AWS CLI

luni, 25 noiembrie 2024 21:34

To configure the cli use: aws configure command and pass the credentials (IAM)

Docs for cli are found at:
aws — AWS CLI 2.22.4 Command Reference

Stored credentials are at:
%USERPROFILE%\.aws\credentials on Windows
./etc/.aws/credentials on linux

Command structure is:
aws [service] [command for service]
To change profile use --profile profile-name

Amazon Q

luni, 25 noiembrie 2024 22:04

Service similar to ChatGPT
Generative AI for all the SDLC phases

Amazon S3

marți, 26 noiembrie 2024 10:13

Offers scalabilty, durability, data availability, security and performance.
Designed for 11 9s durability

Can be configured to generate notifications for create / delete / update.

Use cases:

- Content storage
- Backup and restore
- Archiving
- Data lakes
- Big data analytics
- Disaster recovery
- Static website hosting

An object is the fundamental entity that is stored in Amazon S3.
An object can be any kind of file.
Objects consist of object data and metadata.
The metadata is a set of name-value pairs that describe the object.
Objects can be referred to by their URL.

- Bucket name must be globally unique.
- Buckets are located in regions.
- Objects are referenced through virtual -host-style URLs.
- Prefixes imply a folder structure in an S3 bucket.

User defined metadata:
Must start with x-amz-meta- prefi

Takeaways

marți, 26 noiembrie 2024 11:56

Objects have two types of metadata:

- System-defined metadata (e.g.: creation date)
- User defined metadata (defined with prefix x-amz-meta-)

S3 has three operations:

- PUT
- GET
- DELETE

S3 Select is a powerful tool to query data in place using SQL.
S3 versioning protects objects.
When hosting a website on a S3 bucket, the URL contains .website

PUT object

marți, 26 noiembrie 2024 10:36

Use the put object to upload entire objects to a bucket.
Should use single upload for object up to 5GB in a single PUT operation.
Should use multipart upload for object over 100mb.
Must use multipart upload for objects over 5GB.

CacheControl sets how long the file will be cached by the browser.
Set to 0 for no cache. Useful for files that are changing often.

GET object

marți, 26 noiembrie 2024 10:44

Used to retrieve objects from a specific bucket.
Requires read access to the bucket.
Can retrieve the whole object or retrieve parts of it by specifying a range of bytes.

SELECT object

marți, 26 noiembrie 2024 10:46

Used to query S3 bucket objects based on SQL.
You send a SQL query and a data serialization format for S3 object.
! Requires s3:GetObject permission.

DELETE object

marți, 26 noiembrie 2024 11:46

NO VERSIONING:

- Used to delete a single or multiple objects.
- Deletes permanently an object by specifying the key of the object.

WITH VERSIONING:

- To permanently delete, a request with key and version id is required
- Each individual version must be deleted in order to remove an object completely
- If only a delete key is specified, S3 adds a delete marker that becomes the current version of the object.
- Objects with a delete marker will return a 404 Not Found error message when it receives a request to retrieve the object.

Versioning

marți, 26 noiembrie 2024 10:48

A way to keep multiple variants of an object in the same bucket.
In versioning enabled buckets each object has a version ID.
Versioned buckets support object locking.
Amazon S3 Object Lock is another option to prevent data from being changed, overwritten or deleted.

Protecting data and managing access

marți, 26 noiembrie 2024 12:00

Data protection = protect data while it's in transit (traveling to and from S3 bucket) and while it’s at rest (stored in S3 bucket)

Securing data in transit can be done by using SSL/TLS with HTTPS or using client side encryption.

To protect data at rest (data stored inside S3):

- With client side encryption data is encrypted by the client and uploaded encrypted tot the bucket. In this case the client manages encryption keys and related tools.
- With server side encryption, S3 encrypts data at the object level and decrypts it when you access it.

Server Side Encryption:

- With Amazon S3-managed Keys (SSE-S3):
  - each object is encrypted with a unique key.
  - Each key is encrypted itself with a root key that rotates regularly.

- AWS KMS-managed Keys (SSE-KMS):
  - similar to SSE-S3.
  - Using an envelope key requires additional permissions.
  - The envelope key protects data encryption keys.
  - SSE-KMS also provides you with an audit trail of when your key was used and by whom.
  - User can create encryption keys or use the default key that is unique to user, service and region.

- Custom provided Keys (SSE-C):
  - User manages the encryption keys.
  - S3 manages encryption as it writes to disk.
  - S3 manages decryption when you access the object.

! S3 doesn't encrypt object that are in the bucket when encryption is enabled. It encrypts only objects that are added after encryption is enabled.

Identity based policies and resource based policies

marți, 26 noiembrie 2024 12:14

Access can be managed by writing *access policies* that grant permissions to other users.
By default all S3 resources are private and only the user that created the resources can access them.

S3 offers two types of access policies:

- Identity-based policies:
  - You attach these policies to AWS IAM users, groups and roles to grant them access to AWS resources.

- Resource-based policies:
  - You attach these policies to your S3 resources.

Access control list (ACLs)

marți, 26 noiembrie 2024 12:19

- Resource-based access policy to manage access at the object level or bucket level.
- Use to grant basic read/write permissions to other AWS accounts.

Object ACL is used in following scenarios:

- An object ACL is the only way to manage access to objects that the owner of the bucket does not own.
    - Scenario: An AWS account that owns a bucket may grant access to another account to upload in that bucket. The bucket owner does not own these objects, hence the AWS account that created the object must grant access to the owner of the bucket by using ACLs.

- Permissions vary by object.
    - You must manage permissions at the object level.

- Object ACLs control only object-level permissions. The entire bucket has a single bucket policy, but object ACLs are specified per object.

Bucket policy

marți, 26 noiembrie 2024 12:35

A bucket policy is another type of resource-based IAM policy.
You can add a new policy to an S3 bucket to grant other AWS accounts or IAM users permissions for the bucket and the objects in it.
Object permissions apply only to the objects that the bucket owner creates.

Presigned URLs

marți, 26 noiembrie 2024 12:37

- Provide access to PUT or GET objects without granting permissions to perform other actions.
- Use the permissions of the user who creates the URL.
- Provide security credentials, a bucket name, an object key, an HTTP method, and an expiration date and time.
- Are valid until the expiration time (max 1 week).

Uploading objects with presigned URLs - Amazon Simple Storage Service

Amazon IAM

miercuri, 27 noiembrie 2024 12:59

Shared responsibility model

miercuri, 27 noiembrie 2024 16:01

Security and **compliance** -> shared responsibility between AWS and the customer
AWS responsibility -> security **of** the cloud (physical infrastructure)

- Securing components (host operating system and virtualization)
- Global infrastructure such as AWS Regions, AZs and edge locations

Customer responsibility -> security **in** the cloud

- Security of the guest operating system (including updates and security patches)
- Security of application software used
- Security of group firewall provided by AWS
- Security group configurations, network configurations and secure account management
- Client and server side encryption
- Data integrity
- Authentication

IAM intro

miercuri, 27 noiembrie 2024 16:02

Used to configure fine-grained access to AWS resources.
Supports *federation from corporate systems* like Microsoft Active Directory and standards-based identity providers.
Support MFA authentication.

Authentication in the context of IAM = who can use your AWS resources
Authorization in the context of IAM = what resources can someone use and in what ways

IAM user

- Entity that you create in AWS to represent a person or an application that interacts with AWS services and resources
- Is given a permanent set of credentials, which stay with the user until a forced rotation occurs

IAM group

- Collections of users
- Used to grant the same set of permissions to multiple users

IAM role

- Similar to a user
- AWS identity to which permission policies can be attached
- Doesn't have long term credentials
- When a person or application assumes a role, they are provided with temporary security credentials for the role session

IAM policy

- JSON type
- Document that explicitly lists permissions
- Policy can be attached to a user, group, role or any combination of these resources

Authentication

miercuri, 27 noiembrie 2024 16:26

When a user assumes a role, the previous user's permissions are forgotten.

By creating a role for external account access, you don’t need to manage user names and passwords for third parties.

When needed to provide trusted users with temporary access credentials use roles.

On mobile apps you don't want to store security credentials as they are can be difficult to rotate and users can extract them.

Storing credential in an AWS credentials file offers some benefits:

- Project credentials are stored outside of the project, thus it's unlikely that you will accidentally commit them to VCS.
- Can define and name multiple sets of credentials (called *profiles*) in one place.
- Can reuse the same credentials across multiple projects.

!!
IAM roles with temporary credentials are preferred for applications that run on EC2 and for mobile applications.
Avoid storing credentials in publicly accessible places. Use credentials files instead.

Authorization

miercuri, 27 noiembrie 2024 16:43

Can control access to resources using IAM policies that grant or deny permissions.

By default, an authenticated IAM user/group/role can't access anything until is granted permissions.

An IAM policy defines:

- The effect
- Actions
- Resources
- Optional conditions under which an entity can invoke API operations

Identity based policies:

1. Are attached to users, groups and roles.
2. Indicate what that identity can do.
3. E.g.: grant a user the ability to access a DynamoDB table

Resource based policies:

1. Are attached to resources such as S3.
2. Indicate what a specified user (or group of users) is permitted to do with it.
3. Principal is the entity that access the resource.
4. E.g.: grant access to an S3 bucket.

Managed and inline policies

miercuri, 27 noiembrie 2024 17:01

Managed policies are standalone, identity-based policies that can be attached to multiple users, groups and roles.

- AWS managed policies are created and managed by AWS.
- Customer-managed policies are created and managed by customer.
- Provide several features, including reusability, central change, management, versioning and rollback, and the ability to delegate permission management to other users.

Inline policies are embedded in a principal entity such as user, group or role.

- Can use the same policy for multiple entities, but those entities do not share the policy. Instead, each entity has its own copy, which makes impossible to centrally manage inline policies.
- Inline policies are useful when you want to maintain a 1 to 1 relationship between a policy and the principal entity to which it is applied.

Evaluation of policies

miercuri, 27 noiembrie 2024 17:09

- By default, all requests are denied.
- An explicit allow overrides the default denied.
- An explicit deny overrides any allow.

The order that the policies are evaluated in has no effect on the outcome of the evaluation.
If one policy allows and one denies the same resource, then the action is denied.
The most restrictive policy is applied.

Principle of least privilege

miercuri, 27 noiembrie 2024 17:11

Start by granting the minimum AWS permissions that are required for the job role.
Grant additional permissions as needed.

NoSQL

vineri, 31 ianuarie 2025 17:12

Relational Databases

vineri, 31 ianuarie 2025 17:12

Used when:

- You must use strict schema rules and enforce data quality
- Database don't require extreme read/write capacity
- If you have a relational dataset that doesn’t require extreme performance, a relational database management solution (RDBMS) can be the lowest-effort solution.

Amazon RDS offers:

- **Aurora:** Database built for the cloud, compatible with MySQL and PostgreSQL
  - In terms of performance, can be 5x faster than MySQL and 3x faster than PostgreSQL (standard).
  - Provides security, availability and reliability at about 1/10 of the cost of commercial dbs.
- **PostgreSQL**
- **MySQL**
- **MariaDB**
- **Oracle database**
- **SQL Server**

Nonrelational Databases

vineri, 31 ianuarie 2025 17:17

Used when:

- You need the database to scale horizontally
- Your data doesn't lend itself well to traditional schemas
- Read/write rates exceed rates that can be economically supported through traditional SQL databases

Are schemaless stores, where items are not required to conform to rules.

The design of nonrelational databases means they can scale by adding more compute to infrastructure.
If needed, servers can be brought online and when demand stops, servers can be shut down to reduce costs.

ElastiCache & Neptune

vineri, 31 ianuarie 2025 17:27

ElastiCache:

- In-memory data store that can boost the performance of databases
- Can improve the performance of caching, session stores, gaming, geospatial services, real-time analytics and queueing.

Neptune:

- Graph database engine optimized for storing billions of relationships.
- Queries run with millisecond latency.
- Can be used for recommendation engines, fraud detection, drug discovery, network security, and other solutions that need complex relationships to find solutions.

DynamoDB

vineri, 31 ianuarie 2025 17:24

- Key-value and document database.
- Is multi-region, multi-active and durable DB
- Delivers single digit milliseconds performance.
- Works well for internet-scale apps.
- Supports ACID transactions, thus business applications can use DynamoDB for persistent storage.

Key cocepts for DynamoDB

vineri, 31 ianuarie 2025 17:30

Basic components of DynamoDB are:

- **Tables:**
  - DynamoDB stores data in tables.
  - A table contains items with attributes
- Items:
  - Each table contains zero or more items
  - An item is a group of attributes that is uniquely identifiable among all the other items
- Attributes:
  - Each item is composed of one or more attributes
  - An attribute is a fundamental data element, something that does not need to be broken down any further
- Primary key:
  - A table has a primary key that uniquely identifies each item in the table
  - No two items can have the same key

Types of primary keys:

1. Simple primary key (partition key only):
   - Simple primary key is composed of one attribute that's known as the partition key
   - DynamoDB builds an unordered index on this primary key attribute
2. Composite primary key:
   - Is composed of two attributes: **partition key** and **sort key**
   - DynamoDB builds an unordered index on the partition key attribute and a sorted index on the sort key attribute
   - In such a table, two items can have the same partition key but their sort key must be different

Each attribute has a name and a value. An attribute value can be one of the following:

1. Scalar types: Number, string, binary, Boolean and null
2. Set types (multi-value types): String set, number set and binary set
3. Document types: List and map

The size of an item is the sum of the lengths of its attribute names and values.
An item can be up to 400 KB in size.

Partitions and data distribution

vineri, 31 ianuarie 2025 17:42

- DynamoDB stores data in partitions.
- A partition is an allocation of storage for a table.
- It's backed by SSDs and is automatically replicated across multiple AZs within an AWS Region.
- Partition management is handled entirely by DynamoDB

If the table has a simple primary key (partition key only), DynamoDB stores and retrieves each item based on its partition key value.
The partition key of an item is also known as its hash attribute.

If the table has a composite primary key (partition key + sort key), DynamoDB will store all the items that have the same partition key value physically close together.
It will then order them by sort-key value in the partition.
The sort key of an item is also known as its range attribute.

Secondary indexes

vineri, 31 ianuarie 2025 17:47

- Used to query data based no non-primary-key attributes
- Secondary index defines an alternate key
- Contain:
  - Alternate-key attributes
  - Primary-key attributes
  - Optional subset of other attributes from the base table (projected attributes)
- Can be one of two types:
  - Global secondary index
  - Local secondary index

Can be used to perform queries on attributes that aren't part of the table's primary key.
With a secondary index, you can query the data in the table by using an alternate key, in addition to queries against the primary key.

In addition to the alternate key attributes and primary key attributes (partition key and sort key), a secondary index contains a subset of the other table attributes. When you create an index, you specify which attributes will be copied (or projected) from the base table to the index.
At a minimum, DynamoDB projects the key attributes from the base table into the index.

DynamoDB supports two types of secondary indexes:

1. **Global secondary index:**
   a. An index with a partition key and a sort key, both of which can be different from the keys in the base table.
   b. A global secondary index is considered **global** because queries on the index can span all the data in the base table, across all partitions
   c. A global secondary index has no size limitations and has its own provisioned throughput settings for read/write activity that are separate from the throughput settings of the table
2. **Local secondary index:**
   a. An index that has the same partition key as the base table, but a different sort key
   b. A local secondary index is **local** because every partition is scoped to a base table partition that has the same partition key value
   c. The total size of indexed items for any one partition key value can't exceed 10GB.
   d. A local secondary index shares provisioned throughput settings for read/write activity with the table that it indexes

Each table in DynamoDB can have up to 20 global secondary indexes (default limit) per table, and up to five local secondary indexes per table.

Read/Write throughput

vineri, 31 ianuarie 2025 18:01

Read consistency:

- DynamoDB automatically replicates your data across multiple AZs in an AWS Region, which provides built-in high availability and data durability
- All copies of the data are usually consistent within a second after a write operation

DynamoDB supports eventually consistent and strongly consistent reads.

Eventually consistent reads:

- Read data might not reflect latest updated data
- May return slightly stale data

Strongly consistent reads:

- Always returns a response with the most up-to-date data
- Might not be available if a network delay or outage occurs
- Are not supported on global secondary indexes

Transactions:

- **All or nothing changes**
- **Provide ACID properties**
- **Similar to SQL**

Provisioned read/write throughput

- Is the maximum amount of capacity that an application can consume from a table or index
- If the application exceeds the provisioned throughput, it's subject to request throttling
- DynamoDB divides throughput evenly among partitions.
- The throughput per partition is the total provisioned throughput divided by the number of partitions
- With provisioned throughput, you specify the throughput capacity in terms of read capacity units (RCUs) and write capacity units (WCUs)

RCU:

- A read capacity unit is the number of strongly consistent reads per second of items that are up to 4 KB in size
- For eventually consistent reads, you use half of the RCU that are provisioned (one read capacity unit is two reads per second for items that are up to 4KB)

WCU:

- Is the number of 1KB writes per second

On demand read/write throughput

- Pay-per-request pricing for read and write requests
- DynamoDB adapts rapidly to accommodate the workload as they scale up or down to any previously reached traffic level (if a workload's traffic level reaches a new peak, it adapts to accommodate the workload)
- Tables that use on-demand mode deliver the same single-digit millisecond latency, service-level agreement and security that DynamoDB already offers
- Can be used with new and existing tables, and can continue using the existing DynamoDB API without changing code
- Offers pay-per-request pricing for read/write requests so that you only pay for what you use

Is a good option when:

1. **You create new tables with unknown workloads**
2. **You have unpredictable application traffic**
3. **You prefer the ease of paying for only what you use**

Activity: Suppose that you must read 20 items that are 11 kb in size every second, with eventual consistency. How many RCUs must you provision?

Steps:

1. Round item size to the next multiple of 4 => 11 rounded to 12
2. Divide by 4 (item size per RCU): 12 / 4 = 3
3. Multiply with the number of items read per second: 3 * 20 = 60
4. Since we need eventual consistency, we divide by 2 (60 is the number of RCUs used for strong consistency and each strong consistent offer 2 eventual consistent): 60 / 2 = 30

Streams and global tables

vineri, 31 ianuarie 2025 18:50

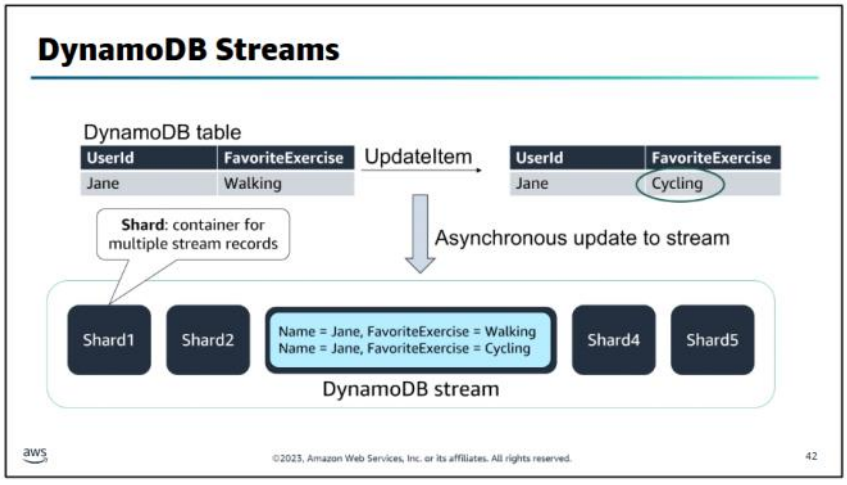
Applications can benefit from capturing changes to items that are stored in a DynamoDB table at the point when these changes occur.

Can enable **DynamoDB Streams as a solution of this type of use case.**
This optional feature captures a time-ordered sequence of item-level modifications in any DynamoDB table (called a stream). It then store this information for up to 24 hours.
Applications can view the data items as they appeared both before and after they were modified, in near-real time.

A stream consists of stream records.
Each stream record represents a single data modification in the DynamoDB table that the stream belongs to.

- Each record is assigned a sequence number, which reflects the order that the record was published to the stream
- When an app creates/updates/deletes items in the table, DynamoDB Streams writes a stream record.
- It records the primary key attributes of the items that were modified.
- Can be configured to record extra information, such as the before and after images of modified items

Stream records are organized into groups or shards.
Each shard acts as a container for multiple stream records, and it contains information that is required for accessing and iterating through these records.
The stream records in a shard are automatically removed after 24 hours.



[Change data capture for DynamoDB Streams - Amazon DynamoDB]

Global tables

- Offer a solution to deploy a multi-Region, multi-active database
- You specify the AWS Regions where you want the table to be available

DynamoDB automatically spreads the data and traffic for your tables over a sufficient number of servers to handle your throughput and storage requirements.

You can use global tables to keep DynamoDB tables synchronized across AWS Regions.

DynamoDB performs all of the necessary tasks to create identical tables in these Regions, and to propagate ongoing data changes to all of them.

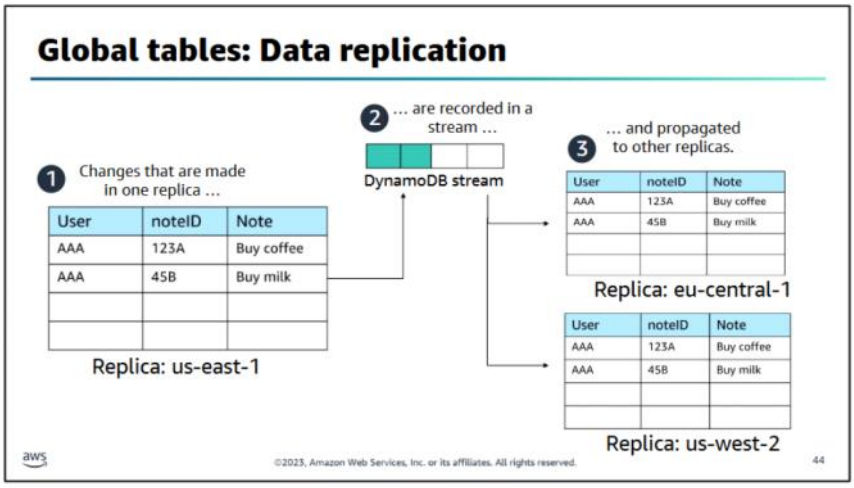
Benefits:

- When one region becomes temporarily unavailable, you can still access the same table data in the other Regions (in contrast to storing multiple distinct table by region)

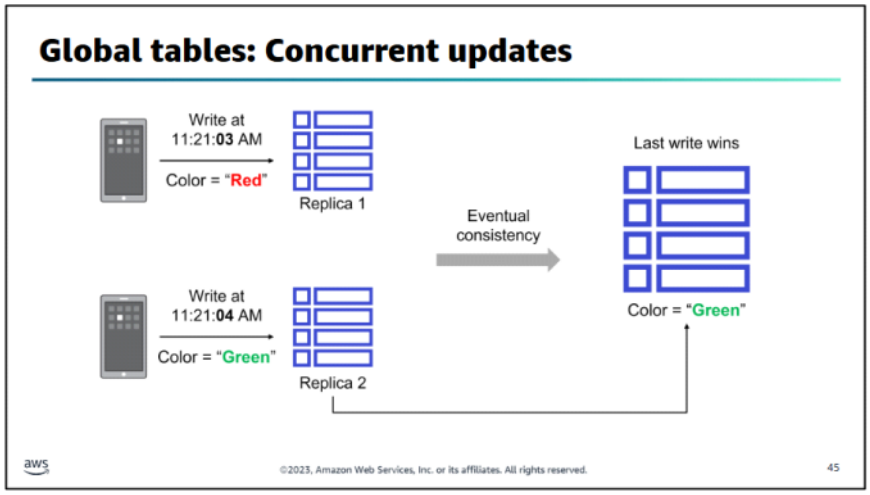
Global table = collection of one or more DynamoDB tables (called replica tables)
Replica table = single DynamoDB table that functions as a part of a global table

Each replica stores the same set of data items.
Any given global table can only have one replica table per Region. Each replica has the same table name and the same primary key schema.

Global tables use DynamoDB Streams to propagate changes between replicas.
A newly added item is usually propagated to all replica tables within seconds.
DynamoDB does not support partial replication of only some of the items.



Concurrent updates:
Conflicts can arise if applications update the same item in different Regions at about the same time.
To achieve eventual consistency, DynamoDB global tables use a *last writer wins* reconciliation.
All replicas will agree on the latest update and they then converge toward a state where they all have identical data.



Read consistency:
An app can read and write data to any replica table.
If the app uses eventually consistent reads, and only issues reads against one AWS Region, then it will work without any modification.
If the app uses strongly consistent reads, then it must perform all of its strongly consistent reads and writes in the same Region.
DynamoDB does not support strongly consistent reads across AWS Regions.

Strongly consistent reads require the use of a replica in the same Region where the client is running.

Transactions are enabled for all single Region DynamoDB tables, and they are disabled on global tables by default. You can choose to enable transactions on global tables by request, but replication across Regions is

asynchronous and eventually consistent. You might observe partially completed transactions during replication to other Regions.

Backup and restore

vineri, 31 ianuarie 2025 19:23

On-demand backups

- DynamoDB provides on-demand backup and restore capabilities.
- Can use DynamoDB to create full backups of your tables so that you can meet the requirements for long-term data retention and archiving for any data regulations that you must follow
- Can restore a DynamoDB table data anytime in the AWS Console or through an API call
- Backup and restore actions run with little impact on table performance or availability
- When created, a time marker of the request is cataloged.
- Backup is created asynchronously by applying all changes until the time of the request to the last full table snapshot.
- Backup requests are processed near instantaneously, and become available for restore within minutes.
- Each time an on-demand backup is created, the entire table data is backed up.
- No limit exists to the number of on-demand backups that you can take
- All backups in DynamoDB work without consuming any provisioned throughput on the table

Point-in-time recovery

- Helps protect DynamoDB tables from accidental write operations or delete operations
- With point-in-time recovery, you don't have to worry about creating, maintaining or scheduling on-demand backups
- If, for example, a script writes accidentally to a production DynamoDB table, point-in-time recovery allows to restore that table to any point in time during the last 35 days
- DynamoDB maintains incremental backups of your table

Basic operations for DynamoDB tables

vineri, 31 ianuarie 2025 19:31

Using DynamoDB API can invoke the following types of operations from an app:

- Control operations:
  - Used to create and manage DynamoDB tables
  - Work with indexes, streams and other objects that depend on tables
- Data operations:
  - Perform create/read/update/delete actions on data in a table.
  - Some data operations also allow you to read data from a secondary index
- Batch operations:
  - Run multiple operations against DynamoDB
  - Developers can run a query, compute some value from the result of the query and then write back to DynamoDB
- Transaction operations:
  - Make coordinated all-or-nothing changes to multiple items both within and across tables

Basic operations on DynamoDB tables - Amazon DynamoDB

Creating a table

- CreateTable (operation)
- Asynchronous operation
- When DynamoDB receives a CreateTable request, it immediately returns a response with a TableStatus of CREATING
- After table is created, DynamoDB sets the TableStatus to ACTIVE
- Can perform read and write operations only on an ACTIVE table
- Optionally, you can define secondary indexes on the new table as part of the CreateTable operation

Creating an item

- PutItem (operation)
- Creates a new item or replaces an earlier item with a new item
- If an existing item in the specified table has the same primary key as the new item, the new item completely replaces the existing item
- Can perform conditional PUT operation (add new item if an item with the specified primary key doesn't exist)
- Can replace an existing item if it has certain attribute values
- Can return the item's attribute values in the same operation by using the ReturnValues parameter

Updating an item

- UpdateItem (operation)
- Used to update an existing item's attributes or add a new item to the table if it doesn’t already exist
- You add, set or remove attribute values
- Can perform conditional update on an existing item (ex: replace an existing name-value pair if it has certain expected attribute

values)
- Can return the item's attribute values in the same operation by using the ReturnValues parameter

Delete an item
- DeleteItem (operation)
- Used to delete an item in a table by using its primary key
- Can perform conditional delete operation that deletes the item if it has an expected attribute value
- Can return the item's attribute values in the same operation by using the ReturnValues parameter

Condition expressions
- For the PutItem, UpdateItem and DeleteItem operations, you can specify a condition expression to determine which items should be modified
- If the condition evaluates to true, the operation succeeds, otherwise fails

Reading an item
- GetItem (operation)
- Used to retrieve a specific item from a DynamoDB table
- Must specify the table name and full primary key (partition key and sort key, if any) to retrieve a single item from a table
- Optionally, can specify a projection expression to retrieve only certain attributes instead of retrieving the entire item (by default, all attributes of the item are returned)
- Can also request that the operation use a strongly consistent read instead of the default eventually consistent read

Querying a table
- Query (operation)
- Used to read only the items that match the primary key from a table or secondary index
- The primary key is specified in the key condition expression
- If a filter expression is specified, then the Query operation further refines the result set based on the filter
- Must explicitly specify the name of the table or secondary index that you want to query
- Returns a result set with the items that match the conditions that were specified
- If no items satisfy the give criteria, returns empty result set

Scanning a table
- Similar to Query operation, but reads all items from the table or index
- Result set can be refined by using a filter expression
- Expensive operation since reads all items
- More efficient to perform a Query operation that has appropriate key condition expression to return only the data that the app needs
- If must do a Scan operation, perform a parallel scan when possible

Optimizing performance and cost
- Can limit the amount of data that Query and Scan return
- Two factors affect the number of items that a Query or Scan

operation returns:
    - Pagination limit:
        - By default, DynamoDB divides the results of Query and Scan operations into pages of data that are 1 MB in size (or less)
        - An app can process the first page of results, then the second page, and so on
        - A single Query or Scan will only return a result set that fits within the 1 MB size limit

    - Limit parameter value
        - Can limit the number of items that Query and Scan return in the result
        - To achieve this, set the Limit parameter to the maximum number of items that you want

Batch operations

- BatchGetItem
    - Read up to 16 MB data that consists of up to 100 items from multiple tables

- BatchWriteItem
    - Write up to 16 MB of data that consists of up to 25 PUT or DELETE requests to multiple tables
    - Individual items to be written can be as large as 400 KB

- If one request in a batch fails, the entire operation does not fail
- Retry with failed keys and data that is returned by the operation

Transactional

- TransactWriteItems
    - Contains a write set
    - Includes one or more PutItem, UpdateItem and DeleteItem operations across multiple tables
    - Can optionally check for prerequisite conditions that must be satisfied
    - If any condition is not met, the transaction is rejected

- TransactGetItems
    - Contains a read set
    - Includes one or more GetItem operations across multiple tables
    - If a TransactGetItems request is issued on an item that is part of an active write transaction, the read transaction is canceled
    - To get the previously committed value, can use a standard read

- If one operation fails, the entire transaction fails

API Gateway

joi, 20 februarie 2025 12:48

Fully managed service that provides API proxy and simplifies API development.

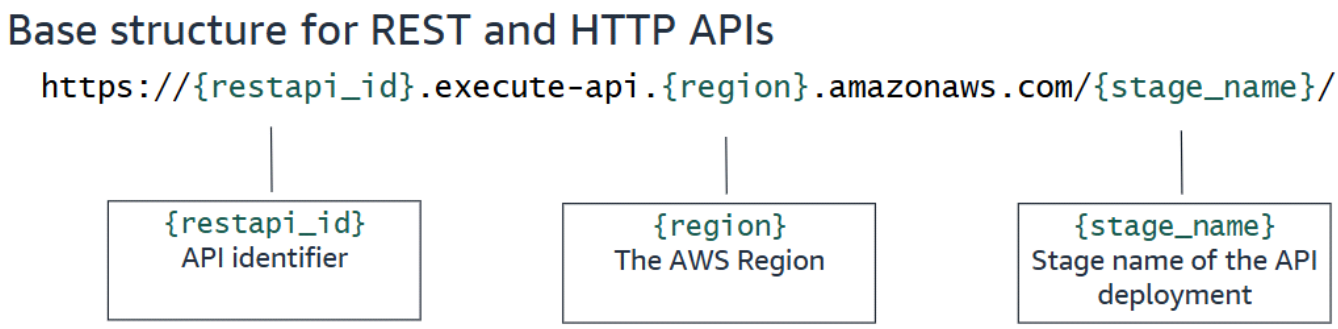
Two types of RESTful APIs in API Gateway:

- Rest API
  - Full control over API requests and responses
  - Have fine-grained control over highly customized authentication options
  - Can set up mock endpoints

- HTTP API
  - Simplifies the development of APIs that require only API proxy functionality
  - Designed for lowest cost and lowest latency

Creating APIs

joi, 20 februarie 2025 12:48



Greedy path variable: {proxy+}

- Set up a single Lambda integration that absorbs any nested paths included on API calls
- Example: https://apiurl.com/route-name/{proxy+} will handle requests like: https://apiurl.com/route-name/34/stuff/more/evenmore/stuff