Migrating to Apache HBase on Amazon S3 on Amazon EMR

Guidelines and Best Practices

October 2018

© 2018, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Notices

This document is provided for informational purposes only. It represents AWS's current product offerings and practices as of the date of issue of this document, which are subject to change without notice. Customers are responsible for making their own independent assessment of the information in this document and any use of AWS's products or services, each of which is provided "as is" without warranty of any kind, whether express or implied. This document does not create any warranties, representations, contractual commitments, conditions or assurances from AWS, its affiliates, suppliers or licensors. The responsibilities and liabilities of AWS to its customers are controlled by AWS agreements, and this document is not part of, nor does it modify, any agreement between AWS and its customers.

Contents

Introduction 1
Introduction to Apache HBase 1
Introduction to Amazon EMR 2
Introduction to Amazon S3 3
Introduction to EMRFS 3
Running Apache HBase directly on Amazon S3 with Amazon EMR 3
Use cases for Apache HBase on Amazon S3 5
Planning the Migration to Apache HBase on Amazon S3 6
Preparation task 7
Selecting a Monitoring Strategy 7
Planning for Security on Amazon EMR and Amazon S3 9
Encryption 9
Authentication and Authorization 10
Network 12
Minimal AWS IAM Policy 13
Custom AMIs and Applying Security Controls to Harden your AMI 13
Auditing 14
Identifying Apache HBase and EMRFS Tuning Options 16
Apache HBase on Amazon S3 configuration properties 16
EMRFS Configuration Properties 36
Testing Apache HBase and EMRFS Configuration Values 40
Options to approach performance testing 40
Preparing the Test Environment 42
Preparing your AWS account for performance testing 42
Preparing Amazon S3 for your HBase workload 43
Amazon EMR Cluster Setup 44

Troubleshooting 48

Migrating and Restoring Apache HBase Tables on Apache HBase on Amazon S3 48

Data Migration 48

Data Restore 50

Deploying into Production 51

Preparing Amazon S3 for Production load 52

Preparing the Production environment 52

Managing the Production Environment 52

Operationalization tasks 52

Conclusion 56

Contributors 56

Further Reading 56

Document Revisions 57

Appendix A: Command Reference 58

Restart HBase 58

EMRFS TTL sub-commands 58

Appendix B: AWS IAM Policy Reference 60

Minimal EMR Service Role Policy 60

Minimal Amazon EMR Role for Amazon EC2 (Instance Profile) Policy 63

Minimal Role Policy for User Launching Amazon EMR Clusters 65

Appendix C: Transparent Encryption Reference 68

Abstract

This whitepaper provides an overview of Apache HBase on Amazon S3 and guides data engineers and software developers in the migration of an on-premises or HDFS backed Apache HBase cluster to Apache HBase on Amazon S3. The whitepaper offers a migration plan that includes detailed steps for each stage of the migration, including data migration, performance tuning, and operational guidance.

Introduction

In 2006, Amazon Web Services (AWS) began offering IT infrastructure services to businesses in the form of web services—now commonly known as cloud computing. One of the key benefits of cloud computing is the opportunity to replace up-front capital infrastructure expenses with low variable costs that scale with your business. With the cloud, businesses no longer need to plan for and procure servers and other IT infrastructure weeks or months in advance. Instead, they can instantly spin up hundreds or thousands of servers in minutes and deliver results faster. Today, AWS provides a highly reliable, scalable, low-cost infrastructure platform in the cloud that powers hundreds of thousands of businesses in 190 countries around the world.

Many businesses have been taking advantage of the unique properties of the cloud by migrating their existing Apache Hadoop workloads, including Apache HBase, to Amazon EMR and Amazon Simple Storage Service (Amazon S3). The ability to separate your durable storage layer from your compute layer, have flexible and scalable compute, and the ease of integration with other AWS services provide immense benefits and opens up many opportunities to re-imagine your data architectures.

Introduction to Apache HBase

Apache HBase is a massively scalable, distributed big data store in the Apache Hadoop ecosystem. It is an open-source, non-relational, versioned database that runs on top of the Apache Hadoop Distributed Filesystem (HDFS). It is built for random, strictly consistent, real time access for tables with billions of rows and millions of columns. It has tight integration with Apache Hadoop, Apache Hive, Apache Phoenix, and Apache Pig, so you can easily combine massively parallel analytics with fast data access through a variety of interfaces. The Apache HBase data model, throughput, and fault tolerance are a good match for workloads in ad tech, web analytics, financial services, applications using time-series data, and many more.

Here are some of the features and benefits when you run Apache HBase:

- Strongly consistent reads and writes – when a writer returns, all of the readers will see the same value.

- Scalability – individual Apache HBase tables comprise billions of rows and millions of columns. Apache HBase stores data in a sparse form to conserve space. You can use column families and column prefixes to organize your schemas and to indicate to Apache HBase that the members of the family have a similar access pattern. You can also use timestamps and versioning to retain old versions of cells.
- Graphs and timeseries – you can use Apache HBase as the foundation for a more specialized data store. For example, you can use Titan for graph databases and OpenTSDB for time series.
- Coprocessors – you can write custom business logic (similar to a trigger or a stored procedure) that runs within Apache HBase and participates in query and update processing (refer to Apache HBase Coprocessors to learn more).
- OLTP and analytic workloads - you can run massively parallel analytic workloads on data stored in Apache HBase tables by using tools such as Apache Hive and Apache Phoenix. Apache Phoenix provides ACID transaction capabilities via standard SQL and JDBC APIs. For details on how to use Apache Hive with Apache HBase refer to Combine NoSQL and Massively Parallel Analytics Using Apache HBase and Apache Hive on Amazon EMR.

You also get easy provisioning and scaling, access to a pre-configured installation of HDFS, and automatic node replacement for increased durability.

Introduction to Amazon EMR

Amazon EMR provides a managed Apache Hadoop framework that makes it easy, fast, and cost-effective to process vast amounts of data across dynamically scalable Amazon Elastic Compute Cloud (Amazon EC2) instances. You can also run other popular distributed engines, such as Apache Spark, Apache Hive, Apache HBase, Presto, and Apache Flink in Amazon EMR, and interact with data in other AWS data stores, such as Amazon S3 and Amazon DynamoDB. Amazon EMR securely and reliably handles a broad set of big data use cases, including log analysis, web indexing, data transformations (ETL), streaming, machine learning, financial analysis, scientific simulation, and bioinformatics. For an overview of Amazon EMR, refer to Overview of Amazon EMR Architecture and Overview of Amazon EMR.

Introduction to Amazon S3

Amazon Simple Storage Service (Amazon S3) is a durable, highly available, and infinitely scalable object storage with a simple web service interface to store and retrieve any amount of data from anywhere on the web.

With regard to Apache HBase and Apache Hadoop, storing data on Amazon S3 gives you more flexibility to run and shut down Apache Hadoop clusters when you need to. Amazon S3 is commonly used as a durable store for HDFS workloads. Due to the durability and performance scalability of Amazon S3, Apache Hadoop workloads that store data on Amazon S3 no longer require the 3x replication as when the data is stored on HDFS. Moreover, you can resize and shut down Amazon EMR clusters with no data loss or point multiple Amazon EMR clusters at the same data in Amazon S3.

Introduction to EMRFS

The Amazon EMR platform consists of several layers, each with specific functionality and capabilities. At the storage layer, in addition to HDFS and the local file system, Amazon EMR offers the Amazon EMR File System (EMRFS), an implementation of HDFS that all Amazon EMR clusters use for reading and writing files to Amazon S3.

EMRFS features include data encryption, data authorization and consistent view. Data encryption allows EMRFS to encrypt the objects it writes to Amazon S3 and to decrypt them during reads. Data authorization allows EMRFS to use different AWS Identify and Access Management (IAM) roles for EMRFS requests to Amazon S3 based on cluster users, groups, or the location of EMRFS data in Amazon S3. Consistent view allows Amazon EMR clusters to check for list and read-after-write consistency for Amazon S3 objects written by or synced with EMRFS. For more information, refer to Using EMR File System (EMRFS).

Running Apache HBase directly on Amazon S3 with Amazon EMR

When you run Apache HBase on Amazon EMR version 5.2.0 or later, you can enable HBase on Amazon S3. By using Amazon S3 as a data store for Apache HBase, you can separate your cluster's storage and compute nodes. This enables you to save costs by sizing your cluster for your compute requirements instead

of paying to store your entire dataset with 3x replication in the on-cluster HDFS.

Many customers have taken advantage of the numerous benefits of running Apache HBase on Amazon S3 for data storage, including lower costs, data durability, and easier scalability. Customers such as Financial Industry Regulatory Agency (FINRA) have lowered their costs by 60% by moving to an HBase on Amazon S3 architecture in addition to the numerous operational benefits that come with decoupling storage from compute and using Amazon S3 as the storage layer.

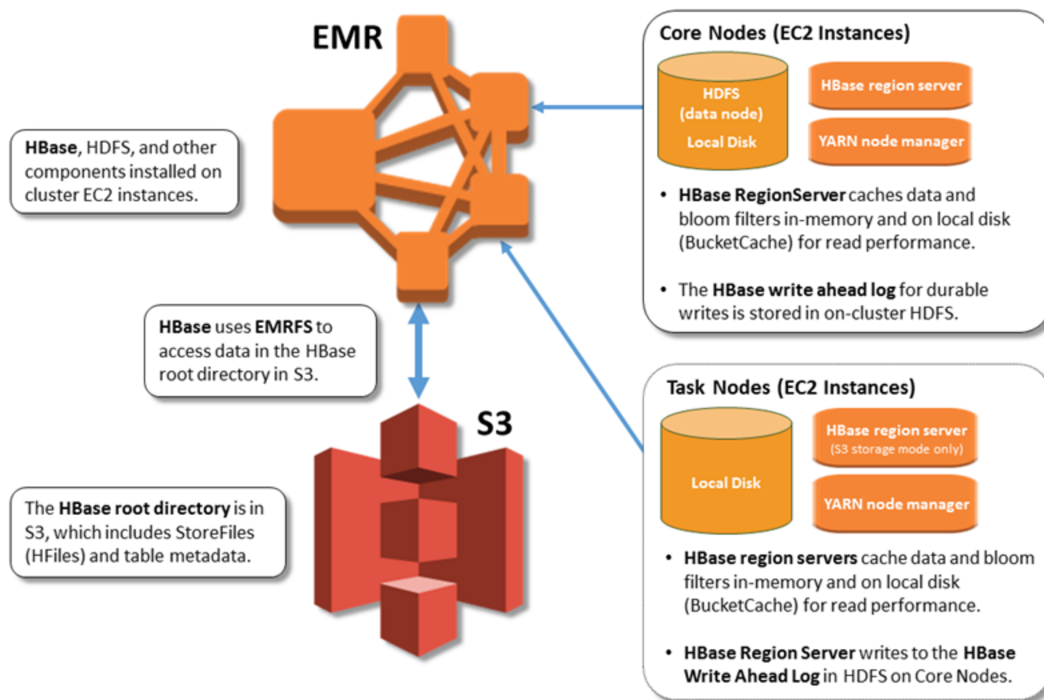

Figure 1: HBase on S3 Architecture

An Apache HBase on Amazon S3 allows you to launch a cluster and immediately start querying against data within Amazon S3. You don't have to configure replication between HBase on HDFS clusters or go through a lengthy snapshot restore process to migrate the data off your HBase on HDFS cluster to another HBase on HDFS cluster. Amazon EMR configures Apache HBase on Amazon S3 to cache data in-memory and on-disk in your cluster, delivering fast performance from active compute nodes. You can quickly and easily scale out or scale in compute nodes without impacting your underlying storage, or you can terminate your cluster to save costs and quickly restore it without having to recover using snapshots or other methods.

Using Amazon EMR version 5.7.0 or later, you can set up a read-replica cluster, which allows you to achieve higher read availability by distributing reads across multiple clusters.

Use cases for Apache HBase on Amazon S3

Apache HBase on Amazon S3 is recommended for applications that require high availability of reads and do not require high availability of writes.

Apache HBase on Amazon S3 can be configured to achieve high requests per second for Apache HBase's API calls. This configuration, together with the proper instance type and cluster size, allows you to find the optimal Apache HBase on Amazon S3 configuration values to support similar requests per second as your HDFS backed cluster. Moreover, as Amazon S3 is used as a storage layer, you can decouple storage from compute, have the flexibility to bring up/down clusters as needed, and considerably reduce costs of running your Apache HBase cluster.

Applications that require high availability of reads are supported by Apache HBase on Amazon S3 via Read Replica Clusters pointing to the same Amazon S3 location. Although Apache HBase on Amazon S3 Read Replica Clusters are not part of this whitepaper, see Further Reading for more details.

Since Apache HBase's Write Ahead Log (WAL) is stored in the cluster, if your application requires support for high availability of writes, Apache HBase on HDFS is recommended. Specifically, you can set up Apache HBase on HDFS with multi-master on an Amazon EC2 custom installation or set up Apache HBase on HDFS on Amazon EMR with an HBase on HDFS replica cluster on Amazon EMR.

Apache HBase on Amazon S3 is recommended if your application does not require support for high availability of writes and can tolerate failures during writes/updates. If you would like to mitigate the impact of Amazon EMR Master node failures (or Availability Zone failures that can cause the termination of the Apache HBase on Amazon S3 cluster or any temporary degradation of service due to an Apache HBase RegionServer operational/transient issues), we

recommend that your pipeline architecture relies on a stream/messaging platform upstream to the Apache HBase on Amazon S3 cluster.

We recommend that you always use the latest Amazon EMR release so you can benefit from all changes and features continuously added to Apache HBase.

Planning the Migration to Apache HBase on Amazon S3

To migrate an existing Apache HBase cluster to an Apache HBase on Amazon S3 cluster, consider the following activities to help scope and optimize performance for Apache HBase on Amazon S3:

- Select a strategy to monitor your Apache HBase cluster's performance
- Plan for security on Amazon EMR and Amazon S3
- Identify Apache HBase and EMRFS tuning options
- Test Apache HBase and EMRFS configuration values
- Prepare the test environment
    - Prepare your AWS account for performance testing
    - Prepare Amazon S3 for your Apache HBase workload
    - Set up Amazon EMR cluster
    - Troubleshoot
- Migrate and restore Apache HBase tables on HBase on Amazon S3
    - Migrate data
    - Restore data
- Deploy into production
    - Prepare Amazon S3 for production load
    - Prepare the production environment
- Manage the production environment
    - Manage operationalization tasks

Preparation task

Before the migration starts, we recommend that you select a strategy to monitor the performance of your cluster.

Selecting a Monitoring Strategy

We recommend you use an enterprise third-party monitoring agent or Ganglia to guide you during the tuning of Apache HBase on Amazon S3. This agent is helpful to understand the changes in performance when changing Apache HBase properties during your tuning process. Moreover, this monitoring allows quick detection of operational issues when the cluster is in production. In addition to monitoring Apache HBase, we also recommend that you monitor EMRFS Consistent View.

Monitoring Apache HBase, subsystems, and dependent systems

To measure the overall performance of Apache HBase, monitor metrics such as those around Remote Procedure Calls (RPCs) and the Java virtual machine (JVM) heap. In addition to Apache HBase metrics, collect metrics from dependency systems, such as HDFS, the OS, and the network.

Monitoring the write path

To measure the performance of the write path, monitor the metrics for the WAL, HDFS (on Apache HBase on Amazon S3 on Amazon EMR, WALs are on HDFS), MemStore, flushes, compactions, garbage collections, and procedure metrics of a related procedure.

Monitoring the read path

To measure the performance of the read path, monitor the metrics for the block cache and the bucket cache. Specifically, monitor the number of evictions, GC time, cache size, and cache hits/misses.

Monitoring with a third-party tool

Apache HBase supports exporting metrics via Java Management Extensions (JMX). Most third-party monitoring agents can then be configured to collect metrics via JMX. For more information refer to Using with JMX. The Configuring HBase to expose metrics via JMX section will provide the

configurations to export Apache HBase metrics via JMX on an Apache HBase on Amazon S3 cluster.

Note that the Apache HBase Web UI allows you access to the available metrics. In the UI, select a RegionServer or the Apache HBase Master, and then click the "Metrics Dump" tab. This tab provides all available metrics collected from the JMX bean and exposes the metrics in JSON format.

For more details on the metrics exposed by Apache HBase, refer to MetricsRegionServerSource.java.

Use the following steps to add monitoring into your Amazon EMR Cluster:

- Create an Amazon EMR bootstrap action to set up the agent of any enterprise monitoring tool used in your environment. (For more information and example bootstrap actions, refer to Create Bootstrap Actions to Install Additional Software.
- Create a dashboard in your enterprise monitoring tool with the metrics to monitor per each Amazon EMR Cluster.
- Create unique tags for each cluster. This tagging avoids multiple clusters writing to the same dashboard.

In addition to monitoring the Amazon EMR Cluster at every layer of the stack, have the monitoring dashboard for your application's API available for use during the performance tests for Apache HBase. This dashboard keeps track of the performance of the application APIs that rely on Apache HBase.

Monitoring Cluster components with Ganglia

The Ganglia open-source project is a scalable, distributed system designed to monitor clusters and grids while minimizing the impact on their performance. When you enable Ganglia on your cluster, you can generate reports and view the performance of the cluster as a whole, as well as inspect the performance of individual node instances. For more information about the Ganglia open-source project, refer to http://ganglia.info/. For more information about using Ganglia with Amazon EMR clusters, refer to Ganglia in Amazon EMR Documentation.

Configuring Ganglia is outside the scope of this whitepaper.

Note that Ganglia produces high amounts of data for large clusters. Consider this information when sizing your cluster. If you choose to use Ganglia to monitor your production cluster, make sure to thoroughly understand Ganglia functionality and configuration properties.

Monitoring EMRFS Consistent View

See Configuring the capacity of the EMRFS metadata table to avoid request throttling for more details on EMRFS Consistent View configurations and monitoring options.

Planning for Security on Amazon EMR and Amazon S3

Many customers in regulated industries, such as financial services or healthcare, require security and compliance controls around their Amazon EMR clusters and Amazon S3 data storage. It is important to consider these requirements as part of an overall data strategy that adheres to any regulatory or internal data security requirements in an industry, such as PCI or HIPAA. This section covers a variety of security best practices around configuring your Amazon EMR environment for HBase on Amazon S3.

Encryption

There are multiple ways to encrypt data-at-rest in your Amazon EMR clusters. If you are using EMRFS to query data on Amazon S3, you can specify one of the following options:

- **SSE-S3:** Amazon S3 manages encryption keys for you
- **SSE-KMS:** An AWS Key Management Service (KMS) customer master key (CMK) encrypts your data server- side on Amazon S3.
- **CSE-KMS/CSE-C:** The encryption and decryption takes place client-side on your Amazon EMR cluster and the encrypted object is stored on Amazon S3. You can use keys provided by AWS KMS (CSE-KMS) or use a custom Java class that provides the master key (CSE-C). When you consider this encryption mode, you should think about the overall ecosystem of tools you will use to access your data and if these tools support CSE-KMS/CSE-C.

In the context of HBase on Amazon S3, many customers use SSE-S3 and SSE-KMS as their methods of encryption because CSE encryption requires additional key management.

Although the bulk of the data is stored on Amazon S3, you still need to consider the following options for local disk encryption:

- **Amazon EMR Security Configuration**: Amazon EMR gives you the ability to encrypt your storage volumes using local disk encryption. It uses a combination of open-source HDFS encryption as well as LUKS encryption. If you want to use this feature, you must specify an AWS KMS key ARN or provide a custom Java class with the encryption artifacts.
- **Custom AMI:** You can create a Custom AMI for Amazon EMR, and specify an Amazon EBS volume encryption to encrypt both your boot and storage volumes.

Amazon EMR security configurations allow you to choose a method for encrypting data in-transit using Transport Layer Security (TLS). You can choose to:

- manually create PEM certificates, zip them in a file, and reference from Amazon S3, or
- implement a certificate custom provider in Java and specify the Amazon S3 path to the JAR.

For more information on how these certificates are used with different big data technologies, refer to In Transit Data Encryption with Amazon EMR. Note that traffic between Amazon S3 and cluster nodes is encrypted using TLS. This encryption is enabled automatically.

Authentication and Authorization

Authentication and authorization, are two crucial components that must be considered when controlling access to data. Authentication is the verification of an entity, whereas authorization is checking whether or not the entity actually has access to the data or resources it is asking for. Another way of looking at it is that authentication is the “are you really who you say you are” check and

authorization is "do you have actually have access to what you're asking for" check. For example, Alice can be authenticated as being Alice, but this does not necessarily mean that Alice has authorization, or access, to look at Bob's bank account.

Authentication on Amazon EMR

Kerberos, a network authentication protocol created by the Massachusetts Institute of Technology (MIT), uses secret-key cryptography to provide strong authentication and avoid sensitive information, such as passwords or other credentials, being sent over the network in an unencrypted and exposed format. With Kerberos, you maintain a set of services (known as a realm) and users that need to authenticate (known as principals) and then provide a means for these principals to authenticate. You can also integrate your Kerberos setup with other realms. For example, you can have users authenticate from an Active Directory domain or LDAP namespace and have a cross-realm trust set up such that these authenticated users can be seamlessly authenticated to access your Amazon EMR clusters.

Amazon EMR installs open source Apache Hadoop ecosystem applications on your cluster, meaning that you can leverage the existing security features in these products. For example, you can enable Kerberos authentication for YARN, giving user-level authentication for applications running on YARN, such as HBase.

You can configure Kerberos on an Amazon EMR cluster (known as Kerberizing) to provide a means of authentication for users who use your clusters. We recommend that you become familiar with Kerberos concepts before configuring Kerberos on Amazon EMR. Refer to Use Kerberos Authentication on the Amazon EMR documentation page. See *Further Reading* for blog posts that show you how to configure Kerberos on your Amazon EMR Cluster.

Authorization on Amazon EMR

Authorization on Amazon EMR falls into three general categories:

- Object-level authorization against objects in Amazon S3.
- Component-specific functionality that is built-in (such as Apache HBase Authorization).

- Tools that provide an intermediary access layer between users running commands on Apache Hadoop components and the storage layer (such as Apache Ranger). (This category is outside the scope of this whitepaper.)

Object-level Authorization

Prior to Amazon EMR version 5.10.0, the AWS Identity and Access Management (IAM) role attached to the Amazon EC2 instance profile role on Amazon EMR clusters determined data access in Amazon S3. Data access to Amazon S3 could only be granular at the cluster-level, making it difficult to have multiple users with potentially different levels of access to data touching the same cluster.

EMRFS fine-grained authorization was introduced with Amazon EMR versions 5.10.0 and later. This authorization allows you to specify the AWS IAM role to assume at the user or group level when EMRFS is accessing Amazon S3. This allows for fine-grained access control for Amazon S3 on multi-tenant Amazon EMR clusters. In addition, it makes it easier to enable cross-account Amazon S3 access to data. For more information on how to configure your security configurations and AWS IAM roles appropriately, refer to Configure AWS IAM Roles for EMRFS Requests to Amazon S3 and Build a Multi-Tenant Amazon EMR Cluster with Kerberos, Microsoft Active Directory Integration and AWS IAM Roles for EMRFS.

HBase Authorization

Authorization on Apache HBase on Amazon S3 is feature-equivalent to Apache HBase on HDFS, with the ability to set authorization rules at the table, column and cell-level. Note that access to the Apache HBase web UIs is not restricted even when Kerberos is used.

Network

The network topology is also important when designing for security and privacy. We recommend placing your Amazon EMR clusters in private subnets, with only outbound internet access via NAT.

Security groups control inbound and outbound access from your individual instances. With Amazon EMR, you can use both Amazon EMR-managed security groups as well as your own to control network access to your instance. By applying the principle of least privilege to your security groups, you can lock

down your Amazon EMR cluster to only the applications and/or individuals who need access.

Minimal AWS IAM Policy

By default, the AWS IAM policies that are associated with Amazon EMR are generally permissive in order to allow customers to easily integrate Amazon EMR with other AWS services. When securing Amazon EMR, a best practice is to start from the minimal set of permissions required for Amazon EMR to function and add permissions as necessary. Since HBase on Amazon S3 depends on Amazon S3 as a storage medium, it is important to ensure that bucket policies are also scoped correctly, such that HBase on Amazon S3 can function while also being secure.

The *AWS IAM Policy Reference* at the end of this paper includes three policies that are scoped around what Amazon EMR minimally requires for basic operation. These policies could be further minimized/restricted by removing actions related to spot pricing and autoscaling.

Custom AMIs and Applying Security Controls to Harden your AMI

Custom AMIs are another approach you can use to harden and secure your Amazon EMR cluster. Amazon EMR uses an Amazon Linux Amazon Machine Image (AMI) to initialize Amazon EC2 instances when you create and launch a cluster. The AMI contains the Amazon Linux operating system, other software, and configurations required for each instance to host your cluster applications.

By default, when you create a cluster, you don't need to think about the AMI. When Amazon EC2 instances in your cluster launch, Amazon EMR starts with a default Amazon Linux AMI that Amazon EMR owns, runs any bootstrap actions you specify, and then installs and configures the applications and components that you select.

Alternatively, if you use Amazon EMR version 5.7.0 or later, you can specify a custom Amazon Linux AMI when you create a cluster and customize its root volume size as well. When each Amazon EC2 instance launches, it starts with your custom AMI instead of the Amazon EMR owned AMI.

Specifying a custom AMI is useful for the following cases:

- Encrypt the Amazon EBS root device volumes (boot volumes) of Amazon EC2 instances in your cluster. For more information, refer to Creating a Custom AMI with an Encrypted Amazon EBS Root Device Volume.

- Pre-install applications and perform other customizations instead of using bootstrap actions, which can improve cluster start time and streamline the startup work flow.

- Implement more sophisticated cluster and node configurations than bootstrap actions allow.

Using a custom AMI, as opposed to a bootstrap action, can allow you to have your hardening steps pre-configured into the images you use, rather than having to run the bootstrap action scripts on instance provision time. You don't have to choose between the two—you can create a custom AMI for the common, less likely to change security characteristics of your cluster and leverage bootstrap actions to pull the latest configurations/scripts that might be cluster-specific.

One approach many of our customers take is to apply the Center for Internet Security (CIS) benchmarks to harden their Amazon EMR clusters. For more details, refer to A step-by-step checklist to secure Amazon Linux. It is important to verify each and every control for necessity and function test against your requirements when applying these benchmarks to your clusters.

Auditing

The ability to audit compute environments is a key requirement for many customers. There are a variety of ways that you can support this requirement within Amazon EMR:

- For Amazon EMR version 5.14.0 and later, EMR File System (EMRFS), Amazon EMR’s connector for Amazon S3, supports auditing of users who ran queries that accessed data in Amazon S3 through EMRFS. This feature is enabled by default and passes on user and group information to audit logs like AWS CloudTrail, providing you with comprehensive request tracking.

- If it exists, application-specific auditing can be configured and implemented on Amazon EMR.
- You can use tools such as Apache Ranger to implement another layer of auditing and authorization.
- AWS CloudTrail, a service that provides a record of actions taken by a user, role, or an AWS service, is integrated with Amazon EMR. AWS CloudTrail captures all API calls for Amazon EMR as events. The calls captured include calls from the Amazon EMR console and code calls to the Amazon EMR API operations. If you create a trail, you can enable continuous delivery of AWS CloudTrail events to an Amazon S3 bucket, including events for Amazon EMR.
- You can also audit the Amazon S3 objects that EMR is accessing via Amazon S3 access logs. AWS CloudTrail only provides logs for AWS API calls, so if a user runs a job that reads/writes data to Amazon S3, the Amazon S3 data that was accessed by Amazon EMR won't appear in AWS CloudTrail. By using Amazon S3 access logs, you can comprehensively monitor and audit access against your data in Amazon S3 from anywhere, including Amazon EMR.
- Because you have full control over your Amazon EMR cluster, you can always install your own third-party agents or tooling via bootstrap actions or custom AMIs to help support your auditing requirements.

Identifying Apache HBase and EMRFS Tuning Options

Apache HBase on Amazon S3 configuration properties

This section helps you optimize components that support the read/write path for your application access patterns by identifying the components and properties to configure. Specifically, the goal of tuning is to prepare the initial configurations to influence cluster behavior, storage footprint behavior, and individual components behavior that support the read and write paths.

The whitepaper covers only HBase tuning properties that were critical to many HBase on Amazon S3 customers during migration. Make sure to test any additional HBase configuration properties that have been tuned on your HDFS backed cluster but not included in this section.-You also need to tune EMRFS properties to prepare your cluster for scale.

This whitepaper should be used together with existing resources or materials on best practices and operational guidelines for HBase.

For a detailed description of the HBase properties mentioned on this document, refer to [HBase default configurations] and [HBase-default.xml (HBase 1.4.6)]. For more details on the metrics mentioned on this document, refer to [MetricsRegionServerSource.java (HBase 1.4.6)]. To monitor changes to some of the properties mentioned on this document, you require access to the Logs for the master and specific Region Servers.

To access the HBase logs during tuning, you can use the HBase Web UI. First select the HBase Master or the specific RegionServer, and then click the "Local Logs" tab. Or, you can SSH to the particular instance that hosts the RegionServer or HBase Master and visualize the last lines added to the logs under `/var/log/hbase`.

Next, we identify the several settings on HBase and later on EMRFS to take into consideration during the tuning stage of the migration.

For some of the HBase properties we propose a starting value or a setting, for others you will need to iterate on a combination of configurations during performance tests to find adequate values.

All of the configuration settings that you decide to set can be applied to your Amazon EMR Cluster via a configuration object that the Amazon EMR service uses to configure HBase and EMRFS when deploying a new cluster. For more details, see *Applying HBase and EMRFS Configurations to the Cluster*.

Speeding up the Cluster initialization

Use the properties that follow when you want to speed up the cluster's startup time, speed up region assignments, and speed up region initialization time. These properties are associated with the HBase Master and the HBase RegionServer.

HBase master tuning

```
hbase.master.handler.count
```

- This property sets the number of RPC handlers spun up on the HBase Master.
- The default value is 30.
- If your cluster has thousands of regions you will likely need to increase this value. Monitor the queue size (`ipc.queue.size`), min and max time in queue, total calls time, min and max processing time, and then iterate on this value to find the best value for your use case.
- Customers at the 20000 region scale have configured this property to 4 times the default value.

HBase RegionServer tuning

```
hbase.regionserver.handler.count
```

- This property sets the number of RPC handlers created on RegionServers to serve requests. For more information about this configuration setting, refer to hbase.regionserver.handler.count.

- The default value is 30.
- Monitor the number of RPC Calls Queued, the 99th percentile latency for RPC calls to stay in queue, and RegionServer memory. Iterate on this value to find the best value for your use case.
- Customers at the 20000 region scale have configured this property to 4 times the default value.

```
hbase.regionserver.executor.openregion.threads
```

- This property sets the number of concurrent threads for region opening.
- The default value is 3.
- Increase this value if the number of regions per RegionServer is high.
- For clusters with thousands of regions, it is not uncommon to see this value at 10 to 20 times the default.

Preventing initialization loops

The default values of the properties that follow may be too conservative for some use cases. Depending on the number of regions, number of RegionServers, and the settings you have chosen to control initialization and assignment times, the default values for the master timeout can prevent your cluster from starting up.

Relevant Master initialization timeouts

```
hbase.master.initializationmonitor.timeout
```

- This property sets the amount of time to sleep in milliseconds before checking the Master's initialization status.
- The default value is 900000 (15 minutes).
- Monitor `masterFinishedInitializationTime` and the HBase Master logs for a "master failed to complete initialization" timeout message. Iterate on this value to find the best value for your use case.

```
hbase.master.namespace.init.timeout
```

- This property sets the time for the master to wait for the namespace table to initialize.
- The default value is 300000 (5 minutes).
- Monitor the HBase Master logs for a "waiting for namespace table to be assigned" timeout message. Iterate on this value to find the best value for your use case.

Scaling to a high number of regions

This property allows the HBase Master to handle high number of regions.

- Set `hbase.assignment.usezk` to false
- For detailed information, refer to [HBase ZK-less Region Assignment.]()

Getting a balanced Cluster after initialization

To ensure that the HBase Master only allocates regions when a target number of RegionServers is available, tune the following properties:

```
hbase.master.wait.on.regionservers.mintostart
hbase.master.wait.on.regionservers.maxtostart
```

- These properties set the minimum and maximum number of RegionServers the HBase Master will wait for before starting to assign regions.
- By default, `hbase.master.wait.on.regionservers.mintostart` is set to 1.
- An adequate value for the min and max is 90 to 95% of the total number of RegionServers. A high value for both min and max results in a more uniform distribution of regions across RegionServers.

```
hbase.master.wait.on.regionservers.timeout
hbase.master.wait.on.regionservers.interval
```

- The `timeout` property sets the time the master will wait for RegionServers to check in. The default value is 4500.

- The `interval` property sets the time period used by the master to decide if no new RegionServers have checked in. The default value is 1500.
- These properties are especially relevant for a cluster with a large number of regions.
- If your use case requires aggressive initialization times, these properties can be set to lower values so that the condition that is dependent on these properties is evaluated earlier.
- Customers at the 20000 region scale and with a requirement of low initialization time, have set `timeout` to 400 and `interval` to 300.
- For more information on the condition used by the master to trigger allocation, refer to HBASE-6389.

Preventing timeouts during Snapshot operations

Tune the following properties to prevent timeouts during snapshot operations:

```
hbase.snapshot.master.timeout.millis
```

- This property sets the time the master will wait for a snapshot to conclude. This property is especially relevant for tables with a large number of regions.
- The default value is 300000 (5 minutes).
- Monitor the logs for snapshot timeout messages and iterate on this value.
- Customers at the 20000 region scale have set this property to 1800000 (30 minutes).

```
hbase.snapshot.thread.pool.max
```

- This property sets the number of threads used by the snapshot manifest loader operation.
- Default value is 8.

- This value depends on the instance type and the number of regions in your cluster. Monitor snapshot average time, CPU usage and your application API to ensure the value you choose does not impact application requests.
- Customers at the 20000 region scale have used 2 to 8 times the default value for this property.

If you will be performing online snapshots while serving traffic, set the following properties to prevent timeouts during the online snapshot operation.

```
hbase.snapshot.region.timeout
```

- Sets the timeout for RegionServers to keep threads in the snapshot request pool waiting.
- Default value is 300000 (5 minutes).
- This property is especially relevant for tables with a large number of regions.
- Monitor memory usage on the RegionServers, monitor the logs for snapshot timeout messages and iterate on this value. Increasing this value consumes memory on the Region Servers.
- Customers at the 20000 region scale have used 1800000 (30 minutes) for this property.

```
hbase.snapshot.region.pool.threads
```

- Sets the number of threads or snapshotting regions on the RegionServer.
- Default value is 10.
- If you decide to increase the value for this property, consider setting a lower value for `hbase.snapshot.region.timeout`.
- Monitor snapshot average time, CPU usage, and your application API to ensure the value you choose does not impact application requests.

Running the balancer for specific periods to minimize the impact of region movements on snapshots

Controlling the operation of the Balancer is crucial for smooth operation of the cluster. These properties provide control over the balancer.

```
hbase.balancer.period
hbase.balancer.max.balancing
```

- The `hbase.balancer.period` property configures when the balancer runs, and the `hbase.balancer.max.balancing` property configures how long the balancer runs.
- These properties are especially relevant if you programmatically take snapshots of the data every few hours because the snapshot operation will fail if regions are being moved. You can monitor the snapshot average time to have more insight into the snapshot operation.

At a high level, balancing requires flushing data, closing the region, moving the region and then opening it on a new RegionServer. For this reason, for busy clusters, consider running the balancer every couple of hours and configure the balancer to only run for one hour.

Tuning the Balancer

Consider the following additional properties when configuring the balancer:

- `hbase.master.loadbalancer.class`
- `hbase.balancer.period`
- `hbase.balancer.max.balancing`

We recommend that you test your current LoadBalancer settings, and then iterate on the configurations. The default LoadBalancer is the Stochastic Balancer. If you choose to use the default LoadBalancer, refer to StochasticLoadBalancer for more details on the various factors and costs associated with this balancer. Most use cases can use the default values.

Access Pattern considerations and read/write path tuning

This section covers tuning the diverse HBase components that support the read/update/write paths.

To properly tune the components that support the read/update/write paths, you start by identifying the overall access pattern of your application.

If the access pattern is read-heavy, then you can reduce the resources allocated to the write path. The same guidelines apply for write-heavy access patterns. For mixed access patterns, you should strive for a balance.

Tuning the Read Path

This section identifies the properties used t when tuning the read path. The properties that follow are beneficial on both random-read and sequential-read access patterns.

Tuning the Size of the BucketCache

The BucketCache is central to HBase on Amazon S3. The properties that follow set the overall size of the BucketCache per instance and allocate a percentage of the total size of the BucketCache to specialized areas, such as single access BucketCache, multiple access BucketCache and in-memory BucketCache. For more details, refer to HBASE-18533.

The goal of this section is to configure the BucketCache to support your access pattern. For an access pattern of random reads and sequential reads, it is recommended to cache all data in the BucketCache, which is stored in disk. In other words, each instance allocates part of its disk to the bucket cache so that the total size of the BucketCache across all the instances in the cluster equals the size of the data on Amazon S3.

To configure the BucketCache, tune the following properties:

```
hbase.bucketcache.size
```

- As a baseline, set the BucketCache to a value equal to the size of data you would like cached.
- This property impacts Amazon S3 traffic. If the data is not in the cache, HBase must retrieve the data from Amazon S3.
- If the BucketCache size is smaller than the amount of data being cached, it may cause many cache evictions which will also increase overhead on Garbage Collection (GC). Moreover, it will increase Amazon S3 traffic.

Set the BucketCache size to the total size of the dataset required for your application's read access pattern.

- Take into account the available disk resources when setting this property. HBase on Amazon S3 uses HDFS for the write path so the total disk available for the BucketCache must consider any storage required by Apache Hadoop/OS/HDFS. See the Amazon EMR Cluster Setup section for recommendations on sizing the cluster local storage for the BucketCache, choosing storage type and its mix (multiple disk versus a single large disk).
- Monitor GC, cache evictions metrics, cache hit ratio, cache miss ratio (you can use HBase UI to quickly access these metrics) to support the process of choosing the best value. Moreover, take into consideration the application SLA requirements to increase or decrease the BucketCache size. Iterate on this value to find the best value for your use case.

```
hbase.bucketcache.single.factor
hbase.bucketcache.multi.factor
hbase.bucketcache.memory.factor
```

- Note that the bucket areas follow the same areas as LRU cache. A block initially read from Amazon S3 is populated in the single-access area (`hbase.bucketcache.single.factor`) and consecutive accesses promote that block into the multi-access area (`hbase.bucketcache.multi.factor`). The in-memory area is reserved for blocks loaded from column families flagged as IN_MEMORY (`hbase.bucketcache.memory.factor`).
- By default, these areas are sized at 25%, 50%, 25% of the total BucketCache size, respectively.
- Tune this value according to the access pattern of your application.
- This property impacts Amazon S3 traffic. For example, if single access is more prevalent than multi access, you can reduce the size allocated to multi access. If multi access is common, ensure that multi access areas are big enough to avoid cache evictions.

```
hbase.rs.cacheblocksonwrite
```

- This property forces all blocks that are written to be added to the cache automatically. Set this property to true.
- This property is especially relevant to read-heavy workloads and setting it to true will populate the cache and reduce Amazon S3 traffic when a read request to the data is issued. Setting this to false in read-heavy workloads will result in reduced read performance and increased Amazon S3 activity.
- HBase on Amazon S3 uses the file base BucketCache together with on-heap cache, BlockCache. This setup is commonly referred as a combined cache. The BucketCache only stores data blocks and the BlockCache stores bloom filters and indices. The physical location of the file base BucketCache is the disk, and the location of the BlockCache is the heap.

Pre-warming the BucketCache

HBase provides additional properties that control the prefetch of blocks when a region is opening. This is a form of cache pre-warming and recommended for HBase on Amazon S3, especially for read access patterns. Pre-warming the BucketCache results in reduced Amazon S3 traffic for subsequent requests. Disabling pre-warming in read-heavy workloads results in reduced read performance and increased Amazon S3 activity.

To configure HBase to prefetch blocks, set the following properties:

```
hbase.rs.prefetchblocksonopen
```

- This property controls whether the server should asynchronously load all of the blocks when a store file is opened (data, meta and index). Note that enabling this property contributes to the time the RegionServer takes to open a region and therefore initialize.
- Set this value to true to apply the property to all tables. This can also be applied per CF instead of using a global setting. You should prefer this over enabling it cluster-wide.
- If you set `hbase.rs.prefetchblocksonopen` to true, the properties that follow increase the number of threads and the size of the queue for the pre-fetch operation:
    - Set `hbase.bucketcache.writer.queuelength` to 1024 as a starting value. The default value is 64.

- Set `hbase.bucketcache.writer.threads` to 6 as a starting value. The default value is 3.
- The values should be configured together and take into consideration the instance type for the RegionServer and the number of regions per RegionServer. By increasing the number of threads, you may be able to choose a lower value for `hbase.bucketcache.writer.queuelength`.
- Property `hbase.rs.prefetchblocksonopen` will control how fast you get data from Amazon S3 during the pre-fetch.
- Monitor HBase logs to understand how fast the bucket cache is being initialized and monitor cluster resources to see the impact of the properties on memory and CPU. Iterate on these values to find the best value for your use case.
- For more details, refer to HBASE-15240.

Modifying the Table Schema to Support Pre-warming

Finally, prefetching can be applied globally or per column family. In addition, the IN_MEMORY region of the BucketCache can be applied per column family.

You must change the schema of the tables to set these properties. For each column family and for the access patterns, you must identify which column families should always be placed in memory and which column families that benefit from prefetching. For example, if one column family is never read by the HBase read path (only read by an ETL job), you can save resources on the cluster and avoid using the PREFETCH_BLOCKS_ON_OPEN Key or the IN_MEMORY for that column family. To modify an existing table to use PREFETCH_BLOCKS_ON_OPEN or IN_MEMORY keys see the following examples:

```
hbase shell
hbase(main):001:0> alter 'MyTable', NAME => 'myCF',
PREFETCH_BLOCKS_ON_OPEN => 'true'
hbase(main):002:0> alter 'MyTable', NAME => 'myCF2', IN_MEMORY
=> 'true'
```

Tuning the Updates/Write Path

This section shows you how to tune and size the MemStore to avoid having frequent flushes and small HFiles. As a result, the frequency of compactions and Amazon S3 traffic is reduced.

```
hbase.regionserver.global.memstore.size
```

- This property sets the maximum size of all MemStores in a Regionserver.
- The memory allocated to the MemStores is kept in the main memory of the RegionServers.
- If the value of `hbase.regionserver.global.memstore.size` is exceeded, updates are blocked, and flushes are forced until the total size of all the MemStores in a RegionServer is at or below the value of `hbase.regionserver.global.memstore.size.lower.limit`.
- The default value is 0.4 (40% of the heap).
- For write-heavy access patterns, you can increase this value to increase the heap area dedicated to all MemStores.
- Consider the number of regions per Region Server and the number of column families with high write activity when setting this value.
- For read-heavy access patterns, this setting can be decreased to free up resources that support the read path.

```
hbase.hregion.memstore.flush.size
```

- This property sets the flush size per MemStore.
- Depending on the SLA of your API, the flush size may need to be higher than the flush size configured on the source cluster.
- This setting impacts the traffic to Amazon S3, the size of HFiles, and the impact of compactions in your cluster. The higher you set the value, the fewer Amazon S3 operations are required, and the higher the size of each resulting HFile.
- This value is dependent on the total number of regions per RegionServer and the number of column families with high write activity.

- Use 536870912 bytes (512 MB) as the starting value, then monitor the frequency of flushes, the Memstore Flush Queue Size, and Application APIs response time. If frequency of flushes and queue size is high, increase this setting.

```
hbase.regionserver.global.memstore.size.lower.limit
```

- When the size of all Memstores exceed this value, flushes are forced. This property prevents the Memstore from being blocked for updates.
- By default, this property is set to 0.95, 95% of the value set for `hbase.regionserver.global.memstore.size`.
- This value depends on the number of Regions per RegionServer and the write activity in your cluster.
- You might want to decrease this value if as soon as the Memstores reach this safety threshold, the write activity quickly fills the missing 0.05 and the MemStore is blocked for writes.
- Monitor the frequency of flushes, the Memstore Flush Queue Size, and Application APIs response time. If frequency and queue size is high, increase this setting.

```
hbase.hregion.memstore.block.multiplier
```

- This property is a safety threshold and controls spikes in write traffic.
- Specifically, this property sets the threshold at which writes are blocked. If the MemStore reaches `hbase.hregion.memstore.block.multiplier` times `hbase.hregion.memstore.flush.size` bytes writes are blocked.
- In case of spikes in traffic, this property prevents the Memstore from continuing to grow and in this way prevents HFiles with large sizes.
- The default value is 4.
- If your traffic has a constant pattern, consider keeping the default value for this property and tune only `hbase.hregion.memstore.flush.size`.

```
hbase.hregion.percolumnfamilyflush.size.lower.bound.min
```

- For the tables that have multiple column families, this property forces HBase to only flush the MemStores of each column family that exceed `hbase.hregion.percolumnfamilyflush.size.lower.bound.min`.
- The default value for this property is 16777216 bytes.
- This setting impacts the traffic to Amazon S3. A higher value reduces traffic to Amazon S3.
- For write-heavy access patterns with multiple column families, this property should be changed to a value higher than the default of 16777216 bytes but less than half of the value of `hbase.hregion.memstore.flush.size`.

```
hfile.block.cache.size
```

- This property sets the percentage of the heap to be allocated to the BlockCache.
- Use the default value of 0.4 for this property.
- By default, the BucketCache stores data blocks, and the BlockCache stores bloom filters and indices.
- You will need to allocate enough of the heap to cache indices and bloom filters, if applicable. To measure HFile indices and bloom filters sizes, access the web UI of one of the RegionServers.
- Iterate on this value to find the best value for your use case.

```
hbase.hstore.flusher.count
```

- This property controls the number of threads available to flush writes from memory to Amazon S3.
- The default value is 2.

- This setting impacts the traffic to Amazon S3. A higher value reduces the MemStore flush queue and speeds up writes to Amazon S3. This setting is valuable for write-heavy environments. The value is dependent on the instance type used by the cluster.
- Test the value and gradually increase it to 10.
- Monitor the MemStore flush queue size, the 99th percentile for flush time, and application API response times. Iterate on this value to find the best value for your use case.

> Note: Small clusters with high region density and high write activity should also tune HDFS properties that allow the HDFS NameNode and the HDFS DataNode to scale. Specifically, properties `dfs.datanode.handler.count` and `dfs.namenode.handler.count` should be increased to at least 3x their default value of 10.

Region size considerations

Since this process is a migration, set the region size to the same region size on your HDFS backed cluster.

As a reference, on HBase on Amazon S3, customer regions fall into these categories: small-sized regions (1-10GB), mid-sized regions (10-50GB), and large-sized regions (50-100GB).

Controlling the Size of Regions and Region Splits

This property sets the size of the regions in your cluster. This property should be configured together with the property `hbase.regionserver.region.split.policy` which is not covered on this whitepaper.

- Use your current cluster's value for `hbase.hregion.max.filesize`
  - As a starting point you can use the value in your HDFS backed cluster.
- Set `hbase.regionserver.region.split.policy` to the same policy in your HDFS backed cluster
  - This property controls when a region should be split.

  - The default value is `org.apache.hadoop.hbase.regionserver.SteppingSplitPolicy.`

- Set `hbase.regionserver.regionSplitLimit` to the same value in your HDFS backed cluster.
  - This property acts as a guideline/limit for the RegionServer to stop splitting.
  - Its default value is 1000.

Tuning HBase Compactions

This section shows you how to configure properties that control major compactions, reduce the frequency of minor compactions, and control the size of HFiles to reduce Amazon S3 traffic.

Controlling Major Compactions

In production environments, we recommend you disable major compaction. However, there should always be a process to run major compactions. Some customers opt to have an application that incrementally runs major compactions in the background, in a table, or RegionServer basis.

Set `hbase.hregion.majorcompaction` to 0 to disable automatically scheduled major compactions.

Reduce the frequency of minor compactions and control the size of HFiles to reduce Amazon S3 traffic

The following properties are dependent on the MemStore size, flush size, and the need to control the frequency of minor compactions.

The properties that follow should be set according to response time needs and average size of generated StoreFiles during a MemStore flush.

To control the behavior of minor compactions, tune these properties:

```
hbase.hstore.compaction.min.size
```

- If a StoreFile is smaller than the value set by this property, the StoreFile will be selected for compaction. If StoreFiles have a size equal or larger

than the value of `hbase.hstore.compaction.min.size`, `hbase.hstore.compaction.ratio` is used to determine if the files are eligible for compaction.

- By default, this value is set to 134217728 bytes.
- This setting depends on the frequency of flushes, the size of StoreFiles generated by flushes, and `hbase.hregion.memstore.flush.size`.
- This setting impacts the traffic to Amazon S3. The higher you set the value, the more frequent minor compactions will occur, and therefore trigger Amazon S3 activity.
- For write-heavy environments with many small StoreFiles, you may want to decrease this value to reduce the frequency of minor compactions and therefore Amazon S3 activity.
- Monitor the frequency of compactions, the overall StoreFile size, and iterate on this value to find the best value for your use case.

```
hbase.hstore.compaction.max.size
```

- If a StoreFile is larger than the value set by this property, the StoreFile is not selected for compaction.
- This value setting depends on the size of the HFiles generated by flushes and the frequency of minor compactions.
- If you increase this value, you will have fewer, larger StoreFiles and increased Amazon S3 activity. If you decrease this value you will have more, smaller StoreFiles, and reduced Amazon S3 activity.
- Monitor the frequency of compactions, the compaction output size, the overall StoreFile size, and iterate on this value.

```
hbase.hstore.compaction.ratio
```

Accept the default of 1.0 as a starting value for this property. For more details on this property, refer to hbase-default.xml.

```
hbase.hstore.compactionThreshold
```

- If a store reaches `hbase.hstore.compactionThreshold`, a compaction is run to re-write the StoreFiles into one.
- A high value will result in less frequent minor compactions, larger StoreFiles, longer minor compactions, and less Amazon S3 activitiy.
- The default value is 3.
- Start with a value of 6, monitor Compaction Frequency, the average size of StoreFiles, compaction output size, compaction time, and iterate on this value to get the best value for your use case.

```
hbase.hstore.blockingStoreFiles
```

- This property sets the total number of StoreFiles a single store can have before updates are blocked for this region. If this value is exceeded, updates are blocked until a compaction concludes or `hbase.hstore.blockingWaitTime` is exceeded.
- For write-heavy workloads, use two to three times the default value as a starting value.
- The default value is 16.
- Monitor the frequency of flushes, blocked requests count, and compaction time to set the proper value for this property.

Minor and major compactions will flush the BucketCache. For more details, refer to HBASE-1597.

Controlling the storage footprint locally and on Amazon S3

At a high level, on HBase on Amazon S3, WALs are stored on HDFS. When a compaction occurs, previous HFiles are moved to the archive and only deleted if they are not associated with a snapshot. HBase relies on a Cleaner Chore that is responsible for deleting unnecessary HFiles and expired WALs.

Ensuring the Cleaner Chore is always running

With HBase 1.4.6 (Amazon EMR version 5.17.0 and later), we recommend that you deploy the cluster with the cleaner enabled. This is the default behavior. The property that sets this behavior is `hbase.master.cleaner.interval.`

We recommend that you use the latest Amazon EMR release. For versions prior to Amazon EMR 5.17.0, see the Operational Considerations section for the HBase shell commands that control the cleaner behavior.

To enable the cleaner globally, set the `hbase.master.cleaner.interval` to 1.

Speeding up the Cleaner Chore

HBASE-20555, HBASE-20352 and HBASE-17215 add additional control to the Cleaner Chore that deletes expired WALs (HLogCleaner) and expired archived HFiles (HFileCleaner). These controls are available on HBase 1.4.6 (Amazon EMR version 5.17.0) and later.

The number of threads allocated to the preceding properties should be configured together and take into consideration the capacity and instance type of the Amazon EMR Master node.

```
hbase.regionserver.hfilecleaner.large.thread.count
```

- This property sets the number of threads allocated to clean expired large HFiles.
- `hbase.regionserver.thread.hfilecleaner.throttle` sets the size that distinguishes between a small and large file. The default value is 64 MB.
- The value for this property is dependent on the number of flushes, write activity in the cluster, and snapshot deletion frequency.
- The higher the write and snapshot deletion activity, the higher the value should be.
- By default, this property is set to 1.
- Monitor the size of the HBase root directory on Amazon S3 and iterate on this value to find the best value for your use case. Consider the Amazon EMR Master CPU resources and the values set for the other

configuration properties identified in this section. For more information, see the Enabling Amazon S3 metrics for the HBase on Amazon S3 root directory section.

```
hbase.regionserver.hfilecleaner.small.thread.count
```

- This property sets the number of threads allocated to clean expired small HFiles.
- The value for this property is dependent on the number of flushes, write activity in the cluster, and snapshot deletion frequency.
- By default, this property is set to 1.
- The higher the write and snapshot deletion activity, the higher the value should be.
- Monitor the size of the HBase root directory on Amazon S3 and iterate on this value to find the best value for your use case. Consider the Amazon EMR Master CPU resources and the values set for the other configuration properties identified in this section.

```
hbase.cleaner.scan.dir.concurrent.size
```

- This property sets the number of threads to scan the oldWALs directories.
- The value for this property is dependent on the write activity in the cluster.
- By default, this property is set to ¼ of all available cores.
- Monitor HDFS use and iterate on this value to find the best value for your use case. Consideration the Amazon EMR Master CPU resources and the values set for the other configuration properties identified in this section.

```
hbase.oldwals.cleaner.thread.size
```

- This property sets the number of threads to clean the WALs under the oldWALs directory.
- The value for this property is dependent on the write activity in the cluster.
- By default, this property is set to 2.
- Monitor HDFS use and iterate on this value to find the best value for your use case. Consider the Amazon EMR Master CPU resources and the values set for the other configuration properties identified in this section.

For more details on how to set the configuration properties to clean expired WALs, refer to HBASE-20352.

Porting existing settings to HBase on Amazon S3

Some properties you have tuned in your on-premesis cluster but were not included in the Apache HBase tuning section. These configurations include the heap size for HBase and supporting Apache Hadoop components, Apache HBase Split Policy, Apache Zookeeper timeouts, and so on. For these configuration properties, use a the value in your HDFS backed cluster as a starting point. Follow the same process to iterate and determine the best value that supports your use case.

EMRFS Configuration Properties

We strongly recommend that Amazon Clusters using Apache HBase on Amazon S3 are configured with EMRFS Consistent View. To enable EMRFS Consistent View, Amazon EMR uses an Amazon DynamoDB table, the EMRFS metadata table, to store object metadata and track consistency with Amazon S3. This table is created automatically for you or you can point the cluster to use a previously created EMRFS metadata table. When Consistent view is enabled, EMRFS adds information about the objects it writes to Amazon S3 to the Amazon DynamoDB table. When EMRFS deletes an object, a listing still remains in the metadata with a deleted state that can be purged via the EMRFS CLI or a TTL.

For every Amazon S3 operation, EMRFS checks the metadata for information about the set of objects in consistent view. If EMRFS finds that Amazon S3 is inconsistent during one of these operations, it retries the operation according to parameters defined in emrfs-site configuration properties. After EMRFS

exhausts the retries, it either throws a ConsistencyException or logs the exception and continues the workflow.

For detailed information on how EMRFS Consistent View tracks objects on Amazon S3, refer to Understanding How EMRFS Consistent View Tracks Objects in Amazon S3. For more information on the EMRFS CLI commands, refer to EMRFS CLI Reference.

For more details about the EMRFS properties included in this section, refer to Configure Consistent View.

Setting the total number of connections used by EMRFS to read/write data from/to Amazon S3

With HBase on Amazon S3, access to data is done via EMRFS. This means that tasks such as an Apache HBase Region opening, MemStore flushes and compactions all will trigger a request to Amazon S3. To support workloads for a large number of regions and datasets, you must tune the total number of connections to Amazon S3 that EMRFS can make (`fs.s3.maxConnections`).

To tune `fs.s3.maxConnections`, account for the average size of the HFiles, number of regions, the frequency of minor compactions, and the overall read and write throughput the cluster is experiencing.

```
fs.s3.maxConnections
```

- The default value for HBase on Amazon S3 is 10000. This value should be set to more than 10000 for clusters with a large number of regions (2500+), large datasets (1TB+), high minor compactions activity, and intense read/write activity.
- Monitor the logs for the ERROR message “Unable to execute HTTP request: Timeout waiting for connection” and iterate on this value. See more details about this error message in the Troubleshooting section.
- Several customers at the +50TB/20k regions scale set this property to 50000.

Setting the behavior of EMRFS in case inconsistencies are detected

The options that follow control the behavior in case EMRFS detects an inconsistency on Amazon S3. Specifically,

`fs.s3.consistent.retryPeriodSeconds` sets the initial backoff time and `fs.s3.consistent.retryCount` sets the maximum number of retries when an inconsistency is detected. After the initial backoff time, the value of `fs.s3.consistent.retryPeriodSeconds` grows exponentially during subsequent retries.

Set `fs.s3.consistent.retryPeriodSeconds` to 1 and set `fs.s3.consistent.retryCount` to 20.

Configuring the capacity of the EMRFS metadata table to avoid request throttling

The following properties set the read and write throughput capacity used by the Amazon DynamoDB table that supports the EMRFS Consistent View.

```
fs.s3.consistent.metadata.read.capacity
fs.s3.consistent.metadata.write.capacity
```

- Accept the default as a starting value.
- Monitor the read and write capacity plus number of throttled requests and iterate on these values.
- See the Monitoring the EMRFS metadata Table section for more details on monitoring.

Updating the read and write capacity of the EMRFS metadata after initial creation

Once the Amazon DynamoDB table that supports EMRFS consistent view is created, you can perform any changes to the read and write capacity throughput via the Amazon DynamoDB API instead of the EMRFS properties.

Monitoring the EMRFS metadata table

This section provides more details on how to monitor the EMRFS metadata table via Amazon DynamoDB Web UI and Amazon CloudWatch.

The Amazon DynamoDB Web UI and Amazon CloudWatch expose metrics on the IOPS consumed by each operation and the number of requests that are throttled. We recommend that you monitor these metrics and update the read and write capacity allocated to the table accordingly. For more information, refer to Monitoring Tools and Monitoring with Amazon CloudWatch.

Preventing the EMRFS Metadata table from growing indefinitely

To avoid the Amazon DynamoDB table that support EMRFS metadata grows infinitely, we recommend that you enable time to live (TTL) Support for the table. This support is only be applied to objects deleted from the metadata.

To enable TTL for each metadata object marked as delete, apply the following `emrfs-site.xml` properties and execute the configuration steps to enable TTL support. The *Amazon EMR Cluster Setup* section provides you with details on how to properly apply any settings to `emrsfs-site.xml` via classifications.

Editing Emrfs-site.xml

Set `fs.s3.consistent.metadata.delete.ttl.enabled` to true.

Set `fs.s3.consistent.metadata.delete.ttl.expiration.seconds` to 3600 (1h).

Enabling TTL Support for the Amazon DynamoDB table

As soon as the metadata table is created, enable TTL on attribute deletionTTL. This step must only be performed once and can be done via the Web UI or command line. For more details on configuring the Amazon DynamoDB table to support the TTL property, refer to New – Manage Amazon DynamoDB Items Using Time to Live (TTL).

Isolating the EMRFS metadata table from other clusters

We recommend that each Apache HBase on Amazon S3 root directory uses a single Amazon DynamoDB table. Associating one Amazon DynamoDB table with a single cluster provides you full control over the Amazon DynamoDB's table provisioned capacity. In opposition, sharing a single Amazon DynamoDB table with multiple clusters can exhaust the table's provisioned capacity as multiple clusters are consuming the table's resources.

To provide a custom name to the EMRFS Metadate table, set `fs.s3.consistent.metadata.tableName` to a custom name.

Testing Apache HBase and EMRFS Configuration Values

Options to approach performance testing

During the testing phase, we recommend that you use the metrics for the relevant HBase sub components together with the overall response times of your application to gauge the impact of the changes made to HBase properties.

We also recommend that you start by testing the HBase configuration settings that contribute to a healthy cluster state at your dataset scale (fast initialization times, balanced cluster, and so on), and then focus on testing the configuration property values for the read and write/update paths.

We provide guidelines on how to size the cluster compute and local storage resources. The R5/R5d instance types are good candidates for a starting point as they are memory-optimized instances.

As you focus on tuning the read and write/update paths, we recommend you iterate on the number of regions per RegionServer (cluster size). As a starting value, you can use the same region density as in your HDFS-backed cluster and iterate according to the behavior indicated by the metrics for the RegionServers resources and HBase read/write path components. For more details, see Sizing Compute Capacity, Selecting an Instance Type. Also, consider costs while you iterate on instance size and type. Refer to the AWS Simple Monthly Calculator to quickly help you estimate costs for the different clusters of your test environment.

To test the HBase configuration values you have selected as a starting value, use one of the following options.

Traffic Segmentation

If the use case permits and the application traffic can be segmented by API/Table, consider creating empty tables pre-partitioned with the same number of regions as the original and then have the test cluster receive 10-50% of the production traffic. Although this won't be an accurate representation of the production load, you will be able to iterate faster on the configurations for most HBase components. This way, as soon as the HBase configuration values

have been identified for the smaller cluster/set up, you can deploy a new cluster, gradually increase the traffic load, and iterate again on the configurations.

Dataset Segmentation

Dataset segmentation is especially relevant for datasets on the terabyte and petabyte scale. If you choose this option and the use case permits, we recommend that you use between 10 to 30% of the overall dataset and iterate to find the HBase configuration values that contribute to a stable cluster and good response time for your application's APIs. Alternatively, you can focus on a few tables at first. As soon as you are satisfied with the performance with a subset of the dataset or some of the tables, you can deploy to a new cluster pointing to the full data set and iterate again on the configurations.

We provide steps on how to migrate and restore the full datasets in the next section.

For both options, when you have identified a set of HBase properties that can be adjusted to improve stability and performance, you can apply the configurations to each node of the cluster with a script and then restart HBase. For more details on the steps to restart HBase, see the Rolling Restart section.

When you are satisfied with the cluster behavior and application response times with segmented traffic and dataset, you can also iterate on the instance size and instance type for both the Amazon EMR Master and Amazon EMR Core/Task Nodes. When you are ready to do so, you can terminate the test cluster, update the Amazon EMR Configuration Settings, and deploy a new cluster. See the Cluster termination without data loss section to follow the correct cluster termination procedure.

Finally, when you are ready to test with the full production traffic and full production dataset, size the cluster accordingly using the metrics for the previous tests as a reference. Then, migrate the data and redeploy a new Amazon EMR Cluster.

Preparing the Test Environment

Preparing your AWS account for performance testing

To identify the optimal configuration of your HBase on Amazon S3 cluster, you will need to iterate on several configuration values during a testing stage. Not only will you make changes to HBase configurations but also to the type and family of the cluster's Amazon EC2 instances.

To avoid any impact on existing workloads on the account used for testing or production, we recommend that you raise the limits identified in this section according to your testing or production account needs.

Increasing Amazon EC2 and Amazon EBS Limits

To avoid any delays during performance tests, raise the following limits in your AWS account since you may need to deploy several clusters at the same time (replicas, clusters pointing to different HBase root directories, and so on). If your cluster size is small, the default values may be sufficient. For more details about the current limits applied into your account, refer to Trusted Advisor (Login Required). If your cluster is expected to have more than 100 instances, open an AWS Support Case (Login Required) to have the following Amazon EC2 and Amazon EBS limits increased:

- R5/R5d family: increase the limit to 200% of your clusters estimated size for xl, 2xl and 4xl.
- Total volume storage of General Purpose SSD (gp2) volumes: increase the limit with additional capacity (4x the total dataset size).

  For example: if dataset is 40TB, the SSD available (instance store or Amazon EBS Volumes) must be at least 40TB. Account for additional storage because you may need to deploy several clusters at the same time (replicas, clusters pointing to different Apache HBase root directories). See the Sizing Local Storage section for more details.

Increasing AWS KMS limits

Amazon S3 encryption works with EMRFS objects read from and written to Amazon S3. If you do not have a security requirement for data at-rest, then you

can skip this section. If your cluster is small, the default values may be sufficient. For additional details about AWS KMS limits, refer to Requests per second limit for each AWS KMS API operation.

Increasing Amazon DynamoDB limits

EMRFS Consistent View relies on an Amazon DynamoDB table that Amazon EMR creates automatically for you. Some of the limits to consider are the default limits for tables per account and default limits for provisioned throughput per account. For additional details about these Amazon DynamoDB limits, refer to Provisioned Throughput Default Limits and Tables Per Account.

Preparing Amazon S3 for your HBase workload

Amazon S3 can scale to support very high request rates to support your HBase on Amazon S3 cluster. It's valuable to understand the exact performance characteristics of your HBase workloads when migrating to a new storage layer, especially when moving to an object store such as Amazon S3.

Amazon S3 automatically scales to high request rates and currently supports up to 3500 PUT/POST/DELETE requests per second and 5500 GET requests per second per prefix in a bucket. If your request rate grows steadily, Amazon S3 automatically scales beyond these rates as needed.

If you expect the request rate per prefix to be higher than the preceding request rate, or if you expect the request rate to rapidly increase instead of gradually increase, the Amazon S3 bucket must be prepared to support the workloads of your HBase on Amazon S3 Cluster. For more details on how to prepare the Amazon S3 bucket, see the Preparing Amazon S3 for production load section. This helps minimize throttling on Amazon S3. To understand how you can recognize that Amazon S3 is throttling the requests from your cluster, see the Troubleshooting section.

Enabling Amazon S3 metrics for the HBase on Amazon S3 root directory

The Amazon CloudWatch request metrics for Amazon S3 enable the collection of Amazon S3 API metrics for a specific bucket. These metrics provide a good understanding of the TPS driven by your HBase cluster and they can be helpful to identify any operational issues.

> **Note:** Amazon CloudWatch metrics incur a cost. For more information, refer to How Do I Configure Request Metrics for an S3 Bucket? and Monitoring Metrics with Amazon CloudWatch.

Enabling Amazon S3 lifecycle rules to end and clean up incomplete multipart uploads

HBase on Amazon S3 via EMRFS uses Amazon S3 Multipart API. The Multipart upload API enables EMRFS to upload large objects in parts. For more details on the Multipart API, refer to Multipart Upload Overview.

> Note: After you initiate a multipart upload and upload one or more parts, you must either complete or abort the multipart upload to stop storage charges of the uploaded parts. Only after you either complete or abort a multipart upload will Amazon S3 free up the parts storage and stop charging you for the parts storage.

Amazon S3 provides a lifecycle rule that, when configured, automatically removes incomplete multipart uploads. For complete steps on how to create a Bucket Lifecycle Policy and apply it to the HBase root directory bucket, refer to Aborting Incomplete Multipart Uploads Using a Bucket Lifecycle Policy. Alternatively, you can use the AWS Console and configure the Lifecycle policy. For more details, refer to Amazon S3 Lifecycle Management Update – Support for Multipart Uploads and Delete Markers. We recommend that you configure the lifecycle policy to end and clean up incomplete multipart uploads after 3 days.

Amazon EMR Cluster Setup

Selecting an Amazon EMR Release

We strongly recommended that you use the latest release of Amazon EMR when possible. Refer to Amazon EMR 5.x Release Versions to find the software versions available at the latest Amazon EMR release. For more details, refer to Migrating from Previous HBase Versions.

We also recommend that you deploy the cluster with only the required applications. This is especially important in production so you can properly use the full resources of the cluster.

Applying HBase and EMRFS Configurations to the Cluster

Amazon EMR allows the configuration of applications by supplying a JSON object with any changes to default values. For more information, refer to [Configuring Applications](#).

Applying HBase configurations

This section includes guidelines on how to construct the JSON object that can be supplied to the cluster during cluster deployment. Most of these properties are configured on the `hbase-site.xml` file.

Other settings of HBase, such as Region and Master server heap size and logging settings, have their own configuration file and their own classification when setting up the JSON object.

For an example JSON object to configure the properties written to `hbase-site.xml`, see [Configure HBase](#). In addition to `hbase-site` classification, you may need to use classification `hbase-log4j` to change values in HBase's `hbase-log4j.properties` file and classification `hbase-env` to change values in HBase's environment.

Configuring HBase to expose metrics via JMX

An example JSON object to configure HBase to expose metrics via JMX can be found below.

```
[
  {
    "Classification": "hbase-env",
    "Properties": {
    },
    "Configurations": [
      {
        "Classification": "export",
        "Properties": {
          "HBASE_REGIONSERVER_OPTS": " -
Dcom.sun.management.jmxremote.ssl=false -
```

```
Dcom.sun.management.jmxremote.authenticate=false -
Dcom.sun.management.jmxremote.port=10102",
          "HBASE_MASTER_OPTS": "-
Dcom.sun.management.jmxremote.ssl=false -
Dcom.sun.management.jmxremote.authenticate=false -
Dcom.sun.management.jmxremote.port=10101"
        },
        "Configurations": [
        ]
      }
    ]
  }
]
```

Configuring the Log Level for HBase

```
  {
    "Classification": "hbase-log4j",
    "Properties": {
      "log4j.logger.org.apache.hadoop.hbase": "DEBUG"
    }
  }
```

Applying EMRFS configurations

For an example JSON object to configure the EMRFS properties, refer to Configure Consistent View.

Sizing the cluster compute and local storage resources

Sizing Compute Capacity, Selecting an Instance Type

When sizing your cluster, you can consider having a large cluster with a smaller instance type or having a small cluster with a more powerful instance type. We recommend extensive testing to find the correct instance type for your application SLA. As a starting point, you can use the latest generation of memory optimized instance types (R5/R5d) and the same region density per RegionServer as in your HDFS backed cluster. R5d instances share the same specifications as R5 instances, and also include up to 3.6TB of local NVMe storage. For more details on these instance types, refer to Now Available: R5, R5d, and z1d Instances. As you progress to tune the read and write path, first

establish a configuration that supports the SLA of your application. Then, increase the region density by reducing the number of nodes in the cluster.

Sizing Local Storage

The disk requirements of the cluster depend on your application SLA and access patterns. As a rule of thumb, read intensive applications benefit from caching data on the BucketCache. For this reason, the disk size should be large enough to cover all caching requirements, HDFS requirements (write path), and OS and Apache Hadoop requirements.

Storage options on Amazon EMR

On Amazon EMR, you have the option to choose an Amazon EBS volume or the instance store. The AMI used by your cluster dictates whether the root device volume uses the instance store or an Amazon EBS volume. Some AMIs use Amazon EC2 instance store, and some use Amazon EBS. When you configure instance types in Amazon EMR, you can add Amazon EBS volumes, which contribute to the total capacity together with instance store (if present) and the default Amazon EBS volume. Amazon EBS provides the following volume types: General Purpose (SSD), Provisioned IOPS (SSD), Throughput Optimized (HDD), Cold (HDD), and Magnetic. They differ in performance characteristics and price to support multiple analytic and business needs. For a detailed description of storage options on Amazon EMR, refer to Instance Store and Amazon EBS.

Selecting and Sizing Local Storage for the BucketCache

Most HBase workloads perform well with General Purpose volumes (GP2) volumes. The volume mix per Amazon EMR Core instances can be either two or more large volumes, or multiple small volumes, in addition to the root volume. Note that when your instance has multiple volumes, the BucketCache is divided across *n-1* volumes. The first volume stores logs and temporary data. See the Tuning the Size of the BucketCache section for details on how to choose a starting value for the size of the BucketCache and the stark disk requirements for your Amazon EMR Core/Task nodes.

Applying Security Configurations to Amazon EMR and EMRFS

You can use Security Configurations to apply the configurations that support at-rest data encryption, in-transit data encryption, and authentication. For more details, see Create a Security Configuration.

Depending on the strategy you choose for authorizing access to HBase, HBase configurations can be applied via the same process included in the Applying HBase and EMRFS Configurations to the Cluster.

Due to performance issues reported when Block encryption is using 3DES, Transparent Encryption is preferred over encrypting block data transfer. For more details on Transparent Encryption, see the Transparent Encryption Reference section.

Troubleshooting

Error message excerpt	Description/Solution
`Please reduce your request rate. (Service: Amazon S3; Status Code: 503; Error Code: SlowDown…)`	Amazon S3 is throttling requests from your cluster due to an excessive number of transaction per second to specific object prefixes. **Find the request rate and prepare the Amazon S3 bucket for that request rate.** Use the metrics for the Amazon S3 bucket location for the HBase root directory to review the number of requests for the previous hour (request rate). See the Preparing Amazon S3 for your HBase workload and Preparing Amazon S3 for Production load sections for details on how to prepare the Amazon S3 bucket location for the HBase root directory for your request rate.
`Unable to execute HTTP request: Timeout waiting for connection from pool`	Increase the value of the `fs.s3.maxConnections` property. See the Setting the total number of connections used by EMRFS to read/write data from/to Amazon S3 section for more details on how to tune this property.

Migrating and Restoring Apache HBase Tables on Apache HBase on Amazon S3

Data Migration

This paper covers using the ExportSnapshot tool to migrate the data. For additional options, see Tips for Migrating to Apache HBase on Amazon S3 from HDFS.

Creating a Snapshot

To create a snapshot, perform the following commands from the HBase shell:

```
hbase shell
hbase(main):001:0>disable 'table_name'
hbase(main):002:0>snapshot 'table_name',
'table_name_snapshot_date'
hbase(main):003:0>enable 'table_name'
```

If you are taking the snapshot from a production HBase cluster and cannot afford service disruption, you do not need to disable the table to take a snapshot. There is minimal performance degradation if you keep the table active. However, there may be some inconsistencies between the state of the table at the end of the snapshot operation and the snapshot contents.

If you can afford service disruption in your production HBase cluster, disabling the table guarantees that the snapshot is fully consistent with the state of the disabled table.

Validating the Snapshot

As soon as the snapshot is concluded, use the following command to check that the snapshot was successful.

```
hbase org.apache.hadoop.hbase.snapshot.SnapshotInfo -stats -
snapshot table_name_snapshot_date
Snapshot Info
----------------------------------------
   Name: table_name_snapshot_date
   Type: FLUSH
  Table: table_name
 Format: 2
Created: 2018-03-29T16:02:06
  Owner:


10 HFiles (0 in archive), total size 48.8 K (100.00% 48.8 K
shared with the source table)
0 Logs, total size 0
```

Exporting a Snapshot to Amazon S3

Next, use `org.apache.hadoop.HBase.snapshot.ExportSnapshot` to copy the data over to the Apache HBase root directory on Amazon S3.

```
hbase org.apache.hadoop.hbase.snapshot.ExportSnapshot -snapshot
<snapshot_name> -copy-to s3://<HBase_on_S3_root_dir>/
```

As an example, the export of 40TB of data with 4x10GB Direct Connect takes approximately four to five hours.

Data Restore

Creating an empty table

If you are restoring data from a snapshot, first create an empty table and then issue a snapshot restore instead of a snapshot clone. A snapshot clone (`clone_snapshot`) produces an actual copy of the files. A snapshot restore (`restore_snapshot`) creates links to the files copied to the Amazon S3 root directory.

```
hbase shell
hbase(main):001:0> create 'table-name','cf1'
hbase(main):002:0> disable 'table-name'
```

Syncing the EMRFS metadata table with Amazon S3

The first step in restoring a table from a snapshot is to SSH into the Amazon EMR Master and perform an `emrfs sync`. You can also use an Amazon EMR Step to run this command. This command adds the objects in the Hbase root directory to the EMRFS metadata table.

```
emrfs sync s3://{bucket}/{HBase-root}/ --read-consumption
<value> --write-consumption <value>
```

Validating the metadata items "deletionTTL" attribute

Next, run the following command to verify that all entries in the metadata have the "deletionTTL" attribute properly set.

```
emrfs validate-ttl -m <MetadataTableName>
```

If some entries are missing a TTL, the output of the `emrfs validate-ttl` command includes entries that have "MISSING TTL FIELD" messages. In this case, run the following command to populate the TTL for entries that already exist in the metadata table:

```
emrfs populate-ttl -m <MetadataTableName> --read-consumption
<value> --write-consumption <value>
```

> **Note:** Given that there is no traffic or activity on the HBase cluster, the value of `--read-consumption` and `--write-consumption` can match the capacity allocated to the Amazon DynamoDB table that supports the metadata.

Restoring the snapshot from the HBase shell

After creating an empty table and loading all of the objects under the Hbase on Amazon S3 root directory to the EMRFS metadata, you can restore the snapshot.

```
hbase(main):004:0> restore_snapshot 'table-name-snapshot'
hbase(main):005:0> enable 'table-name'
```

Deploying into Production

After you complete the steps in this section, you are ready to migrate the full dataset from your HDFS-backed cluster to HBase on Amazon S3 and restore it to an HBase on Amazon S3 cluster running in your AWS production account.

Preparing Amazon S3 for Production load

Analyze the Amazon Cloudwatch metrics for Amazon S3 captured for the HBase root directory in the development account and confirm the number of requests per Amazon S3 API as noted in the Preparing the Test Environment section.

If you expect a rapid increase in the request rate for the HBase on Amazon S3 root directory bucket in the production account to more than the rates in the Preparing the Test Environment section,open a support case to prepare for the workload and to avoid any temporary limits on your request rate. You do not need to open a support case for request rates lower than those in the Preparing the Test Environment section.

Preparing the Production environment

Follow all the steps in the Preparing the Test Environment to prepare your Production Environment with the configuration settings you have found during the testing phase.

To migrate and restore the full dataset into the production environment, follow the steps in the Migrating and Restoring HBase Tables on HBase on Amazon S3 section.

Managing the Production Environment

Operationalization tasks

Node Decommissioning

When a node is gracefully decommissioned by the YARN Resource Manager (during a user initiated shrink operation or node failures such as bad disk), the regions are first closed and then shut down by the RegionServer. You can also gracefully decommission a RegionServer on any active node by stopping the daemon manually. This step may be required while troubleshooting a particular RegionServer in the cluster.

```
sudo stop hbase-regionserver
```

During shutdown, the RegionServer's Znode expires. The HMaster notices this event and considers that RegionServer as a crashed server. The HMaster then reassigns the regions the RegionServer used to serve to other online RegionServers. Depending on the prefetch settings, the RegionServer warms the cache on the new RegionServer that is now assigned to serve the region.

Rolling Restart

A rolling restart restarts HMaster process on the master node and HRegionServer process on all the core nodes.

Check for any inconsistencies and make sure that the HBase balancer is turned off so that the load balancer does not interfere with region deployments.

Use the shell to disable HBase balancer:

```
hbase(main):001:0> balance_switch false
true
0 row(s) in 0.2970 seconds
```

The following is a sample script that performs a rolling restart on an Apache HBase cluster. This script should be executed on the Amazon EMR Master node that has the Amazon EC2 Key Pair (.pem extension) file to login to the Amazon EMR Core nodes.

```
#!/bin/bash
sudo stop hbase-master; sudo start hbase-master
for node in $(yarn node -list | grep -i ip- | cut -f2 -d: | cut
-f2 -d'G' | xargs) ; do
        ssh -i ~/hadoop.pem -t -o "StrictHostKeyChecking no"
hadoop@$node "sudo stop hbase-regionserver;sudo start hbase-
regionserver"
done
sudo stop hbase-master; sudo start hbase-master #Restart HMaster
again to clear out dead servers list and reenable the balancer
hbase hbck #Run hbck utility to make sure HBase is consistent
```

Cluster resize

Nodes can be added or removed from the HBase clusters on Amazon S3 by performing a resize operation on the cluster. If an AutoScaling policy was set based on a specific CloudWatch metric (such as IsIdle), the resize operation happens based on that policy. All these operations are performed gracefully.

Backup and Restore

With HBase on Amazon S3 you can still consider taking snapshots of your tables every few hours (and deleting them after some days) so you have a point in time recovery option available to you. See also the Running the balancer for specific periods to minimize the impact of region movements on snapshots section.

Cluster termination without data loss

If you want to terminate the current cluster and build a new one on the same Amazon S3 root directory, we recommend that you disable all of the tables in the current cluster. This ensures that all of the data that have not been written to S3 yet are flushed from MemStore cache to Amazon S3 in the form of new store files. To do so, the script below uses an existing script (`/usr/lib/hbase/bin/disable_all_tables.sh`) to disable the tables.

```
#!/bin/bash
clusterID=$(cat /mnt/var/lib/info/job-flow.json | jq -r
".jobFlowId")
#call disable_all_tables.sh
bash /usr/lib/hbase/bin/disable_all_tables.sh
#Store the output of "list" command in a temp file
echo "list" | hbase shell > tableListSummary.txt
#fetch only the list of tables and store it in an another temp
file
tail -1 tableListSummary.txt | tr ',' '\n' | tr -d '"' | tr -d [
| tr -d ] | tr -d ' ' > tableList.txt

#prepare for iteration
while true; do
 while read line; do
    flag="N"
    echo "is_enabled '$line'" | hbase shell > bool.txt
    bool=$(tail -3 bool.txt | head -1)
    if [ "$bool" = "true" ]; then
```

```
        flag="Y"
        break
    fi
 done < tableList.txt
echo "flag: "$flag
if [ "$flag" = "N" ]; then
        aws emr terminate-clusters --cluster-ids $clusterID
        break
else
        echo "Tables aren't disabled yet. Sleeping for 5 seconds
to try again"
fi
sleep 5
done

#cleanup temporary files
rm tableListSummary.txt tableList.txt bool.txt
```

The above script can be place on a file and named `disable_and_terminate.sh`. Note that the script does not exist on the instance. You can add an Amazon EMR step to first copy the script to the instance and then run the step to disable and terminate the cluster. To run the script, you can use the Amazon EMR Step properties below.

```
Name="Disable all tables",Jar="command-
runner.jar",Args=["/bin/bash","/home/hadoop/disable_and_terminat
e.sh"]
```

OS and Apache HBase patching

Similar to AMI upgrades on Amazon EC2, the Amazon EMR service team plans for application upgrades with every new Amazon EMR version release. This removes any OS and Apache HBase patching activities from your team. The latest version of Amazon EMR (5.17.0 as of this paper) runs Apache HBase version 1.4.6. Details of each Amazon EMR version release can be found on Amazon EMR 5.x Release Versions.

Conclusion

This paper includes steps to help you migrate from Apache HBase on HDFS to Apache HBase on Amazon S3 on Amazon EMR. The migration plan provided detailed steps and HBase properties to configure when migrating to HBase on Amazon S3 on Amazon EMR.

Using the various best practices and recommendations highlighted in this whitepaper, we encourage you to test several values for HBase configuration properties so your HBase on Amazon S3 on Amazon EMR cluster supports the performance requirements of your application and use case.

Contributors

The following individuals contributed to this document:

- Francisco Oliveira, Senior Big Data Consultant, Amazon Web Services
- Tony Nguyen, Senior Big Data Consultant, Amazon Web Services
- Veena Vasudevan, Big Data Support Engineer, Amazon Web Services

Further Reading

For additional information, see the following:

- HBase on Amazon S3 Documentation
- Tips for Migrating to Apache HBase on Amazon S3 from HDFS
- Low-Latency Access on Trillions of Records: FINRA's Architecture Using Apache HBase on Amazon EMR with Amazon S3
- Setting up Read Replica Clusters with HBase on Amazon S3
- Use Kerberos Authentication to Integrate Amazon EMR with Microsoft Active Directory

Document Revisions

Date	Description
October 2018	First publication

Appendix A: Command Reference

Restart HBase

Commands to run on the master:

```
sudo stop hbase-master

sudo stop hbase-rest

sudo stop hbase-thrift

sudo stop zookeeper-server

sudo start hbase-master

sudo start hbase-rest

sudo start hbase-thrift

sudo start zookeeper-server
```

Commands to run in all core nodes

```
sudo stop hbase-regionserver

sudo start hbase-regionserver
```

EMRFS TTL sub-commands

emrfs populate-ttl [options]

Populate TTL attribute for tombstoned entries.
Options:
-m <value> | --metadata-name <value>
name of the metadata
--read-consumption <value>
amount of dynamodb read capacity to consume, limited by table maximum
--write-consumption <value>
amount of dynamodb write capacity to consume, limited by table maximum

emrfs validate-ttl [options]

Validate that TTL attributes are set correctly. This command will go through the EMRFS Amazon DynamoDB table and validate that the deletionTTL is set properly.
Options:
-m <value> | --metadata-name <value>
name of the metadata

Appendix B: AWS IAM Policy Reference

The policies that follow are annotated with comments - remove the comments prior to use.

Minimal EMR Service Role Policy

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Resource": "*",
            "Action": [
                "ec2:AuthorizeSecurityGroupEgress",
                "ec2:AuthorizeSecurityGroupIngress",
                "ec2:CancelSpotInstanceRequests",
                "ec2:CreateNetworkInterface",
                "ec2:CreateSecurityGroup",
                "ec2:CreateTags",
                "ec2:DeleteNetworkInterface", // This is only needed if you are
launching clusters in a private subnet.
                "ec2:DeleteTags",
                "ec2:DeleteSecurityGroup", // This is only needed if you are
using Amazon managed security groups for private subnets. You can omit this
action if you are using custom security groups.
                "ec2:DescribeAvailabilityZones",
                "ec2:DescribeAccountAttributes",
                "ec2:DescribeDhcpOptions",
                "ec2:DescribeImages",
                "ec2:DescribeInstanceStatus",
                "ec2:DescribeInstances",
                "ec2:DescribeKeyPairs",
                "ec2:DescribeNetworkAcls",
                "ec2:DescribeNetworkInterfaces",
                "ec2:DescribePrefixLists",
                "ec2:DescribeRouteTables",
                "ec2:DescribeSecurityGroups",
                "ec2:DescribeSpotInstanceRequests",
```

```
                "ec2:DescribeSpotPriceHistory",
                "ec2:DescribeSubnets",
                "ec2:DescribeTags",
                "ec2:DescribeVpcAttribute",
                "ec2:DescribeVpcEndpoints",
                "ec2:DescribeVpcEndpointServices",
                "ec2:DescribeVpcs",
                "ec2:DetachNetworkInterface",
                "ec2:ModifyImageAttribute",
                "ec2:ModifyInstanceAttribute",
                "ec2:RequestSpotInstances",
                "ec2:RevokeSecurityGroupEgress",
                "ec2:RunInstances",
                "ec2:TerminateInstances",
                "ec2:DeleteVolume",
                "ec2:DescribeVolumeStatus",
                "ec2:DescribeVolumes",
                "ec2:DetachVolume",
                "iam:GetRole",
                "iam:GetRolePolicy",
                "iam:ListInstanceProfiles",
                "iam:ListRolePolicies",
                "s3:CreateBucket",
                "sdb:BatchPutAttributes",
                "sdb:Select",
                "cloudwatch:PutMetricAlarm",
                "cloudwatch:DescribeAlarms",
                "cloudwatch:DeleteAlarms",
                "application-autoscaling:RegisterScalableTarget",
                "application-autoscaling:DeregisterScalableTarget",
                "application-autoscaling:PutScalingPolicy",
                "application-autoscaling:DeleteScalingPolicy",
                "application-autoscaling:Describe*"
            ]
        },
        {
            "Effect": "Allow",
            "Resource":
["arn:aws:s3:::examplebucket/*","arn:aws:s3:::examplebucket2/*"], // Here you
```

```
can specify the list of buckets which are going to be storing cluster logs,
bootstrap action script, custom JAR files, input & output paths for EMR steps
            "Action": [
                "s3:GetBucketLocation",
                "s3:GetBucketCORS",
                "s3:GetObjectVersionForReplication",
                "s3:GetObject",
                "s3:GetBucketTagging",
                "s3:GetObjectVersion",
                "s3:GetObjectTagging",
                "s3:ListMultipartUploadParts",
                "s3:ListBucketByTags",
                "s3:ListBucket",
                "s3:ListObjects",
                "s3:ListBucketMultipartUploads"
            ]
        },
        {
            "Effect": "Allow",
            "Resource": "arn:aws:sqs:*:123456789012:AWS-ElasticMapReduce-*", //
This will allow EMR to only perform actions (Creating queue, receiving messages,
deleting queue, etc) on SQS queues whose names are prefixed with the literal
string AWS-ElasticMapReduce-
            "Action": [
                "sqs:CreateQueue",
                "sqs:DeleteQueue",
                "sqs:DeleteMessage",
                "sqs:DeleteMessageBatch",
                "sqs:GetQueueAttributes",
                "sqs:GetQueueUrl",
                "sqs:PurgeQueue",
                "sqs:ReceiveMessage"
            ]
        },
        {
            "Effect": "Allow",
            "Action": "iam:CreateServiceLinkedRole",  // EMR needs permissions
to create this service-linked role for launching EC2 spot instances
            "Resource": "arn:aws:iam::*:role/aws-service-
```

```
role/spot.amazonaws.com/AWSServiceRoleForEC2Spot*",
            "Condition": {
                "StringLike": {
                    "iam:AWSServiceName": "spot.amazonaws.com"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": "iam:PassRole", // We are passing the custom EC2 instance
profile (defined below) which has bare minimum permissions
            "Resource": [
                "arn:aws:iam::*:role/Custom_EMR_EC2_role",
                "arn:aws:iam::*:role/EMR_AutoScaling_DefaultRole"
            ]
        }
    ]
}
```

Minimal Amazon EMR Role for Amazon EC2 (Instance Profile) Policy

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Resource": "*",
            "Action": [
                "ec2:Describe*",
                "elasticmapreduce:Describe*",
                "elasticmapreduce:ListBootstrapActions",
                "elasticmapreduce:ListClusters",
                "elasticmapreduce:ListInstanceGroups",
                "elasticmapreduce:ListInstances",
                "elasticmapreduce:ListSteps"
            ]
        },
        {
```

```
            "Effect": "Allow",
            "Resource": [    // Here you can specify the list of buckets which
are going to be accessed by applications (Spark, Hive, etc) running on the nodes
of the cluster
                "arn:aws:s3:::examplebucket1/*",
                "arn:aws:s3:::examplebucket1*",
                "arn:aws:s3:::examplebucket2/*",
                "arn:aws:s3:::examplebucket2*"
            ],
            "Action": [
                "s3:GetBucketLocation",
                "s3:GetBucketCORS",
                "s3:GetObjectVersionForReplication",
                "s3:GetObject",
                "s3:GetBucketTagging",
                "s3:GetObjectVersion",
                "s3:GetObjectTagging",
                "s3:ListMultipartUploadParts",
                "s3:ListBucketByTags",
                "s3:ListBucket",
                "s3:ListObjects",
                "s3:ListBucketMultipartUploads",
                "s3:PutObject",
                "s3:PutObjectTagging",
                "s3:HeadBucket",
                "s3:DeleteObject"
            ]
        },
        {
            "Effect": "Allow",
            "Resource": "arn:aws:sqs:*:123456789012:AWS-ElasticMapReduce-*", //
This will allow EMR to only perform actions (Creating queue, receiving messages,
deleting queue, etc) on SQS queues whose names are prefixed with the literal
string AWS-ElasticMapReduce-
            "Action": [
                "sqs:CreateQueue",
                "sqs:DeleteQueue",
                "sqs:DeleteMessage",
                "sqs:DeleteMessageBatch",
```

```
                "sqs:GetQueueAttributes",
                "sqs:GetQueueUrl",
                "sqs:PurgeQueue",
                "sqs:ReceiveMessage"
            ]
        }
    ]
}
```

Minimal Role Policy for User Launching Amazon EMR Clusters

```
// This policy can be attached to an AWS IAM user who will be launching EMR
clusters. It provides minimum access to the user to launch, monitor and
terminate EMR clusters

{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Statement1",
            "Effect": "Allow",
            "Action": "iam:CreateServiceLinkedRole",
            "Resource": "*",
            "Condition": {
                "StringLike": {
                    "iam:AWSServiceName": [
                        "elasticmapreduce.amazonaws.com",
                        "elasticmapreduce.amazonaws.com.cn"
                    ]
                }
            }
        },
        {
            "Sid": "Statement2",
            "Effect": "Allow",
            "Action": [
                "iam:GetPolicyVersion",
                "ec2:AuthorizeSecurityGroupIngress",
```

```
                "ec2:DescribeInstances",
                "ec2:RequestSpotInstances",
                "ec2:DeleteTags",
                "ec2:DescribeSpotInstanceRequests",
                "ec2:ModifyImageAttribute",
                "cloudwatch:GetMetricData",
                "cloudwatch:GetMetricStatistics",
                "cloudwatch:ListMetrics",
                "ec2:DescribeVpcAttribute",
                "ec2:DescribeSpotPriceHistory",
                "ec2:DescribeAvailabilityZones",
                "ec2:CreateRoute",
                "ec2:RevokeSecurityGroupEgress",
                "ec2:CreateSecurityGroup",
                "ec2:DescribeAccountAttributes",
                "ec2:ModifyInstanceAttribute",
                "ec2:DescribeKeyPairs",
                "ec2:DescribeNetworkAcls",
                "ec2:DescribeRouteTables",
                "ec2:AuthorizeSecurityGroupEgress",
                "ec2:TerminateInstances", //This action can be scoped in similar
manner like it has been done below for "elasticmapreduce:TerminateJobFlows"
                "iam:GetPolicy",
                "ec2:CreateTags",
                "ec2:DeleteRoute",
                "iam:ListRoles",
                "ec2:RunInstances",
                "ec2:DescribeSecurityGroups",
                "ec2:CancelSpotInstanceRequests",
                "ec2:CreateVpcEndpoint",
                "ec2:DescribeVpcs",
                "ec2:DescribeSubnets",
                "elasticmapreduce:*"
            ],
            "Resource": "*"
        },
        {
            "Sid": "Statement3",
            "Effect": "Allow",
```

```
            "Action": [
                "elasticmapreduce:TerminateJobFlows"
            ],
            "Resource":"*",
            "Condition": {
                "StringEquals": {
                  "elasticmapreduce:ResourceTag/custom_key": "custom_value"  //
Here you can specify the key value pair of your custom tag so that this IAM user
can only delete the clusters which are appropriately tagged by the user
                }
              }
        },
        {
            "Sid": "Statement4",
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": [
                "arn:aws:iam::*:role/Custom_EMR_Role",
                "arn:aws:iam::*:role/Custom_EMR_EC2_role",
                "arn:aws:iam::*:role/EMR_AutoScaling_DefaultRole"
            ]
        }
    ]
}
```

Appendix C: Transparent Encryption Reference

To configure Transparent Encryption, use the following EMR Configuration JSON:

```
[{"classification":"hdfs-encryption-
zones","properties":{"/user/hbase":"hbase-key"}}]
```

In addition to the classification above, you must disable HDFS Opensource Security. By default, EMR Security Configurations for at-rest Encryption for Local Disks tie Open-source HDFS Encryption with LUKs encryption.

If you need to configure Transparent Encryption and your application is latency sensitive, do not enable at-rest encryption via EMR Security Configuration. You can configure LUKS via a bootstrap action.

To check that WALs are being encrypted, use the following commands:

```
sudo –u hdfs hdfs dfs -ls /user/HBase/WAL/ip-xx-xx-x-
xx.ec2.internal,16020,1520373175110

sudo –u hdfs hdfs crypto -getFileEncryptionInfo  -path
/user/HBase/WAL/WALs/ip-xx-xx-x-
xx.ec2.internal,16020,1520373175110/ip-xx-xx-x-
xx.ec2.internal%2C16020%2C1520373175110.1520373184129
```

To verify that the oldWALs are being encrypted, the output to the last command should be the following:

```
{cipherSuite: {name: AES/CTR/NoPadding, algorithmBlockSize: 16},
cryptoProtocolVersion:
CryptoProtocolVersion{description='Encryption zones', version=2,
unknownValue=null}, edek:
7c3c2fcf8337f14bbf815697686de5a696c6670c0f41eb71678b53ee5326c33e
```

```
, iv: eac6cf91bdd2eee8496f1ddb19b4fcf8, keyName: HBase-key,
ezKeyVersionName: hbase-key@0}
```

> **Note:** The default configurations grant access to the DECRYPT_EEK operation on all keys (/etc/hadoop-kms/conf/kms-acls.xml).

For more details, see Transparent Encryption in HDFS on Amazon EMR and Transparent Encryption in HDFS.