Hello Cloud Gurus and welcome to this lecture,

which is part two of my EC2 summary.

So let's begin with Route 53

and remember, Route 53 is Amazon's DNS service

and it allows you to map a domain name

to an EC2 instance,

an elastic load balancer or S3 bucket.

And if you remember, we created a web server running httpd,

we added an application load balancer

and then we configured our hosted zone

with a new alias and an A record

to map our domain name

to the DNS address of our application load balancer

and this is one of the most common use cases for Route 53.

And just be aware of the following Route 53 terminology.

So we used a hosted zone,

which is a container for DNS records for our domain

and alias is what allows you to route traffic

addressed to the zone apex

or the top of the DNS namespace,

for example, ilovecloud.com

and send it to a resource within AWS,

for example, an elastic load balancer

and an A record allows you to route traffic to a resource

such as a web server, using an IPv4 address.

Moving on to the AWS CLI,

keep in mind the principle of least privilege

and always give your users the minimum amount of access

required to do their job.

And it's best practice to use groups,

so create identity access management groups

and assign your users to groups.

Group permissions are assigned using IAM policy documents

and your users will automatically inherit

the permissions of the group.

When it comes to generating the secret access key,

remember that you will only see this once

and if you lose it,

then you can delete the access key ID

and secret access key and regenerate them.

And if you do that, you will need to run AWS configure again

and provide the new credentials.

Don't share your key pairs

and each developer should have their own access key ID

and secret access key

and just like passwords,

they should never be shared.

And the AWS CLI supports Linux, Windows and MacOS,

so you can install it on your local machine

and you can also use it on EC2 instances as well.

And of course, when we launch a Linux to EC2 instance,

you get the AWS CLI pre-installed.

Onto using roles with EC2

and we can use a role to give an EC2 instance

access to AWS resources like S3.

And to do that, we need to create

an Identity access management role with S3 access.

Create an EC2 instance

and attach the role that we just created

and then we should be able to access S3

from our EC2 instance.

And using roles in this way is the preferred option

from a security perspective.

And it allows you to avoid hard coding your credentials,

so roles allow you to provide access

without having to manage access keys

and secret access keys

and manually configure them on every single EC2 instance.

We use IAM policies to control a roles permissions

and you can update a policy attached to a role

and it will take immediate effect.

And you can attach and detach roles

to running EC2 instances

without having to stop or terminate the instances

And if you remember, we tried that with our own EC2 instance

and the effect was immediate.

Onto RDS or Relational Database Service.

So we've got a few different RDS database types

there's SQLServer, Oracle,

MySQL, PostgreSQL,

MariaDB and Amazon Aurora

and RDS is for online transaction processing workloads.

So this is great for processing lots of small transactions

like customer orders, banking transactions,

payments and booking systems

and It is not suitable for online analytics processing.

So instead, we should use RedShift for OLAP

and data warehousing type tasks

like analyzing large amounts of data,

reporting and sales forecasting.

And it's important to remember the differences

between automated backups

and database snapshots

and they are also known as manual snapshots.

So with automated backups,

these are enabled by default

and you define the backup window.

They provide point-in-time snapshots

plus transaction logs.

You can define a retention period of up to 35 days

and they can be used to recover your database

to any point in time within the retention period.

Whereas database snapshots are user-initiated and ad-hoc.

They provide point-in-time snapshots only,

there's no retention period

and they are stored indefinitely

until you delete them

and they can be used to backup your database instance

to a known state

and restore to that specific state at any time,

for example, you might run a database snapshot

before making a change to the database.

And when it comes to encrypting your RDS database,

you will need to enable encryption at creation time.

So you cannot enable it later

and when you encrypt your RDS database

it includes all the underlying storage,

automated backups,

snapshots, logs and read replicas as well.

So it's going to encrypt everything.

And RDS integrates with KMS,

so it uses the AWS Key Management Service

for AES-256 bit encryption.

And if you have any existing RDS instance

which you need to encrypt,

of course you cannot add encryption later

but what you can do is take a snapshot of your database

then encrypt the snapshot

and create a new RDS instance

from your encrypted snapshot.

Moving on to multi-AZ and read replica

and it's important to understand the differences

between these two technologies.

So with multi-AZ, you get an exact copy

of your production database

in another availability zone.

These are used for disaster recovery,

so in the event of a failure,

RDS will automatically failover to the standby instance.

Whereas a read replica is a read-only copy

of your primary database

and this can be in the same AZ,

cross-AZ or cross-region.

And read replicas are used to increase

or scale read performance.

And a read replica is great for read-heavy workloads

because it takes the load off your primary database

for read-only workloads,

for example, if you need to run

a business intelligence reporting jobs.

Another way to improve read performance for RDS

is to use ElastiCache.

And if you remember, ElastiCache is an in-memory cache

designed to improve read performance

for read-heavy databases

and there are two options available.

Firstly, we've got memcached

and this is an in-memory, key-value data store.

And this is the one to use

if object caching is your primary goal

and you want to keep things as simple as possible,

you don't need persistence or multi-AZ

and you don't need support

for advanced data types or data sorting.

Whereas with redis, that is also an in-memory

key-value data store

but this is the one to use

if you are performing data sorting and ranking,

such as in gaming leaderboards.

And it's also a great choice

if you have advanced data types

such as lists and hashes

and if you need data persistence and multi-AZ.

So memcached is your basic option

whereas redis provides enterprise features

like persistence and multi-AZ.

And finally, we have parameter store,

which is a service that allows you

to store confidential information

such as passwords, database connection strings

and licensed codes, et cetera.

You can store your values as plain text or encrypt them

and reference them using the parameter name,

for example, in a bootstrap script.

And parameter store is integrated

with lots of different AWS services.

So you can use it with EC2, CloudFormation, Lambda

and it also supports CodeBuild, CodePipeline

and CodeDeploy.

So that's it for this lecture,

if you have any questions please let me know.

Otherwise I will see you in the next lecture.

Thank you.