CHAPTER 2

# Data Science Use Cases

In this chapter, we show how AI and machine learning have disrupted nearly every industry—and will continue to do so in the future. We discuss prominent use cases across industries such as media, advertising, IoT, and manufacturing. As more and more building blocks become available, more and more use cases become tangible. Cloud-native developers have access to these building blocks through ready-to-use AI services such as Amazon Rekognition, fully customizable ML services including Amazon SageMaker, and easy-to-access quantum computers with Amazon Braket.

AI and machine learning have become truly ubiquitous thanks to recent innovations in cloud computing, leaps in computing power, and explosions in data collection. This democratization of AI and machine learning is fueled by an explosion of AI services that are easy to integrate with applications, require very little maintenance, and offer pay-as-you-go pricing.

With no required PhD in data science, we can implement product recommendations to delight our customers, implement highly accurate forecasting models to improve our supply chain, or build virtual assistants to simplify our customer support—all with just a single API call! These AI services free up valuable human resources to focus on domain-specific and product-differentiating features.

## Innovation Across Every Industry

Many AI and machine learning use cases fall in one of two categories: improving business operations or creating new customer experiences. Prominent examples that improve business operations are AI-powered demand forecasting, resource optimization, and fraud detection. Examples for creating new customer experiences include personalized product recommendations and enriched video-streaming experiences.

Without a doubt, AI and machine learning are driving innovation in every industry. Here are a few examples across various industries:

*Media and entertainment*
: Companies are delighting customers with highly engaging, personalized content. AI also enables highly efficient and effective metadata extraction to make media content more easily discoverable and searchable by customers and media production workers.

*Life sciences*
: Companies benefit from AI and machine learning for drug discovery, clinical trial management, drug manufacturing, digital therapeutics development, and clinical decision support.

*Financial services*
: AI and machine learning improve compliance, surveillance, and fraud detection. They help to speed up document processing, create personalized pricing and financial product recommendations, and assist in trading decisions.

*Automotive*
: AI and machine learning power autonomous driving, navigation, and connected vehicles.

*Manufacturing*
: AI and machine learning support engineering design, manage supply chain inventory, and optimize maintenance, repair, and operations. They enhance the assembly line and underpin smart products or factories.

*Gaming*
: The gaming industry leverages AI and machine learning to implement intelligent auto-scaling for their game servers as demand changes throughout the day.

Let's discuss some prominent AI use cases in more detail and see how we can start to implement them with ready-to-use AI services from AWS.

## Personalized Product Recommendations

Over recent decades, consumers have experienced more and more personalized online product and content recommendations. Recommendations are everywhere, including Amazon.com suggesting the next product to buy and Amazon Prime Video recommending the next show to watch.

A lot of recommendation systems find similarities based on how customers collaborate with items in the catalog. An early implementation of such "collaborative filtering" is described in a 2003 paper by Amazon.com, "Amazon.com Recommendations: Item-to-Item Collaborative Filtering".

Today, sophisticated deep learning techniques understand customers' needs at the right time and within the right context. Whether we are shopping on the Amazon.com marketplace, listening to music on Prime Music, watching shows on Prime Video, reading ebooks on the Amazon Kindle, or listening to audiobooks with Audible, we will be presented with new personalized recommendations.

Simple recommendation systems often start as rule-based systems. As the number of users and products increase in our system, it becomes difficult to define rules that are specific enough to each user to provide meaningful recommendations. Broad rules are not typically specialized enough to keep customers coming back.

Even ML-based recommendation systems may face challenges. We also have to handle new users and new items entering our system where we don't have any data on which to base our recommendations. This is the classic "cold start" problem and should be addressed by anyone implementing a recommendation engine in the modern era. Cold start is when we have little or no historical event activity to use as a signal to build our recommender model for a given user or product.

Recommendations should also avoid the "popularity trap," which only recommends popular items and potentially misses delightful recommendations for nonpopular items. This can be solved with recommender systems that explore new items using algorithms such as multiarmed bandits, which we will discuss in Chapter 9.

Also, we'd like to handle real-time changes in a user's intent as they are using our application. This requires a real-time, dynamic recommendation system instead of traditional offline, precomputed recommendations served from a database.

With such a dynamic system, customers will appreciate the relevant and timely content—and our business will realize the following benefits from a more personalized customer experience:

*Increase in product engagement*
: By recommending relevant content to users, we increase the stickiness of our website, encourage users to come back often, and give them reason to stay longer.

*Increase in product conversions*
: Users are more likely to purchase more relevant products.

*Increase in click-through rates*
: We will likely see higher click-through rates with personalized product updates targeted at the individual user.

*Increase in revenue*
: When customers are served the right recommendations at the right time, companies see an increase in revenue.

*Reduction in churn*
: We can reduce overall churn and reduce opt-outs from interesting email campaigns.

Over the last two decades, Amazon has continuously advanced its machine learning research focused on personalization. The paper "Two Decades of Recommender Systems at Amazon.com" by Smith and Linden (2017) provides a great summary of this journey.

More insights into Amazon's scientific research and academic publications are available on Amazon.

## Recommend Products with Amazon Personalize

Like a lot of machine learning, there is no single algorithm that addresses all those challenges in personalization. Wouldn't it be great if anyone could just tap into Amazon.com's extensive experience in creating personalized product and content recommendations and add this capability to our applications? Amazon Personalize offers exactly this.

Amazon Personalize reflects Amazon.com's decades of experience in creating, scaling, and managing personalization technology. Amazon Personalize makes it easy for developers to create individualized product recommendations as well as targeted marketing promotions. This AI service enables developers to build custom personalization models without having to deal with the complexity of managing our own machine learning infrastructure.

To start generating recommendations, we just provide Amazon Personalize with the continuous activity stream from our application (i.e., clicks, page views, signups, purchases) along with the inventory of the products we want to recommend, as shown in Figure 2-1.

The activity data comprises event information about how the user interacts with the system. Some example event activity includes user clicks, shopping cart additions, item purchases, and movie watches. This event activity represents a strong signal to build an effective recommendation model.

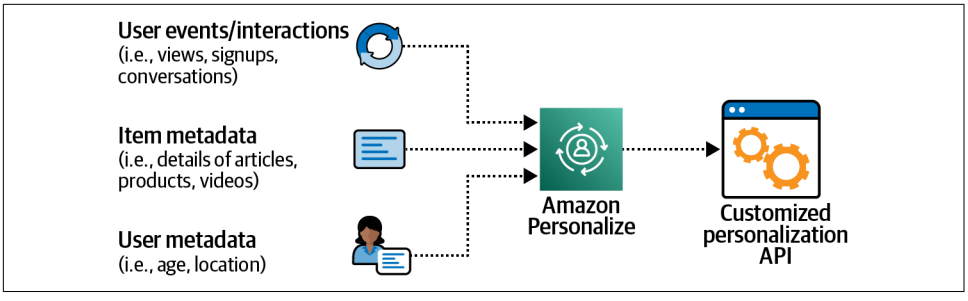


*Figure 2-1. Provide the activity dataset and inventory of products to Amazon Personalize to start generating recommendations.*

We can also provide additional metadata about the users and products involved in the event activity, such as the product category, product price, user age, user location, etc. While this additional metadata is optional, it is helpful when addressing the "cold start" scenario where we have little or no historical event activity to use as a signal to build our recommender model.

Additionally, Amazon Personalize has recently announced a new cold-start algorithm that combines neural networks and reinforcement learning to provide more relevant recommendations when little data is known about the user.

With this event activity and metadata in place, Amazon Personalize trains, tunes, and deploys a custom recommender model for our users and products. Amazon Personalize performs all steps of the machine learning pipeline, including feature engineering, algorithm selection, model tuning, and model deploying. Once Amazon Personalize selects, trains, and deploys the best model for our dataset, we simply call the Amazon Personalize `get_recommendations()` API to generate recommendations in real time for our users:

```
get_recommendations_response = personalize_runtime.get_recommendations(
        campaignArn = campaign_arn,
        userId = user_id
)

item_list = get_recommendations_response['itemList']
recommendation_list = []
for item in item_list:
    item_id = get_movie_by_id(item['itemId'])
recommendation_list.append(item_id)
```

Trained with the popular MovieLens dataset that contains millions of user-movie ratings, Amazon Personalize generates the following recommended movies for our sample user:

| Shrek |
| --- |
| Amelie |
| Lord of the Rings: The Two Towers |
| Toy Story 2 |
| Good Will Hunting |
| Eternal Sunshine of the Spotless Mind |
| Spirited Away |
| Lord of the Rings: The Return of the King |
| Schindler's List |
| Leon: The Professional |

## Generate Recommendations with Amazon SageMaker and TensorFlow

Multitask recommenders create a model that optimizes two or more objectives at the same time. The model performs transfer learning by sharing variables between the tasks during model training.

In the following TensorFlow example that uses the TensorFlow Recommenders (TFRS) library, we will find a model that trains a recommender to predict ratings (ranking task) as well as predict the number of movie watches (retrieval task):

```
user_model = tf.keras.Sequential([
  tf.keras.layers.experimental.preprocessing.StringLookup(
      vocabulary=unique_user_ids),
  # We add 2 to account for unknown and mask tokens.
  tf.keras.layers.Embedding(len(unique_user_ids) + 2, embedding_dimension)
])

movie_model = tf.keras.Sequential([
  tf.keras.layers.experimental.preprocessing.StringLookup(
      vocabulary=unique_movie_titles),
  tf.keras.layers.Embedding(len(unique_movie_titles) + 2, embedding_dimension)
])

rating_task = tfrs.tasks.Ranking(
    loss=tf.keras.losses.MeanSquaredError(),
    metrics=[tf.keras.metrics.RootMeanSquaredError()],
)

retrieval_task = tfrs.tasks.Retrieval(
        metrics=tfrs.metrics.FactorizedTopK(
            candidates=movies.batch(128).map(self.movie_model)
        )
)
```

## Generate Recommendations with Amazon SageMaker and Apache Spark

Amazon SageMaker supports serverless Apache Spark (both Python and Scala) through SageMaker Processing Jobs. We will use SageMaker Processing Jobs throughout the book to perform data-quality checks and feature transformations. However, in this section we will generate recommendations using SageMaker Processing Jobs with Apache Spark ML's collaborative filtering algorithm called *Alternating Least Squares* (ALS). We would use this algorithm if we already have a Spark-based data pipeline and want to generate recommendations using that pipeline.

Here is the *train_spark.py* file that generates recommendations with Apache Spark ML and ALS:

```
import pyspark
from pyspark.sql import SparkSession
from pyspark.sql.functions import *
from pyspark.ml.evaluation import RegressionEvaluator
from pyspark.ml.recommendation import ALS
from pyspark.sql import Row

def main():
    ...
    lines = spark.read.text(s3_input_data).rdd
    parts = lines.map(lambda row: row.value.split("::"))
    ratingsRDD = parts.map(lambda p: Row(userId=int(p[0]),
                                         movieId=int(p[1]),
                                         rating=float(p[2]),
                                         timestamp=int(p[3])))
    ratings = spark.createDataFrame(ratingsRDD)
    (training, test) = ratings.randomSplit([0.8, 0.2])

    # Build the recommendation model using ALS on the training data
    als = ALS(maxIter=5,
              regParam=0.01,
              userCol="userId",
              itemCol="itemId",
              ratingCol="rating",
              coldStartStrategy="drop")
    model = als.fit(training)

    # Evaluate the model by computing the RMSE on the test data
    predictions = model.transform(test)
    evaluator = RegressionEvaluator(metricName="rmse",
                                    labelCol="rating",
                                    predictionCol="prediction")
    rmse = evaluator.evaluate(predictions)

    # Generate top 10 recommendations for each user
    userRecs = model.recommendForAllUsers(10)
    userRecs.show()
```

```
    # Write top 10 recommendations for each user
    userRecs.repartition(1).write.mode("overwrite")\
    .option("header", True).option("delimiter", "\t")\
    .csv(f"{s3_output_data}/recommendations")
```

Now let's launch the PySpark script within a serverless Apache Spark environment running as a SageMaker Processing Job:

```
from sagemaker.spark.processing import PySparkProcessor
from sagemaker.processing import ProcessingOutput

processor = PySparkProcessor(base_job_name='spark-als',
                             role=role,
                             instance_count=1,
                             instance_type='ml.r5.2xlarge',
                             max_runtime_in_seconds=1200)


processor.run(submit_app='train_spark_als.py',
              arguments=['s3_input_data', s3_input_data,
                         's3_output_data', s3_output_data,
              ],
              logs=True,
              wait=False
)
```

The output shows a user ID and list of recommendations (item ID, rank) sorted by rank from most recommended to least recommended:

```
|userId|     recommendations|
+------+--------------------+
|    12|[[46, 6.146928], ...|
|     1|[[46, 4.963598], ...|
|     6|[[25, 4.5243497],...|
+------+--------------------+
```

# Detect Inappropriate Videos with Amazon Rekognition

Computer vision is useful for many different use cases, including content moderation of user-generated content, digital identity verification for secure login, and hazard identification for driverless cars.

Amazon Rekognition is a high-level AI service that identifies objects, including people, text, and activities found in both images and videos. Amazon Rekognition uses AutoML to train custom models to recognize objects specific to our use case and business domain.

Let's use Amazon Rekognition to detect violent content in videos uploaded by our application users. For example, we want to reject videos that contain weapons by using Amazon Rekognition's Content Moderation API as follows:

```
startModerationLabelDetection = rekognition.start_content_moderation(
    Video={
        'S3Object': {
            'Bucket': bucket,
            'Name': videoName,
        }
    },
)

moderationJobId = startModerationLabelDetection['JobId']

getContentModeration = rekognition.get_content_moderation(
    JobId=moderationJobId,
    SortBy='TIMESTAMP'
)

while(getContentModeration['JobStatus'] == 'IN_PROGRESS'):
    time.sleep(5)
    print('.', end='')

    getContentModeration = rekognition.get_content_moderation(
     JobId=moderationJobId,
     SortBy='TIMESTAMP')

display(getContentModeration['JobStatus'])
```

Here is the output that shows the detected labels. Note the `Timestamp`, which represents the offset from the start of the video, as well as the `Confidence`, which represents the confidence of Amazon Rekognition's prediction:

```
{'JobStatus': 'SUCCEEDED',
 'VideoMetadata': {'Codec': 'h264',
  'DurationMillis': 6033,
  'Format': 'QuickTime / MOV',
  'FrameRate': 30.0,
  'FrameHeight': 1080,
  'FrameWidth': 1920},
 'ModerationLabels': [{'Timestamp': 1999,
   'ModerationLabel': {'Confidence': 75.15272521972656,
    'Name': 'Violence',
    'ParentName': ''}},
  {'Timestamp': 1999,
   'ModerationLabel': {'Confidence': 75.15272521972656,
    'Name': 'Weapons',
    'ParentName': 'Violence'}},
  {'Timestamp': 2500,
   'ModerationLabel': {'Confidence': 87.55487060546875,
    'Name': 'Violence',
    'ParentName': ''}}]

Moderation labels in video
==========================================
```

```
At 1999 ms: Violence (Confidence: 75.15)
At 1999 ms: Weapons (Confidence: 75.15)
At 2500 ms: Violence (Confidence: 87.55)
```

We can further improve the confidence of Amazon Rekognition's predictions by training with our own dataset. This feature is called "custom labels"; it's supported by many of the Amazon AI services.

## Demand Forecasting

Demand forecasting is used across many domains to predict demand in areas such as electricity usage, supply-chain inventory, call-center staffing, cash-flow planning, hospital-bed usage, and many other use cases. Forecasting is a time-series problem solved with popular algorithms such as Auto-Regressive Integrated Moving Average, Error Trend Seasonality, Non-Parametric Time Series, Prophet, and DeepAR++.

Businesses use everything from simple spreadsheets to complex time-series software to forecast future business outcomes such as product demand, resource needs, or financial performance. These approaches typically build forecast models using historical time-series data with the assumption that future demand is determined by past activity. These purely time-series approaches have a difficult time producing accurate forecasts with irregular trends and patterns. Also, these approaches often do not factor in important metadata that is relevant to the forecast, such as product price, product category, and store location.

Overforecasting can decrease efficiency and increase cost by overprovisioning resources that remain underutilized. Underforecasting can decrease customer satisfaction, lower revenue by starving the system of necessary resources, and require high-cost alternatives like paying overtime labor wages.

An effective demand forecast system demonstrates the following characteristics:

*Analyze complex relationships rather than just time-series data*
: Combines time-series data with other metadata like product features and store locations.

*Reduce forecast prediction time from months to hours*
: After automatically loading, inspecting, and identifying key dataset attributes, the system will quickly train, optimize, and deploy a custom model that fits our dataset.

*Create forecasts for many different use cases*
: Build forecasts for virtually every use case, including supply chain, logistics, and finance, using a large library of algorithms to automatically determine the best fit for our specific use case.

*Keep data secure*
: Every data point is protected with at-rest and in-flight encryption to keep sensitive information secure and confidential.

*Automatically retrain and redeploy models when needed*
: When new data arrives—or when model evaluation metrics fall below a certain threshold—the model will be retrained and redeployed to improve the forecast predictions.

## Predict Energy Consumption with Amazon Forecast

Amazon Forecast is a fully managed service based on the technology that powers Amazon.com's demand forecasting needs, such as efficient inventory management, immediate product fulfillment, and same-day delivery. Forecast uses machine learning to automatically train, tune, and optimize highly specialized demand forecast models from our dataset. We simply register our historical datasets—and related metadata—with Forecast to start generating demand predictions. Demand forecasts can be exported as CSVs, accessed via the AWS Console UI, or integrated into our application through the Forecast API.

Let's train a demand forecast model to predict individual household energy power consumption for the next 24 hours using Forecast's DeepAR++ algorithm and a public dataset from the UCI Machine Learning Repository.

Here is a snippet of the dataset, including the power consumption per customer:

| timestamp | value | item |
|---|---|---|
| 2014-01-01 01:00:00 | 38.34991708126038 | client_12 |
| 2014-01-01 02:00:00 | 33.5820895522388 | client_12 |
| 2014-01-01 03:00:00 | 34.41127694859037 | client_12 |

Here is the schema that we define in Forecast to represent the public dataset:

```
forecast_schema ={
   "Attributes":[
      {
         "AttributeName":"timestamp",
         "AttributeType":"timestamp"
      },
      {
         "AttributeName":"target_value",
         "AttributeType":"float"
```

```
        },
        {
            "AttributeName":"item_id",
            "AttributeType":"string"
        }
    ]
}
```

Let's register the dataset with Forecast:

```
response=forecast.create_dataset(
                    Domain="CUSTOM",
                    DatasetType='TARGET_TIME_SERIES',
                    DatasetName=forecast_dataset_name,
                    DataFrequency=DATASET_FREQUENCY,
                    Schema = forecast_schema
)
```

Now let's train the demand forecast model with Forecast:

```
forecast_horizon = 24 # hours

algorithm_arn = 'arn:aws:forecast:::algorithm/Deep_AR_Plus'

create_predictor_response = \
    forecast.create_predictor(PredictorName=predictor_name,
                              AlgorithmArn=algorithm_arn,
                              ForecastHorizon=forecast_horizon,
                              PerformAutoML= False,
                              PerformHPO=False,
                              EvaluationParameters= {
                                  "NumberOfBacktestWindows": 1,
                                  "BackTestWindowOffset": 24
                              },
                              InputDataConfig= {
                                  "DatasetGroupArn": forecast_dataset_group_arn
                              },
                              FeaturizationConfig= {
                                  "ForecastFrequency": "H",
                                  "Featurizations": [{
                                      "AttributeName": "target_value",
                                      "FeaturizationPipeline":
                                      [{
                                           "FeaturizationMethodName": "filling",
                                           "FeaturizationMethodParameters": {
                                                "frontfill": "none",
                                                "middlefill": "zero",
                                                "backfill": "zero"
                                           }
                                      }]
                                  }]
                              })
```

Let's make a prediction:

```
forecastResponse = forecastquery.query_forecast(
    ForecastArn=forecast_arn,
    Filters={"item_id":"client_12"}
)
```

## Predict Demand for Amazon EC2 Instances with Amazon Forecast

AWS uses Forecast to predict demand for Amazon EC2 instances within Amazon Redshift clusters. As new data is ingested into Forecast, the Amazon Redshift control plane queries Forecast to adjust the size of the Amazon EC2 warm pool for Amazon Redshift, as shown in Figure 2-2.

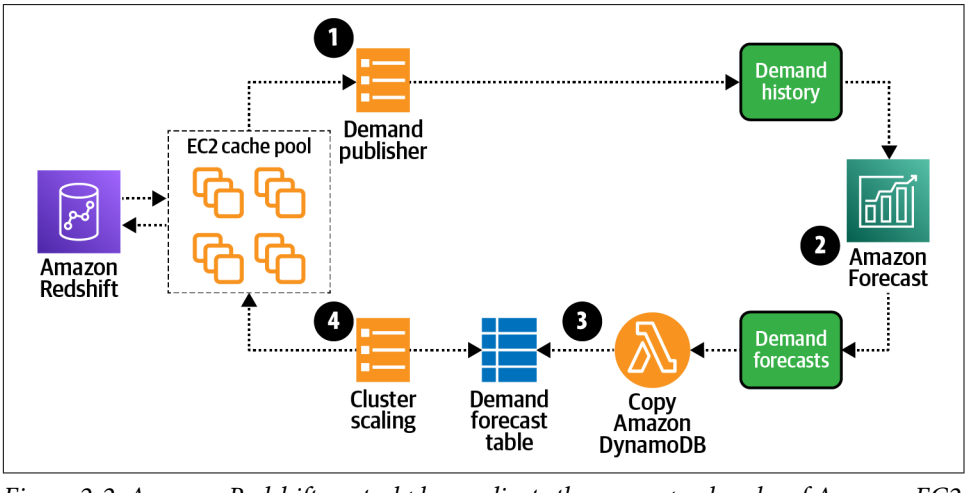


*Figure 2-2. Amazon Redshift control plane adjusts the warm-pool cache of Amazon EC2 instances using Forecast.*

The steps in Figure 2-2 can be described as follows:

1. Changes to Amazon EC2 warm-pool cache demand are published to S3.
2. Forecast ingests demand data from S3 and then creates new forecast predictions.
3. A Lambda function copies the new forecast predictions to Amazon DynamoDB.
4. The Amazon EC2 cluster scaler reads the forecast predictions from DynamoDB and adjusts the warm-pool cache size based on projected demand.

# Identify Fake Accounts with Amazon Fraud Detector

Each year, tens of billions of dollars are lost to online fraud across the world. Online companies are especially prone to bad-actor attacks trying to defraud the system by creating fake user accounts and purchasing items with stolen credit cards. Typical fraud detection systems that detect bad actors often rely on business rules that are slow to adapt to the latest fraud techniques.

Effective fraud detection and privacy-leak prevention systems include the following characteristics:

*Stop bad actors before they affect our business*
: Flag suspicious activity before bad actors can do real harm.

*High-quality fraud-detection models without a lot of data*
: Pre-trained algorithms can analyze even the smallest amount of historical event data and still provide a high-quality fraud-detection model.

*Let fraud teams move faster and with more control*
: Automatically handle the complex tasks required to build, train, tune, deploy, and update our fraud-detection models when new event data is available.

Amazon Fraud Detector is a fully managed service that identifies potentially fraudulent online activities such as online payments and fake accounts. Amazon Fraud Detector uses machine learning and 20 years of fraud-detection expertise from AWS and Amazon.com.

With Amazon Fraud Detector, we create a fraud-detection model with just a few clicks, a relatively small amount of historical data, and minimal code. We simply upload our historical online event data, such as online transactions and account registrations—and Amazon Fraud Detector does the rest, including training, tuning, and deploying a custom fraud-detection model for our business.

Here is the code that trains Amazon Fraud Detector on our transaction dataset:

```
response = client.create_model_version(
    modelId     = MODEL_NAME,
    modelType   = 'ONLINE_FRAUD_INSIGHTS',
    trainingDataSource = 'EXTERNAL_EVENTS',
    trainingDataSchema = trainingDataSchema,
    externalEventsDetail = {
        'dataLocation'     : S3_FILE_LOC,
        'dataAccessRoleArn': ARN_ROLE
    }
)
```

Here is the code to predict whether a given transaction is fraudulent:

```
pred = client.get_event_prediction(
    detectorId = DETECTOR_NAME,
    detectorVersionId = DETECTOR_VER,
    eventId = str(eventId),
    eventTypeName = EVENT_TYPE,
    eventTimestamp = timestampStr,
    entities = [{'entityType': ENTITY_TYPE,
    'entityId':str(eventId.int)}],
    eventVariables = record)

record["score"] = pred['modelScores'][0]['scores']\
                    ["{0}_insightscore".format(MODEL_NAME)]
```

Here is the output of the Amazon Fraud Detector prediction showing the relevant data, prediction outcome, and prediction confidence:

| ip_address | email_address | state | postal | name | phone_number | score | outcomes |
|---|---|---|---|---|---|---|---|
| 84.138.6.238 | synth1@yahoo.com | LA | 32733 | Brandon Moran | (555)784 - 5238 | 5.0 | [approve] |
| 194.147.250.63 | synth2@yahoo.com | MN | 34319 | Dominic Murray | (555)114 - 6133 | 4.0 | [approve] |
| 192.54.60.50 | synth3@gmail.com | WA | 32436 | Anthony Abbott | (555)780 - 7652 | 5.0 | [approve] |
| 169.120.193.154 | synth4@gmail.com | AL | 34399.0 | Kimberly Webb | (555)588 - 4426 | 938.0 | [review] |
| 192.175.55.43 | synth5@hotmail.com | IL | 33690.0 | Renee James | (555)785 - 8274 | 16.0 | [approve] |

# Enable Privacy-Leak Detection with Amazon Macie

A well-instrumented application produces many logs and metrics to increase insight and maintain high system uptime to avoid customer dissatisfaction. However, sometimes these logs contain sensitive account information such as home zip codes or credit card numbers. We need a system that monitors our data for sensitive information, detects access to this sensitive information, and sends notifications if unauthorized access is detected or data is leaked.

Effective systems to detect and monitor access to sensitive information have the following characteristics:

*Continuously assess data sensitivity and evaluate access controls*
: Attacker ROI math dictates that an S3 bucket with sensitive customer data and loosely configured IAM roles is an easy target. We stay ahead of this by continuously monitoring our entire S3 environment and generating actionable steps to respond quickly when needed.

*Support many data sources*
: Assess data sensitivity and evaluate access controls across many different data sources, such as S3, Amazon Relational Database Service (Amazon RDS), Amazon Aurora, emails, file shares, collaboration tools, etc.

*Maintain regulatory compliance*
: In addition to monitoring and protecting sensitive data, compliance teams are required to provide evidence that they are enforcing data security and privacy to meet regulatory compliance requirements.

*Identifying sensitive data during data migrations*
: When migrating large volumes of data into AWS, we want to know if the data includes sensitive data. If so, we likely need to update the security access controls, encryption settings, and resource tags when the data is migrated.

Amazon Macie is a fully managed security service that uses machine learning to identify sensitive data like personally identifiable information in our AWS-based data sources, such as S3. Macie provides visibility into where this data is stored—and who is accessing the data. By monitoring access to sensitive data, Macie can send an alert when it detects a leak—or risk of a leak.

Macie continuously identifies sensitive data and evaluates the security and access controls to this data. Macie helps maintain data privacy and security across all of our data and provides comprehensive options for scheduling our data-sensitivity and access-control analysis to maintain our data privacy and compliance requirements.

We can schedule daily, weekly, or monthly discovery jobs to generate our findings, including evaluation results, time stamps, and historical records of all buckets and objects scanned for sensitive data. These findings are summarized in a standard report compliant with data privacy and protection audits to ensure long-term data retention. For data migrations, Macie automates the configuration of data protection and role-based access policies as our data moves into AWS.

# Conversational Devices and Voice Assistants

Whether it's Alexa or any of those other famous home voices, all of them use state-of-the-art deep learning technologies in the field of automatic speech recognition (ASR) and natural language understanding (NLU) to recognize the intent of our spoken text.

## Speech Recognition with Amazon Lex

Using Amazon Lex to build conversational interfaces for voice and text, we have access to the same deep learning technologies that power Amazon Alexa. Amazon Lex is a fully managed service that uses ASR to convert speech to text. Amazon Lex also uses NLU to recognize the intent of the text. We can build custom responses to a wide set of voice and text queries, such as, "Where is the IT help desk in this office?" and "Reserve this room for the next 30 minutes."

## Text-to-Speech Conversion with Amazon Polly

Amazon Polly is an automated text-to-speech service with dozens of human voices across a broad set of languages, dialects, and genders. We can use Amazon Polly to build speech-enabled applications that turn text into human-like speech for accessibility purposes, for example.

## Speech-to-Text Conversion with Amazon Transcribe

Amazon Transcribe is an ASR service that makes it easy for developers to add speech-to-text capability to their real-time and batch applications. Amazon Transcribe converts speech to text by processing audio either in batch or real time. Popular use cases for Amazon Transcribe include creating image captions and video subtitles.

# Text Analysis and Natural Language Processing

Natural language processing (NLP) is a field of artificial intelligence that focuses on machines' ability to read, understand, and derive meaning from human languages. It is a field that has been studied for a very long time, with research publications dating back to the early 1900s.

Fast-forwarding to 2021, we still experience ground-breaking NLP research with new language models emerging nearly on a monthly basis. In later chapters, we will discuss the evolution of NLP algorithms, discuss the novel Transformer neural-network architecture, and dive deeper into the BERT family of NLP algorithms.

Effective text analysis and cognitive search systems have the following characteristics:

*Fast time to discover*
: New documents should become searchable quickly and without errors that require human correction.

*Efficient processing workflows*
: Document-processing workflows should be automated to increase speed and quality while reducing human effort, custom code, and cost.

## Translate Languages with Amazon Translate

In today's global economy, we need to appeal to international users by translating our content into many localized, region-specific, multilingual versions. Popular use cases include on-demand translation of user-generated content, real-time translations for communication apps, and multilingual sentiment analysis of social media content.

Amazon Translate is a neural machine translation service that creates more accurate and fluent translations than traditional statistical and rule-based translation models.

## Classify Customer-Support Messages with Amazon Comprehend

Customer Obsession is one of Amazon's key leadership principles—customer focus is important for every business and industry. And in many cases, a customer's experience is largely influenced by the quality of customer support. In this section, we will use Amazon Comprehend to classify the sentiment of sample customer-support messages.

Text classification is a popular task in the field of NLP. As described earlier, we can use Amazon Comprehend as a fully managed NLP service to implement text classification without much machine learning experience.

More broadly, Amazon Comprehend can recognize important entities, key phrases, sentiment, language, and topics from our text documents. Important entities include names, places, items, and dates. Key phrases include "good morning," "thank you," and "not happy." Sentiment includes "positive," "neutral," and "negative." Amazon Comprehend currently supports many languages, and new languages are added frequently.

Amazon Comprehend also supports a set of healthcare APIs called *Amazon Comprehend Medical*. Amazon Comprehend Medical has been pre-trained on extensive healthcare datasets and can identify medical conditions, medications, tests, treatments, procedures, anatomy, and protected health information.

Let's take a look at how we can use Amazon Comprehend's out-of-the-box Sentiment Analysis API to classify sample product reviews with just a few lines of code.

First, let's use Amazon Comprehend's `create_document_classifier()` API to create the classifier:

```
training_job = comprehend.create_document_classifier(
    DocumentClassifierName=comprehend_training_job_name,
    DataAccessRoleArn=iam_role_comprehend_arn,
    InputDataConfig={
        'S3Uri': comprehend_train_s3_uri
    },
    OutputDataConfig={
        'S3Uri': s3_output_job
    },
    LanguageCode='en'
)
```

Next, let's use the classifier to predict the sentiment of a sample *positive* review with Amazon Comprehend's `detect_sentiment()` API:

```
txt = """I loved it!  I will recommend this to everyone."""
```

```
response = comprehend.detect_sentiment(
    Text=txt
)
```

Here is the output:

```
{
    "SentimentScore": {
        "Mixed": 0.030585512690246105,
        "Positive": 0.94992071056365967,
        "Neutral": 0.0141543131828308,
        "Negative": 0.00893945890665054
    },
    "Sentiment": "POSITIVE",
    "LanguageCode": "en"
}
```

Next, let's use the classifier to predict the sentiment of a sample *negative* review with Amazon Comprehend's `detect_sentiment()` API:

```
txt = """Really bad.  I hope they don't make this anymore."""

response = comprehend.detect_sentiment(
    Text=txt
)
```

Here is the output for the *negative* review:

```
{
    "SentimentScore": {
        "Mixed": 0.030585512690246105,
        "Positive": 0.00893945890665054,
        "Neutral": 0.0141543131828308,
        "Negative": 0.94992071056365967
    },
    "Sentiment": "NEGATIVE",
    "LanguageCode": "en"
}
```

With Amazon Comprehend Custom Labels, we can train Amazon Comprehend to predict custom labels specific to our dataset.

In Chapter 3, we will train a custom Amazon Comprehend model that classifies support messages into a star rating (1–5) as a finer-grained form of sentiment analysis. We will use the Amazon Customer Reviews Dataset.

## Extract Resume Details with Amazon Textract and Comprehend

Organizations have long struggled to process semistructured documents efficiently to make them easy to index and search. Document processing usually requires significant customization and configuration. Amazon Textract, a fully managed service to accurately extract text from a document, uses optical character recognition (OCR) and machine learning to automatically extract information from our scanned documents.

More than just OCR, Amazon Textract also uses NLP to parse and save the specific words, phrases, dates, and numbers found in the document. Combined with Amazon Comprehend, Amazon Textract can build and maintain a smart index of our document contents. We can also use Amazon Textract to build automated document processing workflows and maintain compliance for our document archives.

After scanning and parsing a PDF resume, Amazon Textract generates this text-based version:

```
NAME
...
LOCATION
...
WORK EXPERIENCE
...
EDUCATION
...
SKILLS
C (Less than 1 year), Database (Less than 1 year),
Database Management (Less than 1 year),
Database Management System (Less than 1 year),
Java (Less than 1 year)
...
TECHNICAL SKILLS
Programming language: C, C++, Java
Oracle PeopleSoft
Internet of Things
Machine Learning
Database Management System
Computer Networks
Operating System worked on: Linux, Windows, Mac
...
NON-TECHNICAL SKILLS
Honest and Hard-Working
Tolerant and Flexible to Different Situations
Polite and Calm
Team-Player
```

Let's train Amazon Comprehend to understand a new concept called "SKILLS" that is specific to our resume domain:

```
comprehend_client = boto3.client('comprehend')

custom_recognizer_name = 'resume-entity-recognizer-'+ str(int(time.time()))

comprehend_custom_recognizer_response = \
    comprehend_client.create_entity_recognizer(
        RecognizerName = custom_recognizer_name,
        DataAccessRoleArn = iam_role_textract_comprehend_arn,
        InputDataConfig = {
            'EntityTypes': [
                {'Type': 'SKILLS'},
            ],
            'Documents': {
                'S3Uri': comprehend_input_doucuments
            },
            'EntityList': {
                'S3Uri': comprehend_input_entity_list
            }
        },
        LanguageCode='en'
    )
```

Using this new Skills entity recognizer that we built with Amazon Comprehend, we can perform entity recognition on the text-based resume that was extracted from the PDF with Amazon Textract earlier:

```
# Start a recognizer Job:
custom_recognizer_job_name = 'recognizer-job-'+ str(int(time.time()))

recognizer_response = comprehend_client.start_entities_detection_job(
    InputDataConfig = {
        'S3Uri': s3_test_document,
        'InputFormat': 'ONE_DOC_PER_LINE'
    },
    OutputDataConfig = {
        'S3Uri': s3_test_document_output
    },
    DataAccessRoleArn = iam_role_textract_comprehend_arn,
    JobName = custom_recognizer_job_name,
    EntityRecognizerArn = comprehend_model_response['EntityRecognizerProperties']\
                          ['EntityRecognizerArn'],
    LanguageCode = 'en'
)
```

Here is the output of our custom Amazon Comprehend entity recognizer that includes the text, offsets within the document, entity type (SKILLS), and prediction confidence:

| Start offset | End offset | Confidence | Text | Type |
|---|---|---|---|---|
| 9 | 39 | 0.9574943836014351 | analytical and problem solving | SKILLS |
| 8 | 11 | 0.7915781756343004 | AWS | SKILLS |
| 33 | 41 | 0.9749685544856893 | Solution | SKILLS |
| 20 | 23 | 0.9997213663311131 | SQL | SKILLS |
| 2 | 13 | 0.9996676358048374 | Programming | SKILLS |
| 25 | 27 | 0.9963501364429431 | C, | SKILLS |
| 28 | 32 | 0.9637213743240001 | C++, | SKILLS |
| 33 | 37 | 0.9984518452247634 | Java | SKILLS |
| 39 | 42 | 0.9986466628533158 | PHP | SKILLS |
| 44 | 54 | 0.9993487072806023 | JavaScript | SKILLS |

# Cognitive Search and Natural Language Understanding

At one point or another, we have all struggled with finding a relevant piece of information buried deep in a website, enterprise content management system, corporate wiki, or corporate file share. We also know the pain of reanswering the same frequently asked questions over and over.

Surfacing relevant and timely search results is an age-old problem that has spawned many open source solutions, including Apache Lucene, Apache SOLR, and Elasticsearch. These solutions are rooted in older NLP techniques created many years ago. When interacting with these solutions, we typically issue keyword searches that, if entered incorrectly or out of order, could result in poor search results.

Cognitive search is a modern solution to the age-old problem of discovering information. Rooted in modern NLU, cognitive search allows end users to issue natural language questions that a human would naturally ask.

Amazon Kendra uses machine learning, NLU, and cognitive search to tackle the enterprise search problem in a modern way. Instead of issuing traditional keyword searches that require extra effort to distill, we can ask Amazon Kendra full natural language questions such as, "On which floor is the IT department located in this office?" and get a specific answer such as "19th floor."

Amazon Kendra integrates with many different data sources, including Amazon S3, SharePoint, Salesforce, ServiceNow, Amazon RDS Databases, OneDrive, and many more. It supports all types of data schemas, including structured, unstructured, and semistructured. It also supports many different formats, including PDF, HTML, rich text, Microsoft Word, and PowerPoint.

While Amazon Kendra comes out of the box with various pre-trained models optimized across multiple domains, we can train Amazon Kendra with our datasets to improve the accuracy of the results. In addition, Amazon Kendra actively learns and retrains itself based on end-user usage patterns, including explicit feedback such as thumbs up and thumbs down on specific search results.

Combining Amazon Kendra and Lex, we can build a support chatbot across a broad range of devices to help answer frequently asked questions. In this example, we also include the popular workforce-collaboration tool Slack, as shown in Figure 2-3.

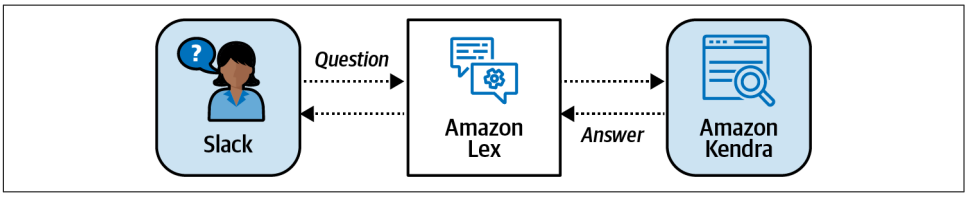


*Figure 2-3. Using Slack, Amazon Lex, and Amazon Kendra to automatically answer questions.*

The following is a sample conversation where Antje is using a Slackbot to ask questions during a live workshop. The chatbot answers commonly asked questions from attendees. This allows hosts of the workshop to focus on more-complicated questions that require human intervention:

*Antje*: "Hi there."

*Slackbot*: "Hello! How can I help?"

*Antje*: "Do you record this workshop?"

*Slackbot*: "Yes, this workshop is being recorded."

*Antje*: "Where do I find the recording?"

*Slackbot*: "The recording will be shared at *https://youtube.datascienceonaws.com* within 24 hours."

*Antje*: "Do you know how to get started with SageMaker?"

*Slackbot*: "I think the answer to your question is: On the Amazon SageMaker Studio page, under Get Started, choose Quick Start, then select Create Studio Domain."

## Intelligent Customer Support Centers

Quality customer support is important for every industry and business (as noted earlier, Customer Obsession is a key leadership principle at Amazon). In many cases, customer support directly affects the customer's perception of the business. Amazon Connect is a cloud contact center solution that implements machine learning to provide intelligent contact center features. With Connect Wisdom, customer support agents can simply enter a question, such as "What is the exchange policy for books?"

and Wisdom returns the most relevant information and best answer. Wisdom also runs machine learning on live-call transcripts to automatically identify customer issues and recommend responses to the agents.

Contact Lens for Amazon Connect adds machine learning capabilities to Amazon Connect, a cloud contact center service based on the same technology that powers Amazon's customer service. Contact Lens uses speech-to-text transcription, NLP, and cognitive search capabilities to analyze customer–agent interactions.

By automatically indexing call transcripts, Contact Lens lets us search for specific words, phrases, and sentiment—as well as redact sensitive information from transcripts to avoid leaks. Contact Lens helps supervisors spot recurring themes with interactions in real time, automatically trains agents to improve their customer support skills, and continuously categorizes contacts based on keywords and phrases used by the customer.

With Contact Lens for Amazon Connect, contact center supervisors have a single view into customer–agent interactions, product feedback trends, and potential compliance risks. Amazon Connect replicates successful interactions, highlights product feedback anomalies, and escalates poor customer–agent interactions to a supervisor.

## Industrial AI Services and Predictive Maintenance

As part of the AWS for Industrial service portfolio, AWS offers a range of AI services and hardware, including Amazon Lookout for Metrics, Lookout for Vision, Lookout for Equipment, Amazon Monitron, and AWS Panorama.

We can create accurate anomaly detection models using Amazon Lookout for Metrics. After uploading our data, Lookout for Metrics will automatically inspect the data and build the anomaly detection model. If the model detects anomalies, the service will group related anomalies together and assign a severity score. Lookout for Metrics comes with built-in connectors to popular data sources, including Amazon S3, Amazon Redshift, Amazon CloudWatch, Amazon RDS, and a variety of SaaS applications. The anomaly detection model leverages human-in-the-loop feedback to continuously improve over time.

We can spot product defects using Amazon Lookout for Vision. Lookout for Vision implements computer vision to identify visual defects in objects. It can help automate the detection of damages to parts, identify missing components, or uncover process issues in our manufacturing lines. Lookout for Vision already comes with a pretrained anomaly detection model. We simply fine-tune it to our specific images.

We can monitor the health and efficiency of our equipment using Amazon Lookout for Equipment. We upload our historical equipment sensor data to Lookout for Equipment, and the service will build a custom machine learning model to detect any

abnormal equipment behavior. Additionally, the service will automatically send an alert so we can take action. Lookout for Equipment works with any time series analog data, including sensor data such as temperature, flow rates, etc.

We can implement an end-to-end predictive maintenance use case with Amazon Monitron, which includes equipment sensors, a gateway device to securely connect to AWS, and a managed service to analyze the data for abnormal machine patterns. Amazon Monitron captures the sensor data from our equipment, identifies healthy sensor patterns, and trains a machine learning model specific to that equipment. We can provide feedback to improve the model via an Amazon Monitron mobile app, for example.

We can enable our on-premises cameras for computer vision via AWS Panorama, which comes with a hardware appliance that we can connect to our network and existing cameras. We can then deploy computer vision applications to the appliance to process the video streams from the connected cameras. Camera device manufacturers can use the AWS Panorama SDK to build new cameras that run computer vision models at the edge.

## Home Automation with AWS IoT and Amazon SageMaker

We live in a world where we estimate five billion people own some sort of mobile device, and more than half of our internet traffic happens through mobile devices. In addition, the industrial Internet of Things (IoT) revolution boasts many more billions of connected sensors and devices across our homes, office buildings, factories, cars, ships, planes, oil drills, agricultural fields, and many more.

This trend toward mobile and IoT devices also pushed computing to the edge, whether we need to analyze and preprocess data before it is sent and ingested into central data lakes (for data-privacy compliance reasons, for example) or improve user experience by serving application responses faster, eliminating the latency of a round trip to the cloud. We also see more and more machine learning happening at the edge. And while the training of machine learning models often requires powerful compute resources, making inferences against these models typically requires far less computational power.

Performing inference at the edge helps to reduce latency and cost as we are saving the round-trip time to the cloud. We can also capture and analyze the prediction results faster, trigger some action locally, or send the analyzed data back to the cloud to retrain and improve our machine learning models.

AWS IoT Greengrass deploys the model from S3 to the edge to make predictions with edge-local data. AWS IoT Greengrass also syncs the model inference results back to an S3 bucket. This prediction data can then be used to retrain and improve the

SageMaker model. AWS IoT Greengrass supports over-the-air deployments, runs locally on each device, and extends AWS to those devices.

Figure 2-4 shows a home automation use case running AWS IoT Greengrass on a local home automation server called the "edge device." AWS IoT Greengrass deploys a SageMaker model to the edge device and processes data received from camera, light switches, and light bulbs using an edge version of Lambda running on the edge device.

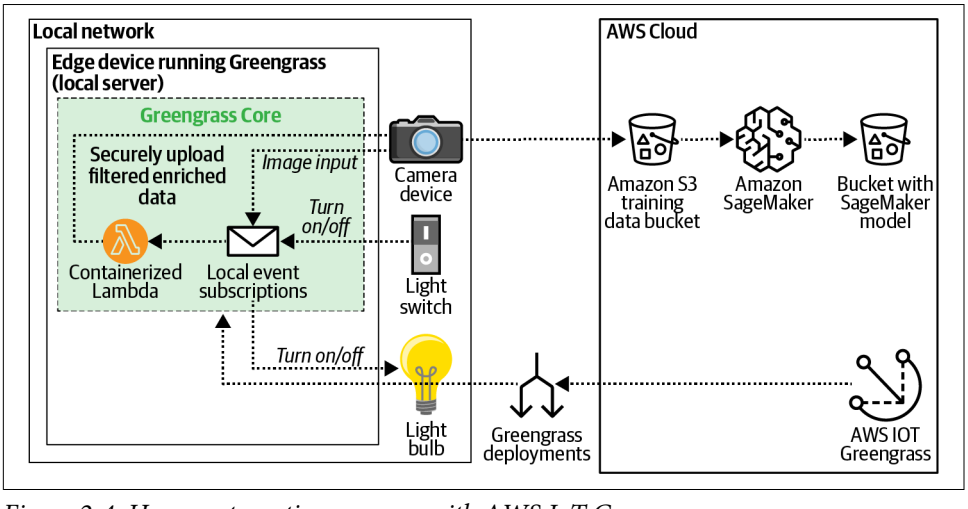


*Figure 2-4. Home automation use case with AWS IoT Greengrass.*

AWS provides a wide range of services to implement machine learning at the edge, including AWS IoT Greengrass for model deployment to the edge, SageMaker Neo for model optimization, and SageMaker Edge Manager for managing models at the edge. We will dive deeper into SageMaker Neo and Edge Manager in Chapter 9.

## Extract Medical Information from Healthcare Documents

In the healthcare space, AWS offers many dedicated services. The services have been developed specifically for the characteristics and needs of healthcare data and comply with healthcare regulations. As part of the Amazon AI HIPAA-eligible healthcare portfolio of services, AWS offers Amazon Comprehend Medical, Amazon Transcribe Medical, and Amazon HealthLake.

Comprehend Medical is an NLP service that has been pre-trained specifically on medical language. Comprehend Medical automates the extraction of health data from medical texts, such as doctors' notes, clinical trial reports, or patient health records.

Transcribe Medical is an ASR service that has been similarly pre-trained on medical language. We can use Transcribe Medical to transcribe medical speech into text. With a simple API call, we can automate clinical documentation workflows or even subtitle telemedicine.

HealthLake is a secure data lake that complies with the Fast Healthcare Interoperability Resources industry standard. In addition to storing, indexing, and transforming healthcare data, Amazon HealthLake leverages machine learning to identify, understand, and extract medical information from the raw data, such as medical reports and patient notes. We can use Amazon QuickSight, Athena, and SageMaker to run advanced analytics and machine learning on our healthcare data.

# Self-Optimizing and Intelligent Cloud Infrastructure

The Amazon AI/ML services that we have introduced so far are not the only services that provide sophisticated machine learning. In fact, more and more existing AWS services are being enriched with machine learning capabilities, and new machine-learning-powered services are being introduced across a variety of use cases. Let's take a quick look at some of these hidden gems.

## Predictive Auto Scaling for Amazon EC2

Amazon EC2, short for Elastic Compute Cloud, provides virtual server instances in the AWS cloud. One of the challenges in running our applications on those Amazon EC2 instances is how we make sure to scale the number of instances to serve the current workload, basically matching supply with demand. Luckily, there's Amazon EC2 Auto Scaling, which helps us with exactly that. Based on changes in demand, we can configure Amazon EC2 Auto Scaling to automatically add or remove compute capacity. This dynamic scaling approach is still reactive, though, as it acts on monitored traffic and Amazon EC2 instance utilization metrics.

We can take this a level further to a proactive approach in combination with a service called *AWS Auto Scaling*. AWS Auto Scaling provides a single interface to set up automatic scaling of multiple AWS services, including Amazon EC2. It combines dynamic and predictive scaling. With predictive scaling, AWS uses machine learning algorithms to predict our future traffic based on daily and weekly trends and provisions the right number of Amazon EC2 instances in advance of anticipated changes, as shown in Figure 2-5.

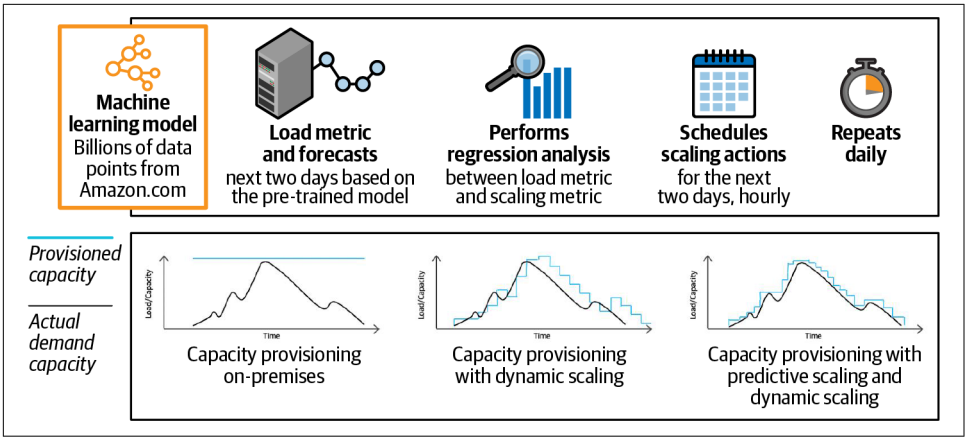


*Figure 2-5. Predictive scaling with AWS Auto Scaling anticipates traffic changes to provision the right number of Amazon EC2 instances.*

## Anomaly Detection on Streams of Data

Streaming technologies provide us with the tools to collect, process, and analyze data streams in real time. AWS offers a wide range of streaming technology options, including Amazon MSK and Amazon Kinesis. While we dive deeper into streaming analytics and machine learning with Amazon Kinesis and Apache Kafka in Chapter 10, we want to highlight Kinesis Data Analytics as a simple and powerful way to create streaming applications with just a few lines of code.

Kinesis Data Analytics provides built-in capabilities for anomaly detection using the Random Cut Forest (RCF) function in Kinesis Data Analytics to build a machine learning model in real time and calculate anomaly scores for numeric values in each message. The score indicates how different the value is compared to the observed trend. The RCF function also calculates an attribution score for each column, which reflects the anomaly of the data in that particular column. The sum of all attribution scores of all columns is the overall anomaly score.

# Cognitive and Predictive Business Intelligence

Many machine learning applications and models assume data being readily available in a data lake (discussed in Chapter 4). In reality, though, much of the world's data is stored and processed in structured, relational databases. In order for us to apply machine learning to this structured data, we have to either export the data or develop a custom application to read the data before applying any machine learning. Wouldn't it be great if we could use machine learning straight from our business intelligence service, our data warehouse, or our databases? Let's see how to do this on AWS.

## Ask Natural-Language Questions with Amazon QuickSight

Amazon QuickSight is a business intelligence service that performs interactive queries and builds visualizations on data sources such as Amazon Redshift, Amazon RDS, Amazon Athena, and Amazon S3. QuickSight can also detect anomalies, create forecasts, and answer natural-language questions from our data through QuickSight ML Insights and QuickSight Q.

QuickSight ML Insights runs the RCF algorithm to identify change in millions of metrics, across billions of data points. ML Insights also enables forecasting based on the observed metrics. The RCF algorithm automatically detects seasonality patterns in our data, excludes any outliers, and imputes missing values.

With QuickSight Q, we can ask natural-language questions such as, "What are the best-selling product categories in the US state of California?" QuickSight uses machine learning to understand the question, apply the question to our data, and create a chart to answer our question, as shown in Figure 2-6. We will dive deep into QuickSight in Chapter 5.


*Figure 2-6. QuickSight Q understands natural-language questions and automatically creates charts to answer the questions.*

## Train and Invoke SageMaker Models with Amazon Redshift

Amazon Redshift is a fully managed data warehouse that allows us to run complex analytic queries against petabytes of structured data. Using Amazon Redshift ML, we can use our data in Amazon Redshift to train models with SageMaker Autopilot as new data arrives. SageMaker Autopilot automatically trains, tunes, and deploys a model. We then register and invoke the model in our Amazon Redshift queries as a user-defined function (UDF). Figure 2-7 shows how we make predictions with the

`USING FUNCTION` SQL clause. We will show a more detailed example of Amazon Redshift ML and SageMaker Autopilot in Chapter 3.

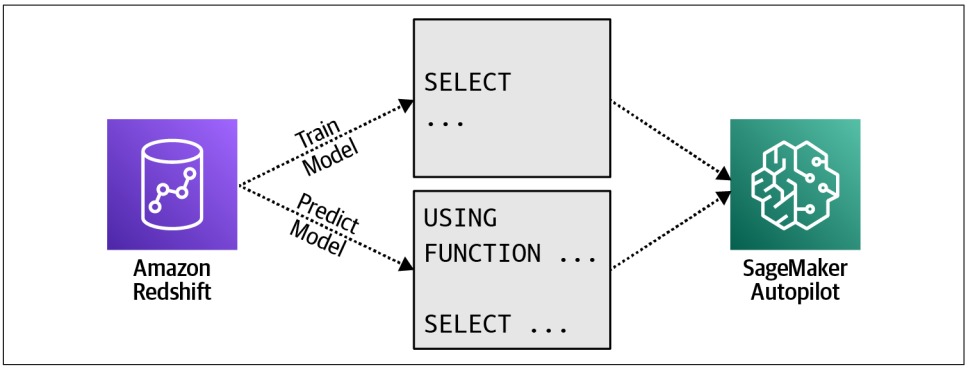


*Figure 2-7. Amazon Redshift uses SageMaker Autopilot to train and invoke a SageMaker model as a UDF.*

We can create a UDF to invoke any AWS service using Lambda Functions. This sample UDF invokes a Lambda Function:

```
USING FUNCTION invoke_lambda(input VARCHAR)
RETURNS VARCHAR TYPE LAMBDA_INVOKE WITH
(lambda_name='<LAMBDA_NAME>') SELECT invoke('<INPUT>');
```

## Invoke Amazon Comprehend and SageMaker Models from Amazon Aurora SQL Database

Aurora, a MySQL- and PostgreSQL-compatible relational database, is natively integrated with Amazon Comprehend and Amazon SageMaker, as shown in Figure 2-8.

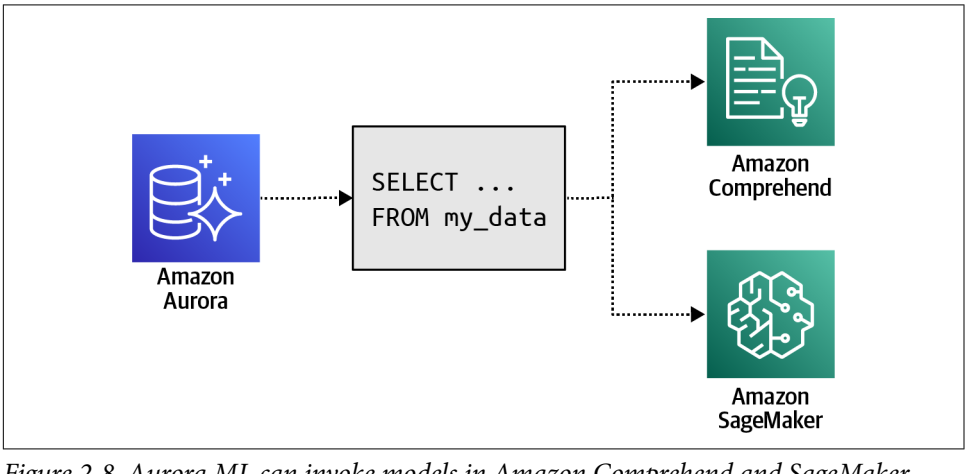


*Figure 2-8. Aurora ML can invoke models in Amazon Comprehend and SageMaker.*

We can use either built-in SQL functions (with Amazon Comprehend) or custom-written SQL functions (with Amazon SageMaker) in our queries to apply machine learning to the data. As shown in previous sections, we could leverage Amazon Comprehend for customer sentiment analysis (built-in SQL function) on maybe a product review or use Amazon SageMaker for any custom-developed machine learning model integration.

Let's say we have some sample product reviews in a relational table:

```
CREATE TABLE IF NOT EXISTS product_reviews (
       review_id INT AUTO_INCREMENT PRIMARY KEY,
       review_body VARCHAR(255) NOT NULL
);

INSERT INTO product_reviews (review_body)
VALUES ("Great product!");
INSERT INTO product_reviews (review_body)
VALUES ("It's ok.");
INSERT INTO product_reviews (review_body)
VALUES ("The worst product.");
```

Then, we can use the following built-in SQL functions to let Amazon Comprehend return us the sentiment and confidence score:

```
SELECT review_body,
       aws_comprehend_detect_sentiment(review_body, 'en') AS sentiment,
       aws_comprehend_detect_sentiment_confidence(review_body, 'en') AS confidence
  FROM product_reviews;
```

This would show a result similar to this:

```
review_body               sentiment         confidence
-----------------------------------------------------------
Great product!            POSITIVE          0.9969872489
It's ok.                  POSITIVE          0.5987234553
The worst product.        NEGATIVE          0.9876742876
```

## Invoke SageMaker Model from Amazon Athena

Similarly, we can use Amazon Athena, a service that lets us use SQL queries to query data stored in Amazon S3, and invoke SageMaker machine learning models for inference directly from those queries, as shown in Figure 2-9.

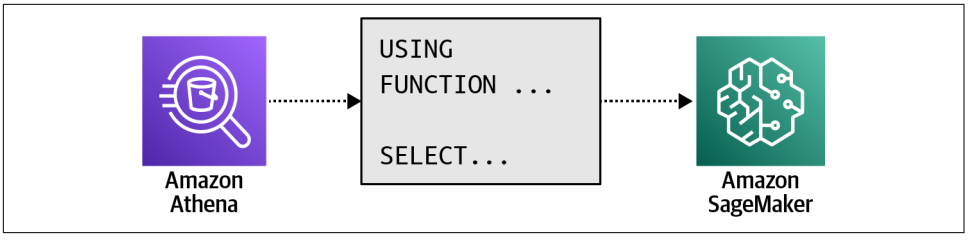


*Figure 2-9. Amazon Athena can invoke SageMaker models.*

We define a UDF with the `USING FUNCTION` SQL statement that invokes a custom-built Amazon SageMaker Endpoint that serves the sentiment predictions. Any subsequent `SELECT` statement in the query can then reference the function to pass values to the model.

Here's a simple example:

```
USING FUNCTION predict_sentiment(review_body VARCHAR(65535))
    RETURNS VARCHAR(10) TYPE
    SAGEMAKER_INVOKE_ENDPOINT WITH (sagemaker_endpoint = '<ENDPOINT_NAME>')

SELECT predict_sentiment(review_body) AS sentiment
     FROM "dsoaws"."amazon_reviews_tsv"
     WHERE predict_sentiment(review_body)="POSITIVE";
```

### Run Predictions on Graph Data Using Amazon Neptune

Amazon Neptune is a fully managed graph database that allows us to build and run applications across highly connected datasets. Neptune ML implements graph neural networks (GNNs) to make predictions using graph data. While algorithms such as XGBoost are designed for traditional tabular datasets, GNNs have been specifically designed to deal with the complexity of graphs and potentially billions of relationships. Neptune ML uses the open source Deep Graph Library and Amazon SageMaker to automatically choose, train, and deploy the best model for our graph data.

## Educating the Next Generation of AI and ML Developers

Amazon and AWS offer many programs and services to help educate the next generation of AI/ML developers. Amazon's Machine Learning University program—used to train Amazon employees—was released to the public in 2020. AWS Training and Certification (T&C) offers a broad range of on-demand and classroom training courses that help to prepare for the AWS Machine Learning specialty certification. In addition, AWS has partnered with Udacity, Coursera, and DeepLearning.AI to create several Massive Open Online Courses to give hands-on experience with the Amazon AI and machine learning stack.

In this section, we discuss the deep-learning-powered AWS devices that provide a fun and educational way to get hands-on experience with computer vision, reinforcement learning, and generative adversarial networks (GANs).

The developer-focused device family consists of the following: AWS DeepLens, DeepRacer, and DeepComposer. AWS DeepLens is a wireless, deep-learning-enabled video camera. AWS DeepRacer is a fully autonomous 1/18th-scale race car driven by reinforcement learning. And AWS DeepComposer is a musical keyboard powered by GANs to transform our melodies into original songs.

## Build Computer Vision Models with AWS DeepLens

AWS DeepLens is a deep-learning-enabled video camera that comes with a rich set of computer vision tutorials and pre-built models. If we want to learn how to build computer vision apps and see our first results in a matter of minutes, we can just use one of the many sample projects that come with pre-trained models and a simple inference function. The camera will perform local inference on the device against the deployed model.

If we are a more experienced developer, we can build and train our custom convolutional neural network (CNN) model in any of the supported deep-learning frameworks, such as TensorFlow, Apache MXNet, or Caffe, and then deploy the project to the AWS DeepLens device. Figure 2-10 shows a typical AWS DeepLens workflow.

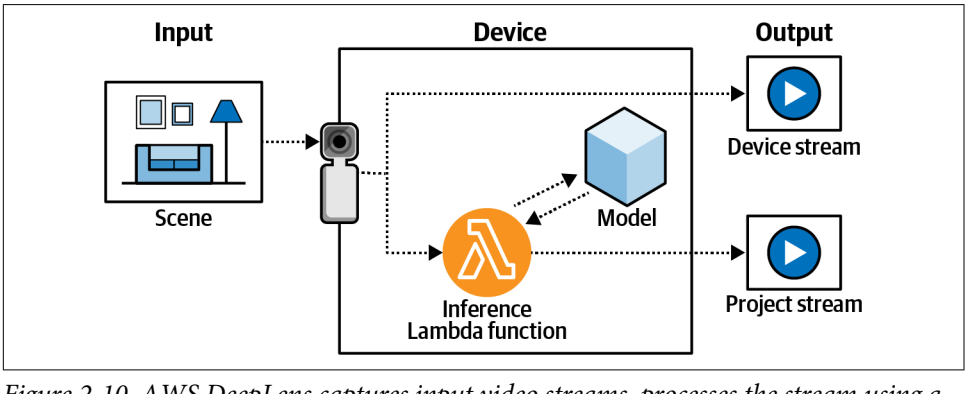


*Figure 2-10. AWS DeepLens captures input video streams, processes the stream using a deployed model, and generates two output video streams.*

AWS DeepLens is both an edge device and a camera. Therefore, AWS DeepLens runs AWS IoT Greengrass Core and can execute its own Lambda functions. New models are pushed to AWS DeepLens using AWS IoT Greengrass. The camera captures the input video stream and produces two output streams: a device stream that's passed through as is, and a project stream that is the result of the deployed model's processed video frames.

Any project we deploy needs to contain a Lambda function to process the input video frames, also called the *Inference Lambda function*. We first bundle that function together with a Lambda runtime and a trained model. Then we deploy the project using AWS IoT Greengrass to an AWS DeepLens device.

## Learn Reinforcement Learning with AWS DeepRacer

AWS DeepRacer is a fully autonomous 1/18th-scale race car driven by reinforcement learning. The car is equipped with two cameras, a LIDAR sensor, and an on-board compute module. The compute module performs the inference in order to drive the car along a track.

Reinforcement learning is applied to a variety of autonomous decision-making problems. It gained broader popularity when the team of scientists, engineers, and machine learning experts at DeepMind released AlphaGo, the first computer program that defeated a professional human Go player back in 2015.

Go is an ancient, strategic board game known for its complexity. It was invented in China about three thousand years ago and is still being played by amateurs and in various professional leagues around the globe.

While AlphaGo learned the game by playing thousands of matches against human players, the subsequent release AlphaGo Zero learned Go by just playing against itself. This has revolutionized the field of reinforcement learning once more as it performed even better than the previous release and showed that the model was able to discover new knowledge and apply unconventional strategies to win.

At a high level, reinforcement learning is a machine learning method that aims at autonomous decision making by an agent to achieve a specific goal through interactions with an environment, as shown in Figure 2-11. Learning is achieved through trial and error.

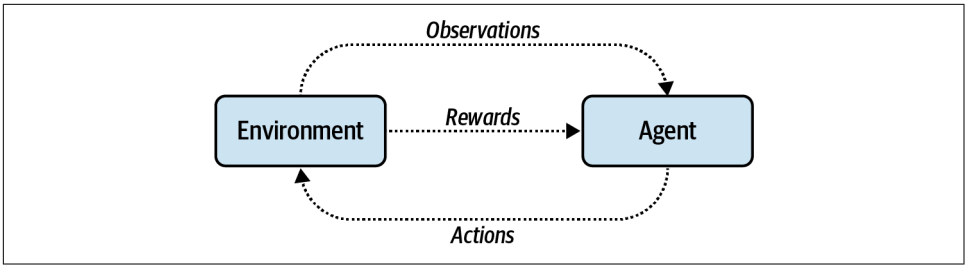


*Figure 2-11. Reinforcement learning is a machine learning method that aims at autonomous decision making by an agent to achieve a specific goal through interactions with an environment.*

We will dive deeper into reinforcement learning to compare models in production with multiarmed bandits in Chapter 9, but let's get back to our autonomous-car racing scenario. In our example, the agent is the AWS DeepRacer car, and the environment consists of track layouts, traveling routes, and traffic conditions. The actions include steer left, steer right, brake, and accelerate. Actions are chosen to maximize a reward function, which represents the goal of reaching the destination quickly without accidents. Actions lead to states. Figure 2-12 shows the AWS DeepRacer's flow of actions, rewards, and states.

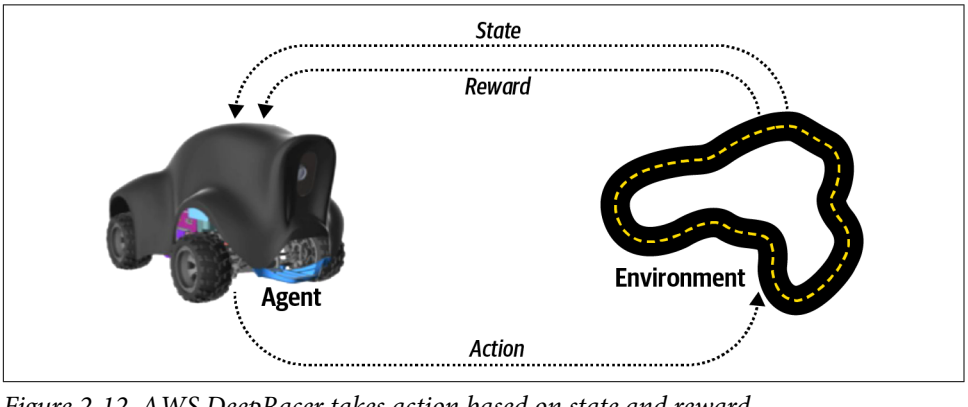


*Figure 2-12. AWS DeepRacer takes action based on state and reward.*

We don't even need a physical track or car to get started. We can start training our custom reinforcement learning model in the AWS DeepRacer console and use the AWS DeepRacer simulator to evaluate our model on a virtual track, as shown in Figure 2-13.

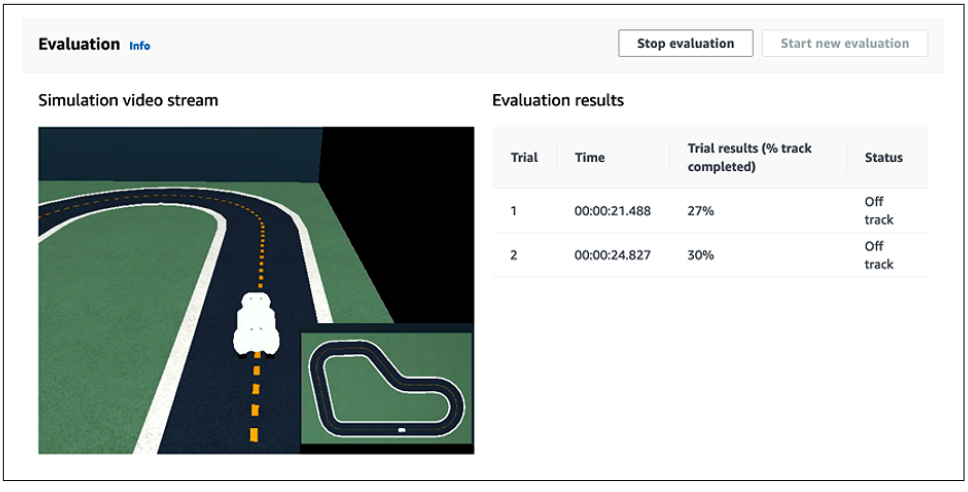


*Figure 2-13. Model evaluation using the AWS DeepRacer simulator. Source: AWS DeepRacer Developer Guide.*

AWS also maintains a global AWS DeepRacer League and leaderboard that ranks vehicle performances from official AWS DeepRacer League racing events happening throughout the year, including both physical and virtual events.

## Understand GANs with AWS DeepComposer

Yes, everyone looked a bit puzzled when AWS introduced the AWS DeepComposer device at the annual AWS re:Invent conference back in December 2019. Soon, however, we started hearing those distinctive sounds coming from hotel hallways throughout Las Vegas. AWS DeepComposer is a musical USB keyboard to help us learn generative AI. It is designed to work with the AWS DeepComposer service to turn a simple melody into an original song. An AWS DeepComposer device is shown in Figure 2-14.

*Figure 2-14. AWS DeepComposer is a musical USB keyboard that helps us learn generative AI. Source: AWS.*

Generative AI, specifically in the form of GANs, is used to generate new content from inputs we provide. This input can be images, text, or—yes—music. Generative AI models automatically discover and learn patterns in data and use this knowledge to generate new data based on the data they were trained on. GANs use two competing algorithms, a generator and a discriminator, to generate new content, as shown in Figure 2-15.

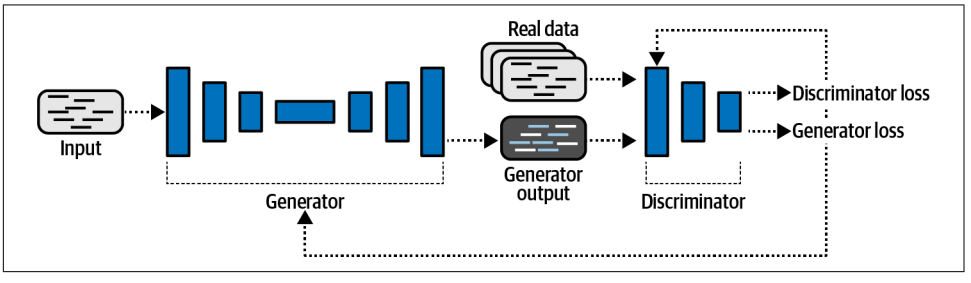


*Figure 2-15. GANs leverage a generator and discriminator algorithm.*

A generator is a CNN that learns to create new content based on the patterns of the input data. The discriminator is another CNN that is trained to actually differentiate between real and generated content. Both the generator and the discriminator are trained in alternating runs to force the generator to create more and more realistic content, while the discriminator improves at identifying synthetic content as opposed to real content.

Applied to our music example, when we play a melody on the keyboard, AWS DeepComposer can add up to three additional accompaniment tracks to create a new composition. The generator network is adapted from the popular U-Net architecture used in Computer Vision and has been trained on a publicly available dataset of Bach's compositions.

## Program Nature's Operating System with Quantum Computing

Building useful quantum applications requires new skills and a radically different approach to problem solving. Acquiring this expertise takes time and requires access to quantum technologies and programming tools.

Amazon Braket helps us explore the potential of quantum hardware, understand quantum algorithms, and retool for a quantum future. Figure 2-16 shows the flywheel of quantum computing growing the ecosystem through better hardware, more developers, and more use cases.

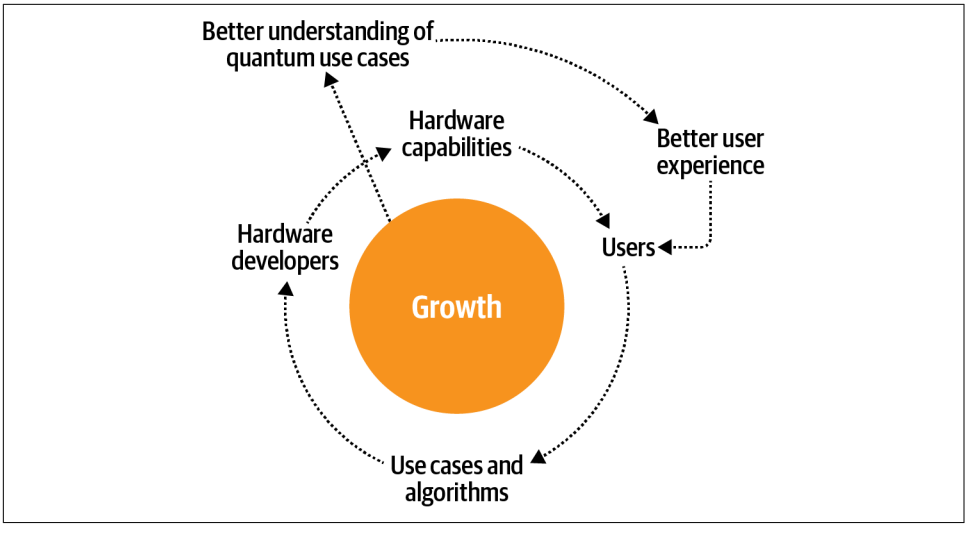


*Figure 2-16. The flywheel of quantum computing grows with Amazon Braket.*

There are many similarities between today's graphics processing units (GPUs) and tomorrow's quantum processing units (QPUs). GPUs revolutionized AI and machine learning though highly parallel, digital computations. GPUs also required a different set of skills, libraries (i.e., NVIDIA's CUDA), and hardware to take advantage of this massive parallelism. Additionally, GPU devices are "off-chip" relative to the CPU devices that traditionally manage the larger computation workflow. Synchronizing data between CPUs and GPUs requires special hardware and software to accommodate the physical separation.

Similarly, QPUs perform computations through massively parallel, quantum computations—many orders of magnitude more parallel than their digital counterparts. Additionally, QPUs require a different set of skills, libraries, and hardware. They are off-chip relative to CPUs and therefore require special hardware and software to perform the synchronization operations similar to GPUs, as shown in Figure 2-17.

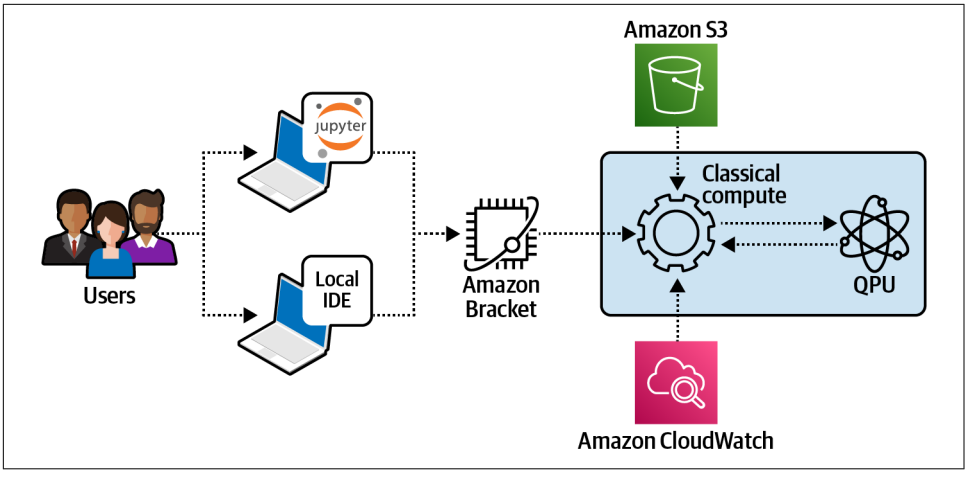


*Figure 2-17. Using a QPU with a classic digital computer.*

## Quantum Bits Versus Digital Bits

Quantum bits (qubits) are the quantum computing equivalent of classical digital bits. However, their state (0 or 1) is probabilistic and therefore requires a READ operation before the value is known. This probabilistic state is referred to as "superposition" and is a key principle behind quantum computing.

Today's accessible quantum computers are around 70 to 100 qubits. However, a significant portion of these qubits are required for error correction given the relative "noisy" environment of the quantum hardware. Cryptography, for example, requires nearly 6,000 clean qubits to break 2048-bit RSA. Six thousand clean qubits requires approximately 1,000,000 error-correcting, redundant qubits to adjust for the noisy environment offered by current quantum hardware.

## Quantum Supremacy and the Quantum Computing Eras

Until recently, we were in the "classically simulatable" phase where we could simulate the performance improvements of quantum computers. However, in 2019, we reached a point of "quantum supremacy" where we are no longer able to simulate and measure additional performance improvements from quantum computers due to limitations with today's digital computers.

The current era is called *Noisy Intermediate-Scale Quantum*. During this era, we are trying to correct for the noise introduced by the quantum computing environment, which requires very specific temperature and vacuum characteristics. Similar to error-correcting registers and RAM chips, we need error-correcting qubits and QRAM to enter the next era, called the *Error-Corrected Quantum Computer* era, as shown in Figure 2-18.

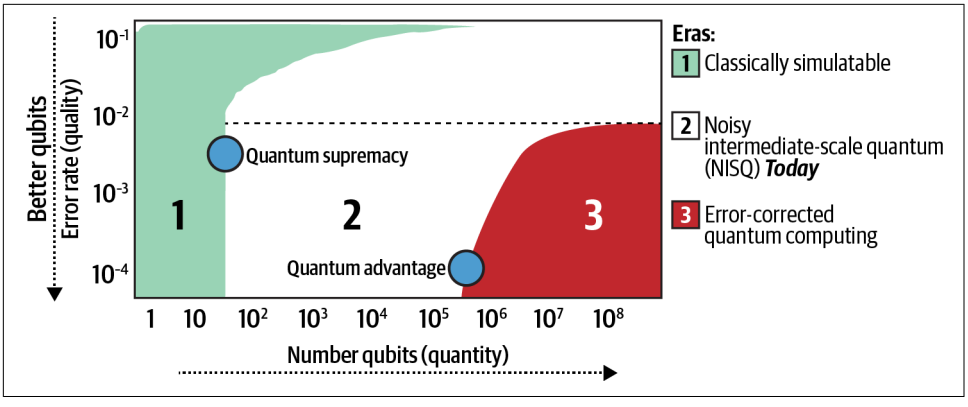


*Figure 2-18. Quantum computing eras.*

## Cracking Cryptography

It is estimated that quantum computers are only 10 or so years away from cracking modern-day RSA cryptography. Today, cryptography is effective because we don't have enough computing power to perform the numeric factorization required to crack the code.

However, with an estimated 6,000 "perfect" qubits (no error correcting needed), we can crack the RSA code within just a few minutes. This is scary and has given rise to "quantum-aware" or "post-quantum" cryptography such as Amazon's s2n open source implementation of the TLS protocol, which uses post-quantum cryptography as opposed to classical cryptography. We dive deeper into post-quantum cryptography in Chapter 12.

## Molecular Simulations and Drug Discovery

Quantum computers have unique parallelization capabilities and can natively manipulate quantum mechanical states. Therefore, they have the potential to solve very important problems, such as mapping the electronic structure of molecules. Quantum simulations will likely lead to discovering new materials, catalysts, drugs, and high-temperature superconductors.

## Logistics and Financial Optimizations

Optimization problems are ubiquitous across many domains, including supply chain logistics and financial services. Finding the optimal approach from an exponential set of possible options can saturate the resources of a classical digital computer. Quantum computers can break through this barrier and accelerate many optimization techniques, including linear programming algorithms and Monte Carlo methods.

## Quantum Machine Learning and AI

Unfortunately, today's use of quantum computers in machine learning and AI is pretty limited. We have seen some early improvements in linear algorithms, such as support vector machines and Principal Component Analysis. We have also seen examples where quantum research has inspired improvements in classical recommendation algorithms. In the future, error-correcting quantum computers will likely lead to a rich class of scalable and high-performance quantum machine learning and AI models.

## Programming a Quantum Computer with Amazon Braket

Amazon Braket supports Jupyter Notebook and offers a Python SDK to allow developers to interact with a quantum computer. Using the Python SDK, we asynchronously submit tasks to a remote QPU. This is similar to how we submitted jobs and "rented" a shared computer back in the early days of computing to complete those jobs. This is also similar to offloading compute from a CPU to a GPU. The key difference, however, is that the CPU and GPU share classical digital fundamentals—the QPU does not.

The following code demonstrates how to build a quantum circuit involving multiple qubits. This example demonstrates how to perform "quantum teleportation" where information (*not* matter) is transported from one qubit to another without using classical digital circuits or network cables:

```
from braket.aws import AwsDevice
from braket.circuits import Circuit, Gate, Moments
from braket.circuits.instruction import Instruction

device = AwsDevice("arn:aws:braket:::device/qpu/ionq/ionQdevice")

# Alice and Bob initially share a Bell pair.
circ = Circuit();
circ.h([0]);
circ.cnot(0,1);

# Define Alice's encoding scheme.
# Define four possible messages and their gates.
message = {
           "00": Circuit().i(0),
           "01": Circuit().x(0),
           "10": Circuit().z(0),
           "11": Circuit().x(0).z(0)
          }

# Alice selects a message to send.  Let's choose '01'.
m = "01"

# Alice encodes her message by applying the gates defined above.
```

```
circ.add_circuit(message[m]);

# Alice then sends her qubit to Bob so that Bob has both qubits.
# Bob decodes Alice's message by disentangling the two qubits.
circ.cnot(0,1);
circ.h([0]);

print(circ)

### OUTPUT ###

T  : |0|1|2|3|4|
q0 : -H-C-X-C-H-
        |   |
q1 : ---X---X---
T  : |0|1|2|3|4|
```

## AWS Center for Quantum Computing

AWS has partnered with Caltech to build the AWS Center for Quantum Computing, scheduled to open in 2021. This center will focus on developing useful quantum applications, error-correcting qubits, quantum programming models, and new quantum hardware.

# Increase Performance and Reduce Cost

What if we were able to double our code speed and reduce our server-pool size by half? We could potentially save significant money. What if we could automatically detect operational issues in our applications and see the recommended fixes to improve availability? Reducing application downtime is another huge cost-saving opportunity.

In this section, we introduce the fully managed services Amazon CodeGuru Reviewer, Amazon CodeGuru Profiler, and Amazon DevOps Guru. CodeGuru Reviewer and Profiler help us improve code performance and reduce our resource requirements, while Amazon DevOps Guru helps to detect operational issues and improve application availability.

## Automatic Code Reviews with CodeGuru Reviewer

Code reviews are a well-known best practice for software development. The idea is that our code is reviewed by a more experienced set of team members to provide feedback on code performance, quality, and security. In addition to domain expertise, these experienced team members possess tacit knowledge of the team's coding idioms as well as an acute sense of code smell.

Sometimes, however, even the most experienced team member will miss subtle performance bottlenecks or mishandled exceptions. These reviewers are often focused on domain-specific issues such as poor implementation of the domain model or misconfigured service integrations. Additionally, reviewers are often limited to a static view of the code versus live metrics into the code's runtime. CodeGuru consists of CodeGuru Reviewer for automated code reviews and CodeGuru Profiler to monitor code performance.

CodeGuru Reviewer automates the code-review process and makes suggestions using machine learning models trained on millions of lines of code from hundreds of thousands of Amazon's internal code bases as well as 10,000+ open source projects on GitHub.

We simply point CodeGuru to our source code repository in a secure and private manner—CodeGuru will start making suggestions. CodeGuru analyzes all pull requests on our source code repositories and automatically flags critical defects such as credential leaks, resource leaks, concurrency race conditions, and inefficient use of AWS resources. It suggests changes to specific lines of code to remediate the defects, as shown in Figure 2-19.

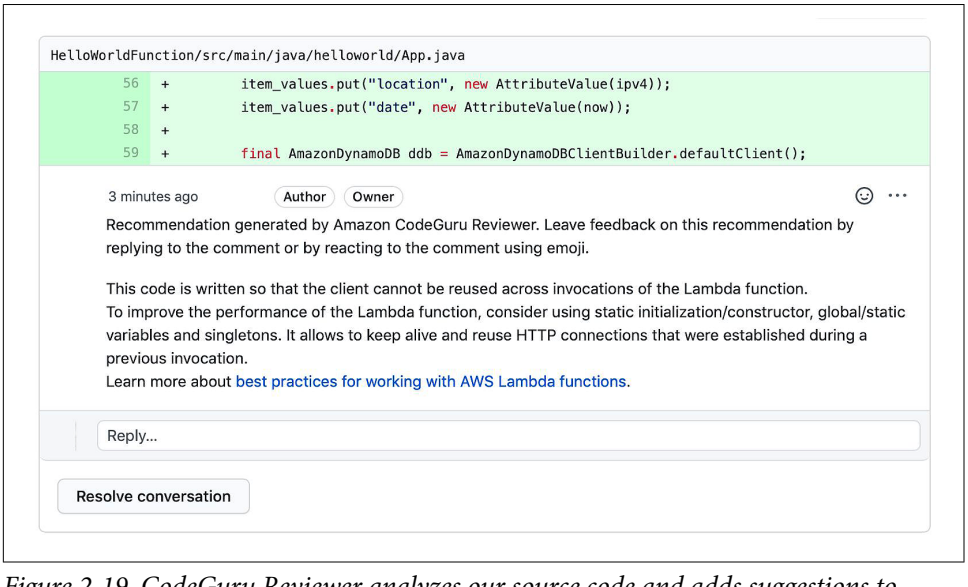


*Figure 2-19. CodeGuru Reviewer analyzes our source code and adds suggestions to improve performance and reduce cost.*

In this case, the original code from a Lambda function was creating a new DynamoDB client on every invocation instead of creating the client once and caching it. Without this change, we will waste unnecessary compute cycles and memory registers as we continuously re-create the same DynamoDB client object on every

invocation. With this change, our Lambda functions can handle more requests per second, which results in fewer resources and lower cost.

CodeGuru Reviewer checks for Python and Java best practices, including connection pooling and exception handling. Reviewer includes Security Detector to detect security issues such as unsanitized arguments passed to OS-level Python subprocess calls. CodeGuru Reviewer also identifies code smells, reduces technical debt, and improves codebase maintainability.

## Improve Application Performance with CodeGuru Profiler

CodeGuru Profiler can detect bottlenecks in our code at runtime by analyzing the application runtime profile, flagging the most expensive line of code, and providing intelligent recommendations. Profiler creates visualizations such as the flame graph in Figure 2-20 to identify where we should spend our time to optimize performance and save the most money.

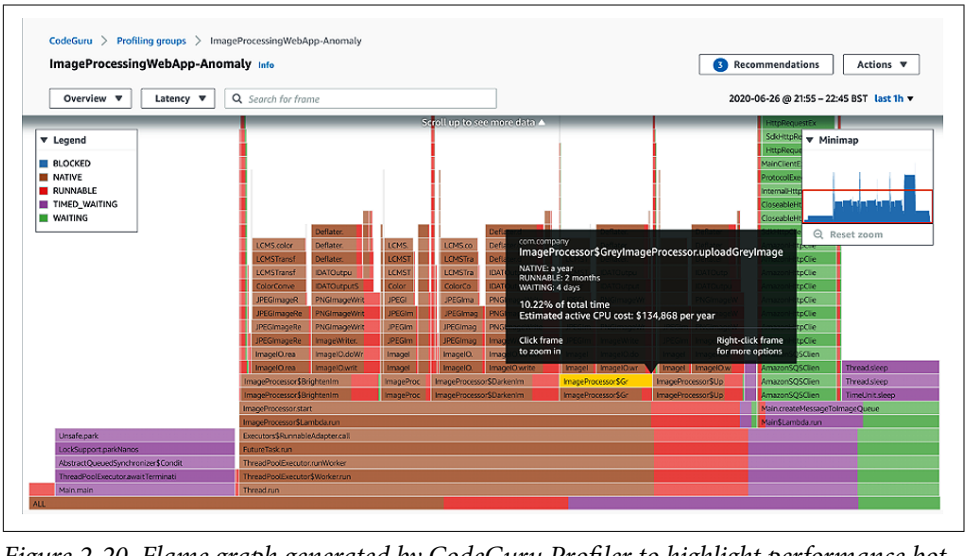


*Figure 2-20. Flame graph generated by CodeGuru Profiler to highlight performance bottlenecks in our code.*

The flame graph shows the call stack in human-readable form with the exact function names. When analyzing flame graphs, we should dive deep into any plateaus that we find. Plateaus often indicate that a resource is stalled waiting for network or disk I/O. Ideally, our flame graph will show a lot of narrow peaks and not a lot of plateaus.

### Improve Application Availability with DevOps Guru

Amazon DevOps Guru is an ML-powered operations service that automatically detects operational issues in applications and recommends fixes. DevOps Guru looks at application metrics, logs, and events to identify any behavior that deviates from normal operating patterns, such as increased response latencies, elevated error rates, and excessive resource utilization. When such a pattern is recognized, DevOps Guru sends an alert together with a summary of related anomalies, the potential root cause, and a possible resolution.

## Summary

In this chapter, we have shown many different use cases that can be solved with various AWS AI and machine learning services out of the box with little or no code. Whether we are application developers and don't know much about machine learning or experienced data scientists who want to focus on the difficult machine learning problems, the managed AI and machine learning services by Amazon are worth exploring.

We can easily enrich our applications with ready-to-use AI services, whether our business use case requires us to bring machine learning to the edge or we are just at the beginning of our AI/ML journey and want to find some fun educational ways to get started with computer vision, reinforcement learning, or GANs.

We've also shown some examples of how to put the high-level AI services to work, including Amazon Personalize for recommendations and Forecast for forecast demand.

We showed how machine learning powers a lot of existing AWS services, including predictive Amazon EC2 auto-scaling and warm-pooling. We also explored how to detect and prevent sensitive data leaks using Macie and prevent fraud using Amazon Fraud Detector. We covered how to improve the customer support experience with Amazon Contact Lens for Amazon Connect, Comprehend, Kendra, and Lex. We also described how we can automate source code reviews and identify performance and cost benefits using CodeGuru Reviewer, CodeGuru Profiler, and DevOps Guru.

In Chapter 3, we will discuss the concept of automated machine learning. We will show how to build predictive models in just a few clicks with Amazon SageMaker Autopilot and Amazon Comprehend.