**Zero Trust Architecture in AWS**

**I. PROJECT OVERVIEW**

This project focuses on the design and implementation of a Zero Trust Architecture (ZTA) using Amazon Web Services (AWS). As organizations increasingly adopt cloud-native infrastructure, the traditional perimeter-based security model is no longer sufficient to defend against modern threats. Zero Trust offers a more robust approach by assuming that no user or system is inherently trustworthy—every request must be explicitly verified, regardless of its source.

1. *Objectives and Design Principles*

The goal of this project was to build a cloud environment that reflects the core principles of Zero Trust:

* Explicit identity-based access control
* Least privilege permissions by default
* Segmentation of internal resources to reduce attack surface
* Continuous monitoring and logging for visibility
* Formal threat modeling to evaluate and mitigate risks

We used a structured approach combining IAM, VPC, Lambda, S3, CloudTrail, and GuardDuty to apply these principles.

1. Role-Based Access Control

To ensure strict separation of duties and minimize the risk of privilege misuse, we established four distinct roles, Architect (Root), CodeOwner, Documenter and Tester, which are customed IAM policies for each. This structure helped validate the Zero Trust concept by testing that only explicitly authorized users could perform sensitive actions such as accessing confidential files or modifying infrastructure.

1. Implementation Highlights

* IAM Policies: Each role had custom-defined JSON policies ensuring only essential permissions. MFA was enforced across all users.
* VPC and Subnets: Resources were segmented between private and public subnets. Lambda functions were deployed in private subnets and accessed the internet securely through a NAT Gateway.
* S3 Access Control: The S3 bucket was configured with strict access policies and verified through role-based testing.
* Monitoring Tools: CloudTrail and GuardDuty were enabled to track all system events and detect suspicious behaviors.
* Threat Modeling: Security threats were formally analyzed using the DREAD framework. Mitigation strategies were proposed for high-risk scenarios such as misconfigured buckets and credential theft.

This Zero Trust implementation not only demonstrated technical controls such as IAM, VPC, and Lambda configuration, but also emphasized the importance of clear role separation, privilege minimization, and continuous validation. By incorporating identity, policy, and monitoring into every layer, we achieved a more secure, auditable, and resilient cloud environment aligned with modern security best practices.

**II. IAM AND ACCESS CONTROL**

In this project, Identity and Access Management (IAM) played a central role in enforcing Zero Trust principles. The core objective was to ensure that only explicitly authorized roles could access sensitive resources, while all others would be denied by default.

1. Role Definition and Assignment

We began by creating IAM users and categorizing them into four predefined roles, each with carefully scoped permissions aligned to their responsibilities:

* Architect (Root): Full administrative control, responsible for overall architecture and setup
* CodeOwner: Has permission to develop and deploy code within the environment
* Documenter: Has access to logs for monitoring purposes.
* TesterWithNoAccessPrivilege: User for validation tests, no access to critical components

Each user was then attached to their respective role using IAM policies. These policies were crafted using the least privilege model, granting only the necessary actions (e.g., s3:GetObject, lambda:InvokeFunction, logs:GetLogEvents, etc.) on explicitly defined resources.

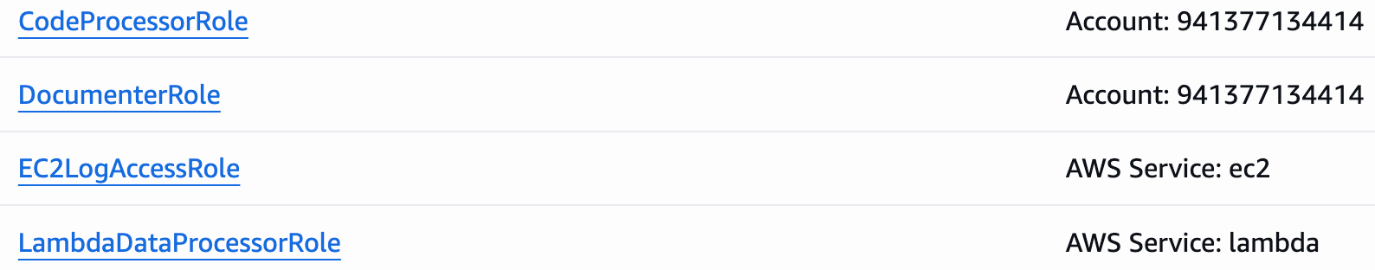
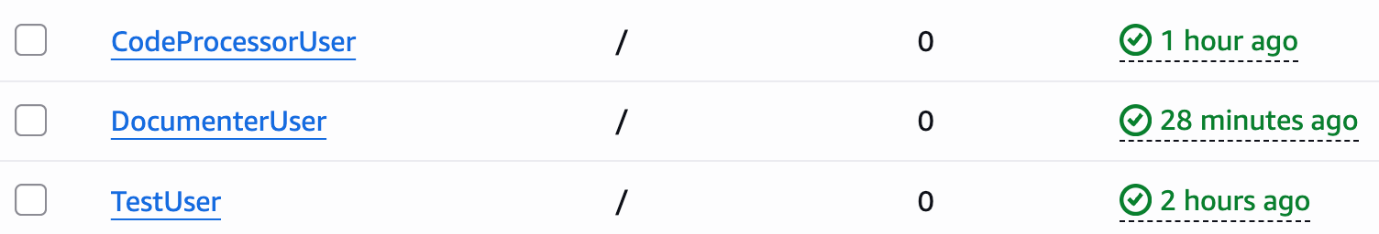


Figure 2.1: Defined IAM Roles in AWS Console

Figure 2.2: IAM Users Overview with Activity Status

1. Securing the Confidential S3 Bucket

We created a secure S3 bucket named confidential-bucket-eric20250713, which included a designated folder for sensitive content.

The Architect and CodeOwner roles were granted access to the folder using fine-grained bucket policy conditions. All other users, including Documenter and TesterWithNoAccess, were explicitly denied access at the policy or resource level. To validate access controls, we uploaded a sample text file inside the folder and simulated different user access attempts.

1. Validation and Results

A series of controlled tests were conducted to confirm the access restrictions:

When CodeOwner or Architect attempted to access the file (via console or Lambda code), access was successfully granted. And when Documenter or Tester attempted to access the same file, they received “Access Denied” errors, as expected.

This validated that the IAM role bindings and S3 bucket policies were functioning correctly, with no implicit trust granted to any user.

A screenshot of a computer

AI-generated content may be incorrect.

Figure 2.3: Permissions Attached to Tester User

This structure reinforces the Zero Trust security model:

* Trust is explicitly granted, never assumed
* Permissions are always role-bound, not user-bound
* Denial by default ensures that any misconfiguration results in restricted access, not accidental exposure

**III. MICRO-SEGMENTATION (VPC & SUBNETS)**

To implement network-level isolation and reduce the risk of lateral movement, we set up micro-segmentation using a custom Virtual Private Cloud (VPC) with isolated subnets and security boundaries.

Our VPC design included two subnets:

* PublicSubnet1 – intended for resources that may require direct internet access, such as a NAT Gateway
* PrivateSubnet1 – reserved for internal services, including our test Lambda function

Although PublicSubnet1 was created, we did not place any directly exposed services in it for this project. Instead, we used it solely to host a NAT Gateway, which acts as a secure intermediary allowing outbound internet access for internal resources.

The Lambda function was deployed inside PrivateSubnet1, ensuring it could not be accessed directly from the internet. To allow the function to communicate with external AWS services (like S3), we updated the route table associated with PrivateSubnet1 to direct all outbound traffic through the NAT Gateway in PublicSubnet1.

In addition to subnet isolation, we applied a dedicated security group (LambdaSG) to the Lambda function:

* All inbound traffic was explicitly blocked
* Only outbound HTTPS (port 443) traffic was permitted

This strict configuration ensures that the Lambda function operates in a tightly controlled environment, consistent with Zero Trust principles—specifically, the idea of “least exposure”. Every path to and from the function was intentionally designed, and no default trust or open access was allowed.

By combining private subnets, routing control, and tightly scoped security groups, we were able to build a network layer that supports Zero Trust by design. It allows secure operations without sacrificing isolation or visibility.



Figure 3.1 – VPC Configuration Overview

****

Figure 3.2– Subnets within ZeroTrustVPC



Figure 3.3– Security Group for Lambda Function

**IV. LEAST PRIVILEGE &CONTINUOUS AUTHENTICATION**

A core principle of Zero Trust is to grant only the minimum level of access required for a task, and to continually verify identity during interactions. In this project, we followed the least privilege model closely and enforced continuous authentication through a combination of IAM policies and user-level security settings.

1. *Least Privilege Implementation*

Each role in the environment was carefully assigned a set of permissions tailored to its operational needs:

* Architect (Root): Granted full administrative access to manage the entire AWS environment.
* CodeOwner: Allowed to develop and deploy Lambda functions, as well as interact with a specific S3 bucket used for confidential storage.
* Documenter: Could view system and application logs via CloudWatch and CloudTrail, but had no access to code or data.
* TesterWithNoAccess: Assigned no effective permissions, used to validate access restrictions.

The IAM policies assigned to each role were written in JSON and scoped as narrowly as possible. Actions were explicitly defined (e.g., lambda:InvokeFunction, s3:GetObject, logs:GetLogEvents), and resources were referenced by ARN to avoid wildcard permissions. This ensures that even within a role, access is purpose-built and cannot be abused beyond its scope. For example:The CodeOwner was permitted to access only the confidential S3 bucket and only through specific actions.The Documenter could query logs, but could not download, upload, or execute any resources.

1. *Continuous Authentication and Identity Enforcement*

To further strengthen identity assurance:

* Multi-Factor Authentication (MFA) was enforced for all users with console access.
* Access keys were only generated when necessary and rotated securely.
* No shared accounts or generic users were used—each person was identified by a named IAM user.

Additionally, session-based credentials (e.g., via STS) could be integrated in future extensions to support short-lived access, enhancing control over active sessions.

By combining least privilege policies with ongoing authentication requirements, the environment minimizes attack surfaces while ensuring legitimate users can operate efficiently and securely.

**V. Monitoring (CloudTrail & GuardDuty)**

To ensure system transparency and enable rapid detection of unusual behavior, we enabled both AWS CloudTrail and Amazon GuardDuty for centralized logging and threat monitoring.

1. *CloudTrail Logging*

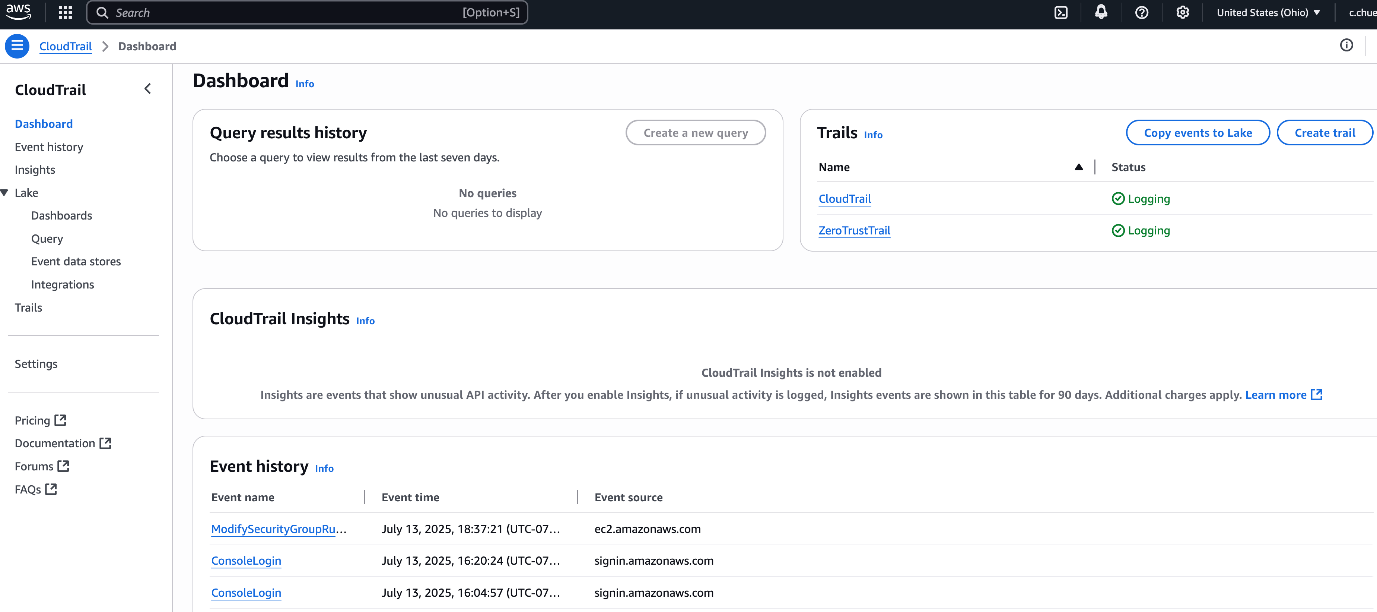
AWS CloudTrail was configured to capture management events across the account. Two separate trails were created: CloudTrail (default) and ZeroTrustTrail (dedicated to project resources).

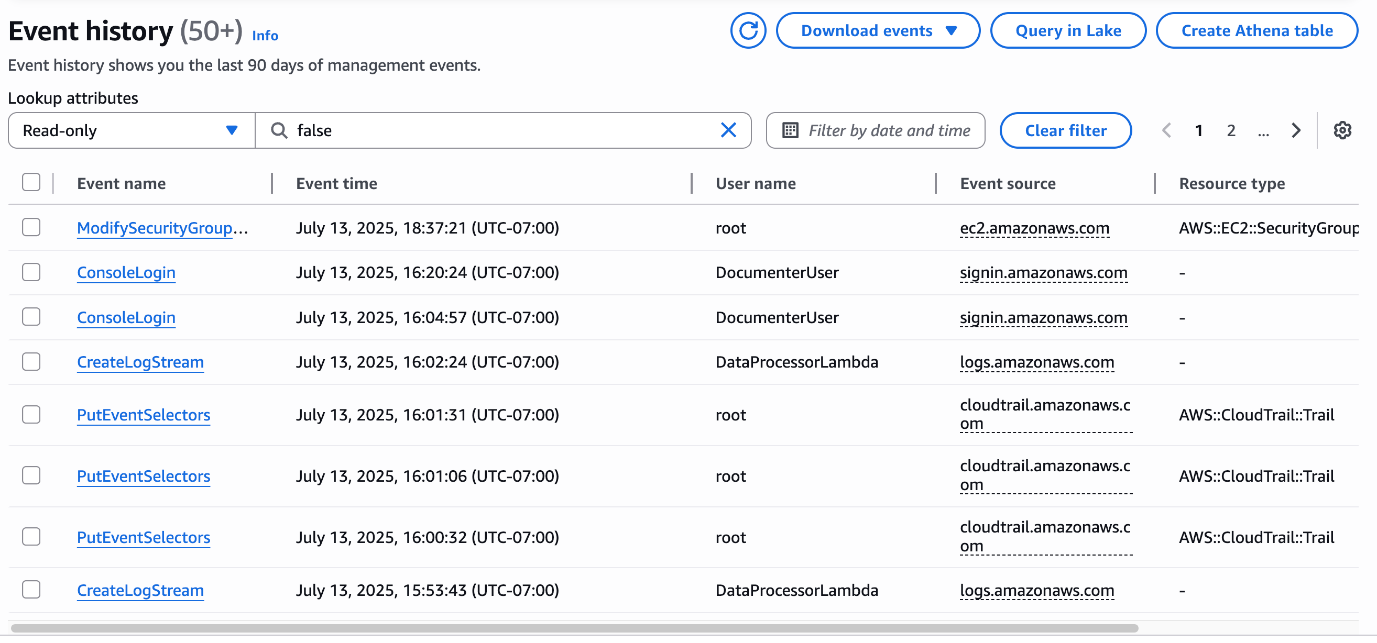
Both trails were set to active logging, allowing the system to capture all administrative activities, including login attempts, role assumption, changes to IAM policies, Lambda executions, and S3 access.

As shown in the dashboard (see Figure 5.1), CloudTrail successfully recorded events such as:

* Console logins from DocumenterUser
* Security group modifications by the root account
* Log stream creations triggered by the Lambda function (DataProcessorLambda)

These events provide a clear audit trail and support accountability across roles. For instance, the presence of a login event from DocumenterUser and corresponding log stream activities by Lambda confirm that permissions were enforced and used as expected.

Figure 5.1: CloudTrail Dashboard showing active trails and recorded events

Figure 5.2: Detailed event history confirming access activity by role (e.g., root, DocumenterUser, Lambda)

1. *GuardDuty Threat Detection*

To complement CloudTrail’s visibility, we also enabled Amazon GuardDuty, AWS’s intelligent threat detection service. While no anomalies were reported during the project execution period, GuardDuty continuously monitored for:

* Unusual API usage patterns
* Possible reconnaissance activity (e.g., port scanning)
* Credential compromise or instance-level threats

Had any high-severity findings been detected, they would have appeared in the GuardDuty console and could have triggered automatic alerts or remediation workflows.

Although the CloudTrail Insights feature was not activated in this deployment, it remains a valuable tool for future scaling, providing deeper behavioral analytics.

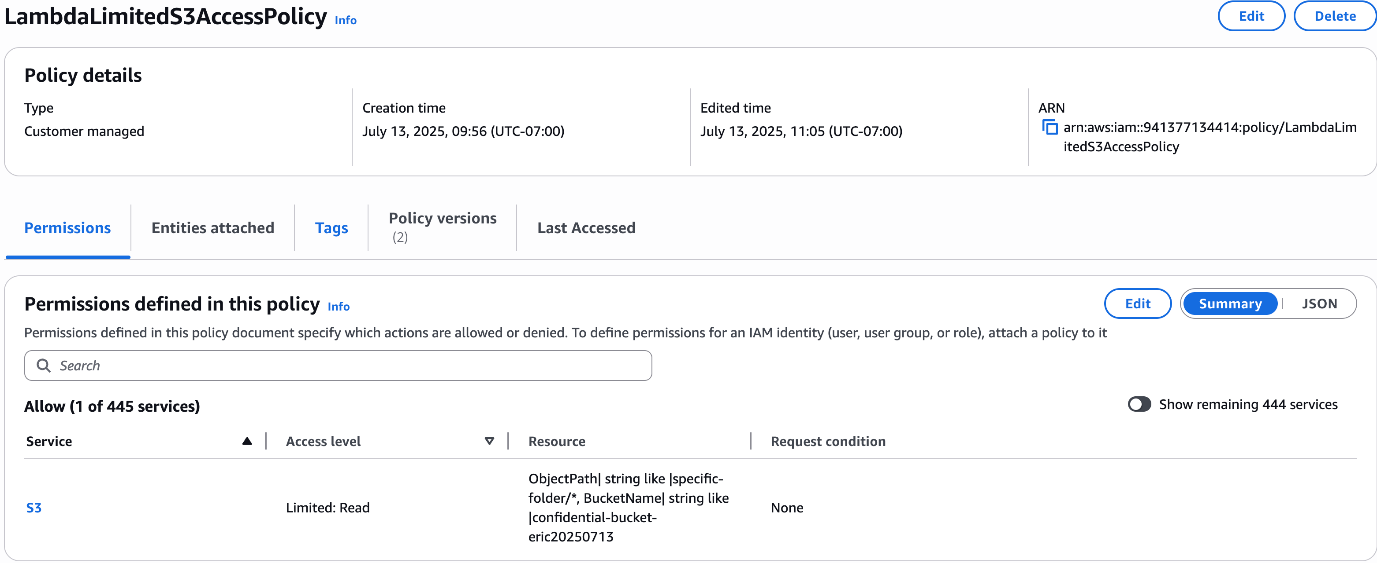
CloudTrail and GuardDuty together provide comprehensive monitoring across the environment. While CloudTrail ensures auditability and traceability of every action, GuardDuty adds real-time threat awareness. Both are foundational to maintaining Zero Trust observability and compliance.

**Ⅵ**. **Access Policies**

Aligned with the Zero Trust principle of granting only the permissions necessary for specific tasks, we defined a set of custom IAM policies that tightly scoped each role's access to resources. Each policy was attached to a specific identity and aligned strictly with their operational responsibilities.

The following custom policies were implemented and validated:

### *LambdaLimitedS3AccessPolicy*

This policy grants read-only access to a specific folder within a confidential S3 bucket. It is assigned to the Lambda execution role, ensuring the function can retrieve required files without broader access.  Figure 6.1: Policy details showing narrow S3 read-only permission for Lambda

The full JSON definition of this policy is shown below:

JSON

{

"Version": "2012-10-17",

"Statement": [

{

"Effect": "Allow",

"Action": "s3:GetObject",

"Resource": "arn:aws:s3:::confidential-bucket-eric20250713/specific-folder/\*"

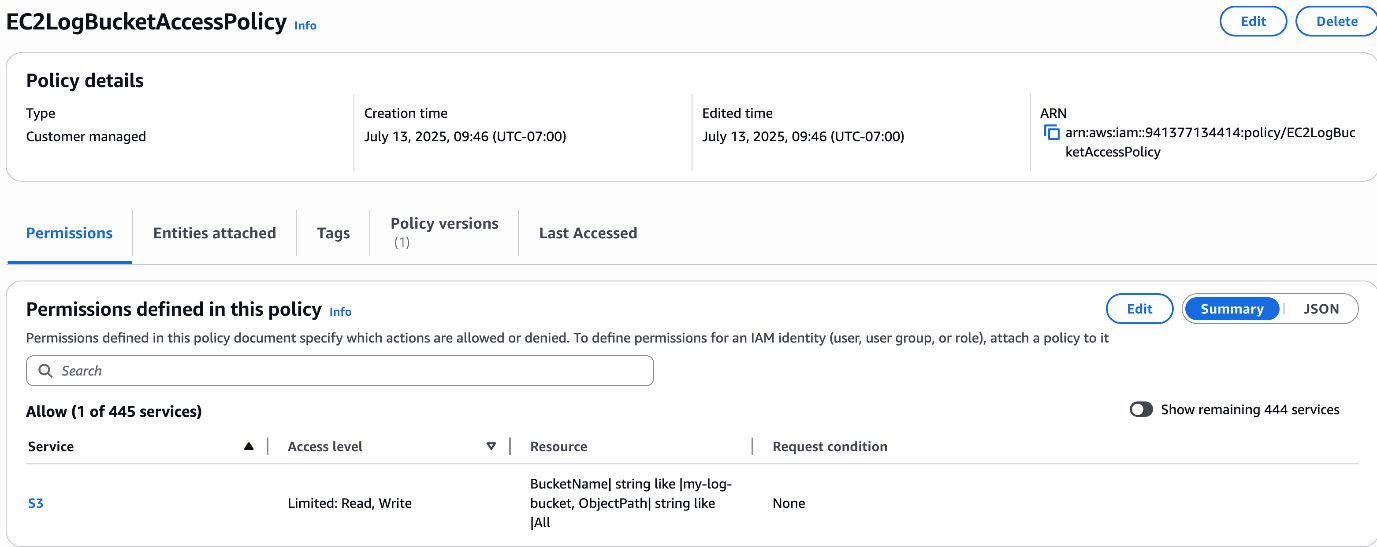
}

]

}

1. *EC2LogBucketAccessPolicy*

This policy is attached to an EC2 logging component or role (not shown in detail), allowing it to both read and write log data to a designated S3 bucket used for storing CloudTrail or system logs.

Figure 6.2: EC2LogBucketAccessPolicy allowing read/write access to log bucket only

The full JSON definition of this policy is shown below:

JSON

{

"Version": "2012-10-17",

"Statement": [

{

"Effect": "Allow",

"Action": [

"s3:GetObject",

"s3:PutObject"

],

"Resource": "arn:aws:s3:::my-log-bucket/\*"

}

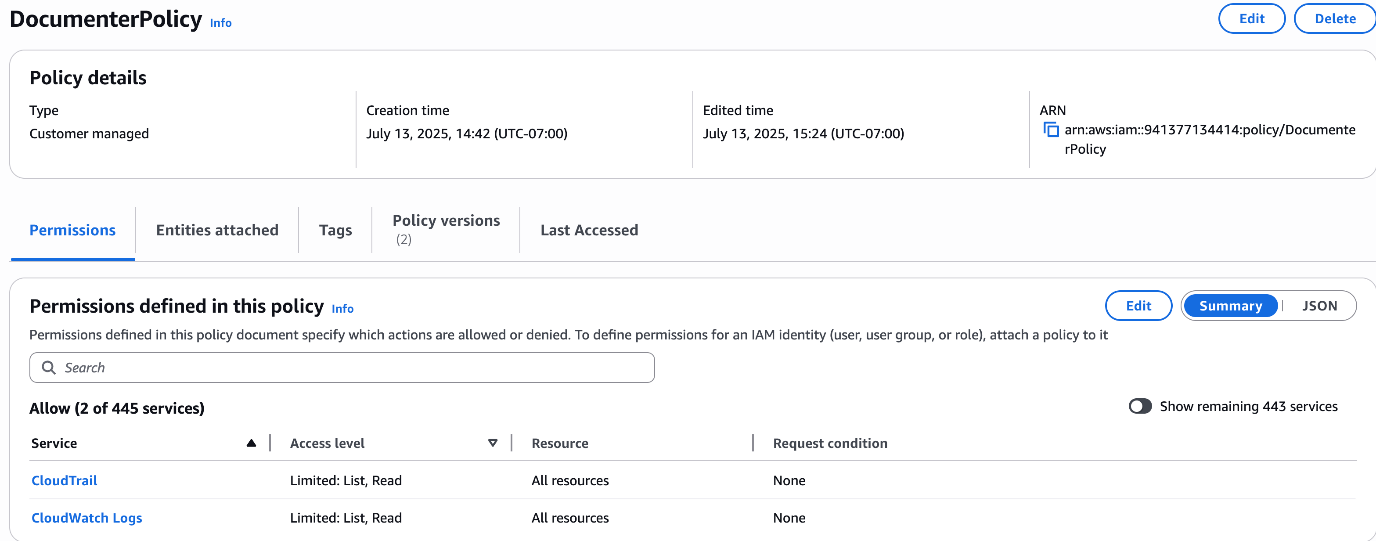
]

}

1. *DocumenterPolicy*

The DocumenterPolicy enables the assigned user to view system logs without exposing any data or infrastructure-modifying capabilities. It is attached to the Documenter role and includes:

This policy ensures that the Documenter can perform monitoring and reporting duties without risk of altering or accessing runtime code or data.

Figure 6.3: DocumenterPolicy allowing read-only log access to CloudTrail and CloudWatch Logs

The full JSON definition of this policy is shown below:

JSON

{

"Version": "2012-10-17",

"Statement": [

{

"Effect": "Allow",

"Action": [

"logs:DescribeLogGroups",

"logs:DescribeLogStreams",

"logs:GetLogEvents",

"cloudtrail:LookupEvents",

"cloudtrail:DescribeTrails",

"cloudtrail:GetTrailStatus",

"cloudtrail:ListEventDataStores"

],

"Resource": "\*"

}

]

}

1. *CodeProcessorPolicy & User Assignment*

This policy allows writing logs to CloudWatch, managing Lambda functions, and interacting with S3. It’s assigned to CodeProcessorUser.

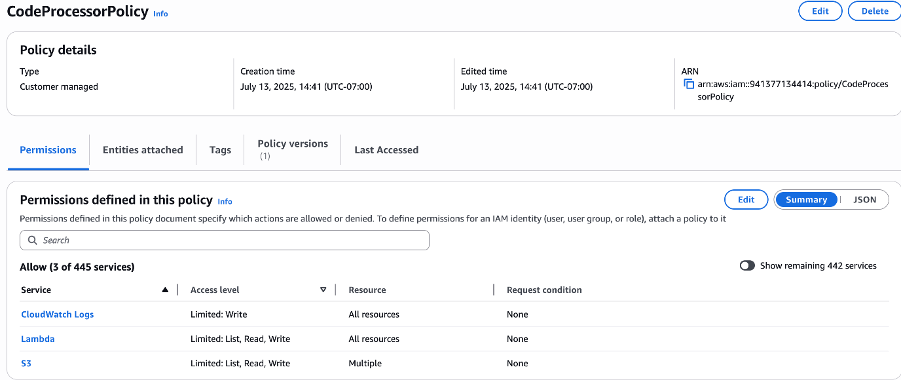


Figure 6.4: CodeProcessorPolicy – Full access for deployment role

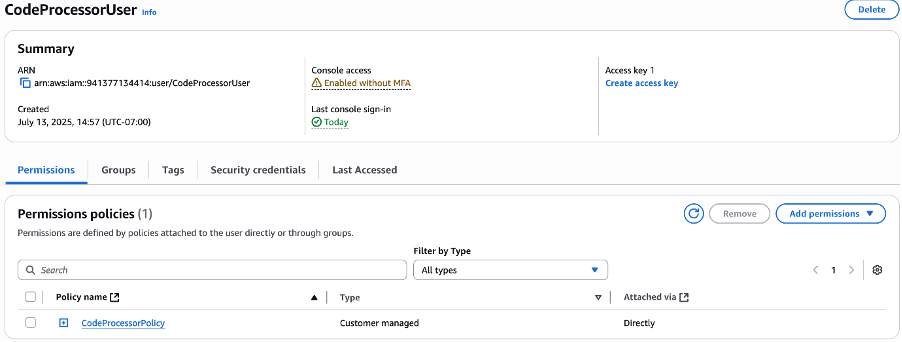


Figure 6.5: CodeProcessorUser assigned to CodeProcessorPolicy

The full JSON definition of this policy is shown below:

JSON

{

"Version": "2012-10-17",

"Statement": [

{

"Effect": "Allow",

"Action": [

"lambda:UpdateFunctionCode",

"lambda:UpdateFunctionConfiguration",

"lambda:InvokeFunction",

"lambda:GetFunction",

"lambda:ListFunctions"

],

"Resource": "\*"

},

{

"Effect": "Allow",

"Action": [

"s3:GetObject",

"s3:PutObject",

"s3:ListBucket"

],

"Resource": [

"arn:aws:s3:::confidential-bucket-eric20250713",

"arn:aws:s3:::confidential-bucket-eric20250713/\*"

]

},

{

"Effect": "Allow",

"Action": [

"logs:CreateLogGroup",

"logs:CreateLogStream",

"logs:PutLogEvents"

],

"Resource": "\*"

}

]

}

## *DocumenterUser Permissions*

Attached to an inline policy, DocumenterUser can only read logs from CloudTrail and CloudWatch Logs.

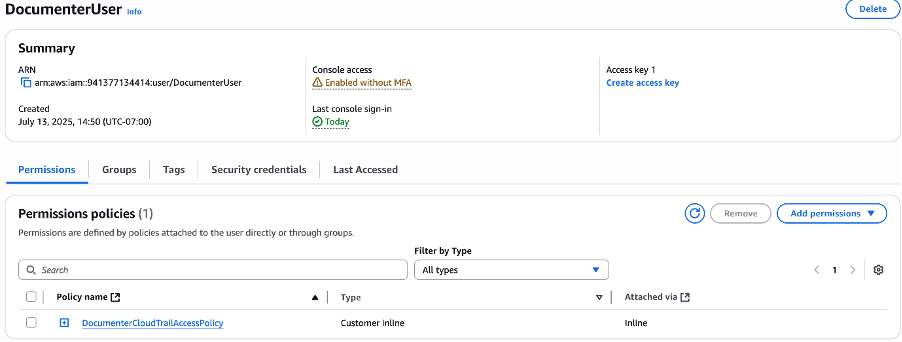


Figure 6.6: DocumenterUser with strict inline log access policy

## *TestUser: Restricted Access for Validation*

TestUser was created to simulate minimal permissions. It was assigned the AWS-managed policy AWSLambda\_ReadOnlyAccess to verify that non-privileged users cannot access sensitive resources.

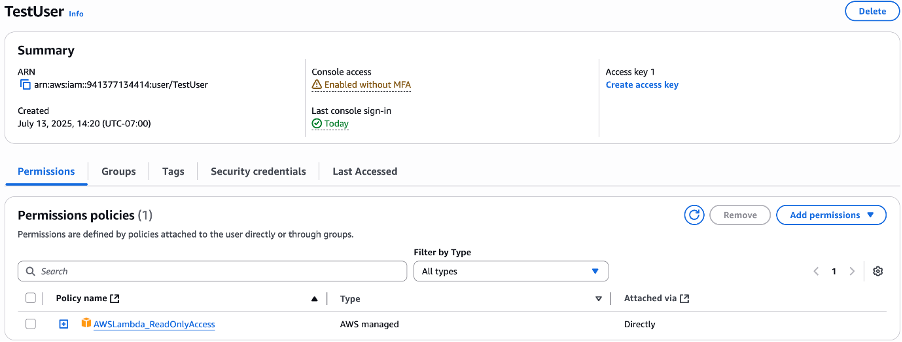


Figure 6.7: TestUser with AWSLambda\_ReadOnlyAccess for validation

**Ⅶ. THREAT MODELING (DREAD)**

In this report, we conduct a formal threat modeling session using the DREAD model to evaluate potential security risks in our AWS-based Zero Trust Architecture. This architecture includes multiple components such as IAM, S3, EC2, VPC segmentation, and monitoring services (CloudTrail, GuardDuty). Our goal is to identify, prioritize, and mitigate key threats to ensure strong security and adherence to Zero Trust principles.

| **DREAD Factor** | **Description** |
| --- | --- |
| D – Damage | How severe the damage would be if the vulnerability were exploited. |
| R – Reproducibility | How easy it is to reproduce the attack. |
| E – Exploitability | How easy it is to launch the attack. |
| A – Affected Users | How many users would be impacted. |
| D – Discoverability | How easy it is to discover the vulnerability. |

Scoring is from 1 (low risk) to 10 (high risk) per category. The average score reflects the severity of the threat.

| # | **Threat Scenario** | **D** | **R** | **E** | **A** | **D** | **Avg.** | **Description and Mitigation** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | Public S3 Bucket Exposure | 9 | 10 | 9 | 10 | 9 | **9.4** | Misconfigured S3 permissions can leak sensitive data. Mitigation: Use bucket policies to restrict access, enable logging. |
| 2 | IAM User Credential Compromise | 8 | 7 | 8 | 8 | 6 | **7.4** | Stolen credentials could allow privilege escalation. Mitigation: Enforce MFA, least privilege IAM policies. |
| 3 | EC2 Port Scanning and Brute-force via SSH | 6 | 6 | 7 | 5 | 7 | **6.2** | Attackers may attempt unauthorized access via open SSH port. Mitigation: Restrict SSH to specific IPs in security groups, use key-based auth. |
| 4 | Privilege Escalation via Misconfigured IAM | 7 | 6 | 7 | 6 | 5 | **6.2** | Users with excessive permissions may gain admin access. Mitigation: Apply least privilege principle, conduct IAM access reviews. |
| 5 | Denial of Service (DoS) Attack on Web App | 8 | 5 | 6 | 7 | 6 | **6.4** | Excessive traffic may make the system unavailable. Mitigation: Enable AWS WAF and rate-limiting. |
| 6 | CloudTrail Logs Disabled by Insider | 9 | 6 | 5 | 7 | 7 | **6.8** | Disabling audit logs hides malicious actions. Mitigation: Use service control policies (SCP), monitor trail status. |

**Ⅷ . CONCLUSION**

This project successfully demonstrates the design and deployment of a Zero Trust Architecture (ZTA) using core AWS services. Through a combination of fine-grained IAM policies, VPC-level micro-segmentation, secure storage access control, real-time monitoring, and formal threat modeling, we built a cloud environment that is secure, auditable, and aligned with modern cybersecurity best practices.

Key outcomes of the project include:

* Role-Based Access Control (RBAC): All users were bound to clearly defined roles with purpose-built permissions. Privileges were assigned based on responsibilities, and least privilege was enforced throughout.
* S3 Data Protection: Confidential data was stored in a private S3 bucket. Access was restricted through policy conditions, and validation tests proved that unauthorized users were correctly denied access.
* Network Isolation: By placing internal services (e.g., Lambda) in private subnets and routing outbound traffic via a NAT Gateway, we ensured that resources were not exposed to the public internet, reinforcing the principle of least exposure.
* Monitoring and Threat Detection: CloudTrail and GuardDuty provided comprehensive logging and security insights. All administrative and runtime activities were traceable, supporting both operational oversight and incident response readiness.
* Access Policy Precision: IAM policies were narrowly scoped to prevent over-permissioning. Users could only invoke specific actions (e.g., s3:GetObject, lambda:InvokeFunction) on explicitly defined resources.
* Threat Modeling with DREAD: Potential vulnerabilities were identified and prioritized based on severity, reproducibility, and exploitability. High-risk issues such as public S3 exposure and credential compromise were addressed with appropriate safeguards.

This project not only validated Zero Trust in a practical AWS context, but also highlighted the importance of disciplined identity management, resource scoping, and real-time auditing. The system now follows a default-deny model, where access must always be earned, not assumed.

Looking ahead, future improvements could include:

* Integration of temporary session-based access tokens (e.g., via AWS STS)
* Automated remediation actions triggered by GuardDuty findings
* Use of AWS Config for continuous compliance monitoring

By applying Zero Trust end-to-end—from identity to network to monitoring—this environment provides a strong foundation for secure cloud operations and scalable security governance.