Working with Multiple AWS Accounts

So far, throughout this book, you've likely been using a single AWS account for all of your infrastructure. For production code, it's more common to use multiple AWS accounts: e.g., you put your staging environment in a stage account, your production environment in a prod account, and so on. This concept applies to other clouds too, such as Azure and Google Cloud. Note that I'll be using the term *account* in this book, even though some clouds use slightly different terminology for the same concept (e.g., Google Cloud calls them *projects* instead of accounts).

The main reasons for using multiple accounts are as follows:

Isolation (aka compartmentalization)

> You use separate accounts to isolate different environments from each other and to limit the "blast radius" when things go wrong. For example, putting your staging and production environments in separate accounts ensures that if an attacker manages to break into staging, they still have no access whatsoever to production. Likewise, this isolation ensures that a developer making changes in staging is less likely to accidentally break something in production.

Authentication and authorization

> If everything is in one account, it's tricky to grant access to some things (e.g., the staging environment) but not accidentally grant access to other things (e.g., the production environment). Using multiple accounts makes it easier to have fine-grained control, as any permissions you grant in one account have no effect on any other account.
>
> The authentication requirements of multiple accounts also help reduce the chance of mistakes. With everything in a single account, it's too easy to make the mistake where you think you're making a change in, say, your staging environment, but you're actually making the change in production (which can be a disaster if the change you're making is, for example, to drop all database tables). With multiple accounts, this is less likely, as authenticating to each account requires a separate set of steps.

Note that having multiple accounts does *not* imply that developers have multiple separate user profiles (e.g., a separate IAM user in each AWS account). In fact, that would be an antipattern, as that would require managing multiple sets of credentials, permissions, etc. Instead, you can configure just about all the major clouds so that each developer has exactly one user profile, which they can use to authenticate to any account they have access to. The cross-account authentication mechanism varies depending on the cloud you're using: e.g., in AWS, you can authenticate across AWS accounts by assuming IAM roles, as you'll see shortly.

Auditing and reporting

A properly configured account structure will allow you to maintain an audit trail of all the changes happening in all your environments, check if you're adhering to compliance requirements, and detect anomalies. Moreover, you'll be able to have consolidated billing, with all the charges for all of your accounts in one place, including cost breakdowns by account, service, tag, etc. This is especially useful in large organizations, as it allows finance to track and budget spending by team simply by looking at which account the charges are coming from.

Let's go through a multi-account example with AWS. First, you'll want to create a new AWS account to use for testing. Since you already have one AWS account, to create new *child accounts*, you can use AWS Organizations, which ensures that the billing from all the child accounts rolls up into the parent account (sometimes called the *root account*) and gives you a dashboard you can use to manage all the child accounts.

Head over to the AWS Organizations Console, and click the "Add an AWS account" button, as shown in Figure 7-3.

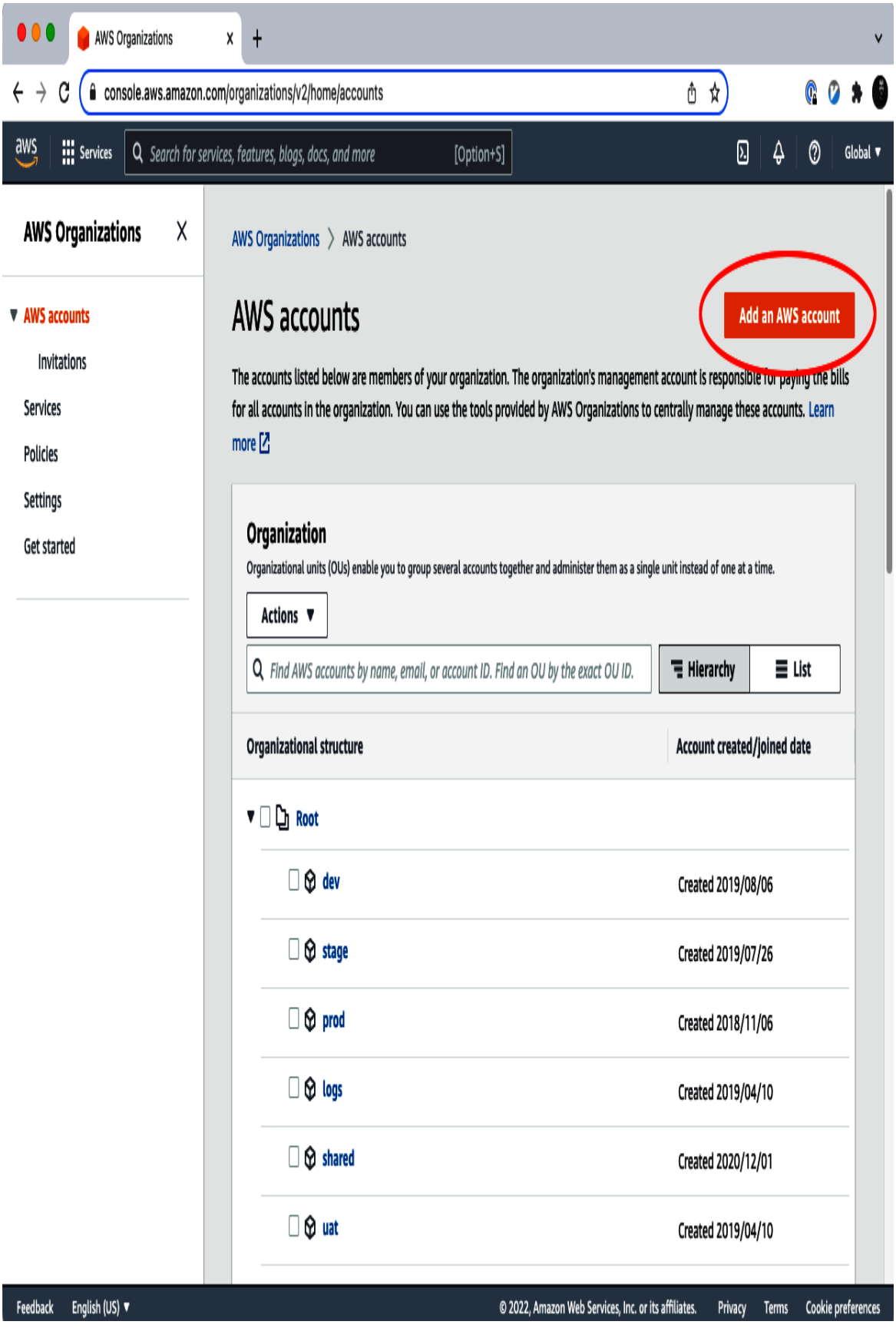
AWS Organizations
console.aws.amazon.com/organizations/v2/home/accounts
Services
Search for services, features, blogs, docs, and more
[Option+S]
Global
AWS Organizations
AWS accounts
Invitations
Services
Policies
Settings
Get started
AWS Organizations > AWS accounts
AWS accounts
Add an AWS account
The accounts listed below are members of your organization. The organization's management account is responsible for paying the bills for all accounts in the organization. You can use the tools provided by AWS Organizations to centrally manage these accounts. Learn more
Organization
Organizational units (OUs) enable you to group several accounts together and administer them as a single unit instead of one at a time.
Actions
Find AWS accounts by name, email, or account ID. Find an OU by the exact OU ID.
Hierarchy
List
Organizational structure
Account created/joined date
Root
dev
Created 2019/08/06
stage
Created 2019/07/26
prod
Created 2018/11/06
logs
Created 2019/04/10
shared
Created 2020/12/01
uat
Created 2019/04/10
Feedback
English (US)
© 2022, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences

Figure 7-3. Use AWS Organizations to create a new AWS account.

On the next page, fill in the following info, as shown in Figure 7-4:

AWS account name

The name to use for the account. For example, if this account was going to be used for your staging environment, you might name it "staging."

Email address of the account's owner

The email address to use for the root user of the AWS account. Note that every AWS account must use a different email address for the root user, so you can't reuse the email address you used to create your first (root) AWS account (see "How to Get Multiple Aliases from One Email Address" for a workaround). So what about the root user's password? By default, AWS does not configure a password for the root user of a new child account (you'll see shortly an alternative way to authenticate to the child account). If you ever do want to log in as this root user, after you create the child account, you'll need to go through the password reset flow with the email address you're specifying here.

IAM role name

When AWS Organizations creates a child AWS account, it automatically creates an IAM role within that child AWS account that has admin permissions and can be assumed from the parent account. This is convenient, as it allows you to authenticate to the child AWS account without having to create any IAM users or IAM roles yourself. I recommend leaving this IAM role name at the default value of `OrganizationAccountAccessRole`.

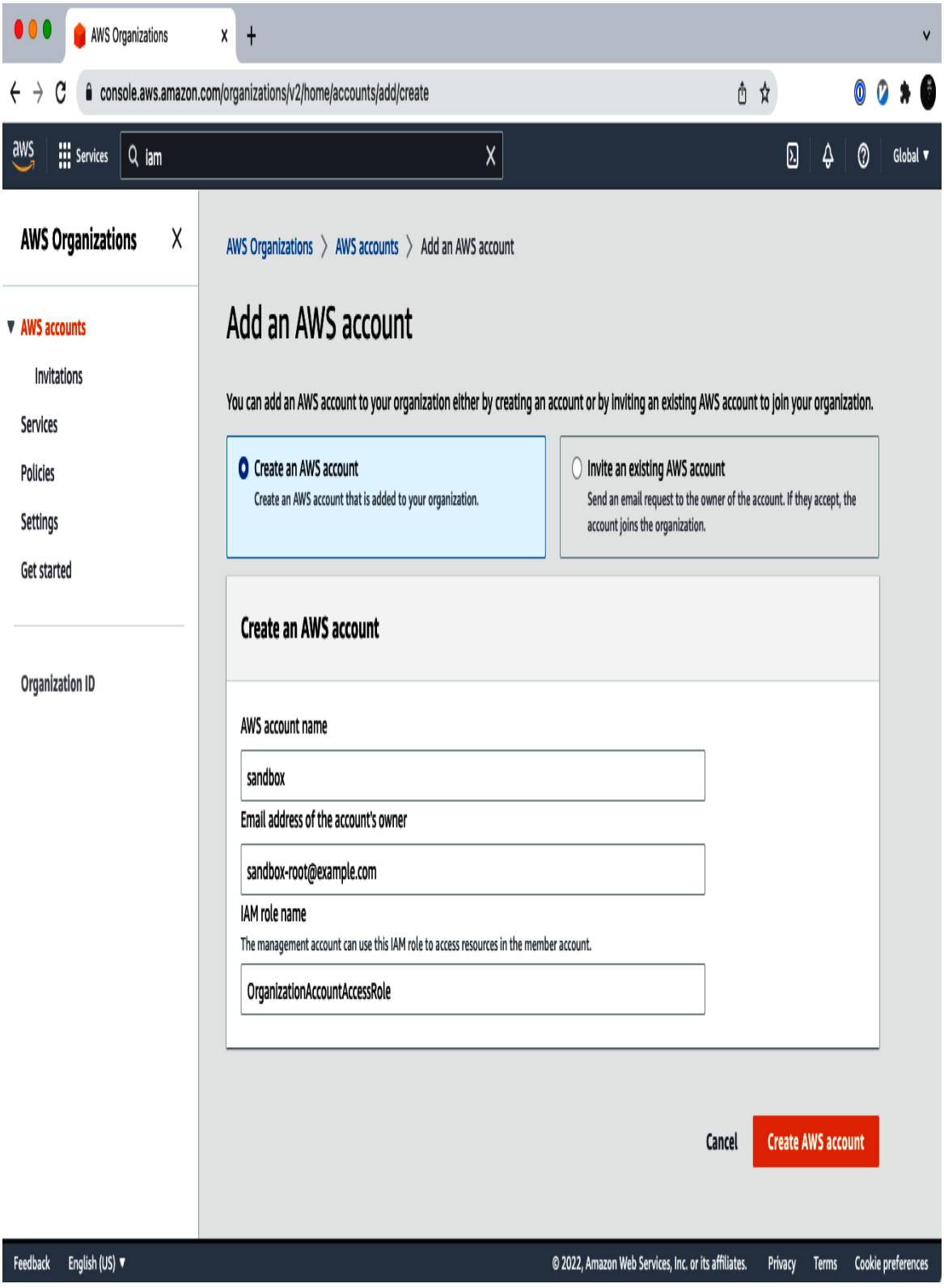

Figure 7-4. Fill in the details for the new AWS account.

HOW TO GET MULTIPLE ALIASES FROM ONE EMAIL ADDRESS

If you use Gmail, you can get multiple email aliases out of a single address by taking advantage of the fact that Gmail ignores everything after a plus sign in an email address. For example, if your Gmail address is *example@gmail.com*, you can send email to *example+foo@gmail.com* and *example+any-text-you-want@gmail.com*, and all of those emails will go to *example@gmail.com*. This also works if your company uses Gmail via Google Workspace, even with a custom domain: e.g., *example+dev@company.com* and *example+stage@company.com* will all go to *example@company.com*.

This is useful if you're creating a dozen child AWS accounts, as instead of having to create a dozen totally separate email addresses, you could use *example+dev@company.com* for your dev account, *example+stage@company.com* for your stage account, and so on; AWS will see each of those email addresses as a different, unique address, but under the hood, all the emails will go to the same account.

Click the Create AWS Account button, wait a few minutes for AWS to create the account, and then jot down the 12-digit ID of the AWS account that gets created. For the rest of this chapter, let's assume the following:

- Parent AWS account ID: `111111111111`
- Child AWS account ID: `222222222222`

You can authenticate to your new child account from the AWS Console by clicking your username and selecting "Switch role", as shown in Figure 7-5.

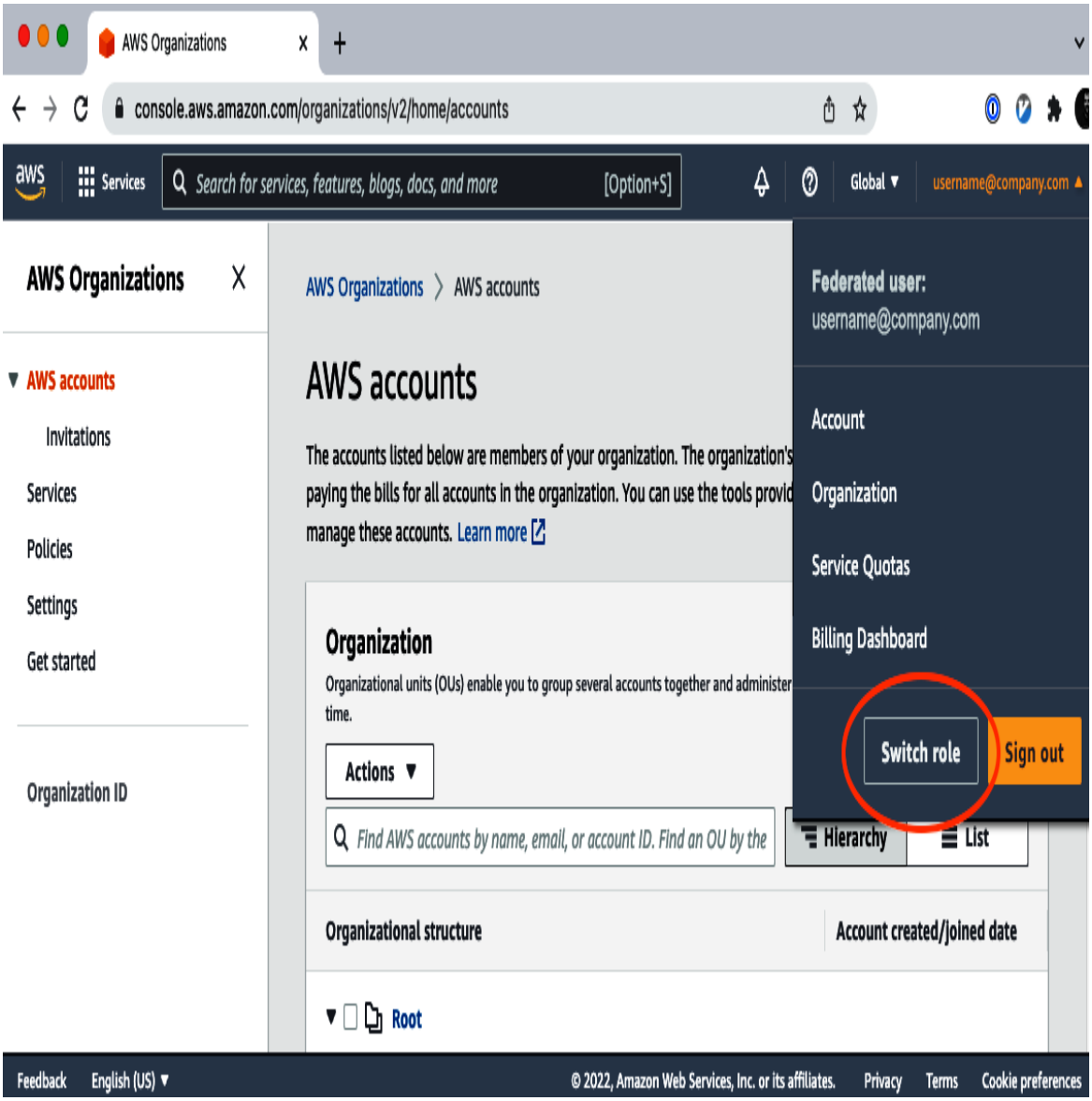

Figure 7-5. Select the "Switch role" button.

Next, enter the details for the IAM role you want to assume, as shown in Figure 7-6:

Account

The 12-digit ID of the AWS account to switch to. You'll want to enter the ID of your new child account.

Role

The name of the IAM role to assume in that AWS account. Enter the name you used for the IAM role when creating the new child account, which is `OrganizationAccountAccessRole` by default.

Display name

AWS will create a shortcut in the nav to allow you to switch to this account in the future with a single click. This is the name to show in this shortcut. It only affects your IAM user in this browser.

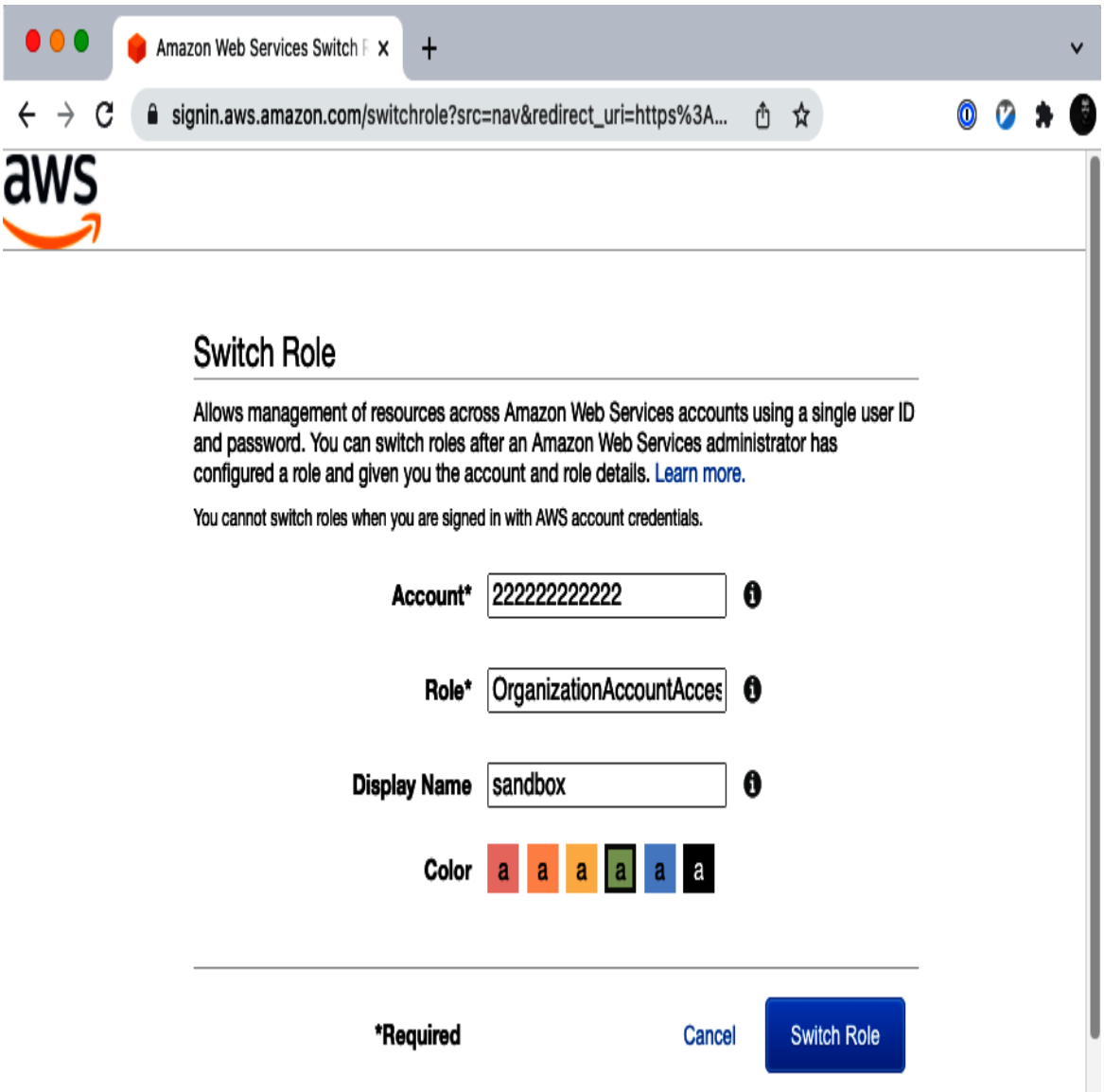

Figure 7-6. Enter the details for the role to switch to.

Click Switch Role and, voilà, AWS should log you into the web console of the new AWS account!

Let's now write an example Terraform module in *examples/multi-account-root* that can authenticate to multiple AWS accounts. Just as with the multiregion AWS example, you will need to add two `provider` blocks in *main.tf*, each with a different alias. First, the `provider` block for the parent AWS account:

```
provider "aws" {
  region = "us-east-2"
  alias  = "parent"
}
```

Next, the `provider` block for the child AWS account:

```
provider "aws" {
  region = "us-east-2"
  alias  = "child"
}
```

To be able to authenticate to the child AWS account, you'll assume an IAM role. In the web console, you did this by clicking the Switch Role button; in your Terraform code, you do this by adding an `assume_role` block to the child `provider` block:

```
provider "aws" {
  region = "us-east-2"
  alias  = "child"

  assume_role {
    role_arn = "arn:aws:iam::<ACCOUNT_ID>:role/<ROLE_NAME>"
  }
}
```

In the `role_arn` parameter, you'll need to replace `ACCOUNT_ID` with your child account ID and `ROLE_NAME` with the name of the IAM role in that account, just as you did when switching roles in the web console.

Here's what it looks like with the account ID `222222222222` and role name `OrganizationAccountAccessRole` plugged in:

```
provider "aws" {
  region = "us-east-2"
  alias  = "child"

  assume_role {
    role_arn =
"arn:aws:iam::222222222222:role/OrganizationAccountAccessRole"
  }
}
```

Now, to check this is actually working, add two `aws_caller_identity` data sources, and configure each one to use a different provider:

```
data "aws_caller_identity" "parent" {
  provider = aws.parent
}

data "aws_caller_identity" "child" {
  provider = aws.child
}
```

Finally, add output variables in *outputs.tf* to print out the account IDs:

```
output "parent_account_id" {
  value       = data.aws_caller_identity.parent.account_id
  description = "The ID of the parent AWS account"
}

output "child_account_id" {
  value       = data.aws_caller_identity.child.account_id
  description = "The ID of the child AWS account"
}
```

Run `apply`, and you should see the different IDs for each account:

```
$ terraform apply

(...)
```

```
Apply complete! Resources: 0 added, 0 changed, 0 destroyed.

Outputs:

parent_account_id = "111111111111"
child_account_id = "222222222222"
```

And there you have it: by using provider aliases and `assume_role` blocks, you now know how to write Terraform code that can operate across multiple AWS accounts.

As with the multiregion section, a few warnings:

Warning 1: Cross-account IAM roles are double opt-in

In order for an IAM role to allow access from one AWS account to another—e.g., to allow an IAM role in account `222222222222` to be assumed from account `111111111111`—you need to grant permissions in *both* AWS accounts:

- First, in the AWS account where the IAM role lives (e.g., the child account `222222222222`), you must configure its assume role policy to trust the other AWS account (e.g., the parent account `111111111111`). This happened magically for you with the `OrganizationAccountAccessRole` IAM role because AWS Organizations automatically configures the assume role policy of this IAM role to trust the parent account. However, for any custom IAM roles you create, you need to remember to explicitly grant the `sts:AssumeRole` permission yourself.
- Second, in the AWS account from which you assume the role (e.g., the parent account `111111111111`), you must *also* grant your user permissions to assume that IAM role. Again, this happened for you magically because, in Chapter 2, you gave your IAM user `AdministratorAccess`, which gives you permissions to do just about everything in the parent AWS account, including assuming IAM roles. In most real-world use cases, your user won't be