Welcome back. This is part two of this video. We're going to carry on immediately from the end of part one. So let's get started. Okay, so it did take a couple of minutes, but at this stage, we're now running with zero capacity units. If I open the cluster and look at logs and events at 00:58 GMT the cluster is being paused and then at 00:59 GMT the database cluster is paused. So it's now in a pause state and in this state, I'm only being charged for the storage that the cluster is using. So straight away I hope you can see the advantages of using Aurora Serverless because if you do allow it to go into a pause state, then you significantly reduce your running cost. Now, this cost reduction doesn't come for free. It does come with some tradeoffs. Now, to illustrate that, if I go to my EC2 instances and copied the DNS address for my web server and then open that in a new tab, you'd be able to see that it's not loading. So there isn't actually an active database instance to service that request. So when it's in a paused state, it does take a number of seconds to spin back up. It's not a lengthy time as you've seen there. It doesn't take that long to spin back up into an operational state, go back to the RDS console and just refresh this as soon as activity is detected, it begins the resume process and approximately 30 seconds later, the database cluster is resumed and ready again for activity. So because we're utilizing the shared storage architecture and because Aurora Serverless keeps this pool of instances ready to use, it is a really rapid process to reallocate instances back into our cluster. So this is critical to understand what's happening behind the scenes is that **AWS is maintaining a pool of hot instances of various different sizes. Now these instances can be allocated for our use or for other AWS customers. They're stateless because they have no attached storage. They just have the Aurora Serverless software installed on them. So the process of them being allocated to our cluster is simply there's a logical allocation process, and then this fleet of proxy instances is informed to be able to direct requests towards these instances and these instances, then have access to the storage, and I can safely load the cat pictures website.**

So because of the Aurora architecture in general, **because of this shared cluster storage, any instance operations are already really rapid and by moving towards Aurora Serverless, you get even more rapid access because AWS is maintaining this warm pool of instances.** So that's really critical to understand.

Now there are some limitations for Aurora Serverless. Essentially, **Aurora Serverless exists in a single availability zone. So if you do have any failure of any infrastructure that's supporting these capacity units then that has to be recreated. Now, it is going to be slower to failover than an Aurora Provisioned cluster because what has to happen is one of these warm instances needs to be added to the cluster and then configured to point at this storage. That is a process that does take longer than if you're doing a failover in a provisioned cluster.**

Now, of course, **a provisioned cluster does have ongoing costs 24/7 while it's running, and so it is a trade off between a slight increase in the amount of time that failover takes versus being able to scale back to zero capacity and then only pay for the storage. So, as with most things in AWS, you're trading between different priorities**. Now, in all of the considerations, as far as applications are concerned, **Aurora Serverless is just a normal database cluster.** This WordPress instance, for example, has no knowledge that it's not a traditional mySQL database. As far as it's concerned it's logging on using a username and password. It's using a certain port, and it's using an endpoint address. **Everything is handled by AWS behind the scenes to make this a really economic platform that can scale all the way down to zero if you don't have any incoming load and so, as I mentioned earlier in this lesson, it makes it ideal for Dev, for testing, for any applications that don't need to run outside business hours, or when you have really unpredictable low usage. Now Aurora Serverless cannot yet be set to be public, so it's not a drop-in replacement for DynamoDB and, of course, it is still a relational database service** versus DynamoDB which I'll talk about in the next topic of the course, which is a nonrelational database.

**Now the cluster, the actual instances that are used to provide these Aurora capacity units, they're not actually hosted inside your VPC. You've got no physical instances. What actually happens is that it uses a service known as private link and private link allows you to place endpoints inside of the VPC to access remote services.** So it's kind of like VPC endpoints that I've talked about earlier in the course. But why this matters is **that you can't currently access an Aurora Serverless cluster from across a VPN or an inter-region VPC peer** and both of those facts might be important for the exam.

**Two more benefits for using an Aurora Serverless cluster is that you can utilize the query editor and the query editor is a web-based tool that allows you to log in to the Aurora Serverless cluster and execute queries.** So I could select the Aurora Lesson Serverless cluster, use the admin user that's already been set up on this database together with the password, and then I could go ahead and connect to the database.

**To use the query editor, you do need to utilize a piece of functionality known as the data API and this is critical because this is one of the main advantages of Aurora Serverless. The data API allows you to access the database using traditional APIs. So rather than having to open a traditional database connection and execute sequel queries, you can connect to it using standard API. So it's much easier if you're designing an application from scratch to code it to utilize Aurora Serverless than to have to worry about traditional database connectivity. But the data API also allows the query edit to work. So you've got to enable the data API on the database and once you've done that, then you can connect to it using this web-based query editor.** So to allow this, I've got to modify the database. So I go to modify, tick the box to enable the data API, hit continue, apply that change immediately, and then modify the cluster. Now that changes really quick, so I can go straight back to the query editor, select the cluster, pick the admin user, and then hit connect and at this point I've connected to the database using this graphical query editor and I could run the query just as I could if I was connected, using a traditional mySQL client, and I'm getting the information from the database using this editor. Critical to understand again, this is an Aurora Serverless only piece of functionality. **The data API only currently works with Aurora Serverless, and so the query editor can also only be used for Aurora Serverless**.