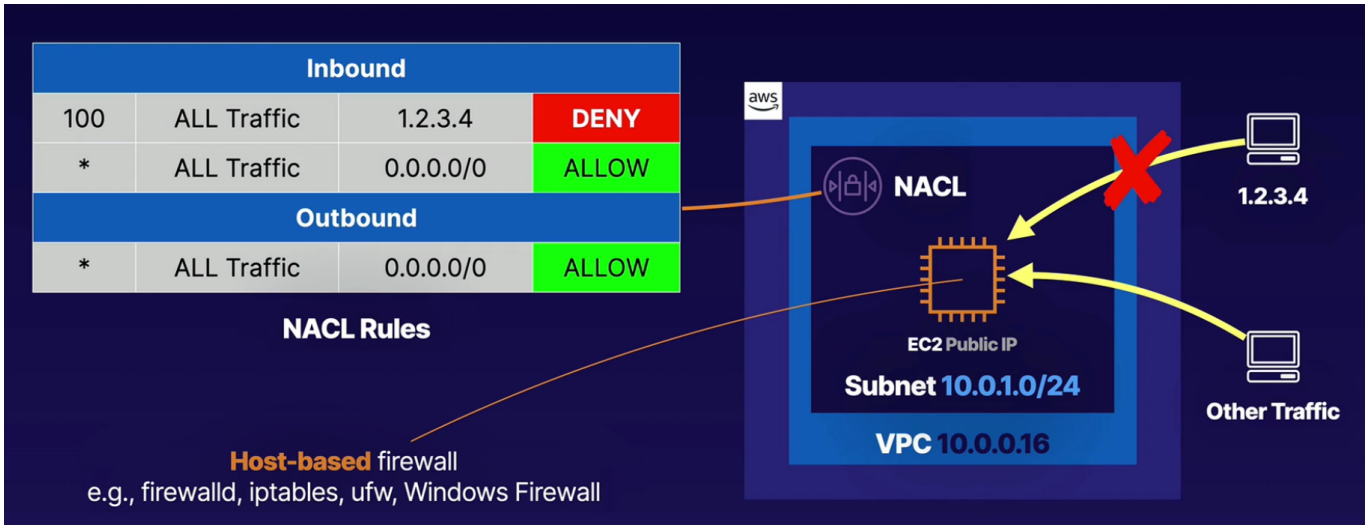
Reducing Security Threats

Bad Actors

- Typically automated processes
- Content scrapers
- Bad bots
- Fake user agent
- Denial of Service (DDOS)

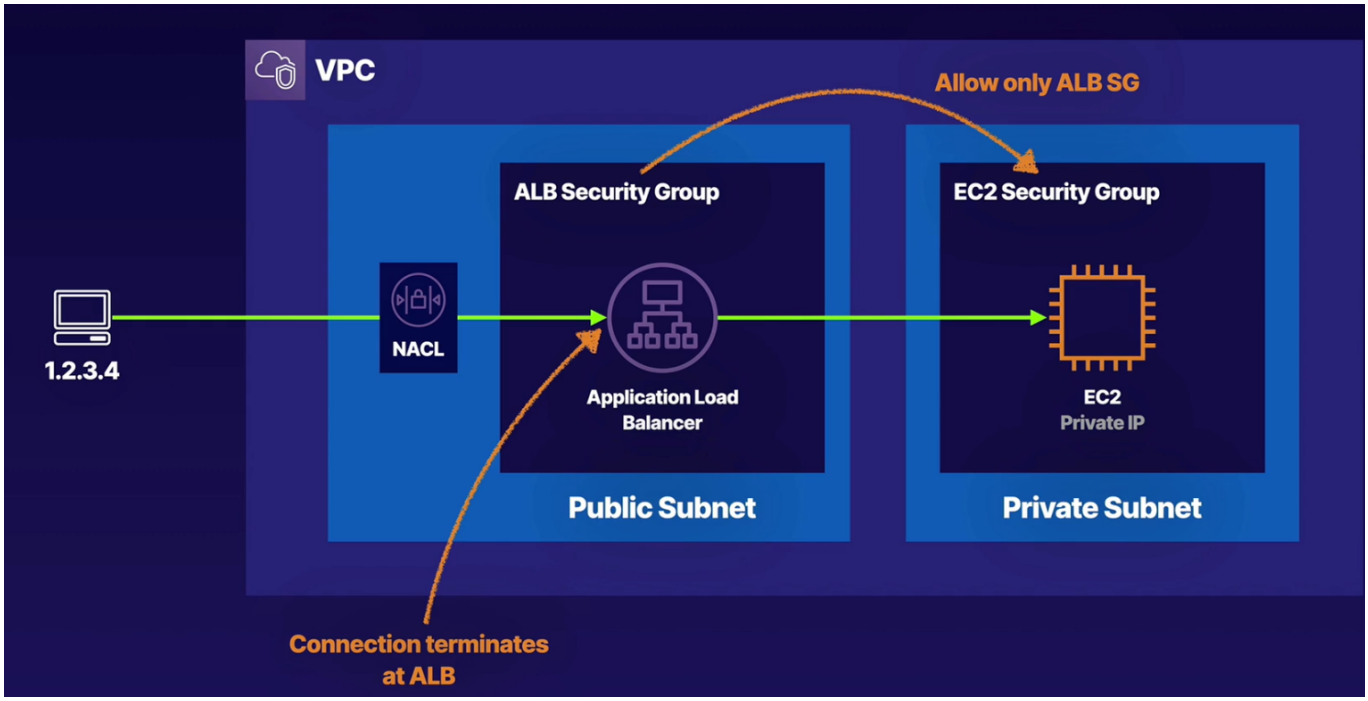
Benefits of preventing bad actors

- Reduce security threats
- lower overall costs



Network Access Control List
If we know there is a bad actor and we have their ip address; we can restrict their access to the EC2 instance using a combination of inbound and outbound rules using NACL and subnets.

We can also employ a host based firewall; these are software that can in installed and can run directly on the EC2 instance.



Application Load Balancer (ALB)

This gets more complicated when we introduce an application load balancer to the environment. With this the incoming connection from the bad actor will terminate at the ALB itself, so the EC2 instance will be compltely unaware of the origin IP.

A host based firewall will be ineffective in such cases; One additional measure you can take is to allow only the ALB security group access to the EC2 security group. This wont completely block traffic to the ALB originating from the bad actor. We will still have to use NACL.

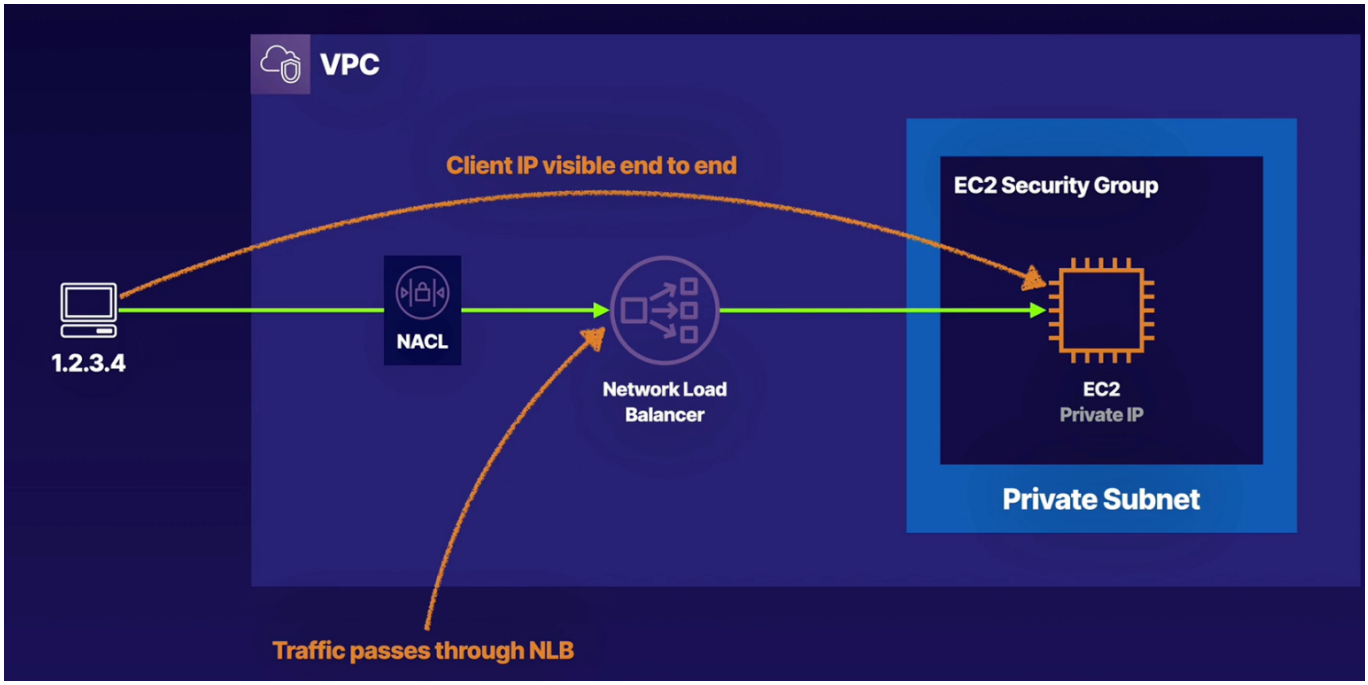
The connection from that bad actor is going to terminate at the ALB not at the EC2 instance.

There's one additional counter measure you can employ and that's to use a web application firewall attached to your load balancer.

This is what's called AWS WAF.

This is a web application firewall service that lets you monitor web requests and protect your web applications from malicious requests from bad actors, you can use WAF to block or allow requests based on conditions that you specify like the origin IP address.

You can also use WAF's pre-configured protections to block common attacks like sequel, injection attacks or cross site scripting attacks.



Network Load Balancer (NLB)
Unlike with an ALB, with an NLB, the traffic doesn't terminate at the NLB. It passes directly through it directly to your EC2 instance, the client IP, the IP address of that bad actor is visible from end-to-end. The only counter-measure you have in this case is to use a NACL to block that IP address.

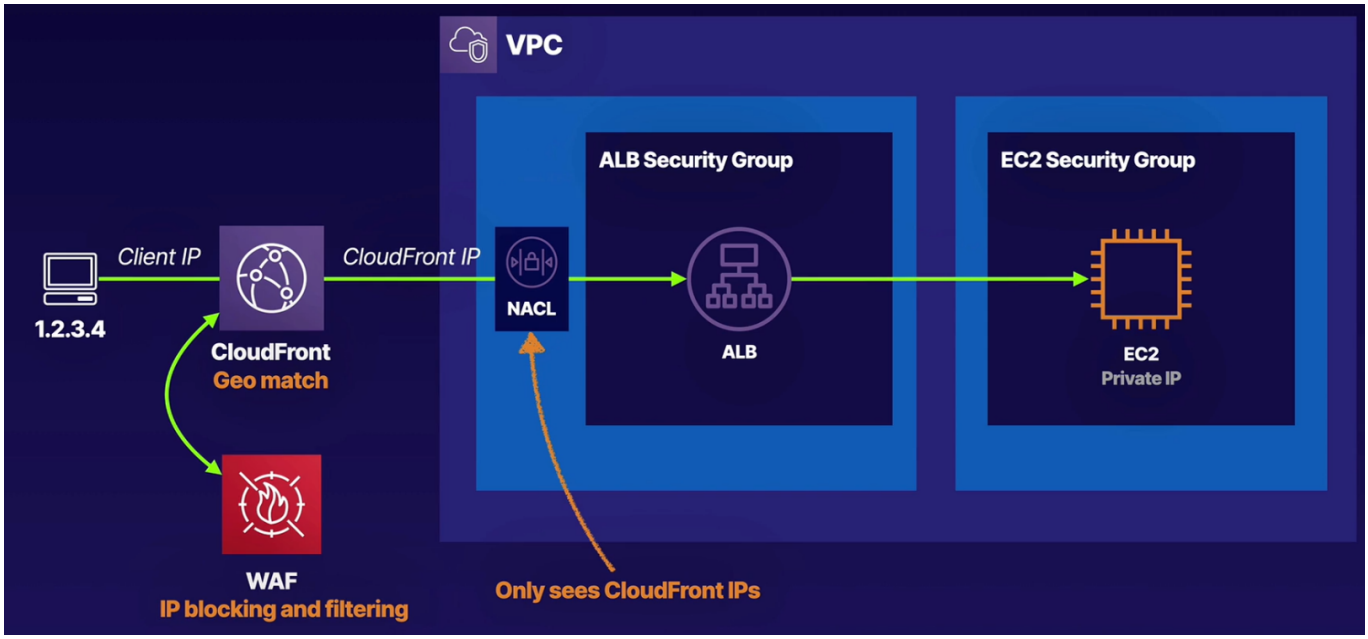
Since the Client IP is visible end to end, a firewall block on the EC2 instance would be possible, but its better to block at the NACL. A WAF rule could be used as well

WAF vs NACL

- If you want to block common exploits like SQL injection or cross-site scripting attacks, its best to use WAF
    - this operates on layer 7 and can inspect this level of traffic for these types of exploits
- if you want to block an ip or a range of ip, then you want to use a NACL which operates on layer 4

If you're operating a public web application, its better to use WAF.

WAF + CloudFront



Just like ALB, you can also attach WAF to your CloudFront distribution, and you have a couple of options here with CloudFront.

Similar to ALB, is that the client's connection terminates at CloudFront, that client IP is not visible to your NACL only the CloudFront IP is passed along to the NACL. So blocking your bad actors, IP, and a NACL when sitting behind a CloudFront distribution will be ineffective.

In these cases, you want to attach a WAF to your CloudFront distribution and use the IP blocking and filtering options. Additionally, if you find that you're getting abuse from a particular country, you can use CloudFront geo match feature to block that country's traffic altogether.

Key Management Service (KMS)

- Regional secure key management and encryption and decryption
- Manages customer master keys (CMKs)
- Ideal for S3 objects, database passwords and aPI keys stored in Systems Manager Parameter Store
- Encrypt and decrypt data up to 4KB in size
- Integrated with most AWS services
- Pay per API call
- Audit capability using CloudTrail - Logs delivered to S3
- FIPS 140-2 Level 2
- Level 3 is ClouHSM

This is the most important service in AWS if you need to perform any kind of encryption.

KMS is regional managed service that makes it easy for you to create and control the encryption keys used to encrypt your data. KMS manages what are called customer master keys or CMKs. A CMK is a logical representation of a key. It's a pointer or reference to some underlying cryptographic material. The CMKs you create exist in a region in AWS and never leave that region or KMS at all.

KMS is ideal for encrypting as three objects or even database passwords and API keys stored in systems manager parameter store. CMKs can encrypt and decrypt data up to four kilobytes in size and KMS is integrated with most AWS services. Now with KMS, you pay per API call.

So API calls like listing your keys, encrypting data decrypting, or reencrypting data you'll pay per each of these API calls and KMS supports audit capability using CloudTrail. Those audit logs are delivered to S3.

it's important to know that KMS is a FIPs 140-2 level two service FIPS is a us government computer security standard used to approve cryptographic modules.
Level two means you just have to show evidence of tampering. There is a level three, which is supported by the cloud HSM product.

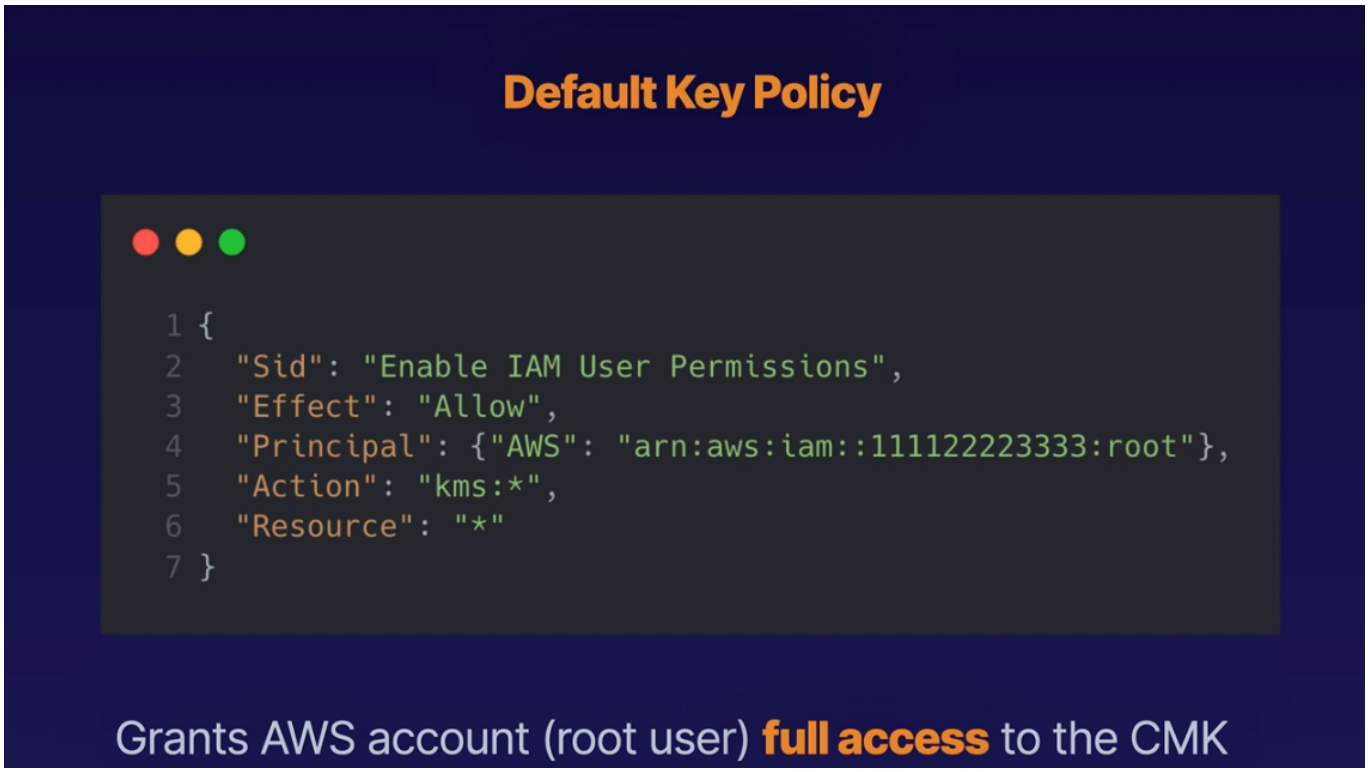
Type	Can View	Can Manage	Dedicated to My Account
Customer Managed	Yes	Yes	Yes
AWS Managed CMK	Yes	No	Yes
AWS Owned CMK	No	Yes	No

- AWS Managed CMK
  - Free; used by default if you pick encryption in most AWS services. Only that service can use them directly
- AWS Owned CMK
  - Used by AWS on a shared basis across many accounts; you typically won't see these
- Customer Managed CMK
  - Allows key rotation; controlled via key policies and can be enabled/disabled

Two types of Encryption

- Symmetric
  - Same key used for encryption and decryption
  - AES-256
  - Never leaves AWS unencrypted
  - Must call the KMS APIs to use
  - AWS services integegrated with KMS use symmetric CMKs
  - Encrypt, decrypt, and re-encrypt data
  - Generate data keys, data key pairs, and random byte strings
  - Import your own key material
- Asymmetric
  - Mathematically related public/private key pair
  - RSA and elliptic-curve cryptography (ECC)
  - Private key never leaves AWS unencrypted
  - Must call the KMS APIs to use private key
  - Download the public key and use outside AWS
  - used outside AWS by users who can't call KMS APIs
  - AWS services integrated with KMS do not support asymmetric CMKs

- Sign messages and verify signatures



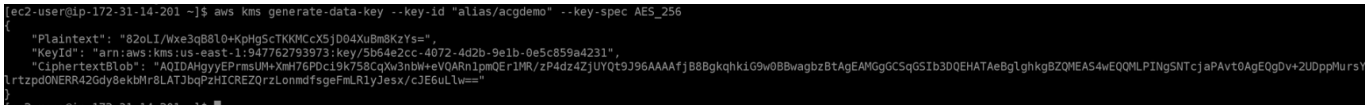
AWS Console → Key Management Service (KMS) → Customer Managed Keys → SSH into EC2 instance

```
aws kms create-key --description "CMK Demo CMK"
```


```
aws kms create-alias --target-key-id [KeyId from the results] --alias-name
"alias/acgDemo"

aws kms list-keys

# encrypt a simple text file

echo "This is file, keep it secret" > topsecret.txt

aws kms encrypt --key-id "alias/acgDemo" --plaintext file://topsecret.txt --
output text --query CiphertextBlob

# now lets decrypt that base 64 blob and save the raw binary data to a file
aws kms encrypt --key-id "alias/acgDemo" --plaintext file://topsecret.txt --
output text --query CiphertextBlob | base64 --decode > topsecret.txt.encrypted

# now we can do the reverse and decrypt the binary
aws kms decrypt --cipertext-blob fileb::://topsecret.txt.encrypted --output text
--query Plaintext

# now we can decrypt the base 64 cipher to plain text
aws kms decrypt --ciphertext-blob fileb://topsecret.txt.encrypted --output text
--query Plaintext | base 64 --decode
```

CMKs can encrypt files up to 4KB in size, if you want to encrypt a file larger than this, you can use a DEK, Data encryption key.

```
aws kms generate-data-key --keyid "alias/acgDemo" --key-spec AWS_256
```



This returns a plaintext data key, and also an encrypted with the specified CMK version of the data key. The encrypted version is referred to as a cipher text blob.

You want to store the returned cipher text blob, this has meta data which tells KMS which CMK was used to generate it and you can use the plain text data key to encrypt any amount of data. So there is no 4KB limit size.

but you should throw away the plain text data key and be sure to store the cipher text block along with the encrypted data.

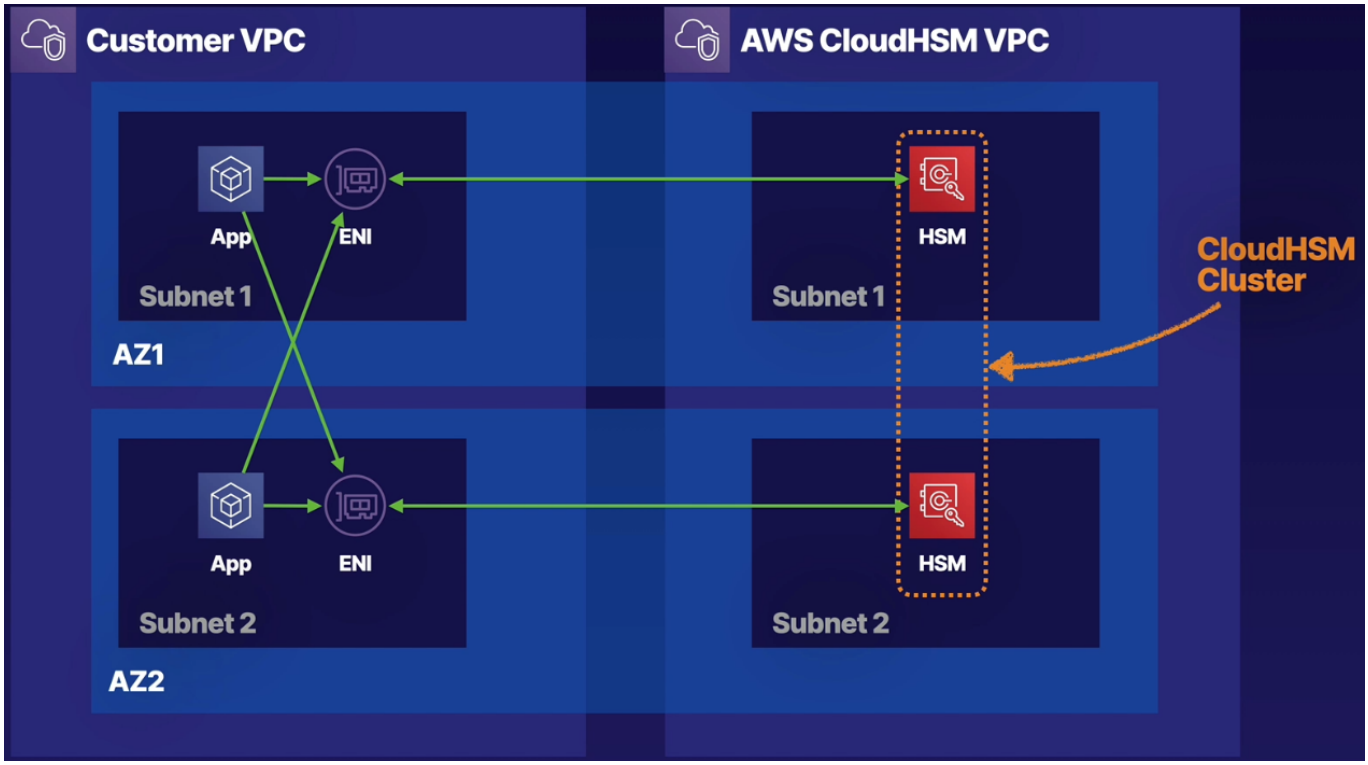
This way the encrypted data won't be compromised as long as the plain text data key is destroyed after each encryption or decryption.

At best someone can get to the cipher text blob, but unless they are able to call KMS to decrypt it, the encrypted data cannot be decrypted with just the cipher text blob. This is envelope encryption. Envelope encryption also reduces network load.

CloudHSM

Hardware Security Modeules (HSM).

- Dedicated hardware security module (HSM)
- FIPS 140-2 Level 3
- Level 2 is KMS
- Manage your own keys with cloudHSM
- No access to the AWS-managed component
- Runs within a VPC in your account
- Single tenant, dedicated hardware, multi-AZ cluster
- Industry-standard APIs - no AWS APIs
- PKCS#11
- Java Cryptography Extensions (JCE)
- Microsoft CryptoNG (CNG)
- Keep your keys safe -- irretrievable if lost



- CloudHSM will operate inside its own VPC dedicated to CloudHSM from a security isolation standpoint
- it will then project ENIs or elastic network interfaces into the VPC of your choosing
- This is how your applications communicate with the CloudHSM cluster inside the cluster you'll create specific instances of HSMs
- CloudHSM is not highly available by default. they'll need to explicitly provision HSMs across availability zones.
- If any HSM fails, or any AZ becomes unavailable; you'll still ahve the other HSM instances.
- Ideally you'll want to place one HSM per subnet in each availablility zone with a minimum of two AZs like you see here in the diagram, which is what's recommened by AWS

Exam Tips

- Regulatory compliance requirements
- FIPS 140-2 level 3

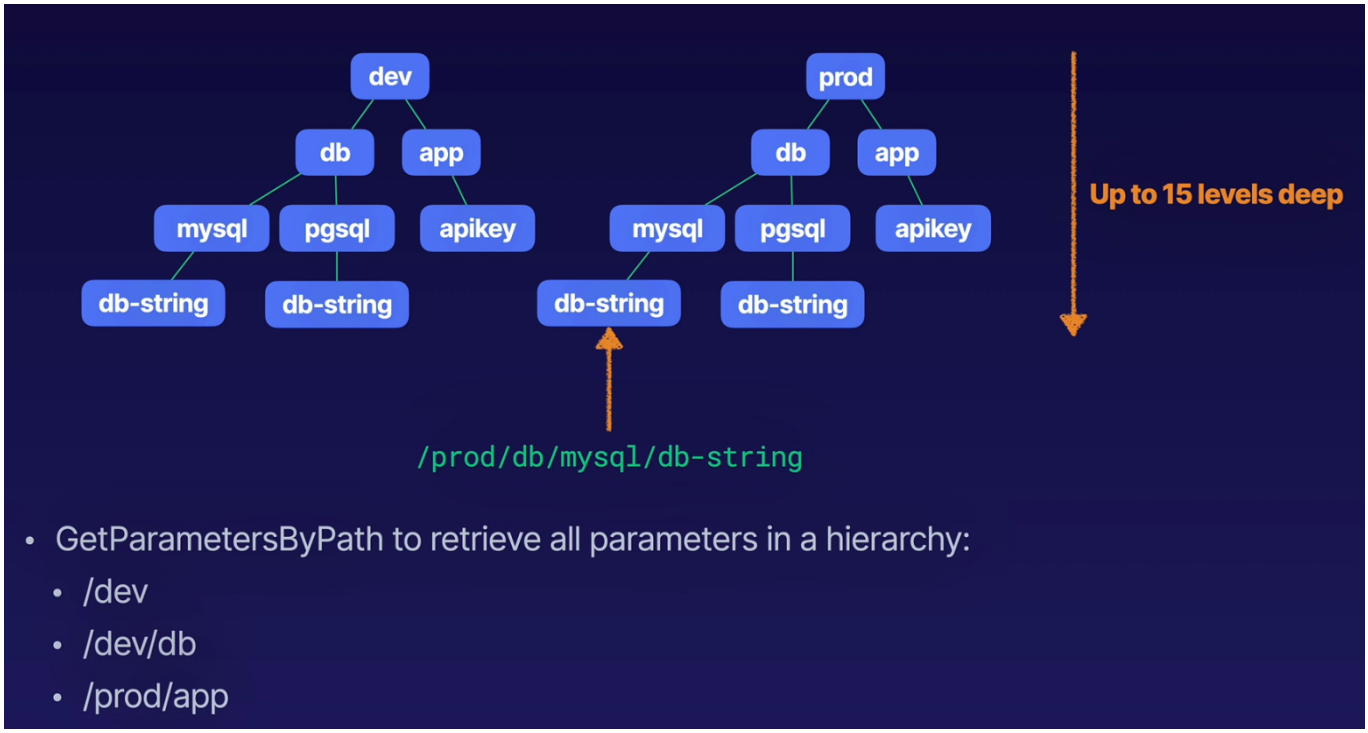
Answer is CloudHSM

Systems Manager Parameter Store

This feature addresses a pretty critical need, securely managing configuration and secrets within AWS. Parameter store is an essential tool for caching and distributing secrets securely to AWS resources.

- component of AWS Systems Manager (SSM)
- their stateless, and designed to be swaped out
- Secure serverless storage for configuration and secrets
    - Passwords
    - Database connection strings
    - License codes
    - API keys
- Values can be stored encrypted (KMS) or plaintext
- Seprarte data from source control
- Store parameters in hierarchies
- Track versions
- Set TTL to expire values such as passwords

Data can be stored hierarchicaly, which means that you can build tree-like structures. At each level of this tree we can retrieve data based on that level. Because its hierarchical we can grant permissions at any point in the tree.



```
Parameters:
  LatestAmiId:
    Type: 'AWS::SSM::Parameter::Value<AWS::EC2::Image::Id>'
    Default: '/aws/service/ami-amazon-linux-latest/amzn2-ami-hvm-x86_64-gp2'

Resources:
 Instance:
    Type: 'AWS::EC2::Instance'
    Properties:
      ImageId: !Ref LatestAmiId
```

Create a lambda function that can access the parameter store

- first create an execution role in IAM, this will define what permissions that lambda function has
- IAM → Policies → Create → JSON

```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Effect": "Allow",
    "Action": [
      "logs:CreateLogGroup",
      "logs:CreateLogStream",
      "logs:PutLogEvents",
      "ssm:GetParameter*",
      "ssm:GetParametersByPath"
    ],
    "Resource": "*"
  }]
}
```

Review → name it → Create

- IAM → Roles → Create → AWS Servce, Lambda → Permissions → search for the policy we created → Review → name it → Create

AWS Console → Lambda → Create → name it, Python, use existing role → Create

```
import json
import os
import boto3

client = boto3.client("ssm")
env = os.environ["ENV"]
app_config_path = os.environ["APP_CONFIG_PATH"]
full_config_path = "/" + env + "/" + app_config_path


def lambda_handler(event, context):

    print("Config Path: " + full_config_path)

    param_details = client.get_parameters_by_path(
        Path=full_config_path, Recursive=True, WithDecryption=True
    )

    print(json.dumps(param_details, default=str))
```

Environment Variables → Edit, add → ENV: prod, APP_CONFIG_PATH: acg → Save

Before we switch to parameter store to create those values, we need to go to KMS to create a customer master key that we can use to encrypt those vlaues.

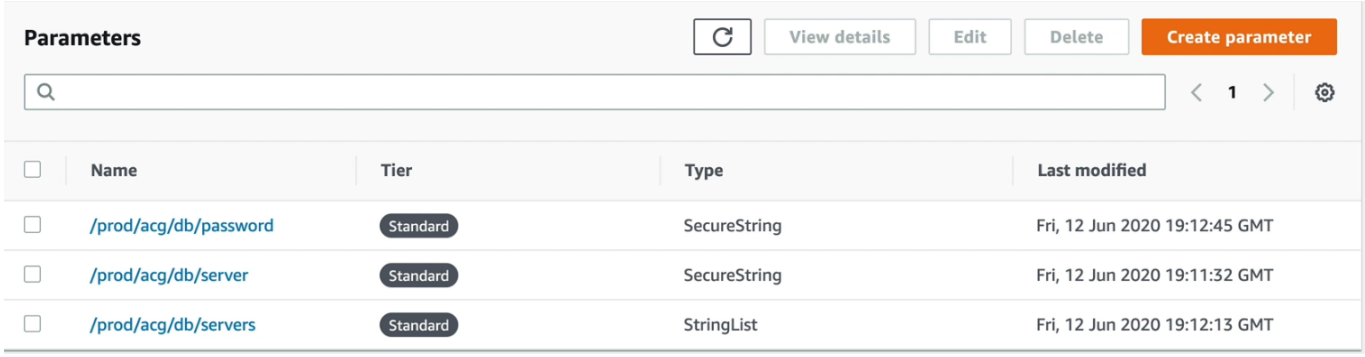
AWS Console → KMS → Create → Symmetric → name it, Next → Key Administrators, select user >> Next, select the lambda role created in IAM → Review → FInish

Now we can go to parameter store and start creating some values

AWS → SSM → Parameter Store → Create → name it; make sure its in a hierarchy fasion `prod/acg/db/server` for example, Standard, SecureString, My current account, use the KMS created. value: anything → Create parameter

Now create a string list → same as above, but select StringList, with a different value → Create

Now create another one, select SecureString → Create



Now we can test the lambda functions

AWS → Lambda → select the function → Test → name the event, Create → Test

Secrets Manager

This is a service that helps you rotate, manage, and retrieve all kinds of secrets, like database credentials, API keys, and so on. Using secrets manager, you can secure audit and manage secrets to access resources in AWS on third party services and on premises.

This is a lot like parameter store.

- Parameter store comes with no additional charges
- Theres a limit on the number of parameters you can store, 10,000 API calls
- Secrets comes with an additional cost, pennies, but it can add up for large organizations

where secrets manager wins over parameter store is the ability to automatically rotate secrets

- out of the box secrets manager provides full key rotation integration with RDS
- Apply the new key/password in RDS for you
- Generate random secrets; random passwords, etc
- Also be shared with other accounts

PArameter store you can store secrets and encrypt them but you can also store un-encrypted data and it's all free. Secrets manager takes things several steps further

AWS Shield

This service can help protect against denial of service attacks.

- Protects against distributed denial of service (DDOS) attacks
- AWS Shield Standard

    - Automatically enabled for all customers at no cost
    - Protects against common layer 3 and 4 attacks
        - SYN/UDP floods
        - Reflection attacks
        - Stopped a 2.3 Tbps DDoS attack for three days in Febraury 2020
- AWS Shield Advanced
    - $3000 per month, per organization
    - Enhanced protection for EC2, ELB, CloudFront, Global Accelerator, Route 53
    - Business and Enterprise support customers get 24×7 access to the DDoS Response Team (DRT)
    - DDoS cost protection

Web Application Firewall (WAF)

Web application firewall that lets you monitor HTTP(S) requests to CloudFront, ALB, or API Gateway

- Control access to content
- Configure filtering rules to allow/deny traffic
    - IP addresses
    - Query string parameters
        - http://example.com/page?foo=bar
    - SQL query injection

WAF Allows three different behaviors

- Allow all requests, except the ones you specify
- Block all requests, except the ones you specify
- Count the requests that match the properties you specify
- Request properties:
    - Originating IP address
    - Originating country
    - Request size
    - Values in request headers
    - Strings in request matching regular expressions (regex) patterns
    - SQL code (injection)
    - Cross-site scripting (XSS)

AWS Firewall Manager

Centrally configure and manage firewall rules across an AWS Organization

- WAF rules:
    - ALB
    - API Gateway
    - CloudFront distributions
- AWS Shield Advanced protections
    - ALB
    - ELB Classic
    - EIP
    - CloudFront Distributions
- Enable security groups for EC2 and ENIs