AWS Identity & Access Management (IAM)

Chetan Agrawal, Solutions Architect
Deven Suri, Account Manager

Agenda

Overview of Security in AWS

What is IAM – Authentication & Authorization

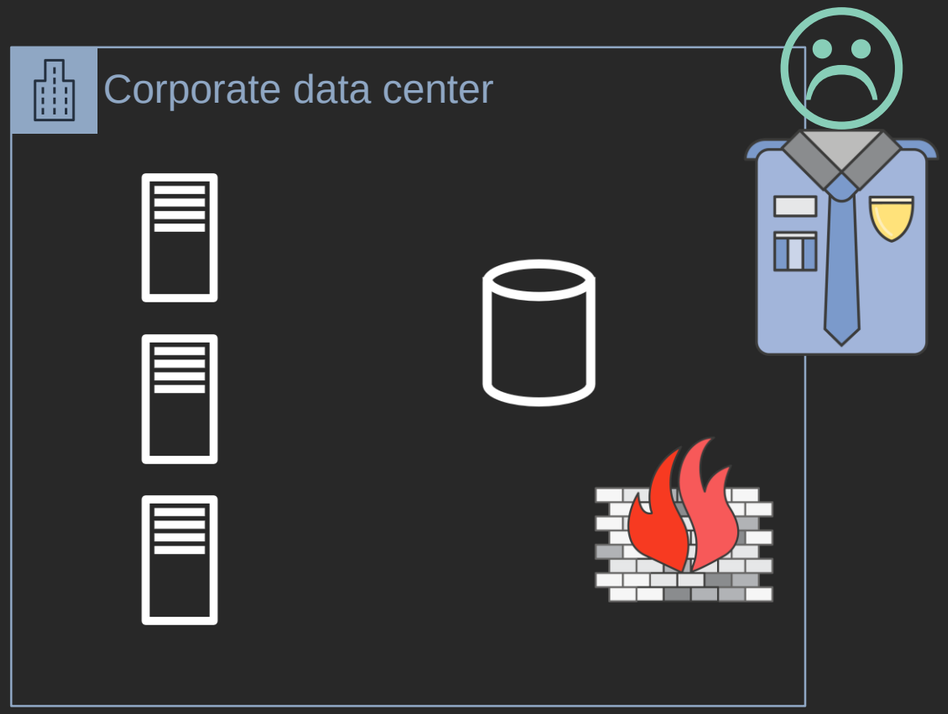
IAM components – Users, Group, Policy, Roles

Multi-account IAM access

IAM examples & best practices

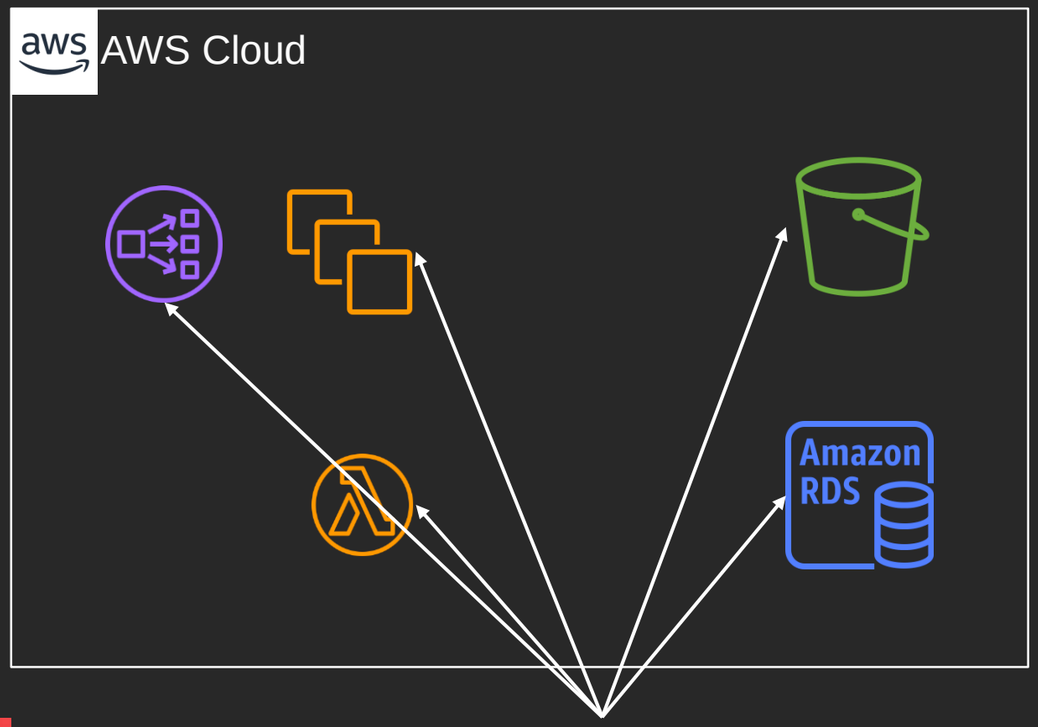
Why learn AWS Identity and Access Management (IAM)

Security before the cloud



Security implemented at perimeter

Security in the cloud



IAM authorization at every resource:

Pervasive security that’s part of your applications

AWS security, identity, and compliance solutions

Identity & access management

AWS Identity & Access Management (IAM)

AWS Single Sign-On

AWS Organizations

AWS Directory Service

Amazon Cognito

AWS Resource Access Manager

Detection

AWS Security Hub

Amazon GuardDuty

Amazon Inspector

Amazon CloudWatch

AWS Config

AWS CloudTrail

VPC Flow Logs

Infrastructure protection

AWS Firewall Manager

AWS Shield

AWS WAF – Web application firewall

Amazon Virtual Private Cloud (VPC)

AWS PrivateLink

AWS Systems Manager

Data protection

Amazon Macie

AWS Key Management Service (KMS)

AWS CloudHSM

AWS Certificate Manager

AWS Secrets Manager

AWS VPN

Server-Side Encryption

Incident response

Amazon Detective

CloudEndure DR

AWS Config Rules

AWS Lambda

AWS Foundational and Layered Security Services


AWS Security Hub

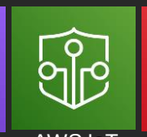
AWS Organizations

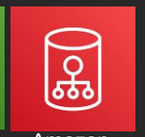

AWS Control Tower

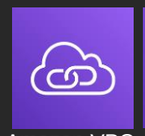

AWS Trusted Advisor

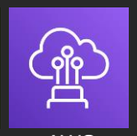

AWS Transit Gateway


Amazon VPC


AWS IoT Device Defender

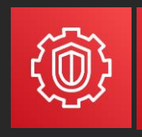

Amazon Cloud Directory

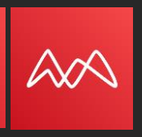

Amazon VPC PrivateLink


AWS Direct Connect


Resource Access manager


AWS Directory Service

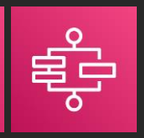

Amazon GuardDuty

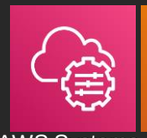

Amazon Macie


Amazon Inspector


Security Hub


Amazon CloudWatch


AWS Step Functions


AWS Systems Manager


AWS Lambda


AWS OpsWorks


AWS CloudFormation

Identify

Protect

Detect

Respond

Recover

YOU ARE HERE

Automate

Investigate


AWS Service Catalog


AWS Config


AWS Well-Architected Tool

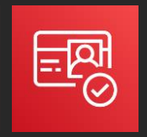

AWS Systems Manager


AWS Shield


IAM


Secrets Manager


KMS

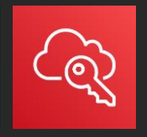

Amazon Cognito


AWS WAF

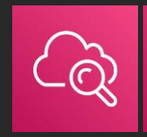

AWS Firewall Manager

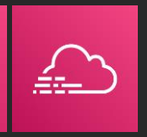

AWS Certificate Manager


AWS CloudHSM


AWS Single Sign-On


Amazon Detective


Amazon CloudWatch


AWS CloudTrail


Personal Health Dashboard

Amazon Route 53

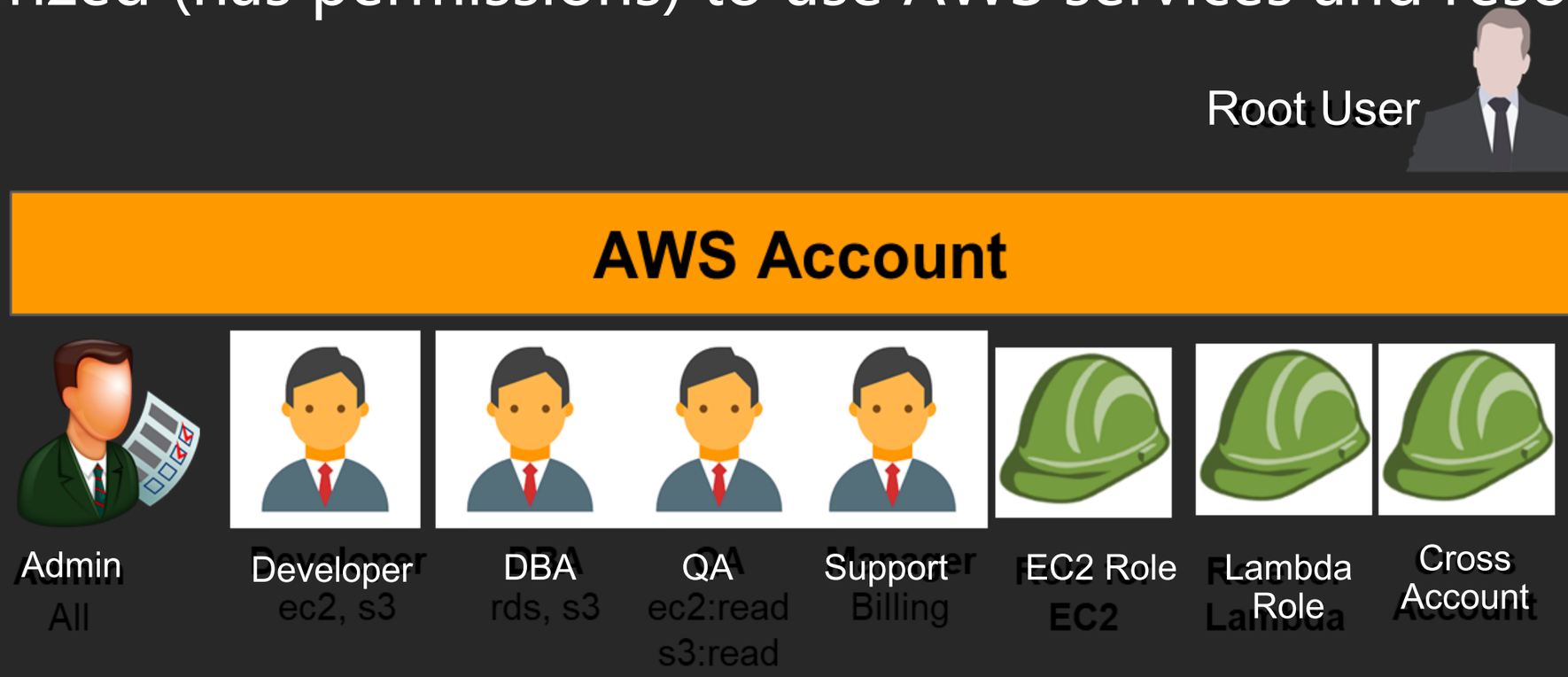
Amazon S3 Glacier

Snapshot

Archive

IAM

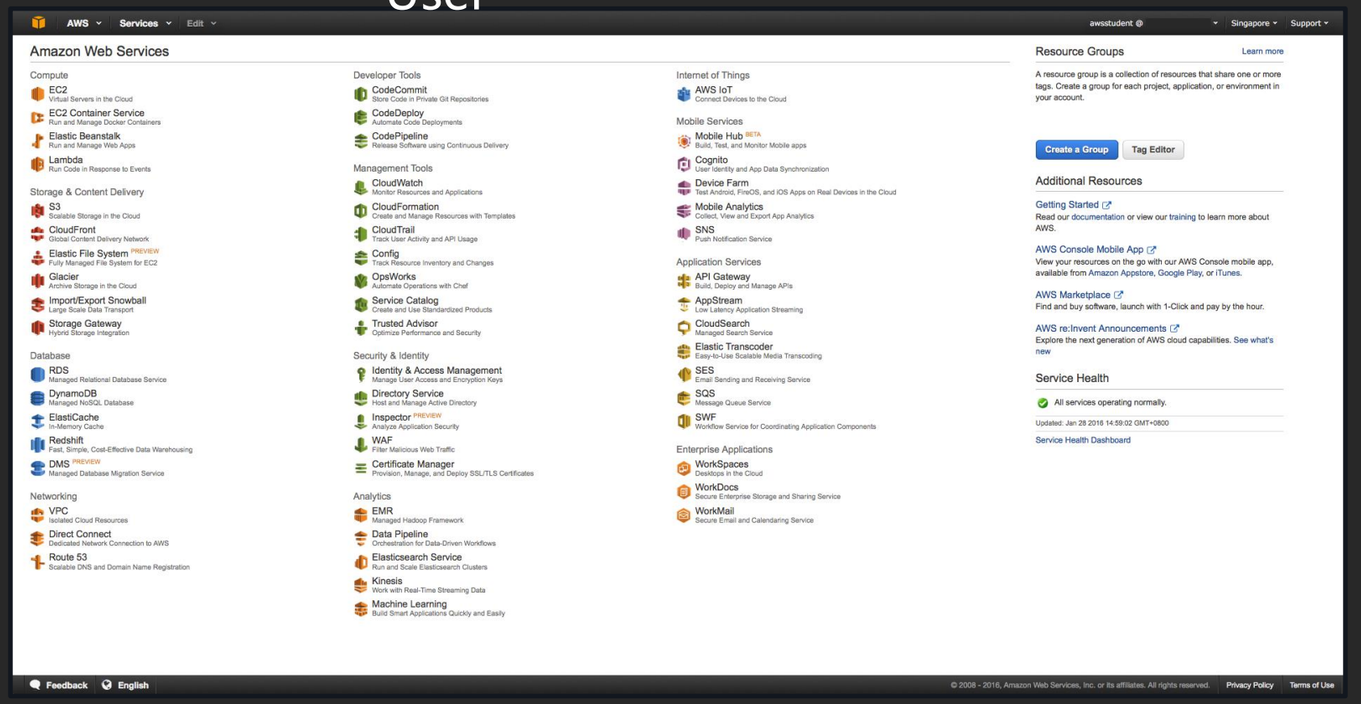
- Allows you to securely control access to AWS Services and Resources
- You use IAM to control who is authenticated (signed in) and authorized (has permissions) to use AWS services and resources.



IAM Authentication

AWS Console Access

IAM
User



AWS IAM Authentication

- AWS CLI or SDK API

  Access Key and Secret Key

IAM User

Access Key ID: AKIAIOSXXXXXXXXXDORIK
Secret Access Key: wJalrXUtnFEXXXXXXXXXXXXXXXXXXXXNIFLEKEY

AWS CLI

```
                :~              $ aws configure
AWS Access Key ID [****************O22A]:
AWS Secret Access Key [****************4m8i]:
Default region name [ap-southeast-1]:
Default output format [json]:
```

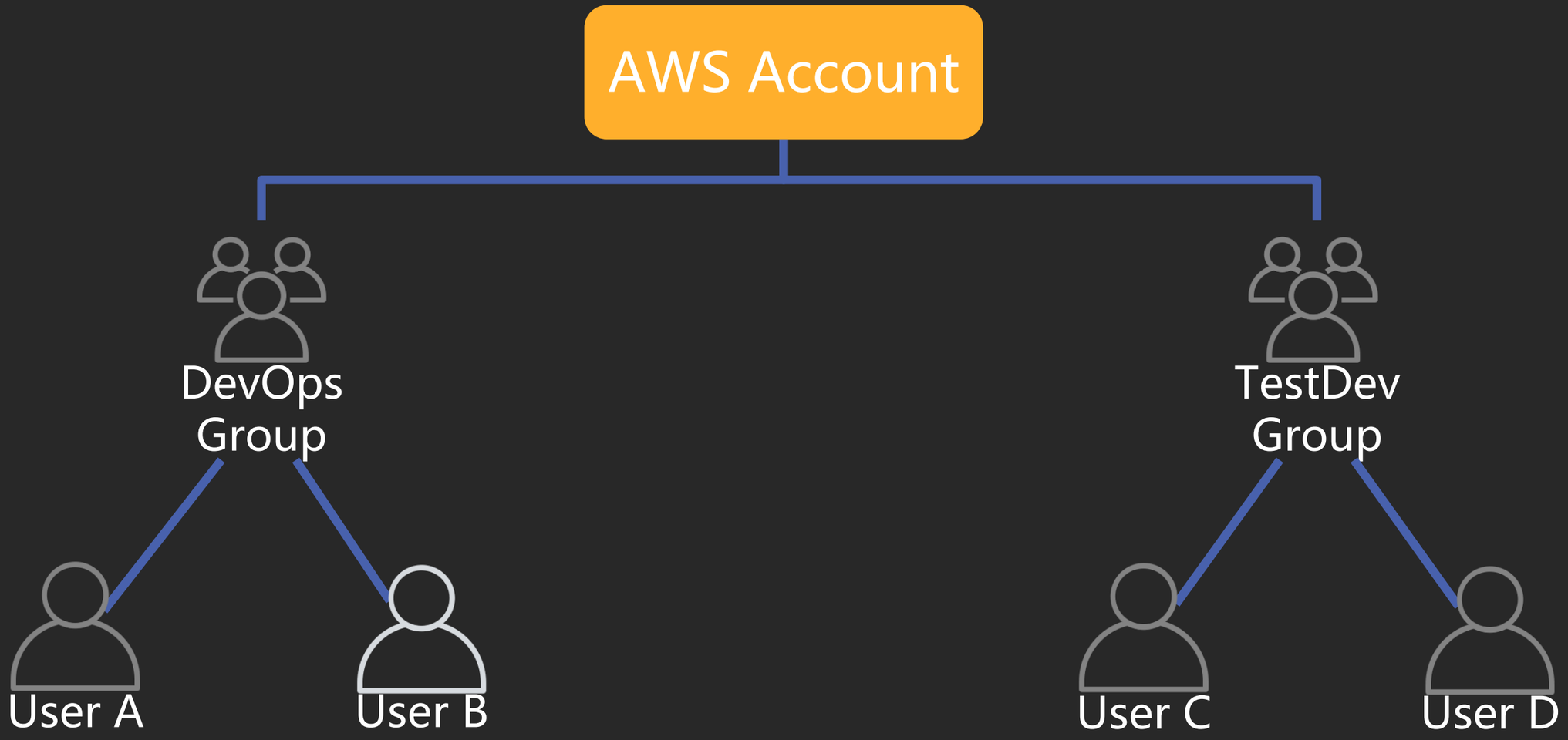
AWS SDK & API

Java

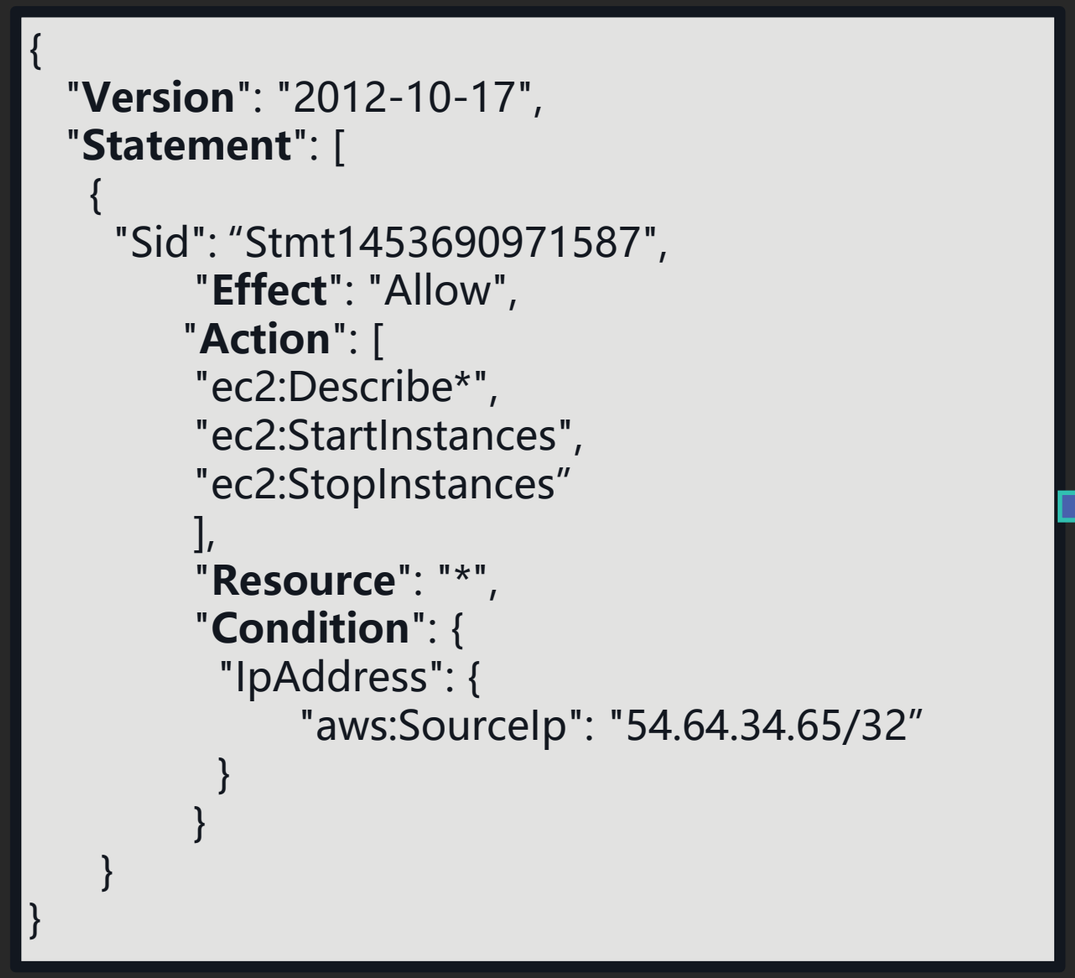
Python

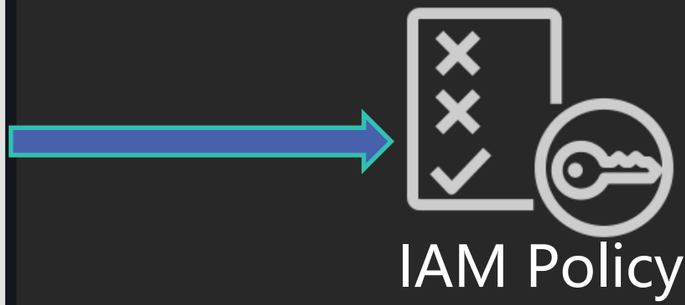
.NET

IAM Users and Groups

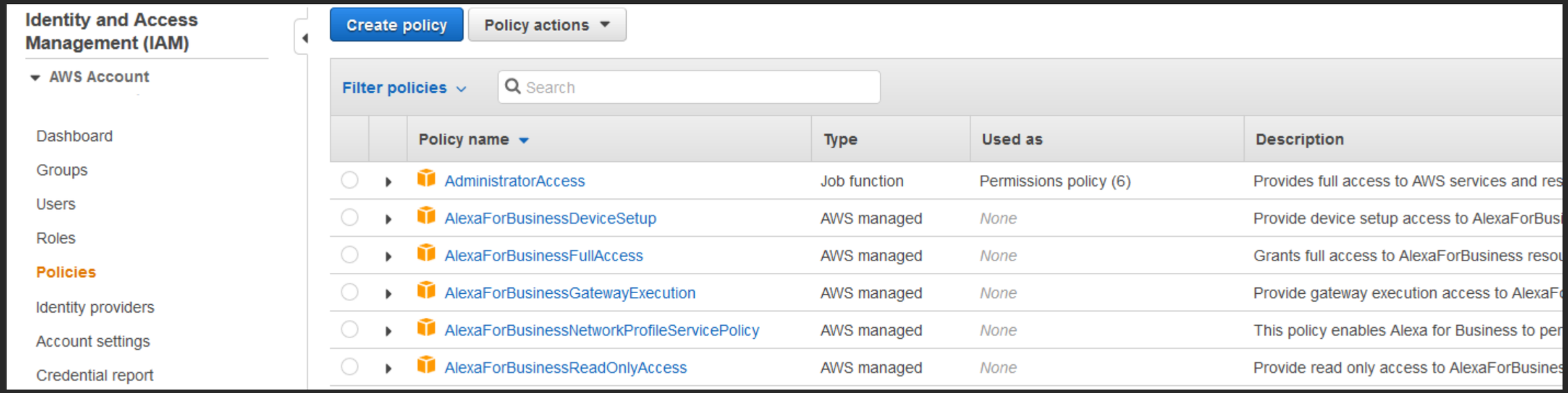


IAM Authorization – IAM Policy



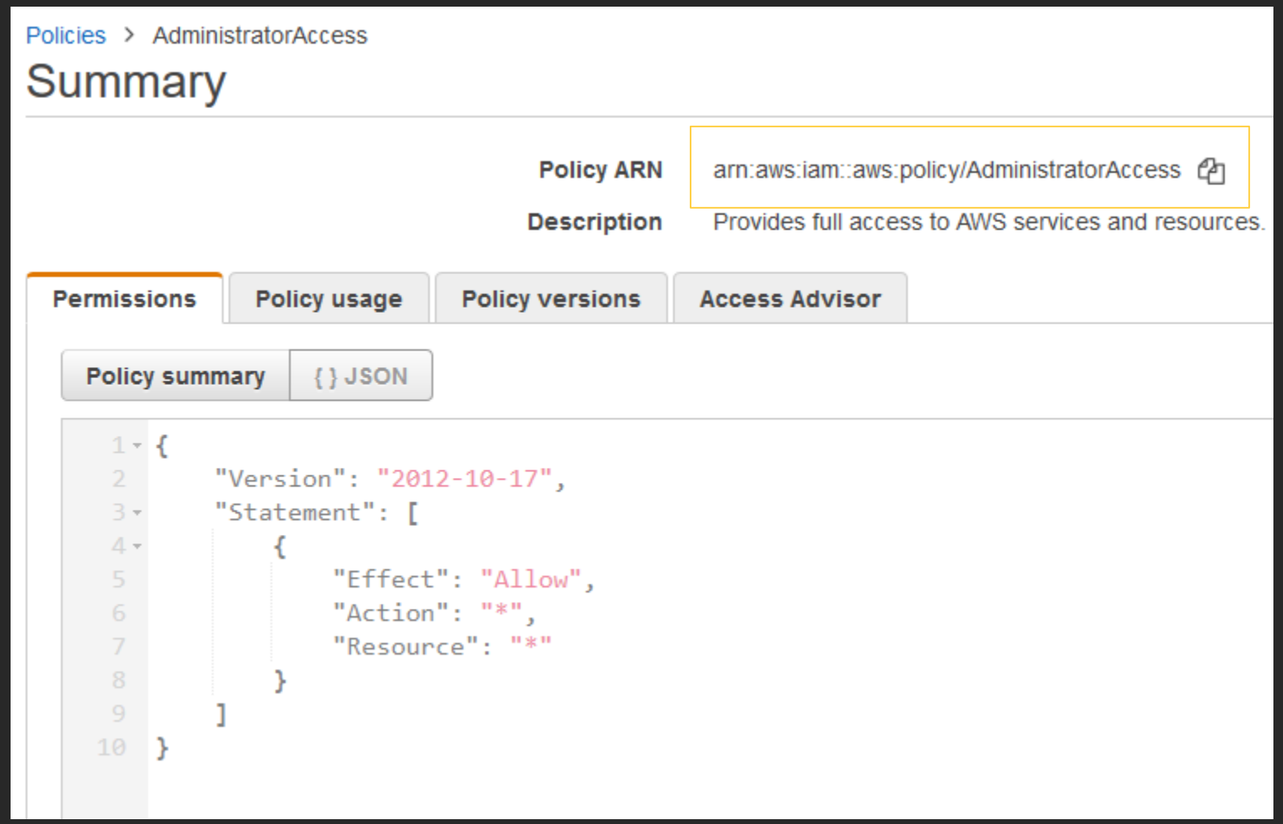


Assigning AWS managed policies

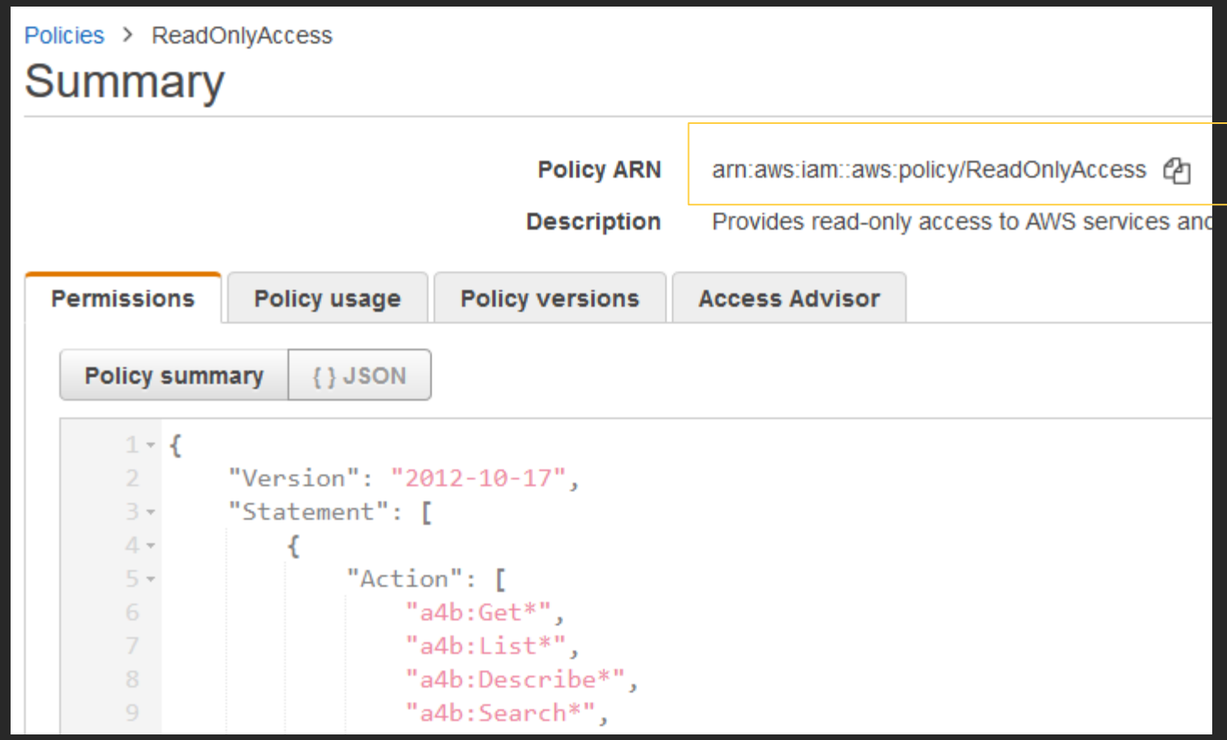


. . . and many other AWS-written policies

Example: Administrator policy



Example: Read-only policy

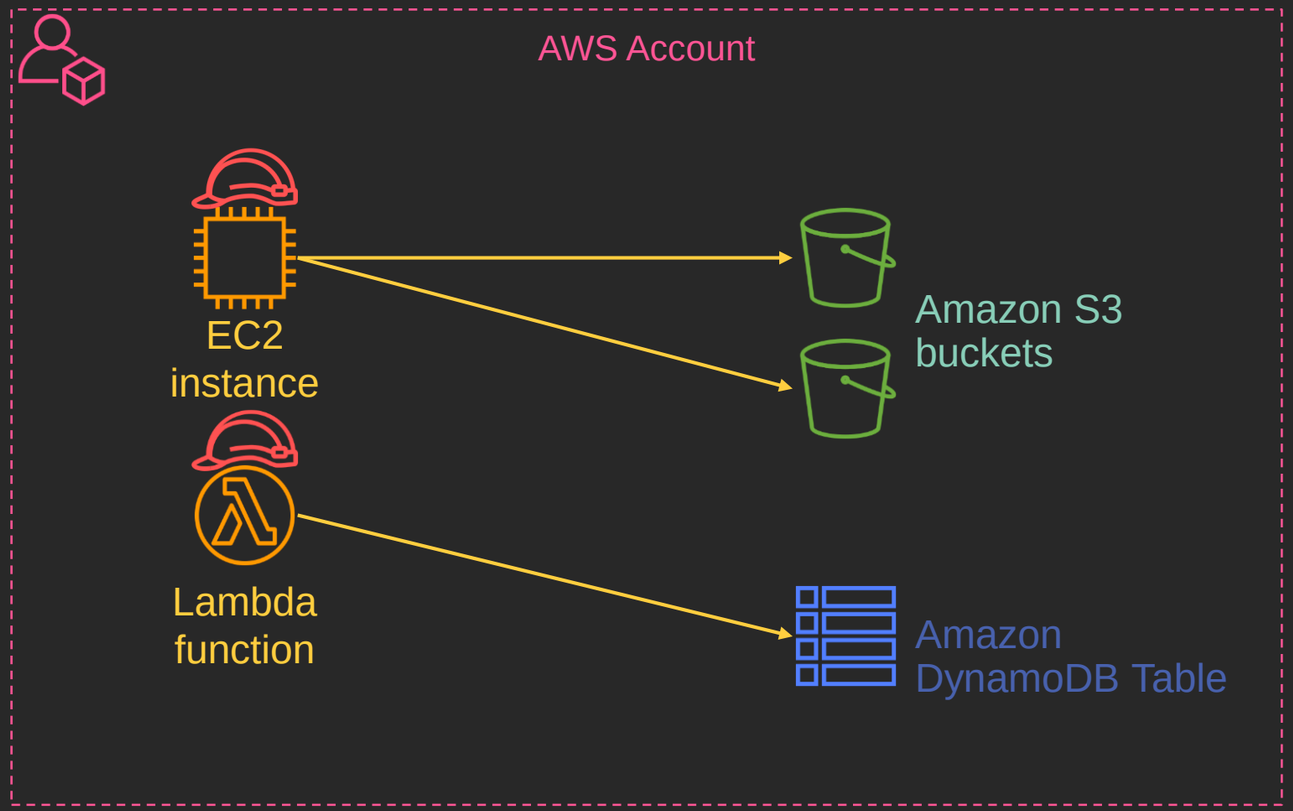


. . . and many other ReadOnly APIs

Let’s create new IAM User and assign some permissions..

- Create new IAM User
- Grant AWS Console Access
- Grant EC2 Read only Access
- Verify the Access
- Now add EC2 Full Access permissions
- Check the effect

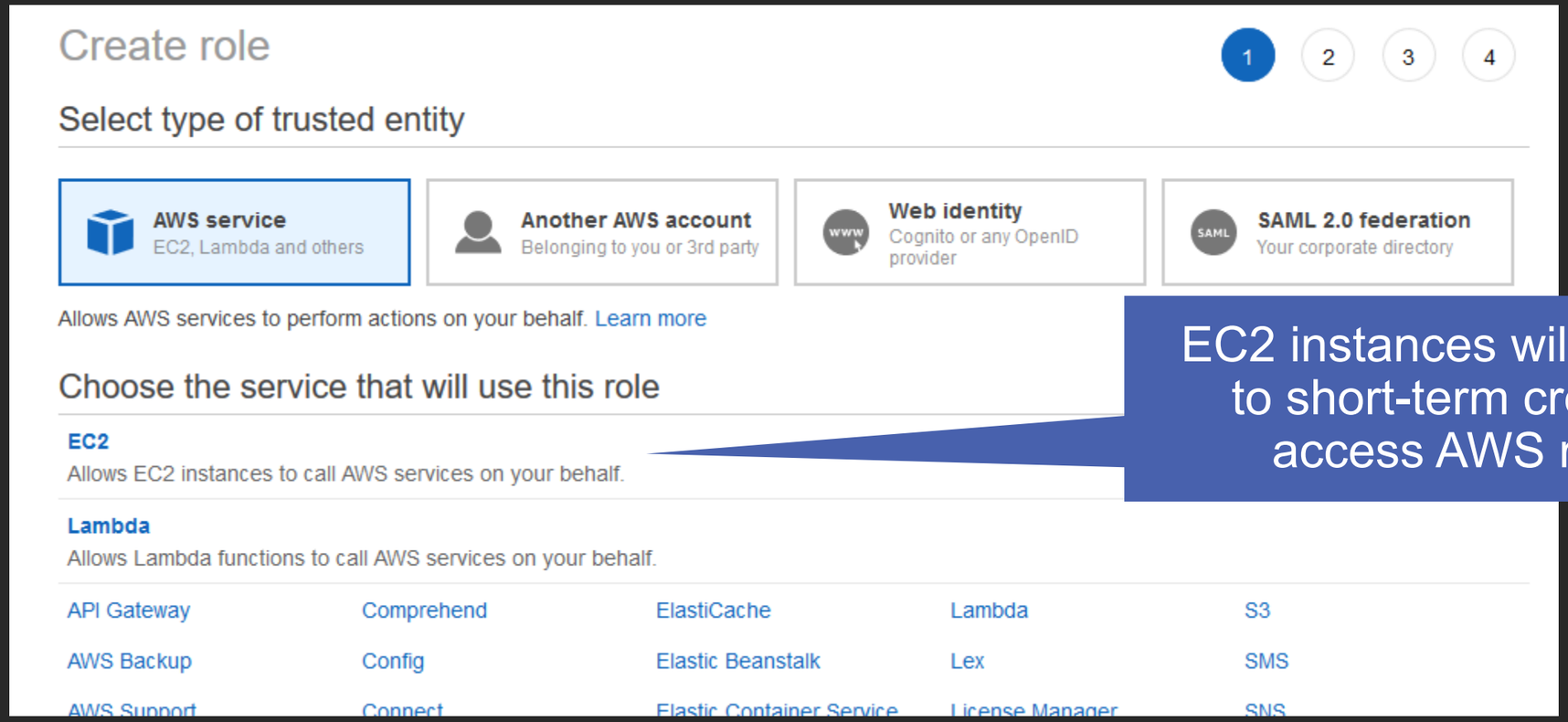
IAM roles for non-human access



Use IAM roles for access to AWS resources from:

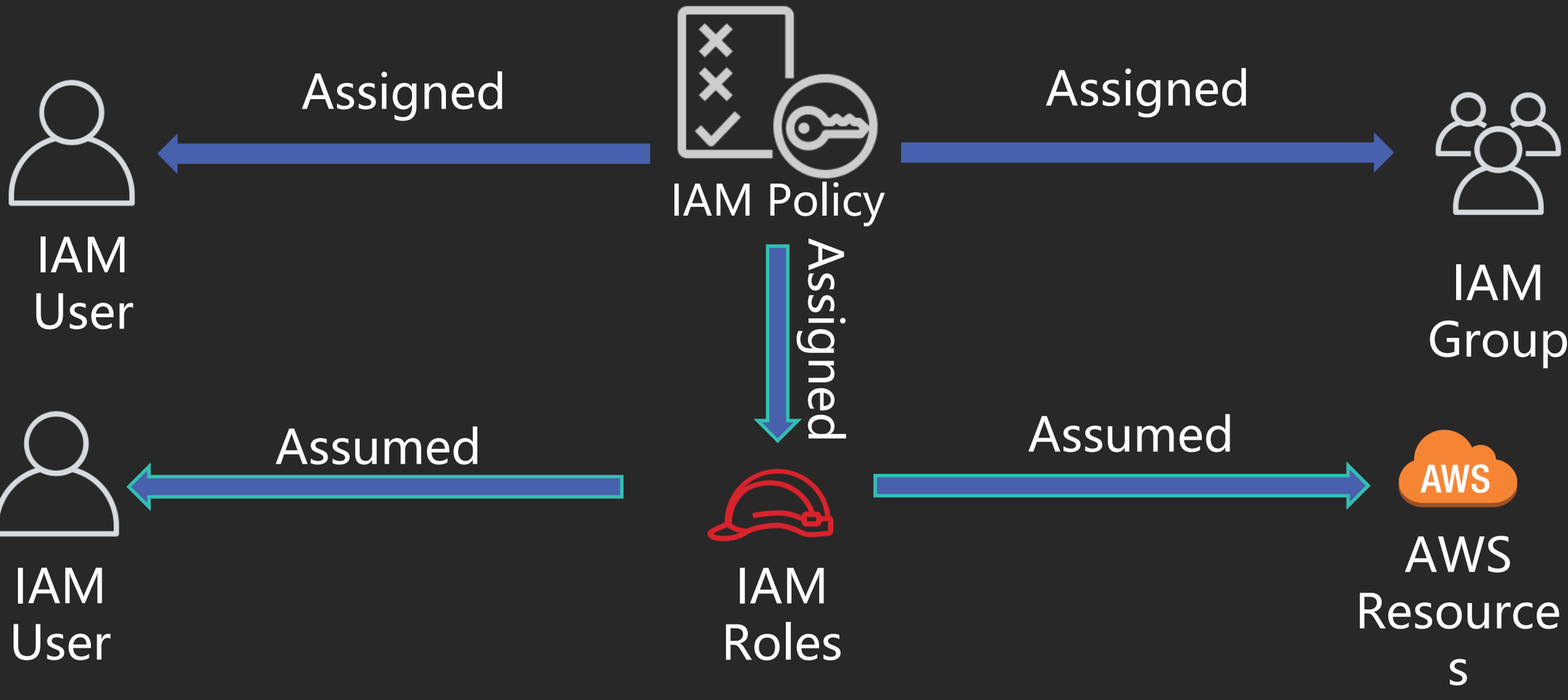
- Your application running on an AWS compute environment, e.g., EC2 instance, Lambda function, etc.
- Permission to an AWS service to access your resources (not shown)

Creating IAM roles for non-human access

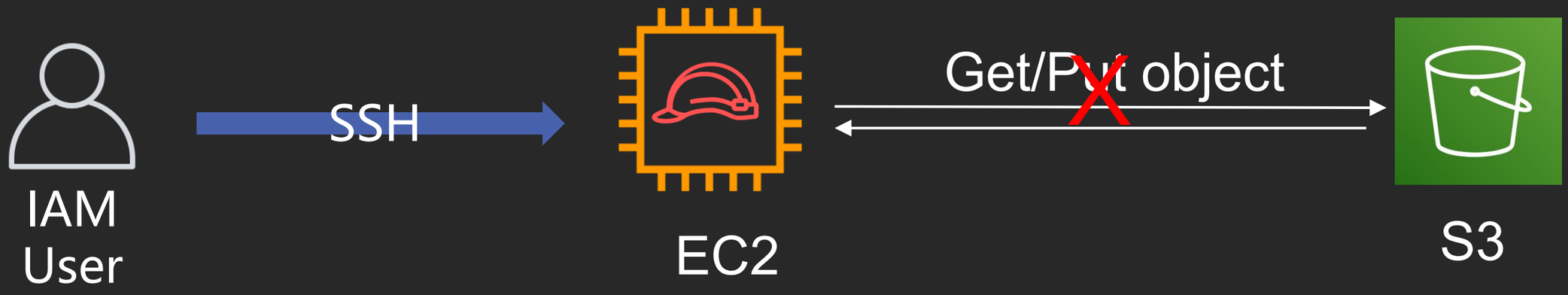


EC2 instances will have access to short-term credentials to access AWS resources

IAM Policy Assignment



Let’s understand how IAM role works

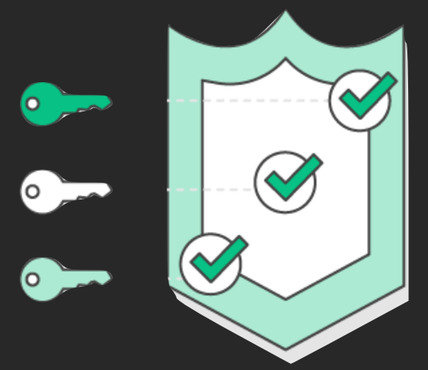


IAM policy for S3 access

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

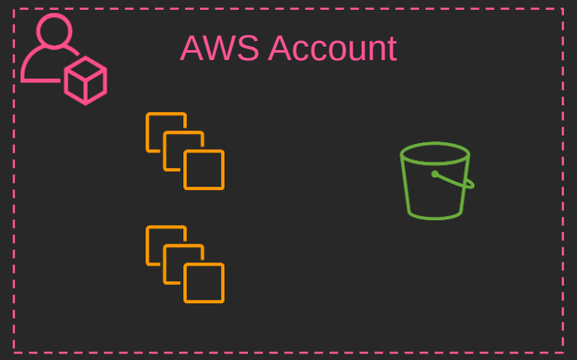
AWS IAM Best Practices

- Create individual IAM users. Don’t share user/credentials.
- Grant least privilege.
- Enable MFA.
- Use roles for applications (instead of sharing credentials).
- Rotate credentials regularly.
- Use policy conditions for extra security.
- Monitor activity in your AWS account.

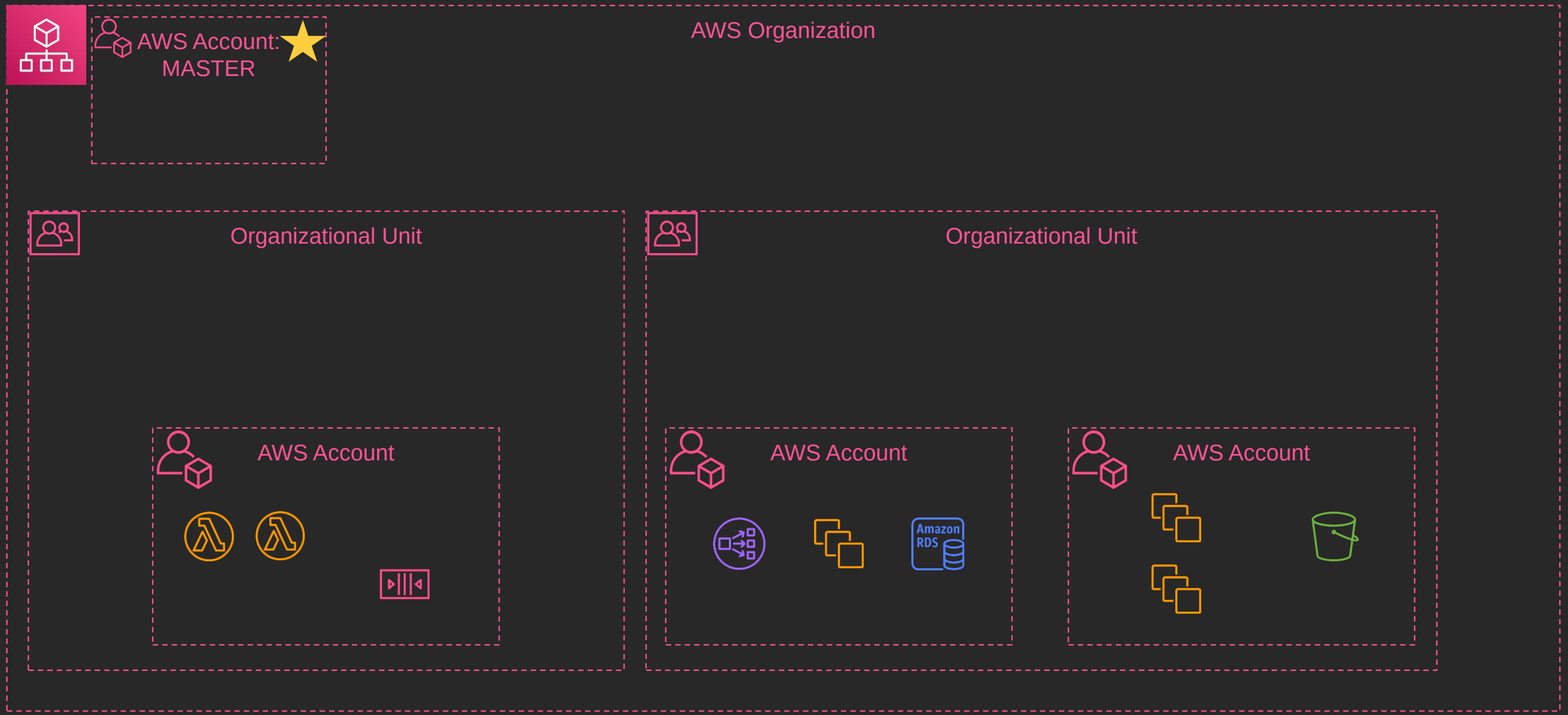


How to manage IAM access when there are more than one AWS accounts?

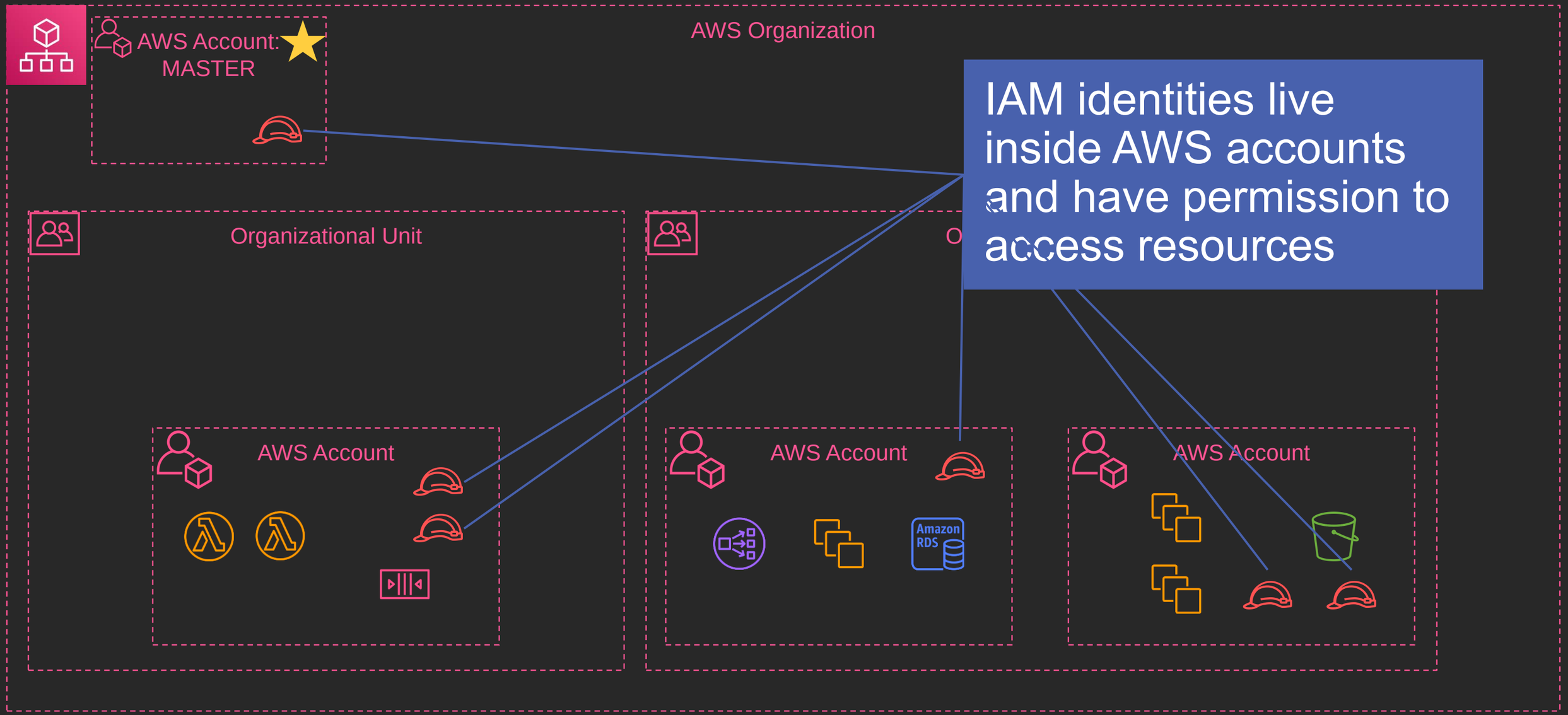
Accounts in AWS



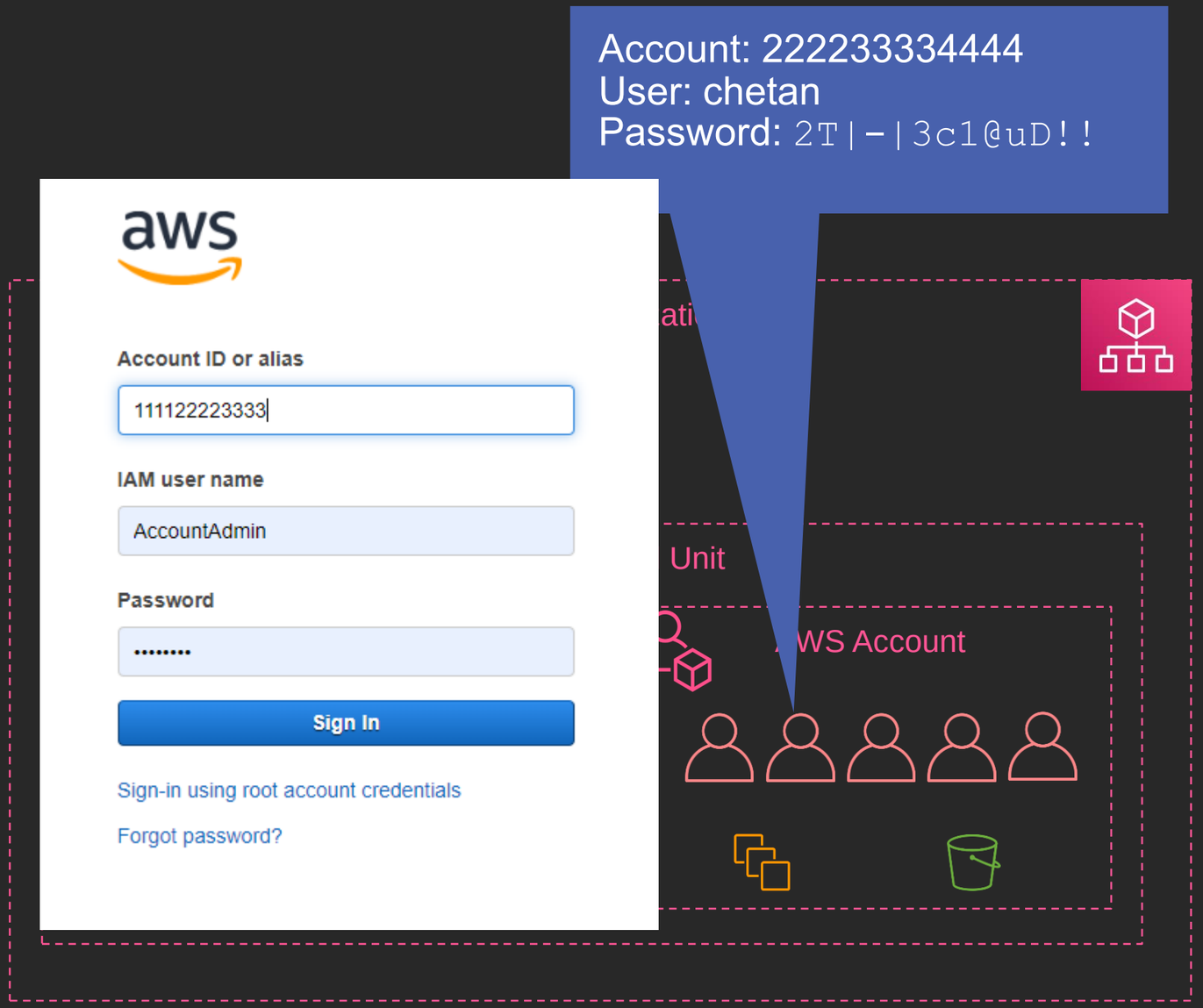
Accounts in AWS



Accounts in AWS



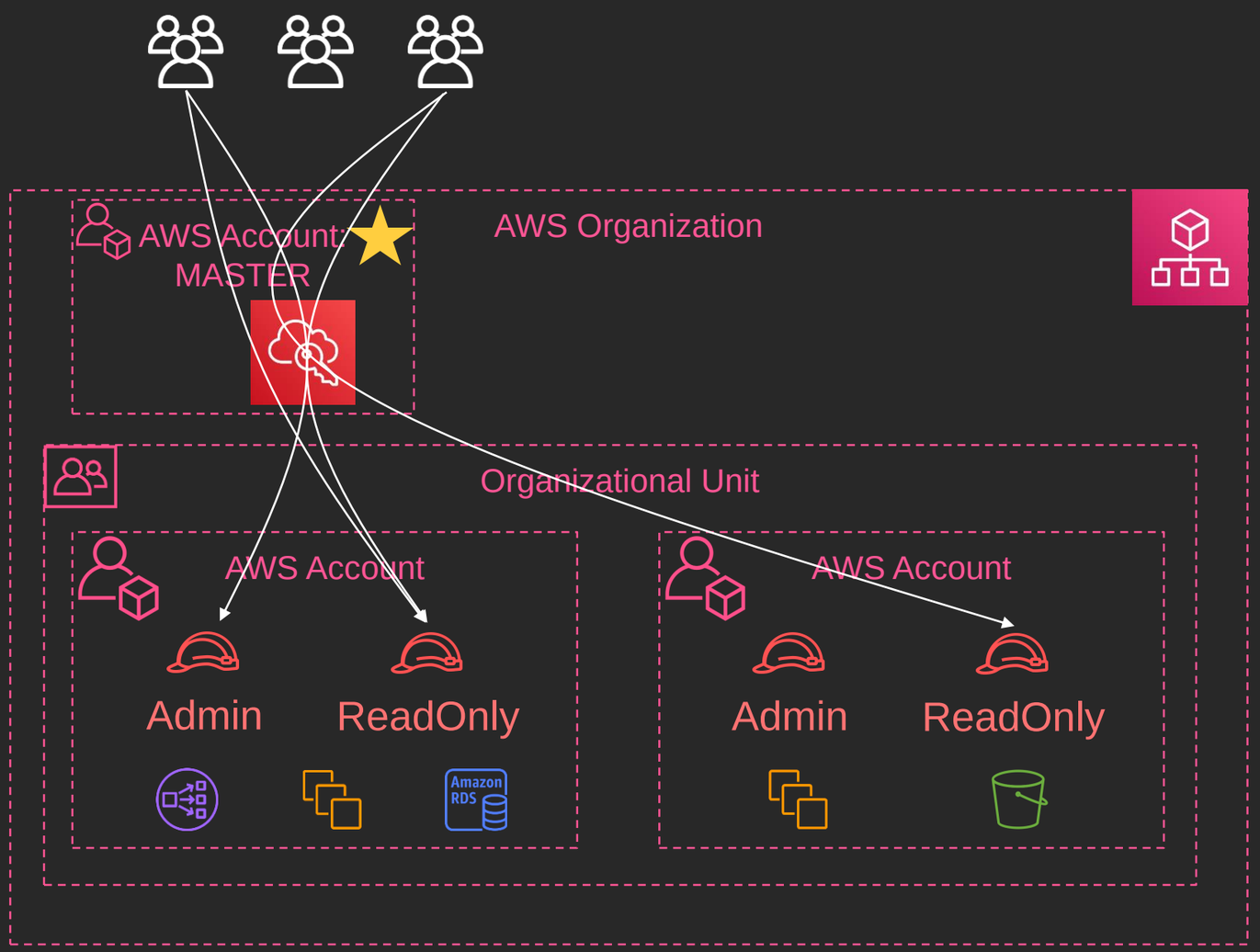
IAM users



Works best when you have:

- A relatively small number of users
- One AWS account, or a relatively small number of them
- A need for long-term credentials
- No user directory, or no ability to connect your directory to AWS
- Your very first AWS account

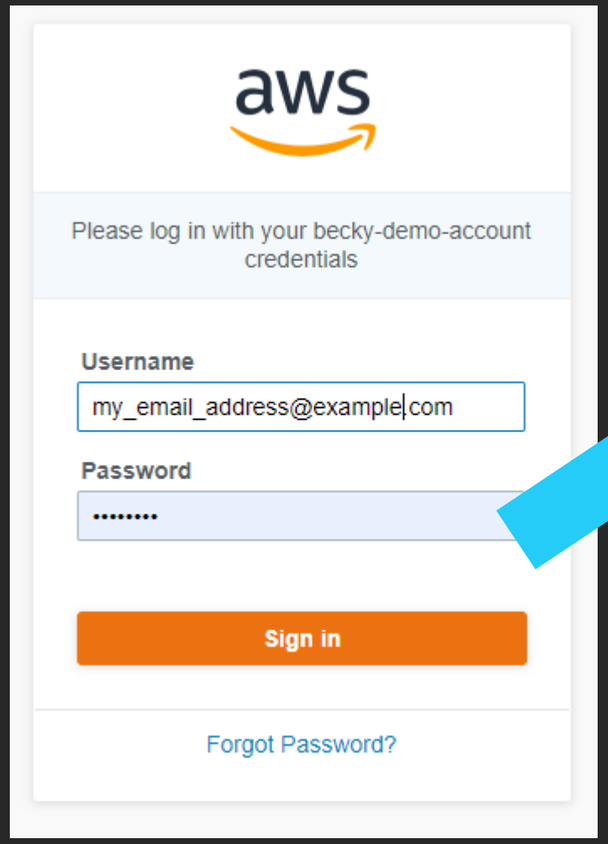
AWS Single Sign-On user pool

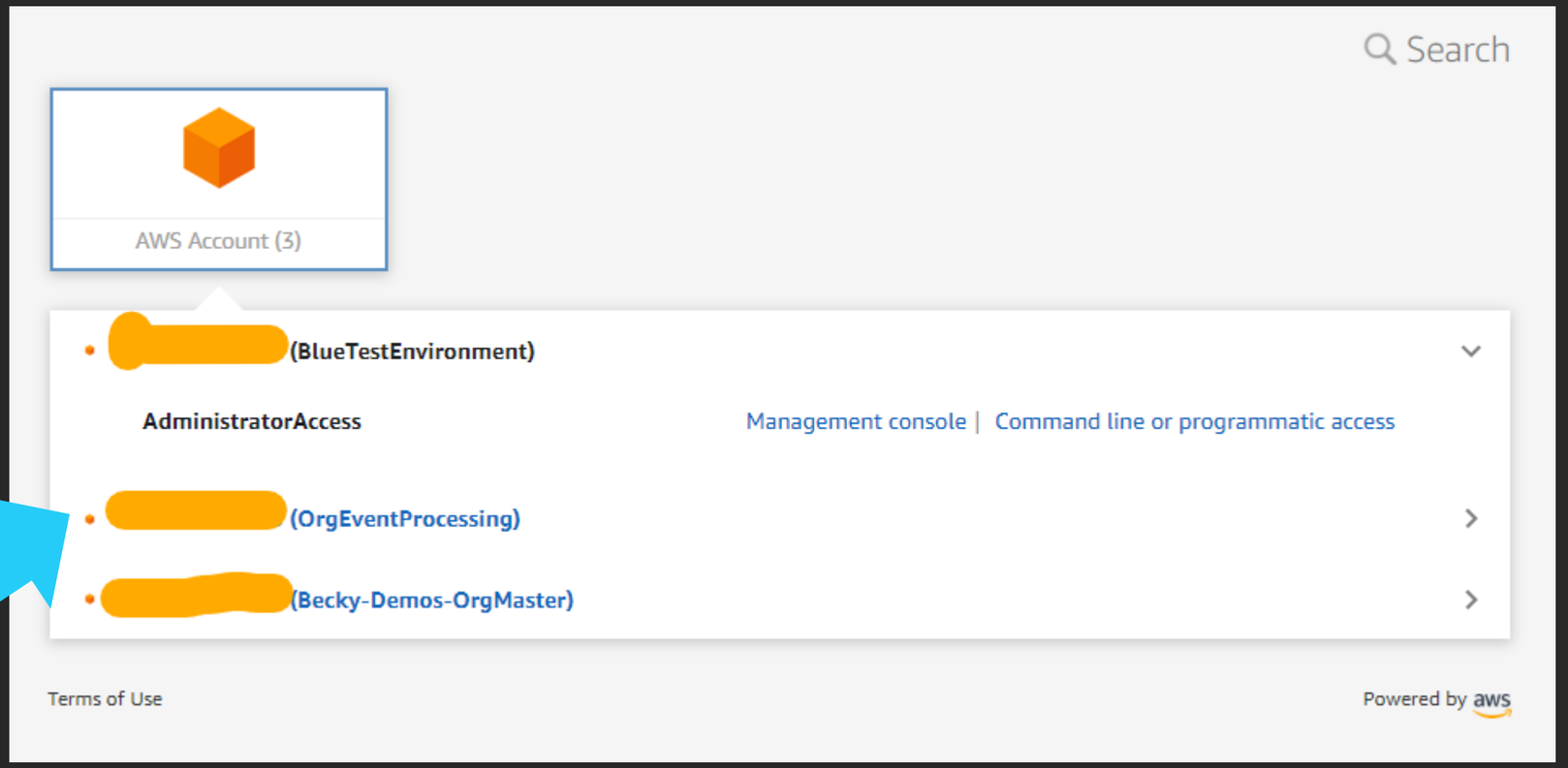


Works best when you have:

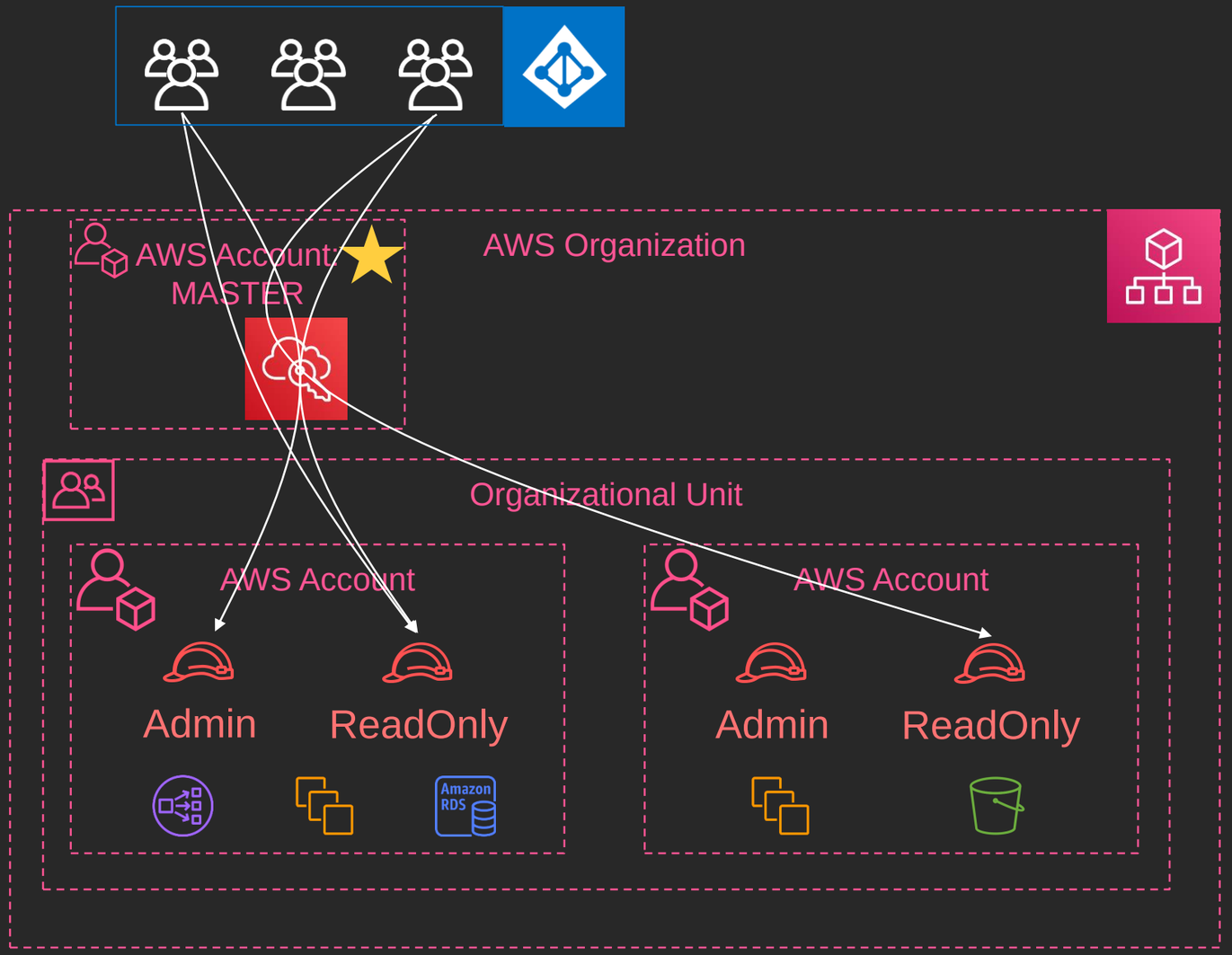
- A relatively small number of users
- Simple authorization schemes of humans into AWS
- Rules to map groups of users to AWS environments
- No user directory, or no ability to connect your directory to AWS

AWS SSO user pool: Sign-in workflow





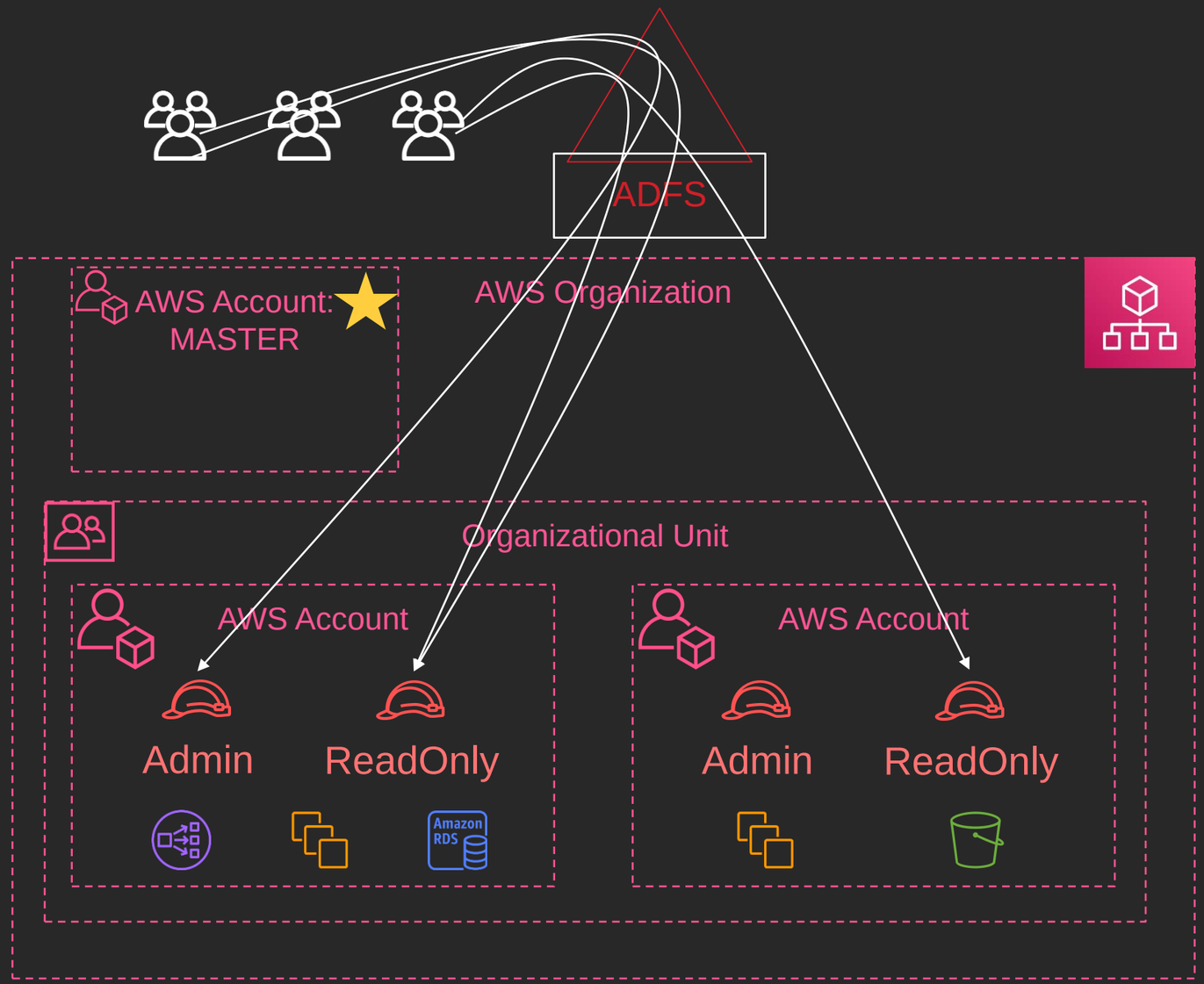
New: AWS Single Sign-On with Azure AD



Works best when you have:

- Azure Active Directory as identity source of truth
- Ability to enable SCIM
- Simple authorization schemes of humans into AWS
- Rules to map groups of users to AWS environments

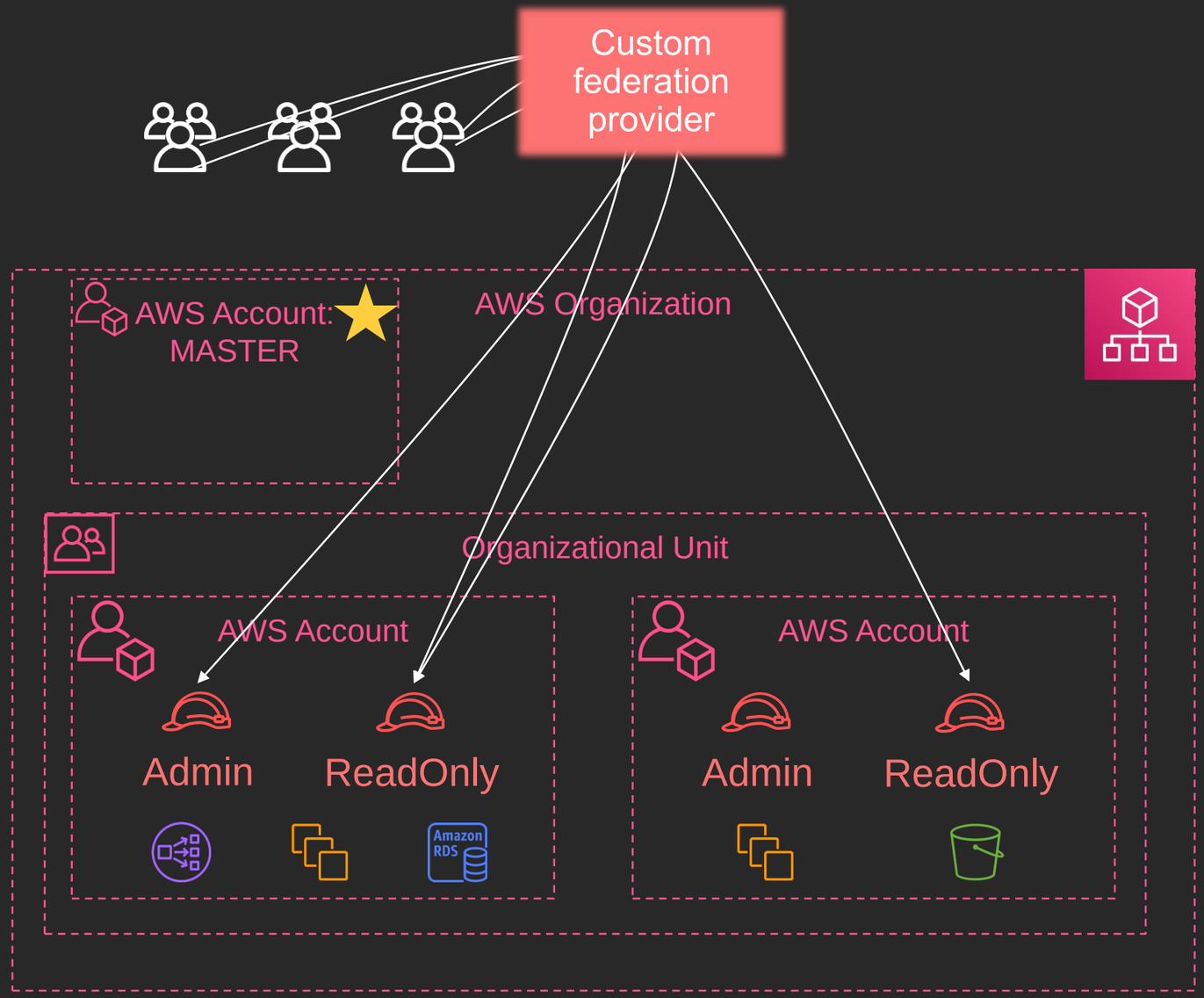
Active Directory Federation Services



Works best when you have:

- Corporate users in a Microsoft Active Directory, either on-premises or managed in AWS
- An ADFS connected to your directory
- Control over ADFS claims
- A need for granular control over user permissions

Custom directory integration



Works best when you have:

- Corporate users in a directory
- Either a third-party SaaS offering or a custom federation solution
- Rules to map groups of users to AWS environments
- A need for granular control over user permissions

Authenticating to AWS: Quick decision framework

If you have an existing user directory:

- AWS SSO with directory integrations
- Bring your own SAML federation (e.g., ADFS)
- Advanced use cases: Custom federation

If you don’t have an existing user directory:

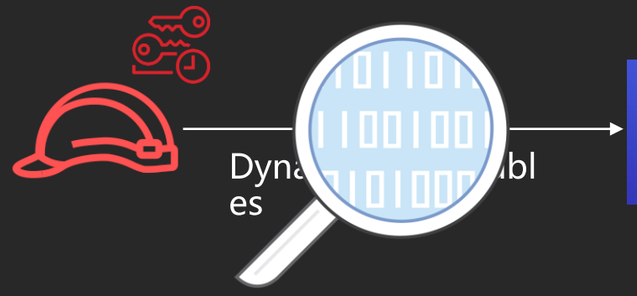
- AWS SSO with user pools
- IAM Users

How IAM authentication and authorization works?

How authentication works in AWS

(You don't need to know this part)

Short-term credentials for IAM role



```
POST https://dynamodb.us-east-2.amazon[illegible].com/ HTTP/1.1
Host: dynamodb.us-east-2.amazonaws.co[illegible]
X-Amz-Date: 20180918T150746Z
X-Amz-Target: DynamoDB_20120810.Lis[illegible]Tables
X-Amz-Security-Token: FQoGZXIvYXd[illegible]EKH///////// …
Content-Type: application/x-amz-[illegible]json-1.0
Authorization: AWS4-HMAC-SHA256
Credential=ASIAXXXXXXXXXXXXXXX/20180918/us-east-
1/dynamodb/aws4_request, SignedHeaders=content-
type;host;x-amz-date;x-amz-security-token;x-amz-target,
Signature=c1b4bc2df0c47c86cbcfa54d932e8aaa455b6b7c38e65d84
0f722254add1ea9e
```

Access key ID: Identifies calling IAM principal to AWS

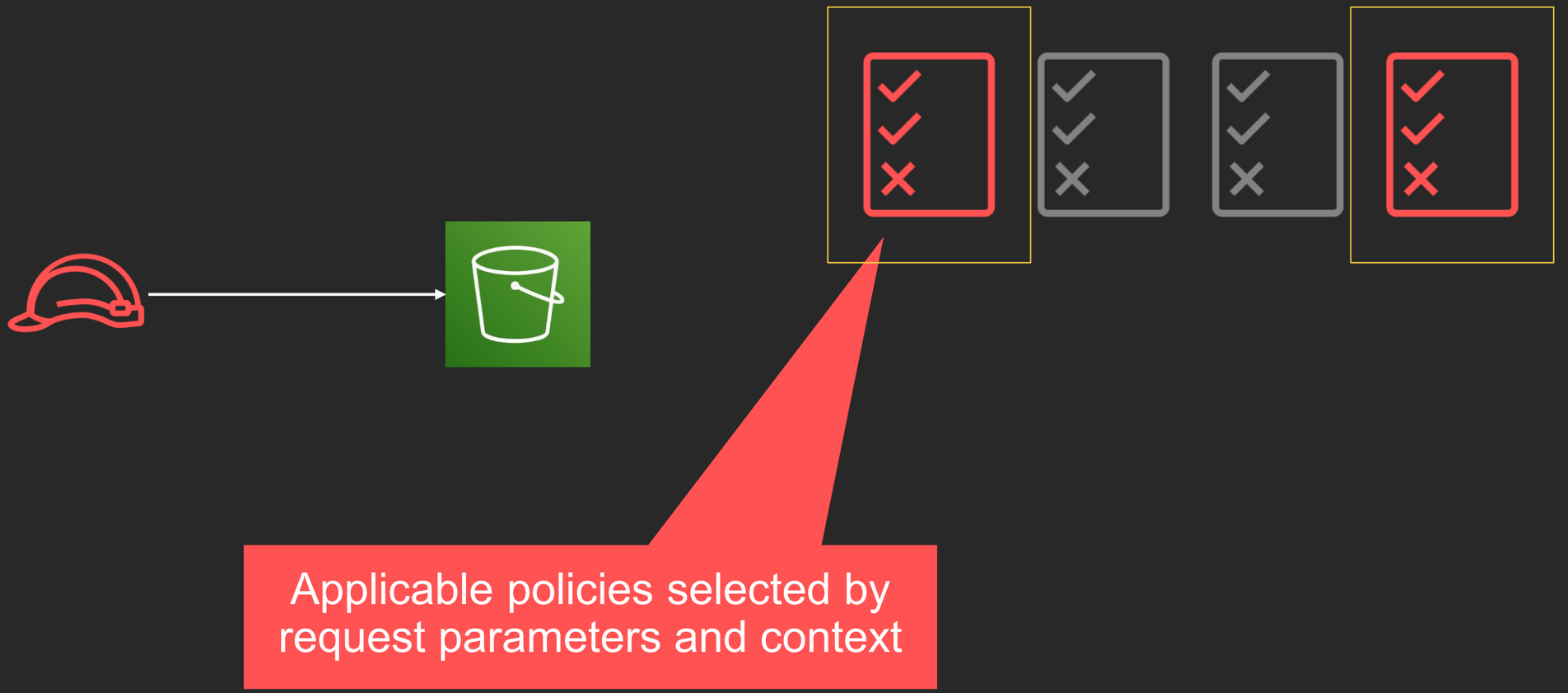
HMAC signature: Generated with secret key; validated by AWS

How authorization works in AWS

(You DO need to know this part)

How authorization works in AWS

(You DO need to know this part)



How authorization works in AWS

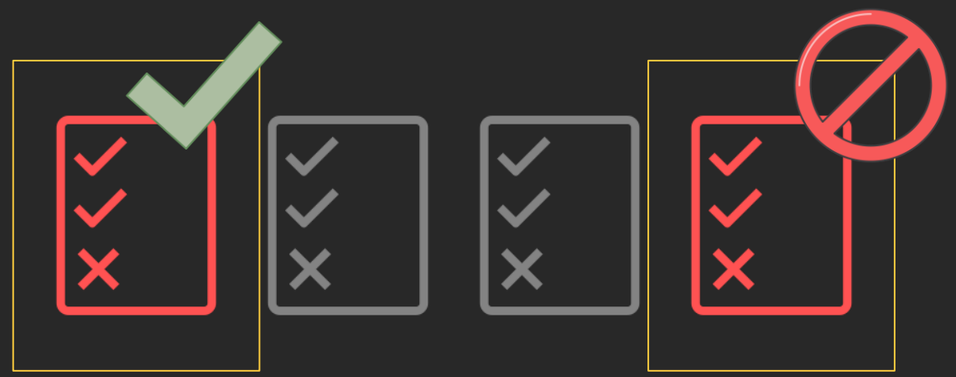
(You DO need to know this part)

The rules:

- If any policy denies → AccessDenied
- If some policy allows → Allow
- Otherwise AccessDenied

How authorization works in AWS

(You DO need to know this part)

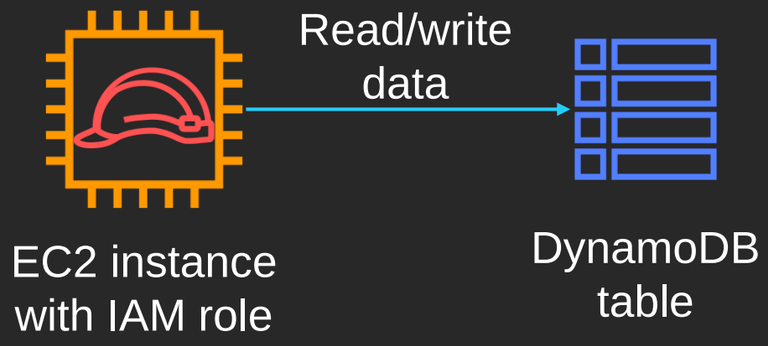


The rules:

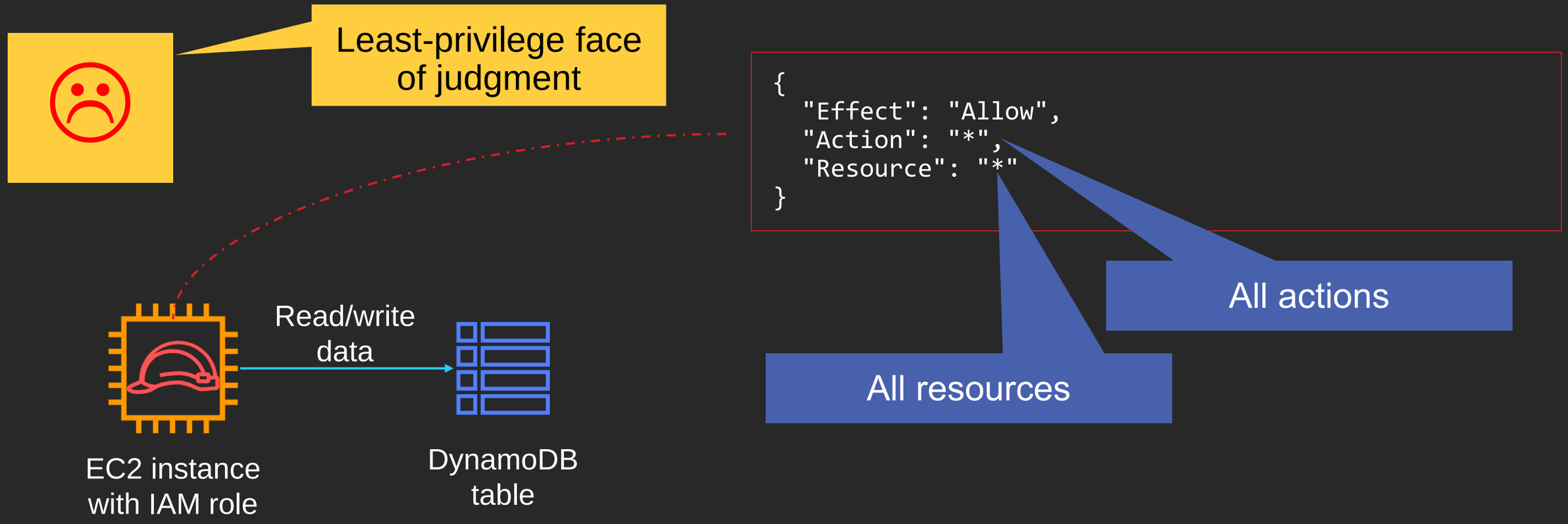
- If any policy denies → AccessDenied
- If some policy allows → Allow
- Otherwise AccessDenied

Reading and writing IAM policy

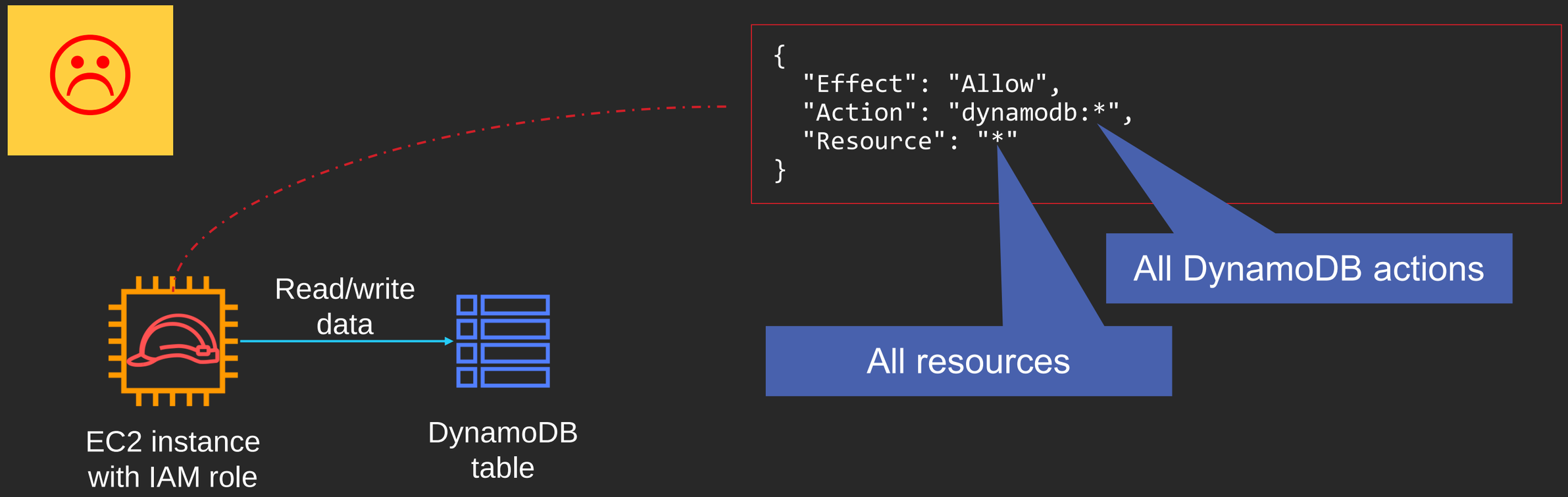
Example 1: Read data from DynamoDB



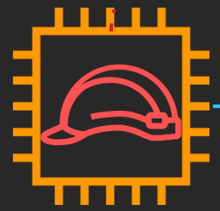
Example 1: Read data from DynamoDB



Example 1: Read data from DynamoDB



Example 1: Read data from DynamoDB



Read/write data

EC2 instance with IAM role

DynamoDB table

```
{
  "Effect": "Allow",
  "Action": [
    "dynamodb:GetItem",
    "dynamodb:PutItem"
  ],
  "Resource": "*"
}
```

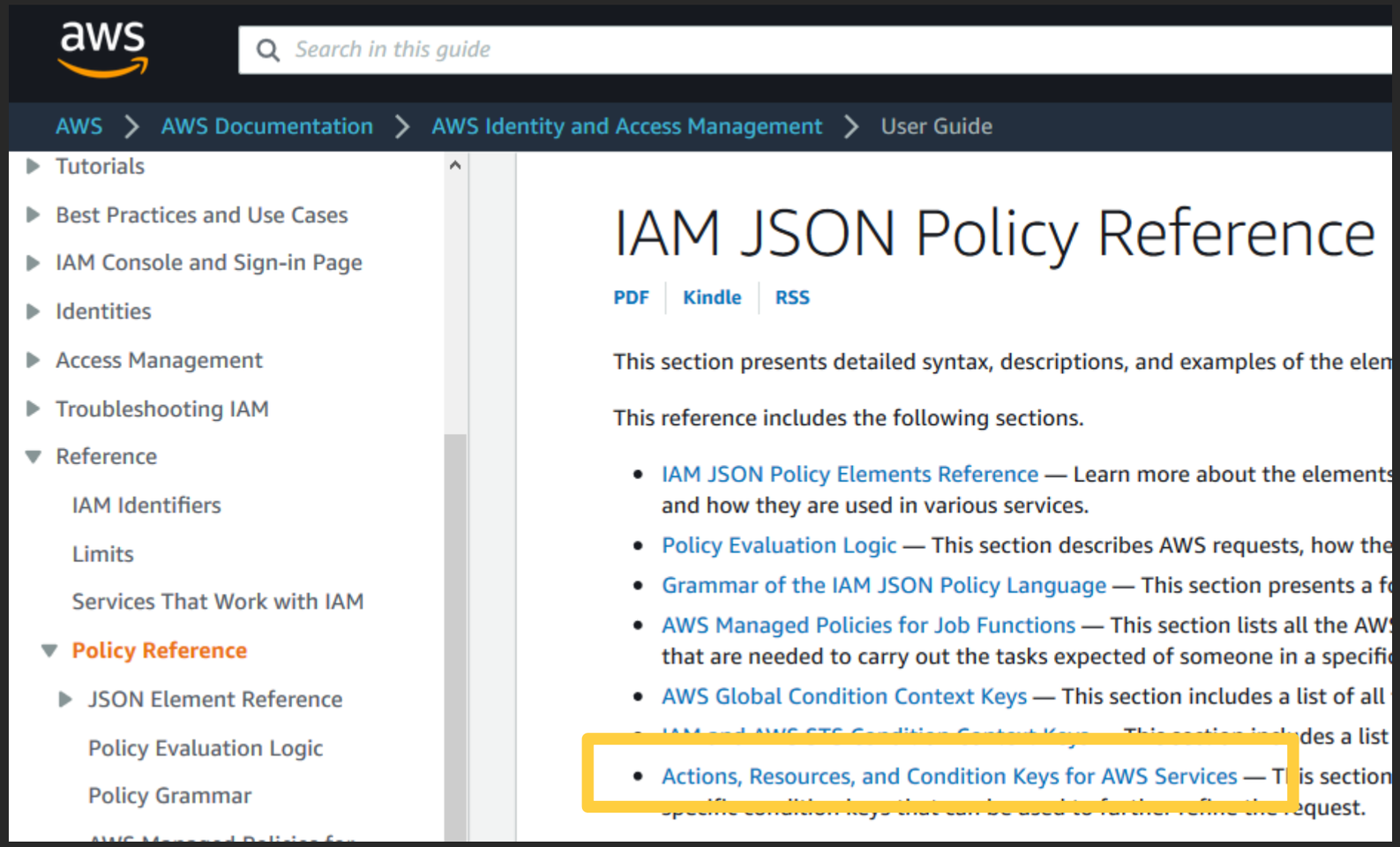
Specific DynamoDB actions for reading items from a table

All resources

Reading the IAM documentation page

Bookmark this page:

https://docs.aws.amazon.com/IAM/latest/UserGuide/reference_policies.html



Reading the IAM documentation page

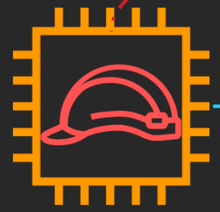
Action

Resource Condition

GetItem	The GetItem operation returns a set of attributes for the item with the given primary key	Read	table*	dynamodb:Attributes dynamodb:EnclosingOperation dynamodb:LeadingKeys dynamodb:ReturnConsumedCapacity dynamodb:Select

Reference: https://docs.aws.amazon.com/IAM/latest/UserGuide/list_amazondynamodb.html

Example 1: Read data from DynamoDB



Read/write data

EC2 instance with IAM role

DynamoDB table

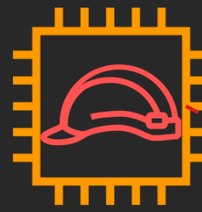
```
{
  "Effect": "Allow",
  "Action": [
    "dynamodb:GetItem",
    "dynamodb:PutItem"
  ],
  "Resource": [
    "arn:aws:dynamodb:us-east-2:111122223333:table/MyTable"
  ]
}
```

DynamoDB row: Reading/writing actions

Specific resource: Table

How authorization works in AWS

```
dynamodb.putItem({
  TableName:"MyTable",
  Item: {
    "Id": {
      S: "a1b2c3d4"
  …
});
```



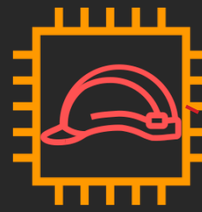
EC2 instance
with IAM role

DynamoDB
table

```
{
  "Effect": "Allow",
  "Action": [
    "dynamodb:GetItem",
    "dynamodb:PutItem"
  ],
  "Resource": [
    "arn:aws:dynamodb:us-east-2:111122223333:table/MyTable"
  ]
}
```

How authorization works in AWS

```
dynamodb.putItem({
  TableName:"MyTable",
  Item: {
    "Id": {
      S: "a1b2c3d4"
  …
});
```



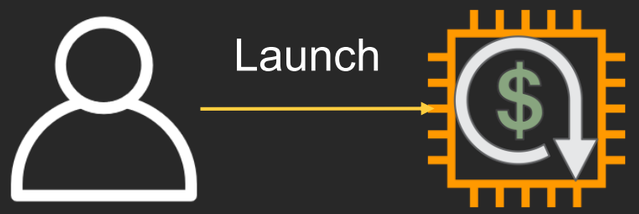
EC2 instance
with IAM role

DynamoDB
table

The “PutItem” action and the “table” resource match the Allow statement, so the request is allowed

```
{
  "Effect": "Allow",
  "Action": [
    "dynamodb:GetItem",
    "dynamodb:PutItem"
  ],
  "Resource": [
    "arn:aws:dynamodb:us-east-2:111122223333:table/MyTable"
  ]
}
```

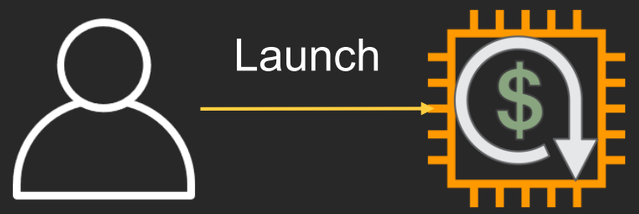
Example 2: Launch specific EC2 instance types




```
{
  "Effect": "Allow",
  "Action": "ec2:RunInstances",
  "Resource": "*",
}
```

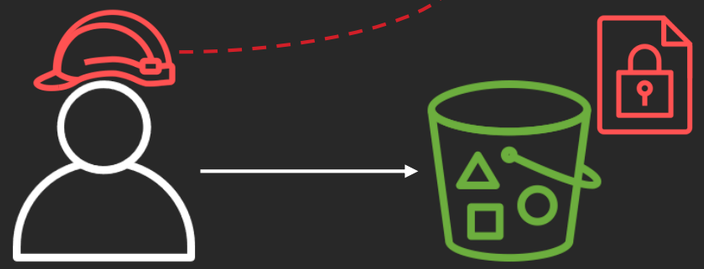
Is this policy good?

Example 2: Launch specific EC2 instance types




```
{
  "Effect": "Allow",
  "Action": "ec2:RunInstances",
  "Resource": "*",
  "Condition": {
    "StringEquals": {
      "ec2:InstanceType":[
        "t3.nano",
        "t3.micro"
      ]
    }
  }
}
```

Example 3: Prevent unencrypted writes to S3



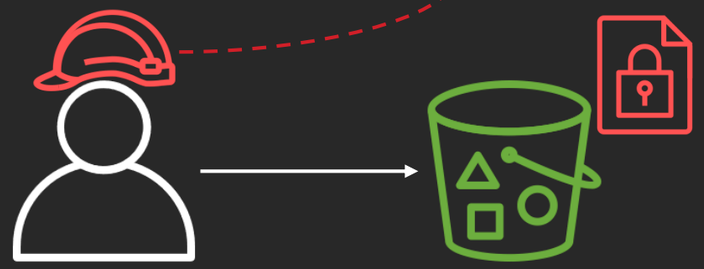
```
{
  "Effect": "Deny",
  "Action": "s3:PutObject",
  "Resource": "arn:aws:s3:::example-bucket/*",
  "Condition": {
    "StringNotEqualsIfExists": {
      "s3:x-amz-server-side-encryption-aws-kms-key-id": [
        "arn:aws:kms:us-east-2:111122223333:key/1234578-1234-
34-1234-12345678abcd"
      ]
    }
```

Deny access

. . . IfExists suffix evaluates to TRUE if the value is absent

Can I write my encrypted data?

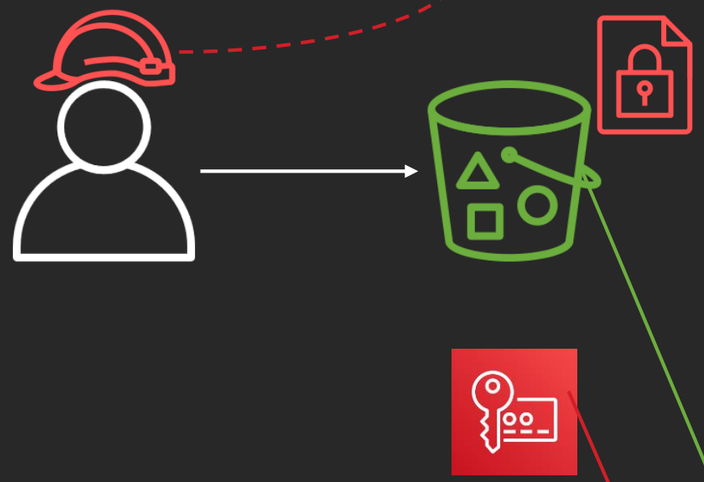
Example 3: Prevent unencrypted writes to S3



```
{
  "Effect": "Deny",
  "Action": "s3:PutObject",
  "Resource": "arn:aws:s3:::example-bucket/*",
  "Condition": {
    "StringNotEqualsIfExists": {
      "s3:x-amz-server-side-encryption-aws-kms-key-id": [
        "arn:aws:kms:us-east-2:111122223333:key/1234578-1234-
1234-1234-12345678abcd"
      ]
    }
  }
},
{
  "Effect": "Allow",
  "Action": [ "s3:GetObject", "s3:PutObject" ],
  "Resource": "arn:aws:s3:::example-bucket/*"
}
```

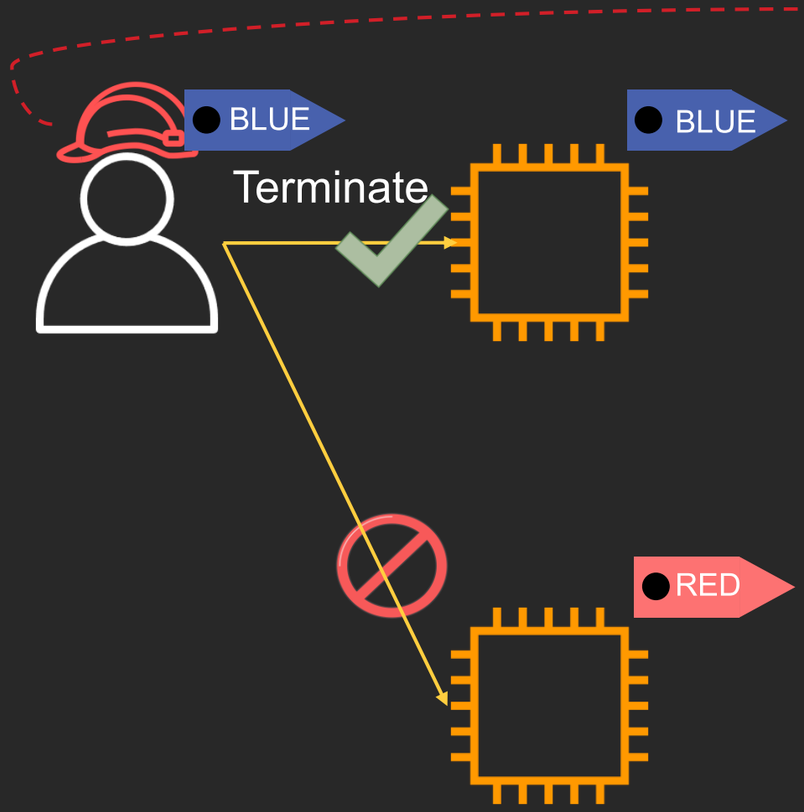
Can I write my encrypted data?

Example 3: Prevent unencrypted writes to S3



```
{
  "Effect": "Deny",
  "Action": "s3:PutObject",
  "Resource": "arn:aws:s3:::example-bucket/*",
  "Condition": {
    "StringNotEqualsIfExists": {
      "s3:x-amz-server-side-encryption-aws-kms-key-id": [
        "arn:aws:kms:us-east-2:111122223333:key/12345678-
1234-1234-1234-12345678abcd"
      ]
    }
  }
},
{
  "Effect": "Allow",
  "Action": [ "s3:GetObject", "s3:PutObject" ],
  "Resource": "arn:aws:s3:::example-bucket/*"
},
{
  "Effect": "Allow",
  "Action": [ "kms:GenerateDataKey*", "kms:Decrypt" ],
  "Resource": "arn:aws:kms:us-east-
2:111122223333:key/12345678-1234-1234-1234-12345678abcd"
}
```

Example 4: Terminate instances in your project




```
{
  "Effect": "Allow",
  "Action": "ec2:TerminateInstances",
  "Resource": "arn:aws:ec2:*:111122223333:instance/*",
  "Condition": {
    "StringEquals": {
      "ec2:ResourceTag/Project": "${aws:PrincipalTag/Project}"
    }
  }
}
```

Resolves to value of the EC2 instance's "Project" tag

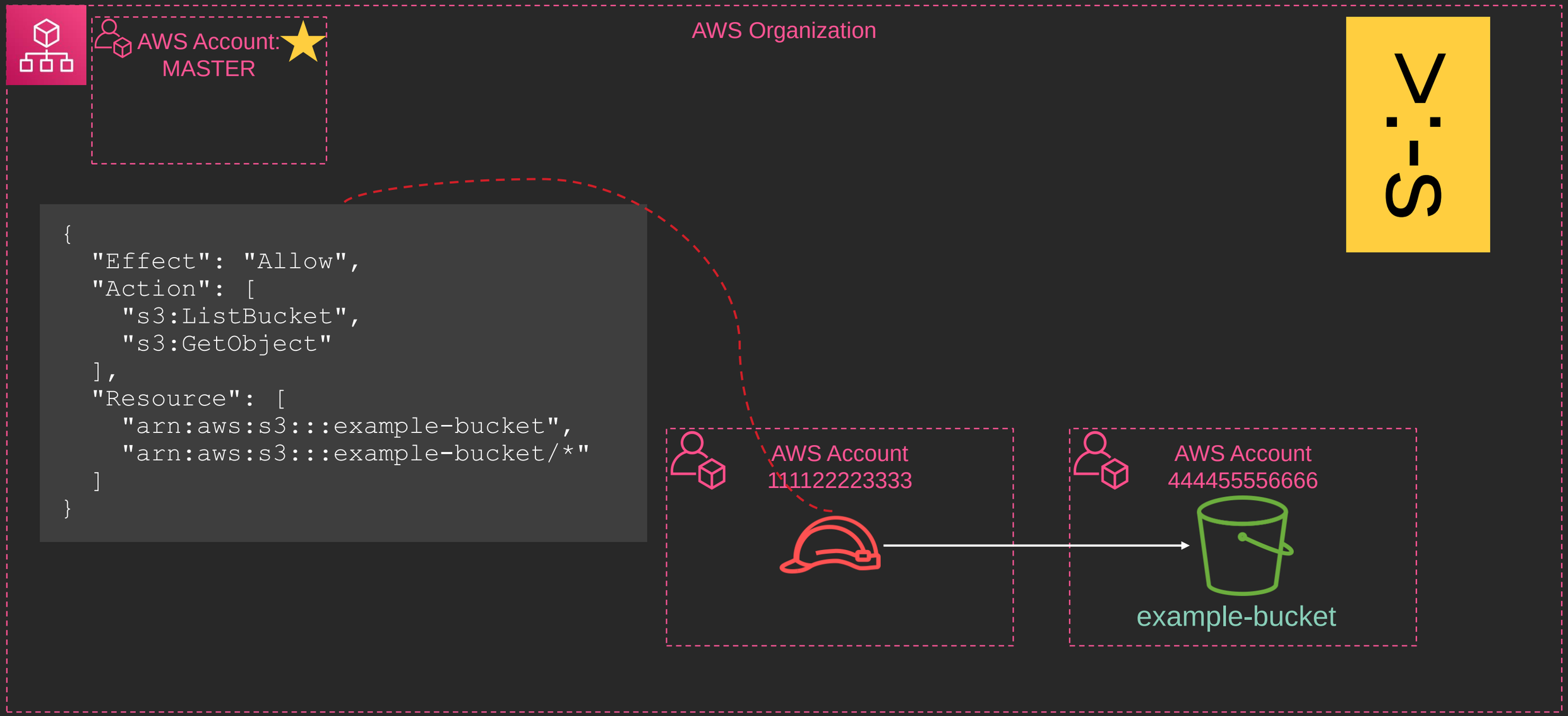
Resolves to value of the caller IAM principal's "Project" tag

Summary of what just happened

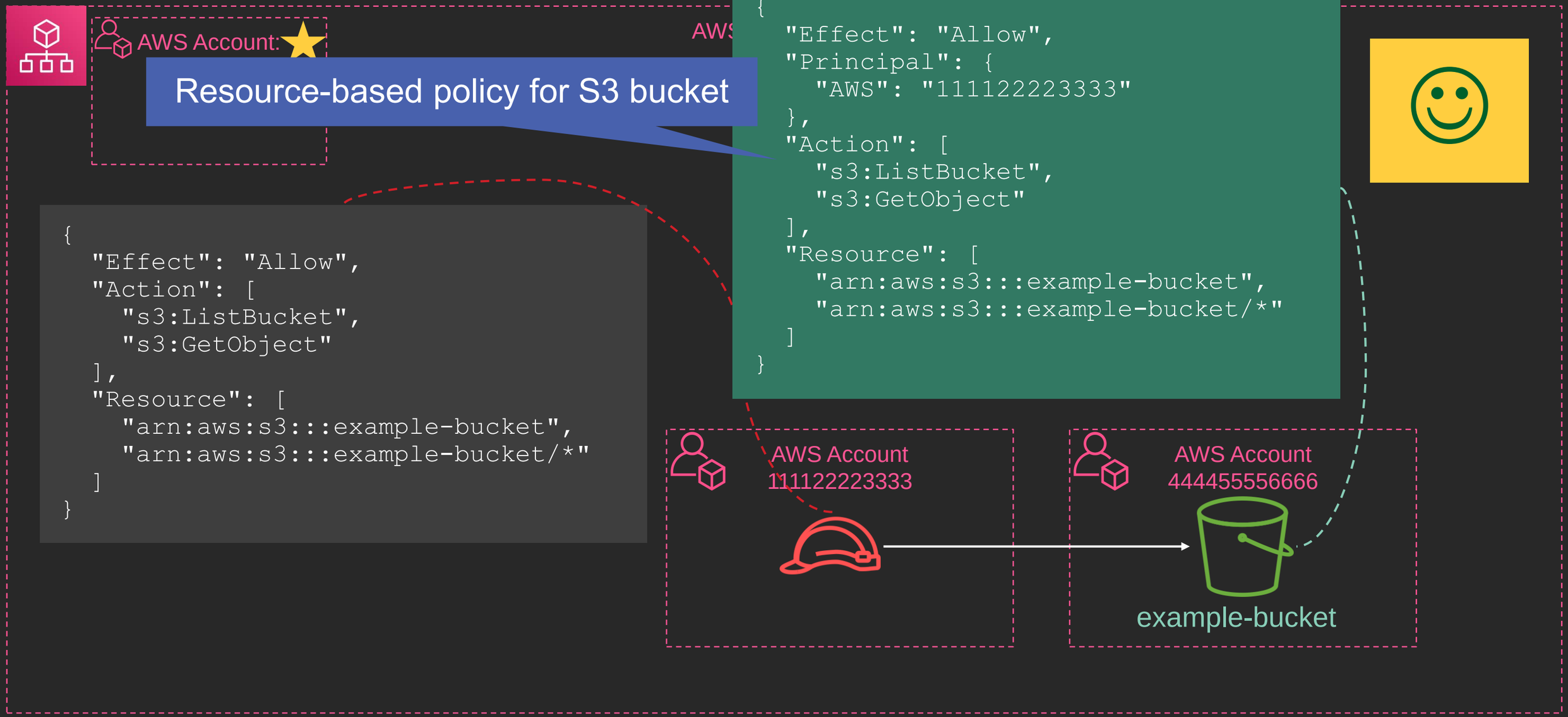
- AWS authorizations perform string-matching, and you need to find:
  - A matching Allow statement
  - No matching Deny statements
- Each resource for an action get authorized, and all must be allowed
- Many resources in AWS offer additional authorization conditions
- Tags can be used in IAM policy for authorization

Accessing AWS resource beyond your AWS account

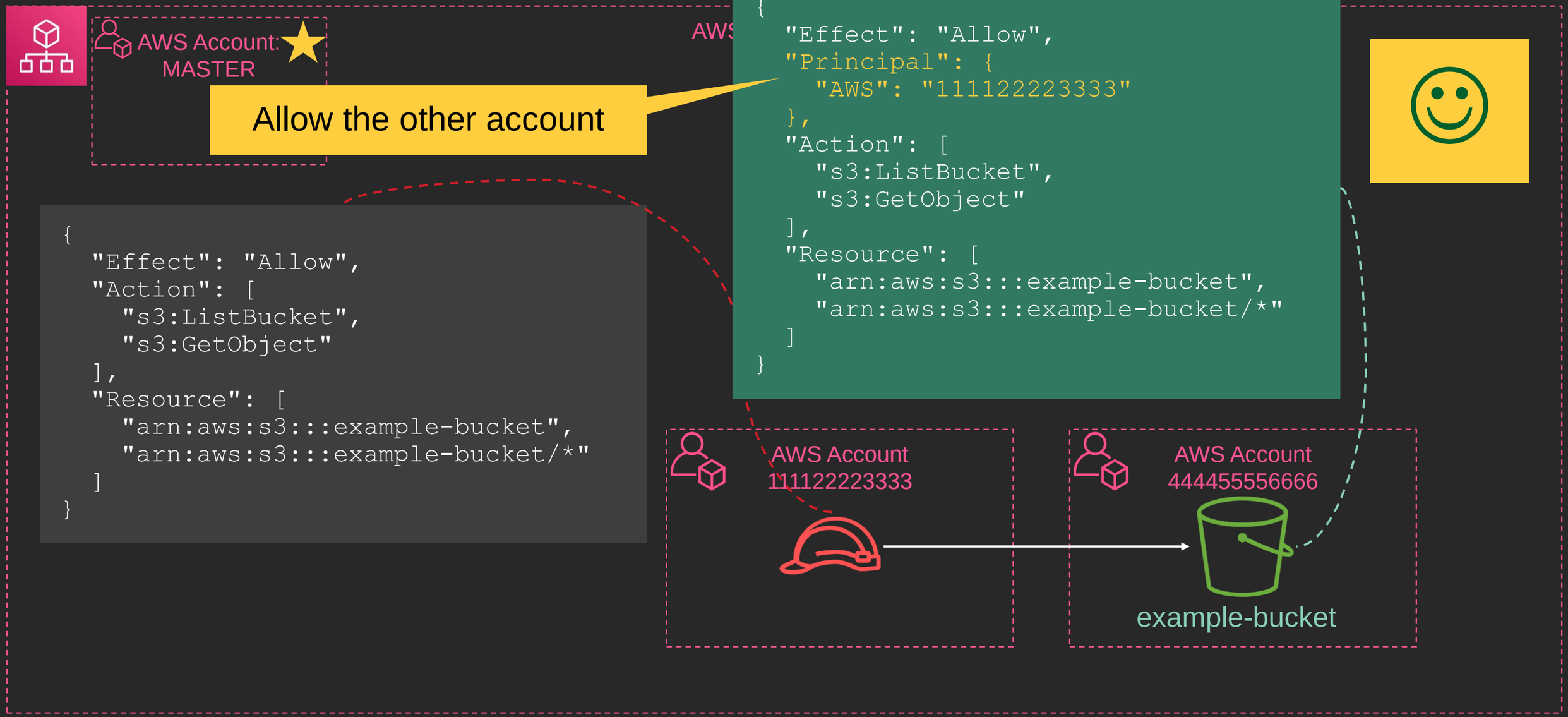
Accessing resources in another AWS account



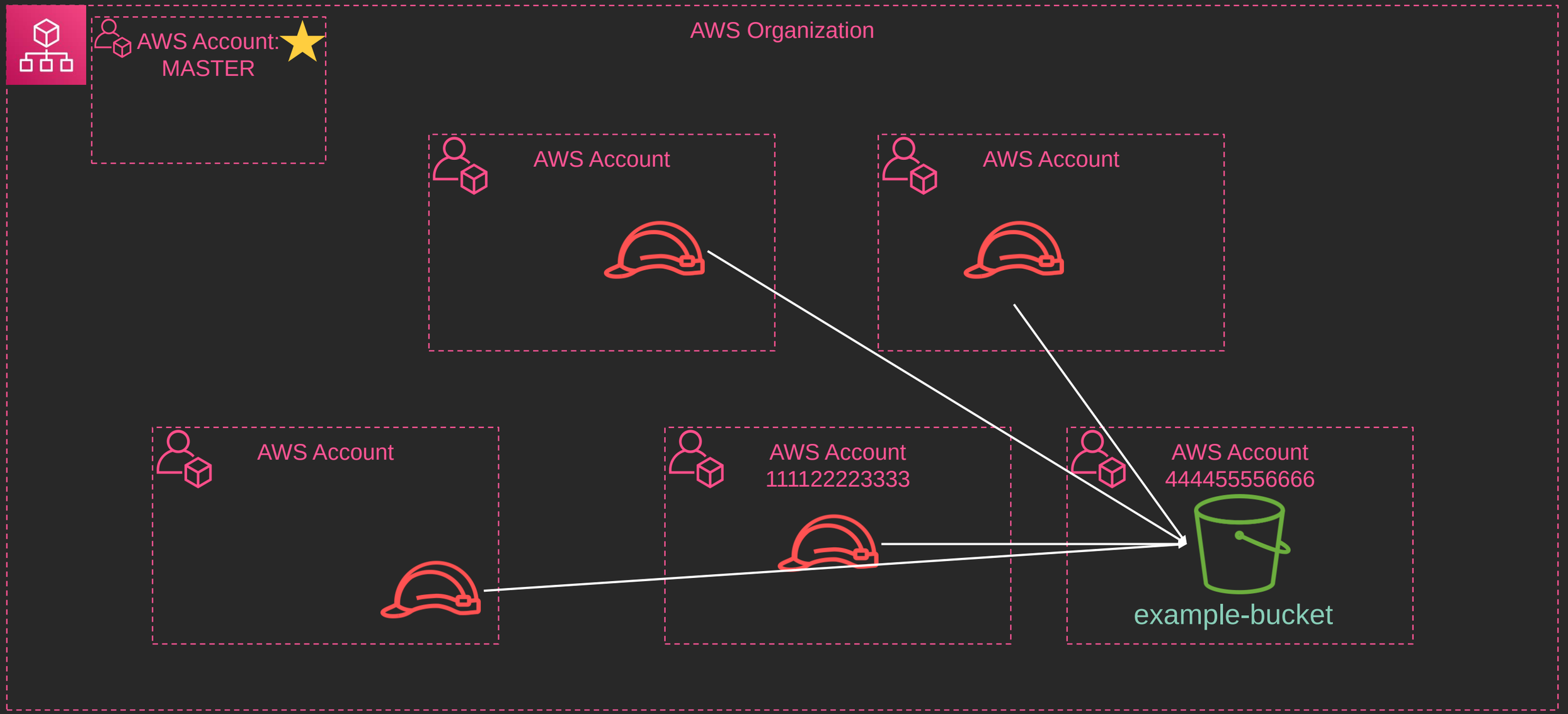
Accessing resources in another AWS account



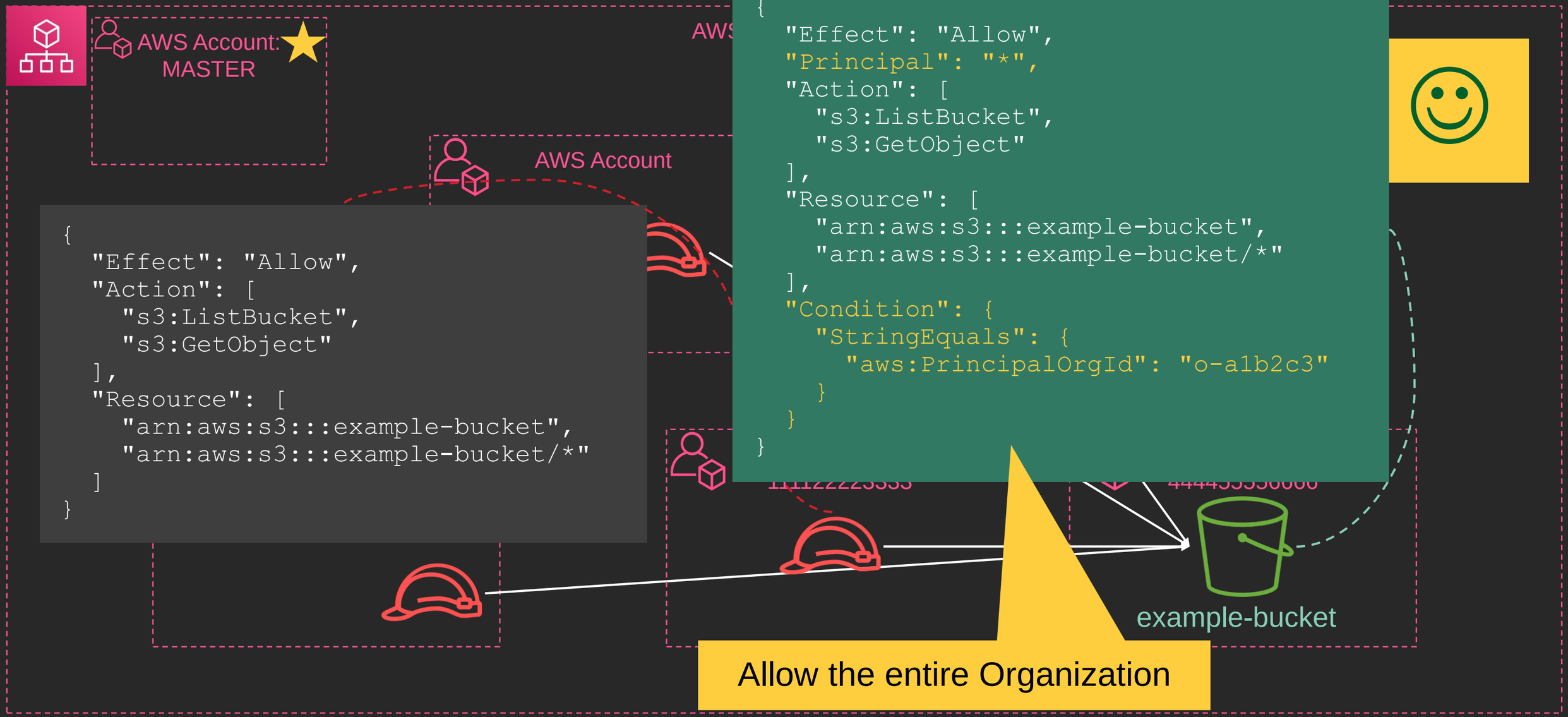
Accessing resources in another AWS account



Accessing resources in another AWS account



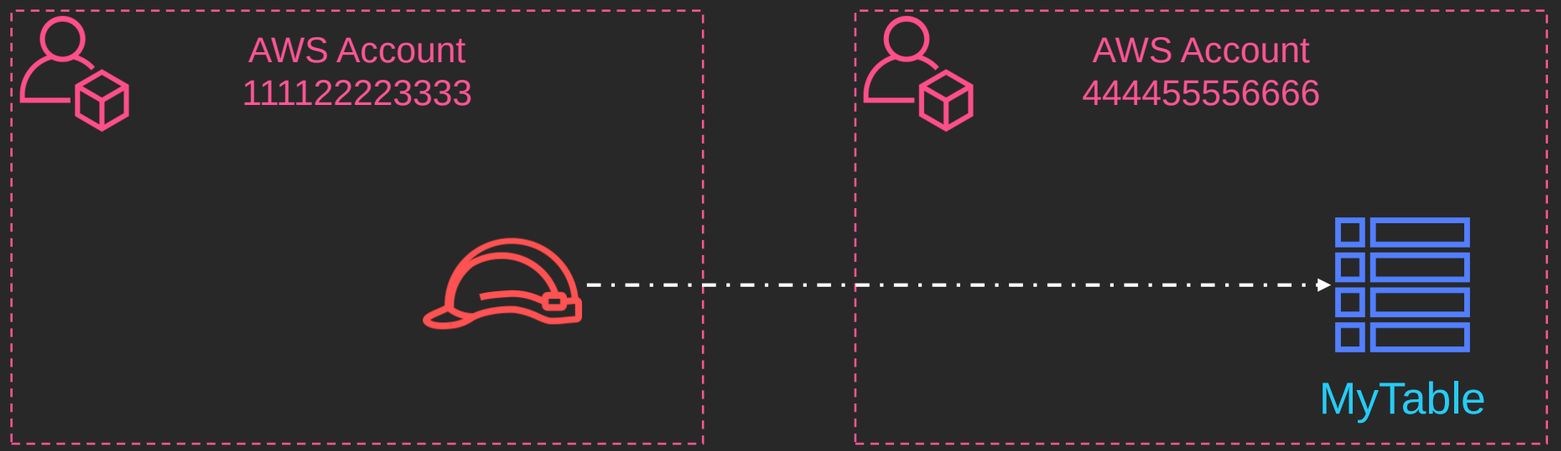
Accessing resources in another AWS account




```
{
  "Effect": "Allow",
  "Action": [
    "s3:ListBucket",
    "s3:GetObject"
  ],
  "Resource": [
    "arn:aws:s3:::example-bucket",
    "arn:aws:s3:::example-bucket/*"
  ]
}
```

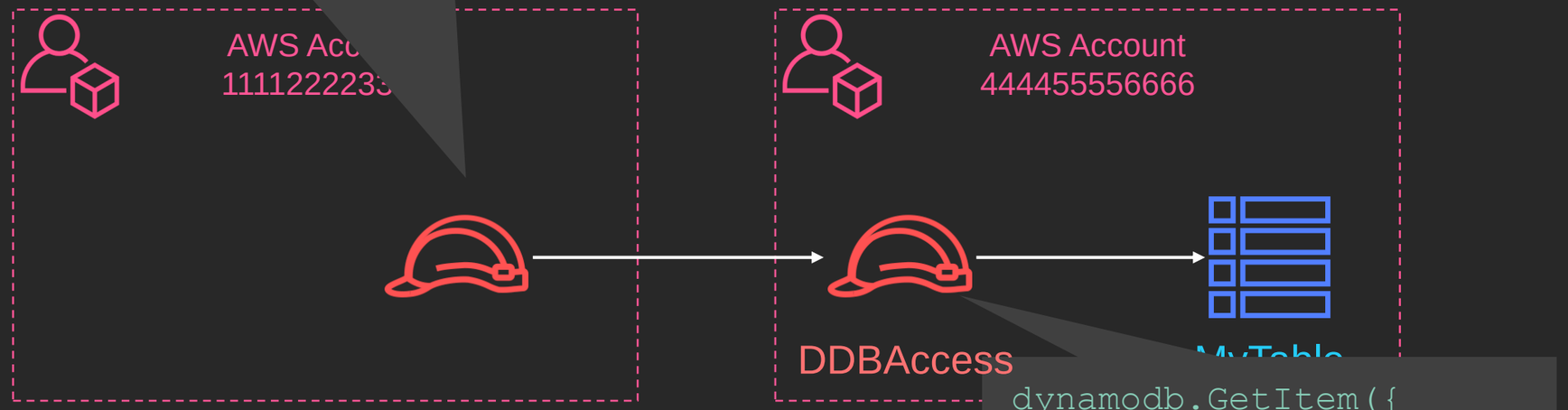
```
{
  "Effect": "Allow",
  "Principal": "*",
  "Action": [
    "s3:ListBucket",
    "s3:GetObject"
  ],
  "Resource": [
    "arn:aws:s3:::example-bucket",
    "arn:aws:s3:::example-bucket/*"
  ],
  "Condition": {
    "StringEquals": {
      "aws:PrincipalOrgId": "o-a1b2c3"
    }
  }
}
```

Accessing resources in another AWS account



Accessing resources in another AWS account: Using IAM roles across accounts

```
sts.AssumeRole({
  RoleArn: 'arn:aws:iam::444455556666:role/DDBAccess',
  …
});
```

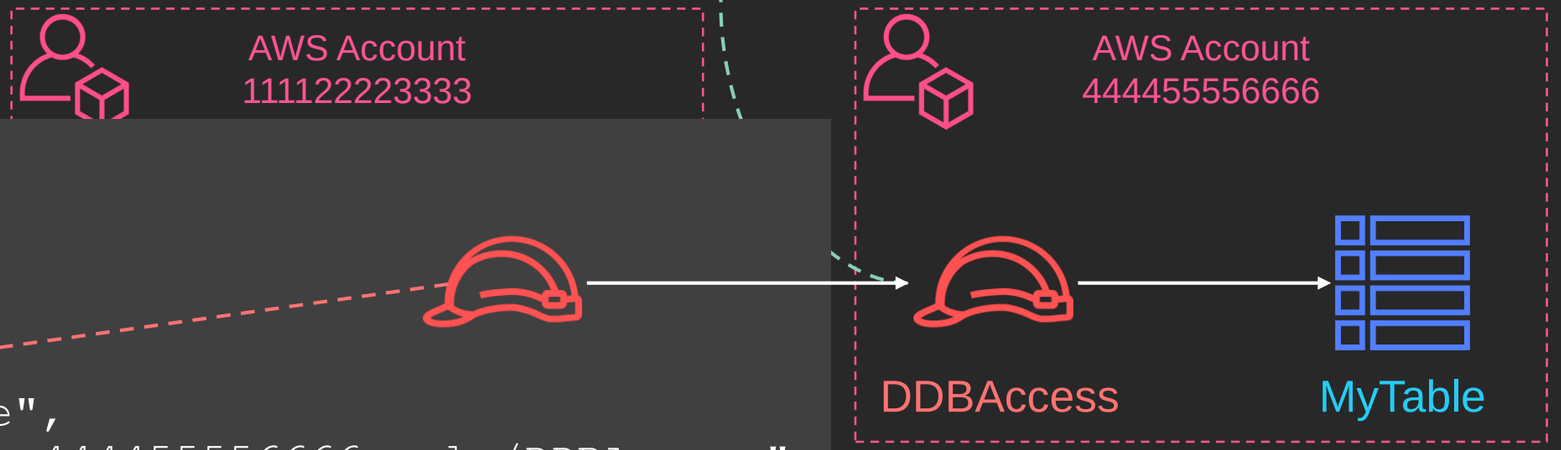



```
dynamodb.GetItem({
  TableName: 'MyTable',
  …
});
```

Accessing resources in another AWS account: Using IAM roles across accounts

IAM role trust policy:

```
{
  "Effect": "Allow",
  "Principal": {
    "AWS": "111122223333"
  },
  "Action": "sts:AssumeRole",
}
```



Principal policy:

```
{

  "Effect": "Allow",
  "Action": "sts:AssumeRole",
  "Resource": "arn:aws:iam::444455556666:role/DDBAccess"
}
```

Recommendations for cross-account access

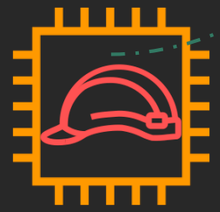
Keep it simple:

- Use resource-based policies when available
- Unless you have a specific reason to do otherwise:
  - Trust the entire other account, or
  - Trust the AWS Organization
- Use IAM roles if resource-based policies are not available
  - Follow the above rules for their trust policies (i.e., resource-based policies for IAM roles)

Role trust policy for aws services

IAM role trust policy:

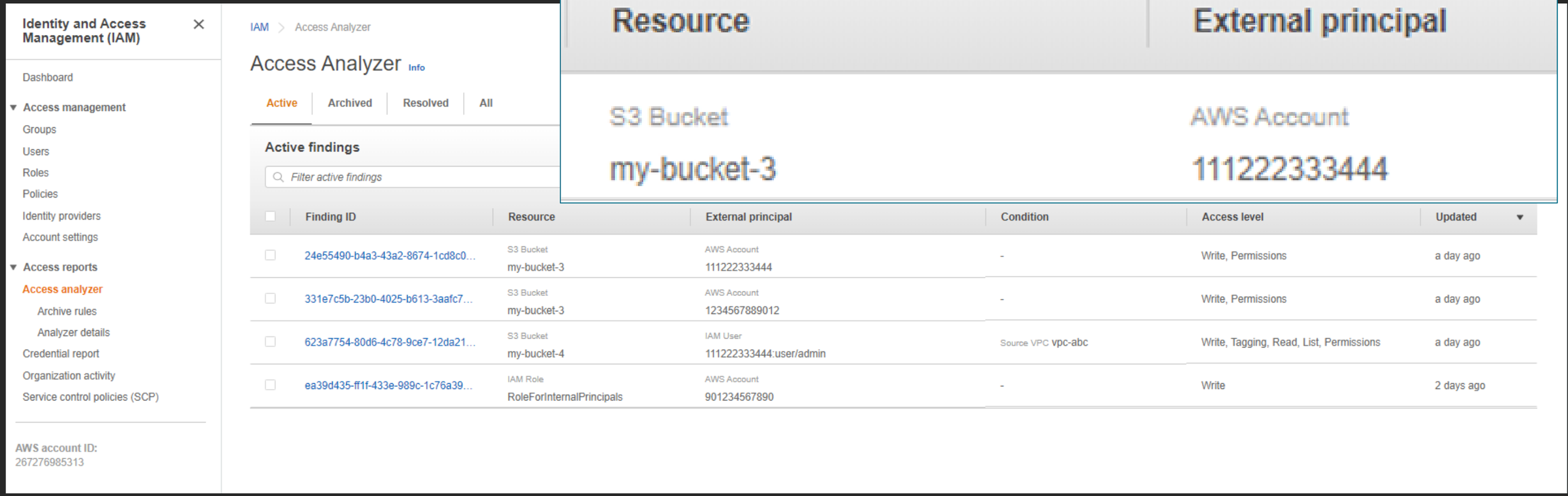
```
{
  "Effect": "Allow",
  "Principal": {
    "Service": "ec2.amazonaws.com"
  },
  "Action": "sts:AssumeRole",
}
```



EC2 instance
with IAM role

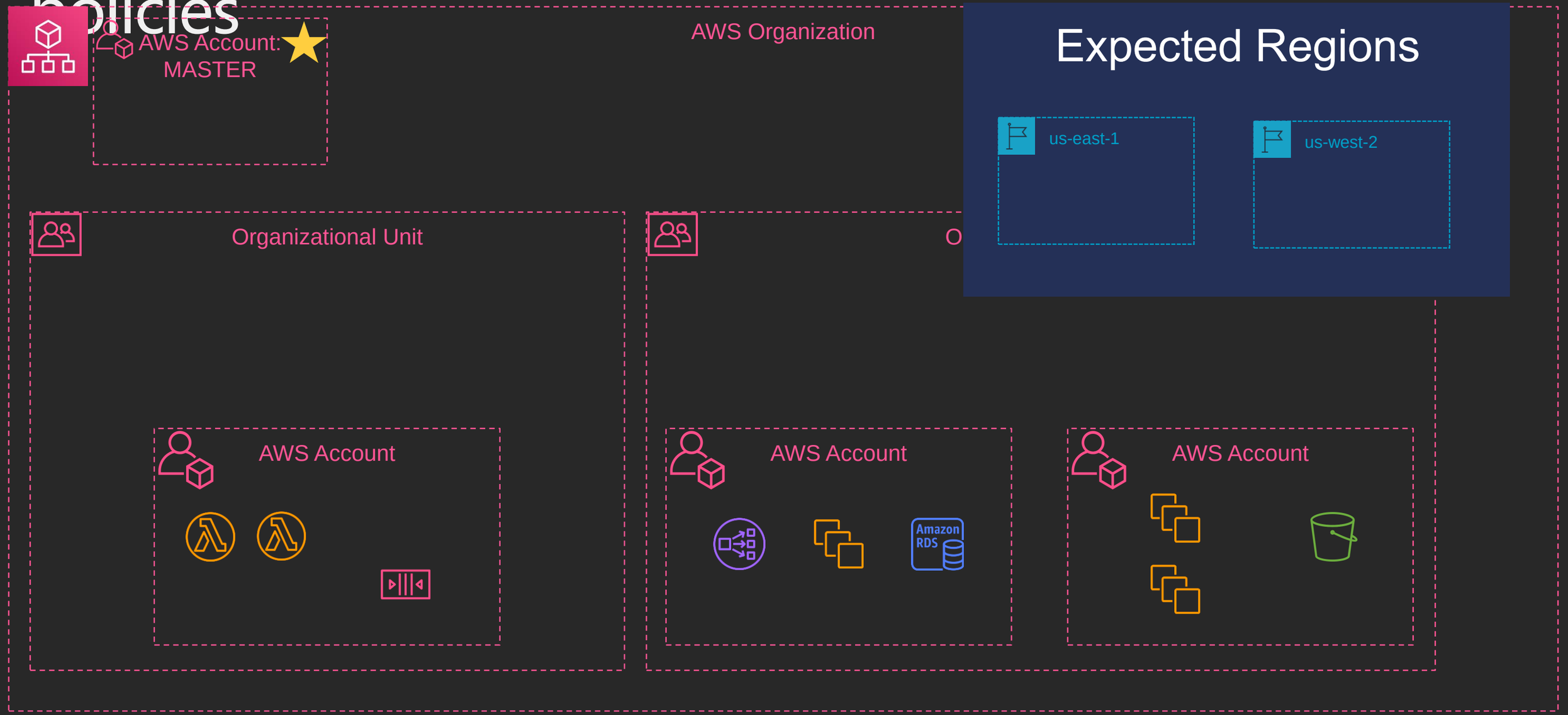
The EC2 service can assume this role and vend short-term credentials to your application

New: Access Analyzer

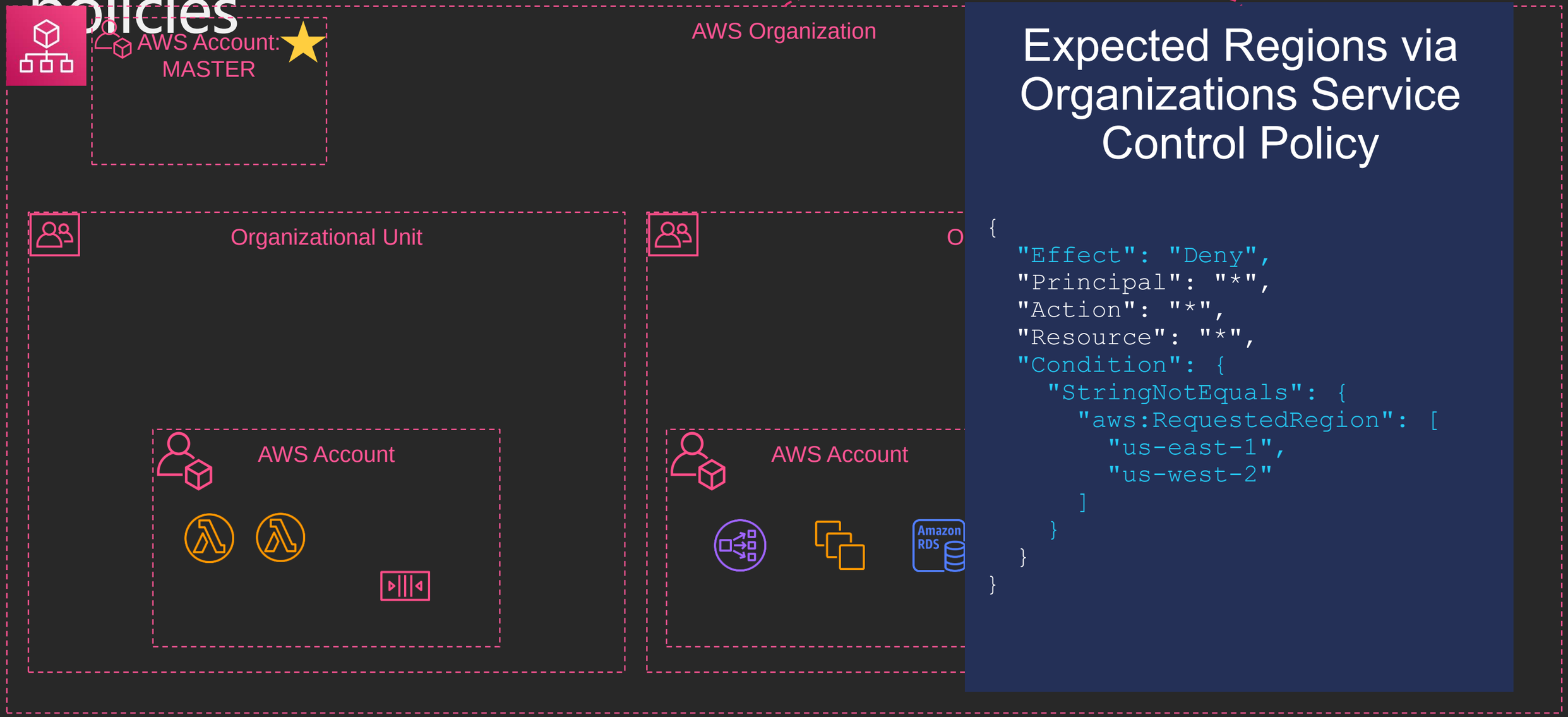


IAM pro move:
Guardrails with Organizations

Guardrails for IAM: Organizations service control policies



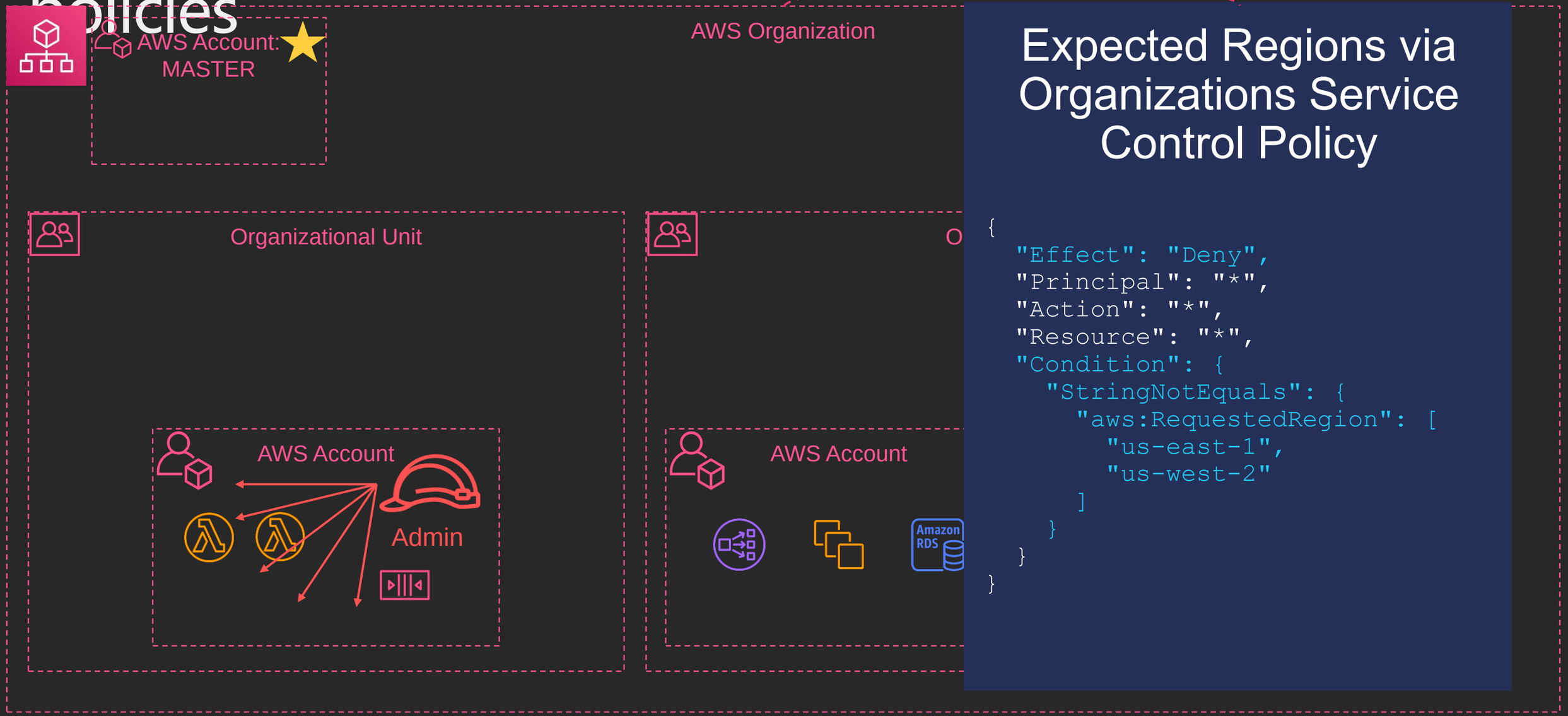
Guardrails for IAM: Organizations service control policies



Expected Regions via Organizations Service Control Policy

```
{
  "Effect": "Deny",
  "Principal": "*",
  "Action": "*",
  "Resource": "*",
  "Condition": {
    "StringNotEquals": {
      "aws:RequestedRegion": [
        "us-east-1",
        "us-west-2"
      ]
    }
  }
}
```

Guardrails for IAM: Organizations service control policies



Expected Regions via Organizations Service Control Policy

```
{
  "Effect": "Deny",
  "Principal": "*",
  "Action": "*",
  "Resource": "*",
  "Condition": {
    "StringNotEquals": {
      "aws:RequestedRegion": [
        "us-east-1",
        "us-west-2"
      ]
    }
  }
}
```

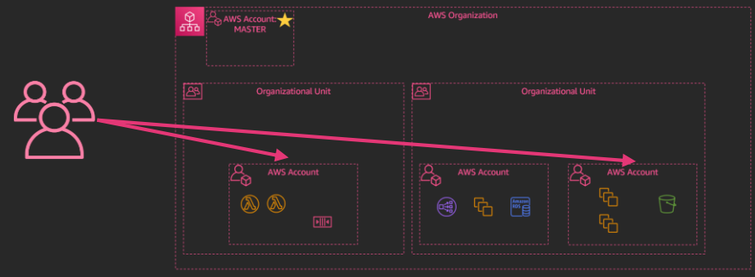
Let’s try SCP

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Statement1",
            "Effect": "Deny",
            "Action": [
                "*"
            ],
            "Resource": [
                "*"
            ],
            "Condition": {
                "StringNotEquals": {
                    "aws:RequestedRegion": [
                        "ap-south-1"
                    ]
                }
            }
        }
    ]
}
```

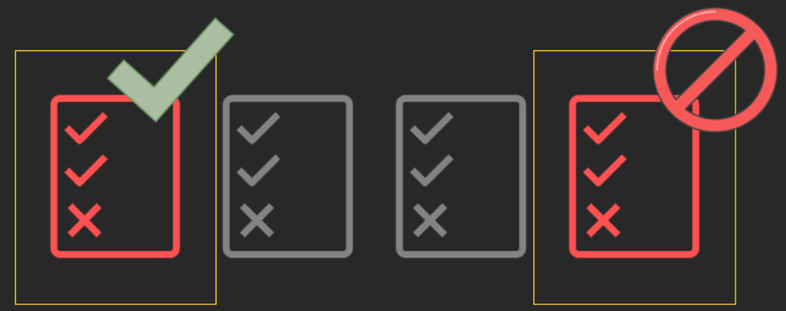
Wrapping up

IAM: Getting in and getting around

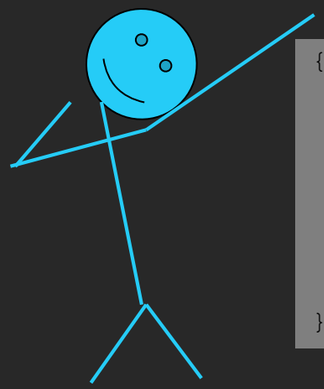
- Patterns for human access to an AWS environment



- How an authorization decision is made in AWS



- How to construct a great IAM policy



```
{
  "Effect": "Allow",
  "Action": [
    "s3:GetObject"
  ],
  "Resource": "arn:aws:s3:::example-bucket/*",
  "Condition": {
    "StringEquals": {
      …
}
```

Thank you!

Chetan Agrawal
agrcheta@amazon.com

Deven Suri
dsuri@amazon.com