3. On the left-hand side, in the **Category** pane, select **Connection** | **SSH** | **Auth** and click **browse** to provide a private key:

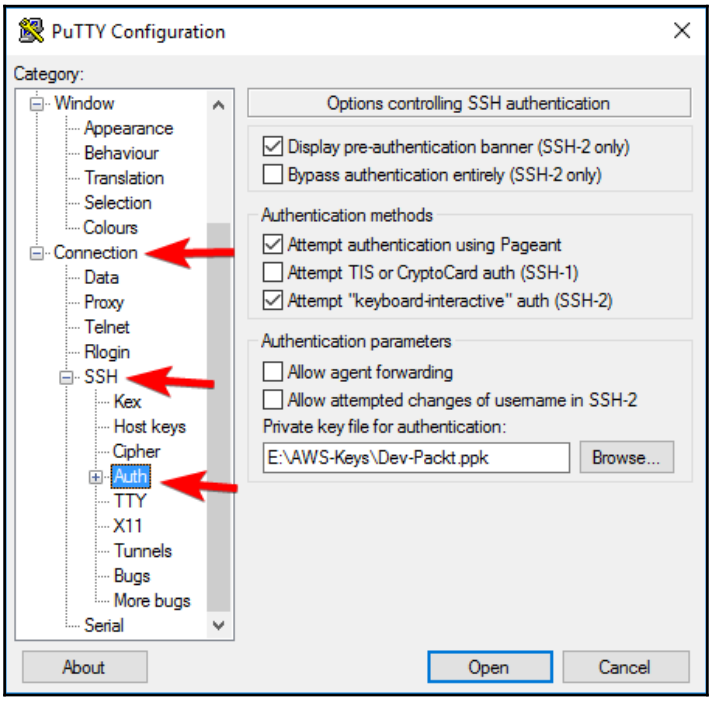


Figure 5.18: Provide private key

4. Once you click **Open**, a security dialog box may appear confirming that you trust the host you are about to connect with. Choose **Yes** and SSH connection takes place.

Troubleshooting SSH connection issues

While establishing an SSH connection with an EC2 instance, if all the required details are properly provided and in spite of that, it fails to establish an SSH connection, check out the following points:

- Ensure that you are giving the correct IP address of the instance.
- Verify the username you have given along with the IP address.

- Make sure an EC2 instance is up and running.
- Ensure that the security group has SSH port `22` open and is accessible.
- Check the OS level firewall and ensure its not blocking the connection.
- If you are behind a network proxy, ensure that your network proxy is not blocking it.
- Ensure that you are using a right `.ppk` file.
- After verifying all the preceding steps, if you are still not able to log in, you can try stopping and restarting the instance.
- You can also diagnose the issue by stopping the instance, detaching its root drive and attaching and mounting it to another healthy EC2 instance as a secondary drive. Once the drive is attached to another EC2 instance, you can diagnose configuration issues.

EC2 instance metadata and user data

Metadata is data about an EC2 instance. EC2 instance details such as AMI ID, hostname, instance ID, instance Type, private IP address, public IP address, and so on are metadata of the instance. EC2 instance metadata can be retrieved by querying `169.254.269.254` on the same machine. From a Linux system, you can use the following command to retrieve the metadata from the EC2 instance:

```
$ curl http://169.254.169.254/
```

Issuing the `curl` command gives the following output, categorizing the metadata based on the date it is introduced by AWS:

```
1.0
2007-01-19
2007-03-01
2007-08-29
2007-10-10
2007-12-15
2008-02-01
2008-09-01
2009-04-04
2011-01-01
2011-05-01
2012-01-12
2014-02-25
2014-11-05
2015-10-20
2016-04-19
2016-06-30
```

```
2016-09-02
latest
```

Furthermore, `latest` metadata is divided into three categories, as shown in the following output:

```
$ curl http://169.254.169.254/latest
dynamic
meta-data
user-data
```

An EC2 instance's individual metadata properties can be retrieved by adding the property at the end of the following command:

```
$ curl http://169.254.169.254/latest/meta-data/
```

For example, you can retrieve `ami-id` by querying the following URL:

```
$ curl http://169.254.169.254/latest/meta-data/ami-id
```

Similarly, you can use the following list of properties with the `curl` command to retrieve its value:

```
ami-id
ami-launch-index
ami-manifest-path
block-device-mapping/
hostname
instance-action
instance-id
instance-type
local-hostname
local-ipv4
mac
metrics/
network/
placement/
profile
public-hostname
public-ipv4
public-keys/
reservation-id
security-groups
```

On an Amazon Linux EC2 instance, the `ec2-metadata` command also gives metadata.

Placement group

A Placement group is a logical grouping of EC2 instances within a single AZ. Placement group provides a possible lowest network latency across all the EC2 instances that are part of the same placement group. All EC2 instances do not support high network throughput (that is, placement group). Before launching an instance in a placement group, you need to ensure that the instance type supports a placement group. It is best practice to create all the EC2 instances required in a placement group, and ensure they are created in a single launch request and have the same instance type. In case multiple instance types are mixed in a placement group then the lowest bandwidth among the EC2 instances is considered as the highest network throughput of the placement group. It is recommended you choose an instance type that supports a 10 GBps or 20 GBps network throughput. There is no additional charge for creating an instance group.

Some important points about a placement group:

- A placement group can span across peered VPCs.
- When network traffic is flowing to and from outside the placement group, network throughput is limited to 5 GBps.
- An existing EC2 instance in the same AZ cannot be moved inside a placement group. It requires creating an AMI of the existing instance and then launching a new EC2 instance inside the placement group.
- Even in the same account, placement groups cannot be merged.
- When you stop and start an instance inside a placement group, it remains in the same placement group.
- If you get a capacity error while launching an instance inside a placement group, you can stop and start all instances in the placement group. Stopping and starting instances automatically migrates the instances to another hardware that has capacity to run them.

Introducing EBS

EBS is an AWS block storage service, that provides block level, persistent storage volumes for EC2 instances. EBS volumes are highly available and reliable storage solution. An EBS volume can be attached only to the EC2 instances running in the same AZ. It provides persistent storage and it is independent from the EC2 instance. That means the data on the EBS volume remains intact even if the instance is restarted. AWS charges for the allocated EBS volume sizes, even if the volume is not attached to any instance. Also, charges are based on the allocated volume size and not based on how much data is stored on the volume. EBS volumes can be formatted into the desired block size and filesystem. It is very suitable for the read and write such as database application or throughput of intensive workloads such as big data. Once EBS volumes are attached to EC2 instances, they are used like a normal physical drive. Multiple EBS volumes can be attached to a single EC2 instance; however, one EBS volume can be attached only to a single EC2 instance.

AWS replicates EBS data at least three times within a single AZ. Unlike S3 data, it does not get replicated in multiple AZs within the same region. It is also important to understand that EBS volumes are not directly attached to the hosts (hypervisor), but they are network-attached block storage.

Types of EBS

Currently, AWS provides the following types of EBS volumes. These EBS types have different performance and prices per GB:

- **Solid State Drive** (**SSD**):
    - General Purpose SSD (gp2)
    - Provisioned IOPS SSD (io1)
- **Hard Disk Drive** (**HDD**):
    - Throughput optimized HDD (st1)
    - Cold HDD (sc1)
- Previous generation volume:
    - Magnetic (Standard)

General Purpose SSD (gp2)

The gp2 volumes are one of the EBS volume types that provides persistent storage. gp2 volume types are ideal for a number of workloads. gp2 volumes are very efficient and provide single-digit millisecond latencies. A gp2 volume is capable of bursting up to 3,000 IOPS for a significant amount of time. You can provision a minimum of 1 GiB size of gp2 volume and a maximum of up to 16 TiB of a gp2 volume. gp2 volume provides 3 IOPS per GiB of volume size. However, if a volume size is 33.33 GiB or less, it provides a minimum of 100 IOPS. As you increase the volume size, the IOPS it provides, also increases. However, a gp2 volume can provide a maximum of 10,000 IOPS. If you use multiple gp2 volumes in an instance, AWS imposes a limit of a maximum of 65000 IOPS per instance.

Where to use gp2 volumes:

- It is recommended for almost all workload types
- Can be used as a root volume for an operating system
- Can be attached to a virtual desktop
- In interactive apps requiring low-latency storage
- Development workloads
- Testing environments

Provisioned IOPS SSD (io1)

Provisioned IOPS SSD (io1) volumes are solid state drive volumes that are intended to address the needs of I/O intensive application workloads. io1 volumes are specifically used for database workloads that require high performance storage and consistent throughput. Unlike gp2 volumes, io1 volume provides a consistent performance. You can specify a consistent IOPS rate while creating the volume. io1 volumes can provide maximum performance out of all other volume types. An io1 volume size can range between 4 GiB to 16 TiB. An io1 volume can have a minimum of 100 IOPS and a maximum of up to 20,000 IOPS. If you use multiple io1 volumes in an instance, AWS imposes a limit of a maximum of 65000 IOPS per instance.

Where to use io1 volumes:

- It can be used in mission critical applications
- Business critical application requiring consistent performance
- It can be used in large databases workloads such as SQL Server, Oracle, and so on

Throughput Optimized HDD (st1)

Throughput Optimized HDD (st1) volumes are designed to provide a financially viable magnetic storage option. st1 volumes are architected to measure the performance in terms of throughput and not on IOPS. st1 volume type is recommended for a large and linear workload such as data warehouse, log processing, Amazon **Elastic MapReduce** (**EMR**), and ETL workloads. It cannot be used as a bootable volume. An st1 volume size can range between 500 GiB to 16 TiB. An st1 volume can have a maximum of 500 IOPS per volumes. If you use multiple st1 volumes in an instance, AWS imposes a limit of a maximum of 65000 IOPS per instance.

Where to use st1 volumes:

- Applications requiring consistent and fast throughput at a low cost
- Big data
- Data warehouse
- Log processing

Cold HDD (sc1)

Cold HDD (sc1) volumes are designed to provide a cost effective magnetic storage option. sc1 volumes are designed to measure the performance in terms of throughput and not on IOPS. sc1 volume type provides a lower throughput limit compared to st1. It is recommended for large, linear cold-data workloads. It's a good low-cost alternative to st1 if you require infrequent access to your data. sc1 volumes cannot be used as bootable root volume. An sc1 volume size can range between 500 GiB to 16 TiB. An sc1 volume can have a maximum of 250 IOPS per volumes. If you use multiple sc1 volumes in an instance, AWS imposes a limit of a maximum of 65000 IOPS per instance.

Where to use sc1 volumes:

- It can be used in throughput-oriented storage
- Use it for large volumes of data when you don't need to access it frequently
- In application needs where there is a need to lower the storage cost

Encrypted EBS

Amazon provides a simple EBS encryption solution that does not require building, maintaining, and securing your own key management infrastructure.

After creating an encrypted EBS volume, when you attach it to a supported instance, it encrypts the following types of data:

- All data at rest, stored inside the volume
- All data that is moving between the volume and the EC2 instance
- All snapshots back up taken from the volume
- AWS encrypts the data on the servers that host EC2 instances and provide encryption of data-in-transit from EC2 instances and on to EBS storage

Amazon EBS encrypts the data using AWS **Key Management Service** (**KMS**) with a customer master key whenever you create an encrypted volume and subsequently any snapshots from them.

When an encryption-enabled EBS volume is attached to the supported EC2 instance type, encryption takes place at EC2 for data-in-transit from EC2 to EBS storage. All future snapshot and disk I/Os are encrypted. An encryption master key from Amazon KMS is used to perform encryption and decryption. Two types of encryption master keys can be used, an Amazon created and custom key or customer provided key. When creating an encrypted EBS volume for the first time in any AWS region, AWS automatically creates a master key in that region. By default, this key can only be used to encrypt the EBS volume. In order to use a custom key to encrypt EBS volumes, you need to create a **Customer Master Key** (**CMK**) in AWS KMS. Creating a CMK gives more control over disabling, defining access control, creating and rotating encryption keys. At present, the root volume attached to the AWS EC2 instance cannot be encrypted. Other than the root, all attached EBS volumes can be encrypted. AWS uses **Advanced Encryption Standard** (**AES-256**) algorithms for encryption.

Monitoring EBS volumes with CloudWatch

Once desired size and type of EBS volumes are created, it is recommended you monitor the performance of the volumes. Monitoring helps in identifying any performance bottleneck, if any, due to any issue. We can use performance logs in the CloudWatch to determine if any volume type needs an upgrade in terms of size, IOPS, or throughput. When an EBS volume is created, AWS automatically creates several CloudWatch metrics for each EBS volumes. Monitoring data is categorized into basic and detailed monitoring. Basic monitoring details are free and include metrics such as read bandwidth (KiB/s), write bandwidth (KiB/s), read throughput (Ops/s), write throughput (Ops/s), and many others.

Only Provisioned IOPS SSD (io1) sends monitoring data to CloudWatch at one-minute intervals. The rest of the EBS volume types such as General Purpose SSD (gp2), Throughput Optimized HDD (st1), Cold HDD (sc1), and Magnetic (standard) sends data to the CloudWatch metrics at five minute intervals.

CloudWatch does not monitor at what rate the disk is being filled or at what percent the disk is utilized or empty. For such requirement, you need to create a custom CloudWatch matrix.

More details about monitoring EC2 instances and EBS volumes are given in `Chapter 7` - *Monitoring with CloudWatch*.

Snapshots

EBS snapshot is an AWS service that provides a mechanism to back up EBS volumes. AWS provides a way to back up your EBS data on S3 by taking a point-in-time snapshot. Snapshots are incremental in nature. That means, it only saves data blocks that are changed after the last snapshot backup taken from the volume. This incremental approach of backing up data saves the time and cost of storage. If there are multiple snapshots for an EBS volume and you delete one of the snapshots, AWS deletes only the data relevant to that snapshot. Other snapshots created out of the same volume, refer to the base data and the incremental change relevant to them.

However, AWS stores the snapshot on S3; snapshots are not directly visible to users on S3. AWS stores the snapshot on a separate area in S3, which is inaccessible to end users. Users can see their snapshots on a snapshot dashboard, given within the EC2 dashboard.

Whether EBS volumes are attached to any instance or not, an EBS snapshot can be taken. Snapshot not only provides an option to perform point-in-time backup of EBS volumes, but it also acts as a baseline for new EBS volumes. Snapshot can be used to migrate existing EBS volumes along with its data from one AZ, region, or AWS account to another. Snapshot of an encrypted volume is also encrypted.

The internal mechanism for a snapshot is to write the copy of data from the EBS volume to the S3 where it is stored redundantly across multiple AZs in the same region. When we access S3 using a web console, CLI, or API, we can't see the snapshots in S3.

It is critical for an organization to draft a backup and retention policy that determines how frequently snapshots are taken for EBS volumes. It is advisable that the backup and retention policy also defines a retention period for each snapshot depending on organizational needs. Housekeeping activities should be automated using scripts or tools to take the snapshots as well as delete unwanted snapshots from the account to control unnecessary cost.

Snapshots are the incremental backup. For any EBS volume, when taking the first snapshot all the written blocks are copied to S3 and finally **Table of Contents** (**TOC**) for the snapshot is written to the S3. TOC points to these blocks. When taking a consequent snapshot for the same EBS volume, it only copies modified blocks from the EBS volume to S3 and creates a relevant table of contents. The table of contents points to the recent blocks copied from EBS to S3 as well as blocks copied during previous snapshots, which are not changed. The following snapshot creation and deletion *Figure 5.19* helps to understand the same. In the same figure, we can see the TOC 2 and TOC 3 is pointing to some of the new and some of the old blocks:

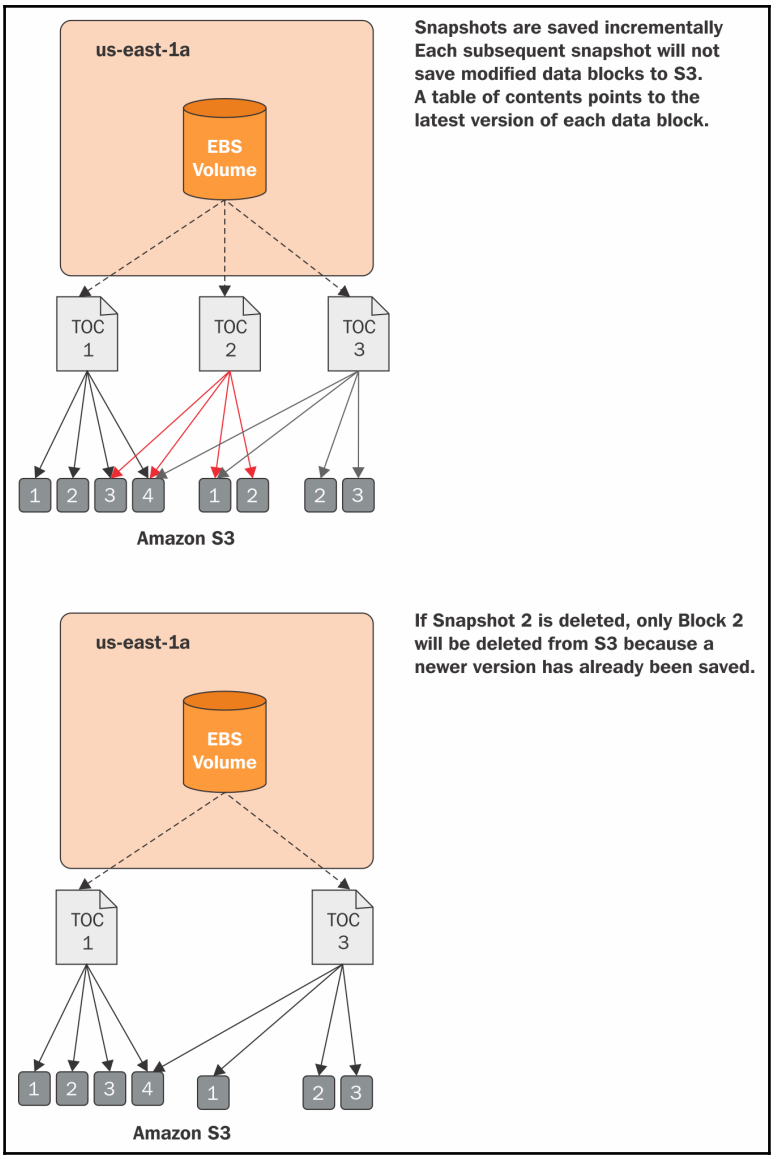


Figure 5.19: Snapshot creation and deletion

Reference URL: http://www.rightscale.com/blog/cloud-industry-insights/amazons-elastic-block-store-explained

It also helps us to understand that when any intermediate snapshot is deleted, only those blocks are deleted that are not referred to by any other snapshot TOCs.

EBS optimized EC2 instances

In a normal EC2 instance, usual network traffic and EBS traffic flows through the same network interface. If network traffic on application processes increases, it adversely affects the EBS performance. Similarly, activities on an EBS volume read and write can adversely affect other network activities on the instance. The following *Figure 5.20* indicates how network traffic from an EC2 instance and EBS volume flows through the same network link:

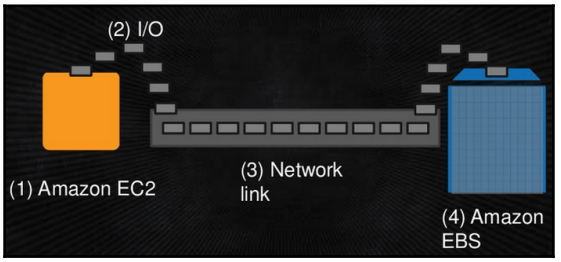


Figure 5.20: Network traffic on EC2 instance and EBS volume

https://cloudnative.io/blog/2015/01/ebs-best-practices-and-performance-tuning/

To handle such performance issues, AWS provides EBS-optimized instance types. EBS optimized instance provides dedicated throughput between EBS volumes and EC2 instance. Such instance types are essential for the application where predictable and consistent disk performance is required. While using Provisioned IOPS SSD volumes, it is recommended you use EBS-optimized instance. It ensures best performance out of Provisioned IOPS SSD volume.

EC2 best practices

The list summarizing EC2 best practices are as follows:

- Ensure that unused EC2 instances are stopped and if not required, terminate them. It reduces unnecessary cost in the monthly AWS billing.
- Closely observe snapshots and AMIs, timely perform housekeeping and discard all the AMIs and snapshots, which are not required. It is recommended you automate and monitor this process.

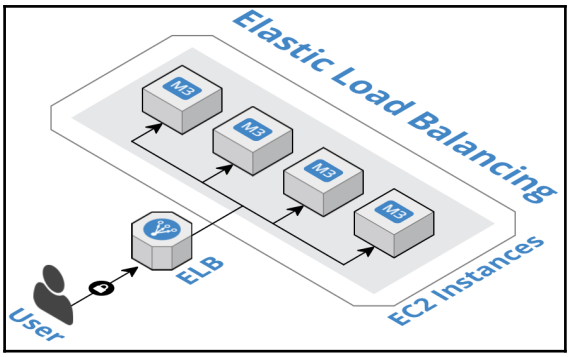
- Ensure that an instance type is set as per the requirement of the application hosted on the instance. It is recommended you optimize instance type as per application performance.
- To match a seasonal spike in compute requirement, plan for Auto Scaling and load balancing.
- Divide your application load in multiple instances rather than going for one big instance, where possible. Dividing the application workload over multiple instances, in different AZ, can avoid single point of failure.
- Ensure to use On-Demand, Spot, and Reserved Instances in the environment based on the need. Balancing the instance types can significantly reduce cost as reserved and spot instances provide a huge cost benefit.
- Always keep your key pairs safely. Once a key pair is lost, it cannot be recovered from AWS.
- Do not embed access key and secret key into the EC2 instance. Where possible, use EC2 roles for accessing AWS resources from an EC2 instance.
- Attach appropriate IAM roles and policies, at the time of creating an EC2 instance to grant access to other AWS services.
- Periodically update security groups and maintain least permissive rules.
- Periodically update and patch the OS to overcome possible security and other vulnerabilities.
- According to the data storage requirement such as persistent or temporary, select EBS or instance-type backed AMI for provisioning an instance.
- It is recommended you use separate volumes for the operating system and data storage.
- Always tag AWS resources with appropriate and relevant tags. It provides a convenient way to identify the right EC2 instance at the time of performing maintenance.
- Create golden AMIs and update them periodically.
- Periodically perform maintenance to delete obsolete and unwanted AMIs as well as snapshots to minimize monthly AWS billing.
- Monitor EC2 and EBS volumes to identify any performance bottleneck in the environment.

6

Handling Application Traffic with Elastic Load Balancing

Introduction to Elastic Load balancer

Elastic Load Balancer (**ELB**) is an AWS service that automatically distributes incoming network or application traffic to a number of EC2 instances. It monitors the health of each of the EC2 instances associated with it and forwards traffic only to healthy instances. ELB provides a single point of contact for the EC2 instances behind ELB. Each of the EC2 instances is marked with a status, either *InService* if it is healthy or *OutOfService* if it is unhealthy. Traffic is routed only to *InService* instances. An ELB provides a single point of contact for the application traffic which is hosted on multiple EC2 instances. By routing traffic only to healthy instances, ELB provides fault tolerance to the application and ensures high availability of the application:



Benefits of using ELB

ELB provides high availability, fault tolerance, elasticity, and security to your environment. We will now look at benefits of ELB in brief:

- **High availability**: An application hosted behind an ELB on a fleet of EC2 instances spread across multiple AZ provides high availability. Consider a scenario where an application is hosted in an EC2 instance without using an ELB. If traffic to the application spikes, EC2 may not be able to handle the traffic and application performance as well as availability is affected. Consider the similar traffic scenario where an application is hosted in multiple E2 instances across AZs behind an ELB. ELB distributes the traffic in a round-robin fashion to all the instances, ensuring that any one EC2 instance is not flooded with traffic.
- **Fault tolerance**: ELB monitors the health of each of the instances associated with it. If an instance is unreachable, it marks the instance as *OutOfService*. Similarly, if an instance is reachable and healthy, it marks the instance as *InService*. The traffic is directed only to the *InService* instances. This way, even if an instance is down, application traffic is not affected and it provides fault tolerance to the application.
- **Elasticity**: In spite of high availability and fault tolerance with a limited number of instances in an ELB, traffic can spike beyond the capacity of the instances in ELB. Other way round, if traffic is very low, the number of instances in an ELB may be underutilized. Over utilization of instances may lead to application performance issues and under utilization results in unnecessary cost. To handle both these scenarios, an ELB can be associated with an Auto Scaling group. It provides elasticity to ELB by automatically increasing the number of instances in an ELB when the traffic is high and automatically reducing the number instances when the traffic is low.
- **Security**: ELB, when used with VPC, provides robust networking and security features. It provides the ability to create an internal-facing ELB, using private IP addresses to route traffic within the VPN. You can also create an internet-facing load balancer to route traffic from the internet to instances in a private subnet. In either case, instances are not directly accessible to the traffic. ELB acts as a frontend to the instances associated with it; and provides a security layer to protect them.

Types of ELB

There are two types of ELB, Classic Load Balancer and Application Load Balancer.

Classic Load Balancer

Classic Load Balancer is one of the initial offerings of AWS. It handles the traffic depending upon application or network level information. It is used for load balancing simple traffic across multiple EC2 instances where the need is to have a highly available, automatic scaling, and secured environment. It is not suitable for the applications that require advanced routing capabilities for handling the application traffic.

Application Load Balancer

Application Load Balancer has been introduced lately in AWS offerings. Unlike Classic Load Balancer, it provides advance application-level routing options. It provides the ability to route the traffic based on application content spread across multiple services, micro services, or container services with multi-tiered application architecture.

Features of ELB

There are several features of an ELB. Each type of load balancer, be it Classic Load Balancer or Application Load Balancer, has its own set of features. The following table illustrates these features and compares the two types of ELBs. This comparison can help in choosing the right load balancer type depending on the requirement:

Features	**Classic Load Balancer**	**Application Load Balancer**
Protocols	HTTP, HTTPS, TCP, SSL	HTTP, HTTPS
Platforms	EC2-Classic, EC2-VPC	EC2-VPC
Sticky sessions (cookies)	✓	Load balancer generated
Idle connection timeout	✓	✓
Connection draining	✓	✓
Cross-zone load balancing	✓	Always enabled

Health checks	✓	Improved
CloudWatch metrics	✓	Improved
Access logs	✓	Improved
Host-based routing		✓
Path-based routing		✓
Route to multiple ports on a single instance		✓
HTTP/2 support		✓
WebSockets support		✓
ELB deletion protection		✓

ELB feature comparison

Let's understand each of the features supported by respective load balancer type.

Protocols: As depicted in the table, Classic Load Balancer supports transport layers as well as application layers. Transport layers consist of TCP and SSL protocol, whereas application layers use HTTP and HTTPS protocols. Application Load Balancer takes routing decisions at application layer and supports only HTTP and HTTPS protocols.

Platforms: Classic Load Balancer, being one of the initial offerings, supports EC2-Classic as well as EC2-VPC, whereas Application Load Balancer is introduced later in AWS offerings for advance application-level options, and supports EC2 instances hosted in a VPC only.

Sticky sessions: Sticky session is a way to consistently route traffic requests from a particular user to the same target instance based on HTTP session, IP address, or a cookie. Classic load balancer supports application cookies or if an application does not have a cookie, you can create a session cookie by specifying stickiness duration in ELB configuration. It creates a cookie named AWSELB for mapping the session to an instance.

Application Load Balancer supports sticky sessions using load balancer generated cookies. It's a mechanism to route traffic requests originated from a user to the same target group every time during a session. When a user sends a request to an application for the first time, an ELB routes the request to one of the instances and generates a cookie. This cookie is included in the response back to the user. When the user sends the subsequent requests, the request contains the same cookie. Based on the cookie, this request is sent to the same target instance until the session duration lasts. The name of the cookie generated in Application Load Balancer is AWSALB.

Idle connection timeout: Every time a user makes a request to an ELB, it maintains two connections. One connection is created with the client and another connection is created with the target EC2 instance. For each of these connections, ELB maintains an idle timeout period. If there is no activity during the specified idle time between the client and the target EC2 instance, ELB closes this connection. In short, if there is no traffic flowing from client to EC2 or vice versa, ELB closes the connection. By default, idle time duration is 60 seconds in an ELB.

Connection draining: An ELB stops sending requests to instances that are either unhealthy or are deregistering from the ELB. This can lead to abrupt closure of an ongoing session initiated by a user. Such abrupt closed sessions give unpleasant experience to an application user. To take care of such user experience issues, AWS supports connection draining in an ELB. When connection draining is enabled and an instance becomes unhealthy or deregistering, an ELB stops sending new requests to such instances; however, it completes any in-flight request made to such instances.

The timeout value for connection draining can be specified between 1 second an 3,600 seconds. The default timeout value for connection draining is 300 seconds. The load balancer forces a connection to close and deregisters it if the time limit is reached.

Cross-zone load balancing: When a Classic Load Balancing is created with instances spread across multiple AZs, it distributes the traffic evenly between the associated AZs. If you have an ELB with 10 instances in US-East-1a and two instances in US-East-1b, the traffic is distributed evenly between two AZs. This means—two instances in US-East-1b serve the same amount of traffic as 10 instances in US-East-1a. This is the default behavior of Classic Load Balancing. If you enable cross-zone load balancing, an ELB distributes traffic evenly between all the EC2 instances across multiple AZs.

Cross-zone load balancing is configurable in Classic Load Balancer; however, it is always enabled in Application Load Balancer.

Health checks: To determine whether an instance is capable of handling traffic or not, an ELB periodically sends pings, tries to establish a connection, or sends HTTP/HTTPS requests. These requests are used to determine the health status of an instance and are called health checks. All the healthy instances in an ELB that serve the traffic have a status as *InService* instances. All the unhealthy instances, which cannot serve the traffic, are called *OutOfService* instances. The ELB routes requests only to the healthy instances. It stops routing traffic requests to *OutOfService* instances. It resumes sending traffic to the instances as soon as the instance status becomes healthy and *InService*.

CloudWatch metrics: AWS sends data points to CloudWatch for load balancers and all the instances associated with it. CloudWatch aggregates those data points and creates statistics in an ordered set of time-series data. This time-series data is called CloudWatch metrics for ELB. With CloudWatch metrics, you can verify the number of healthy EC2 instances in a load balancer during a specific time period. It helps to verify whether the system is consistently performing as expected or not. With CloudWatch, you can create an event trigger in case the metric is outside of the acceptable range. Such event triggers can send a mail to stakeholders or take any specific action based on its association with either Lambda function or Auto Scaling group.

Proxy protocol: When an end user request hits ELB, ELB changes source IP and other request header and forwards it to one of the EC2 instance where the application is hosted. This is the default behavior of ELB, which bars an application from obtaining original client connection information. In some enterprise applications, it is required to have original source connection details to perform traffic analysis. It helps the application to understand more about end user's behavior with such information. However, ELB does not provide original connection information to application by default, it supports proxy protocol, which can be used to obtain connection information from ELB. Proxy protocol is nothing but an **Internet Protocol** (**IP**), which carries client connection information. You can enable or disable Proxy protocol on ELB with the help of AWS CLI.

For more details on enabling or disabling the proxy protocol, you can refer to the URL: `https://www.linkedin.com/pulse/enable-disable-proxy-protocol-support-aws-elb-using-cli-bhavin-parmar/`.

Access logs: ELB generates access logs for each requests sent to it. With each passing request, it captures information such as time of request, source IP address, latencies, request path, and the server response. The access log can be used to analyze the traffic and for troubleshooting any issue. Enabling a access log is optional and it is disabled by default. Once the access log is enabled for an ELB, it captures the log and stores it in an Amazon S3 bucket. The S3 bucket name can be specified while enabling the access log on an ELB.

AWS does not charge any additional amount for access logs, however, you are charged for the storage you use on S3 bucket for access logs.

Host-based routing: Host-based routing refers to a mechanism of routing traffic to a specific target group based on the hostname specified in the host header of the request. For example, requests to `www.example.com` can be sent to target group A, requests to `mobile.example.com` can be sent to target group B, and requests to `api.example.com` can be sent to target group C.

Host-based routing is only supported in Application Load Balancer.

Path-based routing: Application Load Balancer provides a mechanism to route traffic to a specific target group based on the URL path specified in the host header of the request. For example, requests to `www.example.com/production` can be sent to target group A, requests to `www.example.com/sandbox` can be sent to target group B, and requests to `www.example.com/admin` can be sent to target group C.

Path-based routing is only supported in Application Load Balancer.

Route to multiple ports on a single instance: Application Load Balancer supports routing traffic to multiple ports on an EC2 instance. For example, an EC2 instance can run multiple applications on different ports on a single EC2 instance:

- EC2 can run the main web server on port `80`
- It can run the admin application on port `8080`
- It can run the reporting application on port `5000`

In such a scenario, Application Load Balancer can route all the traffic requests with host header as `www.example.com` to port `80` on the instance, all traffic requests with host header `www.example.com/admin` to port `8080` on the instance, and all the traffic requests with host header `www.example.com/reporting` to port `5000` on the instance.

Routing to multiple ports on a single instance is only supported in Application Load Balancer.

HTTP/2 support: HTTP/2, also called HTTP/2.0 is a major revision of the HTTP network protocol used on the internet. Using HTTP/2 features, web applications can increase its speed. It improves the way data is framed and transported. Websites can increase the efficiency and minimize the number of requests required to load an entire web page.

Application Load Balancer supports HTTP/2, industry standard protocol and provides better visibility on the health of the EC2 instances and micro servers or containers.

WebSockets support: WebSockets are an advanced technological development that enables the application to open an interactive communication session between the browser and an application server. If WebSockets are enabled, it allows you to send a message to an application server and receive event-driven responses from the server without polling the server for its response. It provides a persistent connection between a browser and the application server. The browser establishes the connection with the application server using a process named *WebSocket handshaking*. Once a session is established, the browser or the application can start sending the data unilaterally as and when required. Application Load Balancer supports WebSockets, whereas Classic Load Balancer does not support it.

ELB deletion protection: Application Load Balancer supports a configuration option that ensures that an ELB is not accidentally deleted by anybody. This option is called an ELB deletion protection. It is supported only by Application Load Balancer. If ELB deletion protection is enabled, you cannot delete the ELB unless this option is disabled again.

Step by step – Creating a Classic Load Balancer

Before creating a load balancer, it is necessary to ensure that desired configuration of VPC and EC2 instances are in place. If you create an ELB with EC2 instances spread across multiple AZs, ensure that the required instances are in place in each AZ to perform tests after creating the ELB. Also, verify that the security group attached to the respective EC2 instances allows incoming traffic on required ports and protocols. It is also recommended that the desired application or web server is configured properly on the target EC2 instances. With this background, let's follow the steps to create a Classic Load Balancer:

1. Log in to the AWS web console with sufficient credentials to create an ELB.
2. Go to EC2 dashboard and select L**oad Balancers** from the left-hand side pane, as shown in the following *Figure 6.1*:

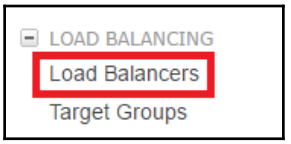


Figure 6.1: Load Balancer option on EC2 dashboard

3. Click on the **Create Load Balancer** button, as shown in the following *Figure 6.2*:

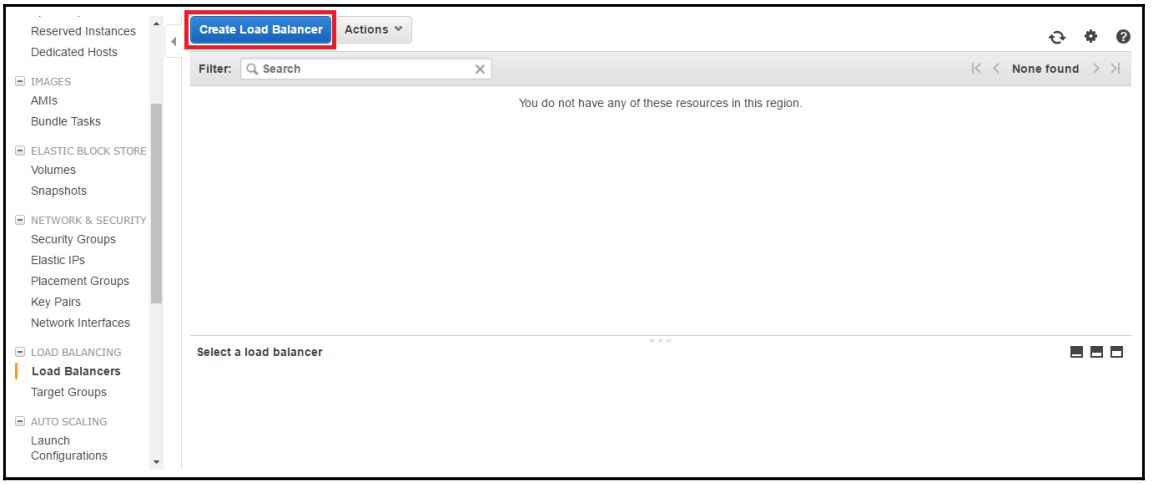


Figure 6.2: Create a Load Balancer button

4. Select the **Classic Load Balancer** type, as shown in the following *Figure 6.3*:

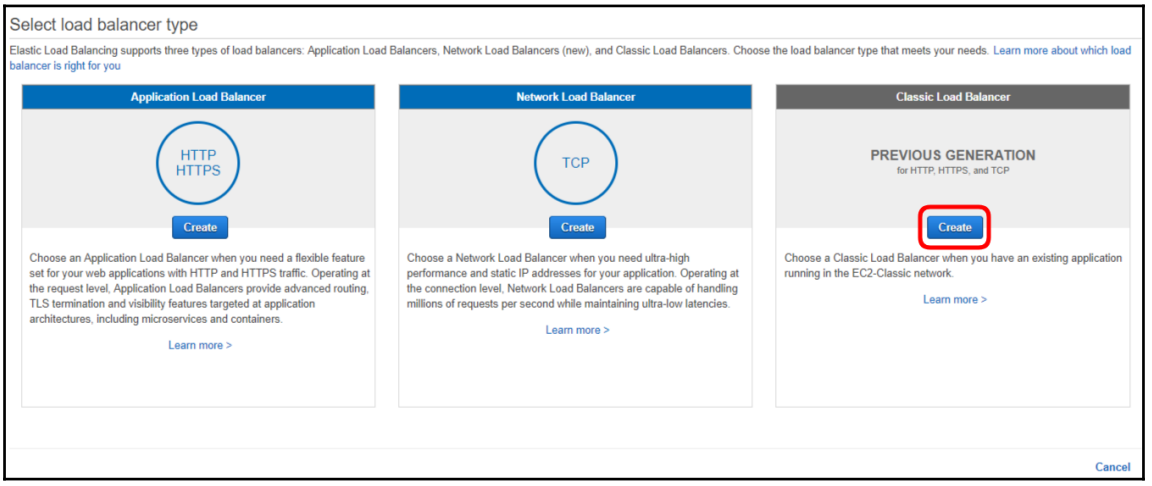


Figure 6.3: Select Classic Load Balancer

5. Next, we define Classic Load Balancer **Basic Configuration**:

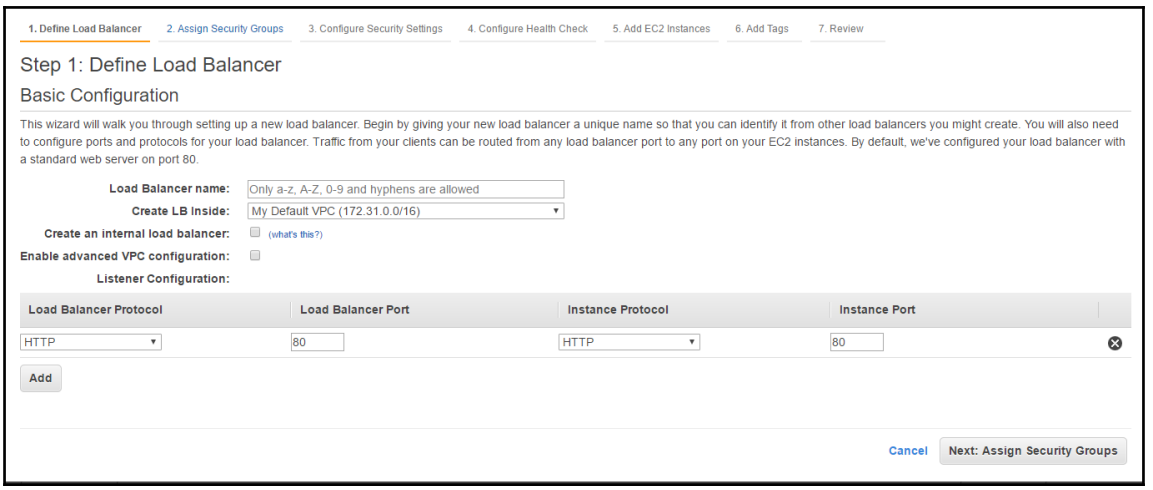


Figure 6.4: Classic Load Balancer -Basic Configuration

In this step, you need to understand the following ELB options:

- **Load Balancer name**: Provide a meaningful and relevant ELB name; it can be alphanumeric (A-Z, a-z, 0-9) and dash (-).
- **Create LB Inside**: Select the desired VPC, where EC2 instances reside. Only one VPC can be selected from the drop-down menu.
- **Create an Internal load balancer**: Select this option when creating a load balancer to manage traffic only from within an AWS environment. An internal load balancer cannot serve internet traffic and can be accessed from within the network or associated VPC.

- **Enable advanced VPC configuration**: Select this option to perform manual selection of available subnets within a selected VPC in the region. When you select this option, other relevant options become visible.
- **Listener Configuration**: Listener defines the ports and protocols which ELB opens for end users to send requests and it defines target ports and protocols on EC2 instances where ELB can forward the incoming traffic. When the listener port on ELB is selected as HTTPS, in consecutive steps, it asks to upload an SSL certificate along with essential cipher configuration.

6. Create a new security group or select an existing security group as shown in the following *Figure 6.5*. It is recommended to create individual security groups for every ELB. Security groups should open only the minimum required ports and protocols:

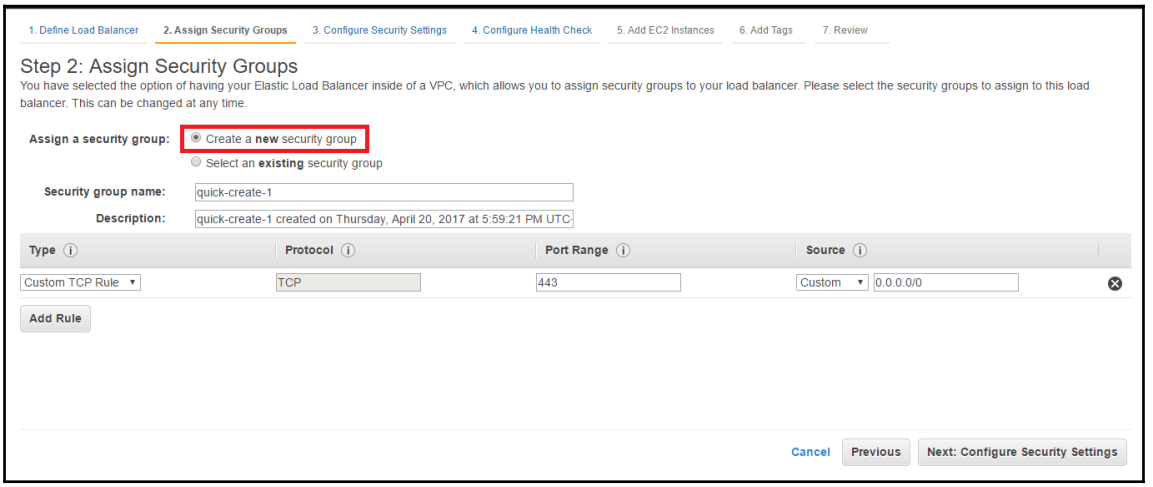


Figure 6.5: Assign Security Groups

7. Configure theSSL certificate. You can either C**hoose an existing certificate from AWS Certificate Manager (ACM)**, C**hoose an existing certificate from AWS Identity and Access Management (IAM)**, or U**pload a new SSL certificate to AWS Identity and Access Management (IAM)** as shown in the following *Figure 6.6*:

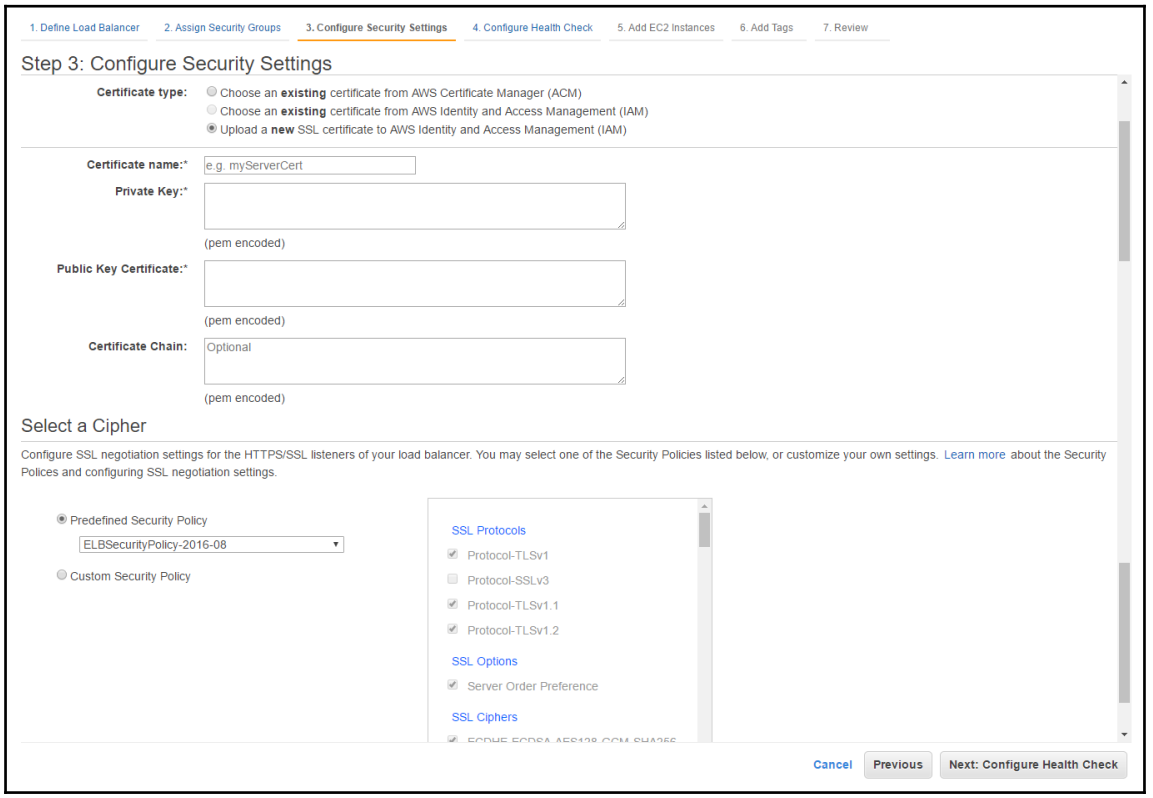


Figure 6.6: Configure Security Settings

When a listener is selected to listen on a HTTPS (`443`) port, it is essential to provide certificate details to move on to the next step.

8. The next step is to **Configure Health Check.** You can use the health check options table as a reference for configuring the health check options shown as follows:

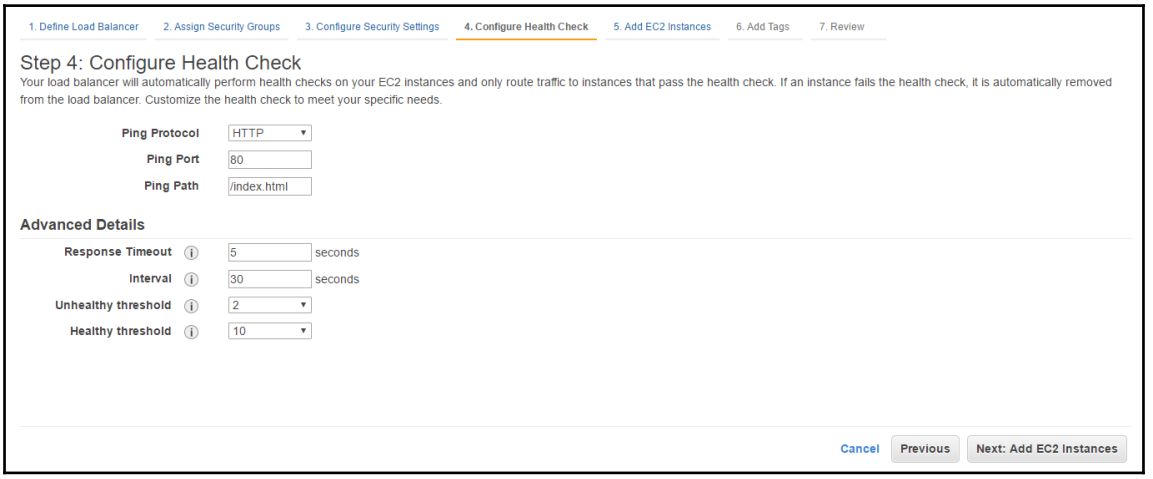


Figure 6.7: Configure Health Check

The various health check options are listed as follows:

Health check option	**Description**
Ping Protocol	Ping Protocol can be either HTTP, HTTPS, TCP, or SSL target instance should allow the selected protocol for pinging it.
Ping Port	This is the port on which ELB can send the ping request to instances. This port should be the port on which EC2 hosts the application. The default port is `80`, which can be changed based on where the application listens.
Ping Path	Ping Path points to a specific document or page on the EC2 instance. Ideally, this should be the path to the initial page loaded on the application. The ping is considered as a success if it is able to reach the page on a given path. To minimize the time to complete each ping, it is recommended to point it to the document root `(/)` rather than `/index.html`.

Healthy threshold	Healthy threshold is the threshold value in the range of 2 to 10. It defines the number of consecutive successful health checks before considering any EC2 instance as healthy instance.
Unhealthy threshold	Unhealthy threshold is the threshold value in the range of 2 to 10. It defines the number of consecutive failed health checks before considering any EC2 instance as unhealthy instance.
Response Timeout	This is an integer value in the range of 2 to 60 seconds. It defines the time interval in seconds. If an instance fails to respond during this time, ELB considers it as a failed health check. The instance is marked as an unhealthy instance if it crosses the Unhealthy threshold. Note: Health check timeout must be smaller than interval.
Interval	Interval can be given as an integer value in the range of 5 to 300 seconds. ELB waits for the number of seconds defined in the interval between two health checks.

9. Add EC2 instances from each of the AZs selected. Select **Enable Cross-Zone Load Balancing** and **Enable Connection Draining** as required. If connection draining is enabled, you can specify the number of seconds against it. ELB waits for the number of seconds defined for in-flight traffic to drain before forcing a session to close on deregistering an EC2 instance:

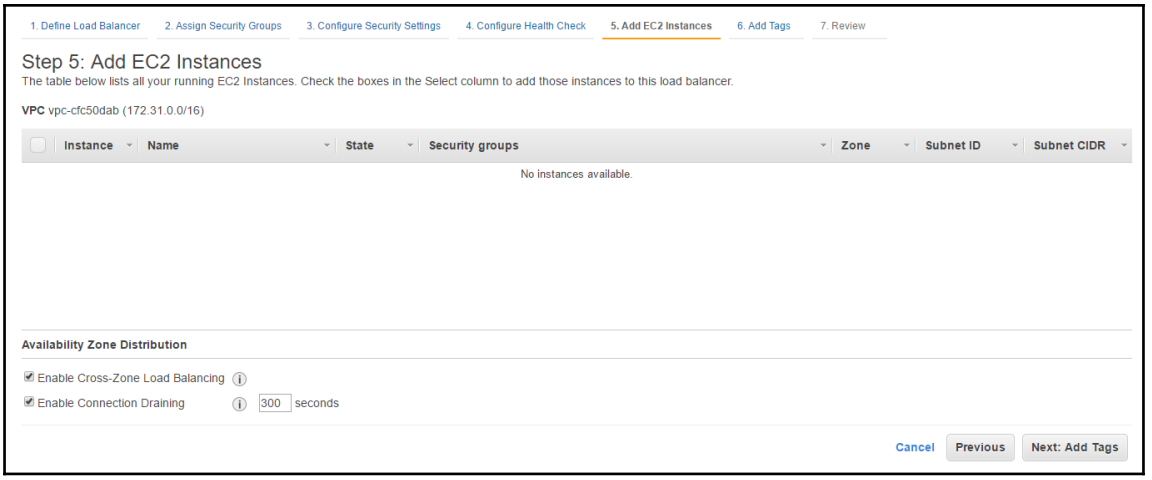


Figure 6.8: Add EC2 Instances to an ELB

10. Add tags as shown in the following *Figure 6.9*. It is recommended to give meaningful and relevant tags on an ELB as per enterprises naming conventions:

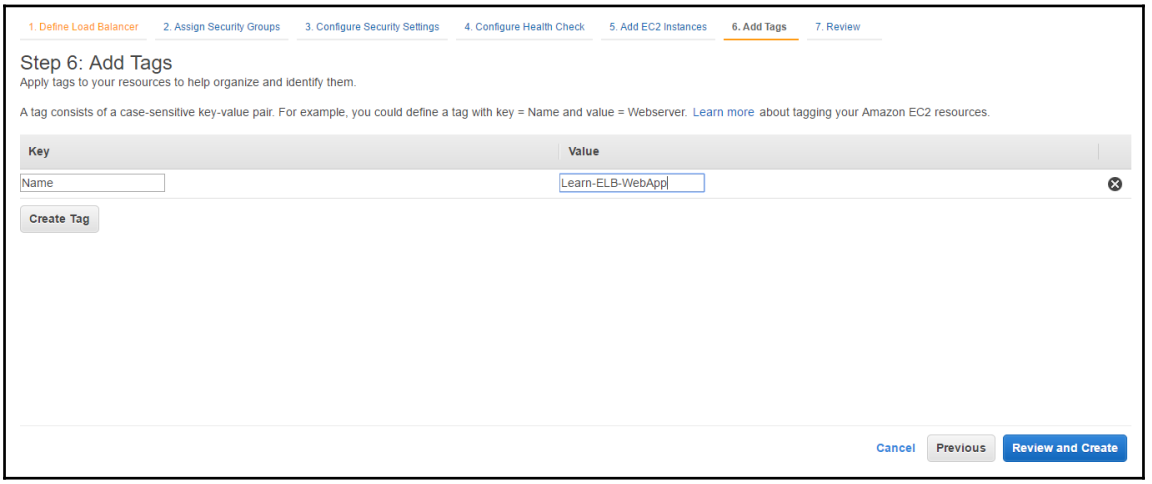


Figure 6.9: Add Tags to the ELB

11. Finally, click on **Review and Create** and verify the configuration. If everything is as per requirement, click on the **Create** button. Once ELB creation is completed, it provides an ELB endpoint. An ELB endpoint can be configured in CNAME record set in DNS to forward end user requests to the ELB.

How ELB works

Since all the ELB terminologies and features are explained, let us understand how exactly ELB works.

The working of a Classic Load Balancer

As per OSI model, Classic Load Balancer runs at Layer 4, which is the transport layer.

A transport layer level load balancer operates at the network protocol level. It does not check the content of actual network packets. A transport layer level load balancer does not check specific details of HTTP and HTTPS requests. In other words, it distributes the load without necessarily knowing much detail about the incoming traffic requests.

A user creates the Classic Load Balancer with instances spread across one or more AZs.

The load balancer configuration includes the following:

- Security group, which defines the security of ELB including the source of traffic that is allowed on the ELB, ports, and protocols open on the ELB for incoming traffic.
- Listener ports and protocols on which ELB listens for incoming traffic and target ports and protocols on EC2 instances where ELB directs the traffic.
- User includes SSL/TLS certification in the configuration.
- User defines health check for the associated instances. Health check enables the ELB to monitor the instances and determines whether an EC2 instance is *InService* or *OutOfService*. ELB routes traffic only to healthy instances with *InService* status.
- User can enable cross-zone load balancing to balance the traffic across all the instances in multiple AZs as required. If cross-zone load balancing is not enabled, traffic is routed on round-robin fashion between AZs.
- User can enable connection draining. If connection draining is enabled, before de-registering an instance from ELB, it allows in-transit sessions to complete for a defined span of seconds.
- Once an ELB is configured and associated with an application, AWS provides an ELB endpoint. ELB endpoint can be used to define a CName in Route53 or any other DNS.
- As described in the following *Figure 6.10*, a user can type in a site URL `www.example.com` in the browser.
- The user's request for the website hits a DNS server, which resolves to an ELB endpoint registered with the DNS and the request is forwarded to the ELB endpoint.
- The ELB endpoint resolves in a public IP address if the ELB is defined as an internet-facing ELB or it resolves in a private IP address if the ELB is defined as an internal-facing ELB:

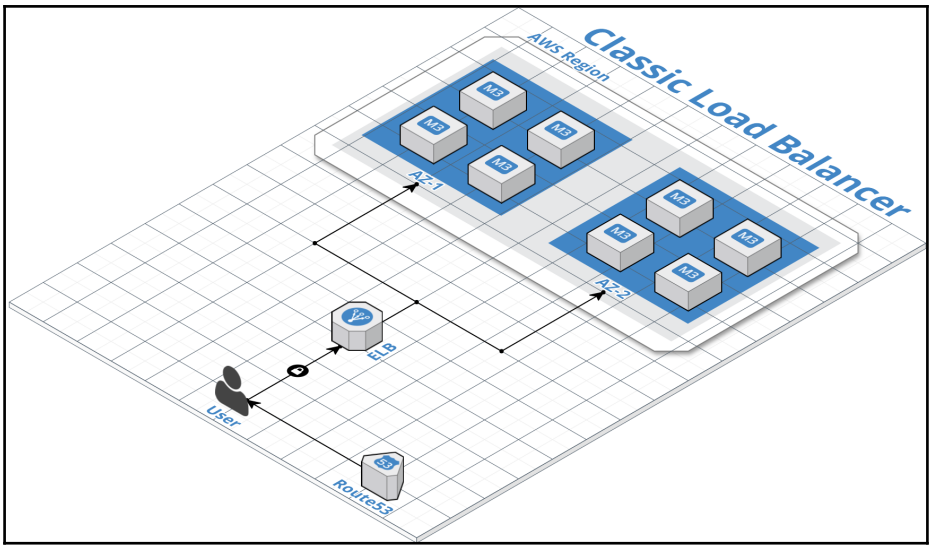


Figure 6.10: Classic Load Balancer

ELB receives the request forwarded by user and checks the source IP, host header, and cookie if sticky session is enabled.

If the source of the request is authorized to access the ELB, it forwards the traffic in a round-robin fashion to the AZs associated with the ELB. If cross-zone load balancing is enabled, the traffic is distributed between all the instances in round-robin fashion instead of distributing it at AZ level. If sticky session is enabled, ELB establishes a persistent session between client browser and EC2 instance. All subsequent requests from this user are forwarded to the same EC2 instance until the user session ends.

The target EC2 instance forwards the response, which is captured by ELB and forwarded back to the user.

The working of a Application Load Balancer

As per OSI model, Application Load Balancer runs at Layer 7, called the application layer.

Application Load Balancer is more powerful than Classic Load Balancer. It checks the traffic packets for more detail. It also has access to HTTP and HTTPS headers of the requests. It fetches more detail from the packets, which empowers it to do a more intelligent job in distributing the load to target instances.

Application Load Balancer supports content-based routing. You can enable host-based and path-based routing on an Application Load Balancer.

It also supports routing to multiple ports on a single instance.

It provides support for HTTP/2, which enables a website to increase efficiency and minimize the number of requests required to load a web page.

With the support of WebSockets, an Application Load Balancer can enable the application to open an interactive communication session between the browser and an application server.

A user creates the Application Load Balancer with instances spread across one or more target group in multiple AZs.

The load balancer configuration includes the following:

- Security Group, which defines the security of ELB including source of traffic that is allowed on the target group, ports, and protocols open on the ELB for incoming traffic.
- Listener ports and protocols on which ELB listens for incoming traffic and target ports and protocols on EC2 instances where ELB directs the traffic.
- Users can also define content-based routing. Content-based routing includes host-based routing or path-based routing. Depending upon the configuration, the traffic requests are forwarded to specific target groups.
- User includes SSL/TLS certification in the configuration.
- User defines health check for the associated instances. Health check enables the ELB to monitor the instances and determines whether an EC2 instance is *InService* or *OutOfService*. ELB routes traffic only to healthy instances with *InService* status.
- Cross-zone load balancing is automatically enabled in Application Load Balancer and the traffic is routed on a round-robin fashion across all the instances.
- User can enable connection draining. If connection draining is enabled, before deregistering an instance from ELB, it allows an in-transit session to complete for a defined span of seconds.
- Once an ELB is configured and associated with an application, AWS provides an ELB endpoint. The ELB endpoint can be used to define a CName in Route 53 or any other DNS.
- A user can type in a site URL `www.example.com` in the browser.

- User request for the website hits a DNS server, which resolves to an ELB endpoint registered with the DNS and the request is forwarded to the ELB endpoint.
- ELB endpoint resolves in a public IP address if the ELB is defined as an internet-facing ELB or it resolves in a private IP address if the ELB is defined as an internal-facing ELB.

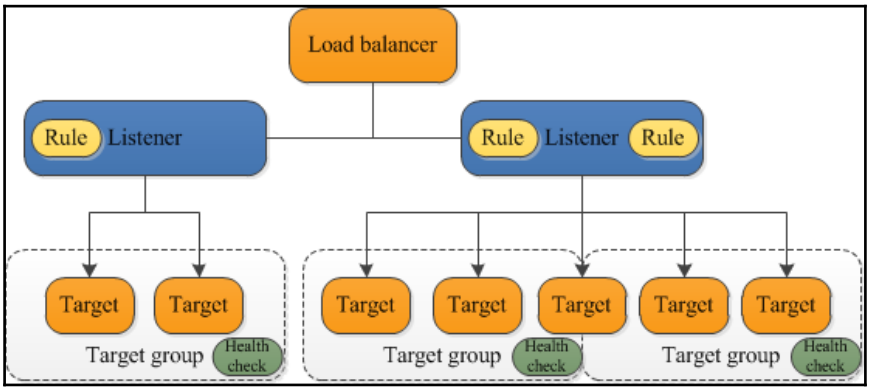


Figure 6.11: Application Load Balancer

ELB receives the request forwarded by the user, and then checks the source IP, host header, and cookie if sticky session is enabled.

If the source of the request is authorized to access the ELB, it forwards the traffic in a round-robin fashion to the target group associated with the ELB. The traffic is distributed between all the instances in round-robin fashion. If sticky session is enabled, ELB establishes a persistent session between client browser and EC2 instance. All subsequent requests from this user are forwarded to the same target until a session lasts.

The target EC2 instance forwards the response, which is captured by ELB and forwarded back to the user.

ELB best practices

ELB best practices are as follows:

- While defining a load balancer, it is recommended to identify target AZs and target group
- Use multiple AZs in ELB as it provides high availability and fault tolerance
- It is highly recommended that a Security group for the ELB opens only required ports and protocols
- Always configure health checks for ELB on appropriate ports and protocols
- If the ELB is created for a web server, use HTTP/HTTPS protocol in health checks instead of TCP protocol
- Do not create internet-facing ELB for internal needs
- Use SSL security certificates to encrypt and decrypt HTTPS connections where possible
- If a heavy traffic spike is expected on a given schedule, contact AWS support and ask them to pre-warm the ELB
- Use ELB deletion protection from accidental deletion
- Use cross-zone load balancing in Classic Load Balancer for evenly distributing the load across all EC2 instances in associated AZs
- Carefully enable connection draining on ELBs associated with critical user applications

7
Monitoring with CloudWatch

CloudWatch is an AWS service, that can be used on the AWS cloud for monitoring various infrastructure and application resources running on your AWS cloud. CloudWatch can be used to collect a number of metrics from the AWS resources. It allows you to track these metrics and also initiate actions based on the threshold you set. CloudWatch can also collect log files, generate metrics out of them, and help to monitor log files. You can set alarms on specific events and trigger an action whenever an event occurs. For example, if CPU utilization for a specific instance crosses a threshold of 80%, you can initiate an action to spin up a new instance.

CloudWatch supports the monitoring of many AWS services such as EC2 instances, DynamoDB, RDS, and so on. You can also generate custom metrics and log files using your own applications and associate them with CloudWatch. Amazon services like Auto Scaling uses CloudWatch alarms to automatically scale an environment up or down, based on the traffic on an environment.

CloudWatch provides a number of graphs and statistics. It gives your system wide insights into how resources are utilized, how to monitor application performances, and track the overall operational health of respective applications in an environment. All these infrastructure and application telemetry data can be used to ensure smooth functioning of your environment.

How Amazon CloudWatch works

CloudWatch acts as a repository of metrics, by collating raw data from various AWS services or applications, converting it into metrics, statistics, graphs, and facilitates certain actions based on specific data points in metrics. The following *Figure 7.1* shows the high-level architecture of CloudWatch:

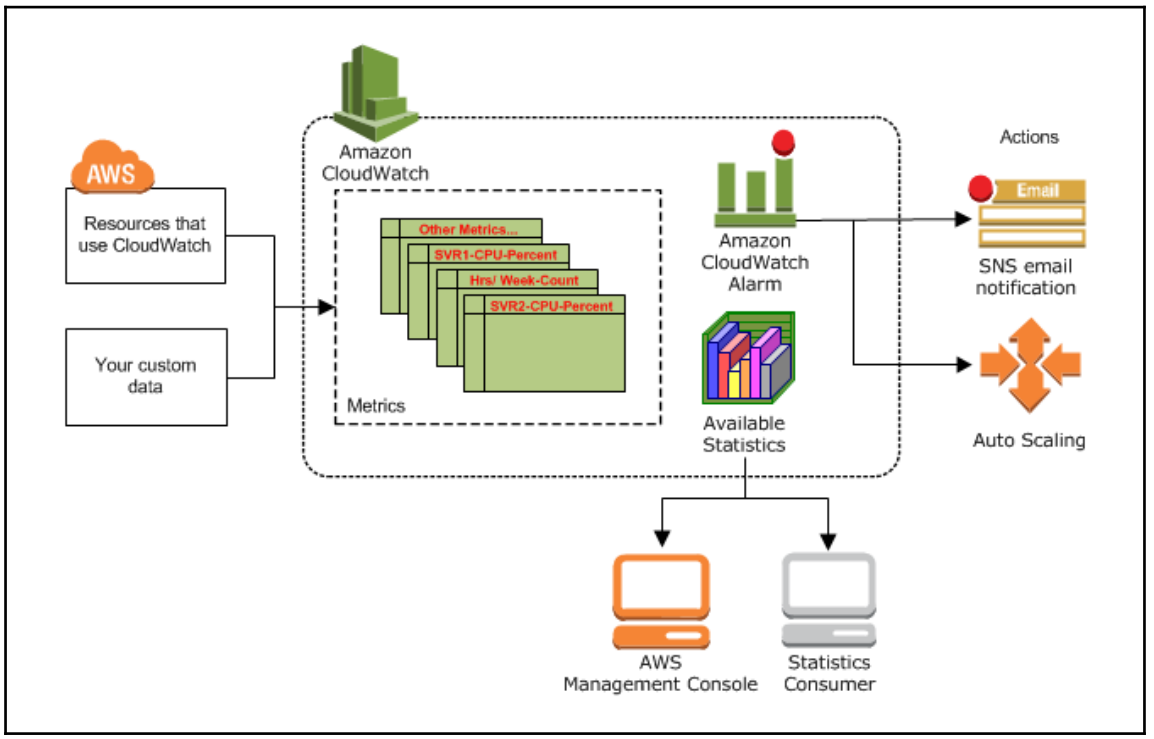


Figure 7.1: High level architecture of CloudWatch

Reference URL: https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/cloudwatch_architecture.html

As shown in the preceding figure, various AWS services and your custom metrics data are stored in CloudWatch. CloudWatch generates various statistical and graphical visualizations out of these metrics which can be consumed directly from the AWS Management Console or by various other means including, but not limited, to AWS CLI, API, custom build applications, and so on. CloudWatch enables the user to set alarms, which can trigger certain actions based on the metrics threshold or events. It can send email notifications or automatically scale an environment up or down using Auto Scaling groups associated with alarms.

Elements of Amazon CloudWatch

To understand and work with AWS-generated and custom metrics, it is important to understand a few basic concepts and terminologies used with Amazon CloudWatch.

Namespaces

CloudWatch namespaces are containers in which metrics for different applications are stored. It is a mechanism to isolate metrics of different applications from each other. Namespaces ensure that an application's metrics, as well as respective statistical data, are not accidentally mixed up with any other application's metrics. All the AWS services that use CloudWatch to register their metrics, use a unique namespace. A namespace name begins with `AWS/` and is generally followed by the application name. If you create a custom application and need to store metrics in CloudWatch, you must specify a unique namespace as a container to store custom metrics. Namespace can be defined at the time of creating the metrics. Namespace name can be a string of up to 256 characters including (`A-Z`, `a-z`, `0-9`), hyphen (`-`), underscore (_), period (`.`), slash (`/`), hash (`#`), and colon (`:`).

Some of the namespaces are given in the following table. For more details on namespace, you can refer to URL: `http://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/aws-namespaces.html`.

AWS product	**Namespace**
Amazon API Gateway	`AWS/ApiGateway`
Amazon CloudFront	`AWS/CloudFront`
Amazon CloudSearch	`AWS/CloudSearch`
Amazon CloudWatch Events	`AWS/Events`
Amazon CloudWatch Logs	`AWS/Logs`
Amazon DynamoDB	`AWS/DynamoDB`
Amazon EC2	`AWS/EC2`
Amazon EC2 (Spot instances)	`AWS/EC2Spot`
Amazon EC2 Container Service	`AWS/ECS`
Amazon EBS	`AWS/EBS`
Amazon EFS	`AWS/EFS`

Amazon Elastic Transcoder	`AWS/ElasticTranscoder`
Amazon ElastiCache	`AWS/ElastiCache`
Amazon Elasticsearch Service	`AWS/ES`
Amazon EMR	`AWS/ElasticMapReduce`
Amazon Kinesis Analytics	`AWS/KinesisAnalytics`
Amazon Kinesis Firehose	`AWS/Firehose`
Amazon Kinesis Streams	`AWS/Kinesis`
Amazon RDS	`AWS/RDS`
Amazon Route 53	`AWS/Route53`
Amazon Simple Email Service	`AWS/SES`
Amazon SNS	`AWS/SNS`
Amazon Simple Queue Service	`AWS/SQS`
Amazon S3	`AWS/S3`
Amazon Simple Workflow Service	`AWS/SWF`

Metrics

Metrics are set of data collected over a period of time with a specific time interval for quantitative assessment, measurement, and comparison of performance data generated by a specific application or a service. For example, CPU utilization data for an EC2 instance is stored in a relevant CloudWatch metrics at time interval of 1 minute. Each AWS service stores several metrics in CloudWatch. For an instance, EC2 stores `CPUUtilization`, `NetworkIn`, `NetworkOut`, `DiskReadOps`, `DiskWriteOps`, and so on. CloudWatch by default, stores metrics data at an interval of 5 minutes. If you enable advance monitoring, it can store a metrics data point at an interval of 1 minute.

CloudWatch retains all metrics for 15 months before discarding them, however lower-level granularity of data is deleted to keep the overall volume of data at a reasonable size.

Granularity of data	**Data retention period**
60 seconds (1 minute)	15 days
300 seconds (5 minutes)	63 days
3600 seconds (1 hour)	455 days (15 months)

CloudWatch also allows you to create custom metrics with a custom name and the required time interval to store data point. Metrics are region based. AWS generated default metrics or custom metrics can be found only in the region where it is created. Metrics cannot be deleted manually. They automatically expire after 15 months from the time of last published data point into the metrics.

Metrics are unique in nature, defined by the combination of a namespace, metric name, and one or more dimensions. Each data point has a timestamp, data point, and optionally, a unit of measurement.

Dimensions

A dimension in a CloudWatch metrics is a mechanism to uniquely identify metrics. It is a name/value pair that is associated with metrics.

For example, CloudWatch stores metrics for EC2 instances in a namespace called `AWS/EC2`. All EC2 metrics are stored in the specific namespace. If you need to retrieve metrics for a specific `InstanceID`, you search the metrics with its InstanceID like `i-09x7xxx4x0x688x43`.

In the preceding example, `InstanceID` is its name and `i-09x7xxx4x0x688x43` is its value which represents a dimension. When you search the metrics with the `InstanceID` value, you are using a dimension. A dimension is a unique identifier of metrics. When you search with a different value of `InstanceId`, CloudWatch provides you with a different metrics that is related with another instance. Similarly, you can search metrics with a different dimension name. For example, you can search the EC2 metrics with an `ImageID`. Metrics can have a maximum of 10 dimensions. Here's the list of dimensions in EC2.

- `InstanceId`
- `ImageId`

- `InstanceType`
- `AutoScalingGroupName`

Just like EC2, other services have their own dimensions. You can also create a metrics with it's custom dimensions.

Statistics

Statistics are a collection of aggregated metrics data for a specific period of time. Metrics data is aggregated using namespace, metric name, dimensions, and several data points in a given time period. CloudWatch provides the following statistics on the metrics data.

Statistic	**Description**
`Minimum`	The least, smallest, or lowest value recorded in the metrics for a specific period of time. For example, lowest CPU utilization recorded during the day on an EC2 instance.
`Maximum`	The largest, biggest, or highest value recorded in the metrics for a specific period of time. For example, highest CPU utilization recorded during the day on an EC2 instance.
`Sum`	Total value resulting from addition of all the matching metrics for a specific period of time. It is useful to study the total volume of activities. For example, total data transferred on an EC2 instance in the past 24 hours.
`Average`	Average value resulting from addition of all the matching metrics for a specific period of time and dividing it by the total number of sample count. For example, average CPU utilization on an EC2 instance for the last one hour.
`SampleCount`	Simply counts the number of data points used for the statistical calculation.
`pNN.NN`	The value represented in percentile. It uses up to two decimal places to specify a number. It helps to study statistics in terms of percentile such as p80.43.

At any given point of time, a data point may contain more than one value as shown in the following table. It describes some statistics with sample data.

Hour	Raw data	Sum	Minimum	Maximum	SampleCount
1	80, 85, 75, 70, 77	387	70	85	5
2	60, 80, 70, 75, 65	350	60	80	5

Percentile

A percentile helps in finding the comparative standing of a value in a set of data. Let us take an example of a data set that contains the CPU utilization of an EC2 instance. The example data is arranged in ascending order for better understanding.

12	17	25	55	58	61	63	70	83	97

Let us consider the percentile for `83` from the preceding data set. In this example, `83` stands at the 90th percentile. That means, `90` percent of data is less than or equal to `83` and the remaining `10` percent is above `83`.

Percentile gives a better insight of the data with better understanding on how the data is distributed. The following AWS services support percentiles.

- EC2
- Application Load Balancer
- Classic Load Balancer
- RDS
- Kinesis
- API Gateway

CloudWatch provides options to monitor systems and applications using various statistics such as maximum, minimum, average, sum, or percentile. If you choose percentile, CloudWatch starts displaying the statistics according to percentile value.

Alarms

CloudWatch alarms help in defining a threshold value that is constantly monitored, and an action is triggered when the threshold condition is breached.

For example, you can define a threshold of 80% CPU utilization on an EC2 instance and trigger an action whenever the CPU utilization is >=80 for three consecutive periods.

The action can be one or more SNS notification, Auto Scaling action, or an EC2 action. An SNS notification can be used to send an alert over mail, an SMS over mobile, and it can also trigger a Lambda Function. Auto Scaling action is used for scaling an environment up by adding more instances or scaling an environment down by reducing the number of instances. EC2 action can be used for rebooting, stopping, terminating, or initiating a recovery on the instance.

An alarm can have three possible states:

- Alarm status displays `OK` when the metric is within the defined threshold
- Alarm status displays `ALARM` when the metric is outside of the defined threshold
- Alarm status displays `INSUFFICIENT_DATA` when the alarm is just configured, the metric is not available, or not enough data is available for the metric to determine the alarm state

Creating a CloudWatch alarm

The following steps describe the process of creating a CloudWatch alarm:

1. Open CloudWatch console by navigating to `https://console.aws.amazon.com/cloudwatch/` on your browser. It brings you to the CloudWatch dashboard.
2. Click on **Alarms** as shown in the following *Figure 7.2*:

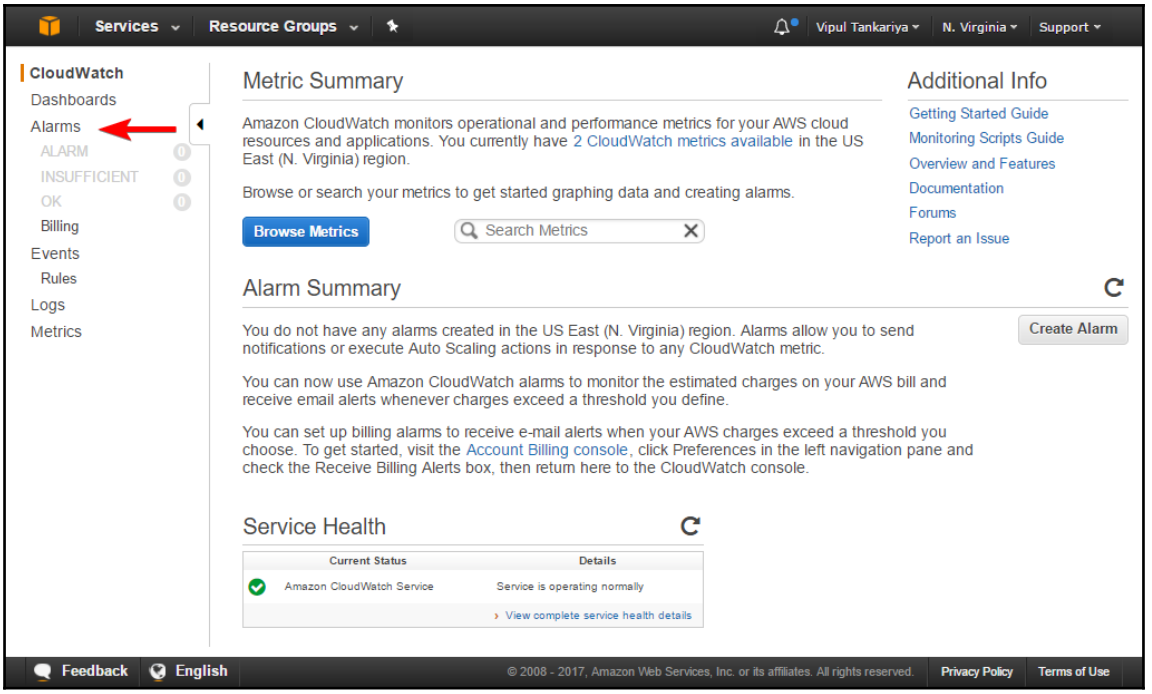


Figure 7.2: CloudWatch dashboard

3. Click on the **Create Alarm** button as shown in the following *Figure 7.3*:

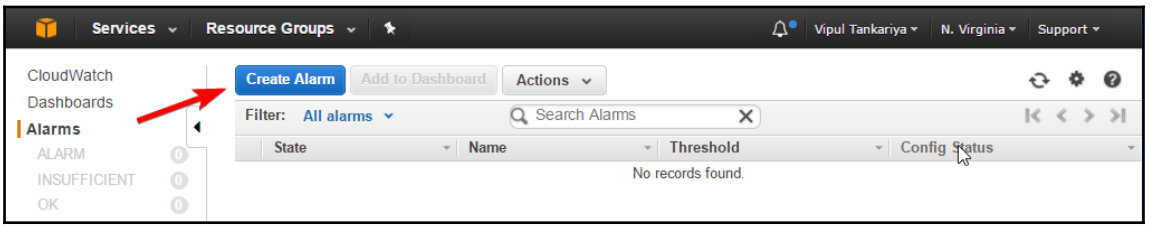


Figure 7.3: Create Alarm

4. Click on **Per-Instance Metrics:** as shown in the next *Figure 7.4*. This window shows multiple categories of metrics depending upon the metrics you have in the account. For example, if you have EC2 instances in the account, it shows **EC2 Metrics:**, if you have ELB resources in the account, it shows **ELB Metrics** and similarly, it shows different categories of metrics for which you have resources in the account. Depending upon the requirement, you can select any of the category metrics as needed:

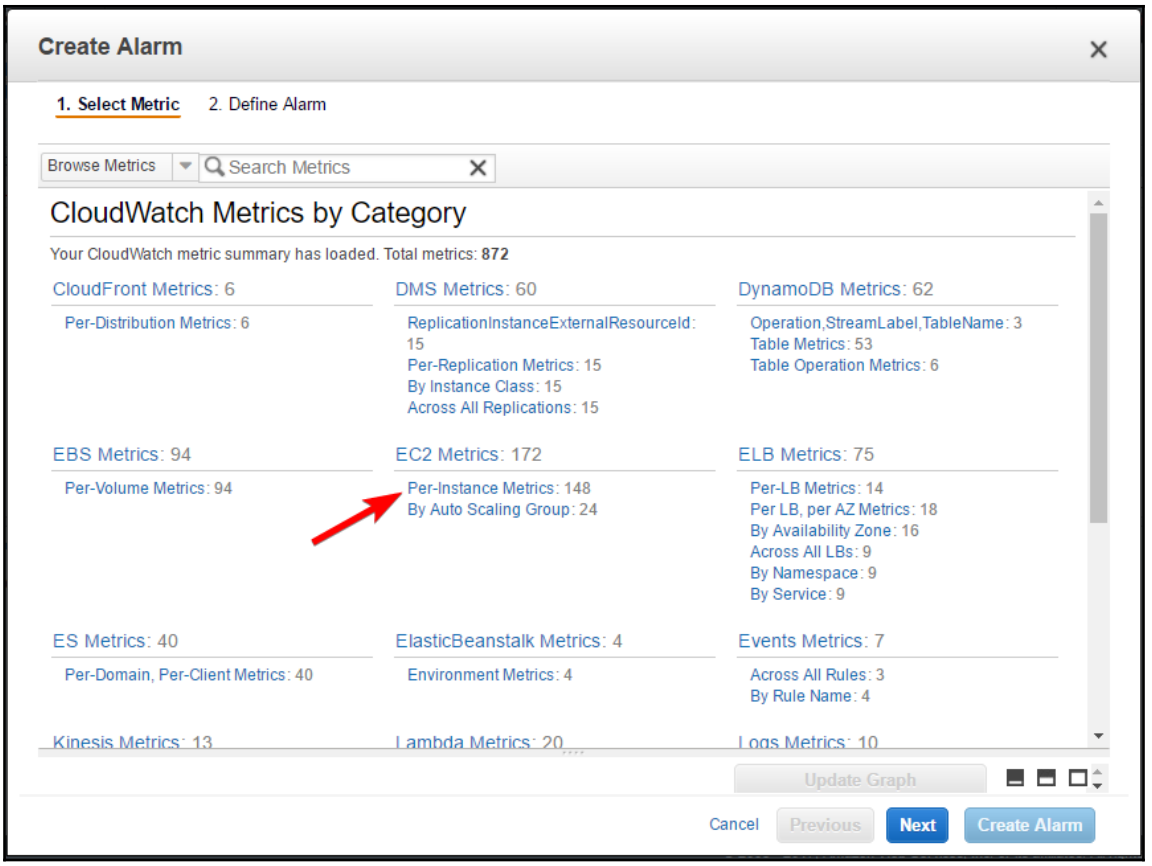


Figure 7.4: CloudWatch Metrics by Category

5. There are a number of metrics such as **CPUUtilization**, **NetworkIn**, **NetworkOut**, and so on. You can select any of these metrics as needed. For this example, select a metrics against an EC2 instance for **StatusCheckFailed_Instance** as shown in the following *Figure 7.5* and click on the **Next** button:

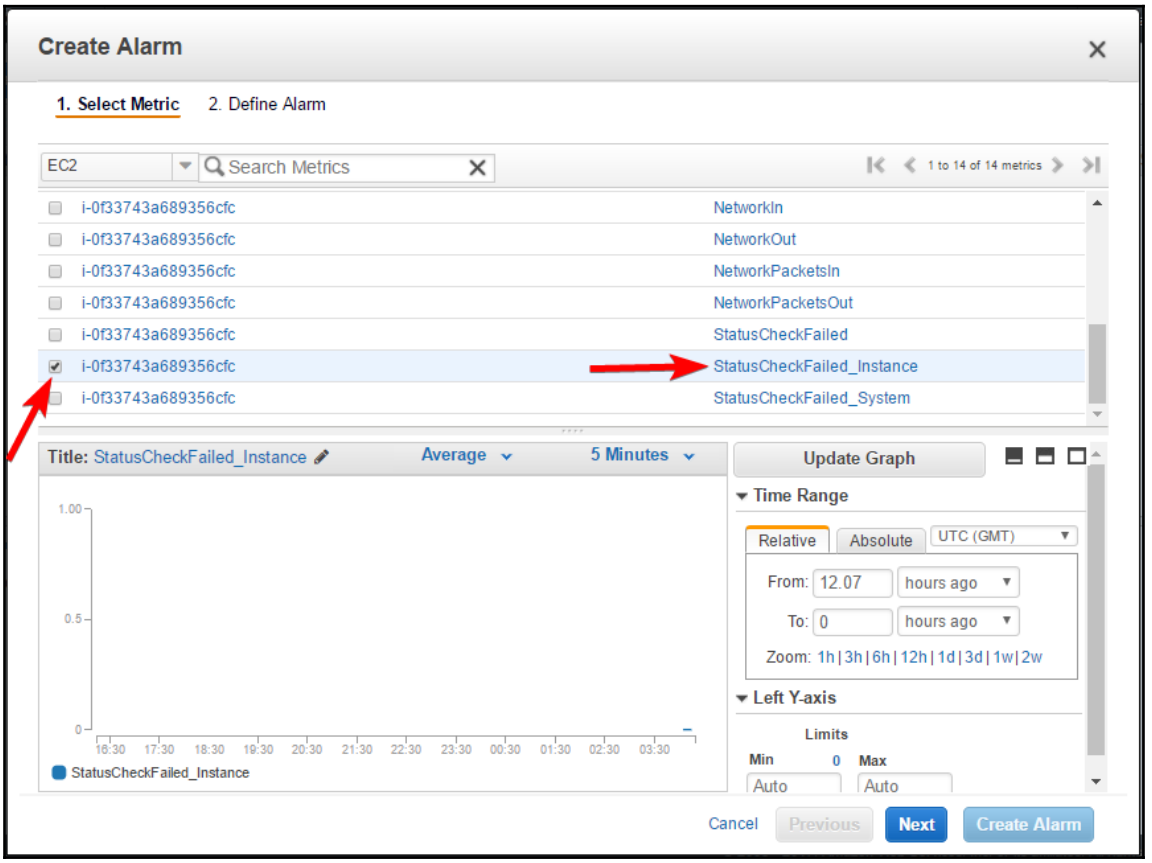


Figure 7.5: Select a Metrics

6. In the subsequent window, provide the alarm **Name**, **Description**, and threshold values:

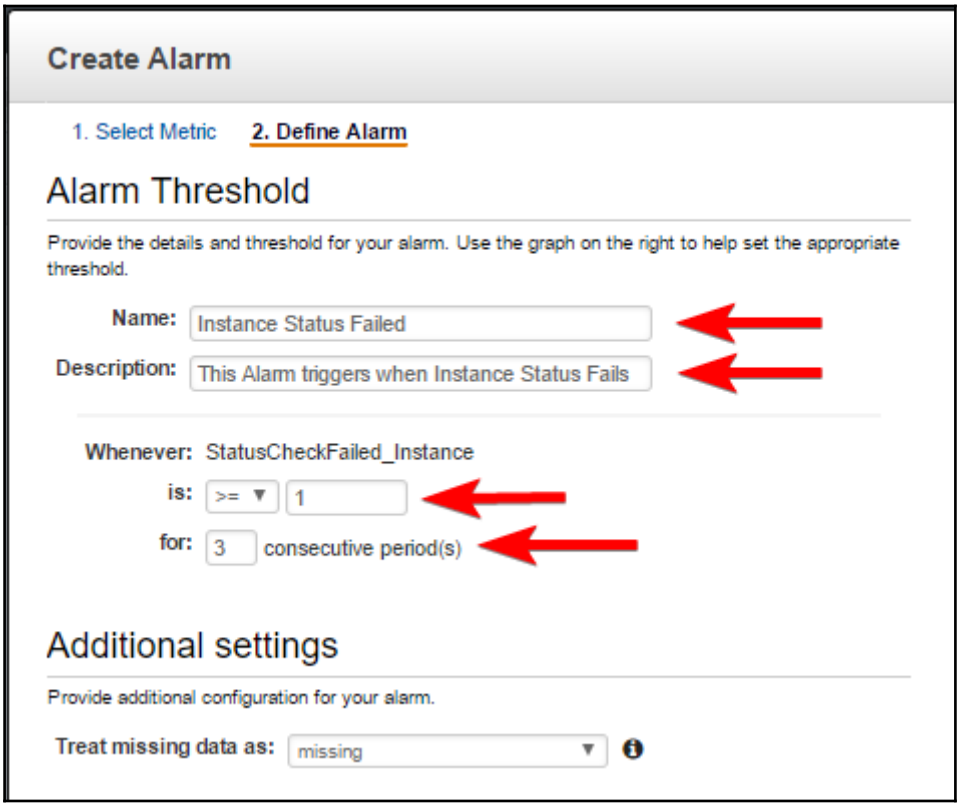


Figure 7.6: Alarm Threshold

7. In the **Actions** section, action can be one or more from an SNS notification, Auto Scaling action, or an EC2 action. An SNS notification can be used to send an alert over mail or an SMS over mobile. Auto Scaling action is used for scaling an environment up by adding more instances or scaling an environment down by reducing the number of instances. EC2 action can be used for rebooting, stopping, terminating, or initiating a recovery on the instance. For this example, in the same window select the **+ EC2 Action** button which is shown as follows:

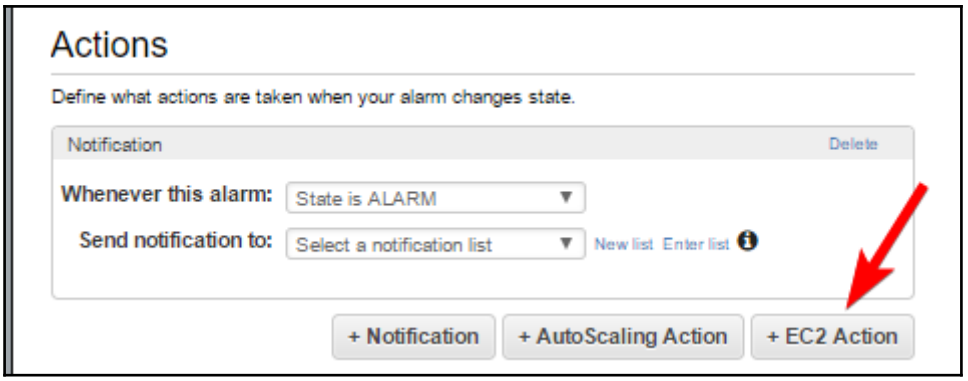


Figure 7.7: Actions

8. When you click **+EC2 Action**, it displays the following *Figure 7.8*. Select **State is ALARM** against **Whenever this alarm** line item and select **Recover this instance** from the action list:

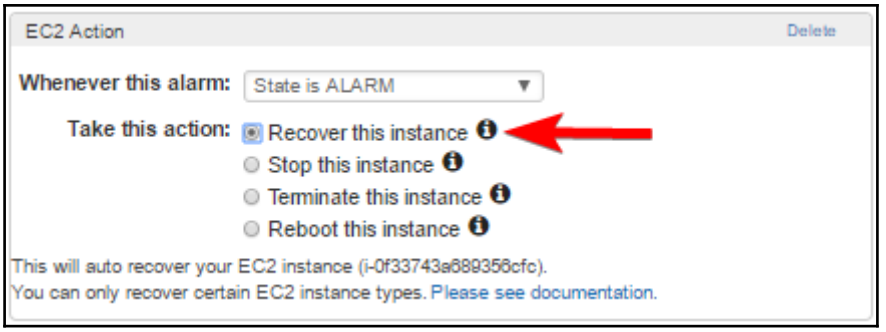


Figure 7.8: EC2 Action

9. If the instance is configured for basic monitoring, you have to select a minimum of 5 minutes or a higher period as shown in the next *Figure 7.9*. An instance can be configured for basic monitoring or it can be configured for advance monitoring. With basic monitoring, instance is monitored at an interval of 5 minutes. With advance monitoring, instance is monitored at an interval of 1 minute. Depending upon the consecutive alarm period selected in the **Alarm Threshold** section and the interval period for checking the alarm, **Alarm Preview** shows the total duration for actions to trigger:

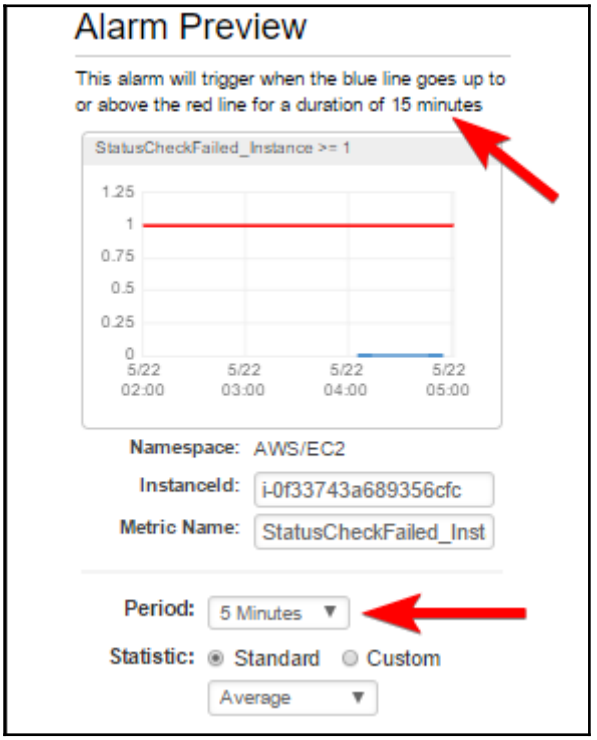


Figure 7.9: Alarm Preview

10. You can click on the **Create Alarm** button after selecting all the required options in the **Create Alarm** window, this creates the alarm. Once the alarm is created, it gets triggered based on the alarm status. In this example, alarm actions are triggered when the instance status is failed for three consecutive periods of 5 minutes each. Instance status failure alarm occurs when there is a hardware issue or any issue with the virtualization host where the EC2 instance is hosted on AWS. If you set this alarm, AWS automatically tries to recover the instance.

You can follow similar steps for creating different types of alarms based on a metric of your choice. For example, if you choose **CPUUtilization** metrics, you can either send an alert to an administrator when the CPU utilization breaches the threshold or you can perform Auto Scaling actions. Auto Scaling actions can add or remove instances in an Auto Scaling group to optimize the resources required for serving the traffic.

Billing alerts

Just as CloudWatch monitors other AWS resources, it can also monitor the monthly billing charges for an AWS account. You can set the threshold for billing. As soon as the billing amount reaches the threshold or shoots above the specified threshold, it notifies the specified administrators. You need to enable billing alerts before you can configure alerts on billing data. You can enable billing alerts from AWS account settings. Remember, only root user can enable the billing alerts, AWS IAM user cannot do that.

The process of enabling billing alerts is discussed as follows:

1. Log in to the AWS account with the root user credentials.
2. Click on the account name and **My Billing Dashboard** as shown in the following *Figure 7.10*:

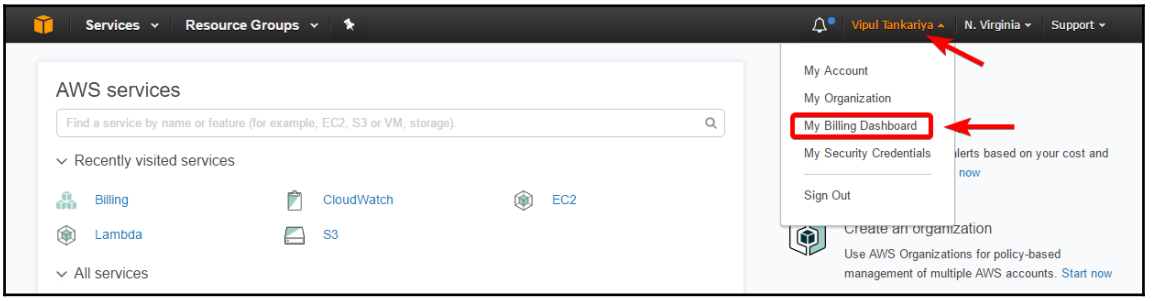


Figure 7.10: Opening My Billing dashboard

3. From the left hand side pane, click on the **Preferences** and check **Receive Billing Alerts**. Optionally, you can also enable **Receive PDF invoice By Email** and **Receive Billing Reports:**

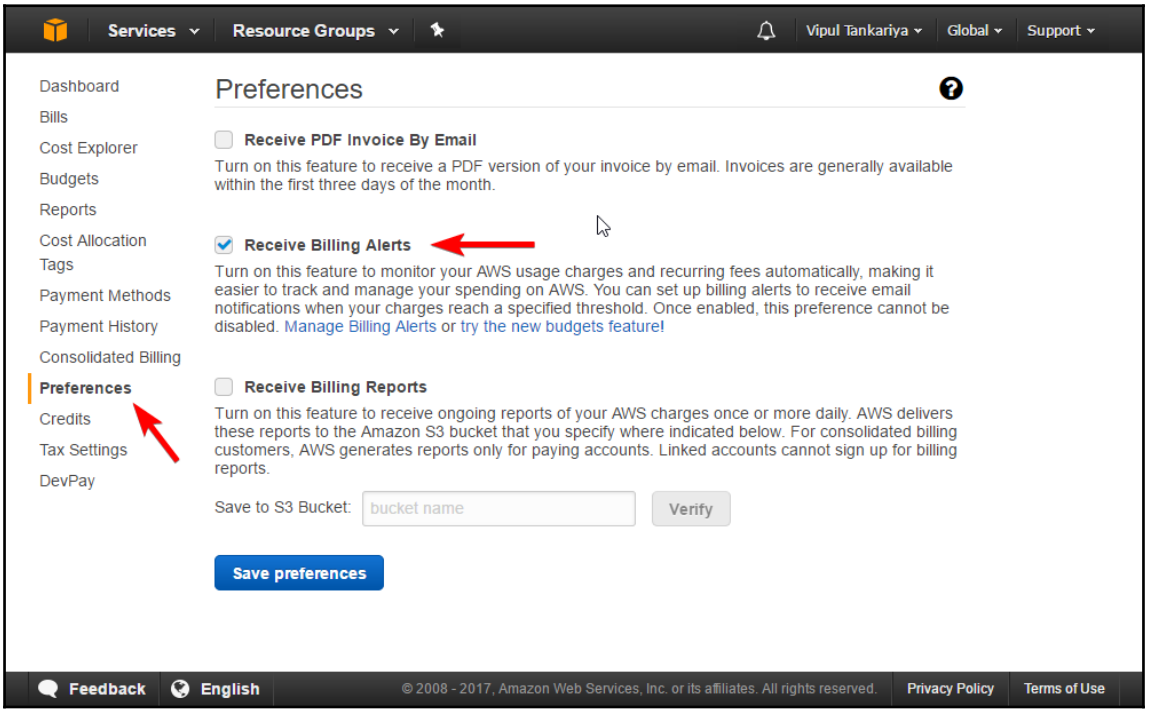


Figure 7.11: Billing dashboard preferences

CloudWatch dashboards

Amazon CloudWatch provides a customizable dashboard inside a web console. It can display a set of critical metrics together. You can create multiple dashboards wherein each dashboard can focus on providing a distinct view of your environment. You can create a custom dashboard to view and monitor the selected AWS resources from the same or different regions. It provides a way to get a single view of critical resource metrics and alarms for observing performance and health of the environment. It gives freedom to add, remove, move, resize, and rename the graphs, as well as change the refresh interval of selected graphs in a dashboard.

Monitoring types – basic and detailed

Amazon CloudWatch monitoring can be broadly categorized into two categories: basic monitoring and detailed monitoring.

- **Basic monitoring:** Basic monitoring is free and it collects data at a 5-minute time interval. By default, when you provision AWS resources, all AWS resources except ELB and RDS start with a basic monitoring mode only. ELB and RDS monitors the resources at a 1-minute interval. For other resources, optionally, you can switch the monitoring mode to detailed monitoring.
- **Detailed monitoring:** Detailed monitoring is chargeable and it makes data available at a 1-minute time interval. Currently, AWS charges $0.015 per hour per instance. Detailed monitoring does not change the monitoring on ELB and RDS which by default collates data at a 1-minute interval. Similarly, detailed monitoring does not change the EBS volumes which are monitored at 5-minute intervals.

You can enable detailed monitoring while launching an instance or after provisioning the instances. While launching an EC2 instance, it provides an option to enable detailed monitoring in the third step as shown in *Figure 7.12*. You can refer to `Chapter 5`, *Getting Started with Elastic Compute Cloud* which provides more details on how to launch an EC2 instance:

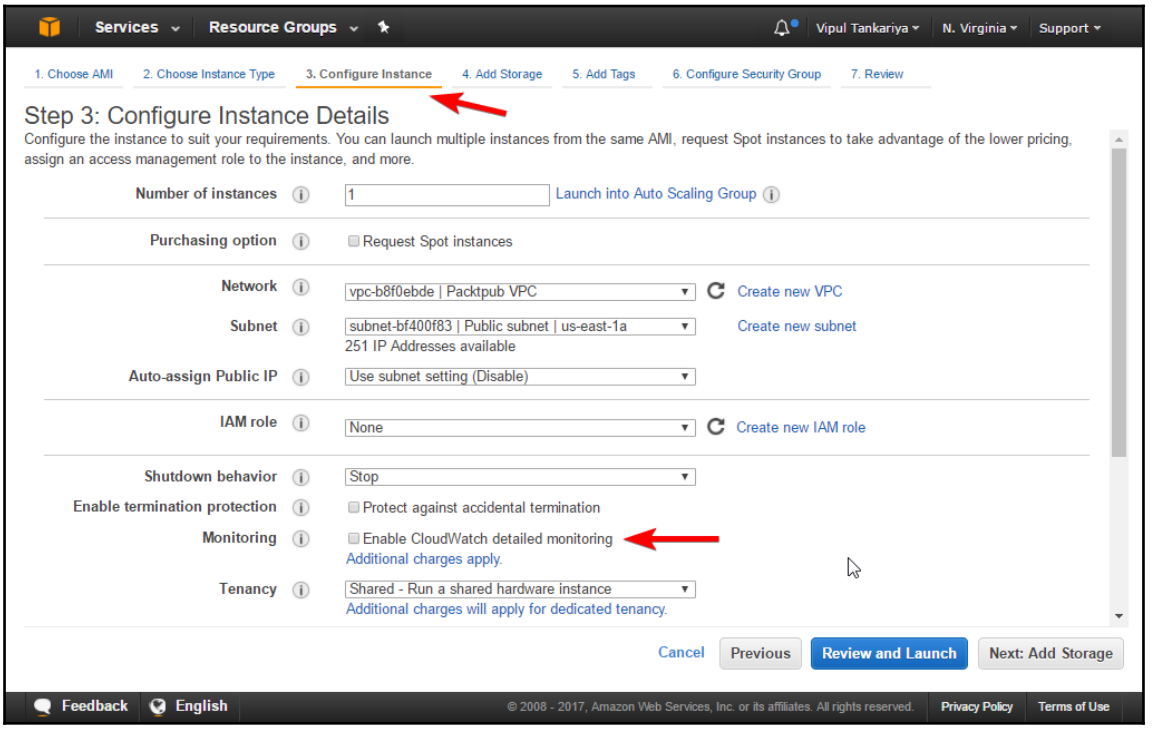


Figure 7.12: Enable detailed monitoring while launching an EC2 instance

For enabling detailed monitoring on an existing EC2 instance, follow these steps:

1. Navigate to `https://console.aws.amazon.com/ec2/` for opening an EC2 console from your browser.
2. From the left hand side navigation pane, select **Instances**.
3. Once the instance list is launched, select an instance for which you want to change the monitoring type.
4. Select **Actions | CloudWatch Monitoring | Enable Detailed Monitoring**.
5. In the **Enable Detailed Monitoring** dialog box choose **Yes, Enable**.
6. Click on **Close**.

CloudWatch best practices

Here are the best practices that we can follow in CloudWatch:

- It is best practice to monitor AWS resources and hosted applications using CloudWatch. It helps in identifying performance bottlenecks and also helps in optimizing resource costs.
- It is recommended that you enable billing alerts for an AWS account. It helps to monitor the monthly costs and keep a tab on them.
- For better understanding of CloudWatch visualization, toggle the metrics between UTC and local time.
- By default, CloudWatch provides basic monitoring for resources and records metrics at a 5-minute interval. It is recommended thay you use detailed monitoring for critical resources, which records metrics at a 1-minute interval.
- Enable custom metrics where required. For example, you can enable memory monitoring on EC2 instance, which is not part of the default EC2 metrics.
- Create custom metrics for monitoring application behaviour and link them to CloudWatch. They provide better insight on the application. For example, custom metrics for monitoring JVM can be created for Java-based applications.
- Enable Auto Recovery and Auto Restart Alarms for your critical EC2 instances. It can automatically recover the instances from hardware related or virtualization host related issues on AWS. Also, Auto Restart Alarm can recover the instance from operating system level issues.
- It is recommended that you automate monitoring tasks as much as possible.
- Upload your critical custom logs to CloudWatch for quick statistical analysis.
- While creating custom metrics, verify log files on your EC2 instances with CloudWatch metrics to ensure that the right data synchronizes with CloudWatch.
- Enable SNS notifications for critical metrics threshold breach.
- AWS provides 10 alarms per month per customer for free. If you're intending to operate in a free tier, ensure that you do not exceed this limit.
- AWS supports a maximum of up to 5000 alarms for a customer in a region. If you are a heavy user of alarms, plan your alarms in such a way that all critical alarms are created before reaching the limit.

- CloudWatch does not validate actions while configuring them. Ensure that configured actions exist and are validated properly. For example, ensure SNS group has valid email IDs associated with it. Similarly, ensure that Auto Scaling group has a valid launch configuration associated with it.
- While testing an application associated with an alarm, you can temporarily disable alarms rather than getting flooded with alerts or unwanted actions being triggered.
- Some AWS resources may not send metric data to CloudWatch in certain conditions. For example, an EBS volume does not send data to CloudWatch if it is not attached to an instance. While trouble shooting any metrics availability issue, ensure that the required conditions are met for monitoring the metrics for the resource.

8

Simple Storage Service, Glacier, and CloudFront

Before we understand what Amazon S3 is, let us understand some basic concepts around storage. Storage services are usually categorized based on how they work and how they are used. Specifically, there are three broad types of storage services: block storage, file storage, and object storage:

- **Block storage**: In simple terms, block storage is a type of storage that is not physically attached to a server, but it is accessed as a local storage device just like a hard disk drive. At the backend, the storage service provider creates a cluster of disks, divided into a number of storage blocks. Each block is virtually connected to a server and treated as a local storage. The server operating system manages the block of storage assigned to it. For example, AWS EBS is a block storage type. When you provision a 100 GB EBS volume, a block of 100 GB is assigned from the cluster of disks to that volume. The EBS volume is associated with an EC2 instance. The volume is subsequently formatted and a filesystem is created on it. This volume is managed by the respective operating system installed on the EC2 instance for storing and retrieving data on it.

  As each block of storage is treated as a local disk, block storage works well for creating filesystems, installing operating systems, and databases. Even though the block storage architecture is comparatively complex, it provides high performance.

- **File storage**: File storage is also known as file-based storage. It is a highly available centralized place for storing your files and folders. You can access file level storage using file level protocols such as **Network File System** (**NFS**), **Server Message Block** (**SMB**), **Common Internet File System** (**CIFS**), and so on. You can use file storage for storing and retrieving files and folders. Just like a block storage, you can edit files stored on file storage. Unlike block storage, you do not have access to format file storage, create a filesystem, and install an operating system on it. It is centrally managed by the service provider.
- **Object storage**: Object storage is a type of storage architecture wherein the data is stored as objects. Each object consists of the data, metadata, and a globally unique identifier. Metadata is data about data, and provides basic information about the data stored in an object. Unlike block storage and file storage, you cannot edit the data stored on object storage. If you need to edit a file, you have to create a new file and delete the old file (or keep multiple versions of the same file).

The following table provides a comparison between block storage, file storage, and object storage:

	Block storage	**File storage**	**Object storage**
Unit of transaction	Blocks	Files	Objects, files with metadata
How you can update	You can directly update the file	You can directly update the file	You cannot update the object directly. You create a new version of the object and replace the existing one or keep multiple versions of the same object
Protocols	SCSI, Fiber Channel, SATA	SMB, CIFS, NFS	REST/SOAP over HTTP/HTTPS
Support for metadata	No metadata support; it stores only filesystem attributes	No metadata support; it stores only filesystem attributes	Supports custom metadata

Usage	For creating filesystems, installing operating systems, and storing transactional data	Used as centralized and shared file storage	Used as a cloud storage for storing static files and data
Strength	High performance	Simple to access, highly available, and easy for sharing files	Scalable, available over the internet and distributed access
Weakness	Restricted to data center capacity	Restricted to data center capacity	Not suitable for in-place editing of data
Can you format it?	Yes, you get complete access to format it and manage filesystems on it	No, you cannot format it. It's a shared service wherein you have access just to manage data on it	No, you cannot format it. It's a shared service wherein you have access just to manage objects on it.
Can you install OS on it?	Yes, block storage can be used as a root volume and you can have an OS on it.	No, you cannot install an OS on it	No, you cannot install an OS on it
Pricing	You are charged based on allocated volume size irrespective of how much data you store on it	You are charged based on the amount of data you store on it	You are charged based on the amount of data you store on it
Example of specific storage service	EBS	Amazon **Elastic File System** (**EFS**)	S3

Amazon S3

S3 is a cloud-based object storage service from Amazon. It is highly scalable and makes it easy to access storage over the internet. You can use S3 for storing and retrieving virtually unlimited amounts of data at any time from anywhere. It provides you with access to a highly scalable, reliable, efficient, and low-cost storage infrastructure that is used by Amazon to run its own global network of websites.

S3 is ideally suggested for storing static content such as graphics files, documents, log files, audio, video, compressed files, and so on. Virtually any type of data in any file format can be stored on S3. Currently, the permissible object size in S3 is 0 bytes to 5 TB. Objects in S3 are stored in a bucket. A bucket is a logical unit in S3 that is just like a folder. Buckets are created at root level in S3 with a globally unique name. You can store objects and also folders inside a bucket. Any number of objects can be stored in each bucket. There is a soft limit of 100 buckets per account in S3.

S3 can be used for content storage and distribution, static website hosting, big data object store, backup and archival, storing application data, as well as for disaster recovery. Using Java Script SDK and DynamoDB, you can also host dynamic applications on S3.

The following section describes the concepts and terminologies used in S3:

- **Buckets**: A bucket is a logical unit in S3, just like a folder. It is a container wherein you can store objects and also folders. Buckets are created at root level in S3 with a globally unique name. Any number of objects can be stored in each bucket. For example, if you store an object named `books/acda.pdf` inside the `packtpub` bucket, then it is accessible using the URL `https://packtpub.s3.amazonaws.com/books/acda.pdf`.
- Buckets are generally used for organizing objects in S3. It is associated with an AWS account that is responsible for storing and retrieving data on the bucket. The account, which owns the bucket, is charged for data transfer. Buckets play a vital role in access control and paves the way for creating usage reports on S3.
- Buckets can be created in a specific region. You can enable version control on a bucket. If version control is enabled on a bucket, it maintains a unique version ID against each object stored in it.
- **Objects**: Objects are the basic entities stored in S3. Each object consists of the data, metadata, and a globally unique identifier. Metadata is data about data, and provides basic information about the data stored in an object. Metadata is stored in a set of name-value pairs, which describes the information associated with the object. For example, `Date Last Modified`, `Content Type`, `Content-Length`, and so on. There can be two types of metadata associated with an object: system-defined metadata and user-defined metadata.
- An object is identified with a unique key (name) within the bucket and a version ID if versioning is enabled on the bucket.

- **Keys**: A key is the name that is assigned to an object. It is a unique identifier or name for an object within a bucket. Every object in a bucket has only one key associated with it. The combination of a bucket, key, and its respective version ID uniquely identifies an object within a bucket. Every object within a bucket has a unique address for accessing it through a web service endpoint. The address URL consists of the bucket name, key, and a version number if versioning is enabled on the bucket.

Example: `https://packtpub.s3.amazonaws.com/books/acda.pdf`. In this example, `packtpub` is the name of the bucket and `books/acda.pdf` is the key as follows:

Figure 8.1: Object URL in S3

- **Region**: A region is a geographical region where Amazon S3 stores a bucket based on user preferences. Users can choose a region while creating a bucket based on the requirement. Ideally, a bucket should be created in the closest geographical region where the bucket is needed to be accessed. Choosing a closest region while creating a bucket optimizes latency while accessing the bucket, reduces costs, and complies with any regulatory requirements an organization may have.

Currently, Amazon has the following regions:

Region	**Location of S3 servers**
US East (N. Virginia)	Northern Virginia
US East (Ohio)	Columbus Ohio
US West (N. California)	Northern California
US West (Oregon)	Oregon
Canada (Central)	Canada
Asia Pacific (Mumbai)	Mumbai
Asia Pacific (Seoul)	Seoul
Asia Pacific (Singapore)	Singapore

Asia Pacific (Sydney)	Sydney
Asia Pacific (Tokyo)	Tokyo
EU (Frankfurt)	Frankfurt
EU (Ireland)	Ireland
EU (London)	London
South America (Sao Paulo)	Sao Paulo

S3 supported regions

When you create an object in a region, it is stored in the same region unless you explicitly copy it over to any other region:

- **S3 data consistency model**: Amazon provides two types of consistency model for S3 data when you perform various input/output operations with it: read-after-write consistency and eventual consistency.

The following table describes both of these data consistency models in S3:

Input/output operation	**Data consistency model**	**Exception, if any**
`PUTS` of new object	read-after-write consistency	S3 gives eventual consistency for `HEAD` or `GET` requests while retrieving the key name before creating an object
Overwrite `PUTS` and `DELETES`	Eventual consistency	No exception

Data consistency model

Amazon provides read-after-write consistency for `PUTS` of a new object. That means, if you create a new object in S3, you can immediately read it.

Amazon provides eventual consistency for overwrite `PUTS` and `DELETES` operations. That means, it takes a few seconds before the changes are reflected in S3 when you overwrite an existing object in S3 or delete an existing object.

Amazon replicates data across the region in multiple servers located inside Amazon data centers. This replication process provides high availability for data. When you create a new object in S3, the data is saved in S3, however, this change must replicate across the Amazon S3 regions. Replication may take some time and you may observe the following behavior:

- After you create a new object in S3, Amazon immediately lists the object keys within the bucket and the new object keys may not appear in the list
- When you replace an existing object and immediately try to read the object, Amazon may return old data until the data is fully replicated
- When you delete an object and immediately try to ready it, Amazon may read the data for the deleted object until the deletion is fully replicated
- When you delete an object in a bucket and immediately list the contents of the bucket, you may still see the deleted object in the content of the bucket until the deletion is fully replicated

Creating a bucket

The following steps describe the process of creating a bucket using the AWS Management Console:

1. Sign-in to your AWS account and go to the S3 console or visit `https://console.aws.amazon.com/s3/`. If you already have any buckets in the account, it displays a list of the buckets or the following *Figure 8.2* stating that you do not have any buckets in the account:

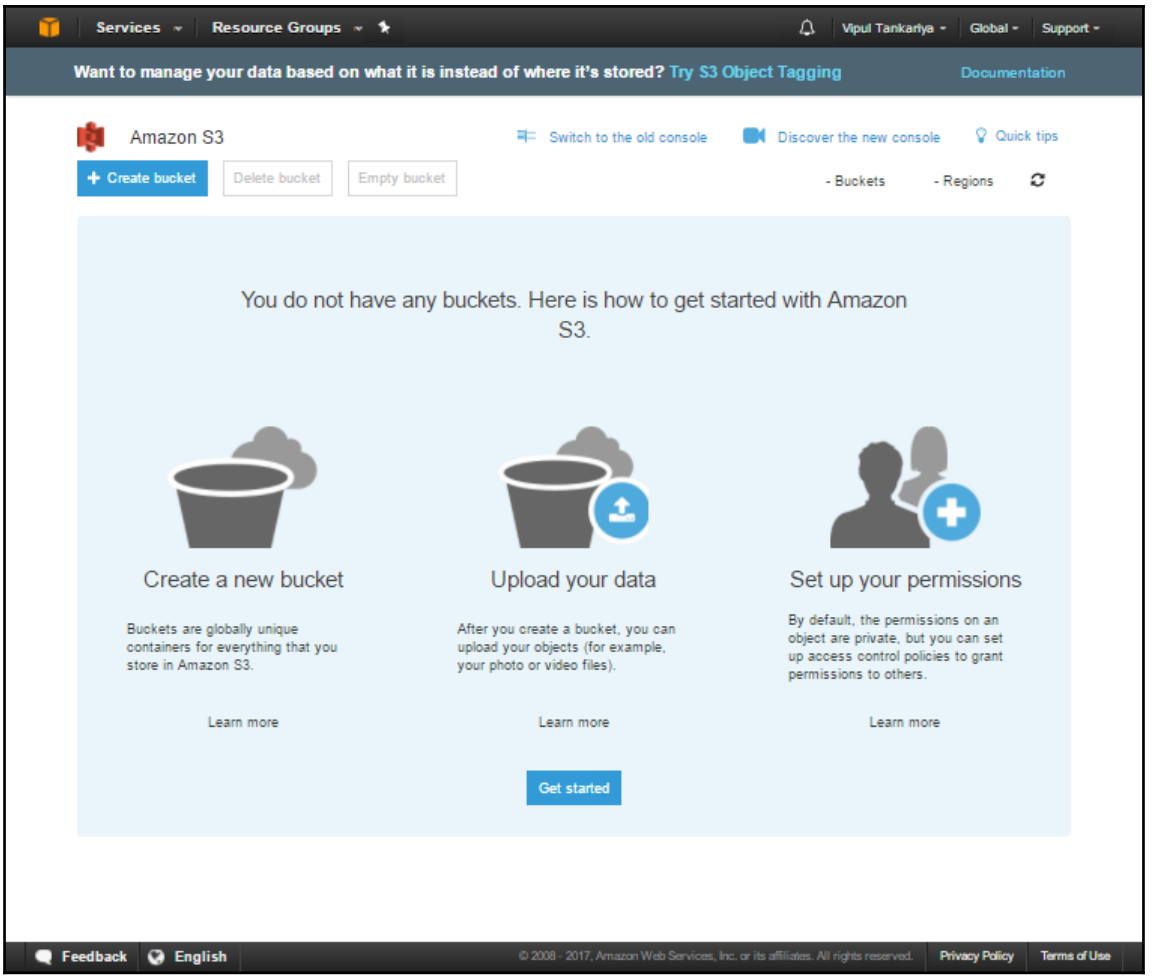


Figure 8.2: S3 console

2. Click on the **Create bucket** icon, as displayed in the following *Figure 8.3*:

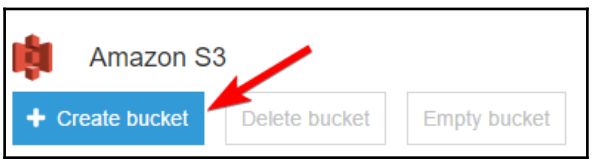


Figure 8.3: Create bucket

3. Clicking on the **Create bucket** button, display a pop-up as shown in the *Figure 8.4*. Enter a DNS compliant bucket name. **Bucket name** must be unique across all existing bucket names in S3. Since S3 is a shared service, it is likely that you may not always get the bucket name you want as it might have been already taken by some one.

   Select the appropriate region where you want to create the bucket from the drop-down menu as indicated in the following *Figure 8.4*. If you already have some buckets, you can **Copy setting from an existing bucket**. You can also click on the **Create** button if you do not want to follow the remaining steps. You need to set bucket properties and permissions later on, if you directly click on the **Create** button. To understand these steps, you can click on the **Next** button:

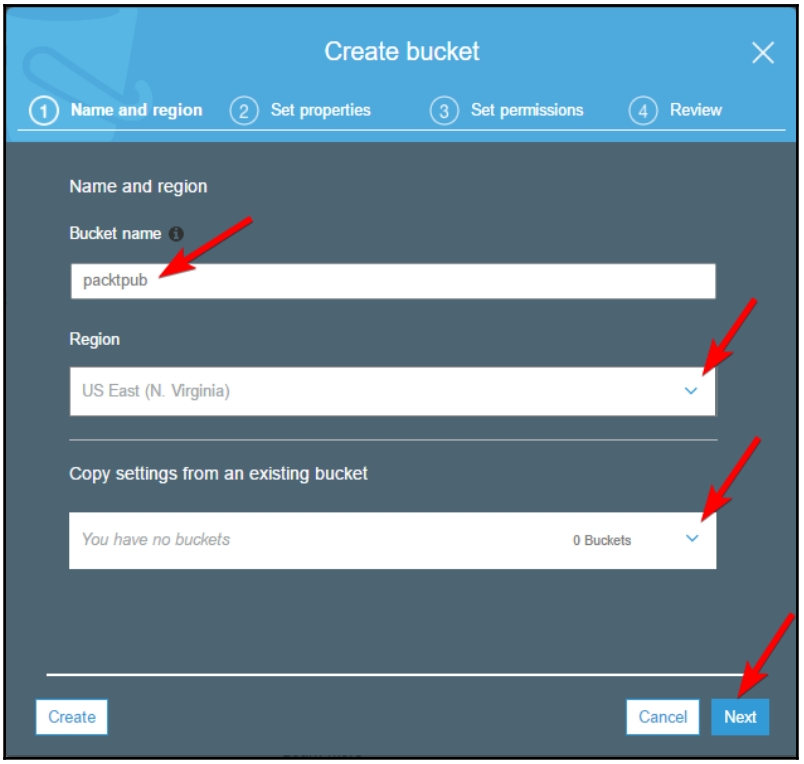


Figure 8.4: Create bucket screen

4. In the subsequent screen, as shown in the following *Figure 8.5*, you can set the required properties. You can see in the screen that by default **Versioning** is **Disabled**, **Logging** is **Disabled**, and there are no **Tags**. You can click on **Versioning** and **Logging** as required or add tags as needed. When you click on these items, it displays respective pop-ups as shown in the following *Figure 8.5*. You can set the required properties as needed:

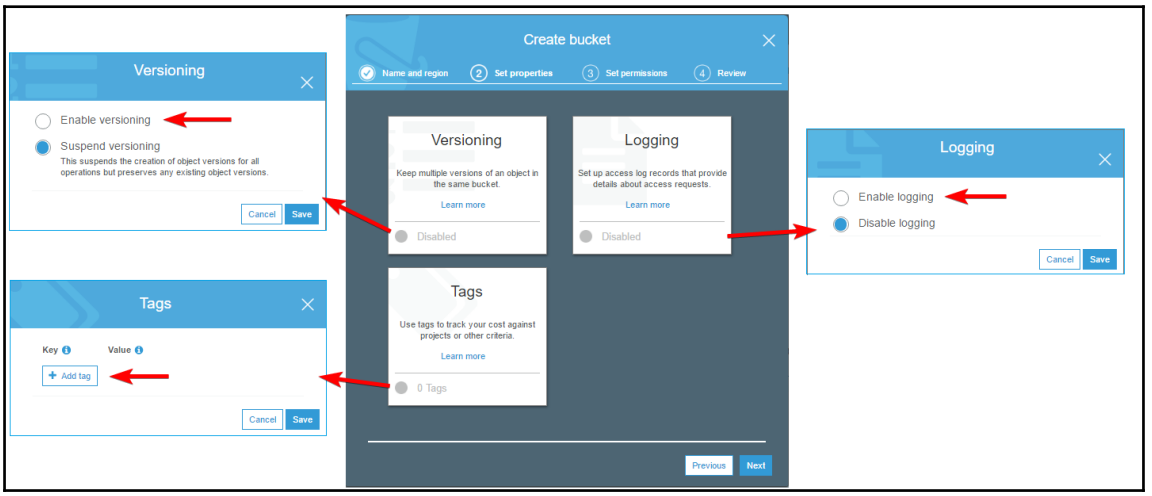


Figure 8.5: Bucket properties in Create bucket wizard

5. In the subsequent screen, as shown in the following *Figure 8.6*, you can set folder permissions. You can set individual user permissions, manage public permissions, and manage system permissions:

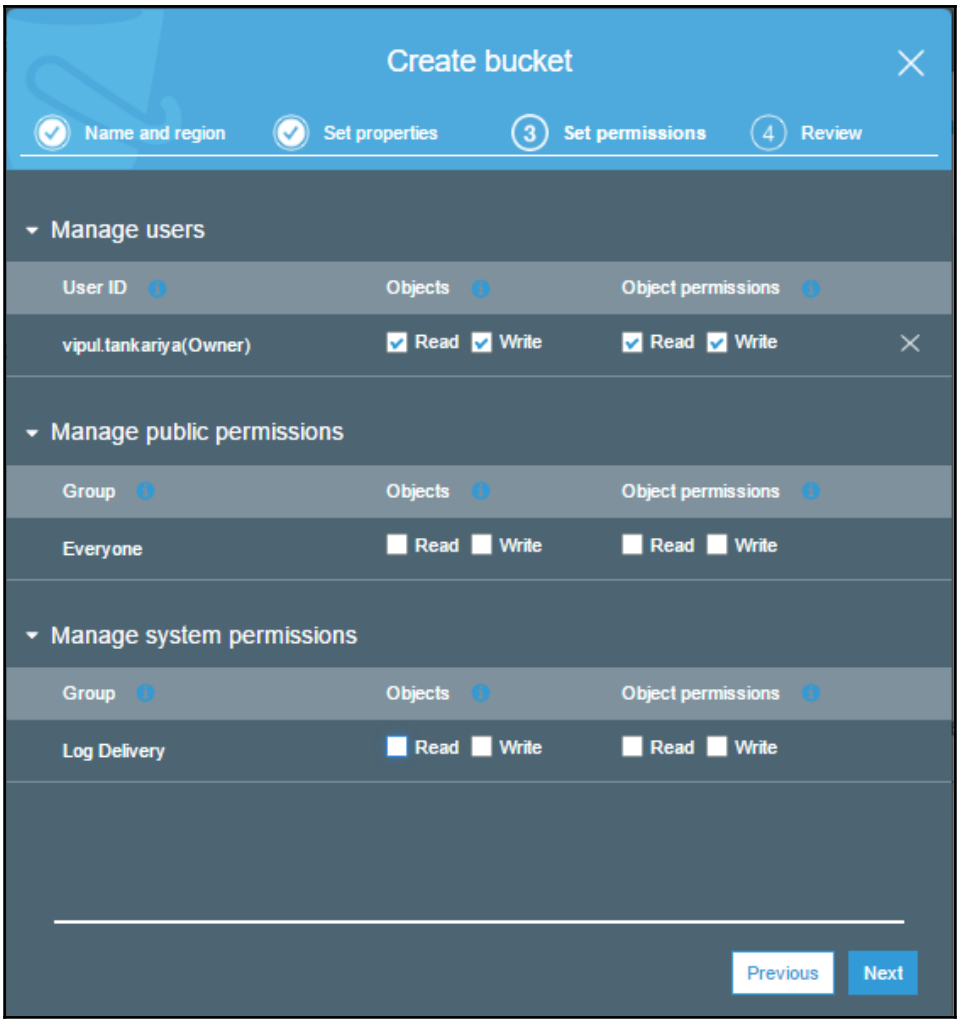


Figure 8.6: Manage bucket permission in Create bucket wizard

6. In the subsequent screen, as shown in the following *Figure 8.7*, review your selection. If required, you can edit your selection under individual categories. After reviewing everything, click on the **Create bucket** button. It creates a bucket as per the input given by you:

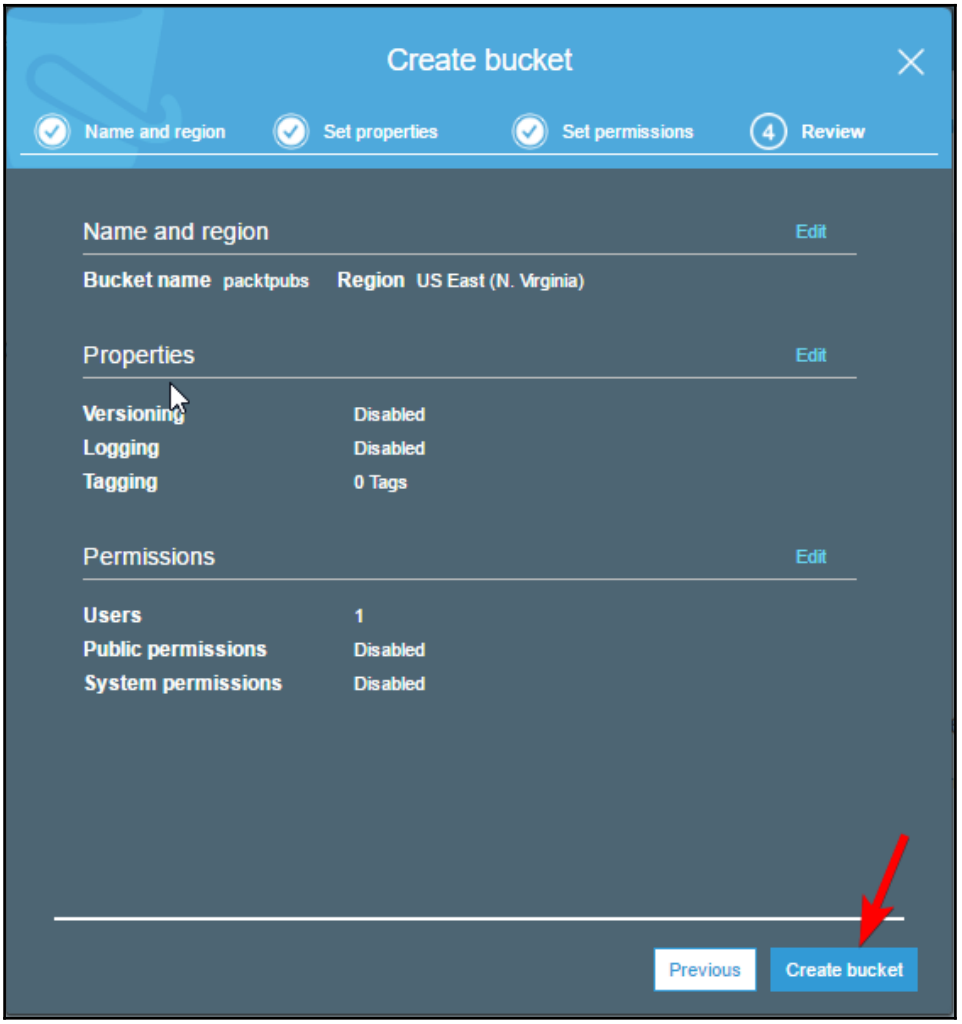


Figure 8.7: Review your steps in the Create bucket wizard

Bucket restriction and limitations

Bucket restrictions and limitations are listed as follows:

- You can create a bucket using the S3 console, APIs, or the CLI.
- Amazon imposes a soft limit of 100 buckets on an AWS account. You can increase this soft limit by raising a support request with Amazon.
- When you create a bucket, it is associated with the AWS account and the user you have used to create it. Bucket ownership cannot be transferred to another AWS account or another user within the same AWS account.
- There is no limit on the number of objects that can be created in a bucket.
- A bucket is created at the root level in S3; you cannot create a bucket inside a bucket.
- If you use an application that automatically creates a bucket, ensure that the application chooses a different bucket name in case the bucket name generated by the application already exists.

Bucket names should comply with DNS naming conventions as follows:

- Bucket names can range between 3 to 63 characters
- Bucket names must be in lowercase letters. They can contain number and hyphens
- Bucket names must start with a lowercase letter or a number and similarly, must end with a lowercase letter or a number
- Bucket names must not be given as an IP address, that is, `192.168.1.10`
- It is recommended to avoid using periods `(.)` in bucket names

Bucket access control

Each bucket in S3 is associated with an access control policy, which governs how objects are created, deleted, and enumerated within the bucket.

When you create an S3 resource, all S3 resources, including buckets, objects, lifecycle policy, or static website configuration, are by default private. Only the resource owner who creates the resource, can access that resource. After creating the resource, the resource owner can optionally grant permissions to other users using an access control policy.

There are two types of access control policy:

- Resource-based policies
- User policies

Access policies that you associate with buckets and objects are called resource-based policies. Bucket policies and **Access Control Lists** (**ACL**) are examples of resource-based policies. Access policies that you associate with users are called user policies. You can use a resource-based policy or user policy and at times a combination of both, to control access to your S3 resources.

Bucket policy

A bucket policy, generally, comprises of the following elements:

- **Resource**: This indicates Amazon S3 resources such as buckets and objects. While creating a policy, you can specify ARN to allow or deny permissions on that resource.
- **Action**: This indicates one or more actions that are either allowed or denied. For example, `s3:GetObject` specifies the permission to read the object data. Similarly, `s3:ListBucket` specifies the permission to list objects in the bucket.
- **Effect**: This specifies action type, either `Allow` or `Deny` access. If permission is not explicitly granted on a resource, by default, access is denied. When you explicitly `Deny` access, it ensures that the user cannot access the resource even if another policy grants access.
- **Principal**: This indicates the account or a user who is allowed or denied access to the resources mentioned in the policy statement. In a user policy, you may not need to specify a principal. A user policy implicitly applies to the associated user.
- **Sid**: This is an optional identifier known as **statement ID**, which is specified for the policy statement. Sid values must be unique within the policy statement.
- Here is an example of a bucket policy. The example policy allows the user `Heramb` following three permissions on the bucket named `packtpubs`:
    - `s3:GetBucketLocation`
    - `s3:ListBucket`
    - `s3:GetObject`

In the policy statement, `Account-ID` should be replaced with the AWS account number:

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Sid": "Statement1",
         "Effect": "Allow",
         "Principal": {
            "AWS": "arn:aws:iam::Account-ID:user/Heramb"
         },
         "Action": [
            "s3:GetBucketLocation",
            "s3:ListBucket",
            "s3:GetObject"
         ],
         "Resource": [
            "arn:aws:s3:::packtpubs"
         ]
      }
   ]
}
```

In the same policy, if you change the effect from `Allow` to `Deny`, it explicitly denies access to the user `Heramb` on the `packtpubs` bucket to perform the specific set of actions mentioned in the policy statement.

User policies

Access policies are associated with users or groups. Unlike a bucket policy, you don't need to specify `Principal` in a user policy. A policy is implicitly applied to the user with whom it is associated.

Example of user policy is as follows:

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "s3:ListAllMyBuckets"
         ],
         "Resource":"arn:aws:s3:::*"
      },
      {
```

```
            "Effect":"Allow",
            "Action":[
               "s3:ListBucket",
               "s3:GetBucketLocation"
            ],
            "Resource":"arn:aws:s3:::packtpubs"
         },
         {
            "Effect":"Allow",
            "Action":[
               "s3:PutObject",
               "s3:GetObject",
               "s3:DeleteObject"
            ],
            "Resource":"arn:aws:s3:::packtpubs/*"
         }
      ]
   }
```

There are three parts to the preceding user policy example:

- The first part describes permission to list all the buckets using a `ListAllMyBuckets` action against `arn:aws:s3:::*`, which signifies all resources in S3 for the account
- The second part describes `ListBucket` and `GetBucketLocation` permissions on the `packtpubs` bucket
- The third part describes permissions to create, read, and delete objects in the `packtpubs` bucket

Once a user policy is created, it can be attached to a user or a group to grant them respective access specified in the policy.

Transfer Acceleration

When you need to transfer a very big amount of data between your on-premises environment and S3, time, efficiency, and the security of the data plays a very vital role. In such requirements, S3 Transfer Acceleration can be very handy. It provides a fast, easy, and secure way to transfer files between S3 and any other source or target of such data transfers. For Transfer Acceleration, Amazon uses CloudFront edge locations. CloudFront edge locations are spread across the Globe, which facilitates the Transfer acceleration process.

The scenarios in which you should use Transfer Acceleration are:

- You have a centralized bucket, which your end customers use from across the Globe for uploading data
- You regularly transfer GBs and TBs of data across continents
- If available bandwidth is underutilized while you transfer data to S3

Enabling Transfer Acceleration

The steps for enabling Transfer Acceleration are as follows:

1. Log in to the AWS Management Console and go to the S3 Console or browse to `https://console.aws.amazon.com/s3`.
2. Open the bucket on which you need to enable Transfer Acceleration.
3. Click on the **Properties** tab as shown in the following *Figure 8.8*:

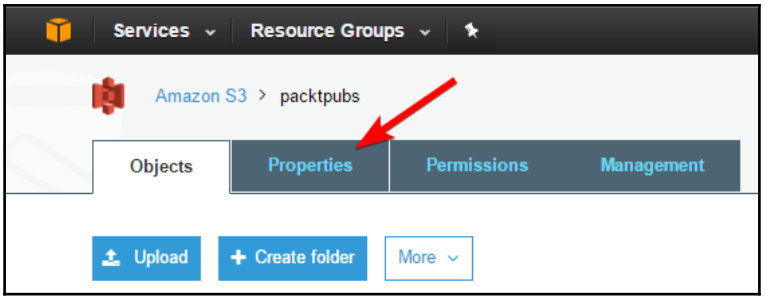


Figure 8.8:Selecting Bucket Properties

4. In the **Properties** tab, click on **Transfer acceleration**. It brings a pop-up to enable or suspend Transfer acceleration as shown in the following *Figure 8.9*. You can select **Enabled** in this pop-up to enable the Transfer Acceleration on the selected bucket. Click on the `Save` button after the selection is done:

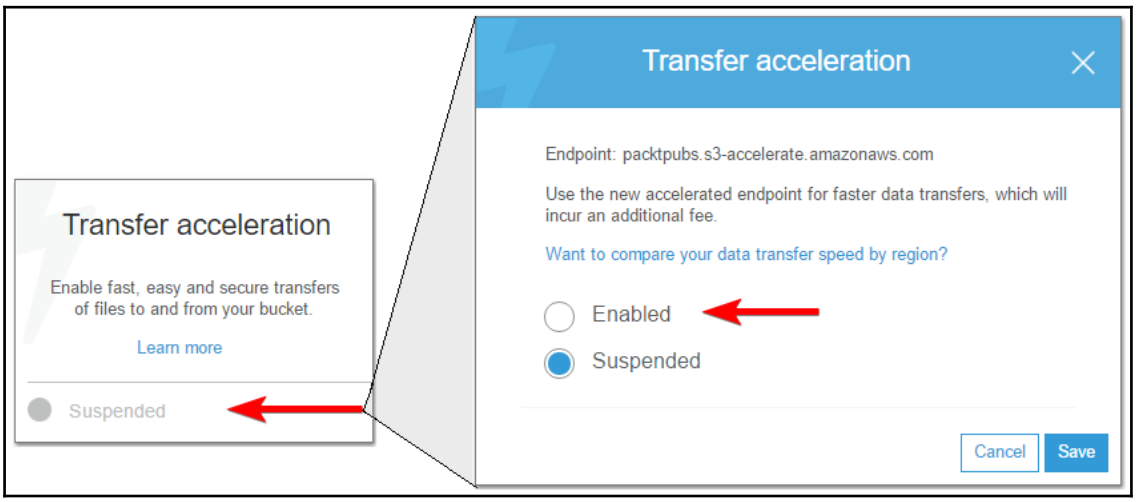


Figure 8.9: Enable Transfer acceleration on a bucket

Requester Pay model

Generally, when you create a bucket in S3, you pay for data storage and data transfer. Based on your usage, charges are added to the AWS account associated with the bucket. Amazon provides an option wherein you can configure your bucket as **Requester Pays** bucket. When you configure a bucket as a Requester Pays bucket, the requester pays for the requests they initiate to download or upload data in the bucket. You just pay for the cost of the data you store in S3:

- You can enable Requester Pays on a bucket when you want to share the data, but do not want to get charged for the requests received, data downloads, or upload operations
- When you enable Requester Pays, AWS does not allow you to enable anonymous access on the bucket
- All requests to Requester Pays buckets must be authenticated. When you enable authentication, S3 can identify requesters and charge them for their respective usage of the bucket

- If a system or application makes requests by assuming an IAM role, AWS charges the account where the assumed role belongs
- If you make calls to the bucket using an application, the request must include `x-amz-request-payer` in the header section if you make `POST`, `GET`, and `HEAD` requests
- If you make a `REST` request, you need to include `x-amz-request-payer` as a parameter in the request
- Requester Pays buckets do not support anonymous request, BitTorrent, and `SOAP` requests
- Amazon does not allow you to enable end user logging on a Requester Pays bucket and similarly, you cannot enable Requester Pays on a bucket where end user logging is enabled

Enabling Requestor Pays on a bucket

You can enable Requestor Pays a bucket in steps that are very similiar to those you followed in Transfer Acceleration:

1. Log in to the AWS Management Console and go to the S3 console or browse to `https://console.aws.amazon.com/s3`.
2. Open the bucket on which you need to enable Transfer Acceleration.
3. Click on the **Properties** tab as shown in the *Figure 8.8*.
4. Click on **Requester Pays** to enable it.

Understanding objects

Objects are the basic entities stored in S3. Amazon has designed S3 as a simple key, value store. You can store a virtually unlimited number of objects in S3. You can segregate objects by storing them in one or more buckets.

Objects consist of a number of elements, that is, key, version ID, value, metadata, subresources, and access control information. Let us understand these object elements:

- **Key**: Key is the name that is assigned to an object. It's just like a filename and can be used to access or retrieve the object.

- **Version ID**: If you enable versioning on a bucket, S3 associates a version ID with each object. The bucket may have one or more objects with the same key, but a different version ID. The version ID helps in uniquely identifying an object when there are multiple objects with the same key.
- **Value**: Value refers to the content or data that is stored on the object. It is generally a sequence of bytes. the minimum size of an object can be zero and the maximum 5 TB.
- **Metadata**: S3 stores reference information related to an object in its metadata in the form of name-value pairs. There are two types of metadata, that is, system-metadata and user-defined metadata. System-metadata is used for managing objects and user-defined metadata is used for managing information related to objects.
- **Subresources**: An object can have subresources associated with it. Subresources are defined and associated with objects by S3. There can be two types of subresources associated with an object, that is, ACL and torrent:
    - ACL contains a list of users and respective permissions granted to them. When you create an object, the ACL entry contains just an owner. Optionally, you can add more users with required permissions for each user.
    - Torrent is another subresource of an object. AWS supports the BitTorrent protocol. It is very simple to access S3 objects using a BitTorrent client. If you assign anonymous permission on an object, that object can be accessed by a BitTorrent client by referring to the object URL with `?torrent` at the end. Once an object URL is accessed with `?torrent` at the end of it, AWS automatically creates a `.torrent` file for that object. Subsequently, you can distribute the `.torrent` file to end users to access the object using BitTorrent client.
- **Access control information**: Amazon S3 enables you to control access on the objects you create using ACL, bucket policies, and user policies. Access control information is nothing but the information containing permissions in the form of ACL, a bucket policy, or user access policies.

Object keys

When you create an object in S3, you need to give a unique key name to the object in the bucket. A key name uniquely identifies an object in the bucket. When you browse a bucket in the S3 console, it displays a list of objects within the bucket. The list of names within the bucket are object keys.

An object key is a sequence of Unicode characters in UTF-8 encoding. A key name can be a maximum of 1024 bytes long.

Object key naming guide

Each application applies its own mechanism to parse special characters. It is recommended to follow best practices while naming an object key. These best practices provide maximum compliance with DNS, web safe characters, XML parsers, and various other APIs:

- An object key name consists of alphanumeric characters [0-9a-zA-Z] and special characters such as !, -, _, ., *, ', (,).
- S3 can store buckets and objects. It does not have any hierarchical structure; however, prefixes and delimiters used in an object key name allows S3 to use folders.
- Key name examples of how S3 supports folders:
    - `projects/acda-guide.xlsx`
    - `books/aws-networking.pdf`
    - `outlines/vpc.xlsx`
    - `help.txt`
- In the previously mentioned examples, S3 uses key name prefixes such as `projects/`, `books/`, `outlines/`. These key name prefixes with `/` as delimiter, enable S3 to represent a folder structure. The following *Figure 8.10* shows the folder structure in S3:

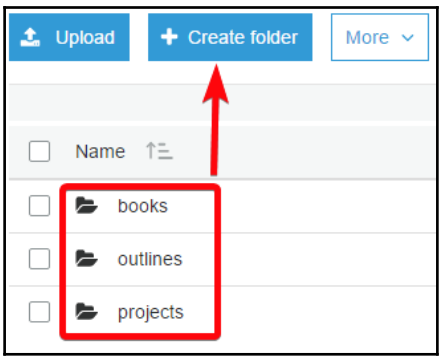


Figure 8.10: Folder structure in S3

When you open a folder, it displays objects inside the folder. The S3 console displays the bucket and folder in the breadcrumb as shown in the following *Figure 8.11*:

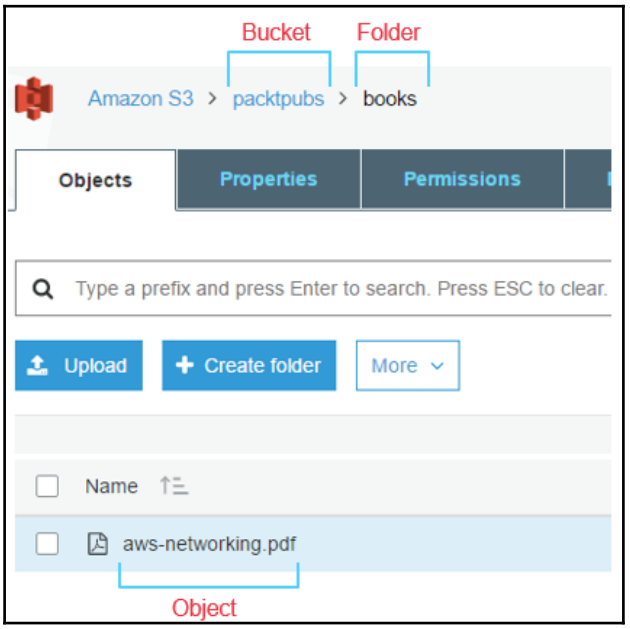


Figure 8.11: Objects inside a folder

Special characters in an object key name

The following is a list of special characters that require special handling if you use them in an object key name. Some characters may not be properly rendered by a browser or application. If you plan to include these characters in S3 object key names, it is recommended to build appropriate logic to handle them in your application.

Ampersand (`&`)	Dollar (`$`)	ASCII character ranges 00-1F hex (0-31 decimal) and 7F (127 decimal)
At (`@`)	Equals (`=`)	Semicolon (`;`)
Colon (`:`)	Plus (`+`)	Space - significant sequences of spaces may be lost in some uses (especially multiple spaces)
Comma (`,`)	Question mark (`?`)	

AWS recommends avoiding following characters in object key names.

Backslash (\)	Left curly brace ({)	Non-printable ASCII characters (128-255 decimal characters)
Caret (^)	Right curly brace (})	Percent character (%)
Grave accent / back tick (`)	Right square bracket (])	Quotation marks
Greater than symbol (>)	Left square bracket ([)	Tilde (~)
Less than symbol (<)	Pound character (#)	Vertical bar / pipe (|)

Object metadata

S3 stores reference information related to an object in its metadata in the form of name-value pairs. There are two types of metadata: system-metadata and user-defined metadata.

System-metadata

Amazon stores a set of system-defined metadata with every object in S3. For example, S3 stores the object creation date as well as size of the object in object metadata. There are two types of system-metadata, one wherein only S3 can change the value of metadata such as object creation date and size. There are other types of system-metadata such as storage class and server-side encryption that can be controlled by users based on selection.

The following table displays a list of system-defined metadata:

Name	**Description**	**Can user modify the value?**
`Content-Length`	Indicates object size in bytes	No
`Content-MD5`	Indicates a base64-encoded 128-bit MD5 digest of the object	No
`Date`	Indicates current date and time	No
`Last-Modified`	Indicates date of last modification on object. It can be the creation date if object is not modified after initial creation	No

`x-amz-delete-marker`	It is displayed against objects in a bucket where versioning is enabled. It indicates if an object is marked for deletion	No
`x-amz-server-side-encryption`	This metadata indicates if server-side encryption is enabled on an object or not.	Yes
`x-amz-server-side-encryption-aws-kms-key-id`	When `x-amz-server-side-encryption` is enabled on an object and it includes `aws:kms`, it indicates the ID of the KMS master encryption key that is used for the object	Yes
`x-amz-server-side-encryption-customer-algorithm`	It indicates if server-side encryption is enabled with customer-provided encryption keys (SSE-C)	Yes
`x-amz-storage-class`	It indicates what storage class is used for storing the object	Yes
`x-amz-version-id`	When versioning is enabled on a bucket, this metadata indicates version of the object	No
`x-amz-website-redirect-location`	When website hosting is enabled on an S3 bucket, this metadata indicates the redirection URL if request redirection is configured	Yes

User-defined metadata

Amazon S3 allows users to assign user-defined metadata to an object. When you create an object in S3, you can provide optional metadata as name-value pair.

User-defined metadata is generally used for associating additional information with an object. Such metadata can help in identifying objects. It can also be used for automating data management tasks using scripts. For example, a script may traverse through all the objects in a bucket and check for specific metadata on an object. If a desired key value pair of a metadata is assigned to an object, the script may further process the data in the object. User-defined metadata must begin with `x-amz-meta-`.

Here is how you can assign metadata to an object using the S3 console:

1. Log in to the AWS console and go to the S3 console.
2. Open the required bucket.
3. Click on the object on which you want to define metadata.
4. Click on the **Properties** tab.
5. Click on **Metadata**.
6. Click on **Add Metadata**.
7. Select **x-amz-meta-book-type** from the drop-down and type the remaining value in the **Key** as well as the **Value** box as shown in the following *Figure 8.12*.
8. Click on **Save:**

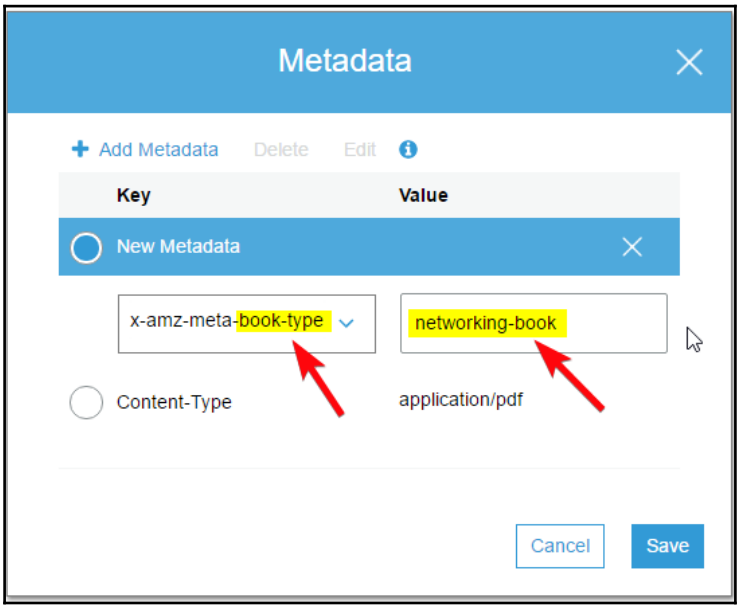


Figure 8.12: Add metadata to an S3 Object

Versioning

S3 allows you to keep multiple versions of an object in a bucket. Versioning can be enabled at bucket level. Once versioning is enabled, it protects you from accidental updates and deletes on an object. When you overwrite or delete an object, it keeps multiple copies with version numbers.

For example, when you enable versioning on a bucket called `packtpub`, for each action on an existing object in the bucket, S3 creates a new version and associates a version ID with it, as shown in the following table:

Object	Last activity	Version ID
`developer-guide.pdf`	Jun 12, 2017 9:42:02 AM	VAgAtLChtLoMkKF4ZVoq.NAGRRBA1hSp
`developer-guide.pdf`	Jun 11, 2017 8:41:23 AM	3mWAzx.l25VRt3.V.1ExutyOAEG1npX3
`developer-guide.pdf`	Jun 10, 2017 5:39:58 PM	hV_2iz3GgRvOTt1NoiL8KXg3FpLJkFI7

When you delete an object in a version-enabled bucket, S3 does not actually delete the object but instead adds a delete marker to it.

Enabling versioning on a bucket

The steps for enabling versioning on a bucket are as follows:

1. Sign in to your AWS Management Console and go to the S3 console on `https://console.aws.amazon.com/s3/`.
2. Click on the bucket on which you want to enable versioning, as shown in the following *Figure 8.13*:

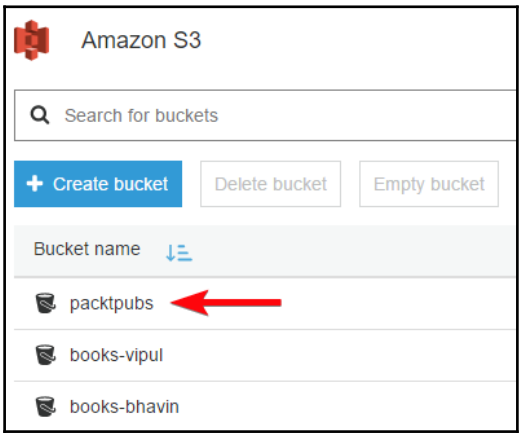


Figure 8.13: Select bucket to enable versioning

3. Click on the **Properties** tab:

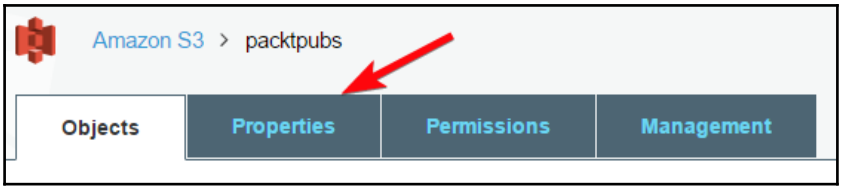


Figure 8.14: Select bucket properties

4. Click on **Versioning**, **Enable versioning**, and save the changes:

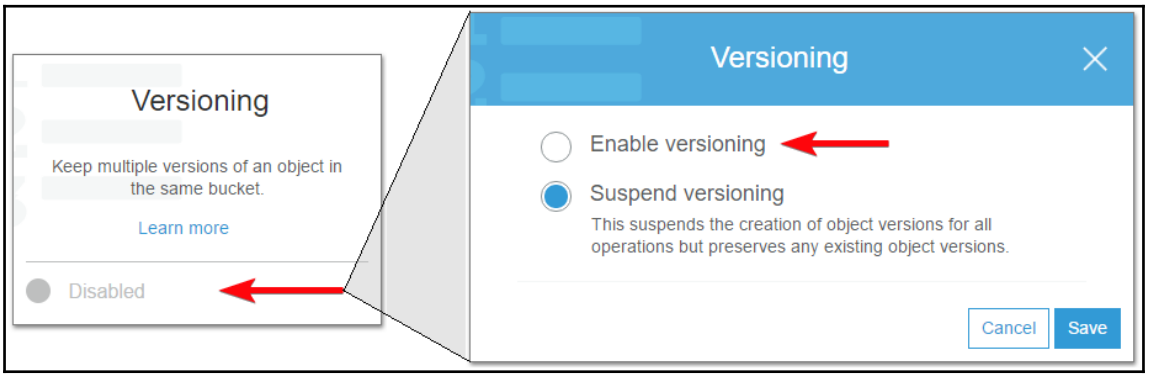


Figure 8.15: Enabling Versioning

Object tagging

S3 allows you to add tags to your objects. Tagging an object helps in categorizing the objects. Each tag is a key and value pair.

Example of tags on an object: Let's consider a scenario wherein an application processes data stored in an S3 bucket. While traversing through the objects in a bucket, it checks for a tag before processing the data in the object. In such scenarios, you may add the following tag to the objects:

```
Processed=True
```

Or:

```
Processed=False
```

The application may check for the tag in an object before processing the data in it. If the tag indicates `Processed=False` then the application should process the data stored in the object and change the tag to `Processed=True`.

You can add tags to an object from object properties in the S3 console. You can also add tags to an object using the AWS CLI as follows:

AWS CLI syntax for adding tags to an object:

```
aws s3api put-bucket-tagging --bucket <Bucket> --tagging
'TagSet=[{Key=<key>,Value=<value>}]'
```

Example:

```
aws s3api put-bucket-tagging --bucket packtpubs --tagging
'TagSet=[{Key=Processed,Value=True}]'
```

S3 storage classes

Amazon S3 provides a number of storage classes for different storage needs. Storage classes are divided into four main types, based on how they are used.

Storage classes include:

- S3 Standard storage
- S3-**Infrequently Accessed** (**IA**) storage
- S3 **Reduced Redundancy Storage** (**RRS**)
- Glacier

S3 Standard storage

S3 Standard storage is used as general-purpose storage for frequently accessed data. It provides high availability, durability, and high-performance storage for frequently accessed data. S3 Standard storage can be used in content distribution, cloud applications, big data analytics, mobile or gaming applications, and dynamic websites. The key features of S3 Standard storage are listed as follows:

- Provides low-latency and high-throughput performance
- Ensures 99.999999999% durability for objects

- Provides 99.99% availability in a year backed by Amazon S3 **Service Level Agreement** (**SLA**)
- Enables SSL encryption of data in transit using SSL
- Supports AES-256 encryption of data at rest
- Supports data lifecycle management for automatically migrating data from one class of storage to another

S3-IA storage

S3-IA storage is meant for data that is less frequently used, but needs to be available immediately when needed. It provides low-latency, high throughput, and durable data storage. It incurs relatively low per GB storage and retrieval costs. Being a low-cost and high performance storage, S3-IA is best suited for backups, disaster recovery, and any long-term storage needs. You can keep S3-IA class objects within the same bucket with other class objects. It also supports object lifecycle policies for automatically transitioning objects to other storage classes without requiring any modification of applications using objects. The key features of S3-IA are as follows:

- Suitable for long-term data storage, backups, and disaster recovery
- Provides low-latency and high-throughput performance same as S3 Standard
- Ensures 99.999999999% durability for objects
- Provides 99.99% availability in a year backed by Amazon S3 SLA
- Enables SSL encryption of data in transit using SSL
- Supports AES-256 encryption of data at rest
- Supports data lifecycle management for automatically migrating data from one class of storage to another

S3 RRS

As the name suggests, S3 RRS provides reduced levels of redundancy as opposed to standard S3 storage. It is suitable for storing noncritical and reproducible data. It is a highly available storage solution for content distribution as a secondary storage for data that is available elsewhere as well. It is ideal for storing thumbnails, transcoded media, or any other processed data that can be reproduced.

S3 stores RRS objects across multiple facilities and provides 400 times durability than a local disk drive; however, RRS objects are replicated relatively infrequently compared to S3 Standard objects:

- Provides reduced level of redundancy
- Comparatively cheaper than S3 Standard storage
- It is backed by Amazon S3 SLA
- Provides 99.99% durability in a given year
- Provides 99.99% availability in a given year
- Architected for absorbing data loss in a single facility
- Enables SSL encryption of data in transit using SSL
- Supports AES-256 encryption of data at rest

Glacier

Glacier is very low-cost, secure, and durable data archival storage. You can virtually store unlimited amounts of long-term data on Glacier for much cheaper rate. Glacier is ideal for storing long term data, backups, archives, and data for disaster recovery. Unlike S3, data stored on Glacier is not immediately available for access. You need to initiate a data retrieval request for accessing data on Glacier. For keeping the costs low and still making it suitable for different retrieval requirements, Glacier provides the following three options for data retrieval:

Data retrieval option	Minimum time for retrieval	Comparative costs
Expedited retrieval	1 to 5 minutes	$$$
Standard retrieval	3 to 5 hours	$$
Bulk retrieval	5 to 12 hours	$

Comparison of S3 storage classes and Glacier

The following table compares the three storage classes of S3 with glacier:

Description	**Standard**	**Standard-IA**	**Reduced Redundancy**	**Glacier**
Availability SLA	99.9%	99%	N/A	N/A
Concurrent facility fault tolerance	2	2	1	N/A
Availability	99.99%	99.9%	99.99%	N/A
Durability	99.999999999%	99.999999999%	99.99%	99.999999999%
First byte latency	milliseconds	milliseconds	milliseconds	Select minutes or hours
Lifecycle transitions	yes	yes	yes	yes
Minimum object size	N/A	128KB	N/A	N/A
Maximum object size	5TB	5TB	5TB	40TB
Minimum storage duration	N/A	30 days	N/A	90 days
Retrieval fee	N/A	per GB retrieved	N/A	per GB retrieved
SSL support	yes	yes	yes	yes
Storage class	object level	object level	object level	object level
Supported encryption at rest	AES-256	AES-256	AES-256	AES-256
Data retrieval time	immediately	immediately	immediately	minimum 3 to 5 hours
Recommended multipart upload size	100 mb	100 mb	100 mb	100 mb

Lifecycle management

Lifecycle management is a mechanism in S3 that enables you to either automatically transition an object from one storage class to another storage class or automatically delete an object, based on configuration. Lifecycle rules can be applied to a group of objects based on filter criteria set in the rule.

S3 allows you to configure one or more lifecycle rules, wherein each rule defines a specific action. There are two types of action you can define in Lifecycle rules:

- **Transition actions**: This defines when an object storage class changes from an existing storage class to target storage class. For example, you can define a rule for all object keys starting with `data/` in a bucket to transition from Standard storage to STANDARD_IA after 15 days. Similarly, you can define a rule to transition for all objects keys starting with `data/` from STANDARD_IA to Glacier storage. Let's say, you have a bucket named `packtpubs` and inside the bucket you have a folder named `data`. Within the data folder you have `.csv` files. In such scenarios, this transition rule applies to all the files present in the data folder.
- **Expiration actions**: This defines when objects expire. When objects expire, Amazon S3 automatically deletes them for you. For example, you can set a rule for object keys starting with `backup/` in a bucket to expire after 30 days. In such scenarios, all the files from the backup folder in a specific bucket expire after 30 days and are automatically deleted from S3.

Lifecycle configuration use cases

It is advisable to configure lifecycle rules on objects wherein there is absolute clarity on the lifecycle of the objects. The following are example scenarios wherein you can consider defining lifecycle rules:

- You have an application that generates and upload logs to an S3 bucket. The application does not need these logs after a week or a month. In such scenarios, you may want to delete these logs.
- You have a bucket wherein users and applications are uploading objects. These objects are frequently accessed for a few days. After a few days of uploading, these objects are accessed less frequently.

- You are archiving data to S3 and you need to keep this data only for regulatory compliance purposes. You need this data in an archive for a specific period of time to cater for regulatory needs and subsequently this data can be deleted.
- You are taking a back up of your databases on S3 and your organization has a predefined retention policy for this data. Based on the retention policy, you may want to keep the backup for a specific period and then delete it.

Defining lifecycle policy for a bucket

Object lifecycle rules can be configured using the Amazon S3 console, using the AWS SDK, or using the REST API. The following list describes the steps for configuring lifecycle rules using the Amazon S3 console:

1. Sign in to your AWS account and go to the S3 console on `https://console.aws.amazon.com/s3`.
2. Click on the bucket for which you want to create the lifecycle policy.
3. Click on the **Management** tab and then click on **+ Add lifecycle rule** as in following *Figure 8.16*:

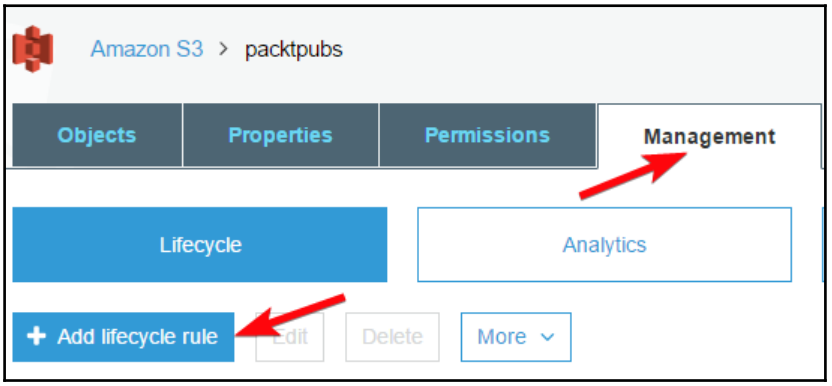


Figure 8.16: Adding lifecycle rule

4. In the subsequent window enter the name for your rule as shown in the following *Figure 8.17*. The rule name must be unique in the bucket. You cannot create more than one rule with the same name in a bucket.

5. Specify filter criteria for filtering the objects in a bucket. The filter criteria can be a string expression, for example `backup/`. You can also specify one or more object tags and limit the scope of the rule accordingly, for example, `backup/ | processed`. You can select a prefix and tags while entering the filter value as shown in the following *Figure 8.17*. You can initially enter prefix value `backup/` and then click on tag as shown in *Figure 8.17* for entering tag value. S3 user pipe (`|`) delimiter for separating prefix and tag in the rule:

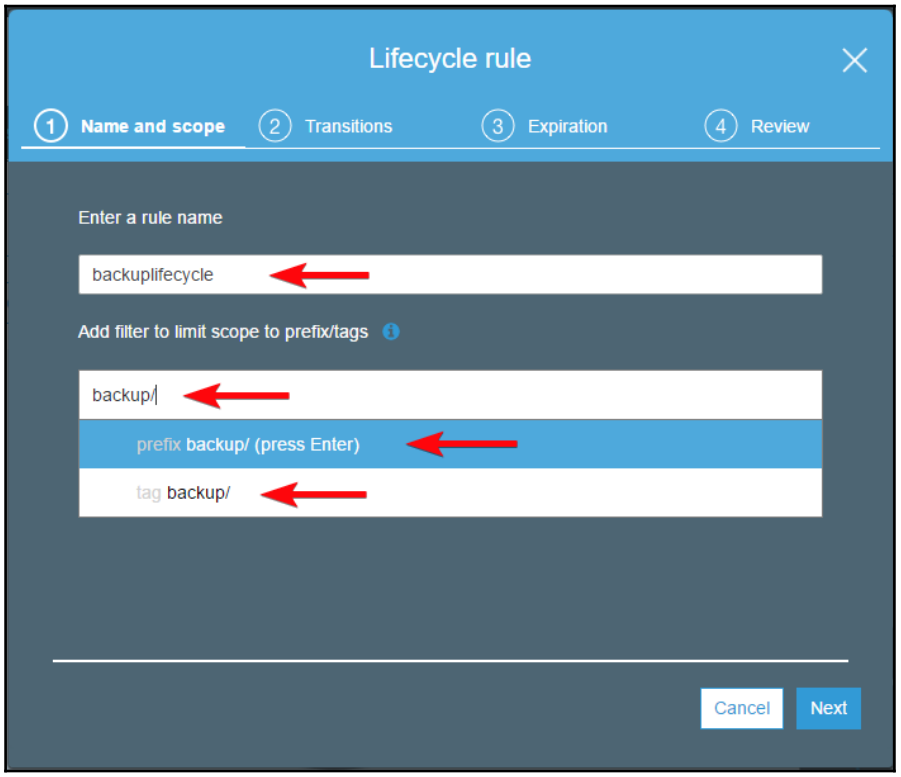


Figure 8.17: Enter lifecycle rule filter

6. In the next screen you can define whether the rule you create applies to the current and latest version of the object or the previous version. If versioning is enabled on the bucket, there may be more than one version of an object. Based on your preference you can select **Current version** or **Previous versions** or both as required:

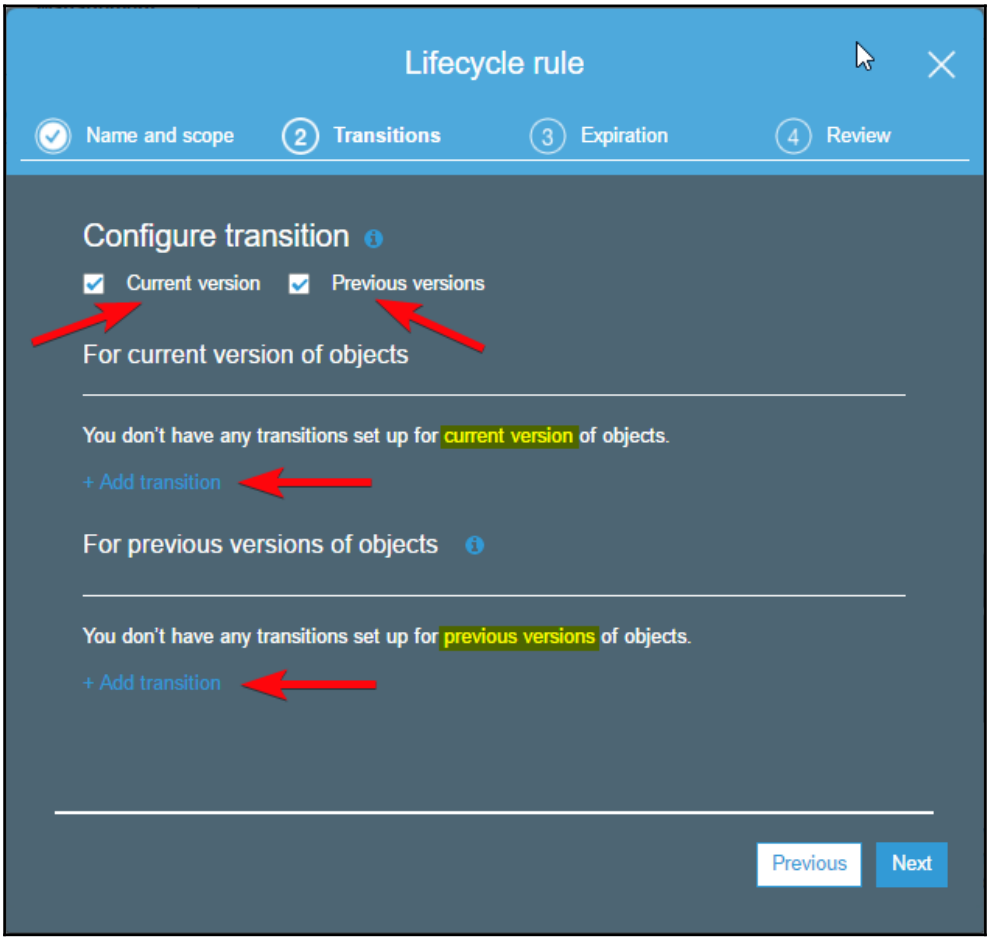


Figure 8.18: Select Current or Previous versions for applying lifecycle policy

7. Click on **+Add transition** as shown in the preceding *Figure 8.18*. It expands the options for selecting transition options as shown in the following *Figure 8.19*. Select the transition action from the combo box, either **Transition to Standard-IA after** or **Transition to Amazon Glacier after**. Also enter the number of days after object creation when an object should transition, as shown in the following *Figure 8.19*. You can also specify similar transition criteria for **Previous versions** of objects:

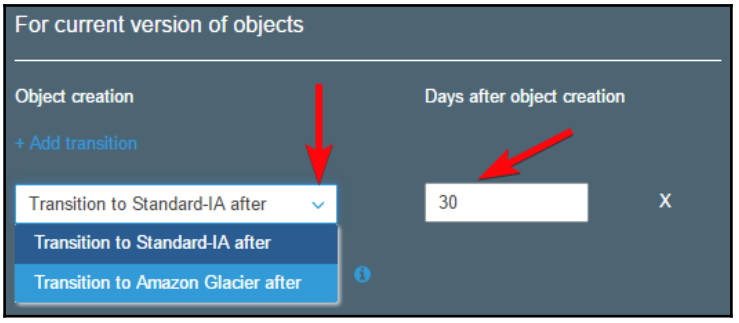


Figure 8.19: Object transition options for lifecycle policy

8. In the subsequent screen, configure expiration options. Similar to the previous steps, you can select either **Current version** or **Previous versions** or both of them as required. You can additionally select to clean up expired object delete markers and clean up incomplete multipart uploads:

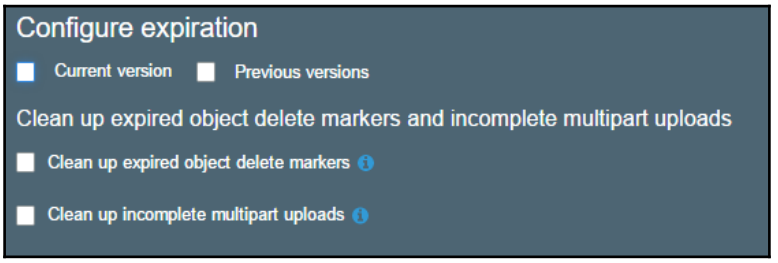


Figure 8.20: Configure expiration options

Based on the version selected, you get options for further selection. As shown in the following *Figure 8.21*, you can choose to expire the current version after a specific number of days. You can also choose to C**lean up expired object delete markers**. Delete markers are not created for expired objects. If you choose to expire objects, you cannot select the option to clean up delete markers. Optionally, you can opt to C**lean up incomplete multipart uploads** after a specific number of days. This is useful in a situation wherein you upload a large object to S3 and the upload process is abruptly closed. S3 can automatically clean up such incomplete multipart uploads based on the selection here:

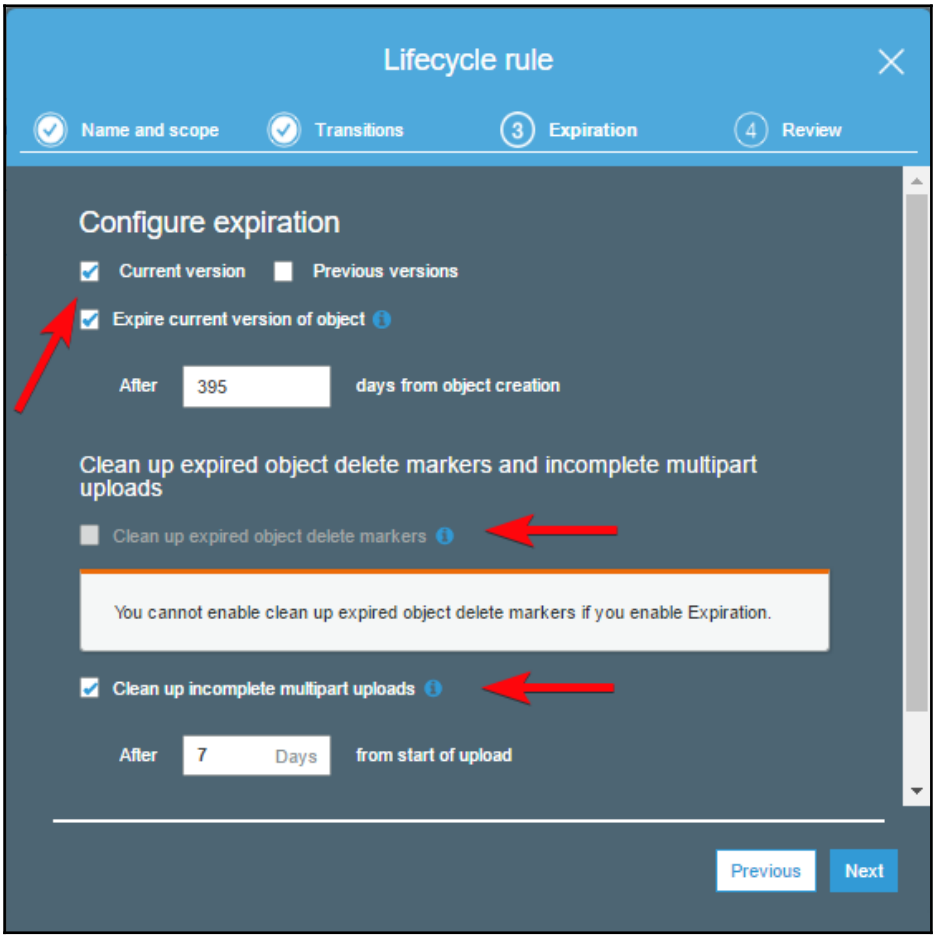


Figure 8.21: Provide additional data for expiration options

9. In the subsequent screen, review the **Lifecycle rule** and click on the **Save** button:

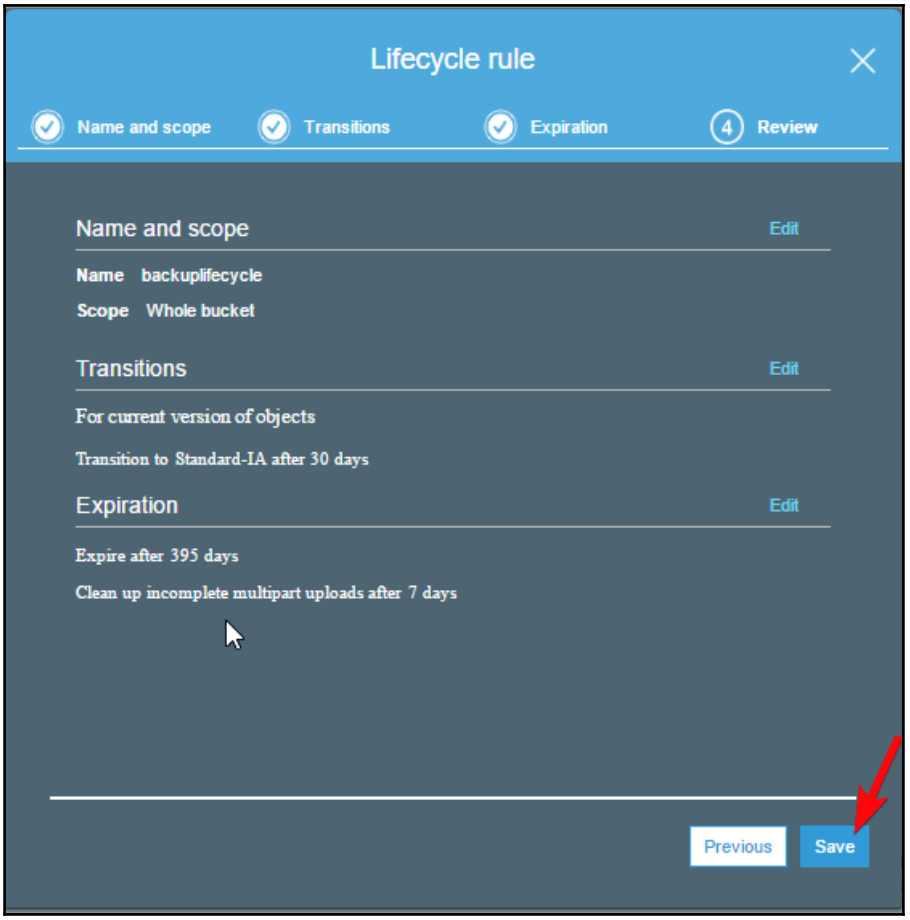


Figure 8.22: Review lifecycle rule and save

Hosting static website on S3

Amazon S3 allows you to host a static website. A static website can contain web pages with static content as well as client-side scripts. S3 does not support service side scripting and due to that, you cannot host a site with any server side scripting such as PHP, JSP, ASP.Net.

You can host HTML pages, CSS, client-side scripts like JavaScripts, and so on. Here's step-by-step process to enable static website hosting on an S3 bucket:

1. Sign in to your AWS console and go to S3 console at `https://console.aws.amazon.com/s3`.
2. Click on the bucket on which you want to enable static website hosting.
3. Click on **Properties** tab as shown in following *Figure 8.23*:

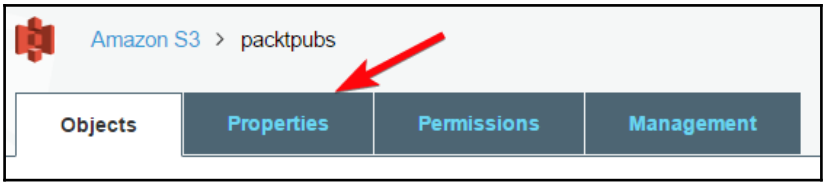


Figure 8.23: Bucket properties tab

4. Click on **Static website hosting** as shown in the following *Figure 8.24*:

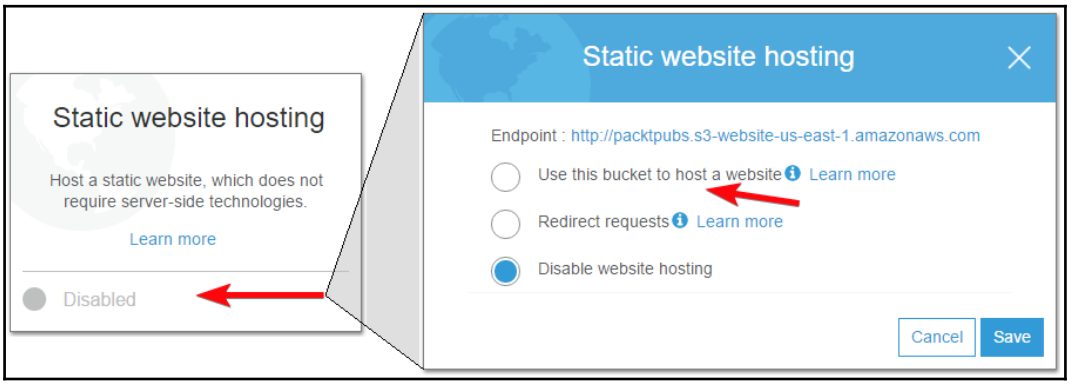


Figure 8.24: Enable static website hosting

5. Specify index and error document for your website as shown in following *Figure 8.25* and click on **Save**. You can also configure **Redirect requests** as needed and optionally specify **Redirection rules**. After configuring the options, you can browse your site from the endpoint URL of the bucket shown as follows:

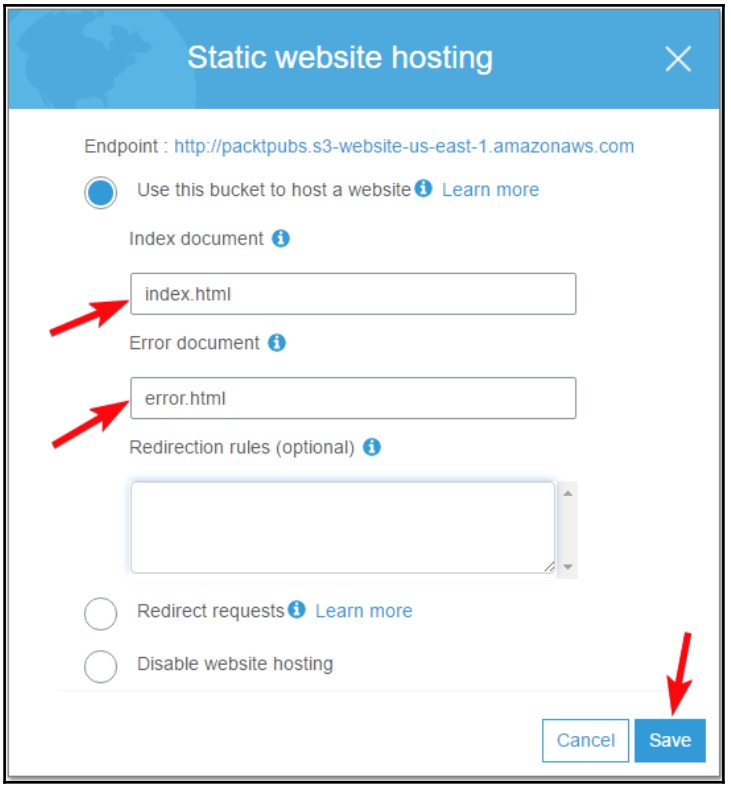


Figure 8.25: Specify index and error documents for static website

Cross-Origin Resource Sharing (CORS)

Before understanding CORS, let us understand the significance of same origin policy. The cross-origin policy is a critical aspect of web application security model. In web application security model, by default, a web browser does not allow a script file associated with a web page to access data associated on a page in different hostname, domain, or port number. The purpose of cross-origin policy is to prevent any malicious script embedded on one page to access sensitive data on another web page.

For example, a script hosted in a page `books.html` on `www.packtpub.com`, can access **Document Object Model** (**DOM**) of any page within the same domain that is, `www.packtpubs.com`. If it tries to access DOM of a page hosted on another domain, the access is denied. Even if a page is hosted on a subdomain like `books.packtpubs.com`, tries to access DOM of another page on `projects.packtpubs.com`, it is denied the access. This is a way to maintain the security of the page based on cross-origin web application security model.

The CORS as the name suggests, is an exact opposite of cross-origin policy.

CORS is a mechanism for client web applications hosted on one domain to use resources hosted on another domain. You can host rich client-side web applications using CORS support on S3. You can selectively enable CORS support on S3 using S3 console, S3 REST API, and AWS SDKs.

Using CORS in different scenarios

- **Use case 1**: Suppose you host a website on an S3 bucket named `packtpubs`. End users can access this site using the URL: `https://packtpubs.s3-website-us-east-1.amazonaws.com`. Amazon's S3 API endpoint for the bucket is assigned as `packtpubs.s3.amazonaws.com`. If you try to make authenticated GET and PUT javascript requests on the pages hosted in the bucket using S3 API endpoint, these requests are blocked by a browser. You can allow such requests using CORS by explicitly allowing requests from `packtpubs.s3-website-use-east-1.amazonaws.com`.

- **Use case 2**: Consider a scenario wherein you host a web site on a bucket and need to load fonts from a different bucket. In such scenario, browser denies access to fonts bucket as it refers to a different origin. CORS can help in such scenario. You can explicitly allow cross-origin request from font bucket.

Configuring CORS on a bucket

For configuring CORS on a bucket, you need to create an XML document which defines the rule to allow cross-origin access on your bucket. You can either open full access to all domains or open access for specific origin domains or URL. For maintaining the security of your site, it is recommended to open access for specific domain. To further strengthen the security of the site, you can allow a specific HTTP methods like GET, POST, PUT, DELETE, and so on.

CORS configuration example XML

```
<!-- Sample policy -->
<CORSConfiguration>
  <CORSRule>
    <AllowedOrigin>*</AllowedOrigin>
    <AllowedMethod>GET</AllowedMethod>
    <MaxAgeSeconds>3000</MaxAgeSeconds>
    <AllowedHeader>Authorization</AllowedHeader>
  </CORSRule>
</CORSConfiguration>
```

The preceding policy, is the default policy that you see when you enable CORS on a bucket. It allows GET requests from all origin. `MaxAgeSeconds` is number of seconds browser can cache response from S3. `AllowedHeader`, by default, allows authorization requests. If you want to allow all headers, you can specify `*` in `AllowedHeader`. It is recommended to exercise caution while configuring CORS and create one or more rules to allow specific domain and HTTP actions. The following example is more specific:

```
<!-- Sample policy -->
<CORSConfiguration>
<CORSRule>
    <AllowedOrigin>http://www.packtpub.com</AllowedOrigin>
<AllowedMethod>PUT</AllowedMethod>
<AllowedMethod>POST</AllowedMethod>
<AllowedMethod>DELETE</AllowedMethod>
<AllowedHeader>*</AllowedHeader>
</CORSRule>
</CORSRule>
<CORSRule>
  <AllowedOrigin>*</AllowedOrigin>
  <AllowedMethod>GET</AllowedMethod>
</CORSRule>
</CORSConfiguration>
```

There are two rules in the preceding example, first rule allows PUT, POST, and DELETE actions from `http://www.packtpub.com`. The second rule allows GET requests from all origins with `AllowedOrigin` as `*`.

Enabling CORS on a bucket

1. Sign in to your AWS console and go to S3 console at `https://console.aws.amazon.com/s3`
2. Click on the bucket on which you want to enable CORS.
3. Click on **Permissions** tab as shown in following *Figure 8.26*:

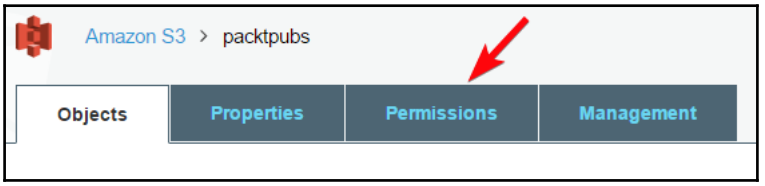


Figure 8.26: Bucket permission tab

4. Click on **CORS configuration** button as shown in the following *Figure 8.27*:

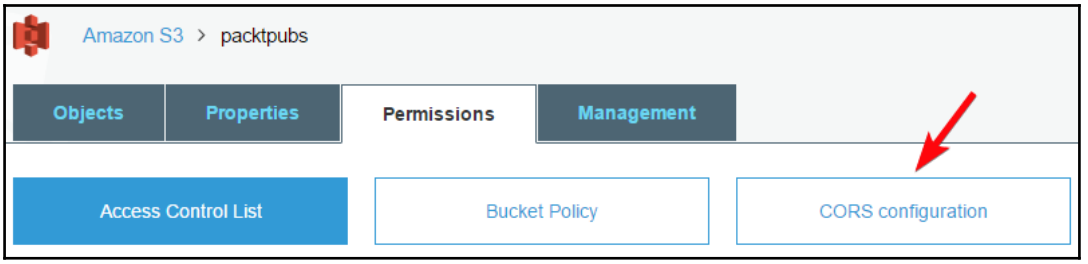


Figure 8.27: Click on CORS configuration

5. Edit the configuration XML as required and click on **Save** as shown in the following *Figure 8.28:*

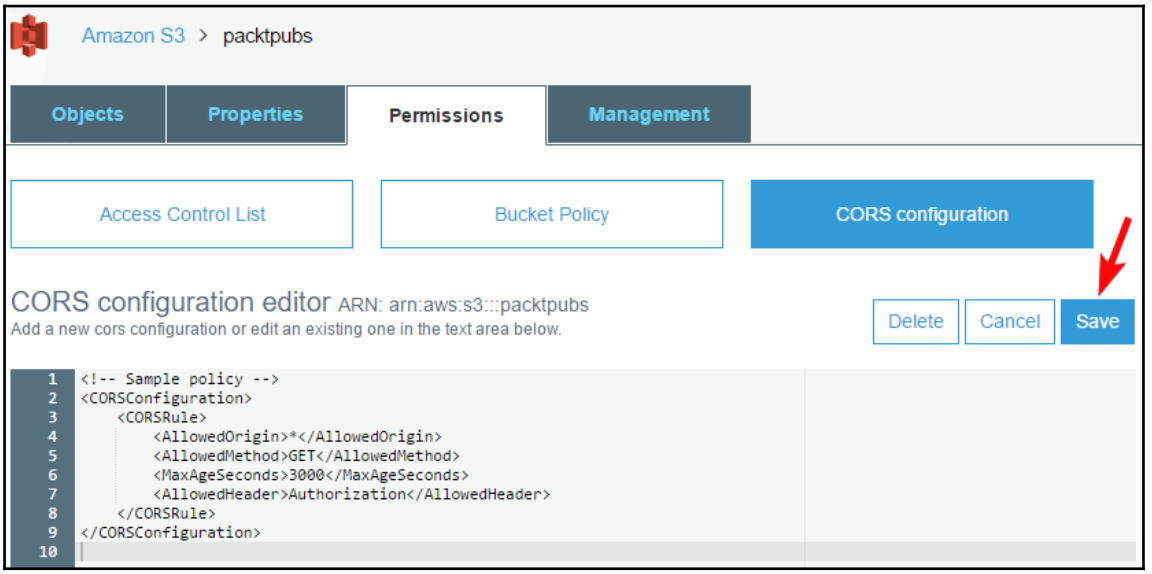


Figure 8.28: Edit CORS configuration and save

Cross-region replication

Amazon S3 enables you to automatically and asynchronously copy objects from a bucket in one AWS region to another AWS region. This is a bucket level feature, which can be configured on source bucket. In the replication configuration, you can specify destination bucket where you want your source bucket objects to be replicated. In the configuration, you can specify a key-name prefixes. S3 replicates all the objects starting with the specific key prefixes to destination bucket. Cross-region replication is generally used for compliance requirements, for minimizing latency in accessing objects, and for any operational reasons wherein compute resources in multiple regions need to access data from region specific bucket.

There are some requirements for enabling cross-region replication:

- Both, source as well as destination bucket must have versioning enabled on them
- Source and destination buckets must be in different regions
- S3 allows you to replicate objects from a source bucket to only one destination
- You must provide permission to Amazon S3 for replicating objects from source to destination bucket
- If source and destination bucket owners are different, source bucket owner must have permission for `s3:GetObjectVersion` and `s3:GetObjectVersionACL` actions
- If source and destination bucket are in different AWS accounts, source bucket owner must have access to replicate objects in the destination bucket

Enabling cross-region replication

1. Sign in to your AWS console and go to S3 console at `https://console.aws.amazon.com/s3`.
2. Click on the bucket on which you want to enable cross-region replication.
3. Click on **Properties** tab as shown in the following *Figure 8.29*:

Figure 8.29: Bucket properties tab

4. Click on **Cross-region replication** and **Enable cross-region replication** as shown in following *Figure 8.30*:

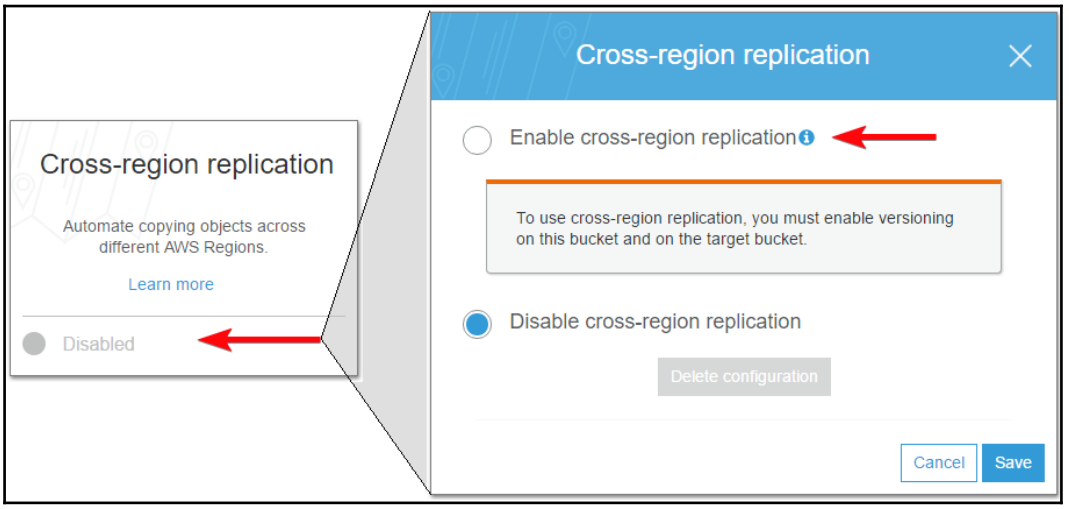


Figure 8.30: Enable cross-region replication

5. As shown in the following *Figure 8.31*, you can select **Whole bucket** or a specific key-name prefix in the bucket for source objects. You need to select a destination region from the drop-down menu. Depending upon available bucket in the destination region, which have versioning enabled, it displays a list of buckets for selection in destination buckets. In addition, you can also select an existing role or create a new role for the cross-region replication on the bucket:

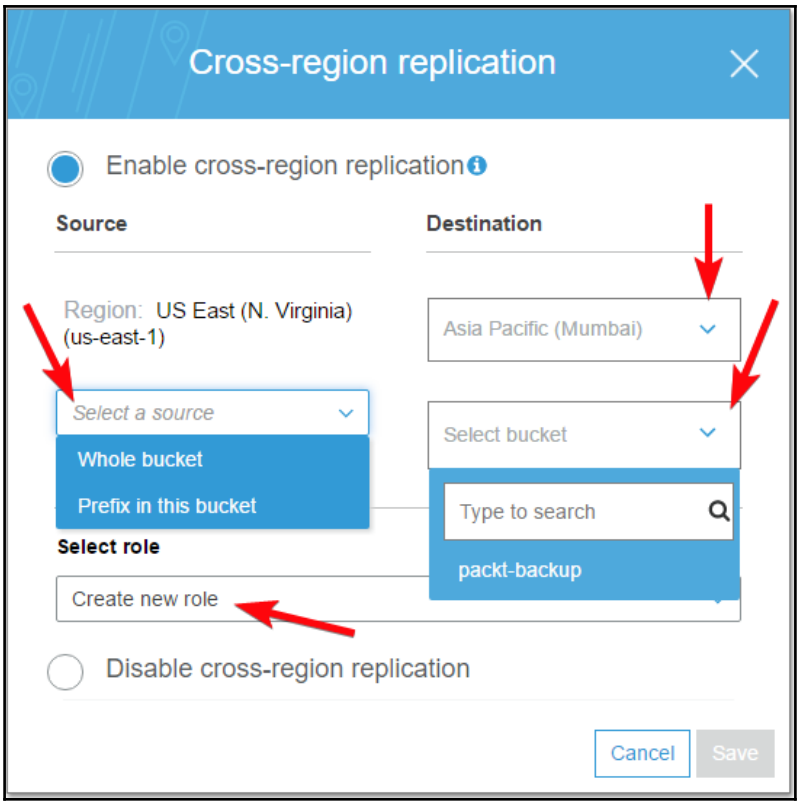


Figure 8.31: Configure cross-region replication

9

Other AWS Storage Options

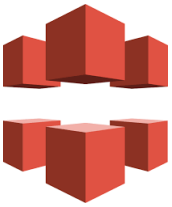
AWS offers a variety of highly available, scalable, reliable, and secure storage services to address various organizational needs. It provides a rich web console for all of the services, which is easy to access and navigate. It's easy to use UI complements efficient services for quickly performing day-to-day administrative. AWS also provides a set of API and CLI interfaces. You can use API and CLI to perform advanced operations or to automate various tasks using customized applications and scripts. The following table describes a number of storage and backup services provided by AWS.

AWS service	**Description**
Amazon S3 	S3 is a cloud-based object storage over the internet. It is ideally suggested for storing static content such as graphics files, documents, log files, audio, video, compressed files, and so on. Virtually, any type of data in any file format can be stored on S3. Currently, permissible object size in S3 is 0 bytes to 5 TB. Objects in S3 are stored in a bucket. A bucket is a logical unit in S3 which is just like a folder. Buckets are created at root level in S3 with a globally unique name. You can store objects and also folders inside a bucket. Any number of objects can be stored in each bucket. There is a soft limit of 100 buckets per account in S3. **Common usage**:S3 can be used for content storage and distribution, static website hosting, big data object store, backup and archival, storing application data as well as for DR. Using Java Script SDK and DynamoDB, you can also host dynamic applications in S3.

Amazon Glacier 	Glacier is a highly secure, durable, and a very low-cost cloud storage service for archiving data and taking long-term backups. Each file or an object stored in Amazon Glacier is called an archive. These stored archives are immutable, which means that contents of the archive cannot be modified. If required, another version of the archive can be stored and the existing version can be deleted. Size of each archive can range from 1 byte to 40 TB. With the help of the S3 lifecycle rules, objects from S3 can be automatically transferred to Glacier. These archives can be logically isolated in containers called vaults. A maximum of 1,000 vaults per account per region can be created. A few important characteristics: • Very economical for storing long term archival data, which is rarely accessed • Retrieval incurs charges and may take a minimum of 3 to 4 hours or more depending on the size of the data • Amazon charges early deletion fees if data is deleted within three months from the date of storing **Common usage**: Glacier can be mainly used for data archival. It is widely used for media asset archiving, healthcare information archiving, regulatory and compliance archiving, scientific data storage, digital preservation, magnetic tape replacement, and so on. It is rarely retrieved for audit or other business purposes.
Amazon EFS 	AWS EFS is a simple to use and scalable file storage, which can be used with EC2 instances. It is a fully managed storage service from AWS which can be used for storing GBs to TBs of data. EFS volumes can be mounted and accessed by multiple EC2 instances at the same time. It uses the **Network File System versions 4.1** (**NFSv4.1**) protocol. When using any EFS volume for the first time, you simply need to mount and format it to the desired filesystem. Subsequently, you can mount this volume on other EC2 instances directly and start using it. EFS volumes can also be accessed from on-premise environments using Direct Connect. You cannot access it from an on-premise environment over VPN connectivity. EFS is available in two modes, General Purpose mode and Max I/O mode. **Common usage**: EFS is designed to provide very high disk throughput. It can be used for big data and analytics, media, content management, web serving, and home directories.

Amazon EBS 	EBS is a persistent, block-level storage service from Amazon. Persistent storage is a type of storage which retains the data stored on it even after power to the device is turned off. Block level storage is a type of storage which can be formatted to support a specific filesystem like NFS, NTFS, SMB, or VMFS. EBS volumes can be attached to an EC2 instance. Because of its persistent nature, data on EBS volume remains intact even after restarting or stopping an EC2 instance. There are five variants of EBS: 1) General Purpose SSD (gp2) 2) Provisioned IOPS SSD (io1) 3) Throughput optimized HDD (st1) 4) Cold HDD (sc1) 5) Magnetic (Standard) Each of these variants, differs in terms of price and performance. EBS volumes are connected as a network storage to an EC2 instance. It can be sized from 1 GB to 16 TB. You can take a snapshot of an EBS volume. A Snapshot is a point-in-time backup of an EBS volume. Snapshots can be used to restore the volume as and when required. **Common usage**: EBS volumes can be used as a root partition and for installing operating systems. It is also used for storing enterprise applications, application data, and databases.
Amazon EC2 instance store 	Instance store is a temporary block-level storage service from Amazon. Unlike EBS, instance store is temporary in nature. Data stored in instance store volume is deleted when EC2 instance is either restarted, stopped, or terminated. Instance store volumes are directly attached to the underlying hosts where an EC2 instance is provisioned. Instance store volumes are faster in comparison to EBS, however, it is a temporary data store. Performance of the instance store volume attached to an EC2 instance, size of each of the volumes, and the number of such volumes that can be attached to an EC2 instance, depending on the EC2 instance type. **Common usage**: It is widely used to store swap files, temporary files, or in applications where good disk throughput is required but data persistence is not required.

AWS Storage Gateway 	AWS Storage Gateway is a hybrid storage service which connects on-premise environments with cloud storage using a software appliance. It seamlessly connects on-premise environments with Amazon's block-level and object-level storage services such as EBS, S3, and Glacier. Storage Gateway uses standard storage protocols such as NFS and iSCSI. It provides low-latency for exchanging data from on-premise to S3, Glacier, or EBS volumes and vice versa. Storage Gateway can provide high performance for frequently accessed data by caching them at source in on-premise environment. **Common usage**: Storage Gateway can be configured for use as a file server in conjunction with S3. It can also be used as a virtual tape library for backup on S3 and virtual tape shelf for archival on Glacier. It can also be configured to be used as a local iSCSI volume. Storage Gateway can also be handy for transferring data from on-premise environments to AWS or transferring the data from AWS to on-premise environments.
AWS Snowball 	AWS Snowball is a petabyte-scale level data transport solution that uses physical appliances to transfer large-scale of data from on-premise environments to the AWS cloud and vice versa. A single Snowball appliance can transport up to 80 TB of data. Snowball comes in two sizes, 50 TB and 80 TB. Data can be copied over to multiple physical appliances and transported to and from an AWS. Transferring large scale data over the internet can take a significant amount of time depending upon the size of data. The purpose of the Snowball service is to minimize the data transfer time by transferring the data using a physical medium rather than transferring data over the internet. Snowball can efficiently compress, encrypt, and transfer data from the on-premise host to the intended Snowball device. Once the data is copied over to one or more snowball devices, these devices are transported back to the nearest AWS data center. Subsequently, AWS transfers data from Snowball devices to S3. **Common usage**: Snowball is used for rapidly and securely transferring bulk data between on-premise data centers and the AWS cloud at a very economical rate.

AWS Snowmobile 	AWS Snowmobile is an exabyte-scale data transport solution that uses physical containers to transfer extremely large-scale of data from on-premise environment to the AWS cloud and vice versa. A Snowmobile container literally comes in a truck which can transfer up to 100 PB of data per snowmobile. The truck carries a high cube shipping container which is 45 foot long, 8 foot wide, and 9.6 foot tall. If your data is more than 100 PB, you can ask Amazon for more than one Snowmobile. At a time, more than one Snowmobile can be connected to the on-premise network for transferring data. When connected to an on-premise network, Snowmobile appears as a standard NFS mounting point on the network. It may require up to 350 KW of power supply. Once the data is transferred from the on-premise network to the Snowmobile, it returns to the nearest AWS data center in the region and subsequently, the data is transferred to the S3 of the respective customer account. **Common usage**: Snowmobile is used for rapidly and securely transferring extremely large scale data between the on-premise data center and the AWS cloud at a very economical rate.
Amazon CloudFront 	Amazon CloudFront is a **Content Delivery Network** (**CDN**) offered by AWS. It is a system of distributed servers spread across edge locations. It is mainly used for caching static content such as web page, stylesheets, client-side scripts, images, and so on. It can also speed up dynamic content distribution. When a user hits a URL which is served through CloudFront, it routes the user request to the nearest edge location. The nearest edge location gives minimum latency in serving the request and provides the best possible performance to the user. **Common usage**: CloudFront is used for providing seamless performance on delivery of a website or web application for a user base spread across multiple geographic locations. It can be used for distributing software or other large files, streaming media files, offering large downloads, and delivering live events.

S3, Glacier, EBS, EC2 instance store, and CloudFront are elaborated in other relevant chapters. Subsequent section of this chapter touches up on EFS, AWS Storage Gateway, AWS Snowball, and AWS Snowmobile.

Amazon EFS

AWS EFS is a simple to use and scalable file storage which can be used with EC2 instances. It is a fully managed storage service from AWS which can be used for storing GBs to TBs of data. EFS volume can be mounted and accessed by multiple EC2 instances at the same time. It uses the NFSv4.1 protocol. When using any EFS volume for the first time, you simply need to mount and format it to the desired file system. Subsequently, you can mount this volume on other EC2 instances directly and start using it. EFS volumes can also be accessed from an on-premise environment using Direct Connect. You cannot access it from on-premise environment over VPN connectivity. EFS is available in two modes, General Purpose mode and Max I/O mode.

In the industry, it is a common requirement to share file systems across the network, which be used as a common data source. EFS is a simple, secure, fully managed, scalable, and reliable block storage, to fulfill common file storage requirements. For using EFS with Linux EC2 instance, you may require installing the latest NFS packages. AWS recommends using the NFSv4.1 client on EC2 instances. Unlike EBS, EFS does not require provisioning a fixed volume size in advance. Being a managed service, you can store as much data as you need and pay only for what you use.

Currently, EFS does not support Windows-based EC2 instances.

An EFS volume is created at the VPC level. At the time of creating an EFS volume, you can specify the AZ from where it can be accessed. EC2 instances from all selected AZ within the same VPC can access the EFS volume. Optionally, you can add tags to your EFS volume. It is recommended to provide a relevant and meaningful name to your EFS volume along with tags for better identification and reference. While creating an EFS volume, it is essential to select the type of EFS volume. Types of EFS volume are General Purpose and Max I/O. The default EFS volume type is General Purpose. Once an EFS volume is successfully created, it returns a DNS endpoints. You can mount the EFS volumes on an EC2 instance or on-premises environment using the endpoint. Remember, you can mount EFS volume on an on-premise network only if you use Direct Connect.

Successful creation of an EFS volume also creates mount points in each AZ. EFS carries properties such as mount target ID, the file system ID, private IPv4 address, the subnet ID in which it is created and the mount target status. It is possible to mount EFS volume using a DNS name. The following *Figure 9.1* elaborates an EFS:

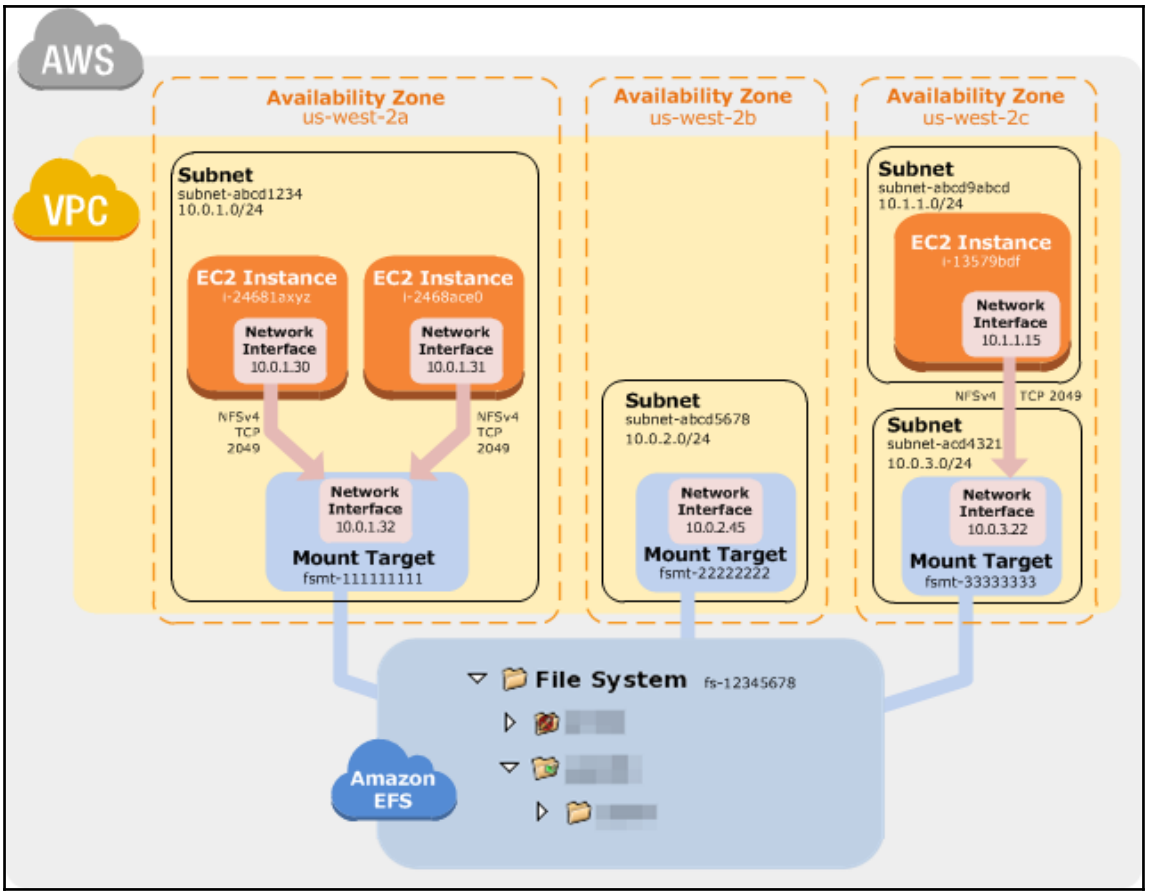


Figure 9.1: EFS

Reference URL: https://docs.aws.amazon.com/efs/latest/ug/how-it-works.html

An EC2 instance does not require a public or elastic IP to mount an EFS volume. You can enable or disable any existing EFS volume as required. You can perform all such changes from the **Manage file system access** option. Once you delete any EFS volume, it cannot be recovered.

Snapshots can be created for EFS volumes. It is also possible to design a backup solution using AWS Data Pipeline for copying data from one EFS volume to another EFS volume. You can also configure a copy operation schedule.

AWS Storage Gateway

AWS Storage Gateway is a hybrid storage service provided by Amazon. With Storage Gateway services, your on-premises applications can seamlessly use AWS cloud storage. The following are some of the important points of AWS Storage Gateway:

- AWS Storage Gateway connects on-premise software appliances with the AWS cloud storage to provide a seamless integration experience and data security between the on-premises data center and the AWS storage services
- It is a scalable and cost-effective storage solution which also maintains data security
- It provides an iSCSI interface, which can be used to mount a volume as a local drive for easily integrating it with the existing backup applications
- AWS Storage Gateway uses incremental EBS snapshots for backing up data to AWS
- AWS provides a VM image for running Storage Gateway on an on-premise data center and you can also run it as an EC2 instance on AWS and in case of any issue, such as if the on-premise data center goes offline, you can deploy the gateway on an EC2 instance
- You can use a Storage Gateway hosted on an EC2 instance for DR, data mirroring, and as an application storage
- By default, Storage Gateway uploads data using SSL and provides data encryption at rest using AES-256 when the data is stored on S3 or Glacier
- Storage Gateway compresses data in-transit and at-rest for minimizing the data size

AWS Storage Gateway provides three types of solutions: file gateways, volume gateways, and tape-based gateways. A file gateway creates a file interface into Amazon S3. It allows you to access S3 using the **Network File System** (**NFS**) protocol. When using volume gateways, you can mount a volume as a drive in your environment. Tape-based gateways can be used similarly to a tape drive for backup.

File gateways

A file gateway creates a file interface into Amazon S3. It allows you to access S3 using the NFS protocol. When you opt for a file gateway, a software appliance is hosted in the on-premise environment on a virtual machine running on VMware ESXi. Once the file gateway is created, it enables you to directly access S3 objects as files using an NFS volume mounted on a server.

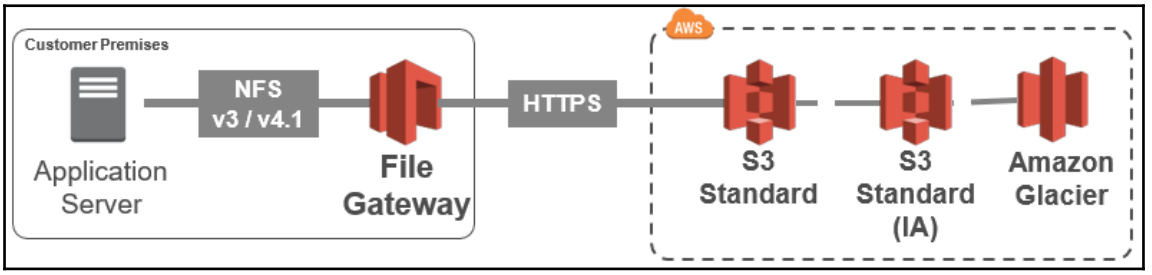


Figure 9.2: File gateway

Here is what file gateways can do for you:

- It allows you to directly store files on S3 using the NFS 3 or NFS 4.1 protocol
- You can directly retrieve files from S3 using the same NFS mount point
- It also allows you to manage S3 data with lifecycle polices, manage cross-region replication, and enable versioning on your data

Volume gateways

When you create a volume gateway, it creates a cloud-backed storage volume, which you can mount as iSCSI devices on your on-premises servers where iSCSI stands for **Internet Small Computer System Interface**. Volume gateways stores all data securely on AWS. There are two types of volume gateway, which determine how much data is stored on-premises and how much data is stored on AWS storage and are discussed as follows:

Gateway–cached volumes

Cached volumes enable you to store complete data on S3 and cache a copy of only frequently used data on on-premise. By reducing the amount of data stored on on-premise environment, you can reduce the overall storage cost. It also boosts performance by providing low-latency access to frequently accessed data using a cache.

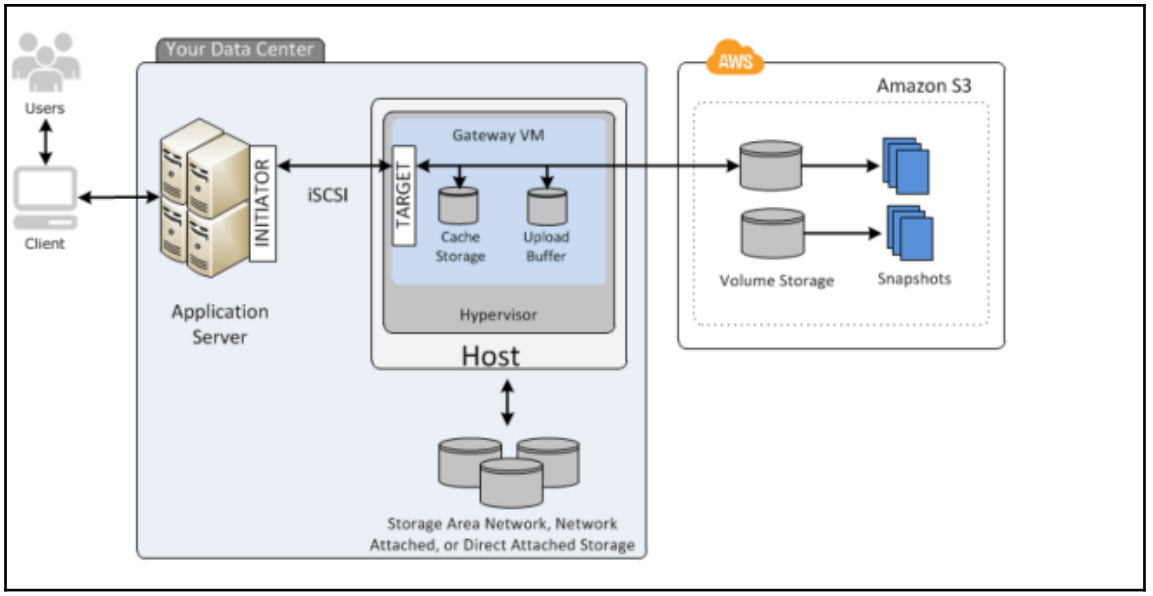


Figure 9.3: Gateway-cached volumes

The key features of gateway-cached volumes are as follows:

- A cached volume stores data in S3 and serves as a primary data storage
- It creates a local copy of frequently accessed data, which provides low-latency access for subsequent data access requests from applications
- By reducing the amount of data stored on an on-premise environment, you can reduce the overall storage cost
- You can create up to 32 gateway-cached volumes in a single storage gateway
- You can store from 1 GiB to 32 TiB in each volume with a maximum storage volume limit of 1,024 TiB (1 PiB)
- You can attach gateway-cached volumes as iSCSI devices on on-premise application servers
- You can take incremental snapshots of gateway-cached volumes
- Gateway-cached volume snapshots are stored on S3 as EBS snapshots
- Gateway-cached volume snapshots can be restored as gateway storage volume or you can create an EBS volume out of them and use it on an EC2 instance
- A maximum size of an EBS volume created out of a snapshot is 16 TiB and you cannot create an EBS volume out of the snapshot if it is more than 16 TiB in size
- AWS stores gateway-cached volume data and snapshots in Amazon S3 and the data is encrypted at rest with **server-side encryption** (**SSE**); you cannot access the data with S3 API or any other tools
- Gateway VM allocates storage in two parts:
    - Cache storage
        - It serves as on-premise durable storage
        - It caches the data locally before uploading it to S3
        - It provides low-latency access to frequently accessed data
    - Upload buffer
        - Upload buffer serves as a staging location prior to uploading the data to s3
        - It uploads data on an encrypted SSL connection to AWS and stores it in an encrypted format on S3

Gateway–stored volumes

You can use gateway-stored volumes when you need low-latency access to your entire data set. It stores all your data locally first and then asynchronously takes a point-in-time backup of this data as a snapshot to S3. It is generally used as an inexpensive off-site backup option for DR.

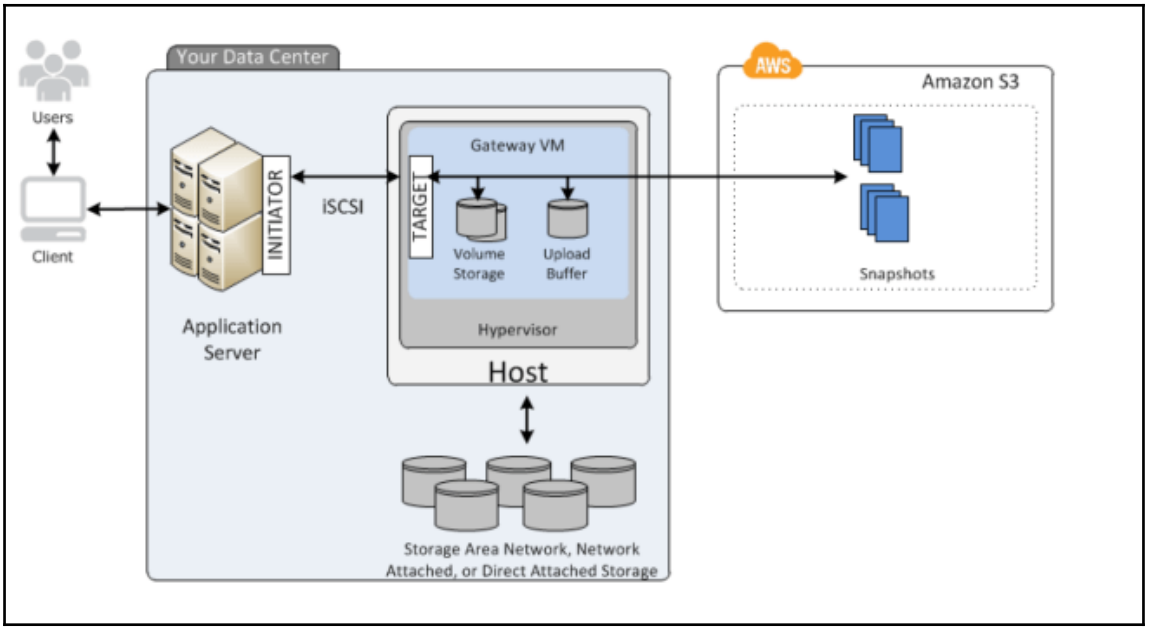


Figure 9.4: Gateway stored volume

The key features of a gateway-stored volume are as follows:

- It maintains the entire data set locally, providing low-latency access to the data
- It stores all your data locally first and then asynchronously takes a point-in-time backup of this data as a snapshot to S3
- You can attach gateway-stored volumes as iSCSI devices on on-premise application servers
- It supports up to 12 gateway-stored volumes per storage gateway applications
- Each gateway stored volume can be from 1 GiB to 16 TiB in size with a total volume storage limit of 192 TiB

- Gateway-stored volumes can be restored as an EBS volume on EC2 instance
- Gateway-stored volume snapshots can be restored as gateway storage volume or you can create an EBS volume out of it and use it on an EC2 instance
- A maximum size of an EBS volume created out of a snapshot is 16 TiB and you cannot create an EBS volume out of the snapshot if it is more than 16 TiB in size
- AWS stores gateway-stored volume data and snapshots in Amazon S3 and the data is encrypted at rest with SSE; you cannot access the data with S3 API or any other tools
- Gateway VM allocates storage in two parts:
    - Volume storage
        - It is used for storing actual data
        - You can map it to an on-premise **DAS** (**Direct-Attached Storage**) or **SAN** (**Storage Area Network**)
    - Upload buffer
        - Upload buffer serves as a staging location, before uploading the data to S3
        - It uploads data on an encrypted SSL connection to AWS and stores it in an encrypted format on S3

Tape-based storage solutions

A tape gateway serves as a replacement for an on-premise tape drive for backup purposes. It stores data on Amazon Glacier for long-term archival. It provides a virtual tape, which can scale based on requirement. It also reduces the burden of managing the physical tape infrastructure.

There are two types of tape-based storage solutions: **Virtual Tape Library** (**VTL**) and **Virtual Tape Shelf** (**VTS**).

VTL

VTL is a scalable and cost effective virtual tape infrastructure, which seamlessly integrates with your existing backup software.

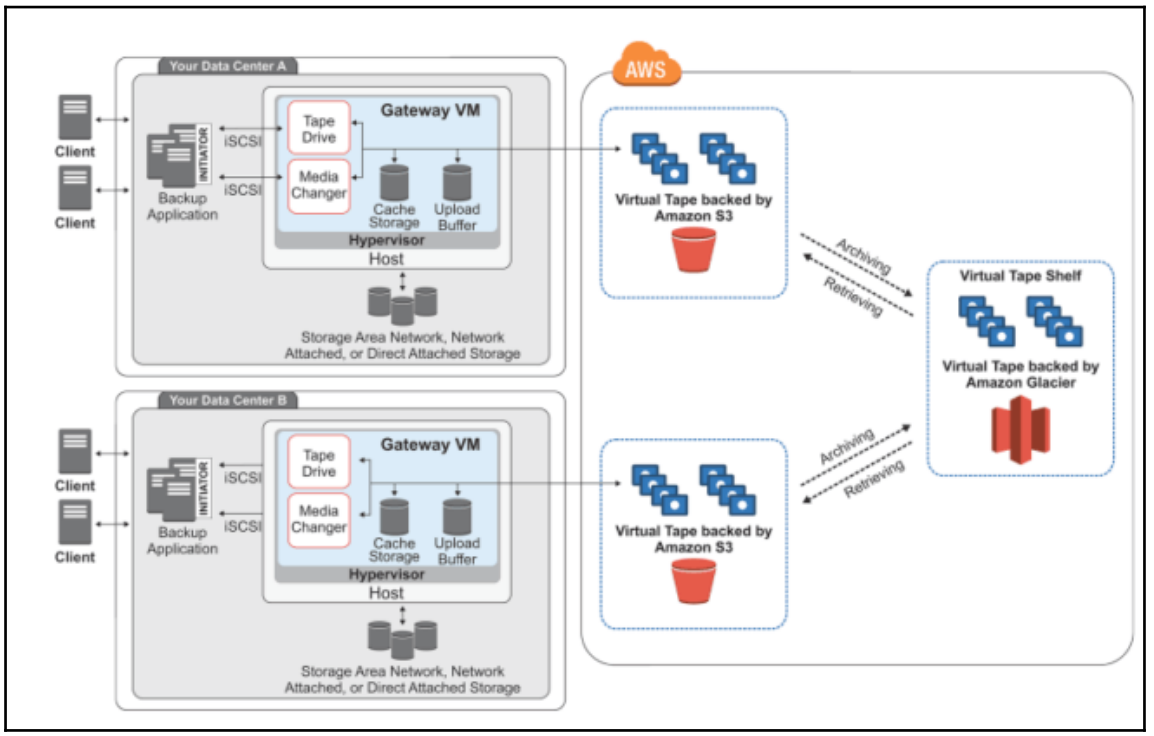


Figure 9.5: Gateway-Virtual tape library

- It provides a low-cost and long-duration archival option in Glacier.
- It provides a virtual tape infrastructure, which can scale based on requirements. It also reduces the burden of managing the physical tape infrastructure.
- It allows you to continue using your existing tape-based backup software for storing data on virtual tape cartridges, which can be created on a gateway-VTL.
- Each gateway-VTL is equipped with preconfigured media changer and tape drives. These are made available to existing backup applications as iSCSI devices. You can add tape cartridges as needed for archiving the data.

- A Gateway VTL contains the following components :
    - Virtual tape
        - Virtual tape emulates a physical tape cartridge wherein the data is stored in AWS storage solutions
        - You can have up to, 1500 tapes in each gateway or up to 150 TiB of total tape data
        - Each tape can store from 100 GiB to a maximum of 2.5 TiB of data
    - VTL
        - VTL emulates a physical tape library wherein the data is stored in S3
        - When a backup software writes data to the gateway, at first the data is stored locally and, subsequently, asynchronously uploaded to virtual tapes in S3
    - VTS
        - VTS works just like an offsite tape holding facility
        - In VTL, data is stored on S3 whereas VTS stores data in Glacier
        - As it uses Glacier for data archival, it becomes an extremely low-cost data archival option
        - VTS resides in the same region where the Storage Gateway is created and there is always only one VTS irrespective of the number of gateways created in an AWS account
        - The gateway moves a virtual tape to VTS when the backup software ejects a tape
        - You can retrieve tapes from VTS only after retrieving the tapes from VTL and it takes around 24 hours for the tapes to be available in the VTL

- Gateway allocates storage in two parts:
  - Cache storage
    - It serves as an on-premise durable storage
    - It caches the data locally before uploading it to S3
    - It provides low-latency access to frequently accessed data
  - Upload buffer
    - Upload buffer serves as a staging location, prior to uploading the data to S3
    - It uploads data on an encrypted SSL connection to AWS and stores it in an encrypted format on S3

AWS Snowball

AWS Snowball comes in a hardware form and can be used with the AWS dashboard or API. It is available in two different sizes, 50 TB and 80 TB. It can be used to transfer **petabytes** (**PB**) of data into and from AWS S3. Dedicated Snowball software is made available by AWS to perform data transfer in a compressed, encrypted, and secure manner. You can attach multiple AWS Snowball devices at the same time to on-premises network backbone. Perfom the following steps to obtain AWS Snowball:

- Sign in to your AWS account and create a job inside the AWS Snowball management console. While creating a job, you need to provide information such as shipping details to receive Snowball device(s), job details mentioning the region, the AWS S3 bucket name, and so on. You also need to provide security details such as ARN of the AWS IAM role and master key from AWS KMS.

- Once the job is created, the Snowball device is shipped to the given shipping address. The following *Figure 9.6* illustrates a Snowball device:

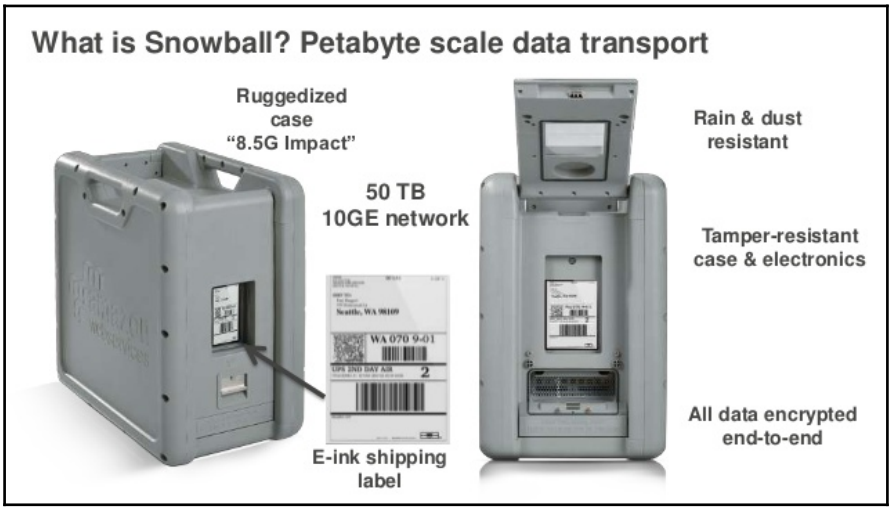


Figure 9.6: Snowball device and its features

Reference URL:
https://image.slidesharecdn.com/clouddatamigration1272016final-160127210855/95/aws-january-2016-webinar-series-cloud-data-migration-6-strategies-for-getting-data-into-aws-14-638.jpg?cb=1466106757

- Once the device is received, connect it to the network. It has two panels, one in the front and another in the back. Flipping the front panel on the top gives access to the E Ink-based touch screen to operate. Network and power cables can be connected on the back side.
- When a Snowball is connected to an on-premise network, it becomes ready to transfer data. Snowball requires credentials to start a data transfer job. These credentials can be retrieved from the AWS dashboard or API. These credentials are encrypted with a manifest file and unlock code. Without the manifest file and unlock code, it is not possible to communicate with the Snowball. AWS provides a Snowball client, to transfer data from on-premise to the Snowball device.
- It is highly recommended that you do not delete the on-premise copy of the data until the data is successfully migrated to the AWS S3 bucket.

- Once the job is complete, disconnect the device from the network and return the device to the shipping address displayed on the display panel. When you create a job, at that time only regional shipping carriers are assigned. For India, Amazon logistics and for rest of the world, UPS are the shipping carrier partners.
- Once the device is shipped back to AWS, the job progress status indicating the movement of data from Snowball to AWS S3 bucket can be tracked on the AWS dashboard or through APIs.

AWS Snowmobile

AWS Snowmobile is an exabyte data transfer service. This hardware come in a high cube shipping container of dimensions, 45 feet long, 8 foot wide, and 9.6 foot tall. Each Snowmobile truck can store up to 100 PB of data and at the same time multiple Snowmobile trucks can be connected to an on-premise infrastructure. It uses 256-bit encryption and a master key for encryption can be managed with AWS KMS. It comes with GPS tracking, alarm monitoring, 24/7 video surveillance, and an optional escort security vehicle while in transit.

When you request for a Snowmobile, AWS performs an assessment and subsequently transports the snowmobile to your designated location. An AWS resource configures it so that you can access it as network storage. During the entire period when the Snowmobile stays at your location, AWS personnel work with your team for assistance. The AWS personnel connect a network switch from your Snowmobile to your local network. Once the setup is ready, you can start the data transfer process from multiple sources in your network to the Snowmobile.

10
AWS Relation Database Services

AWS **Relational Database Service** (**RDS**) is a fully managed relational database service from Amazon. RDS makes it easier for enterprises and developers who want to use a relational database in the cloud without investing much time and resources in managing the environment. AWS RDS supports six database engines: Amazon **Aurora**, **PostgreSQL**, **MySQL**, **MariaDB**, **Oracle**, and **Microsoft SQL Server**. It provides easy to use, cost-effective, and scalable relational databases in the cloud.

The advantages of Amazon RDS as follows:

- It's a fully managed service, which automatically manages backups, software and OS patching, automatic failover, and recovery.
- It also allows taking a manual backup of the database as a snapshot. Snapshots of a database can be used to restore a database as and when required.
- RDS provides fine-grained access control with the help of AWS IAM.

AWS RDS does not provide root access to the RDS instance. In short, RDS not allow the user to access the underlined host operating system. That means, you cannot login to server operating system. It also restricts access to certain system procedure and tables, which may require advance privileges.

After launching RDS in their service offerings, AWS was not providing option to stop an RDS instance for a very long time. Recently, an option to stop the RDS instance is introduced by Amazon. However, unlike EC2 instances, there are some limitations in stopping an RDS instance:

- Only a single AZ RDS instance can be stopped.

- An RDS instance can be stopped for maximum 7 consecutive days. After 7 days, the instance is automatically restarted

This way, by stopping an RDS instance, you can cut the cost for limited period of time. However, there is no limitation on restarting the instance or terminating all unused RDS DB instances to stop incurring the cost.
If a manual snapshot is not taken before terminating the RDS DB instance, it prompts you to take a final snapshot. Once an RDS DB instance is deleted, it cannot be recovered.

Amazon RDS components

Amazon RDS components are as follows:

DB instances

Each Amazon RDS engine can create an instance with at least one database in it. Each instance can have multiple user-created databases. Databases names must be unique to an AWS account and are called DB instance identifier. Each DB instance is a building block and an isolated environment in the cloud. These databases can be accessed using the same tools that are used to access standalone databases hosted in a data center. On top of standard tools AWS RDS instance can also accessed by the AWS Management Console, the API, and the CLI.

Each DB engine has its own version. With the help of a DB parameter group, DB engine parameters can be configured. These parameters help to configure DB instance performance. One DB parameter group can be shared among the same instances types of the same DB engine and version. These sets of allowed parameters vary according to the DB engine and its version. It is recommended to create individual DB parameter groups for each database to have legacy to fine tune as per business need each of them individually. When you choose an RDS instance type, it determines how many CPUs and memory is allocated to it. The most suitable instance type can be selected based on the performance need. Each DB instance can store a minimum of 5 GB and a maximum of 6 TB.

However, there some exceptions like Microsoft SQL Server RDS DB instances supports up to 4 TB of storage. Also, AWS periodically keeps revising this limit for different RDS engines. The minimum and maximum supported storage capacity may vary for each instance type. RDS supports magnetic, general purpose (SSD), and provisioned IOPS (SSD) storage types. RDS instances can be deployed within VPC. Based on the architectural needs, it can be deployed in a public subnet for accessing over the internet or in a private subnet for accessing it within the network.

Region and AZs

AWS hosts its computing resources in data centers, spread across the Globe. Each geographical location where the data centers are located is called a region. Each region comprises multiple distinct locations that are called AZs. Amazon creates AZs in isolated locations such that a failure in one AZ does not impact other AZs in the region. AZs are interconnected with low-latency network connectivity within a region. When you launch an application in multiple AZs, it provides you with high availability and protects you from the failure of an AZ.

An RDS DB instance can be provisioned in several AZs by selecting the Multi-AZ deployment option. It can also be used for DR sites. It is advisable to create RDS in multiple AZs for avoiding single points of failure. It automatically maintains synchronous replicas across multiple AZs. RDS synchronizes DBs between primary and secondary instances. In case a primary instance fails, the load is automatically shifted to a secondary instance.

Security groups

Security groups acts like a firewall. They controls access to a RDS DB instance, by specifying the allowed source port, protocol and IPs. Three types of security group can be attached with Amazon RDS DB instances – DB security groups, VPC security groups, and EC2 security groups.

In general, a DB security group is used when the RDS instance is not in the VPC. The VPC security group is used when RDS instance is within the VPC. The EC2 security group can be used with EC2 instances as well as RDS instances.

DB parameter groups

Over a period when a RDS instance is used in enterprise applications, it may be required to tune certain allowed and common parameters to optimize the performance based on the data insertion and retrieval pattern. The same DB parameter group can be attached to one or more DB instances of the same engine and version type. If not specified then the default DB parameter group with default parameters will be attached. Before creating an RDS instance it is recommended to create DB parameter groups.

DB option groups

DB options groups are used to configure RDS DB engines. With the help of the DB option groups, some of the DB engines can provide additional features for data management, database management, and can also provide additional security features. RDS supports DB options group for MariaDB, Microsoft SQL Server, MySQL, and Oracle. Before creating an RDS instance, it is recommended to create DB option groups.

Amazon RDS charges are based on instance type, running time, storage size, type and I/O requests, total backup storage size, and data in and out transfers.

RDS engine types

Amazon RDS supports six DB engine types: **Amazon Aurora**, **MySQL**, **MariaDB**, **Microsoft SQL Server**, **Oracle**, and **PostgreSQL**. The following table helps us understand the connecting port and protocol for each of these DB instances:

Amazon RDS engine types	Default port	Protocol
Aurora DB	3306	TCP
MariaDB	3306	TCP
Microsoft SQL	1433	TCP
MySQL	3306	TCP
Oracle	1521	TCP
PostgreSQL	5432	TCP

Default port and protocol to connect with each of Amazon RDS engine types

The Amazon RDS engine for Microsoft SQL server and Oracle supports two licensing models: license included and **Bring Your Own License** (**BYOL**). In case you are already invested in purchasing licenses for such databases, it can also be used as a BYOL with Amazon RDS to minimize monthly billing.

Supported instance types may vary for each Amazon RDS engine.

Amazon Aurora DB

Amazon Aurora is a MySQL and PostgreSQL-compatible fully managed **Relational Database Management System** (**RDBMS**). It provides a rare combination of performance and reliability like commercial databases and the cost effectiveness of open source databases. Amazon RDS also provides push-button migration tools to convert your existing Amazon RDS for MySQL applications to Amazon Aurora. It is also possible to use the code, tools, and applications you use today with your existing PostgreSQL databases with Aurora (PostgreSQL).

Creating an Amazon Aurora DB instance will create a DB cluster. It may consist of one or more instances along with a cluster volume to manage the data. These clusters consist of two types of instance: Primary instance and Aurora Replica. Actually the Aurora cluster volume is a virtual database storage volume of type SSD and it spans across multiple AZs in the same region. Each AZ will have a copy of the cluster data. Each Aurora cluster grows automatically as the amount of data in the database grows. It can grow up to 64 TB. Table size is limited to the cluster volume size hence the table can grow up to 64 TB in size:

- **Primary instance**: Performs read, writes, and modifies data to the cluster volume. Each Aurora DB cluster has one primary instance.

- **Aurora Replica**: Performs only read operations. Each Aurora DB cluster supports up to 15 Aurora Replicas plus one primary instance. Amazon RDS Aurora instance availability can be increased by spreading Aurora Replicas across multiple AZs. The following *Figure 10.1* helps to understand this:

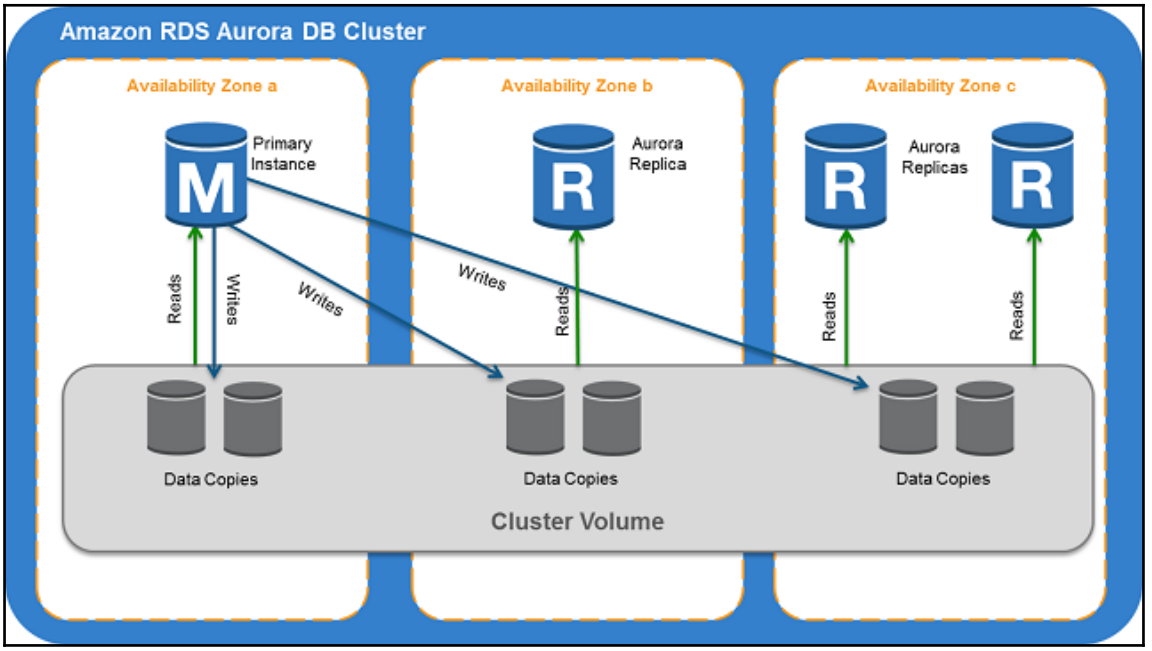


Figure 10.1: Amazon RDS Aurora primary and replica

Reference URL: https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/Aurora.Overview.html

With the help of various endpoints such as the cluster endpoint, reader endpoint, and instance endpoint it is possible to connect to the Aurora DB cluster. Each endpoint consists of a domain name and port separated by a colon and both are discussed as follows:

An endpoint is a URL to access an AWS resource. It can be used to access the DB instance from an application, script, or as a CNAME in a DNS.

- **Cluster endpoint**: To connect with primary instances to perform data read, write, and modification operations. The primary instance also has its own endpoint. An advantage of the cluster endpoint is, it always points to the current primary instance.

- **Reader endpoint**: To connect with one of the Aurora Replicas to perform read operations. This endpoint automatically loads a balanced connection across available Aurora Replicas. In case a primary instance fails, then one of the Aurora Replicas will be promoted as a primary instance and in that situation all the read requests will be dropped.
- **Instance endpoint**: To directly connect with the Primary or Aurora replica instance.

It is designed to be a highly durable, fault tolerant, and reliable and provides the following features:

- **Storage Auto-repair**: It maintains multiple copies of data in three AZs to minimize the risk of disk failure. It automatically detects the failure of a volume or a segment and fixes it to avoid data loss and point-in time recovery.
- **Survivable cache warming**: It *warms* the buffer pool page cache for known common queries, every time a database starts or is restarted after failure to provide performance. It is managed in a separate process to make it survive independently of the database crash.
- **Crash recovery**: Instantly recovers from a crash asynchronously on parallel threads to make a database open and available immediately after crash.

Amazon RDS upgrades the newer major version of Aurora to the cluster only during the system maintenance windows. Timing may vary from region to region and cluster settings. Once a cluster is updated, the database restarts and may experience a downtime for 20 to 30 minutes. It is highly recommended to configure maintenance windows setting to match an enterprise's business requirement to avoid unplanned downtime. But in the case of a minor version upgrade, Amazon RDS schedules an automatic upgrade for all Aurora DB database engines for all Aurora DB clusters. It is optional to allow that update at that scheduled time. It can be manually selected and updated at the desired schedule. Otherwise, it gets applied at the next automatic upgrade for a minor version release.

Amazon Aurora offers *lab mode*. By default it is disabled. It can be enabled for testing current instance and available features in the currently offered version. New features can be tested before applying to production instance.

Comparison of Amazon RDS Aurora with Amazon RDS MySQL

The following table helps in understanding difference between Aurora and MySQL DB engines:

Feature	Amazon RDS Aurora	Amazon RDS MySQL
Read scaling	Supports up to 15 Aurora Replicas with minimal impact on the write performance.	Supports up to only five Read Replicas with some impact on the write operation.
Failover target	Aurora Replicas are automatic failover targets with no data loss.	Manually Read Replicas are promoted as a master DB instance with potential data loss.
MySQL version	Supports only MySQL version 5.6.	Supports MySQL version 5.5, 5.6, and 5.7.
AWS region	Not available in some regions.	Available in all regions.
MySQL storage engine	It supports only InnoDB storage engine type. Tables from other types of storage engine are automatically converted to InnoDB.	Supports both MyISAM and InnoDB.
Read replicas with a different storage engine than the master instance	MySQL (non-RDS) Read Replicas that replicate with an Aurora DB cluster can only use InnoDB.	Read Replicas can use both MyISAM and InnoDB.
Database engine parameters	Some parameters apply to the entire Aurora DB cluster and are managed by DB cluster parameter groups. Other parameters apply to each individual DB instance in a DB cluster and are managed by DB parameter groups.	Parameters apply to each individual DB instance or Read Replica and are managed by DB parameter groups.

Detailed comparison between Amazon RDS Aurora and Amazon RDS MySQL

Reference URL: https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/Aurora.Overview.html

MariaDB

MariaDB is a community version of MySQL RDBMS under GNU GPL license. It maintains a high level of compatibility with MySQL.

Amazon RDS MariaDB manages versions as X.Y.Z where X.Y denotes a major version and Z is the minor version. For example, a version change from 10.0 to 10.1 is considered a major version change, while a version change from 10.0.17 to 10.0.24 is a minor version change. In general, within three to five months it will be introduced in Amazon RDS MariaDB. Amazon RDS Management Console, CLIs, or APIs can be used to perform common tasks such as creating an instance, resizing the DB instance, creating and restoring a backup, and so on.

Minor version support may not be available in all AWS regions.

Amazon RDS MariaDB supports multiple storage engines, but all of them are not optimized for recovery and durability. At present, it fully supports the **XtraDB** storage engine. It supports point in time restore and snapshot restore. It also supports the **Aria** storage type engine, but it may have a negative impact on recovery in the case of instance failure. However, to manage spatial geographical data, it is suggested to use the Aria storage type as the XtraDB storage type doesn't support.

Amazon RDS MariaDB is available in all regions except AWS GovCloud (US) (us-gov-west-1).

Amazon RDS supports two kinds of upgrade for running instances: major version upgrades and minor version upgrades. Minor version upgrades can take place automatically when auto minor version upgrade is enabled from the instance configuration options. In all other cases, upgrading minor versions or major versions requires manual upgrades.

Microsoft SQL Server

It is possible to run Microsoft SQL Server as an RDS instance. It supports various versions of MS SQL such as from SQL Server 2008 R2 to SQL Server 2016. There are a few limitations for Microsoft SQL Server DB instances:

- Each Amazon RDS Microsoft SQL instance can have a maximum of 30 databases. Master and model databases are not counted as a database in this count.
- Some ports are reserved for internal purposes, and cannot be used for general purposes.
- It is not possible to rename a database when an RDS instance with Microsoft SQL Server is deployed in Multi-AZ mirroring.
- The minimum storage required is 20 GB with maximum 400 GB for the Web and Express edition. For the Enterprise and Standard edition a minimum of 200 GB and a maximum of 4 TB of storage is required. In case of larger storage is required, with the help of sharding across multiple DB instances this can be achieved.
- It is recommended to allocate storage based on future considerations. Once storage volume is allocated it cannot be increased due to the extensibility limitations of striped storage attached to Windows Server.
- It doesn't support some of the features of SQL Server such as SQL Server Analysis Services, SQL Server Integration Services, SQL Server Reporting Services, Data Quality Services, and Master Data Services. To use these features it is required to configure Microsoft SQL Server on an Amazon EC2 instance.
- Due to the limitations of Microsoft SQL Server, point in time restore may not work properly until the database has been dropped successfully.

Amazon RDS Microsoft SQL instances support two licensing options: License Included and BYOL. License Included mode is good for the enterprise if you have not already purchased a license. In case you have already purchased a license and are using it in an existing infrastructure, when migration to AWS cloud has been done, once the instance is running with the help of management console or CLI, BYOL can be implemented. When it is deployed in a Multi-AZ mode, the secondary instance is passive and only provides read operations until failover takes place. BYOL is supported for the following Microsoft SQL Server database editions:

- Microsoft SQL Server Standard Edition (2016, 2014, 2012, 2008 R2)
- Microsoft SQL Server Enterprise Edition (2016, 2014, 2012, 2008 R2)

It may be required to upgrade the Amazon RDS Microsoft SQL Server instance, Amazon RDS supports major version and minor version upgrades. In either type, it is essential to perform such upgrades manually. It requires downtime and the total time depends on the engine version and the size of the DB instance.

MySQL

Amazon RDS supports various versions of MySQL. It is also compliant with many leading industry leading standards such as HIPAA, PHI, BAA, and many others. MySQL versions are organized as X.Y.Z where X.Y indicates a major version and Z indicates a minor version. Most of the major versions are supported in most of the regions, but it is recommended to check the availability of desired major and minor version in region, where you are planning to create primary and DR sites. A new version of MySQL is available with the Amazon RDS MySQL instance usually within three to five months. While upgrading MySQL to a newer version, it is possible to maintain compatibility with specific MySQL versions. Major versions can be upgraded from MySQL 5.5 to MySQL 5.6 and then MySQL 5.6 to MySQL 5.7. Usually major version upgrades complete within 10 minutes, but it may vary based on the DB instance type. Minor versions automatically get updated when AutoMinorVersionUpgrade is enabled. Amazon RDS policy on deprecation of MySQL is as follows:

- The major version is supported for three years from the release such as 5.5, 5.6, 5.7, and upcoming
- The minor version is supported for a year from release such as MySQL 5.546
- Three months of grace are provided from the date of version deprecation date

It is essential to perform an OS update (if any are available) before upgrading Amazon RDS MySQL 5.5 DB instance to MySQL 5.6 or later.

Amazon RDS MySQL 5.6 and later supports memcached in an option group. Amazon RDS MySQL also supports various storage engines, but point in time recovery is only supported by InnoDB. Amazon RDS currently does not support the following MySQL features:

- Global Transaction IDs
- Transportable table space
- Authentication plugin

- Password strength plugin
- Replication filters
- Semi-synchronous replication

It is possible to create a snapshot for an Amazon RDS MySQL instance storage volume. Each snapshot is based on the MySQL instance engine version. As it is possible to upgrade the version of an Amazon RDS MySQL instance, it is also possible to upgrade the engine version for DB snapshots. It supports DB snapshot upgrades from MySQL 5.1 to MySQL 5.5.

Oracle

At the time of writing, the following Oracle RDBMS versions are supported by Amazon RDS Oracle engine:

- Oracle 12c, Version 12.1.0.2
- Oracle 11g, Version 11.2.0.4

Amazon RDS Oracle engine also supports the following Oracle RDBMS versions, but soon they will be deprecated:

- Oracle 12c, Version 12.1.0.1
- Oracle 11g, Version 11.2.0.3 and Version 11.2.0.2

Oracle RDS can be deployed within VPC and can perform point-in-time recovery and scheduled or manual snapshots. Optionally, it can be deployed in Multi-AZ to get high-availability and failover. At the time of creating a DB instance master user gets DBA privileges with some limitations, for example SYS user, SYSTEM user, and other DB administrative user accounts cannot be used.

Amazon RDS Oracle instances support two licensing options: License Included and BYOL. Once an instance is running with the help of management console or CLI, BYOL can be implemented. In the case of License Included, it supports the following Oracle database versions:

- Oracle Database Standard Edition One (SE1)
- Oracle Database Standard Edition Two (SE2)

BYOL supports the following license models:

- Oracle Database Enterprise Edition (EE)
- Oracle Database Standard Edition (SE)
- Oracle Database Standard Edition One (SE1)
- Oracle Database Standard Edition Two (SE2)

Amazon allows you to change Oracle RDS instance types, however, if your DB instance uses a deprecated version of Oracle, you cannot change the instance type. Such RDS instances are autocratically updated to new version based on a cut-off date provided by Amazon. For more details on supported versions, deprecated version, and cut-off date for upgrading the deprecated versions, you can check following URL: `http://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/CHAP_Oracle.html`.

It supports both major and minor version upgrades. While upgrading an Amazon RDS Oracle instance from 11g to 12c is a major version upgrade, it has to be done manually and it requires downtime. This downtime may vary based on the current engine version and size of the DB instance. While upgrading engine version it takes two snapshots. The first snapshot is taken just before upgrading, in the case of failure due to any reason it can be used to restore the database. The second snapshot is taken just after engine upgrade is completed. Once DB engine is successfully upgraded, it cannot be undone. If there is any requirement to rollback to previous version, you can create a new instance with the snapshot taken before upgrading the version. Oracle engine upgrade path may vary depending up on the current version running on the instance.

PostgreSQL

The Amazon RDS PostgreSQL engine supports various versions of PostgreSQL. It also supports point-in-time recovery using periodically or manually taken snapshots, Multi-AZ deployment, provisioned IOPS, Read Replicas, SSL connection to DB and VPC. Applications such as *pgAdmin* or any other tool can be used to connect to PostgreSQL and run SQL queries. It is also compliant with many industry leading standards such as HIPAA, PHI, BAA, and many others.

At the time of creating an Amazon RDS PostgreSQL instance master user (super user) a system account is assigned to `rds_superuser` role with some limitations. More details about various supported PostgreSQL supported versions and their features, can be obtained from the URL: `http://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/CHAP_PostgreSQL.html`.

Amazon RDS supported and unsupported database engines can be installed and configured on Amazon EC2 instances as well. Compared to Amazon RDS, installing DB on Amazon EC2 gives more power to fine-tune database engines as gets root level access. Usually, when enterprise is looking for a managed service solution, Amazon RDS is preferred and when they are looking for more detailed fine-tuning hosting on Amazon EC2 is preferred.

Creating an Amazon RDS MySQL DB instance

Amazon RDS MySQL DB instances can be created using Amazon Management Console, CLIs, or APIs and the steps are as follows:

1. Log in to the AWS Management Console with the appropriate user privileges and go to the Amazon RDS dashboard.
2. Select **Launch a DB Instance** as shown in the following *Figure 10.2*:

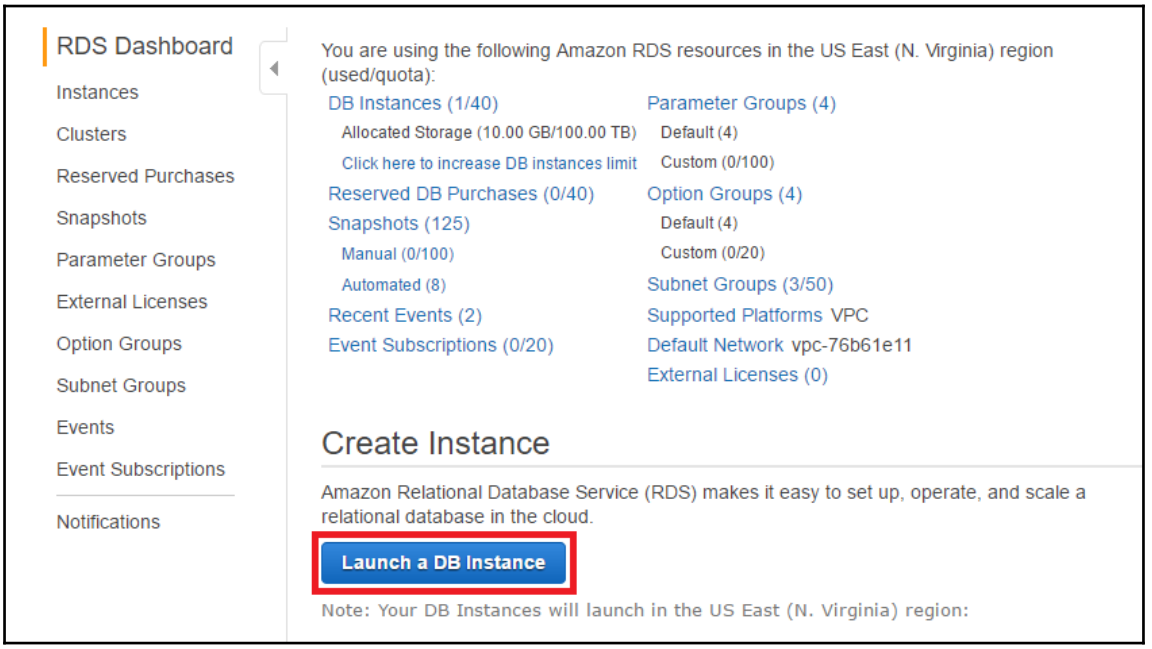


Figure 10.2: Select a DB instance

3. Select the engine type as MySQL, as shown in the following *Figure 10.3*:

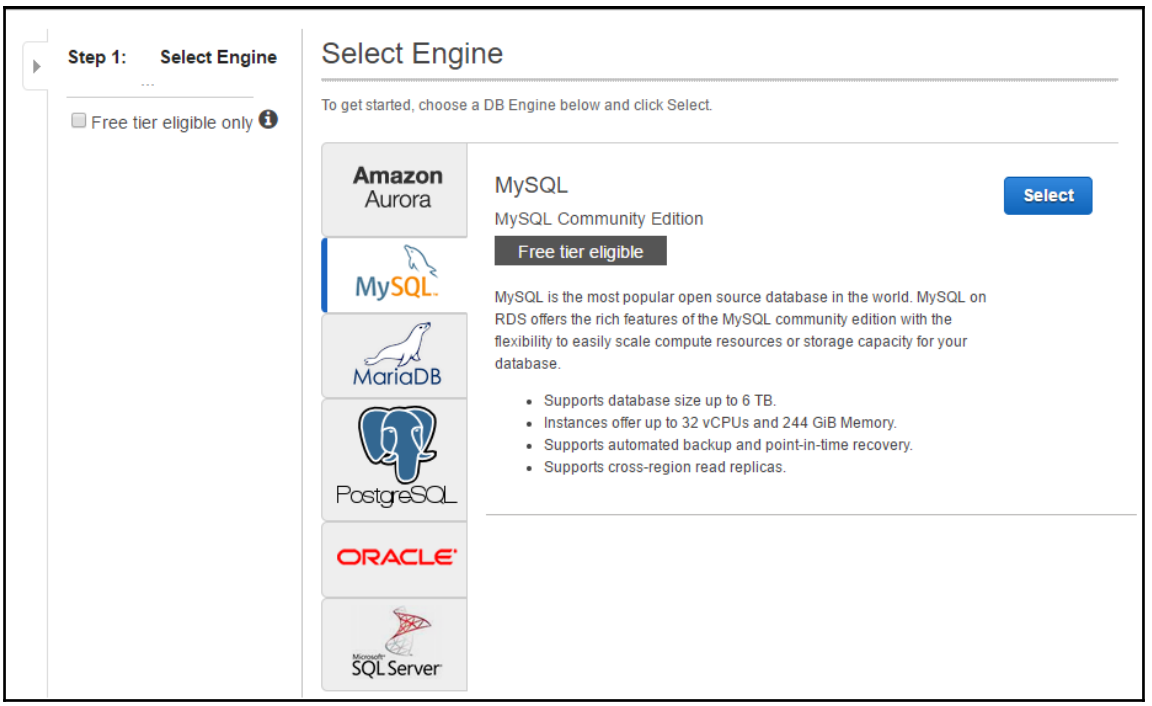


Figure 10.3: Select Amazon RDS engine type as MySQL

Free tier allows us to create a `t2.micro` single-AZ instance for the first year.

4. Select the **Production** type: **Dev/Test** or **MySQL** Multi-AZ, as shown in the following *Figure 10.4*. It is also suggested to switch to Amazon Aurora as it is seamlessly compatible with MySQL:

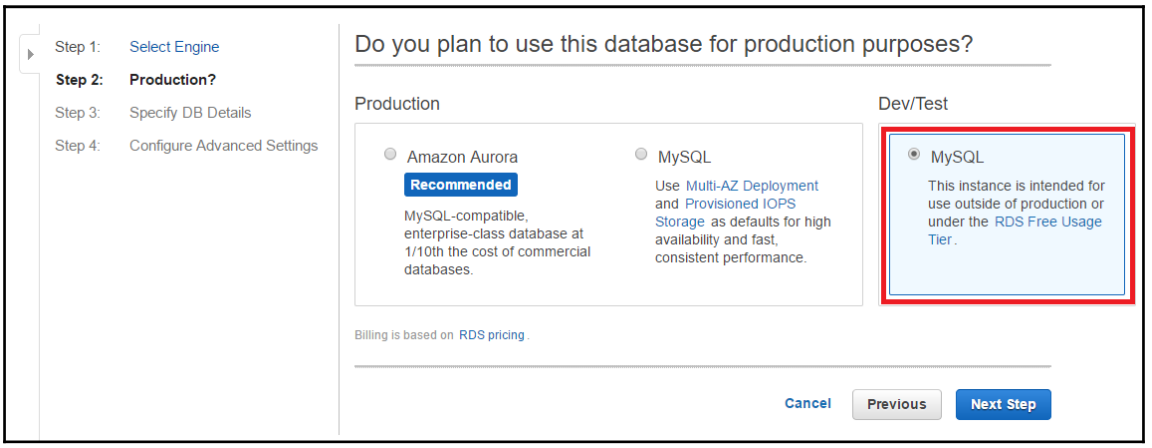


Figure 10.4: Select Amazon RDS MySQL instance to deploy in a single or Multi-AZ

5. Specify Amazon RDS MySQL DB details as follows:

- **Licence Model**: At present it has only one license model: **general-public-licence**.
- **DB Engine Version**: Amazon RDS MySQL engine supports various versions. Based on the enterprise IT requirement, the optimum and latest can be selected.
- **DB Instance Class**: Select the RDS instance type. It decides the size of RAM, CPU, network performance, and EBS performance.
- **Multi-AZ Deployment**: Select **Yes** to enable a standby replica of DB instance in another AZ for failover support.
- **Storage Type**: Supports three types: **Magnetic**, **General Purpose (SSD)**, and **Provisioned IOPS (SSD)**. Storage type can be selected based on the required number of read/write operations.
- **Allocated Storage**: Size of the storage volume to attach to the Amazon RDS DB instance.
- **DB Instance Identifier**: It is a unique DB name for each DB within the AWS account.

- **Master Username** and **Master Password**: Master user is the user with the highest level of privileges within each Amazon RDS instance. It is used to create enterprise level users and grant them privileges to perform day-to-day activities and applications. It also defines passwords.

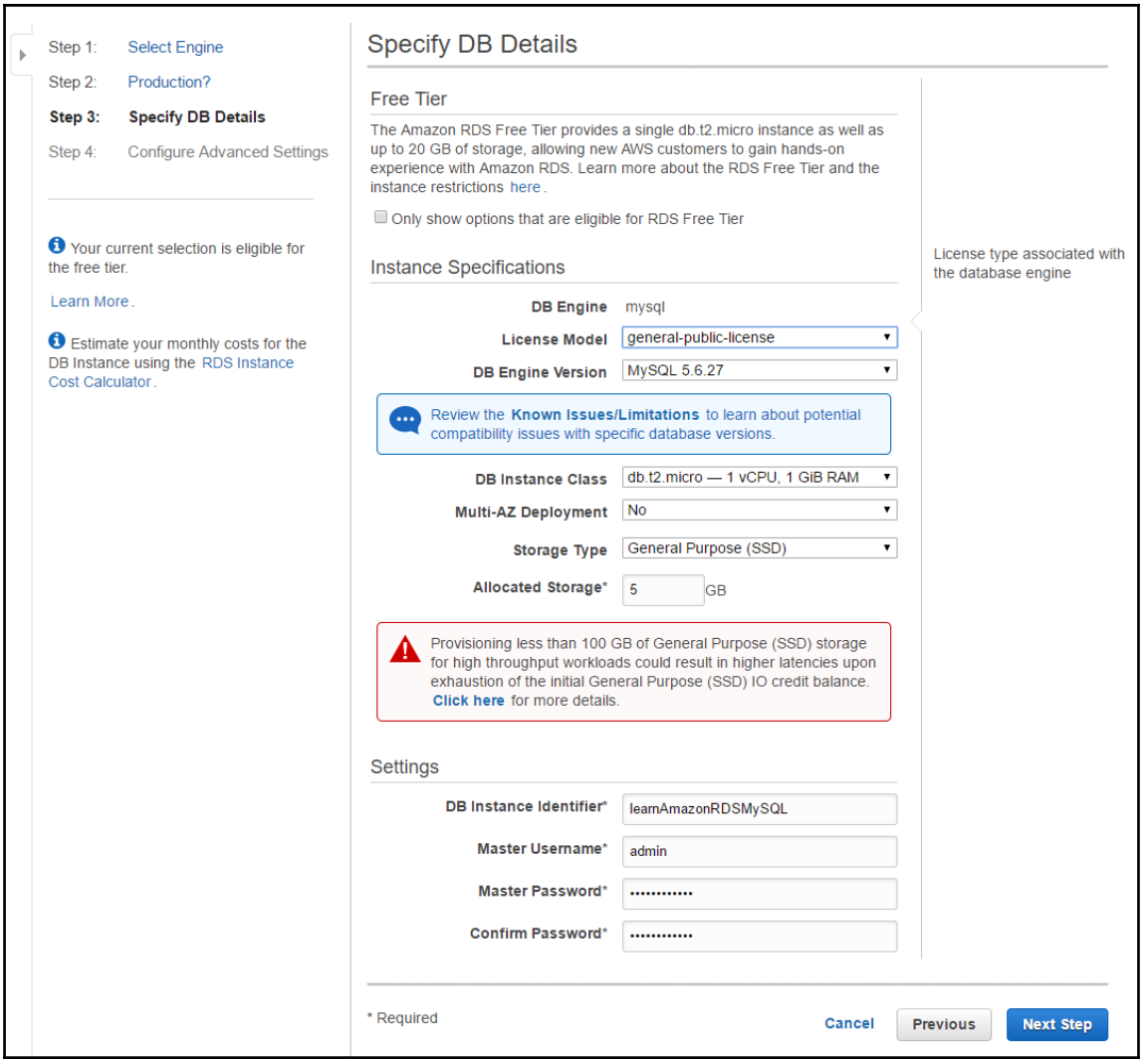


Figure 10.5: Amazon RDS MySQL DB instance details

6. **Configure Advanced Settings** as shown in *Figure 10.6*:

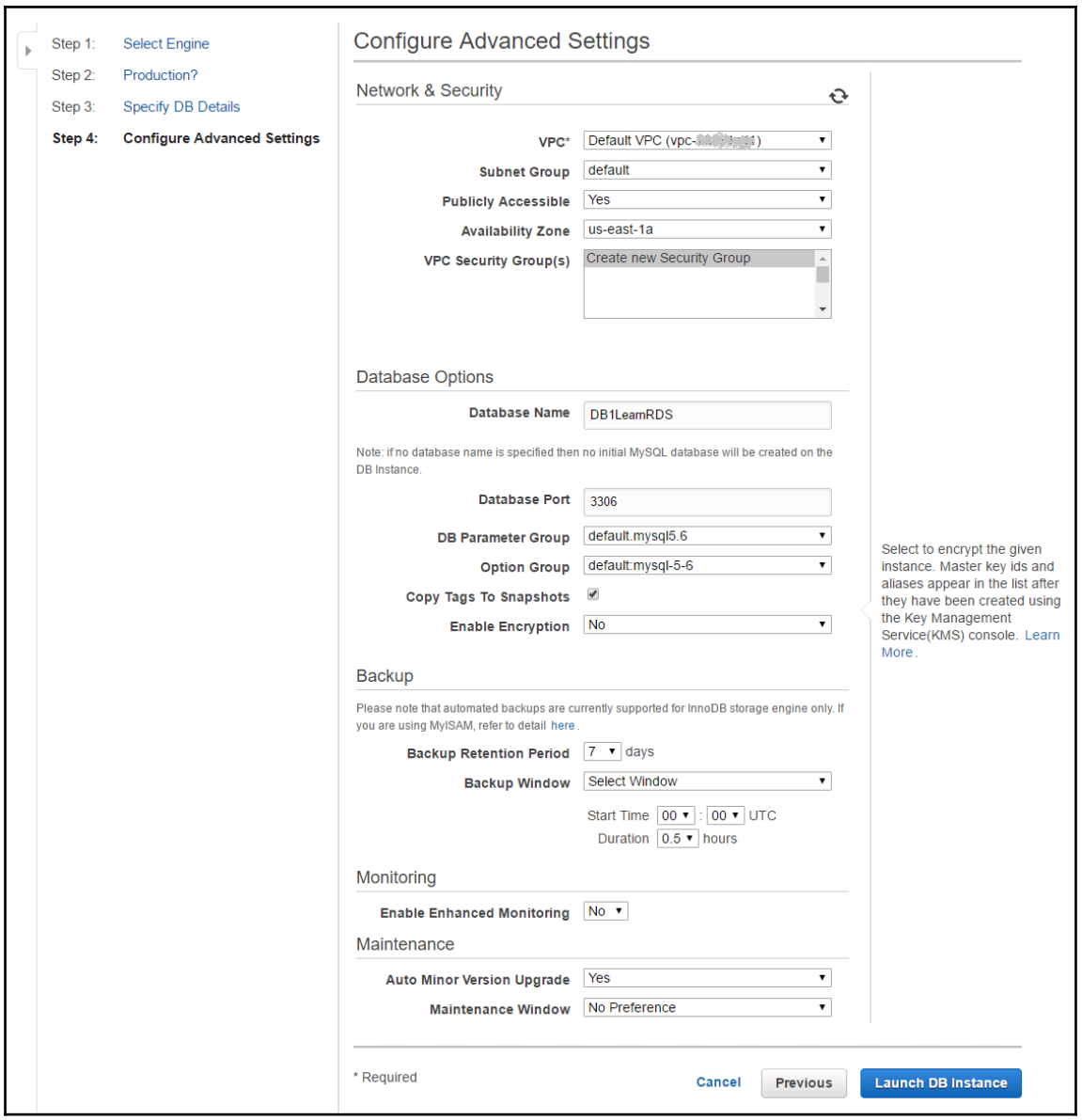


Figure 10.6: Amazon RDS MySQL DB instance advanced configuration

- **VPC**: Select a suitable network VPC.
- **Subnet Group**: Subnet selection depends on architectural design. It can be public or private based on requirements.
- **Publicly Accessible**: It should be selected as **Yes**, if you want to allow the access to the DB from the Internet. It creates a public DNS endpoint, which is Globally resolvable. Select No if you want this DB instance to be accessible only from within the network or VPC.
- **Availability Zone**: If you have any preference on which AZ you want to launch your instance, you choose the specific AZ as required. If you do not have any preference AZ, you can select **No Preference**. In this case, Amazon automatically launches the instance in appropriate AZ to balance the resource availability.
- **VPC Security Group(s)**: Security groups act as a software firewall. One or more security groups can be attached to each Amazon RDS instance.
- **Database Name**: It can be a maximum of 64 alpha-numeric characters. For a given name, it will create a database within the DB instance. It can be blank also.
- **Database Port**: For Amazon RDS MySQL DB instances the default port is `3306`. A default port list for all supported DB engines is given in a table.
- **DB Parameter Group**: It helps to configure DB engine parameters. It is recommended to create the DB parameter group before creating a DB instance. Once it is created it will appear in available drop-down list to use at the time of creating a DB instance. If it is not created before creating a DB instance then the default DB parameter group will be created. This group of parameters can be applied to one or more DB instances of the same engine type. When any dynamic parameter value is changed in the DB parameter group it gets applied immediately whether **Apply Immediately** has been enabled or not. In the case of static parameter value change to get effect, it is required to manually reboot the DB instance.
- **Option Group**: It is supported by the MariaDB, Microsoft SQL Server, MySQL, and Oracle Amazon RDS engines. With the help of option groups it is possible to fine-tune databases and manage data. Option groups can consist of two types of parameter: permanent and persistent.
- To change the persistent options value, it is required to detach the DB instance from the DB option group. When the option group is associated with any DB snapshot, then to perform point-in-time recovery using that DB snapshot it is required to create a new DB instance with the same DB options group. On the other hand, it is not possible to remove permanent options from option group. Also, an option group with permanent options cannot be detached from a DB instance.

- **Copy Tags To Snapshots**: In a backup window, it creates an instance level backup (that is, a snapshot) for the entire volume. By enabling this parameter, each snapshot will copy tags from the DB instance. These metadata can be very helpful to manage access policies.
- **Enable Encryption**: Enabling this option will encrypt data at rest in the DB instance's storage volume and subsequent snapshots. The industry standard AES-256 encryption algorithm is used. Amazon RDS automatically takes care of authentication and encrypts/decrypts data with a minimal impact on performance.
- **Backup Retention Period**: You specify snapshot retention period here in number of days. Snapshots which are older than specified number of days are automatically deleted after specified number of days. Any snapshot can be retained for maximum of 35 days.
- **Backup Window**: An automated backup time window can be specified in a UTC. During this scheduled time, everyday a snapshot will be taken. When any snapshot is aged for a backup retention period it will be automatically obsolete. It will help to achieve an organizational backup retention policy and minimize AWS billing by obsolete old snapshots.
- **Enable Enhanced Monitoring**: Amazon RDS maintains various performance metrics in an Amazon CloudWatch. The main difference between normal and enhanced monitoring is the source of the data. In the case of normal monitoring, CPU utilization of data is derived from the hypervisor. But for enhanced monitoring it is derived from the agent installed on a hypervisor. Data collection from these two sources may vary. In the case of small instance types, this difference can be bigger.
- **Auto Minor Version Upgrade**: Amazon RDS Instances can have two types of upgrade: major and minor. Major version upgrades may require down-time hence they are not performed automatically. Also it may not be possible to revert a major version upgrade. Minor version upgrades are compatible with previous versions and may not require down-time; hence it is performed automatically during scheduled maintenance.

- **Maintenance Window**: Amazon allows you to specify maintenance window. During the maintenance window, Amazon may upgrade DB instance's minor version or DB cluster's OS. Upgrade of the underlined OS or DB version may bring performance implications. Considering this, you should carefully define the maintenance window. Maintenance window definition allows you to define the starting day of the week, hour of the day, minute of the hour, and the total allocated time to perform maintenance activity. Once the maintenance activity begins and if it requires more time to complete the maintenance, it doesn't terminate in between. It stops only after completing the maintenance tasks.

Monitoring RDS instances

Once an Amazon RDS instance is created as per the present need, it is very important to observe its performance with constantly changing business requirements and application loads. It is possible to monitor the instance's CPU utilization, DB connections, free storage space, free memory, and many other parameters. It helps to identify bottlenecks and also will give the opportunity to minimize monthly billing by reducing the resource size if it is underutilized.

An alarm can be configured to take action on a specified threshold. For example, if CPU usage is above 70% for a specified consecutive time period, then send SNS notifications to the DBA. Such an alarm can be created either from the CloudWatch dashboard or from the Amazon RDS dashboard.

To create a CloudWatch alarm from the Amazon RDS dashboard perform the following steps:

1. Go to the Amazon RDS dashboard and select the desired DB instance from the list of running DB instances.
2. Click **Show Monitoring** to get the list of supported metrics. For example, here we have selected the CPU utilization metric and selected **Create Alarm**.

3. Create an alarm by specifying the threshold and other relevant details such as the SNS topic to use to send notifications, CPU utilization threshold, consecutive time period, and alarm name as shown in the following *Figure 10.7*:

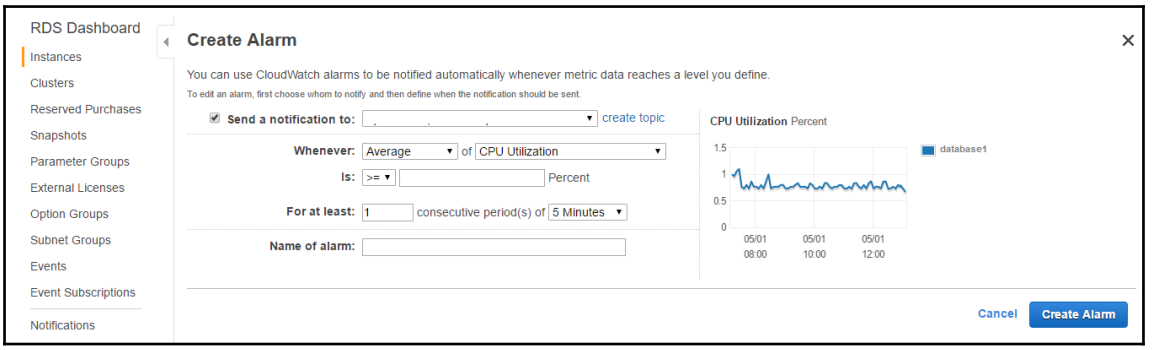


Figure 10.7: Create CloudWatch alert and action from Amazon RDS dashboard.

Creating a snapshot

A snapshot is a frozen image of the DB instance's storage volume. It helps to restore a database to a particular point-in-time. Usually, point-in-time recovery is performed when a database is corrupted or by mistake some data has been dropped (that is, deleted) to bring a database back to the last healthy state. At the time of creating an Amazon RDS instance, a daily snapshot schedule has been already configured. But sometimes it may be required to take a manual snapshot of the DB instance before performing any maintenance task on the database. Snapshot will back up an entire DB instance. It will include all databases and tables and other resources existing on it.

Creating a snapshot for a Multi-AZ DB instance doesn't bring many performance implications. But taking a snapshot for a single-AZ DB instance may suspend DB I/O for a few seconds to minutes. Manual snapshots can be taken using Amazon Management Console, CLI, or APIs. To take a manual snapshot using the management console perform the following steps:

1. Select the desired DB instance.
2. Select **Take Snapshot** from the drop-down menu **Instance Actions,** available above the list of the running RDS instances:

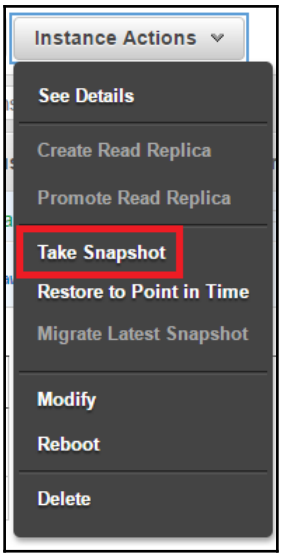


Figure 10.8: Take manual Amazon RDS DB instance snapshot

3. Provide the relevant **Snapshot Name** as shown in the following *Figure 10.9*:

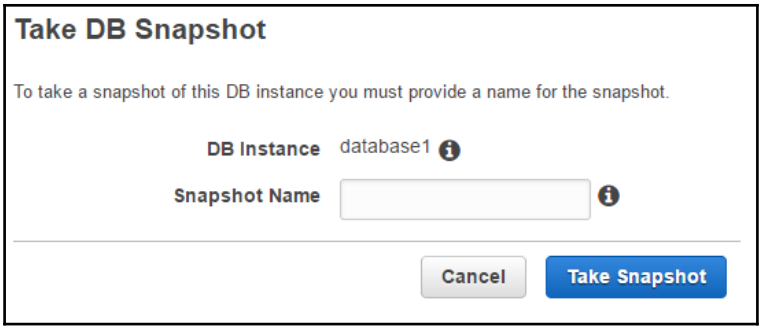


Figure 10.9: Provide Snapshot Name, while taking manual snapshot

Restoring a DB from a snapshot

A snapshot can only be restored by creating a new instance. You cannot restore a snapshot to an existing instance. While restoring the snapshot to a new RDS instance, you can have a different storage volume type from the one used in the snapshot.

Creating an RDS DB instance from a snapshot, automatically attaches a default parameter group and security group to it. Once a DB instance is created, it is possible to change the attached parameter group and security group for that instance.

By restoring a snapshot, the same option group associated with the snapshot will get associated to the newly created RDS DB instance. Options groups are platform-specific: VPC or EC2-Classic.

Creating a RDS DB instance inside a particular VPC will link a used option group with that particular VPC. It means when the snapshot is created for that DB instance it cannot be restored in a different VPC. To do that it requires us to either attach a default options group or create a new options group and attach it to the newly created DB instance from the snapshot.

Creating a DB instance from a snapshot also requires us to provide parameters such as **DB Engine**, **Licence Model**, **DB Instance Class**, **Multi-AZ Deployment**, **Storage Type**, **DB Instance Identifier**, **VPC**, **Subnet Group**, **Publicly Accessible**, **Availability Zone**, **Database Name**, **Database Port**, **Option Groups**, and other parameters that we define at the time of creating a new Amazon RDS DB instance.

It is also possible to copy and share an Amazon RDS snapshot from one region to another and share it among multiple AWS accounts respectively. It may require us to create a DB instance from a snapshot in a different region or AWS account.

Changing a RDS instance type

An RDS instance type is generally changed to accommodate additional resource requirement or for downgrading an existing instance type which is underutilized. For changing the instance type, perform the following steps:

1. From the list of RDS DB instances select the desired instance to modify the instance type and select **Modify** from the **Instance Actions** drop-down menu. The drop-down menu is shown in the following *Figure 10.10*:

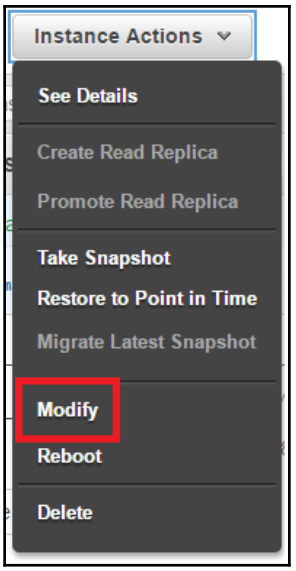


Figure 10.10: Instance Actions drop down menu to select Modify

2. Modifying a DB instance does not only allow us to change the DB instance type, it also allows us to change many other parameters that are provided at the time of creating a DB instance such as subnet group, security group, and many more options. At the end of the parameters that can be changed, an option is available to apply changes now or wait until a next maintenance window, as shown in the following *Figure 10.11*:

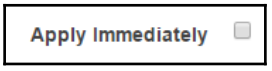


Figure 10.11: DB instance change parameters to Apply Immediately or wait till next maintenance window

3. If DB performance is throttling, you can change the DB instance parameters and apply them immediately. If you do not apply them immediately, the changes are automatically applied during the next maintenance window. Some modifications such as parameter group changes may require us to reboot DB instances. It is advisable to test any changes in a test environment first, before making the changes into productions environment directly.

It is best practice to test such changes in a test environment first, before making changes into productions directly.

Amazon RDS and VPC

Before 2013, AWS used to support EC2-Classic. All AWS account created after 2013-1-04, it only supports EC2-VPC. If an AWS account only supports EC2-VPC, then a default VPC is created in each region and a default subnet in each AZs. Default subnets are public in nature. To meet enterprise requirements, it is possible to create a custom VPC and subnets. This custom VPC and subnet can have a custom CIDR range and can also decide which subnet can be public and which one can be private. When an AWS account only supports EC2-VPC, it has no custom VPC is created, then Amazon RDS DB instances are created inside a default VPC. Amazon RDS DB instances can also be launched into a custom VPC just like EC2 instances. Amazon RDS DB instances have the same functionality in terms of performance, maintenance, upgrading, recovery, and failover detection capability, irrespective of whether they are launched in a VPC or not.

Amazon RDS and high availability

ELB and Auto Scaling can be used with Amazon EC2 to perform load balancing and launching or terminating an EC2 instance to match the load requirement. Auto Scaling cannot be used with Amazon RDS. Amazon RDS supports Multi-AZ deployment to provide high availability and failover. By enabling Multi-AZ deployment, Amazon RDS creates two instances of the same instance type and configuration with individual endpoints in two separate AZs. The sole purpose of another DB instance is to maintain a synchronous standby replica. The standby replica receives traffic only when failover takes place. It can not be used for load balancing or serving read-only traffic. For serving read-only traffic, read replicas can be created, which is different from creating Multi-AZ instances. At present while writing this book, Amazon RDS supports six DB engines. Four out of the six DB engines that is, Oracle, PostgreSQL, MySQL, and MariaDB can perform failover from primary DB instance to secondary DB instance using Amazon's failover mechanism. Microsoft SQL Server RDS engine uses SQL Server mirroring for high availability. Amazon Aurora cluster creates at least three copies of data across Multi-AZs within the same region, which can fulfill high availability requirement. In Amazon Aurora, in case of primary DB instance fails, one of the Aurora Replicas is promoted as a primary.

The following *Figure 10.12* helps to understand the Amazon RDS DB primary and secondary instance in a VPC where the primary instance is denoted as **M** and secondary instance is denoted as **S**:

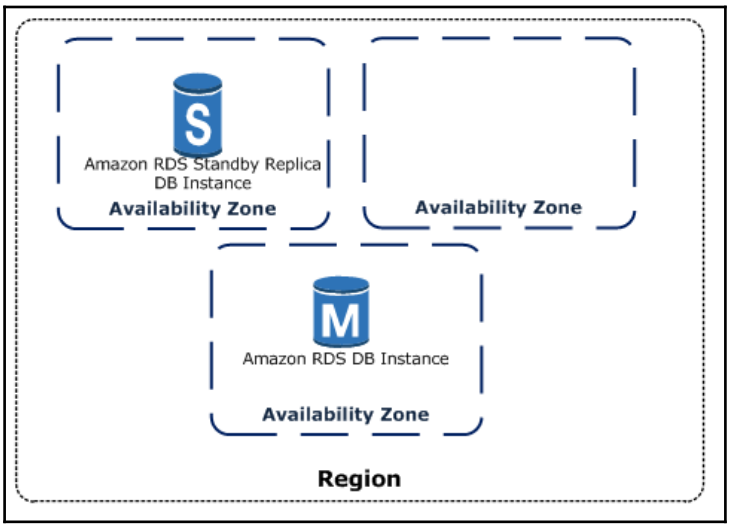


Figure 10.12: Amazon RDS DB instance in a Multi-AZ

Reference URL : https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/Concepts.MultiAZ.html

When using the BYOL licensing model, you must have a license for both the primary instance and the standby replica.

Connecting to an Amazon RDS DB instance

Once the Amazon RDS DB instance is created, you can connect to it for performing read/write operation as well as for performing day-to-day maintenance activities. Before connecting to the DB instance, ensure that the port to connect with DB instance is allowed in the firewall or security group. Also ensure that the source IP from where you need to connect to DB instance is allowed in the security group.

Connecting to an Amazon Aurora DB cluster

Aurora DB clusters consist of a primary instance and Aurora Replica. A separate endpoint is available for the primary instance, Aurora Read instance, or a group of Aurora Read Instances. In line with the task you want to carry out, it is possible to use any of these endpoints in scripting, application, or manually connecting them. Tools used to connect with MySQL databases can be used to connect to Amazon Aurora cluster DB instances.

You can refer to following syntax for connecting to an Aurora DB:

```
mysql –h <aurora–cluster–endpoint> ––ssl–ca=[full path]rds–combined–ca–
bundle.pem ––ssl–verify–server–cert
```

Connecting to a MariaDB Instance

Amazon RDS MariaDB instance up and running has a valid endpoint. It can be used with an application, client, or tool to connect with the DB instance. By default it uses port `3306` and the TCP protocol.

Amazon RDS MariaDB instances can be accessed from the `mysql` command line utility. HeidiSQL is a GUI-based utility and it can be used to connect MariaDB instances.

The following mysql command helps to understand syntax:

```
mysql –h <endpoit> –p 3306 –u <masteruser> –p
```

It is possible to provide a password immediately after the `–p` parameter. Best practice is not to provide it with the command, but to provide it at runtime when prompted for it. Based on the memory (instance type) the number of connection limit is derived. DB instances with higher memory have a higher connection limit. MariaDB maximum possible connection number is defined in the `max_connections` parameter.

Connecting to a MySQL instance

By providing an Amazon RDS endpoint, MySQL DB instances can be connected using a standard MySQL client application or utility. Connecting to MySQL DB instances is similar to connecting MariaDB using a MySQL command-line tool:

```
mysql –h <endpoit> –p 3306 –u <masteruser> –p
```

Amazon RDS DB instance endpoints can be obtained from the RDS console or using CLI `describe-db-` instances.

Optionally it is also possible to use SSL encryption to connect Amazon RDS MySQL DB instance. The `--ssl-ca` parameter is used to provide a public key (`.pem`) for SSL encrypted communication.

Following two tips are repeating, same as MariaDB.

It is possible to pass a password immediately after `-p` parameter. Best practice is not to provide it with the command but to provide at runtime when prompted. The number of connection limit for a MySQL instance is dependent instance type. DB instances with higher memory have a higher connection limit. MySQL's maximum possible connection number is defined in the `max_connections` parameter.

Connecting to an Oracle instance

SQL*Plus is an Oracle command-line utility. It can be downloaded from the Oracle website. Before connecting to the Amazon RDS Oracle DB instance, it is essential to find out Amazon RDS endpoint, port, and protocol. When connecting for the first time, we must connect using master user credentials. Once Amazon RDS relevant application and real entity users are created, it can be used for day-to-day maintenance activity. The following `sqlplus` command line example helps us to understand this:

```
    sqlplus 'mydbusr@(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(HOST=<dns name of
db instance>)(PORT=<listener port>))(CONNECT_DATA=(SID=<database name>)))'
```

Where:

- User: `mydbuser` could be the master user or any other valid user
- `PROTOCOL`: TCP is a protocol and it remains TCP only
- `PORT`: By default, Oracle DB can be connected on `1521`
- `SID`: Database name, intended to connect where one instance may have more than one database

RDS best practices

RDS best practices are as follows:

- Create an individual AWS IAM user to perform DBA tasks. Grant the minimum privileges required to perform day-to-day tasks. Remove unused access key and secret key. Have a strong password policy and rotate the password periodically.
- Before creating an RDS instance identify Amazon RDS essential characteristics to be specified such as VPC, security group, failover or Read Replica requirement, the region and AZs to use, and storage and backup requirements.
- Before creating an RDS instance it is recommended to create a DB options group and DB parameter group.
- Monitor Amazon RDS instance resources such as CPU, memory, and astorage to avoid performance bottlenecks.
- It is recommended to keep some extra buffer in memory and a storage volume while choosing RDS instance types.
- It is recommended to test your environment for failover as it may take different time depending on the use case, instance type, and underlined data size.
- Amazon RDS provides an endpoint to connect to the RDS instance. The IP address beneath that endpoint may change after failover takes place. So if an application caches DNS IP address, set the TTL to under 30 seconds in your application environment.

11

AWS DynamoDB - A NoSQL Database Service

DynamoDB is an easy to use and fully managed NoSQL database service provided by Amazon. It provides fast and consistent performance, highly scalable architecture, flexible data structure, event driven programmability, and fine-grained access control.

Before we get into much details about DynamoDB, let us understand some fundamental characteristics of RDBMS/SQL and NoSQL databases. For a long time, the developer community has been working with **Relational Database Management Service** (**RDBMS**) and **Structured Query Language** (**SQL**). If you have used RDBMS and SQL, you will naturally want to compare and understand the fundamental differences as well as similarities between SQL and NoSQL database.

Let us first understand what an RDBMS is

RDBMS enables you to create databases that can store related data. A database is a collection of information that stores data in database objects, called tables. A table is a collection of related data entries, which consists of columns and rows.

RDBMS enables you to create a link between these tables by establishing a relationship between them. Such a relational model helps in obtaining related information from multiple tables using SQL. You can see in the following *Figure 11.1* that there are three tables, `Employee_Master`, `Department_Master`, and `Emp_Dept`, respectively. All these tables are related with a key field which is called as the primary key. In the following example, you can see how the `Emp_Dept` table, which provides department detail for employees, is linked with the `Employee_Master` and `Department_Master` tables:

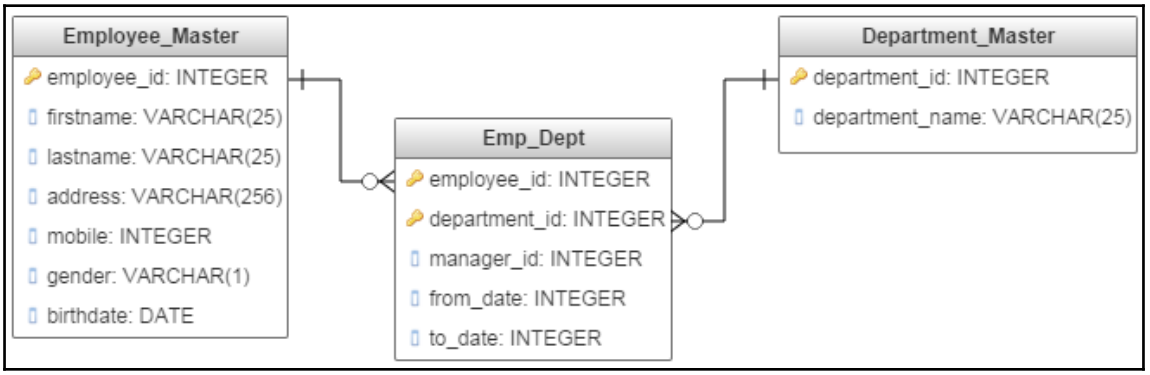


Figure 11.1: Relation between tables in an RDBMS

This is how in a nutshell, RDBMS co-relates with data stored in tables.

What is SQL?

SQL is a standardized language to interact with relational databases. It can execute queries against a database and retrieve data from one or more tables. It can insert, update, and delete records from database tables and perform many other database-related activities. In short, SQL can help you manage your databases.

Here's a simple example of an SQL statement.

```
Select * from Employee_Master
```

The preceding example simply retrieves all the records from `Employee_Master` database. Let's understand one more simple example of an SQL statement, which retrieves related information from multiple tables.

```
Select a.employee_id, b.firstname, b.lastname, c.department_name from
Emp_Dept a, Employee_Master b, Department_Master c where a.employee_id =
b.employee_id and a.department_id = c.department_id
```

The preceding example retrieves related employee information from three different tables. The key to retrieving the information is in the relationship between them. The relationship is established based on key fields in the respective tables.

This is how, in a nutshell RDBMS and SQL work.

What is NoSQL?

A NoSQL database provides a way to store and retrieve data that is in a non-tabular format. It is also referred to as **Non SQL**, **Non Relational** or **Not only SQL** database. NoSQL databases are used for managing large sets of data that are frequently updated in a distributed system. It eliminates the need for a rigid schema associated with an RDBMS.

There are basically four types of NoSQL databases:

- Key-value pair databases
- Document databases
- Graph databases
- Wide column stores

Key-value pair databases

It uses a very simple data model that stores data in a pair of unique keys and the associated value. Commonly, it is used for storing time series data, click stream data, and application logs.

Examples of key-value pair databases are: DynamoDB, Riak, Redis, Aerospike.

Key	**Value**
Name	Abhishek
Mobile	0987654321
Address	Mumbai

DynamoDB is a key-value pair database. In subsequent sections of the chapter, we will get into more details of a key-value pair databases.

Document databases

It stores data elements in a structure that represents a document-like format such as JSON, XML, YAML and so on. Document databases are commonly used for content management and monitoring applications.

Examples of document databases: MongoDB, CouchDB, MarkLogic, and so on.

Unlike RDBMS, the document database schema design is flexible and can combine multiple entities in a single schema. The following example shows how employee information can be stored in a MongoDB document:

```
db.employee.insert(
{
   {
     employee_id:'10001', firstname: 'Tony', lastname: 'Stark', birthdate:
'1965-04-04'
   },
   {
     employee_id:'10002', firstname: 'Thor', lastname: 'Odinson',
birthdate: '1983-08-11'
   },
   {
     employee_id:'10003', firstname: 'Natalia', lastname: 'Romanoff',
birthdate: '1984-11-22'
   }
})
```

Graph databases

A graph database is a NoSQL database type that uses graph structures and stores related data in nodes. It emphasizes on the connection between the data elements to accelerate query performance. It is mainly used for storing geographical data and recommendation engines.

Examples of graph databases: Allegrograph, IBM graph, Neo4J, and so on.

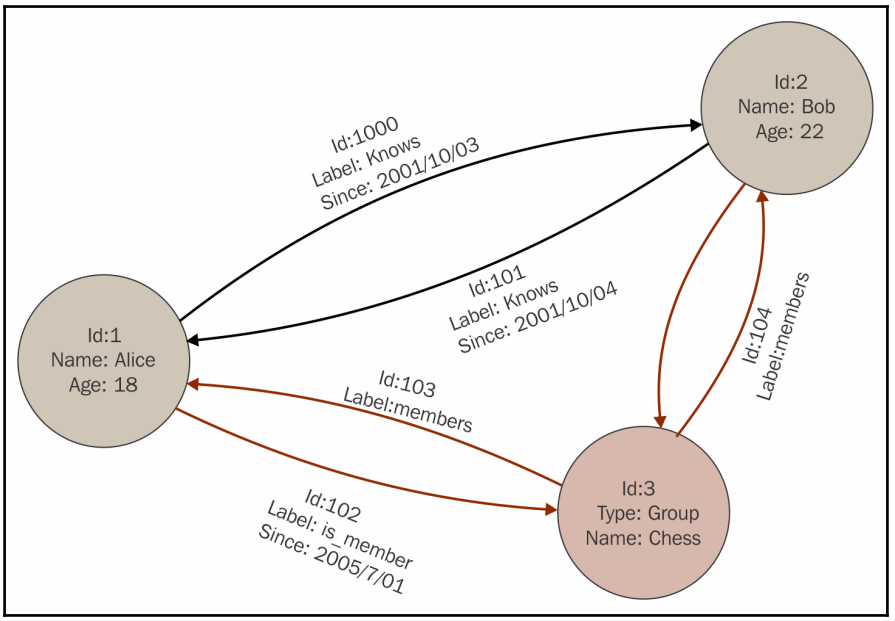


Figure 11.2: Graph database

Image Source: https://upload.wikimedia.org/wikipedia/commons/3/3a/GraphDatabase_PropertyGraph.png

Wide column databases

The wide column database is a type of NoSQL database that stores data using a column-oriented model. It is also called a table-style database or column store database. It stores data in a table-like structure and it can store large number of columns.

Wide column databases are generally used for storing data related to internet search and other similar large-scale web applications.

Examples of wide column databases: Cassandra, HBase, SimpleDB, Accumulo, Hypertable, and so on.

Employee_id	Firstname	Lastname	Birthdate
10001	Tony	Stark	1965-04-04
10002	Thor	Odinson	1983-08-11
10003	Natalia	Romanoff	1984-11-22

In a wide column database, each column is stored in a separate file as depicted in the preceding example.

When to use NoSQL databases?

A relational database stores data in one or more related tables. The relational model and tabular format minimizes data duplication. However, scaling relational databases can become very resource-intensive. In contrast, NoSQL databases stores related data in a single document, which can improve accessibility and scalability. A NoSQL database trades some of the query and transaction capabilities of RDBMS in favor of better performance and high scalability.

NoSQL is generally used for big data, advertisement technologies, gaming, mobile applications, time series data, logs, IoT, and many other applications where heavy write performance, reduced latency, and a dynamic schema are required.

SQL versus NoSQL

Following are some of the key differences between SQL and NoSQL databases:

SQL	NoSQL
Database systems are termed RDBMS.	Database systems are termed non-relational or distributed systems.
It follows a rigid and pre-defined schema model.	It uses a dynamic schema.
It stores data in tabular form and is also known as a tabular database.	It stores data in a collection of key-value pair, graph database, documents, and wide column stores.
Databases can be scaled vertically.	Databases can be scaled horizontally.

It uses SQL for defining and managing data.	It uses an unstructured query language, which varies from database to database.
It is best suited for complex queries.	It is not suitable for complex queries as it does not use the relational data model.
It is not suitable for hierarchical data stores.	It is best suited for hierarchical data stores.
Oracle, SQL Server, MySQL, PostgreSQL, and so on are SQL databases.	DynamoDB, MongoDB, Bigtable, Cassandra, Hbase, CouchDB, and so on are NoSQL databases.

Introducing DynamoDB

Amazon DynamoDB is a fully managed NoSQL database service from Amazon that provides fast and flexible NoSQL database service for applications that need consistent and low-latency access at any scale. It supports key-value and document data models. It provides a dynamic schema model and predictable performance. DynamoDb is best suited for big data, advertisement technologies, gaming, mobile applications, time series data, logs, IoT, and many other applications where heavy write performance, reduced latency, and a dynamic schema are required.

DynamoDB allows you to store any amount of data and handle any level of user traffic. It allows you to scale up or down a table's read/write capacity without affecting the up time and performance of the table. You can use the management console for monitoring DynamoDB resource utilization and it's effective performance metrics.

It helps you reduce storage usage by automatically deleting the expired items from a table. Since it can help you automatically delete expired data, the cost for storing data can be significantly optimized.

DynamoDB tables are spread across a cluster of servers that are sufficient for handling the desired throughput and the required storage for consistent and reliable performance. It stores data in **Solid State Drive** (**SSD**) and the data is replicated across multiple AZs for obtaining high availability and data durability.

DynamoDB components

There are basically three core components of a DynamoDB table: tables, items, and attributes.

Let us understand these core components in detail:

- **Tables**: DynamoDB stores data in an entity called a table. A table consists of a set of data; for example, the following employee table shows how you can store employee information in a DynamoDB table.
- **Item**: A table consists of multiple items. An item consists of a group of attributes. An item is like a record or row in an RDBMS table. In the following employee table example, you can see the data of two employees. Each employee data represents an item in DynamoDB.
- **Attributes**: An item in a table consists of multiple attributes. An attribute is a basic data element of an item. It is similar to a field or a column in an RDBMS. However unlike RDBMS, attributes in a table item can have sub attributes. You can see that in the following employee data example, the address attribute is further broken down into multiple attributes for representing specific data:

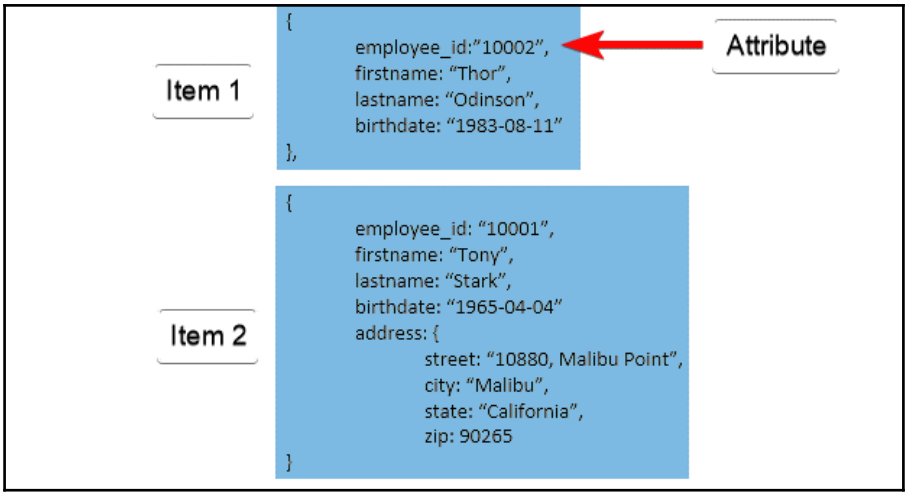


Figure 11.3: DynamoDB table items and attributes

As you can see in the preceding example, the first record in a table does not contain an address whereas, the next record has an address and its subset attributes in the record. This shows the flexibility in the schema of a DynamoDB table. You can have different attributes in subsequent records based on need.

Primary key

While creating a DynamoDB table, you need to specify the table name and the primary key of the table. It is mandatory to define a primary key in the DynamoDB table. A primary key is a mechanism to uniquely identify each item in a table. A primary key does not allow two items with the same key value in a table. There are two types of primary keys:

- **Partition key**: It is a simple primary key that is composed of a single attribute named partition key. DynamoDB partitions the data in sections based on the partition key value. The partition key value is used as an input to an internal hash functions, which determines in which partition the data is stored. This is the reason partition key is also called the *hash* key. No two items in a table can have the same partition key. Since the data is divided into partitions based on its partition key value, data retrieval becomes faster. You can observe in the preceding *Figure 11.3*, `employee_id` is an example of a simple primary key. You can rapidly access employee information from a table by providing the `employee_id`.

- **Partition key and sort key**: Partition key and sort key are composed of two attributes and that's why it is also called as a composite primary key. As the name suggest, the first attribute of the key is the partition key and the second attribute is the sort key. Just like the partition key, the composite key also uses the partition key as an input to an internal hash function. This hash function determines the place of an item in a partition. The partition is a physical storage, which is internal to DynamoDB, for arranging the item to get the best possible performance from the table.

DynamoDB stores all items with the same partition key together. In a partition, items are ordered based on a sort key value. Thus, the partition key determines the first level of sort order in a table whereas the sort key determines the secondary sort order of the data stored in a partition. A sort key is also called a *range key*.

A table with both partition key and sort key can have more than one item with the same partition key value; however, it must have a different sort key. In the following example, you can see that `department_id` is a partition key and `employee_id` is a sort key. A department can have multiple employee records, which leads to repeating the `department_id`; however, `employee_id` cannot repeat with the same department name:

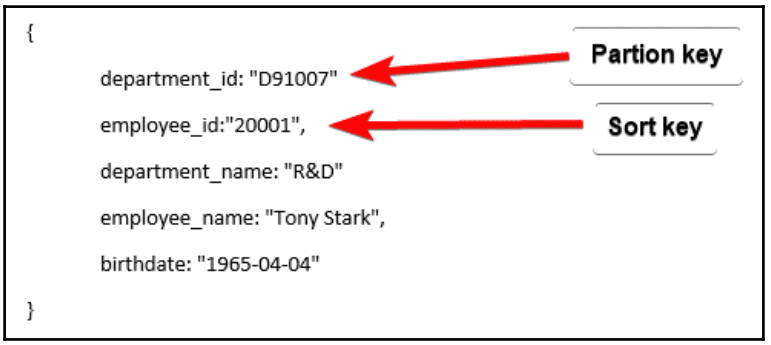


Figure 11.4: Partition key and sort key

In a nutshell, there can be two types of primary keys:

- Single attribute *partition key*
- Two attributes: *partition key and sort key*

Secondary indexes

DynamoDB allows you to create secondary indexes on a table. It is an alternate way to query table data in addition to querying it using the primary key. It is not necessary to use indexes, but using secondary indexes provides some flexibility in querying the data.

There are two types of secondary indexes:

- **Global Secondary Index** (**GSI**): It consists of a partition key and a sort key, which has to be different from the primary keys defined on the table
- **Local Secondary Index** (**LSI**): It uses the same partition key as of the table but uses a different sort key

DynamoDB allows you to create 5 GSI and 5 LSI on a table.

- Every index is associated with a table, called the base table for the index.
- DynamoDB automatically maintains the indexes. Whenever data is added, updated, or deleted from the base table, DynamoDB adds, updates, or deletes the corresponding item in the related indexes.
- While defining the index, you can choose what all attributes are copied to the index. At minimum, DynamoDB projects at least the key attributes from the table.

DynamoDB Streams

DynamoDB Streams provides an optional feature that can capture data modification events whenever a DynamoDB table is changed. The event data is captured in the steam in near real time in chronological order as the event occurs. Each of the events are recorded by a stream record.

When you enable a stream on a DynamoDB table, it writes a stream record as and when one of the following events occurs:

- When a new item is added to a table, the stream captures an image of the entire item including all of the item attributes
- When an item is updated, the stream captures the *before* and *after* values of all the attributes modified during the process
- When an item is deleted from the table, the stream captures an image of the entire item before it is deleted

A stream record consists of the name of the table, the timestamp when the event occurs, and other metadata. A stream record can last for 24 hours; after that it is automatically deleted from the stream.

You can also use AWS Lambda to create a trigger along with DynamoDb streams. A Lambda function can execute whenever a defined event occurs in a stream. Let us consider the `Employee` table used in the previous examples. When a new employee record is created, you want to send an email to all the employees in the department to welcome the new joinee. In such cases, you can enable a stream on the `Employee` table and then associate the stream with a pre-defined Lambda function, which sends an email to all the employees in a department where a new employee joins.

The Lambda function executes whenever there is a new record available in the stream; however, it processes only new items added to the employee table. The Lambda function invokes the Amazon **Simple Email Service** (**SES**) for sending emails to the users in the department. The following *Figure 11.5* illustrates this scenario:

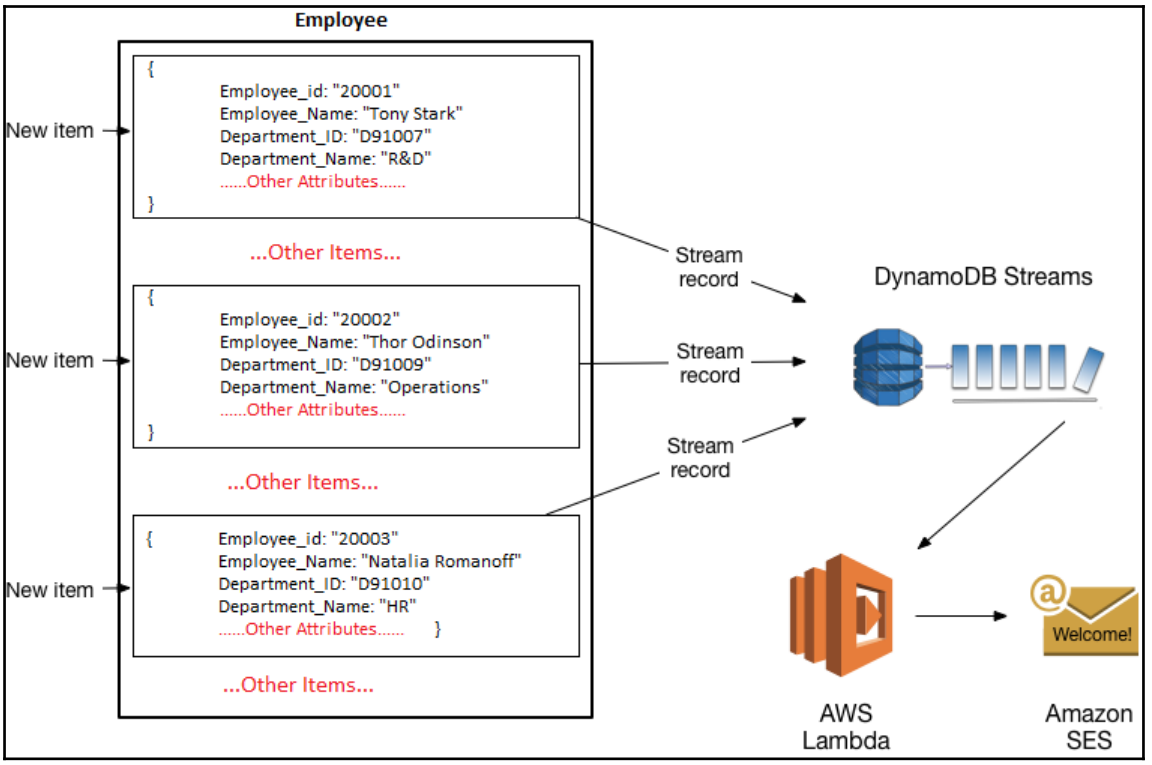


Figure 11.5: DynamoDB Streams with Lambda Function

Apart from triggers, DynamoDB streams can be used for data replication within a specific region or across multiple regions. They can also be used for materialized views of DynamoDB tables. A materialized view is a database object that contains the result of a query just like v*iews* in RDBMS terminology. DynamoDB stream can also be used for data analysis with Kinesis materialized views.

Read consistency model

Amazon provides DynamoDB in multiple AWS regions across the globe. All of these regions are independent and physically isolated from each other. If you create a DynamoDB table `Employee` in us-east-1 region and similarly create another table `Employee` in the us-west-2 region, these tables are two separate and isolated tables. An AWS region consists of multiple AZs. Each of the AZs are isolated from failures in any of the AZs in a region. Amazon provides an economical and low-latency network connection between all the AZs in a region.

Whenever you write data to a DynamoDB table, AWS replicates this data across multiple AZs to provide high availability. After writing data to a DynamoDB table, you get an `HTTP 200` response. `HTTP 200 (OK)` indicates that the data is safely updated to all the replicated copies stored in different AZs. AWS provides two types of read consistency models: eventually consistent read and strong consistent read. These models are based on the mechanism of how soon the data is replicated across the AZs.

Eventually consistent reads

While reading data from a DynamoDB table using with eventually consistent reads, the result you may get might not reflect any recently completed write operations. Since the data is stored in multiple AZs, it takes few seconds to synchronize the data in multiple location. Eventually, the consistency of data is achieved. In such cases, if you repeat your read operation after a short while, it returns the latest copy of the data.

Strong consistent reads

If you opt for a strongly consistent read on a DynamoDB table, it provides the response with the most up-to-date data. It reflects changes from all prior write operations which were successful. While working with strongly consistent reads, if there is any outage or delay in the network, the data may not be available immediately.

By default, DynamoDB uses eventually consistent reads. If you need to use strongly consistent reads, you need to specify as much while creating the table.

Read operations such as GetItem, Query and Scan provide a `ConsistentRead` parameter. When you set this parameter to True, DynamoDB uses strongly consistent reads.

Naming rules and data types

DynamoDB supports a number of data types. This section describes these data types as well as DynamoDB naming rules.

Let's start by understanding naming rules first.

Naming rules

Each table, attributes, and any other object in DynamoDB should have a names. All the names that you use in DynamoDB should be concise and meaningful. For example, a table can be named `Employee`, `Department`, `Books`, and so on. Just like these names, whatever name you use should be self-explanatory.

Here are some of the naming rules for DynamoDB:

- Names are case-sensitive and must be encoded in UTF-8
- A table name and an index name may range between 3 to 255 characters
- Names may contain only the following characters:
    - a-z
    - A-Z
    - 0-9
    - _ (underscore)
    - - (dash)
    - . (dot)
    - Attributes can have a number of characters between 1 to 255.

There are a number of special characters and reserved words in DynamoDB. These reserved words are given in the link `http://docs.aws.amazon.com/amazondynamodb/latest/developerguide/ReservedWords.html`.

Apart from these reserved words, the following characters have special meaning in DynamoDB:

- # (hash)
- : (colon)

Although, DynamoDB allows you to use this list of reserved names and special characters, it is recommended you not use them for naming purposes in DynamoDB.

Data types

DynamoDB supports a number of data types for attributes in a table. These data types can be categorized in three parts

- **Scalar types**: Scalar data type represents one value. It includes data types such as number, string, binary, boolean, and null.
- **Document types**: A document data type contains of a complex structure with nested attributes. Examples of such structures are JSON and XML. DynamoDB supports list and map document data types.
- **Set types**: A set data type contains a group of scalar values. The set types supported by DynamoDb are string set, number set, and binary set.

You need to specify names and respective data types for each of the primary key attributes while creating a DynamoDB table. In addition to that, each primary key, be it a partition key or sort key, must be defined as one of the following scalar data types:

- string
- number
- binary

DynamoDB is a schemaless NoSQL database. While creating a table, you need to define the primary key and its respective data types. Apart from that you do not need to define other attributes and their data types while creating a table. In contrast, while creating a table in RDBMS, you need to specifically define a schema with field names and its data types. DynamoDB has a flexible schema that allows you to directly store the data without defining the schema. You just need to define the primary key;the rest of the attributes can be taken care dynamically as you store the data in the table.

The following section describes each of these data types along with an example format.

Scalar data types

The scalar data type includes number, string, binary, boolean and null.

- **String**

Encoding	Unicode with UTF-8 binary encoding
Length	Greater than 0 and less than 400 KB
Partition key length	Maximum 2048 bytes

Sort key length	Maximum 1024 bytes
Usage	To represent string, alternatively date, and timestamp in string format

- **Number**

Can store	Positive, negative or zero
Precision	Maximum up to 38 digits
Positive range	1E-130 to 9.99999999999999999999999999999999999999E+125
Negative range	-9.99999999999999999999999999999999999999E+125 to -1E-130

- **Binary**

Can Store	Binary data such as images, compressed text, encrypted data
Length	Great than 0 and less than 400 KB
Binary attribute as partition key	Maximum length 2048 bytes
Binary attribute as sort key	Maximum length 1024 bytes
Supported encoding	Application must send the data in base64-encoded format

- **Boolean:** Boolean is also a scalar data type and it can store either *true* or *false* values
- **Null:** Null data type indicates an undefined or unknown state of the data

Document types

DynamoDB supports two document data types: **List** and **Map**. You can create a complex data structure by nesting these data types up to 32 levels deep.

DynamoDB does not limit on the number of values it can store in a list of a map data types; however, a table item must not exceed a total item size of 400 KB..

DynamoDB does not support empty scalar data types or set data types; however, it does support empty list and maps data types.

- **List:** A list data type can store a collection of values in square brackets [...]. You can compare the list data types with a JSON array. You can store any data types within the list and all elements in the list may or may not be of the same data type.
- Simple list example:
    - TechGiants: ["Amazon", "Apple", "Google", "Facebook"]
- Multiple data type list example:
    - DeckOfCards: ["Ace", 2, 3, 4, 5, 6, 7, 8, 9, 10, "Jack", "Queen", "King"]

- **Map:** A map data type attribute can consist of a collection of name-value pairs. This collection of values can be in any order. Map values are stored in curly braces { ... }. You can compare map with a JSON format. DynamoDB does not restrict you from storing any data type in a map element. You can also store multiple data types in an element. They do not need to be of the same size.
- Generally, maps are used for storing JSON documents in DynamoDB.

Example of map with a nested list:

```
{
employee_id: "D10007",
employee_name: "Tony Stark",
address: [
         home:
         {
               street: "10880, Malibu Point",
               city: "Malibu",
               state: "California",
               zip: 90265
         },
         office:
         {
               street: "890, Fifth Avenue, Manhattan",
               city: "New York",
               state: "New York",
               zip: 10019
         }

   ]
}
```

Set types

DynamoDB supports set data types that contains a group of scalar values. The set types supported by DynamoDB are string set, number set, and binary set.

When you use a set type, all the values within a set must be of the same data type. If you declare a set of type string, the set can contain only string values. If you declare an attribute of type number, the set can contain only number values within the set.

DynamoDB does not restrict you on the number of values within a set; however, the item containing the set data cannot exceed the DynamoDB item size limit of 400 KB.

The set must contain unique values. DynamoDB does not preserve the order of set values. Considering that, your application should not rely on any static element order in a set; also, it is important to note that DynamoDB does not support an empty set.

Examples of string, number and binary sets are as follows:

```
["Amazon", "Google", "Microsoft", "Facebook", "Apple"]
[-3.14159, 0, 1, 1.4142, 2, 4, 8, 16, 32]
["eNrLz0sFAAKRAUM=","eNorKc8HAAK8AVs=", "eNoryShKTQUABm4CGQ=="]
```

Creating a DynamoDB table – basic steps

After going through the core components of a DynamoDB table, it's time to create a table. This section describes the process of creating a DynamoDB table:

1. Log in to your AWS account and open the DynamoDB console at `https://console.aws.amazon.com/dynamodb/`.
2. Click on the **Create Table** button.
3. In the C**reate DynamoDB table** dialog, you can choose various options as given in the following *Figure 11.6*:

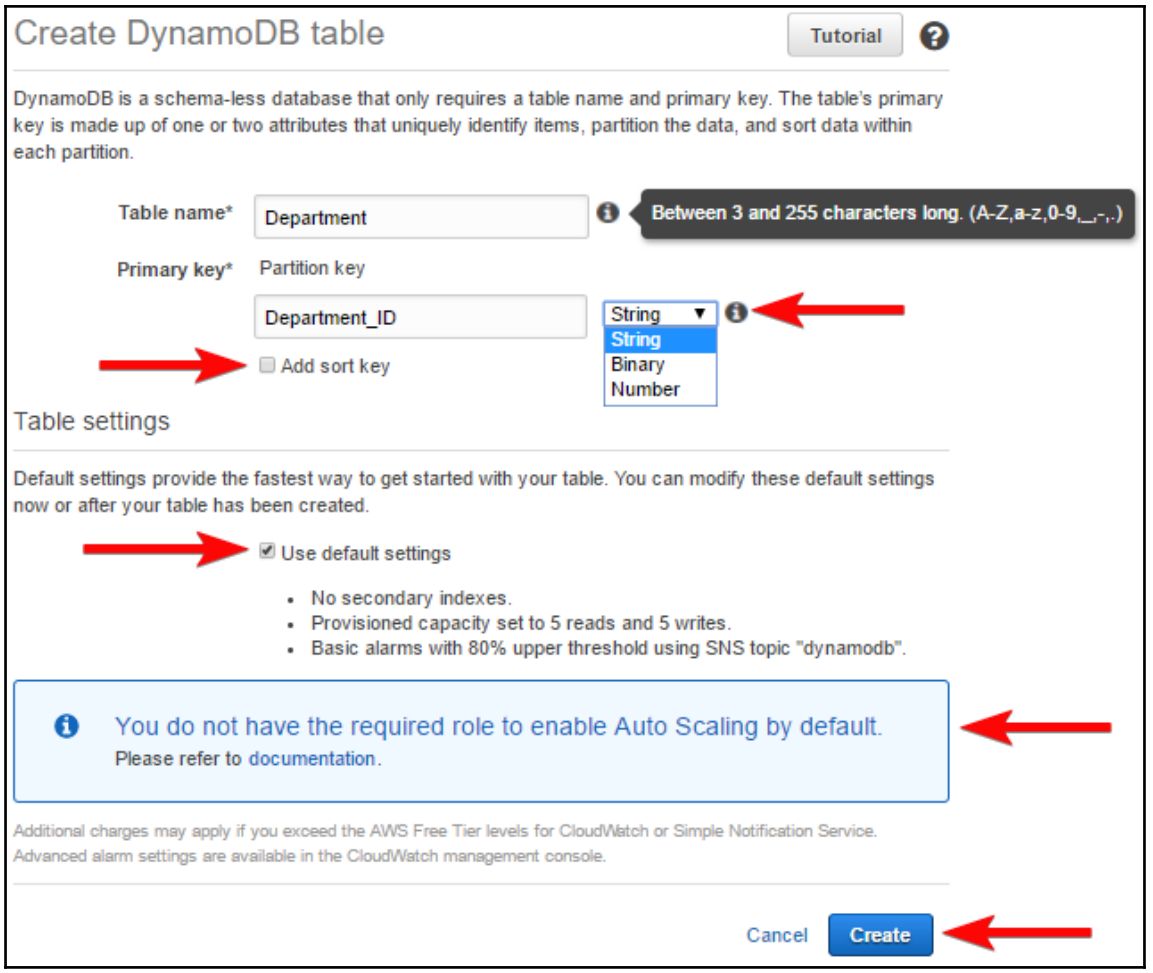


Figure 11.6: Create DynamoDB table

1. Enter the desired table name as shown in the preceding screen. **Table name** can be between 3 to 255 characters long.

2. Enter the partition key name and select the key type from the drop-down menu. Any primary key can be only a scalar data type that is **String**, **Binary** or **Number**.

3. Check **Add sort key** if required and enter the sort key name and data type. This is required only if you want to create a composite primary key with a combination of a *partition key* and a *sort key*.

4. With the default settings, Amazon creates a table without any secondary indexes and provisions a 5 read with 5 write capacity. It also creates a **Basic alarms with 80% upper threshold using SNS topic "dynamodb"**. You can customize these settings by unchecking the **Use default settings** checkbox.

5. As you can see in the preceding *Figure 11.6*, there is a warning saying that **You do not have the required role to enable Auto Scaling by default**. This warning comes only if you do not have any such role in your IAM roles. You can safely ignore this, as DynamoDB creates a new role when you create a table with default settings. In the advanced settings, there is an option to create a new Auto Scaling role for DynamoDB or to use an existing one. This is explained in the subsequent points.

4. Click on **Create**, after choosing the appropriate options.

Now, let us understand some of the advanced settings in the table creation process.

Adding a sort key while creating a DynamoDB table

You can add a sort key while creating a table by selecting the **Add sort key** checkbox as follow:

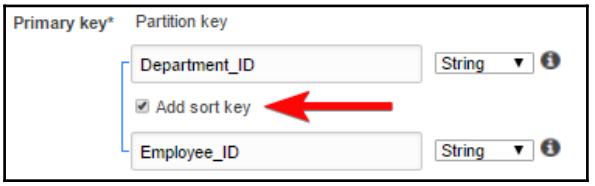


Figure 11.7: Adding a sort key

Using advanced settings while creating a DynamoDB table

With the default setting, Amazon creates a table without any secondary indexes and provisions a 5 read with 5 write capacity. It also creates a B**asic alarm with 80% upper threshold using the SNS topic "dynamodb"**. You can customize these settings by unchecking the **Use default settings** checkbox. Let us understand the options in advanced settings.

On the create table screen uncheck **Use default settings**. It displays the screen shown as follow:

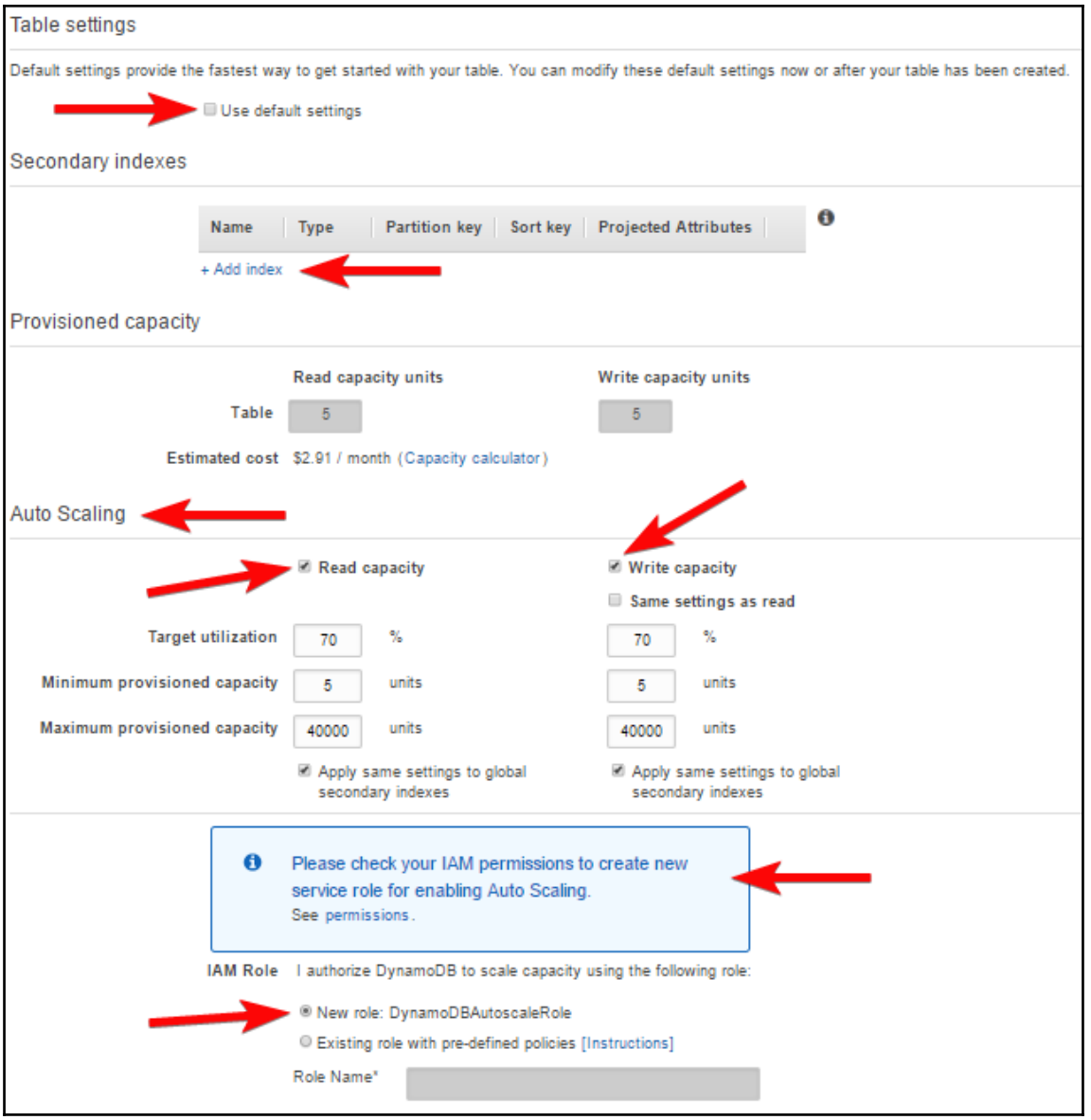


Figure 11.8: Create table screen - advanced settings

As you can see in the advanced settings, there is a provision to create secondary indexes and set up Auto Scaling and also an option to create a **New role: DynamoDBAutoScaleRole** or to choose an existing role for Auto Scaling permissions. Let us understand each of these options one by one.

Creating secondary indexes – table settings

The create table screen allows you to create secondary indexes while creating a table. For creating secondary indexes, click on **Add index** in the **Table settings** screen, as shown in the previous *Figure 11.8*.

Clicking on **Add index** brings up the following *Figure 11.9* with further options:

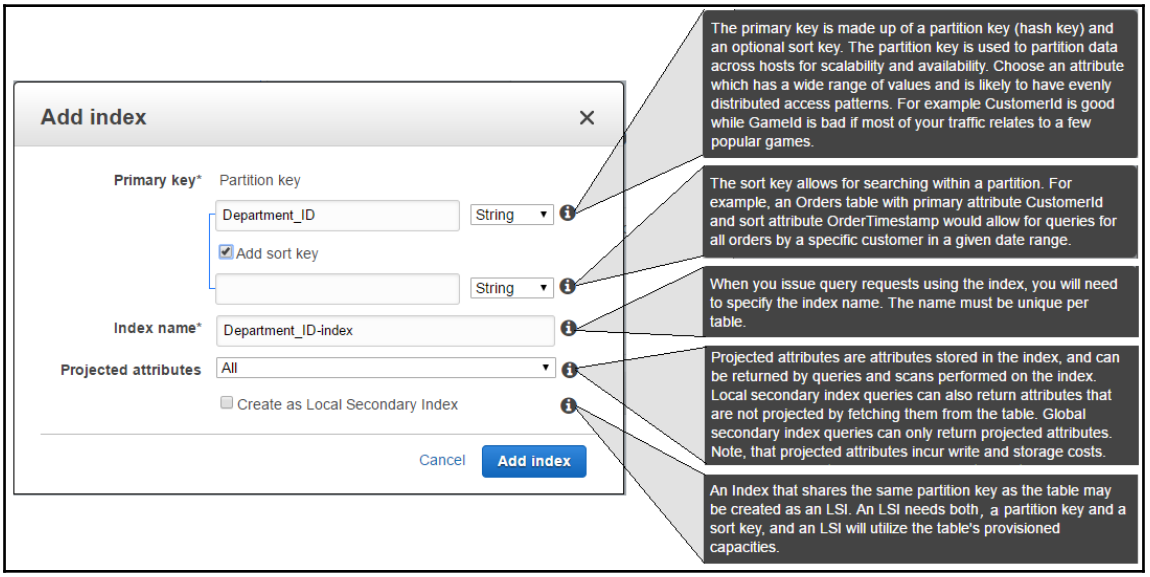


Figure 11.9: Create secondary index

As shown in the preceding *Figure 11.9*, you can enter the **Partition key** name and optionally a sort key name based on the requirement. Index name is automatically populated with the **Partition key** name and the `-index` suffix. You can change the index name, if required. The index name must be unique per table.

Projected attributes are attributes which are stored in the index. While creating an index, you can specify what all attributes you want to add to the index. You can select these projected attributes from the drop-down box. There are three options in the drop-down box. You can either choose **All** attributes or you can choose **Keys only** or you can select **Include** and add specific attributes that you want to add to the index. Whatever attributes you choose here, are returned by query and scan performed on the index.

Provisioned capacity – table settings

If you choose to customize the default settings, you can configure the **Provisioned capacity** for the table. Provisioned capacity settings are disabled if Auto Scaling is enabled. For a custom read or write capacity, you need to disable the **Read capacity** and **Write capacity** checkbox. After disabling the Auto scaling, you can choose the specific **Read capacity** units and **Write capacity** units. You can refer to *Using advanced settings while creating a DynamoDB Table* section. Table read and write capacity is automatically taken care if you enable Auto Scaling on the table.

Auto Scaling – table settings

You can configure the Auto Scaling options from **Table settings**. You can selectively choose Auto Scaling for Read Capacity and/or Write Capacity as per your needs. There are three options you need to configure in order to use Auto Scaling shown as follow:

- **Target utilization**: Default value for **Target utilization** is `70`%. You can change this value based on the need. DynamoDB automatically scales up the read/write capacity of the table , in case the utilization goes beyond the target utilization percentage configured here. The table capacity scales up to **Maximum provisioned capacity** configured. You can individually specify **Target utilization** thresholds for read as well as write operations.
- **Minimum provisioned capacity:** While creating the table, you can specify the **Minimum provisioned capacity** for a table. Irrespective of utilization, DynamoDB provisions the minimum read and/or write capacity as configured here. DynamoDB does not scale itself down beyond **Minimum provisioned capacity**. You can separately specify the **Minimum provisioned capacity** for read as well as write operations.

- **Maximum provisioned capacity:** While using Auto Scaling on a table, you can configure the maximum scaling capacity of a table. DynamoDB does not scale beyond this capacity during Auto Scaling events. You can separately specify the **Maximum provisioned capacity** for read as well as write operations:

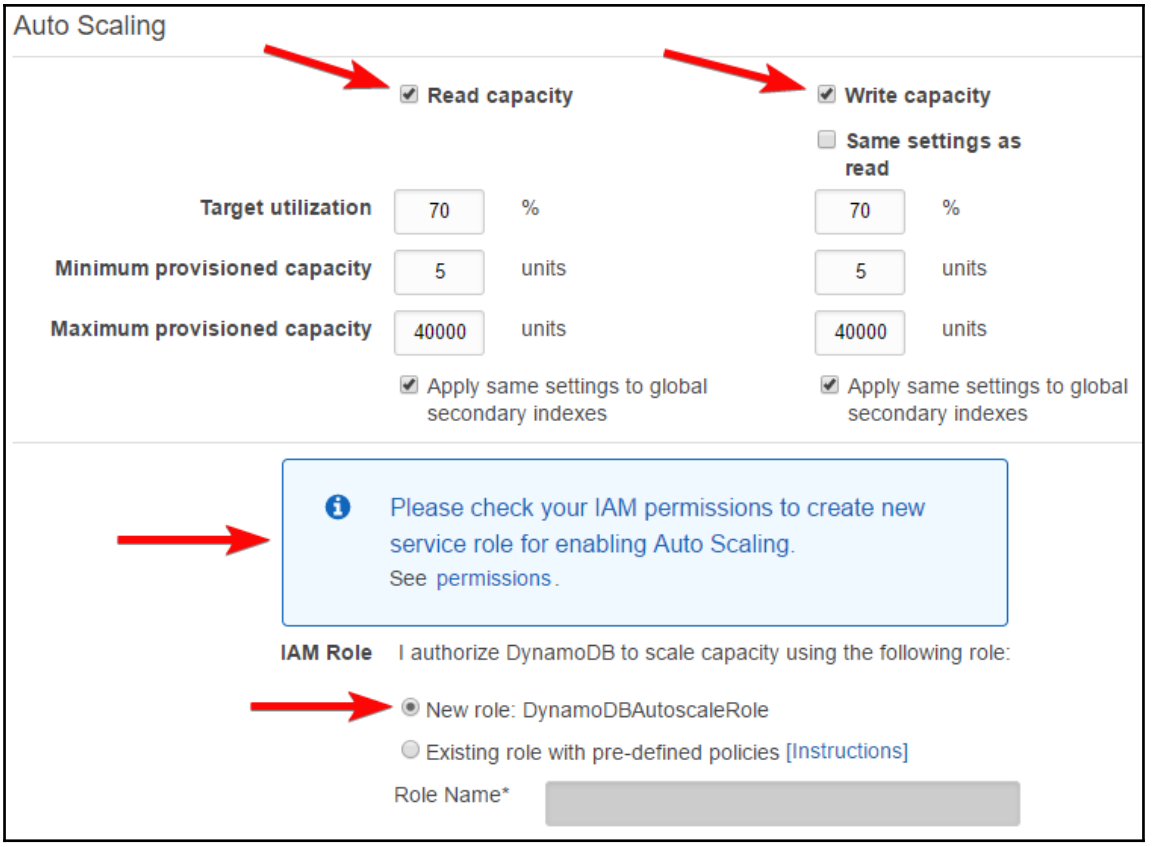


Figure 11.10: Create table -Auto Scaling settings

You can enable or disable Auto Scaling for **Read capacity** and **Write capacity** separately as displayed in the preceding *Figure 11.10*. Unchecking **Read Capacity** disables Auto Scaling for read capacity. Similarly, unchecking the **Write Capacity** disables Auto Scaling for write capacity.

For Auto Scaling the table capacity, DynamoDB requires permission. As shown in the previous snapshot, you need to either select **New role: DynamoDBAutoscaleRole** or choose E**xisting role with pre-defined policies**. If you select **Existing role with pre-defined policies**, you need to specify an existing role name, which carries sufficient permissions for Auto Scaling the DynamoDB table.

For more details on creating a role for DynamoDB Auto Scaling, you can refer to the URL: `http://docs.aws.amazon.com/amazondynamodb/latest/developerguide/AutoScaling.CLI.html#AutoScaling.CLI.CreateServiceRole`.

Methods of accessing DynamoDB

Amazon provides a DynamoDB console, CLI and API interface to access DynamoDB resources.

Let's understand each of these interfaces.

DynamoDB console

Amazon provides a DynamoDB AWS Management Console. You can access the console on the URL: `https://console.aws.amazon.com/dynamodb/home`.

Here's what you can do with the DynamoDB AWS Management Console:

- You can access the DynamoDB dashboard for monitoring recent alerts, the total provisioned capacity of tables, health of the service, and latest news on DynamoDB.
- The console allows you to create, update, and delete tables. It also provides a capacity calculator that can help you estimate the capacity units you may need based on the information you provide.
- You can manage DynamoDB Streams.
- The console also helps you to see items stored in a table, add new items, update existing items, and delete items.
- You can also manage **Time To Live** (**TTL**) for items stored in a table. TTL is defined for automatically deleting an item from the table when it expires.
- Console also helps you to query items in a table and perform a scan on a table.

- The interface also helps you to create and view alarms for monitoring a table's capacity usage.
- You can view in real time a table's top monitoring metrics graph, directly from CloudWatch onto the DynamicDB console.
- Console provides you an interface to change the provisioned capacity of a table.
- There is a provision in the console to create and delete GSIs.
- It enables you to create triggers that can connect DynamoDB streams to a Lambda function.
- You can also add tags to your DynamoDB resources for better identification and categorization of resources.
- Using Console, you can also purchase reserved capacity.

The console also provides a number of navigation tabs. The following list provides a quick reference to each of the tabs available on the console:

Navigation tab	Description
Overview	Displays table details and manage Streams and TTL
Items	Manages table items and executes queries and scans against table and indexes
Metrics	Views and monitors CloudWatch metrics related to DynamoDB
Alarms	Views and manages CloudWatch alarms
Capacity	Views and updates provisioned capacity of a table
Indexes	Views and manages GSIs
Triggers	Manages triggers that can connect DynamoDB streams to a Lambda function
Access control	Manages fine-grained access control and configures web identity federation
Tags	Adds tags to resources for better identification and categorization of resources

DynamoDB CLI

AWS provides CLI to manage AWS services using the command line. You can use CLI for a number of purposes such as creating a script to automate a task or creating a table and performing many other DynamoDB tasks using utility scripts.

If you have not already setup AWS CLI, you can follow the instructions for downloading and configuring it at the URL: `http://aws.amazon.com/cli.`

Working with DynamoDB CLI is very simple if you understand the basics right. Here's the syntax for working with DynamoDB CLI:

```
aws dynamodb <operation-name> <--parameters-name> <parameter-value> ... <--
parameters-name> <parameter-value>
```

The following command creates a DynamoDB table named `employee` with the attributes `Employee_ID` and `Employee_Name`. It also creates a partition key on the attribute `Employee_ID`:

```
aws dynamodb create-table \
   --table-name employee \
   --attribute-definitions \
         AttributeName=Employee_ID,AttributeType=S \
         AttributeName=Employee_Name,AttributeType=S \
   --key-schema AttributeName=Employee_ID, KeyType=HASH
   --provisioned-throughput ReadCapacityUnits=1,WriteCapacityUnits=1
```

Similarly, the following commands add new items to the `employee` table:

```
aws dynamodb put-item \
--table-name employee \
--item \
    '{"Employee_ID": {"S": "10001"}, "Employee_Name": {"S": "Vipul
Tankariya"}, "Country": {"S": "India"}}' \
--return-consumed-capacity TOTAL

aws dynamodb put-item \
--table-name employee \
--item \
    '{"Employee_ID": {"S": "10002"}, "Employee_Name": {"S": "Bhavin
Parmar"}, "Country": {"S": "India"}}' \
--return-consumed-capacity TOTAL
aws dynamodb put-item \
    --table-name employee \
    --item '{ \
        "Employee_ID": {"S": "10003"}, \
        "Employee_Name": {"S": "Gajanan Changadkar"}, \
        "Country": {"S": "India"} }' \
    --return-consumed-capacity TOTAL
```

Sometimes the complex JSON format may create problems on the command line. AWS provides a way to handle this format using a file as an argument on the command line. The following example shows how you can run CLI with JSON file arguments.

Let us assume that all the command line arguments are stored in a JSON file called `condition-file.json` for creating items in the employee table:

```
aws dynamodb query --table-name employee --key-conditions file://condition-file.json
```

AWS provides a number of commands to work with DynamoDB.

You can refer to the URL `http://docs.aws.amazon.com/cli/latest/reference/dynamodb/index.html` for more of such commands.

Working with API

Apart from the AWS Management Console and CLI, AWS also provides APIs to work with DynamoDB. These APIs can be used to develop applications that can manage various DynamoDB operations. For using APIs, you need to install AWS SDKs. AWS provides SDKs for a number of programming languages such as Java, JavaScript in the browser, .Net, Node.js, PHP, Python, Ruby, C++, Go, Android, and iOS.

The following table describes where you can start working with these SDKs:

Language	**Reference URL**
Java	`https://aws.amazon.com/sdk-for-java`
JavaScript in the browser	`https://aws.amazon.com/sdk-for-browser`
+.Net	`https://aws.amazon.com/sdk-for-net`
Node.js	`https://aws.amazon.com/sdk-for-node-js`
PHP	`https://aws.amazon.com/sdk-for-php`
Python	`https://aws.amazon.com/sdk-for-python`
Ruby	`https://aws.amazon.com/sdk-for-ruby`
C++	`https://aws.amazon.com/sdk-for-cpp`
Go	`https://aws.amazon.com/sdk-for-go`
Android	`https://aws.amazon.com/mobile/sdk/`
iOS	`https://aws.amazon.com/mobile/sdk/`

DynamoDB provisioned throughput

DynamoDB provides the Auto Scaling feature for automatically scaling the read and write capacity of a table; however, if you do not use it, you need to manually handle the throughput requirement of your table. DynamoDB measures the throughput capacity using read and write capacity units.

Read capacity units

DynamoDB processes the read operations based on the type of read consistency used. Using one read capcity unit, DynamoDb can process one strongly consistent read per second. In the same line, DynamoDB can process two eventual consistent reads per second using one read capacity unit. Using one read capacity unit, DynamoDB can process an item of up to 4 KB in size. If the item size is more than 4 KB, it requires an additional read capacity unit to process it. In short, item size and consistency model determines the total number of read capacity units required to process it.

- 1 read capacity unit = 1 strongly consistent read
- 1 read capacity unit = 2 eventual consistent reads
- 1 read operation can process an item of up to 4 KB in size
- If an item is more than 4 KB in size, it requires additional read operations
- If an item is less than 4 KB in size, it still requires 1 read capacity units

Write capacity units

Using one write capcity unit, DynamoDb can process one write per second and write an item of maximum 1 KB in size. If the size of the item is larger than 1 KB, it requires additional write capacity units. In short, the number of write capacity units required to process an item depends up on the size of the item.

- 1 write capacity unit = 1 write operation of up to 1 KB in size
- If an item is larger than 1 KB, it requires additional write capacity units
- If an item is less than 1 KB in size, it still requires 1 write capacity unit

Calculating table throughput

If you create a table with a throughput of 5 read capacity units and 5 write capacity units:

- It can perform a strongly consistent read of up to 20 KB per second
    - 5 read capacity units x 4 KB

- It can perform an eventual consistent read of up to 40 KB
    - 5 read capacity units x 4 KB x 2
- It can write 5 KB per second
    - 5 write capacity units x 1 KB
- When you manually configure the throughput of a table, it determines the highest amount of capacity an application can utilize from a table or associated index. If your application consumes more throughput than configured in the provisioned throughput settings, application requests start throttling. This can either crash the application or give a lackluster performance.
- Let's consider a couple of examples to understand the throughput calculation:

Example 1: You have an application that requires reading 15 items per second. Each item is of 3 KB in size. If the application requires strongly consistent reads, what read capacity is required to address this need?

Explanation:

One read capacity unit can process one strongly consistent read of up to 4 KB item in size. If the item size is less than 4 KB, it still requires one read capacity to process the read operation.

Let us formulate this understanding:

Read throughput = (Item size rounded-up in multiples of 4KB) / 4 KB x number of items

Read throughput = 4 / 4 x 15 (Item size is 3 KB, which is rounded-up to 4 KB)

Read throughput = 1 x 15

Read throughput = 15

The answer is 15 read capacity units.

In the same example if the requirement changes to an eventual consistent read, then you just need to divide the result by 2 as one read capacity unit can process two eventual consistent reads.

Example 2: You have an application that requires reading 80 items per second. Each item is 5 KB in size. If the application requires using strongly consistent read, what read capacity is required to address this need?

In this example, the item size is 5 KB, which is greater than 4 KB. We need to round it up to a multiple of 4 KB, which is 8 KB in this case. Remember, if an item is more than 4 KB in size, it requires additional read capacity. This is the reason why we need to always use multiples of 4 while calculating the read throughput.

Let's add the values in the formula for this example:

Read throughput = (Item size rounded-up in multiples of 4KB) / 4 KB x number of items

Read throughput = 8 / 4 x 80

= 2 x 80

= 160

The answer is 160 read capacity units.

Similarly, if the requirement changes from strong consistent read to eventual consistent read, you need to divide the answer by 2.

Example 3: You have an application that requires reading 60 items per second. Each item is 3 KB in size and the application requires using eventual consistent read. What read capacity is required to address this need?

Here's the formula for calculating read capacity for eventual consistency:

Read throughput = ((Item size rounded-up in multiples of 4 KB) / 4 KB x number of items) / 2

= (4 / 4 x 60) / 2

= (1 x 60) / 2

= 60/2

= 30

The answer is 30 read capacity units.

Example 4: You have an application that writes 10 items per second with each item being 8 KB in size. How many write capacity units are required to address this need?

Remember, 1 write capacity unit = 1 write operation of up to 1 KB in size. Here's the formula that can help us in calculating the required write throughput

Write Throughput = (Item size rounded-up in multiples of 1KB) / 1KB x number of items

= Item size rounded-up in multiple of 1KB x number of items

= 8 x 10

= 80

The answer is 80 write capacity units.

DynamoDB partitions and data distribution

Table partitioning is a mechanism to segregate a large table into smaller, more manageable parts without creating a separate table for each part. A partitioned table physically stores data in groups of rows. These groups of rows are called partitions. You can separately access and maintain each partition.

DynamoDB also manages data in partitions. DynamoDB uses SSDs for storing data and automatically replicates data across multiple AZs in an AWS region. DynamoDB automatically manages partitions; you as a consumer do not need to manage the partitions.

While creating a table, DynamoDB allocates a sufficient number of partitions to the new table such that it can handle any provisioned throughput needs. However, DynamoDB can allocate additional partitions to a table in certain situations. The following are the scenarios when DynamoDB allocates additional partitions:

- In case a table's provisioned throughput goes beyond what the existing partitions can handle
- In case an existing partition consumes allocated storage space and more storage space is required

Partition management tasks are performed automatically in the background without affecting the provisioned throughput of a table. It is also important to note that GSIs are also segregated in partitions. The data in an index is stored separately from the base data of a table. Index partitions and table partitions act in a similar manner in DynamoDB.

Data distribution – partition key

As we have seen earlier in this chapter, DynamoDB allows you to create a primary key either with a single attribute partition key or with a composite key comprising of a combination of a partition key and sort key. If we create a table with only the partition key, each item in the table is retrieved based on the partition key value.

DynamoDB uses an internal hash function for writing an item to the table. The partition key value acts an input to the hash function. Based on the partition key value, the hash function determines the target partition for storing the item.

It is required to provide the partition key value for reading an item from the table. This value is used in the hash function to locate the partition where the item can be found.

Let us consider an example to understand how the items are stored in a partition. The following *Figure 11.11* describes the details of a `Cars` table. The table spans in multiple partitions. The primary key for the table is `CarType`. For simplifying the concept, only the primary key attribute is included in the *Figure 11.11*. While storing data in a table, DynamoDB uses an internal hash function to determine the target partition for storing a new item. In this example, the hash function determines the target partition based on the hash value of `CarType`, which is `Sedan`. Here, it is important to understand that the items are not stored in a sorted order. The location of each item is determined based on the value of the partition key:

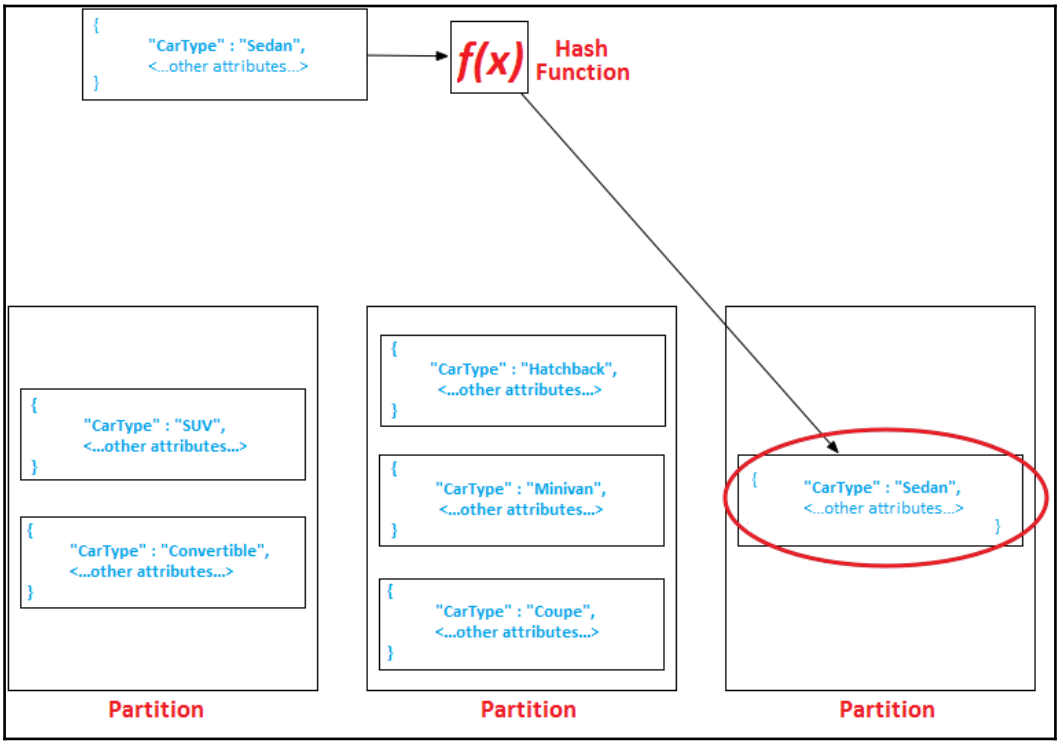


Figure 11.11: Partitioning and hash function

The preceding example with `CarType` is given for simplifying the data distribution concept in DynamoDB partition. While designing a DynamoDB table, you should choose a partition key that has a large number of distinct values. If the partition key values are similar, data may end up getting stored in some specific partitions and may not give optimum performance.

Let's understand this scenario with an example:

If you're creating a Vendor table, `Vendor_ID` is a good candidate for the partition key, however, you need to ensure that the values in the `Vendor_ID` are distinct.

If you choose values in Vendor_ID as `V0001`, `V0002`, `V0003`, and so on, it works and items are distributed in multiple partitions depending up on the table size. However, the best way to optimize partitioning is to use more distinct values such as `FMG001`, `ITV001`, `SRV001`. In this example, `Vendor_ID` includes the vendor type like **FMG** for **Fast-moving consumer goods** vendor, **ITV** for **IT vendor**, **SRV** for **service vendor,** and so on. Such approaches create more distinct values in the table and distributes data optimally in more partitions.

Data Distribution – partition key and sort key

When you use the composite key in a DynamoDB table, which includes the partition key and the sort key, the approach for calculating the hash value remains the same. In addition to keeping partition key values physically close to each other, DynamoDB orders the data by the sort key value.

As described in the previous section, while storing an item in a table, DynamoDB uses the partition key value. The partition key value is supplied to the hash function, which determines the target partition for storing the item. The target partition may already have a number of items. In such scenarios, DynamoDB stores the item in the ascending order of sort key values in the table.

For reading an item from the table, you need to supply the partition key value as well as sort key value. DynamoDB uses the partition key value to determine the source partition where the item can be found. DynamoDB even allows you to read multiple items from a table using a single query. However for reading multiple items in a single query requires that the items must have the same partition key value. While querying the table you can optionally apply a condition to the sort key such that the times within a specific range value is returned.

DynamoDB global and LSI

DynamoDB provides primary keys for quickly accessing items in a table by supplying primary key values in a query. The primary key is useful and it can certainly speed up data retrieval from the table; however, in certain scenarios, applications can take advantage of secondary indexes. Secondary indexes can speed up item retrieval from a table based on any other attributes than the primary key. For fulfilling such requirements, you need to create secondary indexes on the DynamoDB table. Once a secondary index is created on an attribute, you can use a `Query` or `Scan` request on specific indexes to retrieve items.

Secondary index refers to a data structure which is made up of a subset of attributes in a table. The main purpose of a secondary index is to provide an alternate key for query operations. You can use secondary indexes to read data using a query, in the same manner as you query a DynamoDB table. DynamoDB allows you to create multiple secondary indexes, which can help you to access the data using different query patterns.

A secondary index is linked with a table, which becomes the source of data for it. The source table from which an index takes data, is also called as the base table for the index. While defining an index, you need to define an alternate key, which can be a partition key and sort key. You can also choose what all attributes you want to associate with the index. You can choose, all attributes, primary keys or you can choose a specific set of attributes from the table. DynamoDB copies all the attributes you choose, into the index along with primary key attributes. Once the index is created, you can query or scan the index in the same way as you query a table.

DynamoDB automatically maintains secondary indexes. When you change anything on the base table, the change automatically reflects into indexes. If you add, modify, or delete any item in the table, DynamoDB automatically updates this change in the index.

There are two types of secondary indexes:

- **GSI**: It is an index, which can have a different partition key and sort key as compared to the base table. With GSI, you can query data spanned across all the partitions in a base table. In short, when any query is executed against a GSI, its scope spans across all the partitions in a table. This is the reason it is called **Global Secondary Index** (**GSI**) .

- **LSI**: It is an index which has the same partition key as its base table, however, it has a different sort key. Every partition in a LSI is mapped with a base table partition, which carries the same partition key value. In short, LSIs scope is associated with its base table partition and this is the reason it is called **Local Secondary Index** (**LSI**).

Difference between GSI and LSI

Characteristic	Global secondary index	Local secondary index
Key schema	Primary key of a GSI can be a single attribute key which is called a partition key or, it can be a composite key with two attributes, the partition key and the sort key.	Primary key of a LSI has to be a composite key, a combination of the partition key and sort key.
Key attributes	Partition key and sort key of the index, can be any attributes from the base table.	Partition key of the index must be the same as the base table partition key. Sort key can be any attribute from the base table.
Size restrictions per partition key value	There is no restriction on the size of the index.	Total size of all items for a specific partition key value must be less than or equal to 10 GB in size.
Online index operations	GSI can be created while creating a table. DynamoDB also allows you to add additional GSIs to an existing table or delete an existing index as and when required.	LSIs can be created while creating a table, however, DynamoDB does not allow you to add additional LSIs to an existing table. It does not even allow you to delete any existing LSI.
Queries and partitions	You can query the entire table stored across all the partitions in a GSI.	You can query only a single partition with a partition key value specified in the query.
Read consistency	GSI supports only eventual consistency reads.	LSI supports eventual consistency or strong consistency reads based on your requirement.

Provisioned throughput consumption	You need to configure a separate provisioned throughput for GSI. You need to explicitly specify the read and write capacity units while creating a GSI. When you query a GSI, it uses read and write capacity units on top of base table consumption.	LSI does not have their own provisioned throughput. It utilizes read and write capacity units from the base table when you query or scan a table or write data to base table.
Projected attributes	While executing a query or scan or a write request against a GSI, you can specify only the attributes, which are projected while creating the index. It cannot handle any request for attributes which are not defined as part of the index.	While executing a query or scan or a write request against a LSI, you can specify any attributes from the table, even if these attributes are not projected while creating the index. When you query any additional attributes, which are not part of the index, DynamoDB automatically gets these attributes from the base table.

DynamoDB query

DynamoDB query is a mechanism to request items from a table. Using queries, you can request data from a table or any secondary index that has a composite primary key. While querying a table or index, you have to provide the name of the partition key attribute and a value for the same. The query returns all the items with that partition key value. You can also select a sort key attribute and filter the search result using any of the comparison operators.

For working with Query in GUI, you need to go to the DynamoDB dashboard in the URL: `https://console.aws.amazon.com/dynamodb`. From the dashboard, you can select **Tables** -> **Movies -> Items** tab. In this example, we are working with the **Movies** table. You need to select the specific table name that you want to work with on your dashboard.

In the following *Figure 11.12* , you can see the Query window for the **Movies** table, which has **year** as the partition key and **title** as the sort key:

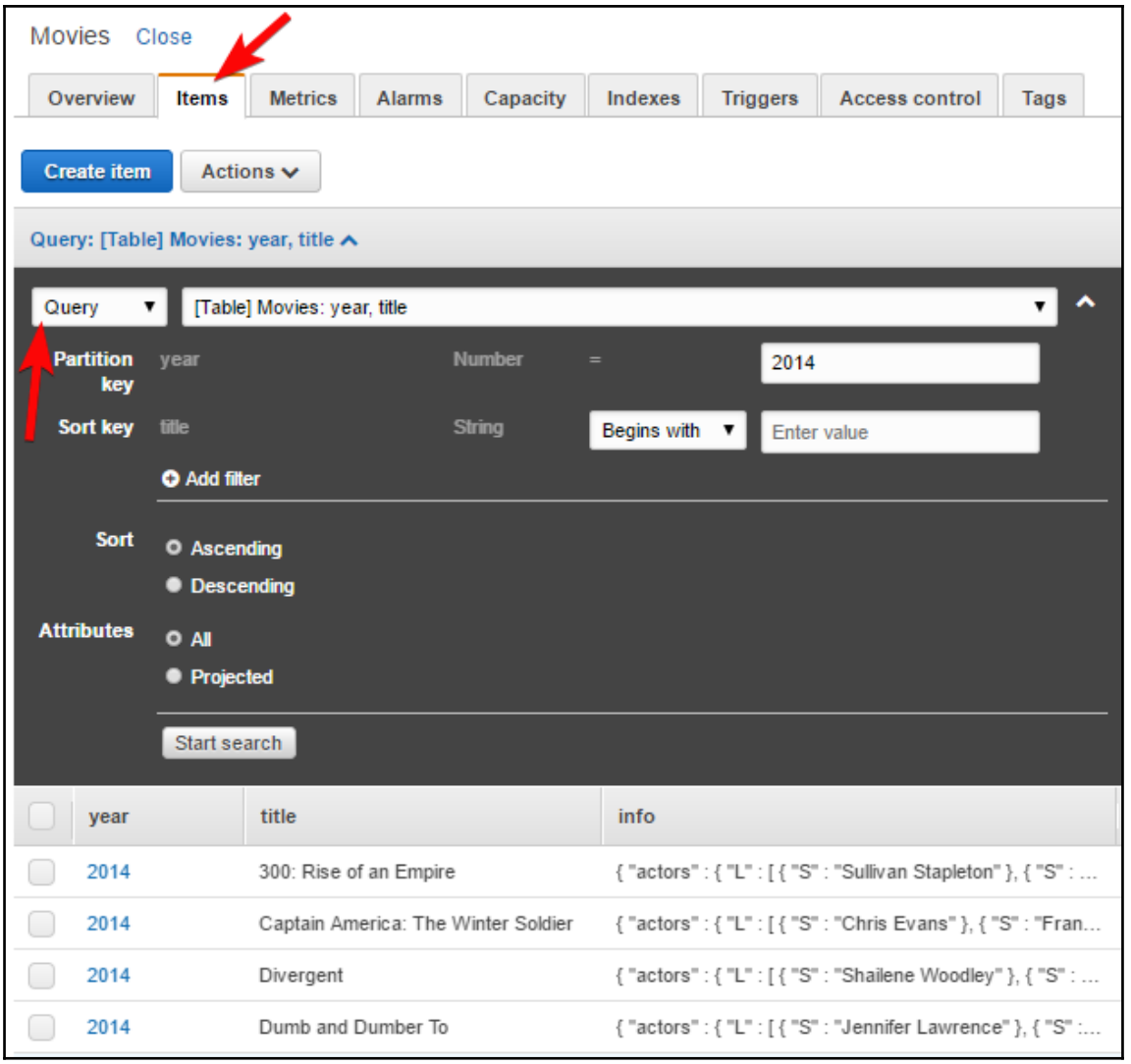


Figure 11.12: DynamoDB query

Do remember to select **Query** from the dropdown box as shown in the preceding figure. By default, the item window is loaded with the **Scan** option as selected instead of **Query**.

For querying the table, you need to provide the partition key value and optionally provide the sort key value. In this example, the partition key is the **year** and **2014** is given as the value. You can click on the **Start search** button after entering the required values. As you can see in the preceding *Figure 11.12*, the table displays all items with the partition key value as **2014**.

For sort keys and filters, you can use a number of key condition expressions. These expressions are described in the following table:

Key condition exptressions

Operator	Example	Description
=	a = b	true if the attribute a is equal to the value b
<	a < b	true if a is less than b
<=	a <= b	true if a is less than or equal to b
>	a > b	true if a is greater than b
>=	a >= b	true if a is greater than or equal to b
`Between`	a BETWEEN b AND c	true if a is greater than or equal to b, and less than or equal to c
`Begins with`	begins_with (a, substr)	true if a begins with a specified substring

While working with Query, you can sort the output in ascending or descending order. You can opt to display all the attributes of the table or you can choose to project specific attributes from the table.

Query with AWS CLI

Here are some examples of using DynamoDB Query with AWS CLI:

```
aws dynamodb query \
    --table-name Items \
    --key-condition-expression "Item_ID = :id" \
    --expression-attribute-values  '{":id":{"S":"i10001"}}'
```

The query retrieves all records for `Item_ID` `i10001` from the table `Items`. As you can see in the preceding example, for initiating a query request to DynamoDB, you need to use the `aws dynamodb query` command on AWS CLI. `aws dynamodb query` requires some parameters. Let's understand the query with the following table:

Command/Expression	**Description**
`aws dynamodb query`	AWS CLI command to initiate a query request to DynamoDB.
`\`	Backward slash `\` is used for indicating continuation of code in next line.
`--table-name`	Indicates the name of the DynamoDB table.
`--key-condition-expression`	Describes the key condition. In this example it indicates the attribute `Item_ID` with `id` as the expression attribute. Expression attributes can be any string to describe the expression. It is different from table attributes. It is a custom name given to an expression. The value of the custom expression is declared in `--expression-attribute-values` as described in the example.
`--expression-attribute-values`	Describes the value for the key-condition-expression attribute defined in `--key-condition-expression`. In this example, a custom expression attribute `id` is declared. Here you need to declare the value type and the value for the expression attribute. In this example `id` is of type `S` (String) with the value `i10001`. For the numeric expression attribute, you need to specify `N` instead of `S`.

If you need to deep dive into Queries, you can take a look at the URL. `http://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Query.html`.

DynamoDB Scan

When you use Query on DynamoDB, it uses only primary key attribute values to perform a search on the table. You can further refine the result by using filters on attributes other than primary keys. Unlike a Query, the Scan operation can perform a search on any attribute of the table. It also allows you to refine the search by applying filters to the scan result.

When you perform a Scan operation on a table, it reads all the items in the table or indexes and by default returns all the items and attributes. If you do not want to retrieve all the attributes, you can use the `ProjectionExpression` parameter to retrieve only specific attributes.

Irrespective of whether the items are found with the matching criteria or not, Scan always returns a result set. If items with the specified criteria are found, it returns the result set with the items else it returns an empty result set.

Every Scan operation can retrieve a maximum of 1 MB data. You can apply a filter to the scan for further narrowing down the result based on the filter conditions.

Example: The following AWS CLI example scans the `Movies` table and returns only the items with `ReleaseYear` as `2017`:

```
aws dynamodb scan \
     --table-name Movies \
     --filter-expression "ReleaseYear = :name" \
     --expression-attribute-values '{":name":{"N":"2017"}}
```

Reading an item from a DynamoDB table

You can use the GetItem operation for reading an item from a DynamoDB table. While performing a GetItem operation, you need to provide a table name and the primary key of the table item.

Example: Following example shows how to read an item from the employee table using AWS CLI. In this example, `employee_id` is the primary key of the table:

```
aws dynamodb get-item \
   --table-name employee \
   --key '{"employee_id":{"S":"E10001"}}'
```

For reading an item from the table, it is necessary to specify the entire primary key. If a table has a composite key, you need to specify the partition key as well as the sort key in `get-item`. It performs an eventual consistent read by default; however, you can use a strongly consistent read by using the `ConsistentRead` parameter. Also by default, `get-item` returns all the attributes of a table. If you want to return specific attributes of the table, you can use the project expression parameter. You can also set the `ReturnConsumedCapacity` parameter to `TOTAL` for returning the number of read capacity used by the `get-item` operation.

Example:

```
aws dynamodb get-item \
    --table-name employee \
    --key '{"employee_id":{"S":"E10001"}}' \
    --consistent-read \
    --projection-expression "FirstName, LastName, JoiningDate, Gender,
DateOfBirth" \
    --return-consumed-capacity TOTAL
```

Writing an item to a DynamoDB table

DynamoDB provides the following operations for creating, updating, and deleting an item from a table:

- PutItem
- UpdateItem
- DeleteItem

For performing either of these operations, you need to specify the complete primary key. If the table has just a partition key, you can provide the partition key. If the table has a composite key, you need to provide both, the partition key as well as the sort key. Providing just the partition key or the sort key alone does not work for these operations. You need to specify both the keys for composite key table.

If you want the operation to return write capacity consumed by the operation, you can set `ReturnConsumedCapacity` parameter with one of the following values:

- `TOTAL`**:** Indicates total number of write capacity units consumed by the operation
- `INDEXES`: Indicates total number of write capacity units consumed by the table along with the secondary indexes affected by the operation

- `NONE`: Does not return any details for write capacity consumed by the operation. If you do not explicitly specify the `ReturnConsumedCapacity` parameter, by default it considers `NONE`.

PutItem

It is used for writing a new item in the table. If the table already has an item with the same key, it is replaced with the new item.

Example:

```
aws dynamodb put-item \
    --table-name employee \
    --item file://employee-item.json
```

Details of the --item argument are stored in the file `employee-item.json`:

```
{
    "FirstName": {"S": "Gini"},
    "LastName": {"S": "Davidson"},
    "JoiningDate": {"S": "20120817"},
    "Gender": {"S": "Female"},
    "DateOfBirth": {"S": "19850719"}
}
```

UpdateItem

UpdateItem is used for updating an item in a table. If you use an existing primary key with UpdateItem, it updates the existing item. If the specified key does not exist, it creates a new item in the table.

While, you can specify the attributes that you want to modify with the update expression. Along with the udpate expression, you can use expression attribute values that act as placeholders for the real values.

Example:

```
aws dynamodb update-item \
    --table-name employee \
    --key file://employee-key.json \
    --update-expression "SET FirstName = :fname, LastName = :lname,
JoiningDate = :jdate" \
    --expression-attribute-values file://expression-attribute-values.json \
    --return-values ALL_NEW
```

The values for the arguments against `--key` parameter are stored in the file `employee-key.json`:

```
{
    "employee_id": {"S": "E10001"},
    "DateOfBirth": {"S": "19850719"}
}
```

The values for arguments against `--expression-attribute-values` are stored in the file `expression-attribute-values.json`:

```
{
    ":fname": {"S":"Johnson"},
    ":lname": {"S":"David"},
    ":jdate": {"S":"19850720"}
}
```

DeleteItem

The deleteItem operation is used for deleting an item from a DynamoDB table. You need to supply a specific key value as a parameter with the DeleteItem operation.

```
aws dynamodb delete-item \
    --table-name employee \
    --key file://key.json
```

It is important to note that if the table has a composite key, you need to specify both the partition key and the sort key.

Contents of the `key.json` file:

```
{
    "employee_id": {"S": "E10001"},
    "DateOfBirth": {"S": "19850719"}
}
```

Conditional writes

You can perform write operations in a DynamoDB table using PutItem, UpdateItem, and DeleteItem. By default, these operations are unconditional in nature; this means that when you perform write operations using these statements, they overwrites an existing item with the same primary key value specified in the operation.

If you do not want to overwrite an existing item, or write an item only based on a specific condition, you can use conditional write with these operations. Conditional write succeeds only in case it meets an expected condition; else, it returns an error. Conditional writes can be used in multiple scenarios.

Example scenarios when conditional writes can be used:

- You want to write an item using the PutItem operation, only if the item with the specific key does not exist
- You want to update an item using the UpdateItem operation, only if the item attribute contains a specific value
- You want to update an item using the UpdateItem operation, only if it is not already been modified by another user

Conditional writes can be handy in situations where multiple users try to update the same item. Let's consider the following *Figure 11.13* to understand this scenario in which Vipul and Bhavin are trying to update the same item in a DynamoDB table:

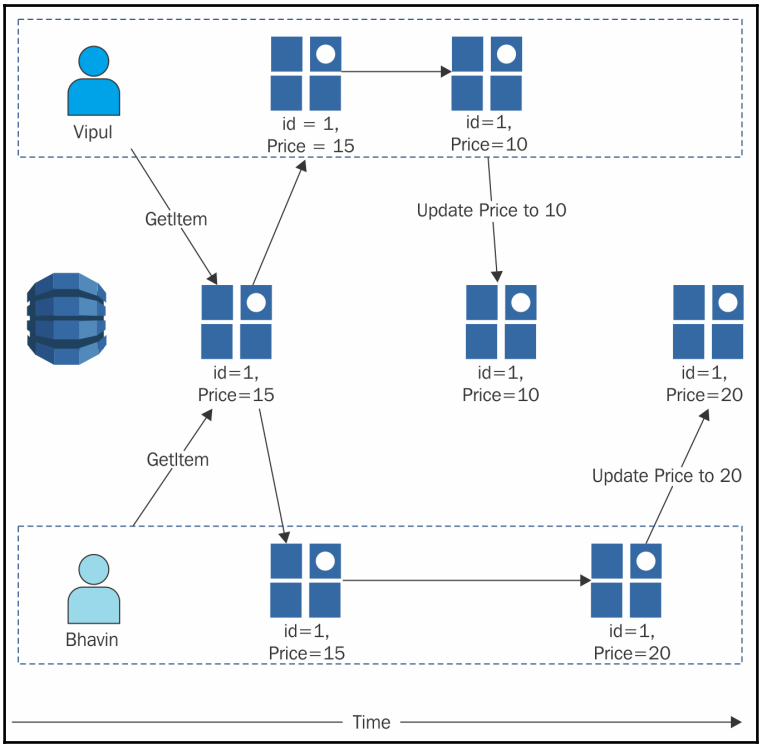


Figure 11.13: No Condition write

Let's assume that Vipul tries to update the `Price` attribute to `10` in the `ProductMaster` table:

```
aws dynamodb update-item \
    --table-name ProductMaster \
    --key '{"Id":{"N":"1"}}' \
    --update-expression "SET Price = :newprice" \
    --expression-attribute-values file://expression-attribute-values.json
```

As we explained earlier, the arguments for `--expression-attribute-values` should be stored in a separate JSON file. In this case, arguments are stored in the file named expression`-attribute-values.json` with the following content:

```
{
    ":newprice":{"N":"10"}
}
```

Now let's consider that Bhavin updates the same item using the UpdateItem request and changes the price to `20`. For Bhavin, the `--expression-attribute-values` parameter file can contain the following values:

```
{
    ":newprice":{"N":"20"}
}
```

The UpdateItem operation initiated by Bhavin succeeds but it overwrites the change made by Vipul.

For performing a conditional write, you need to specify the condition expression along with PutItem, DeleteItem, or UpdateItem. A condition expression contains a string with attribute names, conditional operators, and built-in functions. The operation executes only if the entire expression evaluates to true else; if not, the operation fails.

Let us understand the conditional write using the following figure, which shows how the conditional write prevents Bhavin from overwriting Vipul's changes on the same item. You can compare the previous *Figure 11.13* and the following *Figure 11.14* and observe how the condition is highlighted shown as follow:

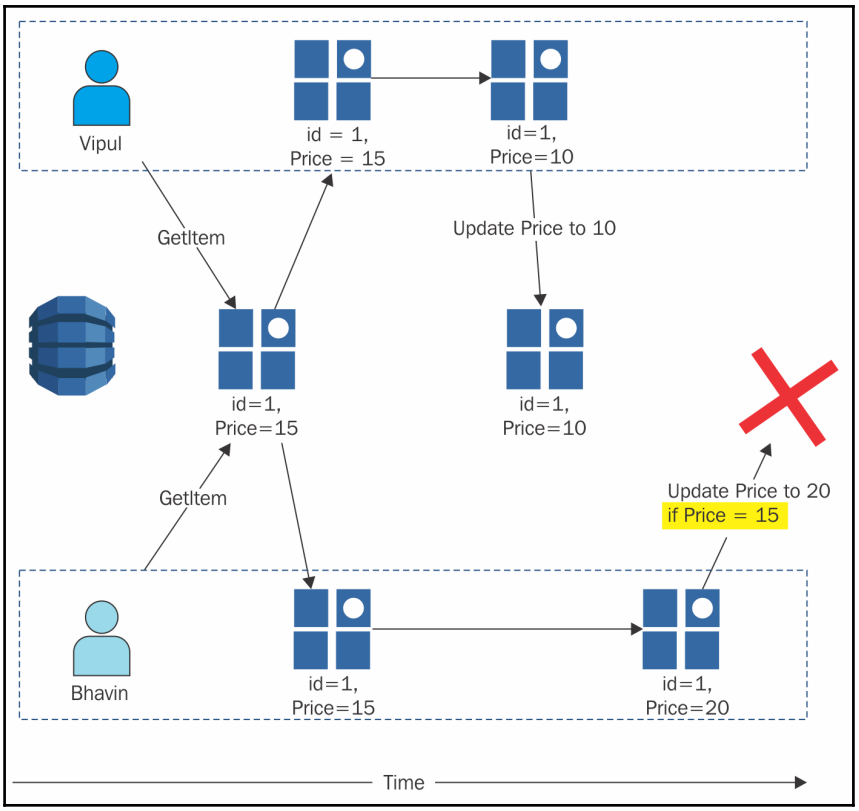


Figure 11.14: Conditional write

Let us understand how you can achieve this using AWS CLI.

While updating the `Price`, Vipul gives a condition to update the Price to 10 only if the Price is 15:

```
aws dynamodb update-item \
    --table-name ProductMaster \
    --key '{"Id":{"N":"1"}}' \
    --update-expression "SET Price = :newprice" \
    --condition-expression "Price = :currprice" \
    --expression-attribute-values file://expression-attribute-values.json
```

The arguments for `--expression-attribute-values` should be stored in a separate JSON file. In this case, arguments are stored in the file named `expression-attribute-values.json` with the following content:

```
{
    ":newprice":{"N":"10"},
    ":currprice":{"N":"15"}
}
```

Vipul's update succeeds as the condition evaluates to be true.

Similarly, Bhavin tries to update the `Price` to `20` with a conditional expression that checks for the current price as 15 before updating it.

For Bhavin, the `--expression-attribute-values` parameter file can contain the following values:

```
{
    ":newprice":{"N":"20"},
    ":currprice":{"N":"15"}
}
```

Since Vipul has already changed the `Price` to `10`, the condition expression in Bhavin's request evaluates to false and his update fails.

User authentication and access control

For accessing DynamoDB you need credentials. The credentials should have the permission to access the DynamoDB table. This section provides details on how you can use IAM to secure DynamoDB resources.

There are a number of ways in which you can access DynamoDB resources:

- AWS root user account
- IAM user
- IAM role
    - Identity federation
    - Cross-account access
    - AWS service access
    - Application running on EC2

If you have valid credentials that can authenticate against DynamoDB you can initiate the request to access but, unless you have permissions associated with your credentials, you cannot perform any operation against DynamoDB. For example, for creating a new DynamoDB table, you need to have the table creation permission.

Before understanding these permissions, let us understand the resources in DynamoDB.

Tables are the primary resources in DynamoDB. Apart from tables, there are indexes and streams, which are part of DynamoDB. Indexes and streams are associated with a table and they are sub-resources of a table. DynamoDB maintains unique ARNs with each of the resources and sub-resources as shown in the following table:

Resource type	**ARN format**
Table	arn:aws:dynamodb:*region*:*account-id*:table/*table-name*
Index	arn:aws:dynamodb:*region*:*account-id*:table/*table-name*/index/*index-name*
Stream	arn:aws:dynamodb:*region*:*account-id*:table/*table-name*/stream/*stream-label*

Managing policies

Access to any resource is determined by a permission policy. In `Chapter 3`, *Getting Familiar with Identity and Access Management*, managing policy is discussed in detail. This chapter emphasizes on IAM with respect to DynamoDB. There are two types of policy: identity-based policies and resource-based policies. Identity-based policies are attached to an IAM identity and resource-based policies are attached to a resource. This section discusses about identity-based policies only, as DynamoDB does not support resource-based policies.

There are three identities in IAM

- User
- Group
- Role

You can attach a policy to a user or a group and grant them access to DynamoDB resources such as table, indexes, and streams.

You can attach a policy to a role for granting cross-account permissions.

Here's an example of a permission policy. This example policy allows `dynamoDB:ListTables` permission for all resources:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ListTables",
            "Effect": "Allow",
            "Action": [
                "dynamodb:ListTables"
            ],
            "Resource": "*"
        }
    ]
}
```

Here's one more example of a permission policy. In this example policy, access is granted on three actions, namely DescribeTable, Query, and Scan. As you can observe in the previous policy, access is granted to all resources with `*` and, unlike the previous policy, in this policy, access is granted to a specific table using an ARN for the table:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "DescribeQueryScanEmployeeTable",
            "Effect": "Allow",
            "Action": [
                "dynamodb:DescribeTable",
                "dynamodb:Query",
                "dynamodb:Scan"
            ],
            "Resource": "arn:aws:dynamodb:us-east-1:account-
id:table/employee"
        }
    ]
}
```

DynamoDB API permissions

While setting up a permission policy, you can refer to the following table which lists DynamoDB API operations, associated actions, and the ARN format for the resource. For more information, refer to `http://docs.aws.amazon.com/amazondynamodb/latest/developerguide/api-permissions-reference.html`.

API Actions	Resources
dynamodb:BatchGetItem	`arn:aws:dynamodb:region:account-id:table/table-name` or `arn:aws:dynamodb:region:account-id:table/*`
dynamodb:BatchWriteItem	`arn:aws:dynamodb:region:account-id:table/table-name` or `arn:aws:dynamodb:region:account-id:table/*`
dynamodb:CreateTable	`arn:aws:dynamodb:region:account-id:table/table-name` or `arn:aws:dynamodb:region:account-id:table/*`
dynamodb:DeleteItem	`arn:aws:dynamodb:region:account-id:table/table-name` or `arn:aws:dynamodb:region:account-id:table/*`
dynamodb:DeleteTable	`arn:aws:dynamodb:region:account-id:table/table-name` or `arn:aws:dynamodb:region:account-id:table/*`
dynamodb:DescribeLimits	`arn:aws:dynamodb:region:account-id:*`
dynamodb:DescribeReservedCapacity	`arn:aws:dynamodb:region:account-id:*`
dynamodb:DescribeStream	`arn:aws:dynamodb:region:account-id:table/table-name/stream/stream-label` or `arn:aws:dynamodb:region:account-id:table/table-name/stream/*`
dynamodb:DescribeTable	`arn:aws:dynamodb:region:account-id:table/table-name` or `arn:aws:dynamodb:region:account-id:table/*`
dynamodb:DescribeTimeToLive	`arn:aws:dynamodb:region:account-id:table/table-name` or `arn:aws:dynamodb:region:account-id:table/*`
dynamodb:GetItem	`arn:aws:dynamodb:region:account-id:table/table-name` or `arn:aws:dynamodb:region:account-id:table/*`
dynamodb:GetRecords	`arn:aws:dynamodb:region:account-id:table/table-name/stream/stream-label` or `arn:aws:dynamodb:region:account-id:table/table-name/stream/*`
dynamodb:GetShardIterator	`arn:aws:dynamodb:region:account-id:table/table-name/stream/stream-label` or `arn:aws:dynamodb:region:account-id:table/table-name/stream/*`
dynamodb:ListStreams	`arn:aws:dynamodb:region:account-id:table/table-name/stream/*` or `arn:aws:dynamodb:region:account-id:table/*/stream/*`
dynamodb:ListTables	`*`

dynamodb:ListTagsOfResource	`arn:aws:dynamodb:region:account-id:table/table-name`
	or
	`arn:aws:dynamodb:region:account-id:table/*`

dynamodb:PutItem	`arn:aws:dynamodb:region:account-id:table/table-name`
	or
	`arn:aws:dynamodb:region:account-id:table/*`

dynamodb:Query	To query a table:
	`arn:aws:dynamodb:region:account-id:table/table-name`
	or:
	`arn:aws:dynamodb:region:account-id:table/*`
	To query an index:
	`arn:aws:dynamodb:region:account-id:table/table-name/index/index-name`
	or:
	`arn:aws:dynamodb:region:account-id:table/table-name/index/*`

dynamodb:Scan	To scan a table:
	`arn:aws:dynamodb:region:account-id:table/table-name`
	or:
	`arn:aws:dynamodb:region:account-id:table/*`
	To scan an index:
	`arn:aws:dynamodb:region:account-id:table/table-name/index/index-name`
	or:
	`arn:aws:dynamodb:region:account-id:table/table-name/index/*`

dynamodb:TagResource	`arn:aws:dynamodb:region:account-id:table/table-name`
	or
	`arn:aws:dynamodb:region:account-id:table/*`

dynamodb:UpdateItem	`arn:aws:dynamodb:region:account-id:table/table-name`
	or
	`arn:aws:dynamodb:region:account-id:table/*`

dynamodb:UpdateTable	`arn:aws:dynamodb:region:account-id:table/table-name`
	or
	`arn:aws:dynamodb:region:account-id:table/*`

dynamodb:UpdateTimeToLive	`arn:aws:dynamodb:region:account-id:table/table-name`
	or
	`arn:aws:dynamodb:region:account-id:table/*`

dynamodb:UntagResource	`arn:aws:dynamodb:region:account-id:table/table-name`
	or
	`arn:aws:dynamodb:region:account-id:table/*`