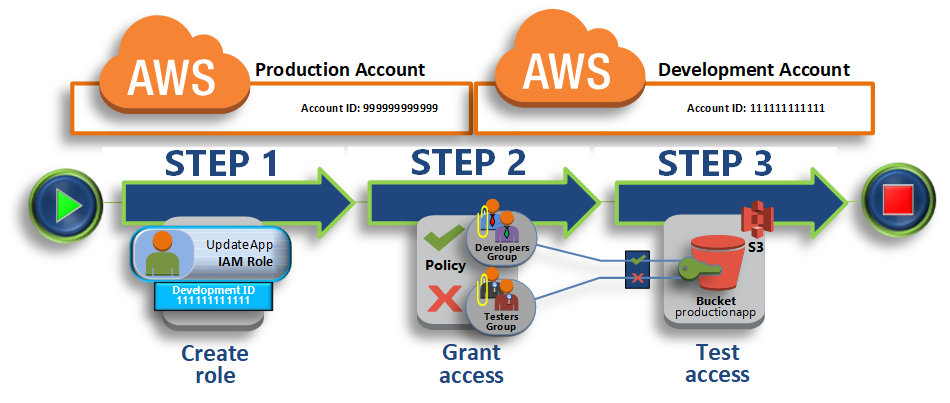
**Delegate Access Across AWS Accounts Using IAM Roles**

This tutorial teaches you how to use a role to delegate access to resources that are in different AWS accounts that you own (Production and Development). You share resources in one account with users in a different account. By setting up cross-account access in this way, you don't need to create individual IAM users in each account. In addition, users don't have to sign out of one account and sign into another in order to access resources that are in different AWS accounts. After configuring the role, you see how to use the role from the AWS Management Console, the AWS CLI, and the API.

In this tutorial, imagine that the Production account is where live applications are managed, and the Development account is a sandbox where developers and testers can freely test applications. In each account, application information is stored in Amazon S3 buckets. You manage IAM users in the Development account, where you have two IAM groups: Developers and Testers. Users in both groups have permissions to work in the Development account and access resources there. From time to time, a developer must update the live applications in the Production account. These applications are stored in an Amazon S3 bucket called productionapp.

At the end of this tutorial, you have a role in the Production account (the trusting account) that allows users from the Development account (the trusted account) to access the productionapp bucket in the Production account. Developers can use the role in the AWS Management Console to access the productionapp bucket in the Production account. They can also access the bucket by using API calls that are authenticated by temporary credentials provided by the role. Similar attempts by a Tester to use the role fail.

This workflow has three basic steps.



[**Step 1 - Create a Role**](https://docs.aws.amazon.com/IAM/latest/UserGuide/tutorial_cross-account-with-roles.html#tutorial_cross-account-with-roles-1)

First, you use the AWS Management Console to establish trust between the Production account (ID number 999999999999) and the Development account (ID number 111111111111). You start by creating an IAM role named *UpdateApp*. When you create the role, you define the Development account as a trusted entity and specify a permissions policy that allows trusted users to update the productionapp bucket.

**[Step 2 - Grant Access to the Role](https://docs.aws.amazon.com/IAM/latest/UserGuide/tutorial_cross-account-with-roles.html" \l "tutorial_cross-account-with-roles-2)**

In this step of the tutorial, you modify the IAM group policy so that Testers are denied access to the UpdateApp role. Because Testers have PowerUser access in this scenario, we must explicitly deny the ability to use the role.

**[Step 3 - Test Access by Switching Roles](https://docs.aws.amazon.com/IAM/latest/UserGuide/tutorial_cross-account-with-roles.html" \l "tutorial_cross-account-with-roles-3)**

Finally, as a Developer, you use the UpdateApp role to update the productionapp bucket in the Production account. You see how to access the role through the AWS console, the AWS CLI, and the API.

**Prerequisites**

This tutorial assumes that you have the following already in place:

* Two separate AWS accounts that you can use, one to represent the Development account, and one to represent the Production account.
* Users and groups in the Development account created and configured as follows:

|  |  |  |
| --- | --- | --- |
| **User** | **Group** | **Permissions** |
| David | Developers | Both users are able to sign in and use the AWS Management Console in theDevelopment account. |
| Theresa | Testers |

* You do not need to have any users or groups created in the Production account.
* An Amazon S3 bucket created in the Production account. We call it ProductionApp in this tutorial, but because S3 bucket names must be globally unique, you must use a bucket with a different name.

**Step 1 - Create a Role**

To allow users from one AWS account to access resources in another AWS account, create a role that defines who can access it and what permissions it grants to users that switch to it.

In this step of the tutorial, you create the role in the Production account and specify the Development account as a trusted entity. You also limit the role's permissions to only read and write access to the productionapp bucket. Anyone who is granted permission to use the role can read and write to the productionapp bucket.

Before you can create a role, you need the account ID of the Development AWS account. The account ID is a unique identifier assigned to each AWS account.

**To obtain the Development AWS account ID**

1. Sign in to the AWS Management Console as an administrator of the Development account, and open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In navigation bar, choose **Support**, and then **Support Center**. The **Account Number** is in the upper right corner immediately below the **Support** menu. The account ID is a 12-digit number. For this scenario, we pretend the Development account ID is 111111111111; however, you should use a valid account ID if you are reconstructing the scenario in your test environment.

**To create a role in the Production account that can be used by the Development account**

1. Sign in to the AWS Management Console as an administrator of the Production account, and open the IAM console.
2. Before creating the role, prepare the managed policy that defines the permissions that the role requires. You attach this policy to the role in a later step.

You want to set read and write access to the productionapp bucket. Although AWS provides some Amazon S3 managed policies, there isn't one that provides read and write access to a single Amazon S3 bucket. You can create your own policy instead.

In the navigation pane on the left, choose **Policies** and then choose **Create policy**.

1. Choose the **JSON** tab and copy the text from the following JSON policy document. Paste this text into the **JSON** text box, replacing the resource ARN (arn:aws:s3:::productionapp) with the real one appropriate to your S3 bucket.
2. {
3. "Version": "2012-10-17",
4. "Statement": [
5. {
6. "Effect": "Allow",
7. "Action": "s3:ListAllMyBuckets",
8. "Resource": "\*"
9. },
10. {
11. "Effect": "Allow",
12. "Action": [
13. "s3:ListBucket",
14. "s3:GetBucketLocation"
15. ],
16. "Resource": "*arn:aws:s3:::productionapp*"
17. },
18. {
19. "Effect": "Allow",
20. "Action": [
21. "s3:GetObject",
22. "s3:PutObject",
23. "s3:DeleteObject"
24. ],
25. "Resource": "*arn:aws:s3:::productionapp/\**"
26. }
27. ]

}

The ListBucket permission allows users to view objects in the productionapp bucket. The GetObject, PutObject, DeleteObject permissions allows users to view, update, and delete contents in the productionapp bucket.

1. When you are finished, choose **Review policy**. The [Policy Validator](https://docs.aws.amazon.com/IAM/latest/UserGuide/access_policies_policy-validator.html) reports any syntax errors.

**Note**

You can switch between the **Visual editor** and **JSON** tabs any time. However, if you make changes or choose **Review policy** in the **Visual editor** tab, IAM might restructure your policy to optimize it for the visual editor. For more information, see [Policy Restructuring](https://docs.aws.amazon.com/IAM/latest/UserGuide/troubleshoot_policies.html#troubleshoot_viseditor-restructure).

1. On the **Review** page, type **read-write-app-bucket** for the policy name. Review the policy**Summary** to see the permissions granted by your policy, and then choose **Create policy** to save your work.

The new policy appears in the list of managed policies.

1. In the navigation pane on the left, choose **Roles** and then choose **Create role**.
2. Choose the **Another AWS account** role type.
3. For **Account ID**, type the Development account ID.

This tutorial uses the example account ID **111111111111** for the Development account. You should use a valid account ID. If you use an invalid account ID, such as **111111111111**, IAM does not let you create the new role.

For now you do not need to require an external ID, or require users to have multi-factor authentication (MFA) in order to assume the role. So leave these options unselected. For more information, see [Using Multi-Factor Authentication (MFA) in AWS](https://docs.aws.amazon.com/IAM/latest/UserGuide/id_credentials_mfa.html)

1. Choose **Next: Permissions** to set the permissions that will be associated with the role.
2. Select the box next to the policy that you created previously.

**Tip**

For **Filter**, choose **Customer managed** to filter the list to include only the policies that you have created. This hides the AWS created policies and makes it much easier to find the one you're looking for.

Then choose **Next: Review**.

1. Type **UpdateApp** for the role name.
2. (Optional) For **Role description**, type a description for the new role.
3. After reviewing the role, choose **Create role**.

The UpdateApp role appears in the list of roles.

Now you must obtain the role's Amazon Resource Name (ARN), which is a unique identifier for the role. When you modify the Developers and Testers group's policy, you will specify the role's ARN to grant or deny permissions.

**To obtain the ARN for UpdateApp**

1. In the navigation pane of the IAM console, choose **Roles**.
2. In the list of roles, choose the UpdateApp role.
3. In the **Summary** section of the details pane, copy the **Role ARN** value.

The Production account has an account ID of 999999999999, so the role ARN is arn:aws:iam::999999999999:role/UpdateApp. Ensure that you supply the real AWS account ID for your 'production' account.

At this point, you have established trust between the Production and Development accounts by creating a role in the Production account that identifies the Development account as a trusted principal. You also defined what users who switch to the UpdateApp role can do.

Next, modify the permissions for the groups.

**Step 2 - Grant Access to the Role**

At this point, both Testers and Developers group members have permissions that allow them to freely test applications in the Development account. Here are the steps required to add permissions to allow switching to the role.

**To modify the Developers group to allow them to switch to the UpdateApp role**

1. Sign in as an administrator in the Development account, and open the IAM console.
2. Choose **Groups**, and then choose **Developers**.
3. Choose the **Permissions** tab, expand the **Inline Policies** section, and then choose **Create Group Policy**. If no inline policy exists yet, then the button does not appear. Instead, choose the link at the end of "To create one, click here."
4. Choose **Custom Policy** and then choose **Select** button.
5. Type a policy name like **allow-assume-S3-role-in-production**.
6. Add the following policy statement to allow the AssumeRole action on the UpdateApp role in the Production account. Be sure that you change *PRODUCTION-ACCOUNT-ID* in the Resource element to the actual AWS account ID of the Production account.
7. {
8. "Version": "2012-10-17",
9. "Statement": {
10. "Effect": "Allow",
11. "Action": "sts:AssumeRole",
12. "Resource": "arn:aws:iam::*PRODUCTION-ACCOUNT-ID*:role/UpdateApp"
13. }

}

The Allow effect explicitly allows the Developers group access to the UpdateApp role in the Production account. Any developer who tries to access the role will succeed.

1. Choose **Apply Policy** to add the policy to the Developer group.

In most environments, the following procedure is likely not needed. If, however, you use Power User permissions, then some groups might already be able to switch roles. The following procedures shows how to add a "Deny" permission to the Testers group to ensure that they cannot assume the role. If this procedure is not needed in your environment, then we recommend that you do not add it. "Deny" permissions make the overall permissions picture more complicated to manage and understand. Use "Deny" permissions only when there is not a better option.

**To modify the Testers group to deny permission to assume the UpdateApp role**

1. Choose **Groups**, and then choose **Testers**.
2. Choose the **Permissions** tab, expand the **Inline Policies** section, and then choose **Create Group Policy**.
3. Choose **Custom Policy** and then choose the **Select** button.
4. Type a policy name like **deny-assume-S3-role-in-production**.
5. Add the following policy statement to deny the AssumeRole action on the UpdateApp role. Be sure that you change *PRODUCTION-ACCOUNT-ID* in the Resource element to the actual AWS account ID of the Production account.
6. {
7. "Version": "2012-10-17",
8. "Statement": {
9. "Effect": "Deny",
10. "Action": "sts:AssumeRole",
11. "Resource": "arn:aws:iam::*PRODUCTION-ACCOUNT-ID*:role/UpdateApp"
12. }

}

The Deny effect explicitly denies the Testers group access to the UpdateApp role in the Production account. Any tester who tries to access the role will get an access denied message.

1. Choose **Apply Policy** to add the policy to the Tester group.

The Developers group now has permissions to use the UpdateApp role in the Production account. The Testers group is prevented from using the UpdateApp role.

Next, you'll learn how David, a developer, can access the productionapp bucket in the Production account by using the AWS Management Console, the AWS CLI commands, and the AssumeRole API call.

**Step 3 - Test Access by Switching Roles**

After completing the first two steps of this tutorial, you have a role that grants access to a resource in the Production account. You also have one group in the Development account whose users are allowed to use that role. The role is now ready to use. This step discusses how to test switching to that role from the AWS Management Console, the AWS CLI, and the AWS API.

**Important**

You can switch to a role only when you are signed in as an IAM user or a federated user. Additionally, if you launch an Amazon EC2 instance to run an application, the application can assume a role through its instance profile. You cannot switch to a role when you are signed in as the AWS account root user.

**Switch Roles (Console)**

If David needs to work with in the Production environment in the AWS Management Console, he can do so by using **Switch Role**. He specifies the account ID or alias and the role name, and his permissions immediately switch to those permitted by the role. He can then use the console to work with the productionapp bucket, but cannot work with any other resources in Production. While David is using the role, he also cannot make use of his power-user privileges in the Development account. That's because only one set of permissions can be in effect at a time.

**Important**

Switching roles using the AWS Management Console works only with accounts that do not require an ExternalId. If you grant access to your account to a third party and require an ExternalId in a Condition element in your permission policy, the third party can access your account only by using the AWS API or a command line tool. The third party cannot use the console because it cannot supply a value for ExternalId. For more information about this scenario, see [How to Use an External ID When Granting Access to Your AWS Resources to a Third Party](https://docs.aws.amazon.com/IAM/latest/UserGuide/id_roles_create_for-user_externalid.html), and [How to Enable Cross-Account Access to the AWS Management Console](http://aws.amazon.com/blogs/security/how-to-enable-cross-account-access-to-the-aws-management-console) in the *AWS Security Blog*.

There are two ways that David can use to enter the **Switch Role** page:

* David receives a link from his administrator that points to a pre-defined Switch Role configuration. The link is provided to the administrator on the final page of the **Create role** wizard or on the **Role Summary** page for a cross-account role. Choosing this link takes David to the **Switch Role** page with the **Account ID** and **Role name** fields already filled in. All David needs to do is choose **Switch Role**and he's done.
* The administrator does not send the link in email, but instead sends the **Account ID** number and **Role Name** values. David must manually type them to switch roles. This is illustrated in the following procedure.

**To assume a role**

1. David signs into the AWS console using his normal user that is in the Development group.
2. He chooses the link that his administrator sent to him in email. This takes him to the **Switch Role** page with the account ID or alias and the role name information already filled in.

—or—

He chooses his name (the Identity menu) on the navigation bar, and then chooses **Switch Role**.

If this is the first time David tries to access the Switch Role page this way, he will first land on a first-run **Switch Role** page. This page provides additional information on how switching roles can enable users to manage resources across AWS accounts. David must choose the **Switch Role** button on this page to complete the rest of this procedure.

1. Next, in order to access the role, David must manually type the Production account ID number (999999999999) and the role name (UpdateApp).

Also, to help him stay aware of which role (and associated permissions) are currently active, he types PRODUCTION in the **Display Name** text box, selects the red color option, and then chooses **Switch Role**.

1. David can now use the Amazon S3 console to work with the Amazon S3 bucket, or any other resource to which the UpdateApp role has permissions.
2. When he is done with the work he needs to do, David can return to his original permissions. To do that, he chooses the **PRODUCTION**role display name on the navigation bar and then chooses **Back to David @ 111111111111**.
3. The next time David wants to switch roles and chooses the Identity menu in the navigation bar, he sees the PRODUCTION entry still there from last time. He can simply choose that entry to switch roles immediately without having to reenter the account ID and role name.

**Switch Roles (AWS CLI)**

If David needs to work in the Production environment at the command line, he can do so by using the [AWS CLI](http://aws.amazon.com/cli/). He runs the aws sts assume-role command and passes the role ARN to get temporary security credentials for that role. He then configures those credentials in environment variables so subsequent AWS CLI commands work using the role's permissions. While David is using the role, he cannot use his power-user privileges in the Development account because only one set of permissions can be in effect at a time.

Note that all access keys and tokens are examples only and cannot be used as shown. Replace with the appropriate values from your live environment.

**To assume a role**

1. David opens a command prompt window, and confirms that the AWS CLI client is working by running the command:

aws help

**Note**

David's default environment uses the David user credentials from his default profile that he created with the aws configure command. For more information, see [Configuring the AWS Command Line Interface](http://docs.aws.amazon.com/cli/latest/userguide/cli-chap-getting-started.html#cli-quick-configuration) in the *AWS Command Line Interface User Guide*.

1. He begins the switch role process by running the following command to switch to the UpdateApp role in the Production account. He got the role ARN from the administrator that created the role. The command requires that you provide a session name as well, you can choose any text you like for that.

aws sts assume-role --role-arn "arn:aws:iam::999999999999:role/UpdateApp" --role-session-name "David-ProdUpdate"

David then sees the following in the output:

{

"Credentials": {

"SecretAccessKey": "wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY",

"SessionToken": "AQoDYXdzEGcaEXAMPLE2gsYULo+Im5ZEXAMPLEeYjs1M2FUIgIJx9tQqNMBEXAMPLE

CvSRyh0FW7jEXAMPLEW+vE/7s1HRpXviG7b+qYf4nD00EXAMPLEmj4wxS04L/uZEXAMPLECihzFB5lTYLto9dyBgSDy

EXAMPLE9/g7QRUhZp4bqbEXAMPLENwGPyOj59pFA4lNKCIkVgkREXAMPLEjlzxQ7y52gekeVEXAMPLEDiB9ST3Uuysg

sKdEXAMPLE1TVastU1A0SKFEXAMPLEiywCC/Cs8EXAMPLEpZgOs+6hz4AP4KEXAMPLERbASP+4eZScEXAMPLEsnf87e

NhyDHq6ikBQ==",

"Expiration": "2014-12-11T23:08:07Z",

"AccessKeyId": "AKIAIOSFODNN7EXAMPLE"

}

}

1. David sees the three pieces that he needs in the Credentials section of the output.
   * AccessKeyId
   * SecretAccessKey
   * SessionToken

David needs to configure the AWS CLI environment to use these parameters in subsequent calls. For information about the various ways to configure your credentials, see [Configuring the AWS Command Line Interface](http://docs.aws.amazon.com/cli/latest/userguide/cli-chap-getting-started.html#config-settings-and-precedence). You cannot use the aws configure command because it does not support capturing the session token. However, you can manually type the information into a configuration file. Because these are temporary credentials with a relatively short expiration time, it is easiest to add them to the environment of your current command line session.

1. To add the three values to the environment, David cuts and pastes the output of the previous step into the following commands. Note that you might want to cut and paste into a simple text editor to address line wrap issues in the output of the session token. It must be added as a single long string, even though it is shown line wrapped here for clarity.

**Note**

The following example shows commands given in the Windows environment, where "set" is the command to create an environment variable. On a Linux or Mac computer, you would use the command "export" instead. All other parts of the example are valid in all three environments.

set AWS\_ACCESS\_KEY\_ID=AKIAIOSFODNN7EXAMPLE

set AWS\_SECRET\_ACCESS\_KEY=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY

set AWS\_SESSION\_TOKEN=AQoDYXdzEGcaEXAMPLE2gsYULo+Im5ZEXAMPLEeYjs1M2FUIgIJx9tQqNMBEXAMPLECvS

Ryh0FW7jEXAMPLEW+vE/7s1HRpXviG7b+qYf4nD00EXAMPLEmj4wxS04L/uZEXAMPLECihzFB5lTYLto9dyBgSDyEXA

MPLEKEY9/g7QRUhZp4bqbEXAMPLENwGPyOj59pFA4lNKCIkVgkREXAMPLEjlzxQ7y52gekeVEXAMPLEDiB9ST3UusKd

EXAMPLE1TVastU1A0SKFEXAMPLEiywCC/Cs8EXAMPLEpZgOs+6hz4AP4KEXAMPLERbASP+4eZScEXAMPLENhykxiHen

DHq6ikBQ==

At this point, any following commands run under the permissions of the role identified by those credentials. In David's case, the UpdateApp role.

1. Run the command to access the resources in the Production account. In this example, David simply lists the contents of his S3 bucket with the following command.

aws s3 ls s3://productionapp

Because Amazon S3 bucket names are universally unique, there is no need to specify the account ID that owns the bucket. To access resources for other AWS services, refer to the AWS CLI documentation for that service for the commands and syntax that are required to reference its resources.

**Using AssumeRole (AWS API)**

When David needs to make an update to the Production account from code, he makes an AssumeRole call to assume the UpdateApp role. The call returns temporary credentials that he can use to access the productionapp bucket in the Production account. With those credentials, David can make API calls to update the productionapp bucket. However, he cannot make API calls to access any other resources in the Production account, even though he has power-user permissions in the Development account.

**To assume a role**

1. David calls AssumeRole as part of an application. He must specify the UpdateApp ARN:arn:aws:iam::999999999999:role/UpdateApp.

The response from the AssumeRole call includes the temporary credentials with an AccessKeyId, a SecretAccessKey, and an Expiration time that indicates when the credentials expire and you must request new ones.

1. With the temporary credentials, David makes an s3:PutObject call to update the productionappbucket. He would pass the credentials to the API call as the AuthParams parameter. Because the temporary role credentials have only read and write access to the productionapp bucket, any other actions in the Production account are denied.

For a code example (using Python), see [Switching to an IAM Role (AWS API)](https://docs.aws.amazon.com/IAM/latest/UserGuide/id_roles_use_switch-role-api.html).

**Related Resources**

* For more information about IAM users and groups, see [Identities (Users, Groups, and Roles)](https://docs.aws.amazon.com/IAM/latest/UserGuide/id.html) .
* For more information about Amazon S3 buckets, see [Create a Bucket](http://docs.aws.amazon.com/AmazonS3/latest/gsg/CreatingABucket.html) in the *Amazon Simple Storage Service Getting Started Guide*.

**Summary**

You have completed the cross-account API access tutorial. You created a role to establish trust with another account and defined what actions trusted entities can take. Then, you modified a group policy to control which IAM users can access the role. As a result, developers from the Development account can make updates to the productionapp bucket in the Production account by using temporary credentials.