**6**

**Harnessing the Power of Cloud Computing**

Technology is transforming the world, yet according to IDC, on average enterprises only put 15% of their IT budgets toward innovation, and the other 85% is spent on maintaining existing infrastructure. In data centers, servers are at the core of any IT workload and consume most of the IT effort and budget. To run any application, you need computing. Even though the cloud brings with it the concept of serverless computing, there are still servers in the background managed by cloud vendors.

The AWS compute platform helps you to shift your budgets from maintaining existing infrastructure to driving innovation. You will learn about some of the basic compute services available in AWS and how these services came to be. In addition, you will learn about serverless compute and hybrid compute. You will also learn how AWS handles the fundamental services of computing.

In this chapter, we will discuss the following topics:

* Compute in AWS
* Learning about Amazon **Elastic Compute Cloud** (**EC2**)
* Reviewing Amazon EC2 best practices
* Amazon Elastic Load Balancing
* Learning serverless compute with AWS Lambda and Fargate
* High-performance computing
* Hybrid compute

By the end of this chapter, you will be familiar with the various compute options available in AWS, and you will be able to choose the right compute option for the right workload. Let’s dive deep into the world of AWS compute.

**Compute in AWS**

The cloud has changed the way we see compute today. A decade ago, there was no such word as “compute,” and “server” was the most used terminology. Running your application code or database was about servers with a CPU attached. These servers could be dedicated to physical bare-metal machines or VMs hosted on physical machines. The cloud started with the same concept of providing on-demand servers, which are VMs hosted in cloud providers’ data centers. In AWS terminology, these servers, called Amazon **Elastic Compute Cloud** (**EC2**), are on-demand VMs available based on a per-second billing model.

With EC2, AWS takes care of the physical server, but there is still a maintenance overhead involved in patching and securing the underlying OS in these EC2 instances. Also, cloud providers such as AWS are looking to provide more optimized solutions and help you focus on the coding part to build business logic. AWS launched a serverless compute service called **AWS Lambda**to reduce OS maintenance overhead in 2014. It was the first offering where you just write a piece of code and run it using the service without worrying about servers and clusters, which generated the term **function-as-a-service** (**FaaS**). Lambda pointed the entire IT industry in the direction of building compute services without servers. However, Lambda still runs on servers behind the scenes, but that is abstracted from the end-user, resulting in the term **serverless compute**.

Initially, Lambda used to run small functions, mainly automating operations such as spin-up infrastructure and triggering CI/CD pipelines. However, AWS Lambda became more powerful over time, and organizations built complex applications such as dynamic e-commerce websites using Lambda. It’s become a low-cost and scalable option for new businesses, and it helps them succeed. For example, in 2017, A Cloud Guru built an entire training content distribution website using Lambda, which scales to 300,000 students at a meagre cost. Read this SiliconANGLE article to find out more: <https://siliconangle.com/2017/08/15/a-cloud-guru-uses-lambda-and-api-gateway-to-build-serverless-company-awssummit/>.

Further down the line, customers started using serverless compute services to run their container workloads. This resulted in AWS Fargate, launched in 2017 for **Elastic Container Service** (**ECS**), which allows the customer to run their Docker containers in AWS without a server. Later, in 2019, Fargate launched **Elastic Kubernetes Service** (**EKS**) to allow customers to run Kubernetes serverless.

Now, AWS is going all-in on serverless. In 2020, they launched various serverless options for analytics services, such as Amazon Redshift Serverless for building petabyte-scale data warehouses in the cloud without a server, **Elastic MapReduce** (**EMR**) Serverless for transforming terabytes of data using a serverless Hadoop system in the cloud, and **Managed Streaming for Kafka** (**MSK**) Serverless for running Kafka workloads in the cloud without worrying about the server. You will learn about these serverless compute services throughout this book.

So now you know why the term is not called **server** but **compute**, because **server** only refers to a physical server or VM, while **compute** is much more than that, with AWS Lambda and serverless compute. AWS provides choices in how you consume compute to support existing applications and build new applications in a way that suits your business needs, whether in the form of instances, containers, or serverless compute. Let’s dive into the computing world and start with AWS’s core service, EC2.

**Learning about Amazon EC2**

As you learned in the previous section, Amazon EC2 is AWS’s way of naming servers. It’s nothing but virtual machines hosted on a physical server residing inside the AWS data center in a secure environment. It is all about standardizing infrastructure management, security, and growth, and building an economy of scale to quickly meet client demand for services in minutes and not months. AWS takes full advantage of virtualization technologies and can slice one computer to act like many computers. When you are using AWS, you can shut off access to resources with the same speed and agility as when you requested and started the resources, with an accompanying reduction in the billing for these resources.

EC2 was first developed to be used in Amazon’s internal infrastructure. It was the idea of Chris Pinkham. He was head of Amazon’s worldwide infrastructure from around 2003. Amazon released a limited beta test of EC2 to the public on August 25, 2006, providing limited trial access. In October 2007, Amazon expanded its offerings by adding two new types of instances (Large and Extra-Large), and in May 2008, two additional instance types were added to the service (High-CPU Medium and High-CPU Extra Large).

Amazon EC2 is probably the most essential and critical service on the AWS cloud computing platform. If you are trying to create something using AWS, and no other service offers the functionality you desire, you will probably be able to use EC2 as the foundation for your project. You can think of the EC2 service as a computer of almost any size and capability that you can turn on or off at any point and stop being charged when you shut it down.

We briefly mentioned that EC2 allows various computer sizes and capabilities. Let’s revisit that idea and see how many choices are available. Amazon EC2 delivers a large variety of instance types. Each type addresses different needs and is optimized to fit a specific use case. Instance types are defined by a combination of their memory, CPU, GPU, storage, and networking capabilities. Each different type can provide a sweet spot for your individual use case. Each EC2 instance type that Amazon provides has a different size, allowing you to match the right instance size with your target workload.

As better CPU cores and memory chips become available, AWS continually improves its EC2 offerings to take advantage of these new components. The Amazon EC2 service is constantly improving. Staying on top of the constant changes can be quite challenging.

For example, when the T instance types were launched, AWS called them T1, but now better instances such as T4 instances types are available as AWS keeps evolving its offerings. To see the complete list of instance types offered by AWS, see the *EC2 instance families* section later in this chapter.

In general terms, the EC2 instance types and their classifications have remained unchanged, but each type’s models and sizes continue to evolve. What used to be the *top-shelf* offering last year might be a medium-level offering this year due to improvements to the underlying components. Depending on budgetary constraints and workload needs, different models and sizes might offer the optimal solution for your project.

AWS supports various processors for their EC2 instances, such as Intel, AMD, NVIDIA, and their own homegrown processor Graviton. Let’s take a quick peek at the Graviton processor.

**AWS Graviton**

From a processor perspective, for AWS, the longest-standing processor choice is Intel. Since 2006, Intel processors have been essential to powering AWS’s most powerful instances. AWS recently released the latest generation of Intel instances with the second generation of Xeon Scalable for compute-intensive workloads. In 2018, AWS launched AMD processor-based instances, and at re:Invent 2018, AWS launched its own processor, an Arm-based Graviton processor. For GPU, AWS uses an NVIDIA chip to provide choice for training machine learning models.

One of the essential things that keeps AWS ahead of the game is their innovation. When AWS launched its processor in 2018, it was a masterstroke that helped them to gain a lead over other cloud providers. Graviton is an Arm chip from Annapurna Labs, a chip design company in Israel that AWS acquired in 2015.

The most important thing about having your own Graviton processor is that it is custom-built to suit your needs. To complement the Graviton processor, AWS launched a Nitro hypervisor as the backbone, and now they have an entire infrastructure system optimized for the cloud-native workload. It resulted in an added advantage for AWS as they can offer low prices and high performance for workload-focused custom instances.

EC2 A1 instances are based on the first generation of Graviton processors, with up to 16 vCPUs, 10 Gbps enhanced networking, and 3.5 Gbps EBS bandwidth. In 2019, AWS launched Graviton 2, which is their second-generation chip. These new AWS Graviton 2 processors have 4x the number of compute cores, deliver up to 7x better performance, have more support for machine learning, video, and HPC, and offer 5x faster memory compared to the Graviton 1.

The Amazon EC2 M6g, C6g, and R6g instances are powered by AWS Graviton 2 processors, which offer a significant improvement in price performance compared to current-generation x86-based instances for various workloads, such as web and application servers, media processing, and machine learning. This translates to up to 40% cost savings while maintaining high performance levels, making these instances an excellent choice for a wide range of computing requirements. These processors are designed for use in AWS’s cloud computing services and are intended to provide even higher performance and lower cost than the first generation of Graviton processors. Now AWS has launched Graviton 3 processors, which have even better (25% better) compute performance than Graviton 2 processors.

Graviton processors are not available for all EC2 instances, but wherever possible it is recommended to use Graviton-backed instances for better price and performance. Let’s learn more about the advantages of EC2.

**Advantages of EC2**

Amazon EC2 has a vast selection of instances for you to choose from. Let’s look at some of them and understand the key advantages of using EC2:

* **The diverse set of instances to choose from**

EC2 offers more than 400 instances to enable customers to run virtually every workload. You can choose from instances that offer less than a single processor (T3 instances) and up to 96 processors (C5 instances). Memory is available from hundreds of megabytes (T3 instances) to 24 terabytes (High Memory instances), the most memory of any major IaaS provider. Network performance ranges from a single gigabit (T2 instances) to 100 gigabits (Elastic Fabric Adapter).

EC2 instances can be built with extreme low-latency NVMe and massively scaled remote and block-level storage using Elastic Block Store. These instances can be built with up to eight peer-to-peer (P3 instances) connected GPUs or even a dedicated FPGA (F1 instances). EC2 also offers many instances as bare metal, with no hypervisor presence, for the most demanding performance environments or customers who intend to use their hypervisor.

EC2 even offers Apple Mac systems with macOS the highest frequency (M5 instances). AWS offers Intel Xeon Scalable processors in the cloud for high-performance computing (HPC) workloads, you can have the most local storage in the cloud with D3 and D3en instances, you can use G4 instances with the best economics for graphics processing in the cloud, and you can get extreme performance for network storage bandwidth, up to 60 Gbps, with R5 instances. AWS offers four times more EC2 instance types than it did just a couple of years ago. AWS provides Arm-based host CPUs (Graviton 2) built by AWS and customized for various workload needs, such as machine learning inference. You will learn more about these instances in the *EC2 instance families* section.

* **Scalability**

You can scale up from a single system to thousands as your business needs grow. What if you no longer need the systems? You can scale back as quickly as you scale up. If you don’t know how much capacity you will need, EC2 offers auto-scaling. Auto-scaling helps to scale out EC2 instances automatically based on defined metrics. For example, you can add two EC2 instances if your instance memory utilization reaches 60% or CPU utilization reaches 70%. You can learn more about auto-scaling in the AWS docs at <https://aws.amazon.com/autoscaling/>.

* **Performance**

AWS Nitro gives EC2 significant and unique performance advantages. Generally, cloud server instances across the industry are given to customers as VMs because there is a hypervisor between the physical hardware and the virtual system. This abstraction layer provides flexibility and ease of use but can also negatively impact performance. In the case of EC2, virtualized instances use the Nitro hypervisor and Nitro offload cards.

The Nitro hypervisor has been optimized for cloud instances, improving performance characteristics. Nitro takes many networks, storage, security, and system management functions.

It offloads them from the system’s CPU, relieves the CPU so it can perform better on customers’ workloads, and improves the actual speed of those functions because they run on dedicated ASICs rather than a general-purpose CPU**,** which is a significant overall differentiator for EC2. You can learn more about the Nitro hypervisor by visiting the AWS page: <https://aws.amazon.com/ec2/nitro/>.

* **Reliability**

EC2 is built on 24 Regions and over 76 global Availability Zones and serves millions of customers ranging from large enterprises to start-ups. EC2 has an SLA commitment of 99.99% availability for each Region. Overall, this makes AWS EC2 very reliable.

* **Security**

Every EC2 instance is built by default to be secure. Client systems that are allowed to connect to EC2 must also meet strict security standards such as PCI, SOC, and FedRAMP. If you are connecting to the AWS cloud on-premises, you can create entire networks under IPsec to define a logically isolated section of AWS using a **Virtual Private Network** (**VPN**). One example of a hardware and software innovation is AWS Nitro. It gives EC2 differentiated security experiences, taking per-system security functions off the main host processor and running them on a dedicated processor, resulting in individual system protection and a minimized attack surface because security functions run on a dedicated chip rather than the system’s host CPU. You can learn more about it on the Nitro Enclaves page: <https://aws.amazon.com/ec2/nitro/nitro-enclaves/>.

Let’s look at EC2 instance families in more detail.

**EC2 instance families**

AWS provides more than 400 types of instances based on processor, storage, networking, operating system, and purchase model. As per your workload, AWS offers different capabilities, processors, platforms, instances, and more. All these options allow you to select the best instances for your business needs. As of June 2022, these are the instance types that Amazon offers:

* General Purpose (A1, M6, T4, Mac)
* Compute Optimized (C7, Hpc6a)
* Accelerated Computing (P4, G5, F1)
* Memory Optimized (R6, X2, High Memory)
* Storage Optimized (H1, D3, I4)

These are EC2 instances in a broad five-instance family, highlighting only the latest generation instances in parentheses. You can look at all older version instances in each family by visiting <https://aws.amazon.com/ec2/instance-types/>.

Instance type names combine the instance family, generation, and size. AWS provides clues about instance families through their naming convention, which indicates the key characteristics of the instance family. These are prefixes or postfixes for various instance types:

* a – AMD processors
* g – AWS Graviton processors
* i – Intel processors
* d – Instance store volumes
* n – Network optimization
* b – Block storage optimization
* e – Extra storage or memory
* z – High frequency

AWS has a standard naming convention for instance types. For example, if the instance name is c7g.8xlarge, the first position indicates the instance family, c. The second position indicates the instance generation, in this case, 7. The remaining letters before the period indicate additional capabilities, such as g for a Graviton-based instance. After the period is the instance size, which is a number followed by a size, such as 8xlarge. You can learn more about instance naming in the AWS user docs: <https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/instance-types.html>.

Let’s look at AWS’s five instance families in more detail.

**General Purpose (A1, M6, T4, Mac)**

General Purpose instances balance CPU, memory, and network resources, providing a balance between cost and functionality. They are ideal for running web servers, containerized microservices, caching fleets, and development environments. Until you have identified the specific needs for your application workload in terms of compute, storage, networking, and so on, it’s a good idea to start with a General Purpose instance. One of the main distinctions within this class is between instances with fixed (such as M5a) and burstable (such as T3a) performance.

**T family instance** types include T2/T3/T4 EC2 instances. Theseare AWS workhorse instances. The T family of instance types is the most popular among General Purpose compute instances. AWS provides t2.micro in their free tier, so most people starting with AWS tend to pick a T instance as their first choice.

T4g uses a Graviton 2 processor, the T2/T3 family uses Intel chips, and T3a uses AMD chips with burstable performance. Burstable performance is used when you have uneven workloads that require an occasional boost. This burst in capacity can be used to scale up compute quickly.

You need to earn CPU credits for bursts. When the CPU is not running at maximum capacity, CPU credits are earned. The credits are used when there is a need for a boost or burst. A CPU credit gives you the performance of a full CPU core for one minute.

Here’s an example to explain the concept of burstable credits better. One example of a burstable instance is the t2.small instance, which is designed for workloads that do not require a consistently high level of CPU performance. The t2.small instance has a baseline performance level of 20% of a CPU core, but it can burst to higher levels of performance when needed.

When using a t2.small instance, you are allocated a certain number of burstable credits that can be used to burst to higher levels of performance. These credits are replenished at a rate of 24 credits per hour, and they are used at a rate of 1 credit per minute of bursting. For example, if your t2.small instance is idle and not using any CPU resources, it will accrue 24 credits per hour, which can be used to burst to higher levels of performance when needed. If your t2.small instance is running a workload that requires a higher level of CPU performance, it will use up its burstable credits at a rate of 1 credit per minute, and once all of the credits are used up, the instance will operate at its baseline performance level until more credits are accumulated.

If the credit allotment is exceeded, this will result in an extra charge to your bill, but it will still enable you to process requests and handle the traffic. Suppose you consistently find yourself paying for burstable traffic. In that case, it is probably an indication that it is time to add more instances to your architecture or move your application to a more powerful instance.

**M family instance** types include M4/M5/M6 EC2 instances, which are similar to T family instances. They are also General Purpose instances and deliver a good balance between compute, memory, and network resources. The latest M6 instances are available across all three processors: Intel Xeon (M6i instances), Graviton 2 (M6g instances), and AMD (M6a instances).

**A1 instances** are the first EC2 instances powered by Graviton processors. Amazon EC2 A1 instances offer good value for the amount of functionality supplied. They are often used in scale-out workloads as they are well suited for this type of job. A1 instances use an Arm chip. Therefore, this type is more suitable for application development that runs open-source languages such as Java, Ruby, and Python.

Mac instances are built on the AWS Nitro system and are powered by Apple Mac mini computers with Intel Core i7-8700 processors. This EC2 family gives you access to macOS to develop, test, and sign applications that require Apple’s Xcode IDE.

Let’s continue our journey through the different instance types offered by AWS.

**Compute Optimized (C7, Hpc6a)**

Compute Optimized instances, such as the C series instances, are designed for applications that are highly compute-intensive and require a high level of CPU performance. These instances are equipped with powerful processors, such as Intel Xeon Scalable processors, and offer a high ratio of CPU to memory to support demanding workloads.

Compute Optimized instances are well suited for a variety of workloads, including **high-performance computing** (**HPC**) applications, video encoding and transcoding, machine learning and deep learning, and other applications that require a high level of CPU performance. These instances can help users achieve high levels of performance and efficiency, making them an excellent choice for applications that are compute-intensive.

**C family instance** types include C4/C5/C6 and the latest C7 instances.These instance types are ideal for compute-intensive applications and deliver cost-effective high performance. C instances are a good fit for applications that require raw compute power. The latest C7g instance uses a Graviton 3 processor and C6 instances are available across all three processors: Intel Xeon (C6i instances), Graviton 2 (C6g/C6gn instances), and AMD (C6a instances) processors. Benchmarks show that Graviton-based C6 instances achieve 40% better price performance than the previous family of C5 instance types.

**Hpc6a instances** are optimized for high-performance computing workloads that are compute intensive. Hpc6a is available in a low-cost AMD processor and delivers network performance up to 100 Gbps with the help of an **Elastic Fabric Adapter** (**EFA**) for inter-node network bandwidth. These instances are designed for the following workloads:

* Molecular dynamics
* Weather forecasting
* Computational fluid dynamics

Let’s learn about another popular instance type – the Accelerated Computing family of instances.

**Accelerated Computing (P4, G5, F1)**

Accelerated Computing instances include additional hardware dedicated to the instance, such as GPUs. For General Purpose, Accelerated Computing, or graphics-intensive computing, FPGAs, or inferencing chips, provide massive amounts of parallel processing for tasks such as graphics processing, machine learning (both learning and inferencing), computational storage, encryption, and compression. These instances have hardware accelerators that enable them to evaluate functions, such as floating-point number calculations, graphics, modeling, and complex pattern matching, very efficiently.

**P family instance** types include P2 NVIDIA K80/P3 V100/P4 NVIDIA A100 Tensor Core instances that can deliver high performance with up to 8 NVIDIA V100 Tensor Core GPUs and up to 400 Gbps of networking throughput. P4 instances can dramatically reduce machine learning training, sometimes from days to minutes.

**G family instance** types include G3/G4/G5 instances. G instances are cost-effective and versatile GPU instances that enable the deployment of graphics-intensive programs and machine learning modeling. G5 instances are optimized for machine learning workloads that use NVIDIA libraries. The NVIDIA A10G for G5 is often used for graphics applications and 3D modeling.

**F1** instances rely on FPGAs for the delivery of custom hardware accelerations. A **Field-Programmable Gate Array** (**FPGA**) is an **Integrated Circuit** (**IC**) that is customizable in the field for a specific purpose. A regular CPU is burned at the factory and cannot be changed once it leaves the factory floor. An example of this is the Intel Pentium chip. Intel manufactures millions of these chips, all of them precisely the same. FPGAs are field-programmable, meaning the end-user can change them after they leave the factory. FPGAs can be customized for individual needs and burned by the customer.

The AWS Inferential-based Inf1 instance is a custom-designed AWS chip optimized for running deep learning inference workloads. Inf1 is ideal for customers with large amounts of image, object, speech, or text data and runs substantial machine learning on that data. Inf1 instances deliver up to 2.3x higher throughput and up to 80% lower cost per inference than Amazon EC2 G4 instances.

AWS has launched the following Accelerated Computing instances to address the specific needs of machine learning workloads:

* **DL1** – Built for training deep learning models based on an Intel Xeon processor.
* **Trn1** – Built for training deep learning models powered by AWS Trainium. They are useful for search, recommendation, ranking, machine learning training for natural language processing, computer vision, and more.
* **VT1** – Optimized for low-cost real-time video transcoding. They are useful for live event broadcasts, video conferencing, and just-in-time transcoding.

The next instance type family we will learn about is Memory Optimized instances.

**Memory Optimized (R6, X2, High Memory)**

Memory Optimized instances are used for anything that needs memory-intensive applications, such as real-time big data analytics, in-memory databases, enterprise-class applications that require significant memory resources, or general analytics such as Hadoop or Spark. These instances can deliver fast performance by allowing us to load large, complete datasets into memory for processing and transformation.

**R family instance** types include R4/R5/R6 instances.**R6i** uses the Intel chip, and R6g uses the Graviton 2 chip. These instances are best suited for memory-intensive applications. The R6 instance types use the AWS Nitro System, which reduces costs compared to its competitors.

**X1** and **X2** instances deliver a high ratio of memory to compute. AWS launched a diverse set of X2 instances across processors such as Graviton 2 (X2gd) and Intel Xeon (X2idn/X2). X2 instances offer up to 50% better price-performance than X1 instances. The X1e type delivers the highest memory-to-compute ratio of all EC2 instance types.

**High Memory instances** deliver the most significant amount of available RAM, providing up to 24 TB of memory per server. Like X2 instances, High Memory instances are best suited for production environments of petabyte-scale databases.

In addition to that, AWS offers **z1d instances**,which provide both high compute capacity and a high memory footprint for **electronic design automation** (**EDA**).

Let’s look at the final category of Storage Optimized instances.

**Storage Optimized (H1, D3, I4)**

Storage Optimized instances are ideal for tasks requiring local access to large amounts of storage, extreme storage performance, or both. Instances include both a large-capacity HDD and an extremely low-latency local NVMe. And don’t forget, all EC2 instances have access to Amazon Elastic Block Store for block-level storage at any scale. These instances are best suited for workloads that need massive storage, particularly sequential read-write-like log analysis.

The **H1** and **D3** instance types form part of the dense storage family of servers that can supply sequential reads and writes with petabyte-scale datasets. These instances provide storage on HDDs. H1 instances can supply up to 16 TB, and D3 can supply up to 48 TB. Compared to EC2 D2 instances, D3 instances offer significantly faster read and write disk throughput, with an improvement of up to 45%. This enhanced disk performance enables faster data processing and storage, making D3 instances a compelling option for workloads that require high-speed access to data.

The latest **I4 instances**launched across diverse processors such as Graviton 2 (Im4gn/Is4gen). I4 instances provide SSD storage up to 30 TB while supplying lower latency than HDD-based storage. I4 instances deliver markedly improved I/O latency, with up to 60% lower latency and a reduction of up to 75% in latency variability when compared to I3 and I3en instances. Additionally, I4 instances come with always-on encryption, ensuring the security of data at rest. These performance and security enhancements make I4 instances an attractive choice for workloads that require low-latency, high-throughput I/O operations, such as data warehousing, data processing, and other demanding enterprise applications.

Amazon will likely continue to enhance its EC2 service by offering new instance types and improving the current ones. You can use AWS’s new **EC2 Instance Type Explorer** to keep yourself up to date with new EC2 instance offerings. EC2 Instance Type Explorer helps you navigate and discover the right instances for your customers. Use filters to narrow down the instance family by category or hardware configuration quickly. You can access it using <https://aws.amazon.com/ec2/instance-explorer>.

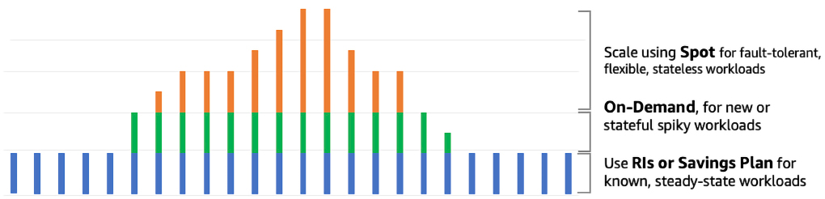
You have now learned about the various EC2 types. As cost is one of the main factors when it comes to the cloud, let’s learn more about the EC2 pricing model.

**EC2 pricing model**

While the standard cloud price model is the pay-as-you-go model, AWS provides multiple options to further optimize your costs. As servers are a significant part of any IT infrastructure, it is better to understand all the available cost options to get the most out of your dollar. The following are the four different ways to purchase compute in AWS:

* **On-Demand**: Pay for compute capacity by the second without any long-term commitment. It is best suited for fluctuating workloads, for example, stock trading or e-commerce website traffic. It is the default choice when you spin up an instance and is also suitable for quick experiments.
* **Reserved Instance (RI)**: You can commit 1 or 3 years to a specific EC2 instance family and receive a significant discount of up to 72%off On-Demand prices. This is best for a steady workload that you know will not fluctuate much, for example, an internal HR portal. An RI is like a coupon: you pay in advance, and it applies automatically when your spin-up instance belongs to the same EC2 instance family for which you pay the RI price. AWS also provides **Convertible RIs**, where you can exchange one or more Convertible RIs for another Convertible RI with a different configuration, including instance family, operating system, and tenancy. The new Convertible RI must be of an equal or higher value than the one you’re exchanging. You can find details on Reserved Instance pricing in AWS at <https://aws.amazon.com/ec2/pricing/reserved-instances/pricing/>.
* **Savings Plan**: This is like an RI, but monetary commitment and compute can be used across Fargate, EC2, and AWS Lambda. In a savings plan, you don’t have to make commitments to specific instance configurations but commit to a spending amount. You can get significant savings, up to 72% off On-Demand instances, with the flexibility to apply it across instance families. AWS has two types of Savings Plans: EC2 Instance Savings Plans and Compute Savings Plans.
* **Spot Instances**: Same as the pay-as-you-go pricing model of On-Demand, but at up to 90% off. EC2 can reclaim Spot Instances with a 2-minute warning. They are best for stateless or fault-tolerant workloads. You can leverage the scale of AWS at a fraction of the cost with a simplified pricing model. A Spot Instance is only interrupted when EC2 needs to reclaim it for On-Demand capacity. You don’t need to worry about your bidding strategy. Spot prices gradually adjust based on long-term supply and demand trends.

All four purchasing options use the same underlying EC2 instances and AWS infrastructure across all Regions. You can combine multiple options to optimize the cost of your workload. As shown in the following diagram, you can use auto-scaling to use all four options to optimize cost and capacity.



*Figure 6.1: EC2 pricing model to optimize cost*

Let’s take an example of an e-commerce website’s workload. As shown in the preceding diagram, for daily traffic patterns, you can use an RI, with which you can save up to 72% compared to On-Demand instances. But if you run a deal, such as 20% off Apple products, and get a sudden spike, then auto-scaling spins up On-Demand instances to handle the spike. At the end of the day, when you are processing orders for fulfillment, you can use a Spot Instance to expedite the order queue with 90% savings compared to an On-Demand instance.

There are so many choices in EC2 that you can get confused easily, and to address that problem, AWS provides Compute Optimizer.

**AWS Compute Optimizer**

AWS offers Compute Optimizer, a service that suggests the most appropriate instances from over 140 options available in the M, C, R, T, and X families for Amazon EC2 and Amazon EC2 Auto Scaling groups. By analyzing resource utilization patterns, Compute Optimizer can identify the most cost-effective and efficient instances to help optimize application performance and reduce costs. It uses machine learning models trained on millions of workloads to help customers optimize their compute resources for cost and performance across all the workloads they run. The following are the benefits of using Compute Optimizer:

* Get instance type and Auto Scaling group recommendations, making it easier for you to choose the right compute resources for specific workloads.
* Get a deep analysis of a workload’s configuration, resource utilization, and performance data to identify a range of defining characteristics, such as whether the workload is CPU intensive and exhibits a daily pattern. Compute Optimizer then uses machine learning to process these characteristics to predict how the workload would perform on various hardware platforms, delivering resource recommendations.
* Get up to three recommended options for each AWS resource analyzed to choose the correct size and improve workload performance. Compute Optimizer predicts your workload’s expected CPU and memory utilization on various EC2 instance types. Compute Optimizer provides the added benefit of allowing you to assess the performance of your workload on the recommended instances before you adopt them. This helps to reduce the risk of unforeseen issues arising from transitioning to new instance types and enables you to make informed decisions regarding which instances to choose for your workload. By simulating performance metrics, Compute Optimizer helps you gauge the effectiveness of recommended instance types and provides a level of confidence in the decision-making process.

When you spin up an EC2 instance, the first thing to select is an **Amazon Machine Image** (**AMI**) to decide which operating system you want to use. Let’s learn more about AMIs.

**Amazon Machine Images (AMIs)**

Even though there are so many EC2 instance types to choose from, the number of instance types pales in comparison to the number of AMIs available. An AMI contains the information needed to start an instance. An AMI needs to be specified when launching an instance.

The chosen AMI will determine the characteristics of the EC2 instance, such as the following:

* **Operating system**: The currently supported operating systems are as follows:
  1. Ubuntu
  2. Amazon Linux
  3. CentOS
  4. Debian
  5. Red Hat Enterprise Linux
  6. FreeBSD
  7. SUSE
  8. Fedora
  9. Gentoo
  10. macOS
  11. Mint
  12. OpenSolaris
  13. Windows Server
* **Architecture**: The architecture that will be used:
  1. 64-bit (Arm)
  2. 32-bit (x86)
  3. 64-bit (x86)
  4. 64-bit (Mac)
* **Launch permissions**: The launch permissions will determine when and where the AMI can be used:
  1. Public: All AWS accounts can launch this AMI.
  2. Explicit: Only specific AWS accounts can launch the AMI.
  3. Implicit: Implicit launch permission is given to launch the AMI.
* **Root device storage**: Another option that can be specified when choosing an AMI is how the data in the root device is persisted. The options include the following:
  1. Amazon EBS: Uses an Amazon EBS volume launched using an Amazon EBS snapshot as its source
  2. Instance store: Uses an instance store volume launched from a template store in S3

Multiple instances can be launched from a single AMI. This is useful when multiple instances need to be launched with the same configuration. It does not matter if you need one instance or a thousand instances. They can be launched with the same effort by clicking a few buttons.

An AMI comprises the following:

* An EBS snapshot or a template (in the case of an instance-backed AMI) for the root volume for an EC2 instance. For example, an operating system, an application, or a server.
* Launch permissions that can be used to control the AWS accounts that will be allowed to use the AMI to generate new instances.
* A block device mapping that specifies which volumes need to be attached to the instance when it is started.

AWS enables running thousands of AMIs. Some are AMIs created by AWS, the AWS community creates some, and some are offerings from third-party vendors.

You have learned about the vast number of options available when creating and launching Amazon EC2 instances. Now let’s explore the best practices to optimize the Amazon EC2 service.

**Reviewing Amazon EC2 best practices**

How you use and configure EC2 is going to depend on your use case. But some general EC2 best practices will ensure the security, reliability, durability, and availability of your applications and data. Let’s delve into the recommended practices for handling security, storage, backup management, and so on.

**Access**

Like with almost any AWS service, it’s possible to manage the access and security of your EC2 instances, taking advantage of identity federation, policies, and IAM. You can create credential management policies and procedures to create, rotate, distribute, and revoke AWS access credentials.

You should assign the least privilege possible to all your users and roles, like any other service. As they say in the military, your users should be given access on a *need-to-know* basis.

One advantage or disadvantage of using EC2 directly is that you are entirely in charge of managing the OS changes. For that reason, ensure that you regularly maintain and secure the OS and all the applications running on your instance.

**Storage**

One advantage of using EC2 instances is that you will stop getting charged when you shut down the instance. The reason that AWS can afford to do this is the fact that once you shut the instance down, AWS then reassigns the resource to someone else. But what does this mean for any data you had stored in the instance? The answer lies in understanding the difference between instance stores and EBS-backed instances.

When you launch an EC2 instance, by default, it will have an instance store. This store has high performance. However, anything that is persisted in the instance store will be lost when you shut down the instance.

**IMPORTANT NOTE**

If you want data to persist after you shut down the instance, you will need to ensure that you include an EBS volume when you create the instance (or add it later) or store the data in an S3 bucket. If you store the data in the instance store attached by default to the EC2 instance, it will be lost the next time it is shut down.

**EBS volumes** are mountable storage drives. They usually deliver lower speed and performance than instance stores but have the advantage that the data will persist when you shut down the instance.

It is also recommended to use one EBS volume for the OS and another EBS volume for data.

If you insist on using the instance store for persistence purposes, make sure there is a plan in place to store the data in a cluster of EC2 instances with a replication strategy in place to duplicate the data across the nodes (for example, using Hadoop or Spark to handle this replication).

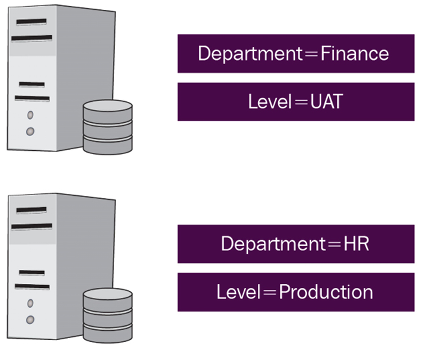
**Resource management**

When launching an EC2 instance, AWS can include instance metadata and custom resource tags. These tags can be used to classify and group your AWS resources.

Instance metadata is data specified for an instance that can then be used to customize, label, and maintain the instance. Instance metadata can be classified into topics such as the following:

* The name of the host
* Events
* Security groups
* Billing tags
* Department or organizational unit tags

The following diagram illustrates the basics of tagging your EC2 instances:



*Figure 6.2: EC2 instance tags*

In this example, two tags are assigned to each one of the instances—one tag is given the key **Department**, and another is given the key **Level**. In this example, you can identify and consolidate the HR and finance department workloads for billing, automation, or to apply tag-based security. You can learn more about tag strategies by referring to the AWS docs at <https://docs.aws.amazon.com/general/latest/gr/aws_tagging.html>.

Tags are a powerful yet simple way to classify EC2 instances. They can help in development, code testing, environment management, and billing. Every tag also has a corresponding value.

**Limit management**

By default, AWS sets limits for a variety of parameters for Amazon EC2 (as well as for other services). If you are planning to use Amazon EC2 in production environments, you should get intimately familiar with those limits.

AWS enforces service limits to safeguard against unanticipated over-provisioning and malicious actions intended to inflate your bill, and to protect service endpoints. These limits are put in place to prevent unexpected charges from excessive resource usage and to mitigate potential security risks that may result from unauthorized access to service endpoints. By implementing these limits, AWS helps to ensure the reliability, availability, and security of its services and provides customers with greater control over their usage and expenditure. AWS has the concepts of soft limit and hard limit to save you from incurring accidental costs. For example, Amazon VPC has a soft limit of 5 and a hard limit of 100 per Region, which can be increased after putting in a request to AWS.

Knowing the limits will allow you to plan when to ask for a limit increase before it becomes critical. Increasing the limits involves contacting the AWS team. It may take a few hours or a few days to get a response. These are some of the default limits for an AWS account:

* 20 instances per Region
* 5 Elastic IPs per Region (including unassigned addresses)

Keep in mind that these soft limits can be increased by contacting AWS. You can learn about AWS’ service limits by referring to <https://docs.aws.amazon.com/general/latest/gr/aws_service_limits.html>.

**EC2 backup, snapshots, and recovery**

When it comes to backing up EC2, you have two main components: EBS volumes and AMIs. It is important to have a periodic backup schedule for all EBS volumes. These backups can be performed with EBS snapshots. You can also build your own AMI by using an existing instance and persisting the current configuration. This AMI can be used as a template to launch future instances.

To increase application and data availability and durability, it is also important to have a strategy to deploy critical application components across multiple Availability Zones and copy the application’s data across these Availability Zones.

Make sure that your application’s architecture can handle failovers. One of the simplest solutions is to manually attach an Elastic IP address or a network interface to a backup instance. Elastic IP provides a fixed IP address to your instance, where you can route traffic. An Elastic IP can be assigned to an **Elastic Network Interface** (**ENI**), and those ENIs can be easily attached to instances. Once you have the solution in place, it is highly recommended to test the setup by manually shutting down the primary server. This should fail the traffic over to the backup instance, and traffic should be handled with little or no disruption.

While all of these best practices will help you to get the most out of EC2, one particularly useful way to manage and distribute traffic among EC2 instances is to use AWS’s load balancer service.

**Amazon Elastic Load Balancing**

**Elastic Load Balancing** (**ELB**) in AWS allows you to assemble arrays of similar EC2 instances to distribute incoming traffic among these instances. ELB can distribute this application or network traffic across EC2 instances or containers within the same AZ or across AZs.

In addition, to help with scalability, ELB also increases availability and reliability. A core feature of ELB is the ability to implement health checks on the managed instances. An ELB health check determines the *“health”* or availability of registered EC2 instances and their readiness to receive traffic. A health check is simply a message or request sent to the server and the response that may or may not be received. If the instance responds within the 200 range, everything is fine. Any other response is considered *“unhealthy”*. If an instance does not return a healthy status, it is considered unavailable, and ELB will stop sending application traffic to that instance until it returns to a healthy status. To learn more about return statuses, see <https://docs.aws.amazon.com/elasticloadbalancing/latest/classic/ts-elb-http-errors.html>.

Before we delve into the nitty-gritty of the different types of ELB services, let’s understand some fundamental concepts.

**ELB rules**

ELB rules are typically comprised of one or more conditions and one or more actions. The conditions specify the criteria that must be met for the rule to be applied, while the actions define what should be done with the request if the rule’s conditions are met.

For example, a listener rule may have a condition that requires traffic to be received on a specific port and protocol, and an action that directs that traffic to a specific target group. Similarly, a path-based routing rule may have a condition that requires a request’s path to match a specific value, and an action that directs that request to a different target group.

In some cases, you may also be able to specify a priority for your rules, which determines the order in which they are evaluated. This can be important if you have multiple rules that apply to the same traffic and you need to ensure that the correct rule is applied in a specific order. Let’s go into more detail about the parts that make up a rule:

* **Conditions** – A condition is a regular expression indicating the path pattern that needs to be present in the request for the traffic to be routed to a certain range of backend servers.
* **Target groups** – A target group is a set of instances. Whenever a condition is matched, traffic will be routed to a specific target group to handle requests. Any of the instances in the group will handle the request. Target groups define a protocol (HTTP, HTTPS, FTP, and others) and a target port. A health check can be configured for each target group. There can be a one-to-many relationship between ALBs and target groups. Targets define the endpoints. Targets are registered with the ALB as part of a target group configuration.
* **Priorities** – Priorities are definitions to specify in which order the ALB will evaluate the rules. A rule with a low number priority will have higher precedence than a high one. As the rules are evaluated by priority, the rules get evaluated. Whenever a pattern is matched in a rule, traffic is routed to a target group, and the evaluation stops.

Like many other AWS services, an ELB can be created and configured via the AWS console, the AWS CLI, or the Amazon API. Let’s look at some specific ELB rules.

**Listener rules**

A listener rule is a type of rule that defines how incoming traffic is forwarded to a target group by an ELB based on the protocol and port of the incoming request. Here’s an example of how a listener rule works. Suppose you have a web application running on a group of EC2 instances and you want to distribute incoming traffic across these instances using an ELB. You can create a listener rule that specifies which target group the traffic should be forwarded to based on the protocol and port of the incoming request.

For example, you could create a listener rule that forwards all HTTP traffic on port 80 to a target group named “web-servers.” The listener rule would specify the protocol (HTTP), port (80), and target group (“web-servers”). Any incoming HTTP requests on port 80 would then be automatically forwarded to the EC2 instances in the “web-servers” target group. A listener rule provides a flexible way to route incoming traffic to the appropriate target group based on specific criteria. By creating multiple listener rules, you can route traffic to different target groups based on the protocol and port of the incoming request.

**Target group rules**

Target group rules are a type of rule in ELB that determines how traffic is distributed across the targets within a target group. Target groups are groups of resources that receive traffic from a load balancer, such as Amazon EC2 instances or containers. Target group rules help you manage how traffic is distributed across these resources. Here’s an example of how target group rules work:

Suppose you have a target group named “web-servers” containing a group of EC2 instances running your web application. You can create a target group rule that specifies how incoming traffic is distributed across these instances based on specific criteria, such as response time or connection count.

For example, you could create a target group rule that distributes traffic based on the response time of each instance. The target group rule would evaluate the response time of each instance and send more traffic to the instances with the lowest response times. This helps ensure that the load is balanced evenly across the EC2 instances, and that users experience consistent performance when accessing your web application.

Target group rules provide a powerful way to manage traffic distribution across your resources. By creating multiple target group rules, you can fine-tune how traffic is distributed across your resources based on specific criteria, helping you optimize performance and ensure high availability for your applications.

**Host-based routing rules**

**Host-based routing** enables the routing of a request based on the host field, which can be set in the HTTP headers. It allows routing to multiple services or containers using a domain and path.

Host-based routing provides the ability to transfer more of the routing logic from the application level, allowing developers to focus more on business logic. It allows traffic to be routed to multiple domains on a single load balancer by redirecting each hostname to a different set of EC2 instances or containers. These rules allow you to direct incoming requests to different target groups based on the hostname in the request URL. For example, you can create a host-based routing rule that directs all requests with a certain hostname to a specific target group.

**Path-based routing rules**

Simply put, path-based routing is the ability to route traffic from the ELB to particular instances on the ELB cluster based on a substring in the URL path. These rules allow you to direct incoming requests to different target groups based on the path of the request URL. For example, you can create a path-based routing rule that directs all requests with a certain path to a specific target group. An example would be the following path-based routing rules:

/es/\*

/en/\*

/fr/\*

\*

We could use these rules to forward the traffic to a specific range of EC2 instances. When we deploy our servers, we could ensure that the en servers are using English, the es servers have the Spanish translation, and the fr servers use French strings. This way, not only will the users be able to see our content in their desired language, but the load will be distributed across the servers. In this example, it might be beneficial to monitor traffic by language constantly and deploy enough servers for each language. Even more powerfully, we could create rules that automatically launch new servers for the different language clusters based on demand by language.

**Query string rules**

Query string rules are a type of rule in ELB that allow you to route traffic based on the query string parameters of incoming requests. The query string is a portion of a URL that contains key-value pairs separated by an ampersand (&) symbol, and is commonly used in web applications to pass information between the client and server. Here’s an example of how query string rules work. Suppose you have a web application that uses a query string parameter called “category” to identify the type of content to display. You want to route traffic to different target groups based on the value of this parameter. For example, you want traffic with the “category” parameter set to “books” to be routed to a target group containing your book servers, while traffic with the “category” parameter set to “movies” is routed to a target group containing your movie servers.

Query string rules provide a powerful way to route traffic based on the content of incoming requests. By creating multiple query string rules, you can route traffic to different target groups based on different query string parameters, allowing you to manage your resources more efficiently and provide a better experience for your users.

Now that we understand more about ELB rules, let’s have a look at the different types of ELBs that AWS offers.

**Elastic Load Balancer types**

In August 2016, AWS launched a new service called **ALB** (**Application Load Balancer**). ALB allows users of the service to direct traffic at the application level.

The old Elastic Load Balancer service offering can still be used, and it was renamed Classic Load Balancer. Also more types of ELB were later launched, named the Network Load Balancer and Gateway Load Balancer. This section will try to understand their differences and when to use one versus the other.

**Classic Load Balancers**

The **Classic Load Balancer** (**CLB**) in AWS can route traffic using the following methods:

* **Round robin**: In this method, the load balancer routes traffic to each registered instance in turn, providing an equal share of the traffic to each instance. Round Robin is the default load balancing algorithm for CLBs.
* **Least connections**: In this method, the load balancer routes traffic to the instance with the fewest active connections. This method is useful for balancing traffic across instances with varying processing capacities.
* **IP hash**: In this method, the load balancer routes traffic to a particular instance based on the source IP address of the client. This method ensures that traffic from the same source IP address is consistently routed to the same instance, which can be useful for stateful applications.

In addition to these load balancing methods, Classic Load Balancers can also route traffic based on protocols, ports, and health checks. For example, you can configure the load balancer to route traffic based on the protocol and port used by the client (such as HTTP or HTTPS on port 80 or 443), and perform health checks on each registered instance to ensure that only healthy instances receive traffic.

It’s worth noting that Classic Load Balancers are a legacy technology in AWS and are being phased out in favor of newer load balancing solutions such as Application Load Balancers and Network Load Balancers, which offer more features and better performance.

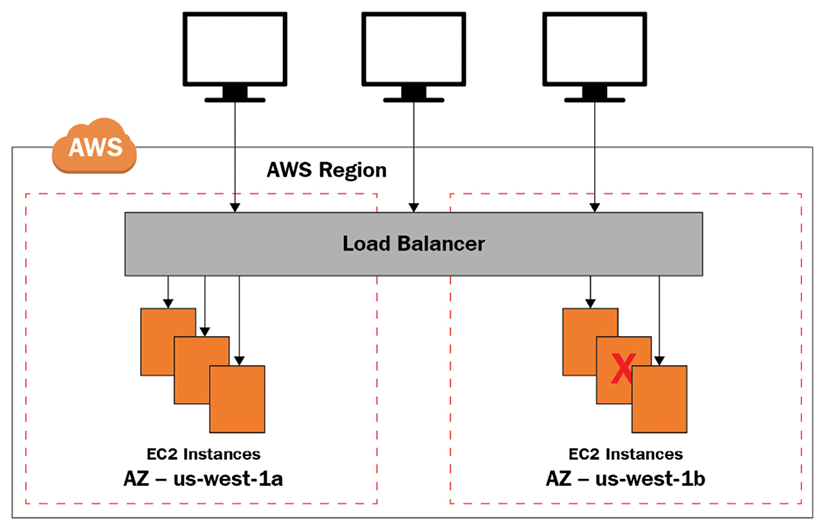
CLBs currently have a requirement where the load balancer needs a fixed relationship between the instance port of the container and the load balancer port. For example, you can map the load balancer using port 8080 to a container instance using port 3131 and to the CLB using port 4040. However, you cannot map port 8080 of the CLB to port 3131 on a container instance and port 4040 on the other container instance. The mapping is static, requiring the cluster to have at least one container instance for each service that uses a CLB.

CLBs can operate at the request level or the connection level. CLBs don’t use host-based routing or path-based routing. CLBs operate at Layer 4 of the OSI model. This means the CLB routes traffic from the client to the EC2 instances based on the IP address and TCP port.

Let’s go through an example:

1. An ELB gets a client request on TCP port 80 (HTTP).
2. The request gets routed based on rules defined in the AWS console for the load balancer to direct traffic to port 4040 to an instance in the provisioned pool.
3. The backend instance processes the instructions from the request.
4. The response is sent back to the ELB.
5. The ELB forwards the payload for the response to the client.

To better understand how traffic is handled with CLBs, look at the following diagram. As you can see, all traffic gets directed to the load balancer first, which in turn directs traffic to instances that the load balancer decides are ready to handle traffic:



*Figure 6.3: CLB architecture*

From the client’s perspective, the request appears to be fulfilled by the ELB. The fact that the backend servers beyond the ELB actually handled the request will be entirely transparent for the client.

Even though it is absolutely possible to set up an ELB with only one EC2 instance supporting the load, it defeats the purpose of having an ELB. Additionally, it is best practice to set up your supporting EC2 instances across AZs in case of an AZ disruption.

Under the default configuration, the load will be distributed evenly across the enabled AZs. When using the default configuration, it is recommended to use a similar number of instances per AZ.

**Application Load Balancers**

The Application Load Balancer is a load balancing solution in AWS that provides advanced routing capabilities for modern web applications. It operates at the application layer (Layer 7) of the OSI model, allowing it to route traffic based on the content of the request, rather than just the source and destination IP addresses and ports. Here are some key features of the Application Load Balancer:

* **Advanced routing**: The Application Load Balancer can route traffic based on multiple criteria, including the URL path, query string parameters, HTTP headers, and source IP address. This allows you to create more sophisticated routing rules for your application.
* **Content-based routing**: The Application Load Balancer can route traffic based on the content of the request, such as the user agent, language, or MIME type. This can be useful for serving different content to different types of clients.
* **Load balancing**: The Application Load Balancer can distribute incoming traffic across multiple targets, such as EC2 instances, IP addresses, and containers. It supports both round-robin and least outstanding requests load balancing algorithms.
* **SSL/TLS termination**: The Application Load Balancer can terminate SSL/TLS connections, reducing the workload on your application servers and providing an additional layer of security.
* **Sticky sessions**: The Application Load Balancer can maintain a session affinity between a client and a target, ensuring that subsequent requests from the same client are always routed to the same target.

The Application Load Balancer is integrated with other AWS services, such as AWS Certificate Manager, AWS CloudFormation, and AWS Elastic Beanstalk, making it easier to deploy and manage your applications. The Application Load Balancer is ideal for modern web applications that use microservices and container-based architectures, as it provides advanced routing and load balancing capabilities that are optimized for these types of applications.

ALBs deliver advanced routing features that can provide host-based routing and path-based routing, and they can support containers and microservices. When you operate in the application layer (an HTTP or HTTPS request on Layer 7), the ELB can monitor and retrieve the application content, not just the IP and port. This facilitates the creation of more involved rules than with a CLB. Multiple services can be set up to share an ALB, taking advantage of path-based routing.

ALBs also easily integrate with **Elastic Container Service** (**ECS**) by enabling a Service Load Balancing configuration. Doing so enables the dynamic mapping of services to ports. This architecture can be configured in the ECS task definition. In this case, several containers point to the same EC2 instance, executing multiple services on multiple ports. The ECS task scheduler can seamlessly add tasks to the ALB.

**Network Load Balancers**

The **Network Load Balancer** (**NLB**) operates at the transport layer (Layer 4) of the OSI model. In this layer, there is no opportunity to analyze the request headers, so the network load balancer blindly forwards requests.

When using network load balancing, the availability of the application cannot be determined. Routing decisions are made exclusively on TCP-layer and network values without knowing anything about the application. An NLB determines *“availability”* by using an **Internet Control Message Protocol** (**ICMP**) ping and seeing if there is a response or completing a three-way TCP handshake.

The NLB is designed to handle high-throughput, low-latency workloads, such as those used by web applications, gaming, and media streaming services. The NLB can distribute traffic across multiple targets, such as EC2 instances, IP addresses, and containers, using a flow-based load balancing algorithm that ensures that each flow is consistently routed to the same target. The NLB is highly scalable and provides high availability, with the ability to automatically recover from failed targets and distribute traffic across healthy targets.

An NLB cannot distinguish whether certain traffic belongs to a certain application or another. The only way this could be possible would be if the applications were to use different ports. For this reason, if one of the applications crashes and the other doesn’t, the NLB will continue sending traffic for both applications. An ALB would be able to make this distinction.

AWS has launched another load balancer called the **Gateway Load Balancer** (**GWLB**), which is a load balancing solution that provides scalable and highly available network-level load balancing for virtual network appliances, such as firewalls, intrusion detection and prevention systems, and other security appliances. The NLB is optimized for high-throughput, low-latency workloads, while the GWLB is optimized for virtual network appliances. The NLB is a Layer 4 load balancer that routes traffic based on IP protocol data, while the GWLB is a Layer 3 load balancer that routes traffic based on the content of the IP packet.

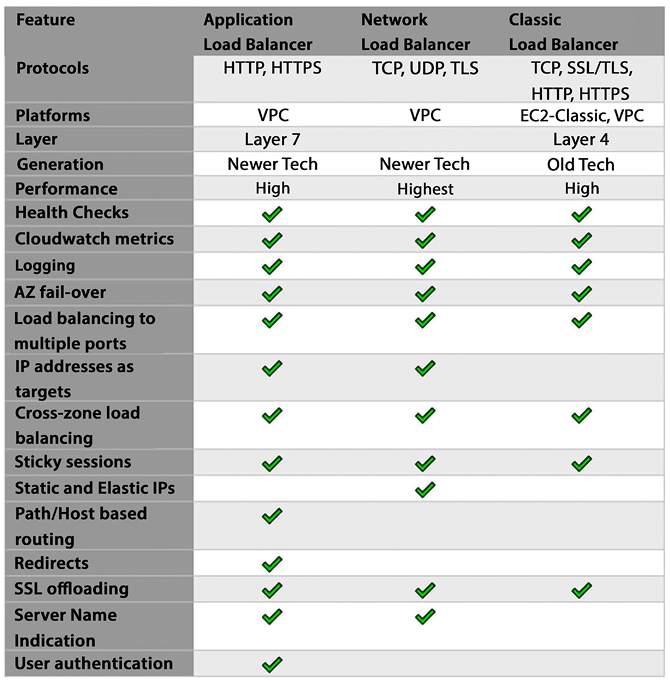
Now that we have learned about the three types of load balancers offered in AWS, let’s understand what makes each type different.

**CLB versus ALB versus NLB comparison**

The ALB is optimized for Layer 7 traffic, such as web applications, microservices, and APIs, while the NLB is optimized for Layer 4 traffic, such as gaming, media streaming, and other TCP/UDP traffic. The CLB is a legacy load balancer that supports both Layer 4 and Layer 7 traffic, but it is less feature-rich than the ALB and NLB. The GWLB is designed for more complex network architectures and provides greater control and visibility. Here’s a brief overview of each load balancer:

* **Application Load Balancer (ALB)**: The ALB operates at the application layer (Layer 7) of the OSI model, allowing it to route traffic based on the content of the HTTP or HTTPS request. It is designed for use cases such as web applications, microservices, and APIs. The ALB supports advanced features like path-based routing, host-based routing, and container-based routing. It also supports SSL/TLS offloading, content-based routing, and sticky sessions.
* **Network Load Balancer (NLB)**: The NLB operates at the transport layer (Layer 4) of the OSI model, allowing it to route traffic based on the IP protocol data, such as TCP and UDP. It is designed for use cases such as high-throughput, low-latency workloads, such as gaming, media streaming, and other TCP/UDP traffic. The NLB supports advanced routing features like source IP affinity, session stickiness, and cross-zone load balancing.
* **Classic Load Balancer (CLB)**: The CLB operates at both the application layer (Layer 7) and the transport layer (Layer 4) of the OSI model, allowing it to route traffic based on the content of the HTTP/HTTPS request as well as the IP protocol data. It is designed for use cases that require basic load balancing, such as simple web applications. The CLB supports basic features like SSL/TLS termination, cross-zone load balancing, and connection draining.
* **Gateway Load Balancer (GWLB)**: The GWLB is a load balancer that is designed for more complex and demanding network architectures. It operates at the network layer (Layer 3) of the OSI model, allowing it to route traffic based on IP addresses and ports. The GWLB is designed to handle traffic for multiple VPCs, allowing you to build more complex architectures with greater control and visibility.

The following diagram illustrates the differences between the three types of ELBs. This will help in our discussion to decide which load balancer is best for your use cases:



*Figure 6.4: ELB comparison*

As a general rule of thumb, it usually is better not to use a CLB on a new project and only use it in legacy applications built when CLBs were the only option.

**CLB and ALB commonalities**

Even though there are differences, CLBs and ALBs still have many features in common. Both support all these features:

* **Security Groups** – Leveraging a **Virtual Private Cloud** (**VPC**) architecture, a security group can be mapped with AWS services, including EC2 instances and ELBs, providing extra security to the overall architecture.
* **SSL termination** – Terminating the SSL connection at the ELB level offloads the processing of SSL traffic from the backend instances to the ELB. This removes load from the backend instances and enables them to focus on handling application traffic. It also simplifies the management of SSL certificates by centralizing the management of the certificates in one place (the ELB).
* **Idle connection timeout** – ALBs and CLBs both support the idle connection timeout period configuration. Connection timeouts enable the termination of connections that exceed a predefined threshold when the server receives no traffic from the client.
* **Connection draining** – Connection draining allows you to gracefully terminate instances and remove them from the ELB while allowing existing transactions to complete. Connections are not terminated until all pending traffic has been processed.

Now that we understand the different characteristics of the three ELB types, let’s understand when it’s best to use each.

**The best choice of ELB by use case**

One item of note when using AWS is that you don’t see version numbers. There is no such thing as S3 version 1.0 and S3 version 2.0. At any given point in time, AWS offers the best version of a given service, and upgrades happen transparently. This puts a tremendous responsibility on AWS, but it makes our job easy. The fact that AWS still offers three load balancers instead of just one tells us that they see value in offering all three. And all three still exist because each could be the best for the job depending on the use case.

And that’s the perfect segue. CLBs are probably the best option for legacy applications where a CLB is already in place. Many deployments were done when CLBs were the only option (at that point, they were called ELBs when that was the only option). Only later were they renamed CLBs to distinguish them from the new ALBs.

If your deployment already uses CLBs, but you are considering migrating to ALBs, these are usually some of the features that might compel you to migrate because they are not supported on a CLB:

* Support for **AWS Web Application Firewall** (**AWS WAF**)
* Support for targets for AWS Lambda
* Support for targets for IP addresses
* The ability to add several TLS/SSL certificates using **Server Name Indication** (**SNI**)

If you have a greenfield deployment and are considering which ELB to use, the best option will most likely be to use an ALB. ALBs integrate with the following:

* **Amazon Elastic Container Service** (**ECS**)
* **Amazon Elastic Kubernetes Service** (**EKS**)
* AWS Fargate
* AWS Lambda

You should have a compelling reason to choose a CLB during a brand-new deployment. A good reason to use NLBs is if performance is one of your utmost priorities and if every millisecond counts. Other reasons to choose an NLB are as follows:

* The ability to register targets using IP addresses, including a target outside the VPC that contains the load balancer.
* Support for containerized applications.
* NLBs support directing requests to multiple applications deployed on the same EC2 instance. Each instance or IP address can be registered with the same target group by assigning different ports.
* Support for traffic with high variability and high volume (millions of requests per second) for inbound TCP requests.
* Support for static or elastic IP addresses. Elastic IPs enable you to keep using the same IP address even if a physical instance goes down.

NLB supports **UDP** (**User Datagram Protocol**) traffic in addition to **TCP** (**Transmission Control Protocol**) traffic. This makes it ideal for use cases such as gaming, media streaming, IoT, and other applications that require low latency and high throughput. The NLB can load balance UDP traffic across a set of backend instances in a target group, using source IP affinity or session stickiness to ensure that each client is directed to the same backend instance for the duration of its session.

Up until this point, we have examined servers in the cloud. Let’s now look at serverless compute options in AWS.

**Learning about serverless computing with AWS Lambda and Fargate**

AWS uses the word “serverless” to describe AWS technologies or applications that have these characteristics: no server management, automatic scaling, high availability built in, and a pay-as-you-go billing model.

AWS has serverless technologies for compute, integration, and databases, though many AWS users associate serverless with AWS’s event-driven compute service, AWS Lambda.

Building serverless applications is one of the primary advantages of moving to the cloud. They reduce the admin overhead of managing infrastructure, thus increasing productivity and further reducing the **total cost of ownership** (**TCO**) in the cloud. A serverless app can be highly performant due to the ease of parallelization and concurrency. Serverless computing is the foundation of serverless apps as it manages to scale automatically, is optimized to reduce latency and cost, and increases throughput. Let’s learn about the serverless compute options available in AWS.

**AWS Lambda**

When it comes to serverless computing, AWS Lambda comes to mind first. Lambda is a serverless compute service that allows users to run code in response to events, without having to provision or manage any underlying infrastructure. Lambda is designed to be scalable, highly available, and cost-effective, making it a popular choice for a wide range of applications.

With Lambda, users can upload their code in the form of a function and specify the events that should trigger that function. When an event occurs, Lambda automatically runs the code in response, without the user having to worry about managing any underlying infrastructure. This allows users to focus on building and running their applications, without having to worry about the details of infrastructure management.

Lambda supports a variety of languages and runtime environments, including Node.js, Java, Python, and .NET, and can be easily integrated with other AWS services, such as Amazon S3, Amazon DynamoDB, and Amazon Kinesis. This makes it an attractive option for a wide range of applications, from simple web services and microservices to complex, distributed systems.

Lambda is very economical in terms of pricing. To start with, AWS provides 1 million free requests a month with the AWS free tier. The pricing component includes the number of requests and duration metered in 1 ms increments based on the function memory setting; for example, 100 ms with 2 GB of RAM costs the same as 200 ms with 1 GB of RAM.

One example use case for AWS Lambda is building a simple web service that allows users to upload images to Amazon S3 and then automatically processes those images using machine learning. Here is how this could work:

1. The user creates a Lambda function that uses a machine learning model to process images and extract information from them.
2. The user configures an Amazon S3 bucket to trigger the Lambda function whenever a new image is uploaded to the bucket.
3. When a user uploads an image to the S3 bucket, S3 automatically triggers the Lambda function, passing the image data as input.
4. The Lambda function runs the machine learning model on the image data and extracts the desired information.
5. The Lambda function returns the results of the image processing to the caller, which could be the user’s application or another AWS service.

In this use case, Lambda is used to automatically process the uploaded images using machine learning, without the user having to manage any underlying infrastructure or worry about scaling or availability. This allows the user to focus on building their application and adding new features, without having to worry about the details of infrastructure management. Serverless architecture means that the cost, size, and risk relating to change reduce, thereby increasing the rate of change.

Lambda acts as a compute service, running the user’s code in response to events, providing a highly scalable, highly available, and cost-effective execution environment. The user can focus on building and running their application, without having to worry about managing any underlying infrastructure.

The purpose of this section is to introduce you to the concept of AWS Lambda. To understand it better, you need to understand how Lambda can help to achieve different architecture patterns, which you will learn about in *Chapter 14*, *Microservice Architectures in AWS*.

Containers are becoming famous for building microservice architectures and deploying complex code. AWS has provided the option to deploy serverless containers using Fargate. Let’s learn about it in more detail.

**AWS Fargate**

Adopting containers requires a steep learning curve for deployment, cluster management, security, and monitoring customers. AWS serverless containers can help you focus time and resources on building applications, not managing infrastructure. Amazon **ECS** provides a simple managed control plane for containers, and AWS Fargate provides serverless container hosting. For Kubernetes, you can use Amazon **EKS** with Fargate for container hosting.

Amazon ECS and AWS Fargate give you a choice to start modernizing applications through fully managed, native container orchestration, standardized and compliant deployment paths, and automated patching and provisioning of servers. ECS Fargate’s serverless model eliminates the operational complexity of managing container hosts and AMIs.

The following are the advantages of Fargate:

* **NoOps**: Yes, you read it right. You have heard about DevOps, DevSecOps, MLOps, and so on, but wouldn’t life be easier if there were NoOps? AWS Fargate removes the complexity of infrastructure management and shifts primary responsibilities, such as OS hardening and patching, onto AWS. You can reduce the resources spent on these tasks and instead focus on adding value to your customers. ECS Fargate automates container orchestration with compute, networking, storage, container runtime config, auto - scaling, and self-healing and provides serverless computing without AMIs to patch, upgrade, and secure your OS.
* **Lower TCO**: In Fargate, each task runs in its dedicated host, and the resources can be tailored to the task’s needs. This dramatically improves utilization, delivering significant cost savings. With fewer container hosts to manage, fewer people are needed to focus on a container infrastructure, which significantly reduces TCO. You only pay for what you provision in Fargate. You are billed for CPU and memory utilization on a per-second billing model at the container task level. You can have further savings with Spot Instances and a Savings Plan.
* **Seamless integrations**: AWS ECS Fargate integrates all the significant aspects of deployment, networking, monitoring, and security. An extensive set of third-party partners, such as Datadog, Aqua Security, Harness, and HashiCorp, provide solutions that work for the monitoring, logging, and runtime security of workloads deployed in ECS Fargate.
* **Security**: In Fargate, each task runs in its dedicated host. This security isolation, by design, eliminates multiple security concerns of sensitive containers co-existing on a host. With complete integration for IAM, security groups, key management and secret management, and encrypted, ephemeral disks, customers can deploy with peace of mind for security.

From a cost-saving perspective, Fargate Spot allows you to leverage similar savings as in your EC2 Spot. With Fargate Spot, AWS runs Fargate tasks on the same EC2 Spot instance and provides a flat discount rate of 70% compared to regular Fargate pricing.

Again, this section aims to introduce you to the concept of a serverless container. To understand it better, you need to understand how containers help to build microservice architecture patterns, which you will learn in *Chapter 13*, *Containers in AWS*.

**High-Performance Computing**

**High-Performance Computing** (**HPC**) is a field of computing that involves using specialized hardware and software to solve complex, compute-intensive problems. HPC systems are typically used for scientific and engineering applications that require a high level of computational power, such as weather forecasting, molecular modeling, and oil and gas exploration. HPC systems require a high level of performance and scalability, but they can help organizations achieve results faster and more efficiently, making them an essential tool for many scientific and engineering applications.

HPC workloads are characterized by a combination of multiple technologies, such as storage, compute, networking, **Artificial Intelligence** (**AI**), machine learning, scheduling and orchestration, and streaming visualization, combined with specialized third-party applications.

HPC workloads are classified into categories that help identify the AWS services and solutions that can best match your needs. These categories include fluid dynamics, weather Modeling, and reservoir simulation; these workloads are typically called scale-up or tightly coupled workloads. The other HPC workloads are financial risk modeling, genomics, seismic processing, and drug discovery, typically called scale-out or loosely coupled workloads. Both of these categories of workloads require a large amount of compute power using EC2 Spot instances; application orchestration using AWS Batch, AWS ParallelCluster, and scale-out computing; and high-performance storage such as FSx for Lustre and S3. Tightly coupled workloads also require high network performance using Elastic Fabric Adapter. Here are some example workloads where you should use HPC:

* Autonomous vehicle simulation and training
* Weather and climate
* Financial services, risk analysis simulation, and actuarial modeling
* **Computer-Aided Engineering** (**CAE**) for manufacturing and design applications
* Drug discovery
* Genomics
* Seismic processing, reservoir simulation/energy
* **Electronic Design Automation** (**EDA**)

AWS offers a number of services and tools that can be used to run HPC workloads on the cloud. These include:

* **EC2 P3 instances**: These instances are designed for HPC workloads and are equipped with powerful NVIDIA Tesla V100 GPUs, making them well suited for tasks such as machine learning and high-performance data analytics.
* **Amazon Elastic Fabric Adapter (EFA)**: EFA is a high-performance network interface that can be used with EC2 instances to improve the performance of HPC workloads that require low-latency communication.
* **AWS ParallelCluster**: AWS ParallelCluster is a fully managed HPC platform that allows users to easily deploy and manage HPC clusters on AWS. It includes a variety of tools and utilities that can help users run HPC workloads on AWS, such as a job scheduler and a resource manager.
* **AWS Batch**: AWS Batch is a service that allows users to run batch computing workloads on AWS. Batch workloads typically involve large numbers of independent tasks that can be run in parallel, making them well suited for HPC applications. AWS Batch can help users manage and scale their batch workloads on AWS.

Overall, AWS provides a variety of services and tools that can be used to run HPC workloads on the cloud. HPC is a vast topic; the idea of this section was to provide a basic understanding of HPC. You can learn more by visiting the AWS HPC page: <https://aws.amazon.com/hpc/>.

**Hybrid compute**

While you want to benefit from the advantages of using the cloud, not all your applications can be migrated to AWS due to latency or the need for local data processing. Latency-sensitive applications such as patient care flow require less than 10 ms responses, and any delays can affect critical processes, so you want your compute to be near your equipment. Similarly, there are instances when you can’t afford downtime due to intermittent networking and want local data processing, for example, manufacturing execution systems, high-frequency trading, or medical diagnostics.

If you can’t move to a Region because of data residency, local processing, or latency requirements, you have to build and maintain the on-premises infrastructure at your facility. In that case, you must maintain an IT infrastructure, which involves a complex procurement and provisioning process from multiple vendors with a months-long lead time. In addition, you will have the overhead to patch and update applications running on-premises and schedule maintenance downtime or arrange for on-site resources, impacting operations.

Developers need to build for different APIs and services to accommodate various other infrastructures and build different tools for automation, deployment, and security controls. That leads to separate code bases and processes for on-premises and the cloud, creating friction and operational risk. This leads to business challenges such as delayed access to new technologies and longer deployment timelines from testing to production, affecting the business’ ability to deliver new features and adapt to changing dynamics.

AWS hybrid cloud services are a set of tools and services that allow you to connect on-premises infrastructure to the cloud, creating a hybrid cloud environment. This allows users to take advantage of the benefits of both on-premises and cloud computing, allowing them to run workloads in the most appropriate environment for their needs.

AWS hybrid cloud services include a range of tools and services that can help users connect their on-premises infrastructure to the cloud, such as AWS Direct Connect and AWS VPN. These services allow users to securely and reliably connect their on-premises environments to the cloud, allowing them to easily move data and workloads between on-premises and cloud environments.

AWS hybrid cloud services also include a variety of tools and services that can help users manage and operate their hybrid cloud environments, such as AWS **Identity and Access Management** (**IAM**) and AWS Systems Manager. These services can help users control access to resources, monitor and manage their environments, and automate common tasks, making it easier to operate and maintain their hybrid cloud environments.

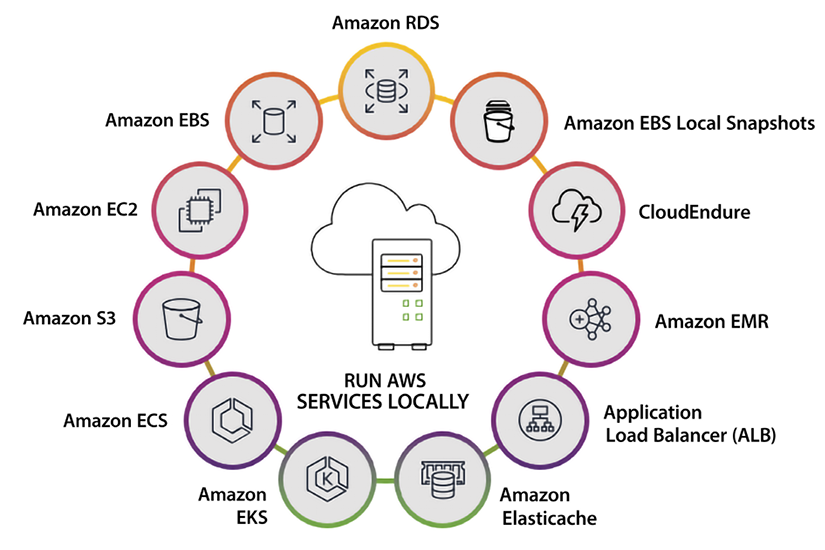
From a compute perspective, let’s look at AWS Outposts, which brings the cloud on-premises.

**AWS Outposts**

Compute and storage racks built with AWS-designed hardware, which are fully managed and scalable, are referred to as AWS Outposts. These can be used to extend AWS infrastructure, services, and tools to on-premises locations. With Outposts, users can run a consistent, native AWS experience on-premises, allowing them to use the same APIs, tools, and services that they use in the cloud.

There are two versions of Outposts: VMware Cloud on AWS Outposts, which allows users to run VMware Cloud on AWS on-premises, and AWS Outposts, which allows users to run a native AWS experience on-premises. Outposts is designed to be easily deployed and managed, and can be used to run a variety of workloads, including compute-intensive HPC workloads, latency-sensitive workloads, and applications that require access to on-premises data or resources.

Outposts’ servers are 1–2 rack units high (1.75”–3.5”) and fit into standard EIA-310 19” width racks. You can launch AWS Nitro-based Amazon EC2 instances with instance storage, among other services available to Outposts locally. The following services are available in Outposts.



*Figure 6.5: AWS Outposts locally available services*

As shown in the preceding diagram, you can run virtual machines with EC2 instances in your on-premises facility using Outposts. Each instance will have instance storage that can be used for AMI launches and data volumes. You can deploy containerized workloads using Amazon ECS or EKS. You can network with Amazon VPC to introduce a logical separation between workloads and secure communication within a workload that spans the Region and your site. Outposts is a good option for IoT workloads to deploy IoT Greengrass for IoT messaging and SageMaker Neo for inference at the edge.

Outposts can help operate smaller sites such as retail stores or enterprise branch offices that run point-of-sale systems, security monitoring, smart displays, and developing next-gen facial or voice recognition to customize the customer experience. It can be helpful for healthcare providers to run the latest tech to assess patient images and process medical data quickly but benefit from cloud tools and long-term storage.

For factories, warehouses, and distribution centers, Outposts can power automation, integrate IoT data, monitor systems, and provide feedback to operators.

AWS Outposts can help users extend the benefits of AWS to on-premises locations, allowing them to run a consistent, native AWS experience on-premises and easily move workloads between on-premises and cloud environments.

In 2017, VMware launched a public cloud offering on AWS called VMware Cloud. VMware is very popular when it comes to running virtual machines. VMware Cloud simplifies migrating and extending your VMware workloads into the AWS cloud. Recently VMware Cloud became available on Outposts as well.

**VMware Cloud on AWS**

**VMware Cloud** (**VMC**) on AWS is a hybrid cloud solution built by engineers from AWS and VMware that provides a fast and efficient path to the cloud if you run VMware vSphere workloads in on-premises data centers. VMC accelerates cloud transformation with operational consistency and flexibility while scaling global business demand. VMware Cloud on AWS is a fully managed service that provides users with a consistent, native VMware experience in the cloud, allowing them to easily migrate and run their VMware-based workloads on AWS.

VMware Cloud on AWS is available in a variety of sizes and configurations, with varying levels of performance and capabilities. It can be used for a wide range of workloads, including traditional enterprise applications, cloud-native applications, and workloads that require access to on-premises data or resources.

Here are some VMC use cases:

* **Migrating VMware-based workloads to the cloud**: VMware Cloud on AWS allows users to easily migrate their VMware-based workloads to the cloud, without having to redesign their applications or learn new technologies. This can help users take advantage of the benefits of cloud computing, such as elastic scalability, high availability, and low cost, while still using the tools and technologies they are familiar with from VMware.
* **Running hybrid cloud environments**: VMware Cloud on AWS can be used to run hybrid cloud environments, allowing users to easily move workloads between on-premises and cloud environments and take advantage of the benefits of both. This can help users achieve greater flexibility, scalability, and efficiency in their environments.
* **Running disaster recovery and backup**: VMware Cloud on AWS can be used to run disaster recovery and backup environments, allowing users to quickly and easily recover from disasters and outages. This can help users ensure the availability and integrity of their applications and data, even in the face of unexpected events.
* **Running test and development environments**: VMware Cloud on AWS can be used to run test and development environments, allowing users to quickly and easily spin up new environments for testing and development purposes. This can help users speed up their development and testing processes, allowing them to release new features and applications faster.

The features of VMC allow customers to dynamically scale their environment by leveraging disaster recovery systems, highly available compute clusters, vSAN storage clusters, and NSX network virtualization.

**Summary**

In this chapter, you learned about compute services available in AWS, which will help you choose the right to compute per your workload requirement. You learned about why terms changed from servers to compute recently due to the broad set of options provided by the cloud.

The most popular compute service is EC2, which is the foundation for the rest of the services provided by AWS. For example, a service such as Amazon SageMaker or Amazon DynamoDB under the hood relies on core services such as EC2. You learned about various EC2 families, pricing models, and advantage.

There are so many EC2 options available, which may be confusing when it comes to optimizing your cost model. You learned about AWS compute optimization, which can help you choose the right compute option and optimize cost. You also learned about AMI, which helps you choose the operating system for your workload and spin up EC2 per your needs. Further, you learned about EC2 best practices.

For distributed computing, a load balancer is often needed to distribute the workload across multiple servers and prevent any single server from becoming overloaded. You learned about AWS-provided load balancers, which can be used for distributed computing, including CLBs, ALBs, NLBs, and GWLBs.

Compute is not limited to servers, and serverless compute is becoming more popular because it helps you to focus on your business logic and avoid the admin overhead of scaling and patching servers. You learned about the two most popular serverless compute options, AWS Lambda and AWS Fargate. You learned about HPC in AWS, which can help you to have your own mini-supercomputer to solve complex algorithmic problems.

Not every workload can be hosted on the cloud. You have learned about hybrid compute and how AWS brings its infrastructure on-premises and provides the same experience as the cloud. In the next chapter, you will get knee-deep into another set of fundamental services in AWS that are the workhorse of many successful start-ups and multinationals. That is the beautiful world of databases.