

Chapter 1

Introduction to AWS Cloud API

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

Domain 2: Security

- ✓ 2.1 Make authenticated calls to AWS services.

Domain 3: Development with AWS Services

- ✓ 3.4 Write code that interacts with AWS services by using APIs, SDKs, and AWS CLI.



Introduction to AWS

The AWS Cloud provides infrastructure services, such as compute, storage, networking, and databases, and a broad set of platform capabilities such as mobile services, analytics, and machine learning (ML). These services are available on demand, through the internet, and with pay-as-you-go pricing.

Think of AWS as a programmable data center. Rather than making a phone call or sending email to provision servers or other resources, you can manage all of your resources programmatically, via *application programming interfaces* (APIs). For example, you can provision virtual servers on demand in minutes and pay only for the compute capacity you use. The same is true for de-provisioning those servers; make a single API call to stop paying for resources that you no longer need. AWS operates many data centers worldwide, so you are not limited to a single data center.

In this chapter, you are introduced to AWS and shown how to make your first API calls. The AWS infrastructure behind the API calls follows. Afterward, you will learn how to manage the API credentials and permissions that you need to make API calls.

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 <https://aws.amazon.com/free>.

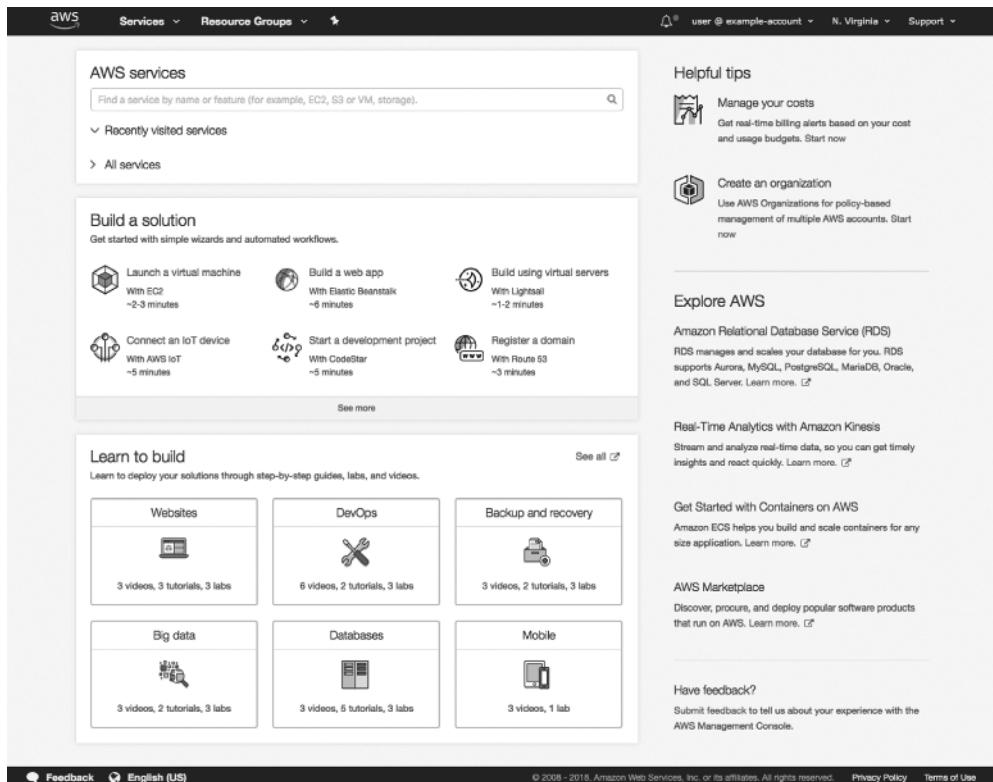
AWS Management Console

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 console as the root user for the account.

The console is a web interface where you can create, configure, and monitor AWS resources in your account. You can quickly identify the AWS services that are available to you and explore the functionality of those services. Links are also provided to learning materials to help you get started.

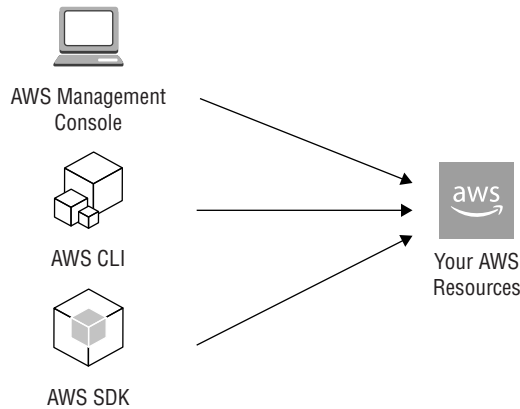
Sign in to the console, as shown in Figure 1.1, at <https://signin.aws.amazon.com/console>.

FIGURE 1.1 AWS Management Console



Because all the functionality of AWS is exposed through APIs, AWS provides more than only the web interface for managing resources. For example, the 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 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 the AWS Mobile SDK and AWS 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.

The Python SDK for AWS is called AWS SDK for Python (Boto). If Python 2 or Python 3 is already installed on your machine, install boto3 using pip, the Python package manager.

To install boto3, open a terminal and run the following command:

```
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 <https://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 CLI*, which is available on Windows, Linux/Unix, and macOS.

The AWS CLI allows you to perform actions similar to those from the SDKs but in an interactive scripting environment. 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 provided, such as the Elastic Beanstalk command line interface and AWS SAM Local. For more information about these tools and installation, see <https://aws.amazon.com/tools/#cli>.

Calling an AWS Cloud Service

The functionality of AWS is powered by web services that are agnostic to the programming language and SDK. In this section, you use the AWS Python SDK to make an API request.

This is an overview of both making an API call and the parameters to configure the SDK. Subsequent sections will describe those parameters.



Locate the API reference documentation about the underlying web services and programming language-specific documentation for each SDK at <https://aws.amazon.com/documentation>.

API Example: Hello World

In the following example, you will make a request to Amazon Polly. *Amazon Polly* provides text-to-speech service with natural-sounding speech, and it is able to provide speech in multiple languages with a variety of male and female voices. Furthermore, you can modify attributes, such as pronunciation, volume, pitch, or speed, by defining *lexicons* or supplying *Speech Synthesis Markup Language* (SSML).

This Python code example uses the AWS SDK for Python (Boto) and Amazon Polly to generate an audio clip that says, “Hello World.”

```
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)
```

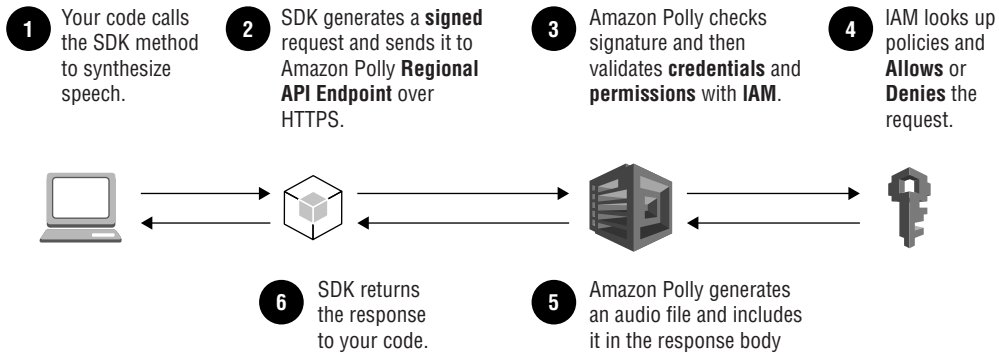
The AWS SDK maps the function call to an HTTPS request to an Amazon Polly API endpoint that is determined by the region name (`region_name`) parameter.

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 *AWS Identity and Access Management* (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 1.3.

FIGURE 1.3 API request and authorization



API Requests

Examine the request that is being transmitted in step 2 of Figure 1.3. When the SDK makes the request to Amazon Polly, it submits a JSON body using a standard HTTP POST to `https://polly.us-west-2.amazonaws.com/v1/speech`.

The SDK sets the following properties in the request:

```
POST /v1/speech HTTP/1.1
host: polly.us-west-2.amazonaws.com
content-Type: application/json
x-amz-date: 20180411T051402Z
authorization: AWS4-HMAC-SHA256 Credential=AKIAIO5FODNN7EXAMPLE/20180411/
us-west-2/polly/aws4_request, SignedHeaders=content-length;content-type;host;x-
amz-date, Signature=d968197e88a6a8de69d1a7bcab414669eecd5f841e13dc90e4a7852
c2c428038

{
  "OutputFormat": "mp3",
  "Text" : "Hello World!",
  "VoiceId": "Aditi"
}
```

Notice that the API endpoint or host URL includes the AWS Region parameter (`us-west-2`). The SDK also generates an authorization header by taking in the AWS access key credentials and applying the AWS Signature Version 4 signing process to the request.

API Responses

The API response corresponds to step 5 of Figure 1.3. For the example API request, the following headers are set in the response:

```
HTTP/1.1 200
status: 200
content-type: audio/mpeg
date: Wed, 11 Apr 2018 05:14:02 GMT
x-amzn-requestcharacters: 12
x-amzn-requestid: 924141bb-b0a6-11e8-b565-b1fabccdcdbd9
transfer-encoding: chunked
connection: keep-alive
```

The headers include a standard HTTP response code. In this case, the status is 200. In general, AWS responds with standard HTTP response codes, such as the following:

- HTTP/200, if successful
- HTTP/403, if authorization is denied

You can also use an *x-amzn-requestid* header to troubleshoot when contacting AWS Support.

The response body includes the audio stream; in this case, the audio stream is in MP3 format.

The AWS SDK wraps the web response and returns an object to the application. This step corresponds to step 6 of Figure 1.3. If the HTTP status code is not HTTP/200, the SDK generates an exception that your code can handle.



The *AWS Signature Version 4* signing process incorporates the current date into the process to sign API requests, so make sure that the clock on the computer making API requests is accurate. AWS API requests must be received within 15 minutes of the timestamp in the request to be valid.

SDK Configuration

In the previous example, the AWS Region and AWS credentials are provided explicitly in the code. The SDK client initialization code from the earlier example is shown again here:

```
# Explicit Client Configuration
polly = boto3.client('polly',
    region_name='us-west-2',
    aws_access_key_id='AKIAIO5FODNN7EXAMPLE',
    aws_secret_access_key='ABCDEF+c2L7yXeGvUyrPgYsDnWRRc1AYEXAMPLE'
)
```

This explicit approach of hardcoding credentials into the code is not recommended, because it carries the risk of checking the credentials into a source-control repository. This would expose the keys to everyone who has access to the repository and could even result in public disclosure. To prevent this, configure the SDK credentials separately from the application source code.

The SDK and AWS CLI automatically check several locations for credentials, and for the region if they are not explicitly provided in the code. These locations include environment variables, programming language-specific parameter stores, and local files.

To configure an AWS access key on your local machine in a local file, create a credentials file in the `.aws` folder in the home folder for the current user. Within this file, specify credentials for the default profile. You may optionally include additional named profiles beyond the default as needed.

```
[default]
aws_access_key_id=AKIAIO5FODNN7EXAMPLE
aws_secret_access_key=ABCDEF+c2L7yXeGvUyrPgYsDnWRRc1AYEXAMPLE
```

From File: `~/.aws/credentials`

Furthermore, hardcoding the AWS Region into the code makes it difficult to deploy your application in different AWS Regions. Instead, create a config file also within the `.aws` folder within your current user's home directory. Within this file, specify a region to use with the default profile.

```
[default]
region = us-west-2
```

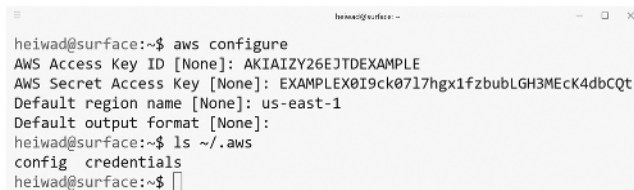
From File: `~/.aws/config`

As an alternative to creating the credentials and config files manually, you can use the AWS CLI to generate the credentials and config files for the default profile as follows:

```
aws configure
```

This command prompts for *credentials* and *region* settings. When the command completes, the config and credentials files are generated, as shown in Figure 1.4.

FIGURE 1.4 Configuring API credentials



```
heiwad@surface:~$ aws configure
AWS Access Key ID [None]: AKIAIZY26EJTDEXAMPLE
AWS Secret Access Key [None]: EXAMPLEX0I9ck0717hgx1fzbubLGH3MEck4dbCQT
Default region name [None]: us-east-1
Default output format [None]:
heiwad@surface:~$ ls ~/.aws
config  credentials
heiwad@surface:~$
```

When the configuration is complete, 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:

```
# 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.

Working with Regions

Now 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 AWS Region is located in a separate geographic area and maintains its own, isolated copies of AWS services. For many AWS services, you are required to select a specific region to process API requests and in which to provision your resources.

FIGURE 1.5 AWS Regions, Availability Zones, and planned regions (as of February 2019)



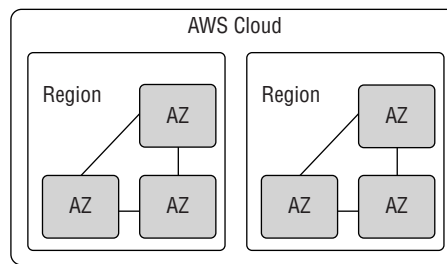
Customers expect that their data is durably held and that services remain highly available. In this section, you will explore how the structure of a region lends itself to providing reliable service and how to choose an appropriate region for your application.

Regions Are Highly Available

Each AWS *Region* contains multiple data centers, grouped together to form *Availability Zones*. Regions are composed of multiple Availability Zones, which allows AWS to provide highly available services in a way that differentiates them from traditional architectures with single or multiple data centers.

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, Availability Zones are connected via low-latency, high-throughput redundant networking.

FIGURE 1.6 Regions and Availability Zones



AWS customers can improve the resilience of their applications by deploying a copy of each application to a second Availability Zone within the same region. This allows the application to remain available to customers even in the face of events that could disrupt an entire data center. Similarly, many of the AWS services automatically replicate data across multiple Availability Zones within an AWS Region to provide high availability and durability of the data.

An example of an AWS service that replicates data across Availability Zones within a region is Amazon Simple Storage Service (Amazon S3). Amazon S3 enable 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 Availability Zones within the region hosting the bucket. This design protects data even against the loss of one entire Availability Zone.

Working with Regional API Endpoints

Many AWS services expose regional API endpoints. When making web service calls to regional endpoints, the region can typically be identified in the URL that you invoke. API calls to a regional endpoint usually affect only the resources within the specific AWS Region that corresponds to that endpoint.

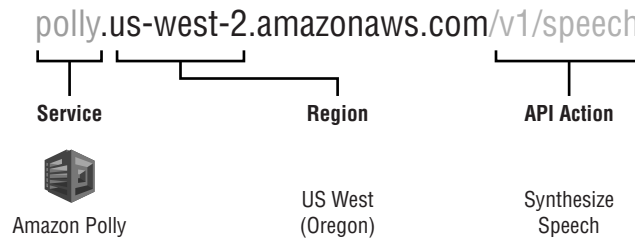
To explore this concept, revisit the previous example of making a request to Amazon Polly to synthesize speech from text.

```
# Initializing SDK Client with Explicit Region Configuration
polly = boto3.client('polly', region_name='us-west-2')
result = polly.synthesize_speech(Text='Hello World!',
                                OutputFormat='mp3',
                                VoiceId='Aditi')
```

To explicitly configure the AWS SDK to use the US West (Oregon) Region, set the `region_name` parameter to `us-west-2` when initializing the SDK client, as in the previous example.

This configuration results in the SDK computing the following URL for the API request, as shown in Figure 1.7.

FIGURE 1.7 A regional API endpoint and API action



You can see regional isolation in practice by uploading a lexicon to Amazon Polly. A *lexicon* stores custom pronunciation information that can be used when synthesizing speech from text. For example, you can require Amazon Polly to expand the acronym AWS to “Amazon Web Services” in the generated audio file by providing the following XML lexicon. The file tells Amazon Polly to speak the *alias* “Amazon Web Services” when it encounters the *grapheme* “AWS” in text.

```
<?xml version="1.0" encoding="UTF-8"?>
<lexicon version="1.0"
  xmlns="http://www.w3.org/2005/01/pronunciation-lexicon"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.w3.org/2005/01/pronunciation-lexicon
    http://www.w3.org/TR/2007/CR-pronunciation-lexicon-20071212/pls.xsd"
  alphabet="ipa"
  xml:lang="en-US">
  <lexeme>
    <grapheme>AWS</grapheme>
    <alias>Amazon Web Services</alias>
  </lexeme>
</lexicon>
File: aws-lexicon.xml
```

To use this lexicon when you synthesize speech, you must first upload it to Amazon Polly. The following shell snippet uses the AWS CLI to upload to the lexicon to the specified region:

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

You can use the `awsLexicon` after it is uploaded. The following example generates a speech request that will be customized by the lexicon. This request is also being made to the `us-west-2` API endpoint.

```
# Synthesizing speech with custom lexicon in the same region
aws polly synthesize-speech --text 'Hello AWS World!' --voice-id Joanna
--output-format mp3 hello.mp3 --lexicon-names="awsLexicon" --region us-west-2
{
  "ContentType": "audio/mpeg",
  "RequestCharacters": "15"
}
```

Assuming that the CLI is configured correctly with an appropriate access key, this request succeeds. In the downloaded audio file, you will hear Joanna say “Hello Amazon Web Services World,” confirming that the lexicon is in effect.

However, if you run the same API request again, but change the region to the US East (N. Virginia) Region, as in the following example, you will get a different result:

```
# Trying again against a different Regional API endpoint
aws polly synthesize-speech --text 'Hello AWS World' --voice-id Joanna --output-
format mp3 hello-custom.mp3 --lexicon-names="awsLexicon" --region us-east-1
```

An error occurred (`LexiconNotFoundException`) when calling the `SynthesizeSpeech` operation: Lexicon not found

In this case, an error occurs because the `awsLexicon` resides only in the US West (Oregon) Region where you placed it. When working with AWS services that are regional in scope, you are in control over where the data resides—AWS does not automatically copy your data for these services to other regions without an explicit action on your part. If you must use the lexicon in regions other than US West (Oregon), you could upload the lexicon to each region in which you plan to use it.

Identifying AWS Regions

When working with AWS services, the AWS Management Console refers to regions differently from the parameters used in the AWS CLI and SDK.

Table 1.1 lists 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 console, the region selector in the upper-right corner of the console displays “Global.” The API endpoint for IAM is the same regardless of the region. Table 1.2 lists some API endpoints.

TABLE 1.2 Selected IAM Service API Endpoints

Region Name	API Endpoint
US East (N. Virginia)	iam.amazonaws.com
US East (Ohio)	iam.amazonaws.com
US West (N. California)	iam.amazonaws.com

In the case of IAM, having IAM resources available in multiple regions is a useful strategy. IAM provides a way to create API credentials, and this means you can use the same set of API credentials to access resources in different AWS Regions.

For each AWS service, you can find the regions in which that service is available, along with the corresponding API endpoints, in the AWS General Reference documentation. The following link provides a comprehensive list of AWS services and their regional API endpoints: <https://docs.aws.amazon.com/general/latest/gr/rande.html>.



The exam may ask you to identify a URL or endpoint for an AWS resource, such as an Amazon S3 bucket, that has been deployed to a specific region. While the test does not require memorization of the region list, AWS recommends that you become familiar with the naming convention for regions and how it is related to the naming convention for Availability Zones.

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 include latency, price, and data residency. Table 1.3 describes selection criteria to include when choosing an AWS Region.

TABLE 1.3 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 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.

API Credentials and AWS Identity and Access Management

Now that you have seen how to make API calls and identified the infrastructure provided by the AWS Cloud, take a closer look at the access keys needed to make API calls. In AWS, an *access key* is a type of security credential that is associated with an identity. So, to make API calls, first you will create an identity in AWS Identity and Access Management (IAM).

To manage authentication and authorization for people or applications, IAM provides users, groups, and roles as identities that you can manage. IAM authenticates the security credentials used to sign an API call to verify that the request is coming from a known identity. Then, IAM authorizes the request by evaluating the policies associated with the identity and resources affected by the request. This section provides reviews users, groups, roles, and policies.



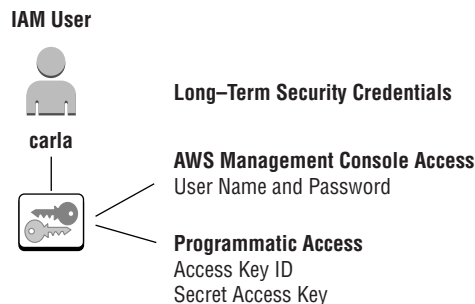
When you first create an account and sign in with your email address and password, you are authenticating as the root user for your account. Few AWS operations require root user permissions. To protect your account, do not generate an access key based on the root user. Instead, create an IAM user and generate an access key for that user. To provide administrator access, add that user to a group that provides administrator permissions.

Users

IAM users can be assigned long-term security credentials. You might create an IAM user when you have a new team member or application that needs to make AWS API calls. Manage the API permissions of the user by associating permissions policies with the user or adding the user to a group that has permissions policies associated with it.

After you create an IAM user, you can assign credentials to allow AWS Management Console access, programmatic access, or both, as shown in Figure 1.8.

FIGURE 1.8 IAM user long-term credentials



AWS Management Console Access

To sign in to the console, IAM users authenticate with an IAM user name and password. As part of the sign-in process, IAM users are prompted to provide either the account ID or alias so that IAM user names only need to be unique within your account. If *multi-factor authentication* (MFA) is enabled for an IAM user, they must 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

For *programmatic access to AWS*, create an access key for the IAM user. An *AWS access key* is composed of the following two distinct parts:

1. Access key ID
2. Secret access key

Here is an example of an AWS access key:

```
aws_access_key_id = AKIAJXR7IOGGTEIVNX7Q
aws_secret_access_key: oe/H0e2Ptj/fvwr dj6Wedo43Vsm05DHDADZ+tnP5
```

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.

Other Credentials for IAM Users

In addition to passwords, multifactor devices, and access keys, IAM users can have other types of security credentials. You can have X.509 certificates, which are used with SOAP APIs, or you can have GIT credentials as either Secure Shell (SSH) keys or passwords to interact with the AWS CodeCommit service.

Groups

To help you manage the permissions of collections of IAM users, IAM provides *IAM groups*. IAM groups do not have their own credentials, but when an IAM user makes an API call with their access key, AWS looks up that user's group memberships and finds the relevant permissions policies. 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. In this way, each team member has unique credentials while they are also given the same permissions.

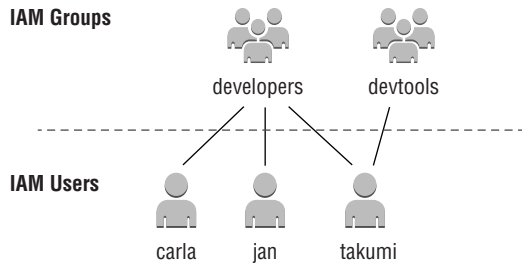
You may create additional groups. For example, you may create a second group for the team members responsible for changing the build and deployment pipeline to which you can assign a name such as *devtools*.

The relationship between IAM users and IAM groups is *many-to-many*. 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. 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.

In the example shown in Figure 1.9, *carla* inherits permissions from the IAM user *carla* and from the group *developers*, and *takumi* inherits the union of all of the policies from *developers* and from *devtools*, in addition to any policies directly associated with *takumi*.

FIGURE 1.9 IAM groups and IAM users



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

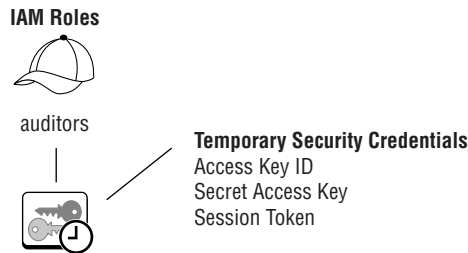
There are situations in which you might not want to create and manage new sets of long-term credentials for team members or applications.

In a large company with many employees, you can use your existing corporate identity store instead of creating new identities and credentials for each team member who manages AWS.

Alternatively, you may delegate permissions to an AWS service to perform actions on your behalf. One common example of this is when application code running on an AWS *compute* service, such as Amazon EC2, needs permissions to make AWS API calls. In this case, AWS recommends allowing Amazon EC2 to manage the credentials for each instance.

In both situations, rather than creating new IAM users, create an *IAM role* to assign permissions. IAM roles can be assumed for short-term sessions, as shown in Figure 1.10.

FIGURE 1.10 IAM roles



To control access to an IAM role, define a *trust policy* that specifies which *principals* can assume a role. Potential principals include AWS 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.

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

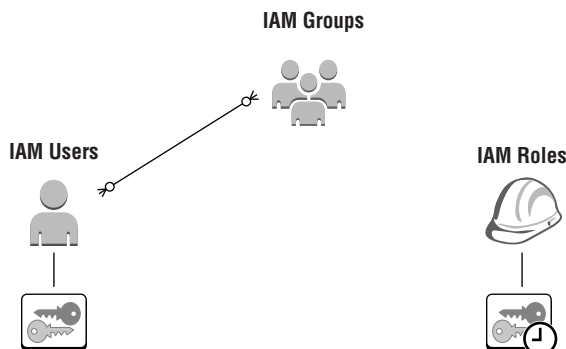
```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "ec2.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

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 Service* (AWS STS). 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 credentials that are generated when the role is assumed:

```
{
  "AccessKeyId": "ASIAJHP2KG65VIKQU2XQ",
  "SecretAccessKey": "zkvPEbYxCLVVD0seWdRnesc8krNDPHEX1cFMyI5W",
  "SessionToken":
    "FQoDYXdzEMf////////wEaDL1b0Wd7VTA3J25cNyL4ARzNSRczH4U3f8gJwi1W8XiDLWJIE9EdX4l4KXTiST40gPoWc9Do9QkcN2xRHk6/qVT6W23d0u6+5YFY9C2wnoEeTTmiQBT5SMjqku5MYlhrCDyFQAVbo6RKUeOZXXSG8REshuFGBtaCNmv95LFF6srCT1b4FZtTtULE7WV3LMcDs6Z2XuN+6aGTawhY50RMnlKRL1w6yHq++RysQWbBHkuNeK/VqjueDINFODPOje9ZnYePVjR5uLmL8ZARWYVBFRB2tpxG07/dseUS902q1hMP8DJuEfsbaiK2ASsmXSRA8v0Znuu4AsBq6ERasBw5EcpICP/Ne8zdK0/93tYF",
  "Expiration": "2018-04-18T22:55:59Z"
}
```

When these short-term credentials are used, AWS looks up the permissions policies associated with the IAM role that was assumed. This is true even if the principal that assumed the role was an IAM user—policies that were associated with the IAM user or their groups are not evaluated when the role credentials are used to make a request. The IAM role is a distinct identity with its own permissions. Furthermore, you cannot nest IAM roles or add IAM roles to IAM groups, as shown in Figure 1.11.

FIGURE 1.11 IAM roles are distinct from IAM users and groups.

Choosing IAM Identities

Consider the following to determine how to define authorization and authentication.

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. The associated long-term credentials are easy to work with on a local development laptop, or on other hardware in your control, such as on-premises servers. 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 disable their corporate access in only one place—the external directory.

Use roles in cases in which you need to make AWS API 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.

Table 1.4 describes the use cases for IAM identities.

TABLE 1.4 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

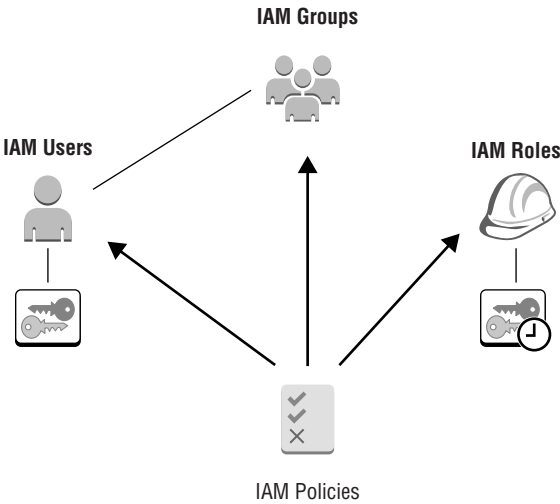


The exam tests your knowledge of the recommended practices for distributing AWS credentials to your code depending on where that code is running.

Managing Authorization with Policies

Manage the permissions for each user, group, or role by assigning *IAM policies* that either *allow* or *deny* permissions to specific API actions, as shown in Figure 1.12. Any API 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 1.12 IAM policies and IAM identities



One method of granting permissions is to use AWS managed policies. AWS provides these policies to support common tasks and are automatically updated as new services and API operations are added.

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. If they need more access later, they can ask for it, and you can update the permissions then.

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 Amazon 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, this operation modifies a state in Amazon Polly. To grant permissions to these actions, 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 Amazon Polly API actions.

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

```

```

        "*"
    ]
    }
}

```

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 API 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 Amazon 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:

```

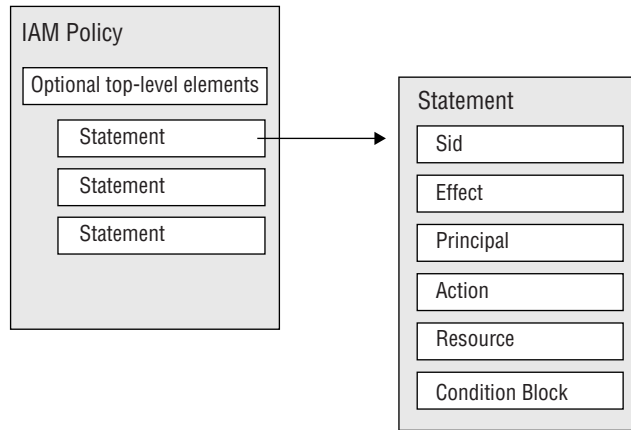
<!-- Amazon Polly Lexicon -->
arn:aws:polly:us-west-2:123456789012:lexicon/awsLexicon

<!-- 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 1.13.

FIGURE 1.13 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 <https://policysim.aws.amazon.com>.

Summary

In this chapter, you learned about the AWS Management Console, the AWS CLI, and the AWS SDKs that AWS uses to configure and manage your resources. You learned how to make API request calls to the AWS Cloud, use configuration files, select an AWS Region, manage AWS API credentials, and identify regional API endpoints. The chapter also discussed AWS account root users, IAM, IAM policies, IAM groups, IAM 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 the AWS Management Console are options for managing the AWS resources within your account.

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

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 API calls.

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

Resources to Review

AWS Free Tier:

<https://aws.amazon.com/free>

Getting Started Resource Center: Create an AWS Account:

<https://aws.amazon.com/getting-started>

Tracking Your Free Tier Usage:

<https://docs.aws.amazon.com/awsaccountbilling/latest/aboutv2/tracking-free-tier-usage.html>

AWS Documentation:

<https://aws.amazon.com/documentation>

AWS Management Console:

<https://signin.aws.amazon.com/console>

AWS Command Line Interface (AWS CLI):

<https://docs.aws.amazon.com/cli/latest/userguide/cli-chap-welcome.html>

SDKs, Toolkits, and Installation Directions:

<https://aws.amazon.com/tools/#sdk>

Amazon Polly:

<https://docs.aws.amazon.com/polly/latest/dg/what-is.html>

AWS Regions and Regional API Endpoints:

<https://docs.aws.amazon.com/general/latest/gr/rande.html>

AWS IAM Documentation:

<https://docs.aws.amazon.com/IAM/latest/UserGuide/getting-started.html>

AWS IAM Best Practices:

<https://docs.aws.amazon.com/IAM/latest/UserGuide/best-practices.html>

AWS IAM FAQs:

<https://aws.amazon.com/iam/faqs/>

AWS Signature Version 4 Signing Process:

<https://docs.aws.amazon.com/general/latest/gr/signature-version-4.html>

AWS Whitepapers (Introduction to AWS):

<https://aws.amazon.com/whitepapers/>

AWS Training and Certification:

<https://aws.amazon.com/training>

AWS Events and Webinars:

<https://aws.amazon.com/about-aws/events>

AWS Glossary:

<https://docs.aws.amazon.com/general/latest/gr/glos-chap.html>

Exercises

EXERCISE 1.1

Sign Up for an Account

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

1. Open your browser and go to <https://aws.amazon.com/free/>.
2. Choose **Create a Free 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 in to the AWS Management Console.

EXERCISE 1.2

Create an IAM Administrators Group and User

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

1. Sign in to the AWS Management Console (at signin.aws.amazon.com/console).
 2. Select **All Services**.
 3. To open the IAM dashboard, select **IAM**.
-

-
4. To view the list of IAM groups, select **Groups**.
If this is a new account, the list is empty.
 5. Choose **Create New Group**.
 6. For **Group Name**, enter **Administrators**.
 7. Choose **Next Step**.
 8. On the **Attach Policy** page, select the AdministratorAccess policy.
 9. Choose **Next Step**.
 10. On the **Review** page, choose **Create Group** to create the Administrators group.
 11. To view the list of IAM users, select **Users**.
If this is a new account, the list is empty.
 12. Choose **Add user**.
 13. Set the user name to **DevAdmin**.
 14. Select both Access type check boxes: **Programmatic access** and **AWS Management Console access**.
 15. Choose **Next: Permissions**.
 16. To add this user to the Administrators group, select the **Administrators group** check box.
 17. Clear the **Require password reset** check box.
 18. Choose **Next: Tags**.
 19. Provide a tag with a key of **project** and a value of **dev-study-guide**.
Use tags to add customizable key-value pairs to resources so that you can more easily track and manage them.
 20. Choose **Next: Review**.
 21. Choose **Create user**.
 22. Download the `credentials.csv` file.
 23. Rename the file to **devadmin-credentials.csv**, and move the file to a folder where you would like to keep it.
 24. 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 user name, password, access key ID, secret access key, and console login link. Use the DevAdmin user name, password, and console sign-in link to sign in to 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**Install and Configure the AWS CLI**

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

1. Install Python from <https://www.python.org/downloads/>.
2. Open a terminal window.
3. To install the AWS CLI, run the following command:

```
pip install aws-cli --upgrade --user
```
4. (Optional) If you encounter issues with step 3, review the AWS CLI Installation guide for alternative installation options here:
<https://docs.aws.amazon.com/cli/latest/userguide/installing.html>
5. To configure the AWS CLI with a default profile for credentials, run the following command:

```
aws configure
```
6. Enter the following values when prompted:
 - AWS Access Key ID: Paste the value from the CSV you downloaded in Exercise 1.2.
 - AWS Secret Access Key: Paste the value from the CSV you downloaded in Exercise 1.2.
 - Default region name: Enter us-east-1.
 - Default output format: Press Enter to leave this blank.
7. 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**Download the Code Samples**

In this exercise, you'll download the code snippets to execute future exercises.

1. If you do not already have Git installed, install it from <https://git-scm.com/downloads>.
 2. Open a command terminal.
-

-
3. Create a folder on the hard drive to store the examples.
 4. Navigate to a folder to host the code.
 5. To download the samples using Git, run the following command:

```
git clone http://example.com/example.git
```

A folder named <<example>> contains the code examples for this study guide.

EXERCISE 1.5

Run a Python Script that Makes AWS API Calls

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

1. Open a terminal window and navigate to the folder with the book sample code.
2. To install the AWS SDK for Python (Boto), 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 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
```
-

(continued)

EXERCISE 1.6 (continued)

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` 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 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 and that you need to update the permissions for that user to be more permissive.

1. Sign in to the AWS Management Console (at `aws.amazon.com`) using the credentials for `DevAdmin` from Exercise 1.2.
 2. Select **Services**.
 3. Select **IAM** to open the IAM dashboard.
 4. Select **Users** to view the list of IAM users.
-

-
5. Choose **Add user**.
 6. Set the user name to **DevRestricted**.
 7. For **Access type**, select **Programmatic access**.
 8. Choose **Next Permissions**.
 9. Select **Attach existing policies directly**.
 10. Select the **AmazonPollyReadOnlyAccess** policy.
 11. To narrow the options, in **Filter**, enter **polly**.
 12. Choose **Next: Tags**.
 13. Define a tag as follows:
 - Key: **project**
 - Value: **dev-study-guide**
 14. Choose **Next: Review**.
 15. Choose **Create User**.
 16. To configure the SDK in the following steps, download the `credentials.csv` file.
 17. 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.
 18. Open a terminal window and navigate to the folder with the sample code.
 19. Navigate to the `chapter-01` folder.
 20. (Optional) Review the code in `upload-restricted.py`.
 21. 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 you downloaded.
 - AWS Secret Access Key: Copy the value from the CSV you downloaded.
 - Default region name: Enter **us-east-1**.
 - Default output format: Press Enter to retain the default setting.
 22. 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
```
-

(continued)

EXERCISE 1.7 (continued)

23. Return to the AWS Management Console for IAM, and in the left navigation, select **Users**.
24. To view a user summary page, select **DevRestricted user**.
25. Choose **Add permissions**.
26. Select **Attach existing policies directly**.
27. To filter out other policies, in the search box, enter **polly**, and select the **AmazonPollyFullAccess** policy.
28. Choose **Next: Review**.
29. Choose **Add permissions**.
30. Repeat step 22 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.

31. 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 calls to AWS services?
 - A. Customer master key (CMK)
 - B. AWS access key
 - C. IAM user name and password
 - D. Account number
2. When you make API 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 API calls, which of the following is the simplest secure method of obtaining an API credential?
 - 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 user name 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 API 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 recreate 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?
- DynamoDB tables created in the AWS Management Console are not accessible from the API.
 - Your SDK may be listing your resources from a different AWS Region in which the table does not exist.
 - The security group applied to the `Movies` table is keeping it hidden.
 - Listing tables is supported only in C# and not in the Java SDK.
7. You make an API 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?
- Your API credentials have been rejected.
  - You have incorrectly configured the AWS Region for your API call.
  - Amazon Polly does not offer a feature to describe the list of available voices.
  - 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"
  }
}
```
- The policy grants full access to read the objects in the Amazon S3 bucket.
 - The policy grants the holder the permission to list the contents of the Amazon S3 bucket called `example_bucket`.
 - Nothing. The policy was valid only until October 17, 2012 (2012-10-17), and is now expired.
 - 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, that user's long-term credentials are valid in which context?
- Only in the AWS Region in which their identity resides
 - 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. When you use identity federation to assume a role, where are the credentials you use to make AWS API calls generated?
 - A. Access key ID and secret access key are generated locally on the client.
 - B. The AWS Security Token Service (AWS STS) generates the access key ID, secret access key, and session token.
 - C. The AWS Key Management Service (AWS KMS) generates a customer master key (CMK).
 - D. Your Security Assertion Markup Language (SAML) identity provider generates the access key ID, secret access key, and session token.
- 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 `TableAuditor`. How can you give the `TableAuditor` user read access to new DynamoDB tables as soon 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 been working with the Amazon Polly service in your application by using the Python SDK for Linux. You are building a second application in C#, and you would like to run that application on a separate Windows Server with .NET. How can you proceed?

 - A.** Migrate all your code for all applications to C#, and modify your account to a Windows account.
 - B.** Go to the Amazon Polly service, and change the supported languages to include .NET.
 - C.** Install the AWS SDK for .NET on your Windows Server, and leave your existing application unchanged.
 - D.** Implement a proxy service that accepts your API requests, and translate them to Python.
- 15.** You are 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.