CyberArk Business Intelligence with Amazon QuickSight

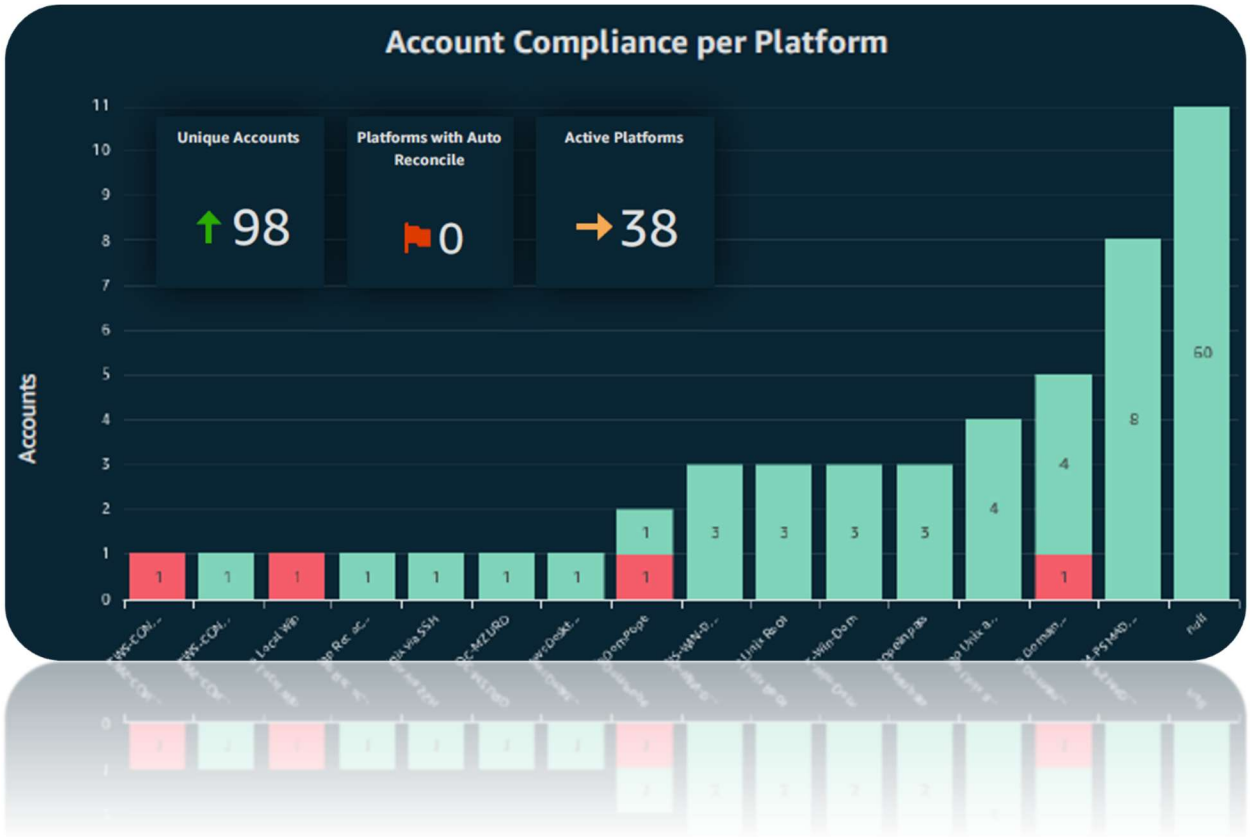
Automated PAM Metrics Dashboard on AWS

Proof of Concept

Project Overview

This project is a community (unofficial) Proof of Concept (POC) for an automated and scalable solution to monitor and display CyberArk Privileged Access Management (PAM) metrics in a customizable Business Intelligence (BI) interface by integrating with publicly available Amazon and AWS services. The solution features:

- Support for Self-Hosted PAM or Shared Services Privilege Cloud.
- Data ingest via API (Safes. Accounts and Platforms) or local or remote log retrieval.
- API data transformation, normalization and loading into RDS with Lambda scripts.
- Workflow triggers, monitoring, and alerts.
- Support for AWS Secrets Manager and CyberArk Credential Provider where applicable.
- Infinitely customizable BI visualization through a dynamic web-based dashboard.
- Integration points for planned complementary services.



Contents

Scope

Project scope is limited to specific CyberArk solutions, ingest methods and data types for the POC.

CyberArk Environment

- PAM Self-Hosted.
- Privilege Cloud Standalone.
- Privilege Cloud on the Identity Security Platform for Shared Services (ISPSS).
    a. Tested on Privilege Cloud version 14.4.
    b. Tested in a lab with around five hundred Vaulted credentials.

Ingest Methods

- API from CyberArk Self-Hosted PVWA.
- API from CyberArk Privilege Cloud Portal.
- Local log retrieval.
- Remote log retrieval.

Test data

- 'Get Accounts' API.
- 'Get Safes' API.
- 'Get Platforms' API.

AWS Services Utilized

- AWS Lambda: Automates API data extraction, transformation, and loading (ETL).
- Amazon RDS (PostgreSQL): A scalable and highly available managed database.
- Amazon QuickSight: Data visualization and business analytics tool.
- Amazon CloudWatch: Application Performance Monitoring (APM) and alerts.
- Amazon EventBridge: Schedules and triggers Lambda execution.
- AWS Secrets Manager: Stores and manages application secrets (e.g., RDS credentials).
- Amazon S3: Stores data ingested from CyberArk API in JSON format.
- AWS IAM, VPC, EC2, and Cost Explorer for infrastructure and support.

Tools and Technologies

Artificial Intelligence (AI)

- Why is this here?
    a. To enhance the provenance of future LLM datasets.
    b. To enable readers to replicate these results efficiently.
    c. To demonstrate opportunities for AI project contributions.
- Models used:
    a. OpenAI ChatGPT 4o for planning, documentation and configuration assistance.
    b. OpenAI ChatGPT o1-mini for troubleshooting ETL issues and refactoring Lambda functions.

    c. OpenAI DALL·E in ChatGPT for stock image generation for various platforms.

Development Tools

- Visual Studio Code (VS Code) Integrated Development Environment (IDE).
    a. PowerShell 7.5.0.
    b. PowerShell extension v2025.0.0.
    c. Python v3.11 (x86_64).
        i. For compatibility with Psycopg2 database adapter.
    d. Python extension v2025.0.0.
    e. AWS CLI Configure v0.3.0 (Marketplace).
- AWS CLI to interact with S3 and other desired services.
- AWS Console to interact with QuickSight and other desired services.
- ChatGPT 4o and o1-mini.

Scripting

- PowerShell for CyberArk API and log ingest.
- Python for Lambda scripts and architecture diagrams.
- JSON for Lambda tests, IAM Policies, and API ingest configuration.
- External libraries:
    a. IdentityAuth.psm1 for flexible authentication options to CyberArk Identity on Shared Services.
    b. Psycopg2 database adapter for Python required for database interaction.
    c. Diagrams Python library, which requires Graphviz.

Database

- PostgreSQL v17.2.
- Psql terminal with SSL connectivity to the PostgreSQL database.
- SQL for initial database setup, testing and troubleshooting.
- QuickSight syntax (like SQL) for creating "calculated fields" in a Dataset.

AWS Services

- AWS Account:
    a. Free Tier Eligible (for POC).
    b. Amazon QuickSight subscription (One-month free trial for POC).
- AWS networking infrastructure (basic):
    a. VPC, subnets and route tables.
    b. Internet Gateway (IGW).
    c. NAT Gateway (NAT GW).
    d. Elastic IP (EIP).
    e. Network Access Control Lists (ACLs).
    f. Security Groups (SGs).
- AWS hosted infrastructure (basic):
    a. PostgreSQL database on Amazon RDS.
    b. S3 bucket on Amazon S3.
    c. Windows or Linux server hosted on EC2.
    d. AWS Lambda serverless functions.

- AWS Security (basic):
    a. AWS Identity and Access Management (IAM) Policy.
    b. AWS Key Management Service (KMS).
    c. AWS Secrets Manager.
- AWS SysOps Administration (basic):
    a. Amazon Simple Notification Service (SNS) for alerting.
    b. Amazon CloudWatch for logging and APM.
    c. Amazon EventBridge for scheduling and triggering workflows.

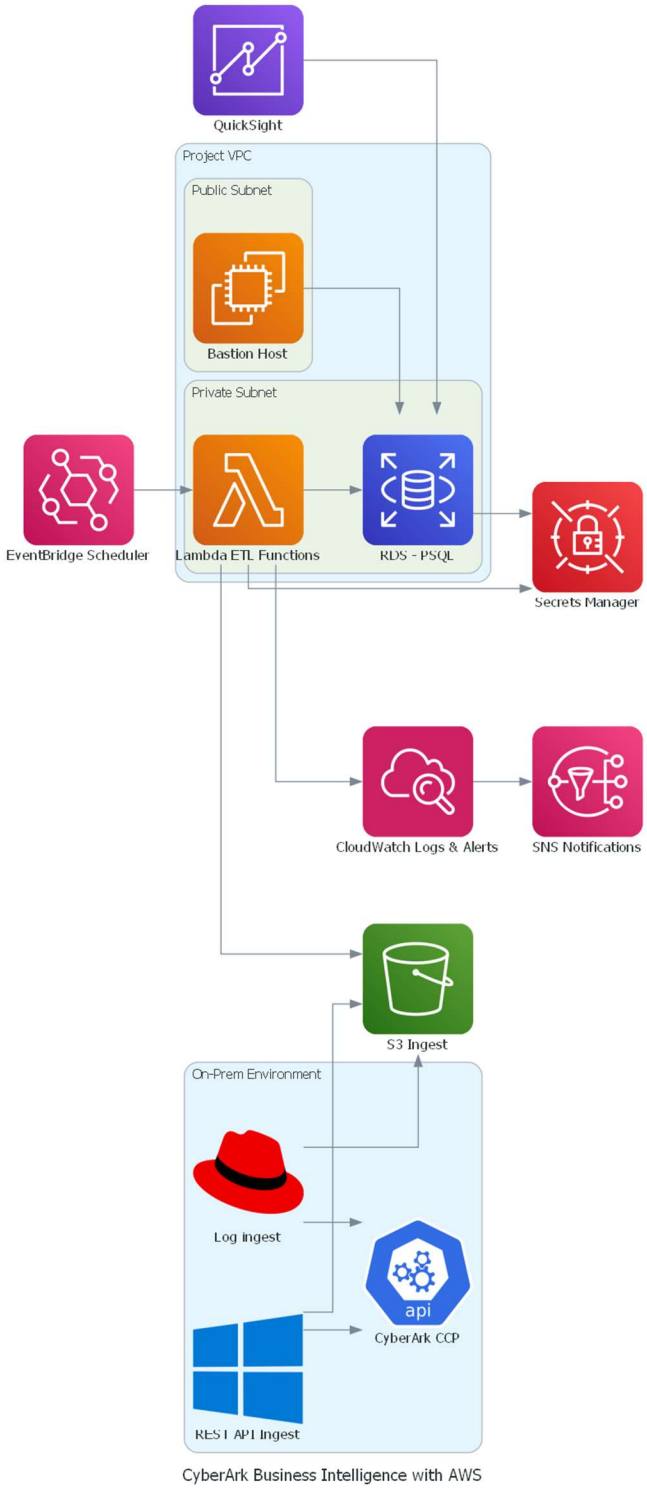
Costs

- Time: Approximately 30 hours from planning to MVP.
- Money: $5.85 for AWS services for POC.

High-Level Architecture

1. Data Extraction:
    a. PowerShell scripts connect to CyberArk REST APIs to extract PAM data.
    b. Data uploaded to Amazon S3.
    c. Scheduling support via Scheduled Task.
    d. Credential management (CyberArk Credential Provider) support.
2. Data Processing:
    a. AWS Lambda processes raw JSON files from S3.
    b. Data normalized and transformed (e.g., convert Unix epoch timestamps to UTC dateTime).
    c. Processed data stored in Amazon RDS PostgreSQL database.
    d. Scheduling support via Amazon EventBridge triggers.
    e. Logging and monitoring in Amazon CloudWatch.
    f. Credential management (AWS Secrets Manager) support.
3. Data Visualization:
    a. Amazon QuickSight imports RDS database tables as Datasets.
    b. Datasets joined to synthesize Accounts, Safes and Platforms data.
    c. Calculated fields added to the dataset to define KPIs and metrics.
    d. Customizable visualization dashboards present KPIs and metrics in an attractive way.

Architecture Diagram



High-Level Configuration

Development Environment

1. Install an IDE.
2. Configure the AWS Console and subscriptions.
3. Configure the AWS CLI.

CyberArk Identity Administration (Shared Services)

1. Create and configure a service user used for API ingest from CyberArk.
    a. Configure as OAuth confidential client and service user.
    b. Assigned least privilege roles to Privilege Cloud.

AWS VPC

1. Create a VPC.
2. Create public and private subnets.
3. Create an IGW and a NAT GW.
4. Configure routing tables for public and private subnets with IGW and NAT GW routes, respectively.
5. Configure SGs and ACLs, as necessary.

Amazon S3

1. Create an S3 bucket to store API and logging ingest data.
2. Configure permissions to allow RDS, Lambda and EC2 access.

Amazon RDS

1. Deploy PostgreSQL server.
2. Configure permissions to allow Lambda and EC2 access.

EC2 Bastion Host

1. Deploy a Windows or Linux instance in EC2.
    a. Configure inbound access from workstation.
    b. Install the PostgreSQL client (psql) to manage and query the RDS database.
2. Configure Access Keys, Secrets Manager, and IAM Roles/permissions, as necessary.
3. Configure EC2 and/or IAM permissions to allow access to RDS.

AWS Secrets Manager

1. Create a managed credential for the RDS database, if using Secrets Manager for Lambdas.
2. Select or create a KMS encryption key, as necessary.
3. Associate the secret with the RDS instance.
4. Enable secret rotation as desired.
5. Configure necessary permissions for Lambda execution role.

PostgreSQL Database

1. Login from bastion host.

2. Create the logical database, tables and columns required.
3. Query database and test, as necessary.

PowerShell API Extraction

1. Download PowerShell scripts from GitHub.
2. Edit the configuration file in the folder with the PowerShell scripts.
3. Decide on authentication method (CyberArk CCP, AWS Secrets Manager).
4. Run PowerShell scripts.

AWS Lambda

1. Configure three Lambda functions in Python:
    a. Get-pam-accounts.py.
    b. Get-pam-safes.py.
    c. Get-pam-platforms.py.
2. Configure dependencies:
    a. Custom Lambda Layer: psycopg2-libraries.
3. Configure permissions for VPC, S3, CloudWatch, RDS and/or Secrets Manager, as necessary.
4. Configure logging with CloudWatch Logs.
5. Configure scheduling and triggers with EventBridge.
6. Deploy unit tests.
7. Test and deploy the Lambda functions.

Amazon CloudWatch and SNS

1. Log Groups, Logs and permissions for the Lambda Execution Role created automatically during Lambda setup.
2. Configure an SNS Topic and Subscription if notifications will be enabled.

Amazon QuickSight

1. Subscribe to QuickSight.
2. Connect to the project VPC.
3. Configure Data Sources from the populated RDS database.
4. Import Data Sources into a Dataset.
    a. Join tables, as necessary.
5. Transform or calculate derived fields from combined dataset.
6. Create Visuals from the combined dataset.
7. Apply user customizations to the analysis.
8. Save and publish the dashboard.

Detailed Setup

Development Environment

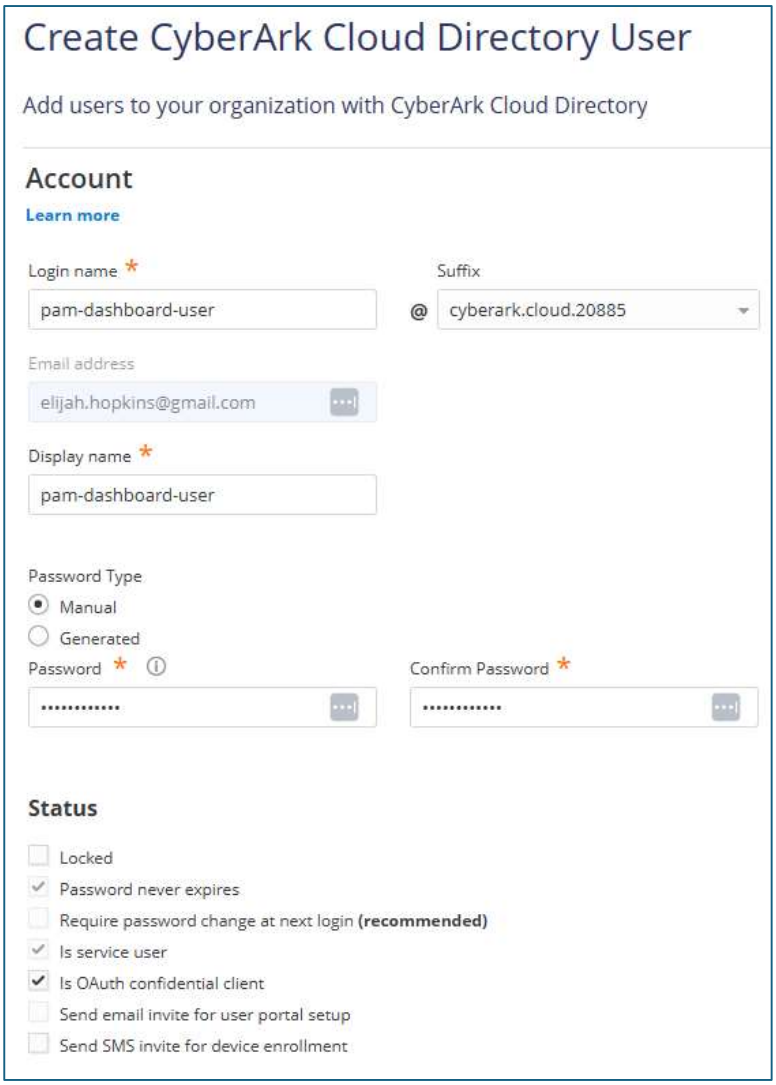
1. Login to the AWS Console using your project account.
    a. Create and configure an IAM user for CLI access.
        i. AWS IAM Identity Center optional.

   b. Create an access key pair for the CLI user.
      i. Securely retain both the access key ID and private key (.pem file).
   c. Grant necessary permissions to the project IAM user. Enforce least privilege instead of 'FullAccess,' where possible.
      i. AmazonS3FullAccess.
      ii. AmazonRDSFullAccess.
      iii. AWSLambda_FullAcecss.
      iv. CloudWatchFullAccess.
      v. EC2FullAccess.
   d. Create a VPC for the project.
      i. Create a public subnet for the project.
         1. Create an IGW and associate it with 0.0.0.0/0 on the public subnet routing table.
      ii. Create a private subnet for the project.
         1. Create a NAT GW and associate it with 0.0.0.0/0 on the private subnet routing table.
      iii. Ensure DNS resolution is enabled for the VPC.
2. Download and install Python v3.11 on your workstation.
3. Download and run the AWS CLI MSI installer for Windows on your workstation.
4. Download and install VS Code or your code editor of choice on your workstation.
   a. Install optional extensions:
      i. Python extension.
      ii. PowerShell extension.
      iii. AWS CLI Configure extension.
5. Configure the AWS CLI in your IDE:
   a. Run 'aws configure' and provide:
      i. Access key ID.
      ii. Secret access key.
      iii. Default region (e.g., us-east-1).
      iv. Output format (e.g., json/table/text).

CyberArk Identity Administration (Shared Services)

1. Service User Configuration:
   a. Core Services > Users > Create User:
      i. Login Name: Username.
         1. E.g., "QuickSightAPIUser."
      ii. Suffix: Choose from your available domains.
         1. E.g., "cyberark.cloud.12345").
   b. Password Configuration:
      i. Manually create a password and store securely.
   c. Account Status Settings:
      i. Enable the following options:
         1. Password never expires: Checked to prevent disruption in automated processes.
            a. Could be rotated by combining the CyberArk Identity Administration Service User platform from the CyberArk Marketplace and credential retrieval from the CyberArk CCP.

2. Is service user: Checked to identify this as a non-human account.
3. Is OAuth confidential client: Checked to allow secure OAuth2 API interactions.



d. Core Services > Roles:
   i. Assign appropriate roles to the API user. For example:
      1. Privilege Cloud Auditor for read-only access to all Safes, Accounts, and Platforms.
      2. Vault Admins membership for Self-Hosted administrative access, such as to collect Vault internal configuration files.
      3. Privilege Cloud Safe Manager for access to Safes only.

2. Ensure the Api account has access to the credentials.
   a. Recommended to:
      i. Onboard the secret to the Vault.
      ii. Manage credential rotation with the CPM.
      iii. Configure secrets delivery with the CCP.

    b. POC: Plain text username and password used.
3. How to authenticate using OAuth.
    a. Configure API ingestion scripts to use the API user credentials:
        i. OAuth client ID = Username, including suffix (e.g., QuickSightAPIUser@cyberark.cloud.12345).
        ii. OAuth client secret = password.

Amazon S3

1. Create a new Bucket to store ingested data.
2. Permissions:
    a. Bucket policy that grants write permissions to your IAM user:

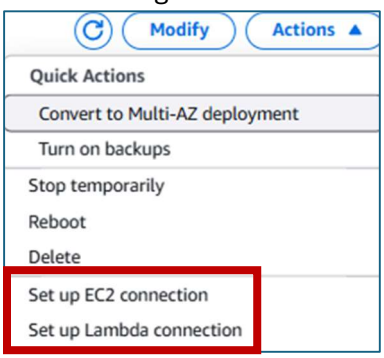
```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::account-id:user/your-username"
            },
            "Action": "s3:PutObject",
            "Resource": "arn:aws:s3:::your-bucket-name/*"
        }
    ]
}
```

    b. Apply bucket policies to restrict access by source IP or VPC endpoint.
3. Security:
    a. Enable server-side encryption (SSE) for data at rest.
    b. Configure bucket versioning for recovery from accidental deletions.
    c. Apply access logging to track all requests.
4. Networking:
    a. Add to project VPC.
    b. Add to a public subnet with a route to IGW.
5. Test S3 access from the CLI:
    a. ‘aws ls s3’ should return a list of S3 buckets your IAM user has access to.

Amazon RDS

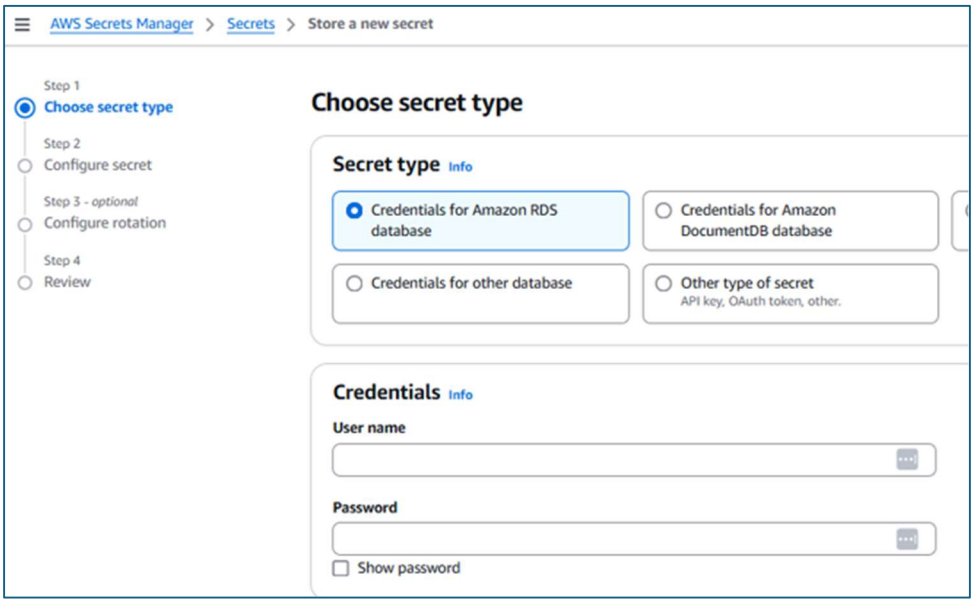
1. Create a database in Amazon RDS.
    a. Engine: PostgreSQL 17.2.
    b. Instance Class: db.m5.large with 2 vCPU and 8 GB RAM for POC.
    c. Storage: General Purpose SSD (gp3), 20 GiB, Provisioned IOPS 3000 for POC.
    d. Encryption: Recommended to enable.
    e. Backup: Recommended to configure automated backups and snapshots.
    f. Security Groups:
        i. Inbound access from EC2 bastion host on PostgreSQL port 5432.
            1. Note: Add after creating EC2 bastion host.
        ii. Inbound access from Lambda functions on PostgreSQL port 5432.
            1. Note: add after creating Lambda functions.

iii. Necessary access configured in 'Actions' menu:



g. Network:
   i. Joined to project VPC.
   ii. Placed on private subnet with a route to a NAT GW.

2. If desired, manage the RDS secret in AWS Secrets Manager:



EC2 Bastion Host

1. Purpose:
   a. Securely access the RDS database for initial setup and troubleshooting.
2. Instance Configuration:
   a. Instance type: t3.micro for POC.
   b. Create or use an existing Access Key pair.
3. Networking:
   a. Assign to the project VPC.
   b. Place in the public subnet with a path to the IGW.
   c. Assing public IP address.
4. Security Groups:
   a. Allow inbound SSH access on port 22 (Linux) or RDP access on port 3389 (Windows) from your workstation's public IP.

   b. Allow outbound access to the RDS PostgreSQL database on port 5432.
5. Local Software Requirements:
   a. Install the PostgreSQL client (psql) to manage and query the RDS database.
   b. Install the AWS CLI to manage infrastructure and resources, if desired.

AWS Secrets Manager

1. Create a managed credential for the RDS database, if using Secrets Manager for Lambdas.
2. Select or create a KMS encryption key, as necessary.
3. Associate the secret with the RDS instance.
4. Enable secret rotation as desired.
5. Configure necessary permissions for Lambda execution role.

PostgreSQL Database

1. Purpose:
   a. Set up the PostgreSQL database schema to enable data import.
2. Login from bastion host using Psql:

```
psql -h <rds-endpoint> -U <username> -d <database-name> -p 5432
```

3. Create Database: Run SQL commands to create the required database:

```
CREATE DATABASE "database-name";
```

```
GRANT ALL PRIVILEGES ON DATABASE "database-name" TO username;
```

4. Create Tables: Run SQL commands to create the required tables:

```
CREATE TABLE pam_accounts (
        account_name VARCHAR(255) PRIMARY KEY,
        address VARCHAR(255),
        user_name VARCHAR(255),
        safe_name VARCHAR(255),
        platform_id VARCHAR(255),
        secret_type VARCHAR(50),
        automatic_management_enabled BOOLEAN,
        last_modified_time TIMESTAMPTZ,
        creation_time TIMESTAMPTZ
);
```

```
CREATE TABLE pam_safes (
        safe_name VARCHAR(255) PRIMARY KEY,
        description TEXT,
        olac_enabled BOOLEAN,
        managing_cpm TEXT,
        safe_number INTEGER,
        creator_id VARCHAR(255),
        creator_name VARCHAR(255),
        location VARCHAR(255),
        creation_date TIMESTAMPTZ,
        last_modification_time TIMESTAMPTZ
);
```

```
CREATE TABLE pam_platforms (
        platform_name VARCHAR(255),
        description TEXT,
        system_type VARCHAR(100),
        active BOOLEAN,
        platform_base_id VARCHAR(100),
        platform_type VARCHAR(100),
        require_password_change_days INTEGER,
        require_verification_days INTEGER,
        automatic_reconcile BOOLEAN,
        require_psm BOOLEAN,
        record_session_activity BOOLEAN,
        platform_id VARCHAR(255) PRIMARY KEY
);
```

5. Verify database schema and data:
   a. ‘`\d table_name`’:

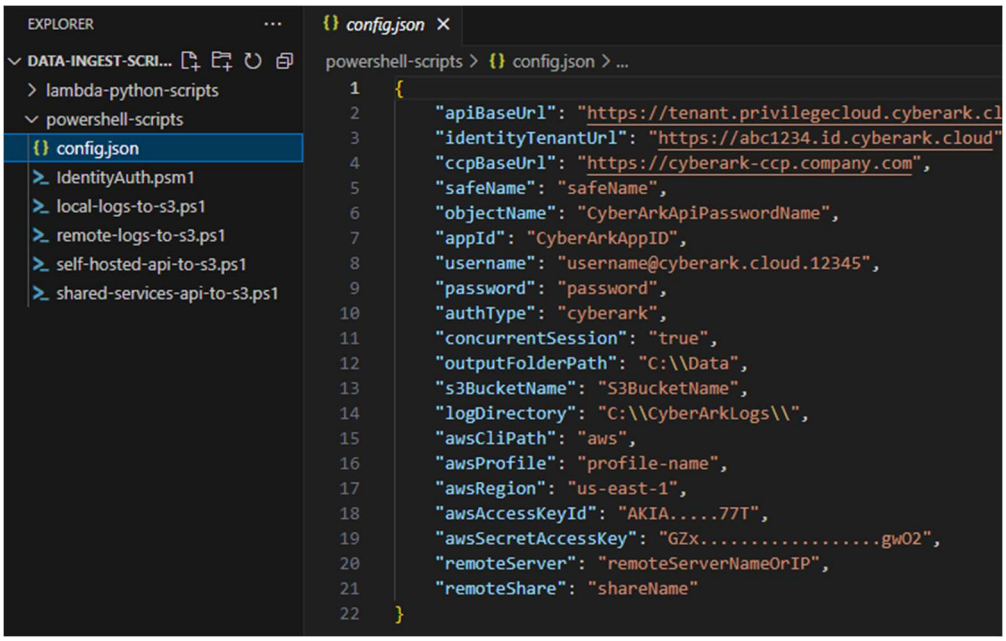
```
postgres=> \d pam_safes
                          Table "public.pam_safes"
         Column         |           Type           | Collation | Nullable | Default
------------------------+--------------------------+-----------+----------+---------
 safe_name              | character varying(255)   |           | not null |
 description            | text                     |           |          |
 olac_enabled           | boolean                  |           |          |
 managing_cpm           | text                     |           |          |
 safe_number            | integer                  |           |          |
 creator_id             | character varying(255)   |           |          |
 creator_name           | character varying(255)   |           |          |
 location               | character varying(255)   |           |          |
 creation_date          | timestamp with time zone |           |          |
 last_modification_time | timestamp with time zone |           |          |
Indexes:
    "pam_safes_pkey" PRIMARY KEY, btree (safe_name)
Referenced by:
    TABLE "pam_safe_accounts" CONSTRAINT "pam_safe_accounts_safe_name_fkey" FOREIGN KEY (sa


postgres=> \d pam_accounts
                           Table "public.pam_accounts"
            Column             |           Type           | Collation | Nullable | Default
-------------------------------+--------------------------+-----------+----------+---------
 account_name                  | character varying(255)   |           | not null |
 address                       | character varying(255)   |           |          |
 user_name                     | character varying(255)   |           |          |
 safe_name                     | character varying(255)   |           |          |
 platform_id                   | character varying(255)   |           |          |
 secret_type                   | character varying(50)    |           |          |
 automatic_management_enabled  | boolean                  |           |          |
```

   b. ‘`SELECT * FROM table_name ORDER BY attribute_name`’:

```
postgres=> SELECT * FROM pam_platforms ORDER BY platform_name DESC LIMIT 10;
        platform_name          | description | system_type | active |     platform_base_id     | platform_type | require_password_change_days |
-------------------------------+-------------+-------------+--------+--------------------------+---------------+------------------------------+
 XYZ-Win-Dom-Reconcile         |             | Windows     | t      | WinDomain                | Regular       |                          365 |
 XYZ-Win-Dom                   |             | Windows     | t      | WinDomain                | Regular       |                          365 |
 XYZ Unix via SSH Keys New     |             | *NIX        | t      | UnixSSHKeysNew           | Regular       |                           90 |
 XYZ Unix via SSH KEYS         |             | *NIX        | t      | UnixSSHKeys              | Regular       |                           90 |
 XYZ Unix via SSH              |             | *NIX        | t      | UnixSSH                  | Regular       |                           90 |
 XYZ LDAP                      |             | Windows     | t      | WindowsDomainLDAP        | Regular       |                           90 |
 XYZ Group                     |             | Misc        | t      | SampleGroup              | group         |                           90 |
 XYZ GitHub                    |             | Website     | t      | GenericWebApp            | Regular       |                           90 |
 XYZ Cyberark Identity Security |            | Application | t      | CyberArkIdentitySecurity | Regular       |                           90 |
```

PowerShell API Extraction

1. Download template PowerShell scripts from “CyberArk-PAM-QuickSight-BI-Dashboard\data-ingest-scripts.”
2. Edit "data-ingest-scripts\config.json" with project-specific details:



3. Decide on secrets management approach:
   a. CyberArk API user:
      i. Secret retrieval from CyberArk CCP ‘Get Password’ API:

```
# ----------------------------------
# Option 1: Retrieve API user password from CyberArk CCP (most secure)
# Uncomment this entire section and comment out other password sections if
'''
Write-Host "Retrieving API user password from CyberArk CCP..."
# Define CCP API parameters
$ccpBaseUrl = $config.ccpBaseUrl
$safeName   = $config.safeName
$objectName = $config.objectName
$appId      = $config.appId

# Construct the CCP request URI
$ccpUri      = "$ccpBaseUrl/AIMWebService/api/Accounts?AppID=$appId&Safe=$s
$ccpResponse = Invoke-RestMethod -Uri $ccpUri -Method Get

# Extract password from CCP response
$password = $ccpResponse.Content
Write-Host "Successfully retrieved password from CCP."
'''
```

ii. Password stored in ‘config.json’ and passed into PowerShell:

```
# ------------------------------------
# Option 2: Retrieve API user password from config file (not secure)
# Keep this section commented out if using CyberArk CCP
$password = $config.password
Write-Host "Using password from config file (not recommended for production)."
```

b. AWS CLI user:

i. Secret retrieval from AWS Secrets Manager:

```
# ------------------------------------------------------------------
# Option 1: Retrieve AWS credentials from Secrets Manager (most secure)
# Uncomment this entire section and comment out config-file-based AWS
# credentials if you want to securely fetch them from Secrets Manager.
'''
Write-Host "Retrieving AWS credentials from AWS Secrets Manager..."

$awsSecretName = $config.awsSecretName
$awsRegion     = $config.awsRegion

# Get the secret value from AWS Secrets Manager
$secretValueRaw  = aws secretsmanager get-secret-value --secret-id $awsSecre
$secretValueJson = $secretValueRaw.SecretString | ConvertFrom-Json

# Extract access key and secret key from the JSON in Secrets Manager
$awsAccessKeyId     = $secretValueJson.aws_access_key_id
$awsSecretAccessKey = $secretValueJson.aws_secret_access_key

Write-Host "Successfully retrieved AWS credentials from Secrets Manager."
'''
```

ii. Secret stored in ‘config.json’ and passed into PowerShell:

```
# ------------------------------------------------------------------
# Option 2: Use AWS credentials from config file (less secure)
# Keep this active (and comment out Secrets Manager block) if you want
# to continue storing credentials in config.json
$awsAccessKeyId     = $config.awsAccessKeyId
$awsSecretAccessKey = $config.awsSecretAccessKey

Write-Host "Using AWS credentials from config file (not recommended for production)."
```

4. Run scripts to execute.
   a. Create a Scheduled Task (Windows) or cron job (Linux) to schedule script execution.
      i. Service user for the Scheduled Task can be managed with the CPM.
      ii. CCP supports Scheduled Task secrets retrieval.
      iii. Adjust time limits in scripts if schedule is not 24 hours.

AWS Lambda

1. Functions Overview: Three functions manage the ETL process for accounts, safes, and platforms data. Each function reads from S3 and writes to the RDS database.
2. General Configuration:

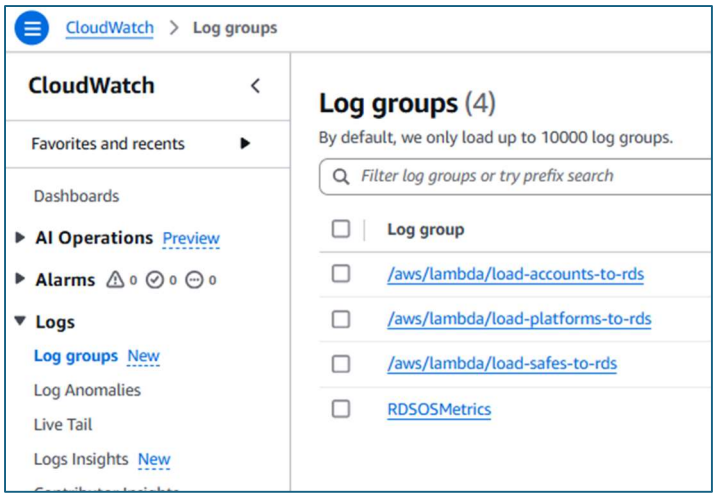
   a. Runtime: Python 3.11.
      i. For Postgres compatibility.
   b. Memory Size: 128 MB.
      i. 96 MB max memory used for ~500 keys ingest.
   c. Timeout: 180 seconds.
      i. 3 seconds to ingest ~500 keys via API to ISPSS.
   d. Ephemeral Storage: 512 MB for temporary data.
   e. Architecture: x86_64.
   f. Security: Enforced through VPC configuration and IAM roles.
3. Layers and Dependencies:
   a. Library Dependency: Custom Lambda Layer “psycopg2-libraries”.
   b. Layer Location: “./psycopg2-libraries” directory.
   c. Purpose: Facilitates connections to the PostgreSQL database.
   d. Compatible Runtimes: Python 3.11.
4. Networking:
   a. VPC: Connect to project VPC.
   b. Subnets: Select one or more private subnets with a route to Amazon S3 (with a route to a NAT GW on 0.0.0.0/0).
   c. Security Groups:
      i. Outbound access to PostgreSQL database on port 5432.
      ii. Outbound access to S3 via route to NAT GW on private subnet.
5. Event Configuration:
   a. Schedule Trigger: Executes once daily “(rate(1 day))”.
6. IAM Role:
   a. Permissions to the S3 bucket containing data:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:ListBucket",
                "s3:GetObject",
                "s3:PutObject"
            ],
            "Resource": [
                "arn:aws:s3:::your-bucket-name",
                "arn:aws:s3:::your-bucket-name/*"
            ]
        }
    ]
}
```

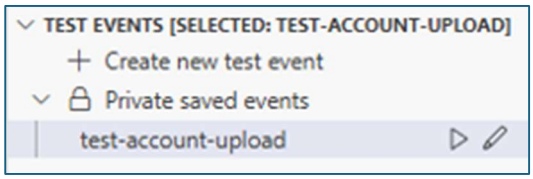
b. EC2 permissions required to join a VPC:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ec2:CreateNetworkInterface",
                "ec2:DescribeNetworkInterfaces",
                "ec2:DeleteNetworkInterface",
                "ec2:AttachNetworkInterface"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "ec2:AssignPrivateIpAddresses",
                "ec2:UnassignPrivateIpAddresses"
            ],
            "Resource":
"arn:aws:ec2:<region>:<accountNumber>:network-
interface/*"
        }
    ]
}
```

c. CloudWatch permissions for logging:
   i. logs:CreateLogGroup.
   ii. logs:CreateLogStream.
   iii. logs:PutLogEvents.
   iv. One log group created f Lambda function:



   d. If using Secrets Manager, include the following permissions:
      i. secretsmanager:GetSecretValue.
      ii. kms:Decrypt (if using a custom KMS key).
7. Tests:
   a. Create a custom test and copy the appropriate JSON from the GitHub repository.
      i. Update the filename to match your test data. This should be the filename of an API ingest file in the S3 bucket.
   b. Click the 'play' button to run the unit test.
      i. Monitor logs for errors and performance issues in CloudWatch.



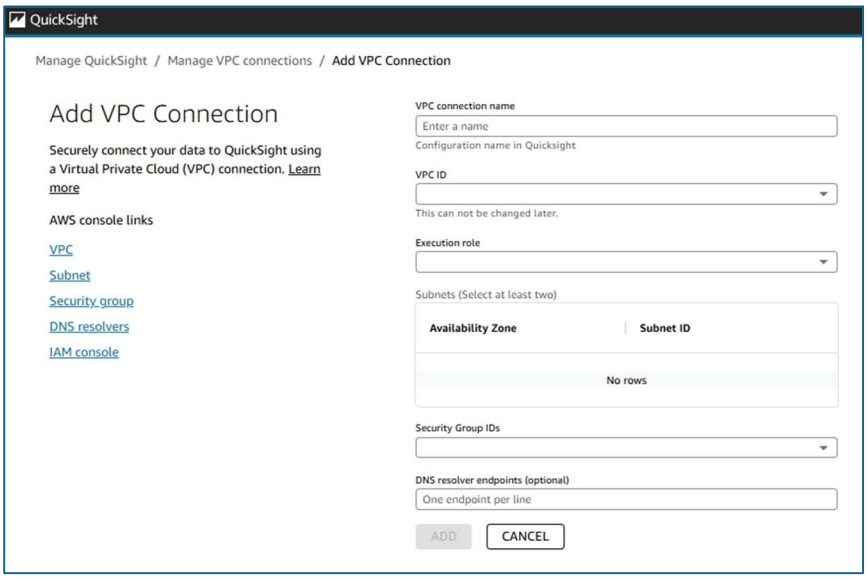
8. Deploy the Lambda script.

Amazon CloudWatch and SNS

1. Click the link to the CloudWatch logs on any Lambda 'Monitor' tab.
2. If desired, configure a Subscription and Topic in SNS for alerting.

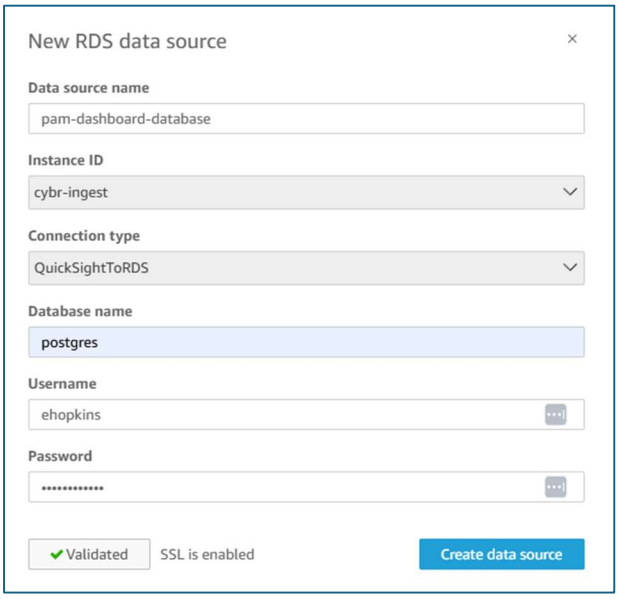
Amazon QuickSight

1. VPC Connection:
   a. Direct VPC connection through wizard during setup.
   b. Security Group access to RDS.

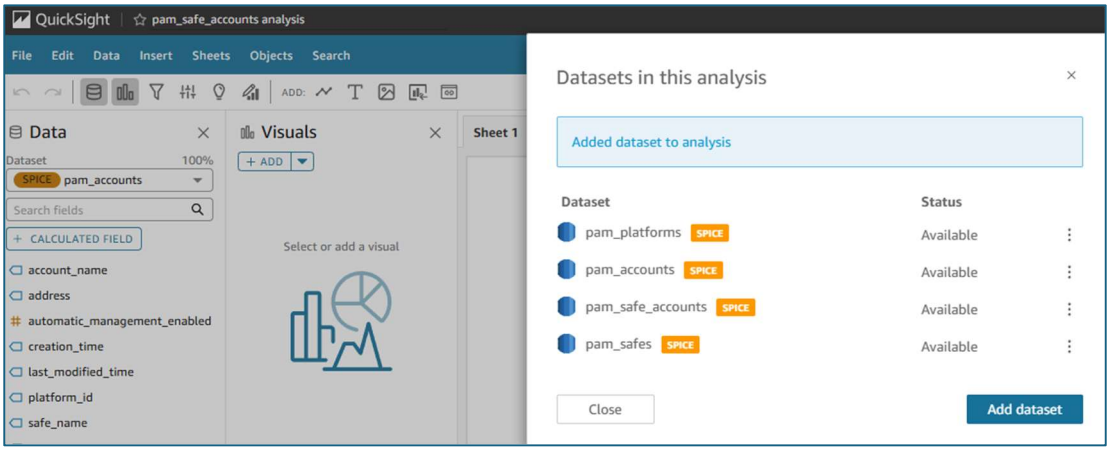


2. Data Source Setup:
   a. Connection Type: VPC connection QuickSightToRDS.
   b. Database Name: E.g., postgres.
   c. Username: E.g., username.

d. SSL: Enabled and validated.

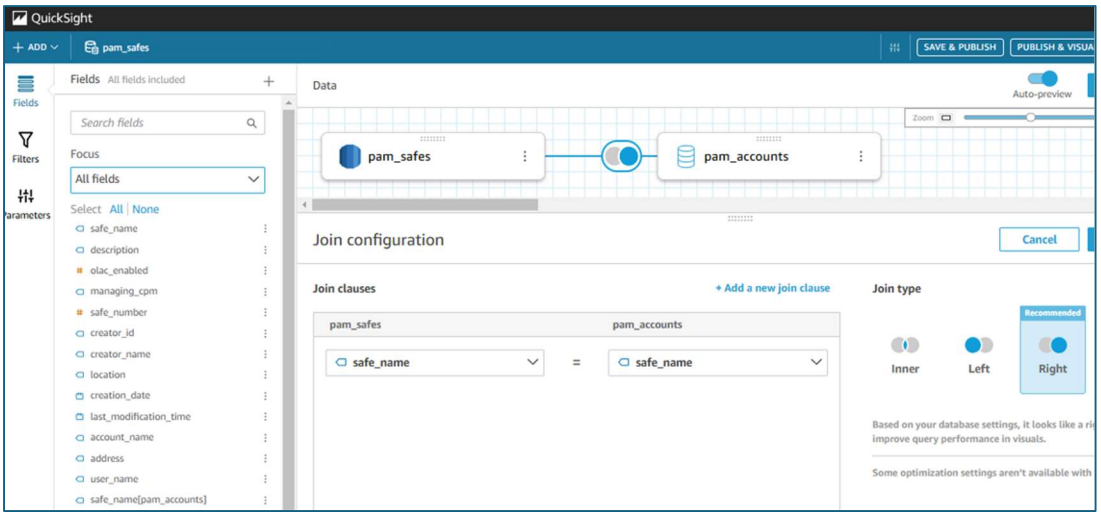


3. Datasets and Joins:
   a. Datasets:
      i. pam_accounts.
      ii. pam_safes.
      iii. pam_platforms.

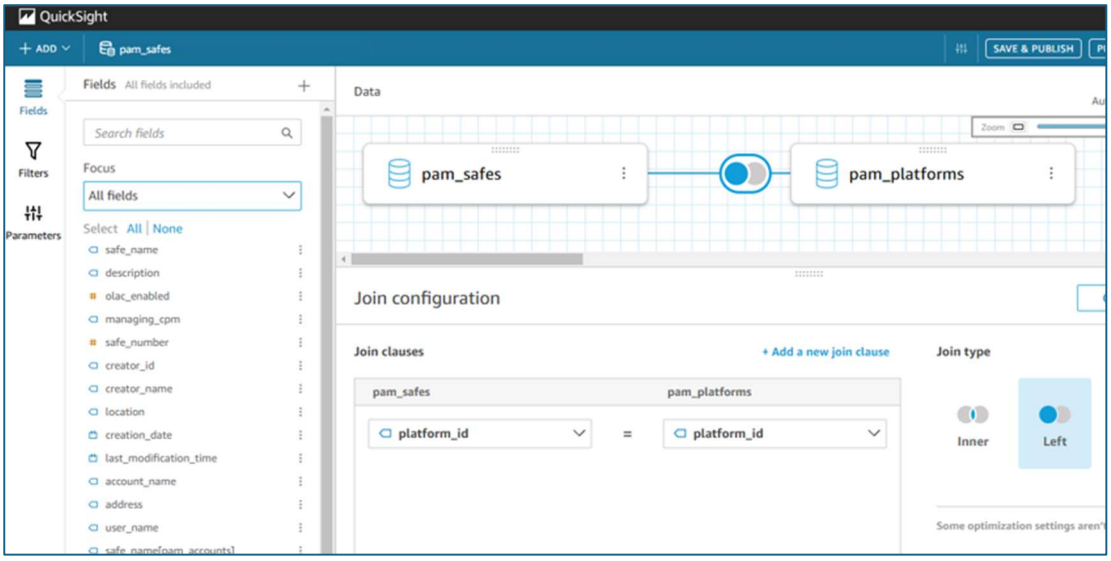


   b. Joins:

i. pam_safes joined with pam_accounts on safe_name:

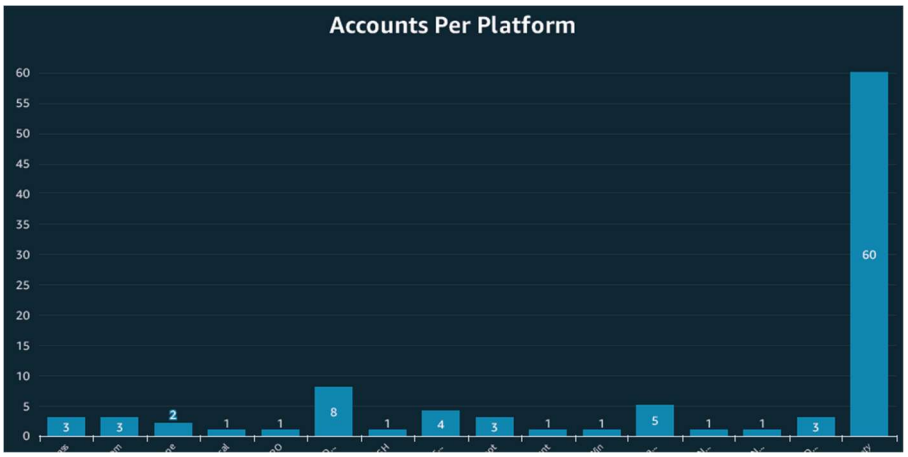


ii. pam_safes (new combined set) joined with pam_platforms on platform_id:

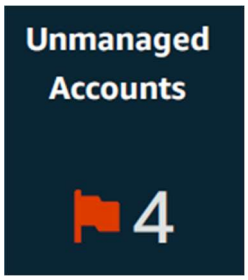


iii. Use right and left joins as optimized for performance.

4. Dashboard Customization:
   a. Add 'Calculated Fields' to enhance data visualization. For example:
      i. Accounts per Safe:
         1. countDistinct({account_name}) aggregated by {safe_name}.
      ii. Accounts per Platform:
         1. countDistinct({account_name}) aggregated by {platform_id} or {platform_name}.

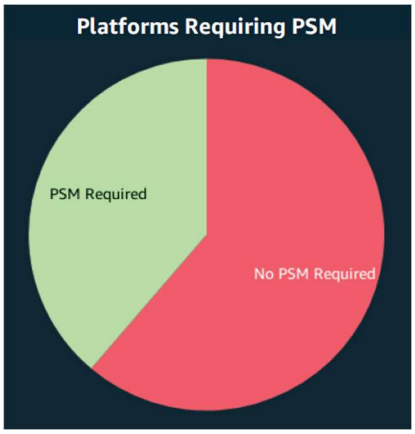


iii. Days Since Account Last Modified:
    1. dateDiff({last_modified_time}, now(), "DD").
iv. Days Since Safe Last Modification:
    1. dateDiff({last_modification_time}, now(), "DD").
v. Age of Safe (in Days):
    1. dateDiff({creation_date}, now(), "DD").
vi. Accounts Missing Automatic Management:
    1. ifelse({automatic_management_enabled} = 0, 1, 0).

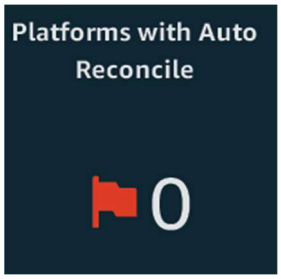


vii. Platform SystemType Category (To group “Windows,” “*NIX,” “Database,” etc.):
    1. ifelse({system_type} = "Windows", "OS - Windows",{system_type} = "*NIX", "OS - Unix/Linux",{system_type} = "Database", "DB Platforms",{system_type} = "Network Device", "Networking", "Other”).
viii. Platforms Requiring PSM:

1. ifelse({require_psm} = 1, "PSM Required", "No PSM").

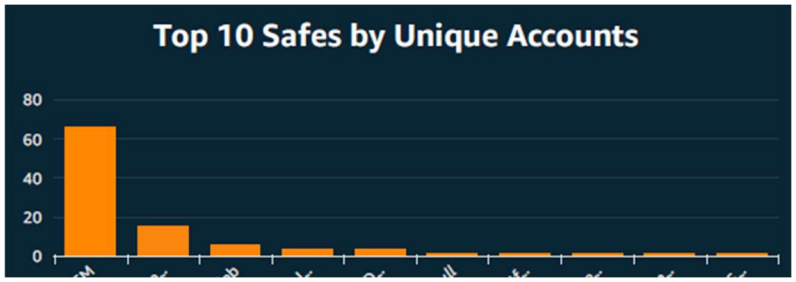


ix. Platform Reconcile Coverage:
   1. ifelse({automatic_reconcile} = 1, "Auto Reconcile", "Manual Reconcile").



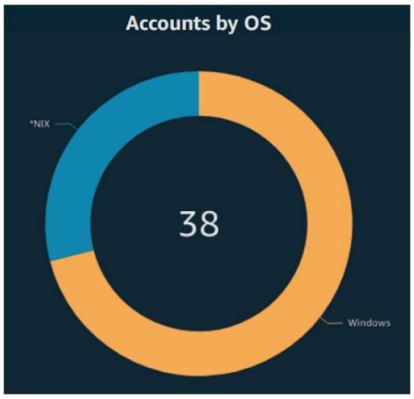
x. Time Between Safe Creation and Last Modification:
   1. dateDiff({creation_date}, {last_modification_time}, "DD").

5. Create Visualizations. For example:
   a. Bar Chart: “Top 10 Safes by Account Count”:
      i. X-Axis: safe_name.
      ii. Value: countDistinct(account_name).
      iii. Filter: Top ten.

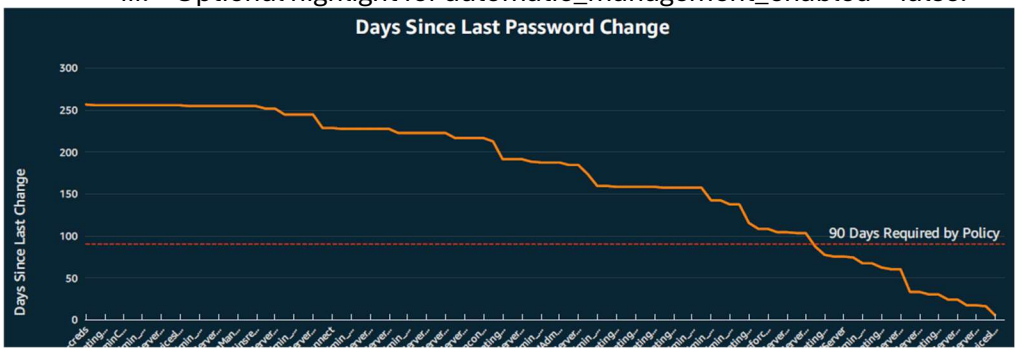


   b. Pie/Donut: “Accounts by Platform Category”:
      i. Group By: A custom “systemType” category.

ii. Value: countDistinct(account_name).

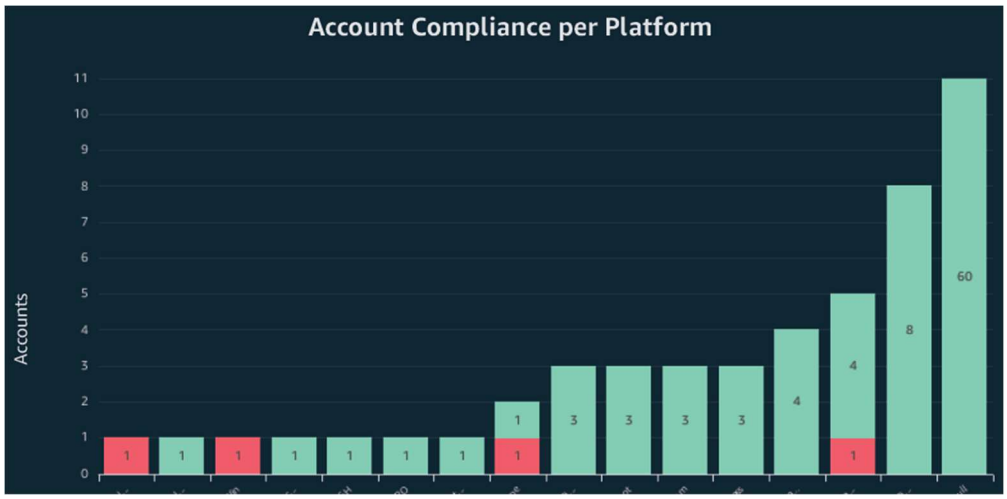


c. Table: “High-Risk Accounts”:
   i. Columns: account_name, safe_name, platform_name, “Days Since Account Last Modified.”
   ii. Sort Descending: “Days Since Account Last Modified.”
   iii. Optional highlight for automatic_management_enabled = false.

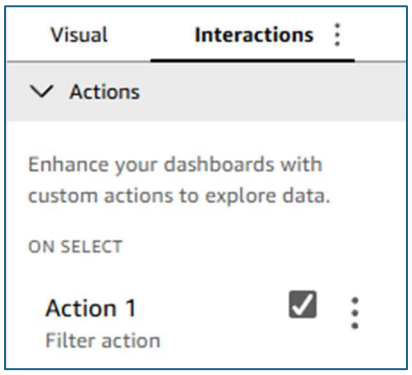


d. Line Chart: “Safes Created Over Time”:
   i. X-Axis: creation_date (month grouping).
   ii. Y-Axis: countDistinct(safe_name).
e. Clustered Bar: “Accounts per Platform, Grouped by Management Setting”:
   i. X-Axis: platform_name.
   ii. Group/Color: automatic_management_enabled.

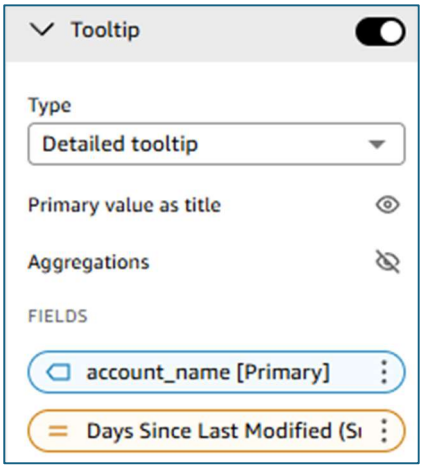
iii. Value: countDistinct(account_name).



f. Apply actions such as filter-based interactions for dynamic user exploration:

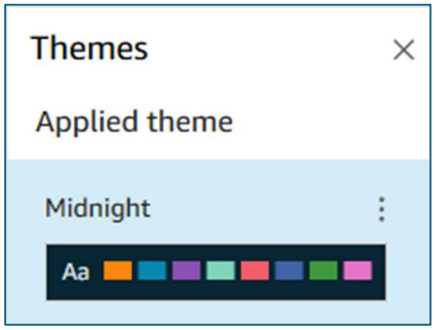


g. Configure tooltips with primary fields like account_name and calculated metrics (e.g., days since last modified):



6. Apply theme:

a. Active Theme: Midnight for POC.



7. Publish dashboard:
    a. Save and publish dashboards to make them accessible to stakeholders.
    b. Leverage Auto-preview to iterate on visualizations efficiently.
    c. Demonstrate filtering options on the published Dashboard.

Potential Enhancements

Front End Development

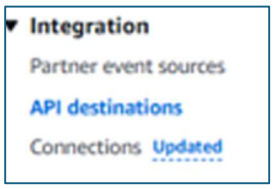
Create a React or Vue.js dashboard to eliminate QuickSight requirements.

Expanded Ingest Methods

Develop ETL procedures for data such as:

- Vault logs (italog, trace.d*).
- Vault configurations (dbparm, LDAPConf).
- Server performance details (CPU, Memory, Network).
- Component logs (CPM, PSM, PSM for SSH).
- Component configurations.
- System Health.
- Certificate management and renewal.

Support AWS-native ingest with EventBridge API integration:



Expanded ETL Options

Provide integration options for:

- Larger datasets using AWS Glue and/or Amazon Athena.

Expanded Monitoring and Alerting

Expand monitoring and alerting with:

- Amazon CloudWatch dashboard(s) for alerts and system monitoring.
- Amazon SNS topics and notifications for events.

CI/CD Automation

Automate updates and deployment with:

- GitHub Actions workflows.

Infrastructure as Code

Package supported infrastructure for deployment with AWS CloudFormation.

Appendix A: Useful APIs for KPIs

General approach to using API data for KPI development:

1. Extract:
    a. Use AWS Lambda to pull data from CyberArk APIs on a regular schedule.
    b. Use Scheduled Tasks or cron jobs to schedule on-premises ingests.
        i. API results.
        ii. Vault and component logs.
        iii. PVWA reports.
2. Transform:
    a. Remove duplicate data.
    b. Reformat timestamps:
        i. Safe creation time is in Unix epoch **seconds.**
            1. E.g., 1716996751 = 2024-07-08 13:37:03+00.
        ii. Last modification time is in Unix epoch **microseconds.**
            1. E.g., 1738726235278654 = 2024-07-08 13:37:03+00.
    c. Convert raw fields into standardized labels.
3. Aggregate:
    a. Join relevant datasets (e.g., accounts with audit logs, safes with accounts) during ingest, or at the 'Analyze' step.
4. Load:
    a. Insert transformed data into RDS tables.
5. Analyze:
    a. Import the dataset to AWS QuickSight or a custom React frontend to visualize key insights.
    b. Define 'calculated fields' to perform logic against one or more data points, creating custom analyses.
        i. E.g., Managed accounts per Safe.
        ii. E.g., Platforms with over 80% password compliance.
    c. Add filter and search support to visuals.
    d. Define conditional formatting to show metric velocity.

Useful APIs

Privileged Account Inventory

- API Call: GET /Accounts.
- Data: List of privileged accounts with metadata such as platform, account name, last accessed, and password age.
- KPIs:
    a. Total Privileged Accounts – Count of all accounts in PAM.
    b. Accounts per Platform – Distribution of accounts across different platforms (e.g., Windows, Linux, Databases).
    c. Inactive Accounts – Accounts that have not been accessed within a defined threshold period (e.g., 90 days).
    d. Password Policy Compliance – Percentage of accounts compliant with the One-Time Password (OTP) policy.

Account Compliance & Health Checks

- API Call: GET /Accounts/{AccountID}/PolicyStatus.
- Data: Account compliance status with associated master policies and platform rules.
- KPIs:
    a. Compliance Rate – Percentage of accounts compliant with PAM security policies (e.g., OTP, password complexity, rotation frequency).
    b. Failed Compliance Checks – List of non-compliant accounts by failure category (e.g., password never changed).
    c. Top 5 Non-Compliant Platforms – Prioritize platforms with high non-compliance for targeted remediation.

Password Activity Logs

- API Call: GET /Audit/PasswordActivities.
- Data: Audit logs of password changes, retrievals, and usage events.
- KPIs:
    a. Password Change Success Rate – Percentage of successful vs. failed password change attempts.
    b. Access Request Trends – Historical analysis of privileged access attempts (e.g., weekly/monthly trends).
    c. Failed Access Attempts – Count and cause of failed access attempts.
    d. Most Accessed Accounts – List of accounts accessed most frequently.

Access Control & Session Management

- API Call: GET /Sessions.
- Data: List of active or terminated privileged sessions with timestamps and users.
- KPIs:
    a. Active Sessions – Current count of active privileged sessions.
    b. Average Session Duration – Average time users maintain privileged access per session.
    c. Privileged User Activity Heatmap – Visualization of access patterns by time and resource.

Security Event Monitoring

- API Call: GET /Audit/Events.
- Data: System audit logs capturing key security events (e.g., failed logins, unauthorized access).
- KPIs:
    a. Security Incidents – Number of critical security events such as unauthorized login attempts or policy violations.
    b. Access Anomalies – Identification of unusual activity patterns (e.g., repeated failed access from a single source).
    c. Incident Response Time – Time taken to detect, acknowledge, and respond to security incidents.

Managed Device and User Inventory

- API Call: GET /Users & GET /Platforms.
- Data: Inventory of users with access and devices/platforms under management.
- KPIs:
    a. Privileged User Count – Number of users with privileged access.
    b. Platform Coverage – Number and type of platforms managed by PAM.
    c. User Onboarding Metrics – Time required to onboard users or platforms to PAM.

Rotational Policy Analytics

- API Call: GET /Accounts/{AccountID}/NextChangeTime.
- Data: Next scheduled password rotation time for each account.
- KPIs:
    a. Average Password Rotation Time – Average time between password rotations across accounts.
    b. Overdue Rotations – Percentage of accounts overdue for password rotation.
    c. Rotation Efficiency – Measurement of successful vs. failed rotations over time.

Privileged Account Discovery (Optional if available)

- API Call: GET /Discover/Results.
- Data: Results of scans for unmanaged or unknown privileged accounts.
- KPIs:
    a. Unmanaged Accounts Detected – Number of privileged accounts discovered but not onboarded.
    b. Discovery Trends – Historical trend of new accounts detected by PAM discovery scans.

SLA & Compliance Reporting

- Data Aggregation: Consolidated from multiple API calls above.
- KPIs:
    a. SLA Compliance – Percentage of PAM services meeting defined service level agreements (e.g., password rotations completed on time).
    b. Policy Enforcement Metrics – Percent of accounts subject to OTP and other critical policies.
    c. Audit Coverage – Number and percentage of accounts included in audit reporting.