MarkLogic on AWS

Rahul Bhartia

December 2014

Contents

Abstract

Amazon Web Services (AWS) is a flexible, cost-effective, easy-to-use cloud computing platform. Running your own NoSQL data store on Amazon Elastic Compute Cloud (EC2) is the ideal scenario for users whose applications require the unique benefits of NoSQL systems. NoSQL systems are some of the most widely deployed software packages in the AWS Cloud.

This whitepaper is intended to help you understand one of the popular NoSQL options available with the AWS Cloud computing platform—the Enterprise NoSQL database MarkLogic. It also provides an overview of general best practices and examines important MarkLogic implementation characteristics such as performance, durability, and security with particular attention to identifying features that support scalability, high-availability, and fault-tolerance.

Introduction

MarkLogic Corporation developed MarkLogic Server (MarkLogic), an Enterprise NoSQL database platform that is fully scalable and provides ACID (Atomicity, Consistency, Isolation, and Durability) transactions for large datasets. MarkLogic is architected to support the following features:

- Native XML, JSON, RDF, and geospatial data storage
- Schema-agnostic, flexible data model
- Built-in search
- Scalability and elasticity
- ACID transactions
- High availability and disaster recovery
- Government-grade security
- Monitoring and management tools
- Hadoop integration

MarkLogic is the NoSQL platform of choice for many mission-critical, customer-facing operational and analytic applications in large enterprises including investment banking, healthcare, and US Federal Government systems.

Tips for Using MarkLogic on Amazon EC2

- Use 64-bit Amazon EC2 instances; there is no 32-bit version of MarkLogic.
- As a fully ACID-compliant transactional data store, MarkLogic requires substantial input/output (I/O) throughput. AWS provides many ways to support the I/O throughput requirements. The General Purpose (SSD) Amazon Elastic Block Store (EBS) volume type provides good performance along with data durability and should be the first choice for storage.
- MarkLogic supports Amazon Simple Storage Service (S3) as a data store. Because Amazon S3 does not currently support "append," it cannot be used with journaling and is not appropriate as primary storage. However, for archival copies, Amazon S3 is ideal because data can be queried and accessed directly, without needing to restore to another file system.
- The MarkLogic AWS CloudFormation templates provide patterns for production deployments. Use the AWS CloudFormation templates in conjunction with DevOps tools like Ansible, Puppet, or Chef to manage configuration and deployment.
- Although storage-optimized instance types offer extremely good I/O throughput, the storage is ephemeral. Because of the risk of data loss should a host fail, we do not currently recommend using ephemeral volumes for forest data, unless data loss is an acceptable risk for the application.
- The M3 general purpose series Amazon EC2 instance types are good fits for MarkLogic deployments, as are the compute-optimized C3 series instances. For applications requiring large numbers of range indexes or semantics triple store use cases, the R3 memory-optimized instance types may also be appropriate.
- Using forest replication on Amazon EBS-optimized instance types with Amazon EBS General Purpose SSD (gp2) will provide excellent overall performance as well as high availability in case of failure. On smaller instance types, attach two forests per host, one primary and one replica, each forest on its own Amazon EBS volume.
- Use enhanced networking on the instance, if available.
- Use Database Replication for in-region replication to secondary MarkLogic clusters in different Availability Zones to help create a highly available infrastructure. MarkLogic clusters should not normally be distributed across regional boundaries.
- On Linux Amazon Machine Images (AMIs), the noop scheduler is the preferred I/O scheduler. If you are using supplied AWS CloudFormation templates to deploy, this configuration is done automatically for you; otherwise, you will need to set the scheduler yourself after installation.

Installing MarkLogic on AWS

These instructions assume that you already have an Amazon Web Services account. (If you don't have an AWS account you can go to aws.amazon.com to sign up for one.)

1. Go to the AWS Marketplace and search for MarkLogic.
2. In the MarkLogic project page, click the Accept Terms button.

   IMPORTANT: Do not click Launch Instance at this point.
3. Create a new AWS Identity and Access Management (IAM) role with resource creation privileges to use with your instance.
4. Generate a key pair if you don't already have one.
5. Create an Amazon Simple Notification Service (SNS) topic to monitor messages from the AWS CloudFormation template deployment process.
6. Create an AWS CloudFormation stack using the provided AWS CloudFormation template.

   The following parameters are associated with the AWS CloudFormation templates. You can review and proceed with defaults or make customizations as needed:

Parameter Name	Description
AdminUser	The username you want to use to log in as the MarkLogic Administrator.
AdminPass	The password you want to use to log in as the MarkLogic Administrator.
IAMRole	Name of the AWS IAM role you created in step 3.
Zone1 Zone2 Zone3	The Availability Zones on which to create each host in the cluster. Each Availability Zone in your cluster should be in the same AWS region, such as us-east or us-west. The possible Availability Zones for the us-east region are: us-east-1a us-east-1b us-east-1c us-east-1d us-east-1e The possible Availability Zones for the us-west region are: us-west-2a us-west-2b us-west-2c The possible Availability Zones for the eu-west region are:

Parameter Name	Description
	eu-west-1a eu-west-1b eu-west-1c
VolumeSize	The initial Amazon EBS volume size (GB). The range of valid values is 1 - 1000. The default is 10.
NodesPerZone	The number of nodes (hosts) to create for each zone. For example, a value of 1 will create one node for each zone, a total of three nodes for the cluster. A value of 0 will pause all nodes.
LogSNS	The Amazon SNS notification needed for logging. Enter the entire topic Amazon Resource Name (ARN) as it appears in the Amazon SNS dashboard.
InstanceType	The type of Amazon EC2 instance to launch.
KeyName	Name of the key pair to use with this instance.
Licensee	Enter `none` unless you are bringing your own license; you need to fill out Licensee per your license keys. This is left blank if you are running the Marketplace AMIs with the Marketplace License.
LicenseKey	Enter `none` unless you are bringing your own license; you need to fill out Licensee per your license keys. This is left blank if you are running the Marketplace AMIs with the Marketplace License.

The AWS CloudFormation templates create all the resources for the stack. The last items to come up are the instances themselves. After the operating system (OS) boots, the MarkLogic cluster is formed. It will take 5-10 minutes typically for the process to complete. On the Outputs tab of the AWS CloudFormation administration interface, there will be a Load Balancer URL. This can be used to attach to the cluster (albeit not to a particular node). Log in using the username and password you used as parameters for the template.

For more detailed instructions on the installation process, go to the MarkLogic Server on Amazon EC2 Guide.

Architecture

The design of your MarkLogic installation on Amazon EC2 is largely dependent on the scale at which you're trying to operate. To gain the benefits of high availability and scalability, the best practice is to run MarkLogic as a cluster. The MarkLogic AWS CloudFormation templates provide the easiest way to make use of the ***Managed Clusters*** feature-set, the benefits of which include:

- Automatic handling of changes in Amazon EC2 instance hostnames.

- Automatic placement and management of data, to insure that nothing is lost on transient volumes.
- Configuration of best-practice security, load balancing, routing, and handling of volatile instances.
- Automatic restarting and reconfiguring of failed or unhealthy nodes in a cluster.
- Pausing of entire clusters, while maintaining all configuration and data.
- Simplified re-configuration of cluster instance types and topologies.

We recommend a minimum of three instances, but the actual sizing will depend on the overall amount of data that you want to store and on your performance requirements.

As demand for your application increases, you can add instances to your cluster to cope with additional RAM, CPU, or I/O loads. You can expand your MarkLogic cluster without taking it offline by using the online rebalance operation, as described in the Expanding Your MarkLogic Cluster section.

Building Blocks and Definitions

Forests

A forest is a collection of documents implemented as a filesystem directory on a storage volume. Each forest holds a set of documents and all their indexes. Forests are created on hosts and attached to databases to appear as a contiguous set of content for query purposes. A forest can only be attached to one database at a time. You cannot load data into a forest that is not attached to a database.

A forest contains in-memory and on-disk structures called *stands*. Each stand is composed of self-consistent XML, binary, and/or text fragments, stored in document order. As new stands are written to disk, MarkLogic merges older stands to optimize performance.

A single instance might manage several forests. Forests are queried in parallel, so placing more forests on a multi-core instance can help with both update and query concurrency. Depending on the workload, you might have one forest corresponding to every physical core on an instance, or you might have a forest for every two cores, with each forest holding millions or tens of millions of documents and 100 to 500 gigabytes. In a clustered environment, you can have a set of servers, each managing their own set of forests, all unified into a single database.

Host Functions: D-Nodes and E-Nodes

There are two functions a host in a MarkLogic cluster can perform.

- Evaluator node (e-node)

- Data node (d-node)

E-nodes evaluate requests. If the request does not need any forest data to complete, then an e-node request is evaluated entirely on the e-node. If the request needs forest data (for example, a document in a database), then it communicates with one or more d-nodes to service the forest data. After it gets the content back from the d-node, the e-node finishes processing the request (performs the filter portion of query processing, if specified by the query) and sends the results to the application. E-nodes retain forest data in a cache to speed up subsequent operations utilizing the same data.

D-nodes are responsible for maintaining transactional integrity during insert, update, and delete operations. This transactional integrity includes forest journaling, forest recovery, backup operations, and on-disk forest management. D-nodes are also responsible for providing forest optimization (merges), index maintenance, and content retrieval. D-nodes service e-nodes when the e-nodes require content returned from a forest. A d-node gets the communication from an e-node then sends the results of the index resolution back to the e-node. The d-node part of the request includes the index resolution portion of query processing. Also, each d-node performs the work needed for merges for any forests hosted on that node.

In single instance configurations, both e- and d-node activities are carried out by a single host. In a cluster, it is also possible for some or all of the hosts to have shared e-node and d-node duties. For smaller clusters, it makes sense to have all hosts perform both e- and d-node functions. By default, every MarkLogic host acts as both an e-node and a d-node. As systems are scaled for performance, the roles can be separated out to different hosts.

Caches

There are several different caches involved in MarkLogic. The default behavior is to have each node act as both an e- and d-node, so the default caches are fine for initial configuration. However, when separating nodes into d- and e-nodes, it is important to adjust cache sizes accordingly.

List cache: This cache contains termlist data for all on-disk stands. Termlists are used to resolve queries against the database.

Compressed tree cache: When data is sent from a forest in response to a query request, that data is first sent as a compressed tree. The compressed tree cache contains the compressed tree data for on-disk stands.

Expanded tree cache: When a document is returned to a client, the document is transformed from its compressed form into an expanded tree form; at that time, the document is stored in the expanded tree cache.

Triple cache: This cache stores RDF triples data for on-disk stands.

Replicas and Failover

To provide high availability for MarkLogic, each forest in the database allows you to create replicas that are copies of the primary forest. These replicas become active should a primary node fail. To configure an environment for high availability, use a cluster (minimum of three nodes) and forest-level replication.

The replicas contain the same data as the primary forest, and are kept up to date transactionally as updates to the forest occur. Each replica forest must be on a different host from the primary forest so that if the host for the primary forest goes down, another host with a copy of the primary forest's data can take over.

Each replica needs adequate I/O bandwidth. Multiple replicas can reside on the same EBS volume as long as it is provisioned properly for adequate I/O bandwidth.

Database Replication for Disaster Recovery

Database replication enables replication of data from one cluster to another. Replication is configured in one direction only. When configuring replication, whether between clusters within Amazon EC2, or between Amazon EC2 and non-Amazon EC2 clusters, care should be taken with the network configuration and port access. Information replicated between clusters can be encrypted using SSL.

RAM Recommendations

MarkLogic uses a number of caches to optimize performance and reduce storage volume I/O. A general rule of thumb for RAM recommendations follows:

- One-half the RAM for caches and queries
- One-half the RAM for forests

Storage and I/O

You should ensure that the provisioned size of the storage volumes for your data is sufficient to store the documents and indexes for each forest assigned to your instance. You can measure forest and index size by viewing the Database Status page in the Administration Console, or by viewing the real-time metrics in the Monitoring Console. See the Administrator's Guide and the Monitoring Guide for more details.

The I/O throughput an application requires is dependent on the quantity and complexity of the queries and on the number of queries executed against the server. The Sizing section of this paper will help you estimate the throughput needed for an application.

Both the document data and the index information are written to storage in an append-only format. Changes to the documents cause the old documents to be marked for deletion. At a later time, MarkLogic will run merges to clean up old deleted fragments and combine stands. Merges are also a way of self-tuning the performance of the system, and MarkLogic periodically assesses the state of each database to see if it would benefit from self-tuning through a merge. In most cases, the default merge settings and the dynamic nature of merges will keep the database tuned optimally at all times. Because merges can be resource intensive (both I/O and CPU), they can be scheduled to occur during specific times.

To help prevent I/O throughput from impairing performance, you need enough throughput to accommodate queries as well as updates and merges. Operational activities such as backup/restore and database replication will require additional I/O resources.

CPU Requirements

When forests are stored on local storage, CPU requirements can change based upon the number of forests per host. Forests are in most ways independent from each other, so with multi-core systems you can often benefit by having more forests (up to the number of cores that you have) to get more parallel execution.

When using Amazon EBS General Purpose (SSD) volumes it is likely that the network will be saturated before CPU resources are exhausted, but the limit of the network varies by instance type. Therefore, it is generally preferable to have multiple forests per instance.

With Amazon EBS volumes in play, CPU becomes much more important during queries because forests can be queried in parallel by different threads running on different cores. CPU will be driven by query load—more queries and more application server threads will require more CPU.

Network Configuration

For production applications, we recommend the use of Amazon Virtual Private Cloud (VPC) instead of Amazon EC2 Classic. Amazon VPC allows you to isolate your application in your own network configuration. Amazon VPC can support complex networking configurations, and we recommend that you follow the AWS best practices on configuring your Amazon VPC.

In general, the purpose of your Amazon VPC should be to isolate the different application layers into network topologies that are appropriate for your use case. Instead of using publicly available IP addresses, Amazon VPCs can be configured with subnets and routing including network address translation (NAT). Amazon VPC configuration should be dictated by your data center best practices.

For communication between instances within the cluster, you must have opened the appropriate ports for communication by using a suitable security group.

If your clients are outside Amazon EC2 or Amazon VPC, you must configure each Amazon EC2 instance with a public IP address and then use this IP address to identify and add each instance to the cluster. MarkLogic exchanges the cluster map by using the registered IP address. You can combine the public IP address with a DNS solution such as Amazon Route 53 to provide a convenient name for your cluster instances.

For administration, particularly if you want to use the MarkLogic Administration Console, you will need to use the public DNS address or enable a public IP address on one or more instances within your cluster. You must also ensure that your security group is configured to allow communication on the administration port (8001) and the intra-cluster communication port (7999) so the administration client can communicate with your Amazon EC2 instances and all Amazon EC2 instances within the cluster can communicate with each other.

If you use the AWS CloudFormation templates to deploy your cluster, you'll also have an Elastic Load Balancing (ELB) service load balancer public address that clients (either internal or external) can use to access the cluster. Requests will be load balanced across all nodes in the cluster (or all e-nodes, in the case of e/d splits).

Sizing

Measuring and predicting the size of your dataset is critical to understanding the optimum configuration for your cluster. There are some basic figures that you can use to help provide a rough estimate of your cluster size and configuration requirements.

- **Storage space**. MarkLogic uses a set of internal and configurable indexes to quickly resolve searches and other operations. Some combinations of index settings can result in data expansion where indexed documents inside MarkLogic take up more space than the raw data does on storage volumes. Other combinations may result in storage that is close to, or potentially less than, the raw data. Because index settings can influence document size, it is important to sample data inside MarkLogic with appropriate index settings to determine average document storage size. After the average document size has been determined, multiply the average size with the total expected number of documents. MarkLogic is an append-only system, offering significant performance enhancements over traditional databases. Periodically MarkLogic cleans up deleted documents and merges multiple stands; this process typically requires 1x the storage space + 64 GB per forest. Therefore, the total storage is the average doc size x total number of documents + 64 GB x total number of forests.
- **RAM requirements**. Ingest a sample dataset into MarkLogic with appropriate index settings, and then measure the average storage size of each document. Multiply this with the total number of expected documents, and you will have the total RAM required for forests. This represents 1/3 of the total RAM required, so multiply by 3 to get the total RAM.

- **I/O operations**. Balancing I/O cost and performance requires understanding the drivers and schedule of your loads. Each write must be journaled—you can only write to the database as fast as your I/O. Reads might be cached so more RAM will help, but you still need to consider how long warming the cache will take and what the impact will be of cache misses on your user's queries. Merges can be throttled or even deferred, but only up to a point, and then merge activity will move to the foreground and slow down reads and writes.

As an example, let's say you want to store 20,000,000 documents total. In general, try to sample at least 1% of the content; so take a sample of 200,000 documents to use as a basis for estimating the total storage space and RAM required. After loading the 200,000 documents into a single forest, first validate that you've loaded 200,000 documents by hitting the forest counts management URL ({host} should be your host and {forest-name} should be your forest name):

http://{host}:8002/manage/v2/forests/{forest-name}?view=counts

Then, view the in-memory and on-disk size of the forest by hitting this management URL:

http://{host}:8002/manage/v2/forests/{forest-name}?view=status

To return the status formatted as XML or JSON, simply append `&format=xml` or `&format=json` to the end of the URL.

Here's example output from the view status for a forest:

```
<stand>
     <stand-id>17436730133011644403</stand-id>
     <path>/var/opt/volume1/0000001</path>
     <stand-kind units="enum">Active</stand-kind>
     <is-fast units="bool">false</is-fast>
     <label-version units="quantity">50397184</label-version>
     <disk-size units="MB">2820</disk-size>
     <memory-size units="MB">45</memory-size>
     <list-cache-hits units="quantity">35044</list-cache-hits>
     <list-cache-misses units="quantity">188172</list-cache-misses>
     <list-cache-hit-rate units="fraction">0</list-cache-hit-rate>
     <list-cache-miss-rate units="fraction">0</list-cache-miss-rate>
     <compressed-tree-cache-hits units="quantity">155316</compressed-tree-
cache-hits>
     <compressed-tree-cache-misses units="quantity">32663</compressed-
tree-cache-misses>
     <compressed-tree-cache-hit-rate units="fraction">0</compressed-tree-
cache-hit-rate>
     <compressed-tree-cache-miss-rate units="fraction">0</compressed-tree-
cache-miss-rate>
```

```
     <triple-cache-hits units="quantity">0</triple-cache-hits>
     <triple-cache-misses units="quantity">0</triple-cache-misses>
     <triple-cache-hit-rate units="fraction">0</triple-cache-hit-rate>
     <triple-cache-miss-rate units="fraction">0</triple-cache-miss-rate>
     <triple-value-cache-hits units="quantity">0</triple-value-cache-hits>
     <triple-value-cache-misses units="quantity">0</triple-value-cache-
misses>
     <triple-value-cache-hit-rate units="fraction">0</triple-value-cache-
hit-rate>
     <triple-value-cache-miss-rate units="fraction">0</triple-value-cache-
miss-rate>
</stand>
```

This forest contains 200,000 documents, with a total storage size of 2820 MB (from the disk-size element) and a total memory size of 45 MB (from the memory-size element). So the average document size is:

2820M / 200,000 = 0.0141 MB x 1024 = 14.4 KB

The average in-memory size is:

45M / 200,000 = 0.000225 MB x 1024 = 0.2304 x 1024 = 236 B

Next, we extrapolate out to our 20,000,000 documents. So, for storage, we'll need:

0.0141 x 20,000,000 = 282,000 MB x 1.5 = 423,000 / 1024 = ~414 GB

And, for memory, we'll need:

0.000225M x 20,000,000 = 4,500 MB / 1024 = ~5 GB x 3 = ~15 GB

Now, you now know roughly what you need in terms of overall storage for the database (about half a terabyte) and the overall memory (about 15 GB). You can then split this up into multiple instances and choose appropriate instance types depending upon your requirements.

Production Designs

This section identifies common best practices for designing production MarkLogic implementations.

Given the building blocks discussed in the previous section, how would a system scale to accommodate growing load over time?

Scaling out is an excellent way to add capacity as your data volume and/or request load grows. The following reference architectures represent good starting points;

Small scale: Use Amazon EBS volumes and Amazon EBS-optimized instance types. Amazon EBS has significant write cache, superior random I/O performance (which MarkLogic uses for reads), and provides enhanced durability in comparison to the ephemeral disks on non-SSD instance types. The use of smaller Amazon EBS General Purpose (SSD) volumes is a good fit for applications that don't require very high peak I/O. Amazon EBS-optimized instances provide dedicated throughput to Amazon EBS that will smooth out any response time spikes that might occur without the dedicated pipe. If you're going to use ephemeral disks, be sure to have multiple copies of forests to enhance operational durability. Remember, if the instance is stopped, fails, or is terminated, the ephemeral disks are gone, and so is your data.

For production scale applications, we recommend a minimum of 4 vCPU, so the m3.xlarge or c3.xlarge Amazon EC2 instance types would be a good fit. Depending upon the volume of data, a single node might be adequate. In that case, using database replication to copy data to a second node might be a good option from a disaster recovery/ high availability perspective. Using Amazon S3 as a backup location might also be an acceptable plan, depending upon recovery time objectives. It might also be an option to do frequent Amazon EBS volume snapshots, which would then be available for recovery as needed.

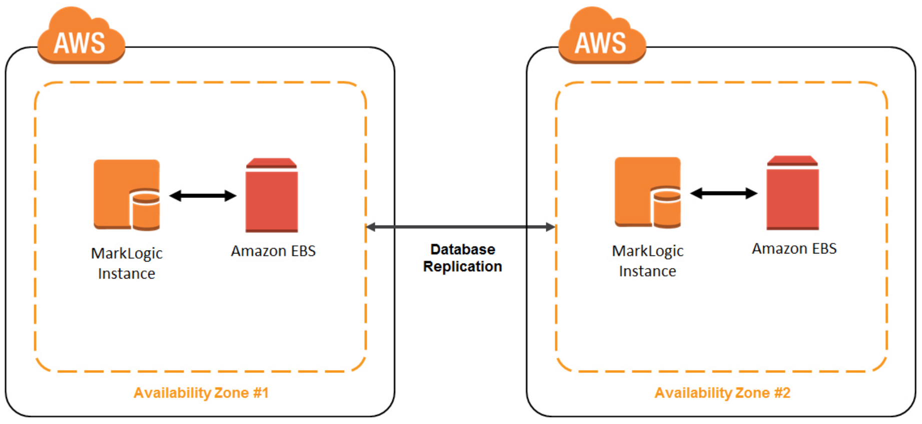

Figure 1: Small scale—single node with Amazon EBS and database replication

Medium scale: At medium scale, we recommend a three-node MarkLogic cluster with shared e- and d-nodes. From a storage perspective, medium scale deployments should use Amazon EBS-optimized instance types and Amazon EBS General Purpose (SSD) volumes. Assuming the need for high availability, each node should have a single primary and single replica forest, each on its own Amazon EBS volume.

Depending upon the query volume, it might make sense at this point to move up to 8 vCPU instances (m3.2xlarge, c3.2xlarge), but 4 vCPU instances might also work here (m3.xlarge or c3.xlarge).

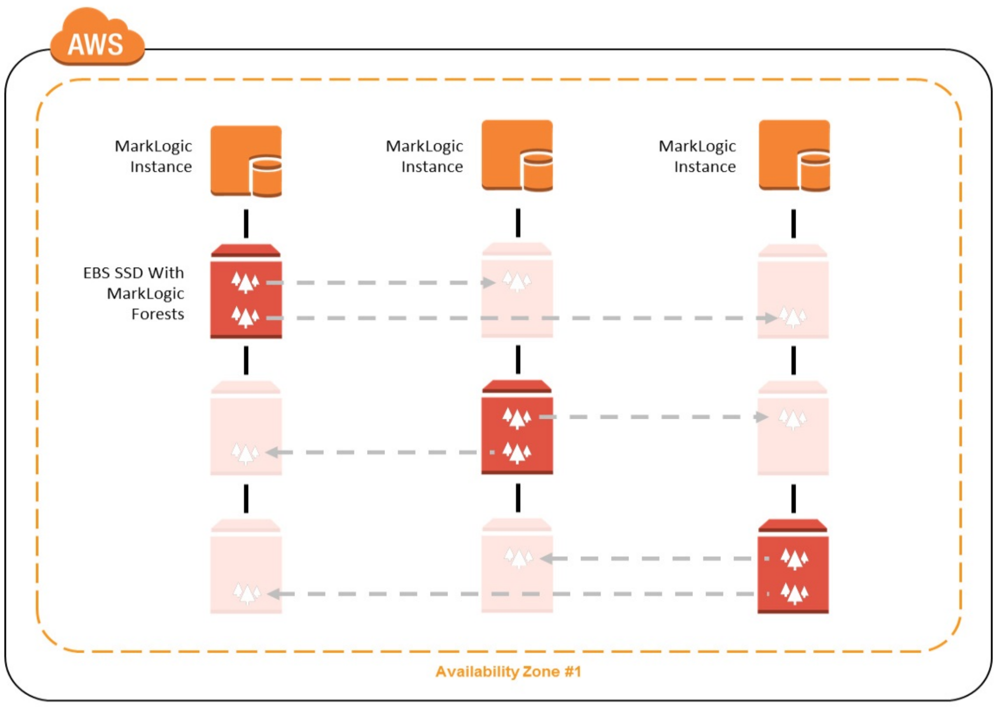

Figure 2: Medium scale—3 nodes with local disk failover, multiple Amazon EBS volumes

Large scale: At this point, depending upon the characteristics of your application, it may make sense to separate data and evaluation nodes. After you reach between 8 to 16 nodes, having a set of dedicated e-nodes to handle incoming queries helps separate workloads. In addition, separating hosts into e- and d-nodes means that you can make more effective use of caches and potentially gain better performance by increasing MarkLogic's list and expanded tree cache. Having more of the search index in memory will result in better overall performance. The cache hit ratios are available from the MarkLogic monitoring application, typically found at http://{host}:8002/history on the host where MarkLogic is running. If you have a cache hit ratio that is low along with a high query volume, it might make sense to separate out to e- and d-nodes and then increase cache sizes appropriately.

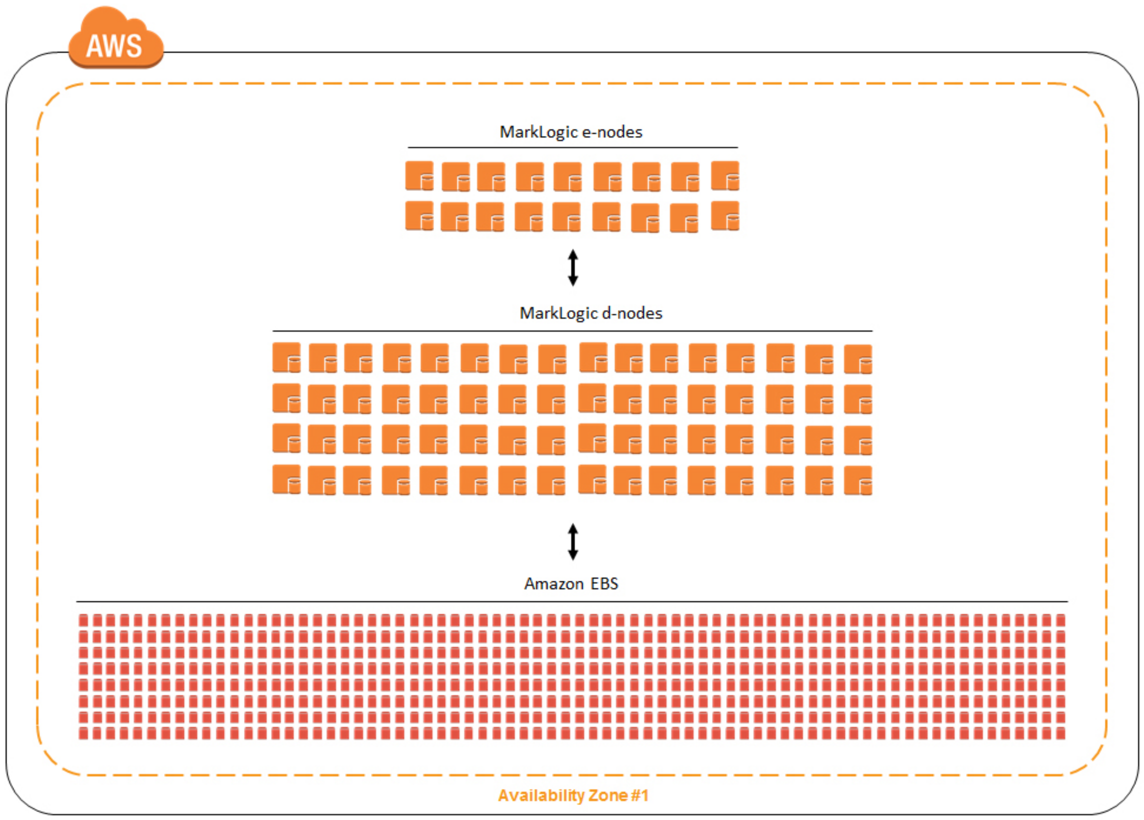

Figure 3: Large scale—Support for 100 TBs with 18 e-nodes, 48 d-nodes, and 576 Amazon EBS volumes

When nodes are separated into e- and d-nodes, the following cache changes should be made.

D-nodes:

- Decrease expanded tree cache to a minimal number; d-nodes host data and don't need the expanded tree cache.
- Increase list cache. The increased list cache will improve query performance. Use some of the reclaimed memory from the expanded tree cache to increase the list cache.
- Increase compressed tree cache. The data nodes will use more compressed tree cache. Use some of the reclaimed memory from the expanded tree cache to increase the compressed tree cache.

E-nodes:

- Reduce the compressed tree cache to a minimal number. E-nodes won't use compressed tree cache since they don't host data.

- Decrease the list cache. Since e-nodes don't host data, list cache is not important for them.
- Increase the expanded tree cache. Use the reclaimed memory from reducing the compressed tree and list caches. E-nodes use the expanded tree cache to send data back to the client; increasing this cache can improve performance.

At this point, the memory-optimized R3 Amazon EC2 instance types might become more appropriate. The r3.8xlarge with a 10 Gig network or the r3.4xlarge with Amazon EBS would be good choices here. If memory is not a bottleneck, then the c3.8xlarge or the c3.4xlarge instance types with Amazon EBS would also be good choice.

If you're not using an instance that supports a 10 Gigabit network, use Amazon EBS-optimized instance types and Amazon EBS General Purpose (SSD) volumes.

Extra-large scale: At this point, recommendations vary depending upon the use case. We think of MarkLogic applications as being on a continuum from high performance to high capacity use cases. High volume, high concurrency search applications are on one end, and data warehouse type applications on the other. Both of these can be considered extra-large scale—we might need to support a large number of concurrent queries over a large volume of data with subsecond response times. Or, we might need to support a smaller number of users over petabyte-scale data with more tolerance for longer-running reports.

For the high performance end of the spectrum, you'll need a 10 Gig network and many cores to support query concurrency. For the high storage end of the spectrum, you may want to scale out on compute or memory optimized instances with a smaller number of cores.

Instance types at this scale include r3.8xlarge or c3.8xlarge for the high performance applications, and r3.4xlarge, r3.2xlarge, or c3.4xlarge for the high capacity use cases.

For workloads with a high bias to reads and very large datasets, high I/O instances can deliver more than 120,000 4 input/output operations per second (IOPS) and between 10,000 and 85,000 4 KB random write IOPS (depending on the active logical block addressing span) to applications across two 1 TB local SSD data volumes. Exercise caution when using hi1.4xlarge instances as primary storage for data; as with all ephemeral disks, stop commands or instance failures will cause irrecoverable loss of data. Snapshots are not available on ephemeral disks, so we strongly recommend operating with hi1.4xlarge replicas and adding another replica that uses Amazon EBS volumes (on a CC2 instance), with replication for greatly simplified durable backups to Amazon S3 by snapshot.

Expanding Your MarkLogic Cluster Using AWS CloudFormation Templates

If you have created your stack using the 3+ AWS CloudFormation template, you can add nodes and forests to scale up your cluster for periods of heavy use and then remove them later when fewer resources are needed.

Adding more hosts to a cluster is simple. Simply use the Update Stack feature to reapply the 3+ AWS CloudFormation template and provide a larger number for the NodesPerZone setting. Alternatively, you can add hosts by means of your Auto Scaling groups. After you've added hosts, you'll need to create and attach Amazon EBS volumes for the hosts. SSH into each host and then run:

```
/opt/MarkLogic/bin/mlcmd init-volumes-by-system
```

After the volumes have been initialized, you can create forests using the MarkLogic REST API. An example of how to script forest creation follows:

```
$ cat forest-create.xml
 <forest-create xmlns="http://marklogic.com/manage">
   <forest-name>example-1</forest-name>
   <data-directory>/var/opt/volume1</data-directory>
   <host>my-host</host>
 </forest-create>

 $  curl --anyauth --user user:password -X POST -d @./forest-create.xml
\
      -i -H "Content-type: application/xml" \
      http://localhost:8002/manage/v2/forests

 ==> Create a forest named "example-1" on host "my-host". MarkLogic
Server
     responds with output similar to the following. You can use the
     reference in the Location header to retrieve status and other
     information about the new forest.

 HTTP/1.1 201 Created
 Location: /manage/v2/forests/2721944172534009539
 Content-type: application/xml
 Cache-Control: no-cache
 Expires: -1
 Server: MarkLogic
 Content-Length: 0
 Connection: Keep-Alive
 Keep-Alive: timeout=5
```

Most MarkLogic management operations can be scripted via the Management REST interface. To mount a forest on a specific Amazon EBS volume, set the data-directory to the Amazon EBS volume path, as in the above example.

After the cluster has been expanded, some forests might need to be migrated so that replica forests are spread evenly across a cluster. More information about forest migration can be found in the Administrator's Guide.

After the cluster is expanded, content can be redistributed across the database to take best advantage of the new hosts. MarkLogic uses a built-in rebalancer to move content around a cluster. The rebalancer must be enabled on the database configuration in order for the rebalancing to occur. For more information about the rebalancer, see the Administrator's Guide.

Shrinking Your MarkLogic Cluster Using AWS CloudFormation Templates

1. Use MarkLogic and AWS tools to identify an equal number of hosts in each ASG to delete. Never delete the host with the Security database, or any of the other built-in MarkLogic databases, such as Meters, App-Services, Modules, and so on.

2. Delete or move the data from the hosts to be removed to other hosts. Forests can be marked as "retired." Change the forest to "retired" and enable the rebalance; this will move existing content from that forest to other forests in the cluster. This can be scripted using the REST Management API. For more details, see the Administrator's Guide.

3. As a super user, run the leave-cluster -terminate command on each host to be removed. This will cause the node to leave the cluster, and adjust the AutoScaling Group DesiredCount setting. For details, see leave-cluster.

4. Update the AWS CloudFormation template to represent the downsized cluster and use the Update Stack feature to reapply the template to the stack to alert AWS of the updated configuration.

5. Detach and delete any unused volumes.

Anti-Patterns

In this section, we will identify practices that are generally best to avoid.

1. Failure to use Managed Clusters and AWS CloudFormation templates means that the operational complexity falls on the user, rather than the system. Many of the operational complexities of using MarkLogic on AWS have been considered and worked through in the AWS CloudFormation templates; choosing not to use the templates greatly increases the risks for misconfiguration. AWS

CloudFormation stack deployment can be automated with popular DevOps tools like Chef, Puppet, or Ansible.

2. Creating a large cluster far in excess of either your RAM or storage I/O requirements will provide you with no significant benefits except the ability to grow into your cluster without having to add instances and rebalance. Instead, construct a cluster that can cope with your known data requirements with suitable headroom for expansion, and then increase the cluster as required before it becomes overloaded.

3. Avoid using non-optimized Amazon EBS instance types (unless your instance includes 10 Gig network capacity). Not having enough I/O capacity slows down ingestion and updates, but MarkLogic also requires I/O at run-time for queries; not having consistent read capacity will cause unpredictable query performance.

4. Failure to monitor and anticipate growth can mean that a cluster is initially undersized. Rebalancing places additional memory and I/O load on your cluster because it involves physical data movement across instances in the cluster. As a result, you should aim to increase the size of your cluster before you reach cluster capacity. Keeping the equivalent RAM and I/O capacity of one or two instances in your cluster as a buffer before you expand is good practice, and it will enable you to expand your cluster before you run out of capacity. The rebalance operation should be run only when the cluster is in a healthy state.

Operations

In this section, we will discuss ongoing maintenance and cluster administration tasks.

Backup Using Amazon EBS Snapshot

You can use the Amazon EBS snapshot system to back up your running MarkLogic database. To provide a consistent, usable snapshot, you must temporarily stop data from being written to the database so that the files on storage are not being actively updated. You can then create the snapshot. Recovery can be done by restoring the data from the snapshot and then setting the database to “update” mode so that new documents can be added.

First, to quiesce the database, you must put all forests into “flash-backup” mode. When this update type is set on a forest, update transactions on the forest are retried until either the update type is reset or the Default Time Limit set for the App Server is reached. This operation can be scripted, or XQuery can be used from the Query Console in MarkLogic (http://host:8000/qconsole):

```
xquery version "1.0-ml";
import module namespace admin =
http://marklogic.com/xdmp/admin
```

```
        at "/MarkLogic/admin.xqy";
let $config := admin:get-configuration()
return
admin:forest-set-updates-allowed($config,
admin:forest-get-id($config, "Test"),
        "flash-backup")
```

This operation would need to use the correct forest name (not "Test" as in the above example). The operation needs to be completed for all forests attached to the database.

After this operation has returned, you can then use AWS to create an Amazon EBS snapshot of the volume where the forest resides. When the snapshot creation is complete, you can execute the following in the MarkLogic Query Console to allow the forest to again accept updates:

```
xquery version "1.0-ml";
import module namespace admin =
http://marklogic.com/xdmp/admin
        at "/MarkLogic/admin.xqy";
let $config := admin:get-configuration()
return
admin:forest-set-updates-allowed($config,
admin:forest-get-id($config, "Test"),
        "all")
```

Restore from an Amazon EBS Snapshot

Restoring a forest from an Amazon EBS snapshot is to be used when there has been a significant failure in your system, and you need to bring the instance and cluster back to the most recent snapshot status. You are, in effect, re-creating the volume and data status at the time of the snapshot.

To restore from an EBS snapshot:

1. Create a new instance, or use an existing instance, and detach the existing volume used for data storage.
2. Create a new volume from your EBS snapshot: `$ ec2-create-volume -K key -C certificate -s 70 -z us-east-1c --snapshot "backup1 "`
3. Attach a new volume created from the EBS snapshot in the previous step: `$ ec2-attach-volume -K key -C certificate VOLNAME -i INSTANCEID-d`

```
/dev/sdh
```

4. Connect to the host using SSH and run `init-volumes-from-system`. Your volume should now be mounted and ready to be used by your forests.

5. Run `sync-volumes-to-mdb`. This will save the information about your EBS volume to the metadata database.

Backup

You should back up your MarkLogic data frequently. Backups can be performed on a live and running system, but the method and frequency you use will depend on the quantity and rate of updates, and the data availability required. Because replicas and failover support within a MarkLogic cluster help to support online availability, data backups should be used mainly for long-term data retention and disaster recovery.

Because the MarkLogic backup utilities provide transactional consistency, use the MarkLogic-supplied backup capabilities to backup data. You can back up a MarkLogic database using either the Administration Console or script backups with the Admin API. For more information on database backups through the console, see the MarkLogic Administrator Guide. For scripting details, see Scripting Administrative Tasks Guide.

Amazon S3 is an excellent storage choice for MarkLogic database backups. You can connect directly to Amazon S3 from MarkLogic and store your database or forest in an Amazon S3 bucket.

In addition, MarkLogic supports Amazon S3 as a data store. Since Amazon S3 does not currently support "append", it cannot be used with journaling and is not appropriate as primary storage. However, for archival copies, Amazon S3 is ideal because data can be queried and accessed directly, without needing to restore to another file system.

To back up your MarkLogic database or Forest on Amazon S3, first follow the instructions in the MarkLogic Guide titled MarkLogic Server on Amazon EC2. This guide contains instructions for Configuring MarkLogic for Amazon S3. After you have set up an Amazon S3 bucket, configured the Amazon S3 endpoint for your group, and configured Amazon S3 credentials, you are ready to back up to Amazon S3.

To execute the backup, log into the MarkLogic Admin Console, select the database, and select the Backup/Restore tab. Enter the path to the Amazon S3 bucket for the backup directory. The following format represents an Amazon S3 path in MarkLogic:

```
s3://bucket/bucketdirectory
```

Restore

You can restore the MarkLogic database from backup using either the Administration Console or the Admin API. It is extremely important to test the restore process to make sure that the process is well-understood.

To restore an entire database from a backup using the Administration Console, perform the following steps:

1. Log into the Admin Interface as a user with the admin role.
2. Click the Databases link in the left menu of the Admin Interface.
3. Click the database name for the database you want to restore, either on the tree menu or on the summary page. This database should have the same configuration settings (index options, fragmentation, and range indexes) as the one that was backed up.
4. Click the Backup/Restore tab. The Backup/Restore screen appears.
5. Enter the directory in which the backup exists in the Restore from directory field.
6. If you enter a directory that contains multiple backups of the same database, the latest one is used. If you want to choose a particular backup to restore, enter the *date_stamp* subdirectory corresponding to the backup you want to restore. For details of the directory structure, see Backup Directory Structure.
7. If you have configured forests for local-disk failover, you can optionally set Include Replica Forests to true if you want to restore the replica forests from the backup. In order to use this option, you must have enabled the option to include the replica forests in the backup.
8. Leave Journal Archiving false.
9. Click OK.
10. The Confirm restore screen appears and lists all the forest selected for restoring.
11. The Confirm restore screen also lists the date the backup was performed and the server version used for the backup you selected.
12. By default, all of the forests associated with a database are checked to restore. If you do not want to restore all of the forests, deselect any forests you do not want to restore.
13. If you deselect any of the forests to restore, you might not be restoring a completely consistent view of the database. Only deselect a forest if you are sure you understand the implications of what you are restoring. To guarantee the

exact same view of the database, restore all of the forests associated with the database, including the Schemas and Security database forests.

14. Click OK to begin the restore operation.

Monitoring

AWS provides robust monitoring of Amazon EC2 instances, Amazon EBS volumes, and other services via the Amazon CloudWatch service. Amazon CloudWatch can alarm, via SMS or email, upon user-defined thresholds on individual AWS services. A great example would be alarming on excessive storage throughput, provided here. Another approach would be to write a custom metric to Amazon CloudWatch, for example, to monitor current free memory on your instances, and to alarm or trigger automatic responses off of those measures.

MarkLogic also provides a built-in monitoring application out of the box. More information about monitoring MarkLogic can be found in the Monitoring MarkLogic Guide.

Security

Network

To enable access to the Administration Console, you must open port 8001 within the security group. In addition, you will need to open whatever ports your application uses for its various application servers.

For communication between instances within the cluster, you will need to open port 7999.

If you want to use SSL for client server communication or for intra-cluster communication, you'd need to open those ports as well.

You should configure the MarkLogic security groups to allow access to those ports from the security groups for your application servers. All other ports should be restricted.

Authentication

Authentication is not required for client applications to access databases and application servers, but you can set a password on each application server that the client must supply in order to connect to the server. For administration, you must initially use the user name and password that were configured during setup, but you can create multiple admin users and grant the admin role to those users. For more detail on MarkLogic Security, go to the Understanding and Using Security Guide.

Conclusion

The AWS Cloud provides a unique platform for any NoSQL application, including MarkLogic. With capacities that can meet dynamic needs, cost that is based on use, and easy integration with AWS products like Amazon CloudWatch, the AWS Cloud enables you to run a variety of NoSQL applications without having to manage the hardware yourself.

In addition, management features inherent in NoSQL systems can be leveraged in the cloud. This paper provides specific information on MarkLogic, including hardware configurations, architecture designs, high availability and data recovery approaches, and security details.

For information on Amazon AWS, see http://aws.amazon.com.

Further Reading

For additional help, please consult the following sources:

MarkLogic Server on Amazon EC2 Guide, http://docs.marklogic.com/guide/ec2

Scalability, Availability and Failover Guide, http://docs.marklogic.com/guide/cluster

Monitoring MarkLogic Server Guide, http://docs.marklogic.com/guide/monitoring