#___________________Section 9 and 10 ________________________________#

Now let's talk about databases and AWS.
And first I want to give you an introduction
as to what is a database
and the different types of databases we have today.
So when you are storing data on disk,
would it be on an EBS drive, an EBS volume,
an EC2 Instance Store, Amazon S3, you have your limits.
If you want to store data in the structure way,
you may want to store it in a database.
And this structure is going to allow you
to build indexes and to efficiently query
or search through the data.
So well we have with EFS EBS, EC2 Instance Store
and Amazon S3, we can do per files operations
with databases, it's going to be a lot more structured.
We can even define relationships between your datasets.
So these databases nowadays,
they're all optimized for a purpose
and they will come with different features,
shapes and constraints.
From an exam perspective, it's up to you to understand
which database is going to fit best
at the use case given to you by the question.
So the first kind of database that has been very common
and it's still a very common today,
is called the relational databases.
So if you used Excel before, you know, Excel spreadsheets,
this is just the same as Excel spreadsheets,
but do you have links between each table.
So for example, we have the students table,
and as we can see, we have four columns
and there's a student ID, department ID,
a name and an email and then for example,
you would have a second spreadsheet
called the department spreadsheets
with text the department ID and as more information.
And the idea is that's in a relational database,
you are defining a relation between the department ID,
the second column in the students table,
into the first column of the departments table.
And you can even define more relations,
for example, you can define a subject's stable
and link the table of the students
to the subjects by defining another relation.
This is why it's called a relational database.
Now in relational databases,
the particularity clarity is that you can use
the SQL language to perform queries or lookups.

So these, whenever you hear SQL think relational databases.
Next, we have NoSQL Databases, NoSQL means non-SQL,
which means non relational databases.
So NoSQL are in more modern kind of databases.
They're built for a specific purpose
with a specific data model in mind
and have a flexible schema for building modern application.
The schema is basically the shape of the data.
So the benefits of using a NoSQL database
is that you have more flexibility.
It's easier to make the data model evolve,
it's scalable, so it's designed to scale out
by adding distributed servers.
So, the example from before the relational database,
it's not easy to add servers to scale it,
so you have to scale up by doing some vertical scaling,
but with no sequel databases, you can do horizontal scaling.
It's also high performance,
it's optimized for a specific data model.
It's highly functional,
the types are optimized for the model.
And finally, some examples of NoSQL databases
are going to be key-value databases,
document, graph, in-memory or a search databases.
We will be seeing those in this section.
So, for example, in the NoSQL database,
you can have the data in JSON format.
So JSON stands for JavaScript Object Notation,
and we've seen it before
this is the same language we've been using
when we defined our IAM policies,
and this is what a JSON document looked like.
As we can see, this does not look at all
like an Excel documents.
So JSON is a very common way to describe data
and as we can see in here,
we have different sub nesting of the data.
We have different fields, different name, different types.
And so, it is a very common way
to muddle data in a NoSQL model.
The data can be nested, for example,
if you look at address right here,
it is nested within the higher object elements.
So address and types are going to be nested.
The fields can change over time.
So it's up to us to add a new field into our JSON documents
and their support for arrays, for example,
John who is 30, has three different cars,
Ford, BMW, and Fiat, he's a lucky man.
Okay, so databases, as you can see get a different forms
and we'll be seeing them in details in the section.

So just some words around databases
and the shared responsibility model on AWS.
So AWS will offer to manage different databases for us.
So these are the offering we'll see
and the benefits of using a managed database,
is that it's very quick to provision.
Usually a database will have,
high availability in mind when designing it.
And if you wanna scale your database,
you can do vertical and horizontal scaling pretty easily.
They will also include some utilities
to do automated backup and restore of your databases
as well as operations and upgrades.
And if you need to patch the operating system
of the underlying instance,
actually it's not your responsibility anymore.
It's AWS's responsibility.
So AWS will be responsible for the entire database
in terms of patching.
The monitoring and alerting as well
are going to be integrated.
And so this is why using a managed database
on AWS is very, very important and very beneficial.
So, you are obviously able
to run your own database technology on an EC2 instance,
but if you do so, you need to handle yourself,
all the things associated with resiliency, backup, patching,
high availability, fault tolerance, and scaling.
So this is why in this case, using a managed database
is going to be a lifesaver
for so many of our use cases, okay?
So in this section,
we'll have a look at all the managed databases from AWS
and which use cases they fit.
So we'll see you in the next lecture.

The first kind of databases
we're going to see is the relational databases,
and for it we have RDS.
RDS stands for Relational Database Service,
so it's very clear that it's only for relational databases
and it will be using the SQL language.
So it's a managed database service for databases
that will use SQL as a query language,
and it will allow you to create databases
in the cloud that will be managed by AWS.
These databases can be of different kinds.
It could be a Postgres. MySQL, MariaDB.
Oracle, Microsoft SQL Server, IBM DB2,
and finally, something called Aurora,
which is a proprietary database from AWS,

which we'll be seeing in this lecture as well.
So why would we use RDS
instead of deploying our own databases on EC2?
Well, RDS is a managed database service,
so provisioning the database will be automatic.
The patching of the operating system will be done by AWS.
We will have continuous backups
and restore options with Point in Time Restore.
We'll have monitoring dashboards
to see if our database is doing good.
We'll be able to scale the reads by creating read replicas
and improve the read performance.
We will have the way to set up Multi AZ
to make sure that we're ready
to have a plan for disaster recovery
against a whole availability zone going down.
And finally, we can set up maintenance windows for upgrades
and we can scale vertically and horizontally.
This storage is going to be backed by EBS.
And the only thing that we cannot do
with an RDS database is that we cannot SSH
into your RDS database instance, okay?
We're just using the service,
AWS manage entirely the database for us,
so we cannot use the SSH utility
to see what's going on within our database.
Okay, so where does RDS fit
within our solution architecture?
Well, we have our load balancer,
and it's fronting multiple backend EC2 instances,
they're possibly into an auto scaling group,
and they need to store and share the data somewhere.
And this is structured data,
so they're not going to use EBS
or EFS or EC2 instance store,
they will be using a database and in this example,
it is a relational database, so SQL based.
And the EC2 instances will be connecting
to the database and doing read writes all at once.
So they will be sharing the database in the backend,
and this is a classic solution architecture,
not just using RDS, but using any kind of database.
You have the load balancer layer
that will be taking the web request,
the backend EC2 instances doing the application logic,
and the last tier, which is the right hand side,
which is going to be the database tier
doing the reads and the writes of the data.
Okay, so this is for RDS.
So now for Aurora.
So, Aurora is going to be a database technology

created by AWS, it's not open source,
and it works the same way as RDS.
We have our EC2 instances connecting directly
into Amazon Aurora.
And Aurora supports two kinds of database technologies.
It supports PostgreSQL and MySQL.
The idea with Aurora is that it's going
to be cloud optimized,
and so there is a 5x performance improvement
over MySQL on RDS and a 3x performance improvement
over using Postgres on RDS.
On top of things, we don't need to worry
about the storage of an Aurora database
because the storage will grow automatically
in increments of 10 gigabytes up to 128 terabytes.
Even if Aurora is going to be more expensive than RDS,
about 20% more, it's going to be more efficient,
and so it should be more cost effective.
Finally, Aurora is not something
that's included into the free tier of AWS, but RDS is, okay?
So from an exam perspective,
RDS and Aurora are going to be
the two ways for you to create relational databases on AWS.
They're both managed,
and Aurora is going to be more cloud native,
whereas RDS is going to be running the technologies
you know directly as a managed service.
And we also have a serverless option for Amazon Aurora.
So this is where the database
instantiation is going to be automated.
And on top of it, you will have auto-scaling
based on actual usage of your database.
So both PostgreSQL and MySQL are supported as engines
of Aurora Serverless Database.
You don't need to do any kind of capacity planning.
There's no management because you don't manage servers
and you're going to pay per second,
so it can be a lot more cost effective
than provisioning an Aurora cluster yourself.
So this is going to be great when you have infrequent
or intermittent or unpredictable workloads.
So how does that work?
Well, from the client perspective, it's super easy.
It connects to a proxy fleet that is managed by Aurora.
And Aurora, behind the scenes,
is going to instantiate database instances
when it needs to scale up or down.
And these Aurora databases are going
to be sharing the same storage volume no matter what.
So from an exam perspective,
if you see Aurora with no management overhead and so on,

think of Aurora Serverless.
Okay, so that's it for this lecture.
I hope you liked it.
And I will see you in the next lecture.

So let's go ahead
and create our first database in RDS.
And we're not going to go over all the options,
but we're going to see the general idea
of RDS and how it works.
So, on the left hand side, I will go under Databases
and I will create a database.
So here, we have the interface.
And so we have a standard create and an easy create.
So an easy create is to use recommended
best-practice configurations,
whereas the standard create allows you
to just customize everything.
So, for the purpose of this demonstration,
I will use a standard create.
So we have different options of the engine.
We have Aurora, MySQL, MariaDB,
PostgreSQL, Oracle, and Microsoft SQL Server.
So, in this demo, we're going to launch
a very simple managed MySQL database using RDS,
but we can see that there is a lot
of options available to us.
We'll use the MySQL Community,
and then we'll use the latest version that is recommended,
8.0.23, but it really doesn't matter.
Here, we have some templates
to launch our MySQL RDS database.
We have the Production template,
the Dev/Test template, or the Free Tier template,
which is quite nice because it pre-fills
some of the settings from below.
And we'll see what I mean.
So if we go into the free tier,
because we wanna stay within the free tier.
Let's scroll down.
For the DB identifier, I will use database-1. This is fine.
Here, there is a master username. So admin is great.
And then, the password, I will enter password twice.
And then, I will scroll down.
The database class is now a burstable class of T2 micro.
Well, that's the only one I can actually enter here
because I went with the Free Tier setting.
But if I went with, for example, Production setting,
I could be choosing another type of class,
for example, any kind of class in here.
But again, we'll stay within the free tier

and use a T2 micro type of database.
I'll scroll down, we have some settings around storage.
We can enable 20 gigabytes of gp2 SSD storage,
and then we can have storage autoscaling.
In case we are having more storage than 20 gigabytes,
it will automatically increase the storage for us
with a maximum of one terabyte, 1,000 gigabytes in here.
Availability and durability,
for now, we will not go over these settings.
Connectivity, so where do we want to launch the database?
So we specify a virtual private cloud, or a VPC,
as well as a subnet group.
Do we want to have public access to our database? Yes or no?
Well, you need to specify yes
if you want to be able to connect
to your database from your computer
if you don't have direct connectivity into AWS,
which should be the case for you,
so we'll keep it as yes public access.
Then, we should assign some security group.
So you can create a new security group.
I'll call it demo-database-rds.
And the AZ is going to be no preference. This is fine.
And database port is 3306. Excellent.
Now, how do we want to authenticate the database?
Using password or password and IAM or password and Kerberos.
We'll keep it very simple
and use it with password authentication.
And we will not go over any additional configuration for now
because there could be a lot for you.
But so, as we can see here, we have the RDS Free Tier.
It's available for us for 12 months,
and then we'll get 725,
basically, a full month of RDS free every month
for the db.t2.micro instance
as well as 20 gigabytes free of SSD,
so, again, this is what we selected,
and some space for backups.
So we'll create this database.
So it took a few minutes, but now my database is created.
And here, we have some summary.
So we know where it is launched,
for example, eu-central-1a.
We get some information around the endpoint.
If you wanted to connect to it,
this would be the endpoint and the port to be using.
We can have a look at the VPC security group
that has been created for my RDS instance
and the inbound rules.
We can see there's an inbound rule on the port,
and if I scroll right,

on the port 3306 into my instance.
So that means that I can use the correct port
to connect to my MySQL database instance.
Perfect.
If we look into Monitoring,
because this is a managed service,
we have some information around the CPU utilization,
which is great.
So we start really seeing the power of Managed Services,
login events, and so on.
Configuration, Backups, and Tags.
So, as we can see, having a database in the cloud
is really helpful when it's managed
because, well, we have all the nice monitoring,
configuration, settings, and so on, associated with it,
so this is a huge advantage.
Other things you can do
is that you can take a snapshot of your database,
and this snapshot will allow you, for example,
to restore the database into another one.
So let's do it, demo-snapshot, and we'll take the snapshot.
So we can't do it because my database is still
in a backup stage, so what I'm going to do is wait for it
to not be backed up, but to be available.
Okay, so my database is now available.
So let's go back to Actions, Take Snapshot,
and here's my snapshot name.
Take the snapshot.
Now, my snapshot is being created.
And with it, I have a few options.
So I can click on it and I can do a few things.
So let's wait for it to be created.
Right now, it's in the Creating status.
So my snapshot is now available,
and what I can do is do Action, Restore Snapshot,
and this would allow me to create a new database
out of this snapshot.
And the reason why I would do so is that, for example,
you wanted to create a bigger database
or create a copy of the database and so on,
or create a different settings for your database.
Then, restoring a snapshot would be the way to do it.
So I'm going to cancel this.
Other things you can do with your snapshot is to copy it.
So you can copy it to a different region.
And this would be extremely helpful
if you wanted to restore the database
into another region of AWS,
for example, for disaster recovery.
So yet another very handy feature.
Another thing you can do is share the snapshot.

And sharing the snapshot allows you
to share with other accounts
so they can restore their database
directly from your snapshot.
So again, another very handy feature.
So that's it for RDS, just as a quick overview.
So again, we've created a database,
and we've created a snapshot out of it,
and we saw different options.
So you can see, Managed Services really have a good impact
on how you use the cloud,
and they really speed it up for you
to make sure that you don't manage too much infrastructure.
So just to finish this hands-on,
let's delete this snapshot,
and then we're also going to delete this database
by doing Action, Delete,
and then you don't want to create a final snapshot,
you do not want to retain automated backups,
and you acknowledge that everything will be deleted,
then you type delete me, and you're good to go.
So that's it for this lecture.
I hope you liked it,
and I will see you in the next lecture.

So when you deploy RDS databases, you need to understand
there are multiple architectural choices
you need to make, that you can make, actually.
So the first is to use RDS Read Replica.
So let's take the example of your application,
that reads from your main RDS database.
But say that now you need to scale the read workloads,
we have more and more applications
that need to read more and more data from RDS.
The way you can do it is by creating Read Replica.
So that means that there is gonna be some copies,
some replicas of your RDS database
that are going to be created.
And this is going to allow your applications
to read from this Read Replica also.
And therefore, you're distributing the reads
to many different RDS databases.
So you can create up to 15 Read Replicas.
So as you can see, in this example,
I have two Read Replicas on top of my main RDS database,
and my applications can read from all of them.
Now, when it comes to writing data,
writing data is only done to the main database,
so your application still have to write

to the only one central RDS database, okay.
So Read Replicas are used to scale reads.
Now we have Multi-AZ.
And so this is helpful when you have
to have failover in case of an AZ outage.
So in case in the availability zone,
like crashes, and this gives you high availability.
So in this example, your applications to read and write
from the same main RDS database, okay.
But we are going to set up
a replication cross AZ,
so in another different availability zone.
And this is going to be a failover database,
and this is why it's called Multi-AZ
because it's in a different AZ.
Now, in case the main RDS database crashes,
for whatever reason,
because maybe there's an issue with it,
or maybe because the AZ is having problems,
then RDS will trigger a failover.
And then your application will failover
to developer database in a different AZ.
So in this case, data is only read and written
to the main database.
The failover DB is passive,
it's not accessible until there is an issue
with the main database.
And you can only have one other AZ zone as a failover AZ.
The last kind of deployment you can do
is called Multi Region.
So this is for read replicas,
but this time, instead of being in the same region,
they are across different regions.
So to take an example,
we have EU-West one for RDS database,
and we're going to create a read replica in US-East two.
And so applications in US-East two can read locally
from this read replica.
But anytime this application needs to write data,
the writes need to happen across region.
And so we need to write the data to US one.
Same if you were to have also another region
in AP-Southeast two so in Australia,
you'd have the same concepts.
So why would you want a multi region type of deployments?
Well, number one,
because you want to set up a disaster recovery strategy,
in case of a region issue.
So if EU-West one is having a regional issue,
then you have a backup in either ES-East two,
or AP-Southeast two,

and this is why it's a disaster recovery strategy.
And also, as you can see,
our applications that are in different regions
get better performance, because they're read
from a local database, so they have less latency.
But finally, when you do this,
you need to understand that
because you are replicating data across regions,
then there is going to be a replication cost associated
with a network transfers of data between regions.
So that's it for RDS deployments,
the scenarios that will be proposed to you at the exam
will make it clear what department you choose.
So hope you liked it and I will see you in the next lecture.

So now let's talk about a second type
of database you will see on AWS,
which is Amazon ElastiCache.
So the same way you use RDS to get
managed Relational Databases,
you're going to get ElastiCache to get a managed,
Redis or Memcached database.
These caches are going to be in-memory databases
with high performance, low latency.
So anytime in the exam you see something that says
you need an in-memory database,
you have to think directly ElastiCache.
Now ElastiCache will be helpful
to reduce load off of databases
that have a read intensive workloads.
The idea is that if we have an RDS database
and we're doing a lot of query on it
and they're the same queries all the time,
we put a bit of pressure onto that RDS database.
Instead what we could be doing,
is to use a cache to reduce the pressure
off of the database by making sure the queries
are directly going onto my in-memory database
through ElastiCache.
And because this is a managed database,
then AWS will take care of all the operating system
maintenance and patching, the optimizations,
the setup, the configuration, the monitoring,
failure recovery and backups.
So we've already learned a lot about caches in this exam
but we need to know it's an in-memory database
and the solution architecture around using a cache
looks like this.
So you're Elastic Load Balancer
will go to your EC2 Instances.

Possibly in an ASG.
They will be reading and writing data from
your Amazon RDS database, which is slow.
And then if possible there will be caching some values
into an Amazon ElastiCache database
and this will be very fast because it's in-memory.
And so with ElastiCache, there will be pressure
taken off the main RDS database
and put it on to the ElastiCache database
and that's the whole idea behind the cache.
You want to save the queries somewhere else,
so that they're very readily available.
Easily accessible and they can relieve, again pressure
from the main database.
So that's it, that's all you need to know going
into the exam and I will see you in the next lecture.

Okay, so let's go ahead and practice
DynamoDB a little bit.
So I'm going to create a table,
and I'll call this one DemoTable.
Now in DynamoDB what you need to do
is specify a partition key, so specify user_id,
and sort keys are definitely out of scope for the exam,
so let's just consider just the partition key.
Okay, so, now for the settings in DynamoDB, again,
I'm going to leave it as a default settings,
you don't need to know the details of how it works,
and then you scroll down and you click on Create table.
So our table is now creating,
and what I'm going to notice is that we are creating
a table without creating a database.
So the database already exists, it's serverless,
we don't need to provision servers.
We just want you to say,
"Hey, look, I want this table, please create it for me
and I don't care how it's being run."
And that's the whole power of DynamoDB,
that's the whole power of serverless services.
So now that the table is ready
we can click on View items, and practice a little bit
inserting some data into this table.
So currently zero items are returned
because I haven't created anything,
but we can create an item in DynamoDB.
And for user_id I can say 1234,
so that will be my user ID.
We can have a first name to be equal to Stephane,
and we can have a last name to be equal to Maarek,
and then, finally, we could have a number.

So I can see lot of different types we can have, by number,
and it could be my favorite number, and it's 42,
and click on Create item.
And here we go, my item was written into DynamoDB.
So this is super easy, yet, again,
I do not have to specify any database, just my table,
write some item, and so on.
And we didn't have to specify this schema,
it just get automatically inferred,
now we have four attributes or columns.
Now if I can create a second item, so 45678,
and then add a new string, so, for example, first_name.
In here we're going to have Alice, and click on Create item.
So, as we can see, in this example,
I didn't specify a favorite number or last name for Alice,
I just specified first name Alice,
and it was still accepted by DynamoDB.
So it's a very flexible type of database,
it's a very flexible way to insert data,
and this whole, like, test sets of features
and particularities make DynamoDB really, really good.
But the difference between DynamoDB and, say, RDS
is that DynamoDB will have all the data
living within one single table,
and there's no way to join it with another table.
So it's not a relational database,
that's when DynamoDB is a non SQL database,
so not only SQL database, so NoSQL.
And the idea, here, is that, yes,
we cannot link this table to another table,
so we need to make sure that all the relevant data
is well formatted within our main DynamoDB table.
So it changes a little bit
how you see database designs and so on.
But that's it for DynamoDB, a very quick hands-on
to show you an overview, but with DynamoDB
it's a lot more to learn and this is the focus
of the Certified Developer exam,
not the Cloud Practitioner exam.
So, for this example, this is enough,
and when you're done and ready
you can just delete the table.
You can delete all the CloudWatch alarms
associated with it, and just type delete in this box
and you'll be good to go.
So that's it for this lecture, I hope you liked it,
and I will see you in the next lecture.

Okay, so now let's talk about

DynamoDB global tables.
So this is a one feature need to know about for DynamoDB.
It's a way for you to make DynamoDB table
accessible with low latency and this is the key
in multiple regions.
So let's take an example.
We have our DynamoDB table in us-east-one,
and we'll set up as a global table.
So the basics, of course, our users can do read and write
to this DynamoDB table in Northern Virginia us-east-one.
But it is possible for us
to set up some replication for this global table.
So we can create a global table in Paris eu-west-three
and we'll say they're the same,
so there's two way replication between these tables.
That means that the same data is in us-east-one
and in eu-west-three,
but users, you know, there are close to the Paris region,
can access this global table with low latency in Paris.
So this is true for one to 10 regions, if you want it to.
Okay.
A global table is truly global
and users can read and write
to the table in any specific region,
there will just be replication between these two.
So the fact that there is read/write access
to any region of AWS on this global table,
makes it an active-active replication
because you can actively write to any region
and it will actively be replicated into other regions. Okay.
So that's it, all you need to know for the exam.
I will see you in the next session.

Okay, next type of database we have
is Redshift.
Redshift is a database
that is based on PostgreSQL,
but it's not used for OLTP.
OLTP stands for Online Transaction Processing.
That is what RDS was good for.
Instead, Redshift is OLAP,
or Online Analytic Analytical Processing,
which is used to do analytics and data warehousing.
So anytime in the exam you're saying a database
needs to be a warehouse and to do analytics on it,
then Redshift is going to be your answer.
With Redshift, you don't load the data continuously.
You load it, for example, every hour.
And the idea with Redshift that is really,
really good at analyzing data and making some computations.
So it boasts 10x better performance

than other data warehouses and scales to petabytes of data.
The data is stored in columns,
so it's called a columnar storage
of data instead of row-based.
So anytime you see columnar, again,
think Redshift, and it has something called
the Massively Parallel Query Execution MPP engine
to do these computations very, very quickly.
It's pay as you go based
on the instances you have provisioned,
and has a SQL interface to perform the queries.
It's also finally integrated with BI,
so business intelligence tools such as QuickSight or Tableau
if you wanted to create dashboards on top of your data
in your data warehouse.
So that's it.
Just a high level overview, but a data warehouse is used
to do some computation on your data sets
and do some analytics,
and possibly build some kind of visualizations
through dashboards on it.
And so for that use case, Redshift
would be perfect.
We now have a feature
for Redshift called Redshift Serverless,
and this allows you to run Redshift but not worry
about scaling the data warehouse or provisioning it.
(indistinct) does it for us,
that's what it's called, serverless.
So the idea is that you're going
to run your analytics workloads
without managing data warehouse infrastructure,
which is very handy, and you're only going to pay
for what you use, which allows you to save on costs.
So the use cases will be to do reporting
or dashboarding applications, or realtime analytics,
but again, without managing the underlying capacity
and the infrastructure
of your Redshift Serverless database.
So how does that work?
Well, you just enable Amazon Redshift Serverless
on your account.
Then you connect the Redshift Query Editor
or any other tool to start writing your queries,
and then automatically Redshift Serverless is going
to run these queries and provision and scale capacity based
on the workload and the query itself.
Finally, you only pay for compute
and storage used during the analysis,
which makes it a very cost efficient option
to running Redshift,

and the exam can test you on it as well.
Okay, so that's it for this lecture.
I hope you liked it and I will see you in the next lecture.

Another type of database we have
on AWS is Amazon EMR, and EMR stands for Elastic MapReduce.
So EMR is actually not really a database.
It's to create what's called a Hadoop cluster
when you wanna do big data on AWS,
and a Hadoop cluster is used to analyze
and process vast amount of data.
So Hadoop is an open source technology,
and they allow multiple servers that work in a cluster
to analyze the data together, and so when you're using EMR,
you can create a cluster made of hundreds of EC2 instances
that will be collaborating together to analyze your data.
So part of the Hadoop ecosystem, the Big Data ecosystem,
you will see projects names such as Apache Spark, HBase,
Presto, and Flink, and all these things
will be working on top of your Hadoop cluster.
So what is EMR then?
Well, EMR takes care of provisioning all these EC2 instances
and configuring them so that they work together
and can analyze together data from a big data perspective.
Finally, it has auto-scaling
and it is integrated with Spot instances,
and the use cases for EMR will be data processing,
machine learning, web indexing, or big data in general.
So from an exam perspective,
any time you see Hadoop cluster,
think no more, it's going to be Amazon EMR.
That's it, I hope that was helpful,
and I will see you in the next lecture.

So now let's talk about Amazon Athena.
Amazon Athena is a serverless query service
to perform analytics against your objects
stored in Amazon S3.
So the idea is that you would use the SQL query language to
create these files, but you don't need to load them.
They just need to be in S3 and Athena will do the rest.
So these files can be formatted in different ways,
such as CSV, JSON, ORC, Avro, and Parquet
and the Athena is built on the Presto engine,
if you must know.
Now, how does it work?
So users will load it into Amazon S3,
and then Amazon Athena will be used
to query and analyze the data.
Very, very simple.

And then if you wanted to,
you could have some reporting on top of Athena,
such as using Amazon QuickSight.
Now the pricing for Athena is around
$5 per terabyte of data scanned.
And if you use compressed
or data stored in a columnar fashion,
then you're going to have cost savings
because there is less scan of the data being made.
So the use cases of Athena are multiple,
but anytime you see Business intelligence,
analytics, or reporting,
or to analyze Flow Logs in VPC or ELB Logs,
or CloudTrail logs, or platform logs,
all these kinds of logs in AWS,
then Athena's going to be a really, really good option.
So from an exam perspective,
whenever you see serverless analyze data in S3
use SQL, then think Amazon Athena.
That's it.
I hope you liked it.
And I will see you in the next lecture.

Now let's talk about Amazon QuickSight.
So it's a serverless, machine learning-powered
business intelligence service
to create interactive dashboards.
So behind this very complicated tagline,
all you have to remember is that Amazon QuickSight
allows you to create dashboards on your databases
so we can visually represent your data
and show your business users
the insights they're looking for, okay.
So QuickSight allows you to create
all these kind of cool graphs, charts, and so on.
So it's fast, it's automatically scalable.
It's embeddable and there's per-session pricing,
so you don't have to provision any servers.
The use cases are business analytics,
building visualizations, performing ad-hoc analysis,
get business insights using data.
And in terms of integrations, there are so many,
but, for example, QuickSight can run on top
of your RDS database, it can run on top of Aurora,
Athena, Redshift, Amazon S3, and so on.
So QuickSight is your go-to tool for DI in AWS.
That's it. I will see you in the next lecture.

Now let's talk about DocumentDB.
So the same way we had Aurora as the way for AWS

to implement a sort of big cloud native version
of PostgreSQL and MySQL,
we have DocumentDB, which is an Aurora version for MongoDB.
So MongoDB, if you don't know,
that's the logo on the top right corner of your screen,
it is another no SQL database
and you need to remember this for the exam.
So DocumentDB is a no SQL database
and it's based on top of the MongoDB technology.
So it's compatible with MongoDB.
So MongoDB is used to store query and index JSON data,
and you have the same similar deployment concept
as Aurora with DocumentDB.
So that means it's a fully managed database,
it's highly available.
Data is replicated across three availability zones
and DocumentDB storage automatically will grow
in increments of 10 gigabytes,
and DocumentDB has been engineered
so it can scale to workloads
with millions of requests per second.
So at the exam, if you see anything related to MongoDB,
think DocumentDB, or if you see anything related
to no SQL databases, think DocumentDB and also DynamoDB.
That's it for this lecture.
I hope you liked it,
and I will see you in the next lecture.

Now, let's talk about Amazon Neptune.
Neptune is a fully-managed graph database.
So an example of what a graph dataset would be,
would be, for example, something we all know
which is a social network.
So, if we look at a social network,
people are friends, they like, they connect,
they read, they comment and so on.
So users have friends, posts will have comments,
comments have likes from users,
users shares and like posts
and so, all these things are interconnected
and so, they create a graph.
And so, this is why Neptune is a great choice
of database when it comes to graph datasets.
So, Neptune has replication across 3 AZ,
up to 15 read replicas.
It's used to build and run applications
that are gonna be with highly connected datasets,
so like a social network,
and because Neptune is optimized to run queries
that are complex and hard on top of these graph datasets.
You can store up to billions of relations on the database

and query the graph with milliseconds latency.
It's highly available with application
across multiple Availability Zones.
And it's also great for storing knowledge graphs.
For example, the Wikipedia database is a knowledge graph
because all the Wikipedia articles are interconnected
with each other, fraud detection,
recommendations engine and social networking.
So, coming from an exam perspective,
anytime you see anything related to graph databases,
think no more than Neptune.
That's it.
I hope you liked it and I will see you in the next lecture.

So now let's talk about Amazon Timestream.
And, as there's a hint in the name, it is for time series.
So it's a database that is fully managed, fast, scalable,
and serverless for time series data.
So what is time series data?
Well, it's data that is evolving over time.
So, for example, here on the vertical axis we have a number,
and on the horizontal axis we have the year
that is evolving from an older date to a newer date.
And so because the date is evolving over time,
we are talking about a time series datasets.
So Timestream is just for that,
and there's automatic scaling up and down
based on capacity and the need of computes.
You can store and analyze trillions
of events per day in time series form.
It's about a thousand times faster
and one 1/10th of the cost of relational databases.
On top of it, if you want to analyze
the time series data in real time,
you could have time series analytics function
to find your pattern in your database.
So, whenever at the exam you see time series data,
think no more than Amazon Timestream.
All right, that's it.
I hope you liked it, and I will see you in the next lecture.

Now let's talk about Amazon QLDB,
which stands for Quantum Ledger Database.
So what is it?
A ledger is a book recording financial transactions
and so QLDB is going to be just to have a ledger
of financial transactions.
It's a fully managed database, it's serverless,
highly available, and has replication of data

across three availability T zones.
And it's used to review history of all the changes
made to your application data over time,
that's why it's called a ledger.
So it's an immutable system as well,
that means that once you write something to the database,
it can not be removed or modified.
And there is also a way to have a cryptographic signature
to make sure that indeed nothing has been removed.
So how does it work?
Well, there is behind the scenes a journal,
and so a journal has a sequence of modifications.
And so anytime a modification is made,
there is a cryptographic hash that is computed
which guarantees that nothing has been deleted or modified
and so this can be verified by anyone using the database.
So this is extremely helpful for financial transactions
because you wanna make sure that
obviously no financial transaction
is disappearing from your database
which makes QLDB a great ledger database in the cloud.
So you get two to three times better performance
than common ledger blockchain frameworks
and also you can manipulate data using SQL.
Now, as you'll see in the next lecture
there's also another database technologies
called Amazon Managed Blockchain.
But the difference between QLDB and Managed Blockchain
is that with QLDB, there is no concept of decentralization.
That means that there's just a central database
owned by Amazon that allows you to write this journal.
Okay.
And this is in line with many financial regulation rules.
So the difference between QLDB and Managed Blockchain
is that QLDB has a central authority component
and it's a ledger, whereas managed blockchain
is going to have a de-centralization component as well.
Okay.
So that's it, anytime you see financial transactions
and ledger, think QLDB.
I will see you in the next lecture.

Now let's talk about,
Amazon Managed Blockchain.
So what is Blockchain?
It makes it possible to build applications
where multiple parties can execute transactions
without the need for trusted central authority.
So there's a decentralization aspect to a Blockchain.
So managed Blockchain by Amazon,

is a service to join a public Blockchain networks
or create your own scalable private
Blockchain network within AWS.
And it's compatible with two Blockchain so far,
the framework Hyperledger Fabric,
and the framework Ethereum.
So from an exam perspective,
if you see anything related to Blockchains
or Hyperledger fabric or Ethereum,
you have to think Amazon managed Blockchain
which is also a decentralized Blockchain, okay?
So make sure you remember that and that's it,
all you want to know for the exam,
you'll need to be Blockchain experts.
So that's it, I hope you liked it,
and I will see you in the next lecture.

Okay, now let's talk about AWS Glue.
So, Glue is a managed extract, transform, and load service,
or ETL, and from an exams perspective,
that's all you need to know.
But let's go a little bit deeper dive,
to understand how that works.
So, what is ETL?
Well, ETL is very helpful when you have some datasets
but they're not exactly in the right form,
or the right format that you need,
to do your analytics on them.
And so the idea is that you would use an ETL service
to prepare and transform that data.
So, Glue does that, but traditionally you use servers
to do it, but with Glue it's a fully serverless service,
so you just worry about the actual data transformation,
and Glue does the rest.
So, in a diagram for example, Glue ETL sits in the middle,
and say we wanted to extract data from both an S3 Bucket
and an Amazon RDS database.
So, for this, we'd use Glue to extract the data
from both these sources, and then,
once the data is extracted, it is in a Glue service,
and we would write a script to do a transform part.
So here, Glue would help us transform the data,
and then, once it's transformed,
we need to actually analyze it so we can load up that data
into, for example, an Amazon Redshift database,
where we can do our analytics the right way.
And so, Glue sits here, okay?
It's a very powerful tool, because you can do any kind
of instruction transformation
and then you can load it into many different places.

So, that's for Glue ETL, and then there's another service
called Glue Data Catalog, which I think is not at the exam,
but I will still mention it to you 'cause it's important
to know it as part of the Glue family.
So, the Glue Data Catalog, as the name indicates,
is a catalog of your datasets in your Alias infrastructure,
and so this Glue Data Catalog will have a alert reference
of everything, the column names, the field names,
the field types, et cetera, et cetera.
And this can be used by services such as Athena,
Redshift and EMR to discover the datasets
and build the proper schemas for them, okay?
So, that's it. I hope you liked this lecture,
and I will see you in the next lecture.

So we have seen in this section
so many different database technologies
and the question is how do you migrate data
from one database to another?
For this we can use DMS which is properly named
the Database Migration Service.
So we use source database and once we extract the data
so we'll run an EC2 instance
that will be running the DMS software.
We'll extract the data from the source database
and then DMS will insert the data back
into a target database that will be somewhere else.
So with DMS we get a quick and secure database migration
into AWS that's going to be resilient and self healing.
And the cherry on the cake,
the source database remains available during the migration
so we don't have to take it down.
It supports many kinds of migration.
So one it's called an homogeneous migration
where you have Oracle to Oracle for example.
So it's the same database technology for the source database
and the target database.
Or it supports heterogeneous migrations
when the source database technology
and the target are different for example
a Microsoft SQL Server to Aurora.
And in that case, DMS is smart enough
to know how to convert data
from the source into the target.
So any time in the exam you see migration of a database,
DMS is going to be the answer for it.
I hope that was helpful
and I will see you in the next lecture.

So let's summarize everything we know

about databases and analytics in AWS.
And you just need to know what database corresponds
to what use case going into the exam.
So if you have a relational database
and you have OLTP, so online transaction processing,
you have to think RDS and Aurora,
and both these databases support the SQL language,
Sequel language to query your data.
You also need to know for RDS the difference
between a multi-AZ deployment, read replicas,
and multi-regions, as well as their use cases.
If you need to find an in-memory database
or in-memory cache, think ElastiCache.
If you're looking for a key/value database,
think DynamoDB, which is a serverless database.
And if you need caching technology for DynamoDB,
then use the DAX technology,
which is a cache made specifically for DynamoDB.
If you're looking for a web data warehousing
or OLAP online analytical processing,
then you need to look at Redshift,
which is a warehousing technology.
And you can also use the SQL language
to query data on Redshift.
If you're trying to build a Hadoop cluster
to do big data analysis, use the EMR service.
If you want to query data on Amazon S3
in a serverless fashion with a SQL language,
then use Athena.
QuickSight is a way to create dashboards
of visually interactives, visuals,
and so on that can be interactive
as well on your data in a serverless fashion,
then you would use Amazon QuickSight,
also used for business intelligence.
DocumentDB is what I call the Aurora of MongoDB.
So anytime you see MongoDB, think DocumentDB,
which is also using the JSON type of data sets.
And this is a NoSQL database.
So this is another NoSQL database on top of DynamoDB.
Then we have Amazon QLDB,
which is a financial transaction ledger.
Anytime you would see financial transaction,
immutable journal,
something that is cryptographically verifiable,
think Amazon QLDB.
And this is a central database
which is opposed to a decentralized database,
which is Amazon Managed Blockchain,
in which case we can have managed Hyperledger Fabric
and Ethereum blockchains on AWS.

Finally, if you want to have a managed extract,
transform and load tool, so ETL, we can use Glue,
which also has a data catalog service
to discover data sets into your various databases in AWS.
If you need to move data between databases,
please use DMS, which is Database Migration Service.
If you want a graph database, you must choose Neptune.
And if you want to use a time-series database,
you must use Amazon Timestream.
Okay, so that's it for this lecture.
I hope you liked it, and I will see you in the next lecture.

#____

So we are getting into the ECS Section,
but before talking about ECS we need to talk about Docker.
So Docker is something you may have heard before.
I'll try to simplify it.
You can make a 12 hour long course on Docker.
We'll try to make it short in four minutes.
So what is Docker?
Docker is a software development platform to deploy apps.
So the way we've been deploying applications from before
is to install them on Linux
and then they will work,
but with Docker, you will package your app
into something called a container
and that container is very special,
because it can be run on any operating system very easily.
The app, once in a container
will run the exact same way,
regardless of where they're run.
So it could be any machine,
no compatibility issues with predictable behavior,
less work, easier to maintain and deploy.
It will work with any programming language,
any operating system, any technology
and with Docker you can scale containers up and down
very quickly in a matter of seconds.
So that makes Docker
something that is extremely powerful nowadays
to deploy applications.
So if we talk about Docker on an EC2 Instance,
we would have, for example,
a Docker running Java code,
a Docker running NodeJS code,
a Docker running in My SQL Database,
a Docker running Java and so on
all onto the same EC2 Instance.
So the idea is that if we managed to package our application
in a Docker container,

then it will become very easy for us to run it
on an EC2 Instance.
So Docker images, you need to create them.
This is how your container will be run
and they can be stored
in something called Docker Repositories.
So there is a public Docker Repository called the Docker Hub
available at this address.
And you can find the base images for many technologies
or operating systems.
So for Ubuntu, which is a Linux Operating System,
or My SQL, this is a database technology,
NodeJS Java Programming Languages
or as we'll see in this section,
we can use Amazon ECR,
which is a private Docker Repository.
This is where you can store your private Docker images.
So this is more advanced,
but the question is, do you wanna use Docker
or virtual machine?
So Docker is sort of a visualization technology,
but not exactly.
So the resources are shared with a host.
That means that you can have many containers on one server.
So if we look at comparing EC2 and Docker,
this is all in the works.
So we have the infrastructure which is on AWS,
the Host Operating System,
then the Hypervisor
and this is stuff we don't have access to.
And then finally, when we get an EC2 Instance,
we have our application onto the Guest Operating System
and so if we want another EC2 Instance,
it will be created like this
and a third EC2 Instance, it will be credited like that.
So this works great
and this is what happens when we create EC2 Instances.
But in the case of Docker, we have the infrastructure,
the Host OS which is the EC2 Instance
and then the Docker Daemon.
And then as soon as the Docker Daemon is running,
we can have many containers running on to the Docker Daemon
and they're more light weights.
They don't package,
they don't come with a full operating system
and a virtual machine, all of them.
And so that makes Docker very versatile, very easy to scale
and very easy to run.
So this is just an overview of Docker.
You don't need to know what Docker is going into the exam,
but I wanted to give you that sweet little overview

in case you were curious
and in the next lecture,
we'll deal about how to do Docker on AWS
and that will be ECS.
So I will see you in the next lecture.

So now let's talk about ECS.
ECS stands for elastic container service,
and this is used to launch the docker containers
we just talked about on AWS.
For it to work,
we need the docker containers to run somewhere,
and so for ECS, you must provision
and maintain the infrastructure yourself.
So that means you need to create EC2 instances in advance.
AWS will take care of starting
or stopping the containers for you,
and it has integration
with an application balancer,
if you want to create a web application on ECS.
So as a diagram, you would have multiple EC2 instances
and we would need to create these EC2 instances in advance
and they will be running different containers
by the ECS service.
Now the ECS service,
any times it has a new docker container,
it will be smart enough to find on which EC2 instance
to place that docker container.
So anytime in the exam you see,
I want to run docker containers on AWS, think of ECS.
Now let's talk about Fargate.
So Fargate is also used to launch docker containers on AWS.
But this time with Fargate,
we don't need to provision any infrastructure.
So we don't need to create any EC2 instances
and manage them.
And this is a much more simple offering from AWS.
This is actually a serverless offering
because we don't manage any servers.
AWS will just run the containers that we need
based on the specification
of CPU and RAM for each container.
So Fargate is going to be a lot simpler to use
and I like it personally a lot.
Fargate is here.
Say we have to have a new docker container
to be run on Fargate, then Fargate will
automatically run that container for us.
We don't exactly know where, but it will be run.
And so the idea here is that with Fargate,
we don't manage any EC2 instances

and so it is easier to use.
So before with ECS, we first created our EC2 instances,
but with Fargate, we don't.
But both services will allow you
to run docker container on AWS.
Finally, for storing these docker images,
so that it can be run on the AWS,
you need to use a container registry.
And for this, we can use ECR,
which stands for elastic container registry.
It is a private docker registry on AWS,
and this is where you're going to store you docker images
so that it can be run
by the ECS service or the Fargate service.
So the second example,
we have ECR and Fargate.
We're going to store our images of our application
onto Amazon ECR, and then Fargate will be able
to look at these images and create a container from them,
and run them directly on the Fargate service.
So it could be one container here, one container there,
but this is ECR, so we could have multiple images as well,
creating different containers on Fargate.
So that's it.
Very, very simple.
So remember, ECS versus Fargate versus ECR,
that's all you need to know going into the exam.
I hope you liked it,
and I will see you in the next lecture.

So now, I want to formally introduce
the Serverless Section to you.
So what is severless?
Severless is a catchy word but it is a new paradigm
in which the developers don't manage the servers anymore.
They just do what they do best,
they just deploy the codes or they deploy functions.
So initially, serverless was pioneered
as a Function as a Service with AWS Lambda,
and that means that you just deploy your code
and each function will be run independently
by the Lambda service, but nowadays,
anything that's serverless
is mostly mentioned as something that is managed
and that does include servers managed by the user
so that includes serverless databases,
messaging, storage, et cetera.
So serverless does not mean that there are no servers.
There are, of course, servers behind the scenes.
Otherwise, services would not work,
but it just means that as an end user,

you don't manage, provision, or even see the servers.
So in this course,
we have been using some serverless services
from the beginning.
So Amazon S3 was one of them
because we used it as a storage layer,
but we didn't manage any servers at all, okay?
Amazon S3 can scale infinitely, there was no servers,
it was just uploaded file, and that was it.
DynamoDB was another one.
So in DynamoDB, we just went ahead, we created a table,
but we didn't provision a server for that table,
and that server was, that table could auto scale
based on the load it was receiving.
So that makes it another serverless service.
Fargate is another one
so Fargate was to run Docker containers,
and as I said with ECS, you create EC2 instances
to run the Docker containers
so that would not be serverless, but with Fargate,
you just send the Docker containers
and Fargate will automatically find a way for it to be run
so that makes it another serverless service,
and in this section, we're going to see Lambda,
which was the pioneer of serverless services.
So Lambda allows you to run functions in the cloud.
So this is it for the quick intro to serverless,
I will see you in the next lecture to talk about Lambda.

Okay, so now let's talk about AWS Lambda.
So if we use an EC2 instance,
we have a virtual server in the cloud
but we are bounded by the amount of memory
and CPU power we give it.
It is continuously running
even though sometimes we don't use it.
And if we want to scale we can use an autoscaling group,
but that means that we need to add
or remove servers over time.
That may be a little slow
or there may be sometimes very complicated to implement.
With Lambda this is a new way to think about it.
In this case, we don't have servers,
we just have virtual functions
and these functions are limited by time.
So they're intended for shorter type of executions.
They will run on demand.
So that means that whenever we run a function,
it will be there to be run.

But whenever we don't need a function,
it will not be run and we will not be built for it.
And in case we need scaling, it's already automated
as part of the Lambda service
and this is why Lambda is a very popular service from AWS.
So the benefits of using AWS Lambda
is that the pricing is first of all super easy.
You're going to pay per request and per compute time.
And the free tier is also very generous.
So you get every month one million Lambda invocations
and 400,000 gigabytes seconds of compute time.
What this means is that you can run on Lambda
some pretty good services for free.
Now it is integrated with the whole AWS suite of services.
So we have integration with so many of the services
we've seen so far and it is very important event-driven.
So the functions will only get invoked by AWS
when something happens,
when an event happens or when needed.
So that makes Lambda the reactive type of service
which is important going into the exam.
It is fully integrated with many programming languages.
You get easy monitoring through CloudWatch.
We haven't seen what a CloudWatch is
but it will be the monitoring solution in AWS.
And finally, it's easy to get more resources per function.
We can get up to 10 gigabytes of RAM per function.
And if you do increase the RAM, it will also improve the CPU
and the network quality, so all in all very good.
Lambda can run many languages.
It can run JavaScript through Node.js,
Python, Java, C#, Golang, C# PowerShell, Ruby
and any language you want through the Custom Runtime API.
And one last runtime is called the Lambda Container Image
and this allows you to run actual Docker containers
on top of Lambda but these container images must implement
the Lambda runtime API,
which is not the case for every single Docker image.
So they are very very specific Docker images
that you can run on Lambda.
And so coming from an exam perspective,
you don't have to remember all the languages up there
but still is good for you to have a list.
But if it comes to running Docker images on AWS,
ECS and Fargate is going to be a preferred way
of running these arbitrary Docker images
so any kind of Docker images
but the Lambda Container Image does exist
if the Docker image is compliant
with the Lambda runtime API.
They maybe too specific for the exam

but I wanted to let you know of this detail.
Here is a very common use case of Lambda
which is to create a serverless thumbnail creation service.
So say we have an S3 buckets and we add images in it.
So our users are uploading a beach image into S3 buckets.
The S3 buckets will trigger a Lambda function
once the image is uploaded
and that Lambda function will take that image
and will change it to create a thumbnail.
It will push the thumbnail back into Amazon S3.
So the thumbnail is a smaller version of the image
or it will also push some metadata
about the thumbnail into DynamoDB.
That includes the image size, the image name,
the creation dates, etc, etc.
And all of this is fully event-driven and fully serverless.
With S3 we don't provision servers.
With Lambda we don't provision servers
and we've been in with Dynamo DB as well
we don't provision any servers.
So that is a great pattern because this serverless
thumbnail creation will scale it really, really well.
And we will be able to not worry
about provisioning servers to make it scale.
Now there's another very common use case for Lambda
which is to create a serverless CRON Job.
So CRON allows you to define a schedule
for example, every hour, every day or every Monday
and based on that schedule to run a script.
And by default, a CRON Job is run on an Linux AMI
so on a Linux machine.
But we are serverless
so we cannot provision an EC2 instance.
So instead we'll be using something
called CloudWatch Events or EvenBridge
and this service that we'll see later on in this course
will be triggering every one hour our Lambda function
to perform a task and effectively we have no servers in this
because CloudWatch events is serverless
and Lambda is serverless.
And so effectively we are launching a script
every hour through a Lambda function.
So I hope you can see now the trigger of it
the Lambda functions is really for serverless
functions in the cloud.
Now let's just talk about the pricing.
So you can find the Lambda pricing at this URL,
but it's very simple.
You pay per call.
So that means the first one million Lambda invocations
are free and then it's also very, very cheap.

You're going to pay $0.20 per 1 million requests thereafter.
You also going to pay for the duration.
So the free tier as I said is 400,000 gigabytes
seconds of compute time for free.
And that means it's 400,000 seconds
if the function has one gigabyte of RAM
or 3.2 million seconds
if the function has 120th megabyte of RAM.
After that you're going to pay $1
for 600,000 gigabyte seconds.
So all in all the bottom line
is that it's going to be very cheap to run Lambda on AWS.
And so it's a very popular service
to run your serverless applications and websites.
And going into the CCP exam,
you need to remember that Lambda pricing
is based on calls and duration.
So that's it for this lecture, I hope your liked it
and I will see you in the next lecture.

Okay,
so we're going to practice using the Lambda service.
And when you go in Lambda, you may end up on this screen,
but there's another screen I really like.
And so if you're in your URL,
you replace /discover by /begin,
you may end up on this screen,
which I really like because it has an educational value
I want to show you.
So Lambda is here to help you run code
without thinking about servers,
and this makes it a truly serverless service.
So the idea is that we can use any type
of programming language we want.
For example, .Net Core, Go, Java, Node JS, Python, Ruby,
or any custom runtime,
if you want to have a runtime provided by an open source,
for example, the Lambda Rust Runtime is possible.
And so from this code, you will have it written,
upload it into the Lambda console,
and then AWS Lambda will run the code for you.
So let's take a very simple code.
For example, Python, we have a Lambda handler,
it's going to print the event,
and then say, "Return Hello from Lambda."
So let's click on Run and we get Hello from Lambda.
So that means that the Lambda function just ran the code
that we have provided right here.
Very simple, right?
Next, how does Lambda functions get invoked?
So we can click on the Run button obviously,

but also, we can have Lambda respond to events.
And this is what I want to show you,
I think it's really cool.
So as we can see, events can come from various sources.
For example, in this one, this is streaming analytics.
And so as the streaming analytics is sending events
into a Lambda function,
the Lambda function is returning Hello from Lambda,
Hello from Lambda, Hello from Lambda,
and so on.
But it's not just those streaming analytics.
If you click on the phone right here,
it's going to send a message into a mobile IoT backend.
And this IoT backend will also invoke our Lambda function.
Same for if we take a photo,
and upload it into an S3 bucket for data processing,
then the Lambda function will be invoked.
But the cool thing is that
if you start clicking a lot on one of these sources,
you can see on the right hand side, we have more cog wheels.
So as the left hand side flow,
and invocations of Lambda scales,
then the number of Lambda invocations,
and concurrent Lambda functions running
will be scaling as well.
So it's really cool because that means that
as we have more load,
automatically Lambda will scale with our load.
And that is a whole power of using Lambda
as a compute platform,
and this is why Lambda is serverless,
and it scales really well.
So if we go in here, as we can see,
as the Lambda function is invoked,
we get more invocations over time,
and the cost remains zero because there's a free tier.
And as soon as we will pass 1 million invocations,
then the Lambda function will start incurring some charges.
So if we go in there and start having
over 1 million invocations, here we go,
now, we're getting some cents.
As you can see, we are at almost 2 million invocations,
and we have only 14 cents as a cost.
So it's very, very, very cost-efficient
to have a Lambda function and to run some workload at scale.
So as you can see, you can play around,
and seeing that the more you have invocations,
the more the cost goes, but it's really, really controlled.
And Lambda is, can be quite a cost-saving mechanism
if you use it at scale.
So this is just for the introduction to Lambda.

So if I click on create a function,
I can choose between three options,
and I'm going to choose to use a blueprint for this one.
And it's going to be the hello world function.
So I'm just going to search by hello world,
and then I will choose the Python version.
So Python 3.10 in this example,
but this could be a different version for you,
as long as it's Python.
Next, for the function name,
we're going to enter demo-lambda,
and then we're going to create a new role
with basic Lambda permissions.
This is going to be the IAM role of our Lambda function.
Then we can have a look at the function code.
So this is what a Lambda function looks like.
So we have a Lambda handler.
This is the function actually handles the event
of invoking our Lambda function.
And right now, it just prints a bunch
of values and returns the key1.
So we'll have a look at this in details
when we test our function.
So when you're ready, just click on create function.
Okay, so my function is created and as we can see,
the code has the code source in here.
And if I click on lambda function.py and open it,
we can see that the function code we had
from before is now loaded into this code editor.
So why don't we go ahead and test the function.
So I'm going to click on test and we need
to create a new test event, which is a hello world event,
which contains key1, value1, key2, value2, key3, value3
as adjacent documents.
So I'll call this one DemoEvent,
and click on Create.
So now, the demo event was successfully saved.
So if I click on test now,
it was going to run the DemoEvent,
and the response is value1,
the function log is that value1 equals value1,
value2 equals value2, value3 equals value3,
which is just a result of this, three print statements.
And finally, the response value is value1,
again, due to this line of code.
So it may not look like a match,
but from a programmer's perspective, as you can see,
you had just some code and it was uploaded into Lambda,
and then it was run by Lambda very quickly.
So this is a huge improvement if you're a developer,
as you can see, to deploy the code and have it run.

And on top of it, it runs seamlessly,
and it will scale automatically and it is fully serverless.
Okay?
We didn't deploy any servers.
Now, in terms of build duration,
so if you go in here and scroll down, sorry.
Then as you can see, the duration was 2.32 millisecond.
We've been billed for three millisecond of execution,
and here's the memory size that we've provisioned,
and the one that was used and how much the init was,
because this is the first time that
we used our Lambda function.
Now, if I run it again, oops,
if I go back to my function, excuse me,
and run this one again, as we can see now,
the function duration was 1.33 millisecond.
And on the right hand side, there was no initiation,
because my Lambda function was ready to be used.
Okay?
So that's one way of doing things.
Now, the other thing I'm going to show you is that
from this Lambda function, we're able to configure it.
So if I go into general configuration,
we get some of the most important settings.
The first one is around memory.
So we can have the memory from anywhere
between 128 megabytes up to 10,240 megabytes of memory.
Obviously, if you have more memory,
you're going to get billed more.
In terms of timeouts,
we can go anywhere between three seconds or five seconds,
all the way to, as you can see, 15 minutes.
So the maximum timeout is 15 minutes,
but you wanna make sure that you're only using the function
for the time that you think it's going to be used for.
And then the execution role is the one that was created
by Lambda in the beginning.
Okay, so those are some of the most important settings.
Okay?
And the other thing we can look at is monitoring.
So in monitoring, we're able to see
what is going on with the Lambda function,
so how many times it was invoked.
So here, one time, how long it lasted,
whether or not there was errors or successes and so on.
So it could be quite helpful.
But we have integration with CloudWatch,
metrics and also CloudWatch Logs.
So right now, we have nothing but we could look
at the CloudWatch Logs right now when the function runs.
So to do so, just refresh on the right hand side

the recent invocations and we can see
that there's a log stream right here.
So if I click on it,
I am taken directly into the CloudWatch Logs,
and we can see that on the CloudWatch Logs,
we have all the logs of the invocation
of this Lambda function.
And this is within a log group
called AWS Lambda, Demo Lambda,
and Demo Lambda is the name
of my Lambda function within my region, obviously.
And we have one log stream right here.
So we get all the logs into Lambdas.
And other things you can try is to modify the code.
For example, if we take this code,
we go to the Lambda function,
and I will comment this line of code
by having a hashtag and uncomment this one,
this is going to raise an exception.
So to do so, I need to first deploy the changes
by clicking on Deploy.
Now, the changes have been deployed,
and now, I can test my function.
And in this type, we get a execution result,
which is an error where something went wrong,
type is exception, as we can see,
something went wrong which was triggered
by this code or line of code right here.
So any type of errors also will be reported by Lambda,
and you could look into CloudWatch Logs
to understand as well where the log of the error happens.
So if I go back into CloudWatch Logs,
there's a second log stream right here,
which I'm going to open and we get
this exception right here.
So it's possible for us to go back as well to the logs
of the exception within CloudWatch Logs
to understand the root cause of the issue.
So fairly easy.
Now, if you wanna make the function run again fine,
you just reverse what you did and you click on Test.
And again, now, this time,
this Lambda execution function will work.
So some of the last things we may want to check out
is the fact that in here, the runtime settings,
we're running Python 3.7 in this example,
but you may get an updated version maybe on your end.
And the handler is lambda function.lambda handler,
which is saying to look at the lambda function.py file,
and the function named lambda handler.
So if you go up, you can see the function name

is the lambda_function.py,
and within it, we have the lambda_handler function.
So this is why it knew to invoke this function particularly.
Also, we have written to CloudWatch Logs.
And the reason we were able to do so
is that if we go into the configuration of our Lambda,
and go to the permissions tabs,
we have a role name called demo lambda role
that was created for us by the Lambda console.
And if you look at the policy itself,
we can see within the policy summary
that we have access to write to CloudWatch Logs,
which is something you can also see right here
by resource summary,
that CloudWatch Log has three actions and two resources,
which allow us to create log stream,
and log groups and also send the logs to CloudWatch Logs.
So this is the whole idea
behind this permissions tabs right here.
So that's it for this lecture.
I hope you liked it.
And if you are in the certified developer course on AWS,
then get ready for an extra few hours of content on Lambda.
And if you're not, well, this is enough for your exam.
I hope you liked it, and I will see you in the next lecture.

Now let's talk about a quick service
called the Amazon API Gateway.
So this comes from the use case from an exam perspective
of you wanting to build a server less HTTP API.
So in this example, we have server less technologies
so we're using Lambda,
and we're Reading, Creating, Updating and Deleting data
from DynamoDB,
which are both server less technologies
but we want external clients to be able to access our Lambda
function.
But a Lambda function is not exposed as an API right away.
For this we need to expose it
through an API Gateway
which is going to provide the client with the rest HTTP API
to connect directly to your website.
And so, as we can see the client
will talk to the API Gateway.
The API gateway will proxy the request
to your lender functions
which will execute the transformations on your data.
And so API Gateway is used
as a fully managed service
that is going to allow developers
to easily create, publish, maintain,

monitor, and secure APIs in the cloud.
It is a server less technology,
and it is fully scalable.
It supports RESTful APIs,
and also WebSocket APIs for real time streaming of data.
It supports also security, user authentication,
API throttling, API keys, monitoring, and so on.
So when you see going to exam
which creates a server less API
you need to think about API Gateway and Lambda.
That's it for this lecture.
I hope you liked it.

So now let's talk about a service that is named after
what it does it is AWS Batch.
So batch is a fully managed batch processing service
that can allow you to do batch processing at any scale.
And with the batch service,
you can efficiently run hundreds of thousands
of computing batch jobs on AWS very easily.
So what is a batch job?
Well, a batch job is a job that has a start and an end.
And that is opposed to say, a continuous
or a streaming job that really doesn't ever end
it's always running.
But a batch job say,
for example, starts at 1 a.m. and finishes at 3 a.m.
So a batch job has a point of time when it happens
and so the batch service will
dynamically launch EC2 instances or Spot Instances
to accommodate with the load
that you have to run these batch jobs.
So batch will provision the right amount of compute
and memory for you to deal with your batch queue.
And you just submit or scheduled batch jobs
into the batch queue and the batch service does the rest.
Now how do you define a batch job?
Well, it is simply a Docker image
and a test definition that you run on the ECS service.
So this is pretty much saying that anything
that can run on ECS can run on batch.
And this is going to be very helpful to use batch
to run these batch jobs.
And because it automatically scales
the right number of ECS2 instances or Spot Instances,
to do these jobs,
then you get lots of cost optimizations
and you focus a lot less on the infrastructure,
you just focus on your batch jobs.
So this should be more than enough for going to the exam,
but I just want to show you a small diagram that I made.

So for example, say we wanted to process images submitted
by users into Amazon S3 in a batch way.
So image will be put into Amazon S3,
and this will trigger a batch job.
And so batch will automatically have
an ECS cluster made of EC2 instances,
or Spot Instances and batch would make sure that
you have the right amount of instances
to accommodate the load of batch jobs
you have in the batch queue.
And then these instances will be running
your Docker images that will be doing your job.
And then maybe that job will be to insert
the processed object.
Maybe it's a filter on top of the image
into another Amazon S3 buckets.
So the question you may have is
what is the difference between batch and Lambda
because they look similar?
So Lambda has a time limit,
it's 15 minutes,
and you only get access to a few programming languages.
On top of it, you have limited temporary disk space
if you want to run your jobs,
and it's going to be serverless,
whereas batch is very different.
So batch has no time limit,
because it relies on EC2 instances.
It's any runtime that you want as long
as you package it as a Docker image.
And for storage,
you rely on the storage that comes with an EC2 instance.
So it could be an EBS volume,
or an EC2 instance store for disk space,
which can be a lot more than for Lambda functions.
And then finally, batch is not a serverless service.
It's a managed service,
but it relies on actual EC2 instances being created.
But these EC2 instances are managed by AWS
so we don't have to worry
about the auto scaling and so on.
So I hope that was helpful
and I will see you in the next lecture.

Amazon Lightsail is a bit of an oddball in AWS
because it is kind of a stand-alone service.
So with Lightsail,
you can get virtual servers,
storage,
databases
and networking in one place.

It gives you low and predictable pricing.
And so the reason people use Amazon Lightsail
is that it is a much simpler alternative
to using services
that we've been learning
such as EC2 RDS, ELB, EBS, Route53 and so on.
So Lightsail really the intent of it from AWS
is for people that have little cloud experience
and don't want to learn
how the services work intricately,
for example,
how their networking works,
how the storage works,
the server works etc.
This is not something you would use
your learning how AWS works properly
but if you're someone with little
to no cloud experience
then Lightsail maybe a service for you.
From Lightsail you can also set up some monitoring
and notifications for your Lightsail resources
and so use cases for Lightsail would be
for example,
if you want to deploy a very simple web application
for example,
if you have LAMP Stack, Nginx, MEAN, Node.js
there's templates for that on Lightsail
or if you have a very simple website,
for example,
if you have a WordPress and Magento,
Plesk, Joomla
all these kind of things can be deployed
very easily on Lightsail.
Finally, Lightsail is also great
if you have development and test environments in AWS.
There is a concept of high availability in Lightsail
but there's no auto-scaling
and there really are limited AWS integrations.
So to summarize,
Lightsail really is something
that you wouldn't use today on your own
but from an exam perspective
if you see someone
that has no cloud experience
and need to get started quickly
with a low and predictable pricing
without configuring things much
that will be Lightsail,
otherwise Lightsail will almost
always be a wrong answer.
So I hope that was helpful

and I will see you in the next lecture.

So I want to quickly show you
what Lightsail looks like
we're not going to play much with it.
But I want to show you what it looks like.
So Lightsail, as you can see, looks nothing like AWS,
it is a separate service
because it doesn't have any kind
of good integrations with AWS.
So let's get started with Lightsail.
They need you to create an instance.
And so say, where is this going to be.
And so as you can see, you can change the region
and the AZ from here for that instance.
And there's limited amount of regions available to you.
So for example, we can set at one in Ireland,
then based on where you select your instance to be,
you're going to get some parameters
to configure your instance,
but you're going to get a lot less.
so you can pick an instance image
which would correspond to an AMI a but again,
all these things are hidden from you in lightsail.
And so for example, we can say okay,
we want to use a blueprint such as a template,
and we can use the OS only.
This would be very similar to EC2,
but we can go with App NOS for example.
We can say okay,
I want to launch a WordPress websites right now.
and you click on this And you have a WordPress website.
So as you can see,
lightsail is a much lighter as the name indicates.
And it's really meant for you to be quickly up
and going on AWS without understanding,
the entire infrastructure that goes behind it.
So this is for very simple applications.
So we can have an optional launch grid,
which is an EC2 use data,
we can change the SSH key pair if you wanted to.
But you can see all these things are kind of hidden.
They don't want you
to modify these parameters well, right away.
The cool thing here is the instance plan.
So we have a lot less customization here,
we don't see the EC2 instance types,
we just see some instances with a price indicated,
so it's easy to understand the price is going to cost me
For example, this one will cost me 3.5 USD,
but it's the first month is going to be free.

So maybe that's the one I want.
And we can see how much RAM CPU I get and disk space.
So this is really, really easy to use.
As you can see,
we don't have that many instance types possible.
We can go all the way to one
that costs 160 US dollars per month,
and we see 32 gigabytes of memory, eight VCPU 640 ASG
and seven terabytes of network transfer,
so fairly easy, I'm going to just pick this one
because it's in the free tier,
and then you have to make sure you delete at the end.
So it's going to be WordPress one,
and it will create this instance.
So we can see here, no security groups, no networking,
no EBS volumes, really, it's meant to be very, very simple.
And that's the main idea behind lightsail.
So while this instance is pending, again,
we can go ahead and create,
we could go ahead and create a database.
So you could create database and again,
this is not RDS, this is lightsail.
So it's going to be much more easy.
You choose again, the region
and the AZ you pick the database type.
And then you can specify some credentials for the login,
if you want to standard database or high availability,
and then the price you're willing to pay.
And again, this database right here,
will get you the first month three,
but then you would be $15 per month.
So as you can see, again,
it's a very simplified version of AWS,
hence the name lightsail.
We could do the same with networking.
I won't show you right now.
But you can create all these three things,
including a load balancer,
In the networking section,
we could do storage to get some more storage disk
and snapshots for your backups.
So this is a very simplified one view of AWS.
Now let's look at our WordPress.
So here, if I click here,
this actually gets me into an SSH terminal,
into my EC2 instance.
So I need to wait a little bit for it to boot up.
So if I now try again,
I'm able to connect to my instance using,
a version of SSH through the browser.
So again, it's really intended to be very easy to be used.

So we're not going to do any commands on the SSH.
But as we can see a WordPress was deployed.
And so I can click on it and view it
and then we can connect using SSH again from the same thing.
But if we look at the public IP that is running,
and open this in a new tab,
then we're going to connect into our WordPress instance.
And so we are getting our hello world blog.
And I'm not a WordPress expert.
So I'm not going to show you how to modify this,
but the idea is that yes,
Something is being created on our instance.
And that was very, very easy.
So lightsail, I hope you get an idea of what it is.
It's meant to be a lightweight version of AWS,
but it's not something you're going to use
in your job, I believe as a RDS developer,
as a subs lab practitioners, solutions architect, etc, etc.
Still a good idea to know about the service
and it can come up at, the exam, usually as a destructor.
Very, very rarely as a service you should get the answer of.
So that's it when you're done,
please make sure to delete that instance.
So you click on delete,
and then delete the instance
so that you don't incur any charges.
So that's it. I hope you liked it,
and I will see you in the next lecture.

Okay, so let's do a summary of all
these new compute services we've just learned about.
And the first thing is that we learned about Docker,
which is a technology,
a container technology that allows you to run applications.
And we've seen how to run Docker on AWS.
The first way is to use ECS.
And this is going to allow you to run your Docker containers
on EC2 instances, but you have to provision
these instances in advance.
For Fargate, it is the exact same way
but this time we run the Docker containers
without provisioning the infrastructure,
it is transparent to us.
And then next Fargate, a serverless offering
because we don't manage any EC2 instances
to run these Docker containers.
These Docker containers can be stored on the AWS using ECR,
which has a Private Docker Images Repository.
And we've also seen about the batch service.
Batch allows you to run batch jobs on AWS
across a set of managed EC2 instances.

And actually, the batch service actually runs
on top of the ECS service.
We've finally seen about a new type of service
to run predictable and low pricing simple applications
and database techs, which is LightSail,
which is most likely going to be a distractor at the exam
but we've done a hands-on, and now we knew what to expect
and how the service works.
So now let's talk about Lambda.
Lambda is going to be a serverless service
which is going to give you
a capability of function as a service,
and it's going to have seamless scaling.
So from one invocation
to thousands of invocation per second,
and it's fully reactive.
You have two components for building of Lambda.
You have by the time run,
times the amount of memory provisions
for the Lambda function.
And also by the number of times your Lambda function
has been invoked.
In terms of language supports,
it supports many different kinds of programming languages.
And even though it supports container images
and then need to implement a specific runtime API.
And so I like to say that Lambda does not support
arbitrary Docker images,
for this you would use ECS and Fargate,
but if your Docker image does implement
the Lambda container Runtime API,
then you can run Docker images on Lambda,
but this is not the standard again.
The invocation time is up to 15 minutes
and the use cases for Lambda is to create funnels
for images uploaded onto Amazon history, for example,
or to run a serverless Cron job.
Finally, if we wanted to expose our Lambda functions
as APIs, we would use another serverless service
called the API Gateway,
that would allow us to expose our functions as HTTP APIs,
but also give us capabilities around security, front Ling,
API keys, and so on.
So that's it for this section, I hope you liked it.
And I will see you in the next section.

#______________________________________