aws professional services

AWS Identity & Access Management

AWS Security Workshop

Agenda

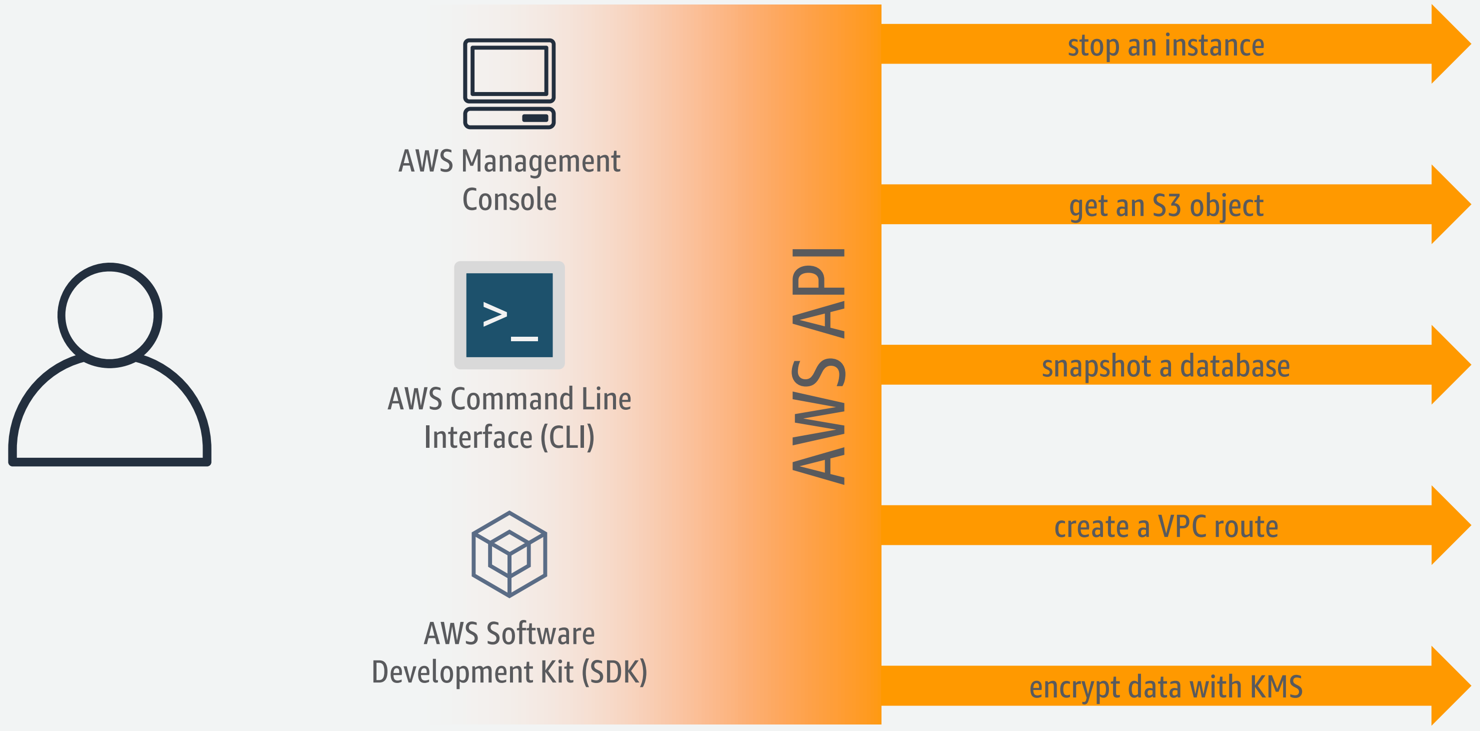
- The AWS APIs
- AWS Identity and Access Management (IAM)
- Authentication
- Authorization
- Cross Account Access

Goals

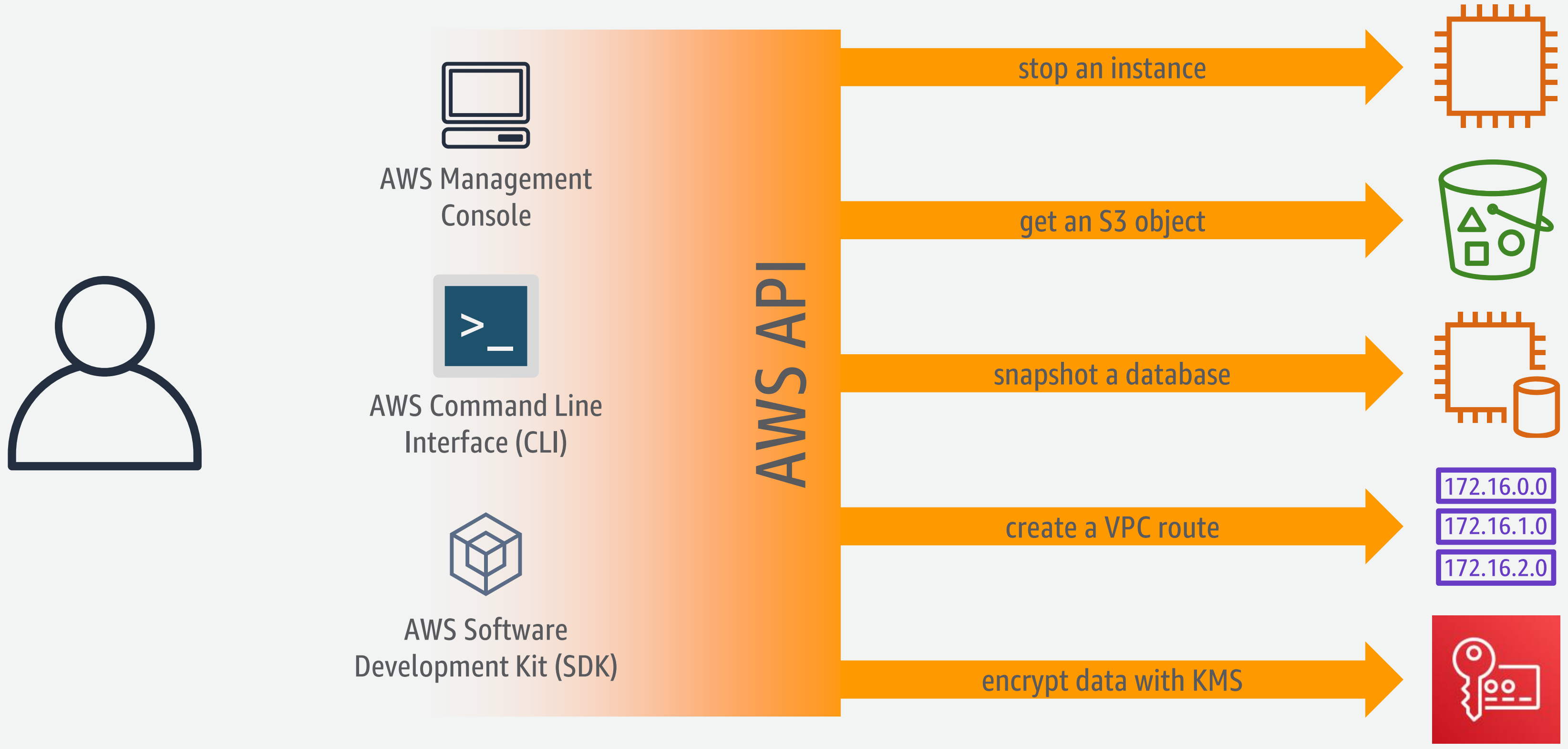
- Learn AWS Identity & Access Management
- Understand when and where to use AWS IAM
- Discovery identity federation options
- Introduction to AWS IAM policy language
- Decision on roles and responsibilities

AWS API Calls

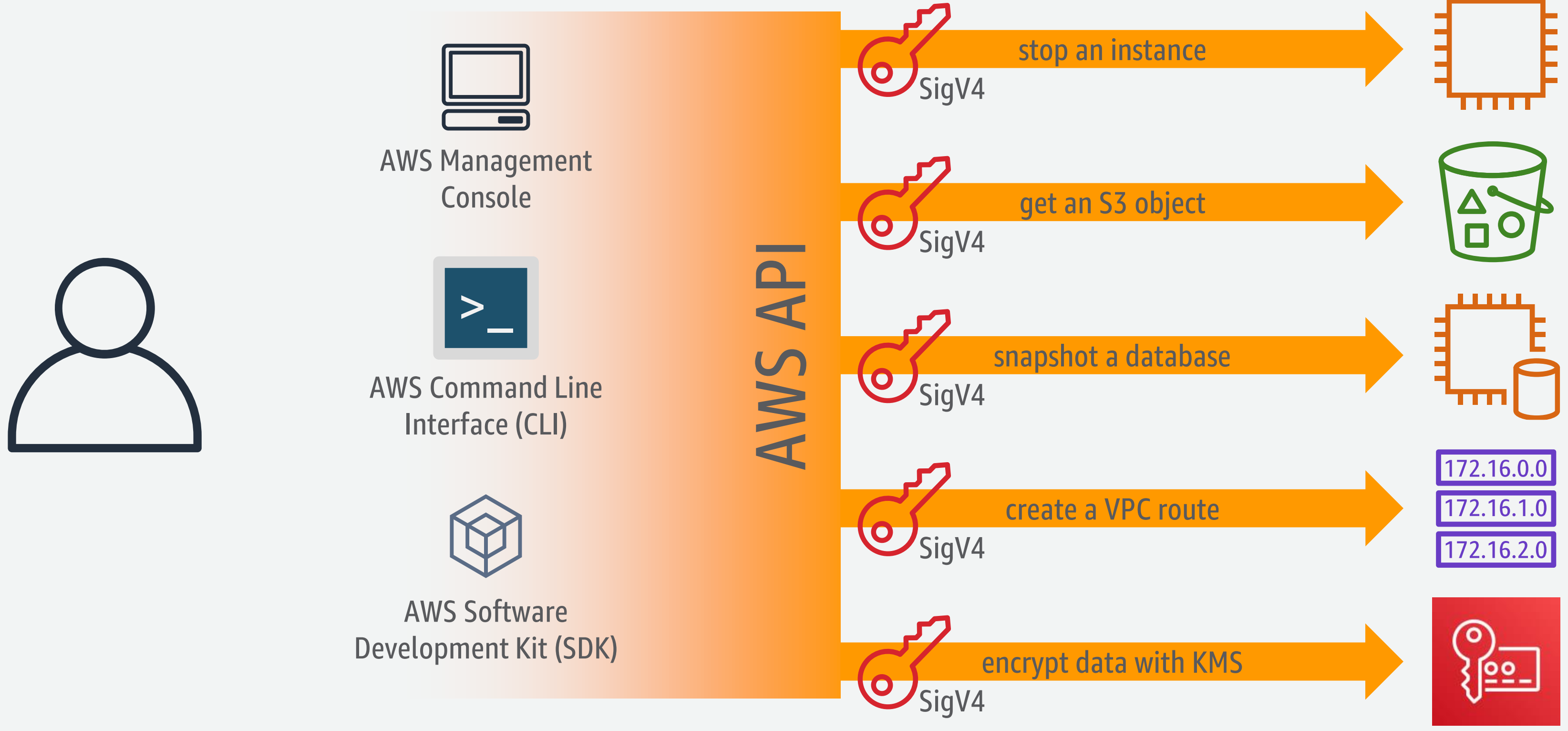
Making API Calls



Making API Calls



Making API Calls



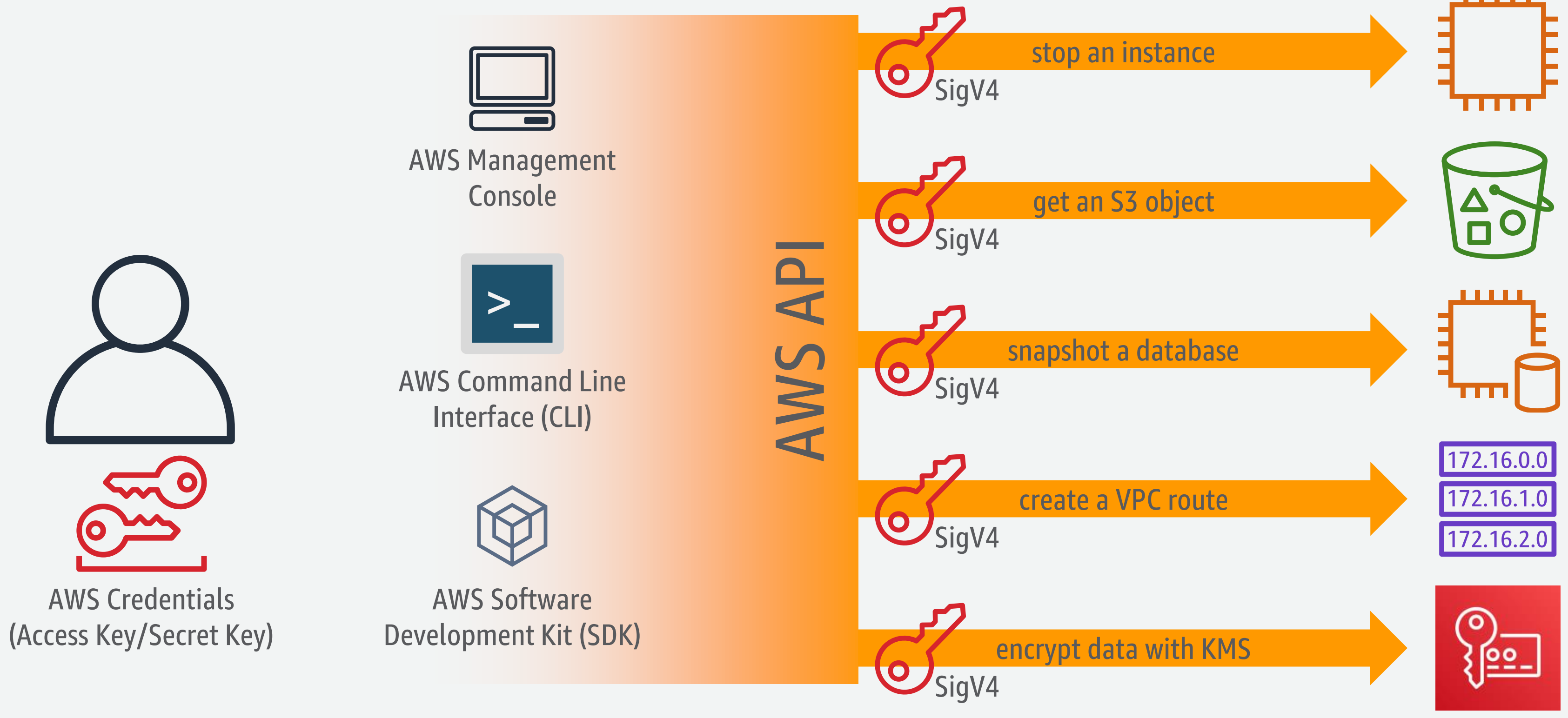
AWS Signature Version 4

- AWS Signature Version 4 is the process to add authentication information to AWS requests.
    - The AWS SDKs or CLI tools will construct, sign, and send requests for you, with the **access keys** you provide.
    - If you are constructing AWS API requests yourself, you will have to include code to sign the requests.

More information can be found here:

http://docs.aws.amazon.com/general/latest/gr/signature-version-4.html

Making API Calls



AWS Identity & Access Management

AWS IAM Concepts

AWS Account

- Centrally controls all the resources
- Protected by the Root Account Credentials
  - Unrestricted and unrestrictable access (root account)
- Pay the bill

Do not use the Root Account Credentials!

Protect the credentials!

AWS IAM Concepts

AWS Accounts

- Strong separation of duties
- Consolidate billing into a single account
- Plan your account strategy in advance (e.g. per function, per criticality, etc.)

AWS IAM Concepts

AWS Resources

Defined uniquely by an **Amazon Resource Name** (ARN)

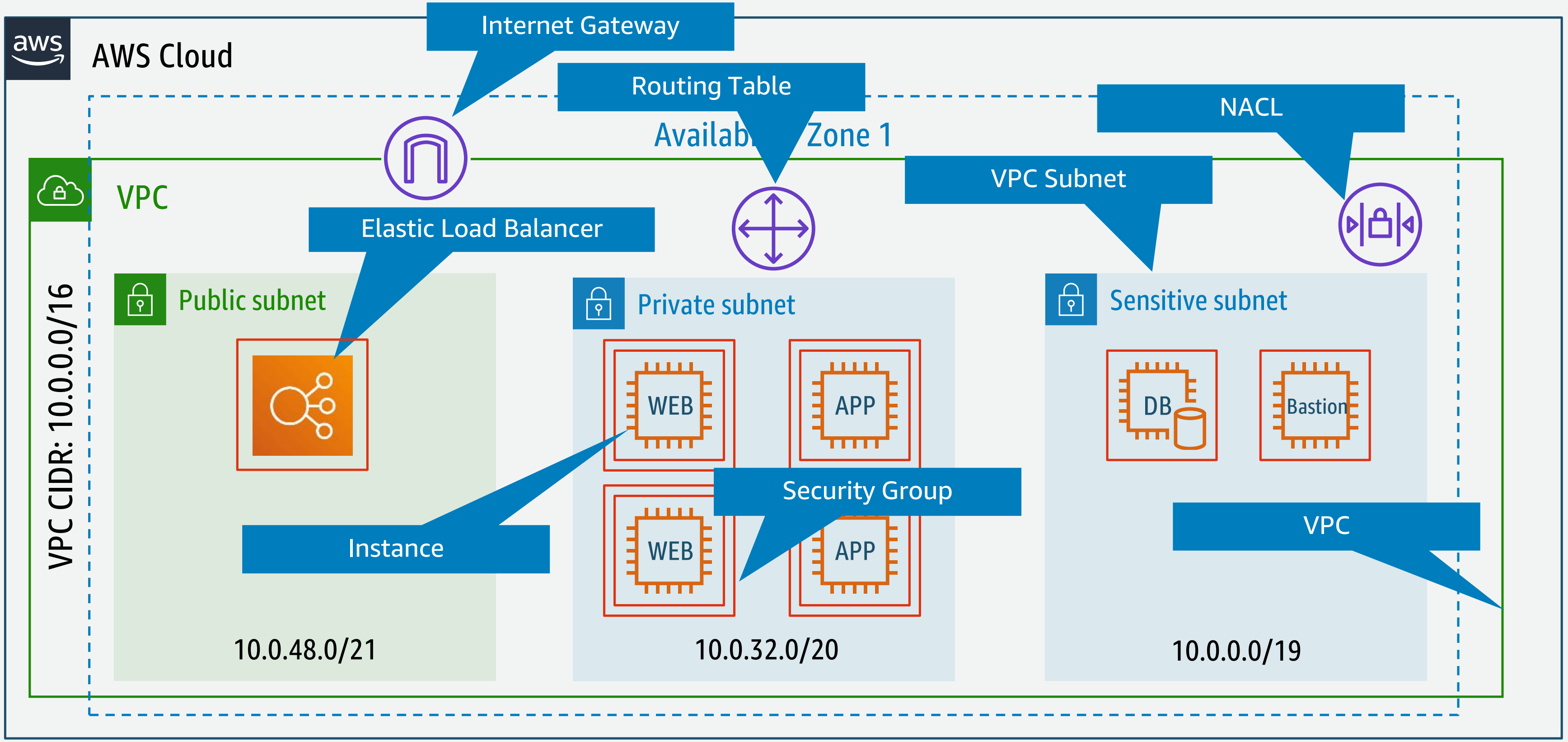
Ex: EC2 instance, DynamoDB table, IAM user, etc.

Not: OS installed on EC2, data inside an EBS volume, etc.

```
arn:aws:service:region:account:resource
```

```
<!- Amazon EC2 instance -->
arn:aws:ec2:us-east-1:123456789012:instance/i-1a2b3c4d

<!-- Amazon RDS tag -->
arn:aws:rds:eu-west-1:123456789012:db:mysql-db

<!-- Amazon S3 all objects in a bucket -->
arn:aws:s3:::my_corporate_bucket/*
```

AWS IAM Concepts

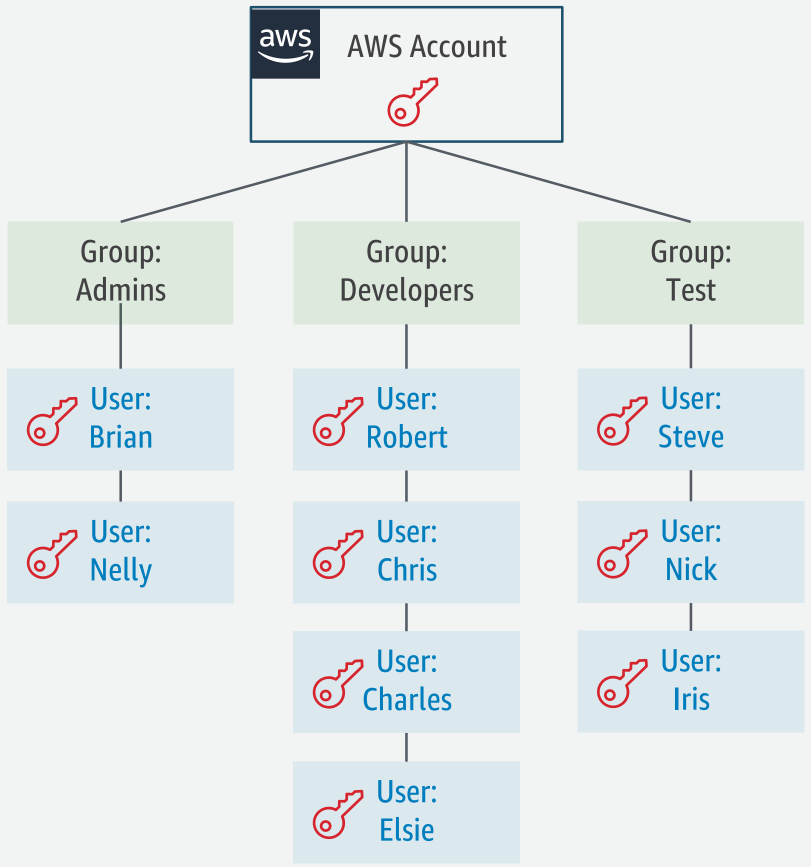


AWS IAM Concepts - Resources

```
<-- S3 Bucket -->
"Resource":"arn:aws:s3:::my_corporate_bucket/*"

<-- SQS queue-->
"Resource":"arn:aws:sqs:us-west-2:123456789012:queue1"

<-- Multiple DynamoDB tables -->
"Resource":["arn:aws:dynamodb:us-west-2:123456789012:table/books_table",
          "arn:aws:dynamodb:us-west-2:123456789012:table/magazines_table"]

<-- All EC2 instances for an account in a region -->
"Resource": "arn:aws:ec2:us-east-1:123456789012:instance/*"
```

AWS IAM Concepts

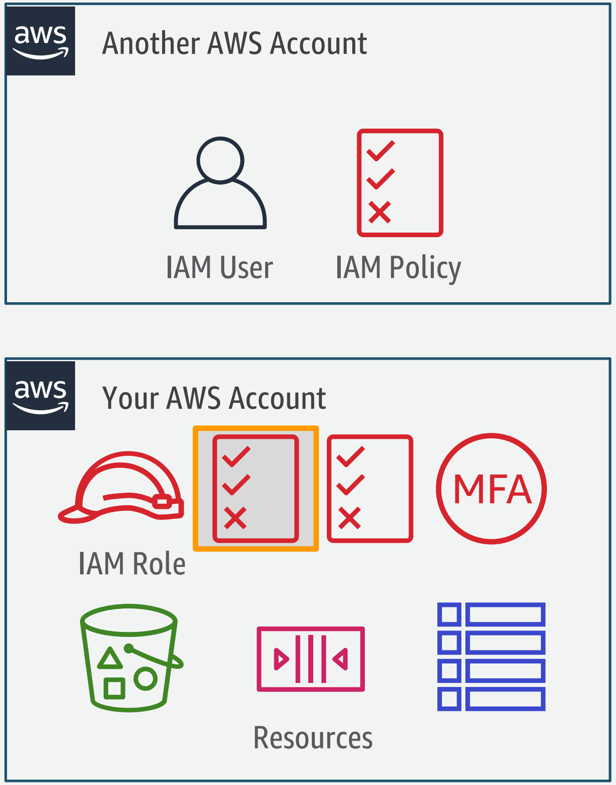
- A username for each user
- Groups to manage multiple users
- Centralised access control
- Optional provisions:
    - Password for console access
    - Policies to control access
    - Use Access Key to sign API calls
    - Multifactor Authentication



AWS IAM Concepts - Roles

- Set of permissions granted to a trusted entity
- Assumed by IAM users, applications or AWS services like EC2
    - Use case:
    - Cross-services
    - Temporary access
    - Cross-account
    - Federation
- Benefits
    - Security: no sharing of secrets
    - Control: revoke access anytime

AWS IAM Concepts - Roles



Create an **IAM Role**

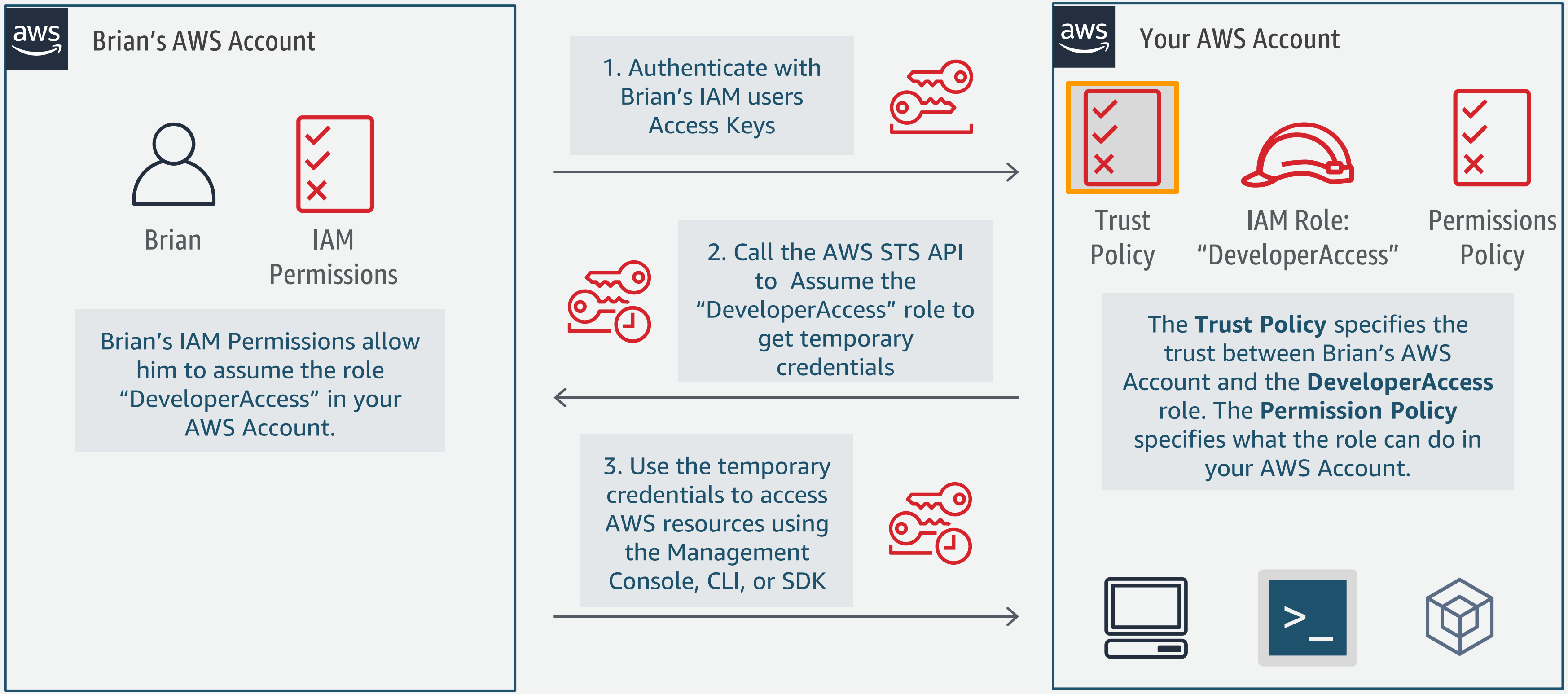
Trust Policy: Trust another AWS Account

Permission Policy: Grant Permissions

Another account's **IAM user** can assume the role if his **Permission Policy** allows him to

Use **MFA** to protect role assumption for privileged access

AWS IAM Concepts - Roles



Brian's IAM Permissions allow him to assume the role "DeveloperAccess" in your AWS Account.

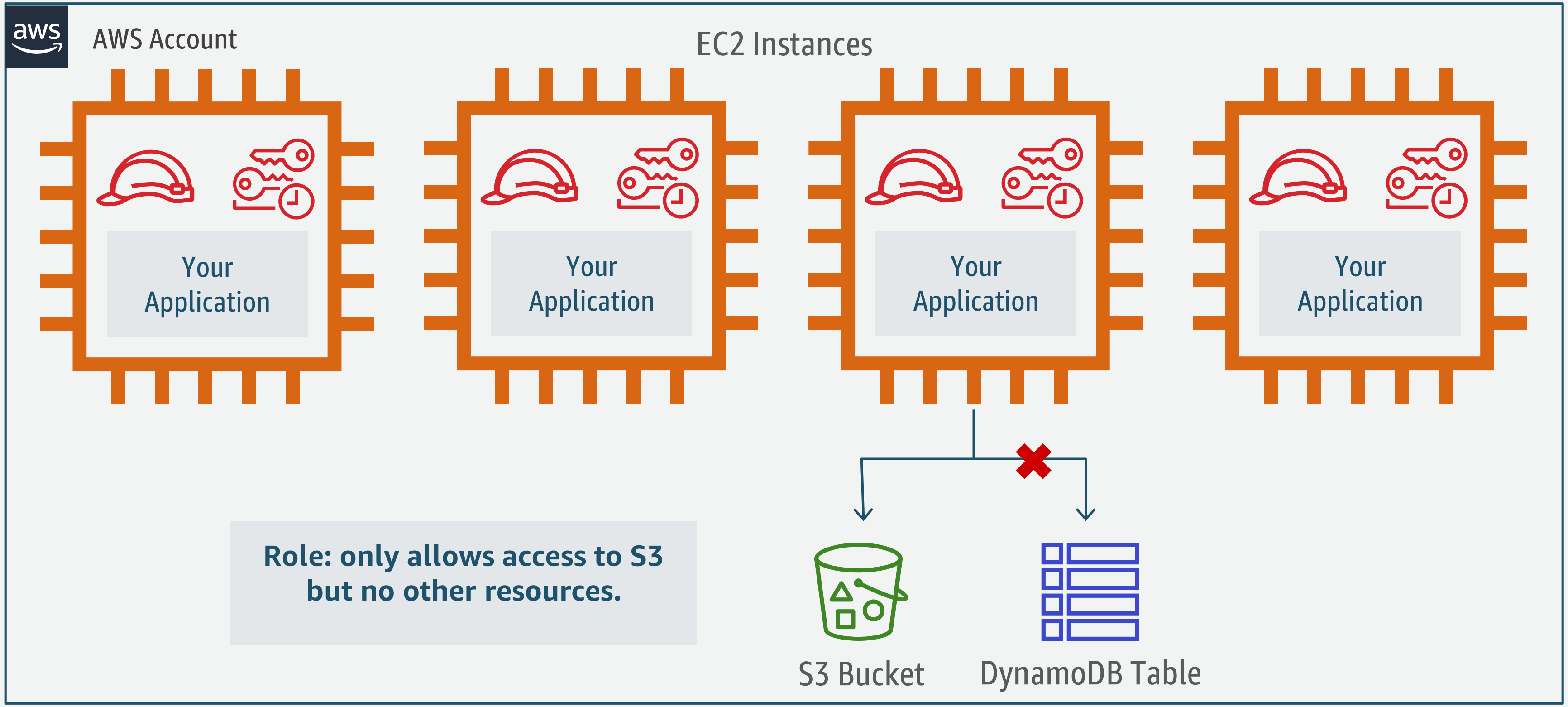
1. Authenticate with Brian's IAM users Access Keys

2. Call the AWS STS API to Assume the "DeveloperAccess" role to get temporary credentials

3. Use the temporary credentials to access AWS resources using the Management Console, CLI, or SDK

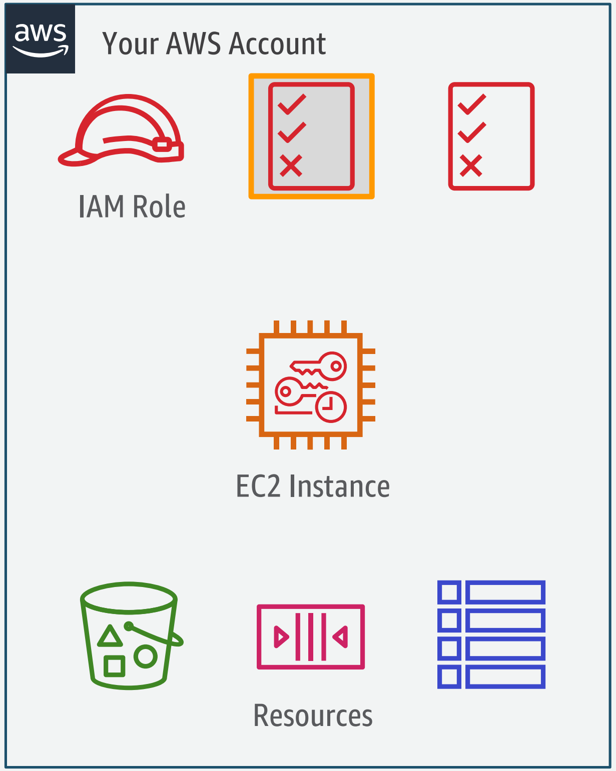
The **Trust Policy** specifies the trust between Brian's AWS Account and the **DeveloperAccess** role. The **Permission Policy** specifies what the role can do in your AWS Account.

AWS IAM Concepts - Roles



Role: only allows access to S3 but no other resources.

AWS IAM Concepts - Roles



Create an **IAM Role**

Trust Policy: Allow EC2 instances to assume this role

Permission Policy: Grant Permissions to resources

Launch an **EC2 Instance** with the **IAM Role** or attach the **IAM Role** to an existing **EC2 Instance**

Temporary Credentials are available on the EC2 Instance through the metadata URL.

AWS IAM Concepts - Roles

Roles

- Set of permissions granted to a trusted entity
- Assumed by IAM users, applications or AWS services like EC2
- Use case:
  - Cross-services
  - Temporary access
  - Cross-account
  - Federation
- Benefits
  - Security: no sharing of secrets
  - Control: revoke access anytime

Analogy

Account Owner ID (Root Account)

- Access to all subscribed services.
- Access to billing.
- Credentials can't be disabled.
- Access to console and APIs.

DO NOT USE after initial set-up

Door Key
Keys to the Kingdom

IAM Users

- Access to specific services.
- Access to console and/or APIs.
- Credentials can be revoked and invalidated.

Employee ID Badge

Temporary Security Credentials / IAM Roles

- Access to specific services.
- Access to console and/or APIs.

Hotel Key

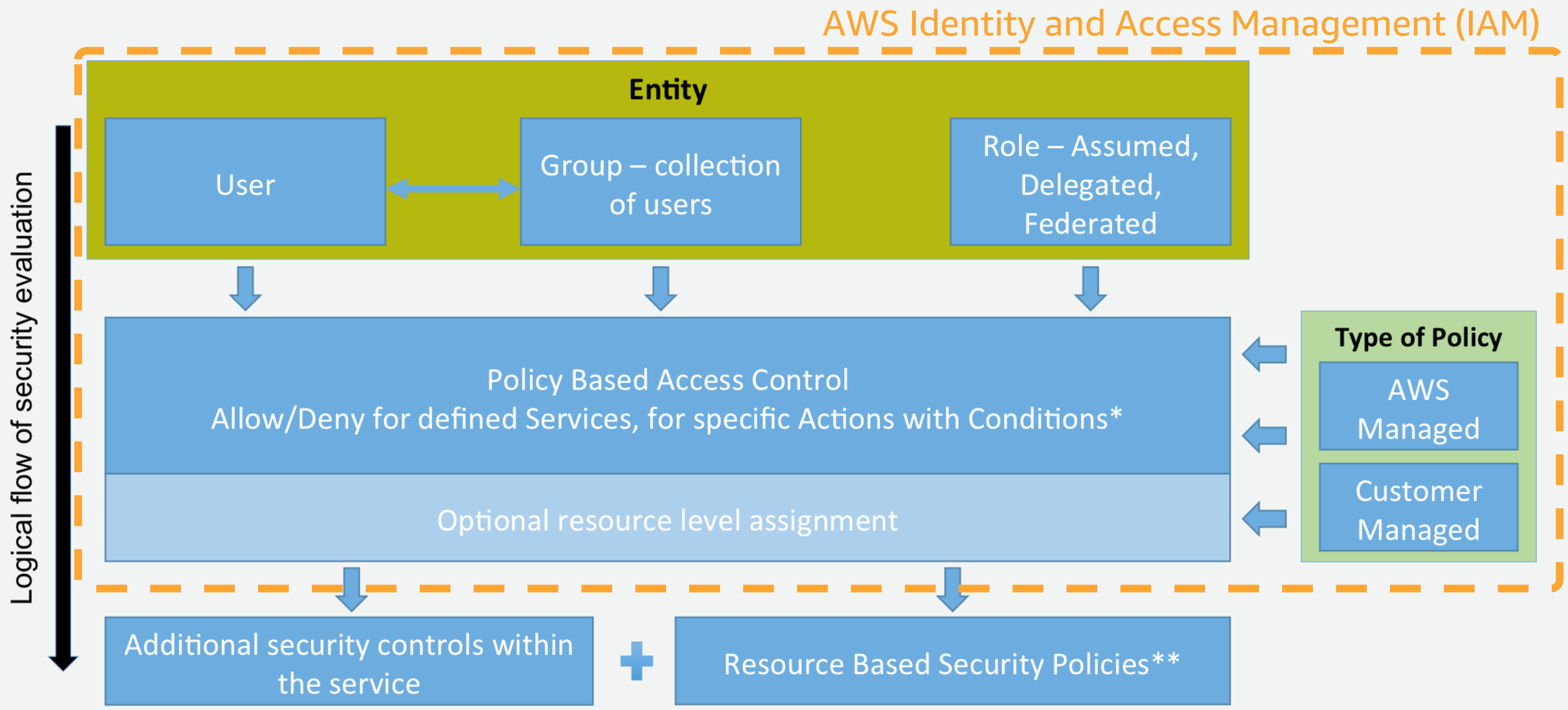
AWS IAM Concepts

- Permissions
  - Authorize (or not) to perform an action
  - Use Policies to grant permission
- Policy
  - Set of instructions which define permission
  - Can be simple or very granular

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[ "s3:ListBucket" ],
         "Resource": “*”
       }
    ]
}
```

aws professional services

Secure Access with IAM


AWS Identity and Access Management (IAM)
Entity
User
Group – collection of users
Role – Assumed, Delegated, Federated
Policy Based Access Control
Allow/Deny for defined Services, for specific Actions with Conditions*
Optional resource level assignment
Type of Policy
AWS Managed
Customer Managed
Additional security controls within the service
Resource Based Security Policies**
Logical flow of security evaluation

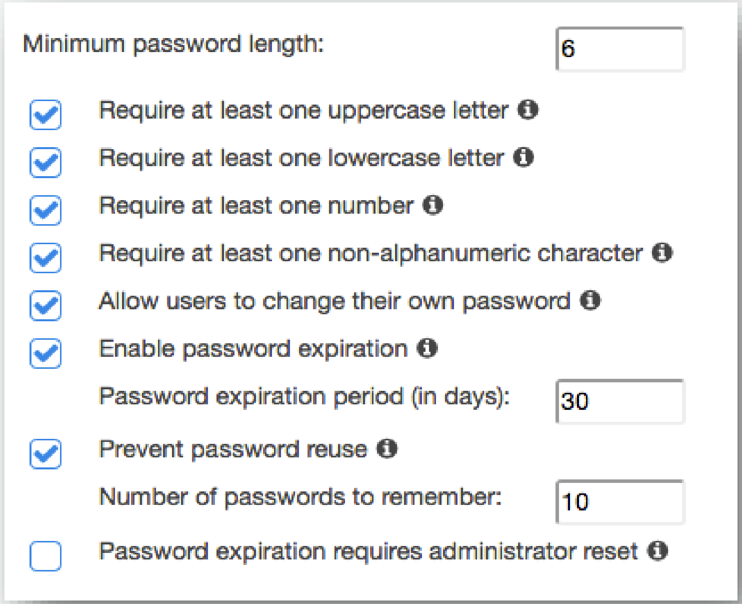
* Service specific conditions also available, e.g. EC2, RDS, KMS, Elastic Beanstalk, etc.
** Available for S3, SQS, SNS, KMS, VPC Endpoint

Authentication

Authentication

Username/Password

- Console access
- Can set an IAM Password Policy



Minimum password length: 6

- [✓] Require at least one uppercase letter
- [✓] Require at least one lowercase letter
- [✓] Require at least one number
- [✓] Require at least one non-alphanumeric character
- [✓] Allow users to change their own password
- [✓] Enable password expiration
  - Password expiration period (in days): 30
- [✓] Prevent password reuse
  - Number of passwords to remember: 10
- [ ] Password expiration requires administrator reset

Authentication

Access Key

- CLI/API access
- Used to sign requests without sending the Secret on the network
- Not retrievable from AWS again – you lose it, generate a new pair

Identifier ACCESS KEY ID

```
AKIAIOSFODNN7EXAMPLE
```

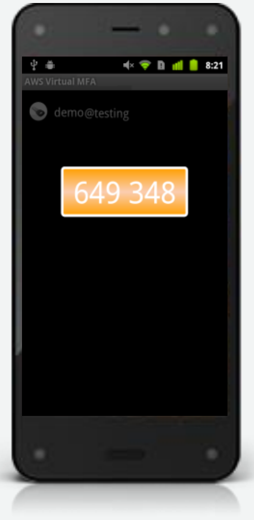
Secret SECRET KEY

```
UtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
```

Authentication

Multifactor Authentication (MFA)

- Helps prevent anyone with unauthorized knowledge of your credentials from impersonating you.
- Virtual, Hardware, U2F
- Works with
  - Root credentials
  - IAM Users
  - Application
- Integrated into
  - AWS API
  - AWS Management Console
  - Key pages on the AWS Portal
  - S3 (Secure Delete)



aws professional services

Keys or Password?

Depends on how your users will access AWS

- Console → Password
- API, CLI, SDK → Access keys

In either case, make sure to rotate credentials regularly

- Use Credential Report to audit credential rotation
- Configure password policy
- Configure policy to allow access key rotation

Authorization

Authorization

Permissions are to specify

Who can access to AWS resources

What action can be performed on those AWS resources

How is it done?

- Organized in **Policies** *(JSON)*

Authorization

Identity-Based Permissions

User: Brian

Can Read, Write, List

On Resource X

Group: Admins

Can Read, Write, List

On Resource XYZ

Group: Developers

Can Read, List

On Resource YZ

Resource-Based Permissions

Resource X

Brian: Read, Write, List
Admins: Read, Write, List
Developers: List

Resource Y

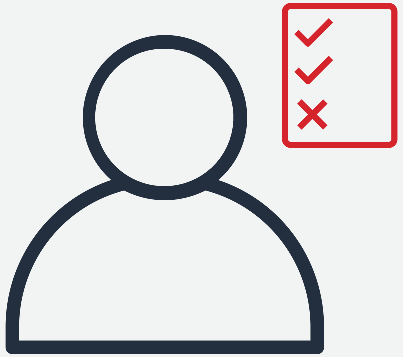
Brian: Read, Write, List
Bob: List
Iris: Read

Resource Z

Admins: Read, Write, List
Developers: Read

Authorization – Identity-Based Permissions

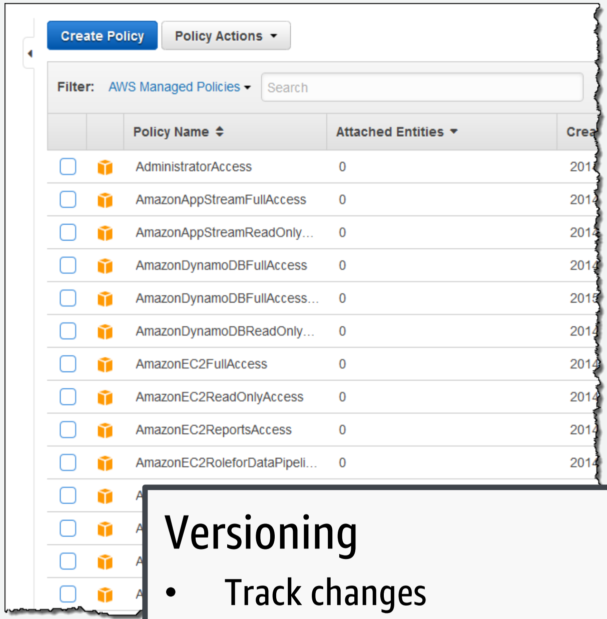
- Are built in Policies
- Attached to an IAM user, group, or role
- Enable you specify what that user, group, or role can do
- User-based policies: **managed** or **inline**



Authorization – Identity-Based Permissions

- Managed Policies
  - AWS managed policies
  - Customer managed policies
  - Reusable
  - Versioning

- Inline Policies
  - Embedded into a user, group or role
  - Disposable / Temporary



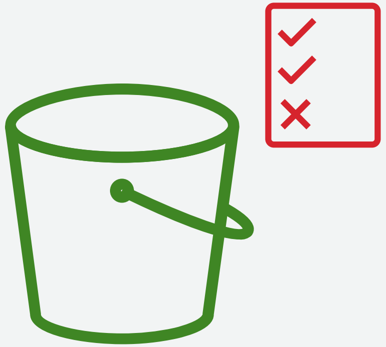
		Policy Name	Attached Entities	Crea
		AdministratorAccess	0	201
		AmazonAppStreamFullAccess	0	2014
		AmazonAppStreamReadOnly...	0	2014
		AmazonDynamoDBFullAccess	0	2014
		AmazonDynamoDBFullAccess...	0	2015
		AmazonDynamoDBReadOnly...	0	2014
		AmazonEC2FullAccess	0	2014
		AmazonEC2ReadOnlyAccess	0	2014
		AmazonEC2ReportsAccess	0	2014
		AmazonEC2RoleforDataPipeli...	0	2014
		A		
		A		
		A		
		A		

Versioning

- Track changes
- Enables rollback
- Keep up to five versions

Authorization – Resource-Based Permissions

- Are built in Policies
- Attached to a resource
- Only available on
  - Amazon S3 buckets
  - Amazon Glacier vaults
  - Amazon SNS topics
  - Amazon SQS queues
  - VPC Endpoints
  - AWS Key Management Service encryption keys
- Specify who has access to the resource and what actions they can perform on it
- Resource-based policies : inline only



Authorization – Resource-Based Permissions

- JSON-formatted documents
- Contain a statement (permissions) that specifies:
  - Which actions a principal can perform
  - Which resources can be accessed

```
{
 "Statement":[{
   "Effect":"effect",
   "Principal":"principal",
   "Action":"action",
   "Resource":"arn",
   "Condition":{
     "condition":{
       "key":"value" }
     }
   }
  ]
}
```

Principal

Action

Resource

Condition

You can have multiple statements and each statement is comprised of PARC.

Authorization – Policies

Identity-Based versus Resource-Based

```
{
 "Statement":[{
   "Effect":"effect",
   "Action":"action",
   "Resource":"arn",
   "Condition":{
     "condition":{
       "key":"value" }
     }
   }
  ]
 }
```

Identity-based Policy

```
{
 "Statement":[{
   "Effect":"effect",
   "Principal":"principal",
   "Action":"action",
   "Resource":"arn",
   "Condition":{
     "condition":{
       "key":"value" }
     }
   }
  ]
 }
```

Resource-based Policy

Principal

Action

Resource

Condition

Authorization – Policies

```
{
   "Version": "2012-10-17",
   "Statement": {
     "Effect": "Allow",
     "Action": "s3:ListBucket",
     "Resource": "arn:aws:s3:::example_bucket”
  }
}
```

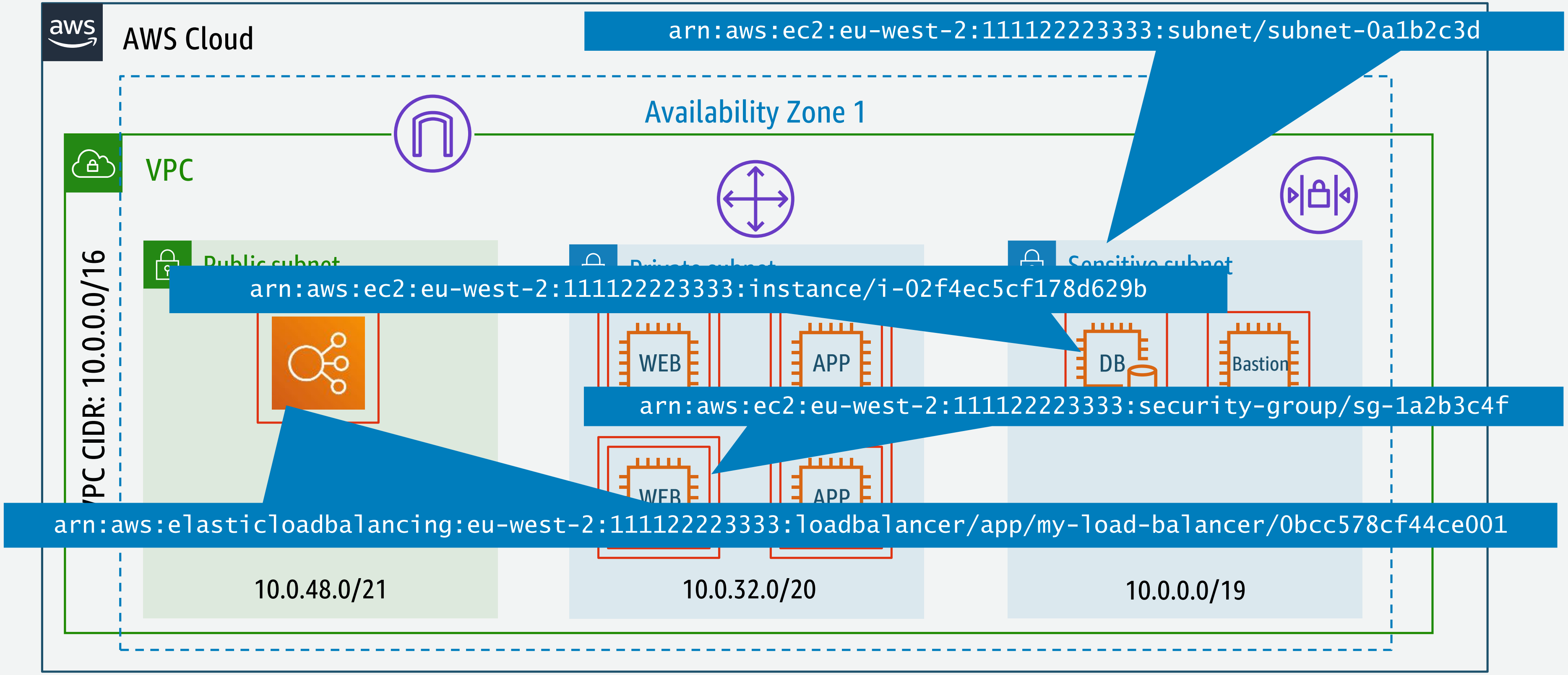
You can attach this policy to an IAM user or group. If that's the only policy for the user or group, the user or group is allowed to perform only this one action (ListBucket) on one Amazon S3 bucket (example_bucket).

Authorization – Amazon Resource Name (ARN)

```
arn:partition:service:region:account-id:resource
arn:partition:service:region:account-id:resourcetype/resource
arn:partition:service:region:account-id:resourcetype:resource
```

- **Partition**: For standard AWS regions, the partition is `aws`.
- **Service**: The service namespace (example: `iam`)
- **Region**: The region the resource resides in (example: `us-west-2`)
- Note that the ARNs for some resources do not require a region, so this component might be omitted.
- **Account-id**: Example: `123456789012`

Authorization – Amazon Resource Name (ARN)



arn:aws:ec2:eu-west-2:111122223333:subnet/subnet-0a1b2c3d

arn:aws:ec2:eu-west-2:111122223333:instance/i-02f4ec5cf178d629b

arn:aws:ec2:eu-west-2:111122223333:security-group/sg-1a2b3c4f

arn:aws:elasticloadbalancing:eu-west-2:111122223333:loadbalancer/app/my-load-balancer/0bcc578cf44ce001

Authorization – Principals

```
<!-- Everyone (anonymous users) -->
"Principal":"AWS":"*.*"

<!-- Specific account or accounts -->
"Principal":{"AWS":"arn:aws:iam::123456789012:root" }
"Principal":{"AWS":"123456789012"}

<!-- Individual IAM user -->
"Principal":"AWS":"arn:aws:iam::123456789012:user/username"

<!-- Federated user (using web identity federation) -->
"Principal":{"Federated":"www.amazon.com"}
"Principal":{"Federated":"graph.facebook.com"}
"Principal":{"Federated":"accounts.google.com"}

<!-- Specific role -->
"Principal":{"AWS":"arn:aws:iam::123456789012:role/rolename"}

<!-- Specific service -->
 "Principal":{"Service":"ec2.amazonaws.com"}
```

Replace with your AWS account number

Authorization – Actions

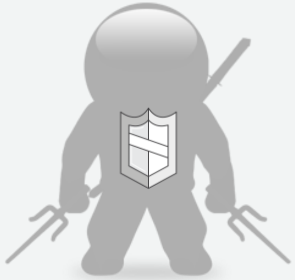
```
<!-- EC2 action -->
"Action":"ec2:StartInstances"

<!-- IAM action -->
"Action":"iam:ChangePassword"

<!-- S3 action -->
"Action":"s3:GetObject"

<!-- Specify multiple values for the Action element -->
"Action":["sqs:SendMessage","sqs:ReceiveMessage"]

<-- Use wildcards (* or ?) as part of the action name. This would cover
Create/Delete/List/Update -->
"Action":"iam:*AccessKey*"
```

Authorization – NotAction

```
{
  "Version": "2012-10-17",
  "Statement": [ {
      "Effect": "Allow",
      "NotAction": "iam:*",
      "Resource": "*"
    }
  ]
}
```

```
{
  "Version": "2012-10-17",
  "Statement": [{
      "Effect": "Allow",
      "Action": "*",
      "Resource": "*"
    },
    {
      "Effect": "Deny",
      "Action": "iam:*",
      "Resource": "*"
    }
  ]
}
```

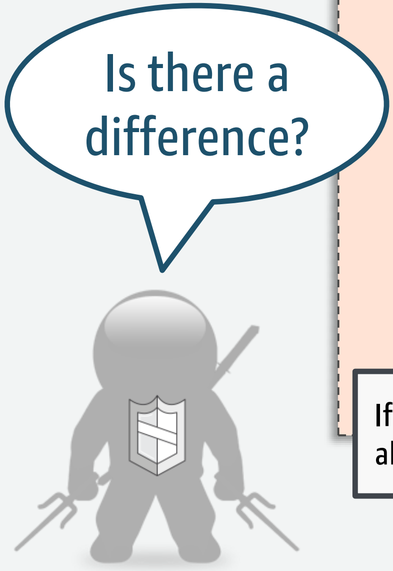
Is there a difference?



Authorization – NotAction

```
{
  "Version": "2012-10-17",
  "Statement": [ {
      "Effect": "Allow",
      "NotAction": "iam:*",
      "Resource": "*"
    }
  ]
}
```

This is not a **Deny**. A user could still have a separate policy that grants **IAM:***



```
{
  "Version": "2012-10-17",
  "Statement": [{
      "Effect": "Allow",
      "Action": "*",
      "Resource": "*"
    },
    {
      "Effect": "Deny",
      "Action": "iam:*",
      "Resource": "*"
    }
  ]
```

If you want to prevent the user from ever being able to call IAM APIs, use an **explicit deny**.

Authorization – Conditions

Restricting access to a time frame and IP address

```
    "Condition" :  {
      "DateGreaterThan" : {"aws:CurrentTime" : "2015-10-08T12:00:00Z"},
AND   "DateLessThan": {"aws:CurrentTime" : "2015-10-08T15:00:00Z"},
      "IpAddress" : {"aws:SourceIp" : ["192.0.2.0/24", "203.0.113.0/24"]}
    }
                                                     OR
```

Allows a user to access a resource under the following conditions:

- The time is after 12:00 P.M. on 10/8/2015 **AND**
- The time is before 3:00 P.M. on 10/8/2015 **AND**
- The request comes from an IP address in the 192.0.2.0 /24 **OR** 203.0.113.0 /24 range

All of these conditions must be met in order for the statement to evaluate to TRUE.

aws professional services

Authorization – Conditions

Examples:

- aws:CurrentTime
- aws:EpochTime
- aws:MultiFactorAuthAge
- aws:MultiFactorAuthPresent
- aws:SecureTransport
- aws:UserAgent
- aws:PrincipalOrgID
- aws:PrincipalType
- aws:Referer
- aws:RequestedRegion
- aws:RequestTag/*tag-key*
- aws:ResourceTag/*tag-key*
- aws:SourceAccount
- aws:SourceArn
- aws:SourceIp
- aws:SourceVpc
- aws:SourceVpce
- aws:TagKeys
- aws:TokenIssueTime
- aws:userid
- aws:username

Authorization – Policy Variables

- Predefined variables based on service request context
  - **Global** keys (`aws:SourceIP`, `aws:MultiFactorAuthPresent`, etc.)
  - **Principal-specific** keys (`aws:username`, `aws:userid`, `aws:PrincipalType`)
  - **Provider-specific** keys (`graph.facebook.com:id`, `www.amazon.com:user_id`)
  - **SAML** keys (`saml:cn`, `saml:edupersonassurance`)
  - See documentation for service-specific variables
- Benefits
  - Simplify policy management
  - Reduce the need for hard-coded, user-specific policies

Authorization – Policy Variables

Applicable to Brian:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": ["iam:*AccessKey*"],
      "Effect": "Allow",
      "Resource": ["arn:aws:iam::123456789012:user/Brian"]
    }
  ]
}
```

Applicable to all users:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": ["iam:*AccessKey*"],
      "Effect": "Allow",
      "Resource": ["arn:aws:iam::123456789012:user/${aws:username}"]
    }
  ]
}
```

Authorization – Policy Variables

Grants a user access to a home directory in S3 that can be accessed programmatically

```
{
  "Version": "2012-10-17",
  "Statement": [{
     "Effect": "Allow",
     "Action": ["s3:ListBucket"],
     "Resource": ["arn:aws:s3:::myBucket"],
     "Condition":
                  {"StringLike":
                     {"s3:prefix":["home/${aws:username}/*"]}
        }
    },
    {
      "Effect":"Allow",
      "Action":["s3:*"],
      "Resource": ["arn:aws:s3:::myBucket/home/${aws:username}",
                   "arn:aws:s3:::myBucket/home/${aws:username}/*"]
    }
  ]
}
```

Version is required

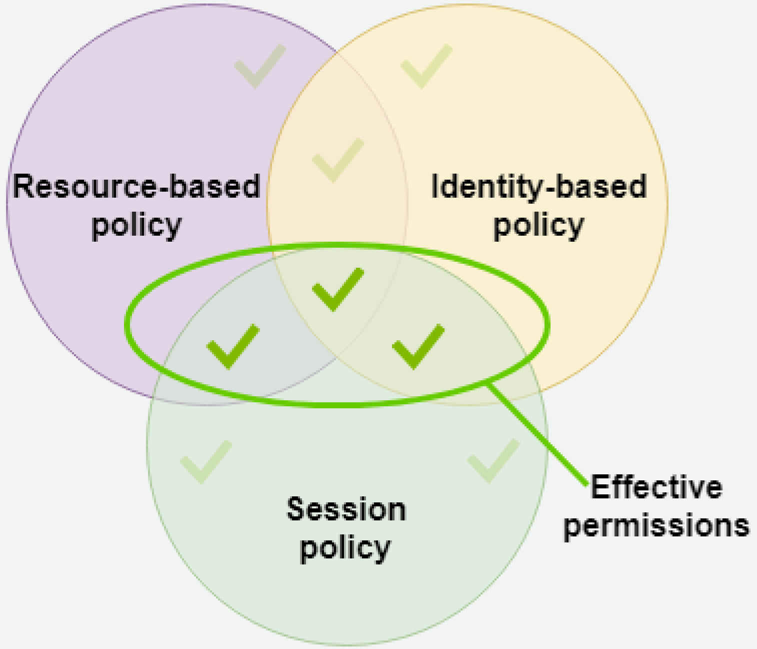
Variable in conditions

Variable in resource ARNs

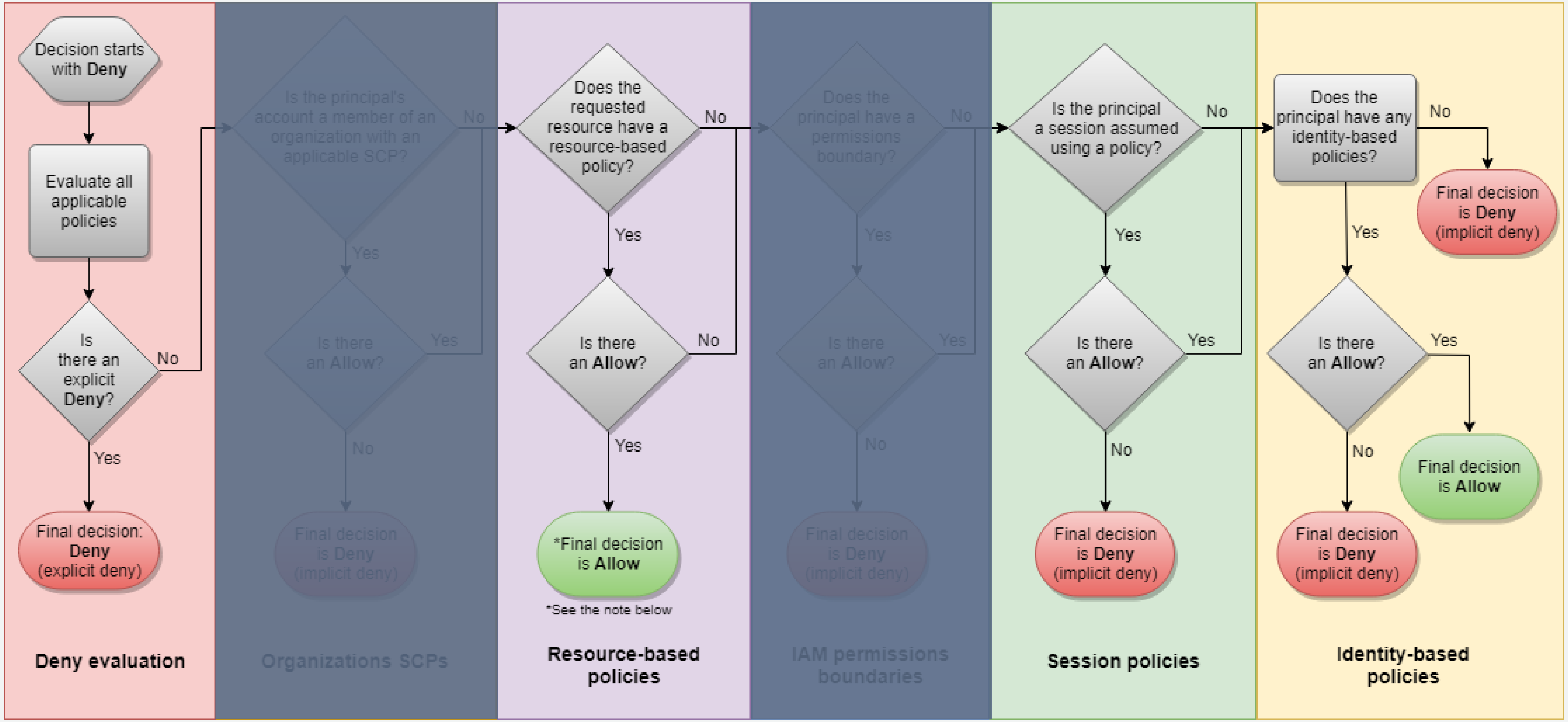
Authorization – Session Policies

Session Policies

- Can be passed as a parameter for programmatically created sessions.
- Effective permissions come from:
    - Identity-based permissions
    - Resource-based permissions
    - Session-based based permissions



IAM Evaluation Logic



Decision starts with Deny
Evaluate all applicable policies
Is there an explicit Deny?
No
Yes
Final decision: Deny (explicit deny)
Deny evaluation
Is the principal's account a member of an organization with an applicable SCP?
No
Yes
Is there an Allow?
Yes
No
Final decision is Deny (implicit deny)
Organizations SCPs
Does the requested resource have a resource-based policy?
No
Yes
Is there an Allow?
No
Yes
*Final decision is Allow
*See the note below
Resource-based policies
Does the principal have a permissions boundary?
No
Yes
Is there an Allow?
Yes
No
Final decision is Deny (implicit deny)
IAM permissions boundaries
Is the principal a session assumed using a policy?
No
Yes
Is there an Allow?
Yes
No
Final decision is Deny (implicit deny)
Session policies
Does the principal have any identity-based policies?
No
Final decision is Deny (implicit deny)
Yes
Is there an Allow?
Yes
Final decision is Allow
No
Final decision is Deny (implicit deny)
Identity-based policies

AWS Organizations

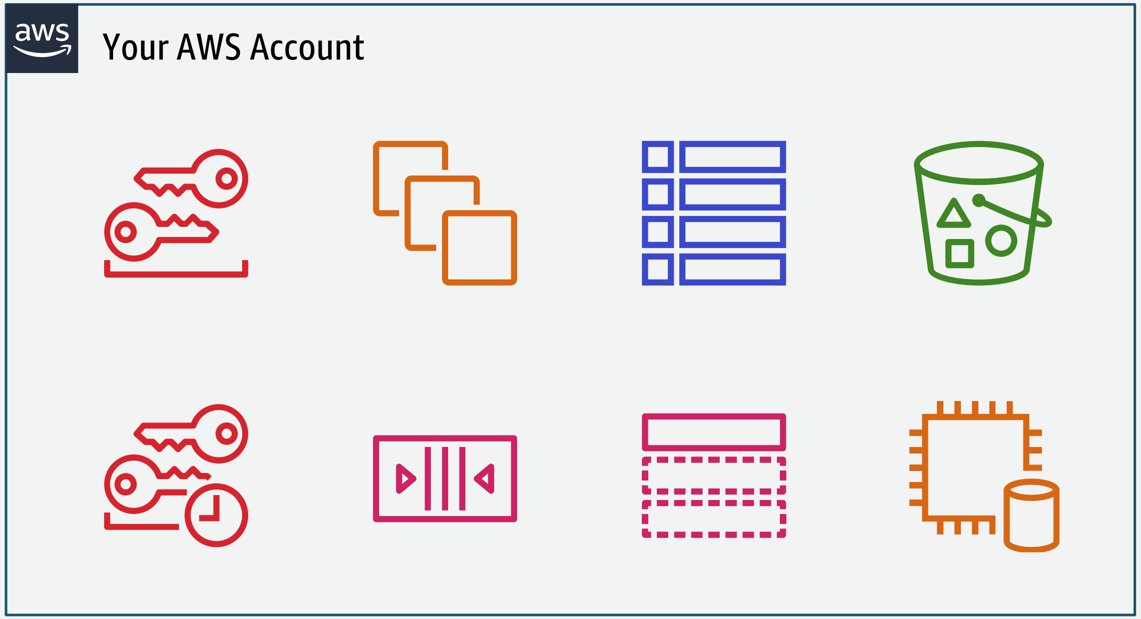
AWS Organizations

- Manage/control multiple AWS accounts centrally
- Enable multi-account functionality for AWS services

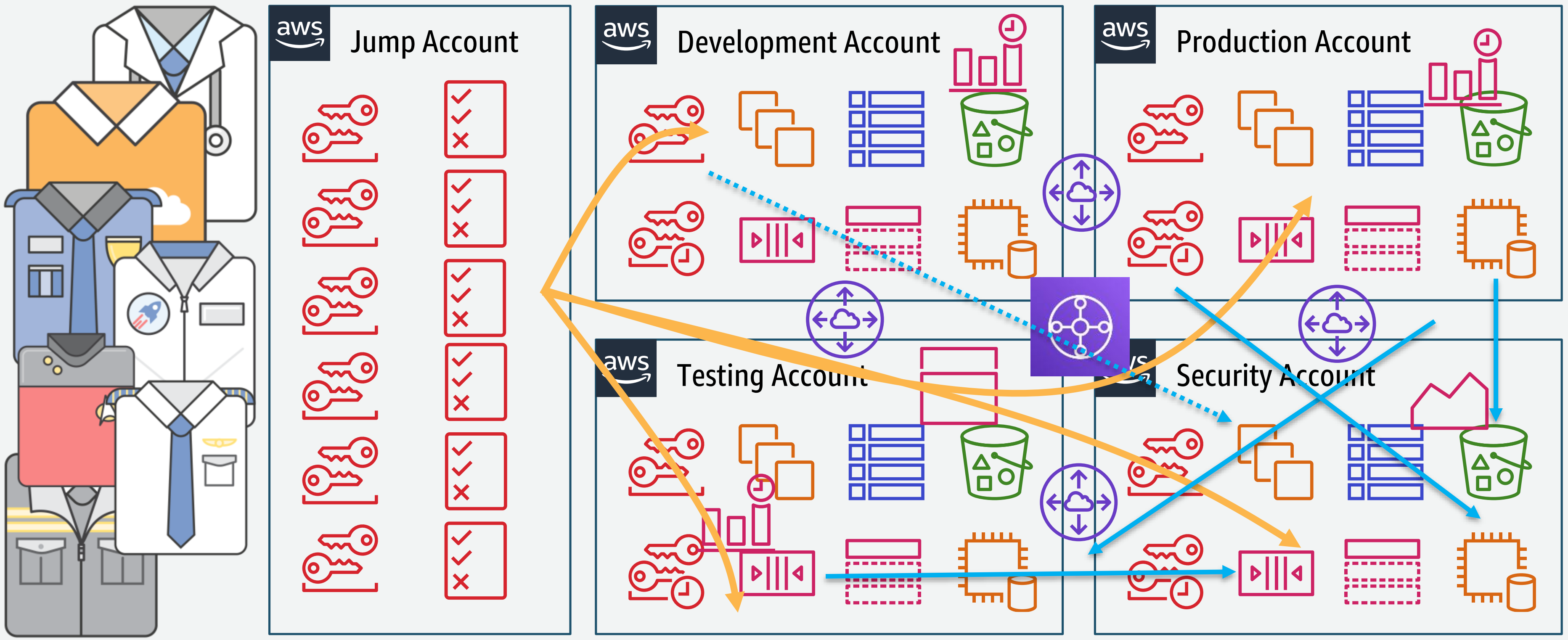
Key features:

- Simplified creation of new AWS accounts
- Logically group AWS accounts for management convenience
- Apply organizational policies to control AWS services
- Consolidate billing and usage across all accounts into one bill

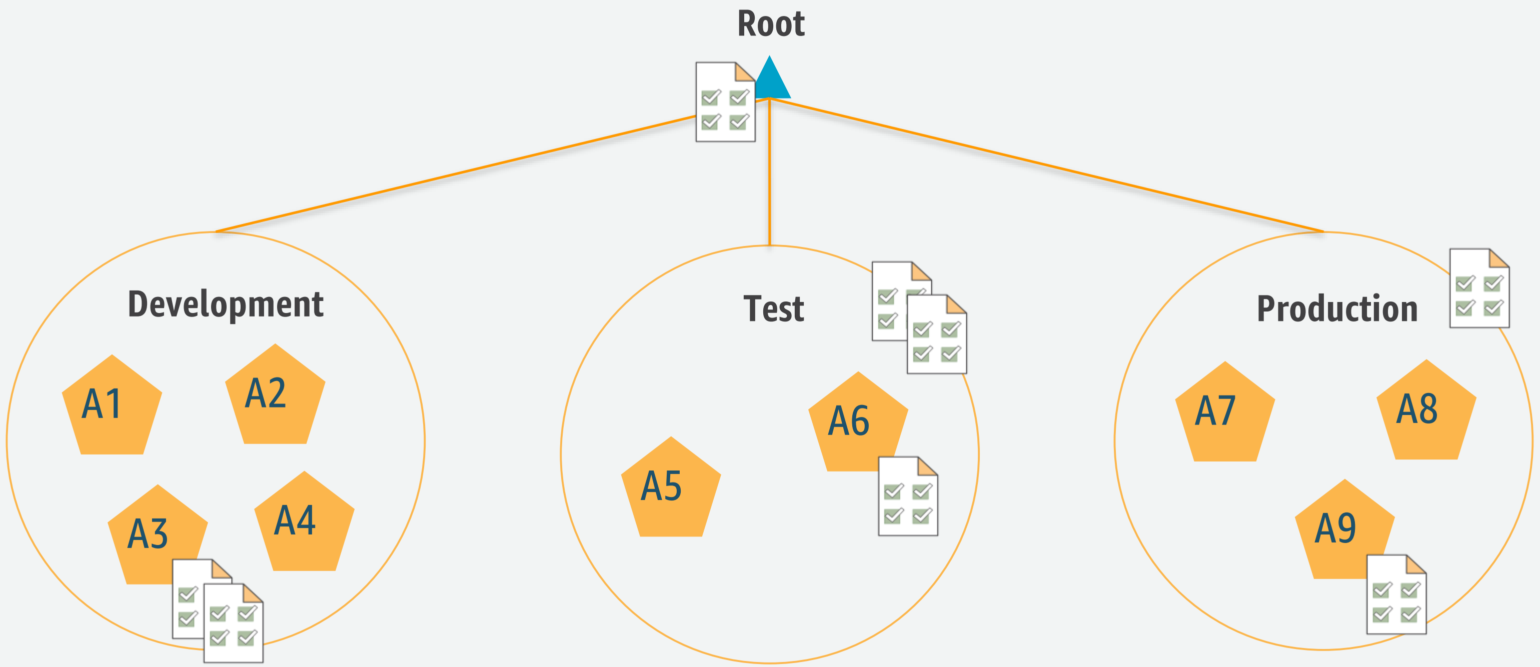
AWS Organizations – In the beginning…



AWS Organizations – Today

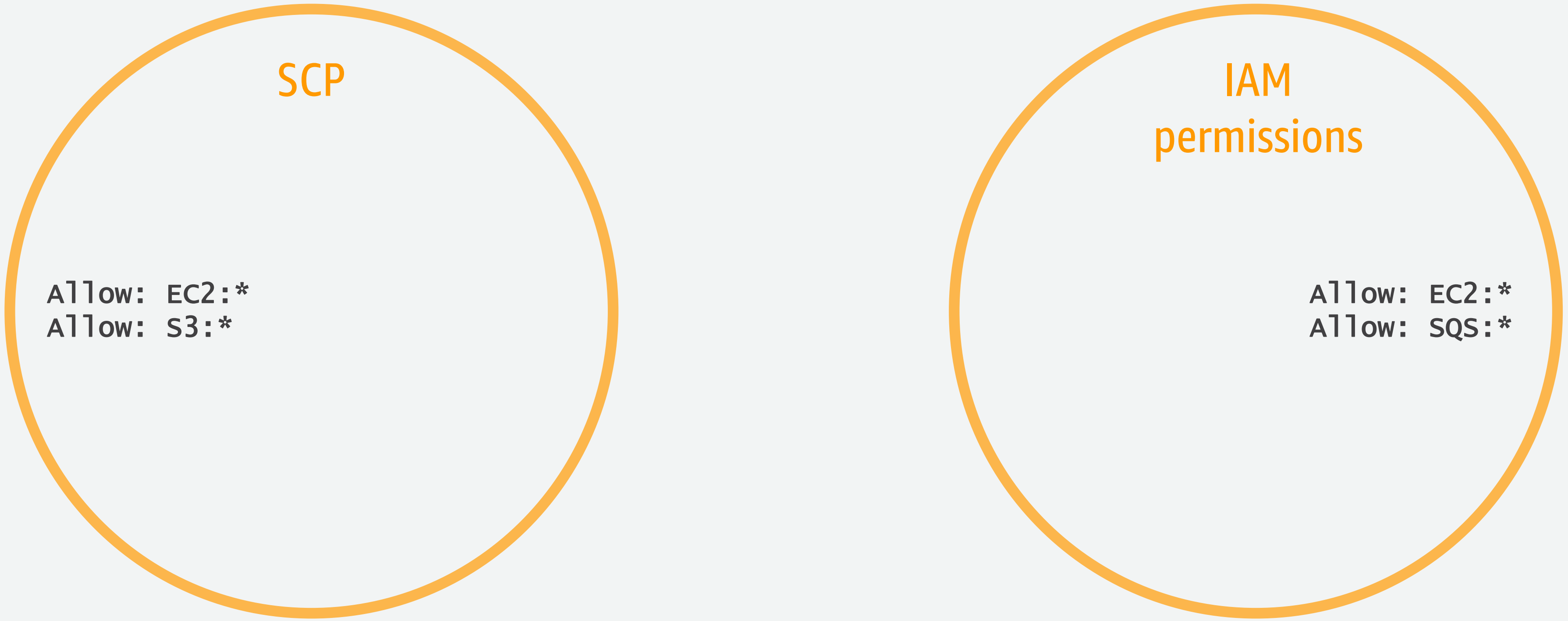


AWS Organizations – Hierarchy and Policies

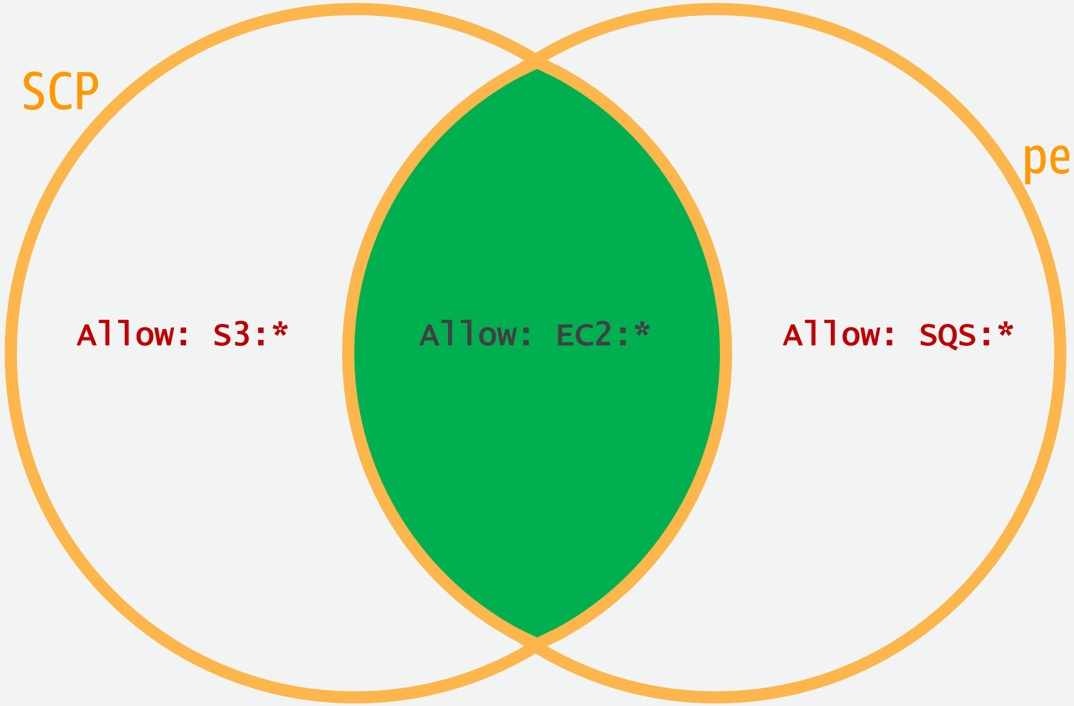


Service Control Policies use the IAM policy language

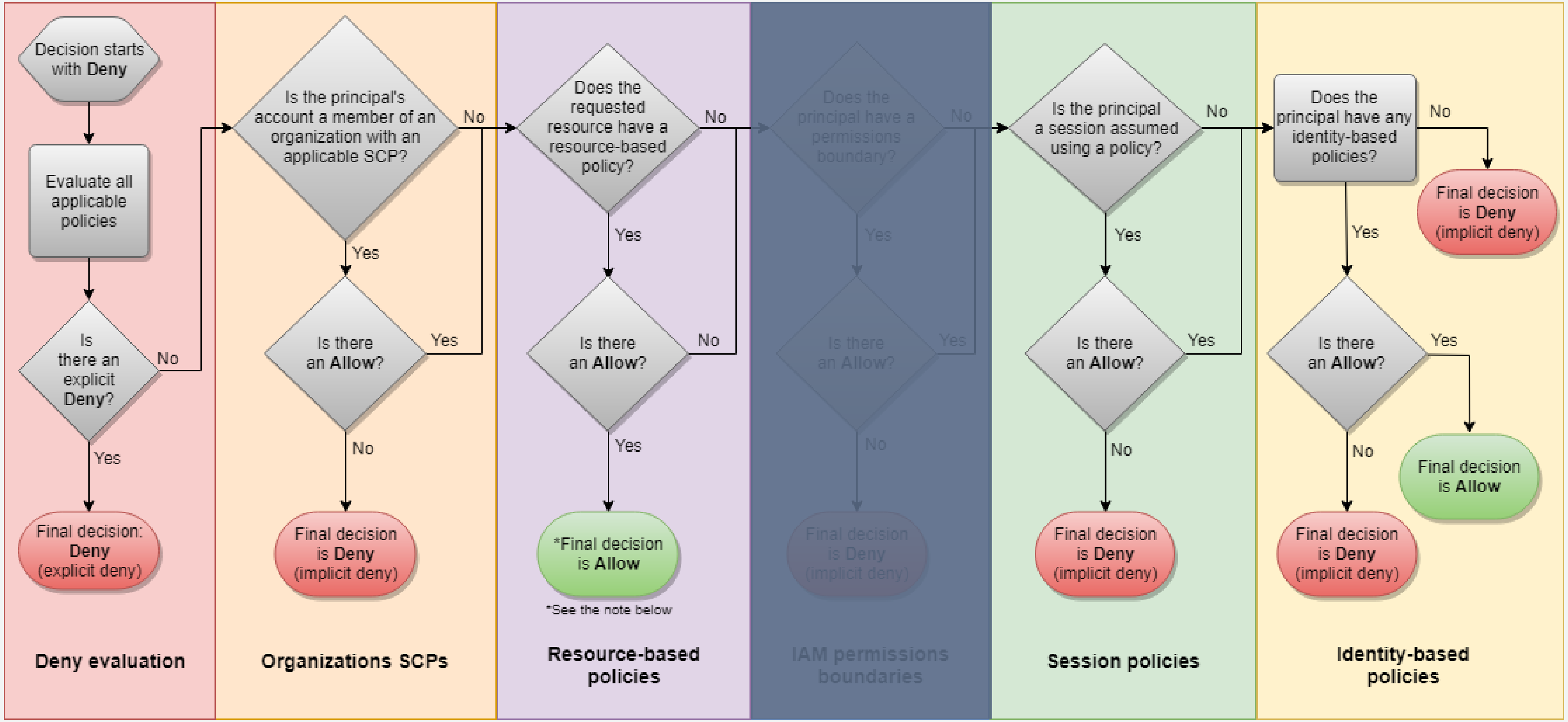
AWS Organizations – Hierarchy and Policies



AWS Organizations – Hierarchy and Policies



IAM Evaluation Logic



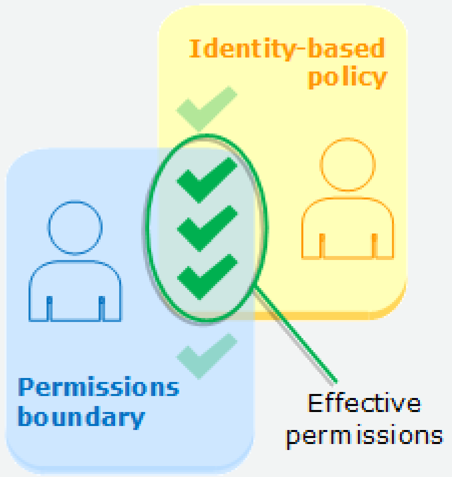
Decision starts with Deny
Evaluate all applicable policies
Is there an explicit Deny?
No
Yes
Final decision: Deny (explicit deny)
Deny evaluation
Is the principal's account a member of an organization with an applicable SCP?
No
Yes
Is there an Allow?
Yes
No
Final decision is Deny (implicit deny)
Organizations SCPs
Does the requested resource have a resource-based policy?
No
Yes
Is there an Allow?
No
Yes
*Final decision is Allow
*See the note below
Resource-based policies
Does the principal have a permissions boundary?
No
Yes
Is there an Allow?
Yes
No
Final decision is Deny (implicit deny)
IAM permissions boundaries
Is the principal a session assumed using a policy?
No
Yes
Is there an Allow?
Yes
No
Final decision is Deny (implicit deny)
Session policies
Does the principal have any identity-based policies?
No
Final decision is Deny (implicit deny)
Yes
Is there an Allow?
Yes
Final decision is Allow
No
Final decision is Deny (implicit deny)
Identity-based policies

Permission Boundaries

Permissions Boundaries

Permission Boundaries

- limit the maximum permissions that a principal can have
- Can be used to delegate IAM tasks but limit the permissions granted to the IAM principals they create
- Use the same policy language as regular IAM policies



Permissions Boundaries – Use Case

- **IAM Administrator**: Principal responsible for provisioning user, roles, and policies for the enterprise.
- **Delegated IAM Administrator**: Principal with delegated responsibility to provision users, roles, and policies for their business unit, team, or workload.
- **Business unit, team, or workload member**: Principal interacting with AWS to accomplish their business unit, team, or workload goals.

Permissions Boundaries – Tasks

1. IAM Administrator creates MyAppPermissionsBoundary.
2. IAM Administrator creates Delegated IAM Administrator role.
3. IAM Administrator creates Delegated IAM Administrator Permissions Boundary and Permissions Policy.
4. Attach permissions boundary and policy to role.

Permissions Boundaries – How does it work?

```
"Effect": "Allow",
            "Action": [
                "iam:DetachRolePolicy",
                "iam:DeleteRolePolicy",
                "iam:PutRolePermissionsBoundary",
                "iam:CreateRole",
                "iam:AttachRolePolicy",
                "iam:PutRolePolicy"
            ],
            "Resource": "arn:aws:iam:::role/apps/my_app/*",
            "Condition": {
                "StringEquals": {

"iam:PermissionsBoundary":"arn:aws:iam::<account>:policy/MyAppPermissionsBoundary"
                }
            }
```

Permissions Boundaries – How do I enforce?

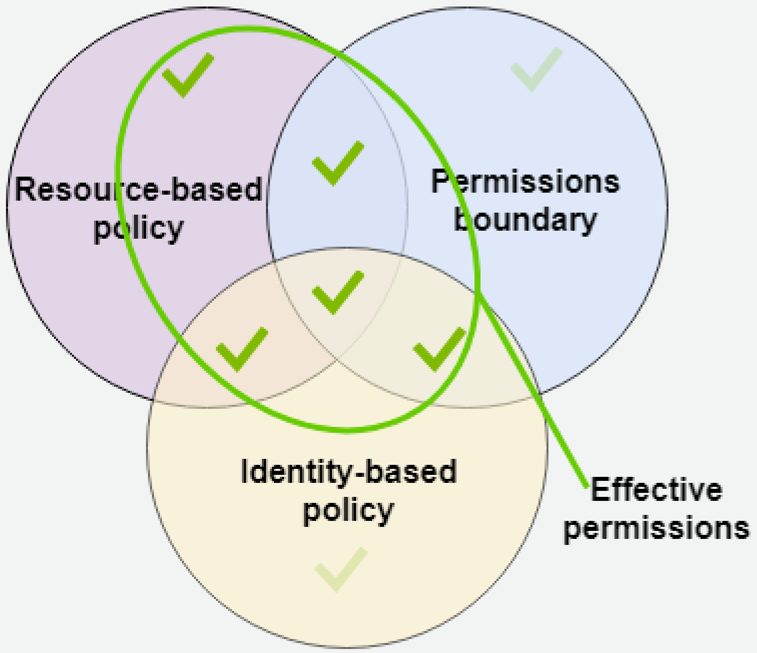
```
{
            "Sid": "DenyDeletePermBoundary",
            "Effect": "Deny",
            "Action": [
                "iam:DeletePolicy",
                "iam:DeleteRolePermissionsBoundary"
            ],
            "Resource": [
                "arn:aws:iam::<account>:policy/MyAppPermissionsBoundary",
                "arn:aws:iam:::policy/apps/my_app/DelegatedPermissionsBoundary",
                "arn:aws:iam::*:role/*"
            ]
}
```

aws professional services

Permissions Boundaries

Resource-based Policies

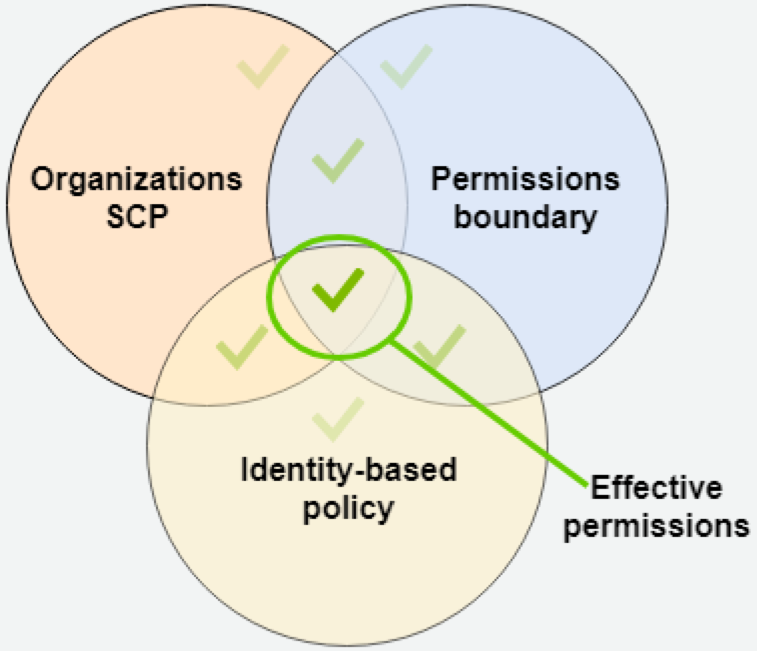
- Boundaries do not effect permissions granted through resource-based policies.
- Effective permissions consist of everything that is allowed by the resource-based
  policy and everything that is allowed by both the permissions boundary and the identity-based policy.



Permissions Boundaries

Organizations SCP's

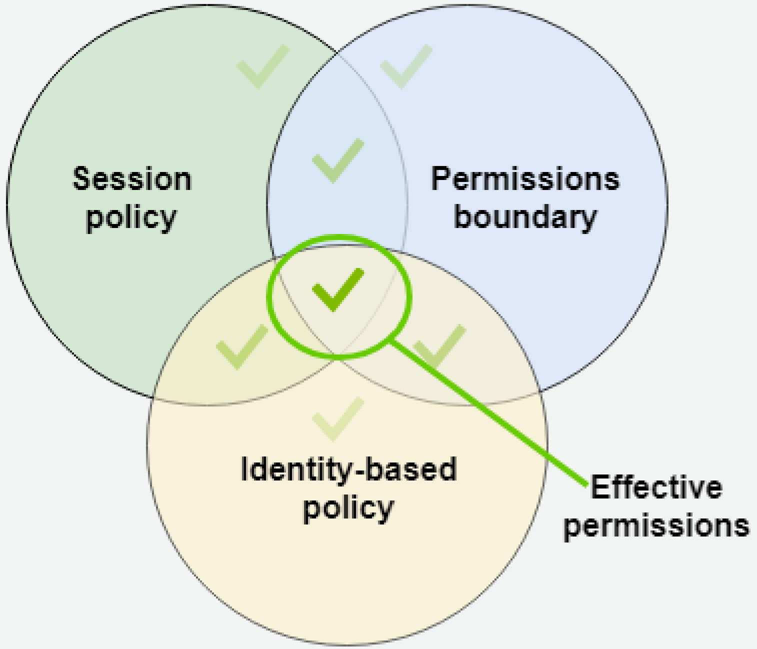
- SCP's do not grant any permissions to a principal.
- SCP's only limit the operations in an account and apply to all principals.
- Effective permissions consist of any operations that is allowed by any of the three policies.



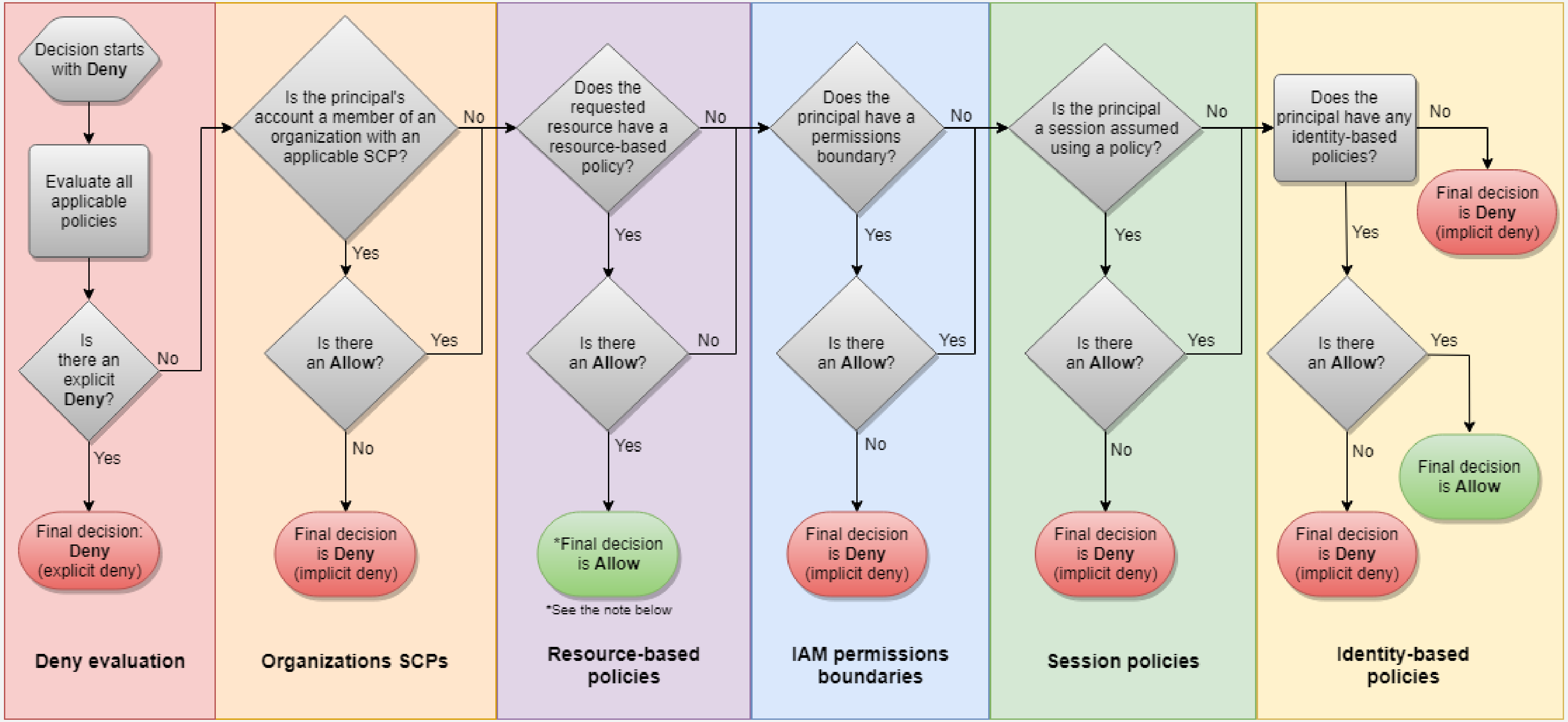
Permissions Boundaries

Session Policy

- SCP's do not grant any permissions to a principal.
- SCP's only limit the operations in an account and apply to all principals.
- Effective permissions consist of any operations that is allowed by any of the three policies.



IAM Evaluation Logic



Decision starts with Deny
Evaluate all applicable policies
Is there an explicit Deny?
Yes
Final decision: Deny (explicit deny)
No
Is the principal's account a member of an organization with an applicable SCP?
Yes
Is there an Allow?
No
Final decision is Deny (implicit deny)
Yes
No
Does the requested resource have a resource-based policy?
Yes
Is there an Allow?
Yes
*Final decision is Allow
*See the note below
No
No
Does the principal have a permissions boundary?
Yes
Is there an Allow?
No
Final decision is Deny (implicit deny)
Yes
No
Is the principal a session assumed using a policy?
Yes
Is there an Allow?
No
Final decision is Deny (implicit deny)
Yes
No
Does the principal have any identity-based policies?
No
Final decision is Deny (implicit deny)
Yes
Is there an Allow?
Yes
Final decision is Allow
No
Final decision is Deny (implicit deny)
Deny evaluation
Organizations SCPs
Resource-based policies
IAM permissions boundaries
Session policies
Identity-based policies

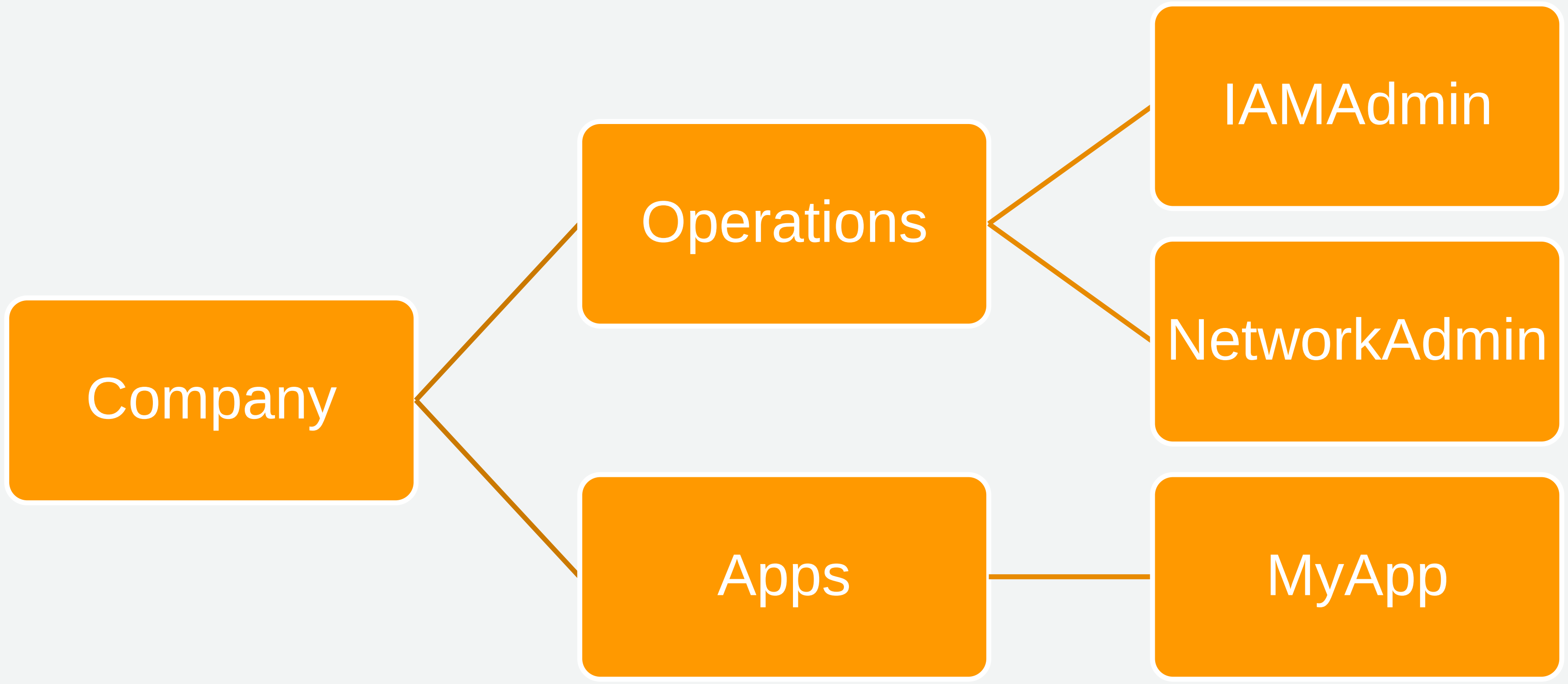
aws professional services

AWS IAM Path

AWS IAM Path

- If using the API or AWS CLI to create IAM entities, you can also give the entity an optional path.
- Use a single path, or nest multiple paths as if they were a folder structure. Example:
  /division_abc/subdivision_xyz/product_1234/engineering/
- Principals in the same path does not automatically grant access.

AWS IAM Path - Example



AWS IAM Path – How does it work?

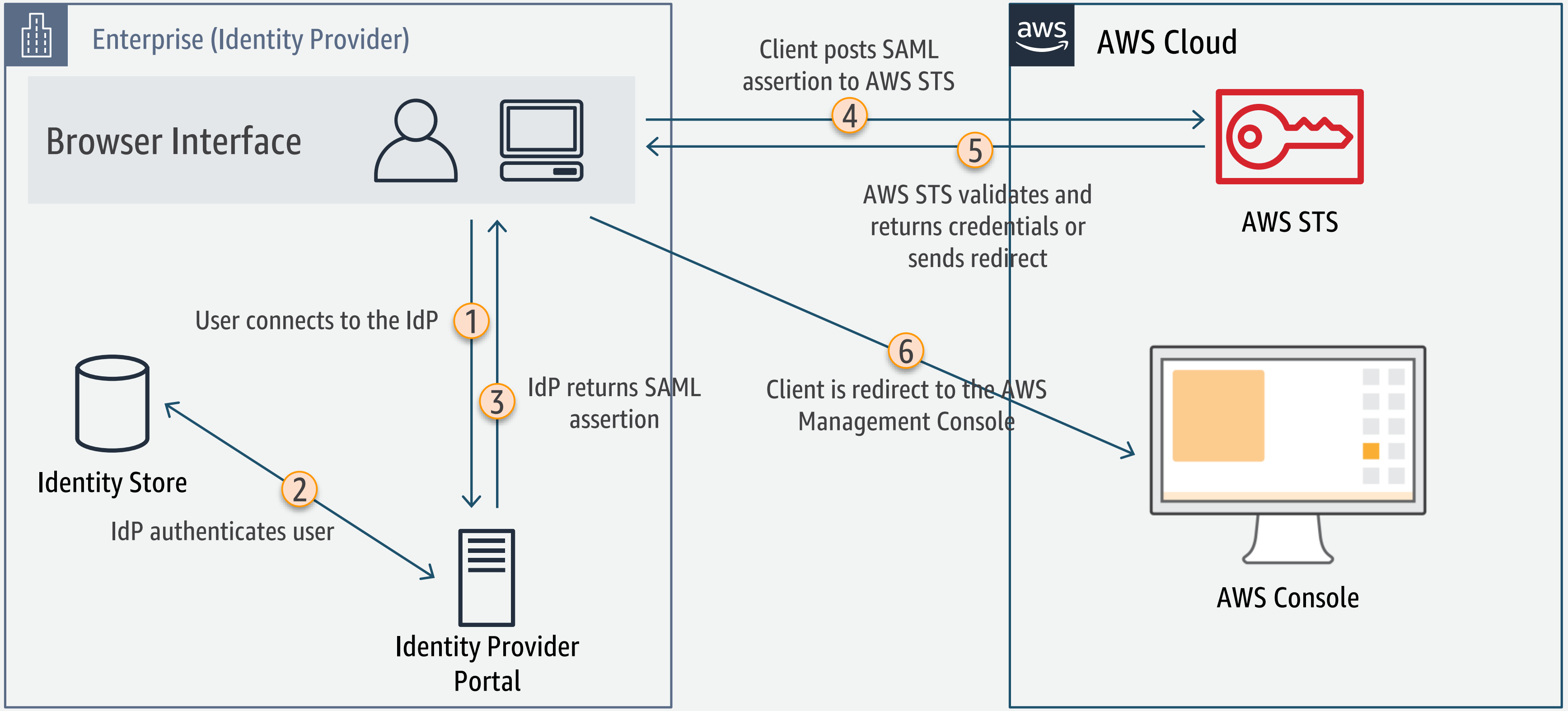
```
"Effect": "Allow",
            "Action": [
                "iam:DetachRolePolicy",
                "iam:DeleteRolePolicy",
                "iam:PutRolePermissionsBoundary",
                "iam:CreateRole",
                "iam:AttachRolePolicy",
                "iam:PutRolePolicy"
            ],
            "Resource": "arn:aws:iam:::role/apps/my_app/*",
            "Condition": {
                "StringEquals": {

"iam:PermissionsBoundary":"arn:aws:iam::<account>:policy/MyAppPermissionsBoundary"
                }
            }
```

Federation

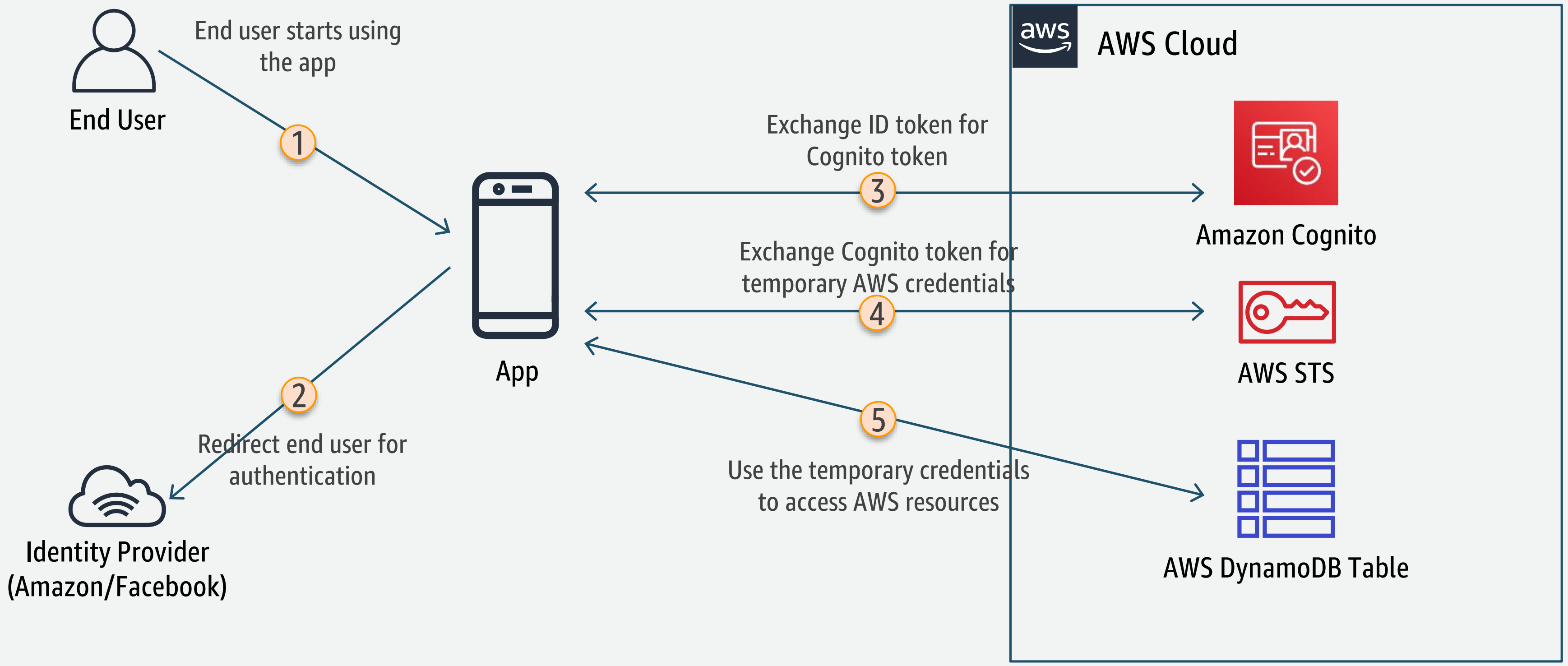
Federation

- Amazon STS supports SAML 2.0
- **Benefits:**
  - Open standards
  - Quicker and easier to implement federation
  - Leverage existing identity management software to manage access to AWS resources
  - **No coding required**

Federation – SAML2.0


Enterprise (Identity Provider)
Browser Interface
User connects to the IdP
1
Identity Store
2
IdP authenticates user
3
IdP returns SAML assertion
Identity Provider Portal
Client posts SAML assertion to AWS STS
4
5
AWS STS validates and returns credentials or sends redirect
6
Client is redirect to the AWS Management Console
aws
AWS Cloud
AWS STS
AWS Console

Federation – Cognito

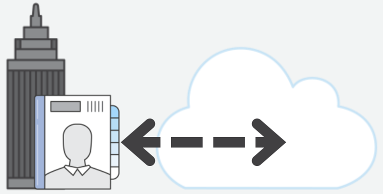

End User
End user starts using the app
1
App
2
Redirect end user for authentication
Identity Provider (Amazon/Facebook)
AWS Cloud
Exchange ID token for Cognito token
3
Amazon Cognito
Exchange Cognito token for temporary AWS credentials
4
AWS STS
5
Use the temporary credentials to access AWS resources
AWS DynamoDB Table

Federation – AWS Directory Service

AWS Directory Service

Simple AD

AWS Managed Microsoft AD



AD Connector

Federation – Simple AD

Simple AD is a Microsoft Active Directory–compatible directory from AWS Directory Service that is powered by Samba 4. Simple AD supports commonly used Active Directory features such as user accounts, group memberships, domain-joining EC2 instances running Linux and Microsoft Windows.

When to use

In most cases, Simple AD is the least expensive option and your best choice if you have 5,000 or less users and don't need the more advanced Microsoft Active Directory features.

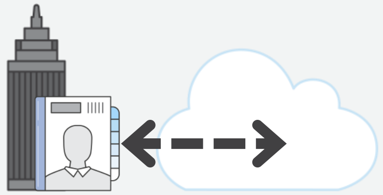
Federation – AWS Managed Microsoft AD

AWS Managed Microsoft AD is a managed Microsoft Active Directory hosted on the AWS Cloud. It provides much of the functionality offered by Microsoft Active Directory plus integration with AWS applications. With the additional Active Directory functionality, you can, for example, easily set up trust relationships with your existing Active Directory domains to extend those directories to AWS services.

When to use

Microsoft AD is your best choice if you have more than 5,000 users and/or need a trust relationship set up between an AWS hosted directory and your on-premises directories.

Federation – AD Connector



AD Connector is a proxy service for connecting your on-premises Microsoft Active Directory to the AWS Cloud without requiring complex directory synchronization or the cost and complexity of hosting a federation infrastructure.

When to use

AD Connector is your best choice when you want to use your existing on-premises directory with AWS services.

Roles & Responsibilities

Roles & Responsibilities

What roles are needed?

- Human roles are needed for human operators to assume into the AWS account to use services.
- System roles may be needed for your applications running on EC2 instances or for AWS services to utilize your AWS resources.

Roles & Responsibilities

What roles are needed?

- Consider the current roles in your organization, and compare to the end-goal job roles your organization wants to achieve (i.e. DevOps).
- Consider increasing business agility by removing operational blockers
- Consider separation of duties and least privilege

Roles & Responsibilities

What roles are needed?

- Administrator
- IAMAdmin
- NetworkAdmin
- SystemOperator
- Developer
- SecurityOperator
- ReadOnlyAnalyst
- FinanceAdmin
- ComplianceAnalyst
- StarPortalWebTier
- StarPortalAppTier
- StarPortalDataTier
- ApplicationDelivery

Questions?

Appendix A – IAM Naming Convention

Example IAM User Naming Convention

IAM User Names

- Syntax
  - [<organization/business unit short name>][<team/project name>]<product/technology>
  - <user alias>
- Examples
  - TeamASecMonitoring
  - TeamBSecMonitoring
  - AppMonitoring
  - JohnDoe

Example IAM Group Naming Convention

IAM Group Names

- Syntax
  - [<organization/business unit short name>][<team/project name>]<function name>
- Examples
  - CompanyTeamEngineering
  - Developers
  - Admins

Example IAM Role Naming Convention

IAM Role Names

- Syntax
    - [<organization/business unit short name>]<team/project name><entity name>
    - [<organization/business unit short name>]<team/project name><app name><app tier>
- Examples
    - CompanyTeamDevelopers
    - Developers
    - TeamAStarPortalWebTier
    - StartPortalAppTier

Appendix B – IAM Best-Practices

IAM Best-Practices

- Lock away your AWS account (root) access keys
- Create individual IAM users
- Use groups to assign permissions to IAM users
- Grant least privilege
- Configure a strong password policy for your users
- Enable MFA for privileged users

IAM Best-Practices

- Use roles for applications that run on Amazon EC2 instances
- Delegate by using roles instead of by sharing credentials
- Rotate credentials regularly
- Remove unnecessary credentials
- Use policy conditions for extra security
- Monitor activity in your AWS account

More info: http://docs.aws.amazon.com/IAM/latest/UserGuide/best-practices.html

Appendix C – AWS Services That Work With IAM

aws professional services

AWS Services That Work With IAM

The following webpage describes the IAM permission types each service supports, specifically:

- Action-level permissions
- Resource-level permissions
- Resource-based policies
- Tag-based authorization
- Requests via temporary security credentials
- Service-linked roles

http://docs.aws.amazon.com/IAM/latest/UserGuide/reference_aws-services-that-work-with-iam.html

Appendix D – Service Last Accessed Data

Service Last Accessed Data

- The IAM console & API's provide information about when IAM users and roles last attempted to access AWS services.
- You can use this information to identify unused and not recently used permissions in your IAM policies.
- Knowing if and when an IAM entity last exercised a permission can help you remove unnecessary permissions and tighten your IAM policies with less effort.

https://docs.aws.amazon.com/IAM/latest/UserGuide/access_policies_access-advisor.html