All Articles

How to Migrate an existing DynamoDB Table to a Global Table

Note: Since this article was published, AWS has added the ability to add regions to an existing table. Thus, the advice around migrating to a global table is less useful. The principles are still useful whenever you need to make a schema change or migration in your existing table.

Amazon DynamoDB is a fully-managed NoSQL database that's exploding in popularity. It provides low-latency reads and writes via HTTP with low maintenance in a way that fits with high-scale applications.

At re:Invent 2017, AWS announced DynamoDB Global Tables. With Global Tables, you can write to a DynamoDB table in one region, and AWS will asynchronously replicate items to the other regions.

This feature allows you to make reads and writes in the region closest to your user — allowing for lower latency — without manually managing cross-region replication. This is a pretty big deal.

There's one problem with DynamoDB Global Tables — you can't change an existing table to be a Global Table. A Global Table needs to be completely empty during configuration.

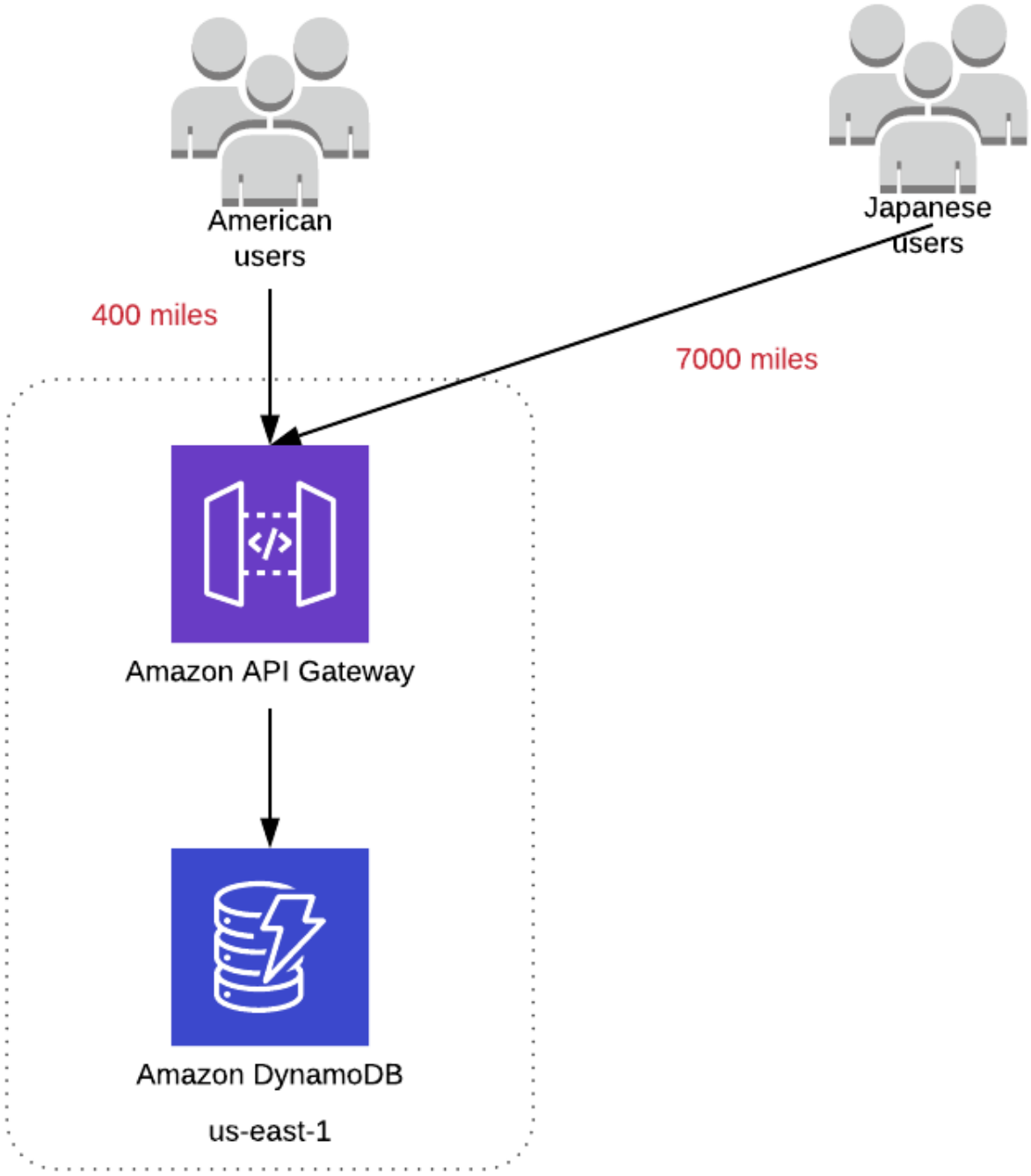
In this post, we'll learn how to migrate an existing table to a new Global Table. It requires some coding, and one of the things on my #awswishlist is that AWS will provide an automatic mechanism to make this migration easy. Until then, you can use this.

Background

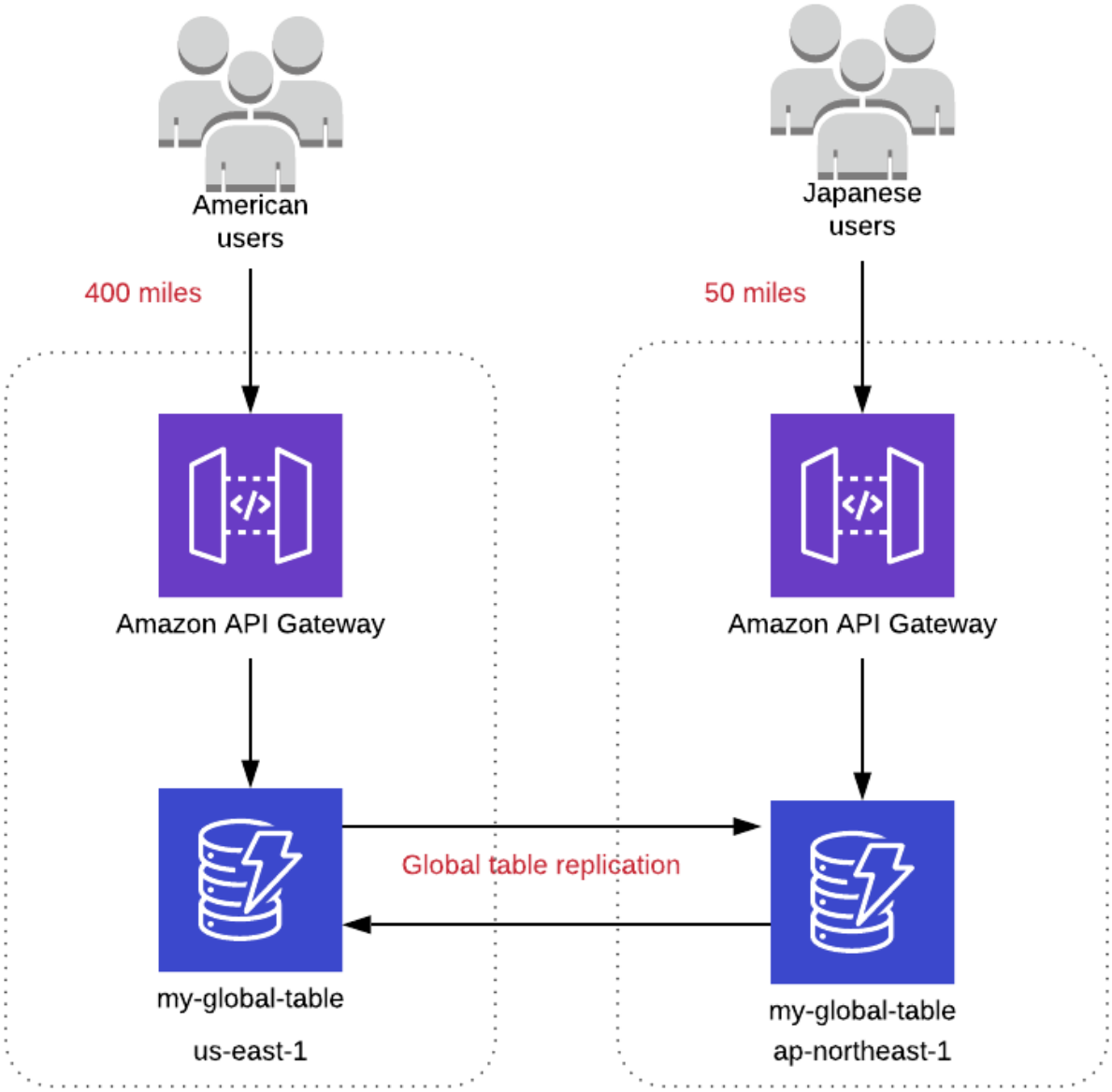
Before we dive into the architecture, let's set up the situation.

Imagine you have an existing DynamoDB table with >1 million items located in the `us-east-1` region of AWS. This table has served you well, but your users in Japan have been

complaining about the latency. Each read and write from Japan needs to cross the ocean, the Rocky Mountains, and the great state of Nebraska to make it to AWS datacenters in Northern Virginia.



To solve this problem, you want to set up a second copy of your architecture in the `ap-northeast-1` region in Tokyo:



To make this transition, you'll need to migrate all items from your existing, single-region table into a new Global Table.

Walkthrough: Migration to a Global Table

In the walkthrough below, I'll show you how to migrate an existing DynamoDB table to a Global Table. For the walkthrough, I'll use `my-table` for the name of the existing table and `my-global-table` for the name of the new table.

Step 1: Create a Global Table

The first step is to create your new Global Table.

I'm a strong proponent of infrastructure-as-code (IaC). The two most popular IaC tools for working with AWS resources are CloudFormation, a service provided by AWS and Terraform, an open-source tool created by Hashicorp.

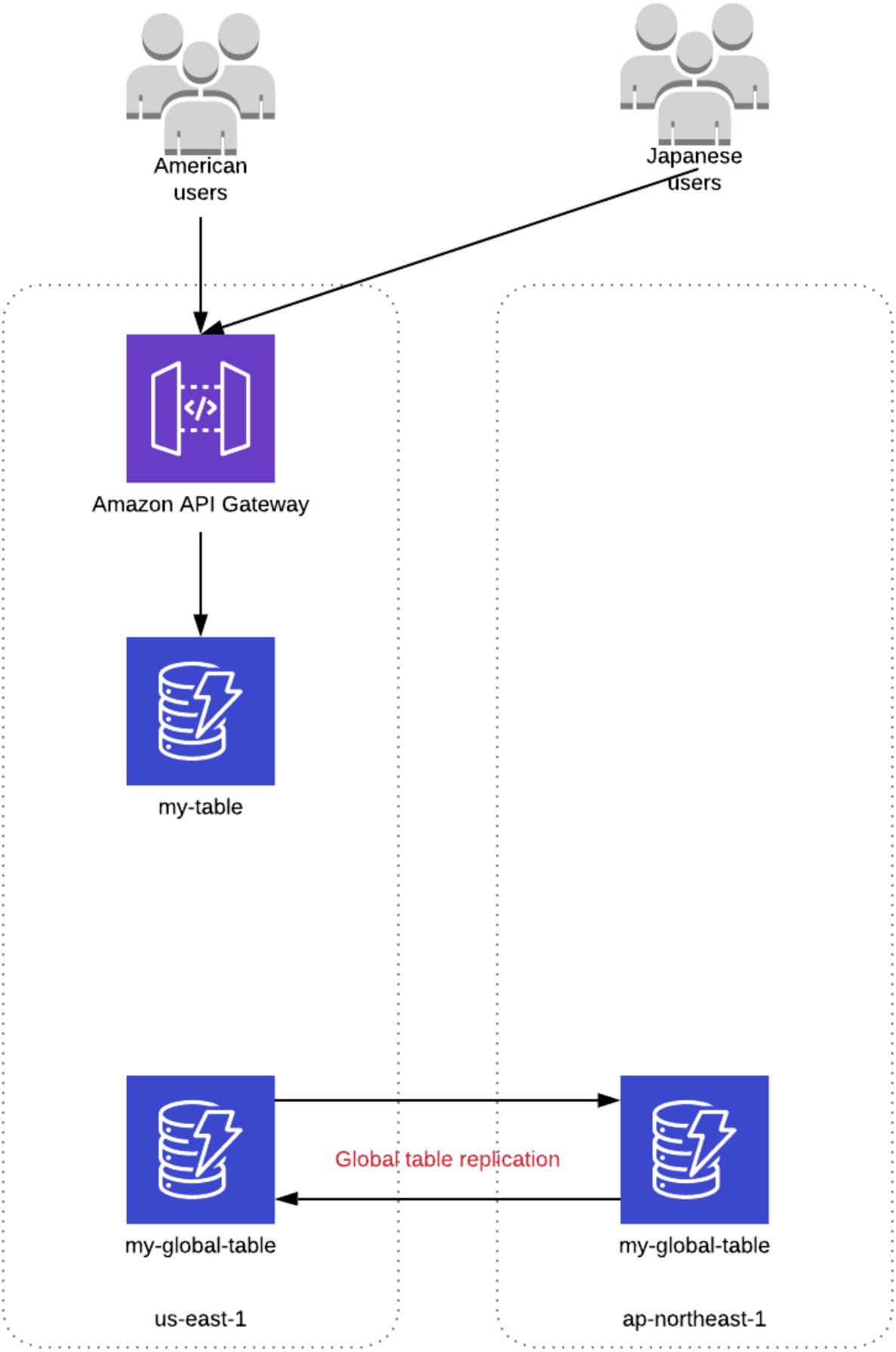
With Terraform, you can create your DynamoDB Global Table just like any other Terraform resource.

With CloudFormation, it's a little trickier. CloudFormation doesn't support Global Tables as a resource, so you'll need to do a bit of manual work.

For CloudFormation, you should:

1. Provision your base DynamoDB tables in each of the regions you want.
2. Use the `create-global-table` command in the AWS CLI to turn your base tables into a global table.

At the end of this step, our architecture looks as follows:


American users
Japanese users
Amazon API Gateway
my-table
Global table replication
my-global-table
my-global-table
us-east-1
ap-northeast-1

Both American and Japanese users are still hitting our us-east-1 API and using our original table. We have a Global Table set up with instances in both `us-east-1` and `ap-northeast-1`, but they don't have any items in them.

Step 2: Streaming updates to our Global Table

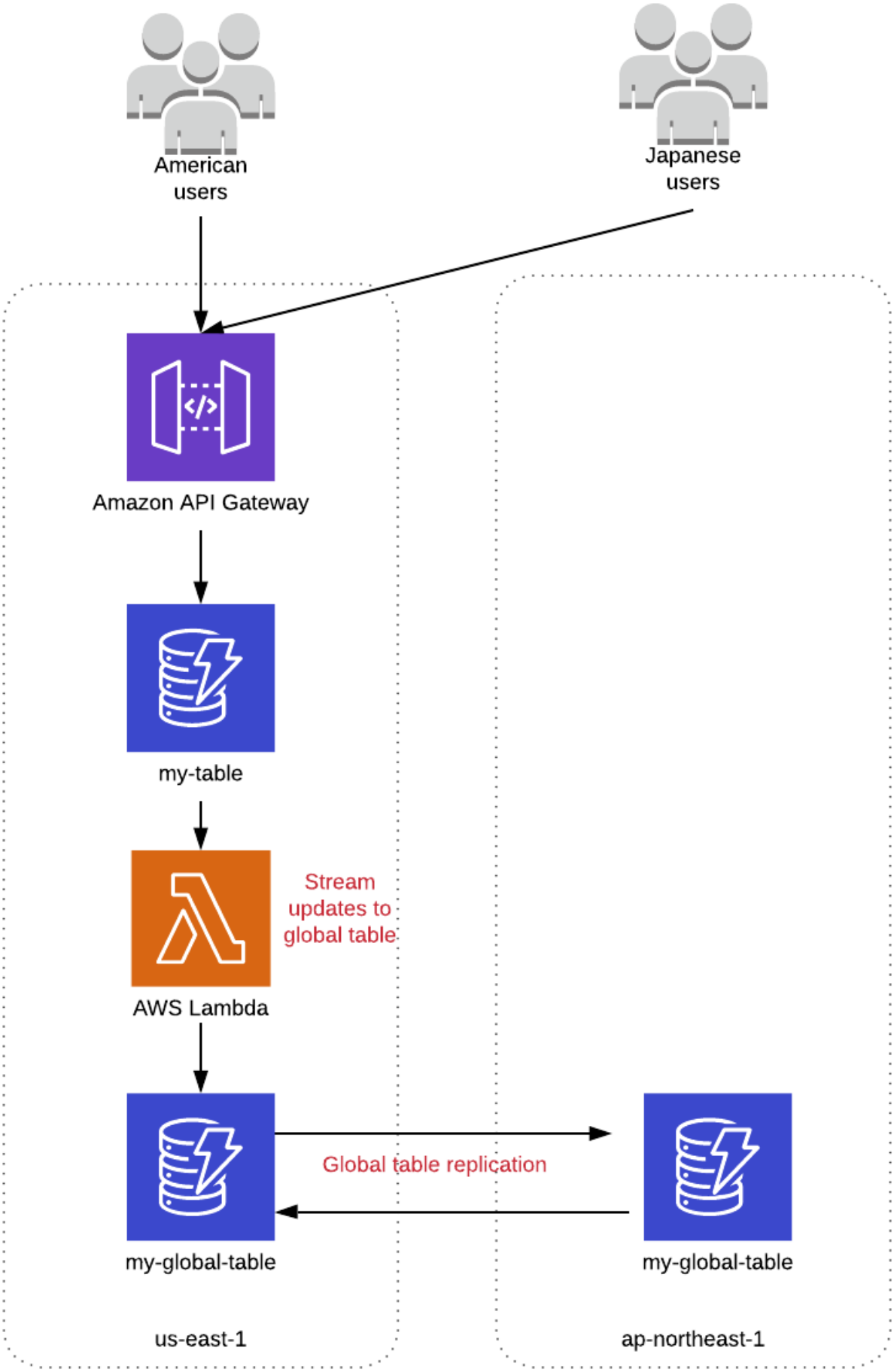
Now that we have our Global Table configured, we're going to use DynamoDB Streams to replicate all writes from our existing table to our new Global Table.

DynamoDB Streams is a feature you can enable on your DynamoDB table which gives you a changelog of your DynamoDB table in a time-ordered sequence. Every update that happens on your table — creating a new item, updating a previous item, deleting an existing item — is represented in your DynamoDB stream.

You can then configure an AWS Lambda function to consume the DynamoDB Stream. In your function, copy the corresponding change from the original table to your new Global Table.

Important: As you copy these changes, make sure you have an *updatedAt* field or some property to indicate when the item was last updated. This will be necessary later on.

After this step, our architecture will look as follows:


American users
Japanese users
Amazon API Gateway
my-table
Stream updates to global table
AWS Lambda
Global table replication
my-global-table
my-global-table
us-east-1
ap-northeast-1

All reads and writes are still going through our original table. Any writes that happen are replicated to our Global Table via the DynamoDB stream.

Step 3: Backfilling existing items

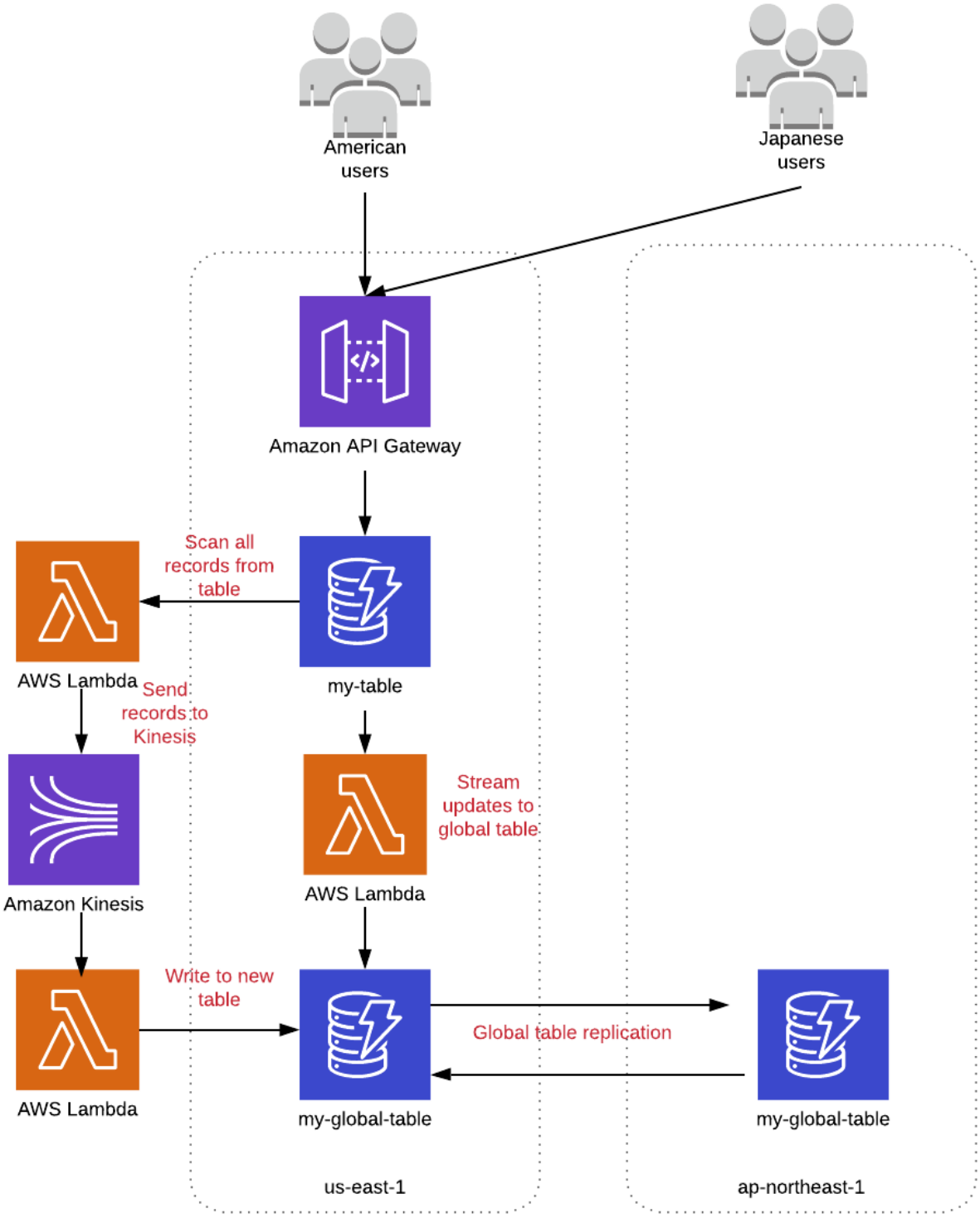
Now that we have our DynamoDB Stream configured, we are getting all updates from our existing table into our new Global Table. However, we still have a problem — all of our existing items that haven't been updated are not in our new table. We need to backfill these items into our new table.

This is the hardest part of the job. A backfill will require a full table scan of your existing table, which you usually want to avoid.

Further, you'll need to then write those items to your new table. This will require a lot of write throughput on your Global Table.

For this portion of the migration, I would recommend using DynamoDB On-Demand. This is a new feature announced at re:Invent 2018 where you don't need to capacity plan for your table's throughput. AWS will manage all the scaling for you on the backend at a slightly higher price than if you were managing throughput manually.

That caveat aside, let's dig into how we would accomplish this. The architecture would look as follows:



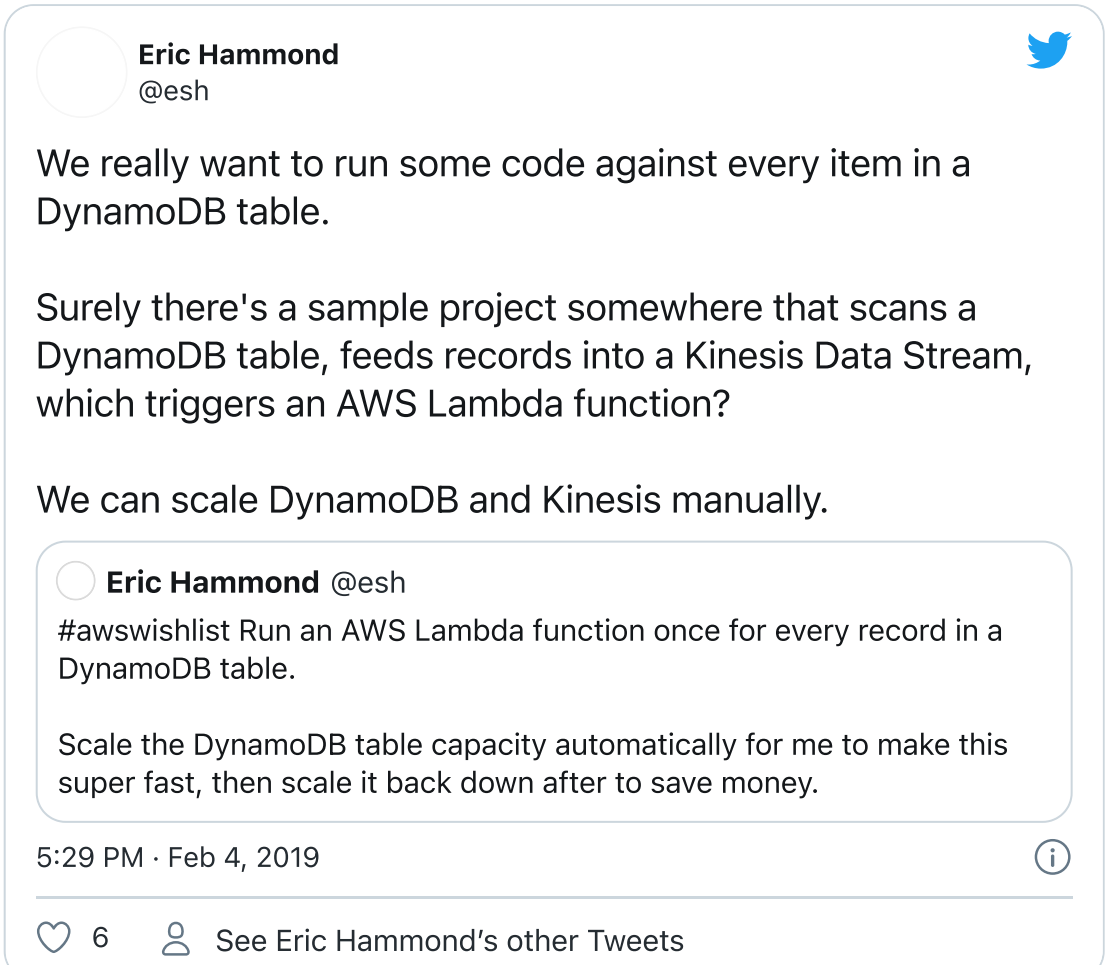
Pay attention to the elements on the left-most column of the architecture diagram. There are two things going on:

1. A Lambda function is scanning our existing table and writing each item to a Kinesis stream.

2. A second Lambda function is reading from the Kinesis stream and inserting the records into the new Global Table.

   Note: this second function should do a conditional put operation such that it only inserts the item if the updatedAt field on the item to insert is greater than the updatedAt field on the existing item (if any).

The first part of this design was inspired by a tweet from AWS Community Hero Eric Hammond:

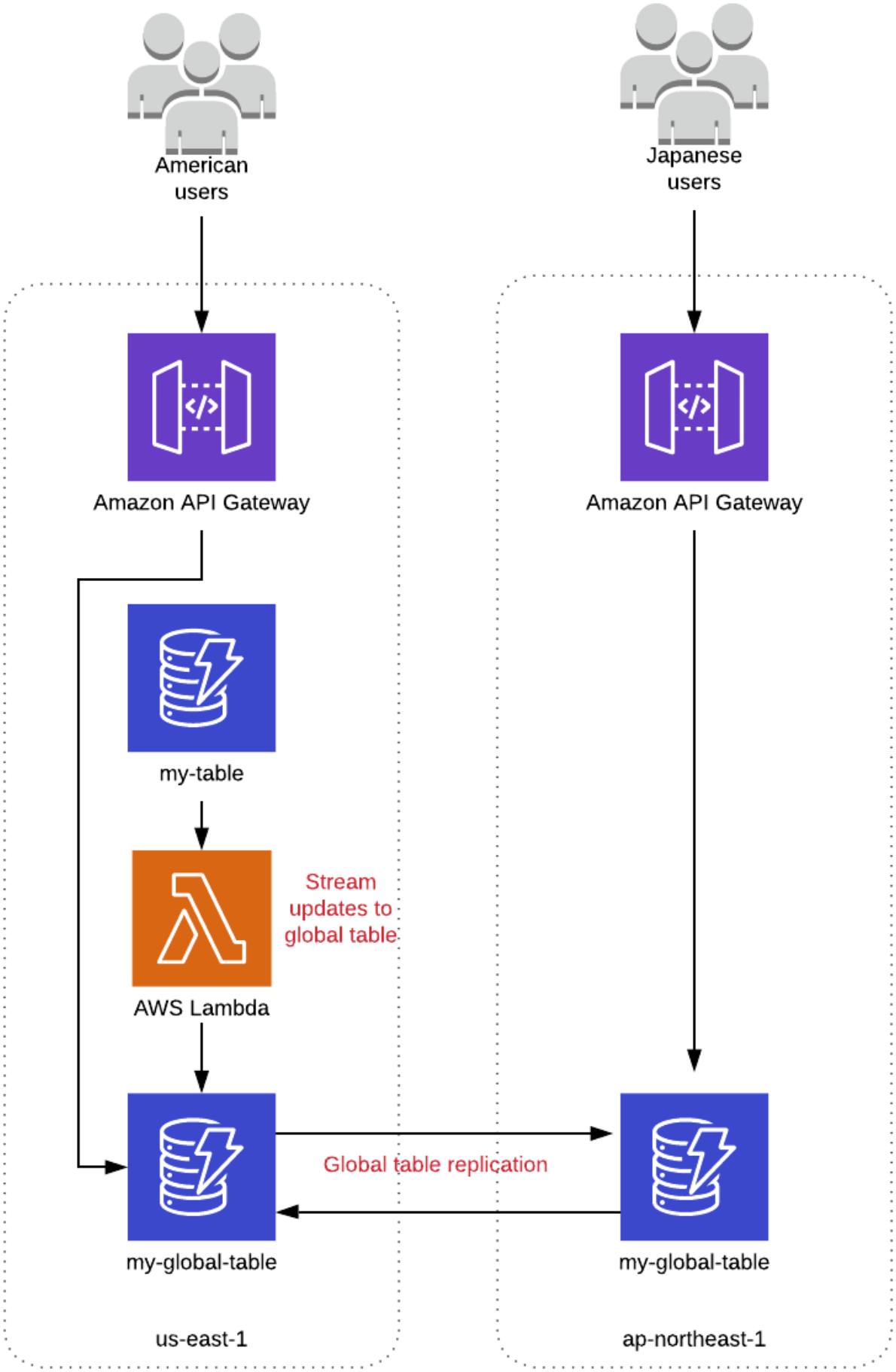


You can see an example of the architecture and code for the first element in my serverless-dynamodb-scanner project.

The Lambda function stores its location in the scan after each iteration and recursively invokes itself until the entire scan is done. As a result, you get a fully-serverless DynamoDB table scanner.

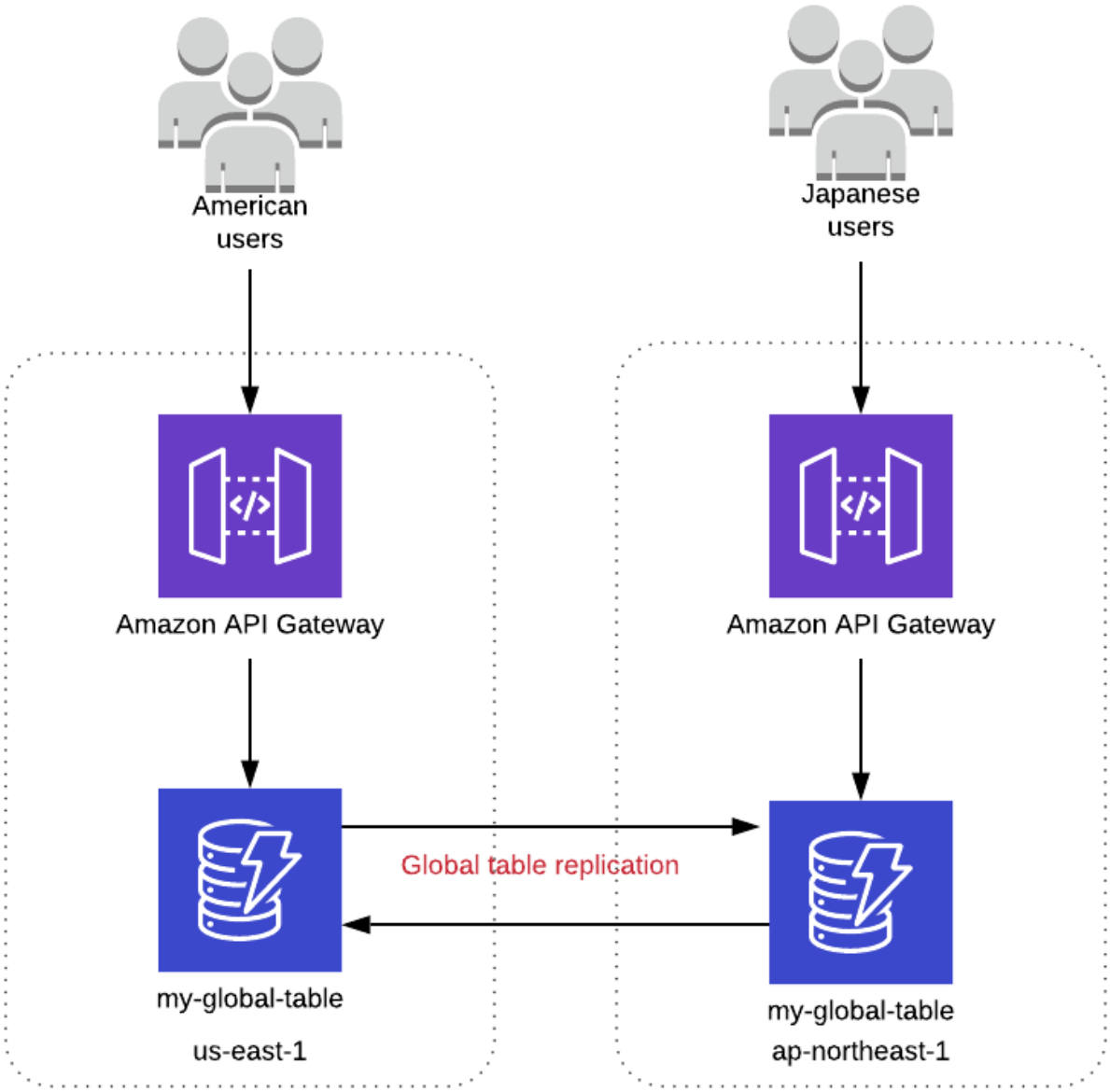
Step 4: Shift traffic to your global table

Once your backfill is complete, shift your traffic so that reads and writes come from your Global Table:


American users
Japanese users
Amazon API Gateway
Amazon API Gateway
my-table
Stream updates to global table
AWS Lambda
Global table replication
my-global-table
my-global-table
us-east-1
ap-northeast-1

Your American users are hitting your `us-east-1` endpoints and your Japanese users are hitting the `ap-northeast-1` endpoints.

Be sure to keep the existing table and DynamoDB stream replication up until all traffic has stopped going there and the stream is completely drained. Then, you can remove that infrastructure to simplify your architecture:



Ta-da! Your users are happy and you have a fully-managed global database! 🎉

Conclusion

In this article, we learned how to migrate an existing DynamoDB table to a Global Table in a fairly low-maintenance fashion.

I'd love it if this migration was seamless in the future. With the DynamoDB team's pace of innovation, I doubt this will be a manual process for long.

Published 5 Mar 2019

AWS DynamoDB Serverless

AWS Data Hero providing training and consulting with expertise in DynamoDB, serverless applications, and cloud-native technology.
Alex DeBrie on Twitter

ALSO ON **ALEXDEBRIE**

My DynamoDB Wish List

a year ago • 13 comments

In this post, I talk about my #awswishlist for DynamoDB, including …

The What, Why, and When of …

10 months ago • 31 comments

AWS recommends using just a single DynamoDB table for your entire …

A Detailed Overview of AWS API Gateway

2 years ago • 23 comments

Look inside the black box of AWS API Gateway to understand authorization, …

SQ Sc

a ye

Unc rela sca

What do you think?

6 Responses

👍 Upvote | 😝 Funny | 😍 Love | 😲 Surprised | 😤 Angry | 😥 Sad

2 Comments **alexdebrie** **Disqus' Privacy Policy** **1 Login**

Recommend Tweet Share **Sort by Best**

Join the discussion…

LOG IN WITH

OR SIGN UP WITH DISQUS ?

Name

uday • a year ago

https://aws.amazon.com/blog...

Reply • Share ›

Alex D Mod ➚ uday • a year ago

Nice catch, **@uday** ! Updated with a note at the top.

Reply • Share ›

Subscribe **Add Disqus to your siteAdd DisqusAdd** **Do Not Sell My Data**