Privaclave AWS Deployment Manual-Version_1.0

Pre-requisites:

Create an EC2 instance of at least *t3.medium* configuration with Red Hat Enterprise Linux.

Step 1: Prepare the Setup

- Gain root access:

```bash
sudo -i
```

- Update and install **wget**

```bash
sudo yum update -y
sudo yum install -y wget
```

- Install **unzip** (if not already installed)

```bash
sudo yum install -y unzip
```

- Download the **privaclave_setup_1.0.zip** file
  wget https://github.com/privaclave-internal/privaclave-setup/raw/main/privaclave_setup_1.0.zip -O privaclave_setup_1.0.zip

- Unzip the setup file

```bash
unzip privaclave_setup_0.0.1.zip
```

Step 2: Configure AWS Credentials

- Navigate to the unzipped privaclave_setup_0.0.1 directory

```bash
cd privaclave_setup_0.0.1
```

Then Run the following command to get required permissions

- Apply (Read , Write and execute) permission → **chmod 777 privaclave_setup_0.0.1**
- **yum install dos2unix**
- **dos2unix setup.sh**

- Open **AwsConfig.properties** file for editing

Type : vi AwsConfig.properties

*Press **Insert** button -> Update the file with your **AWS credentials***

```properties
AwsAccessKeyId=YOUR_AWS_ACCESS_KEY_ID
AwsSecretAccessKey=YOUR_AWS_SECRET_ACCESS_KEY
EC2HostedRegion=YOUR_EC2_REGION
privaclaveComponentRoleArn=arn:aws:iam::014223972444:role/thirdpartys3access
```

- **privaclaveComponentRoleArn** role is fixed from Privaclave technical team so no need to change that.
- **Press Escape -> :wq (Write and Quit) and save the changes.**

Step 3: Obtain AWS Access Key and Secret Access Key

1. Sign in to the AWS Management Console:
2. Open AWS Management Console.
3. Sign in with your AWS account credentials.
4. Navigate to the IAM Console:

- In the AWS Management Console, type **IAM** in the search bar and select **IAM**. **Select the User:**
- In the IAM console, click on Users in the navigation pane.
- Click on the name of the user for whom you want to create access keys.

- Create Access Keys:

Click on the Security credentials tab. Scroll down to the Access keys section. Click on Create access key.

A dialog box will appear showing the **Access key ID and Secret access key**.

Important: *Copy the Secret access key immediately as you will not be able to retrieve it later. Store it securely*.

Step 4: Run the Setup Script

- Run the setup script

```bash
./setup.sh
```

This will install the followings

1. **JDK 1.8.341** : verify installation using **'java –version'** command.
2. **Apache Tomcat Web Server 9.0.91** : It will be installed at **/opt/tomcat** folder.

Step 5: Install MariaDB

- Run the MariaDB installation script

```bash
./mariadbinstall.sh
```

Step 6: Deploy Privaclave Components

- Run the deployment command

```bash
java -jar PRIVACLAVE_AWS_COMPONENT_DEPLOYER_0.0.1.jar
```

This will perform the following operations

1. Create Lambda functions as per the defined configuration.
2. Pull **AutoPilot<VERSION_ID>.ja**r from **PRIVACLAVE_COMPONENTS_BUCKET** and upload it to the Lambda function.
3. Pull **CockpitEngine.war** and **CockpitAgent.war** from **PRIVACLAVE_COMPONENTS_BUCKET** and deploy them to the**/opt/tomcat/webapps** directory.
4. Configure **Cockpit Engine DB** and related DB Configuration.

Step 7 : IAM User Creation and Inline Policy Configuration

Overview

This document outlines the steps to create an IAM user named `ApiGateWayAuthenticator`, attach the `AmazonAPIGatewayInvokeFullAccess` AWS managed policy, and create an inline policy named `IAMPolicyForPolicyURL`. The inline policy will govern the decryption capabilities based on specific roles defined by the user.

Steps to Create IAM User and Attach Policies

1. Create an IAM User:

- **User Name:** `ApiGateWayAuthenticator`
- **Permissions:**
  - Attach the AWS managed policy `AmazonAPIGatewayInvokeFullAccess` to the IAM user.
  - Create an inline policy named `IAMPolicyForPolicyURL` with the following content.

2. Inline Policy: `IAMPolicyForPolicyURL`

This policy controls access to a KMS key (`arn:aws:kms:<EC2_Region>:<AWS_ACCOUNT_ID>:key/GetPolicyURL`) based on the role specified in the request tags. The policy grants or denies decryption permissions depending on the role of the user.

Policy Document Explanation

Version:

- The version number of the policy language (`2012-10-17` is the latest version and most commonly used).

Statements:

1. **Allow Statement:**
   - **Effect:** `Allow`
   - **Action:** `kms:Decrypt` – Grants permission to decrypt using the KMS key.
   - **Resource:** Specifies the KMS key (`arn:aws:kms:<EC2_Region>:<AWS_ACCOUNT_ID>:key/GetPolicyURL`).
   - **Condition:** Applies this permission only if the `Role` tag in the request is `USER_APP_DEFINED_ROLE_X`.
2. **Deny Statement:**
   - **Effect:** `Deny`
   - **Action:** `kms:Decrypt` – Denies permission to decrypt using the KMS key.
   - **Resource:** Specifies the KMS key (`arn:aws:kms:<EC2_Region>:<AWS_ACCOUNT_ID>:key/GetPolicyURL`).
   - **Condition:** Applies this denial only if the `Role` tag in the request is `USER_APP_DEFINED_ROLE_Y`.

Sample Inline Policy: `IAMPolicyForPolicyURL`

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "kms:Decrypt",
            "Resource": "arn:aws:kms:<EC2_Region>:<AWS_ACCOUNT_ID>:key/GetPolicyURL",
            "Condition": {
                "StringEquals": {
                    "aws:RequestTag/Role": "USER_APP_DEFINED_ROLE_X"
                }
            }
        },
        {
            "Effect": "Deny",
            "Action": "kms:Decrypt",
            "Resource": "arn:aws:kms:<EC2_Region>:<AWS_ACCOUNT_ID>:key/GetPolicyURL",
            "Condition": {
                "StringEquals": {
                    "aws:RequestTag/Role": "USER_APP_DEFINED_ROLE_Y"
                }
            }
```

```
        }
    ]
}
```

HashiCorp Vault Installation on Red Hat EC2 Instance

This guide provides step-by-step instructions to install HashiCorp Vault on an EC2 instance running Red Hat. We will use wget to download the Vault binary, unzip to extract it, and configure it to run as a systemd service.

Prerequisites

- An EC2 instance running Red Hat.
- sudo or root access to the instance.

Step 1: Update and Install Dependencies

First, update the package list and install necessary dependencies like wget and unzip:

- *sudo yum update -y*
- *sudo yum install -y wget unzip*

Step 2: Download HashiCorp Vault

Download the latest version of HashiCorp Vault from the official HashiCorp website:

- *wget https://releases.hashicorp.com/vault/1.14.0/vault_1.14.0_linux_amd64.zip*

Note: Replace 1.14.0 with the latest version available from the official HashiCorp Vault downloads page.

Step 3: Unzip the Vault Binary

After downloading the Vault binary, unzip the file:

- unzip vault_1.14.0_linux_amd64.zip

Move the vault binary to /usr/local/bin to make it accessible from anywhere:

- *sudo mv vault /usr/local/bin/*

Step 4: Verify the Installation

Check that Vault is correctly installed by running:

- *vault --version*

This should display the version of Vault installed.

Step 5: Set Up Vault as a Systemd Service

5.1 Create a Vault User

For security purposes, it is recommended to run Vault under a non-root user. Create a dedicated vault user:

- *sudouseradd --system --home /etc/vault.d --shell /bin/false vault*

5.2 Create Configuration Directory

Create a configuration directory for Vault:

- *sudo mkdir /etc/vault.d*
- *sudo chown -R vault:vault /etc/vault.d*
- *sudo chmod 755 /etc/vault.d*

5.3 Create a Systemd Service File

Create a systemd service file for Vault:

- *sudo nano /etc/systemd/system/vault.service*

Add the following content to the file:

```
[Unit]
Description=HashiCorp Vault
Requires=network-online.target
After=network-online.target
[Service]
EnvironmentFile=-/etc/sysconfig/vault
ExecStart=/usr/local/bin/vault server -config=/etc/vault.d/config.hcl
Restart=always
User=vault
Group=vault
LimitNOFILE=65536
[Install]
WantedBy=multi-user.target
```

Save the file and exit.

Step 6: Enable and Start Vault Service

Reload systemd to recognize the new service:

- *sudo systemctl daemon-reload*

Start and enable the Vault service to run at startup:

- *sudo systemctl start vault*
- *sudo systemctl enable vault*

Check the status of the Vault service:

- *sudosystemctl status vault*

Step 7: Configure Environment Variables

Set the VAULT_ADDR environment variable to point to your Vault server:

- *export VAULT_ADDR='http://127.0.0.1:8200'*

To make this persistent across sessions, add it to your shell profile:

- *echo "export VAULT_ADDR='http://127.0.0.1:8200'" | sudo tee -a /etc/profile*

Step 8: Secure the Vault Configuration

Ensure that the Vault configuration files and directories are owned by the vault user and have the correct permissions:

- *sudo chown -R vault:vault /etc/vault.d*
- *sudo chmod -R 640 /etc/vault.d/**

References

- HashiCorp Vault Official Documentation(Documentation | Vault | HashiCorp Developer)
- HashiCorp Vault Downloads Page

(Install | Vault | HashiCorp Developer)

- Running Vault in Production

(Deploy Vault | Vault | HashiCorp Developer)

This documentation provides the steps required to install and configure HashiCorp Vault on a Red Hat-based EC2 instance. You can refer to the official documentation for more detailed information..

Configure AWS KMS for Hashicorp vault

1. Creating an IAM Policy for KMS

Before attaching the policy, you may need to create a custom IAM policy that grants specific permissions to use your AWS KMS key.

Steps to Create an IAM Policy:

1. **Navigate to IAM in AWS Management Console:**
   - Go to the AWS IAM Console.
   - In the left navigation pane, click on Policies.
2. **Create a New Policy**:
   - Click on the Create policy button.
   - You can either use the Visual editor or switch to the JSON tab for more direct control.
3. **Define the Policy Permissions:**
   - In the Visual editor:
     - Choose Service: Select KMS.
     - Actions: Choose the specific actions your application will need:
       - kms:Encrypt: Allows the role/user to encrypt data using the KMS key.
       - kms:Decrypt: Allows the role/user to decrypt data.
       - kms:GenerateDataKey: Allows the role/user to generate a data key for encryption.
       - kms:DescribeKey: Allows the role/user to view details about the KMS key.
     - Resources: Specify the ARN of the KMS key you created. This restricts the permissions to that specific key.
   - JSON Example:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "kms:Encrypt",
        "kms:Decrypt",
        "kms:GenerateDataKey",
        "kms:DescribeKey"
      ],
```

```
      "Resource": "arn:aws:kms:us-east-1:123456789012:key/abcd-efgh-ijkl-mnop-qrstuvwx"
    }
  ]
}
```

- *Replace the Resource ARN with your specific KMS key's ARN.*

4. **Review Policy:**
   - Click Next: Tags if you want to add tags for easier identification (optional).
   - Click Next: Review to proceed.
   - Give the policy a meaningful name, such as VaultKMSAccessPolicy.
   - Review the policy summary and click Create policy.
5. Attaching the IAM Policy to a Role/User

Once the policy is created, you need to attach it to the IAM role or user that Vault or your application will use.

Steps to Attach a Policy:

1. **Navigate to the IAM Console:**
   - In the IAM dashboard, go to Roles or Users, depending on what your application is using.
2. **Choose the Role/User:**
   - Click on the role or user that you want to attach the policy to.
   - If you don't have a specific role or user, you can create one:
     - For a Role: Go to Roles>Create Role. Choose a trusted entity (e.g., EC2 if Vault is running on an EC2 instance) and then proceed with attaching the policy.
     - For a User: Go to Users>Add user and follow the prompts.
3. **Attach Policy**:
   - In the role/user's summary page, click on the Add permissions button.
   - Choose Attach policies directly.
   - Search for the policy you created earlier (e.g., VaultKMSAccessPolicy).
   - Select the policy and click Next: Review.
   - Finally, click Add permissions to attach the policy.
4. **Verification**

To ensure that the policy is attached correctly:

1. **Check Policy Attachment:**
   - In the role/user's permissions tab, verify that the policy appears under Permissions.
2. **Preparing the Vault Configuration File**

Vault's configuration file (vault.hcl or similar) is where you define how Vault interacts with AWS KMS for auto unsealing. Here's how to configure this:

Vault Configuration Structure

1. **Locate or create the Vault configuration file:**
   - The file is typically named vault.hcl, but the name may vary depending on your setup.
   - If the file doesn't exist, you can create it.
2. **Configure the Seal Stanza:**
   - Add a seal stanza in the configuration file to specify AWS KMS as the seal provider.
   - The seal stanza will look something like this:

```
seal "awskms" {
  region     = "us-east-1"                        # AWS Region where your KMS key is located
kms_key_id = "arn:aws:kms:us-east-1:123456789012:key/abcd-efgh-ijkl-mnop-qrstuvwx"  # ARN of your KMS key
}
```

3.Parameters Explained:

- region: The AWS region where your KMS key is located (e.g., us-east-1).
- kms_key_id: The ARN of the KMS key you created earlier. This tells Vault which key to use for encryption and decryption of the seal keys.
- Additional Configuration (Optional):
  - You can also configure additional parameters like endpoint, access_key, and secret_key if you need to use specific credentials or endpoints. However, in most cases, Vault will use the IAM role or credentials configured on the instance.

Example with additional parameters:

```
seal "awskms" {
  region     = "us-east-1"
kms_key_id = "arn:aws:kms:us-east-1:123456789012:key/abcd-efgh-ijkl-mnop-qrstuvwx"
access_key = "YOUR_ACCESS_KEY"                # Optional: Your AWS access key
secret_key = "YOUR_SECRET_KEY"                # Optional: Your AWS secret key
  endpoint   = "https://kms.us-east-1.amazonaws.com"  # Optional: Custom KMS endpoint
}
```

4. **Save and Exit:**
   - After making the necessary changes, save the configuration file.
5. **Starting or Restarting Vault**

Once you've configured the Vault configuration file, the next step is to start or restart the Vault service to apply the changes.

1. Start Vault (if not already running):
   - Use the command below to start Vault if it's not already running:

- vault server -config=**/path/to/your/vault.hcl**
- Ensure the path to your configuration file is correct.
- Restart Vault (if already running):
  - If Vault is already running, restart the service to apply the new configuration.
  - How you restart Vault depends on how it's managed (systemd, init.d, Docker, etc.).

For example, if you're using systemd:

- sudo systemctl restart vault
- Check the Status:
  - Once Vault has started, you can check its status to ensure everything is working correctly:
- vault status

6.If everything is configured correctly, Vault should be in an unsealed state without requiring manual unsealing.

API Gateway Configuration

1. **Create an API in API Gateway**

- Go to the **API Gateway** service in the AWS Console.
- Click **Create API**.
- Select **HTTP API** or **REST API** depending on your use case.
  - **HTTP API** is simpler and offers better performance, while **REST API** has more configuration options.
- Click **Build** or **Create API** depending on the type chosen.

2. **Configure a Lambda Integration in API Gateway**

- After creating the API, click **Add Integration**.
- Choose **Lambda Function** as the integration type.
- Select your Lambda function from the drop-down menu or type the name of the Lambda function you created earlier.
- Click **Add Integration**.

3. **Configure Routes**

- Go to **Routes**.
- Add a route by selecting a method (e.g., **GET**, **POST** or **ANY**) and providing the route path (e.g., /**hello**).
- Link this route to the Lambda integration created in the previous step.

4. **Configure Resources and Methods**

- Go to **Resources**.
- Click **Create Resource** and define a path for your API (e.g., /users).
- Under this resource, choose **Create Method** (e.g., **GET**, **POST**).
- Select **Lambda Function** as the integration type.
- In the Lambda Function box, type the name of your function.

5. **Deploy the API**

- In API Gateway, navigate to **Stages**.
- Click **Create** or **Deploy**.
- Choose a stage (e.g., dev).
- Click **Deploy**.
- After deployment, you will be provided with an **Invoke URL** for the API.

6. **Test the API**

- Use a tool like curl or Postman to make a request to the API's invoke URL (e.g., *https://<api-id>.execute-api.<region>.amazonaws.com/<stage>/<resource>*).
- The Lambda function should be invoked and return a response.