AWS Security & Encryption:KMS,Encryptio…

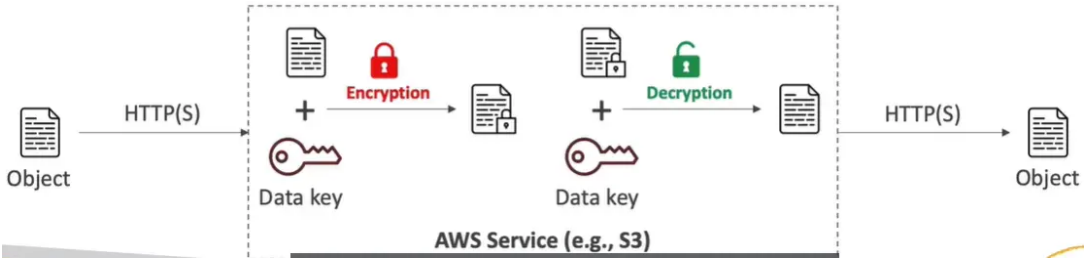
why encryption ?

Encryption in flight (TLS / SSL)

- data is encrypted before sending and decrypted after receiving
- TLS certificates help with encryption (HTTPS)
- encryption in flight ensures no MITM (man in the middle attack) can happen

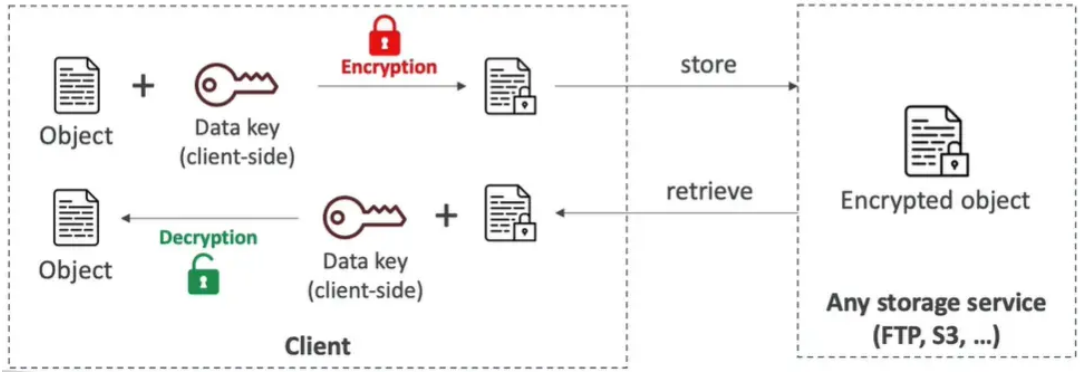
Server-side encryption at rest

- data is encrypted after being received by the server
- data is decrypted before being sent
- it's stored in an encrypted form thanks to a key (usually a data key)
- the encryption / decryption keys must be managed somewhere, and the server must have access to it



Client-side encryption

- data is encrypted by the client and never decrypted by the server
- data will be decrypted by a receiving client
- the server should not be able to decrypt the data
- cloud leverage Envelope Encryption



AWS KMS (Key Management Service)

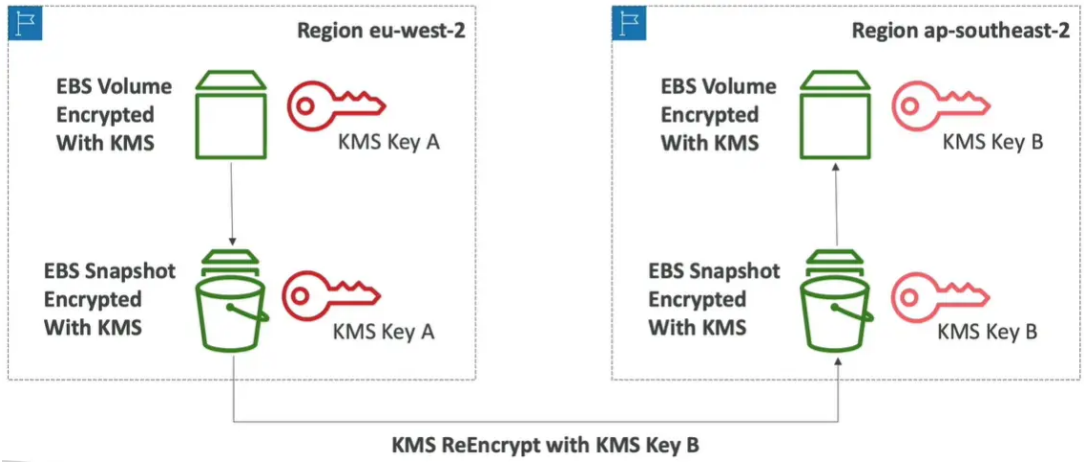
- anytime you hear ”encryption” for an AWS service, it’s most likely KMS
- AWS manages encryption keys for us
- fully integrated with IAM for authorization
- easy way to control access to your data
- able to audit KMS key usage using CloudTrail
- seamlessly [无缝地] integrated into most AWS services (EBS,S3, RDS,SSM...)
- Never ever store your secrets in plaintext, especially in your code !
  - KMS key encryption also available through API calls (SDK, CLI)
  - encrypted secrets can be stored in the code / environment variables

KMS keys types

- KMS keys is the new name of KMS customer master key
- Symmetric [对称] (AES – 256 keys)
  - single encryption key that is used to encrypt and decrypt
  - AWS services that are integrated with KMS use symmetric CMKs
  - you never get access to the KMS key unencrypted (must call KMS API to use)
- Asymmetric [非对称](RSA & ECC key pairs)
  - public (encrypt) and private key (decrypt) pair
  - used for encrypt/ decrypt, or sign / verify operations
  - the public key is downloadable, but you can’t access the private key unencrypted
  - use case : encryption outside of AWS by users who can’t call the KMS API

- Types of KMS keys
  - AWS Owned keys (free): SSE-S3, SSE-SQS,SSE-DDB (default key)
  - AWS Managed Key :free (aws/service-name, example:aws/rds or aws/ebs)
  - customer managed keys created in KMS: $1/month
  - customer managed keys imported : $1/month
  - + pay for API call to KMS ($0.03 / 10000 calls)
- Automatic key rotation
  - AWS-managed KMS key : automatic every 1 year
  - customer-managed KMS key : (must be enabled) automatic every 1 year
  - imported KMS key : only manual rotation possible using alias

cpoying snapshots across regions



KMS Key Policies

- control access to KMS keys,"similar" to S3 bucket policies
- difference: you cannot control acess without them

- Default KMS key policy
  - created if you don't provide a specific KMS key policy
  - complete access to the key to the root user = entire AWS account
- Custom KMS key policy
  - define users, roles that can access the KMS key
  - define who can administer the key

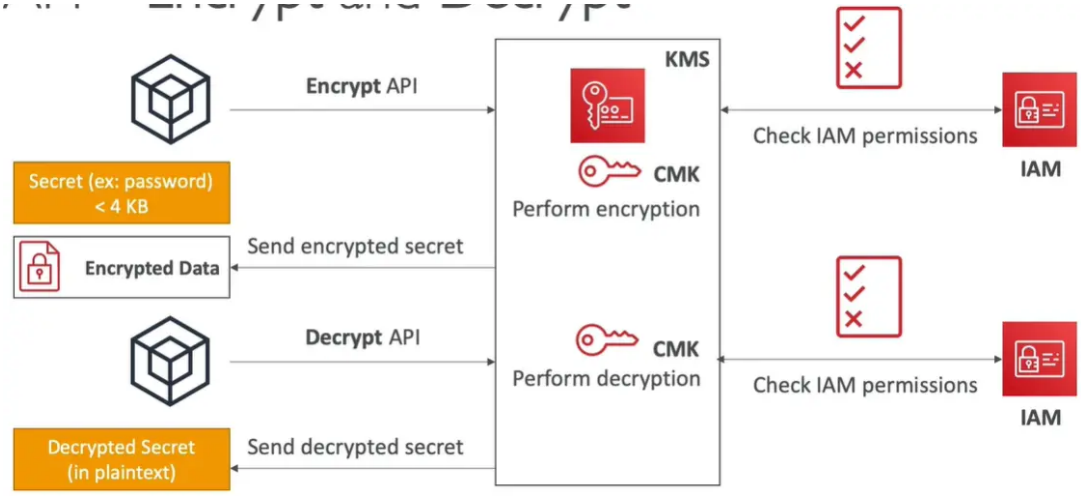
copying snapshots across accounts

```
{
    "Sid": "Allow use of the key with destination account",
    "Effect": "Allow",
    "Principal": {
      "AWS": "arn:aws:iam::TARGET-ACCOUNT-ID:role/ROLENAME"
    },
    "Action": [
      "kms:Decrypt",
      "kms:CreateGrant"
    ],
    "Resource": "*",
    "Condition": {
      "StringEquals": {
        "kms:ViaService": "ec2.REGION.amazonaws.com",
        "kms:CallerAccount": "TARGET-ACCOUNT-ID"
      }
    }
}
```

KMS Key Policy

1. create a snapshot, encrypted with your own KMS key (customer Managed key)
2. attach a KMS key policy to authorize cross-account access
3. share the encrypted snapshot
4. (in target) create a copy of the snapshot, encrypt it with a CMK in your account
5. create a volume from the snapshot

how dose KMS work ? API-Encrypt and Decrypt

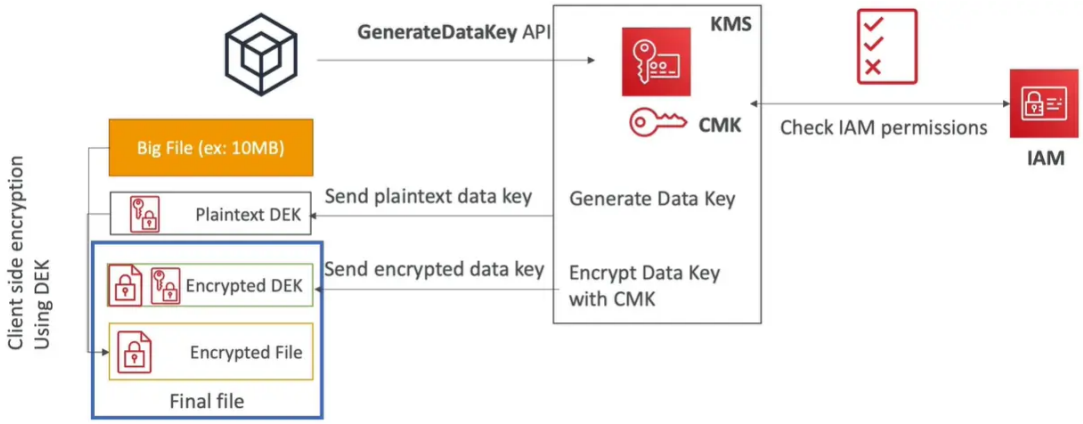


envelope encryption

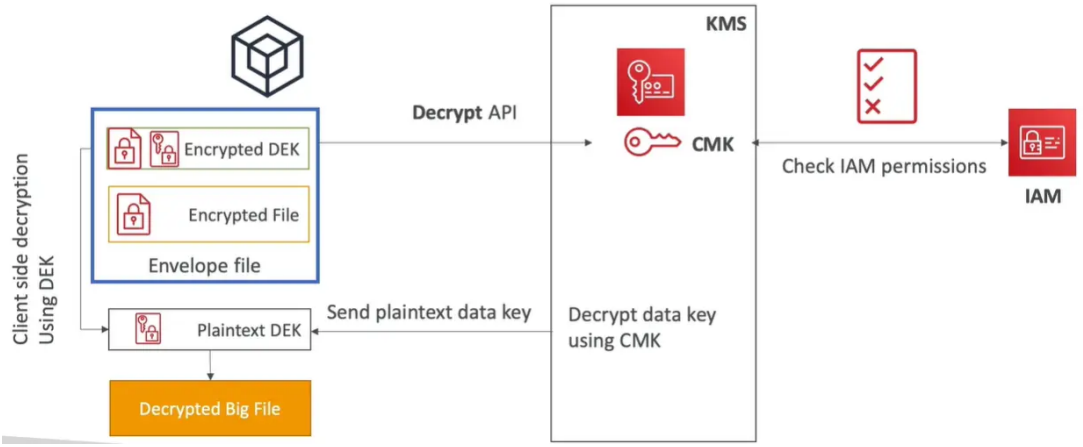
- KMS encrypt API call has a limit of 4KB
- if you want to encrypt > 4KB, we need to use Envelope Encryption
- the main API that will help us is the GenerateDataKey API

- for the exam : anything over 4KB of data that needs to be encrypted must use the envelope encryption == GenerateDataKey API

Deep dive into envelope encryption GenerateDataKey API



Deep dive into envelope encryption Decrypt envelope data



encryption SDK

- the AWS encryption SDK implemented envelope encryption for us

- the encryption SDK also exists as a CLI tool we can install
- implementations for java, python,c , javascript

- feature – data key caching
  - re-use data keys instead of creating new ones for each encryption
  - helps with reduing the number of calls to KMS with a security trade-off
  - use localCryptoMaterialsCache (max age, max bytes, max number of mesages)

KMS Symetric – API summary

- Encrypt: encrypt up to 4KB of data through KMS
- GenerateDataKey : generates a unique symmetric data key (DEK)
  - returns a plaintext copy of the data key
  - AND a copy that is encrypted under the CMK that you specify
- GenerateDataKeyWithoutPlaintext:
  - generate a DEK to use at some point (not immediately)
  - DEK that is encrypted under the CMK that you specify (must use Decrypt later)
- Decrypt: decrypt up to 4KB of data (including data encryption keys)
- GenerateRandom : Returns a random byte string

KMS Request Quotas [配额]

- when you exceed [超过] a request quota, you get a ThrottlingException [节流异常]

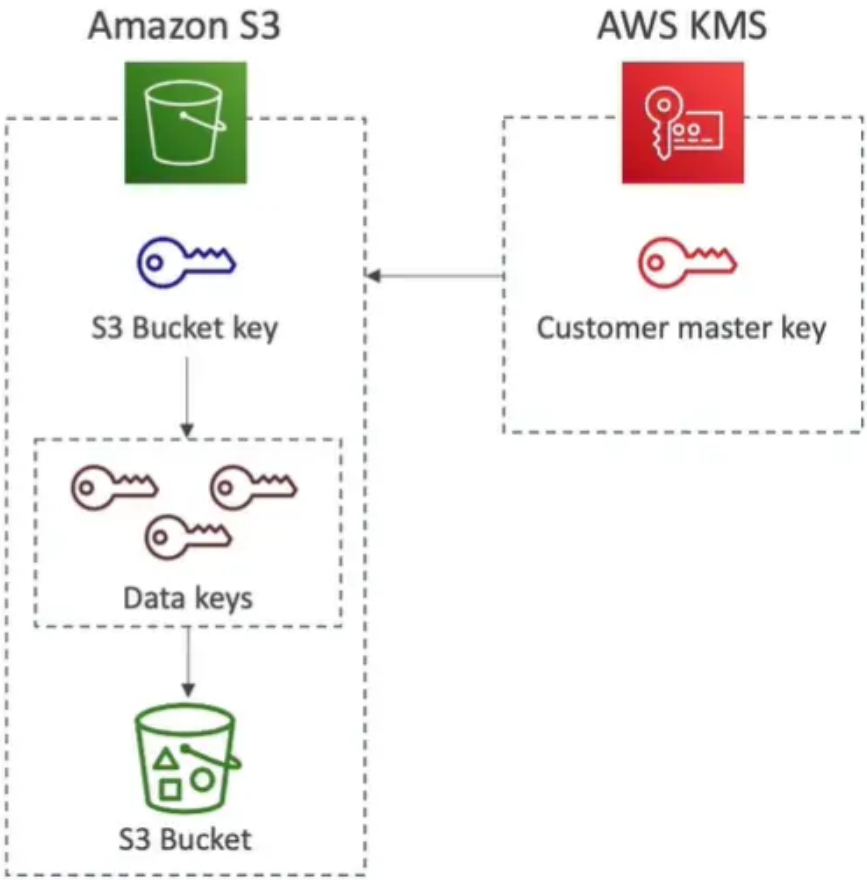
```
You have exceeded the rate at which you may call KMS. Reduce the frequency of your calls.
(Service: AWSKMS; Status Code: 400; Error Code: ThrottlingException; Request ID: <ID>
```

- to respond ,use exponential [指数] backoff (backoff [退避] and retry)
- for cryptographic operations, they share a quota
- this includes requests made by AWS on your behalf (ex: SSE-KMS)
- for GenerateDataKey, consider using DEK caching from the Encryption SDK
- you can request a request quotas increase through API or AWS support

API operation	Request quotas (per second)

Decrypt
Encrypt
GenerateDataKey (symmetric)
GenerateDataKeyWithoutPlaintext (symmetric)
GenerateRandom
ReEncrypt
Sign (asymmetric)
Verify (asymmetric)
	These shared quotas vary with the AWS Region and the type of CMK used in the request. Each quota is calculated separately.
	
	Symmetric CMK quota:
	• 5,500 (shared)
	• 10,000 (shared) in the following Regions:
	  • us-east-2, ap-southeast-1, ap-southeast-2, ap-northeast-1, eu-central-1, eu-west-2
	• 30,000 (shared) in the following Regions:
	  • us-east-1, us-west-2, eu-west-1
	
	Asymmetric CMK quota:
	• 500 (shared) for RSA CMKs
	• 300 (shared) for Elliptic curve (ECC) CMKs

S3 Bucket Key for SSE-KMS encryption



- new setting to decrease
  - number of API calls made to KMS form S3 by 99%
  - costs of overall KMS encryption with Amazon S3 by 99%
- this leverages data keys
  - a S3 bucket key is generated
  - that key is used to encrypt KMS objects with new data keys
- you will see less KMS CloudTrail events in CloudTrail

key policy –example

Default KMS Key Policy

```
{
    "Effect": "Allow",
    "Action": "kms:*",
    "Principal": {
        "AWS": "arn:aws:iam::123456789012:root"
    },
    "Resource": "*"
}
```

Allow Federated User

```
{
    "Effect": "Allow",
    "Action": [
        "kms:Encrypt",
        "kms:Decrypt",
        "kms:ReEncrypt*",
        "kms:GenerateDataKey*",
        "kms:DescribeKey"
    ],
    "Principal": {
        "AWS": "arn:aws:sts::123456789012:federated-user/user-name"
    },
    "Resource": "*"
}
```

Principal options in IAM policies

- AWS Account and Root User

```
"Principal": { "AWS": "123456789012" }
"Principal": { "AWS": "arn:aws:iam::123456789012:root" }
```

- IAM Roles

```
"Principal": { "AWS": "arn:aws:iam::123456789012:role/role-name" }
```

- IAM Role Sessions

```
"Principal": { "AWS": "arn:aws:sts::123456789012:assumed-role/role-name/role-session-name" }
"Principal": { "Federated": "cognito-identity.amazonaws.com" }
"Principal": { "Federated": "arn:aws:iam::123456789012:saml-provider/provider-name" }
```

- IAM Users

```
"Principal": { "AWS": "arn:aws:iam::123456789012:user/user-name" }
```

- Federated User Sessions

```
"Principal": { "AWS": "arn:aws:sts::123456789012:federated-user/user-name" }
```

- AWS Services

```
"Principal": {
    "Service": [
        "ecs.amazonaws.com",
        "elasticloadbalancing.amazonaws.com"
    ]
}
```

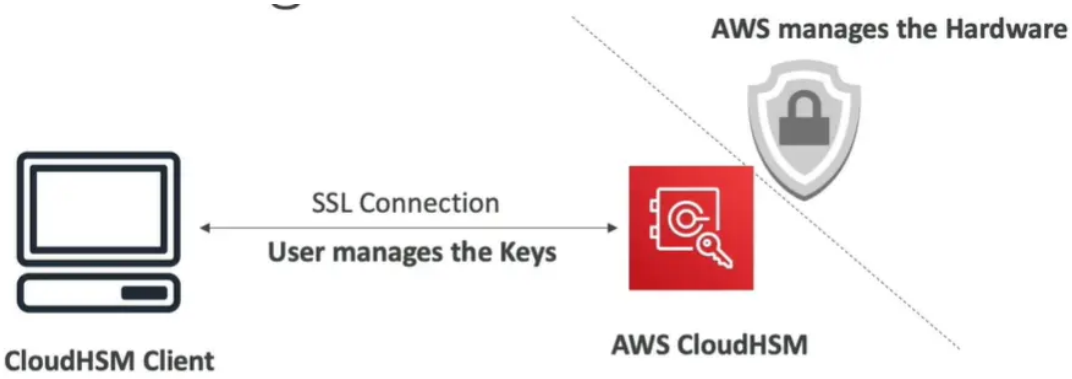
- All Principals

```
"Principal": "*"
"Principal": { "AWS": "*" }
```

CloudHSM

- KMS => AWS manages the software for encryption
- CloudHSM => AWS provisions encryption **hardware**
- Dedicated [专用] Hardware (HSM = Hardware Security Module)
- you manage your own encryption keys entirely (not AWS)
- HSM device is tamper [篡改] resistant [抵抗], FIPS 140-2 level 3 compliance [承诺]
- supports both symmetric and **asymmetric** encryption (SSL / TLS keys)
- no free tier available
- must use the CloudHSM Client Software
- Redshift supports CloudHSM for database encryption and key management
- **good option to use with SSE-C encryption**

CloudHSM Diagram

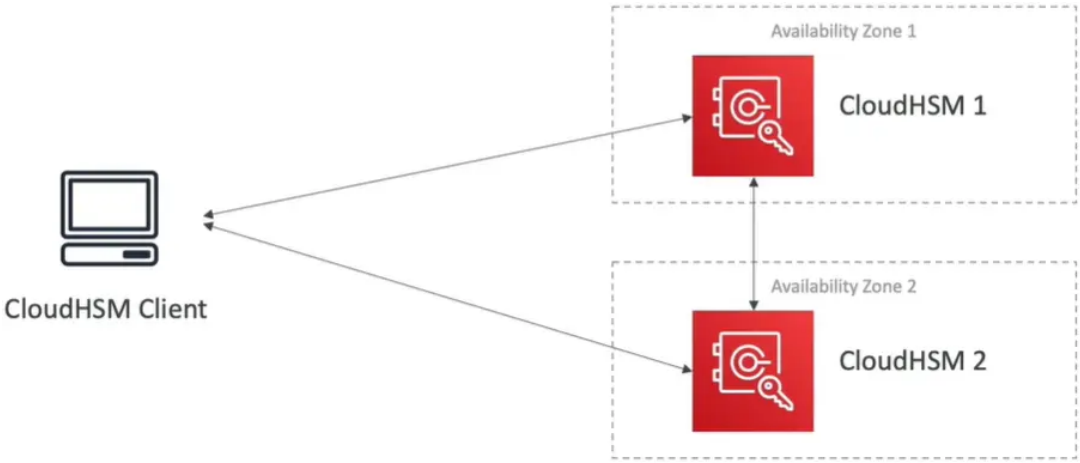


- IAM permissions

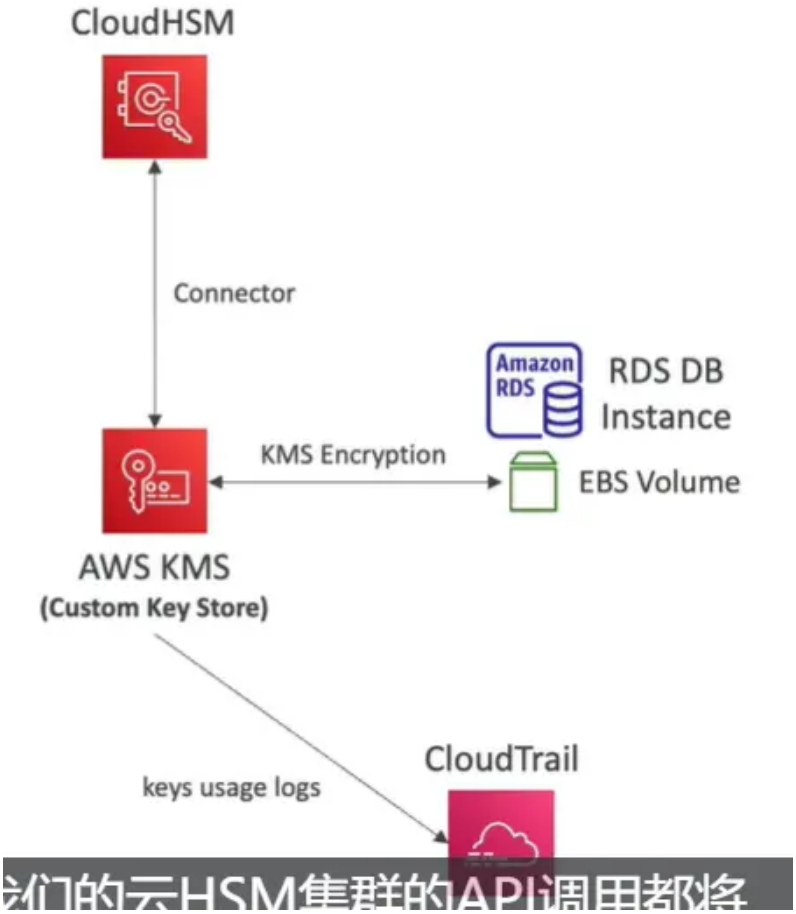
- CRUD an HSM Cluster
- CloudHSM Software:
  - Manage the keys
  - Manage the Users

CloudHSM – High Availability

- CloudHSM clusters are spread across Multi AZ (HA)
- great for availability and durability



integration with AWS services



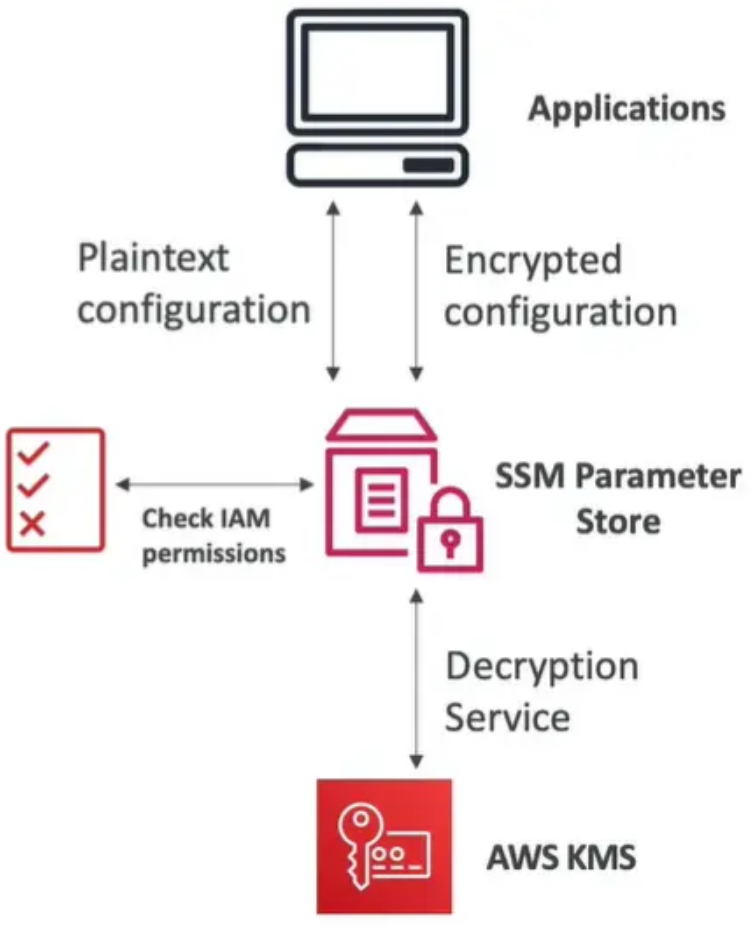
- through integration with AWS KMS
- configure KMS custom key store with CloudHSM
- example: EBS,S3,RDS...

CloudHSM vs KMS

Feature	AWS KMS	AWS CloudHSM
Tenancy	Multi-Tenant	Single-Tenant
Standard	FIPS 140-2 Level 3	FIPS 140-2 Level 3
Master Keys	• AWS Owned CMK • AWS Managed CMK • Customer Managed CMK	Customer Managed CMK
Key Types	• Symmetric • Asymmetric • Digital Signing	• Symmetric • Asymmetric • Digital Signing & Hashing
Key Accessibility	Accessible in multiple AWS regions (can't access keys outside the region it's created in)	• Deployed and managed in a VPC • Can be shared across VPCs (VPC Peering)
Cryptographic Acceleration	None	• SSL/TLS Acceleration • Oracle TDE Acceleration
Access & Authentication	AWS IAM	You create users and manage their permissions

Feature	AWS KMS	AWS CloudHSM
High Availability	AWS Managed Service	Add multiple HSMs over different AZs
Audit Capability	• CloudTrail • CloudWatch	• CloudTrail • CloudWatch • MFA support
Free Tier	Yes	No

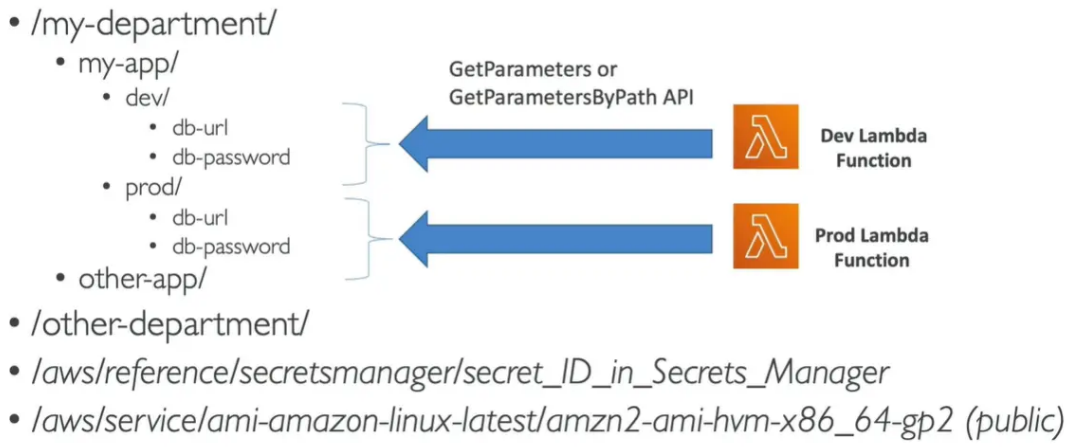
SSM Parameter Store



- secure storage for configuration and secrets
- optional seamless encryption using KMS
- serverless,scalable,durable,easy SDK
- version tracking of configurations / secrets
- security through IAM
- notification with Amazon EventBridge

- integration with CloudFormation

SSM Parameter Store Hierarchy [结构]



Standard and advanced parameter tiers

	Standard	Advanced
Total number of parameters allowed (per AWS account and Region)	10,000	100,000
Maximum size of a parameter value	4 KB	8 KB
Parameter policies available	No	Yes
Cost	No additional charge	Charges apply
Storage Pricing	Free	$0.05 per advanced parameter per month

parameters policies (for advanced parameters)

- allow to assign [指定] a TTL to a parameter (expiration date) to force updating or deleting sensitive data such as passwords
- can assign multiple policies at a time

Expiration (to delete a parameter)

```
{
    "Type":"Expiration",
    "Version":"1.0",
    "Attributes":{
        "Timestamp":"2020-12-02T21:34:33.000Z"
    }
}
```

ExpirationNotification (EventBridge)

```
{
    "Type":"ExpirationNotification",
    "Version":"1.0",
    "Attributes":{
        "Before":"15",
        "Unit":"Days"
    }
}
```

NoChangeNotification (EventBridge)

```
{
    "Type":"NoChangeNotification",
    "Version":"1.0",
    "Attributes":{
        "After":"20",
        "Unit":"Days"
    }
}
```

AWS Secrets Manager

- newer service, meant for storing secrets
- capability to force rotation of secrets every X days
- automate generation of secrets on rotation (use lambda)
- integration with Amazon RDS (MySQL,PostgreSQL,Aurora)
- secrets are encrypted using KMS
- Mostly meant for RDS integration

multi-region secrets

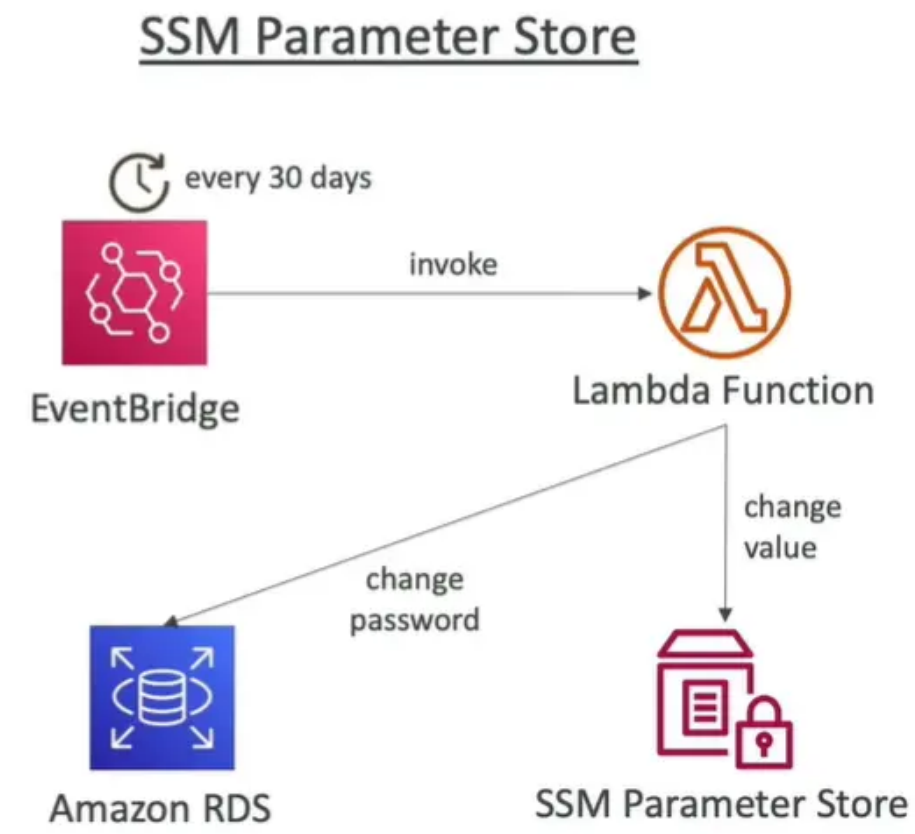
- replicate secrets across multiple AWS Regions
- Secrets Manager keeps read replicas in sync with the primary secret
- ability to promote a read replica secret to a standalone secret

- use cases: multi-region apps, disaster recovery strategies,multi-region DB..

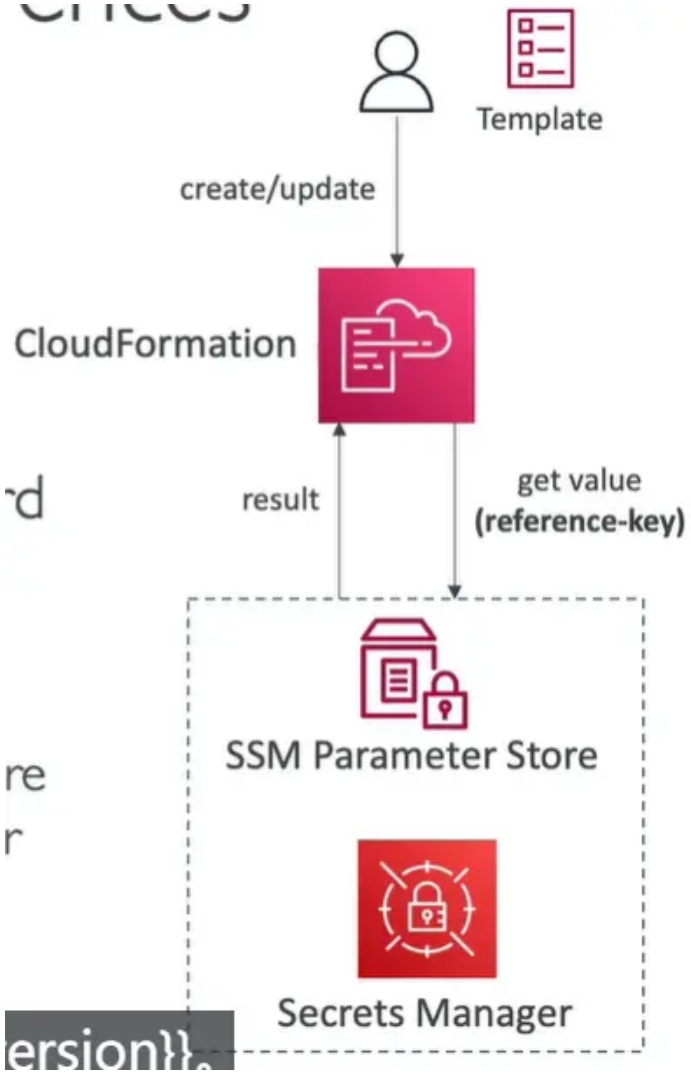
SSM Parameter Store vs Secrets Manager

- Secrets Manager ($$$)
  - automatic rotation of secrets with AWS Lambda
  - Lambda function is provided for RDS, Redshift,DocumentDB
  - KMS encryption is mandatory [强制的]
  - can integration with CloudFormation
- SSM Parameter Store ($)
  - simple API
  - no secret rotation (can enable rotation using Lambda triggered by EventBridge)
  - KMS encryption is optional
  - Can integration with CloudFormation
  - can pull a secrets manager secret using the SSM Parameter store API

SSM Parameter Store vs Secrets Manager Rotation



CloudFormation – Dynamic References



- reference external values stored in Systems Manager Parameter Store and Secrets Manager within CloudFormation templates
- CloudFormation retrieves the value of specified reference during create / update/delete operations
- for example: retrieve RDS DB instance master password from Secrets Manager
- supports
  - ssm – for plaintext valuues stored in SSM parameter store
  - ssm–secure – for secure strings stored in SSM paramater store
  - secretsmanager – for secret values stored in secrets manager

‘{{resolve:*service-name*:*reference-key*}}’

SSM

{{resolve:**ssm**:*parameter-name*:*version*}}

```
Resources:
  S3Bucket:
    Type: AWS::S3::Bucket
    Properties:
      AccessControl: '{{resolve:ssm:S3AccessControl:2}}'
```

SSM Secure

{{resolve:**ssm-secure**:*parameter-name*:*version*}}

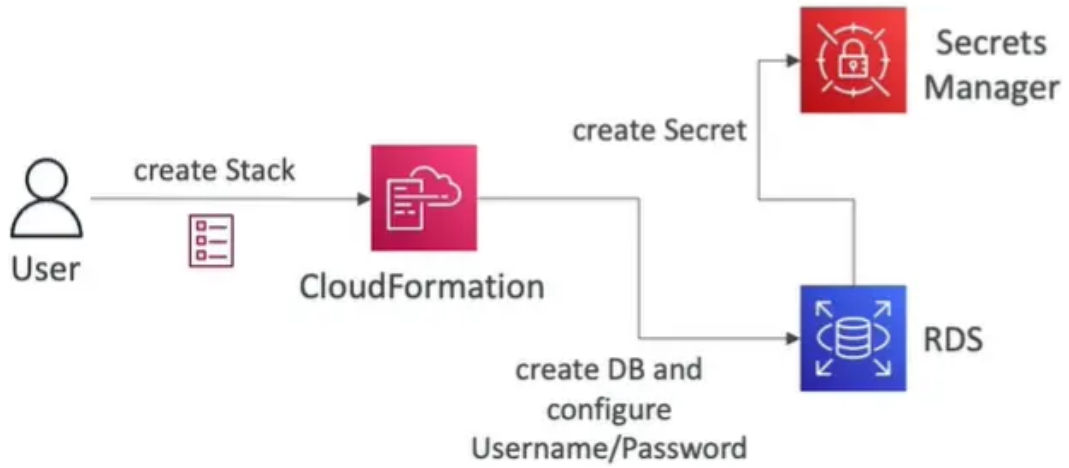
```
Resources:
  IAMUser:
    Type: AWS::IAM::User
    Properties:
      UserName: john
      LoginProfile:
        Password: '{{resolve:ssm-secure:IAMUserPassword:10}}'
```

Secrets Manager

{{resolve:**secretsmanager**:*secret-id*:*secret-string*:*json-key*:*version-stage*:*version-id*}}

```
Resources:
  DBInstance:
    Type: AWS::RDS::DBInstance
    Properties:
      DBName: MyRDSInstance
      MasterUsername: '{{resolve:secretsmanager:MyRDSSecret:SecretString:username}}'
      MasterUserPassword: '{{resolve:secretsmanager:MyRDSSecret:SecretString:password}}'
```

cloudformation , secrets manager & RDS option 1 – ManageMasterUserPassword




```yaml
Resources:
  MyCluster:
    Type: AWS::RDS::DBCluster
    Properties:
      Engine: aurora-mysql
      MasterUsername: masteruser
      ManageMasterUserPassword: true

Outputs:
  Secret:
    Value: !GetAtt MyCluster.MasterUserSecret.SecretArn
```

- ManageMasterUserPassword – create admin secret implicitly [隐含的]
- RDS，Aurora will manage the secret in Secrets Manager and its rotation

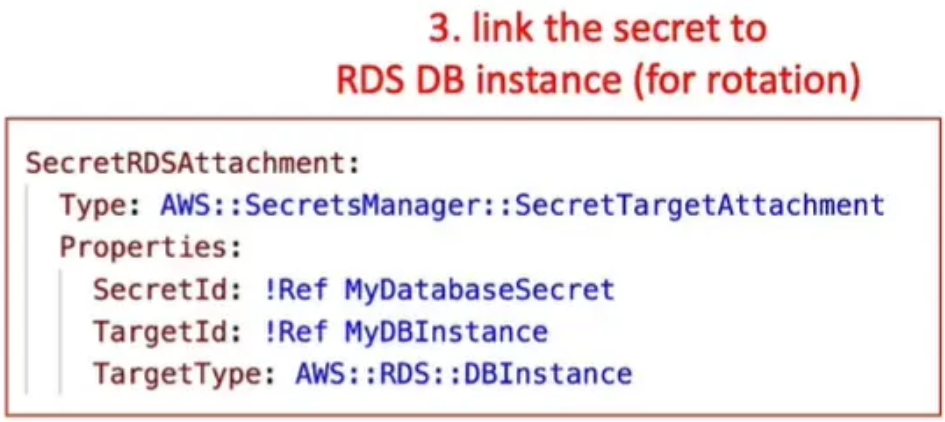
cloudformation , secrets manager & RDS option 2– Dynamic Reference

1. secret is generated

```yaml
Resources:
  MyDatabaseSecret:
    Type: AWS::SecretsManager::Secret
    Properties:
      Name: MyDatabaseSecret
      GenerateSecretString:
        SecretStringTemplate: '{"username": "admin"}'
        GenerateStringKey: "password"
        PasswordLength: 16
        ExcludeCharacters: '"@/\'
```

2. Reference secret in RDS DB instance

```yaml
MyDBInstance:
  Type: AWS::RDS::DBInstance
  Properties:
    DBName: mydatabase
    AllocatedStorage: 20
    DBInstanceClass: db.t2.micro
    Engine: mysql
    MasterUsername: '{{resolve:secretsmanager:MyDatabaseSecret:SecretString:username}}'
    MasterUserPassword: '{{resolve:secretsmanager:MyDatabaseSecret:SecretString:password}}'
```



CloudWatch Logs – Encryption

- you can encrypt CloudWatch logs with KMS keys
- Encryption is enabled at the log group level, by associating a CMK with a log group, either when you create the log group or after it exists
- you cannot associate a CMK with a log group using the CloudWatch console
- you must use the CloudWatch logs API
  - `associate-kms-key`: if the log group already exists
  - `create-log-group`: if the log group doesn't exist yet

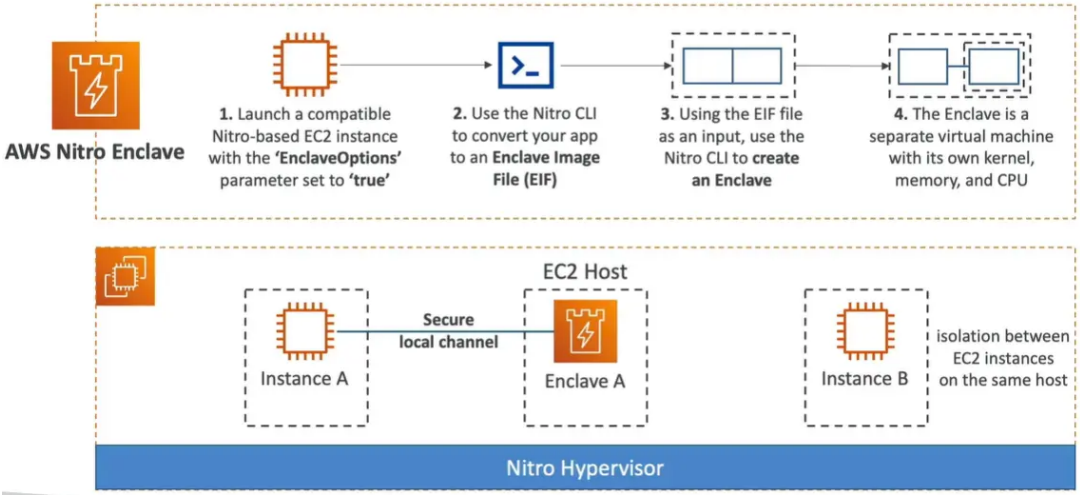
CodeBuild Security

- to access resource in your VPC, make sure you specify a VPC configuration for your CodeBuild

- secrets in CodeBuild
- don't store them as plaintext in environment variables
- instead
  - environment variables can reference parameter store parameters
  - environment variables can reference secrets manager secrets

AWS Nitro Enclaves

- process highly sensitive data in an isolated compute environment
  - personally identifiable information (PII),healthcare, financial....
- fully isolated virtual machines, hardened, and highly constrained [限制]
  - not a container, not persistent [持久] storage, no interactive [交互], no external networking
- helps reduce the attack surface for sensistive data processing apps
  - Cyptographic Attestation – only authorized code can be running in your Enclave
  - only Enclaves can access sensitive data (integration with KMS)
- Use cases: securing private keys, processing credit cards, secure multi-party computation..


AWS Nitro Enclave
1. Launch a compatible Nitro-based EC2 instance with the 'EnclaveOptions' parameter set to 'true'
2. Use the Nitro CLI to convert your app to an Enclave Image File (EIF)
3. Using the EIF file as an input, use the Nitro CLI to create an Enclave
4. The Enclave is a separate virtual machine with its own kernel, memory, and CPU

EC2 Host
Instance A
Secure local channel
Enclave A
Instance B
isolation between EC2 instances on the same host
Nitro Hypervisor