**Provision AWS Infrastructure for Claim Check Solution Using Terraform**

**Overview of the problem**

The application team needs to process a type of request that can take several minutes to complete. The input payloads will range in size and can be arbitrarily large. They’ve decided to use a messaging architecture for these requests and would like new infrastructure setup in their cloud provider. They intend to follow the claim check pattern and you've been asked to set up an object storage container that payloads will be written into. Whenever an object is added, an entry that references the new object must be added into a message queue managed by the cloud provider. The application team will design their application to work against the cloud provider's message queue.

**Solution summary**

**Graphical user interface, application

Description automatically generated**

I’ve designed and implemented a solution that will allow a user or application to upload an object (file) into an AWS S3 bucket. As soon the file is being uploaded to an S3 bucket, an event for that action is also created. The solution filters for that file upload to S3 event and sends an event notification SQS. So, while the file is being place in the S3 bucket, metadata about that file such as the filename and the S3 bucket it is stored in is being place into an SQS Queue. Now the developers can have their application poll the SQS Queue for the objects in S3 bucket and process them whenever the application is ready. To simulate that as well as to give the developers another place to view what is in the S3 bucket, I wrote a Lambda function in Python, that would periodically poll the SQS Queue, every 5 minutes and upload info about each file into a DynamoDB database table.

**Pre-requisites for implementation of the solution**

The following is required to setup the environment in order to implement this solution.

1. An AWS Account.
2. The AWS account must have the proper IAM Role and Polices that will allow Terraform the permission to create and destroy all of the AWS Resources needed to implement the solution. This includes the ability to create and destroy IAM Roles, S3 buckets, SQS Queues, Lambda functions, DynamoDB, etc..
3. Setup AWS CLI locally on your laptop or desktop. Make sure to [configure](https://docs.aws.amazon.com/cli/latest/userguide/cli-chap-configure.html) your AWS CLI
4. Setup Terraform locally on your laptop or desktop.  For steps, see [Terraform downloads](https://www.terraform.io/downloads.html)
5. Create 2 buckets in AWS account in us-east-1 called
   1. “claim-check-global” stores the tfstate files for global resources
   2. “claim-check-env” stores the tfstate files for all 3 environments (test, dev, prod)

**Overview of some AWS Services used in this solution**

Simple Storage Service (S3) -- an object storage service that offers industry-leading scalability, data availability, security, and performance. Customers of all sizes and industries can use Amazon S3 to store and protect any amount of data for a range of use cases, such as data lakes, websites, mobile applications, backup and restore, archive, enterprise applications, IoT devices, and big data analytics. Amazon S3 provides management features so that you can optimize, organize, and configure access to your data to meet your specific business, organizational, and compliance requirements.

Simple Queue Service (SQS) – a fully managed message queuing service that enables you to decouple and scale microservices, distributed systems, and serverless applications. SQS eliminates the complexity and overhead associated with managing and operating message-oriented middleware, and empowers developers to focus on differentiating work. Using SQS, you can send, store, and receive messages between software components at any volume, without losing messages or requiring other services to be available.

Lambda Function – a [serverless compute](https://aws.amazon.com/serverless/) service that runs your code in response to events and automatically manages the underlying compute resources for you. These events may include changes in state or an update, such as a user placing an item in a shopping cart on an ecommerce website. You can use AWS Lambda to extend other AWS services with custom logic, or create your own backend services that operate at AWS scale, performance, and security. AWS Lambda automatically runs code in response to [multiple events](http://docs.aws.amazon.com/lambda/latest/dg/intro-core-components.html#intro-core-components-event-sources), such as HTTP requests via [Amazon API Gateway](https://aws.amazon.com/api-gateway/), modifications to objects in [Amazon Simple Storage Service](https://aws.amazon.com/s3/) (Amazon S3) buckets, table updates in [Amazon DynamoDB](https://aws.amazon.com/dynamodb/), and state transitions in [AWS Step Functions](https://aws.amazon.com/step-functions/).

DynamoDB –  a fully managed NoSQL database service that provides fast and predictable performance with seamless scalability. DynamoDB lets you offload the administrative burdens of operating and scaling a distributed database so that you don't have to worry about hardware provisioning, setup and configuration, replication, software patching, or cluster scaling. DynamoDB also offers encryption at rest, which eliminates the operational burden and complexity involved in protecting sensitive data.

**Walk thru of the solution**

Directly below you’ll see a directory tree for all of the Terraform files used to implement this solution. The Terraform was set up in a way that allows you to implement this in multiple environments. So, a developer can run this solution in separate test, dev and prod environments. These environments are separated by regions as the test environment is ran in us-west2, the dev environment in us-west-1 and the prod environment in us-east-1. I’m able to do this by taking advantage of Terraform Workspaces. Each environment is separated by region and has its own Terraform Workspace. Since I do not have multiple AWS accounts I did it this way. However, in a real-world scenario it would be best to use separate AWS accounts for each environment as per AWS best practices.

├── Provision\_AWS\_Infrastructure\_For\_Claim\_Check\_Solution\_Using\_Terraform.docx

├── env

│   ├── backend.tf

│   ├── dev.tfvars

│   ├── lambda2

│   │   └── sqs\_check.py

│   ├── lambda2.zip

│   ├── main.tf

│   ├── prod.tfvars

│   ├── provider.tf

│   ├── test.tfvars

│   └── vars.tf

├── global\_resources

│   ├── gbl\_main.tf

│   ├── global\_backend.tf

│   └── vars.tf

**To setup global AWS resources. This includes IAM Roles and Policies**

cd to ../ claim\_check/global\_resources$

terraform init

terraform plan

terraform apply

**To set up environment (test /dev /prod)**

* cd to ../ claim\_check/env$
* terraform init
* terraform workspace list
* terraform workspace new test
* terraform workspace select test
* terraform plan -var-file=test.tfvars
* terraform apply -var-file=test.tfvars

Note: If you’d like to setup an environment such as dev or prod run the same commands listed above except replace the word test for each step with either dev or prod accordingly.

In order to simulate a file being uploaded into a bucket by an application or user, now that the test environment is built out, do the following….

* Log into the AWS console and go to S3 resource click Buckets
* Click the bucket named “claim-check-test-bucket”. If you were in the dev or prod environments, you would look for the bucket named “claim-check-dev-bucket” or “claim-check-prod-bucket” accordingly
* Upload a file or several files
* To check SQS first change the current AWS Region to us-west-2. For dev use the us-west-1 region and for prod use the us-east-1 region accordingly.
* Go to the SQS resource section and click on queues
* Click on the queue named “claim-check-s3-event-notification-queue”
* Click on send and receive messages
* Scroll down and click on poll for messages
* You should now see the SQS messages in the queue

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

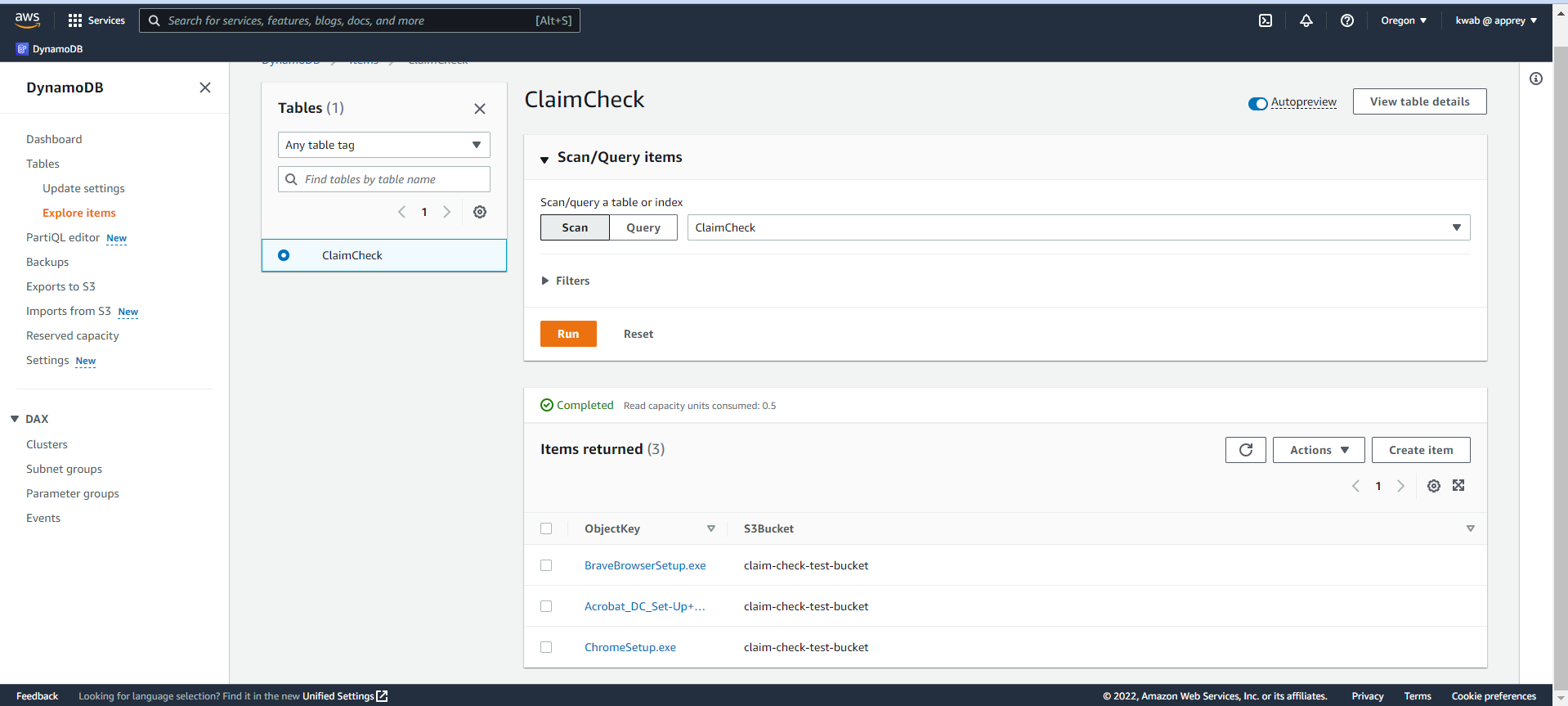
**Testing**

As a way to test this solution, I’ve created a Lambda Function that periodically runs a python script that will check the SQS queue every 5 minutes and will copy the object key and bucket details into a DynamoDB table. This is to simulate an application that will poll a queue after some buildup and some time.

* Add objects to s3 buckets as instructed above and wait 5 minutes to simulate a queue building up.
* Then to check DynamoDB Table first change the current AWS Region to us-west-2 if you are not there already. For dev use the us-west-1 region and for prod use the us-east-1 region accordingly.
* Go to the DynamoDB resource section and click on Tables

Click on the table named “ClaimCheck”

* Click on explore table items
* Click run and you should see a table whose items include columns consisting of the Object Key listing the title of each object in the bucket from the SQS queue messages as well as the bucket name.



**Future Improvements**

The items listed below should be implemented prior to using this in production. I began implementing some of these items but ran into issues. So, in the interest of time these items can be implemented later while working in the test environment.

* Enable encryption on S3 bucket
* Make S3 bucket private
* Enable encryption on SQS queue
* Add a Lambda Function python script or s3 bucket lifecycle policy that would delete the objects in the s3 bucket after they’ve been proceed.
* Add feature to delete SQS messages and DynamoDB table items after being procced by app
* Use separate AWS accounts for test, dev, prod environments