Chapter 1

Introduction to Amazon Web Services

THE AWS CERTIFIED DEVELOPER – ASSOCIATE EXAM TOPICS COVERED IN THIS CHAPTER MAY INCLUDE, BUT ARE NOT LIMITED TO, THE FOLLOWING:

- ✓ **Domain 1: Development with AWS Services**
  - Task Statement 1: Develop code for applications hosted on AWS
- ✓ **Domain 2: Security**
  - Task Statement 1: Implement authentication and/or authorization for applications and AWS services

Introduction to AWS

Amazon Web Services (AWS) is the world's leading provider of cloud infrastructure services, offering compute, storage, networking, and databases, plus a broad set of platform capabilities such as deployment pipelines, analytics, and machine learning, all available on demand with pay-as-you-go pricing.

Think of AWS as a programmable data center. Once upon a time, if you wanted to launch a new web application or provision a database, you started by choosing a server configuration, purchasing it, waiting for it to ship to you, and installing and configuring it on-premises—before you could even begin to log in, configure software, and provide services to your users.

With *infrastructure-as-a-service* (IaaS) provided by AWS, all that can be done in a matter of seconds, with a few clicks or lines of code, freeing you up to focus on delivering value for your users. You can provision virtual servers on demand in minutes and pay only for the compute capacity you use. This ability to tailor capacity and compute costs to demand makes AWS a truly elastic service that can meet your needs however they may vary. Not only that, but AWS operates many data centers worldwide, configured to offer redundancy and high availability, improving the uptime and user experience of your data and services.

In this chapter, you are introduced to AWS and shown how to make your first service calls to build and manage resources. We'll then dive into the infrastructure behind AWS, and you'll learn how to manage the credentials and permissions that you need to securely access these powerful cloud capabilities.

Getting Started with an AWS Account

The AWS Certified Developer – Associate is designed for developers who have hands-on experience working with AWS services. To help you prepare, this book has recommended exercises at the end of each chapter.

To work with AWS, you'll need an account. While you must provide contact and payment information to sign up for an account, you can test many of these services through the *AWS Free Tier*. The AWS Free Tier limits allow you to become familiar with the APIs for the included services without incurring charges.

The AWS Free Tier automatically provides usage alerts to help you stay in control of usage and identify possible charges. You can define additional alerts with AWS Budgets. To best take advantage of the AWS Free Tier and reduce costs, take some time to review the AWS Free Tier limits and make sure to shut down or delete resources when you are done using them.

To create an account, sign up at `aws.amazon.com/free`.

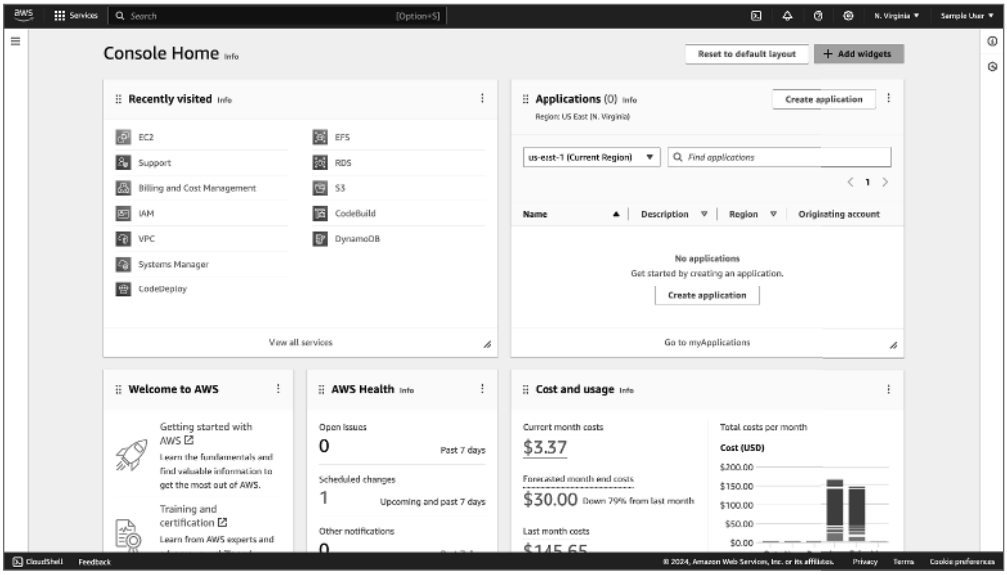
AWS Management Console

After you have created an account, you will be prompted to sign in to the *AWS Management Console*. As part of the sign-up process, you define an email address and password to sign in to the AWS Management Console as the root user for the account.

The AWS Management Console is a web interface where you can create, configure, and monitor AWS resources in your account and explore the functionality of AWS's many services. The console also summarizes your overall monthly spend and provides links to learning materials to help you get started.

Sign in to the AWS Management Console, as shown in Figure 1.1, at `http://signin.aws.amazon.com/console`.

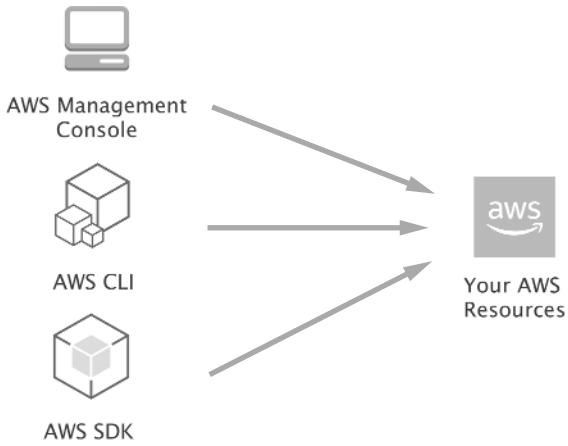
FIGURE 1.1 AWS Management Console



Because all the functionality of AWS is exposed through APIs, AWS provides more than just the web interface for managing resources. For example, the AWS Management Console is also available as a mobile app for iOS and for Android.

After you become familiar with a service, you can manage AWS resources programmatically through either the AWS Command-Line Interface (AWS CLI) or the AWS Software Development Kits (AWS SDKs), as shown in Figure 1.2.

FIGURE 1.2 Options for managing AWS resources



AWS Software Development Kits

AWS software development kits (SDKs) are available in many popular programming languages such as Java, .NET, JavaScript, PHP, Python, Ruby, Go, and C++. AWS also provides specialty SDKs such as its Mobile SDK and Internet of Things (IoT) Device SDK.

Although the instructions for installing and using an AWS SDK vary depending on the operating system and programming language, they share many similarities. In this chapter, the examples are provided in Python, using its SDK, which is named "boto." If Python is already installed on your machine, install boto3 using pip, the Python package manager:

```
pip install boto3 --upgrade -user
```

For documentation on the Python SDK, see `http://boto3.readthedocs.io`.

For more information on SDKs for other programming languages or platforms, see `http://aws.amazon.com/tools/#sdk`.

AWS CLI Tools

In addition to the AWS Management Console and SDKs, AWS provides tools to manage AWS resources from the command line. One such tool is the *AWS Command-Line Interface (CLI)*, which is available on Windows, Linux/UNIX, and macOS.

The AWS CLI allows you to perform actions similar to the SDKs from a terminal. Because the AWS CLI is interactive, it is a good environment for experimenting with AWS features. Also, the AWS CLI and the SDK on the same server can share configuration settings.

If you prefer to manage your resources using PowerShell, use the AWS Tools for PowerShell instead of the AWS CLI. Other specialty command-line tools are also available, such as the Elastic Beanstalk command-line interface and AWS SAM Local. For more information about these tools and installation, see `http://aws.amazon.com/tools/#cli`.

Once it's installed, you can verify basic functionality of the CLI by typing **`aws --version`**.

Creating an Administrative IAM User

Before you can use the CLI to work with AWS resources, you configure it with the credentials of the user you wish to use to authenticate and whose permissions will be used by the CLI. While you will have initially logged into AWS as the root user of your account, it is not recommended to act as the root user when performing tasks via the CLI (or SDK). We must create a subuser, or in AWS terms, an *Identity and Access Management (IAM)* user.

Because this book covers so many IAM services, the user we create will be given wide permissions; however, this is still more secure than the root user, which is capable of changing billing information or even closing your account entirely! It's important to protect the root user login information as best you can.

To create an administrative user for use with the CLI or SDK, do the following:

1. Log in to the management console as the root user.
2. Use the Services search bar to navigate to IAM.
3. Click Users in the left-hand menu.
4. Click Create User.
5. For User Details, type **AdminUser**.
6. Select the option Provide User Access To The AWS Management Console. Keep the defaults for Autogenerated Password and "Users must create a new password at next sign-in" options.
7. Click Next.
8. Choose Attach Policies Directly, then filter for and select AdministratorAccess.
9. Click Next.
10. Click Create User.
11. Note the Sign-in Details on this page, including the password, which you can reveal by clicking Show.

From now on, you can use the Console Sign-in URL shown on this page, along with the User Name and Console Password, to log in to the management console. Try it now, logging out as your root user and logging back in with the AdminUser you just created.

Next, we will obtain the access keys for the user, which you can use to configure the CLI.

Obtaining Access Keys for Programmatic Access

Now that you are logged in as AdminUser, let's generate access keys for use with the CLI. Once again, head to the IAM console:

1. Log in to the management console as the root user.
2. Use the Services search bar to navigate to IAM.
3. Click Users in the left-hand menu.
4. Click AdminUser.
5. Select the Security Credentials tab.
6. Scroll to the Access Keys section and click Create Access Key.
7. Choose the use case Command Line Interface, scroll past the recommended alternatives, and select the "I understand the above recommendations" check box.
8. Click Next.
9. Provide the description **Keys for CLI**, and click Next.

On the final page, you will be given a chance to copy two values: the Access Key, which you can always obtain, and the Secret Access Key, which you will not see again.

Make sure to treat these values like passwords and keep them safe.

Configuring the AWS CLI

Now that you have installed the CLI and obtained your access key pair, let's configure the CLI. On your command line, type **`aws configure`** to initiate a series of prompts for values. You'll be prompted for four values. You have the first two:

- AWS Access Key ID.
- AWS Secret Access Key.
- Default Region Name: You'll learn about these later in this chapter. For now, take the default, `us-east-1`.
- Default Output Format: Keep the default, `json`.

The CLI is now configured for this user. You can verify this by running a command that fetches the active account's user details:

```
aws iam get-user
```

For macOS and Linux systems, the AWS CLI stores these values at `~/.aws/credentials` and `~/.aws/config`. If you open the credentials file, you will see something like this:

```
[default]
aws_access_key_id = AKIA1234567890
aws_secret_access_key = <secret_string>
```

Similarly, the config file contains your chosen region and format.

CLI Named Profiles

It is possible to configure the AWS CLI to support multiple user profiles at once. First, create another IAM user with different privileges, then choose a profile name. The profile name does not need to correlate with the IAM username you plan to configure; in this example, we use the name "developer." Then run this command:

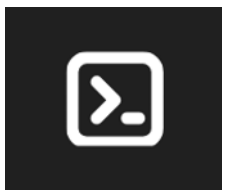
```
aws configure --profile developer
```

You will be prompted for the configuration values again. From then on, you may append `--profile developer` to any AWS CLI command to run it as this alternate user.

Using Cloud Shell to Avoid Managing Credentials

AWS provides Cloud Shell, which is another way to get easy CLI access for any user. Log in to the AWS Management Console and locate this button on the toolbar, as shown in Figure 1.3.

FIGURE 1.3 Cloud Shell button



Clicking this button will launch a terminal window in your web browser. This ephemeral environment has no long-term persistent storage, but it comes with the CLI preinstalled and configured for the currently logged-in user. Cloud Shell is a great way to quickly run CLI scripts from any machine without the need to set up a local environment or handle (or even generate) credentials, making it a convenient and secure option.

Using the CLI

While a deep dive into the CLI is outside the scope of this book, you should know that the CLI is capable of programmatically issuing any command that you can perform with your

configured user via the Management Console. CLI commands are invoked in the format `aws <service_name> <command>`, followed by any additional parameters or flags. For example, you may list details of all S3 buckets in your account with **`aws s3 ls`** or show details of all EC2 instances using **`aws ec2 describe-instances`**. To see all services, use **`aws help`** and to see all commands for a service, use **`aws <service_name> help`**.

Calling an AWS Service

The CLI is just one way to invoke AWS commands. In this section, you'll use boto, the Python SDK, to make service requests, and learn how to configure the SDK with your user credentials.

Locate the API reference documentation about the underlying web services and programming language–specific documentation for each SDK at `http://aws.amazon.com/documentation`.

SDK Example: Hello World

The following example makes a request to *Amazon Polly*. Polly provides text-to-speech services, and it's simple enough to make a great first SDK example.

This Python code example uses boto and Polly to generate an audio clip that says, "Hello World," using made-up credentials to communicate with the AWS service.

```
import boto3

#Explicit Client Configuration
polly = boto3.client('polly',
        region_name='us-west-2',
        aws_access_key_id='AKIAIO5FODNN7EXAMPLE',
        aws_secret_access_key='ABCDEF+c2L7yXeGvUyrPgYsDnWRRC1AYEXAMPLE'
        )

result = polly.synthesize_speech(Text='Hello World!',
                                 OutputFormat='mp3',
                                 VoiceId='Aditi')

# Save the Audio from the response
audio = result['AudioStream'].read()
with open("helloworld.mp3","wb") as file:
    file.write(audio)
```

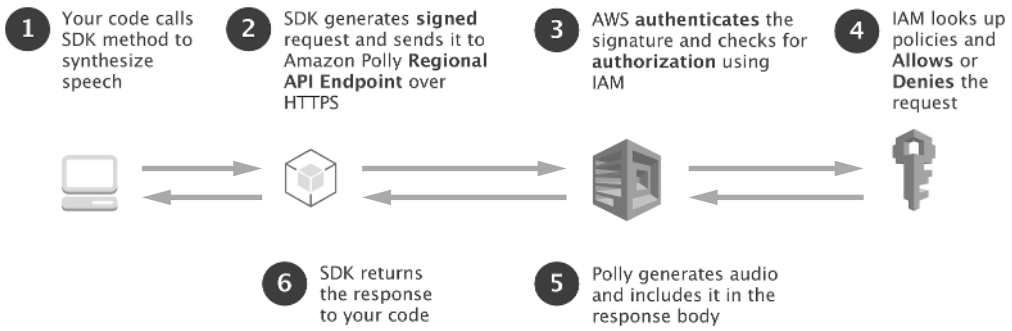
Behind the scenes, boto maps the SDK function call to an HTTPS request to an Amazon Polly API endpoint that is determined by the region name (`region_name`) parameter.

The SDK also adds authorization information to your request by signing the request using a key derived from the AWS secret access key.

When your request is received at the Amazon Polly API endpoint, AWS authenticates the signature and evaluates IAM policies to authorize the API action.

If authorization succeeds, Amazon Polly processes the request, generates an MP3 audio file, and then returns it to the SDK client as part of the response to the HTTPS request, as shown in Figure 1.4.

FIGURE 1.4 API request and authorization



While it's worth understanding that behind each boto call is an HTTP request to the AWS API, it is not essential to understand. When coding, most developers interact with AWS via an SDK or the CLI and rarely interface directly with the behind-the-scenes API.

Calls to AWS via the SDK or CLI include signatures that incorporate the current date, so make sure your system clock is accurate. AWS requests must be received within 15 minutes of the time stamp in the request to be valid.

SDK Configuration

In the previous example, the AWS region and credentials are provided explicitly in the code:

```
# Explicit Client Configuration
polly = boto3.client('polly',
       region_name='us-west-2',
       aws_access_key_id='AKIAIO5FODNN7EXAMPLE',
       aws_secret_access_key='ABCDEF+c2L7yXeGvUyrPgYsDnWRRC1AYEXAMPLE'
       )
```

This approach of hard-coding credentials is not recommended due to the risk of checking the credentials into a source-control repository, which would expose the keys to everyone who has access to the repository and could even result in public disclosure. Fortunately, you can configure these credentials in other ways that don't require hard-coding.

The SDK and CLI both automatically check several locations for these values when they are not explicitly provided in the code. These locations include environment variables, programming language–specific parameter stores, and local files.

Because we have already configured the AWS CLI, boto can use the default credentials found in `~/.aws/credentials` and configuration found in `~/.aws/config`.

As a result, we can replace this snippet of code:

```
# Explicit Client Configuration
polly = boto3.client('polly',
       region_name='us-west-2',
       aws_access_key_id='AKIAIO5FODNN7EXAMPLE',
       aws_secret_access_key='ABCDEF+c2L7yXeGvUyrPgYsDnWRRC1AYEXAMPLE'
       )
```

with this line of code, which uses the default CLI credentials:

```
# Implicit Client Configuration
polly = boto3.client('polly')
```

By separating your code from the credentials, you make it easier to collaborate with other developers while making sure that your credentials are not inadvertently disclosed to others.

For code running on an AWS compute environment, such as Amazon Elastic Compute Cloud (Amazon EC2) or AWS Lambda, instead of using local files, assign an IAM role to the environment. This enables the SDK to load the credentials automatically from the role and to refresh the credentials as they are automatically rotated. See the section "Identity and Access Management" in this chapter for more on IAM roles.

Working with Regions

Now we'll take a closer look at what it means to configure the AWS SDK with an *AWS region*. AWS operates facilities in multiple regions across the world, as shown in Figure 1.5. Each region is located in a separate geographic area and maintains its own, isolated copies of AWS services. For many services, you are required to select a specific region in which to provision your resources.

FIGURE 1.5 AWS regions, availability zones, and planned regions (as of May 2024)

In this section, you will explore how the structure of a region enables AWS to provide highly available, durable services, and how to choose an appropriate region for your application.

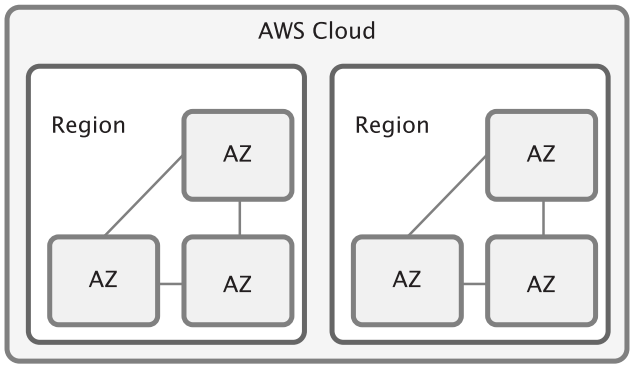
Regions Are Highly Available

Each *AWS region* contains multiple data centers, grouped together to form *availability zones (AZs)*. Availability zones are physically separated from each other and are designed to operate independently from each other in the case of a fault or natural disaster, as shown in Figure 1.6. Even though they are physically separated, AZs are connected via low-latency, high-throughput redundant networking, making multi-AZ architectures a fantastic choice for achieving high availability and scalability in your applications.

In fact, many AWS services automatically replicate data across multiple AZs within a region to provide high availability and durability of the data. An example of an AWS service that replicates data across AZs within a region is Amazon Simple Storage Service (Amazon S3). S3 enables you to upload files and store those files as objects within a bucket. By default, Amazon S3 automatically replicates objects across a minimum of three AZs within the region hosting the bucket, protecting your data against the loss of any one AZ.

In the AWS Management Console, you select from a drop-down list to specify your current region. Because so many AWS resources are region-specific, always pay attention to this selection! More than one new AWS developer has been shocked to find their existing resources gone, only to realize they have selected the wrong region.

FIGURE 1.6 Regions and availability zones



Identifying AWS Regions

When working with AWS Cloud services, the AWS Management Console refers to regions differently from the parameters used in the AWS CLI and SDK. Table 1.1 displays several region names and the corresponding parameters for the AWS CLI and SDK.

TABLE 1.1 Sample of region names and regions

Region name	Region
US East (N. Virginia)	`us-east-1`
US West (Oregon)	`us-west-2`
EU (Frankfurt)	`eu-central-1`
EU (London)	`eu-west-2`
EU (Paris)	`eu-west-3`
Asia Pacific (Tokyo)	`ap-northeast-1`
Asia Pacific (Mumbai)	`ap-south-1`
Asia Pacific (Singapore)	`ap-southeast-1`

There are other AWS services, such as *IAM*, that are not limited to a single region. When you interact with these services in the AWS Management Console, the region selector in the top-right corner of the console displays "Global."

Choosing a Region

One factor for choosing an AWS region is the availability of the services required by your application. Other aspects to consider when choosing a region are latency, price, and data residency. Table 1.2 contains selection criteria to include when choosing an AWS region.

TABLE 1.2 Selecting an AWS region

Selection criteria	Description
Service availability	Choose a region that has all or most of the services you intend to use. Each region exposes its own AWS Cloud service endpoints, and not all AWS Cloud services are available in all regions.
Proximity and latency	Choose a region closer to application users, on-premises servers, or your other workloads. This allows you to decrease the latency of API calls.
Data residency	Choose a region that allows you to stay compliant with regulatory or contractual requirements to store data within a specific geographic region.
Business continuity	Choose a pair of regions based on any specific requirements regarding data replication for disaster recovery. For example, you may select a second AWS region as a target for replicating data based on its distance from the primary AWS region.
Price	AWS service prices are set per region. Consider cost when service availability and latency are similar between candidate regions.

Identity and Access Management

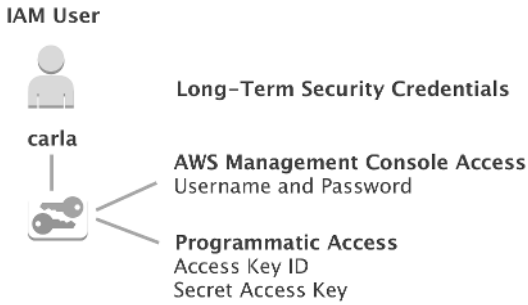
You have already seen how to create an IAM user to avoid making calls with the root user and its credentials and how to obtain and configure its security credentials. Now, let's take a closer look at the IAM service and how it gives you powerful controls for handling who can use your account and how.

To manage authentication and authorization, IAM provides users, groups, and roles. IAM authorizes each request to AWS by evaluating the policies associated with the identity and resources affected by the request. This section reviews users, groups, roles, and policies.

Users

IAM user accounts allow you to provision access to other users in your account, assigning credentials to allow AWS Management Console access, programmatic access, or both, as shown in Figure 1.7.

FIGURE 1.7 IAM user long-term credentials



AWS Management Console Access

To sign in to the web console, IAM users authenticate with an IAM username and password. When logging in, they provide either the account ID or alias so that IAM usernames only need to be unique within your account. If *multifactor authentication (MFA)* is enabled for an IAM user, they must also provide their MFA code when they attempt to sign in.

To simplify sign-in, use the special sign-in link in the IAM dashboard that prefills the account field in the console sign-in form.

AWS IAM User API Access Keys

As you saw when configuring the CLI, for *programmatic access to AWS*, we create an access key for the IAM user. An *AWS access key* is composed of two distinct parts:

- Access key ID
- Secret access key

Each user may have up to two active access keys at any time. These access keys are *long-term* credentials and remain valid until you explicitly revoke them.

Given the importance of the secret access key, you can view or download it only once. If you forget the secret access key, create a new access key and then revoke the earlier key.

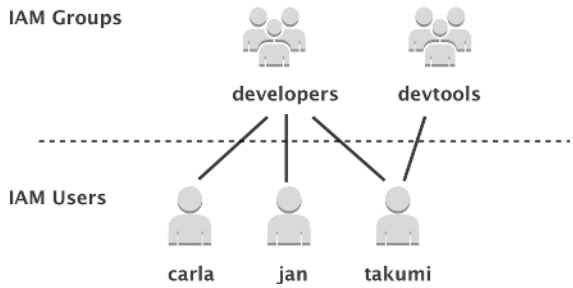
Groups

To help you manage the permissions of collections of IAM users, AWS provides *IAM groups*. You can associate users who need the same permissions with a group and then assign policies to the group instead of associating the permissions directly to each user.

For example, all developers working on a specific project could each have their own IAM user. Each of these users can be added to a group, named *developers*, to manage their permissions collectively.

An individual IAM user can be a member of many IAM groups, and each IAM group can have many IAM users associated with the group (see Figure 1.8). IAM users within an IAM group inherit permissions from the policies attached to their group, plus any permissions from policies that are associated directly with that IAM user.

FIGURE 1.8 IAM groups and IAM users



In the example shown in Figure 1.8, *carla* inherits permissions from the IAM user *carla* and from the group *developers*, and *takumi* inherits the union of all policies from *developers* and from *devtools*, in addition to any policies directly associated with *takumi*.

In the case that multiple permissions policies apply to the same API action, any policy that has the effect *deny* will take precedence over any policy that has the effect *allow*. This order of precedence is applied regardless of whether the policies are associated with the user, group, or resource.

Roles

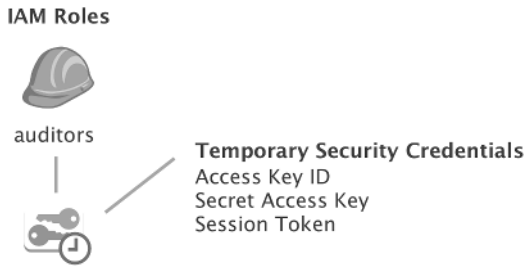
Roles (see Figure 1.9) are a unique feature of AWS IAM. They are similar to users in that you give them a name and assign privileges to them by attaching policies; however, they do not have passwords or credentials and cannot be used to log in.

Instead, the purpose of roles is to give users time-limited, temporary privileges. This can be more secure than assigning persistent privileges to a user, who can *assume* a role momentarily or for a single login session, perform actions allowed by that role, and then switch to a lower-privilege role.

In addition, roles are the mechanism for granting permissions to AWS resources like EC2 instances. The association of a role with an instance, called an *instance profile*, is explored further in Chapter 2, "Introduction to Compute and Networking."

To control access to an IAM role, define a *trust policy* that specifies which *principals* can assume a role. Potential principals include AWS Cloud services and also users who have authenticated using identity federation. Principals could also include users who authenticate with web identity federation, IAM users, IAM groups, or IAM roles from *other* accounts.

FIGURE 1.9 IAM roles



This example trust policy allows Amazon EC2 to request short-term credentials associated with an IAM role:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "ec2.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

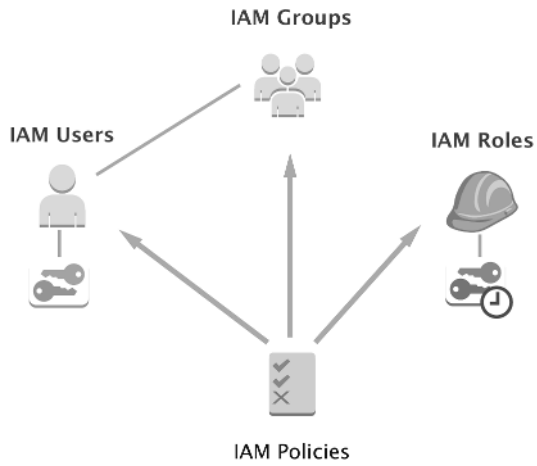
When a principal assumes a role, AWS provides new short-term security credentials that are valid for a time-limited session through the *AWS Security Token* (AWS STS) service. These credentials are composed of an access key ID, secret access key, and, additionally, a session token with a known expiration date.

This example displays the kind of credentials that are generated when the role is assumed:

```
{
       "AccessKeyId": "ASIAJHP2KG65VIKQU2XQ",
       "SecretAccessKey": "zkvPEbYxCLVVD0seWdRnesc8krNDPHEX1cFMyI5W",
       "SessionToken": "FQoDYXdzEMf//////////wEaDL1b0Wd7VTA3J25cNyL4ARzNSR
czH4U3f8gJwi1W8XiDLWJIE9EdX4l4KXTiST40gPoWc9Do9QkcN2xRHk6/qVT6W23d0
u6+5YFY9C2wnoEeTTmiQBT5SMjqku5MYlhrCDyFQAVbo6RKUeOZXXSG8REshuFGBt
aCNmv95lFF6srCT1b4FZtTtULE7WV3LMcDs6Z2XuN+6aGTawhY50RMnlKRL1w6yHq
++RysQWbBHkuNeK/VqjueDINFODPOje9ZnYePVjR5uLmL8ZARWYVBFrB2tpxG07/
dseUS9O2q1hMP8DJuEfsbaiK2ASsmXSRA8vOZnuu4AsBq6ERasBw5EcpICP/Ne8zdKO/93tYF",
       "Expiration": "2018-04-18T22:55:59Z"
}
```

While a user assumes a role, their permissions are limited to what the role can do; any permissions the user has directly attached or inherited from a group are not evaluated. Furthermore, you cannot nest IAM roles or add IAM roles to IAM groups, as shown in Figure 1.10.

FIGURE 1.10 IAM roles are distinct from IAM users and groups.



Choosing IAM Identities

The following scenarios will give you an idea of how to apply the right combination of users, roles, and groups for your situation.

Scenario: During Development

IAM users can be a convenient way to share access to an account with your team members or for application code that is running locally. To manage the permissions of collections of IAM users more simply, add those users to IAM groups.

Scenario: When Deploying Code to AWS

Use IAM roles. AWS compute services can be configured to distribute and rotate the role credentials automatically on your behalf, making it easier for you to manage credentials securely.

Scenario: When You Have an Existing External Identity Provider

When you have an external identity provider, such as Active Directory, use IAM roles. That way, team members can use the single sign-on they already use to access AWS without

needing to remember an extra password. Also, if a team member leaves, you can easily disable their corporate access in only one place—the external directory.

Use roles in cases in which you need to make AWS service calls from untrusted machines because role credentials automatically expire. For example, use IAM roles for client-side code that must upload data to Amazon S3 or interact with Amazon DynamoDB. Please see Table 1.3 for details on when to use IAM users or roles.

TABLE 1.3 IAM users and IAM roles usage

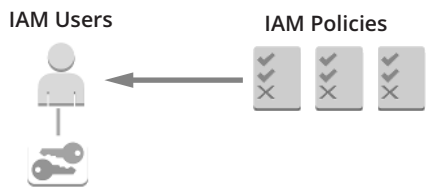
For code running on . . .	Suggestion
A local development laptop or on-premises server	IAM user
An AWS compute environment such as Amazon EC2	IAM role
An IAM user mobile device	IAM role
Enterprise environments with an external identity provider	IAM role

The exam tests your knowledge of the recommended practices for distributing AWS credentials to your code depending on where that code is running.

Managing Authorization with Policies

You manage permissions for each user, group, or role by assigning *IAM policies* that either *allow* or *deny* permissions to specific actions, as shown in Figure 1.11. Any action is *implicitly denied* unless there is a policy that explicitly allows it. If there is a policy that *explicitly denies* an action, that policy always takes precedence. In this way, AWS defaults to secure operation and errs on the side of protecting the resources in cases where there are conflicting policies.

FIGURE 1.11 IAM identities and IAM policies



One method of granting permissions is to use *AWS managed policies*. Managed policies are prebuilt, named policies maintained by AWS to support common tasks; they are automatically updated as new services and API operations are added. Earlier in the chapter, we used the managed policy `AdministratorAccess` to avoid writing our own policy. Other managed policies are clearly named for what privileges they confer, such as `AmazonS3ReadOnlyAccess`. Managed policies can always be examined so you may inspect the privileges they grant.

When choosing permissions policies, AWS recommends that you adopt the principle of *least privilege* and grant someone the minimum permissions they need to complete a task.

Take the example of an application that uses Amazon Polly. If the application uses only Amazon Polly to synthesize speech, use the `AmazonPollyReadOnlyAccess` policy, which grants permissions to Polly actions that do not store any data or modify data stored in AWS. The policy is represented as a JSON document and shown here:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "polly:DescribeVoices",
                "polly:GetLexicon",
                "polly:ListLexicons",
                "polly:SynthesizeSpeech"
            ],
            "Resource": [
                "*"
            ]
        }
    ]
```

If the application needs permission to upload (or delete) a custom lexicon, use the `AmazonPollyFullAccess` policy. The policy is shown here. Notice that the actions granted by the policy shown here are represented as `"polly:*"`, where the `*` provides access to all Polly API actions.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "polly:*"
```

```
            ],
            "Resource": [
                "*"
            ]
        }i
    ]
}
```

Writing Custom Policies

AWS recommends that you use the AWS managed policies whenever possible. However, when you need more control, you can define custom policies.

As shown in the earlier examples, an *IAM policy* is a JSON-style document composed of one or more statements. Each statement has an effect that will either allow or deny access to specific actions on AWS resources. *A deny statement takes precedence over any allow statements*. Use an *Amazon Resource Name (ARN)* to specify precisely the resource or resources to which a custom policy applies.

For example, the following policy authorizes access to the `DeleteLexicon` action in Polly on the resource specified by the ARN. In this case, the resource is a particular lexicon within a specific account and within a specific region.

```
{
  "Version": "2012-10-17",
  "Statement": [{
      "Sid": "AllowDeleteForSpecifiedLexicon",
      "Effect": "Allow",
      "Action": [
         "polly:DeleteLexicon"],
      "Resource": "arn:aws:polly:us-west-2:123456789012:lexicon/awsLexicon"
      }
   ]
}
```

To allow slightly broader permissions in a similar policy, use *wildcards* in the ARN. For example, to allow a user to delete any lexicon within the specified region and account, replace `awsLexicon` with an `*` in the ARN, as shown here:

```
{
  "Version": "2012-10-17",
  "Statement": [{
      "Sid": "AllowDeleteSpecifiedRegion",
      "Effect": "Allow",
```

```
      "Action": [
         "polly:DeleteLexicon"],
      "Resource": "arn:aws:polly:us-east-2:123456789012:lexicon/*"
      }
   ]
}
```

An ARN always starts with `arn:` and can include the following components to identify a particular AWS resource uniquely:

Partition Usually `aws`. For some regions, such as in China, this can have a different value.

Service Namespace of the AWS service.

Region The region in which the resource is located. Some resources do not require a region to be specified.

Account ID The account in which the resource resides. Some resources do not require an account ID to be specified.

Resource The specific resource within the namespace of the AWS service. For services that have multiple types of resources, there may also be a resource type.

These are example formats for an ARN:

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

Here are some examples of ARNs for various AWS resources:

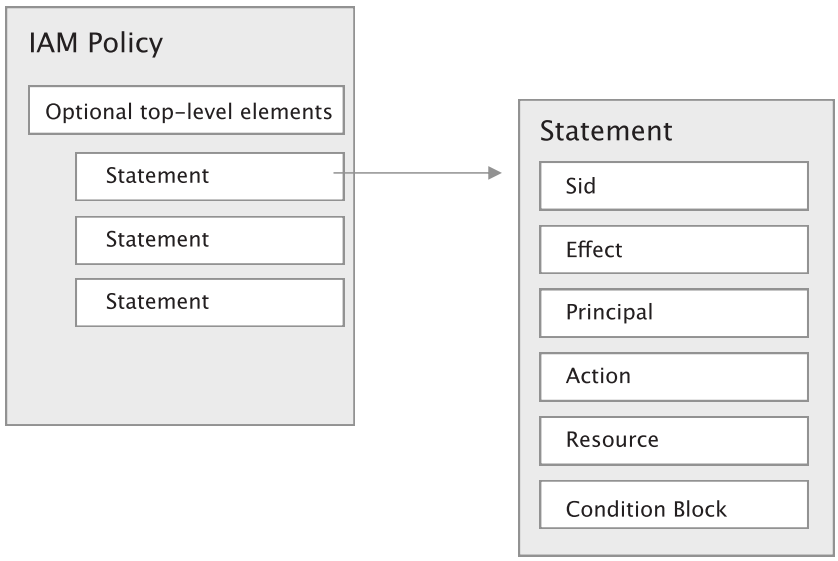
```
<!-- Amazon Polly Lexicon -->
arn:aws:polly:us-west-2:123456789012:lexicon/awsLexicon

<!-- IAM user name -->
arn:aws:iam::123456789012:user/carla


<!-- Object in an Amazon S3 bucket -->
arn:aws:s3:::bucket-name/exampleobject.png
```

A single policy document can have multiple statements. Additional components to a statement may include an optional *statement ID (Sid)* and condition blocks to restrict when the policy applies. If the policy is attached to a resource rather than to an IAM identity, then the policy must also specify a principal (to whom the policy applies), as shown in Figure 1.12.

FIGURE 1.12 IAM policy elements



Write custom policies manually or use tools like the *Visual Policy Editor* in the AWS Management Console to generate policies more easily. To help you test the effects of policies, you can also use the IAM policy simulator at `http://policysim.aws.amazon.com`.

Summary

In this chapter, you learned about the AWS Management Console, the CLI, and the SDKs that you can use to configure and manage resources in the cloud. You learned how to use configuration files, select an AWS region, and manage credentials. The chapter also discussed AWS account root users, IAM, policies, groups, roles, long-term and short-term credentials, the access key ID, and the secret access key.

Exam Essentials

Know the ways to manage AWS resources. Recall that the AWS SDK, AWS CLI, and AWS Management Console are each an option for managing resources within your account.

Know the importance of AWS regions. Be able to identify the impact of region selection on your application code, such as the relationship between region selection and user latency. Also recognize how region selection impacts CLI and SDK calls.

Know about IAM users and IAM roles. Know when it is appropriate to use IAM users or IAM roles for a given application that needs to make AWS calls.

Know how to recognize valid IAM policies. Identify valid IAM policies and predict the effects of policy statements.

Exercises

EXERCISE 1.1

Signing Up for an Account

In this exercise, you'll sign up for an account.

1. Open your browser and go to `http://aws.amazon.com/free`.
2. Choose Create An AWS Account.
3. Provide personal information.
4. Provide payment Information.
5. Verify your phone number.
6. Select a support plan.
7. Choose Sign In To The Console.
8. Sign in to the console.

You are now signed into the AWS Management Console.

EXERCISE 1.2

Creating an IAM Administrators Group and User

In this exercise, you'll define an Administrators group and then add a user to that group. Generate security keys for this user and call this user `DevAdmin`.

1. Sign in to the AWS Management Console (at `http://signin.aws.amazon.com/console`).
2. Click Services and search for **`IAM`**. Take note that this is a global service, and region is not selectable.

EXERCISE 1.2 *(continued)*

3. To view the list of IAM groups, select User Groups.
   If this is a new account, the list is empty.
4. Choose Create Group.
5. For Group Name, enter **`Administrators`**.
6. Under Attach Permissions Policies, select the AdministratorAccess policy.
7. Choose Create Group.
8. To view the list of IAM users, select Users.
   If this a new account, the list is empty.
9. Choose Create User.
10. Set the username to **`DevAdmin`**.
11. Select Provide User Access To The AWS Management Console. Keep the defaults for all other choices.
12. Click Next.
13. To add this user to the Administrators group, select the Administrators Group option under User Groups.
14. Click Next.
15. On the Review And Create page, scroll down to Tags.
16. Provide a tag with a key of **`project`** and a value of **`dev-study-guide`**.
17. Click Create User.
18. Use tags to add customizable key-value pairs to resources so that you can more easily track and manage them.
19. View your password and download the `credentials.csv` file via the Download .csv File button.
20. Rename the file to **`devadmin-credentials.csv`**, and move the file to a folder where you would like to keep it.
21. Sign out of the AWS Management Console by clicking your name in the top bar and selecting Sign Out.

You now have a CSV file that contains a username, password, access key ID, secret access key, and console login link. Use the DevAdmin username, password, and console login link to sign into the AWS Management Console for all future exercises unless otherwise noted. Use the access key to configure the SDK in the following exercises.

EXERCISE 1.3

Installing and Configuring the AWS CLI

In this exercise, you'll install and configure the AWS Command-Line Interface. The AWS CLI requires Python2 or Python3. Install Python using pip, the Python installer.

1. Install the AWS CLI using the instructions for your operating system on this page: `http://docs.aws.amazon.com/cli/latest/userguide/getting-started-install.html`.
2. To configure the AWS CLI with a default profile for credentials, run the following command:

   ```
   aws configure
   ```

3. Enter the following values when prompted:
   - `AWS Access Key ID`: Paste the value from the CSV file you downloaded in Exercise 1.2.
   - `AWS Secret Access Key`: Paste the value from the CSV file you downloaded in Exercise 1.2.
   - `Default region name`: Enter **`us-east-1`**.
   - `Default output format`: Press Enter to leave this blank.
4. Run the CLI command to verify that your CLI is working correctly, and view the available voices for Amazon Polly:

   ```
   aws polly describe-voices --language en-US --output table
   ```

   A table in the terminal lists the available voices for Amazon Polly for the language US English.

EXERCISE 1.4

Downloading the Code Samples

1. If you haven't downloaded the chapter resources from the online test bank, go to `www.wiley.com/go/sybextestprep`.
2. Register to get an access code.
3. Log in and then redeem the access code. The book will be added to the online test bank.

EXERCISE 1.4 *(continued)*

4. Next to Course Dashboard, click Resources.
5. Click Chapter Resources to download the code files.

   You can also download the code files from the book page at

   `https://www.wiley.com/en-us/AWS+Certified+Developer+Study+Guide%3A+Associate+(DVA-C02)+Exam%2C+2nd+Edition-p-9781394274802`
6. Click Downloads to access the online materials for Chapters 1, 2, 11, and 12.

EXERCISE 1.5

Running a Python Script That Makes AWS Cloud API Calls

In this exercise, you'll run the Python script to make an AWS service call.

1. Open a terminal window and navigate to the folder with the book sample code.
2. To install the AWS SDK for Python, run the following command:

   ```
   pip install boto3
   ```
3. Navigate to the `chapter-01` folder where you downloaded the sample code.
4. To generate an MP3 in the `chapter-01` folder, run the `helloworld.py` program:

   ```
   python helloworld.py
   ```
5. To hear the audio, open the generated file, `helloworld.mp3`.
6. (Optional) Modify the Python code to use a different voice. See Exercise 1.3 for an AWS CLI command that provides the list of available voices.

You hear "Hello World" when you play the generated audio file. If you completed the optional challenge, you'll also hear the audio spoken in a different voice from the first audio.

EXERCISE 1.6

Working with Multiple Regions

In this exercise, you'll use Amazon Polly to understand the effects of working with different AWS regions.

1. Open a terminal window and navigate to the folder with the book sample code.

2. Navigate to `chapter-01` in the folder where you downloaded the sample code.
3. Verify that the region is `us-east-1` by running the following command:

   ```
   aws configure get region
   ```

4. Upload `aws-lexicon.xml` to the Amazon Polly service in the default region, which is US East (N. Virginia):

   ```
   aws polly put-lexicon --name awsLexicon --content file://aws-lexicon.xml
   ```

5. The file `helloaws.py` is currently overriding the region to be EU (London). Run the Python code and observe the `LexiconNotFoundException` exception that returns.

   ```
   python.helloaws.py
   ```

6. Upload the lexicon to EU (London) by setting the region to `eu-west-2`:

   ```
   aws polly put-lexicon --name awsLexicon --content file://aws-lexicon.xml
   --region eu-west-2
   ```

7. Run the following Python script again:

   ```
   python helloaws.py
   ```

   Observe that it executes successfully this time and generates an MP3 file in the current folder.

8. Play the generated `helloaws.mp3` file to confirm that it says, "Hello Amazon Web Services."
9. (Optional) Delete the lexicons with the following commands:

   ```
   aws polly delete-lexicon --name awsLexicon
   aws polly delete-lexicon --name awsLexicon --region eu-west-2
   ```

   Even though the text supplied by the API call to synthesize speech was "Hello AWS!" the generated audio file uses the lexicon you uploaded to pronounce it as "Hello Amazon Web Services."

EXERCISE 1.7

Working with Additional Profiles

In this exercise, you'll define a limited user for the account and configure a new profile in the SDK to use these credentials. Notice that the permissions are restrictive for that user, and you need to update the permissions to be more permissive.

1. Sign in to the AWS Management Console using the credentials for `DevAdmin` from Exercise 1.2.

EXERCISE 1.7 *(continued)*

2. Select Services.
3. Select IAM to open the IAM dashboard.
4. Select Users to view the list of IAM users.
5. Click Create User.
6. Set the username to **`DevRestricted`**.
7. Select the option Provide User Access To The AWS Management Console.
8. Select I Want To Create An IAM User, leave the other values at the defaults, and click Next.
9. Select Attach Policies Directly.
10. Filter for the AmazonPollyReadOnlyAccess policy.
11. Click Next.
12. Under Tags, define a tag as follows:
    - Key: **`project`**
    - Value: **`dev-study-guide`**
13. Click Create User.
14. To configure the SDK in the following exercises, click Download .csv File to download the `credentials.csv` file.
15. Rename the downloaded file to **`devrestricted-credentials.csv`** and move it to the same folder where you put the CSV file from Exercise 1.2.
16. Open a terminal window and navigate to the folder with the sample code.
17. Navigate to the `chapter-01` folder.
18. (Optional) Review the code in `upload-restricted.py`.
19. Configure the AWS CLI with a new profile called **`restricted`**. Run the following command:

    ```
    aws configure --profile restricted
    ```

    When prompted, enter the following values:

    - `AWS Access Key ID`: Copy the value from the CSV file you downloaded.
    - `AWS Secret Access Key`: Copy the value from the CSV file you downloaded.
    - `Default region name`: Enter **`us-east-1`**.
    - `Default output format`: Press Enter to keep this blank.

20. Upload the lexicon.

    The upload operation is expected to fail because of the restricted permissions associated with the profile specified in the script. Run the following Python script:

    ```
    python upload-restricted.py
    ```

21. Return to the AWS Management Console for IAM, and in the left navigation, select Users.

22. To view a user summary page, select DevRestricted User.

23. Choose Add Permissions ➢ Add Permissions.

24. Select Attach Policies Directly.

25. To filter out other policies, in the search box, enter **`polly`**, and select the AmazonPollyFullAccess policy.

26. Click Next.

27. Click Add Permissions.

28. Repeat step 20 to upload the lexicon.

    The upload is successful. After the change in permissions, you did not have to modify the credentials. After a short delay, the new policy automatically takes effect on new API calls from `DevRestricted`.

29. Delete the lexicon by running the following command:

    ```
    aws polly delete-lexicon --name awsLexicon --region eu-west-2
    ```

    In this exercise, you have configured the SDK and AWS CLI to refer to a secondary credentials profile and have tested the distinction between the AWS managed IAM policies related to Amazon Polly. You have also confirmed that it is possible to change the permissions of an IAM user without changing the access key used by that user.

Review Questions

1. Which of the following is typically used to sign API (CLI or SDK) calls to AWS services?
   - **A.** AWS KMS Key
   - **B.** AWS access key
   - **C.** IAM username and password
   - **D.** Account number

2. When you make calls to AWS services, for most services those requests are directed at a specific endpoint that corresponds to which of the following?
   - **A.** AWS facility
   - **B.** AWS availability zone
   - **C.** AWS region
   - **D.** AWS edge location

3. When you're configuring a local development machine to make AWS calls, which of the following is the simplest secure method of obtaining and using credentials?
   - **A.** Create an IAM user, assign permissions by adding the user to an IAM group with IAM policies attached, and generate an access key for programmatic access.
   - **B.** Sign in with your email and password, and visit My Security Credentials to generate an access key.
   - **C.** Generate long-term credentials for a built-in IAM role.
   - **D.** Use your existing username and password by configuring local environment variables.

4. You have a large number of employees, and each employee already has an identity in an external directory. How might you manage AWS credentials for each employee so that they can interact with AWS for short-term sessions?
   - **A.** Create an IAM user and credentials for each member of your organization.
   - **B.** Share a single password through a file stored in an encrypted Amazon S3 bucket.
   - **C.** Define a set of IAM roles, and establish a trust relationship between your directory and AWS.
   - **D.** Configure the AWS Key Management Service (AWS KMS) to store credentials for each user.

5. You have a team member who needs access to write records to an existing Amazon DynamoDB table within your account. How might you grant write permission to this specific table and only this table?
   - **A.** Write a custom IAM policy that specifies the table as the resource, and attach that policy to the IAM user for the team member.
   - **B.** Attach the `DynamoDBFullAccess` managed policy to the IAM role used by the team member.
   - **C.** Delete the table and re-create it. Permissions are set when the DynamoDB table is created.
   - **D.** Create a new user within DynamoDB, and assign table write permissions.

6. You created a `Movies` DynamoDB table in the AWS Management Console, but when you try to list your DynamoDB tables by using the Java SDK, you do not see this table. Why?
   A. DynamoDB tables created in the AWS Management Console are not accessible from the API.
   B. Your SDK may be listing your resources from a different AWS region in which the table does not exist.
   C. The security group applied to the `Movies` table is keeping it hidden.
   D. Listing tables is supported only in C# and not in the Java SDK.

7. You make a request to describe voices offered by Amazon Polly by using the AWS CLI, and you receive the following error message:

```
Could not connect to the endpoint URL:
https://polly.us-east-1a.amazonaws.com/v1/voices
```

   What went wrong?
   A. Your credentials have been rejected.
   B. You have incorrectly configured the AWS region for your call.
   C. Amazon Polly does not offer a feature to describe the list of available voices.
   D. Amazon Polly is not accessible from the AWS CLI because it is only in the AWS SDK.

8. To what resource does this IAM policy grant access, and for which actions?

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

   A. The policy grants full access to read the objects in the Amazon S3 bucket.
   B. The policy grants the holder the permission to list the contents of the Amazon S3 bucket called `example_bucket`.
   C. Nothing. The policy was valid only until October 17, 2012 (`2012-10-17`), and is now expired.
   D. The policy grants the user access to list the contents of all Amazon S3 buckets within the current account.

9. When an IAM user makes an API call via the CLI or SDK, that user's long-term credentials are valid in which context?
   A. Only in the AWS region in which their identity resides
   B. Only in the availability zone in which their identity resides
   C. Only in the edge location in which their identity resides
   D. Across multiple AWS regions

10. You have provisioned a user with the managed policy `AmazonS3FullAccess`. A new feature has been added to the S3 API. What steps do you need to take to ensure that the user has access to use this new feature?
    A. Go to the IAM console, find the `AmazonS3FullAccess` policy, and click Update.
    B. Remove and reattach the `AmazonS3FullAccess` policy from the user.
    C. Copy the text of the `AmazonS3FullAccess` policy and use it to create a custom policy with the new permissions.
    D. Do nothing; AWS updates managed policies automatically when services are updated.

11. You have an on-premises application that needs to sample data from all your Amazon DynamoDB tables. You have defined an IAM user for your application called `Table Auditor`. How can you give the `TableAuditor` user read access to new DynamoDB tables as soon as they are created in your account?
    A. Define a custom IAM policy that lists each DynamoDB table. Revoke the access key, and issue a new access key for `TableAuditor` when tables are created.
    B. Create an IAM user and attach one custom IAM policy per AWS region that has DynamoDB tables.
    C. Add the `TableAuditor` user to the IAM role `DynamoDBReadOnlyAccess`.
    D. Attach the AWS managed IAM policy `AmazonDynamoDBReadOnlyAccess` to the `TableAuditor` user.

12. The principals who have access to assume an IAM role are defined in which document?
    A. IAM access policy
    B. IAM trust policy
    C. MS grant token
    D. AWS credentials file

13. A new developer has joined your small team. You would like to help your team member set up a development computer for access to the team account quickly and securely. How do you proceed?
    A. Generate an access key based on your IAM user, and share it with your team member.
    B. Create a new directory with AWS Directory Service, and assign permissions in the AWS Key Management Service (AWS KMS).
    C. Create an IAM user, add it to an IAM group that has the appropriate permissions, and generate a long-term access key.
    D. Create a new IAM role for this team member, assign permissions to the role, and generate a long-term access key.

14. You have developed an app that interacts with Amazon S3 using the Python SDK on a Linux server. You are now building a new application in C#, which will run on a separate Windows Server. What is the most straightforward way to enable your new application to work with S3?

    A. Rewrite your existing Python application in .NET to ensure compatibility with the Windows Server environment.

    B. Go to the Amazon S3 service, and change the supported languages to include .NET.

    C. Install the AWS SDK for .NET on your Windows Server.

    D. Implement a proxy service that accepts your API requests, and translate them to Python.

15. You work for a Virginia-based company, and you have been asked to implement a custom application exclusively for customers in Australia. This application has no dependencies on any of your existing applications. What is a method you use to keep the customer latency to this new application low?

    A. Set up an AWS Direct Connect (DX) between your on-premises environment and US East (N Virginia), and host the application from your own data center in Virginia.

    B. Create all resources for this application in the Asia Pacific (Sydney) region, and manage them from your current account.

    C. Deploy the application to the US East (N Virginia) region, and select Amazon EC2 instances with enhanced networking.

    D. It does not matter which region you select, because all resources are automatically replicated globally.