Architecting with AWS

Identity, Authentication, and Authorization

Identity, Authentication, and Authorization | What we'll cover

1. Authentication, authorization, and where they apply
2. Authentication to AWS Service APIs
3. Authorization Policies
4. Temporary credentials with the Security Token Service
5. Service-specific, OS, and application authentication

Identity, Authentication, and Authorization | What we'll cover

1

Authentication, authorization, and where they apply

What we'll cover

1 The 3 major realms where authentication and authorization occur within AWS.

2 Multi-factor authentication and how to implement it.

3 Your AWS master account.

4 Creating users and groups with IAM.

5 The role of authorization policies.

Identity, Authentication, and Authorization | The Three Realms

Let’s think about **Wordpress**

Wordpress: We want to run it on AWS

Wordpress: We want to run it on AWS

1. Login to Management console and launch EC2 instance

Wordpress: We want to run it on AWS

1. Login to Management console and launch EC2 instance
2. Login to instance, install Wordpress and configure DB connection

Wordpress: We want to run it on AWS

1. Login to Management console and launch EC2 instance
2. Login to instance, install Wordpress and configure DB connection
3. Login to Wordpress and write a blog post

Login to Management console and launch EC2 instance

Authentication and Authorization to AWS APIs:

- Everything is an API at AWS
- You have to make authenticated API requests

Login to Management console and launch EC2 instance

Examples of API requests

- EC2->RunInstance

Login to instance, install Wordpress and configure DB connection

Authentication and Authorization to OS and Database:

- Local Linux user (for example, root@, ubuntu@, ec2-user@)
- Local Windows user (Administrator)

Login to instance, install Wordpress and configure DB connection

Authentication and Authorization to OS and Database:

- MySQL username and password
- SQL Server username and password

Login to Wordpress and write a blog post

Authentication and Authorization to the application:

- Wordpress authenticates to a database
- Some applications authenticate to Active Directory
- Others authenticate via OAuth 2.0, etc.

Identity, Authentication, and Authorization | WordPress Example

Task	Can AWS help?
Login to Management console and launch EC2 Instance	Yes, a lot
Login to instance, install Wordpress and configure DB connection	Yes, some
Login to Wordpress and write a blog post	Depends on the app

Identity, Authentication, and Authorization | Authentication to APIs

2

Authentication to AWS Service APIs

Let’s think about using AWS APIs:

S3 Create Bucket

- REST API
- Management Console
- SOAP API

Let’s think about using AWS APIs:

RDS Snapshot Database

- REST API
- Management Console

Let's think about using AWS APIs:

AutoScaling Execute Scaling Policy

- **REST API**

Three major interfaces to AWS:

	For All Services?	Credential
REST API	★	Access Key, Secret Key
MANAGEMENT CONSOLE	★	Username, Password
SOAP API	★	X.509 Certificate

Multi-factor Authentication (MFA): optional, but <u>recommended</u>

Physical

Virtual

- Android
- iOS
- Windows Phone
- Blackberry

Possible MFA Configurations

For	Require		
Management Console	Username and Password	+	
Billing/Account Info	Username and Password	+	Phone
EC2::Terminate Instance API	Access Key and Secret Key	+	Phone

The Master Account

The Master Account

Every account has a master user

- Equivalent to Root/Administrator

The Master Account

Every master user has:

- A Management Console login
- An Access Key/Secret Key

The Master Account

Best practices:

- **Do not** use the access key/secret key from the Master Account
- **Apply a physical MFA** to the Management Console login
- **Use Identity and Access Management**

Identity and Access Management

Within a Master Account, create:

1. **Users**
   - No credentials or privilege by default

Identity and Access Management

Within a Master Account, create:

1. **Users**
    - **Credentials** can be any of:
        - Console login
        - Access key/secret key
        - MFA
        - X.509 cert

Identity and Access Management

Within a Master Account, create:

1. **Users**
    - **Privilege** via:
        - Individual authorization
        - **Group** membership

Identity and Access Management

Within a Master Account, create:

1. **Users**
   - **Best Practices:**
     - Rotate access key/secrete key
     - Apply a password policy

Identity and Access Management

Within a Master Account, create:

1. **Users**
2. **Groups**
    - A collection of users
    - Defines privilege of members via authorization policies

Identity and Access Management

Within a Master Account, create:

1. **Users**
2. **Groups**
3. **Roles**

Identity and Access Management

Within a Master Account, create:

1. **Users**
2. **Groups**
3. **Roles**
    - Allow your applications (e.g., Java) running on EC2 to securely access other services (e.g., S3, SQS, etc)
    - Allow cross-account management/access
        - Jane in Account A may assume a Role in Account B, giving Jane an Access Key/Secret Key/Token that may be used to make API calls to Account B.

Identity and Access Management

Within a Master Account, create:

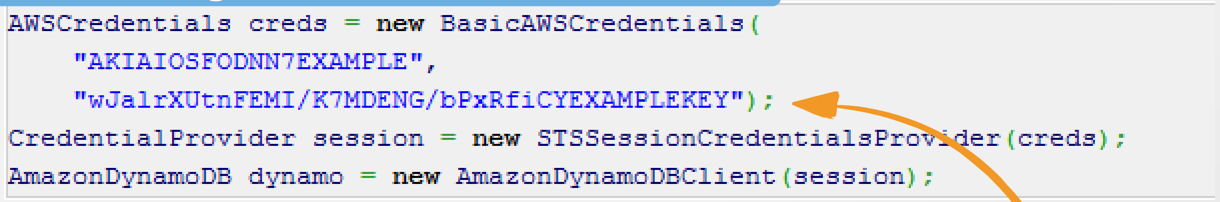
1. **Users**
2. **Groups**
3. **Roles**
    - Allow your applications (e.g., Java) running on EC2 to securely access other services (e.g., S3, SQS, etc)
    - Allow cross-account management/access
        - Jane in Account A may assume a Role in Account B, giving Jane an Access Key/Secret Key/Token that may be used to make API calls to Account B.

Let's look at some example code…

Java App on EC2, accessing DynamoDB

Java App on EC2, accessing DynamoDB

EC2 not using an IAM Role



```
AWSCredentials creds = new BasicAWSCredentials(
    "AKIAIOSFODNN7EXAMPLE",
    "wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY");
CredentialProvider session = new STSSessionCredentialsProvider(creds);
AmazonDynamoDB dynamo = new AmazonDynamoDBClient(session);
```

Credentials embedded in code ☹

Java App on EC2, accessing DynamoDB

EC2 using an IAM Role

```
AmazonDynamoDB dynamo = new AmazonDynamoDBClient();
```

Credentials automatically retrieved from IAM Role! ☺

Identity, Authentication, and Authorization | Authorization Policies

3

Authorization Policies

Authorization Policies

- Defining the fine-grain privilege to IAM Users, Groups, and Roles

Authorization Policy Documents

1. JSON format
2. Action (API)
3. Resource (some services)
4. Condition (optional)

Authorization Policy Documents

1. JSON format
2. Action (API)
3. Resource (some services)
4. Condition (optional)

} Define **least-privilege access** for each user, group, or role in your AWS account

Authorization Policy Documents

1. JSON format
2. **Action (API)**
   - Specific API(s) that you can call, such as:
     - S3::GetObject
     - S3::GetObjectVersion
     - **S3::Get***
3. Resource (some services)
4. Condition (optional)

Authorization Policy Documents

1. JSON format
2. Action (API)
3. **Resource (some services)**
    - Applies to specific resources, such as:
        - arn:aws:s3:::**bucketname/keyname**
        - arn:aws:s3:::my_website/images/header.jpg
        - arn:aws:s3:::my_website/images/*
4. Condition (optional)

Authorization Policy Documents

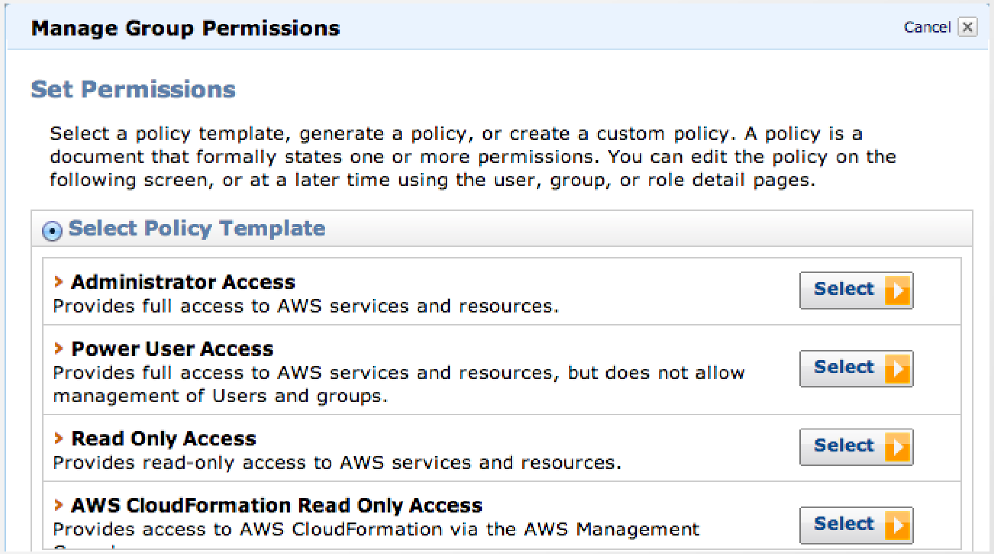
1. JSON format
2. Action (API)
3. Resource (some services)
4. **Condition (optional)**
    - Applies to specific conditions, such as:
        - SSL required
        - Request must originate from specific IP range (CIDR)
        - Rquest requires MFA
        - Request valid until (or after) some date/time

Identity, Authentication, and Authorization | Authorization Policies

```
{ "Statement” : [
   {
      "Effect” : “Allow",
      "Action” : “s3:Get*",
      "Resource” : “arn:aws:s3:::my-bucket/secure/*",
      "Condition” : {
         "IpAddress” : {
            "aws:SourceIp” : [ "174.128.53.0/24” ]
         } }
   },
   { ANOTHER STATEMENT… } ] }
```

Creating a policy document:

- Use Pre-defined policies
  - In IAM Management console at **console.aws.amazon.com/iam**



Manage Group Permissions

Cancel

Set Permissions

Select a policy template, generate a policy, or create a custom policy. A policy is a document that formally states one or more permissions. You can edit the policy on the following screen, or at a later time using the user, group, or role detail pages.

Select Policy Template

Administrator Access
Provides full access to AWS services and resources. Select

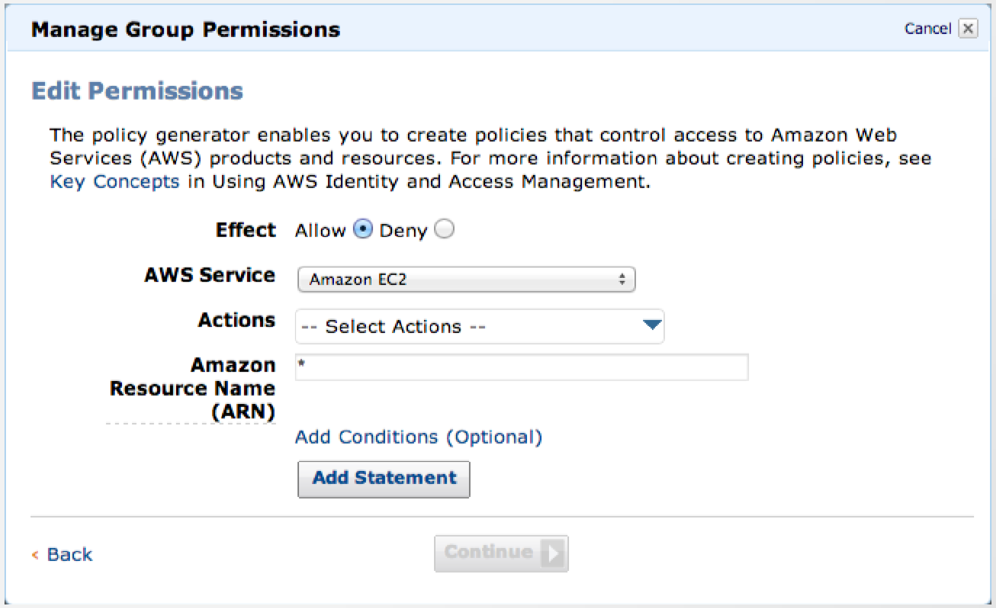
Power User Access
Provides full access to AWS services and resources, but does not allow management of Users and groups. Select

Read Only Access
Provides read-only access to AWS services and resources. Select

AWS CloudFormation Read Only Access
Provides access to AWS CloudFormation via the AWS Management Select

Creating a policy document:

- Use Policy Generator UI
    - In IAM Management console at **console.aws.amazon.com/iam**



Manage Group Permissions

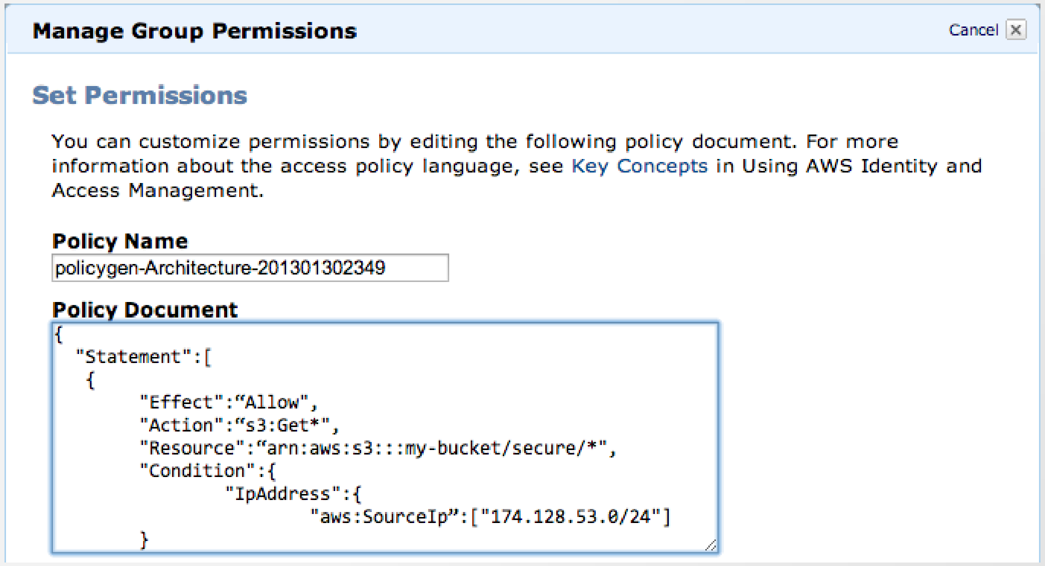
Edit Permissions

The policy generator enables you to create policies that control access to Amazon Web Services (AWS) products and resources. For more information about creating policies, see Key Concepts in Using AWS Identity and Access Management.

Effect	Allow [✓ ●] Deny ○
AWS Service	Amazon EC2
Actions	-- Select Actions --
Amazon Resource Name (ARN)	*

Creating a policy document:

- Define custom policies
    - In IAM Management Console or APIs



Set Permissions

You can customize permissions by editing the following policy document. For more information about the access policy language, see Key Concepts in Using AWS Identity and Access Management.

Policy Name

policygen-Architecture-201301302349

Policy Document

```
{
  "Statement":[
   {
        "Effect":"Allow",
        "Action":"s3:Get*",
        "Resource":"arn:aws:s3:::my-bucket/secure/*",
        "Condition":{
                "IpAddress":{
                        "aws:SourceIp":["174.128.53.0/24"]
        }
```

Identity, Authentication, and Authorization | Temporary Credentials

4

Temporary credentials with the Security Token Service

Security Token Service

- Generate temporary credentials for an IAM User or for users that you authenticate (federated users). Useful for improving security posture, mobile applications, and identity federation.

Identity, Authentication, and Authorization | Temporary Credentials

Temporary Credentials

- Access Key, Secret Key, and **Token**
- **Expire automatically** (15 minutes ~ 36 hours)

IAM Users

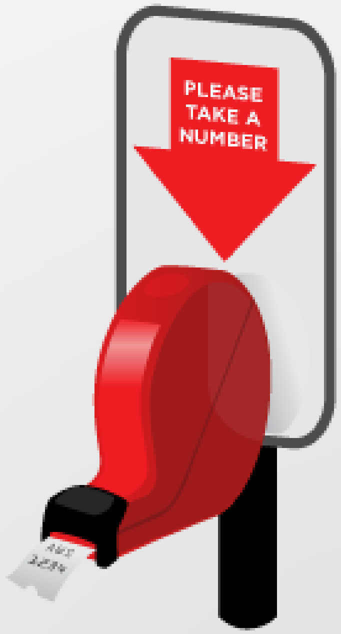
- **Can create temporary credentials for themselves**

Federated Users

- Authenticate users to your identity store
- **SSO** to AWS Management Console
- Enhanced security for **mobile applications**

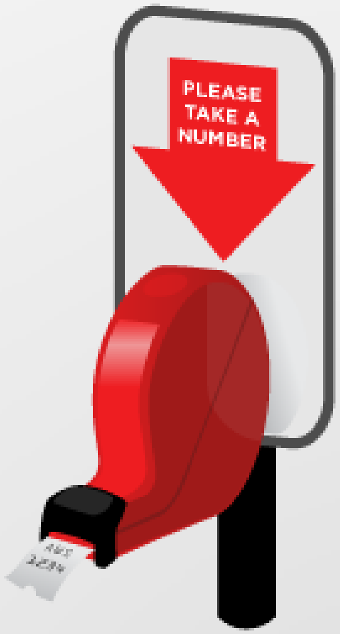
Roles

- Allow trusted **entity** to **assume role**
- **Entity = EC2 Instance(s),** or an **IAM user in another account**



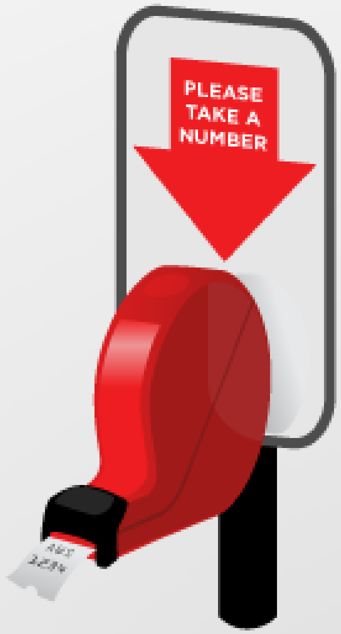
Identity, Authentication, and Authorization | Temporary Credentials

Temporary Credentials	• Access Key, Secret Key, and **Token** • **Expire automatically** (15 minutes ~ 36 hours)
IAM Users	• **Can create temporary credentials for themselves**
Federated Users	• Authenticate users to your identity store • **SSO** to AWS Management Console • Enhanced security for **mobile applications**
Roles	• Allow trusted **entity** to **assume role** • **Entity = EC2 Instance(s),** or an **IAM user in another account**



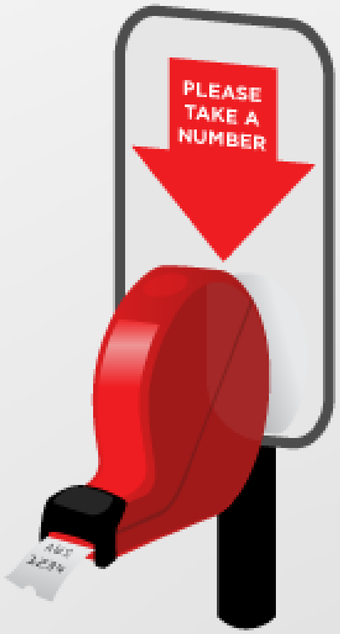
Identity, Authentication, and Authorization | Temporary Credentials

Temporary Credentials	• Access Key, Secret Key, and **Token** • **Expire automatically** (15 minutes ~ 36 hours)
IAM Users	• **Can create temporary credentials for themselves**
Federated Users	• Authenticate users to your identity store • **SSO** to AWS Management Console • Enhanced security for **mobile applications**
Roles	• Allow trusted **entity** to **assume role** • **Entity = EC2 Instance(s),** or an **IAM user in another account**



Identity, Authentication, and Authorization | Temporary Credentials

Temporary Credentials	• Access Key, Secret Key, and **Token** • **Expire automatically** (15 minutes ~ 36 hours)
IAM Users	• **Can create temporary credentials for themselves**
Federated Users	• Authenticate users to your identity store • **SSO** to AWS Management Console • Enhanced security for **mobile applications**
Roles	• Allow trusted **entity** to **assume role** • **Entity = EC2 Instance(s),** or an **IAM user in another account**

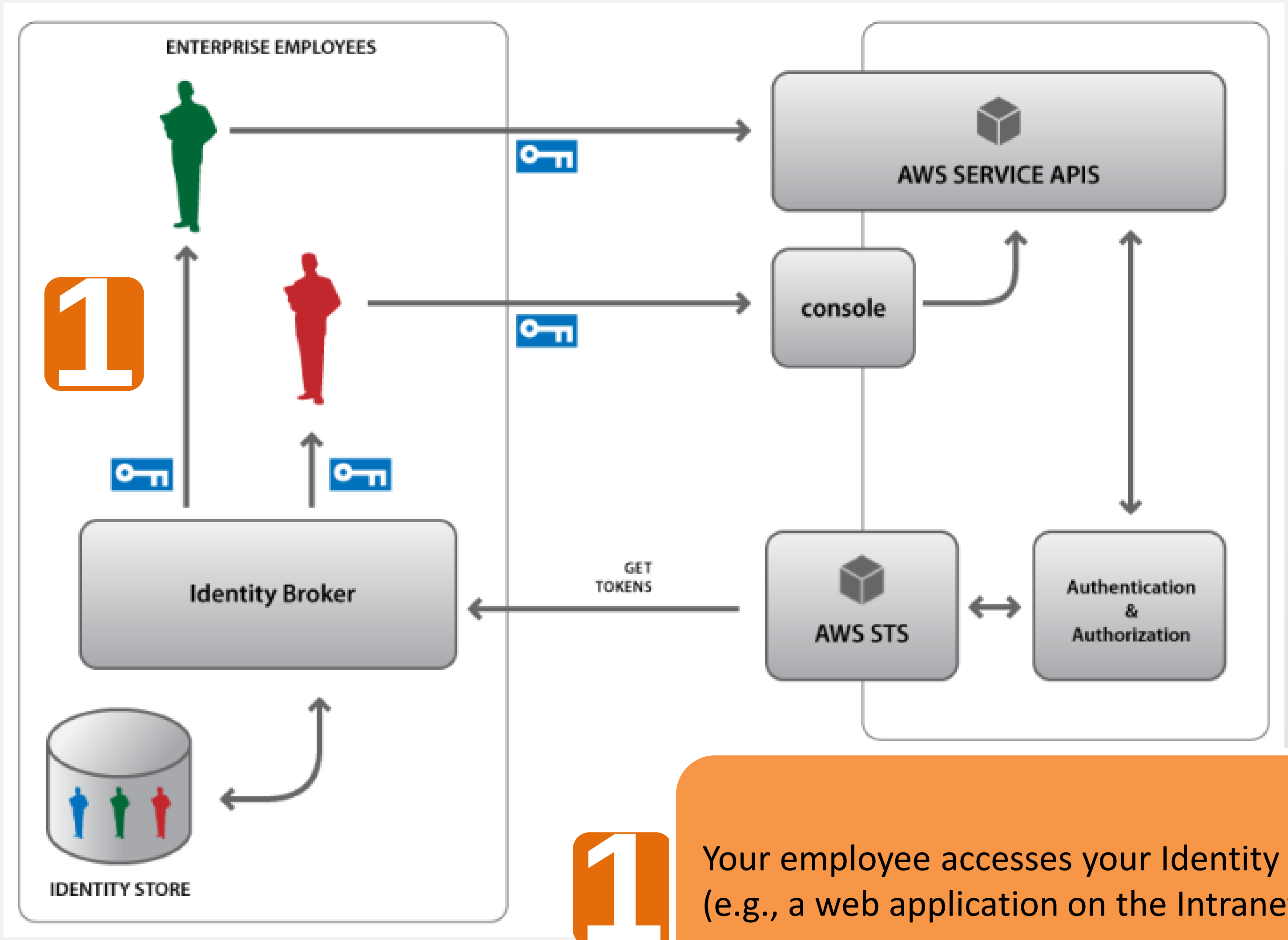


Federated Users

- **Authenticate users to your own identity store**

- You write an "identity broker application"
- Users authenticate to your identity broker
- Your identity broker provisions temporary credentials via STS
- **SSO**: Temporary credentials can be used to sign user directly into the AWS Management Console
- **Let's look at an example…**

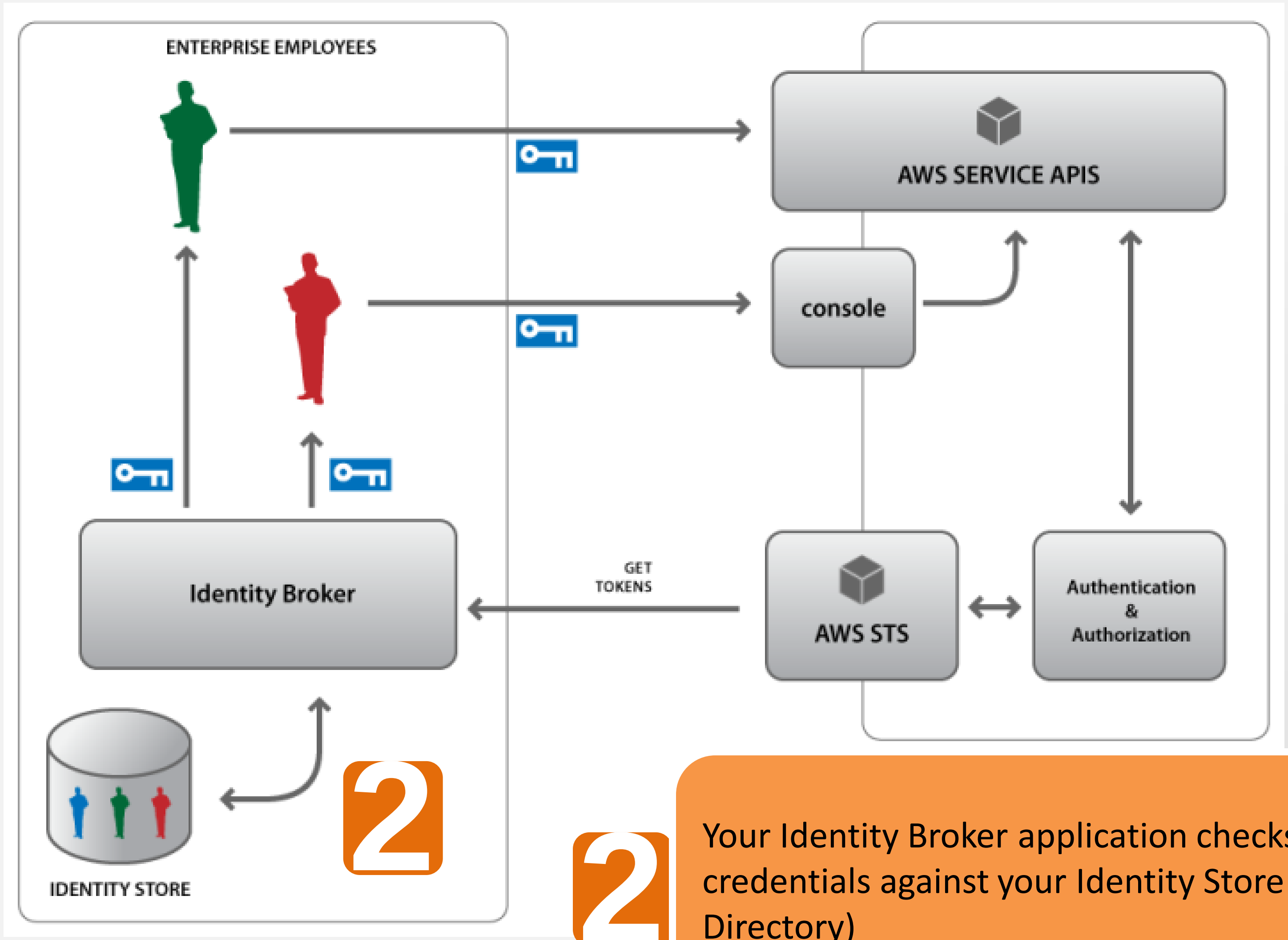
Identity, Authentication, and Authorization | Temporary Credentials



1

Your employee accesses your Identity Broker application (e.g., a web application on the Intranet)

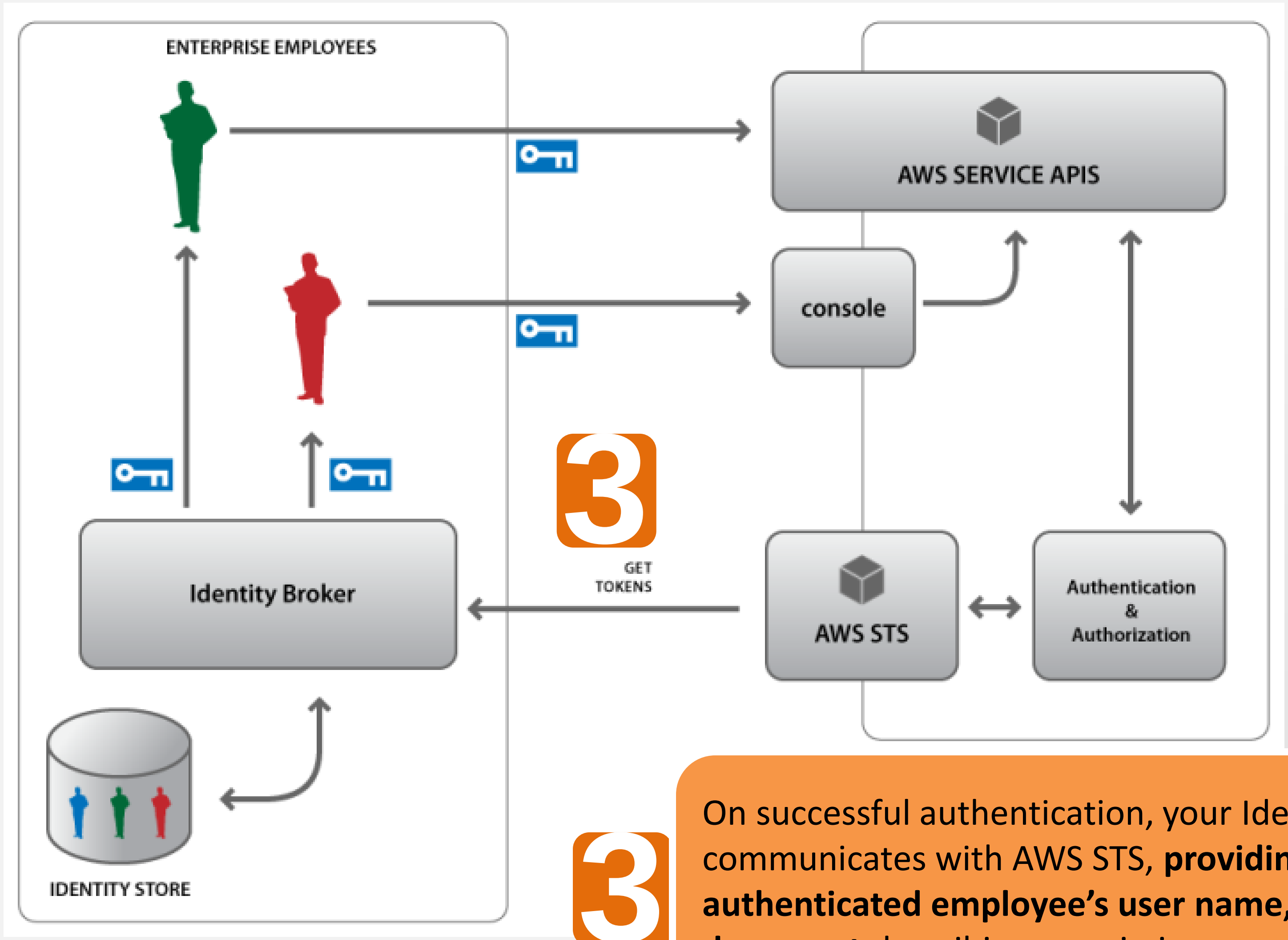
Identity, Authentication, and Authorization | Temporary Credentials



2

Your Identity Broker application checks your employee's credentials against your Identity Store (e.g., Active Directory)

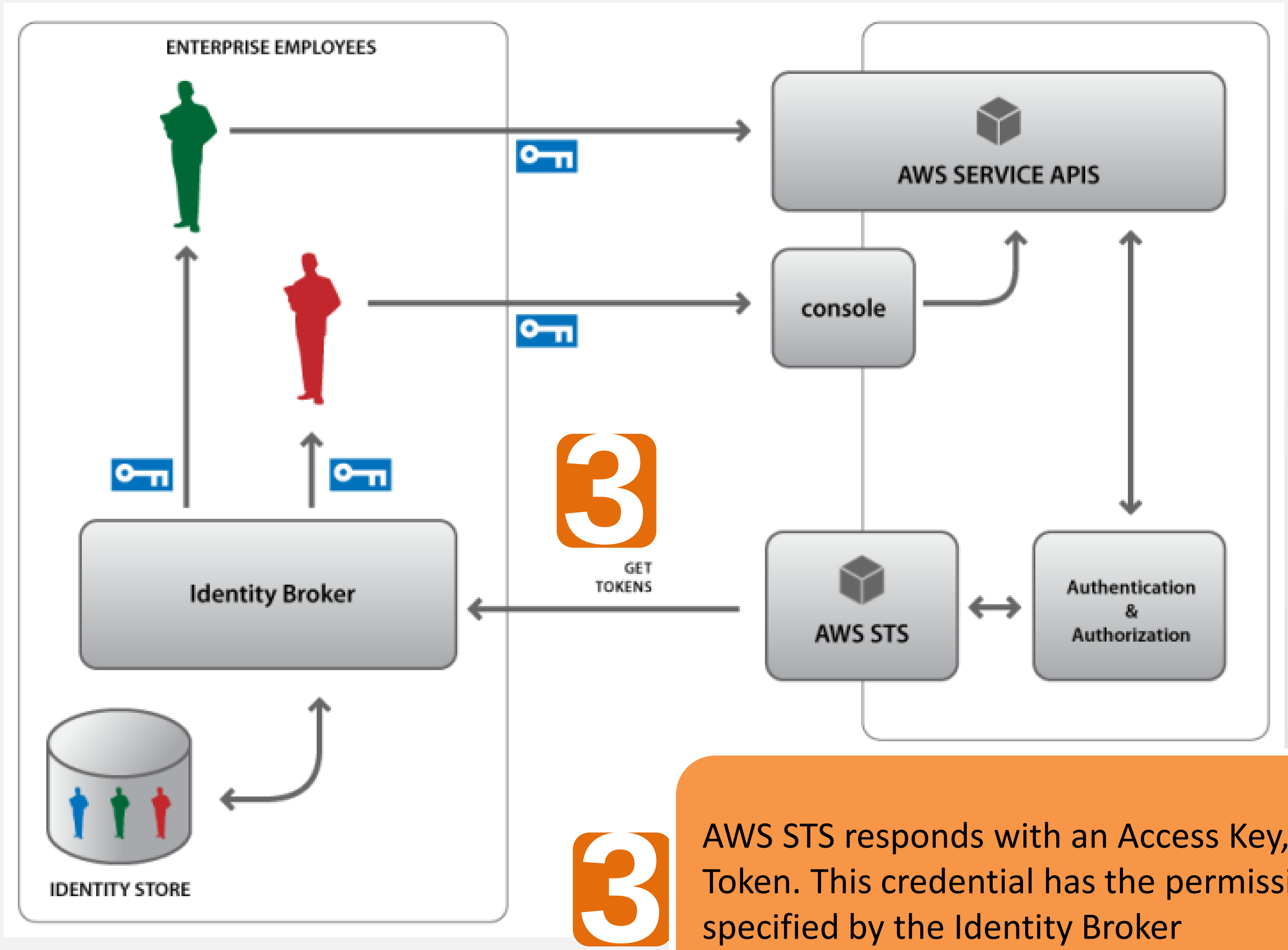
Identity, Authentication, and Authorization | Temporary Credentials



3

On successful authentication, your Identity Broker communicates with AWS STS, **providing the authenticated employee's user name, a policy document** describing permissions, and a timeout.

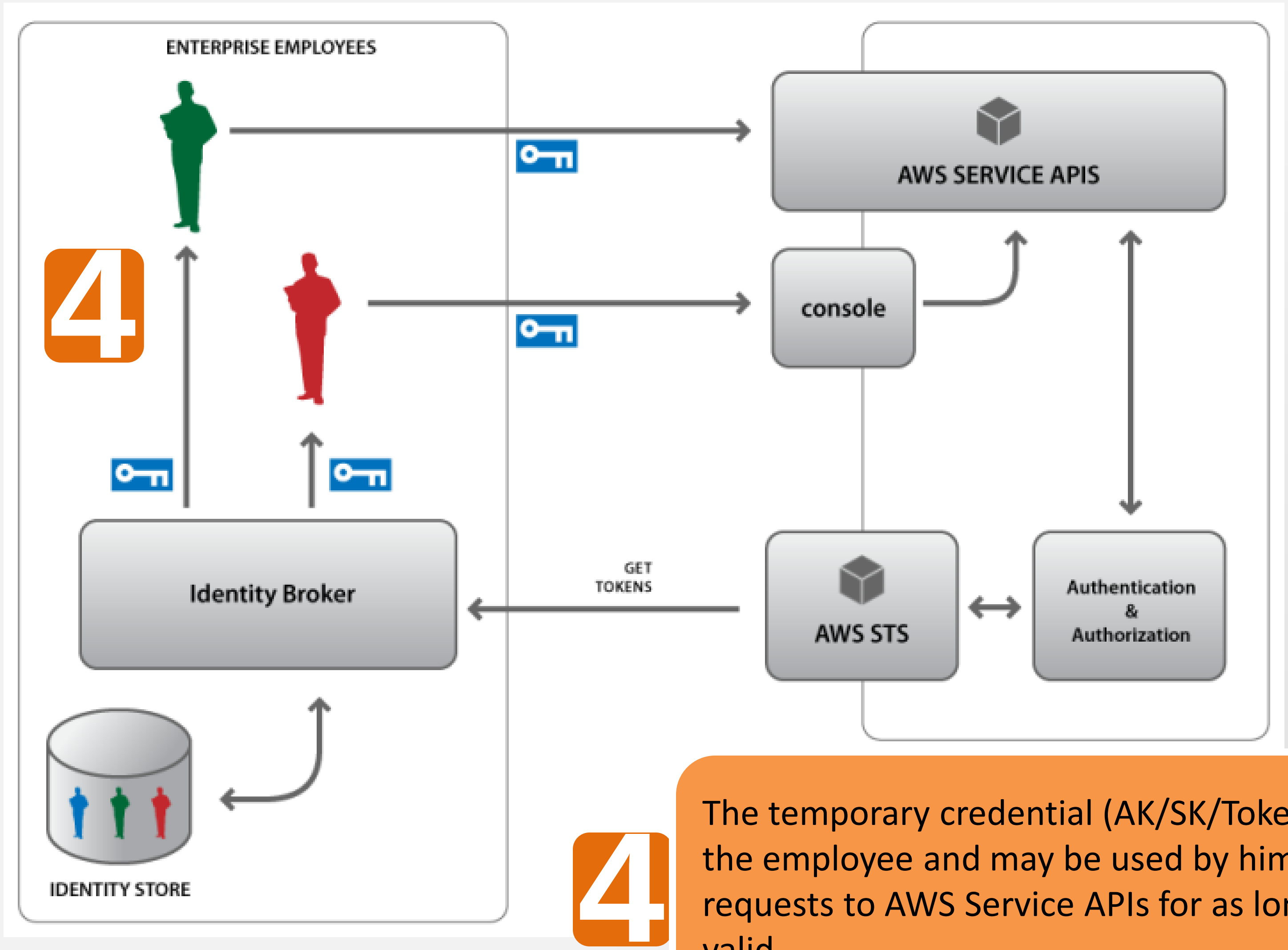
Identity, Authentication, and Authorization | Temporary Credentials



3

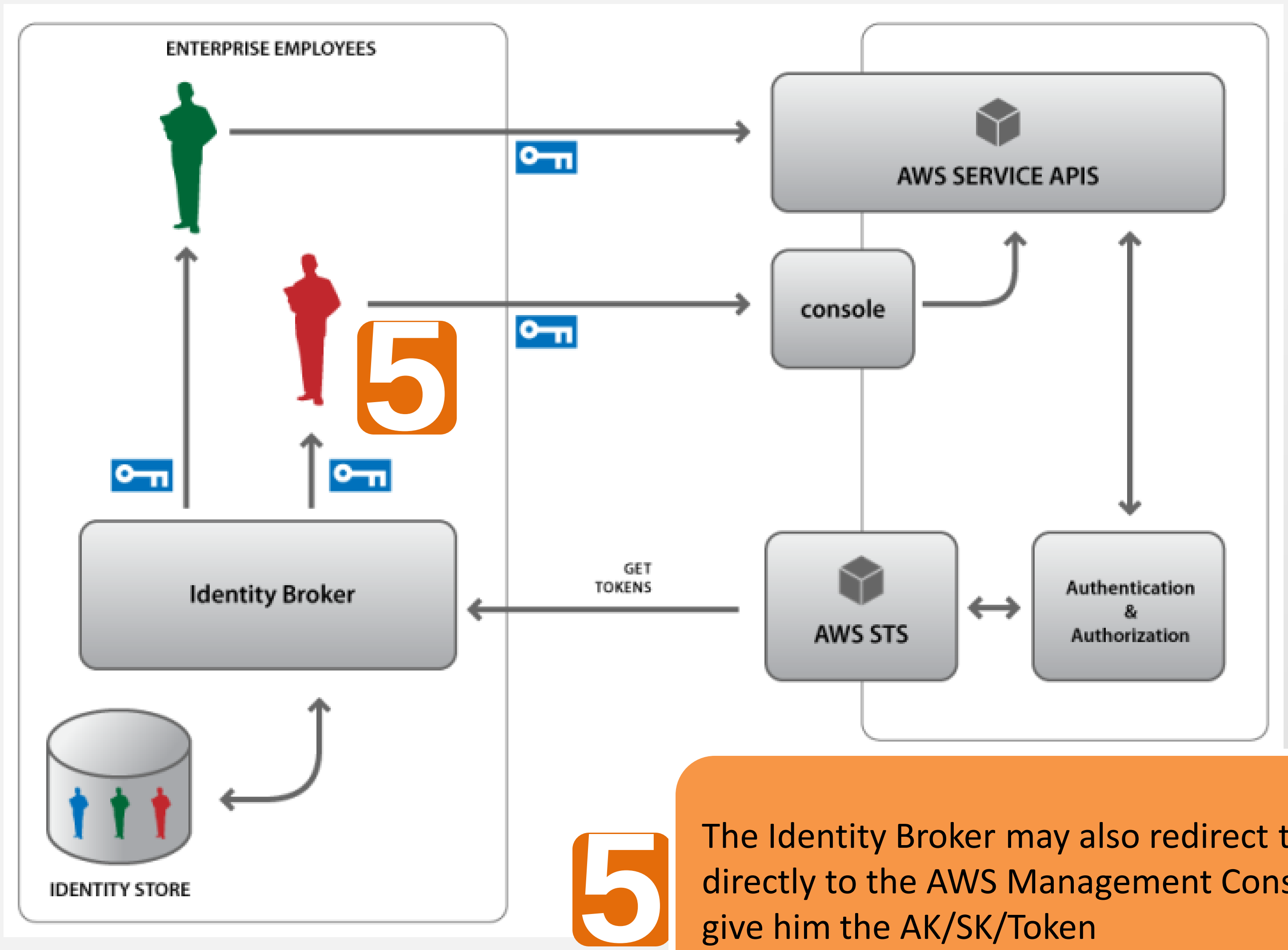
AWS STS responds with an Access Key, Secret Key, and Token. This credential has the permissions and expiration specified by the Identity Broker

Identity, Authentication, and Authorization | Temporary Credentials



The temporary credential (AK/SK/Token) is delivered to the employee and may be used by him to authenticate requests to AWS Service APIs for as long as they are valid.

Identity, Authentication, and Authorization | Temporary Credentials



The Identity Broker may also redirect the employee directly to the AWS Management Console rather than give him the AK/SK/Token

Identity, Authentication, and Authorization | Service-specific, OS, and application authentication

5

Service-specific, OS, and application authentication

Service-specific policy documents

S3

SNS

SQS

Some services have additional mechanisms for authentication and authorization.

You may apply authorization policy documents to these individual services

For example, a policy applied to an S3 bucket may make some objects publically readable

Example S3 Bucket Policy

```
{
    "Statement":[{
    "Sid":"AddCannedAcl",
    "Effect":"Allow",
      "Principal": {
             "AWS": [
                   "arn:aws:iam::111122223333:root",
                   "arn:aws:iam::444455556666:root"
             ]
          },
      "Action":["s3:PutObject","s3:PutObjectAcl"
       ],
       "Resource":["arn:aws:s3:::bucket/*"
       ],
       "Condition":{
         "StringEquals":{
           "s3:x-amz-acl":["public-read"]
         }
       }
     }
   ]
}
```

Allow...

Example S3 Bucket Policy

```
{
    "Statement":[{
    "Sid":"AddCannedAcl”,
    "Effect":"Allow",
      "Principal": {
             "AWS": [
                   "arn:aws:iam::111122223333:root",
                   "arn:aws:iam::444455556666:root”
             ]
          },
      "Action":["s3:PutObject","s3:PutObjectAcl"
       ],
       "Resource":["arn:aws:s3:::bucket/*"
       ],
       "Condition":{
         "StringEquals":{
           "s3:x-amz-acl":["public-read"]
         }
       }
     }
   ]
}
```

…these two AWS accounts (i.e., principals)…

Example S3 Bucket Policy

```
{
    "Statement":[{
    "Sid":"AddCannedAcl”,
    "Effect":"Allow",
      "Principal": {
             "AWS": [
                   "arn:aws:iam::111122223333:root",
                   "arn:aws:iam::444455556666:root”
             ]
          },
      "Action":["s3:PutObject","s3:PutObjectAcl"
       ],
       "Resource":["arn:aws:s3:::bucket/*"
       ],
       "Condition":{
         "StringEquals":{
           "s3:x-amz-acl":["public-read"]
         }
       }
     }
   ]
}
```

…to make these two S3 API calls…

Example S3 Bucket Policy

```
{
    "Statement":[{
    "Sid":"AddCannedAcl”,
    "Effect":"Allow",
      "Principal": {
             "AWS": [
                   "arn:aws:iam::111122223333:root",
                   "arn:aws:iam::444455556666:root”
             ]
         },
      "Action":["s3:PutObject","s3:PutObjectAcl"
       ],
       "Resource":["arn:aws:s3:::bucket/*"
       ],
       "Condition":{
         "StringEquals":{
           "s3:x-amz-acl":["public-read"]
         }
       }
     }
   ]
}
```

…anywhere in this bucket…

Example S3 Bucket Policy

```
{
    "Statement":[{
    "Sid":"AddCannedAcl”,
    "Effect":"Allow",
      "Principal": {
             "AWS": [
                   "arn:aws:iam::111122223333:root",
                   "arn:aws:iam::444455556666:root”
             ]
          },
      "Action":["s3:PutObject","s3:PutObjectAcl"
       ],
       "Resource":["arn:aws:s3:::bucket/*"
       ],
       "Condition":{
         "StringEquals":{
           "s3:x-amz-acl":["public-read"]
         }
       }
     }
   ]
}
```

…as long as the object they’re trying to put includes the ‘public-read’ ACL.

Operating System Authentication

Initial OS login restricted by key pair

- You maintain the private key (.pem file)
- **Has nothing to do with IAM**
- After initial login, you can implement your own authentication system (Active Directory, etc.)

RDS Database Authentication

RDS database has its own username/password

- Manage username/password with AWS Service APIs
- Connect to database with username/password using normal conventions (such as JDBC)
- **Has nothing to do with IAM**

Application Authentication

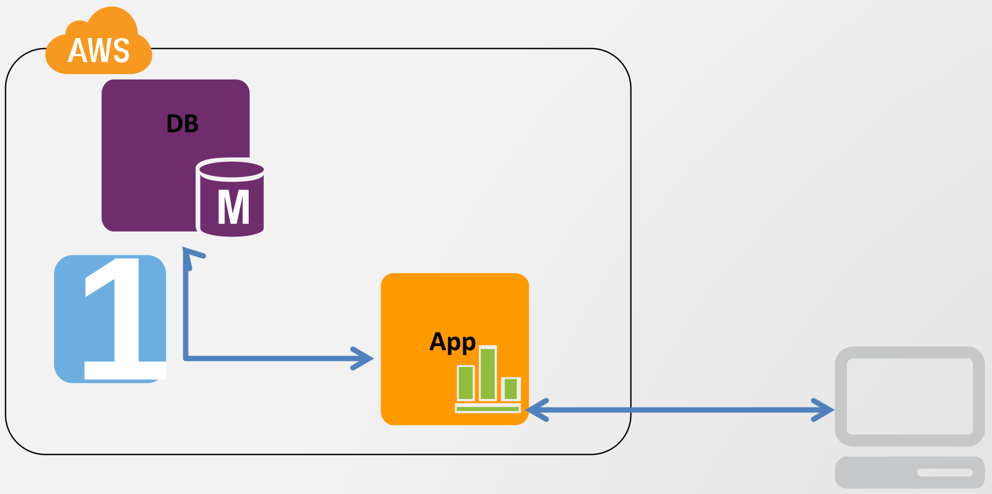
Application authentication up to the user

- IAM is strictly for authenticating/authorizing AWS Service APIs. **IAM is not suitable for application authentication**.
- Let’s look at 3 possibilities for application authentication in EC2…

Application Authentication

Application authentication up to the user

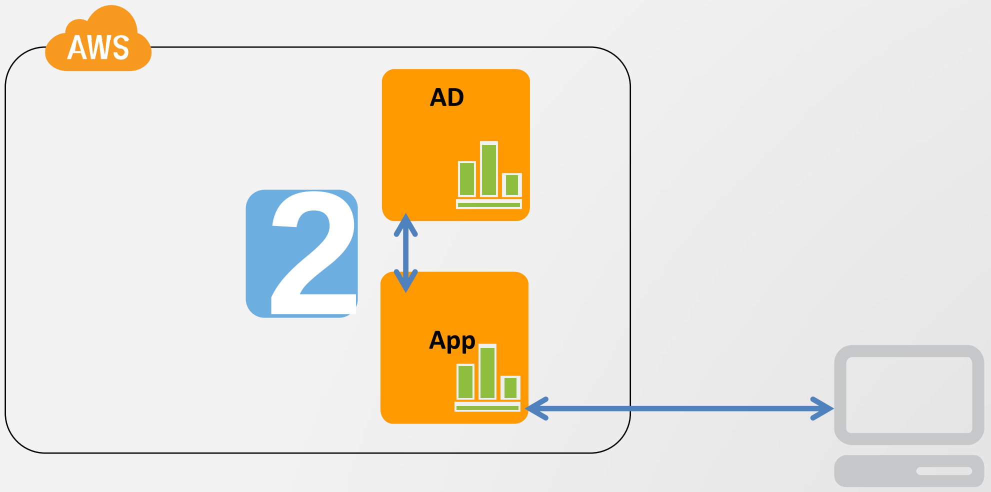
1. Use a local database for credentials and application roles.



Application Authentication

Application authentication up to the user

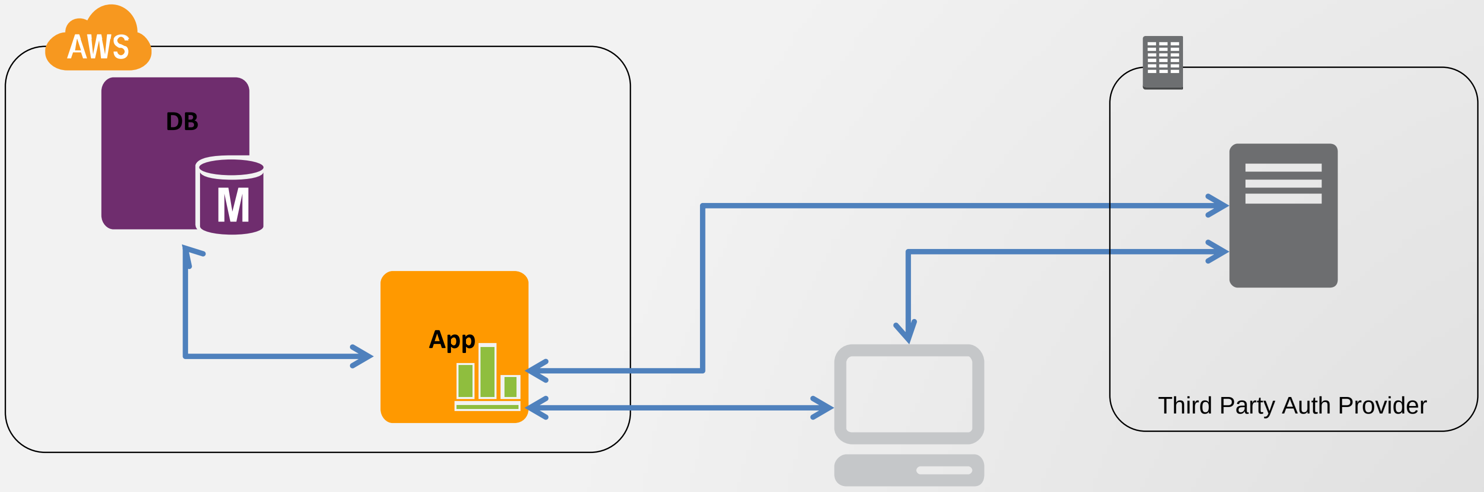
2. Use Active Directory or LDAP to store application credentials and roles



Application Authentication

Application authentication up to the user

3. Integrate the application with a third-party provider (such as OAuth 2.0) and store roles in a local database



Service-specific, OS, and application authentication

For review:

- What are the 3 major realms where authentication and authorization occur within AWS?
- What credentials are required to use an AWS API? The Management Console?
- What is the role of a policy document? How can you create one?
- What service allows federated access to AWS?
- Can you use IAM for application authentication?

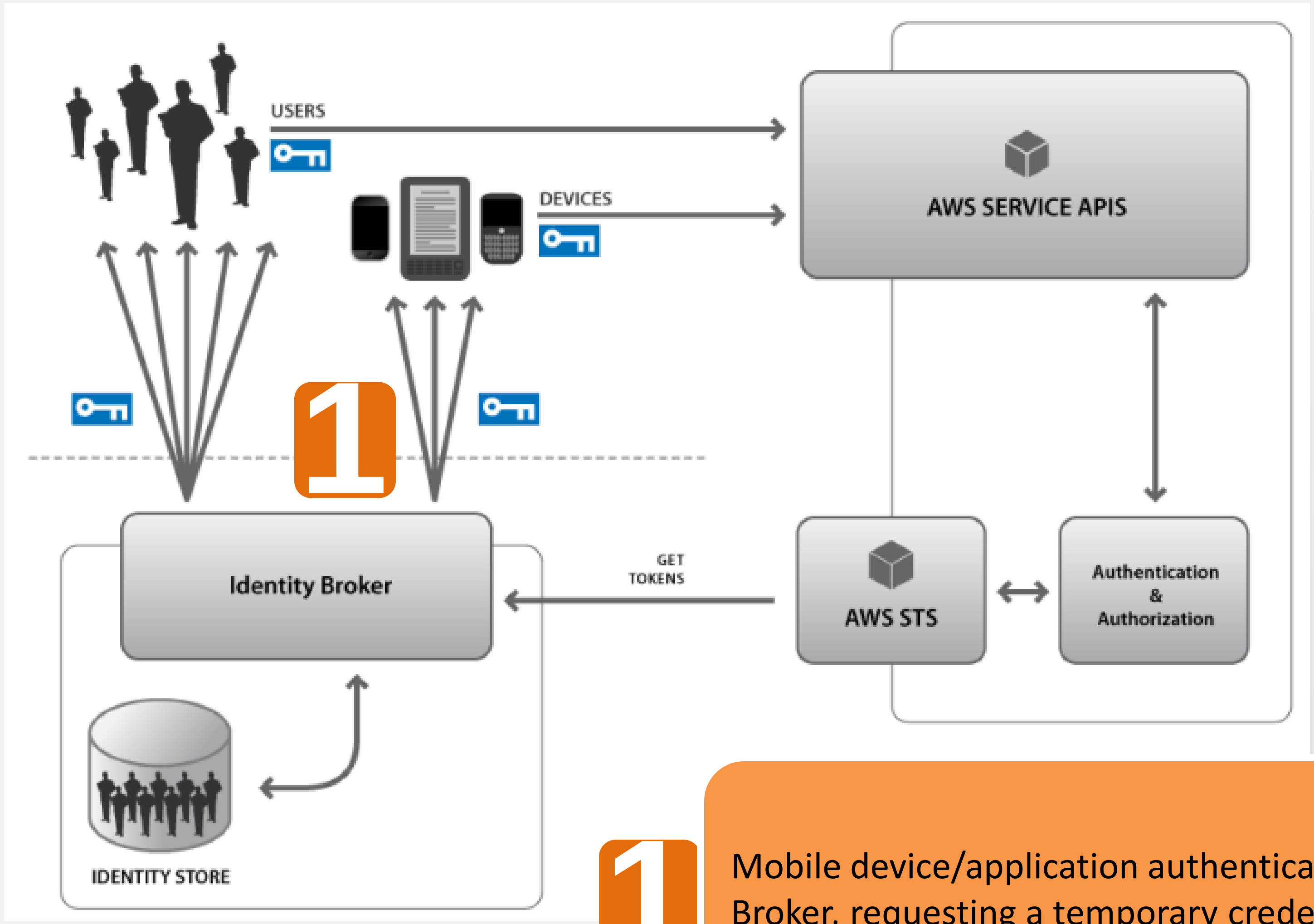
Appendix

Federated Users

- **Allow mobile devices to communicate directly with AWS**

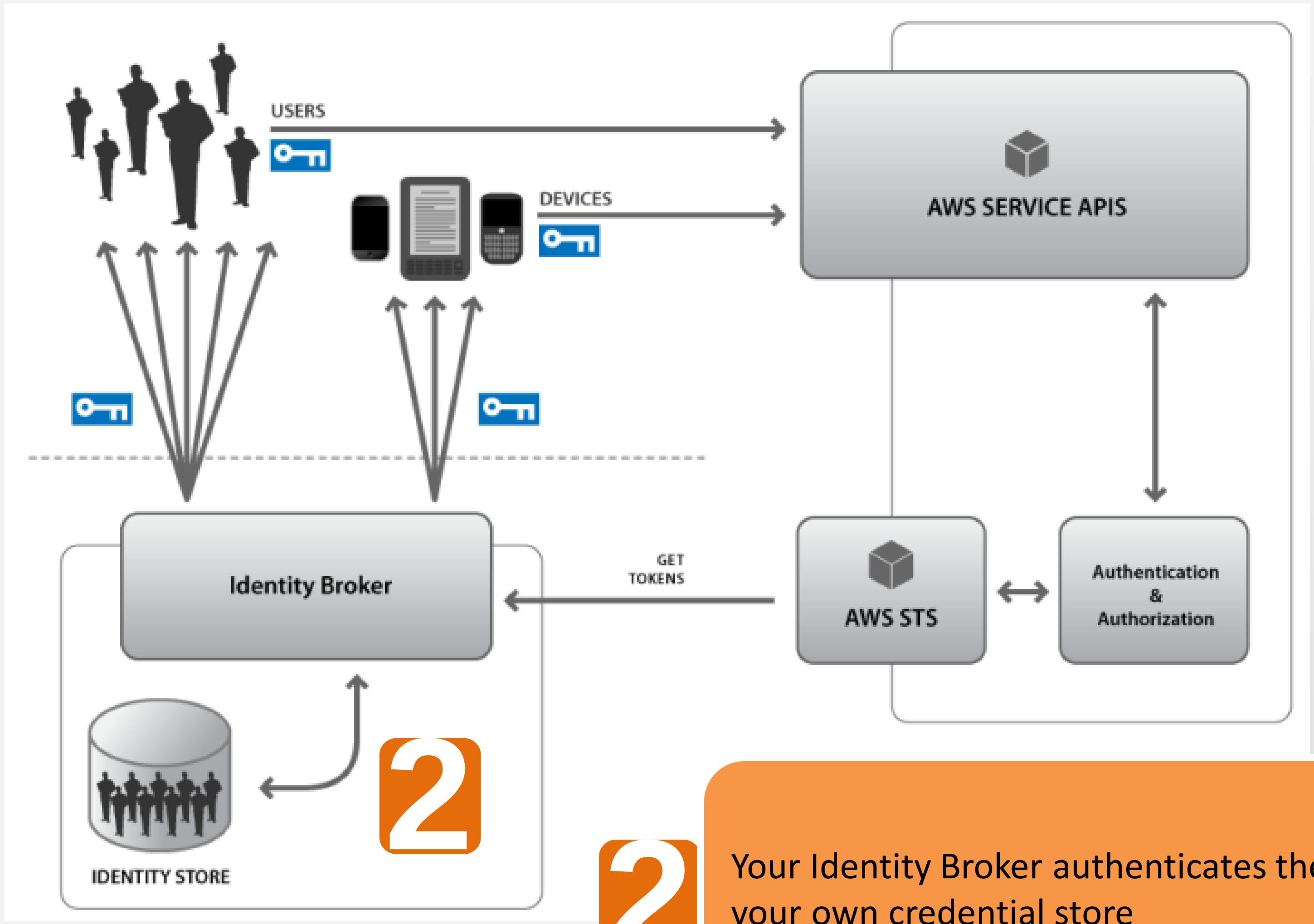
- **Example:** allow a mobile application to upload photos or video directly to S3
- No limit to number of credentials that you can generate
- **Expire automatically**
- **Let's look at an example**

Identity, Authentication, and Authorization | Temporary Credentials



1 Mobile device/application authenticates to your Identity Broker, requesting a temporary credential

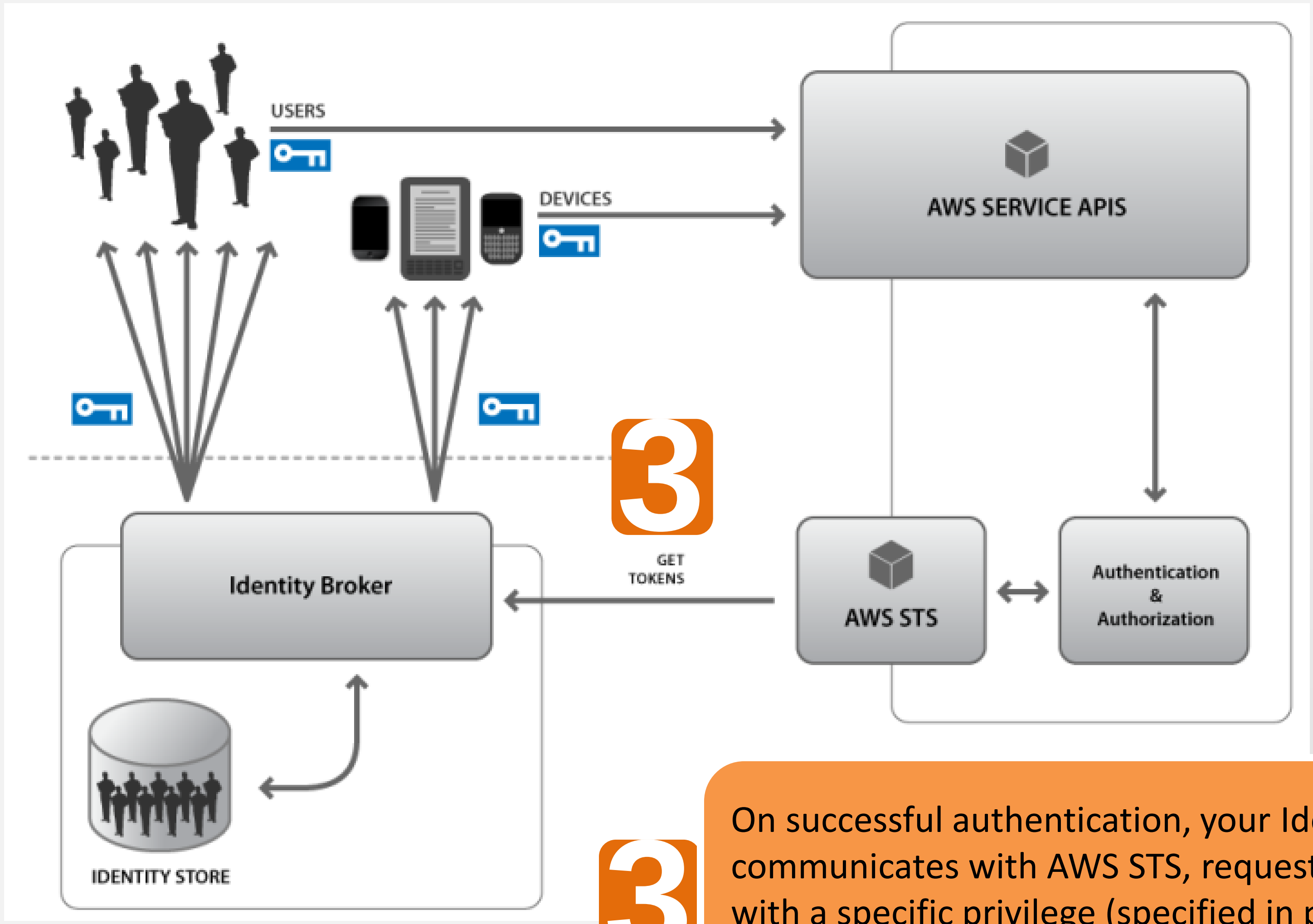
Identity, Authentication, and Authorization | Temporary Credentials



2

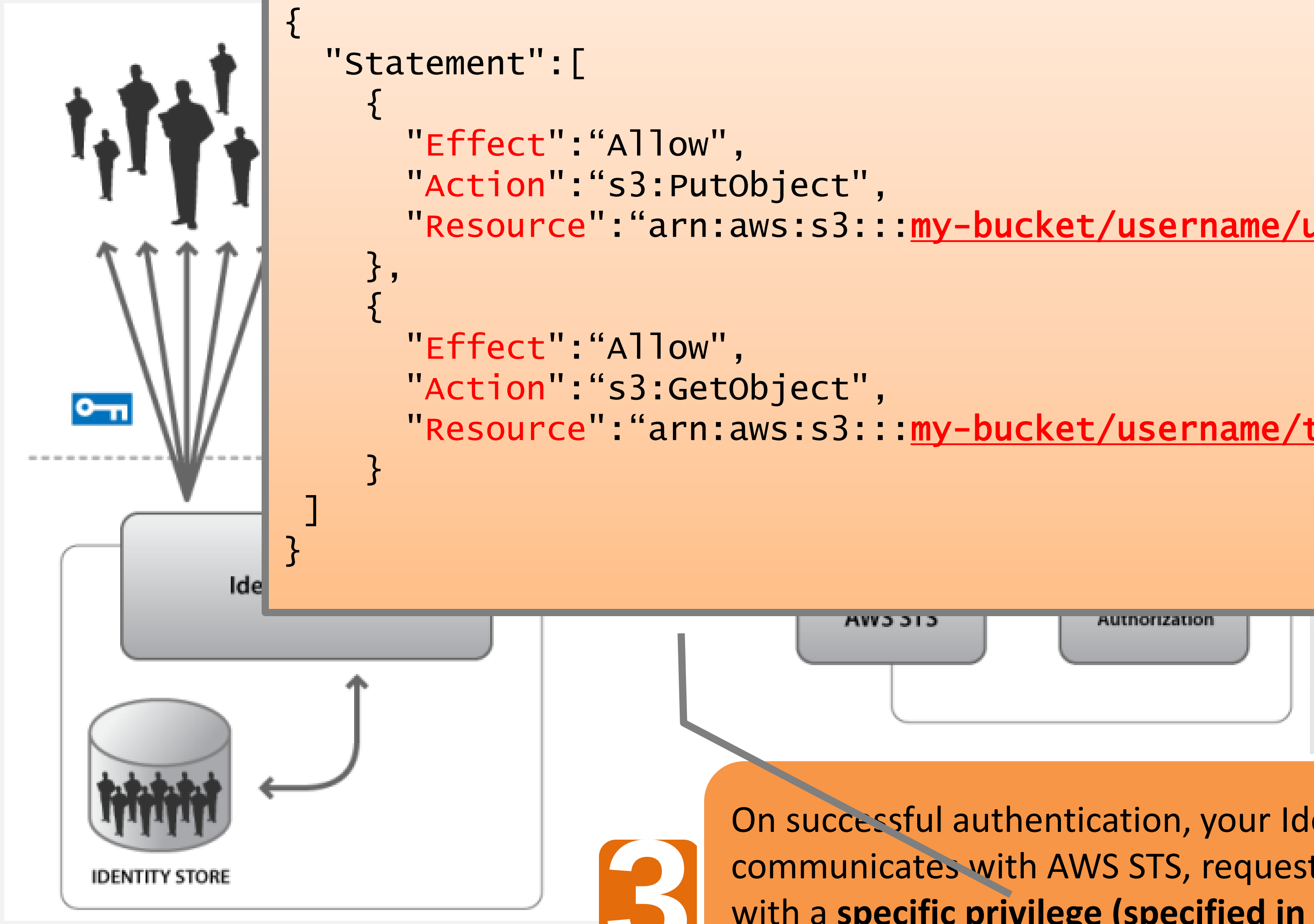
Your Identity Broker authenticates the device against your own credential store

Identity, Authentication, and Authorization | Temporary Credentials



3 On successful authentication, your Identity Broker communicates with AWS STS, requesting a credential with a specific privilege (specified in policy), and a timeout

Identity, Authentication, and Authorization | Temporary Credentials

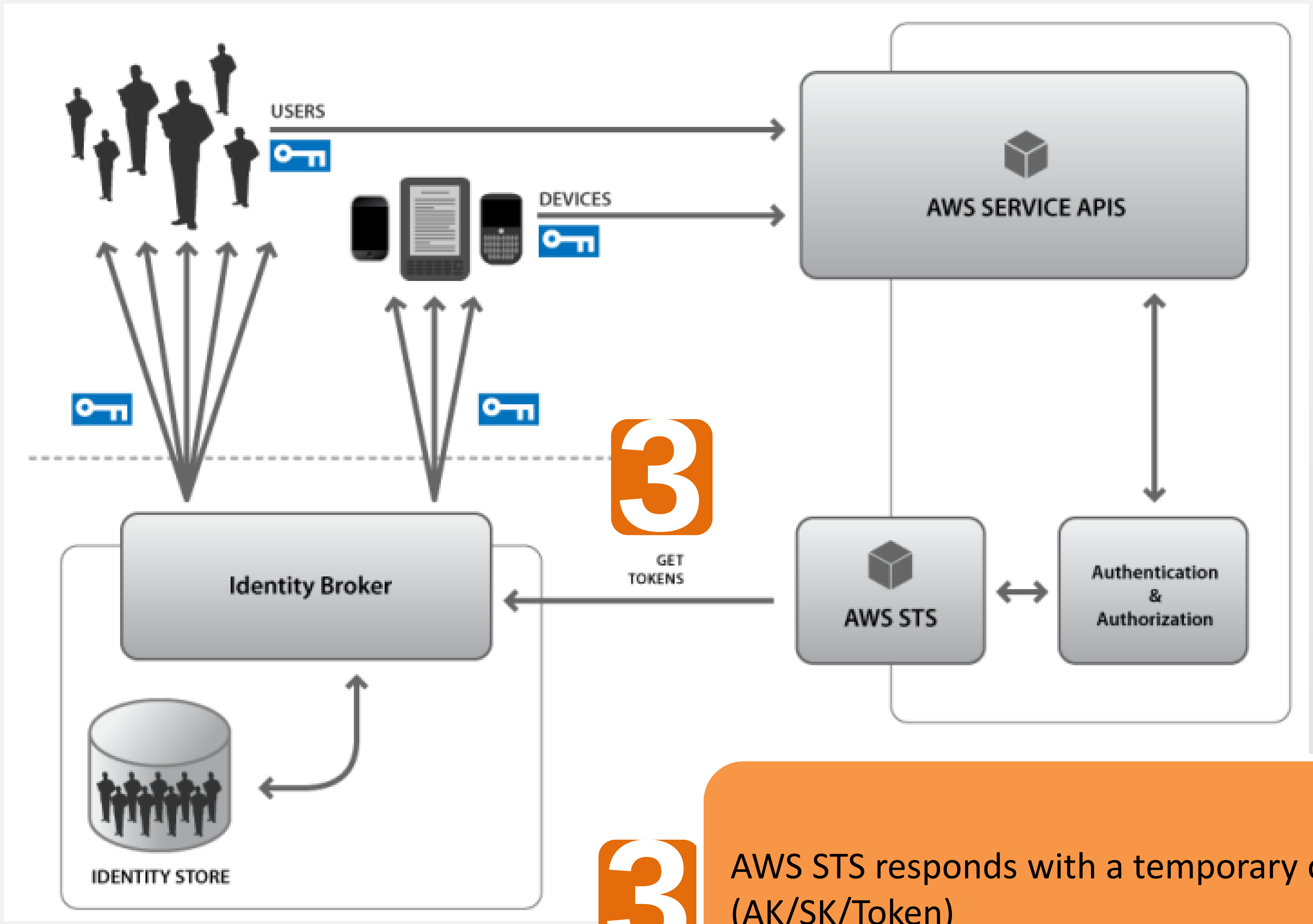


```
{
  "Statement":[
    {
      "Effect":“Allow",
      "Action":“s3:PutObject",
      "Resource":“arn:aws:s3:::my-bucket/username/upload/*",
    },
    {
      "Effect":“Allow",
      "Action":“s3:GetObject",
      "Resource":“arn:aws:s3:::my-bucket/username/thumbnails/*",
    }
  ]
}
```

3

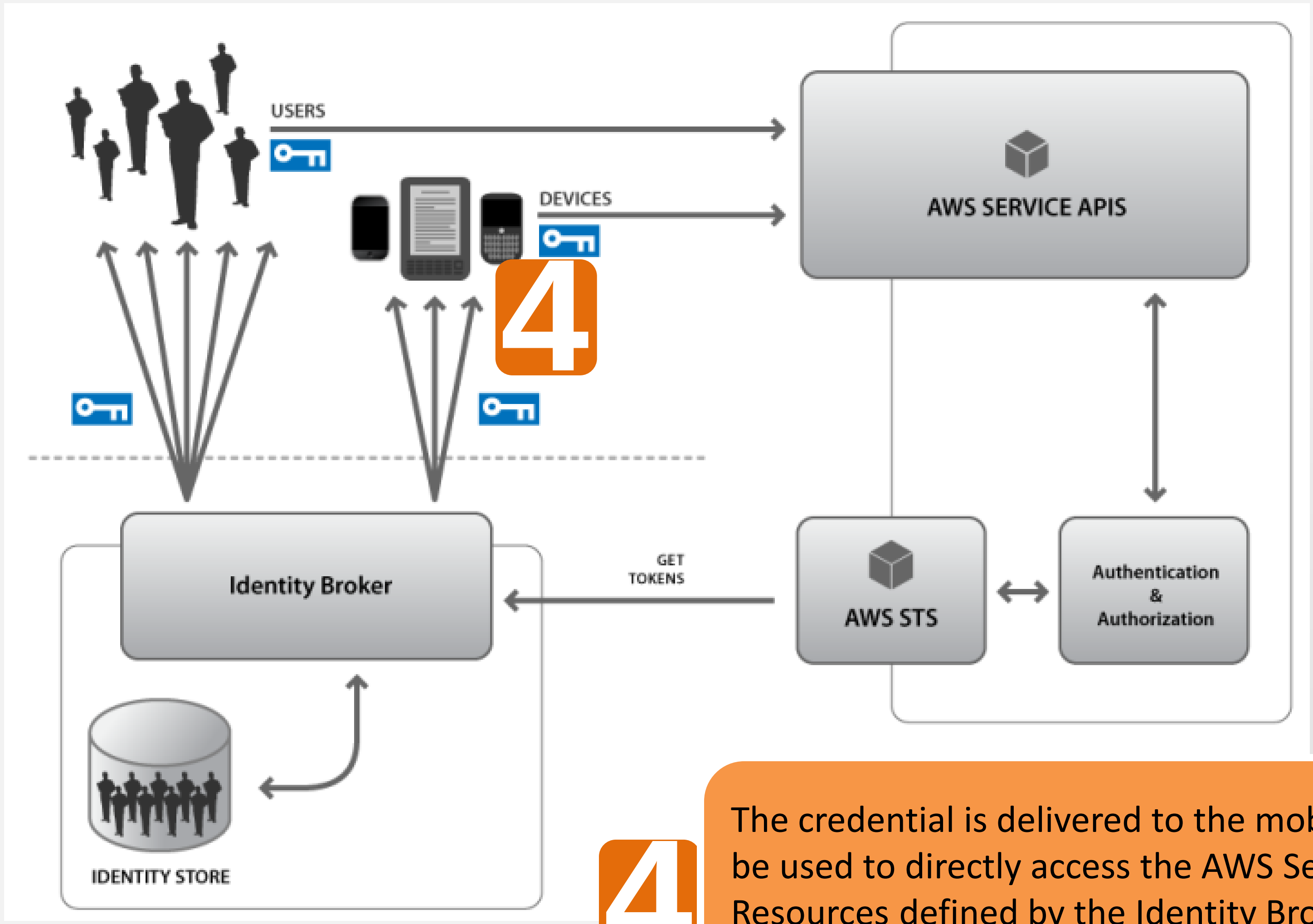
On successful authentication, your Identity Broker communicates with AWS STS, requesting a credential with a **specific privilege (specified in policy)**, and a timeout

Identity, Authentication, and Authorization | Temporary Credentials



3 AWS STS responds with a temporary credential (AK/SK/Token)

Identity, Authentication, and Authorization | Temporary Credentials



4

The credential is delivered to the mobile device and may be used to directly access the AWS Service APIs and Resources defined by the Identity Broker (e.g., Put and Get specific items to a specific S3 bucket)