**Amazon Elastic File System (Amazon EFS)**

**Video transcript**

–

Next up on the list of storage services is Amazon Elastic File System, or what we call EFS. EFS is a managed file system. It's extremely common for businesses to have shared file systems across their applications. For example, you might have multiple servers running analytics on large amounts of data being stored in a shared file system. This data traditionally has been hosted on premises. In this on-premises data center, you would have to ensure that the storage you have can keep up with the amount of data that you are storing. Make sure backups are taken, and that the data is stored redundantly as well as manage all of the servers hosting that data.

Luckily with AWS, you don't need to worry about buying all of that hardware and keeping the whole file system running from an operational standpoint. With EFS, you can keep existing file systems in place but let AWS do all the heavy lifting of the scaling and the replication. EFS allows you to have multiple instances accessing the data in EFS at the same time. It scales up and down as needed without you needing to do anything to make that scaling happen. Super nice, right? Well, you might be thinking, Amazon EBS also lets me store files that I can access from EC2 instances. What exactly is the difference here?

[Blaine] AWS Cloud Practitioners, welcome back!

[Morgan] All right, we don't need to do all of that. The answer is really simple. Amazon EBS volumes attach to EC2 instances and are an Availability Zone-level resource. In order to attach EC2 to EBS, you need to be in the same AZ. You can save files on it. You can also run a database on top of it. Or store applications on it. It's a hard drive. If you provision a two terabyte EBS volume and fill it up, it doesn't automatically scale to give you more storage. So that's EBS. Amazon EFS can have multiple instances reading and writing from it at the same time.

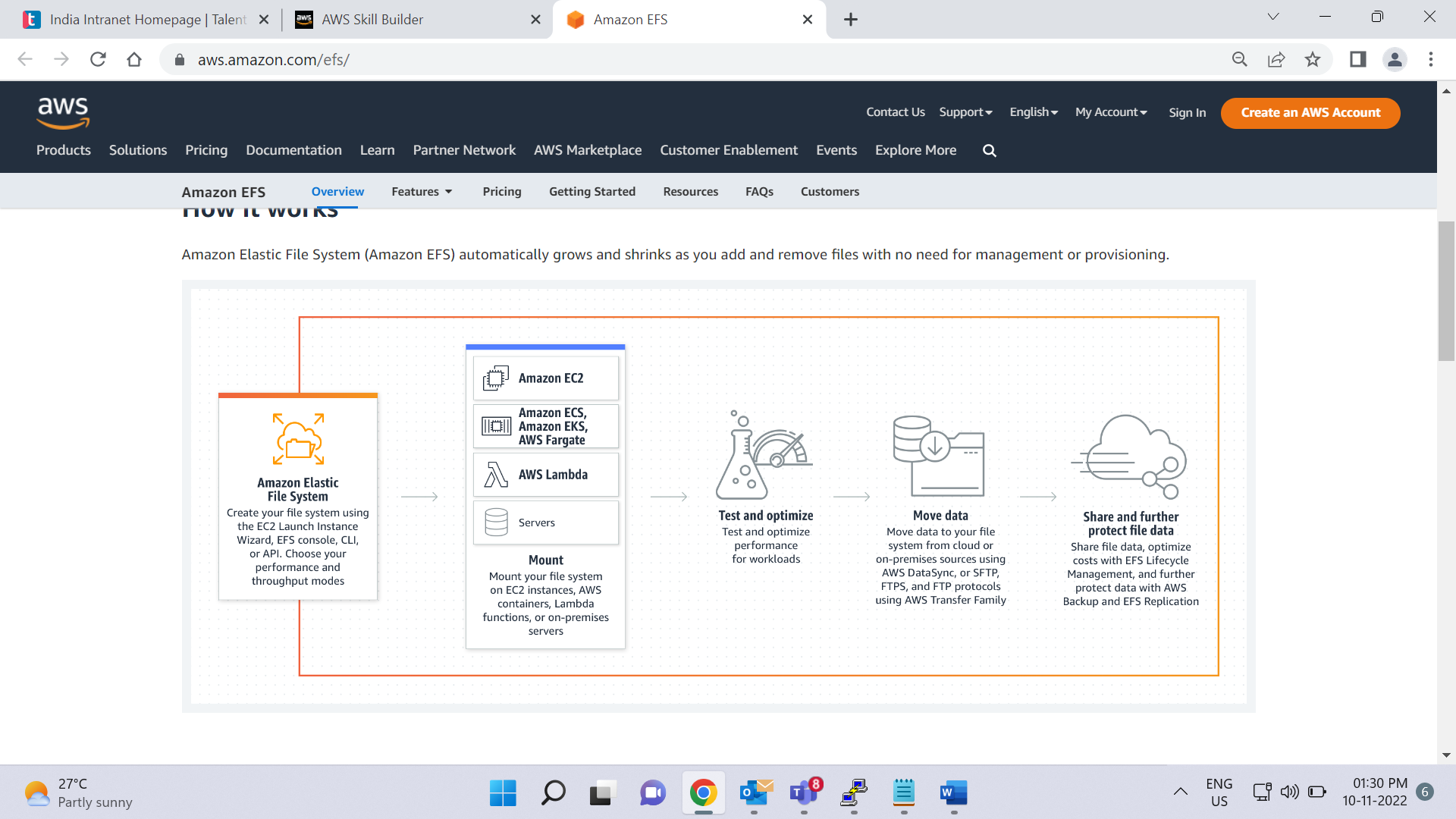
But it isn't just a blank hard drive that you can write to. It is a true file system for Linux. It is also a regional resource. Meaning any EC2 instance in the Region can write to the EFS file system. As you write more data to EFS, it automatically scales. No need to provision any more volumes.

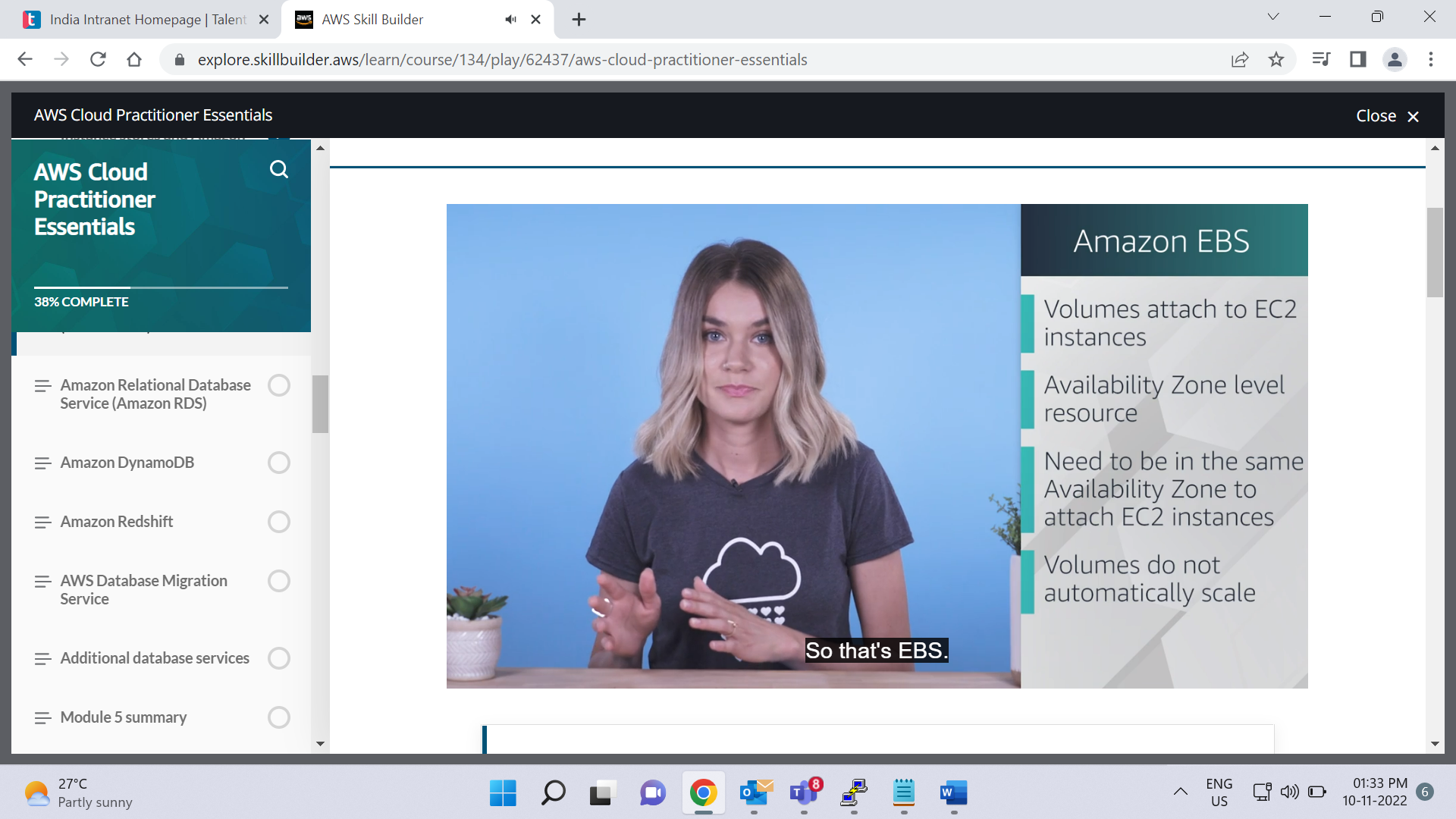
**File storage**

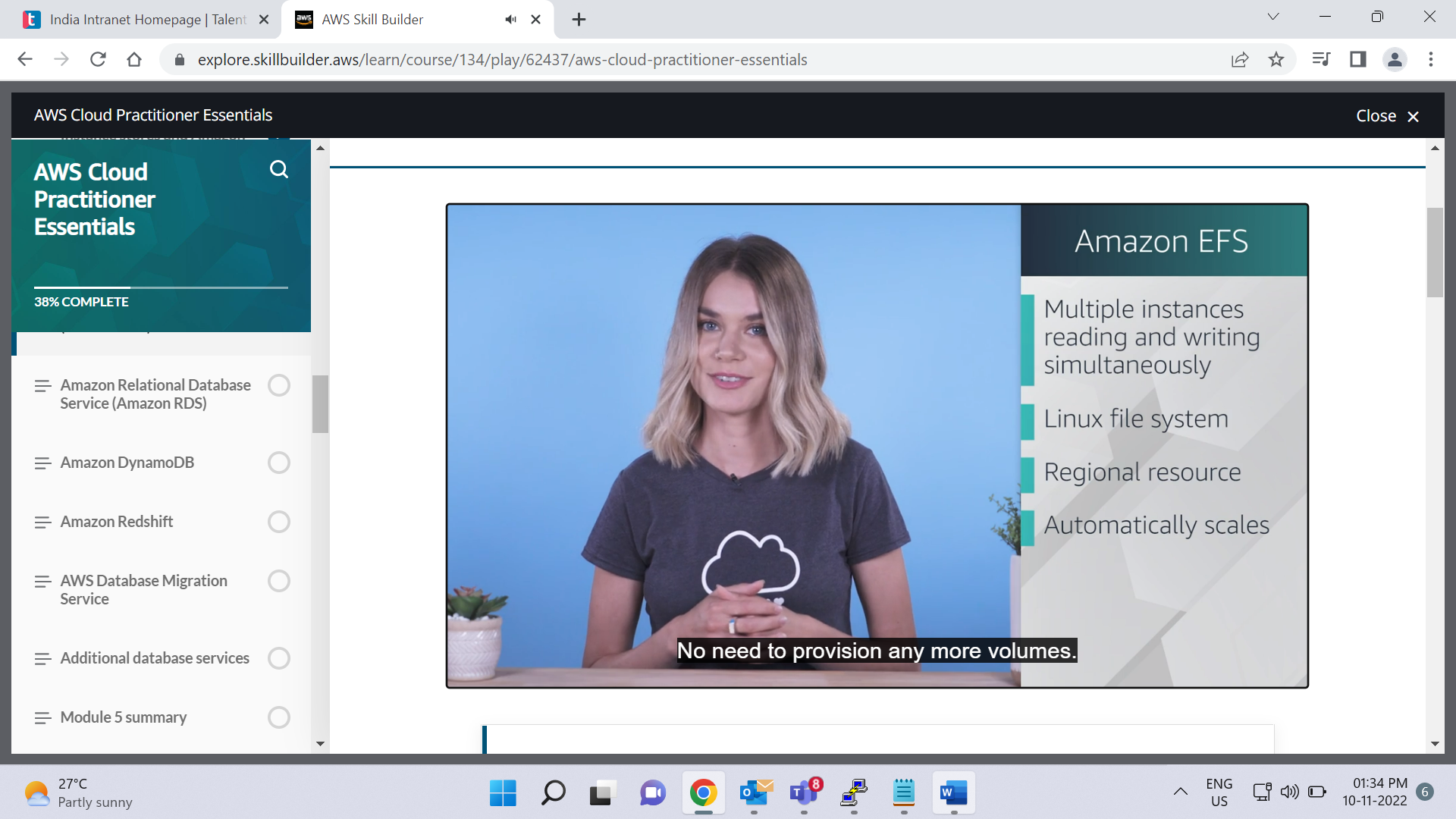
In **file storage**, multiple clients (such as users, applications, servers, and so on) can access data that is stored in shared file folders. In this approach, a storage server uses block storage with a local file system to organize files. Clients access data through file paths.

Compared to block storage and object storage, file storage is ideal for use cases in which a large number of services and resources need to access the same data at the same time.

[**Amazon Elastic File System (Amazon EFS)**](https://aws.amazon.com/efs/) is a scalable file system used with AWS Cloud services and on-premises resources. As you add and remove files, Amazon EFS grows and shrinks automatically. It can scale on demand to petabytes without disrupting applications.







**Comparing Amazon EBS and Amazon EFS**

Select each flashcard to flip it.

Amazon EBS

An Amazon EBS volume stores data in a **single** Availability Zone.

To attach an Amazon EC2 instance to an EBS volume, both the Amazon EC2 instance and the EBS volume must reside within the same Availability Zone.

Amazon EFS

Amazon EFS is a regional service. It stores data in and across **multiple** Availability Zones.

The duplicate storage enables you to access data concurrently from all the Availability Zones in the Region where a file system is located. Additionally, on-premises servers can access Amazon EFS using AWS Direct connect

# Amazon Relational Database Service (Amazon RDS)

**Video transcript**

–

So you're storing data about your coffee shop in various systems. But you're finding that you need to maintain relationships between various types of data. And by relationships, I mean, if say, a customer orders the same drink multiple times, maybe you want to offer them a promotional discount on their next purchase. And you need a way to keep track of this relationship somewhere. In this case, it's best to use a relational database management system, or RDBMS. Essentially, It means we store data in a way such that it relates to other pieces of data.

For example, if we had a customer entry or record, we store that in a customer table. We then could have an entry for the physical address, which we store on a corresponding address table. We then relate the two via a common attribute and can query the data that is housed in both tables.

The most common way to query the data is by writing queries in SQL. And this runs on a variety of database systems. Speaking of database systems, what are some of the more well known ones that AWS supports? Well, there's MySQL, PostgreSQL, Oracle, Microsoft SQL Server, and many more. If you have an on-premises environment, you're probably running one of those and they're most likely housed in your data center.

But is there a way to easily move them to the cloud? Well, the simple answer is yes, you can do what we call a Lift-and-Shift, and migrate your database to run on Amazon EC2. This means you have control over the same variables you do, in your on-premises environment, such as OS, memory, CPU, storage capacity, and so forth. It's a quick entry to the cloud, and you can migrate them using standard practices or using something like Database Migration Service, which we'll cover in a later video.

The other option for running your databases in the cloud is to use a more managed service called Amazon Relational Database Service, or RDS. It supports all the major database engines, like the ones we mentioned earlier, but this service comes with added benefits. These include automated patching, backups, redundancy, failover, disaster recovery, all of which you normally have to manage for yourself. This makes it an extremely attractive option to AWS customers, as it allows you to focus on business problems and not maintaining databases. Which if you're a database admin, can be pretty time consuming and difficult.

So how do we make it even easier for you to run database workloads on the cloud? Well, we go one further and have them migrate or deploy to Amazon Aurora. It's our most managed relational database option. It comes in two forms, MySQL and PostgreSQL. And is priced is 1/10th the cost of commercial grade databases. That's a pretty cost effective database. The other benefits are things like your data is replicated across facilities, so you have six copies at any given time. You can also deploy up to 15 read replicas, so you can offload your reads and scale performance. Additionally, there's continuous backups to S3, so you always have a backup ready to restore. You also get point in time recovery, so you can recover data from a specific period.

And there you have it, relational databases in a nutshell.

**Relational databases**

In a **relational database**, data is stored in a way that relates it to other pieces of data.

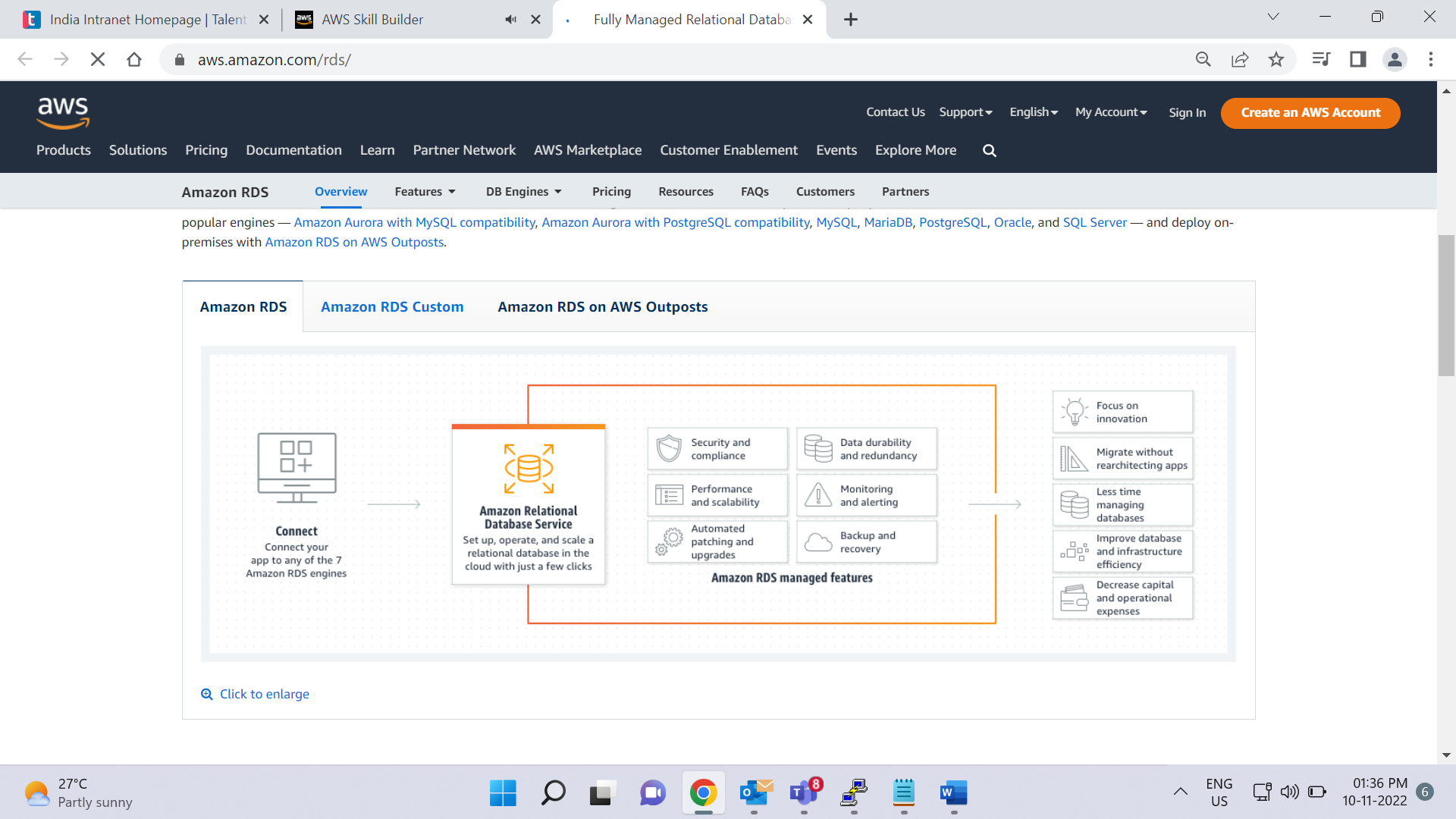
An example of a relational database might be the coffee shop’s inventory management system. Each record in the database would include data for a single item, such as product name, size, price, and so on.

Relational databases use **structured query language (SQL)** to store and query data. This approach allows data to be stored in an easily understandable, consistent, and scalable way. For example, the coffee shop owners can write a SQL query to identify all the customers whose most frequently purchased drink is a medium latte.

Example of data in a relational database:

| **ID** | **Product name** | **Size** | **Price** |
| --- | --- | --- | --- |
| 1 | Medium roast ground coffee | 12 oz. | $5.30 |
| 2 | Dark roast ground coffee | 20 oz. | $9.27 |

**Amazon Relational Database Service**



[**Amazon Relational Database Service (Amazon RDS)**](https://aws.amazon.com/rds/) is a service that enables you to run relational databases in the AWS Cloud.

Amazon RDS is a managed service that automates tasks such as hardware provisioning, database setup, patching, and backups. With these capabilities, you can spend less time completing administrative tasks and more time using data to innovate your applications. You can integrate Amazon RDS with other services to fulfill your business and operational needs, such as using AWS Lambda to query your database from a serverless application.

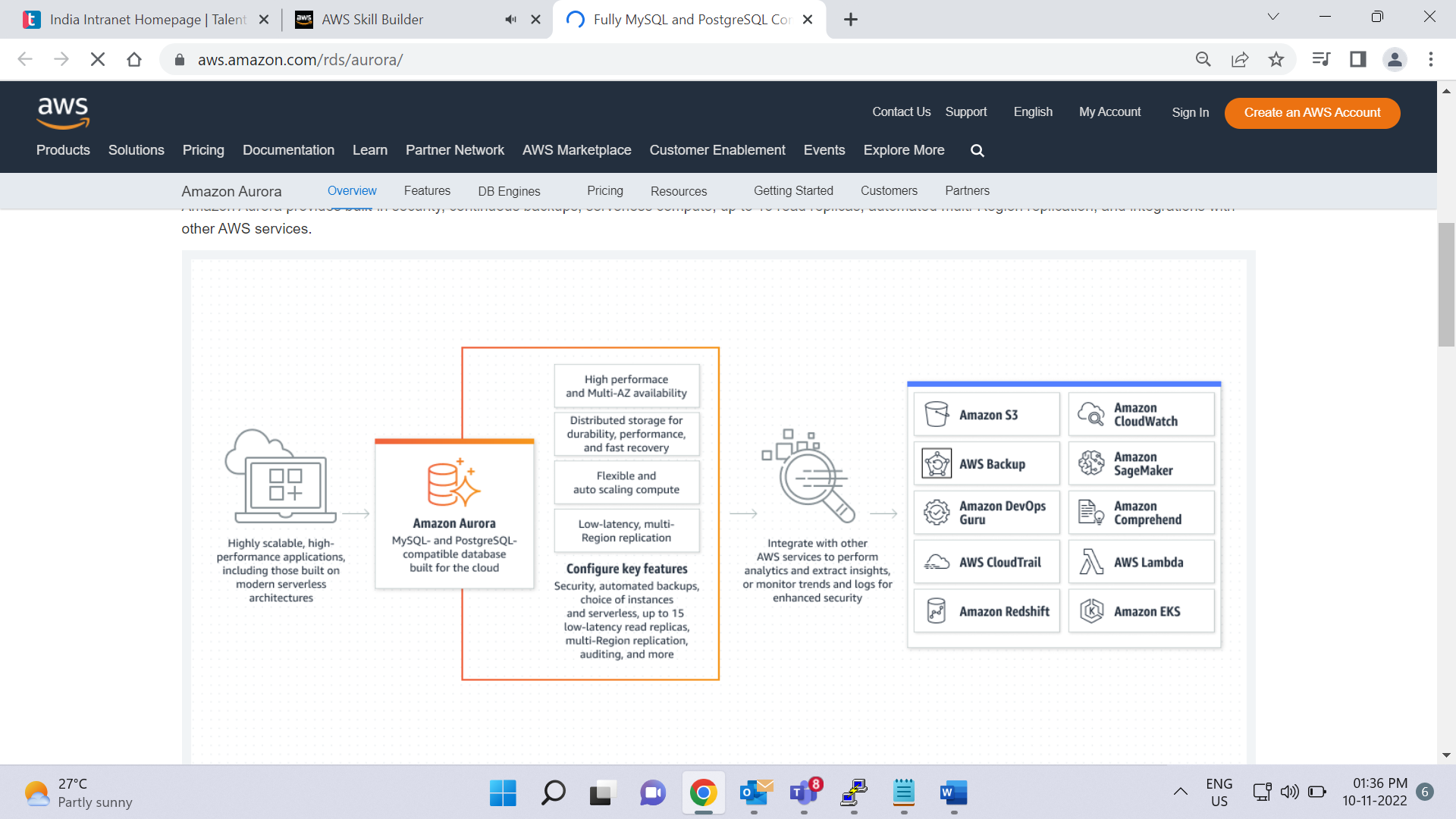
Amazon RDS provides a number of different security options. Many Amazon RDS database engines offer encryption at rest (protecting data while it is stored) and encryption in transit (protecting data while it is being sent and received).

**Amazon RDS database engines**

Amazon RDS is available on six database engines, which optimize for memory, performance, or input/output (I/O). Supported database engines include:

* Amazon Aurora
* PostgreSQL
* MySQL
* MariaDB
* Oracle Database
* Microsoft SQL Server

**Amazon Aurora**



[**Amazon Aurora**](https://aws.amazon.com/rds/aurora/) is an enterprise-class relational database. It is compatible with MySQL and PostgreSQL relational databases. It is up to five times faster than standard MySQL databases and up to three times faster than standard PostgreSQL databases.

Amazon Aurora helps to reduce your database costs by reducing unnecessary input/output (I/O) operations, while ensuring that your database resources remain reliable and available.

Consider Amazon Aurora if your workloads require high availability. It replicates six copies of your data across three Availability Zones and continuously backs up your data to Amazon S3.

# Amazon DynamoDB

**Video transcript**

–

Let's talk about Amazon DynamoDB. At its most basic level, DynamoDB is a database. It's a serverless database, meaning you don't need to manage the underlying instances or infrastructure powering it.

With DynamoDB, you create tables. A DynamoDB table, is just a place where you can store and query data. Data is organized into items, and items have attributes. Attributes are just different features of your data. If you have one item in your table, or 2 million items in your table, DynamoDB manages the underlying storage for you. And you don't need to worry about the scaling of the system, up or down.

DynamoDB stores this data redundantly across availability zones and mirrors the data across multiple drives under the hood for you. This makes the burden of operating a highly available database, much lower.

DynamoDB, beyond being massively scalable, is also highly performant. DynamoDB has a millisecond response time. And when you have applications with potentially millions of users, having scalability and reliable lightning fast response times is important.

Now, DynamoDB isn't a normal database. In the sense that it doesn't use the very widely used query language, sequel, or SQL. Relational databases, like a standard MySQL Database, require that you have a well defined schema, in place. That might consist of one, or many tables that might relate to each other. You then use SQL to query the data.

This works great for a lot of use cases, and has been the standard type of database historically. However, these types of rigid SQL databases, can have performance and scaling issues when under stress. The rigid schema also makes it so that you cannot have any variation in the types of data that you store in a table. So, it might not be the best fit for a dataset that is a little bit less rigid, and is being accessed at a very high rate.

This is where non-relational, or NoSQL, databases come in. DynamoDB is a non-relational database. Non-relational databases tend to have simple flexible schemas, not complex rigid schemas, laying out multiple tables that all relate to each other.

With DynamoDB, you can add and remove attributes from items in the table, at any time. Not every item in the table has to have the same attributes. This is great for datasets that have some variation from item to item. Because of this flexibility, you cannot run complex SQL queries on it. Instead, you would write queries based on a small subset of attributes that are designated as keys.

Because of this, the queries that you run are non-relational databases tend to be simpler, and focus on a collection of items from one table, not queries than span multiple tables. This query pattern, along with other factors, including the way that the underlying system is designed, allows DynamoDB to be very quick in response time, and highly scalable.

So, things to remember: DynamoDB is a non-relational, NoSQL database. It is purpose built. Meaning it has specific use cases, and it isn't the best fit for every workload out there. It has millisecond response time. It's fully managed, and it's highly scalable. One awesome example is on Prime Day in 2019, across the 48 hours of Prime Day, there were 7.11 trillion calls to the DynamoDB API, peaking at 45.4 million requests per second. That's insanely scalable, all without the underlying database management. That's pretty cool.

**Nonrelational databases**

In a **nonrelational database**, you create tables. A table is a place where you can store and query data.

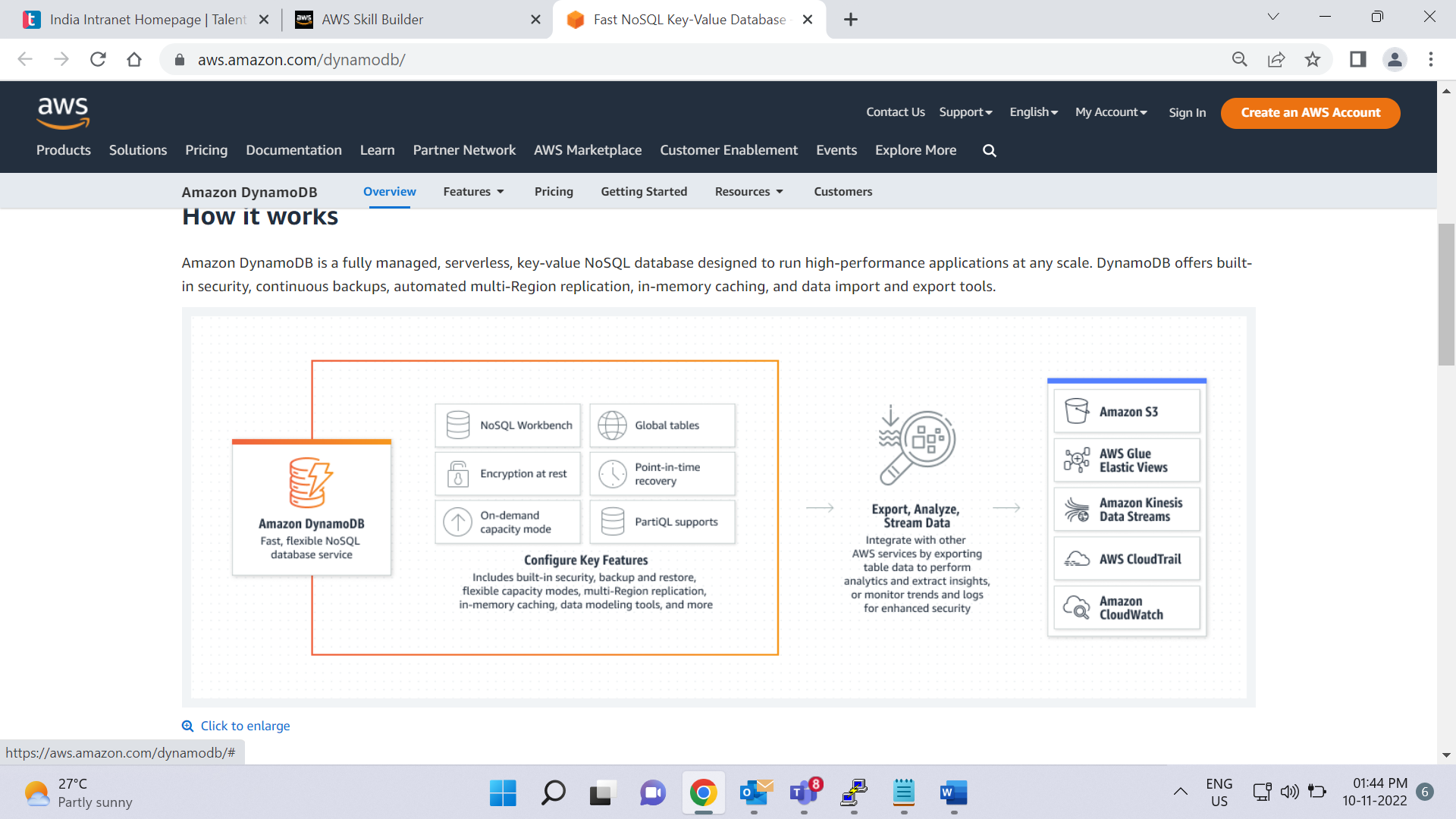
Nonrelational databases are sometimes referred to as “NoSQL databases” because they use structures other than rows and columns to organize data. One type of structural approach for nonrelational databases is key-value pairs. With key-value pairs, data is organized into items (keys), and items have attributes (values). You can think of attributes as being different features of your data.

In a key-value database, you can add or remove attributes from items in the table at any time. Additionally, not every item in the table has to have the same attributes.

Example of data in a nonrelational database:

| **Key** | **Value** |
| --- | --- |
| 1 | **Name**: John Doe  **Address**: 123 Any Street  **Favorite drink**: Medium latte |
| 2 | **Name**: Mary Major  **Address**: 100 Main Street  **Birthday**: July 5, 1994 |

**Amazon DynamoDB**



[**Amazon DynamoDB**](https://aws.amazon.com/dynamodb/) is a key-value database service. It delivers single-digit millisecond performance at any scale.

To learn about features of DynamoDB, select each flashcard to flip it.

Serverless

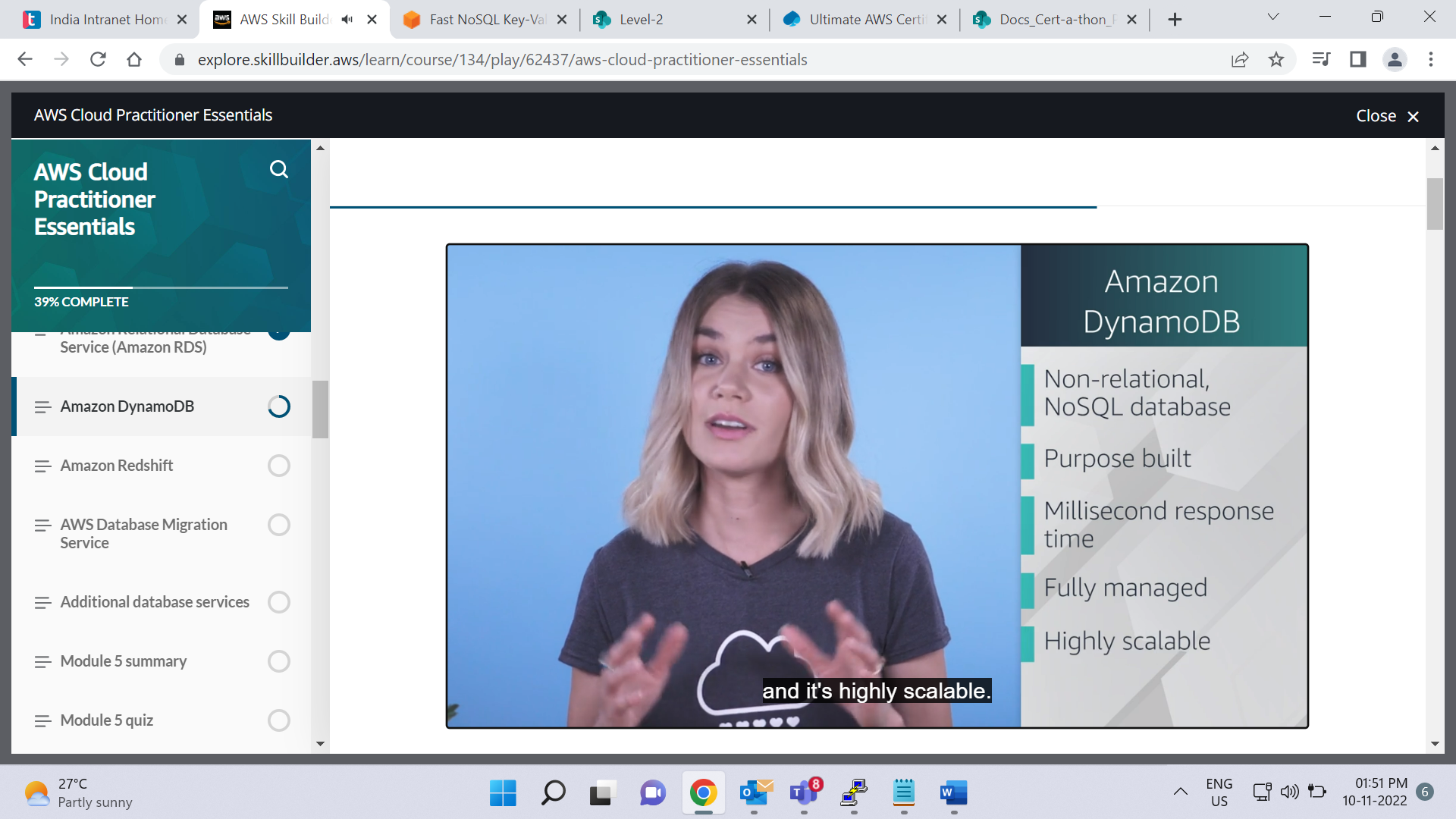
DynamoDB is serverless, which means that you do not have to provision, patch, or manage servers.

You also do not have to install, maintain, or operate software.

Automatic scaling

As the size of your database shrinks or grows, DynamoDB automatically scales to adjust for changes in capacity while maintaining consistent performance.

This makes it a suitable choice for use cases that require high performance while scaling.



**Video transcript**

–

AWS Cloud Practitioners, welcome back to the championship chase of the database! In the relational corner, engineered to remove undifferentiated heavy lifting from your database administrators with automatic high availability and recovery provided. You control the data, you control the schema, you control the network. You are running Amazon RDS. Yes, Yeah.

The NoSQL corner, using a key value pair that requires no advanced schema, able to operate as a global database at the touch of a button. It has massive throughput. It has petabyte scale potential. It has granular API access. It is Amazon DynamoDB.

Head to head. Each database class is engineered to exactly enhance exciting existential environments you envision. Which database will ultimately be victorious in this new world knock out night fight? Once again, the winner will depend on your use case.

Round one, Relational databases have been around since the moment businesses started using computers. Being able to build complex analysis of data spread across multiple tables, is the strength of any relational system. In this round, you have a sales supply chain management system that you have to analyze for weak spots. Using RDS is the clear winner here because it's built for business analytics, because you need complex relational joins. Round one easily goes to RDS.

Round two, the use case, pretty much anything else. Now that sounds weird, but despite what your standalone legacy database vendor would have you believe, most of what people use expensive relational databases for, has nothing to do with complex relationships. In fact, a lot of what people put into these databases ends up just being look-up tables.

For this round, imagine you have an employee contact list: names, phone numbers, emails, employee IDs. Well, this is all single table territory. I could use a relational database for this, but the things that make relational databases great, all of that complex functionality, creates overhead and lag and expense if you're not actually using it. This is where non-relational databases, Dynamo DB, delivers the knockout punch. By eliminating all the overhead, DynamoDB allows you to build powerful, incredibly fast databases where you don't need complex joint functionality. DynamoDB comes out the undisputed champion.

Once again, the winner depends on your individual workload. Each service is the right service for specific needs. And once you understand what you need, you will know again, which service is your champion.

**Knowledge check**

What are the scenarios in which you should use Amazon Relational Database Service (Amazon RDS)? (Select TWO.)

* Running a serverless database

Correctly unchecked

* Using SQL to organize data

Correctly checked

* Storing data in a key-value database

Correctly unchecked

* Scaling up to 10 trillion requests per day

Incorrectly checked

* Storing data in an Amazon Aurora database

Incorrectly unchecked

SUBMIT

**Incorrect**

The two correct response options are:

* Using SQL to organize data
* Storing data in an Amazon Aurora database

The other three response options are scenarios in which you should use Amazon DynamoDB.

