International Journal of Emerging Trends in Computer Science and Information Technology
ISSN: 3050-9246 | https://doi.org/10.63282/3050-9246.IJETCSIT-V6I4P114
Eureka Vision Publication | Volume 6, Issue 4, 102-108, 2025

Original Article

An AWS-Native Pattern for Audited, Just-in-Time SSH Access Using Short-Lived Certificates

Tripatjeet Singh
Senior Cloud Engineer,Dallas-Fort Worth, USA.

Received On: 24/09/2025 ***Revised On: 26/10/2025*** ***Accepted On: 01/11/2025*** ***Published On: 08/11/2025***

Abstract - Traditional SSH access in multi-account AWS environments relies on long-lived keys, bastion hosts, or static IAM mappings, patterns that are costly, difficult to audit, and misaligned with just-in-time (JIT) and zero-standing-privilege principles. This whitepaper introduces an AWS-native approach using short-lived SSH certificates issued by a central SSH CA stored in AWS Secrets Manager, combined with GitHub Actions OIDC federation, OpenSSH user certificates, hardened EC2 access via AWS Systems Manager, and EventBridge-driven one-time network revocation. It also supports optional ServiceNow validation for production governance. The result is a low-cost, reusable framework delivering auditable, short-lived EC2 SSH access across multiple AWS accounts. The paper also outlines architecture, implementation, operational considerations, and comparisons with commercial JIT and AWS SSM Session Manager [9] solutions.

Keywords - AWS, SSH, certificates, Just-In-Time access, GitHub Actions, EventBridge Scheduler, Systems Manager, ServiceNow, Security, Compliance.

1. Introduction

Enterprises are increasingly adopting multi-account architectures in AWS to enforce isolation and governance. In these environments, application and platform teams frequently require shell access to Amazon EC2 instances for diagnostics, break-glass operations, or controlled configuration changes. This has traditionally been enabled by using shared SSH users (e.g., ec2-user) and long-lived key pairs, or by bastion hosts with static authorized_keys files. Such patterns are difficult to audit, prone to key sprawl, and do not align with zero-trust principles. SSH certificate-based authentication, supported natively by OpenSSH, addresses most of these challenges by replacing the static keys with short-lived certificates signed by a trusted CA. A few commercial vendors [1][2][3][7][8][13] and open-source projects, advocate and implement this model for both host and user authentication. It also describes JIT access flows based on short-lived SSH certificates.

However, there is a gap between these building blocks and a prescriptive, AWS-native pattern that:

- Operates across multiple AWS accounts.
- Does not require developers to have AWS permissions.
- Integrates into existing CI/CD tooling (e.g., GitHub Actions).
- Enforces time-boxed network access and user cleanup.
- Provides compelling, immutable evidence for audit and regulatory review.

This paper fills that gap by providing a reference architecture and implementation for a centralized, just-in-time SSH access framework with short-lived OpenSSH user certificates, EventBridge Scheduler one-time schedules, and SSM-driven EC2 instances configuration.

2. Background & Problem Statement

2.1. SSH Key Management Challenges

Conventional SSH public-key authentication requires that public keys be distributed to each target host and for authorized_keys files to be managed over time. Rotating keys, removing access for departed staff, and maintaining per-user accountability are non-trivial tasks at scale, especially across many accounts and networks. Prior work highlights that static keys do not scale and contribute to an expanded attack surface [5].

2.2. SSH Certificates and SSH CAs

OpenSSH introduces certificate-based authentication, where a CA signs user or host public keys with metadata such as principals, and validity period. When configured with the TrustedUserCAKeys directive, sshd can validate user certificates without storing individual public keys. Short-lived SSH certificates address key-management problems by reducing credential lifetime and centralizing trust in the CA.

Multiple commercial vendors provide managed SSH CA services. Some demonstrates SSH CAs integrated with identity providers and OIDC for single sign-on. Others provide similar capabilities with session recording and policy engines [1][2][3][4][5][6].

2.3. Just-in-Time Privileged Access

JIT access elevates users only for the duration and scope required to complete a specific task, reducing standing privileges. One commercial JIT system describes JIT SSH access using short-lived certificates that are generated on demand and linked to centrally managed accounts. Many of these solutions are appliance or SaaS-based and may be

heavy-weight for organizations that prefer to rely primarily on cloud-native services [7][8][13].

2.4. AWS-Native Access Mechanisms

AWS Systems Manager Session Manager [9] provides browser-based and CLI shell access to managed nodes without opening inbound ports or managing SSH keys, and integrates with CloudWatch Logs or S3 for auditing. Session Manager is well-suited when administrators have IAM access and when proxy-style access is acceptable. However, some workloads still require direct SSH connectivity, for example when third-party tools, existing jump hosts, or network inspection systems are already standardized on SSH.

Amazon EventBridge Scheduler is a serverless service that supports cron, rate, and one-time schedules, including per-target execution roles and flexible delivery windows. Previous work demonstrates patterns for scheduling one-time invocations and removing schedules after execution [10][11][12].

2.5. Positioning of This Work

This pattern differs from existing SSH CA products by relying exclusively on AWS services and GitHub Actions, without introducing a dedicated CA microservice. It also differs from Session Manager by preserving native SSH semantics while still providing centralized, short-lived, and auditable access. To my knowledge, no prior public blueprint describes a design that combines GitHub OIDC, a Secrets Manager-backed SSH CA, SSM-enforced SSH CA trust, one-time EventBridge Scheduler revocation, and ServiceNow change-request enforcement into an integrated workflow.

3. Design Goals and Thread Model

3.1. Design Goals

The system is designed to satisfy the following goals:

- Zero standing credentials on EC2. No long-lived SSH users or static authorized_keys; access is always time-bound.
- Developer-friendly access. Developers connect using standard SSH clients and local key pairs; they are not required to have AWS console or CLI permissions.
- Multi-account support. A single central CA and workflow can grant access to instances spread across many AWS accounts.
- Auditability and evidence. Every certificate issuance, network grant, and user lifecycle event is recorded in append-only stores suitable for financial and regulatory audits.
- Integration with change control. For production and uat accounts, access is gated by approved change requests in ServiceNow.
- Low operational overhead. The solution should be low-cost, serverless where practical, and driven by infrastructure-as-code.

3.2. Threat Model

We assume:

- AWS Organizations, centralized logging, and standard IAM hygiene are in place.
- Adversaries may obtain a developer's workstation or key pair, but not both the workstation and the CA private key.
- Adversaries may attempt to reuse, exfiltrate, or modify issued certificates, security group rules, or Linux user accounts.
- The CA private key is stored in AWS Secrets Manager and protected by a dedicated KMS key with monitoring on usage anomalies.

The pattern aims to minimize the blast radius of a compromised developer key or transient misconfiguration by limiting certificate validity, network exposure, and user presence on EC2 instances.

4. System Architecture

4.1. High-Level Overview

4.1.1. Central Security Account

- SSH CA private key stored in AWS Secrets Manager, encrypted with a customer-managed KMS key.
- DynamoDB table for structured audit events.
- S3 bucket with Object Lock enabled for immutable evidence.
- Optional SES configuration for emailing SSH certificates and instructions.

4.1.2. Target Application Accounts

- IAM role (e.g., github-oidc-role) assumable via GitHub OIDC in each account.
- SSM document and association that deploy trusted_user_ca.pub to /etc/ssh/trusted_user_ca.pub and enforces TrustedUserCAKeys plus hardened SSH settings [9].
- Lambda "revoke" function that revokes temporary security group rules and optionally deletes ephemeral Linux users via SSM.
- EventBridge Scheduler, configured by the workflow, to create one-time schedules that invoke the revoke function [10][11][12].

4.1.3. GitHub Actions

- Central repository with a workflow_dispatch workflow.
- Uses GitHub OIDC to assume the central role (to sign certificates and write audit data) and then assumes the target account role (to manipulate SSM and security groups).

4.1.3.1. ServiceNow

- REST API used to validate change request status for production and UAT accounts before granting access.

4.1.3.2. Developer Workstation

- Generates its own long-term SSH key pair.
- Receives a short-lived SSH certificate and clear SSH command via email.

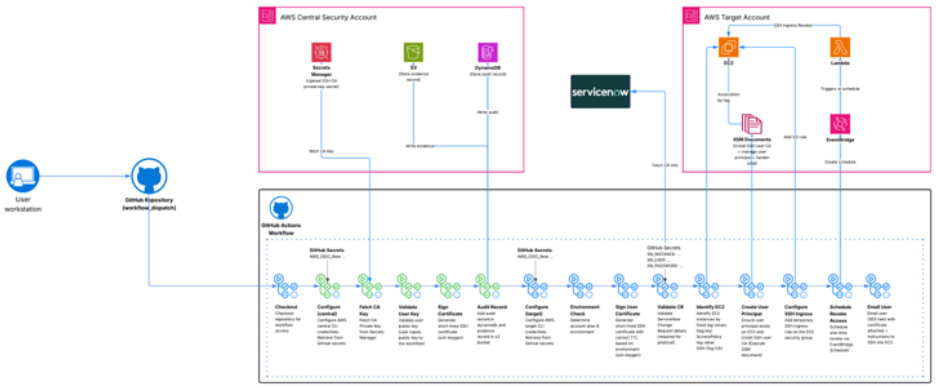

Figure 1. (High Level Architecture) Illustrates the Core Components

4.2. Trust and Identity Flow

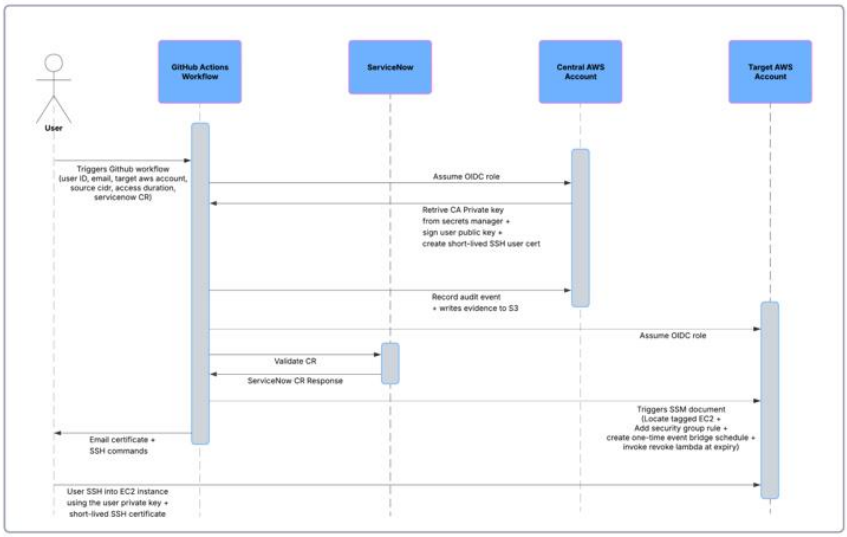

Figure 2. (Sequence Diagram) Shows the Typical Access Workflow

Developer generates a personal SSH key pair locally.

Developer (or an approver) triggers the GitHub workflow, providing:

- User ID (Linux principal name),
- Email address,
- Target AWS account ID,
- Source CIDR(s),
- Access duration,
- Optional ServiceNow change request number.

GitHub Actions uses OIDC to assume the central AWS role.

Workflow retrieves the CA private key from Secrets Manager and uses ssh-keygen to sign the user's public key, creating a short-lived user certificate.

Workflow records an audit item in DynamoDB and writes evidence to S3 (including certificate serial, principals, account ID, source CIDR, duration, workflow run ID, and, when applicable, change-request reference).

Workflow reconfigures AWS credentials to assume the github-oidc-role in the target account.

In the target account, the workflow:

- Uses SSM to ensure the Linux user principal exists and has the appropriate group membership and shell [9].
- Locates tagged EC2 instances (e.g., AccessPolicy=SSH-Org-CA).
- For each instance, adds a temporary security group rule for port 22 from the provided CIDR(s), using a description that embeds the user ID and certificate serial.
- Creates a one-time EventBridge Scheduler schedule per grant that will, at expiry, invoke the revoke Lambda with a payload describing the security group and CIDR [10][11][12].

The workflow emails the developer the certificate and a ready-to-copy SSH command.

At the end of the access window, EventBridge Scheduler invokes the revoke Lambda, which:

- Revokes the security group rule(s),
- Optionally invokes SSM to delete the Linux user, and
- Calls DeleteSchedule to remove its one-time schedule.

4.3. Account Tiering and Governance

The workflow queries the target account alias or tags to classify it as development, test, uat, or production. For production and uat aliases containing, for example, “prod” or “uat”, the workflow requires a ServiceNow change request identifier. It then calls the ServiceNow API to validate that the change is approved and in a valid window before proceeding. If the call fails or the status is not approved, the workflow terminates without issuing the certificate or modifying security groups.

5. Implementation Details

This section summarizes the key implementation aspects. In practice, all resources can be defined using Terraform and GitHub Actions YAML, but we focus on the conceptual structure.

5.1. Central Security Account

KMS Key and Secrets Manager

A customer-managed KMS key is created with key rotation enabled. The SSH CA private key is generated offline via:

```
ssh-keygen -t ed25519 -f ssh_ca -C "Org SSH User CA"
```

The private key is imported into AWS Secrets Manager; the public key (ssh_ca.pub) is kept for distribution to EC2 instances.

5.1.1. Audit Data Stores

Records per certificate are stored in a DynamoDB table, which contains a partition key serial and a global secondary index on the attributes such as username and issuedAt. Other fields include certificate serial, principals, account ID, instance scope, validity period, workflow run identifier, change-request reference, and certificate hash. Textual evidence files are stored in an S3 bucket with Object Lock, and with server-side encryption with KMS. Account ID and serial are encoded into object key names. Object Lock ensures that evidence cannot be tampered with or deleted for a preconfigured retention period, important for financial and regulatory audits.

- Central Role and GitHub OIDC

An IAM role trusts the GitHub OIDC provider and restricts sub to the specific organization and repository. The role permits:

- secretsmanager:GetSecretValue on the CA secret,
- kms:Decrypt on the CA KMS key,
- dynamodb:PutItem and UpdateItem on the audit table,
- s3:PutObject and retention operations in the evidence bucket,
- ses:SendEmail where SES is used,
- Sts:AssumeRole into github-oidc-role in any target account.

This role is assumed at the beginning of the workflow.

5.2. Target Accounts

5.2.1. Instance Hardening via SSM

An SSM document (type Command) is deployed that:

- Writes ssh_ca.pub into /etc/ssh/trusted_user_ca.pub,
- Ensures TrustedUserCAKeys /etc/ssh/trusted_user_ca.pub is present in sshd_config,
- Disables password authentication and root login where permitted,
- Restarts sshd.

An SSM association targets instances with a tag such as AccessPolicy=SSH-Org-CA. Any new EC2 instance tagged accordingly will automatically be configured to trust the central CA [9].

5.2.2. Onboarding Role

Each target account defines a role (e.g., github-oidc-role) that:

- Trusts the GitHub oidc directly,
- Allows ssm:SendCommand, DescribeInstanceInformation, and ListTagsForResource to manage users,
- Allows ec2:DescribeInstances, DescribeSecurityGroups, AuthorizeSecurityGroupIngress, and vokeSecurityGroupIngress,
- Allows scheduler:CreateSchedule, DeleteSchedule, and GetSchedule,
- Allows lambda:InvokeFunction on the revoke function.

5.2.3. Revoke Lambda Function

The revoke function is a small Python or Node.js Lambda that expects an input JSON payload:

```
{"SecurityGroupId": "sg-0123456789",
 "Cidr": "203.0.113.10/32",
 "FromPort": 22,
 "ToPort": 22,
 "Protocol": "tcp",
 "User": "xyz001",
 "DeleteUser": true}
```

It revokes the security group ingress rule and, if DeleteUser is true, calls SSM to execute a script on the instance that removes the Linux user, cleans up home directories, and removes any leftover authorized_keys files. Standard CloudWatch Logs are enabled for observability.

5.3. GitHub Actions Workflow

The workflow is manually triggered (workflow_dispatch) with inputs:

- userId (Linux principal),
- userEmail,
- targetAccountId,
- sourceCidrs (comma-separated),
- durationMinutes (with defaults differing for non-production vs production),
- changeRequestId (optional, but required for uat/prod).

Key steps are:

1. Assume central role via OIDC and validate inputs.
2. Fetch CA secret and use ssh-keygen -s to sign the uploaded user public key into a short-lived certificate:

```
SERIAL=$(date +%s%3N)
TTL=$((
${GITHUB_EVENT_INPUT_DURATIONMINUTES} * 60 ))
PRINCIPAL="${GITHUB_EVENT_INPUT_USERID}"
ssh-keygen -s ssh_ca -I "issuer@${SERIAL}" -n
"${PRINCIPAL}" -V "+${TTL}s" -z "${SERIAL}" user.pub
```

3. ServiceNow Validation. For accounts classified as uat or prod, the workflow calls the ServiceNow REST API to confirm that the change request exists, is approved, and is within its implementation window. Failure immediately aborts the workflow.
4. Audit Writes. The workflow writes an audit record to DynamoDB and an evidence file to S3 containing the serial, principal, account, duration, source CIDR, and change-request reference.
5. Assume target account via oidc role and ensure user exists via SSM. A shell script can, for example, call useradd if the user is missing or adjust group membership if the user already exists.
6. Security Group Grants. For each tagged EC2 instance, the workflow identifies the primary network interface's security group and calls AuthorizeSecurityGroupIngress with a description of the form "xyz001 SSH ACCESS <serial>". This embeds both the identity and the certificate serial in the SG rule for easy forensic mapping.
7. One-Time Schedules. For each grant (instance + CIDR), the workflow computes the expiry timestamp and creates a one-time EventBridge Scheduler schedule using an at() expression that invokes the revoke Lambda with a JSON payload describing the grant.
8. Email Instructions. Finally, the workflow emails the developer using SES, attaching or linking the certificate and including a ready-to-use SSH command:

```
ssh -i ~/.ssh/xyz001 -o CertificateFile=user-cert.pub
                xyz001@<PRIVATE-IP>
```

9. The email also calls out the source CIDR and the exact expiry time, and states that the SG rule and Linux user will be automatically removed.

5.4. Failure Handling and Edge Cases

The design handles several edge cases:

- Duplicate SG Rule: If the rule already exists (e.g., re-run of the workflow), errors from AuthorizeSecurityGroupIngress are caught and logged, and the grant is still recorded.
- Schedule Creation Failure: If EventBridge Scheduler rejects a schedule (e.g., malformed timestamp), the workflow fails before emailing the user, ensuring no orphaned SG rules without auto-revoke.
- Lambda Failure at Revoke Time: CloudWatch alarms and a secondary periodic job can detect any SG rules with expired timestamps in their description and trigger manual cleanup. Patterns for removing one-time schedules after execution are well documented.
- Revocation before Expiry: Security administrators can call the revoke Lambda directly with the grant payload to terminate access early.

6. Evaluation

We evaluate the pattern qualitatively along three axes: mean time to revocation (MTTR), auditability, and blast-radius limitation. We also comment on operational cost.

6.1. Mean Time to Revocation

With static SSH keys, MTTR for revocation is bounded by the time required to locate and remove keys from all authorized_keys files, often measured in hours or days. In this design, MTTR is bounded by:

- The certificate validity period (e.g., 2–4 hours for production),
- The EventBridge Scheduler runtime accuracy (seconds to minutes), and
- Optional manual invocation of the revoke Lambda.

Thus, even without manual intervention, credential validity is limited to the configured TTL. EventBridge Scheduler guarantees one-time invocations at the specified time in UTC. In practice, access can be revoked immediately by revoking SG rules and deleting the user via SSM.

6.2. Auditability

Each issuance generates:

- A structured record in DynamoDB keyed by certificate serial and indexed by username and timestamp.
- An immutable evidence file in S3 Object Lock containing user, account, change-request reference, CIDR, and expiry.
- Security group rule descriptions embedding the user ID and serial.
- Optional CloudWatch Logs entries for SSM commands and Lambda invocations.

Compared to approaches that rely solely on bastion logs or OS-level logging, this pattern provides an explicit, cryptographically tied evidence chain from identity → certificate serial → network grant → host user account.

6.3. Blast-Radius Limitation

The blast radius of a compromised developer workstation or key is constrained by:

- Short-lived certificates (hours, not days or months),
- Per-session, per-CIDR security group rules,
- Hosted CA private key in a central account with strict KMS access policies,
- Optional ephemeral Linux users that are deleted after each window.

Unlike pure Session Manager solutions, which rely on IAM to gate access but may provide broad OS-level privileges within a session, this pattern isolates privileges to specific instances and time windows, while still allowing use of native SSH tooling.

6.4. Operational Cost

As estimated using current AWS pricing, the monthly cost for a central account plus 10 target accounts and 200 EC2 instances is on the order of tens of dollars, dominated by minimal CloudWatch Logs and S3 storage. EventBridge Scheduler and Lambda usage cost is negligible at typical volumes. This compares favorably to commercial JIT access products that may charge per user or per host.

7. Discussion and Limitations

7.1. Comparison to Commercial JIT Systems

Commercial JIT system's SSH CA and single sign-on for SSH pattern demonstrate integrating OIDC identity with SSH certificates, but they generally assume a single control plane and do not describe AWS multi-account specifics such as SSM automation, or EventBridge Scheduler-based revocation. A few commercial JIT SSH access is more feature-rich for privileged access management and session recording, but typically requires dedicated infrastructure and licensing. This pattern trades some advanced features (e.g., screen recording) for a simpler deployment based solely on AWS-native services and existing GitHub pipelines.

7.2. Comparison to Session Manager

Session Manager eliminates the need for inbound SSH entirely and is often the preferred option when all administrators have IAM access and the organization is comfortable with a proxy-based shell experience. However, some scenarios still require direct SSH (e.g., when integrating with existing SOC tooling, jump hosts, or offline scripts). The presented pattern complements Session Manager by providing an SSH-native path that retains similar JIT and audit properties [9].

7.3. Limitations

- The pattern assumes EC2 instances are managed by SSM and that SSM can be used to modify sshd_config and user accounts.
- The use of GitHub Actions as the orchestrator means that outages or rate limits in GitHub may affect access issuance.
- ServiceNow integration hinges on reliable API connectivity and clear CR status semantics.
- While certificate issuance is centralized, host hardening and logging still require standard OS and agent configuration practices.

8. Conclusion

This paper has presented a practical, AWS-native engineering pattern for just-in-time SSH access using short-lived OpenSSH user certificates, GitHub Actions, and AWS services such as Secrets Manager, Systems Manager, EventBridge Scheduler, and Lambda. The design achieves zero standing credentials on EC2 hosts, strong auditability, controlled blast radius, and seamless integration with existing change-control processes. By building on open standards (OpenSSH certificates) and commodity cloud services, the pattern offers an attractive middle ground between fully managed JIT platforms and ad-hoc key-management scripts. Future work includes formal quantitative evaluation of MTTR improvements, and potential integration with additional identity providers and policy engines.

References

[1] Smallstep. SSH Certificate Login Tutorial. [Online].Available: https://smallstep.com/docs/tutorials/ssh-certificate-login/

[2] Smallstep. If You're Not Using SSH Certificates You're Doing SSH Wrong. [Online].Available: https://smallstep.com/blog/use-ssh-certificates/

[3] Smallstep SSH How it Works [Online]. November, 2025.Available: https://smallstep.com/docs/ssh/how-it-works/

[4] H. McLaren. How to Configure SSH Certificate-Based Authentication. [Online]. April, 2022.Available: https://goteleport.com/blog/how-to-configure-ssh-certificate-based-authentication/

[5] SecureW2. How Does SSH Certificate Authentication Work? [Online]. September, 2024.Available: https://www.securew2.com/blog/how-does-ssh-certificate-authentication-work

[6] Infisical. SSH Keys Don't Scale. SSH Certificates Do. [Online]. April, 2025.Available: https://infisical.com/blog/ssh-keys-dont-scale

[7] CyberArk. Just in Time Access with Short-Lived SSH Certificates. [Online]. Available: https://docs.cyberark.com/pam-self-hosted/latest/en/content/pasimp/jit-access-ssh-certificate.htm

[8] CyberArk. What is Just-In-Time Access? [Online]Available: https://www.cyberark.com/what-is/just-in-time-access/

[9] AWS. AWS Systems Manager Session Manager. [User Guide].Available: https://docs.aws.amazon.com/systems-manager/latest/userguide/session-manager.html

[10] AWS. Amazon EventBridge Scheduler, Schedule Types and One-Time Schedules. [User Guide].Available: https://docs.aws.amazon.com/scheduler/latest/UserGuide/schedule-types.html

[11] M.Sachin. One-Time Schedules Using AWS EventBridge Scheduler. [Online]. April, 2023.Available:

https://mathewsachin.github.io/blog/2023/04/21/aws-scheduler-one-time.html

[12] AWS Serverless Land. Remove One-Time EventBridge Schedules After They Run. [Online] Available: https://serverlessland.com/patterns/eventbridge-schedule-remove-one-time-schedules

[13] CyberArk. Connect through PSM for SSH. [Product Documentation].Available: https://docs.cyberark.com/pam-self-hosted/latest/en/content/pasimp/psso-pmsp.htm