**CHAPTER 7**

Deploying and Monitoring Applications on AWS

In this chapter, you will

•   Learn about serverless applications

•   Be introduced to AWS Lambda

•   Learn about API Gateway

•   Learn about Amazon Kinesis Data Steams, Amazon Kinesis Data Firehose, and Amazon Kinesis Data Analytics

•   Explore Amazon CloudFront, Amazon Route 53, AWS WAF, and AWS Shield

•   Learn about AWS SQS, SNS, and Step Functions

•   Learn about Elastic Beanstalk and AWS OpsWorks

•   Understand Amazon Cognito

•   Learn about Amazon Elastic MapReduce

•   Learn about AWS CloudFormation

•   Learn how to monitor the AWS services by exploring the monitoring services such as Amazon CloudWatch, AWS CloudTrail, AWS Config, VPC Flow Logs, and AWS Trusted Advisor

•   Learn how to manage multiple AWS accounts using AWS Organizations

AWS LAMBDA

When you’re building applications, you want them to deliver a great experience for your users. Maybe you want your application to generate in-app purchase options during a gaming session, rapidly validate street address updates, or make image thumbnails available instantly after a user uploads photos. To make this magic happen, your application needs back-end code that runs in response to events such as image uploads, in-app activity, web site clicks, or censor outputs. But managing the infrastructure to host and execute back-end code requires you to size, provision, and scale a bunch of servers; manage operating system updates; apply security patches; and then monitor all this infrastructure for performance and availability. Wouldn’t it be nice if you could just focus on building great applications without having to spend a lot of time managing servers?

AWS Lambda is a compute service that runs your back-end code in response to events such as object uploads to Amazon S3 buckets, updates to Amazon DynamoDB tables, data in Amazon Kinesis Data Streams, or in-app activity. Once you upload your code to AWS Lambda, the service handles all the capacity, scaling, patching, and administration of the infrastructure to run your code and provides visibility into performance by publishing real-time metrics and logs to Amazon CloudWatch. All you need to do is write the code.

AWS Lambda is low cost and does not require any up-front investment. When you use AWS Lambda, you’re simply charged a low fee per request and for the time your code runs, measured in increments of 100 milliseconds. Getting started with AWS Lambda is easy; there are no new language tools or frameworks to learn, and you can use any third-party library and even native ones. The code you run on AWS Lambda is called a *Lambda function*. You just upload your code as a ZIP file or design it in the integrated development environment in the AWS Management Console, or you can select prebuilt samples from a list of functions for common use cases such as image conversion, file compression, and change notifications. Also, built-in support for the AWS SDK makes it easy to call other AWS services. Once your function is loaded, you select the event source to monitor such as an Amazon S3 bucket or Amazon DynamoDB table, and within a few seconds AWS Lambda will be ready to trigger your function automatically when an event occurs. With Lambda, any event can trigger your function, making it easy to build applications that respond quickly to new information.

Is AWS Lambda Really Serverless?

AWS Lambda is a compute service, and the biggest advantage of using AWS Lambda is you don’t have to provision or manage any infrastructure. It is a serverless service. I’ll first explain what is meant by *serverless*. If the platform is to be considered serverless, it should provide these capabilities at a minimum:

•   **No infrastructure to manage** As the name *serverless* implies, there should not be any infrastructure to manage.

•   **Scalability** You should be able to scale up and down your applications built on the serverless platform seamlessly.

•   **Built-in redundancy** The serverless platform should be highly available at all times.

•   **Pay only for usage** On the serverless platform, you have to pay only when you are using the service; if you are not using the service, you don’t have to pay anything. For example, by using Lambda, you pay only when your code is running. If your code is not running, you don’t pay anything. Also, the biggest savings come from the fact that you aren’t paying for a server or the costs associated with running a server, such as data center costs, which include power, cooling, networking, and floor space.

If you study these four characteristics carefully, you will realize that many AWS services that you have studied elsewhere in the book are serverless. Specifically, these AWS services are serverless:

•   Amazon S3

•   Amazon DynamoDB

•   Amazon API Gateway

•   AWS Lambda

•   Amazon SNS and SQS

•   Amazon CloudWatch Events

•   Amazon Kinesis

You may be wondering whether *serverless* is really serverless. Aren’t there any servers running behind the scenes? You’re right; serverless does not literally mean no servers. There are fleets of EC2 servers running behind the scenes to support the serverless infrastructure. AWS takes care of the provisioning, management, stability, and fault tolerance of the underlying infrastructure. AWS keeps everything ready for you; you just need to use the service. For example, for S3, all the infrastructure is already provisioned; you just need to upload your content. Similarly, for Lambda, you just need to execute your code. Since you don’t have to deal with the server infrastructure in the back end, these services are called *serverless*.

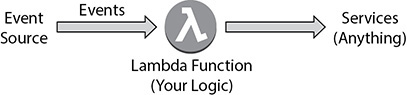
Understanding AWS Lambda

By using AWS Lambda, you get all the benefits obtained via a serverless platform. You are charged based on the number of requests for your functions and their duration (the time it takes for your code to execute). This cost is also based on the memory consumption. You don’t pay anything when your code isn’t running.

With Lambda, you can run code for virtually any type of application or back-end service. Lambda runs and scales your code with high availability. Each Lambda function you create contains the code you want to execute, the configuration that defines how your code is executed, and, optionally, one or more event sources that detect events and invoke your function as they occur.

An event source can be an Amazon SNS function that can trigger the Lambda function, or it can be an API Gateway event (covered in the next section of this book) that can invoke a Lambda function whenever an API method created with API Gateway receives an HTTPS request. There are lots of event sources that can trigger a Lambda function, such as Amazon S3, Amazon DynamoDB, Amazon Kinesis, Amazon CloudWatch, and so on. For the examination, you don’t have to remember all the event sources.

[Figure 7-1](https://learning.oreilly.com/library/view/aws-certified-solutions/9781260470192/ch7.xhtml#fig7_1) shows what the simplest architecture of AWS Lambda looks like.



**Figure 7-1** Architecture of a running AWS Lambda function

After you have configured an event source, as soon as the event occurs (the event can be an image upload, in-app activity, web site click, and so on), your code is invoked (as a Lambda function). The code can be anything; it can be business logic or whatever end result you want. You will look at a couple of reference architectures using Lambda in the “Reference Architectures Using Serverless Services” section, which will give you more exposure to various use cases.

You can run as many Lambda functions in parallel (or concurrently) as you need; there is no limit to the number of Lambda functions you can run at any particular point of time, and they scale on their own. Lambda functions are “stateless,” with no affinity to the underlying infrastructure so that Lambda can rapidly launch as many copies of the function as needed to scale to the rate of incoming events. AWS Lambda allows you to decouple your infrastructure since it provides you with the ability to replace servers with microprocesses. As a result, building microservices using Lambda functions and API Gateway is a great use case.

With AWS Lambda, you can use the normal language and operating system features, such as creating additional threads and processes. The resources allocated to the Lambda function, such as memory, disk, runtime, and network usage, must be shared among all the processes the function uses. The processes can be launched using any language supported by Amazon Linux.

The following are the high-level overview of steps you need to take to use AWS Lambda:

**1.**   Upload the code to AWS Lambda in ZIP format. Alternatively, you can author the code from scratch or browse the serverless application repository to get some sample code.

**2.**   Add an IAM role to the function or create a new role to run the function.

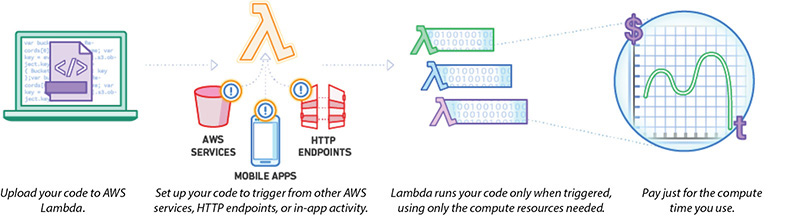
**3.**   Once the function is created, add a trigger to invoke the Lambda function.

**4.**   Configure the destination details, which send invocation records to a destination when your function is invoked asynchronously or if your function processes records from a stream.

**5.**   Once the function is created, you can schedule it and specify how often the function will run. You can also specify whether the function is driven by an event and, if so, what the source of the event is.

**6.**   You can specify various configurations for the Lambda function, such as the compute resource for the event (which can be from 128MB to 3008MB of memory), the timeout period for the event, VPC details, and so on.

Figure 7-2 summarizes how Lambda works.



**Figure 7-2** How AWS Lambda works

AWS Lambda natively supports the following languages:

•   Java

•   Node.js

•   Python

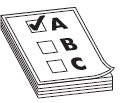
•   C#

•   Go

•   PowerShell

•   Ruby

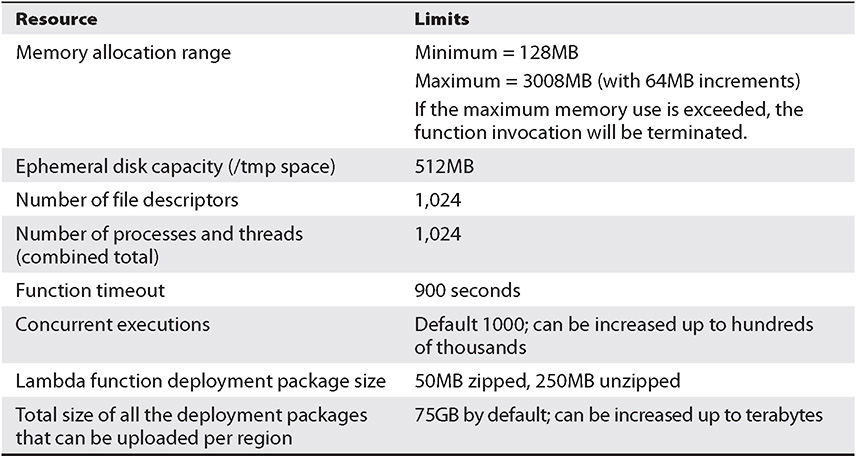
In addition, AWS Lambda provides a runtime API that allows you to use any additional programming languages to author your functions.



**EXAM TIP**     Remember these languages; there might be a question on the exam about the languages that AWS Lambda supports.

AWS

It is important to know the resource limits of AWS Lambda so that you can find the right use case for Lambda, as shown in [Table 7-1](https://learning.oreilly.com/library/view/aws-certified-solutions/9781260470192/ch7.xhtml#tab7_1). For example, if you want a job to run for 12 hours, you won’t be able to do that via AWS Lambda since the maximum execution duration per request is 300 seconds, or 5 minutes.



**Table 7-1** AWS Lambda Resource Limits per Invocation

Also, there is a limit of 1,000 concurrent executions, but just like any other AWS service, you can increase the service limit by creating a support ticket or case.

LAMBDA USAGE PATTERN

The following are some of the most common real-life usage patterns of Lambda. Since Lambda can be directly triggered by AWS services, such as Amazon S3, DynamoDB, Amazon Kinesis Data Streams, Amazon Simple Notification Service (Amazon SNS), and Amazon CloudWatch, it allows you to build a variety of real-time data-processing systems.

•   **Real-time File Processing** You can trigger Lambda to invoke a process when a file has been uploaded to Amazon S3 or modified. For example, you might change an image from color to grayscale after it has been uploaded to Amazon S3.

•   **Real-time Stream Processing** You can use Kinesis Data Streams and Lambda to process streaming data for click stream analysis, log filtering, and social media analysis.

•   **Extract, Transform, Load** You can use Lambda to run code that transforms data and loads that data into one data repository or another.

•   **Replace Cron** Use schedule expressions to run a Lambda function at regular intervals as a cheaper and more available solution than running cron on an EC2 instance.

•   **Process AWS Events** You can use many other services, such as AWS CloudTrail, to act as event sources simply by logging in to Amazon S3 and using S3 bucket notifications to trigger Lambda functions.

AMAZON API GATEWAY

Architecting, deploying, maintaining, and monitoring an API are time-consuming and challenging tasks. If you want to continuously improve as well, this is an even bigger challenge. Often you have to run different versions of the same APIs to maintain the backward compatibility of the APIs for all the clients. The effort required can increase depending on which phase of the development cycle you are in (development, testing, or production).

Also, it is important to handle the access authorization aspect for every API. It is a critical feature for all APIs but complex to build and involves repetitive work. When an API is published and becomes successful, the next challenge is to manage, monitor, and monetize the ecosystem of third-party developers utilizing the API.

Other challenges of developing APIs are throttling requests to protect the back end, caching API responses, transforming requests and responses, and generating API definitions. Sometimes documentation with tools adds to the complexity.

Amazon API Gateway not only addresses those challenges but also reduces the operational complexity of creating and maintaining RESTful APIs.

API Gateway is a fully managed service that makes it easy for developers to define, publish, deploy, maintain, monitor, and secure APIs at any scale. Clients integrate with the APIs using standard HTTPS requests. API Gateway serves as a front door (to access data, business logic, or functionality from your back-end services) to any web application running on Amazon EC2, Amazon ECS, AWS Lambda, or on-premises environment. It has specific features and qualities that result in it being a powerful edge for your logic tier. Thus, you can use API Gateway in the following ways:

•   To create, deploy, and manage a RESTful API to expose back-end HTTP endpoints, AWS Lambda functions, or other AWS services

•   To invoke exposed API methods through the front-end HTTP endpoints

API Gateway is capable of handling all the tasks involved in processing hundreds of thousands of concurrent API calls. It handles the day-to-day challenges for managing an API very well. For example, it can do traffic management, it is able to handle the authorization and access control, it can take care of the monitoring aspect, it can do version control, and so on. It has a simple pay-as-you-go pricing model where you pay only for the API calls you receive and the amount of data transferred out. There are no minimum fees or startup costs.

API Types Supported by API Gateway

Here are the API types supported by the API Gateway:

•   **REST API** REST APIs offer API proxy functionality and API management features in a single solution. REST APIs offer API management features such as usage plans, API keys, publishing, and monetizing APIs.

•   **WebSocket API** Using WebSocket APIs you can maintain a persistent connection between the clients, thereby enabling real-time message communication.

•   **HTTP API** HTTP APIs are optimized for building APIs that proxy to AWS Lambda functions or HTTP back ends, making them ideal for serverless workloads.

Benefits of Amazon API Gateway

These are some of the benefits that you get by using Amazon API Gateway:

•   **Resiliency and performance at any scale** Amazon API Gateway can manage any amount of traffic with throttling so that back-end operations can withstand traffic spikes. You don’t have to manage any infrastructure for API Gateway, and the infrastructure scales on its own depending on your needs.

•   **Caching** API Gateway provides the ability to cache the output of API calls to improve the performance of your API calls and reduce the latency since you don’t have to call the back end every time. As a result, it provides a great user experience.

•   **Security** API Gateway provides several tools to authorize access to your APIs and control service operation access. You can also use the AWS native tools such as AWS Identity and Access Management (IAM) and Amazon Cognito to authorize access to your APIs. API Gateway also has the capability to verify signed API calls. API Gateway leverages signature version 4 to authorize access to APIs.

•   **Metering** API Gateway helps you define plans that meter and restrict third-party developer access to your APIs. API Gateway automatically meters traffic to your APIs and lets you extract utilization data for each API key. (API keys are a great tool to manage the community of third-party developers interacting with the APIs.) API Gateway allows developers to create API keys through a console interface or through an API for programmatic creation. You can set permissions on API keys and allow access only to a set of APIs, or stages within an API. You also have the ability to configure throttling and quota limits on a per API key basis. Thus, API Gateway helps developers create, monitor, and manage API keys that they can distribute to third-party developers.

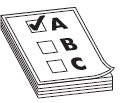
•   **Monitoring** Once you deploy an API, API Gateway provides you with a dashboard to view all the metrics and to monitor the calls to your services. It is also integrated with Amazon CloudWatch, and hence you can see all the statistics related to API calls, latency, error rates, and so on.

•   **Lifecycle management** API Gateway allows you to maintain and run several versions of the same API at the same time. It also has built-in stages. These enable developers to deploy multiple stages of each version such as the development stage, production stage, or beta stage.

•   **Integration with other AWS products** API Gateway can be integrated with AWS Lambda, which helps you to create completely serverless APIs. Similarly, by integrating with Amazon CloudFront, you can get protection against distributed denial-of-service (DDoS) attacks.

•   **Open API specification (Swagger) support** API Gateway supports open source Swagger. Using the AWS open source Swagger importer tool, you can import your Swagger API definitions into Amazon API Gateway. With the Swagger importer tool, you can create and deploy new APIs as well as update existing ones.

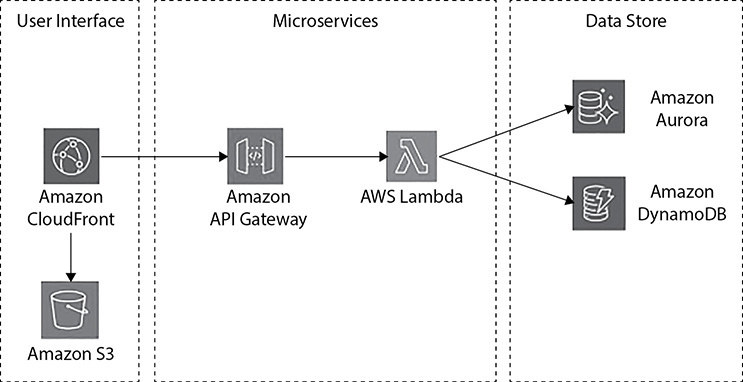
•   **SDK generation for iOS, Android, and JavaScript** API Gateway can automatically generate client SDKs based on your customer’s API definition. This allows developers to take their APIs from concept to integration test in a client app in a matter of hours.



**EXAM TIP**     Amazon API Gateway recently has been added to the associate examination; therefore, you should be able to identify the use cases of API Gateway and articulate the benefits of using API Gateway.

Lambda can be easily integrated with API Gateway. The combination of API Gateway and Lambda can be used to create fully serverless applications.

Figure 7-3 shows the serverless microservices architecture with AWS Lambda.



**Figure 7-3** Microservices architecture with AWS Lambda

AMAZON KINESIS

In the past few years, there has been a huge proliferation of data available to businesses. They are now receiving an enormous amount of continuous streams of data from a variety of sources. For example, the data might be coming from IoT devices, online gaming data, application server log files, application clickstream data, and so on. If you want to get insight from the data, you should be able to quickly process and analyze it. Having the ability to process and analyze becomes extremely important because that governs how you are going to serve your customers. For example, depending on a customer’s purchase patterns, you can customize the promotions, or you can provide personal recommendations based on the patterns of the customer.

Real-Time Application Scenarios

There are two types of use case scenarios for streaming data applications:

•   **Evolving from batch to streaming analytics** You can perform real-time analytics on data that has been traditionally analyzed using batch processing in data warehouses or using Hadoop frameworks. The most common use cases in this category include data lakes, data science, and machine learning. You can use streaming data solutions to continuously load real-time data into your data lakes. You can also update machine learning models more frequently as new data becomes available, ensuring the accuracy and reliability of the outputs. For example, Zillow uses Amazon Kinesis Data Streams to collect public record data and MLS listings and then provides home buyers and sellers with the most up-to-date home value estimates in near real time. Zillow also sends the same data to its Amazon Simple Storage Service (S3) data lake using Kinesis Data Streams so that all the applications work with the most recent information.

•   **Building real-time applications** You can use streaming data services for real-time applications such as application monitoring, fraud detection, and live leaderboards. These use cases require millisecond end-to-end latencies, from ingestion to processing and all the way to emitting the results to target data stores and other systems. For example, Netflix uses Kinesis Data Streams to monitor the communications between all its applications so it can detect and fix issues quickly, ensuring high service uptime and availability to its customers. While the most commonly applicable use case is application performance monitoring, more real-time applications in ad tech, gaming, and IoT are falling into this category.

Differences Between Batch and Stream Processing

You need a different set of tools to collect, prepare, and process real-time streaming data than the tools that you have traditionally used for batch analytics. With traditional analytics, you gather the data, load it periodically into a database, and analyze it hours, days, or weeks later. Analyzing real-time data requires a different approach. Instead of running database queries over stored data, stream-processing applications process data continuously in real time, even before it is stored. Streaming data can come in at a blistering pace, and data volumes can increase or decrease at any time. Stream data–processing platforms have to be able to handle the speed and variability of incoming data and process it as it arrives, meaning often millions to hundreds of millions of events per hour.

The Amazon Kinesis family provides you with solutions to manage huge quantities of data and gain meaningful insights from it. Amazon Kinesis consists of the following products:

•   Amazon Kinesis Data Streams

•   Amazon Kinesis Data Firehose

•   Amazon Kinesis Data Analytics

•   Amazon Kinesis Video Streams

AMAZON KINESIS DATA STEAMS

Amazon Kinesis Data Streams enables you to build custom applications that process or analyze streaming data for specialized needs. Kinesis Data Streams can continuously capture and store terabytes of data per hour from hundreds of thousands of sources such as web site clickstreams, financial transactions, social media feeds, IT logs, and location-tracking events. With the Kinesis Client Library (KCL), you can build Kinesis applications and use streaming data to power real-time dashboards, generate alerts, implement dynamic pricing and advertising, and more. You can also emit data from Kinesis Data Streams to other AWS services such as Amazon S3, Amazon Redshift, Amazon EMR, and AWS Lambda.

Benefits of Amazon Kinesis Data Streams

These are the benefits of Amazon Kinesis Data Streams:

•   **Real time** Kinesis Data Streams allows for real-time data processing. With Kinesis Data Streams, you can continuously collect data as it is generated and promptly react to critical information about your business and operations.

•   **Secure** You can privately access Kinesis Data Streams APIs from Amazon Virtual Private Cloud (VPC) by creating VPC endpoints. You can meet your regulatory and compliance needs by encrypting sensitive data within Kinesis Data Streams using server-side encryption and AWS Key Management Service (KMS) master keys.

•   **Easy to use** You can create a Kinesis stream within seconds. You can easily put data into your stream using the Kinesis Producer Library (KPL) and build Kinesis applications for data processing using the Kinesis Client Library. An Amazon Kinesis Data Streams producer is any application that puts user data records into a Kinesis data stream (also called *data ingestion*). The Kinesis Producer Library simplifies producer application development, allowing developers to achieve high-write throughput to a Kinesis stream.

•   **Parallel processing** Kinesis Data Streams allows you to have multiple Kinesis applications processing the same stream concurrently. For example, you can have one application running real-time analytics and another sending data to Amazon S3 from the same stream.

•   **Elastic** The throughput of a Kinesis data stream can scale from megabytes to terabytes per hour and from thousands to millions of PUT records per second. You can dynamically adjust the throughput of your stream at any time based on the volume of your input data.

•   **Low cost** Kinesis Data Streams has no up-front cost, and you pay for only the resources you use.

•   **Reliable** Kinesis Data Streams synchronously replicates your streaming data across three facilities in an AWS region and preserves your data for up to seven days, reducing the probability of data loss in the case of application failure, individual machine failure, or facility failure.

AMAZON KINESIS DATA FIREHOSE

Amazon Kinesis Data Firehose is the easiest way to load streaming data into data stores and analytics tools. It can capture, transform, and load streaming data into Amazon S3, Amazon Redshift, Amazon Elasticsearch, and Splunk, enabling near real-time analytics with the existing business intelligence tools and dashboards you’re already using today. It is a fully managed service that automatically scales to match the throughput of your data and requires no ongoing administration. It can also batch, compress, and encrypt the data before loading it, minimizing the amount of storage used at the destination and increasing security.

You can easily create a Firehose delivery stream from the AWS Management Console or AWS SDK, configure it with a few clicks, and start sending data to the stream from hundreds of thousands of data sources to be loaded continuously to AWS—all in just a few minutes. With Amazon Kinesis Data Firehose, you pay only for the amount of data you transmit through the service. There is no minimum fee or setup cost.

Amazon Kinesis Data Firehose manages all underlying infrastructure, storage, networking, and configuration needed to capture and load your data into Amazon S3, Amazon Redshift, Amazon Elasticsearch, or Splunk. You do not have to worry about provisioning, deployment, ongoing maintenance of the hardware or software, or writing any other application to manage this process. Firehose also scales elastically without requiring any intervention or associated developer overhead. Moreover, Amazon Kinesis Data Firehose synchronously replicates data across three facilities in an AWS region, providing high availability and durability for the data as it is transported to the destinations.

[Figure 7-4](https://learning.oreilly.com/library/view/aws-certified-solutions/9781260470192/ch7.xhtml#fig7_4) shows how Kinesis Data Firehose works.



**Figure 7-4** How Amazon Kinesis Data Firehose works

Benefits of Amazon Kinesis Data Firehose

These are the benefits of Amazon Kinesis Data Firehose:

•   **Easy to use** Amazon Kinesis Data Firehose provides a simple way to capture and load streaming data with just a few clicks in the AWS Management Console. You can simply create a Firehose delivery stream, select the destinations, and start sending real-time data from hundreds of thousands of data sources simultaneously. The service takes care of stream management, including all the scaling, sharding, and monitoring needed to continuously load the data to destinations at the intervals you specify.

•   **Integrated with AWS data stores** Amazon Kinesis Data Firehose is integrated with Amazon S3, Amazon Redshift, and Amazon Elasticsearch. From the AWS Management Console, you can point Kinesis Data Firehose to an Amazon S3 bucket, Amazon Redshift table, or Amazon Elasticsearch domain. You can then use your existing analytics applications and tools to analyze streaming data.

•   **Serverless data transformation** Amazon Kinesis Data Firehose enables you to prepare your streaming data before it is loaded to data stores. With Kinesis Data Firehose, you can easily convert raw streaming data from your data sources into formats required by your destination data stores, without having to build your own data-processing pipelines.

•   **Near real time** Amazon Kinesis Data Firehose captures and loads data in near real time. It loads new data into Amazon S3, Amazon Redshift, Amazon Elasticsearch, and Splunk within 60 seconds after the data is sent to the service. As a result, you can access new data sooner and react to business and operational events faster.

•   **No ongoing administration** Amazon Kinesis Data Firehose is a fully managed service that automatically provisions, manages, and scales compute, memory, and network resources required to load your streaming data. Once set up, Kinesis Data Firehose loads data continuously as it arrives.

•   **Pay only for what you use** With Amazon Kinesis Data Firehose, you pay only for the volume of data you transmit through the service. There are no minimum fees or up-front commitments.

AMAZON KINESIS DATA ANALYTICS

Amazon Kinesis Data Analytics is the easiest way to process and analyze real-time, streaming data. With Amazon Kinesis Data Analytics, you just use standard SQL to process your data streams, so you don’t have to learn any new programming language. Simply point Kinesis Data Analytics at an incoming data stream, write your SQL queries, and specify where you want to load the results. Kinesis Data Analytics takes care of running your SQL queries continuously on data while it’s in transit and then sends the results to the destinations.

Data is coming at us at lightning speeds because of the explosive growth of real-time data sources. Whether it is log data coming from mobile and web applications, purchase data from e-commerce sites, or sensor data from IoT devices, the massive amounts of data can help companies learn about what their customers and clients are doing. By getting visibility into this data as it arrives, you can monitor your business in real time and quickly leverage new business opportunities—such as making promotional offers to customers based on where they might be at a specific time or monitoring social sentiment and changing customer attitudes to identify and act on new opportunities.

To take advantage of these opportunities, you need a different set of analytics tools for collecting and analyzing real-time streaming data than what has been available traditionally for static, stored data. With traditional analytics, you gather the information, store it in a database, and analyze it hours, days, or weeks later. Analyzing real-time data requires a different approach and different tools and services. Instead of running database queries on stored data, streaming analytics platforms process the data continuously before the data is stored in a database. Streaming data flows at an incredible rate that can vary up and down all the time. Streaming analytics platforms have to be able to process this data when it arrives, often at speeds of millions of events per hour.

Benefits of Amazon Kinesis Data Analytics

These are the benefits of Amazon Kinesis Data Analytics:

•   **Powerful real-time processing** Amazon Kinesis Data Analytics processes streaming data with subsecond processing latencies, enabling you to analyze and respond in real time. It provides built-in functions that are optimized for stream processing, such as anomaly detection and top-K analysis, so that you can easily perform advanced analytics.

•   **Fully managed** Amazon Kinesis Data Analytics is a fully managed service that runs your streaming applications without requiring you to provision or manage any infrastructure.

•   **Automatic elasticity** Amazon Kinesis Data Analytics automatically scales up and down the infrastructure required to run your streaming applications with low latency.

•   **Easy to use** Amazon Kinesis Data Analytics provides interactive tools including a schema editor, a SQL editor, and SQL templates to make it easy to build and test your queries for both structured and unstructured input data streams.

•   **Standard SQL** Amazon Kinesis Data Analytics supports standard SQL. There is no need to learn complex processing frameworks and programming languages.

•   **Pay only for what you use** With Amazon Kinesis Data Analytics, you pay only for the processing resources your streaming application uses. As the volume of input data changes, Amazon Kinesis Data Analytics automatically scales resources up and down and charges you only for the resources actually used for processing. There are no minimum fees or up-front commitments.

Use Cases for Amazon Kinesis Data Analytics

You can use Amazon Kinesis Data Analytics in pretty much any use case where you are collecting data continuously in real time and want to get information and insights in seconds or minutes rather than having to wait days or even weeks. In particular, Kinesis Data Analytics enables you to quickly build applications that process streams from end to end for log analytics, clickstream analytics, Internet of Things (IoT), ad tech, gaming, and more. The three most common usage patterns are time-series analytics, real-time dashboards, and real-time alerts and notifications.

GENERATE TIME-SERIES ANALYTICS

Time-series analytics enables you to monitor and understand how your data is trending over time. With Amazon Kinesis Data Analytics, you can author SQL code that continuously generates these time-series analytics over specific time windows. For example, you can build a live leaderboard for a mobile game by computing the top players every minute and then sending it to Amazon S3. Or, you can track the traffic to your web site by calculating the number of unique site visitors every five minutes and then send the processed results to Amazon Redshift.

FEED REAL-TIME DASHBOARDS

You can build applications that compute query results and emit them to a live dashboard, enabling you to visualize the data in near real time. For example, an application can continuously calculate business metrics such as the number of purchases from an e-commerce site, grouped by the product category, and then send the results to Amazon Redshift for visualization with a business intelligence tool of your choice. Consider another example where an application processes log data, calculates the number of application errors, and then sends the results to the Amazon Elasticsearch Service for visualization with Kibana.

CREATE REAL-TIME ALARMS AND NOTIFICATIONS

You can build applications that send real-time alarms or notifications when certain metrics reach predefined thresholds or, in more advanced cases, when your application detects anomalies using the machine learning algorithm you provide. For example, an application can compute the availability or success rate of a customer-facing API over time and then send the results to Amazon CloudWatch. You can build another application to look for events that meet certain criteria and then automatically notify the right customers using Kinesis Data Streams and Amazon Simple Notification Service (SNS).

AMAZON KINESIS VIDEO STREAMS

Kinesis Video Streams is the latest addition to the Kinesis family. Kinesis Video Streams is not covered in the SAA examination, so we won’t cover it in detail here. However, it is part of Kinesis family, and by using this product you can build your applications, so it is good to know what this product is all about and what it does.

It’s a fully managed service which can be used for ingesting, storing, and processing media files. Using this service you can process any volume of the media files. Since it is a managed service, you don’t have to provision any infrastructure. This service automatically scales the infrastructure for you depending on the requirement. The data is stored across multiple AZs to provide durability, and you can stream your media through easy-to-use APIs. Using this service, you can do the following:

•   Stream video from millions of devices

•   Build real-time vision and video-enabled apps

•   Play back live and recorded video streams

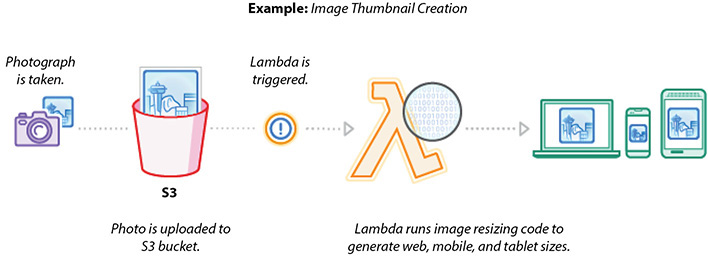
•   Build apps with two-way, real-time media streaming

REFERENCE ARCHITECTURES USING SERVERLESS SERVICES

In this section, you will explore the reference architecture when using AWS Lambda, Amazon API Gateway, and Amazon Kinesis. This will help you to understand the practical implementation aspects of using serverless architecture.

Real-Time File Processing

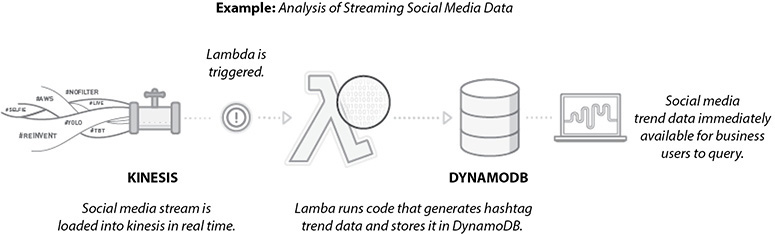
You can use Amazon S3 to trigger AWS Lambda to process data immediately after an upload. For example, you can use Lambda to create thumbnail images, transcode videos, index files, process logs, validate content, and aggregate and filter data in real time. [Figure 7-5](https://learning.oreilly.com/library/view/aws-certified-solutions/9781260470192/ch7.xhtml#fig7_5) shows the reference architecture for real-time file processing.



**Figure 7-5** Reference architecture for real-time file processing

Real-Time Stream Processing

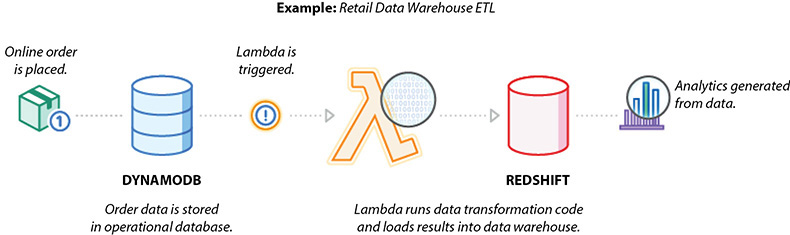
You can use AWS Lambda and Amazon Kinesis to process real-time streaming data for application activity tracking, transaction order processing, clickstream analysis, data cleansing, metrics generation, log filtering, indexing, social media analysis, and IoT device data telemetry and metering. [Figure 7-6](https://learning.oreilly.com/library/view/aws-certified-solutions/9781260470192/ch7.xhtml#fig7_6) shows the reference architecture for real-time stream processing.



**Figure 7-6** Reference architecture for real-time stream processing

Extract, Transformation, and Load (ETL) Processing

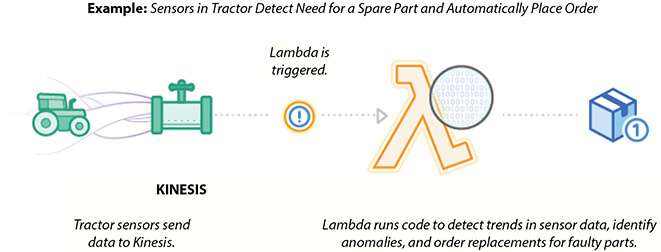
You can use AWS Lambda to perform data validation, filtering, sorting, or other transformations for every data change in a DynamoDB table and load the transformed data into another data store. [Figure 7-7](https://learning.oreilly.com/library/view/aws-certified-solutions/9781260470192/ch7.xhtml#fig7_7) shows the reference architecture for a data warehouse ETL.



**Figure 7-7** Reference architecture for data warehouse ETL

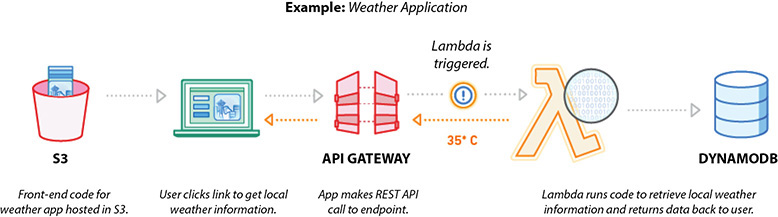
IoT Back Ends

This example leverages a serverless architecture for back ends using AWS Lambda to handle web, mobile, Internet of Things (IoT), and third-party API requests. [Figure 7-8](https://learning.oreilly.com/library/view/aws-certified-solutions/9781260470192/ch7.xhtml#fig7_8) shows the IoT back end.



**Figure 7-8** IoT back end

[Figure 7-9](https://learning.oreilly.com/library/view/aws-certified-solutions/9781260470192/ch7.xhtml#fig7_9) shows the reference architecture for a weather application with API Gateway and AWS Lambda.



**Figure 7-9** Reference architecture for a weather application using Amazon API Gateway and AWS Lambda

AMAZON CLOUDFRONT

Amazon CloudFront is a global content delivery network (CDN) service that allows you to distribute content with low latency and provides high data transfer speeds. Amazon CloudFront employs a global network of edge locations and regional edge caches that cache copies of your content close to your viewers. In addition to caching static content, Amazon CloudFront accelerates dynamic content. Amazon CloudFront ensures that end-user requests are served by the closest edge location. It routes viewers to the best location. As a result, viewer requests travel a short distance, improving performance for your viewers. As of the writing this book, Amazon CloudFront has 217 points of presence (205 edge locations and 12 regional edge caches) in 84 cities across 42 countries. When using Amazon CloudFront, there are no minimum usage commitments; you pay only for the data transfers and requests you actually use. Also, there is no data transfer charges for data transferred between AWS regions and CloudFront edge locations.

These are some of the use cases for Amazon CloudFront:

•   **Caching static assets** This is the most common use case for Amazon CloudFront. It helps in speeding up the delivery of your static content such as photos, videos, style sheets, and JavaScript content across the globe. The data is served to end users via edge locations.

•   **Accelerating dynamic content** Amazon CloudFront has a lot of network optimizations that accelerate the dynamic content. You can integrate CloudFront with your application or web site running on EC2 servers.

•   **Helping protect against distributed denial-of-service (DDoS) attacks** Amazon CloudFront can be integrated with AWS Shield and WAF, which can protect layers 3 and 4 and layer 7, respectively, against DDoS attacks. CloudFront negotiates TLS connections with the highest security ciphers and authenticates viewers with signed URLs.

•   **Improving security** Amazon CloudFront can serve the content securely with SSL (HTTPS). You can deliver your secure APIs or applications using SSL/TLS, and advanced SSL features are enabled automatically. CloudFront’s infrastructure and processes are all compliant with PCI, DSS, HIPAA, and ISO to ensure the secure delivery of your most sensitive data.

•   **Accelerating API calls** Amazon CloudFront is integrated with Amazon API Gateway and can be used to secure and accelerate your API calls. CloudFront supports proxy methods such as POST, PUT, OPTIONS, DELETE, and PATCH.

•   **Distributing software** Amazon CloudFront is used for software distribution. By using Amazon CloudFront for the distribution of your software, you can provide a faster user experience since it is going to result in faster downloads. Since Amazon CloudFront scales automatically, you don’t have to bother about how much content it can serve. You can make your software available right at the edge where your users are.

•   **Streaming videos** Amazon CloudFront can be used for video streaming both live and on demand. It is capable of streaming 4K video.

Amazon CloudFront Key Concepts

In this section, you will learn some Amazon CloudFront key terminology:

•   **Edge location** CloudFront delivers your content through a worldwide network of data centers called *edge locations*. These data centers are located in major cities across the globe. It is likely that an AWS region may not exist at a particular place where an edge location is present.

•   **Regional edge location** The regional edge caches are located between your origin web server and the global edge locations that serve content directly to your viewers. As objects become less popular, individual edge locations may remove those objects to make room for more popular content. Regional edge caches have a larger cache width than any individual edge location, so objects remain in the cache longer at the nearest regional edge caches. This helps keep more of your content closer to your viewers, reducing the need for CloudFront to go back to your origin web server and improving the overall performance for viewers. This regional edge cache feature is enabled by default, and you don’t have to do anything manually to use this feature; it is not charged separately.



**NOTE**    An origin web server is often referred to as an *origin*, which is the location where your actual noncached data resides.

•   **Distribution** A distribution specifies the location or locations of the original version of your files. A distribution has a unique CloudFront.net domain name (such as abc123.cloudfront.net) that you can use to reference your objects through the global network of edge locations. If you want, you can map your own domain name (for example, [www.example.com](https://www.example.com/)) to your distribution. You can create distributions to either download your content using the HTTP or HTTPS protocol or stream your content using the RTMP protocol.

•   **Origin** CloudFront can accept any publicly addressable Amazon S3 or HTTP server, an ELB/ALB, or a custom origin server outside of AWS as an origin. When you create an origin, you must provide the public DNS name of the origin. For example, if you specify an EC2 server, it should be something like ec2-52-91-188-59.compute-1.amazonaws.com.

•   **Behaviors** Behaviors allow you to have granular control of the CloudFront CDN, enforce certain policies, change results based on request type, control the cacheability of objects, and more. You can unleash the whole power of Amazon CloudFront using behaviors. The following sections discuss the important behaviors that can be configured with Amazon CloudFront.

**Path Pattern Matching** You can configure multiple cache behaviors based on URL path patterns for the web site or application for which you are going to use Amazon CloudFront. The pattern specifies which requests to apply the behavior to. When CloudFront receives a viewer request, the requested path is compared with path patterns in the order in which cache behaviors are listed in the distribution, as in images/\*.jpg and /images/\*. The CloudFront behavior is the same with or without the leading /. Based on the path pattern, you can route requests to specific origins, set the HTTP/HTTPS protocol, set the header or caching options, set cookie and query string forwarding, restrict access, and set compression.

**Headers** Using headers you can forward request headers to the origin cache based on the header values. You can detect the device and take actions accordingly. For example, you can have a different response if the user is coming from a laptop or mobile device. Similarly, you can have a different response based on the language; for example, a user can prefer Spanish but will accept British. You can also have a different response based on the protocol. For example, you can forward the request to different content based on the connection type.

**Query Strings/Cookies** Some web applications use query strings to send information to the origin. A query string is the part of a web request that appears after a ? character; the string can contain one or more parameters separated by & characters. For example, the following query string includes two parameters, color=blue and size=small:

http://abc111xyz.cloudfront.net/images/image.jpg?*color=blue&size=small*

Now let’s say your web site is available in three languages. The directory structure and file names for all three versions of the web site are identical. As a user views your web site, requests that are forwarded to CloudFront include a language query string parameter based on the language that the user chose. You can configure CloudFront to forward query strings to the origin and to cache based on the language parameter. If you configure your web server to return the version of a given page that corresponds with the selected language, CloudFront will cache each language version separately, based on the value of the language query string parameter.

In this example, if the main page for your web site is main.html, the following three requests will cause CloudFront to cache main.html three times, once for each value of the language query string parameter:

http://abc111xyz.cloudfront.net/main.html?*language=en*

http://abc111xyz.cloudfront.net/main.html?*language=es*

http://abc111xyz.cloudfront.net/main.html?language=fr

**Signed URL or Signed Cookies** If you move your static content to an S3 bucket, you can protect it from unauthorized access via CloudFront signed URLs. A signed URL includes additional information, for example, an expiration date and time, that gives you more control over access to your content. This is how the signed URL works. The web server obtains temporary credentials to the S3 content. It creates a signed URL based on those credentials that allow access. It provides this link in content returned (a signed URL) to the client, and this link is valid for a limited period of time. This additional information appears in a policy statement, which is based on either a canned policy or a custom policy. Via signed URLs, you can get additional control such as restricting access to content, getting subscriptions for your content, creating digital rights, creating custom policies, and so on.

Signed HTTP cookies provide the same degree of control as a signed URL by including the signature in an HTTP cookie instead. This allows you to restrict access to multiple objects (e.g., whole-site authentication) or to a single object without needing to change URLs. This is how it works. A Set-Cookie header is sent to the user after they are authenticated on a web site. That sets a cookie on the user’s device. When a user requests a restricted object, the browser forwards the signed cookie in the request. CloudFront then checks the cookie attributes to determine whether to allow or restrict access.

**Protocol Policy** If you want CloudFront to allow viewers to access your web content using either HTTP or HTTPS, specify HTTP and HTTPS. If you want CloudFront to redirect all HTTP requests to HTTPS, specify Redirect HTTP to HTTPS. If you want CloudFront to require HTTPS, specify HTTPS Only.

**Time to Live (TTL)** You can control how long your objects stay in a CloudFront cache before CloudFront forwards another request to your origin. Reducing the duration allows you to serve dynamic content. Increasing the duration means your users get better performance because your objects are more likely to be served directly from the edge cache. A longer duration also reduces the load on your origin. You can set up minimum, maximum, and default TTL for all the objects. The time is specified in seconds. By default, each object automatically expires after 24 hours. You can also control the cache duration for an individual object, and you can configure your origin to add a Cache-Control max-age or Cache-Control s-maxage directive or an expires header field to the object.

**Gzip Compression** Gzip compression can be enabled on distributions; your pages can load more quickly because content will download faster, and your CloudFront data transfer charges may be reduced as well. You can configure Amazon CloudFront to automatically apply gzip compression when browsers and other clients request a compressed object with text and other compressible file formats. This means if you are already using Amazon S3, CloudFront can transparently compress this type of content. For origins outside S3, doing compression at the edge means you don’t need to use resources at your origin to do compression. The resulting smaller size of compressed objects makes downloads faster and reduces your CloudFront data transfer charges.

You can create two types of distributions via CloudFront: web and RTMP. Web distribution is used for speeding up the distribution of static and dynamic content, for example, .html, .css, .php, and graphics files. RTMP distribution is used to speed up the distribution of your streaming media files using Adobe Flash Media Server’s RTMP protocol. An RTMP distribution allows an end user to begin playing a media file before the file has finished downloading from a CloudFront edge location. Most of the behaviors mentioned earlier are applicable for web distribution, and some of them may not be applicable for RTMP distribution.

Geo Restriction

When a user requests your content, CloudFront typically serves the requested content regardless of where the user is located. If you need to prevent users in specific countries from accessing your content, you can use the CloudFront geo restriction feature to do one of the following:

•   Allow your users to access your content only if they’re in one of the countries on a whitelist of approved countries

•   Prevent your users from accessing your content if they’re in one of the countries on a blacklist of banned countries

Error Handling

You can configure CloudFront to respond to requests using a custom error page when your origin returns an HTTP 4xx or 5xx status code. For example, when your custom origin is unavailable and returning 5xx responses, CloudFront can return a static error page that is hosted on Amazon S3. You can also specify a minimum TTL to control how long CloudFront caches errors.

AMAZON ROUTE 53

Amazon Route 53 is the managed Domain Name Service (DNS) of Amazon. DNS translates human-readable names such as [www.example.com](https://www.example.com/) into the numeric IP addresses such as 192.0.0.3 that servers/computers use to connect to each other. You can think of DNS as a phone book that has addresses and telephone numbers. Route 53 connects user requests to infrastructure running in AWS, such as Amazon EC2 instances, Elastic Load Balancing load balancers, or Amazon S3 buckets, and it can also be used to route users to infrastructure outside of AWS.

It is highly available and scalable. This is the only service that has a 100 percent SLA. This service is region independent, which means you can configure Route 53 with resources running across multiple regions. It is capable of doing DNS resolution within multiple regions and among AWS VPCs.

In addition to managing your public DNS record, Route 53 can be used to register a domain, create DNS records for a new domain, or transfer DNS records for an existing domain.

Amazon Route 53 currently supports the following DNS record types:

•   A (address record)

•   AAAA (IPv6 address record)

•   CNAME (canonical name record)

•   CAA (certification authority authorization)

•   MX (mail exchange record)

•   NAPTR (name authority pointer record)

•   NS (name server record)

•   PTR (pointer record)

•   SOA (start of authority record)

•   SPF (sender policy framework)

•   SRV (service locator)

•   TXT (text record)

In addition, Route 53 supports alias records (also known as *zone apex support*). The zone apex is the root domain of a web site ([example.com](https://example.com/), without the www). You use CloudFront to deliver content from the root domain, or zone apex, of your web site. In other words, you configure both [http://www.example.com](http://www.example.com/) and [http://example.com](http://example.com/) to point at the same CloudFront distribution. Since the DNS specification requires a zone apex to point to an IP address (an A record), not a CNAME (such as the name AWS provides for a CloudFront distribution, ELB, or S3 web site bucket), you can use Route 53’s alias record to solve this problem.

Route 53 also offers health checks, which allow you to monitor the health and performance of your application, web servers, and other resources that leverage this service. Health checks of your resources with Route 53 are useful when you have two or more resources that are performing the same function. For example, you might have multiple Amazon EC2 servers running HTTP server software responding to requests for the example.com web site. Say you have multiple EC2 servers running across two regions. As long as all the resources are healthy, Amazon Route 53 responds to queries using all of your example.com resource sets (using all the EC2 servers). When a resource becomes unhealthy, Amazon Route 53 responds to queries using only the healthy resource record sets for example.com, which means if a few EC2 servers go down, Route 53 won’t use them, or if an AZ goes down, Route 53 won’t use the EC2 instances from the AZ that went down for the resources. It is going to leverage only the healthy EC2 servers from healthy AZs in a region.

Using Route 53, you can also do Traffic Flow by running multiple endpoints around the world. Using Amazon Route 53 Traffic Flow, your users can connect to the best endpoint based on latency, geography, and endpoint health. Using this feature, you can improve the performance and availability of your application and thereby provide the best user experience.

Amazon Route 53 supports the following routing policies:

•   **Weighted round robin** When you have multiple resources that perform the same function (for example, web servers that serve the same web site) and you want Amazon Route 53 to route traffic to those resources in proportions that you specify (for example, one-quarter to one server and three-quarters to the other), you can do this using weighted round robin. You can also use this capability to do A/B testing, sending a small portion of traffic to a server on which you’ve made a software change (say 10 percent of the traffic going to the newly changed server and 90 percent of the traffic going to the old server).

•   **Latency-based routing** When you have resources in multiple Amazon EC2 data centers that perform the same function and you want Amazon Route 53 to respond to DNS queries with the resources that provide the best latency, you can use latency-based routing. It helps you improve your application’s performance for a global audience. You can run applications in multiple AWS regions, and Amazon Route 53, using dozens of edge locations worldwide, will route end users to the AWS region that provides the lowest latency.

•   **Failover routing** When you want to configure active-passive failover, in which one resource takes all traffic when it’s available and the other resource takes all traffic when the first resource isn’t available, you can use failover routing. For example, you may have all your resources running from a particular region. When this region fails, you can do failover routing and point to a static web site running from a different region.

•   **Geo DNS routing** When you want Amazon Route 53 to respond to DNS queries based on the location of your users, you can use this routing. Route 53 Geo DNS lets you balance the load by directing requests to specific endpoints based on the geographic location from which the request originates. Geo DNS makes it possible to customize localized content, such as presenting detail pages in the right language or restricting the distribution of content to only the markets you have licensed.

AWS WEB APPLICATION FIREWALL

AWS Web Application Firewall (WAF) is a web application firewall that protects your web applications from various forms of attack. It helps to protect web sites and applications against attacks that could affect application availability, result in data breaches, cause downtime, compromise security, or consume excessive resources. It gives you control over which traffic to allow or block to/from your web applications by defining customizable web security rules. The following are some of the use cases of AWS WAF:

•   **Vulnerability protection** You can use AWS WAF to create custom rules that block common attack patterns, such as SQL injection or cross-site scripting (XSS), and rules that are designed for your specific application.

•   **Malicious requests** Web crawlers can be used to mount attacks on a web site. By using an army of automated crawlers, a malicious actor can overload a web server and bring a site down. AWS WAF can protect against those malicious requests. It can also protect from scrapers where someone tries to extract large amounts of data from web sites.

•   **DDoS mitigation (HTTP/HTTPS floods)** This helps protect web applications from attacks by allowing you to configure rules that allow, block, or monitor (count) web requests based on conditions that you define. These conditions include IP addresses, HTTP headers, HTTP body, URI strings, SQL injection, and cross-site scripting.

WAF is integrated with CloudFront. As a result, you can bring the added distribution capacity and scalability of a CDN to WAF. It helps to decrease the load of origin by blocking attacks close to the source, helps in distributing sudden spikes of traffic leveraging CDNs, and avoids single points of failures with increased redundancy of CDNs. WAF can be integrated with application load balancers (ALBs) as well, which can protect your origin web servers running behind the ALBs.

To use WAF with CloudFront or ALB, you need to identify the resource that can be either an Amazon CloudFront distribution or an application load balancer that you need to protect. You then deploy the rules and filters that will best protect your applications. Rules are collections of WAF filter conditions; it either can be one condition or can be a combination of two or more conditions. Let’s understand this in detail.

Conditions define the basic characteristics that you want AWS WAF to watch for in web requests, and these conditions specify when you want to allow or block requests. For example, you may want to watch the script that looks malicious. If WAF is able to find it, it is going to block it. In this case, you create a condition that watches for the request. Let’s take a look at all the conditions you can create using AWA WAF.

•   Using cross-site scripting match conditions, you can allow or block the requests that appear to contain malicious scripts.

•   Using IP match conditions, you can allow or block requests based on the IP addresses that they originate from.

•   Using geographic match conditions, you can allow or block requests based on the country that they originate from.

•   Using size constraint conditions, you can allow or block requests based on whether the requests exceed a specified length.

•   Using SQL injection match conditions, you can allow or block requests based on whether the requests appear to contain malicious SQL code.

•   Using string match conditions, you can allow or block requests based on strings that appear in the requests.

•   Using regex matches, you can allow or block requests based on a regular expression pattern that appears in the requests.

Once you create the condition, you can combine these conditions into rules to precisely target the requests that you want to allow, block, or count.

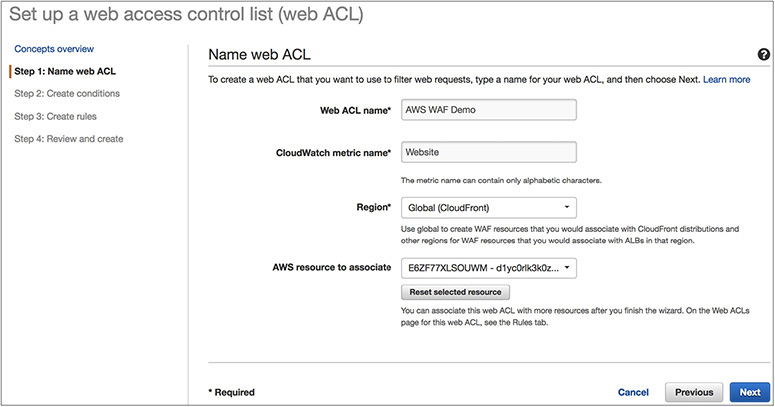
There are two types of rules in AWS WAF: regular rules and rate-based rules. Regular rules use only conditions to target specific requests. For example, you can create a regular rule based on the following conditions: requests coming from 19.152.0.55 and requests that include SQL-like code. In this case, with a regular rule, you have included two conditions. When a rule includes multiple conditions, as in this example, AWS WAF looks for requests that match all conditions—that is, it ANDs the conditions together. Rate-based rules are similar to regular rules, with one addition: a rate limit in five-minute intervals. Say you specify the rate limit as 2,000; then the rate-based rules count the requests that arrive from a specified IP address every five minutes. The rule can trigger an action (block all IPs) that have more than 2,000 requests in the last five minutes.

You can combine conditions with the rate limit. In this case, if the requests match all of the conditions and the number of requests exceeds the rate limit in any five-minute period, the rule will trigger the action designated in the web ACL.

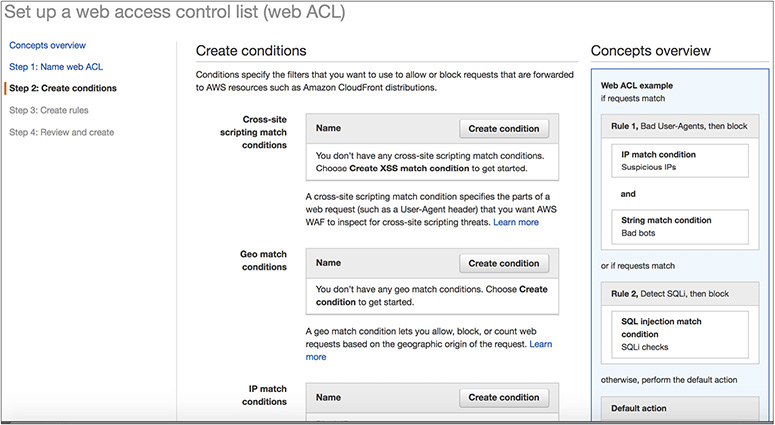
Creating a web access control list (web ACL) is the first thing you need to do to use AWS WAF. Once you combine your conditions into rules, you combine the rules into a web ACL. This is where you define an action for each rule. The action can be set to allow, block, or count. Now when a web request matches all the conditions in a rule, AWS WAF can either block the request or allow the request to be forwarded to Amazon CloudFront or an application load balancer.

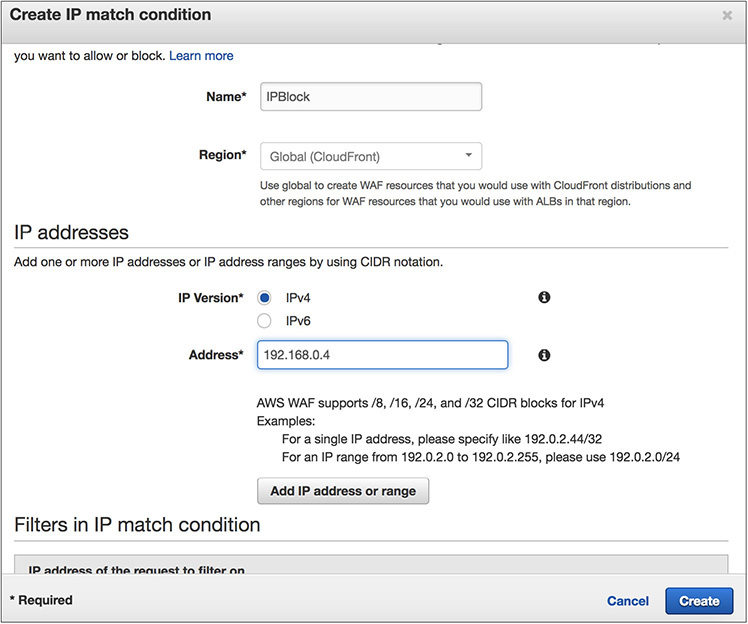
Now that you understand the concept, let’s look at the step-by-step process to configure a WAF from the console:

**1.**   Name the web ACL.

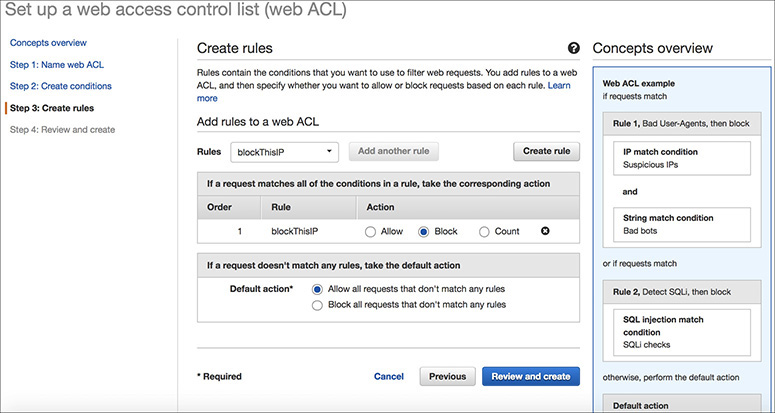


**2.**   Create the conditions.

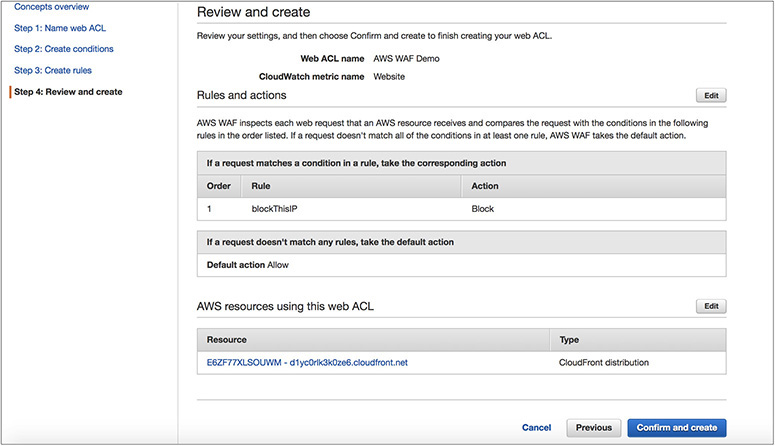




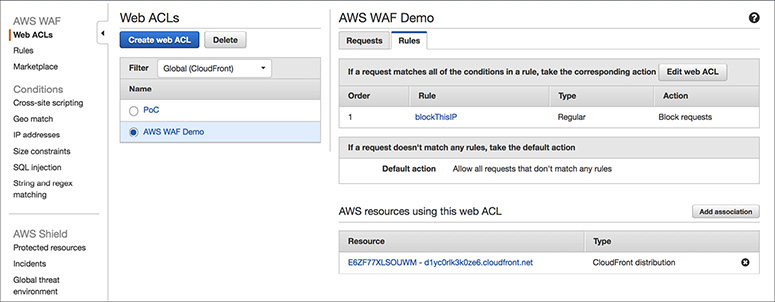
**3.**   Create the rules.



**4.**   Review the rules.



**5.**   Confirm the rules.



AWS WAF resources can be managed with APIs. Therefore, you can do all kinds of actions using APIs such as adding IPs to a list. In addition to APIs, just like any other service, you can configure everything via the AWS console. The previous example showed how to configure AWS WAF via the AWS Management Console. AWS WAF configurations are propagated globally in one minute.

You can watch the real-time metrics using Amazon CloudWatch. One-minute metrics are available in CloudWatch. You can see how many requests were blocked, allowed, and counted, or you can apply your rules for analysis. You can also monitor all the changes made via APIs using CloudWatch.

AMAZON SHIELD

AWS Shield is a managed service that provides protection against distributed DDoS attacks for applications running on AWS. A DoS attack is an intentional attack on your application or website with the intention of making it unavailable to users. The attackers target your application in various ways such as flooding it with network traffic, simulating a huge workload on your application, and so on. There are two types of AWS Shield: AWS Standard and AWS Shield Advanced.

Benefits of AWS Shield

AWS Shield Standard is automatically enabled for all AWS customers at no additional cost. AWS Shield Advanced is an optional paid service. Let’s compare the benefits of both.

AWS SHIELD STANDARD

AWS Shield Standard protects against DDoS attacks occurring at the network and transport layer. There are no charges for using AWS Shield Standard, and this service is available at all AWS regions. AWS Shield Standard monitors and baselines all the incoming traffic and then identifies if there are any anomalies and then creates the mitigations automatically, thereby protecting your applications.

AWS SHIELD ADVANCED

Since AWS Shield Advanced is a paid service, it comes with lots of additional benefits when compared with AWS Shield Standard. The benefits of AWS Shield Advanced are as follows:

•   Access to the AWS Global DDoS response team for any assistance required in mitigating against DDoS attacks

•   Getting more visibility around the DDoS attack

•   Access to the Global Threat Environment dashboard, which provides a comprehensive view of the attacks

•   AWS Shield Advanced users can also access AWS Web Application Firewall (WAF) and AWS Firewall Manager free of cost

•   The service level is enhanced for all AWS Shield Advanced customers

AMAZON SIMPLE QUEUE SERVICE

One of the challenges an architect faces when building new applications for the cloud or migrating existing applications is making them distributed. You need to address scalability, fault tolerance, and high availability, and you need to start thinking more deeply about things such as the CAP theorem, eventual consistency, distributed transactions, and design patterns that support distributed systems.

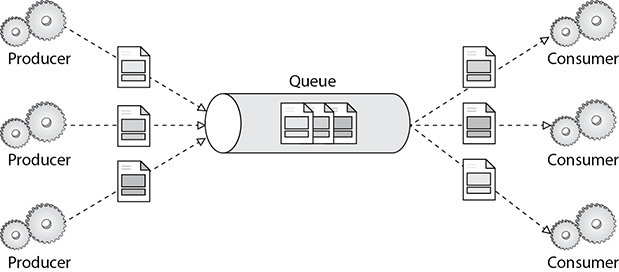


**NOTE**    The CAP theorem states that in a distributed data store it is impossible to provide more than two guarantees out of these three: consistency, availability, partition tolerance. This means either you can have consistency and availability or consistency and partition tolerance or availability and partition tolerance but not all three together.

Messaging can really help you in this case to achieve the goals. A *message queue* is a form of asynchronous service-to-service communication used in serverless and microservice architectures. Messages are stored on the queue until they are processed and deleted. Each message is processed only once by a single consumer. Message queues can be used to decouple the processing of larger jobs into small parts that can be run independently of each other. This can help in terms of performance, making the batch job run faster, and can help during busy workloads.

When you are designing an architecture for the cloud, it is recommended to decouple the applications to smaller, independent building blocks that are easier to develop, deploy, and maintain. Message queues provide communication and coordination for these distributed applications. It can also simplify the coding of decoupled applications, at the same time improving performance, reliability, and scalability.

Message queues allow different parts of a system to communicate and process operations asynchronously. A message queue provides a *buffer,* which temporarily stores messages, and *endpoints,* which allow software components to connect to the queue to send and receive messages. You can put messages into a queue, and you can retrieve messages from a queue. The messages are usually small and can be things such as requests, replies, error messages, or just plain information. The software that puts messages into a queue is called a *message producer,* and the software that retrieves messages is called a *message consumer*. For sending a message, the producer adds a message to the queue. The message is stored on the queue until the receiver of the message (the consumer) retrieves the message and does something with it. [Figure 7-10](https://learning.oreilly.com/library/view/aws-certified-solutions/9781260470192/ch7.xhtml#fig7_10) shows the producer, queue, and consumer.



**Figure 7-10** Producer, queue, and consumer

Amazon Simple Queue Service (Amazon SQS) is a fast, reliable, scalable, and fully managed queue service. Using Amazon SQS, you can quickly build message queuing applications that can run on any system. It can send, store, and receive messages between components. Like most AWS services, it’s accessible through a web API, as well as SDKs in most languages.

These are some of the key features of Amazon SQS:

•   SQS is redundant across multiple AZs in each region. Even if an AZ is lost, the service will be accessible.

•   Multiple copies of messages are stored across multiple AZs, and messages are retained up to 14 days.

•   If your consumer or producer application fails, your messages won’t be lost.

•   Because of the distributed architecture, SQS scales without any preprovisioning. It scales up automatically as and when more traffic comes. Similarly, when the traffic is low, it automatically scales down.

•   The messages can contain up to 256KB of text data, including XML, JSON, and unformatted text.

There are two types of SQS queues: standard and FIFO.

•   **Standard** This is the default queue type of Amazon SQS. It supports almost unlimited transactions per second. It supports at-least-once message delivery. It provides best-effort ordering that ensures that messages are generally delivered in the same order as they’re sent and at nearly unlimited scale. Although a standard queue tries to preserve the order of messages, it could be possible that sometimes a message is delivered out of order. If your system needs order to be preserved, then instead of choosing standard, you should choose FIFO.

•   **FIFO** This is the second type of queue. A first in, first out (FIFO) queue guarantees first in, first out delivery and also exactly once processing, ensuring that your consumer application does not need to consider the message being delivered multiple times. In FIFO queues, the throughput is limited to 300 transactions per second, and FIFO queues support up to 3,000 messages per second

These are the differences between standard queues and FIFO queues:

•   Standard queues support a nearly unlimited number of transactions per second (TPS) per API action, whereas FIFO queues support up to 300 messages per second (300 send, receive, or delete operations per second). You can also batch 10 messages per operation (maximum). FIFO queues can support up to 3,000 messages per second.

•   In standard queues, a message is delivered at least once, but occasionally more than one copy of a message is delivered, whereas in FIFO a message is delivered once and remains available until a consumer processes and deletes it. Duplicates aren’t introduced into the queue.

•   In standard queues, occasionally messages might be delivered in an order different from which they were sent, whereas in FIFO the order in which messages are sent and received is strictly preserved (i.e., first in, first out).

Let’s understand some of the terminology and parameters that you need to know for configuring SQS.

When a producer sends a message to the queue, it is immediately distributed to multiple SQS servers across multiple AZs for redundancy. Whenever a consumer is ready to process the message, it processes the message from the queue. When the message is being processed, it stays in the queue and isn’t returned to subsequent receive requests for the duration of *visibility timeout.* When the visibility timeout expires, the consumer deletes the message from the queue to prevent the message from being received and processed again. Thus, visibility timeout is the length of time (in seconds) that a message received from a queue will be invisible to other receiving components. The value must be between 0 seconds and 12 hours.

•   **Message retention period** This is the amount of time that Amazon SQS will retain a message if it does not get deleted. The value must be between 1 minute and 14 days.

•   **Maximum message size** This is the maximum message size (in bytes) accepted by Amazon SQS. It can be between 1KB and 256KB.

•   **Delivery delay** This is the amount of time to delay or postpone the delivery of all messages added to the queue. It can be anywhere from 0 seconds to 15 minutes. If you create a delay queue, any messages that you send to the queue remain invisible to consumers for the duration of the delay period. For standard queues, the per-queue delay setting is *not retroactive*—changing the setting doesn’t affect the delay of messages already in the queue. For FIFO queues, the per-queue delay setting is *retroactive*—changing the setting affects the delay of messages already in the queue.

•   **Receive message wait time** Using this parameter, you can specify short polling or long polling. Short polling returns immediately, even if the message queue being polled is empty. When you set Receive Message Wait Time to 0 seconds, short polling is enabled. *Long polling* helps reduce the cost of using Amazon SQS by eliminating the number of empty responses (when there are no messages available for a ReceiveMessage request) and false empty responses (when messages are available but aren’t included in a response) and returning messages as soon as they become available. When you specify the parameter between 1 and 20 seconds, long polling is enabled.

•   **Content-based deduplication** This parameter is applicable only for the FIFO queue. Using this parameter you use an SHA-256 hash of the body of the message (but not the attributes of the message) to generate the content-based message deduplication ID.

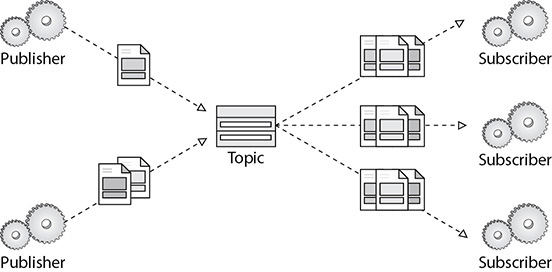
Amazon SQS supports *dead-letter queues,* which other queues (*source queues*) can target for messages that can’t be processed (consumed) successfully. Sometimes messages can’t be processed because of a variety of possible issues, such as erroneous conditions within the producer or consumer application or an unexpected state change that causes an issue with your application code. Dead-letter queues are useful for debugging your application or messaging system because they let you isolate problematic messages to determine why their processing doesn’t succeed. By checking the parameter Use Redrive Policy, you can send messages into a dead-letter queue after exceeding the Maximum Receives setting. Using the parameter Maximum Receives, you can specify the maximum number of times a message can be received before it is sent to the dead-letter queue. The value of Maximum Receives can be between 1 and 1000. You can specify a queue name by adding one for the parameter Dead Letter Queue.

Using server-side encryption (SSE), you can transmit sensitive data in encrypted queues. SSE protects the contents of messages in Amazon SQS queues using keys managed in the AWS Key Management Service (AWS KMS). SSE encrypts messages as soon as Amazon SQS receives them. The messages are stored in encrypted form, and Amazon SQS decrypts messages only when they are sent to an authorized consumer.

AMAZON SIMPLE NOTIFICATION SERVICE

As the name suggests, Amazon Simple Notification Service (Amazon SNS) is a web service used to send notifications from the cloud. It is easy to set up and operate and at the same time highly scalable, flexible, and cost-effective. SNS has the capacity to publish a message from an application and then immediately deliver it to subscribers. It follows the publish-subscribe mechanism, also known as *pub-sub messaging*. It is a form of asynchronous service-to-service communication used in serverless and microservice architectures. In this model, any message published to a topic is immediately received by all the subscribers to the topic. Just like SQS, SNS is used to enable event-driven architectures or to decouple applications to increase performance, reliability, and scalability.

To use SNS, you must first create a “topic” identifying a specific subject or event type. A topic is used for publishing messages and allowing clients to subscribe for notifications. Once a topic is created, the topic owner can set policies for it such as limiting who can publish messages or subscribe to notifications or specifying which notification protocols are supported. To broadcast a message, a component called a *publisher* simply pushes a message to the topic. These topics transfer messages with no or very little queuing and push them out immediately to all subscribers. [Figure 7-11](https://learning.oreilly.com/library/view/aws-certified-solutions/9781260470192/ch7.xhtml#fig7_11) shows the publisher and subscriber model.



**Figure 7-11** Publisher and subscriber model

Subscribers are clients interested in receiving notifications from topics of interest; they can subscribe to a topic or be subscribed by the topic owner. Subscribers specify the protocol and endpoint (URL, e-mail address, etc.) for notifications to be delivered. All components that subscribe to the topic will receive every message that is broadcast, unless a message filtering policy is set by the subscriber. The publishers and subscribers can operate independently of each other, which means publishers need not know who has subscribed to the messages, and similarly, the subscribers don’t have to know from where the message is coming.

These are some of the features of Amazon SNS:

•   It is reliable since the messages are stored across multiple AZs by default.

•   It offers flexible message delivery over multiple transport protocols. It can be HTTP/HTTPS, e-mail, SMS, Lambda, and SQS where the message can be delivered.

•   The messages can be delivered instantly or can be delayed. It follows push-based delivery, which means messages are automatically sent to subscribers.

•   It provides monitoring capability. Amazon SNS and CloudWatch are integrated, so you can collect, view, and analyze metrics for every active Amazon SNS topic.

•   It can be accessed from the AWS Management Console, AWS Command Line Interface (CLI), AWS Tools for Windows PowerShell, AWS SDKs, and Amazon SNS Query API.

•   Amazon SNS messages can contain up to 256KB of text data with the exception of SMS, which can contain up to 140 bytes. If you publish a message that exceeds the size limit, Amazon SNS sends it as multiple messages, each fitting within the size limit. Messages are not cut off in the middle of a word but on whole-word boundaries. The total size limit for a single SMS publish action is 1,600 bytes.

With these three simple steps, you can get started with Amazon SNS:

•   **Create a topic** A topic is a communication channel to send messages and subscribe to notifications. It provides an access point for publishers and subscribers to communicate with each other.

•   **Subscribe to a topic** To receive messages published to a topic, you have to subscribe an endpoint to that topic. Once you subscribe an endpoint to a topic and the subscription is confirmed, the endpoint will receive all messages published to that topic.

•   **Publish to a topic** Publishers send messages to topics. Once a new message is published, Amazon SNS attempts to deliver that message to every endpoint that is subscribed to the topic.

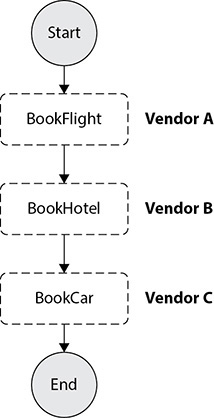
There are several scenarios where you use SNS and SQS together.

Say you have uploaded a new video to S3. The moment a video is uploaded, it triggers a message to be published to the SNS topic and is then replicated and sent to SQS queues. This sends the S3 event to multiple Lambda functions to be processed independently. In this case, the processing can be encoding the video to a different format (360p, 480p, 720p, 1080p) in parallel.

You have an order management system. Whenever someone places an order, an SNS notification is created. It is then sent to the order queue (SQS) and processed by EC2 servers. You can again have different SQSs depending on the priority of the order, say, a high-priority SQS queue and a low-priority SQS queue. When the order goes to the high-priority queue, it will be shipped immediately, and when it goes to the low-priority queue, it will be shipped after two or three days.

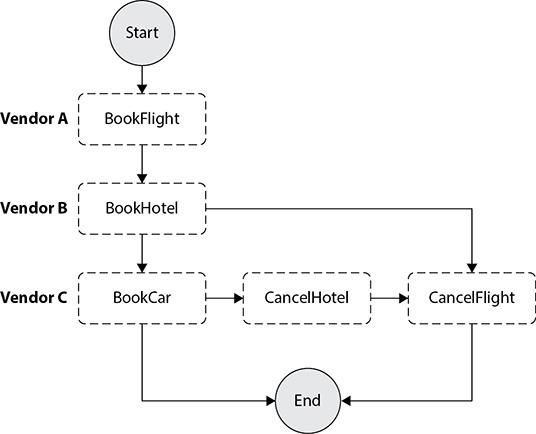
AWS STEP FUNCTIONS AND AMAZON SIMPLE WORKFLOW (SWF)

AWS Step Functions is a fully managed service that makes it easy to coordinate the components of distributed applications and microservices using visual workflow. It is really easy to use and scales down to little one-off shell-script equivalents and up to billions of complex multiphase tasks. Let’s take a simple example to understand this. Say you are planning to go to Europe for your next vacation. For your vacation, you need to do the following three tasks in sequence: book a flight, book a hotel, and book a rental car. For each step you are going to choose a different vendor. This is shown in [Figure 7-12](https://learning.oreilly.com/library/view/aws-certified-solutions/9781260470192/ch7.xhtml#fig7_12).



**Figure 7-12** Steps for vacation planning

Now if you are not able to reserve a rental car because of the unavailability of it, you should be able to automatically cancel the hotel booking and flight ticket ([Figure 7-13](https://learning.oreilly.com/library/view/aws-certified-solutions/9781260470192/ch7.xhtml#fig7_13)).



**Figure 7-13** Cancellation of hotel and flight

In this case, there are multiple ways of solving this problem.

You could create a function for each step and just link your functions together. That’s not terrible, but it does not give you modular, independent functions that each does one thing well. If you invoke one Lambda from another and you do it synchronously, that doesn’t scale because it might want to call another, and another, and so on, depending on how many steps you have. So, you can do it asynchronously, which is actually a better design, but then error handling gets hard. The more steps you add, the more difficult it is going to become to handle the error. Alternatively, you could keep track of your state by writing it into a database, say Amazon DynamoDB, or you could pass your state and control around through queues, but again both of those ideas take a lot of effort. What if you can do this seamlessly? Thus, if you are designing a coordination solution, it must have several characteristics:

•   It needs to scale out as demand grows. You should be able to run one execution or run thousands.

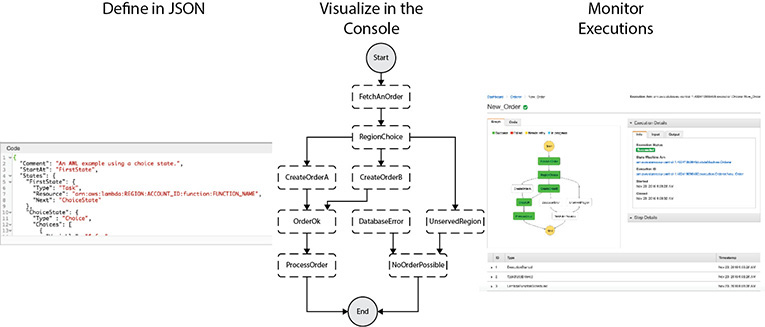
•   You can never lose state.

•   It deals with errors and times out and implements things like try/catch/finally.

•   It is easy to use and easy to manage.

•   It keeps a record of its operation and is completely auditable.

With AWS Step Functions, you define your application as a state machine, a series of steps that together capture the behavior of the app. States in the state machine may be tasks, sequential steps, parallel steps, branching paths (choice), and/or timers (wait). Tasks are units of work, and this work may be performed by AWS Lambda functions, Amazon EC2 instances, containers, or on-premises servers; anything that can communicate with the Step Functions API may be assigned a task. When you start a state machine, you pass it input in the form of JSON, and each state changes or adds to this JSON blob as output, which becomes input to the next state. The console provides this visualization and uses it to provide near-real-time information on your state machine execution. The management console automatically graphs each state in the order of execution, making it easy to design multistep applications. The console highlights the real-time status of each step and provides a detailed history of every execution. Step Functions operate and scale the steps of your application and underlying compute for you to ensure your application executes reliably under increasing demand. [Figure 7-14](https://learning.oreilly.com/library/view/aws-certified-solutions/9781260470192/ch7.xhtml#fig7_14) shows the application lifecycle in AWS Step Functions.



**Figure 7-14** Application lifecycle in AWS Step Functions

As of the writing this book, AWS Step Functions has following state types:

•   **Task** This is a single unit of work. Task states do your work. These call on your application components and microservices. There are two kinds of task states: one pushes a call to AWS Lambda functions, and the other dispatches tasks to applications.

•   **Choice** Using the choice states, you can use branching logic to your state machines.

•   **Parallel** Parallel states allow you to fork the same input across multiple states and then join the results into a combined output. This is really useful when you want to apply several independent manipulations to your data, such as image processing or data reduction.

•   **Wait** You can delay for a specified time by specifying wait in state.

•   **Fail** This stops an execution and marks it as a failure.

•   **Succeed** This stops an execution successfully.

•   **Pass** This passes its input to its output.

•   **Map** The Map state can be used to run a set of steps for each element of an input array.

AWS Step Functions is replacing Amazon Simple Workflow Service (SWF). Amazon SWF continues to exist today for customers who have already built their application using it. If you are building a new application, then you should consider AWS Step Functions instead of SWF. Moreover, AWS SWF is not a managed service; therefore, you need to configure everything manually in the EC2 servers.

AWS ELASTIC BEANSTALK

With Elastic Beanstalk, you can deploy, monitor, and scale an application on AWS quickly and easily. Elastic Beanstalk is the simplest and fastest way of deploying web applications. You just need to upload your code, and Elastic Beanstalk will provision all the resources such as Amazon EC2, Amazon Elastic Container Service (Amazon ECS), Auto Scaling, and Elastic Load Balancing for you behind the scenes. Elastic Beanstalk lets you focus on building applications without worrying about managing infrastructure. Although the infrastructure is provisioned and managed by Elastic Beanstalk, you maintain complete control over it.

If you don’t have much AWS knowledge and want to deploy an application, you might do several tasks. You can start by creating a VPC and then create public and private subnets in different AZs, launch EC2 instances, integrate them with Auto Scaling and ELB, provision a database, and so on. Laying the infrastructure itself can become challenging, and on top of that, if you have to manage everything manually, it adds to more overhead. Elastic Beanstalk solves this problem.

An Elastic Beanstalk application consists of three key components. The environment consists of the infrastructure supporting the application, such as the EC2 instances, RDS, Elastic Load Balancer, Auto Scaling, and so on. An environment runs a single application version at a time for better scalability. You can create many different environments for an application. For example, you can have a separate environment for production, a separate environment for test/dev, and so on. The next component is the application version. It is nothing but the actual application code that is stored in Amazon S3. You can have multiple versions of an application, and each version will be stored separately. The third component is the saved configuration. It defines how an environment and its resources should behave. It can be used to launch new environments quickly or roll back configuration. An application can have many saved configurations.

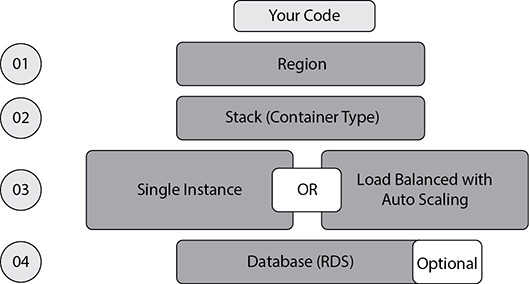
AWS Elastic Beanstalk has two types of environment tiers to support different types of web applications:

•   Web servers are standard applications that listen for and then process HTTP requests, typically over port 80.

•   Workers are specialized applications that have a background processing task that listens for messages on an Amazon SQS queue. Worker applications post those messages to your application by using HTTP.

It can be deployed either in a single instance or with multiple instances with the database (optional) in both cases. When deployed with multiple instances, Elastic Beanstalk provisions the necessary infrastructure resources such as load balancers, Auto Scaling groups, security groups, and databases. It also configures Amazon Route 53 and gives you a unique domain name. The single instance is mainly used for development or testing purposes, whereas multiple instances can be used for production workloads.

Elastic Beanstalk configures each EC2 instance in your environment with the components necessary to run applications for the selected platform. You don’t have to manually log in and configure EC2 instances. Elastic Beanstalk does everything for you. You can add AWS Elastic Beanstalk configuration files to your web application’s source code to configure your environment and customize the AWS resources that it contains. The configuration files can be either a JSON file or YAML. Using these configuration files, you can customize your Auto Scaling fleet as well. You should never manually log in and configure EC2 instances since all the manual changes will be lost on scaling events. [Figure 7-15](https://learning.oreilly.com/library/view/aws-certified-solutions/9781260470192/ch7.xhtml#fig7_15) shows what a deployment in Elastic Beanstalk looks like.



**Figure 7-15** Deployment in AWS Elastic Beanstalk

AWS Elastic Beanstalk provides a unified user interface to monitor and manage the health of your applications. It collects 40+ key metrics and attributes to determine the health of your application. It has a health dashboard in which you can monitor the application. It is also integrated with Amazon CloudWatch.

AWS Elastic Beanstalk supports the following languages and development stacks:

•   Apache Tomcat for Java applications

•   Apache HTTP Server for PHP applications

•   Apache HTTP Server for Python applications

•   Nginx or Apache HTTP Server for Node.js applications

•   Passenger or Puma for Ruby applications

•   Microsoft IIS 7.5, 8.0, and 8.5 for .NET applications

•   Java SE

•   Docker

•   Go

AWS OPSWORKS

When you’re building an application, you want to get new features out to your users fast, but having to manage all the infrastructure that your application needs and respond to changing conditions such as spikes in traffic can be error prone and hard to repeat if you’re configuring everything manually. Wouldn’t it be nice if you could automate operational tasks like software configuration, server scaling, deployments, and database setup so that you could focus on developing instead of doing all that heavy lifting?

AWS OpsWorks is a configuration management service that helps you deploy and operate applications of all shapes and sizes. OpsWorks allows you to quickly configure, deploy, and update your applications. It even gives you tools to automate operations such as automatic instant scaling and health monitoring. You have a lot of flexibility in defining your applications, architecture, and other things such as package installations, software configurations, and the resources your application needs such as storage databases or load balancers.

OpsWorks provides managed instances of Chef and Puppet. Chef and Puppet are automation platforms that allow you to use code to automate the configurations of your servers. OpsWorks lets you use Chef and Puppet to automate how servers are configured, deployed, and managed across your Amazon EC2 instances or on-premises compute environments.

OpsWorks offers three tools: AWS OpsWorks for Chef Automate, AWS OpsWorks for Puppet Enterprise, and AWS OpsWorks Stacks.

AWS OpsWorks for Chef Automate provides a fully managed Chef server and suite of automation tools that give you workflow automation for continuous deployment, automated testing for compliance and security, and a user interface that gives you visibility into your nodes and their status. The Chef server gives you full stack automation by handling operational tasks such as software and operating system configurations, package installations, database setups, and more. The Chef server centrally stores your configuration tasks and provides them to each node in your compute environment at any scale, from a few nodes to thousands of nodes. OpsWorks for Chef Automate is completely compatible with tooling and cookbooks from the Chef community and automatically registers new nodes with your Chef server.

AWS OpsWorks for Puppet Enterprise provides a managed Puppet Enterprise server and suite of automation tools giving you workflow automation for orchestration, automated provisioning, and visualization for traceability. The Puppet Enterprise server gives you full stack automation by handling operational tasks such as software and operating system configurations, package installations, database setups, and more. The Puppet Master centrally stores your configuration tasks and provides them to each node in your compute environment at any scale.

AWS OpsWorks Stacks lets you manage applications and servers on AWS and on-premises. Using OpsWorks Stacks, you model your entire application as a *stack* consisting of various layers. Layers are like blueprints that define how to set up and configure a set of Amazon EC2 instances and related resources.

OpsWorks provides prebuilt layers for common components, including Ruby, PHP, Node.js, Java, Amazon RDS, HA Proxy, MySQL, and Memcached. It also allows you to define your own layers for practically any technology and configure your layer however you want using Chef recipes. After you define all the layers you need to run your application stack, you just choose the operating system and the instance type to add. You can even scale the number of instances running by time of day or average CPU load. Once your stack is up and running, OpsWorks will pull the code from your repository and deploy it on your instances, and you will have a stack up and running based on the layers you defined earlier.

Using OpsWorks to automate, deploy, and manage applications saves you a lot of time. Without OpsWorks, if you needed to scale up the number of servers, you would need to manually configure everything including web framework configurations, installation scripts, initialization tasks, and database setups for each new instance. With OpsWorks, you set up and configure whatever your application needs for each layer once and let OpsWorks automatically configure all instances launched into that layer. It lets you focus on building amazing applications and services for your users without having to spend a lot of time manually configuring instances, software, and databases. It helps automate your infrastructure, gets your application to your users faster, helps you manage scale and complexity, and protects your applications from failure and downtime.

There is no additional charge for using OpsWorks. You pay for the AWS resources needed to store and run your applications.

AMAZON COGNITO

When you’re building a mobile app, you know that your users probably have more than one device—maybe a smartphone for the work commute and a tablet for enjoying movies later. Being able to sync your user’s profile information, whether that’s saved game data or some other kind of information, is really important so they can have a great experience with your app whenever and wherever they’re using it, regardless of which device they use. If you want to create a back end to support that kind of storage and synchronization, it is a lot of work. You have to build it, deploy it, and manage the infrastructure that it runs on. Wouldn’t it be great if you could stay focused on writing your app without having to build your own back end? You just concentrate on syncing and storing users’ data?

Amazon Cognito is a user identity and data synchronization service that makes it really easy for you to manage user data for your apps across multiple mobile or connected devices. You can create identities for users of your app using public login providers such as Google, Facebook, and Amazon, and through enterprise identity providers such as Microsoft Active Directory using SAML. This service also supports unauthenticated identities. Users can start off trying your app without logging in, and then when they do create a profile using one of the public logging providers, their profile data is seamlessly transferred. Amazon Cognito user pools provide a secure user directory that scales to hundreds of millions of users. User pools provide user profiles and authentication tokens for users who sign up directly and for federated users who sign in with social and enterprise identity providers. Amazon Cognito User Pools is a standards-based identity provider and supports identity and access management standards, such as OAuth 2.0, SAML 2.0, and OpenID Connect.

You can use Amazon Cognito to sync any kind of user data and key-value pairs whether that is app preferences, game state, or anything that makes sense for your app. By using Amazon Cognito, you don’t have to worry about running your own back-end service and dealing with identity network storage or sync issues. You just save the user data using the Amazon Cognito API and sync. The user’s data is securely synced and stored in the AWS cloud.

Amazon Cognito provides solutions to control access to AWS resources from your app. You can define roles and map users to different roles so your app can access only the resources that are authorized for each user.

It is really easy to use Amazon Cognito with your app. Instead of taking months to build a solution yourself, it just takes a few lines of code to be able to sync your users’ data. If you’re using other AWS services, Amazon Cognito provides you with even more benefits such as delivering temporary credentials of limited privileges that users can use to access AWS resources. Amazon Cognito lets you focus on building your app and making sure that your users have a consistent experience regardless of the device they’re using without you having to worry about the heavy lifting associated with building your own back-end solution to sync user data.

AMAZON ELASTIC MAPREDUCE

Whatever kind of industry you are in, being able to analyze data coming from a wide variety of sources can help you to make transformational decisions. To be able to make these decisions based on data of any scale, you need to be able to access the right kind of tools to process and analyze your data. Software frameworks like Hadoop can help you store and process large amounts of data by distributing the data and processing across many computers. But at the same time, deploying, configuring, and managing Hadoop clusters can be difficult, expensive, and time-consuming. Traditionally, you had to purchase the underlying servers and storage hardware, provision the hardware, and then deploy and manage the software even before you had a chance to do anything with your data.

Wouldn’t it be great if there was an easier way? Amazon Elastic MapReduce (EMR) solves this problem. Using the elastic infrastructure of Amazon EC2 and Amazon S3, Amazon EMR provides a managed Hadoop framework that distributes the computation of your data over multiple Amazon EC2 instances.

Amazon EMR is easy to use. To get started, you need to load the data into Amazon S3; then you can launch EMR clusters in minutes. Once a cluster is launched, it can start processing your data immediately, and you don’t need to worry about setting up, running, or tuning clusters. Since it is a managed service, Amazon is going to take care of the heavy lifting behind the scenes. You just need to define how many nodes in the cluster you need, what types of instances you need, and what applications you want to install in the cluster. Then Amazon will provision everything for you.

Thus, you can focus on the analysis of your data. When your job is complete, you can retrieve the output from Amazon S3. You can also feed this data from S3 to a visualization tool or use it for reporting purposes. Amazon EMR monitors the job. Once the job is completed, EMR can shut down the cluster or keep it running so it is available for additional processing queries. You could easily expand or shrink your clusters to handle more or less data and to get the processing done more quickly.

In a Hadoop ecosystem, the data remains on the servers that process the data. As a result, it takes some time to add or remove a server from a cluster. In the case of EMR, the data remains decoupled between the EC2 servers and Amazon S3. EC2 only processes the data, and the actual data resides in Amazon S3. EMRFS (EMR file system) is used by an EMR cluster for reading and writing files from EMR directly to S3. As a result, at any point in time, you can scale up or scale down. Say you are running an EMR job and you have selected only one EC2 server for running the job, and let’s say the job takes 10 hours. Let’s say this EC2 server costs $1 per hour, so the total cost of running this job would be $10 ($1 \* 10 hours), and the amount of time it takes is 10 hours. Now instead of processing this job with one EC2 server, if you create an EMR cluster with a 10-node EC2 server, the job is going to be finished in just one hour instead of ten since you added 10 times more compute. Now price-wise, it is going to cost $10 ($1 per server \* 10 servers). In this case, you are processing the same job ten times faster but paying the same amount of money to process the job.

When you store your data in Amazon S3, you can access it with multiple EMR clusters simultaneously, which means users can quickly spin off as many clusters as they need to test new ideas and can terminate clusters when they’re no longer needed. This can help speed innovation and lower the cost of experimentation, and you can even optimize each cluster for a particular application.

You can have three types of nodes in an Amazon EMR cluster:

•   **Master node** This node takes care of coordinating the distribution of the job across core and task nodes.

•   **Core node** This node takes care of running the task that the master node assigns. This node also stores the data in the Hadoop Distributed File System (HDFS) on your cluster.

•   **Task node** This node runs only the task and does not store any data. The task nodes are optional and provide pure compute to your cluster.

Amazon EMR is low cost and provides a range of pricing options, including hourly on-demand pricing, the ability to reserve capacity for a lower hourly rate, or the ability to name your own price for the resources you need with spot instances. Spot instances are a great use case for Amazon EMR; you can use the spot instance for the task node since it does not store any data and is used for pure compute. You can mix and match different types of EC2 instance types for spot instances so that even if you lose a particular type of EC2 instance, the other type is not impacted when someone over bids you.

Amazon EMR automatically configures the security groups for the cluster and makes it easy to control access. You can even launch clusters in an Amazon Virtual Private Cloud (VPC).

With Amazon EMR you can run MapReduce and a variety of powerful applications and frameworks, such as Hive, Pig, HBase, Impala, Cascading, and Spark. You can also use a variety of different programming languages. Amazon EMR supports multiple Hadoop distributions and integrates with popular third-party tools. You can also install additional software or further customize the clusters for your specific use case.

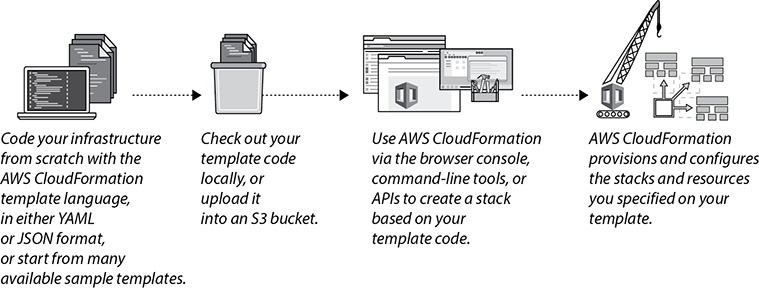
AWS CLOUDFORMATION

When you are managing infrastructure, you might use run books and scripts to create and manage everything. Version controlling and keeping track of changes can be challenging. Things get even harder when you need to replicate your entire production stack multiple times for development and testing purposes. If you want to provision infrastructure stacks directly from a collection of scripts, it is not simple. Wouldn’t it be great if you could create and manage your infrastructure and application stack in a controlled and predictable way?

You can do the same seamlessly using AWS CloudFormation. CloudFormation provisions and manages stacks of AWS resources based on templates you create to model your infrastructure architecture. You can manage anything from a single Amazon EC2 instance to a complex multitier, multiregion application. CloudFormation can be used to define simple things such as an Amazon VPC subnet, as well as provision services such as AWS OpsWorks or AWS Elastic Beanstalk.

It is easy to get started with CloudFormation. You simply create a template, which is a JSON file that serves as a blueprint to define the configuration of all the AWS resources that make up your infrastructure and application stack, or you can select a sample prebuilt template that CloudFormation provides for commonly used architectures such as a LAMP stack running on Amazon EC2 and an Amazon RDS. Next you just upload your template to CloudFormation. You can also select parameters such as the number of instances or instance type if necessary; then CloudFormation will provision and configure your AWS resource stack. You can update your CloudFormation stack at any time by uploading a modified template through the AWS management console, CLI, or SDK. You can also check your template into version control, so you’re able to keep track of all changes made to your infrastructure and application stack. With CloudFormation you can version control your infrastructure architecture the same way you would with software code. Provisioning infrastructure is as simple as creating and uploading a template to CloudFormation. This makes replicating your infrastructure simple. You can easily and quickly spin up a replica of your production stack for development and testing with a few clicks in the AWS Management Console. You can tear down and rebuild the replica stacks whenever you want. Replicating production staff could have been time-consuming and error prone if you did it manually, but with CloudFormation you can create and manage the AWS resource stack quickly and reliably. There is no additional charge for CloudFormation; you pay only for the AWS resources that CloudFormation creates and your application uses. CloudFormation allows you to treat your infrastructure as just code.

To use AWS CloudFormation, you need templates and stacks. Using templates, you can describe your AWS resources and their properties. Whenever you create a stack, CloudFormation provisions the resources as per your template. A CloudFormation template is a JSON- or YAML-formatted text file. You can create a template directly from the editor available from the AWS Management Console or by using any text editor. You can save the template with an extension such as .json or .yaml.txt or .template. These templates serve as a blueprint for building all the resources. Using a template you can specify the resource that CloudFormation is going to build. For example, you can specify a specific type of an EC2 instance as a resource in your CloudFormation template. All the AWS resources collected together is called a *stack*. You can manage the stack as a single unit. For example, using a stack you can create multiple resources like an EC2 instance, VPC, or RDS database via a single CloudFormation template. You can create, update, or delete a collection of resources by creating, updating, or deleting stacks. Using AWS CloudFormation templates, you can define all the resources in a stack. [Figure 7-16](https://learning.oreilly.com/library/view/aws-certified-solutions/9781260470192/ch7.xhtml#fig7_16) shows how AWS CloudFormation works.



**Figure 7-16** How AWS CloudFormation works

Whenever you have to make changes to the running resources in a stack, such as when you want to add an ELB to the existing EC2 instances, you need to update the existing stack. Before making the changes, you can generate a change set that summarizes the proposed change. Using a change set, you can foresee what is going to be the impact on your running resources if the change is being made. Make sure you don’t run into limits; for example, EC2 has a soft limit, and VPC has limits per region, per account. If you think you might hit a limit, you can always increase it with a support ticket.

MONITORING IN AWS

Whenever you build or deploy your applications, you should have the capability to monitor them from end to end. By using the right monitoring tools, you will be able to find out whether the application is performing well, whether there is a security vulnerability in the application, whether the application is taking too much of your resources in terms of CPU or memory, or whether the application is constrained in terms of resources when you build or deploy your applications in AWS. In addition to these questions, you may ask some additional questions such as, How do I capture, view, and act on resource availability and state changes? How do I track the key performance indicators across AWS resources? How can API calls be logged within my AWS accounts? How do I track cost within my AWS accounts? AWS provides you with lots of tools that give you this capability. Apart from technical reasons, there are several other reasons why you should monitor your systems. Via monitoring you can find out whether your customers are getting a good experience or not, whether the changes you are making in the system are impacting overall performance, whether the same problem can be prevented in the future or not, when you need to scale, and so on.



**NOTE**    In AWS, resources are software defined, and changes to them are tracked as API calls. The current and past states of your environment can be monitored and acted on in real time. AWS scaling allows for ubiquitous logging, which can be extended to your application logs and centralized for analysis, audit, and mitigation purposes.

In this section, we will discuss all these tools that can be used to monitor the AWS resources.

AMAZON CLOUDWATCH

Watching the cloud means monitoring all the resources deployed in the AWS cloud. Amazon CloudWatch provides capabilities to gain visibility into what’s going on with your resources. You can monitor the health checks, look at the utilization, and view performance. Amazon CloudWatch monitors your AWS cloud resources and your cloud-powered applications. It tracks the metrics so that you can visualize and review them. You can also set alarms that will fire when a metric goes beyond the limit that you specify. CloudWatch gives you visibility into resource utilization, application performance, and operational health.

Let’s explore what Amazon CloudWatch can do.

Metrics Collection and Tracking

Amazon CloudWatch provides metrics for all the services, and there are more than 100 types of metrics available among all the different services. You can look at these metrics for your EC2 instances (e.g., in an EC2 instance, you can look at CPU, Network In/Out, and so on), RDS, ELBs, EBS volumes, DynamoDB, and so on. Apart from the default metrics available, you can also create your own custom metrics using your application and monitor them via Amazon CloudWatch. Please log in to the AWS console and browse all the metrics you can monitor via CloudWatch. You will see that you will be able to monitor almost everything via CloudWatch, but for monitoring some components you may have to write custom metrics. For example, in order to monitor the memory utilization, you have to write a custom metric. Previously (before November 2016), these metrics used to be retained for 14 days, but now the metrics are retained depending on the metric interval.

•   For the one-minute data point, the retention is 15 days.

•   For the five-minute data point, the retention is 63 days.

•   For the one-hour data point, the retention is 15 months or 455 days.

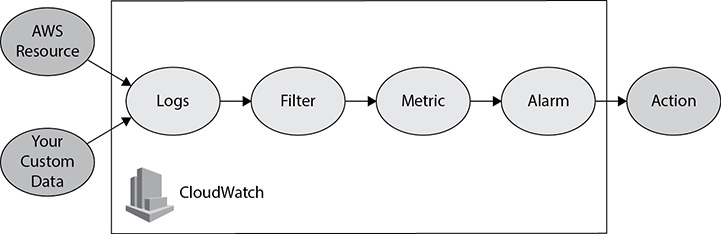
Capture Real-Time Changes Using Amazon CloudWatch Events

Amazon CloudWatch Events helps you to detect any changes made to your AWS resource. When CloudWatch Events detects a change, it delivers a notification in almost real time to a target you choose. The target can be a Lambda function, an SNS queue, an Amazon SNS topic, or a Kinesis Stream or built-in target. You can set up your own rules and can take action whenever you detect a change. There are many ways to leverage Amazon CloudWatch Events. For example, say your company has a strict rule that whenever someone is creating an EC2 instance, they should tag it. You can create a CloudWatch event whenever an EC2 is created and send it to a Lambda function. The Lambda function will check whether the newly created instance has a tag. If it does not have a tag, it can automatically tag the instance as per the logic you define.

Monitoring and Storing Logs

You can use CloudWatch Logs to monitor and troubleshoot your systems and applications using your existing system, application, and custom log files. You can send your existing log files to CloudWatch Logs and monitor these logs in near real time. Amazon CloudWatch Logs is a managed service to collect and keep your logs. It can aggregate and centralize logs across multiple sources. Using the CloudWatch Logs Agent, you can stream the log files from an EC2 instance. The CloudWatch agent is available for both Linux and Windows. In addition to the agent, you can publish log data using the AWS CLI, the CloudWatch Logs SDK, or the CloudWatch Logs API. You can further export data to S3 for analytics and/or archival or stream to the Amazon Elasticsearch Service or third-party tools like Splunk.

CloudWatch Logs can be used to monitor your logs for specific phrases, values, or patterns. [Figure 7-17](https://learning.oreilly.com/library/view/aws-certified-solutions/9781260470192/ch7.xhtml#fig7_17) shows how you can filter a particular pattern from CloudWatch Logs.



**Figure 7-17** Filtering a particular pattern from Amazon CloudWatch Logs

For example, you could set an alarm on the number of errors that occur in your system logs or view graphs of web request latencies from your application logs. You can view the original log data to see the source of the problem if needed.

Set Alarms

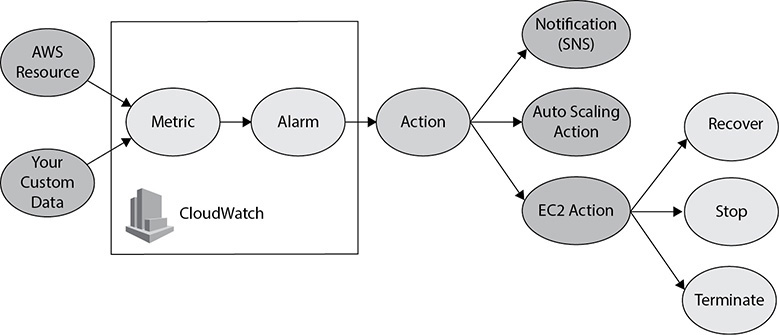
You can create a CloudWatch alarm that sends an Amazon Simple Notification Service message when the alarm changes state. An alarm watches a single metric over a time period you specify and performs one or more actions based on the value of the metric relative to a given threshold over a number of time periods. The action is a notification sent to an Amazon Simple Notification Service topic or Auto Scaling policy. Alarms invoke actions for sustained state changes only. A CloudWatch alarm will not invoke an action just because it is in a particular state; to be invoked, the state must have changed and been maintained for a specified period of time. An alarm has the following possible states:

•   **OK** This state means the metric is within the defined threshold.

•   **ALARM** This state means the metric is outside the defined threshold.

•   **INSUFFICIENT\_DATA** This state means the alarm has just started, the metric is not available, or not enough data is available to determine the alarm state.

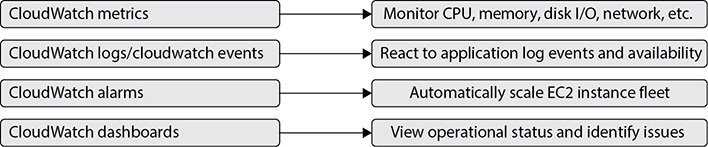
Let’s look at an example to understand this. You have created an alarm to alert you when the CPU runs at or above 75 percent in an EC2 instance. Say you have set the alarm threshold to 3, which means after the third occurrence of 75 percent CPU utilization in the EC2 instance, the alarm is going to invoke the action associated with it. Say you have defined an action that says when the CPU gets to more than 75 percent, start one more EC2 instance via Auto Scaling, and after the third occurrence, Auto Scaling should add more EC2 instances for you. [Figure 7-18](https://learning.oreilly.com/library/view/aws-certified-solutions/9781260470192/ch7.xhtml#fig7_18) shows how Amazon CloudWatch metrics and CloudWatch alarms can be used together.



**Figure 7-18** Using Amazon CloudWatch metrics and Amazon CloudWatch alarms together

View Graphs and Statistics

You can use Amazon CloudWatch dashboards to view different types of graphs and statistics of the resources you have deployed. You can create your own dashboard and can have a consolidated view across your resources. The Amazon CloudWatch dashboard is a single view for selected metrics to help you assess the health of your resources and applications across one or more regions. It acts as an operational playbook that provides guidance for team members during operational events about how to respond to specific incidents. It is a common view of critical resource and application measurements that can be shared by team members for faster communication flow during operational events. [Figure 7-19](https://learning.oreilly.com/library/view/aws-certified-solutions/9781260470192/ch7.xhtml#fig7_19) summarizes all the Amazon CloudWatch capabilities.



**Figure 7-19** Amazon CloudWatch capabilities

AWS CLOUDTRAIL

AWS CloudTrail is a service that logs all API calls, including console activities and command-line instructions. It logs exactly who did what, when, and from where. It can tell you which resources were acted upon in the API call and where the API call was made from and to whom. That means you have full visibility into the accesses, changes, or activity within your AWS environment. You can save these logs into your S3 buckets.

CloudTrail can help you achieve many tasks. You can track changes to AWS resources (for example, VPC security groups and NACLs), comply with rules (log and understand AWS API call history), and troubleshoot operational issues (quickly identify the most recent changes to your environment). Different accounts can send their trails to a central account, and then the central account can do analytics. After that, the central account can redistribute the trails and grant access to the trails.

AWS CloudTrail shows the results of the CloudTrail event history for the current region you are viewing for the last 90 days.

You can have the CloudTrail trail going to CloudWatch Logs and Amazon CloudWatch Events in addition to Amazon S3. This enables you to leverage features to help you archive, analyze, and respond to changes in your AWS resources. You can create up to five trails in an AWS region. [Figure 7-20](https://learning.oreilly.com/library/view/aws-certified-solutions/9781260470192/ch7.xhtml#fig7_20) shows how AWS CloudTrail works.



**Figure 7-20** How AWS CloudTrail works

You can follow these AWS CloudTrail best practices for setting up CloudTrail:

•   Enable AWS CloudTrail in all regions to get logs of API calls by setting up a trail that applies to all regions.

•   Enable log file validation using industry-standard algorithms, SHA-256 for hashing, and SHA-256 with RSA for digital signing.

•   By default, the log files delivered by CloudTrail to your bucket are encrypted by Amazon server-side encryption with Amazon S3 managed encryption keys (SSE-S3). To provide a security layer that is directly manageable, you can instead use server-side encryption with AWS KMS managed keys (SSE-KMS) for your CloudTrail log files.

•   Set up real-time monitoring of CloudTrail logs by sending them to CloudWatch logs.

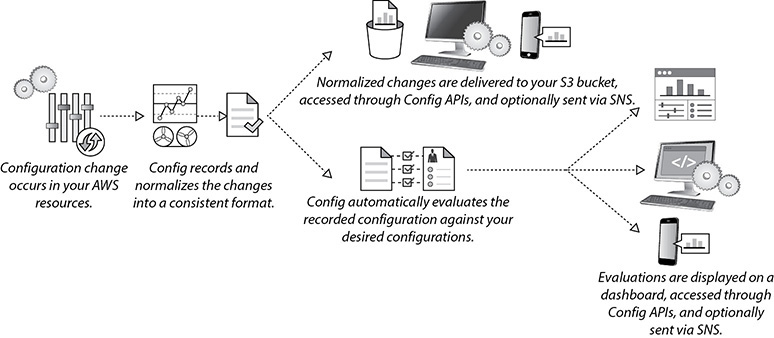
•   If you are using multiple AWS accounts, centralize CloudTrail logs in a single account.

•   For added durability, configure cross-region replication (CRR) for S3 buckets containing CloudTrail logs.

AWS CONFIG

AWS Config is a fully managed service that provides you with a detailed inventory of your AWS resources and their current configuration in an AWS account. It continuously records configuration changes to these resources (e.g., EC2 instance launch, ingress/egress rules of security groups, network ACL rules for VPCs, etc.). It lets you audit the resource configuration history and notifies you of resource configuration changes. Determine how a resource was configured at any point in time, and get notified via Amazon SNS when the configuration of a resource changes or when a rule becomes noncompliant.

A config rule represents desired configurations for a resource and is evaluated against configuration changes on the relevant resources, as recorded by AWS Config. The results of evaluating a rule against the configuration of a resource are available on a dashboard. Using config rules, you can assess your overall compliance and risk status from a configuration perspective, view compliance trends over time, and pinpoint which configuration change caused a resource to drift out of compliance with a rule. [Figure 7-21](https://learning.oreilly.com/library/view/aws-certified-solutions/9781260470192/ch7.xhtml#fig7_21) shows how AWS Config works.



**Figure 7-21** How AWS Config works

These are the things you can do with AWS Config:

•   **Continuous monitoring** AWS Config allows you to constantly monitor all your AWS resources and record any configuration changes in them. As a result, whenever there is a change, you can get instantly notified of it. It can make an inventory of all AWS resources and the configuration of those resources.

•   **Continuous assessment** AWS Config provides you with the ability to define rules for provisioning and configuring AWS resources, and it can continuously audit and assess the overall compliance of your AWS resource configurations with your organization’s policies and guidelines. AWS Config constantly assesses your resources against standard configuration, and whenever there is a deviation, it instantly notifies you.

•   **Change management** AWS Config helps you with change management. It can track what was changed, when it happened, and how the change might affect other AWS resources. This need might arise because of an unexpected configuration change, a suspected system failure, a compliance audit, or a possible security incident. You can also track the relationships among resources and review resource dependencies prior to making changes. Once you make a change, you can look at the configuration history to find out how it looked in the past.

•   **Operational troubleshooting** Since AWS Config tracks all the changes and constantly monitors all your resources, it can be used for troubleshooting against operational issues. It helps you to find the root cause by pointing out what change is causing the issue. Since AWS Config can be integrated with AWS CloudTrail, you can correlate configuration changes to particular events in your account.

•   **Compliance monitoring** You can use AWS Config for compliance monitoring for your entire account or across multiple accounts. If a resource violates a rule, AWS Config flags the resource and the rule as noncompliant. You can dive deeper to view the status for a specific region or a specific account across regions.

AMAZON VPC FLOW LOGS

Amazon VPC Flow Logs captures information about the IP traffic going to and from network interfaces in your VPC. The Flow Logs data is stored using Amazon CloudWatch Logs, which can then be integrated with additional services, such as Elasticsearch/Kibana for visualization.

The flow logs can be used to troubleshoot why specific traffic is not reaching an instance. Once enabled for a particular VPC, VPC subnet, or Elastic Network Interface (ENI), relevant network traffic will be logged to CloudWatch Logs for storage and analysis by your own applications or third-party tools.

You can create alarms that will fire if certain types of traffic are detected; you can also create metrics to help you to identify trends and patterns.

The information captured includes information about allowed and denied traffic (based on security group and network ACL rules). It also includes source and destination IP addresses, ports, the IANA protocol number, packet and byte counts, a time interval during which the flow was observed, and an action (ACCEPT or REJECT).

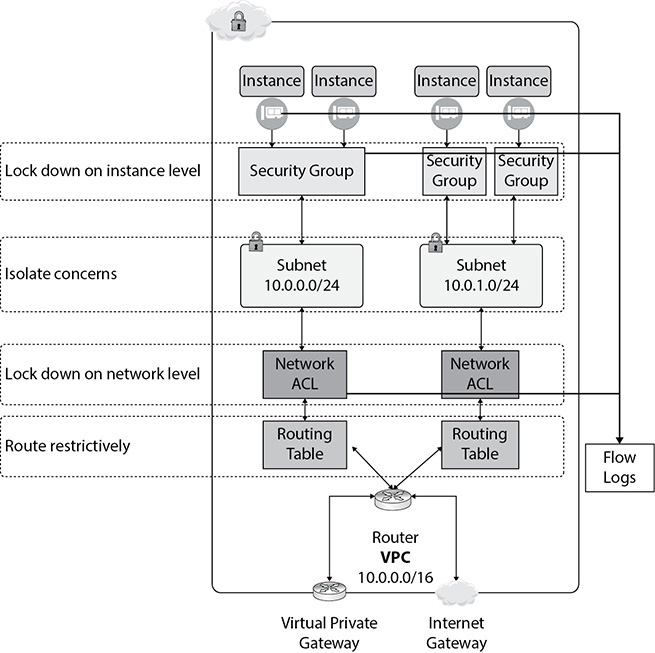
You can enable VPC Flow Logs at different levels.

•   **VPC** This would cover all network interfaces in that VPC.

•   **Subnet** This captures traffic on all network interfaces in that subnet.

•   **Network interface** This captures traffic specific to a single network interface.

[Figure 7-22](https://learning.oreilly.com/library/view/aws-certified-solutions/9781260470192/ch7.xhtml#fig7_22) shows VPC Flow Logs capture from different places.



**Figure 7-22** VPC Flow Logs capture

You can enable VPC Flow Logs from the AWS Management Console or the AWS CLI or by making calls to the EC2 API. VPC Flow Logs does not require any agents on EC2 instances. Once you create a flow log, it takes several minutes to begin collecting data and publishing to CloudWatch Logs. Please note that it should not be used as a tool for capturing real-time log streams for network interfaces. The flow logs can capture either all flows, rejected flows, or accepted flows. VPC Flow Logs can be used both for security monitoring and for application troubleshooting. You can create CloudWatch metrics from VPC log data.

AWS TRUSTED ADVISOR

AWS Trusted Advisor provides best practices (or checks) in five categories:

•   **Cost Optimization** You can save money on AWS by eliminating unused and idle resources or making commitments to reserved capacity.

•   **Security** Helps you to improve the security of your application by closing gaps, enabling various AWS security features, and examining your permissions.

•   **Fault Tolerance** You can increase the availability and redundancy of your AWS application by taking advantage of Auto Scaling, health checks, multi-AZs, and backup capabilities.

•   **Performance** Helps you to improve the performance of your service by checking your service limits, ensuring you take advantage of provisioned throughput, and monitoring for overutilized instances.

•   **Service Limits** It checks for service usage that is more than 80 percent of the service limit. Since the values are based on a snapshot, your current usage might differ. Sometimes it may take up to 24 hours to reflect any change. For example, the default service limit for EC2 instances is 20. If you have already created more than 16 instances it will be displayed on the dashboard.

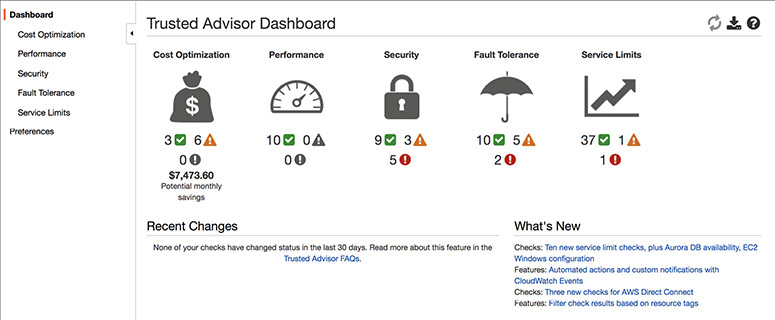
The status of the check is shown by using color coding on the dashboard page.

•   Red means action is recommended.

•   Yellow means investigation is recommended.

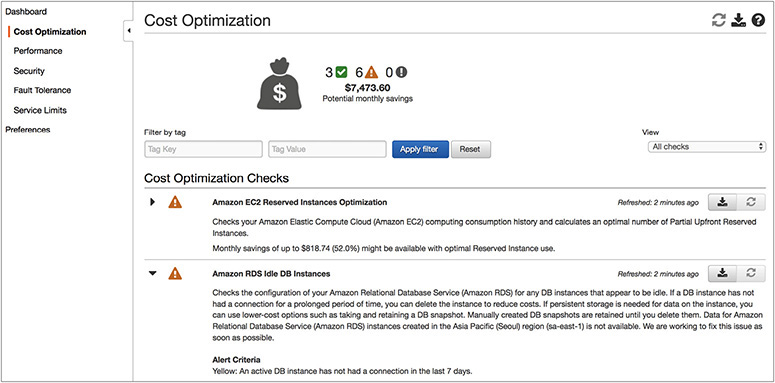
•   Green means no problem is detected.

[Figure 7-23](https://learning.oreilly.com/library/view/aws-certified-solutions/9781260470192/ch7.xhtml#fig7_23) shows the Trusted Advisor Dashboard screen.

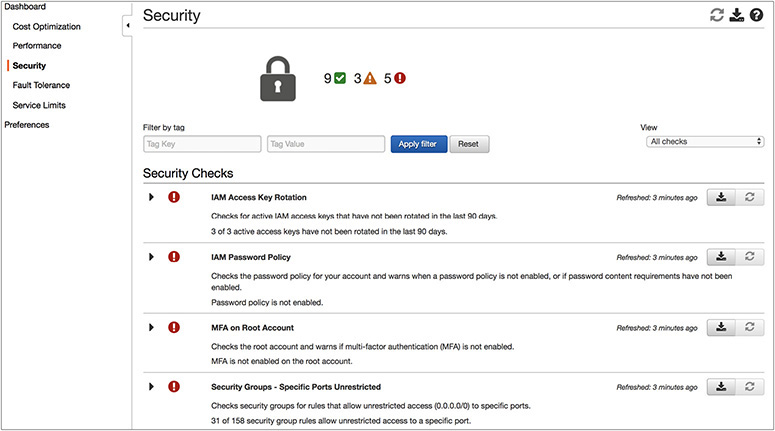


**Figure 7-23** Trusted Advisor Dashboard

These seven Trusted Advisor checks are available to all customers under various categories: Service Limits, S3 Bucket Permissions, Security Groups–Specific Ports Unrestricted, IAM Use, MFA on Root Account, EBS Public Snapshots, and RDS Public Snapshots. Customers can access the remaining checks by upgrading to a Business or Enterprise Support plan. [Figures 7-24](https://learning.oreilly.com/library/view/aws-certified-solutions/9781260470192/ch7.xhtml#fig7_24) and [7-25](https://learning.oreilly.com/library/view/aws-certified-solutions/9781260470192/ch7.xhtml#fig7_25) show a few checks that Trusted Advisor does on the Cost Optimization and Security tabs.



**Figure 7-24** Checks on the Cost Optimization tab from Trusted Advisor



**Figure 7-25** Checks on the Security tab from Trusted Advisor

AWS ORGANIZATIONS

Many AWS enterprises have found themselves managing multiple AWS accounts as they have scaled up their use of AWS for a variety of reasons. Some of these enterprises have added more accounts incrementally as individual teams and divisions make the move to the cloud.

Other enterprises use different accounts for test development and production systems or to meet strict guidelines for compliance such as HIPAA or PCI. As the number of these accounts increases, enterprises would like to set policies and manage billing across their accounts in a simple, more scalable way, without requiring custom scripts and manual processes.

Enterprises also want more efficient automated solutions for creating new accounts with the current policies applied as they create more accounts to meet the needs of their business.

AWS Organizations makes account management simple. It offers policy-based management from multiple AWS accounts. You can create groups of accounts and then apply policies to those groups that centrally control the use of AWS services down to the API level across multiple accounts. It enables you to centrally manage policies across multiple accounts, without requiring custom scripts and manual processes. For example, you can create a group of accounts that are used for production resources and then apply a policy to this group that limits which AWS service APIs those accounts can use. You can also use the organization’s APIs to help automate the creation of new AWS accounts with a few simple API calls. You can create a new account programmatically and then apply the correct policies to the new account automatically. AWS Organizations’ service control policies (SCPs) help you centrally control AWS service use across multiple AWS accounts in your organization.

Using AWS Organizations, you can set up a single payment method for all of these accounts through consolidated building. AWS Organizations is available to all AWS customers free of cost.

CHAPTER REVIEW

In this chapter, you learned about AWS Lambda, Amazon API Gateway, and Amazon Kinesis. You also learned about Amazon CloudFront, Amazon Route 53, AWS WAF, Amazon Simple Queue Service, Amazon Simple Notification Service, AWS Step Functions, Elastic Beanstalk, AWS OpsWorks, Amazon Cognito, Amazon EMR, AWS CloudFormation, Amazon CloudWatch, CloudTrail, AWS Config, VPC Flow Logs, AWS Trusted Advisor, and AWS Organizations.

With Lambda, you can run code for virtually any type of application or back-end service. Lambda runs and scales your code with high availability. Each Lambda function you create contains the code you want to execute, the configuration that defines how your code is executed, and, optionally, one or more event sources that detect events and invoke your function as they occur. AWS Lambda supports Java, Node.js, Python, and C#.

API Gateway is a fully managed service that makes it easy for developers to define, publish, deploy, maintain, monitor, and secure APIs at any scale. Clients integrate with the APIs using standard HTTPS requests. API Gateway serves as a front door (to access data, business logic, or functionality from your back-end services) to any web application running on Amazon EC2, Amazon ECS, AWS Lambda, or on-premises environment.

Amazon Kinesis Data Streams enables you to build custom applications that process or analyze streaming data for specialized needs. Kinesis Data Streams can continuously capture and store terabytes of data per hour from hundreds of thousands of sources such as web site clickstreams, financial transactions, social media feeds, IT logs, and location-tracking events.

Amazon Kinesis Data Firehose is the easiest way to load streaming data into data stores and analytics tools. It can capture, transform, and load streaming data into Amazon S3, Amazon Redshift, Amazon Elasticsearch Service, and Splunk, enabling near real-time analytics with the existing business intelligence tools and dashboards you’re already using today. It is a fully managed service that automatically scales to match the throughput of your data and requires no ongoing administration. It can also batch, compress, and encrypt the data before loading it, minimizing the amount of storage used at the destination and increasing security.

Amazon Kinesis Data Analytics is the easiest way to process and analyze real-time, streaming data. With Amazon Kinesis Data Analytics, you just use standard SQL to process your data streams, so you don’t have to learn any new programming languages. Simply point Kinesis Data Analytics at an incoming data stream, write your SQL queries, and specify where you want to load the results. Kinesis Data Analytics takes care of running your SQL queries continuously on data while it’s in transit and sends the results to the destinations.

Amazon CloudFront is a global CDN service that allows you to distribute content with low latency and provides high data transfer speeds. Amazon CloudFront employs a global network of edge locations and regional edge caches that cache copies of your content close to your viewers.

Amazon Route 53 is the managed DNS service of Amazon. DNS translates human-readable names like [www.example.com](https://www.example.com/) into the numeric IP addresses like 192.0.0.3 that servers/computers use to connect to each other. Route 53 is the only service that has 100 percent SLA. This service is region independent.

AWS WAF is a web application firewall that protects your web applications from various forms of attack. It helps to protect web sites and applications against attacks that could affect application availability, cause data breaches, cause downtime, compromise security, or consume excessive resources. It gives you control over which traffic to allow or block to your web applications by defining customizable web security rules.

AWS Shield is a managed service that provides protection against distributed denial-of-service (DDoS) attacks for applications running on AWS. There are two types of AWS Shield: AWS Shield Standard and AWS Shield Advanced. AWS Shield Standard is automatically enabled for all AWS customers.

Amazon Simple Queue Service is a fast, reliable, scalable, and fully managed queue service. Using Amazon SQS, you can quickly build message queuing applications that can run on any system. It can send, store, and receive messages between components. Like most AWS services, it’s accessible through a web API, as well as SDKs in most languages.

Amazon Simple Notification Service is a web service used to send notifications from the cloud. It is easy to set up and operate and at the same time highly scalable, flexible, and cost-effective. SNS has the capacity to publish messages from an application and then immediately deliver them to subscribers. It follows the publish-subscribe mechanism.

AWS Step Functions is a fully managed service that makes it easy to coordinate the components of distributed applications and microservices using a visual workflow. It is really easy to use and scales down to little one-off shell-script equivalents and up to billions of complex multiphase tasks.

Using AWS Elastic Beanstalk, you can deploy, monitor, and scale an application on AWS quickly and easily. Elastic Beanstalk is the simplest and fastest way to deploy web applications. You just need to upload your code and Elastic Beanstalk will provision all the resources like Amazon EC2, Amazon Elastic Container Service (Amazon ECS), Auto Scaling, and Elastic Load Balancing for you behind the scenes.

AWS OpsWorks is a configuration management service that helps you deploy and operate applications of all shapes and sizes. OpsWorks provides an easy way to quickly configure, deploy, and update your applications. OpsWorks offers AWS OpsWorks for Chef Automate, AWS OpsWorks for Puppet Enterprise, and AWS OpsWorks Stacks.

Amazon Cognito is a user identity and data synchronization service that makes it really easy for you to manage user data for your apps across multiple mobile or connected devices. You can create identities for users of your app using public login providers such as Google, Facebook, Amazon, and through enterprise identity providers such as Microsoft Active Directory using SAML.

Amazon EMR provides a managed Hadoop framework that distributes computation of your data over multiple Amazon EC2 instances. It decouples the compute and storage by keeping the data in Amazon S3 and using Amazon EC2 instances for processing the data.

AWS CloudFormation provisions and manages stacks of AWS resources based on templates you create to model your infrastructure architecture. You can manage anything, from a single Amazon EC2 instance to a complex multitier, multiregion application. AWS CloudFormation allows you to treat your infrastructure as just code.

Amazon CloudWatch is a monitoring service for AWS cloud resources and the applications you run on AWS. You can use Amazon CloudWatch to gain system-wide visibility into resource utilization, application performance, and operational health. You can use these insights to react and keep your application running smoothly. Amazon CloudWatch monitors your AWS cloud resources and your cloud-powered applications. It tracks the metrics so that you can visualize and review them.

AWS CloudTrail is a service that logs all API calls, including console activities and command-line instructions. It logs exactly who did what, when, and from where. It can tell you which resources were acted upon in the API call and where the API call was made from and was made to. That means you have full visibility into any accesses, changes, or activity within your AWS environment. You can save these logs into your S3 buckets.

AWS Config is a fully managed service that provides you with a detailed inventory of your AWS resources and their current configuration in an AWS account. It continuously records configuration changes to these resources (e.g., EC2 instance launch, ingress/egress rules of security groups, network ACL rules for VPCs, etc.).

VPC Flow Logs captures information about the IP traffic going to and from network interfaces in your VPC. The flow log data is stored using Amazon CloudWatch Logs, which can then be integrated with additional services, such as Elasticsearch/Kibana for visualization.

AWS Trusted Advisor provides best practices (or checks) in five categories: cost optimization, security, fault tolerance, performance, and service limits.

AWS Organizations offers policy-based management for multiple AWS accounts. With Organizations, you can create groups of accounts and then apply policies to those groups. Organizations enables you to centrally manage policies across multiple accounts, without requiring custom scripts and manual processes.

Questions

[**1**](https://learning.oreilly.com/library/view/aws-certified-solutions/9781260470192/ch7.xhtml#ch7_1)**.**   What are the languages that AWS Lambda supports? (Choose two.)

**A.**   Perl

**B.**   Ruby

**C.**   Java

**D.**   Python

[**2**](https://learning.oreilly.com/library/view/aws-certified-solutions/9781260470192/ch7.xhtml#ch7_2)**.**   Which product is not a good fit if you want to run a job for ten hours?

**A.**   AWS Batch

**B.**   EC2

**C.**   Elastic Beanstalk

**D.**   Lambda

[**3**](https://learning.oreilly.com/library/view/aws-certified-solutions/9781260470192/ch7.xhtml#ch7_3)**.**   What product should you use if you want to process a lot of streaming data?

**A.**   Kinesis Firehouse

**B.**   Kinesis Data Stream

**C.**   Kinesis Data Analytics

**D.**   API Gateway

[**4**](https://learning.oreilly.com/library/view/aws-certified-solutions/9781260470192/ch7.xhtml#ch7_4)**.**   Which product should you choose if you want to have a solution for versioning your APIs without having the pain of managing the infrastructure?

**A.**   Install a version control system on EC2 servers

**B.**   Use Elastic Beanstalk

**C.**   Use API Gateway

**D.**   Use Kinesis Data Firehose

[**5**](https://learning.oreilly.com/library/view/aws-certified-solutions/9781260470192/ch7.xhtml#ch7_5)**.**   You want to transform the data while it is coming in. What is the easiest way of doing this?

**A.**   Use Kinesis Data Analytics

**B.**   Spin off an EMR cluster while the data is coming in

**C.**   Install Hadoop on EC2 servers to do the processing

**D.**   Transform the data in S3

[**6**](https://learning.oreilly.com/library/view/aws-certified-solutions/9781260470192/ch7.xhtml#ch7_6)**.**   Which product is not serverless?

**A.**   Redshift

**B.**   DynamoDB

**C.**   S3

**D.**   AWS Lambda

[**7**](https://learning.oreilly.com/library/view/aws-certified-solutions/9781260470192/ch7.xhtml#ch7_7)**.**   You have the requirement to ingest the data in real time. What product should you choose?

**A.**   Upload the data directly to S3

**B.**   Use S3 IA

**C.**   Use S3 reduced redundancy

**D.**   Use Kinesis Data Streams

[**8**](https://learning.oreilly.com/library/view/aws-certified-solutions/9781260470192/ch7.xhtml#ch7_8)**.**   You have a huge amount of data to be ingested. You don’t have a very stringent SLA for it. Which product should you use?

**A.**   Kinesis Data Streams

**B.**   Kinesis Data Firehose

**C.**   Kinesis Data Analytics

**D.**   S3

[**9**](https://learning.oreilly.com/library/view/aws-certified-solutions/9781260470192/ch7.xhtml#ch7_9)**.**   What is the best way to manage RESTful APIs?

**A.**   API Gateway

**B.**   EC2 servers

**C.**   Lambda

**D.**   AWS Batch

[**10**](https://learning.oreilly.com/library/view/aws-certified-solutions/9781260470192/ch7.xhtml#ch7_10)**.**   To execute code in AWS Lambda, what is the size of the EC2 instance you need to provision in the back end?

**A.**   For code running less than one minute, use a T2 Micro.

**B.**   For code running between one minute and three minutes, use M2.

**C.**   For code running between three minutes and five minutes, use M2 large.

**D.**   There is no need to provision an EC2 instance on the back end.

[**11**](https://learning.oreilly.com/library/view/aws-certified-solutions/9781260470192/ch7.xhtml#ch7_11)**.**   What are the two configuration management services that AWS OpsWorks supports? (Choose two.)

**A.**   Chef

**B.**   Ansible

**C.**   Puppet

**D.**   Java

[**12**](https://learning.oreilly.com/library/view/aws-certified-solutions/9781260470192/ch7.xhtml#ch7_12)**.**   You are designing an e-commerce order management web site where your users can order different types of goods. You want to decouple the architecture and would like to separate the ordering process from shipping. Depending on the shipping priority, you want to have a separate queue running for standard shipping versus priority shipping. Which AWS service would you consider for this?

**A.**   AWS CloudWatch

**B.**   AWS CloudWatch Events

**C.**   AWS API Gateway

**D.**   AWS SQS

[**13**](https://learning.oreilly.com/library/view/aws-certified-solutions/9781260470192/ch7.xhtml#ch7_13)**.**   Your company has more than 20 business units, and each business unit has its own account in AWS. Which AWS service would you choose to manage the billing across all the different AWS accounts?

**A.**   AWS Organizations

**B.**   AWS Trusted Advisor

**C.**   AWS Cost Advisor

**D.**   AWS Billing Console

[**14**](https://learning.oreilly.com/library/view/aws-certified-solutions/9781260470192/ch7.xhtml#ch7_14)**.**   You are running a job in an EMR cluster, and the job is running for a long period of time. You want to add additional horsepower to your cluster, and at the same time you want to make sure it is cost-effective. What is the best way of solving this problem?

**A.**   Add more on-demand EC2 instances for your task node

**B.**   Add more on-demand EC2 instances for your core node

**C.**   Add more spot instances for your task node

**D.**   Add more reserved instances for your task node

[**15**](https://learning.oreilly.com/library/view/aws-certified-solutions/9781260470192/ch7.xhtml#ch7_15)**.**   Your resources were running fine in AWS, and all of a sudden you notice that something has changed. Your cloud security team told you that some API has changed the state of your resources that were running fine earlier. How do you track who has created the mistake?

**A.**   By writing a Lambda function, you can find who has changed what

**B.**   By using AWS CloudTrail

**C.**   By using Amazon CloudWatch Events

**D.**   By using AWS Trusted Advisor

[**16**](https://learning.oreilly.com/library/view/aws-certified-solutions/9781260470192/ch7.xhtml#ch7_16)**.**   You are running a mission-critical three-tier application on AWS and have enabled Amazon CloudWatch metrics for a one-minute data point. How far back you can go and see the metrics?

**A.**   One week

**B.**   24 hours

**C.**   One month

**D.**   15 days

[**17**](https://learning.oreilly.com/library/view/aws-certified-solutions/9781260470192/ch7.xhtml#ch7_17)**.**   You are running all your AWS resources in the US-East region, and you are not leveraging a second region using AWS. However, you want to keep your infrastructure as code so that you should be able to fail over to a different region if any DR happens. Which AWS service will you choose to provision the resources in a second region that looks identical to your resources in the US-East region?

**A.**   Amazon EC2, VPC, and RDS

**B.**   Elastic Beanstalk

**C.**   OpsWorks

**D.**   CloudFormation

[**18**](https://learning.oreilly.com/library/view/aws-certified-solutions/9781260470192/ch7.xhtml#ch7_18)**.**   What is the AWS service you are going to use to monitor the service limit of your EC2 instance?

**A.**   EC2 dashboard

**B.**   AWS Trusted Advisor

**C.**   AWS CloudWatch

**D.**   AWS Config

[**19**](https://learning.oreilly.com/library/view/aws-certified-solutions/9781260470192/ch7.xhtml#ch7_19)**.**   You are a developer and want to deploy your application in AWS. You don’t have an infrastructure background and are not sure about how to use infrastructure within AWS. You are looking for deploying your application in such a way that the infrastructure scales on its own, and at the same time you don’t have to deal with managing it. Which AWS service are you going to choose for this?

**A.**   AWS Config

**B.**   AWS Lambda

**C.**   AWS Elastic Beanstalk

**D.**   Amazon EC2 servers and Auto Scaling

[**20**](https://learning.oreilly.com/library/view/aws-certified-solutions/9781260470192/ch7.xhtml#ch7_20)**.**   In the past, someone made some changes to your security group, and as a result an instance is not accessible by the users for some time. This resulted in nasty downtime for the application. You are looking to find out what change has been made in the system, and you want to track it. Which AWS service are you going to use for this?

**A.**   AWS Config

**B.**   Amazon CloudWatch

**C.**   AWS CloudTrail

**D.**   AWS Trusted Advisor

Answers

[**1**](https://learning.oreilly.com/library/view/aws-certified-solutions/9781260470192/ch7.xhtml#rch7_1)**.**   **C, D.** Perl and Ruby are not supported by Lambda.

[**2**](https://learning.oreilly.com/library/view/aws-certified-solutions/9781260470192/ch7.xhtml#rch7_2)**.**   **D.** Lambda is not a good fit because the maximum execution time for code in Lambda is five minutes. Using Batch you can run your code for as long as you want. Similarly, you can run your code for as long as you want on EC2 servers or by using Elastic Beanstalk.

[**3**](https://learning.oreilly.com/library/view/aws-certified-solutions/9781260470192/ch7.xhtml#rch7_3)**.**   **B.** Kinesis Data Firehose is used mainly for large amounts of nonstreaming data, Kinesis Data Analytics is used for transforming data, and API Gateway is used for managing APIs.

[**4**](https://learning.oreilly.com/library/view/aws-certified-solutions/9781260470192/ch7.xhtml#rch7_4)**.**   **C.** EC2 servers and Elastic Beanstalk both need you to manage some infrastructure; Kinesis Data Firehose is used for ingesting data.

[**5**](https://learning.oreilly.com/library/view/aws-certified-solutions/9781260470192/ch7.xhtml#rch7_5)**.**   **A.** Using EC2 servers or Amazon EMR, you can transform the data, but that is not the easiest way to do it. S3 is just the data store; it does not have any transformation capabilities.

[**6**](https://learning.oreilly.com/library/view/aws-certified-solutions/9781260470192/ch7.xhtml#rch7_6)**.**   **A.** DynamoDB, S3, and AWS Lambda all are serverless.

[**7**](https://learning.oreilly.com/library/view/aws-certified-solutions/9781260470192/ch7.xhtml#rch7_7)**.**   **D.** You can use S3 for storing the data, but if the requirement is to ingest the data in real time, S3 is not the right solution.

[**8**](https://learning.oreilly.com/library/view/aws-certified-solutions/9781260470192/ch7.xhtml#rch7_8)**.**   **B.** Kinesis Data Streams is used for ingesting real-time data, and Kinesis Data Analytics is used for transformation. S3 is used to store the data.

[**9**](https://learning.oreilly.com/library/view/aws-certified-solutions/9781260470192/ch7.xhtml#rch7_9)**.**   **A.** Theoretically EC2 servers can be used for managing the APIs, but if you can do it easily through API Gateway, why would you even consider EC2 servers? Lambda and Batch are used for executing the code.

[**10**](https://learning.oreilly.com/library/view/aws-certified-solutions/9781260470192/ch7.xhtml#rch7_10)**.**   **D.** There is no need to provision EC2 servers since Lambda is serverless.

[**11**](https://learning.oreilly.com/library/view/aws-certified-solutions/9781260470192/ch7.xhtml#rch7_11)**.**   **A, C.** AWS OpsWorks supports Chef and Puppet.

[**12**](https://learning.oreilly.com/library/view/aws-certified-solutions/9781260470192/ch7.xhtml#rch7_12)**.**   **D.** Using SQS, you can decouple the ordering and shipping processes, and you can create separate queues for the ordering and shipping processes.

[**13**](https://learning.oreilly.com/library/view/aws-certified-solutions/9781260470192/ch7.xhtml#rch7_13)**.**   **A.** Using AWS Organizations, you can manage the billing from various AWS accounts.

[**14**](https://learning.oreilly.com/library/view/aws-certified-solutions/9781260470192/ch7.xhtml#rch7_14)**.**   **C.** You can add more spot instances to your task node to finish the job early. Spot instances are the cheapest in cost, so this will make sure the solution is cost-effective.

[**15**](https://learning.oreilly.com/library/view/aws-certified-solutions/9781260470192/ch7.xhtml#rch7_15)**.**   **B.** Using AWS CloudTrail, you can find out who has changed what via API.

[**16**](https://learning.oreilly.com/library/view/aws-certified-solutions/9781260470192/ch7.xhtml#rch7_16)**.**   **D.** When CloudWatch is enabled for a one-minute data point, the retention is 15 days.

[**17**](https://learning.oreilly.com/library/view/aws-certified-solutions/9781260470192/ch7.xhtml#rch7_17)**.**   **D.** Using CloudFormation, you can keep the infrastructure as code, and you can create a CloudFormation template to mimic the setup in an existing region and can deploy the CloudFormation template in a different region to create the resources.

[**18**](https://learning.oreilly.com/library/view/aws-certified-solutions/9781260470192/ch7.xhtml#rch7_18)**.**   **B.** Using Trusted Advisor, you can monitor the service limits for the EC2 instance.

[**19**](https://learning.oreilly.com/library/view/aws-certified-solutions/9781260470192/ch7.xhtml#rch7_19)**.**   **C.** AWS Elastic Beanstalk is an easy-to-use service for deploying and scaling web applications. You can simply upload your code and Elastic Beanstalk automatically handles the deployment, from capacity provisioning, load balancing, and auto-scaling to application health monitoring.

[**20**](https://learning.oreilly.com/library/view/aws-certified-solutions/9781260470192/ch7.xhtml#rch7_20)**.**   **A.** AWS Config maintains the configuration of the system and helps you to identify what change was made in it.