Data, BigData, Cloud, Anaytics & Virtualizaton
HadoopExam.com | QuickTechie.com

AWS DEVELOPER ASSOCIATE LEVEL REVISION NOTES

Revision Notes for preparing AWS Developer Associate Level Certification

amazon web services	amazon web services	amazon web services	amazon web services	amazon web services
266 Q & A	300 + Q & A	474 + Q & A	144 Q & A	
Click Here	Click Here	Click Here	Click Here	Click Here
AWS Developer Certification Associate Level	AWS Certified SysOps Administrator Associate Level	AWS Certified Solutions Architect Associate Level	AWS Certified Solutions Architect Professional Level	Click Here for AWS Package Deal

WWW.HADOOPEXAM.COM
HadoopExam Learning Resources

Note: This revision notes cum study material should be used in conjunction with the AWS certification Practice Questions Provided by www.HadoopExam.com , and please note that this are the points/topic you need to focus for your real exam. However, it is recommended you have in depth knowledge of the AWS products referred in this doc, to work on AWS as a developer.

EBS: is a virtual block device. You can think of it as a hard drive, although it's really a bunch of software magic to link into another kind of storage device but make it look like a hard drive to an instance.
EBS is just the name for the whole service. Inside of EBS you have what are called volumes. These are the "unit" amazon is selling you. You create a volume and they allocate you X number of gigabytes and you use it like a hard drive that you can plug into any of your running computers (instances).

We take data protection very seriously! Over the years we have added a number of security and encryption features to various parts of AWS. We protect data at rest with Server Side Encryption for Amazon S3 and Amazon Glacier, multiple tiers of encryption for Amazon Redshift, and Transparent Data Encryption for Oracle and SQL Server databases via Amazon RDS. We protect data in motion with extensive support for SSL/TLS in CloudFront, Amazon RDS, and Elastic Load Balancing.

Today we are giving you yet another option, with support for encryption of EBS data volumes and the associated snapshots. You can now encrypt data stored on an EBS volume at rest and in motion by setting a single option. When you create an encrypted EBS volume and attach it to a supported instance type, data on the volume, disk I/O, and snapshots created from the volume are all encrypted. The encryption occurs on the servers that host the EC2 instances, providing encryption of data as it moves between EC2 instances and EBS storage.

Adding encryption to a provisioned IOPS (PIOPS) volume will not affect the provisioned performance. Encryption has a minimal effect on I/O latency. The snapshots that you take of an encrypted EBS volume are also encrypted and can be moved between AWS Regions as needed. You cannot share encrypted snapshots with other AWS accounts and you cannot make them public.

As I mentioned earlier, your data is encrypted before it leaves the EC2 instance. In order to be able to do this efficiently and with low latency, the EBS encryption feature is only available on EC2's M3, C3, R3, CR1, G2, and I2 instances. You cannot attach an encrypted EBS volume to other instance types.

Also, you cannot enable encryption for an existing EBS volume. Instead, you must create a new, encrypted volume and copy the data from the old one to the new one using the file manipulation tool of your choice. Rsync (Linux) and Robocopy (Windows) are two good options, but there are many others.

Each newly created volume gets a unique 256-bit AES key; volumes created from encrypted snapshots share the key. You do not need to manage the encryption keys because they are protected by our own key management infrastructure, which implements strong logical and physical security controls to prevent unauthorized access. Your data and associated keys are encrypted using the industry-standard AES-256 algorithm.

Volumes: can either be created blank or from a snapshot copy of previous volume, which brings us to the next topic.
Snapshots: snapshots of volumes: An exact capture of what a volume looked like at a particular moment in time, including all its data.
You could have a volume, attach it to your instance, fill it up with stuff, then snapshot it, but keep using it. The volume contents would keep changing as you used it as a file system but the snapshot would be frozen in time.
You could create a new volume using this snapshot as a base. The new volume would look exactly like your first disk did when you took the snapshot. You could start using the new volume in place of the old one to roll-back your data, or maybe attach the same data set to a second machine. You can keep taking snapshots of volumes at any point in time. It's like a freeze-frame instance backup that can then easy be made into a new live disk (volume) whenever you need it.
So volumes can be based on new blank space or on a snapshot. Volumes can be attached and detached from any instances, but only connected to one instance at a time, just like the physical disk that they are a virtual abstraction of.

Amazon Instance Store-Backed: Cannot be in stopped state; instances are running or terminated

Amazon EBS-Backed: Can be placed in stopped state where instance is not running, but the root volume is persisted in Amazon EBS. You can stop an Amazon EBS-backed instance, but not an Amazon EC2 instance store-backed instance. Stopping causes the instance to stop running (its status goes from running to stopping to stop). A stopped instance persists in Amazon EBS, which allows it to be restarted. Stopping is different from terminating; you can't restart a terminated instance. Because Amazon EC2 instance store-backed AMIs can't be stopped, they're either running or terminated.

AMI: These are tricky because there are two types. One creates an ephemeral instances where the root files system looks like a drive to the computer but actually sits in memory somewhere and vaporizes the minute it stops being used.
The other kind is called an EBS backed instance. This means that when your instances loads up, it loads its root file system onto a new EBS volume, basically layering the EC2 virtual machine technology on top of their EBS technology. A regular EBS volume is something that sits next to EC2 and can be attached, but an EBS backed instance also IS a volume itself.
An regular AMI is just a big chunk of data that get's loaded up as a machine. An EBS backed AMI will get loaded up onto an EBS volume, so you can shut it down and it will start back up from where you left off just like a real disk would.
Now put it all together. If an instance is EBS backed, you can also snapshot it. Basically this does exactly what a regular snapshot would ... a freeze frame of the root disk of your computer at a moment in time.
In practice, it does two things different. One is it shuts down your instance so that you get a copy of the disk as it would look to an OFF computer, not an ON one. This makes it easier to boot up. So when you snapshot an instance, it shuts it down, takes the disk picture, and then starts up again. Secondly, it saves that images as an AMI instead of as a regular disk snapshot. Basically it's a bootable snapshot of a volume.
You create an EBS AMI from an instance, and you create a snapshot from a volume.

An AMI includes the following:

- A template for the root volume for the instance (for example, an operating system, an application server, and applications)
- Launch permissions that control which AWS accounts can use the AMI to launch instances
- A block device mapping that specifies the volumes to attach to the instance when it's launched
- Amazon Machine Image (AMI) contains a disk image that is used to boot an Amazon EC2 instance. It is created by using the Create Image command on an existing Amazon EC2 instance, and can also be created from an EBS Snapshot (Linux only).
- Each AMI receives a unique ID in the form ami-1234abcd.
  AMIs exist in only one AWS region. When an AMI is copied between regions, it receives a new AMI ID. This means that the "same" image will appear with a different AMI ID in each region.
- When an AMI is copied between regions, it will retain its name and description, but this is not necessarily unique, so it cannot be used to definitively match AMIs between regions.

Using Web Identity Federation : Now suppose that a mobile gaming app uses this table, and that app needs to support thousands, or even millions, of users. At this scale, it becomes very difficult to manage individual app users, and to guarantee that each user can only access their own data in the *GameScores* table. Fortunately, many users already have accounts with a third-party identity provider, such as Facebook, Google, or Login with Amazon — so it makes sense to leverage one of these providers for authentication tasks.

If you are writing an application targeted at large numbers of users, you can optionally use web identity federation for authentication and authorization. Web identity federation removes the need for creating individual IAM users; instead, users can sign in to an identity provider and then obtain temporary security credentials from AWS Security Token Service (AWS STS). The app can then use these credentials to access AWS services.
Web identity federation supports the following identity providers:

- Login with Amazon
- Facebook
- Google

To do this using web identity federation, the app developer must register the app with an identity provider (such as Login with Amazon) and obtain a unique app ID. Next, the developer needs to create an IAM role. (For this example, we will give this role a name of GameRole.) The role must have an IAM policy document attached to it, specifying the conditions under which the app can access GameScores table.

When a user want to play a game, he signs in to his Login with Amazon account from within the gaming app. The app then calls AWS Security Token Service (AWS STS), providing the Login with Amazon app ID and requesting membership in GameRole. AWS STS returns temporary AWS credentials to the app and allows it to access the GameScores table, subject to theGameRole policy document.

DynamoDB: Primary Key

When you create a table, in addition to the table name, you must specify the primary key of the table. The primary key uniquely identifies each item in the table, so that no two items can have the same key.

DynamoDB supports the following two types of primary keys:

- **Hash Type Primary Key—**the primary key is made of one attribute, a hash attribute. DynamoDB builds an unordered hash index on this primary key attribute. Each item in the table is uniquely identified by its hash key value.

- **Hash and Range Type Primary Key**—the primary key is made of two attributes. The first attribute is the hash attribute and the second one is the range attribute. DynamoDB builds an unordered hash index on the hash primary key attribute, and a sorted range index on the range primary key attribute. Each item in the table is uniquely identified by the combination of its hash and range key values. It is possible for two items to have the same hash key value, but those two item.

This means that every row's primary key is the **combination of the hash and range key**. You can make direct gets on single rows if you have both the hash and range key, or you can make a query against the **sorted range index**. For example, get *Get me all rows from the table with Hash key X that have range keys greater than Y*, or other queries to that affect. They have better performance and less capacity usage compared to Scans and Queries against fields that are not indexed.

The choice of which key to use comes down to your Use Cases and Data Requirements for a particular scenario. For example, if you are storing User Session Data it might not make much sense using the Range Key since each record could be referenced by a GUID and accessed directly with no grouping requirements. In general terms once you know the Session Id you just get the specific item querying by the key. Another example could be storing User Account or Profile data, each user has his own and you most likely will access it directly (by User Id or something else).

However, if you are storing Order Items then the Range Key makes much more sense since you probably want to retrieve the items grouped by their Order.

In terms of the Data Model, the Hash Key allows you to uniquely identify a record from your table, and the Range Key can be optionally used to group and sort several records that are usually retrieved together. Example: If you are defining an Aggregate to store Order Items, the Order Id could be yourHash Key, and the OrderItemId the Range Key. Whenever you would like to search the Order Itemsfrom a particular Order, you just query by the Hash Key (Order Id), and you will get all your order items.

Parallel Scan

By default, the Scan operation processes data sequentially. DynamoDB returns data to the application in 1 MB increments, and an application performs additional Scan operations to retrieve the next 1 MB of data.
The larger the table or index being scanned, the more time the Scan will take to complete. In addition, a sequential Scan might not always be able to fully utilize the provisioned read throughput capacity: Even though DynamoDB distributes a large table's data across multiple physical partitions, a Scan operation can only read one partition at a time. For this reason, the throughput of a Scan is constrained by the maximum throughput of a single partition.

To address these issues, the Scan operation can logically divide a table or secondary index into multiple segments, with multiple application workers scanning the segments in parallel. Each worker can be a thread (in programming languages that support multithreading) or an operating system process. To perform a parallel scan, each worker issues its own Scan request with the following parameters:

Segment — A segment to be scanned by a particular worker. Each worker should use a different value for Segment.

TotalSegments — the total number of segments for the parallel scan. This value must be the same as the number of workers that your application will use.

A parallel scan with a large number of workers can easily consume all of the provisioned throughput for the table or index being scanned. It is best to avoid such scans if the table or index is also incurring heavy read or write activity from other applications.

To control the amount of data returned per request, use the Limit parameter. This can help prevent situations where one worker consumes all of the provisioned throughput, at the expense of all other workers. For more information, see "Reduce Page Size

Many applications can benefit from using parallel Scan operations rather than sequential scans. For example, an application that processes a large table of historical data can perform a parallel scan much faster than a sequential one. Multiple worker threads in a background "sweeper" process could scan a table at a low priority without affecting production traffic. In each of these examples, a parallel Scan is used in such a way that it does not starve other applications of provisioned throughput resources.

Although parallel scans can be beneficial, they can place a heavy demand on provisioned throughput. With a parallel scan, your application will have multiple workers that are all running Scan operations concurrently, which can very quickly consume all of your table's provisioned read capacity. In that case, other applications that need to access the table might be throttled.

A parallel scan can be the right choice if the following conditions are met:

The table size is 20 GB or larger.

The table's provisioned read throughput is not being fully utilized.

Sequential Scan operations are too slow.

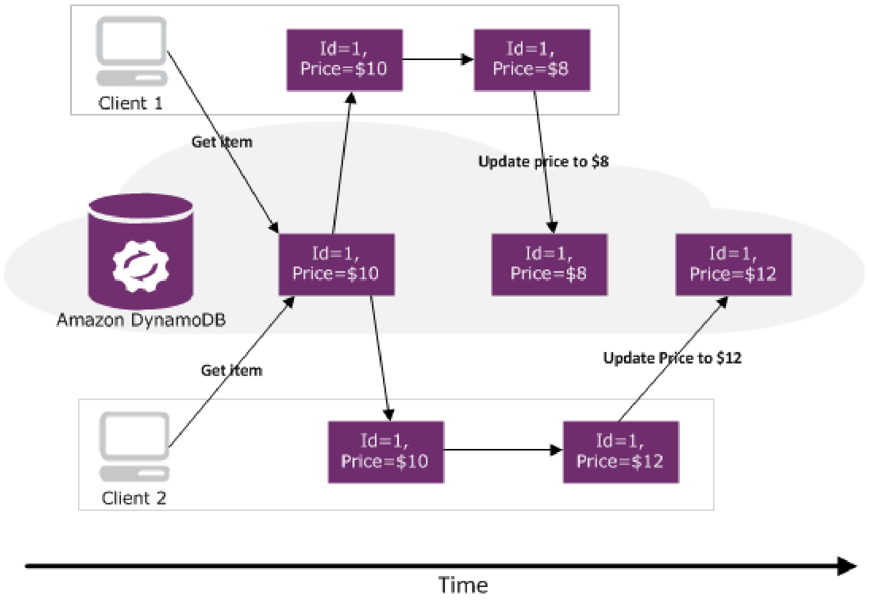
Optimistic Locking With Version Number

Optimistic locking is a strategy to ensure that the client-side item that you are updating (or deleting) is the same as the item in DynamoDB. If you use this strategy, then your database writes are protected from being overwritten by the writes of others — and vice-versa.

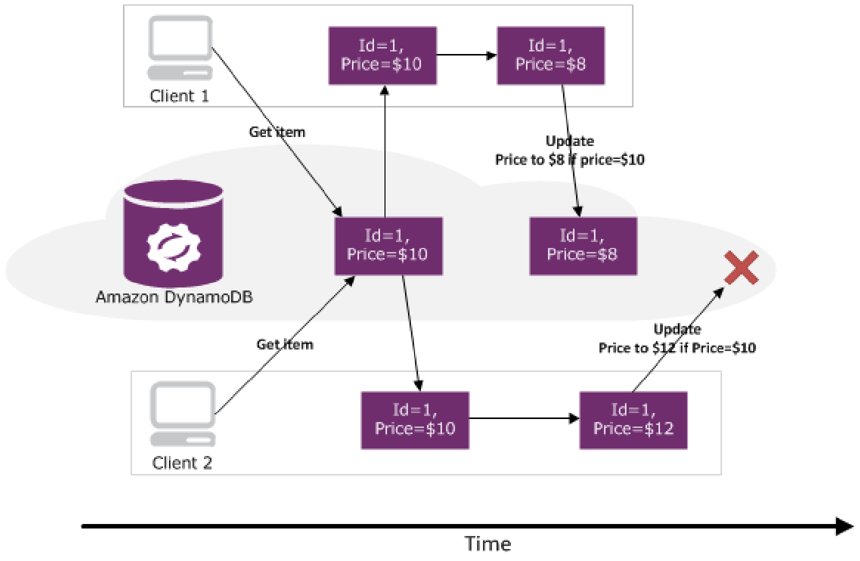
With optimistic locking, each item has an attribute that acts as a version number. If you retrieve an item from a table, the application records the version number of that item. You can update the item, but only if the version number on the server side has not changed. If there is a version mismatch, it means that someone else has modified the item before you did; the update attempt fails, because you have a stale version of the item. If this happens, you simply try again by retrieving the item and then attempting to update it. Optimistic locking prevents you from accidentally overwriting changes that were made by others; it also prevents others from accidentally overwriting your changes.

To support optimistic locking, the AWS SDK for Java provides the `@DynamoDBVersionAttribute` annotation type. In the mapping class for your table, you designate one property to store the version number, and mark it using this annotation type. When you save an object, the corresponding item in the DynamoDB table will have an attribute that stores the version number. The `DynamoDBMapper` assigns a version number when you first save the object, and it automatically increments the version number each time you update the item. Your update or delete requests will succeed only if the client-side object version matches the corresponding version number of the item in the DynamoDB table.

Conditional Writes: In a multi-user environment, multiple clients can access the same item and attempt to modify its attribute values at the same time. However, each client might not realize that other clients might have modified the item already. This is shown in the following illustration in which Client 1 and Client 2 have retrieved a copy of an item (Id=1). Client 1 changes the price from $10 to $8. Later, Client 2 changes the same item price to $12, and the previous change made by Client 1 is lost.



To help clients coordinate writes to data items, DynamoDB supports *conditional writes* for PutItem, DeleteItem, and UpdateItem operations. With a conditional write, an operation succeeds only if the item attributes meet one or more expected conditions; otherwise it returns an error. For example, the following illustration shows both Client 1 and Client 2 retrieving a copy of an item (Id=1). Client 1 first attempts to update the item price to $8, with the expectation that the existing item price on the server will be $10. This operation succeeds because the expectation is met. Client 2 then attempts to update the price to $12, with the expectation that the existing item price on the server will be $10. This expectation cannot be met, because the price is now $8; therefore, Client 2's request fails.



Note that conditional writes are *idempotent*. This means that you can send the same conditional write request multiple times, but it will have no further effect on the item after the first time DynamoDB performs the specified update. For example, suppose you issue a request to update the price of a book item by 10%, with the expectation that the price is currently $20. However, before you get a response, a network error occurs and you don't know whether your request was successful or not. Because a conditional update is an idempotent operation, you can send the same request again. and DynamoDB will update the price only if the current price is still $20.

To request a conditional PutItem, DeleteItem, or UpdateItem, you specify the condition(s) in the ConditionExpression parameter. ConditionExpression is a string containing attribute names, conditional operators and built-in functions. The entire expression must evaluate to true; otherwise the operation will fail.

Tip

For more information, see Conditional Write Operations.

If you specify the ReturnConsumedCapacity parameter, DynamoDB will return the number of write capacity units that were consumed during the conditional write. (Note that write operations only consume write capacity units; they never consume read capacity units.) Setting ReturnConsumedCapacity to TOTAL returns the write capacity consumed for the table and all of its global secondary indexes; *INDEXES* returns only the write capacity consumed by the global secondary indexes; *NONE* means that you do not want any consumed capacity statistics returned.

If a ConditionExpression fails during a conditional write, DynamoDB will still consume one write capacity unit from the table. A failed conditional write will return a*ConditionalCheckFailedException* instead of the expected response from the write operation. For this reason, you will not receive any information about the write capacity unit that was consumed. However, you can view the ConsumedWriteCapacityUnits metric for the table in Amazon CloudWatch to determine the provisioned write capacity that was consumed from the table.

Note

Unlike a global secondary index, a local secondary index shares its provisioned throughput capacity with its table. Read and write activity on a local secondary index consumes provisioned throughput capacity from the table.

ProvisionedThroughputExceededException

If you have checked in the AWS Management Console and verified that throttling events are occurring even when read capacity is well below provisioned capacity the most likely answer is that your hash keys are not evenly distributed. As your DynamoDB table grows in size and capacity, the DynamoDB service will automatically split your table into partitions. It will then use the hash key of the item to determine which partition to store the item. In addition, your provisioned read capacity is also split evenly among the partitions.

If you have a well-distributed hash key, this all works fine. But if your hash key is not well distributed it can cause all or most of your reads to come from a single partition. So, for example, if you had 10 partitions and you had a provisioned read capacity of 1000 on the table, each partition would have a read capacity of 100. If all of your reads are hitting one partition you will be throttled at 100 read units rather than 1000.

Unfortunately, the only way to really fix this problem is to pick a better hash and rewrite the table with those hash values.

CreatePlatformEndpoint

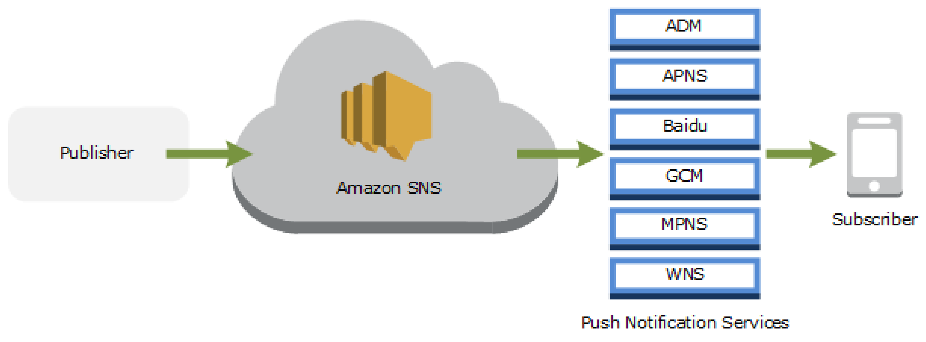
With Amazon SNS, you have the ability to send push notification messages directly to apps on mobile devices. Push notification messages sent to a mobile endpoint can appear in the mobile app as message alerts, badge updates, or even sound alerts.

Creates an endpoint for a device and mobile app on one of the supported push notification services, such as GCM and APNS. CreatePlatformEndpoint requires the PlatformApplicationArn that is returned from CreatePlatformApplication. The EndpointArn that is returned when using CreatePlatformEndpoint can then be used by the Publish action to send a message to a mobile app or by the Subscribe action for subscription to a topic. The CreatePlatformEndpoint action is idempotent, so if the requester already owns an endpoint with the same device token and attributes, that endpoint's ARN is returned without creating a new endpoint.

You send push notification messages to both mobile devices and desktops using one of the following supported push notification services:

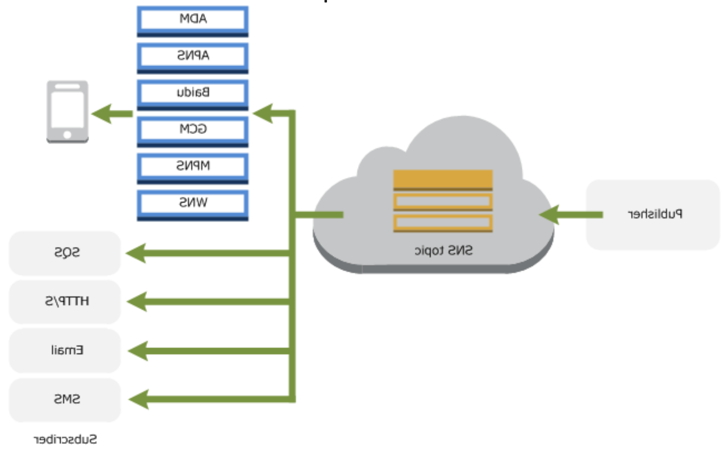
- Amazon Device Messaging (ADM)
- Apple Push Notification Service (APNS) for both iOS and Mac OS X
- Baidu Cloud Push (Baidu)
- Google Cloud Messaging for Android (GCM)
- Microsoft Push Notification Service for Windows Phone (MPNS)
- Windows Push Notification Services (WNS)

The following figure shows an overview of how Amazon SNS is used to send a direct push notification message to a mobile endpoint.



Push notification services, such as APNS and GCM, maintain a connection with each app and associated mobile device registered to use their service. When an app and mobile device register, the push notification service returns a device token. Amazon SNS uses the device token to create a mobile endpoint, to which it can send direct push notification messages. In order for Amazon SNS to communicate with the different push notification services, you submit your push notification service credentials to Amazon SNS to be used on your behalf.

In addition to sending direct push notification messages, you can also use Amazon SNS to send messages to mobile endpoints subscribed to a topic. The concept is the same as subscribing other endpoint types, such as Amazon SQS, HTTP/S, email, and SMS, to a topic. The difference is that Amazon SNS communicates using the push notification services in order for the subscribed mobile endpoints to receive push notification messages sent to the topic. The following figure shows a mobile endpoint as a subscriber to an Amazon SNS topic. The mobile endpoint communicates using push notification services where the other endpoints do not.



Prerequisites

To begin using Amazon SNS mobile push notifications, you first need an app for the mobile endpoints that uses one of the supported push notification services: ADM, APNS, Baidu, GCM, MPNS, or WNS. After you've registered and configured the app to use one of these services, you configure Amazon SNS to send push notification messages to the mobile endpoints.

Registering your application with a push notification service requires several steps. Amazon SNS needs some of the information you provide to the push notification service in order to send direct push notification messages to the mobile endpoint. Generally speaking, you need the required credentials for connecting to the push notification service, a device token or registration id (representing your mobile device and mobile app) received from the push notification service, and the mobile app registered with the push notification service. The specific names will vary depending on which push notification service is being used. For example, when using APNS as the push notification service, you need the *device token*. Alternatively, when using GCM, the device token equivalent is called the *registration ID*. The *device token* and *registration ID* are unique identifiers for the mobile app and device.

To send a push notification message to a mobile app and device using the APIs, you must first use the `CreatePlatformApplication` action, which returns a *`PlatformApplicationArn`*attribute. The *`PlatformApplicationArn`* attribute is then used by `CreatePlatformEndpoint`, which returns an *`EndpointArn`* attribute. You can then use the *`EndpointArn`* attribute with the `Publish` action to send a notification message to a mobile app and device, or you could use the *`EndpointArn`* attribute with the `Subscribe` action for subscription to a topic.

Token: Unique identifier created by the notification service for an app on a device. The specific name for Token will vary, depending on which notification service is being used. For example, when using APNS as the notification service, you need the device token. Alternatively, when using GCM or ADM, the device token equivalent is called the registration ID.

Amazon Simple Notification Service (Amazon SNS) is a web service that coordinates and manages the delivery or sending of messages to subscribing endpoints or clients. In Amazon SNS, there are two types of clients—publishers and subscribers—also referred to as producers and consumers. Publishers communicate asynchronously with subscribers by producing and sending a message to a topic, which is a logical access point and communication channel. Subscribers (i.e., web servers, email addresses, Amazon SQS queues, AWS Lambda functions) consume or receive the message or notification over one of the supported protocols (i.e., Amazon SQS, HTTP/S, email, SMS, Lambda) when they are subscribed to the topic.

Elastic Beanstalk

With Elastic Beanstalk, you can quickly deploy and manage applications in the AWS cloud without worrying about the infrastructure that runs those applications. AWS Elastic Beanstalk reduces management complexity without restricting choice or control. You simply upload your application, and Elastic Beanstalk automatically handles the details of capacity provisioning, load balancing, scaling, and application health monitoring. Elastic Beanstalk uses highly reliable and scalable services that are available in the AWS Free Usage Tier such as:

- Amazon Elastic Compute Cloud
- Amazon Simple Storage Service
- Amazon Simple Notification Service
- Amazon CloudWatch
- Elastic Load Balancing
- Auto Scaling
- Amazon RDS
- Amazon DynamoDB
- Amazon CloudFront
- Amazon ElastiCache

S3 Share an Object with Others

All objects by default are private. Only the object owner has permission to access these objects. However, the object owner can optionally share objects with others by creating a pre-signed URL, using their own security credentials, to grant time-limited permission to download the objects.

When you create a pre-signed URL for your object, you must provide your security credentials, specify a bucket name, an object key, specify the HTTP method (GET to download the object) and expiration date and time. The pre-signed URLs are valid only for the specified duration.

Anyone who receives the pre-signed URL can then access the object. For example, if you have a video in your bucket and both the bucket and the object are private, you can share the video with others by generating a pre-signed URL.
Note: Anyone with valid security credentials can create a pre-signed URL. However, in order to successfully access an object, the pre-signed URL must be created by someone who has permission to perform the operation that the pre-signed URL is based upon.

Time Limited: "Signed URL" is the URL that is valid for a specific period of time. That's why, it is also known as "Time Limited Signed URL" . After the expiry time, the URL will no longer remain active and if user attempts to access the URL once it has expired, he/she will only find some "Request has expired"message. The Signed URL can be generated for all version objects.

Gets the pre-signed URL of a given S3 object and populates a variable with the results. A signed URL is a URL which is valid for a specific time period. Its purpose is to share the pages that have time sensitive information, or when you want to share a file with someone without making it available to everyone on the net. Such information becomes invalid once the specified time period has passed.

You can distribute private content using a signed URL that is valid for only a short time—possibly for as little as a few minutes. Signed URLs that are valid for such a short period are good for distributing content on-the-fly to a user for a limited purpose, such as distributing movie rentals or music downloads to customers on demand. If your signed URLs will be valid for just a short period, you'll probably want to generate them automatically using an application that you develop. When the user starts to download an object or starts to play a media file, CloudFront compares the expiration time in the URL with the current time to determine whether the URL is still valid.

You can also distribute private content using a signed URL that is valid for a longer time, possibly for years. Signed URLs that are valid for a longer period are useful for distributing private content to known users, such as distributing a business plan to investors or distributing training materials to employees. You can develop an application to generate these longer-term signed URLs for you

EC2 Instance Metadata for IP address:

local-ipv4 : The private IP address of the instance. In cases where multiple network interfaces are present, this refers to the eth0 device (the device for which the device number is 0).

public-ipv4 The public IP address. If an Elastic IP address is associated with the instance, the value returned is the Elastic IP address.

Provisioned Throughput in Amazon DynamoDB

When you create or update a table, you specify how much provisioned throughput capacity you want to reserve for reads and writes. DynamoDB will reserve the necessary machine resources to meet your throughput needs while ensuring consistent, low-latency performance.

A unit of *read capacity* represents one strongly consistent read per second (or two eventually consistent reads per second) for items as large as 4 KB. A unit of *write capacity* represents one write per second for items as large as 1 KB.

Items larger than 4 KB will require more than one read operation. The total number of read operations necessary is the item size, rounded up to the next multiple of 4 KB, divided by 4 KB. For example, to calculate the number of read operations for an item of 10 KB, you would round up to the next multiple of 4 KB (12 KB) and then divide by 4 KB, for 3 read operations.

Non-Uniform Workloads : When storing data, Amazon DynamoDB divides a table's items into multiple partitions, and distributes the data primarily based on the hash key element. The provisioned throughput associated with a table is also divided evenly among the partitions, with no sharing of provisioned throughput across partitions. Consequently, to achieve the full amount of request throughput you have provisioned for a table, keep your workload spread evenly across the hash key values. Distributing requests across hash key values distributes the requests across partitions.

For example, if a table has a very small number of heavily accessed hash key elements, possibly even a single very heavily used hash key element, traffic is concentrated on a small number of partitions – potentially only one partition. If the workload is heavily unbalanced, meaning disproportionately focused on one or a few partitions, the operations will not achieve the overall provisioned throughput level. To get the most out of Amazon DynamoDB throughput, build tables where the hash key element has a large number of distinct values, and values are requested fairly uniformly, as randomly as possible.

Another example of a non-uniform workload is where an individual request consumes a large amount of throughput. Expensive requests are generally caused by Scan or Query operations, or even single item operations when items are large. Even if these expensive requests are spread out across a table, each request creates a temporary hot spot that can cause subsequent requests to be throttled.

For instance, consider the Forum, Thread, and Reply tables from the Getting Started section of the developer guide, which demonstrate a forums web application on top of DynamoDB. The Reply table stores messages sent between users within a conversation, and within a thread, are sorted by time.

If you Query the Reply table for all messages in a very popular thread, that query could consume lots of throughput all at once from a single partition. In the worst case, this expensive query could consume so much of the partition's throughput that it causes subsequent requests to be throttled for a few seconds, even if other partitions have throughput to spare.

AWS SDK:

AWS currently offers SDKs for seven different programming languages – Java, C#, Ruby, Python, JavaScript, PHP, and Objective C (iOS), and we closely follow the language trends among our customers and the general software community. Since its launch, the Go programming language has had a remarkable growth trajectory, and we have been hearing customer requests for an official AWS SDK with increasing frequency. We listened and decided to deliver a new AWS SDK to our Go-using customers.

The AWS SDK for Java uses us-east-1 as the default region if you do not specify a region in your code. However, the AWS Management Console uses us-west-2 as its default. Therefore, when using the AWS Management Console in conjunction with your development, *be sure to specify the same region in both your code and the console*.

NAT Instances : Instances that you launch into a private subnet in a virtual private cloud (VPC) can't communicate with the Internet. You can optionally use a network address translation (NAT) instance in a public subnet in your VPC to enable instances in the private subnet to initiate outbound traffic to the Internet, but prevent the instances from receiving inbound traffic initiated by someone on the Internet.

Each EC2 instance performs source/destination checks by default. This means that the instance must be the source or destination of any traffic it sends or receives. However, a NAT instance must be able to send and receive traffic when the source or destination is not itself. Therefore, you must disable source/destination checks on the NAT instance.

For the NAT instance to operate correctly, the `Source/Destination Check` attribute must be disabled. Although the Amazon VPC wizard does this for you, these instructions are included so you can do this yourself if needed. You can also use this procedure to verify that the `Source/Destination Check` attribute has been disabled.

SNS Publish: Sends a message to all of a topic's subscribed endpoints. When a messageId is returned, the message has been saved and Amazon SNS will attempt to deliver it to the topic's subscribers shortly. The format of the outgoing message to each subscribed endpoint depends on the notification protocol selected.
To use the Publish action for sending a message to a mobile endpoint, such as an app on a Kindle device or mobile phone, you must specify the EndpointArn. The EndpointArn is returned when making a call with the CreatePlatformEndpoint action. The second example below shows a request and response for publishing to a mobile endpoint.

Request Parameters

Message

The message you want to send to the topic.

MessageAttributes.entry.N

Message attributes for Publish action.

MessageStructure

Set MessageStructure to json if you want to send a different message for each protocol. For example, using one publish action, you can send a short message to your SMS subscribers and a longer message to your email subscribers.

Subject

Optional parameter to be used as the "Subject" line when the message is delivered to email endpoints. This field will also be included, if present, in the standard JSON messages delivered to other endpoints.

TargetArn

Either TopicArn or EndpointArn, but not both.

TopicArn

The topic you want to publish to.

IAM Policy : Determining Whether a Request is Allowed or Denied

When a request is made, the AWS service decides whether a given request should be allowed or denied. The evaluation logic follows these rules:

- By default, all requests are denied. (In general, requests made using the account credentials for resources in the account are always allowed.)
- An explicit allow overrides this default.
- An explicit deny overrides any allows.

The order in which the policies are evaluated has no effect on the outcome of the evaluation. All policies are evaluated, and the result is always that the request is either allowed or denied.

Elastic IP : An Elastic IP address is a static IP address designed for dynamic cloud computing. With an Elastic IP address, you can mask the failure of an instance or software by rapidly remapping the address to another instance in your account. Your Elastic IP address is associated with your AWS

account, not a particular instance, and it remains associated with your account until you choose to release it explicitly.

We assign each instance in a default VPC two IP addresses at launch: a private IP address and a public IP address that is mapped to the private IP address through network address translation (NAT). The public IP address is allocated from the EC2-VPC public IP address pool, and is associated with your instance, not with your AWS account. You cannot reuse a public IP address after it's been disassociated from your instance.

We assign each instance in a nondefault VPC only a private IP address, unless you specifically request a public IP address during launch, or you modify the subnet's public IP address attribute. To ensure that an instance in a nondefault VPC that has not been assigned a public IP address can communicate with the Internet, you must allocate an Elastic IP address for use with a VPC, and then associate that Elastic IP address with the elastic network interface (ENI) attached to the instance.

Data, BigData, Cloud, Analytics & Virtualizaton HadoopExam.com | QuickTechie.com

	Special offer Hadoop Training + Any One Certification Simulator Offer close in next 3 days Hadoop Training + 50%+20% off Developer Certification Or Admin Certification Or HBase Certification ~~12800~~INR = 5200INR ~~$288~~ = $116 Click Here
	Special Full Package offer 1. Hadoop Training 2. Developer Certification 3. Admin Certification 4. HBase Certification ~~24400~~INR Product == 7900INR Only ~~$528~~ Product == $169 Only 50%+35% off Click Here
	Special offer HBase Training + HBase Certification Simulator Offer close in next 3 days 50%+20% off HBase Training + HBase Certification ~~12800~~INR = 5200INR ~~$288~~ = $116 Click Here

234 Questions for Data Science Certification EMC E20-007 (Data Science Associate) for real exam : Highest Number of Questions

EMC²

234 Q & A

Click Here

EMC Data Scientist Associate Certification

E20-007 (EMCDSA)

Data Science certification really needs a good and in depth knowledge of statistics cum BigData Hadoop knowledge. It also require you to have good knowledge in like the main phases of the Data Analytics Lifecycle, analyzing and exploring data with R, statistics for model building and evaluation, the theory and methods of advanced analytics and statistical modeling, the technology and tools that can be used for advanced analytics, operationalizing an analytics project, and data visualization techniques. Successful candidates will achieve the EMC Proven Professional – Data Science Associate credential. Hence to clear the real exam it realy needs very well preparation. So HadoopExam Learning Resources brings Data Science Certification Simulator with 234 Practice Questions, which can help you to prepare for this exam in lesser time. **Practice - practice - practice**! The EMC:DS E20-007 Exam Simulator offers you the opportunity to take 4 sample Exams before heading out for the real thing. Be ready to succeed on exam day!

163 Questions for OREILLY DataBricks Apache Spark Developer Certification : Practice Questions for real exam

Link:
http://www.hadoopexam.com/spark/Apache_Spark_Oreilly_databricks_developer_certification_exam_dumps.html

Lifetime Accessible and Free Updates e.g. Additional Questions

Apache Spark certification really needs a good and in depth knowledge of Spark , Basic BigData Hadoop knowledge and Its other component like SQL. It also require you to have good knowledge in Broadcast and Accumulators variable, basic coding skill in all three language Java,Scala, and Python to understand Spark coding questions. Understanding of transformations and action API. Spark Streaming , windows function, lazy evaluation, RDD lineage graph and SchemaRDD etc. Successful candidates will achieve the Spark Developer Cdertification credential. Hence to clear the real exam it realy needs very well preparation. So HadoopExam Learning Resources brings Apache Spark Certification Simulator with 163 Practice Questions, including 50 coding questions, which can help you to prepare for this exam in lesser time. **Practice - practice - practice**! Apache Spark Exam Simulator offers you the opportunity to take 3 sample Exams before heading out for the real thing. Be ready to succeed on exam day!