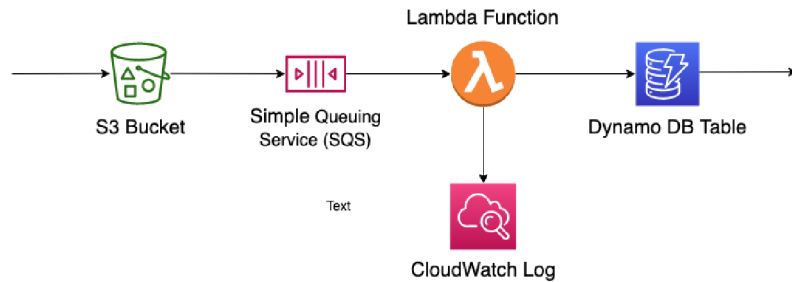


DHS Immersion Day Lab

Background

Today we're securing part of a large workload that processes incoming files. Another part writes files to S3. When they arrive at S3, S3 Events causes a Lambda function to run, which reads the S3 object and writes meta-data about the object to a table in Dynamo DB. Downstream portions of this workload read metadata about S3 objects from Dynamo DB.



CloudWatch Logs retains output from the Lambda function for troubleshooting and auditing.

Goals

This lab steps you through securing parts of this workload using guidance found in AWS and NIST 800-53 documentation.

Security Controls Setup

The NIST 800-53 PDF is available online at <https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-53r5.pdf>

Download a copy of this and keep it handy for the experiments below.

Account Setup

For today's lab, this portion of the workload will be created by a CloudFormation template that creates the S3 bucket, Lambda function, and Dynamo DB table.

Login to your AWS account used for this training (not your personal or work AWS account). Go to the [CloudFormation console](#) and click "Create Stack".

Click "Upload a template file", and then "Choose file". Pick the `cloudformation_template.yaml` file from the Git repository above. Click `Next` at the bottom.

Specify template
A template is a JSON or YAML file that describes your stack's resources and properties.

Template source
Selecting a template generates an Amazon S3 URL where it will be stored.

☐ Amazon S3 URL ☒ Upload a template file

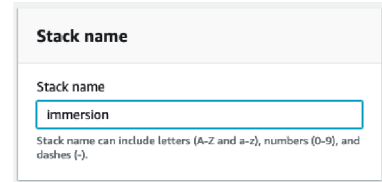
Upload a template file

Choose file `cloudformation_template.yaml`
JSON or YAML formatted file

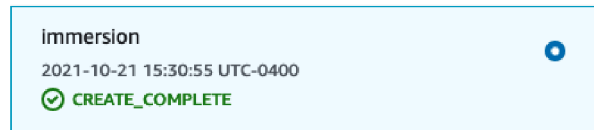
Type in "immersion" as the name of the stack and click `Next`.

Scroll to the bottom of the next page and click `Next`.

On the “Review immersion” page, scroll to the bottom. Click the “I acknowledge that AWS CloudFormation might create IAM resources” checkbox in the light blue box and click “Create stack”.



After a few minutes, you should see CREATE_COMPLETE under stack immersion.



Your workload is ready.

Take it for a spin!

In your production system, other parts of the workload would put objects in the S3 bucket. For today, we’ll simulate this by using the AWS Console to upload documents to your S3 bucket directly.

Go to the [S3 console](#) and find the immersion day S3 bucket. Click the link.

Name	AWS Region
Immersion-fileuploadsbucket-kn0450gsk5g	US East (N. Virginia) us-east-1

Click the Upload button to bring up the Upload page. Click “Add files” and pick a file under the sample_s3_files folder.



When added, click the orange Upload button at the bottom of the page. You should see an “Upload: status” page with a green “succeeded” in the Summary section. This confirms the file was uploaded.

Succeeded

✔ 1 file, 540.0 B (100.00%)

This also triggers S3 to put a message in the SQS queue, and for the Lambda function to pick it up and insert it into Dynamo DB. Check the Dynamo table to see a new record created from this S3 upload.

Go to the [Dynamo DB “Items” console](#) and click on the Immersion-ObjectMetadataTable radio button on the left. This displays all records in the Dynamo table.

You should see a single entry with the S3ObjectName matching the local file you uploaded to S3. If so, you’re ready to proceed: the system is installed and working correctly.

Tables (1)

Tag

Any table tag

Find tables by name

1

Immersion-ObjectMetadataTable-M8FRCQEIGYHK

► immersion-ObjectMetadataTable-M8FRCQEIGYHK

Expand to query or scan items.

Items returned (1)

Find items

<input type="checkbox"/>	S3ObjectName	Bucket	Created...
<input type="checkbox"/>	Grimms' Fairy Tales b...	immersion...	2021-10-2...

Lab 1: Inventory All Assets

Goal: Understand that all components of a workload are discoverable and documentable.

The first step in evaluating the security posture of a workload is to enumerate all the assets. Assets are the individual components in AWS that make up the whole. Assets are S3 buckets, Lambda functions, Dynamo DB tables, SQS queues, IAM Roles used by operators, Policies that establish permissions, and anything else.

Go to the [CloudFormation console](#) and click the hyperlink for “Immersion”, the stack you created above. Click the “Resources” tab at the top of the page to display all assets created by this CloudFormation stack.

Some entries here are hyperlinks. These take you to the AWS Console page for that particular asset.

Compare this list to the `cloudformation_template.yaml` file you uploaded above when you created the workload.

Logical ID ▲	Physical ID ▼	Type ▼
FileUploadsBucket	Immersion-fileuploadsbucket-kn0450gsck5g	AWS::S3::Bucket
InvokeS3EventSubscriptionLambda	2021/10/21/[\$LATEST]0f63aac1a08a488f82fb72949aa99f8d	AWS::CloudFormation::CustomResource
MetadataLambda	Immersion-MetadataLambda-w3sS5EvArktG	AWS::Lambda::Function
MetadataLambdaRole	Immersion-MetadataLambdaRole-18TTOJR68VUN8	AWS::IAM::Role
ObjectMetadataTable	Immersion-ObjectMetadataTable-M8FRCQEIGYHK	AWS::DynamoDB::Table

Each asset created in AWS corresponds to an entry in the YAML file. Developers manage the infrastructure of AWS workloads by writing configuration files under source code control, like a Git repository.

Find the row with Logical ID “S3EventSubscriptionLambdaRole” and click the hyperlink under the Physical ID column. This takes you directly to this Role on the IAM Console. Open the `cloudformation_template.yaml` file in an editor, or view it online at <https://github.com/ralberth/aws-serverless-security-lab>.

Compare the entries in the YAML file to what CloudFormation created in this IAM Role. Click the SystemAdministrator link on the IAM Console to view what permissions this Policy grants. Hint: it’s way too much!

The image shows a comparison between a YAML configuration file and the AWS IAM console. On the left, a snippet of the `cloudformation_template.yaml` file is shown with the following content:

```

S3EventSubscriptionLambdaRole:
  Type: AWS::IAM::Role
  Properties:
    AssumeRolePolicyDocument:
      Version: 2012-10-17
      Statement:
        - Effect: Allow
          Principal:
            Service: lambda.amazonaws.com
          Action: sts:AssumeRole
    ManagedPolicyArns:
      - arn:aws:iam::aws:policy/job-function/SystemAdministrator
  
```

On the right, the AWS IAM console shows the details for the role `Immersion-S3EventSubscriptionLambdaRole-18TTOJR68VUN8`. The role is linked to the `SystemAdministrator` policy, which is highlighted in the console. Arrows indicate the mapping between the YAML file and the console details.

Key Takeaway: All the “moving parts” of a workload are defined in code (YAML files in this example), including the security policies and key security-related configuration options.

Lab 2: Secure the S3 Bucket

Goal: Use AWS documentation to pick security best practices, and implement them on your S3 bucket.

Guidance

Open with the S3 User Guide at <https://docs.aws.amazon.com/AmazonS3/latest/userguide/Welcome.html> Click on the “Security” section on the left, then click on “Security Best Practices”. These are general best practices and a good place to start. AWS User Guides are a good place to learn about the security features of assets in a workload.

Briefly skim this web page looking at the bold subheadings to get an overview of S3 security features we should consider for our workload. In particular, skim the paragraphs under:

1. “Consider encryption of data at rest” (*we’ll do “Server-Side Encryption” today*)
2. “Enable Amazon S3 server access logging”

These correspond to common NIST 800-53 security controls. In the NIST 800-53 PDF document you loaded above, *briefly* skim the sections below to see how AWS security recommendations align with NIST 800-53 controls. In fact, the controls in many external, industry security accreditations align with AWS security best practices.

1. “SC-28 PROTECTION OF INFORMATION AT REST” around page 316
2. “AU-2 EVENT LOGGING” around page 66


Secure the Bucket

From the [CloudFormation Console](#), click the “Immersion” stack, and click the “Resources” tab like you did above. Scroll down and find the “FileUploadsBucket” entry and click the hyperlink next to it. This takes you to the [AWS S3 console page](#) for this S3 bucket. Click the “Properties” tab at top.

This is the list through to the S3 console and look at what settings are available. Notice that:

objects in S3 are not encrypted, and


Default encryption

Automatically encrypt new objects stored in this bucket. [Learn more](#) 

Default encryption
Disabled

access logging is disabled.

Server access logging

Log requests for access to your bucket. [Learn more](#) 

Server access logging
Disabled

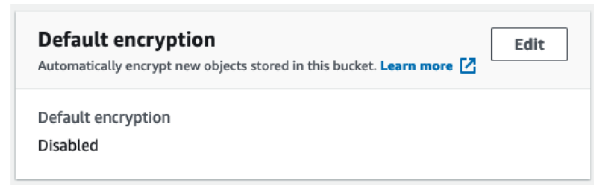
Using the AWS S3 Console, let’s turn on these security features. In a production system, the development team might edit their CloudFormation YAML files to enable these features so they are auditable and pass through an automated pipeline.

Turn on object encryption (“at rest”)

do through the console to get warmed-up.

Scroll to the “Default encryption” section on the page, and click the Edit button.

Click the “Enable” radio button, which shows the “Encryption key type” radio buttons.

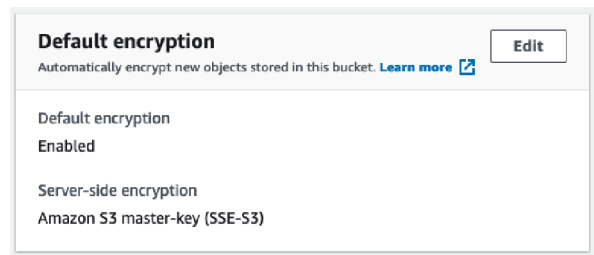


“Amazon S3 key (SSE-S3)” is the simplest type of encryption at rest. With SSE-S3, the S3 product handles encrypting, decrypting, and key management for every S3 object automatically. You don’t write any code and don’t have any burden to maintain or secure the keys. When applications store plain-text objects in S3, they are encrypted automatically before written to disk. Requests for S3 objects are decrypted automatically before returning to the caller.

Click “Amazon S3 key (SSE-S3)” and click the “Save changes” button.

The Default encryption section now shows the SSE-S3 setting.

All new objects written to this bucket will be encrypted at rest.



See it in action!

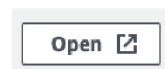
At the top of the page, click the “Objects” tab and click Upload button like you did before. Pick a different file from the sample_s3_files folder and upload it to S3.

Click on the hyperlink for the new file you just uploaded and scroll down to the Default encryption section. This file was automatically encrypted with SSE-S3 before writing it to disk.

Default encryption
Enabled

Server-side encryption
Amazon S3 master-key (SSE-S3)

Click the “Open” button at the top of the page. The encrypted file in S3 is retrieved and decrypted automatically.



Log All Actions

Turn on bucket logging and specify a location to log to.

Go to the [main AWS S3 console](#) page and click “Create bucket”. On the next form page:

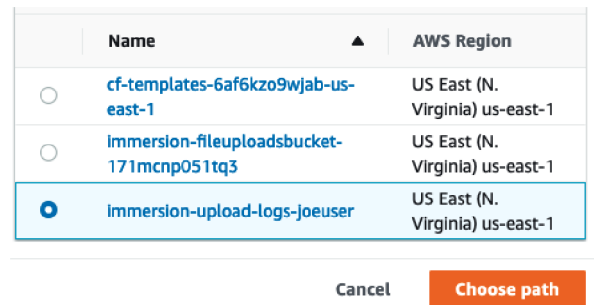
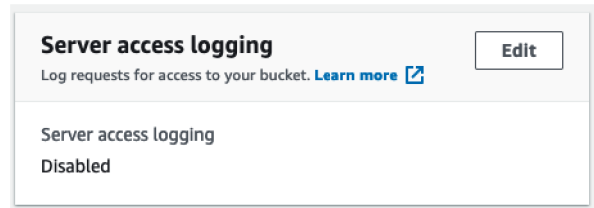
1. Enter “immersion-upload-logs-yourname” (bucket names are globally unique, everyone needs to create a bucket with a distinct name)
2. Leave “Block all public access” checked!
3. Change “Default encryption” on this bucket like the other one: set “Enable” and “Amazon S3 key (SSE-S3)”
4. Click “Create bucket” at the bottom of the page.

You now have two buckets: the original one created by the YAML file and CloudFormation. That bucket is used by your workload to upload files and send events to your Lambda function. You just created another bucket to hold all the access log entries from the 1st bucket. We now need to setup the connection so accesses to the 1st bucket are logged in the 2nd bucket. That's the "Server access logging" section on the `immersion-fileuploadsbucket`.

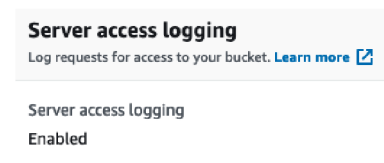
Click on the `immersion-fileuploadsbucket` (not the one you just created!) and then click on "Properties" at the top. Scroll down to "Server access logging" and click "Edit".

Click the "Enable" button, which brings up the "Target bucket" selector box. Click "Browse S3" and pick the `immersion-upload-logs-yourname` bucket. This sets the `immersion-fileuploadsbucket` to log accesses to the `immersion-upload-logs-yourname` bucket.

Click "Choose path", and then "Save changes".



All done! New accesses to this bucket will be recorded in the other bucket. You can use this audit log to track who did what.



See it in action!

Briefly skim the S3 documentation for this feature at <https://docs.aws.amazon.com/console/s3/server-access-logging>.

Server access log records are delivered on a best effort basis. Most requests for a bucket that is properly configured for logging result in a delivered log record. Most log records are delivered within a few hours of the time that they are recorded, but they can be delivered more frequently.

Grab a coffee and stretch, then check the `immersion-upload-logs-yourname` bucket. If lots aren't there, continue with other labs below and peek back later!

Key Takeaway: AWS managed resources have security features built-in that align with common assurance frameworks. Often, securing a workload means changing the configuration of AWS assets.

Lab 3: Secure the Lambda Function

Goal: Setup least privilege Policies that surround the Lambda function code

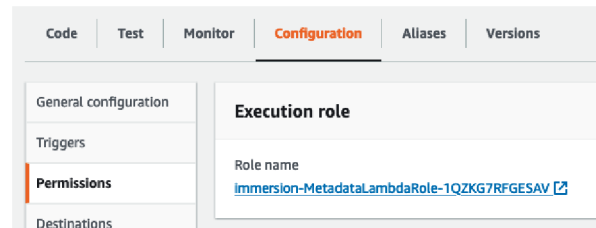
Lambda functions in AWS operate within the boundary setup by an IAM Role. The Role dictates what other AWS resources the Lambda function can interact with, and to what extent. For example, a Lambda function might need to read from a

Dyanmo DB table, but not write to it. A Policy can codify this.

Find and inspect the current Role for the Lambda function by going to the [AWS Lambda functions console](#) and clicking the `Immersion-Metadatalambda` hyperlink.

Click the `Configuration` tab and then click the `Permissions` tab on the left-hand side of the page.

Click the `Immersion-MetadatalambdaRole` hyperlink in the `Execution role` section at right. This is the Role that controls what this Lambda is permitted to do. The link takes you to the Role on the AWS IAM console.

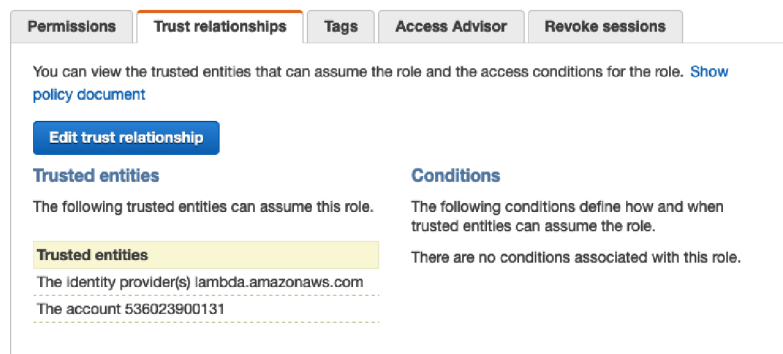


Trust Policy

Click the Trust relationships tab on the IAM console. This shows what other AWS resources are allowed to use this Role.

These aren't others that are allowed to execute the Lambda function.

We're looking at a Role. The Lambda function *uses* this Role to control its security.



The Trusted entities above are the Lambda product (`lambda.amazonaws.com`) and a different AWS account. That account might belong to a developer who was doing testing, or it might be a malicious way for an attacker to gain access to the abilities this Role has.

Click `Edit trust relationship`. In the JSON editor, remove the entire clause that grants access to `arn:aws:iam::536023900131:root`. Click `Update trust policy`. This Role can now only be used by the AWS Lambda runtime. No humans or other parts of AWS can use this Role.

means that only the Lambda product is allowed to use this Role. Humans can't use it, and other parts of AWS cannot use it.

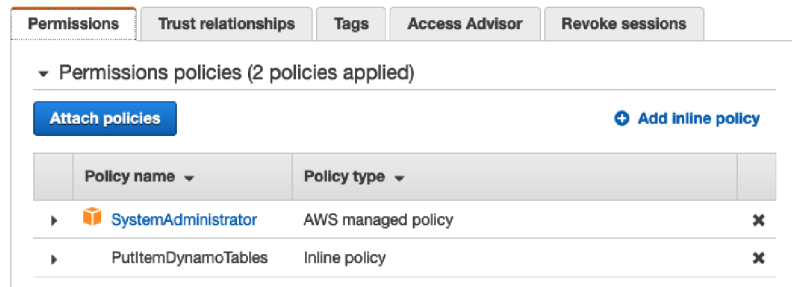
This shows good *isolation*, a key concept in NIST 800-53 and other security controls. *Briefly* look at NIST 800-53 section "SC-7 BOUNDARY PROTECTION", number 11 ("restrict incoming communications traffic") and number 21 ("isolation of system components") for related guidance.

Permissions

Click on the `Permissions` tab to show the abilities of this Role.

This role has the sum of all permissions in the `SystemAdministrator` policy and the `PutItemDynamoTables` policy.

This is too much! Administrator permissions are much more than this Lambda function needs.



Click the `SystemAdministrator` link to see all permissions that this grants. Obviously this Lambda function should not have the ability to launch EC2 instances, or destroy RDS databases. This was probably a starting-point for a developer early in the system's history that didn't get removed.

Click the black “X” on the right side of SystemAdministator to remove it from the policy. Click the red “Detach” button on the confirmation pop-up.

Click on the small, black “turnstile” triangle to the left of `PutItemDynamoTables` to inspect its permissions.

Service ▾	Access level	Resource
Allow (3 of 298 services) Show remaining 295		
CloudWatch Logs	Limited: Write	All resources
DynamoDB	Full access	All resources
SQS	Full access	All resources

It limits what this Role (and therefore what the Lambda function) can do to only these three services.

That's a good start, but there are serious problems left to handle. This Policy does not follow the Least Privilege security principle: grant only the most minimal abilities to actors in a secure system. In the NIST 800-53 document, briefly look at “AC-6 Least Privilege” on page 36 for an overview and related controls.

There are two dimensions to be fixed: limiting what access is granted to the services, and limiting which resources they apply to. As-written, this Role is allowed to do anything (“Full access”), including dropping tables, to any table in the account (“All resources”). This is much more than needed. From the code in the YAML file, the Lambda function only needs to dequeue messages from a single SQS queue and put items into a single Dynamo DB table.

Click `Edit Policy` to make these changes. You've been clicking around the console for a while now, so a summary of the changes to make is below. Ask for help if you get stuck!

1. Remove `PutItemDynamoTables` entirely. We're going to recreate it with Least Privilege permissions.
2. Remove `CloudWatch Logs` entry. No need to have a custom entry here when there is a Managed Policy already present that does what we want.
3. Attach policy `AWSLambdaBasicExecutionRole` which grants what the custom `CloudWatch` entry did above.
4. Add a new Inline Policy named “`DynamoDbPutItemOnly`” that only allows `PutItem` on the Dynamo DB table that starts with `immersion-ObjectMetadataTable`)
5. Add a new Inline Policy named “`RetrieveSqsMessages`” that grants actions `GetQueueAttributes`, `ReceiveMessage`, and `DeleteMessage` to the SQS queue in your account (starts with `immersion-S3NotificationQueue`).

Try it out!

Go back to S3 and upload another story from the `sample_s3_files` folder. The Dynamo DB table should have a new record to match the file you uploaded to S3. This confirms the Lambda function is operating correctly with the limited permissions set above.

Edit the Permissions again, and change `PutItem` to `WriteItem`. Upload another object. It won't be in the Dynamo DB table because the Lambda couldn't execute a `PutItem` Dynamo call when it's policy disallowed it.

Go to the [AWS CloudWatch Logs console](#) and click on the `/aws/lambda/immersion-MetadataLambda` log group. Pick the newest Log stream and click it. You should be able to find the following error message:

```
[ERROR] ClientError: An error occurred (AccessDeniedException) when calling the PutItem operation: User: arn:aws:sts::469426856847:assumed-role/immersion-MetadataLambdaRole-1QZKG7RFGESAV/immersion-MetadataLambda-QwwKtWLjyshU is not authorized to perform: dynamodb:PutItem on resource: arn:aws:dynamodb:us-east-1:469426856847:table/immersion-ObjectMetadataTable-1UVXF0TLDDPNY
```

In short, Role `immersion-MetadataLambdaRole` is not allowed to `PutItem` on table `immersion-ObjectMetadataTable`.

Key takeaway: Roles are assigned to AWS assets like Lambda functions. Roles control what part of AWS has what permissions. Roles help isolate portions of workloads from other workloads.

Lab 4: AWS Config

Goal: Setup an automatic monitor to alert you when key configuration changes.

Direct from the AWS Config User Guide:

You can use AWS Config rules to evaluate the configuration settings of your AWS resources. When AWS Config detects that a resource violates the conditions in one of your rules, AWS Config flags the resource as noncompliant and sends a notification. AWS Config continuously evaluates your resources as they are created, changed, or deleted.

We're going to setup AWS Config for your AWS account so it will flag any S3 key configuration changes that are considered dangerous.

Go to the [AWS Config console](#) and click the `1-Click setup` button. Leave everything as-is and click "Confirm" at the bottom. After a minute you'll see the Dashboard. Your AWS account is now Config-enabled.

Click `Rules` on the left, and then click the orange `Add rule` button. Search for and add the following rules, taking all the defaults:

- `s3-bucket-logging-enabled`
- `s3-bucket-server-side-encryption-enabled`

These are the security settings we turned on in the lab above.

AWS Config can just log when an S3 bucket changes its logging or SSE settings, publish a notification to SNS so your team can be paged, or even fix the problem automatically.

For s3-bucket-logging-enabled, _____

For server-side encryption, let's turn on automatic remediation:

1. Click the radio button next to s3-bucket-server-side-encryption-enabled and select Actions --> Manage Remediation.
2. Change to "Automatic remediation"
3. Under Remediation action details, select AWS-EnableS3BucketEncryption.
4. Click the Save changes button.

The screenshot shows the 'Select remediation method' section with two options: 'Automatic remediation' (selected) and 'Manual remediation'. Below this is the 'Remediation action details' section, which states 'The execution of remediation actions is achieved using AWS Systems Manager Automation'. It includes a dropdown menu for 'Choose remediation action' with 'AWS-EnableS3BucketEncryption' selected, and a description 'Enables Encryption on S3 Bucket'.

Left off here, finish AWS Config, turn on automatic remediation, then "Test it out!" by manually changing the S3 bucket and watching it reset automatically.

<https://aws.amazon.com/blogs/mt/implement-aws-config-rule-remediation-with-systems-manager-change-manager/>

Extra Credit: Limit Operator Permissions

Goal: Setup Least Privilege for humans that interact with the production system

Administrator and root vs. custom-configured.

ACCESS ENFORCEMENT | [ROLE-BASED ACCESS CONTROL](#)

Least Privilege overview

1. S3:
 - a. List objects and view metadata only
 - b. No access to view objects, remove objects, delete buckets
2. SQS:
 - a. Read-only
3. Lambda:
 - a. Manage the lambda, change environment
4. Dynamo DB:
 - a. Manage data
 - b. No access to drop tables or change table configuration

Setup Custom Operators

Create IAM Group with limited permissions. Can create with allowlist or denylist, most secure is allowlist-based.

Everything that a user is permitted to do must be listed.

Create:

1. IAM User Group for all operators
2. Managed Policy attached to the Group
3. IAM Users under the Group to inherit the Policy.

Test it out!

Log in as a User created above, try to drop a bucket or table, try to read an S3 object.