Disaster Recovery for AWS IoT

Implementation Guide

Disaster Recovery for AWS IoT: Implementation Guide

Table of Contents

Implement a failover strategy for your AWS IoT devices with the Disaster Recovery for AWS IoT solution

AWS Solution Implementation Guide

Publication date: **May 2021**

The Disaster Recovery for AWS IoT solution provides a failover strategy for your AWS IoT devices. Customers with critical AWS IoT Core workloads can use this solution to store and process their data in a secondary AWS Region if the primary Region is not accessible by their devices.

This solution provides the following key features:

1. Replicates classic device shadows and registry settings by configuring a global Amazon DynamoDB table in the primary and secondary Regions.
2. Implements active-passive disaster recovery and provides tools to copy existing IoT devices from your primary Region (active) to your secondary Region (passive).
3. Uses Amazon Route 53 with health checks and traffic policies to direct traffic from primary to secondary Region in case of Region failover.

This implementation guide provides infrastructure and configuration information for planning and deploying the AWS Disaster Recovery for AWS IoT in the Amazon Web Services (AWS) Cloud. This guide assumes that all devices will be created and provisioned in the primary Region and includes links to AWS CloudFormation templates that launch and configure the AWS services required to deploy this solution using AWS best practices for security and availability.

The guide is intended for IT infrastructure architects, administrators, and DevOps professionals who have practical experience with IoT solutions in the AWS Cloud with emphasis on AWS IoT Core, AWS IoT Device Management, Amazon DynamoDB, AWS Lambda, and AWS Step Functions.

Cost

You are responsible for the cost of the AWS services used while running this solution. As of May 2021, the cost for running this solution for a small workload with the default settings in the US East (N. Virginia) AWS Region is approximately **$70.14 per month**. Prices are subject to change. For full details, refer to the pricing webpage for each AWS service you will be using in this solution.

The cost of this solution depends on the following factors:

1. Number of devices created in the primary Region which are then copied to the secondary Region.
2. Number of classic device shadows created and updated in the primary Region.
3. Number of devices that are processed by the failed provision runner.

The cost of this solution also includes the following fixed costs:

1. $0.05 for launching and installing the solution using CodeBuild. (one time cost)
2. $5.00 / month ($2.50 / month / Region) for Route 53 health checks on the primary and secondary Regions.
3. $50 per policy record / month if you bring your own domain to Route 53.

This total cost does not include the costs associated with the normal operations of your IoT Core environment which consists of:

- Connectivity
- Messaging
- Device shadow and registry
- Rules Engine

Example 1: 72,000 devices created monthly and 150,000 device shadows monthly

AWS service	Dimensions	Cost (per month)
AWS IoT Core	Register 72,000 devices in secondary Region 150,000 device shadows 222,000 IoT rule invocations	$0.31
Amazon DynamoDB	222,000 provisioning with each request of the size of 2K. Failed provisioning for 10,000 devices	$2.00
AWS Step Functions	222,000 workflow requests (one state transition per workflow)	$5.50
AWS Lambda	444,000 calls invoked by IoT rules and Step Function workflows. Invocations of failed provisioning runner.	$7.33

AWS service	Dimensions	Cost (per month)
Amazon Route 53	Health checks on the primary and secondary Regions	$5.00
Amazon Route 53	Bring your own domain (optional)	$50.00
	Total cost:	**$70.14**

Example 2: 7.2M devices created monthly and 40M device shadows monthly

AWS service	Dimensions	Cost (per month)
AWS IoT Core	Register 7.2M devices in secondary Region. 40M device shadows 47.2M IoT rule invocations	$66.08
Amazon DynamoDB	47.2M provisioning with each request of the size of 2K. Failed provisioning for 10,000 devices.	$27.50
AWS Step functions	47.2M work flow requests (one state transition per workflow)	$118
AWS Lambda	94.8M calls invoked by IoT rules and Step Function workflows. Invocations of failed provisioning runner.	$9.50
Amazon Route 53	Health checks on the primary and secondary Regions	$5.00
Amazon Route 53	Bring your own domain (optional)	$50.00
	Total cost:	**$276.08**

Architecture overview

Deploying this solution with the default parameters builds the following environments in the AWS Cloud:

- A setup to replicate device settings and shadows from the primary Region to the secondary Region.
- Amazon Route53 health checks that can be used for failover.

Apart from the deployments, this solution also provides a Region-to-Region sync to replicate existing devices.

This solution works in two AWS Regions, which are called primary and secondary Regions. Devices are created in the primary Region. Device settings and shadows in the primary Region are replicated in the secondary Region.

> **Note**
> When you create devices in the secondary Region, they will not be replicated in the primary Region.

If the primary Region is not available, devices failover to the secondary Region.

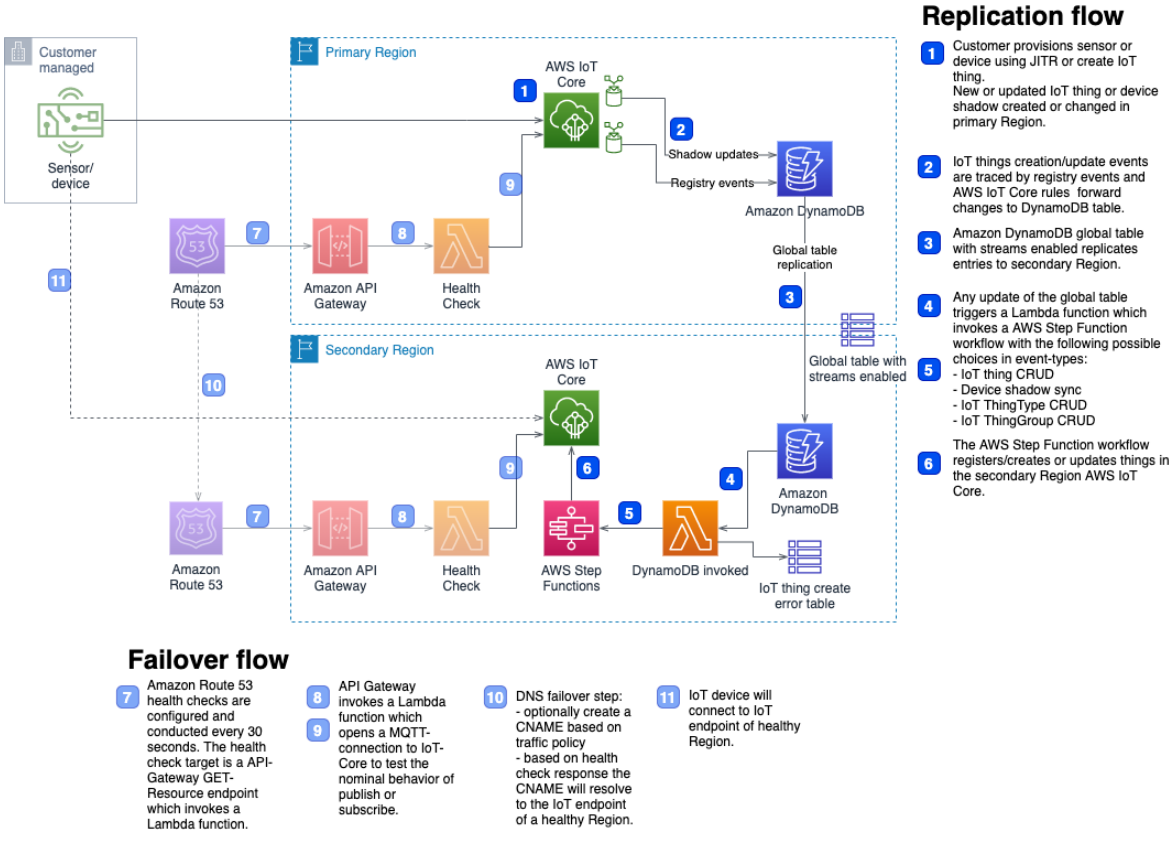

Figure 1: Disaster Recovery for AWS IoT solution architecture

In each Region, AWS CloudFormation templates deploy AWS IoT rules, AWS Lambda functions, an Amazon DynamoDB table, and AWS Identity and Access Management (IAM) roles necessary for device replication settings and to sync the shadow. In the secondary Region, an AWS Step Function setup is deployed additionally.

Replication flow

1. After the AWS CloudFormation templates have created a DynamoDB table in each of the Regions, these tables will be configured as one global table. You must turn on registry events in the primary Region.
2. The registry publishes event messages when IoT things, thing types, and thing groups are created, updated, or deleted. A topic rule forwards these messages to the DynamoDB table in the primary Region. They are automatically replicated to the table in the secondary Region.
3. DynamoDB streams captures the data on arrival in the secondary Region and invokes a Lambda function (`Dynamo trigger`).
4. The `Dynamo trigger` Lambda function initiates a Step Functions workflow to forward the related event types to another Lambda function.
5. The related Lambda function creates, updates or deletes several aspects of IoT things, thing groups, and thing types.
6. The Step Functions workflow creates or updates IoT things in the secondary Region. The Step Functions setup also includes retry rules to handle errors.

By default, X.509 certificates for device authentication are provided by AWS IoT Core. AWS Certificate Manager Private Authority (ACM PCA) can be used to optionally issue device certificates if you choose to use your own Certificate Authority (CA). ACM PCA is not deployed automatically with CloudFormation templates. Jupyter notebooks are provided with the solution to deploy ACM PCA in a Region where the service is available and to register the root CA in the primary and secondary Regions. There is no need to deploy ACM PCA in either the primary or secondary Region.

The device shadow sync is also deployed with the AWS CloudFormation templates for the primary and secondary Regions. The shadow sync architecture is based on the architecture for device replication. It uses the same global DynamoDB table and the same Step Functions workflow. Shadow messages that are accepted by AWS IoT Core are published on the shadow update/accepted topic. Messages are stored with an IoT rule in the global DynamoDB table. These messages are forwarded to the `ChoiceEventType` Step Function workflow and routed to a Lambda function, which updates the device shadow in the secondary Region.

Failover flow

A separate set of AWS CloudFormation templates creates health checks that can be used by Amazon Route 53 in the primary and secondary Regions.

7. Amazon Route 53 with health checks and traffic policies can be used for a Region failover. For more information about failover options, refer to Solution components (p. 6). Amazon Route 53 currently only supports HTTP(s) or TCP health checks. This solution uses the health of the Message Queuing Telemetry Transport (MQTT) message broker from AWS IoT Core.
8. CloudFormation templates deploy an Amazon API Gateway resource, which calls a Lambda function. This Lambda function is configured as a device in AWS IoT Core. When invoked, the Lambda function connects to AWS IoT Core, and subscribes to a topic and publishes a configured number of messages. The Lambda function expects to receive the same number of messages to the topic it has subscribed to.
9. Amazon Route 53 health checks calls the API Gateway resource and tests the MQTT message broker implicitly. As a layer of security, the Lambda function receives a query string before it connects to the message broker. If the query string does not match, the Lambda function issues an error message. The expected query string is configurable.

Solution components

Provisioning options

This solution covers aspects of the device layer and replicates device settings from a primary to a secondary Region. AWS IoT offers multiple options to provision devices. IoT policies can be attached to device certificates or to IoT thing groups. IoT thing groups can be nested. The solution supports the following provisioning options.

- Replicating devices which have a certificate attached. An IoT policy is attached to the device certificate.
- Replicate device deletion from primary to secondary Region. If a device shadow exists, it will also be deleted.
- Syncing the device unnamed shadow from the primary to the secondary Region.
- Region-to-Region sync for existing setups. If you already have devices before you install the solution, you can use the sync to replicate devices from the primary to the secondary Region.
- Route 53 health checks in the primary and secondary Regions. They check the health of the MQTT message broker.
- Based on Route 53 health checks, you can create traffic policies for failover for your solution. For more information, refer to Amazon Route 53 (p. 8).

Amazon DynamoDB expiring items

Items stored in DynamoDB are copied from one Region to another. After they have arrived and processed in the secondary Region, they are no longer needed. To reduce the number of items stored in DynamoDB, they are expired by the DynamoDB Time to Live (TTL) feature. The **expires** attribute defines when items can be removed. The **expires** attribute for the global DynamoDB table is defined in the IoT rules by `(timestamp()/1000)+172800 as expires`.The number 172800 refers to when the item expires, which is after 48 hours. You can change this setting in the CloudFormation templates.

Region syncs

If you already have devices provisioned in your account, you have to replicate them manually after launching this solution. After the solution has been implemented, only newly created devices will be synced to the secondary Region. To sync all existing devices to the secondary Region, Regions syncs are provided.

These Region syncs and the Lambda function code have been copied to an S3 bucket in the primary Region. You can find the S3 location in the outputs section of your main CloudFormation stack under `LambdaS3Url`. The Region syncs are located in the subfolder `iot-dr-region-syncer`.

There are two types of Region syncs:

- `region-to-region`
- `region-to-ddb`

The Region syncs get their settings from environment variables.

Environment variables for the `region-to-region` sync:

- PRIMARY_REGION: primary Region
- SECONDARY_REGION: secondary Region
- SYNC_MODE: default 'smart': In smart mode, a device will not be synced if the IoT thing name can be found in the primary Region
- QUERY_STRING: query string to look up devices, defaults to `"thingName:*"`
- MAX_WORKERS: number of parallel threads to run, defaults to 10

Environment variables for the `region-to-ddb` sync:

- PRIMARY_REGION: primary Region
- SECONDARY_REGION: secondary Region
- SYNC_MODE: default 'smart': In smart mode a device will not be synced if the thing name can be found in the primary Region
- QUERY_STRING: query string to look up devices, defaults to `"thingName:*"`
- DYNAMODB_GLOBAL_TABLE: Name of the global Dynamo DB table where to put device information from the primary Region. You can find the name of the global DynamoDB table in the **Outputs** section of the CloudFormation stack in the primary Region under `GlobalDynamoDBTableName`

Both syncs get the devices from the primary Region from the registry index if indexing is turned on, otherwise by `list-things`. We recommend using registry indexing. If registry indexing is not turned on, the query string will be ignored.

The syncs are implemented in python and were tested with Python 3.7. They require the `boto3`, `dynamodb-json`, and `device_replication.py` libraries to be available on the system. `boto3` and `dyanmodb-json` can be installed with pip. The `device_replication.py` library can be found in the S3 bucket in the primary Region in the subfolder `iot-dr-layer`.

Run Region syncs

Region syncs can be run standalone, for example, on an AWS Cloud9 environment or an EC2 instance. They can also be run as Lambda functions; however, Lambda functions are limited to a runtime of 15 minutes. Syncs can also be run in a Docker container. To run the syncs in a serverless container environment, use AWS Fargate. If you use Fargate you must provide an IAM role for your task definition. For an example IAM role, refer to IAM Roles (p. 10).

region-to-region

The region-to-region sync `iot-region-to-region-syncer.py` gets devices from the primary Region and provisions these devices in the secondary Region.

To build a container, you can use the script `build-docker-image-r2r.sh`. This script builds a Docker image and uploads the image to Amazon Elastic Container Registry (Amazon ECR). Prior to uploading the image to Amazon ECR, you must create a repository and modify `build-docker-image-r2r.sh` to reflect your settings. You must also change the AWS_REGION and AWS_ACCOUNT to reflect your settings.

region-to-ddb

The region-to-ddb sync gets the devices from the primary Region, converts them into the format as registry events messages, and then stores them into the global DynamoDB table. This data is used to

sync devices from the primary to the secondary Region. Devices in the secondary Region are created by the Step Functions workflow.

The sync has been tested standalone on Cloud9, but should also be able to run in a Docker container or as Lambda function. To build a container use the script `build-docker-image-r2r.sh` and modify it to reflect your settings.

Example test results from sync runs

Region-to-region:

- Created 2,400 devices on Fargate. Fargate provisioned in region eu-west-2, primary region eu-central-1, secondary Region eu-north-1. `MAX_WORKERS` set to 20. This takes approximately 10 minutes to replicate devices.

Region-to-DDB:

- Scanned 2,400 devices and store 1,000 into DynamoDB. This takes approximately two minutes.
- All 1000 devices found in secondary Region. This takes approximately three minutes.

Amazon Route 53

Health checks from Route 53 and DNS failover can be used for failover from one Region to another. You can create a CNAME based on a traffic policy and based on health checks, the CNAME will resolve the IoT endpoint of a healthy Region. Your devices can then connect to the CNAME resolution.

Route 53 health checks are created by CloudFormation templates in the primary and secondary Regions. You can find the stacks in each Region and named `R53HealthChecker`*`YYYYMMDDhhmmss`*.

Create a traffic policy

To create a traffic policy with a CNAME, you must have your own domain. Use the following instructions to create a traffic policy for the `iot-dr-us.example.com` CNAME.

1. Sign in to the Amazon Route 53 management console.
2. Select **Traffic policies**.
3. Choose **Create traffic policy**.
4. In **Policy name**, use the example: `IoTDR-primary-region-secondary-region`
5. Choose **Next**.
6. In **DNS type**, select **CNAME: Canonical name**
7. Select **+ Connect to...** and select **Failover rule.**
8. Under **Primary** > **Health checks**, check the box next to **Evaluate target health.**
9. Select **+ Connect to...** and select **New endpoint**
10. In the **Value** field, enter your IoT endpoint from primary Region. For example, `1111111bbbbbbb-ats.iot.us-east-1.amazonaws.com`
11. Under **Primary** > **Health checks**, check the box next to **Evaluate target health**.
12. Select **+ Connect to...** and select **New endpoint**
13. In the **Value** field, enter your IoT endpoint from secondary Region. For example, `1111111bbbbbbb-ats.iot.us-west-2.amazonaws.com`
14. Choose **Create traffic policy.**

15Select a hosted zone from the domains you have registered. For example: `example.com`
16For the Policy record DNS name, use the example: `iot-dr-us`
17Choose **Create policy records**.

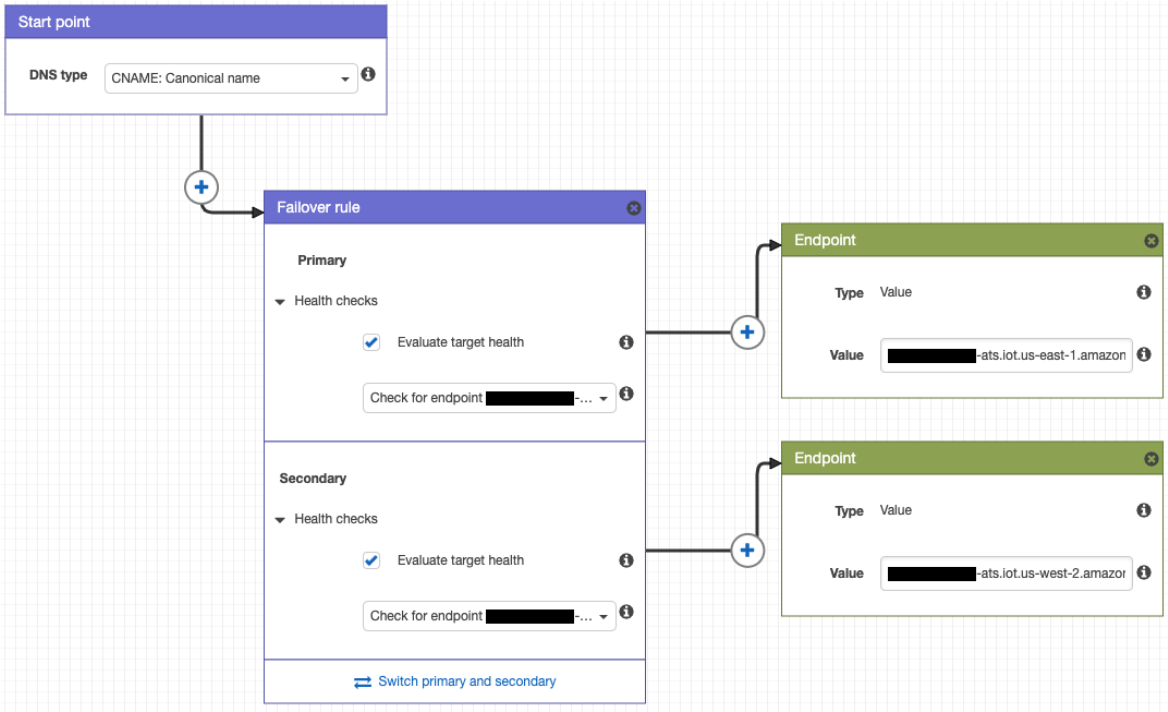

Figure 2: Disaster Recovery for AWS IoT Amazon Route 53 traffic policy

Security

When you build systems on AWS infrastructure, security responsibilities are shared between you and AWS. This shared model reduces your operational burden because AWS operates, manages, and controls the components including the host operating system, the virtualization layer, and the physical security of the facilities in which the services operate. For more information about AWS security, visit the AWS Security Center.

IAM roles

AWS Identity and Access Management (IAM) roles allows you to assign granular access policies and permissions to services and users on the AWS Cloud. This solution creates IAM roles that grant the solution's AWS Lambda functions access to create Regional resources.

You can find the IAM roles and policies in the CloudFormation templates that are used to deploy the solution.

Additional IAM roles

The following IAM roles are not created by CloudFormation templates. They are example roles that can be customized to your need.

IAM role AWS Fargate

The following role is used to run the region-to-region sync (p. 7) on AWS Fargate.

Trust relationship:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "",
      "Effect": "Allow",
      "Principal": {
        "Service": "ecs-tasks.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

Policy:

```
{
    "Action": [
        "dynamodb:DeleteItem",
        "dynamodb:DescribeTable",
        "dynamodb:GetItem",
        "dynamodb:PutItem",
        "dynamodb:Query",
```

```
            "dynamodb:UpdateItem"
        ],
        "Resource": "*",
        "Effect": "Allow"
    },
    {
        "Effect": "Allow",
        "Action": [
            "logs:CreateLogGroup",
            "logs:CreateLogStream",
            "logs:PutLogEvents"
        ],
        "Resource": "arn:aws:logs:*:*:*"
    },
    {
        "Effect": "Allow",
        "Action": [
            "iot:AddThingToThingGroup",
            "iot:AttachPolicy",
            "iot:AttachThingPrincipal",
            "iot:CreateDynamicThingGroup",
            "iot:CreatePolicy",
            "iot:CreateThing",
            "iot:CreateThingGroup",
            "iot:CreateThingType",
            "iot:DeleteCertificate",
            "iot:DeleteDynamicThingGroup",
            "iot:DeletePolicy",
            "iot:DeleteThing",
            "iot:DeleteThingGroup",
            "iot:DeleteThingType",
            "iot:DeprecateThingType",
            "iot:DescribeCertificate",
            "iot:DescribeThing",
            "iot:DescribeThingGroup",
            "iot:DescribeThingType",
            "iot:DetachPolicy",
            "iot:DetachThingPrincipal",
            "iot:GetIndexingConfiguration",
            "iot:GetPolicy",
            "iot:ListAttachedPolicies",
            "iot:ListPrincipalPolicies",
            "iot:ListPrincipalThings",
            "iot:ListThingGroupsForThing",
            "iot:ListThingPrincipals",
            "iot:ListThings",
            "iot:ListThingTypes",
            "iot:ListThingsInThingGroup",
            "iot:RegisterCertificateWithoutCA",
            "iot:RemoveThingFromThingGroup",
            "iot:SearchIndex",
            "iot:UpdateCertificate",
            "iot:UpdateThing",
            "iot:UpdateThingGroup",
            "iot:UpdateThingShadow"
        ],
        "Resource": "*"
    }
```

IAM role for Jupyter notebooks

The Jupyter notebooks provided with the solution call APIs from AWS IoT and AWS Certificate Manager private CA. To allows access to your IoT, use the following permissions for the environment where you run the notebooks.

```
{
    "Effect": "Allow",
    "Action": [
        "acm-pca:*",
        "iot:*"
    ],
    "Resource": "*"
}
```

Design considerations

IoT solution layers

A complete IoT setup or solution can consist of several layers. A disaster recovery setup affects every layer.

Device Layer

The device layer contains the following resources required to connect devices to AWS IoT Core:

- DNS (Amazon Route 53)
- IoT Endpoint
- Device registry
- Device certificates
- IoT policies
- Device shadows

Device settings required to connect devices to AWS IoT Core are replicated from a primary to a secondary Region. The solution provides two approaches to replicate devices, JITR- and complete mode.

JITR-mode

In JITR-mode, certificates are automatically registered and a device is being provisioned when it connects for the first time to an AWS IoT endpoint. With JITR-mode, only device registry settings will be replicated. When a device fails over to a secondary Region the certificate will be registered automatically and an IoT policy will be attached to the device certificate.

Complete provisioning mode

Upon device creation, the IoT thing and the associated certificates and policies will be replicated to the secondary Region. For registering certificates in the secondary Region, the multi-account registration feature (MAR) will be used. Certificates will be retrieved from the primary Region and registered in the secondary Region. Certificates in the primary Region are either issued by AWS IoT Core or by another CA like ACM PCA.

Registry events in the primary Region are used to track creation of devices. Registry information for IoT things, thing-groups and thing-types are replicated with a DynamoDB global table to the secondary Region. IoT things, thing-groups, or thing-types are then created in the secondary Region.

Registry events do not publish messages if a policy or certificate is being created. To retrieve this information, the certificate attached to a IoT thing and the policy attached to the certificate are retrieved from the primary Region after the device has been duplicated in the secondary Region.

A scheduled script will be run that determines if a cert/policy has been attached to the device. Devices with attached certificate and policy will be provisioned.

This script can be executed standalone in a container, for example, on AWS Fargate or as a Lambda function. To provision multiple devices, you can update the runtime limit of a Lambda function to restrict the usage of the Lambda function. It will run until all devices are provisioned.

Registry events constraints

- Merging information (`true | false`) for attributes for IoT things or thing groups are not provided as part of registry events
- Remove thing type information is not provided by registry events

Shadows

Shadows are replicated from the primary to the secondary Region, as shown in Figure 1.

Infrastructure layer

The infrastructure layer of an IoT solution consists of resources, such as IoT rules, Certificate Authorities, or settings. These resources are automatically deployed when you launch the solution in both Regions.

Storage/analytics layer

IoT data are stored or analyzed in the storage/analytics layer. This layer is automatically created when you launch the solution.

The infrastructure and storage/analytics layers are not automatically replicated because different disaster recovery scenarios can be created. You can create limitations with the same functionality in both Regions, or you can use the secondary Region to store your data in case of a disaster and merge it back later to the primary Region.

Performance testing

Before launching this solution, test the performance to ensure that service quotas are handled automatically. If service quotas are not handled, you can request a limit increase.

Before setting up failover, verify that the number of devices connected to the primary Region can also be used the secondary Region. We recommend working with your AWS Support team to have resources available in your primary and secondary regions.

Device replication has been tested with 30,000 devices and shadow replication has been tested with 20,000 shadows.

Device replication with bulk provisioning

To test replicating device settings from the primary to a secondary Region, bulk registration with 30,000 devices is used. To replicate devices, the solution uses the following Lambda functions:

- `DynamoTrigger` function: This function reads messages from the DynamoDB stream and invokes the `IoTDRSecondary`*`Timestamp`*`-SFNDynamoTriggerLambdaFu-`*`UniqueString`* Step Functions workflow.
- `ThingCrud`Lambda function: Replicates device settings in the `IoTDRSecondary`*`Timestamp`*`-SFNThingCrudLambdaFuncti-`*`UniqueString`* Step Functions workflow.

Monitor these Lambda functions for errors and also for concurrent invocations. Errors for the `ThingCrud` functions are handled by the retry configuration of the Step Functions workflow.

During the performance test, some API limits are throttled due to the parallel invocation of Lambda functions. This does not impact the replication of the devices. However, depending on the results of tests in your environment, you might want to consider requesting a quota increase.

For more information about running these Lambda functions, refer to Analyzing Log Data with CloudWatch Logs Insights in the *Amazon CloudWatch Logs User Guide*. The following list of example filters can be used to get more insights into the `ThingCrud` function:

- API throttling:

```
filter @message =~ filter @message =~ "ThingCrudException:
lambda_handler: An error occurred (ThrottlingException)"
```

- Runtime and memory used: `filter @message =~ "REPORT"`
- Timed out Lambda function: `filter @message =~ "Task timed out after"`
- Use this filter to for all of the above:

```
filter @message =~ "ThingCrudException: lambda_handler: An error occurred
 (ThrottlingException)" or @message =~ "Task timed out after" or @message =~ "REPORT"
```

Figure 3 displays the AWS IoT bulk provisioning results, which include:

- number of things for bulk provisioning: 30000
- time to generate keys and CSRs: 11086 secs.
- time for bulk provisioning: 6146 secs.

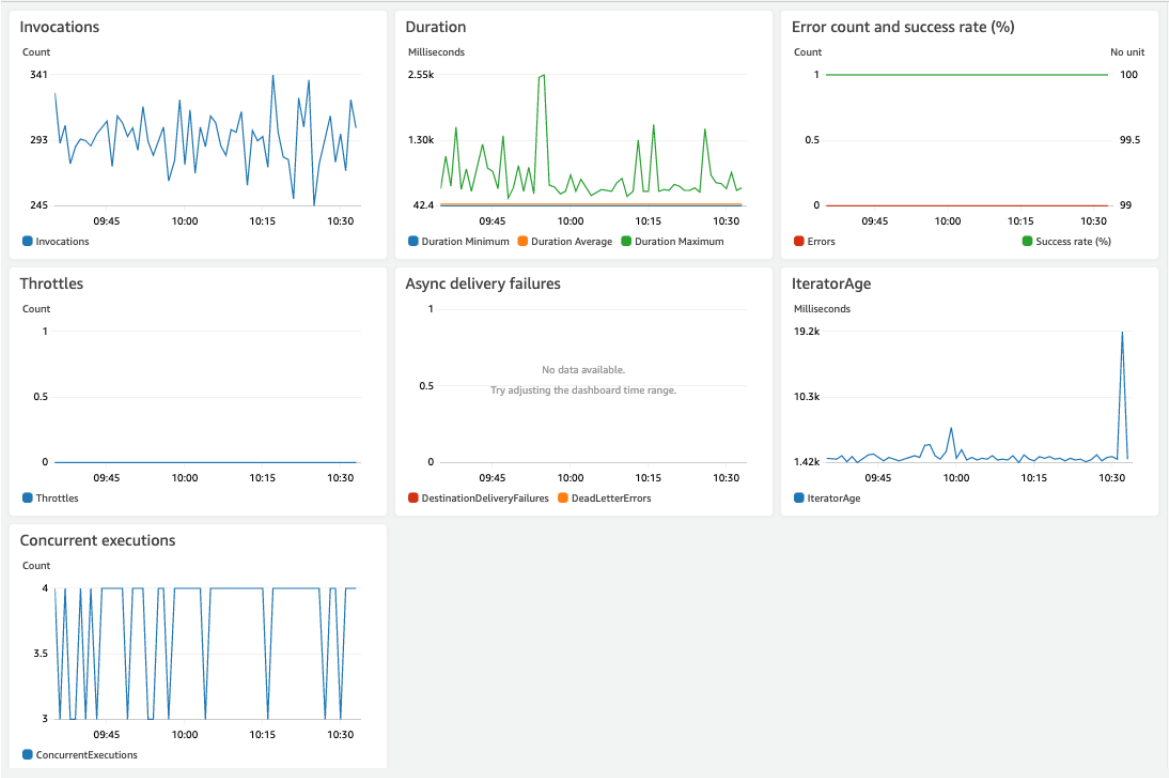

Figure 3: Disaster Recovery for AWS IoT monitoring DynamoTrigger Lambda function

Figure 3 displays typical behavior for the `DynamoTrigger` Lambda function when 30,000 devices are replicated.

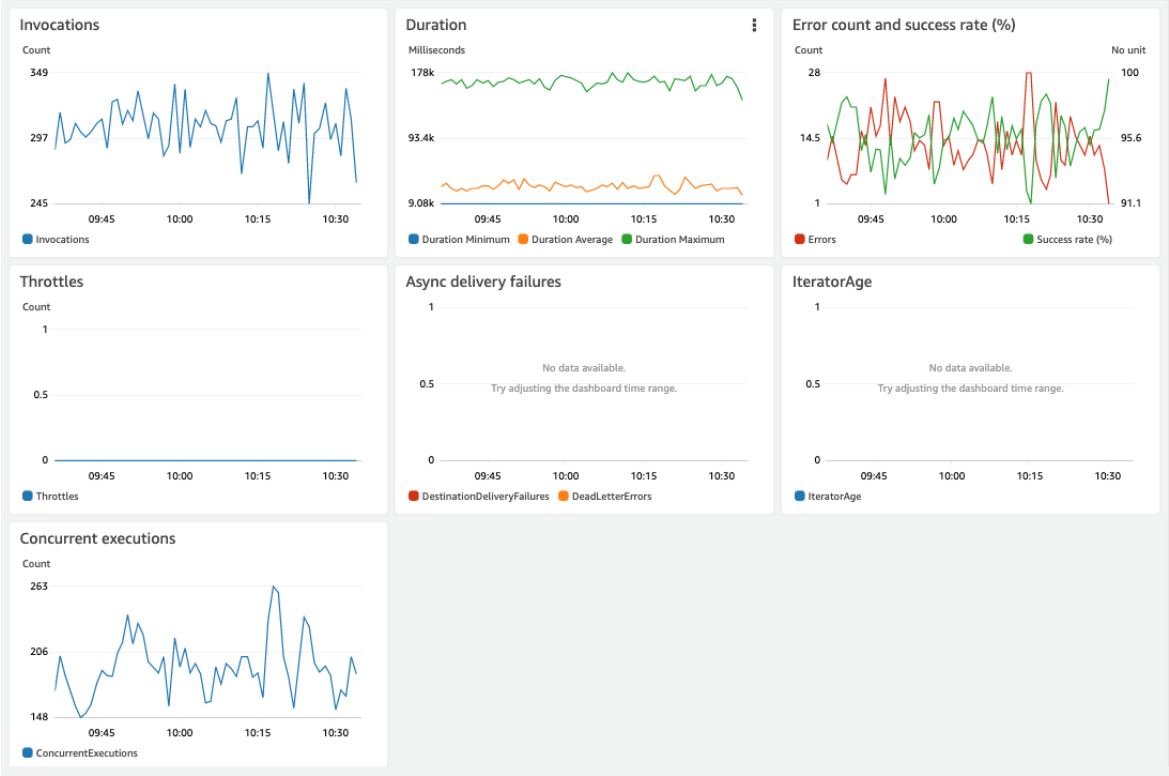

Figure 4: Disaster Recovery for AWS IoT monitoring ThingCrud Lambda function

Refer to Figure 4 for an example of monitoring the `ThingCrud` Lambda function when replicating 30,000 devices. The errors displayed are caused by reaching API limits where they are handled by the Step Functions retry workflow.

Shadow replication

Replicating the shadow has been tested with 20,000 device shadows with the tool `iot-dr-shadow-cmp.py` by using 40 parallel threads.

Results:

```
./iot-dr-shadow-cmp.py --primary-region us-east-1 --secondary-region us-west-2 --num-tests
 20000 --max-workers 40 |tee /tmp/sdw20k.log
2020-11-20 12:49:03,908 [INFO]: MainThread-iot-dr-shadow-cmp.py:156-<module>: cmp: start
2020-11-20 13:02:47,887 [INFO]: MainThread-iot-dr-shadow-cmp.py:221-<module>: cmp: stats:
 NUM_SHADOWS_COMPARED: 20000 NUM_SHADOWS_NOTSYNCED: 0 NUM_ERRORS: 0
2020-11-20 13:02:47,887 [INFO]: MainThread-iot-dr-shadow-cmp.py:223-<module>: cmp: stop
```

Testing tools

This solution uses testing tools that are copied to an S3 bucket in the primary Region. You can find the S3 location in the outputs section of your main CloudFormation stack under `ToolsS3Url`.

The tools are implemented as Bash scripts or in Python3 and have been tested on an Amazon Cloud9 environment. You can use them in any environment where Bash and Python3 is available. The Python scripts use the boto3 library or the AWS IoT SDK v2, which must be installed on your system.

If you want to capture the output of scripts not only to terminal but also to a file, use the *tee* command. For example: `$ ./script-name | tee /tmp/my.log`.

These test tools require setting environment variables. Under `ToolsS3Url`, you will find `toolsrc` file with several predefined environment variables. You can customize it to your needs and set the environment variables before using the tools.

The following scripts are provided to create, delete, and test devices.

> **Note**
> When you use these scripts to create devices you want replicated in the secondary Region, you must create the device in the primary Region. Only devices created in the primary Region will be replicated to the secondary Region.

- `bulk-bench.sh`: Creates a certain number of devices with bulk provisioning.
  - Prerequisites:
    - An IAM role that permits AWS IoT to provision devices on your behalf. The role ARN must be set to the environment variable `ARN_IOT_PROVISIONING_ROLE`. A role has been created already with CloudFormation during solution launch.
    - An S3 bucket that you can use for bulk provisioning. The bucket name must be set to the environment variable `S3_BUCKET`.
    - You can find both environment variables in the file `toolsrc`.

- `$ ./bulk-bench.sh <base_thingname> <num_things>`: The script creates a directory to store keys and CSRs. An IoT thing name is composed out of the `base_thingname` and an ongoing number. The name of the directory is `base_thingname-%Y-%m-%d_%H-%M-%S`.
  - After successful bulk provisioning, the script will download the resulting JSON file containing device certificates. The name of the JSON file is `base_thingname-%Y-%m-%d_%H-%M-%S/results.json`.

- `bulk-result.py`: This script extracts device certificates issued by bulk-bench.sh and writes them to the file system.
  - Run `$ bulk-result.py results.json`
- `script-name <THING_NAME>`: The following scripts create devices. They use the AWS CLI to provision devices. Some of the scripts will expect shell variables to be set. The scripts write device certificates and private and public keys to `$THING_NAME.certificate.pem`,`$THING_NAME.private.key`, `$THING_NAME.public.key`. By default, devices are created in the Region of your AWS CLI configuration. If you want to create devices in another Region, set the environment variable `AWS_DEFAULT_REGION` to the appropriate Region.
  - `create-device-attrs-type.sh`: creates a device with attributes and the thing-type `dr-type03`. You must create the thing-type before using the script.
  - `create-device-attrs.sh`: Creates device with attributes.
  - `create-device-pca.sh`: Creates a device with a device certificate issued by AWS Certificate Manager Private Certificate Authority (ACM PCA). A PCA can be created with the Jupyter notebooks provided by the solution. Environment variable: PCA_ARN must be set to the ARN of you private CA.
  - `create-device-type.sh`: Creates a device with the thing-type `dr-type03`. You must create the thing-type before using the script.
  - `create-device.sh`: Creates a device using the IoT policy from the document `sample-pol1.json`. The script replaces the Region in the `sample-pol1.json` policy. Environment variable `REGION` must be set to the AWS Region that you want to use.

  - Always create devices in the primary Region.
- `delete-things.py`
  - Prerequisite: Registry indexing must be turned on.
  - Run:

```
$./delete-things.py -region <your_region_where_devices_should_be_delted> --query-string
 “thingName:<device_pattern>”
```

  - For the IoT thing name, you can provide a single ioT thing name or use the “*” wildcard in combination with the part of a device name to match multiple devices.
  - Deletes devices that are matched by the query string. The certificate will be detached from the device. The policy associated with the certificate will only be deleted if no other principals are associated with the policy. The certificate will be deleted if no other things are associated with the certificate.
  - Always delete devices in the primary Region.

- `iot-devices-cmp.py`
  - Prerequisite: Registry indexing must be enabled
  - Run:

```
$ ./iot-devices-cmp.py -primary-region <your_primary_region> --secondary-region
 <your_secondary_region> --query-string “thingName:devicename*”
```

    Query-string is optional and defaults to `“thingName:*”`
  - Compares devices in the primary and secondary region to verify the same certificate and policy is attached in both regions to the certificate. The script is meant to work with one certificate and one policy attached to the certificate which has been replicated. It is not a general tool to cover multiple certificates or multiple policies attached to a device or certificate.

- `iot-dr-pubsub.py`: Based on the `pubsub.py` example from the AWS IoT Device SDK v2 for Python with two additional features.
  - Prerequisites: Optional Amazon Route 53 setup with health checkers and traffic policy. See description below in this document. The Python libraries *awsiotsdk* and *dnspython* must be installed.
  - Optional features:
    - *--cname*: instead of connecting directly to the provided endpoint do a CNAME lookup in DNS and connect to the resulting host name.
    - *--dr-mode:* Starts a separate thread which does a CNAME lookup regularly. If the result of the CNAME lookup changes it terminates the current connection to an IoT endpoint and reconnects to the new endpoint.
  - Run:

```
$ ./iot-dr-pubsub.py --endpoint <endpoint> --root-ca <file> --cert <file> --key <file>
 <optional_features>
```

  - Apart from publishing/subscribing the script can look up the CNAME of an IoT endpoint and connect to the result of the CNAME lookup. When the CNAME is created with a traffic policy in Amazon and `-dr-mode` is turned on, the script will failover to another Region in case health checks determine that the current active Region has failed. By using a TXT record in DNS, the script can determine if it is connected to a primary or secondary Region. To use this feature, create a DNS TXT record which points to `_YOUR_CNAME` and provide a JSON object in the following format: `{“primary”: “primary_region”, “secondary”: “secondary_region”}`. A sample lookup will result in the following answer:

```
host -t TXT _iot-dr-us.example.com
_iot-dr-us.example.com descriptive text "{\"primary\": \"us-east-1\", \"secondary\":
 \"us-west-2\"}"
```

- `iot-dr-shadow-cmp.py`
  - Run:

```
$ ./iot-dr-shadow-cmp.py -primary-region <region> --secondary-region <region> --num-
tests <number_of_shadows_to_test> --max-workers <default_10_max_50>
```

  - Test shadow replication. Shadows are created in the primary Region and compared with the shadow in the secondary Region to determine if shadow replication has been successful. After testing shadows will be deleted. Works parallelized to speed up runtime.

- `iot-search-devices.py`
  - Prerequisites: Registry indexing must be turned on.
  - Run `$ ./iot-search-devices.py -query-string <query_string>`
  - Searches all devices for a given query string. Not limited to a number of devices as it makes use of the `next_token` in an answer and continues to get devices. Useful to compare the number of devices in regions. Query string example to find all device which name starts with `iot-dr: “thingName:iot-dr*”`
- `list-thing.py`
  - Run `$ ./list-thing.py <thing_name>`
  - Looks up the given device name and the attached certificate and iot policy
- `sample-pol1.json`, `sample-pol2.json`
  - Sample IoT policies
- `test-dr-deployment.py`
  - Run `$ ./test-dr-deployment -primary-region <region> --secondary-region <region>`
  - An automated end-to-end test to show the working capabilities of the IoT DR Solution. It creates a device in the primary Region and verifies if the replication to the secondary Region has been successful. Afterwards it tests publish and subscribe in both Regions. Afterwards it tests shadow synchronization. When tests have been finished it deletes the resources that have been created.

Failover

This section discusses possible approaches for Region failover.

Use your own domain with CNAME

To use your own domain as CNAME pointing to AWS IoT endpoints, you can use Route 53 health checks for failover or implement a failover logic on your devices.

Route 53 health checks

Health checks are initiated from several AWS Regions, but they cannot determine if your devices are able to reach AWS IoT endpoints. There might be cases where your devices are not able to reach a Region, but health checks can.

Failover logic on devices

You can configure both endpoints for the primary and secondary Region on your devices. If a device cannot reach the primary Region, it can switch to the secondary Region. This device perspective approach always looks for your IoT endpoints.

Failback

Devices using a permanent MQTT-based connection must implement a failback strategy. Without failback strategy, they will stay connected to the failover Region even when the primary Region is already up again. If your devices detect when they are connected to a secondary Region, they can test in regular intervals if the primary Region is reachable again and fail back.

The sample `iot-dr-pubsub.py` implements a strategy to failover as soon as an endpoint change has been detected. If you use any mechanism on your devices to determine to which Region a device is connected, you can build your reconnection strategy upon such a mechanism. For more information, refer to Testing tools (p. 16).

Customizing this solution

This solution is mainly built on AWS IoT registry events to capture and replicate IoT thing, thing-group, or thing-type related settings. It does not cover certificates attached to IoT thing groups or nested thing groups or jobs. Also, by using registry events, it is not possible to track changes at device certificates or IoT policies.

To capture certificate and policy changes, use Amazon EventBridge to replicate them.

Replicating jobs requires some more investigations to determine if and how they can be replicated and handled in case of failover.

Regional deployments

This solution uses AWS IoT Core and Amazon DynamoDB. Both services must be available in the Regions where you deploy the solution. The CloudFormation template allows you to choose only Regions where these services are available.

If your AWS IoT environment also uses other services, you need to select an AWS Region where all of your services are available. For the most current availability by Region, refer to the AWS Service Region Table.

AWS CloudFormation templates

This solution uses AWS CloudFormation to automate the deployment of the Disaster Recovery for AWS IoT in the AWS Cloud. It includes the following CloudFormation templates, which you can download before deployment:

disaster-recovery-for-aws-iot.template: By launching this template (also referred to as main template) you initiate the launch of the whole solution. It creates a CodeBuild project and a Lambda function which starts the CodeBuild project. CodeBuild deploys the solution as such in the primary and secondary Regions.

> **Note**
> The CodeBuild project automatically launches these templates after you launch the `disaster-recovery-for-aws-iot` template.

primary-region.template: Creates resources in the primary Region. A DynamoDB table, topic rules to ingest messages from registry events and shadows into the DynamoDB table, Lambda functions for JITR and IAM roles with permission for AWS IoT and Lambda.

secondary-region.template: Creates resources in the secondary Region. A DynamoDB table with streams enabled, Lambda functions and a step function setup. When messages are ingested in the DynamoDB table a step function workflow is launched by a Lambda reading data from the DynamoDB stream. Step Functions replicate devices and shadows.

r53-health-checker.template: Launches the required AWS resources for Route 53 in the primary and secondary Region. Creates an Amazon API Gateway, a Lambda function which checks the MQTT message broker and a health check in Route 53. It also creates a device in AWS IoT Core which is used to perform MQTT health checks.

CodeBuild project: Apart from launching the CloudFormation template the CodeBuild project executes additional tasks to launch the solution.

- Creates Amazon S3 buckets in both Regions and copies CloudFormation templates and tools to these buckets.
- Creates a global DynamoDB table out of the tables in both Regions.

You can customize all templates to meet your specific requirements.

Automated deployment

Before you launch the solution, review the architecture, solution components, security, and design considerations discussed in this guide. Follow the step-by-step instructions in this section to configure and deploy the solution into your account.

Time to deploy: Approximately 20 minutes

Step 1. Launch the stack

This automated AWS CloudFormation template deploys Disaster Recovery for AWS IoT in the AWS Cloud.

> **Note**
> You are responsible for the cost of the AWS services used while running this solution. For more details, visit to the Cost (p. 2) section in this guide, and refer to the pricing webpage for each AWS service used in this solution.

The main template launches in the US East (N. Virginia) Region by default. Do not change this Region. The other templates will be launched in the primary and secondary Regions independently of the Region of the main stack.

1. Sign in to the AWS Management Console and select the button to launch the `disaster-recovery-for-aws-iot` AWS CloudFormation template.

   Launch Solution

   You can also download the template as a starting point for your own implementation.

By default, the stack name is `IoTDRSolution`. Consider using a more meaningful name, such as `IoTDRSolution-primary-region-secondary-region`. This helps to distinguish disaster recovery setups if you launch multiple stacks.

2. The template launches in the US East (N. Virginia) Region by default. To launch the solution in a different AWS Region, use the Region selector in the console navigation bar.
3. Choose `PrimaryRegion` and `SecondaryRegion`.
4. On the **Review** page, review and confirm the settings. Check the box acknowledging that template will create AWS Identity and Access Management (IAM) resources.
5. Choose **Create stack** to deploy the stack.

You can view the status of the stack in the AWS CloudFormation Console in the **Status** column. You should receive a CREATE_COMPLETE status in approximately 2 minutes.

After the main stack has been created, you can watch the progress of creating the other stacks at the logs of your CodeBuild project:

1. In the **Outputs** section of your main stack, select the link next to **CodeBuildLaunchSolutionProject**.
2. Under **Build history > Build run**, select the link with a name similar to `CodeBuildLaunchSolution-[A_UNIQUE_STRING]`

3. Select **Tail logs**. A window is opened where you can see the progress of the IoT DR solution setup. Several CloudFormation stacks are launched.

Every stack is postfixed with a timestamp in this format: `%Y%m%d%H%M%S`

- Stacks in primary Region:
  - `IoTDRPrimary`*`POSTFIX`*
  - `R53HealthChecker`*`POSTFIX`*
- Stacks in secondary Region:
  - `IoTDRSecondary`*`POSTFIX`*
  - `R53HealthChecker`*`POSTFIX`*

AWS CodeBuild should take approximately 10 minutes to finish. In the **Outputs** section of your main stack, you can find links to the CloudFormation stacks in the primary and secondary Regions.

> **Note**
> This solution includes helper Lambda functions, which only run during initial configuration or when resources are updated or deleted.
> When you run this solution, helper Lambda functions appear in the AWS console.
> Helper Lambda functions use tags, such as `CloudFormation:CustomResource` and `Solution:IoTDR`. You can find these functions in the AWS Lambda console by applying these tags as filters.

Testing the solution

Use the following steps to test the solution after the CloudFormation stacks have been launched.

Device replication

- Create a device in the primary Region.
- Verify that the device has been created with certificate and policy attached in the secondary Region. You can use the AWS CLI, AWS management console or the tool.
- Send and receive messages in both Regions to verify that the device works correctly in either Region. You can use a publish/subscribe of your choice by using the `iot-dr-pubsub.py` tool.

Device shadow replication

Test device shadow replication with the `iot-dr-shadow-cmp.py` tool provided or use your own method.

Sample walkthrough

This sample walkthrough used the tools provided by the solution. Multiple variables are used from the file `toolsrc` in the tools folder.

Copy the tools from your S3 bucket in the primary Region to your environment. You can find the S3 URL for tools in the **Outputs** section of the main stack under **ToolsS3Url**.

```
# Get the tools
mkdir tools
cd tools
aws s3 sync ToolsS3Url

# make scripts executable
chmod +x *.sh
chmod +x *.py

# get the Amazon Root CA
curl https://www.amazontrust.com/repository/AmazonRootCA1.pem -o root.ca.pem

# Source the environment variables
. toolsrc

# assign the thing name to be used to a shell variable
THING_NAME=dr-walkthrough

# create the device in the primary region
AWS_DEFAULT_REGION=$PRIMARY_REGION ./create-device.sh $THING_NAME

# list the device in primary and secondary region
AWS_DEFAULT_REGION=$PRIMARY_REGION ./list-thing.py $THING_NAME
AWS_DEFAULT_REGION=$SECONDARY_REGION ./list-thing.py $THING_NAME


# pub/sub in primary region
```

```
./iot-dr-pubsub.py --endpoint $IOT_ENDPOINT_PRIMARY --cert $THING_NAME.certificate.pem
 --key $THING_NAME.private.key --root-ca root.ca.pem --client-id $THING_NAME --topic dr/
$THING_NAME --count 2 --interval 1

# pub/sub in secondary region
./iot-dr-pubsub.py --endpoint $IOT_ENDPOINT_SECONDARY --cert $THING_NAME.certificate.pem
 --key $THING_NAME.private.key --root-ca root.ca.pem --client-id $THING_NAME --topic dr/
$THING_NAME --count 2 --interval 1
```

You can also use the MQTT test client in the AWS IoT Core console in the primary and secondary Regions to verify message arrival. Subscribe to `dr/$THING_NAME` in each Region. Replace `$THING_NAME` with the name of your device.

Shadow replication tool

You can use the `iot-dr-shadow-com.py` tool to test if device shadows are replicated from the primary to the secondary Region. Replicating a shadow from one Region to the other can take up to approximate 10 seconds. The tool tries to look up the shadow in the secondary Region immediately after it has been created in the primary Region. This attempt fails if the shadow has not been replicated yet. The tool has retry logic implemented and will try to get the shadow again between two to 10 seconds depending on the amount of retry attempts. You might see messages that include retry information, such as the following example:

```
compare_shadow: n: 2 thing_name: 69a14446-dc16-4f29-9db7-1e46875b5c02: no shadow payload,
 retrying in 4 secs.
```

Upon successful comparison of the shadows in both Regions, you will find the following example messages:

```
compare_shadow: i: 4 thing_name: 524a0e2d-7cdb-4def-9341-a5e22553df07 shadows match:
 temperature: 36 temperature_secondary: 36
```

To create five shadows and compare them, run the following command:

```
./iot-dr-shadow-cmp.py --primary-region $PRIMARY_REGION --secondary-region
 $SECONDARY_REGION --num-tests 5
```

Failover testing

To test failover with Amazon Route 53, you can use the tool `iot-dr-pubsub.py`. You must create a CNAME with a traffic policy before you start using the pub/sub sample subscriber with your endpoint and in `dr-mode`.

```
./iot-dr-pubsub.py --endpoint iot-dr-us.example.com --cert $THING_NAME.certificate.pem
 --key $THING_NAME.private.key --root-ca root.ca.pem --client-id $THING_NAME --topic dr/
$THING_NAME --count 0 --interval 5 --use-cname --dr-mode
```

You can invalidate the Route 53 health check in the primary Region by modifying the query string for the health check so that the health check fails. Use the following steps to modify the query string:

1. Sign in to the Amazon Route 53 management console.
2. Select **Health checks**.

3. Check the health check for your primary Region.
4. Edit the health check.
5. Delete the last character from the value for the **Path** field. Record this character so that you can revert the path to the original.
6. Choose **Save**.

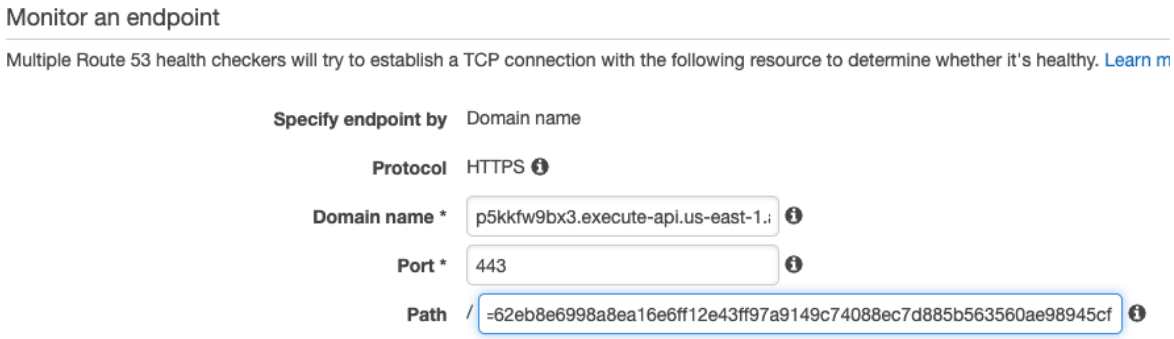

Figure 5: Disaster Recovery for AWS IoT Solution Make a health check fail

After a few minutes, the health check changes the state to **Unhealthy**.

When the health check in the primary Region changes to **Unhealthy**, Amazon Route 53 resolves your CNAME automatically to the secondary Region. This causes the script `iot-dr-pubsub.py` to detect a Region switch and reconnect to the secondary Region. You will then see messages similar to the following:

```
[INFO]: Thread-4-iot-dr-pubsub.py:143-dr_endpoint_verifier: REGION SWITCH detected:
 ENDPOINT_NAME-ats.iot.us-east-1.amazonaws.com -> ENDPOINT_NAME-ats.iot.us-
west-2.amazonaws.com
[INFO]: Thread-4-iot-dr-pubsub.py:145-dr_endpoint_verifier: teminating current
 MQTT_CONNECTION
[INFO]: Thread-4-iot-dr-pubsub.py:147-dr_endpoint_verifier: disconnect_future result:
 <Future at 0x7f7db1bd9240 state=pending>
[INFO]: Thread-4-iot-dr-pubsub.py:150-dr_endpoint_verifier: initiating new MQTT_CONNECTION
 to iot_endpoint: ENDPOINT_NAME-ats.iot.us-west-2.amazonaws.com
[INFO]: Thread-4-iot-dr-pubsub.py:205-connection_start: Connecting to ENDPOINT_NAME-
ats.iot.us-west-2.amazonaws.com with client ID 'dr-walkthrough'...
```

You can use the MQTT test client in the AWS IoT management console to verify to which Region messages are being published.

If you make the health check status in the primary Region **Healthy** by adding the character that you deleted, use the script to switch the health check status back again to the primary Region.

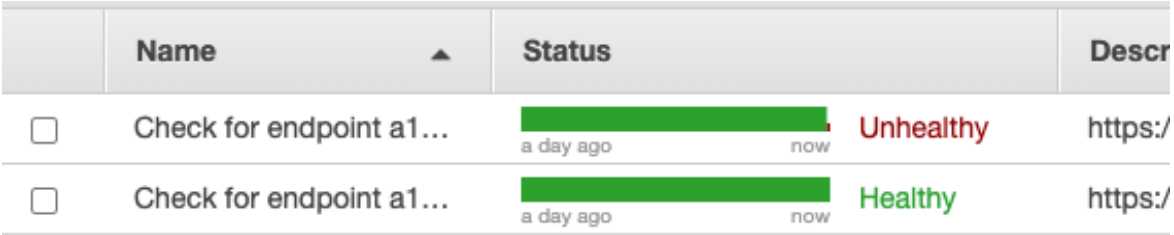

Figure 6: Disaster Recovery for AWS IoT Route 53 health checks

Automated tests

The python script `test-dr-deployment.py` in the `source/tools` folder provides an end-to-end test capability to check if the deployed solution is working. The script tests the following:

- **Device replication** from primary to secondary Region
- **Pub/Sub** to newly created things in primary and secondary Region
- **Shadow** replication
- **Delete thing** replication

Additional resources

AWS services

• AWS IoT Core	• Amazon Route 53
• AWS CloudFormation	• AWS Lambda
• AWS CodeBuild	• AWS Step Functions
• Amazon DynamoDB	• Amazon Simple Storage Service

Uninstall the solution

To uninstall the Disaster Recovery for AWS IoT solution, use the AWS CloudFormation Console or the command line.

Using the AWS Management Console

1. Sign in to the AWS CloudFormation console.
2. Select this solution's main stack. By default, the name is `IoTDRSolution`.
3. Choose **Delete**.

Using AWS Command Line Interface

Determine whether the AWS Command Line Interface (AWS CLI) is available in your environment. For installation instructions, refer to What Is the AWS Command Line Interface in the *AWS CLI User Guide*. After confirming that the AWS CLI is available, run the following command.

```
$ aws cloudformation delete-stack --stack-name <installation-stack-name>
```

Source code

Visit our GitHub repository to download the templates and scripts for this solution, and to share your customizations with others. Refer to the README.md file for additional information.

Revisions

Date	Change
May 2021	Initial release

Contributors

- Hubert Asamer
- Philipp Sacha
- Arun Viswanathan

Notices

Customers are responsible for making their own independent assessment of the information in this document. This document: (a) is for informational purposes only, (b) represents AWS current product offerings and practices, which are subject to change without notice, and (c) does not create any commitments or assurances from AWS and its affiliates, suppliers or licensors. AWS products or services are provided “as is” without warranties, representations, or conditions of any kind, whether express or implied. AWS responsibilities and liabilities to its customers are controlled by AWS agreements, and this document is not part of, nor does it modify, any agreement between AWS and its customers.