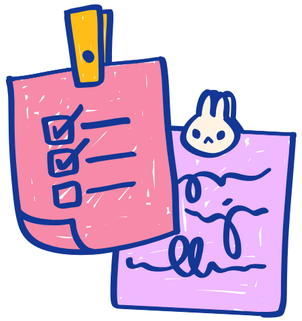
DevKTOps

ARCHITECTING ON aws

Module - 3 : Identity and Access Management (IAM)

Module Overeiw

- IAM Users, Groups, and Roles
- Federated Identity Management
- Amazon Cognito
- AWS Organizations

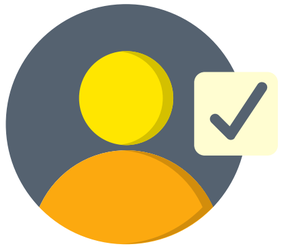


AWS Account Root User

This account has full access to all AWS services and resources.

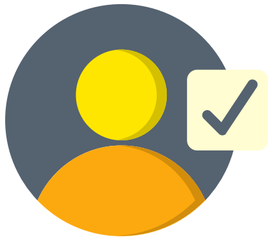
- Billing information
- Personal data
- Your entire architecture and its components

Root User Safety



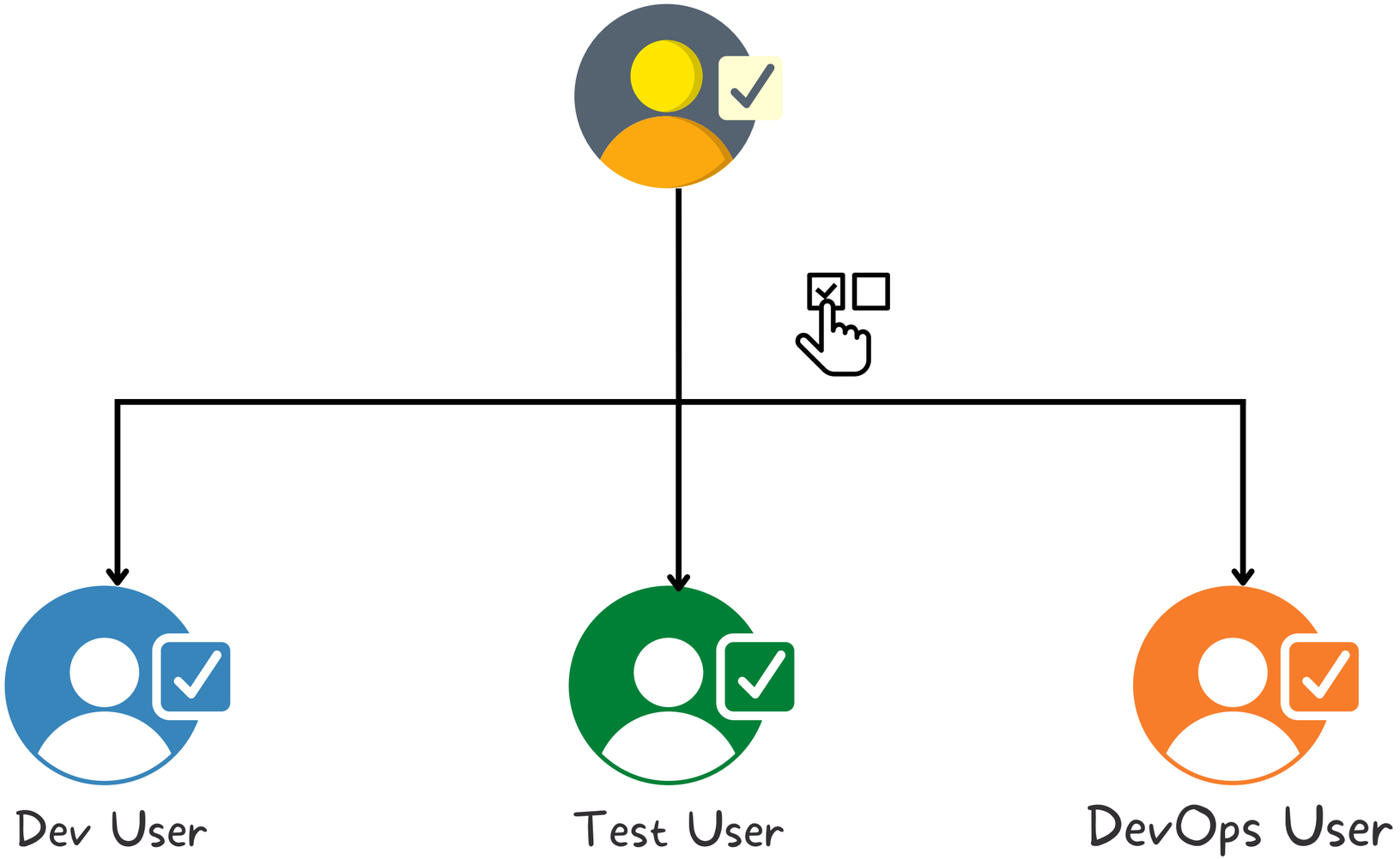
Create IAM Admin User

Store the root user credentials in safe place

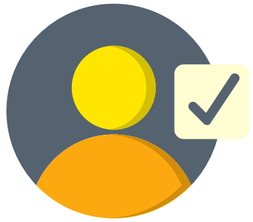


Use IAM admin user

IAM Use Case



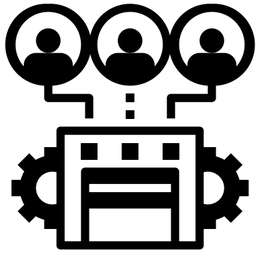
IAM Principals



IAM User

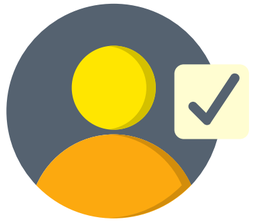
Federated User

IAM Role



Identity Provider (IdP)

IAM Users



IAM User

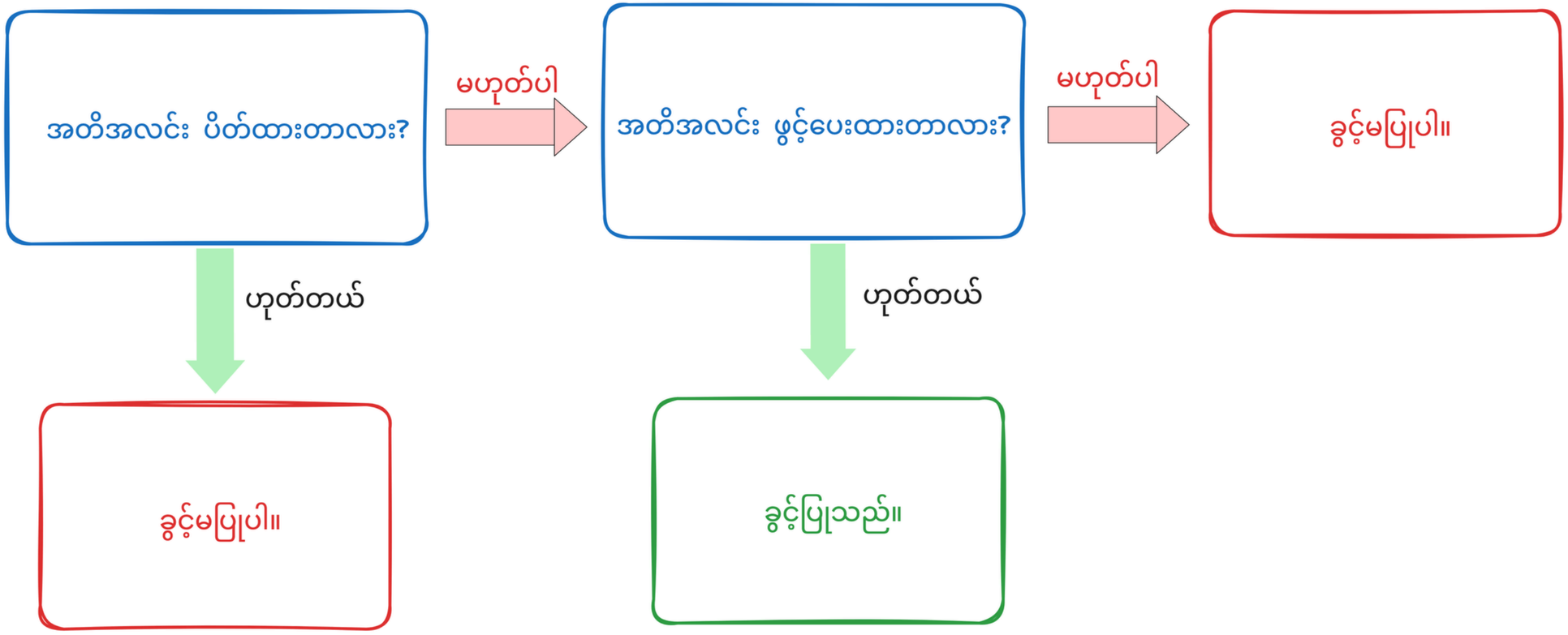
- IAM users are not separate AWS accounts,they are users within your account.
- Each user has their own credentials.
- IAM users are authorized to perform specific AWS actions based on their permissions.
- There are no default permissions.
- Access to the AWS Management Console or CLI must be explicitly granted.

IAM Users - Granting Permission

Policy

- A formal declaration of one or more permissions
- Evaluated at the time of request
- IAM policies ONLY control access to AWS services
- IAM has no visibility above the hypervisor

How IAM Determine Permissions?



IAM Policy

Policy

- Identity-Based - Attached to an IAM principal
- Resourced-Based - Attached to an AWS Resource

Idenitiy-Based Policies

Identity-based

Identity-based policies are permission policies that you can attach to a principal, such as IAM user, role or group.These policies control what actions that identity can perform, on which resources and under what conditions.

There are three type of policies :

- AWS-managed
- Customer-managed
- Inline

How Control works:

- Actions performed
- Which resources
- What conditions are required

Resource-Based Policies

Resource-based

Resource-based policies are JSON policy documents that you attach to a resource such as an Amazon S3 Bucket.

How Control works:

- Actions allowed by specific principal
- What conditions are required
- Are always inline policies
- No AWS-managed resource-based policies

Principal with Attached Permissions

Amazon RDS Policy

Amazon S3 Policy

Amazon EC2 Policy

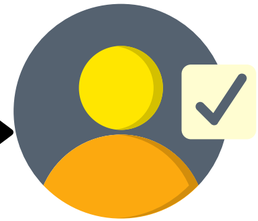
AWS Management Console

AWS CLI

Applying Permissions

Policy

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



ARNs and Wildcards

Resources are identified by using Amazon Resource Name (ARN) format

Example : "sqs" : "arn:aws:sqs:ap-southeast-1:12345678:devktops"

Use a wildcard (*) to give access to all actions for a specific AWS service

Example :

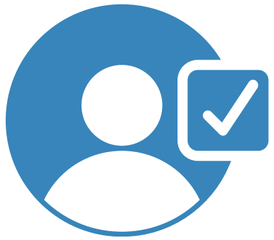
"Action": "s3:*"

Amazon Resource Names

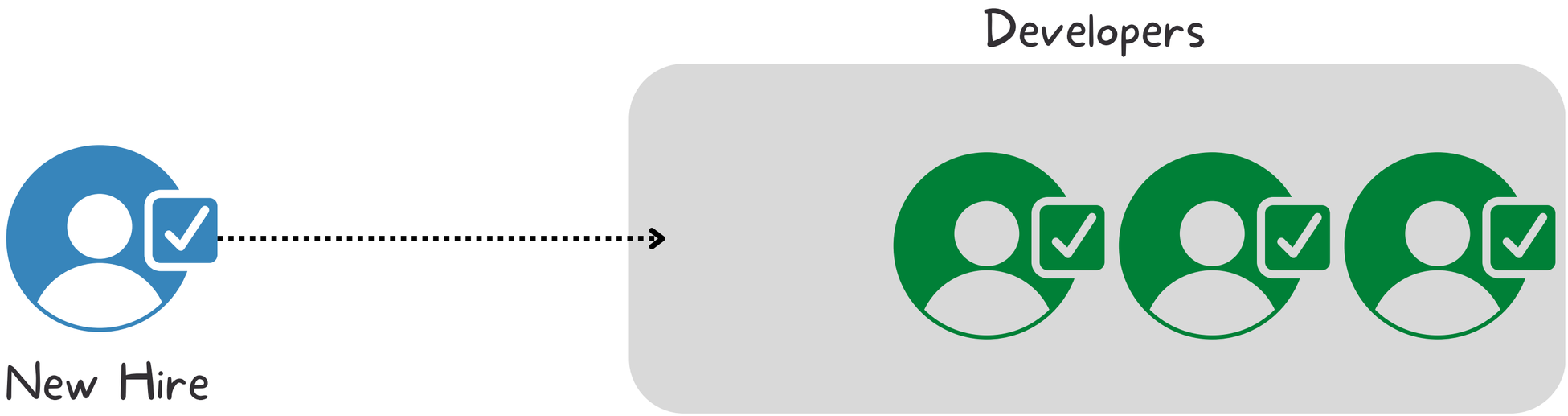
There are six segments in ARNs, each separated by a colon :.

arn:<PARTITION>:<SERVICE>:<REGION>:<ACCOUNT_ID>:<RESOURCE_ID>

Managing Users



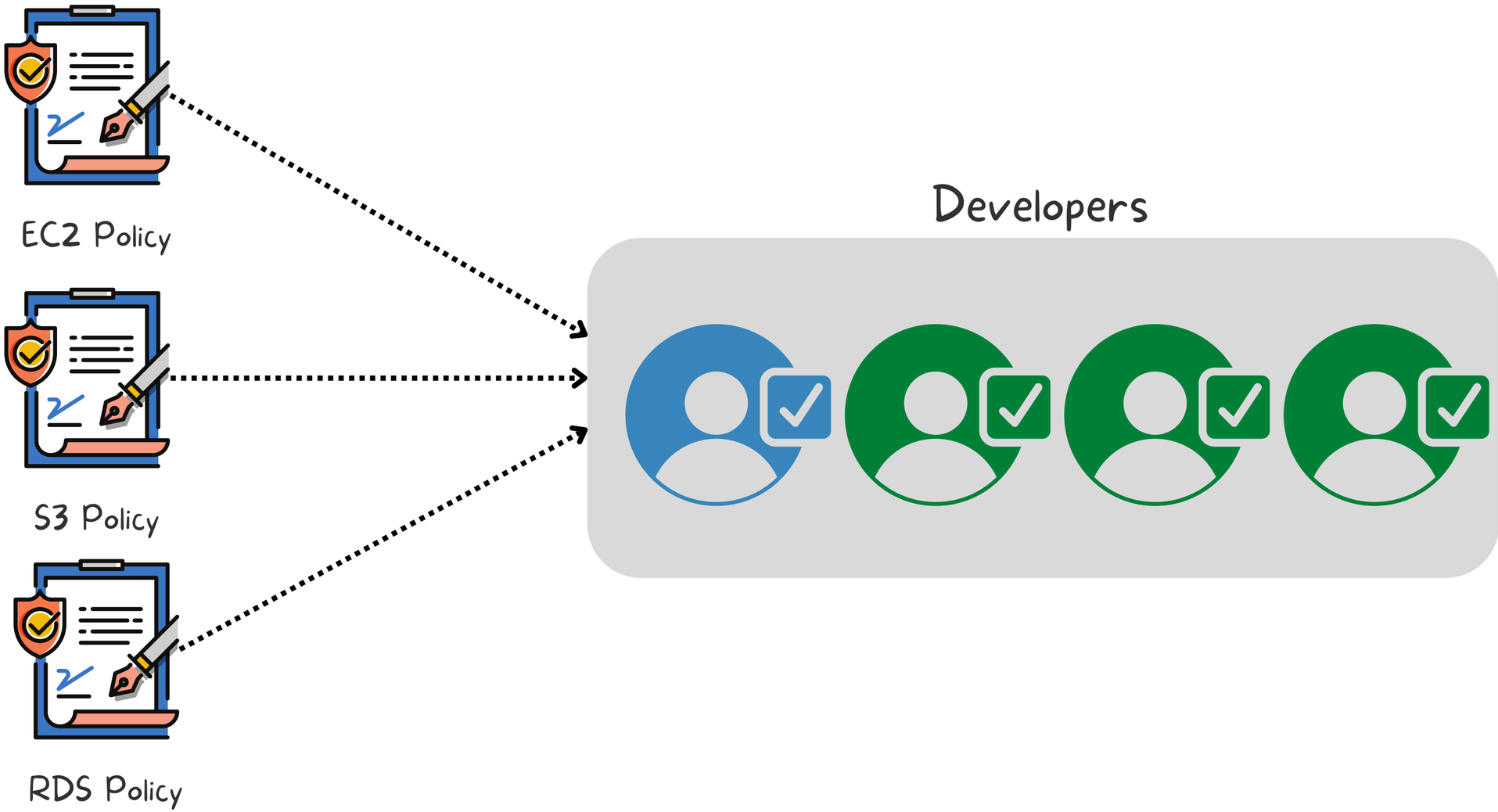
IAM User Group



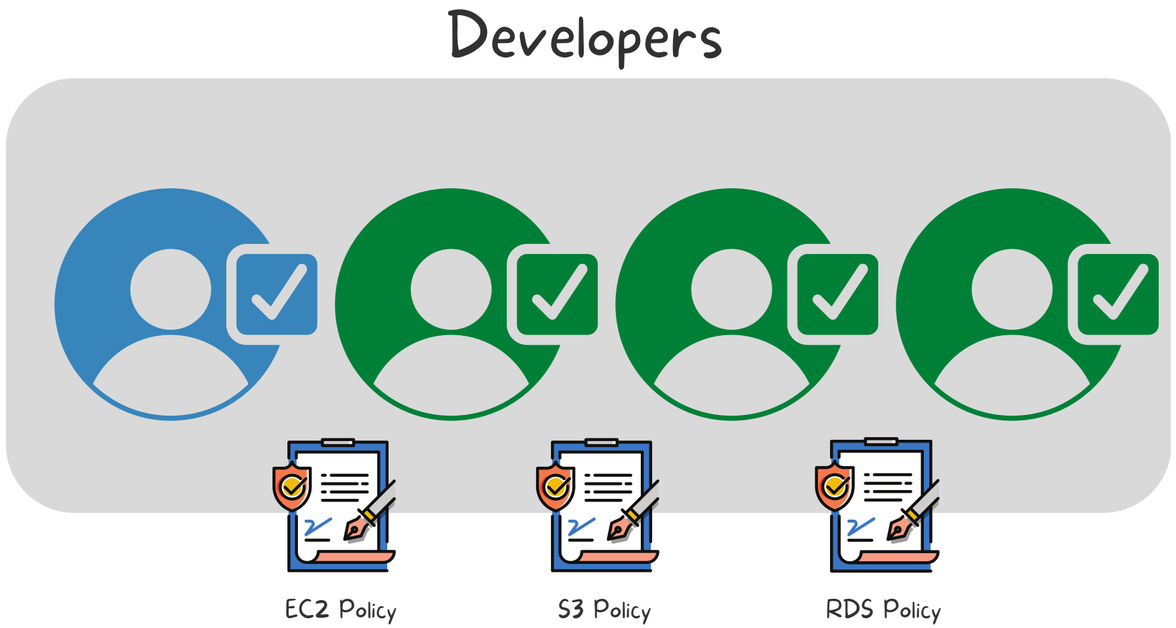
IAM User Group

Developers

IAM User Group



IAM User Group



IAM Roles

A role lets you define a set of permissions to access the resources that a user or service needs.

- The permissions are not attached to an IAM user or group.
- The permissions are attached to a role and the role is assumed by the user or the service.

IAM Roles Use Cases

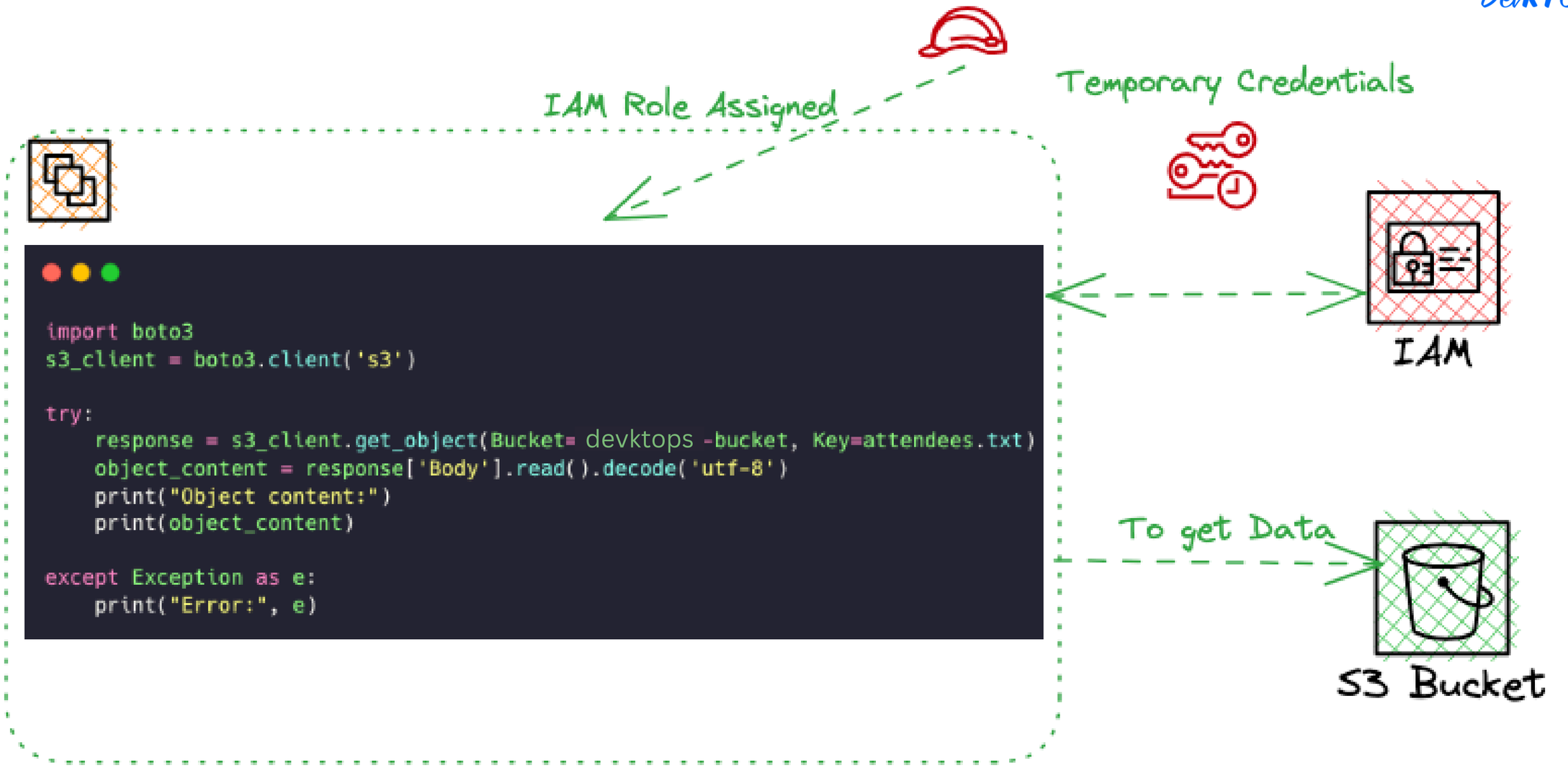
- Provide AWS resources with access to AWS services
- Provide access to externally authenticated users
- Provide access to third parties
- Switch roles to access resources in:
  - Your AWS account
  - Any other AWS account (cross-account access)

IAM User Access Key


```
cat ~/.aws/credentials

[ devktops-aws]
aws_access_key_id=YOUR_ACCESS_KEY_ID
aws_secret_access_key=YOUR_SECRET_ACCESS_KEY
```

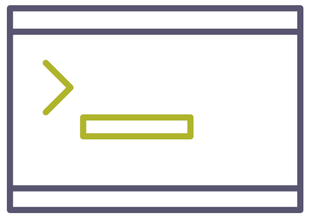
```python
import boto3

aws_access_key_id = 'YOUR_ACCESS_KEY_ID'
aws_secret_access_key = 'YOUR_SECRET_ACCESS_KEY'

bucket_name = ' devktops '
object_key = 'attendees.txt'


s3_client = boto3.client('s3', aws_access_key_id=aws_access_key_id,
aws_secret_access_key=aws_secret_access_key)

try:
    # Get the object from S3
    response = s3_client.get_object(Bucket=bucket_name, Key=object_key)

    # Read the content of the object
    object_content = response['Body'].read().decode('utf-8')

    # Do something with the object content
    print("Object content:")
    print(object_content)

except Exception as e:
    print("Error:", e)
```

IAM Role Usage

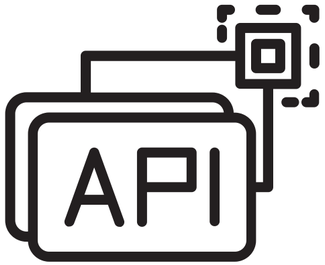


Assume a Role

AWS Management Console



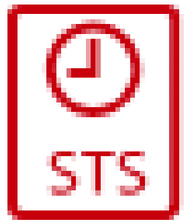
AWS CLI



API Call

AWS Security Token Service (AWS STS)

AWS Security Token Service (AWS STS)



AWS STS

- To Request temporary, limited-privilege credentials
- For IAM users or Federated Users to use this credentials

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "sts:AssumeRole",
            "Resource": "arn:aws:iam::ACCOUNT_ID:role/ROLE_NAME"
        }
    ]
}
```

Allow IAM User to assume a role

Role-based Access Control (RBAC)

Traditional approach to Access Control:

- Grating users specific permissions based on Job function (such as Finance)
- Create a specific IAM role for each permission combination
- Have to update permissions by adding access for each new resource

Attribute-based Access Control (ABAC)

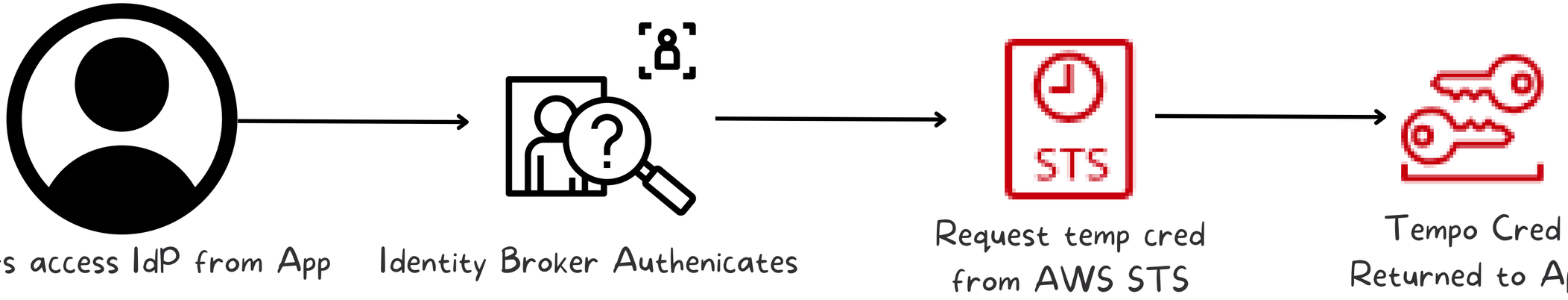
Highly scalable approach to access control

- Attributes are a key or a key-value pair (tag)
  - Example : Team = Developers, Project = DevKTOps
- Permissions (policy) rules are easier to handle with ABAC more than with RBAC
- Outcomes
  - Automatically apply permissions based on attributes
  - No need to update a permissions for every new user or resource

Set up ABAC

1. Set access control attributes(tags) on identities
2. Add attributes to new resources
3. Configure permissions based on attributes
4. Test
    a. Create new resources
    b. Verify the policy

External Authenticated Users



Users access IdP from App

Identity Broker Authenicates

Request temp cred from AWS STS

Tempo Cred Returned to App

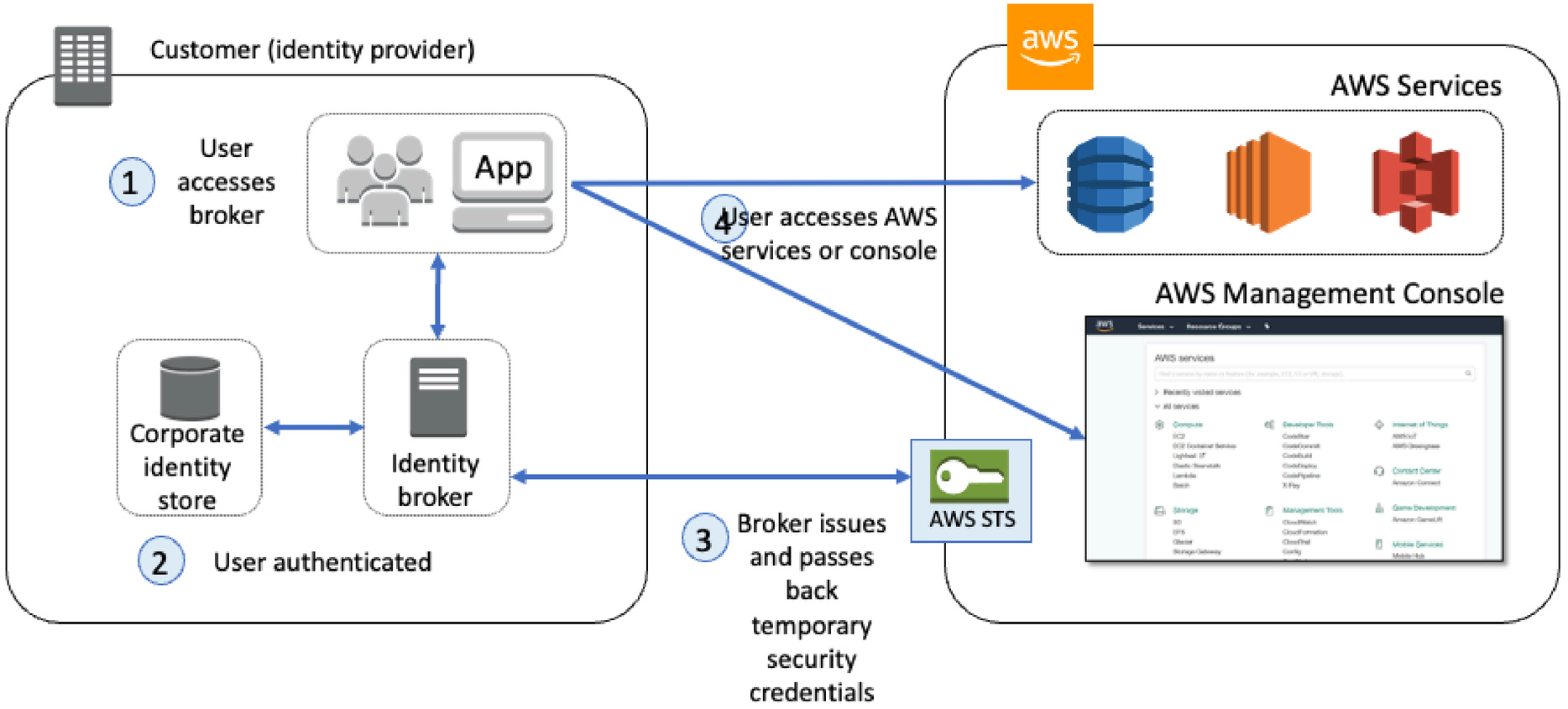
Identity Federation

- External users can able to authenticate to the AWS Account
  - Example : Microsoft AD
- No need to create IAM Users, providing access through existing identities

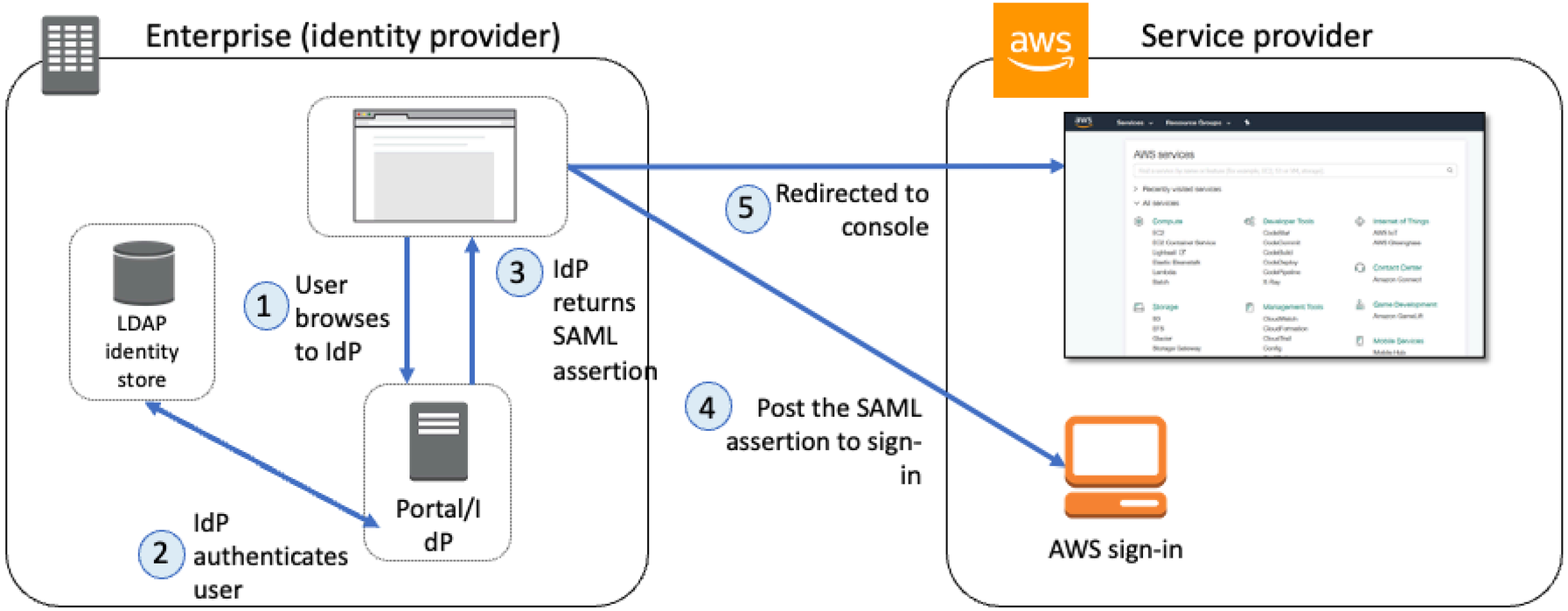
Identity Federation Options

1. AWS STS
   a. Public Identity service providers (IdPs)
   b. Custom identity broker application
2. Security Assertion Markup Language (SAML)
3. Amazon Cognitio

STS Identity Broker Process



SAML


Enterprise (identity provider)
aws
Service provider
LDAP identity store
1 User browses to IdP
3 IdP returns SAML assertion
5 Redirected to console
4 Post the SAML assertion to sign-in
Portal/IdP
2 IdP authenticates user
AWS sign-in

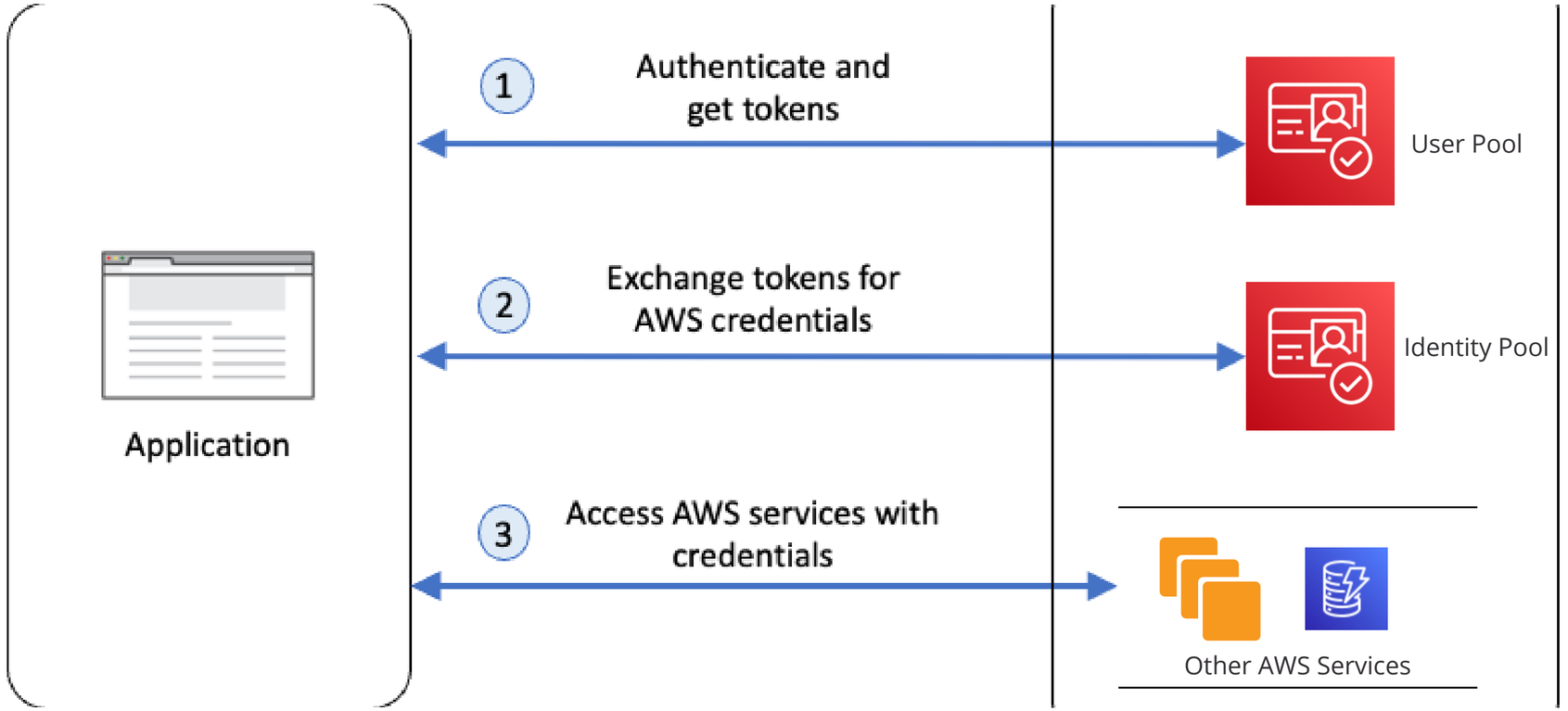
Amazon Cognito

Fully managed service that provides authentication, authorization, and user management for web and mobile apps.

There are two main components :

- User pools - provide sign-up and sign-in options for your app users.
- Identity pools - enable you to grant your users to access to the other AWS service. Identity pools and user pools can be used separately or together.

Amazon Cognito Example



AWS Landing Zone (Control Tower)

A solution that helps customers more quickly set up a secure, multi-account AWS environment based on AWS best practices, featuring:

- Multi-Account Structure
- Account Vending Machine
- User Access
- Notifications

Multiple Accounts

How many AWS accounts does your organization need?

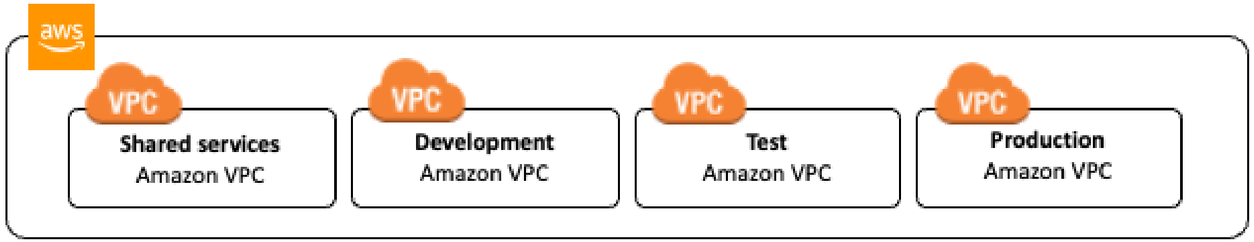
Dev

Test

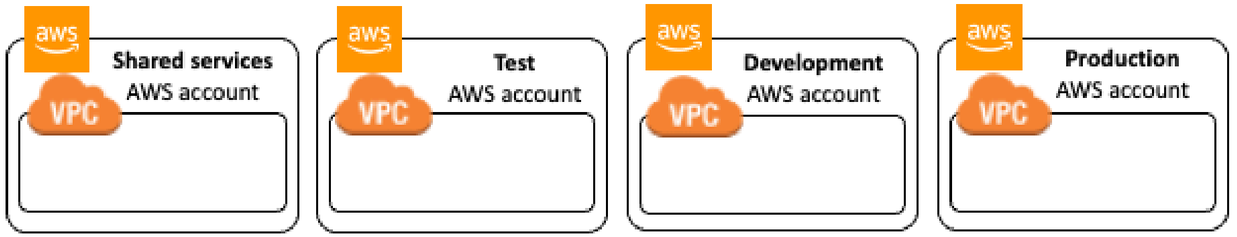
Production

- Can be leveraged for isolation:
  - Separate business units, dev/test/production environments
- Can be leveraged for security:
  - Separate accounts for regulated workloads, different geographical locations, governing other accounts
- Cross-account access is not enabled by default

Best Practice in Accounts



One account – multiple VPCs

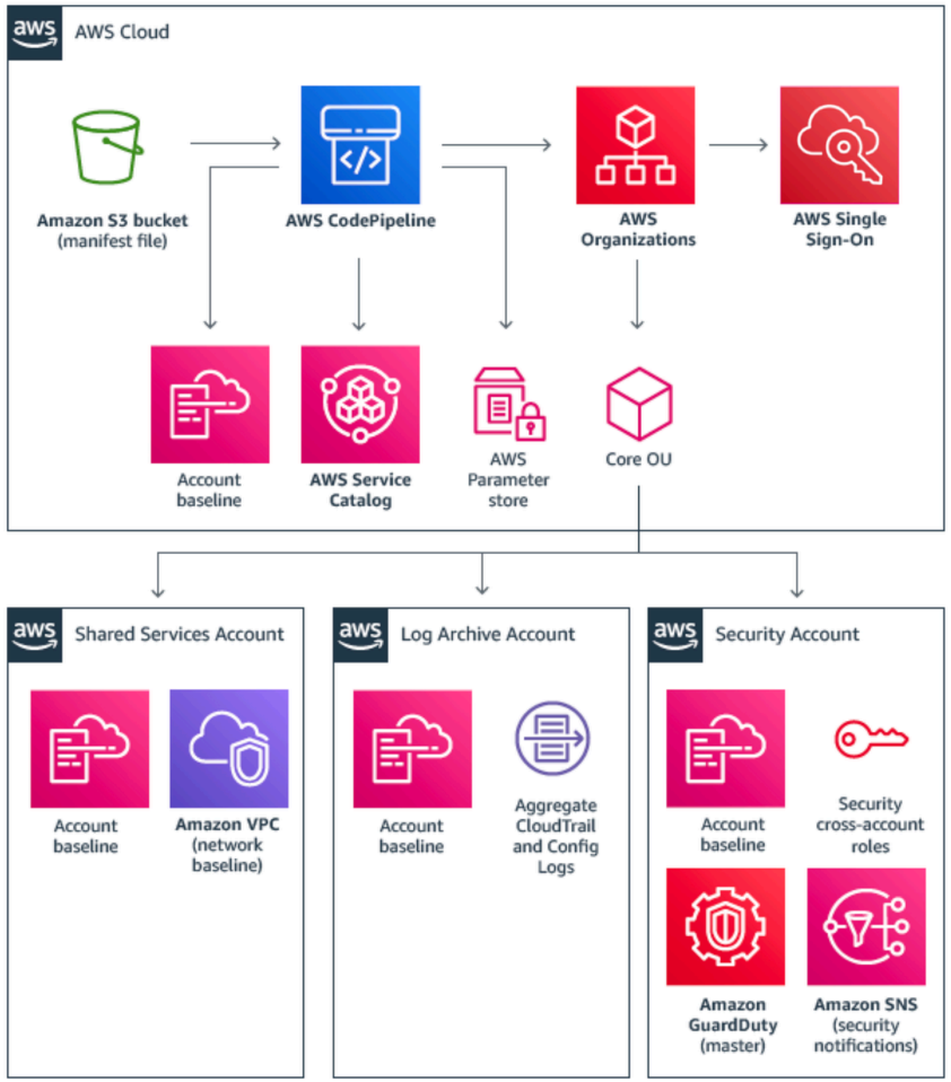


Multiple accounts – One VPC per account

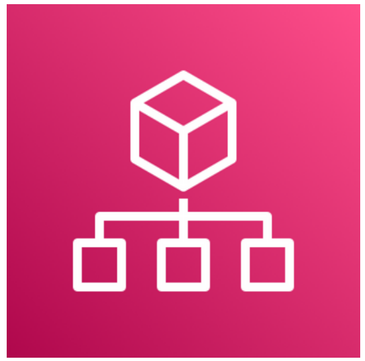
Multiple Accounts Strategies

Centralized security management	Single AWS account
Separation of production, development, and testing environments	Three AWS accounts
Multiple autonomous departments	Multiple AWS accounts
Centralized security management with multiple autonomous independent projects	Multiple AWS accounts

Landing Zone - Mulit Accounts



How to Manage All these Accounts?



AWS Organizations

Centralized account management

- Group-based account management
- Policy-based access to AWS services
- Automated account creation and management
- Consolidated billing
- API-based

Review

If you need to grant temporary permissions to a resource, what would you use?

Review

One of your AWS Account's users can't access an S3 bucket. What should you check to identify the cause of the problem?

Review

1.You have created a mobile application that makes calls to DynamoDB to fetch data.

2.The application is using the DynamoDB SDK and the AWS account root user access key ID and secret access key to connect to DynamoDB from the mobile app.

3.With respect to the best practice for security in this scenario, how should this be fixed?

Dev KT Ops

Thank you!

Architecting on AWS : Module 7