**AWS DynamoDB Terraform Module Documentation**

**Overview**

AWS DynamoDB is a fully managed NoSQL database service provided by Amazon Web Services (AWS). It offers fast and predictable performance with seamless scalability. This Terraform module allows users to configure and deploy AWS DynamoDB tables efficiently, incorporating attributes, indexes, encryption, and various configurations.

**Pre-requisites**

Before using this module, ensure that you have the following:

* An AWS account with appropriate IAM permissions.
* Terraform installed and configured.
* A JSON configuration file defining the necessary parameters for the DynamoDB table.

**Resources Created by the Module**

This module provisions the following AWS DynamoDB resources:

* DynamoDB Table
* Table Attributes
* Local Secondary Indexes (LSI)
* Global Secondary Indexes (GSI)
* Time to Live (TTL) Configuration
* Point-in-Time Recovery
* On-Demand Throughput
* Server-Side Encryption
* Table Replication (if applicable)
* Tags for resource management

**Key Features and Configuration**

**DynamoDB Table Configuration**

The Terraform module defines the following key attributes for the DynamoDB table:

* **Table Name**: Defined dynamically using var.data.name.
* **Billing Mode**: Supports PAY\_PER\_REQUEST or PROVISIONED.
* **Primary Key Configuration**: Includes hash\_key and optional range\_key.
* **Capacity Settings**: If PROVISIONED, it includes read\_capacity and write\_capacity.
* **Stream Configuration**: Enables DynamoDB Streams and sets stream\_view\_type.
* **Table Class**: Standard or STANDARD\_INFREQUENT\_ACCESS.
* **Deletion Protection**: Prevents accidental deletion.

**Attributes Definition**

The attribute block defines the schema attributes for the table, ensuring proper data indexing:

attribute {

name = "id"

type = "N"

}

attribute {

name = "title"

type = "S"

}

attribute {

name = "Timestamp"

type = "S"

}

**Indexing**

**Local Secondary Index (LSI)**

Local secondary indexes allow querying within the same partition key:

local\_secondary\_index {

name = "TimestampSortIndex"

range\_key = "Timestamp"

projection\_type = "ALL"

}

**Global Secondary Index (GSI)**

Global secondary indexes enable queries across different partitions:

global\_secondary\_index {

name = "TitleIndex"

hash\_key = "title"

projection\_type = "ALL"

}

**Point-in-Time Recovery**

Ensures backup and restore capability:

point\_in\_time\_recovery {

enabled = var.data.point\_in\_time\_recovery\_enabled

}

**Time-to-Live (TTL) Configuration**

Specifies an attribute to automatically expire items:

ttl {

enabled = true

attribute\_name = "ttl-attribute"

}

**On-Demand Throughput Configuration**

on\_demand\_throughput {

max\_read\_request\_units = "1"

max\_write\_request\_units = "1"

}

**Server-Side Encryption**

Supports AWS KMS encryption:

server\_side\_encryption {

enabled = true

kms\_key\_arn = var.data.server\_side\_encryption\_kms\_key\_arn

}

**Replication Across Regions**

Defines multi-region replication for high availability:

replica {

region\_name = "us-east-1"

kms\_key\_arn = null

propagate\_tags = false

point\_in\_time\_recovery = false

}

**Timeouts Configuration**

Defines create, update, and delete timeouts:

timeouts {

create = "10m"

delete = "60m"

update = "20m"

}

**Tagging Support**

Ensures proper resource categorization:

tags = {

"Terraform" = "true"

"Environment" = "staging"

"Name" = var.data.name

}

**Module Usage**

The module uses a JSON input configuration (dynamodb\_config.json) that defines the table's attributes and parameters. Example:

{

"data": {

"name": "my-table-test1",

"stream\_enabled": false,

"stream\_view\_type": "NEW\_AND\_OLD\_IMAGES",

"create\_table": true,

"hash\_key": "id",

"range\_key": "title",

"billing\_mode": "PAY\_PER\_REQUEST",

"ttl": { "attribute\_name": "ttl-attribute" },

"point\_in\_time\_recovery\_enabled": true

}

}

**Provisioning Steps**

1. **Clone the Repository**:
2. git clone https://github.com/example/terraform-dynamodb.git
3. cd terraform-dynamodb
4. **Modify Configuration**: Edit dynamodb\_config.json based on requirements.
5. **Initialize Terraform**:
6. terraform init
7. **Apply the Configuration**:
8. terraform apply -var-file=dynamodb\_config.json
9. **Verify the Table**: Check the AWS console for the provisioned DynamoDB table.

**Best Practices**

* Use **AWS KMS** for encryption to secure sensitive data.
* Enable **point-in-time recovery** for data protection.
* Optimize **read and write capacities** based on workload requirements.
* Use **Terraform state locking** to prevent concurrent modifications.

**Troubleshooting**

* Ensure IAM policies allow **DynamoDB resource creation**.
* Validate **KMS key permissions** if using encryption.
* Confirm **IAM roles have DynamoDB full access**.

**Conclusion**

This Terraform module simplifies AWS DynamoDB provisioning, providing flexibility and automation for managing NoSQL databases in AWS. With support for attributes, indexes, encryption, and replication, users can deploy scalable and secure database solutions efficiently.