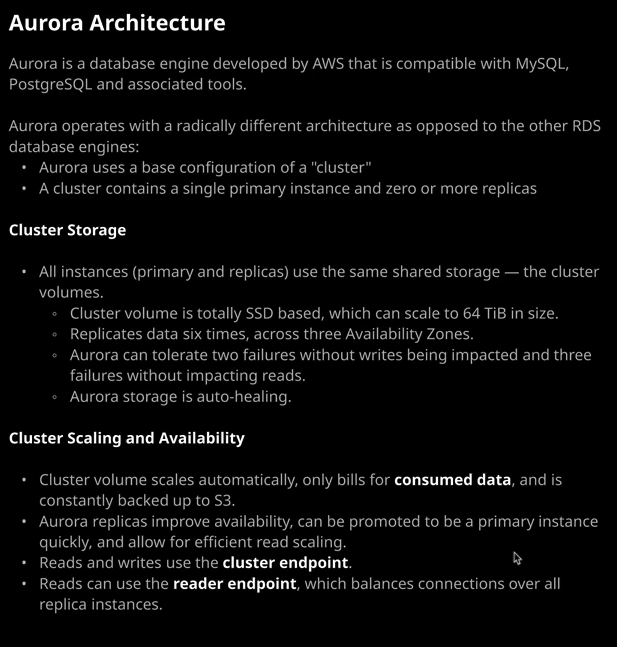
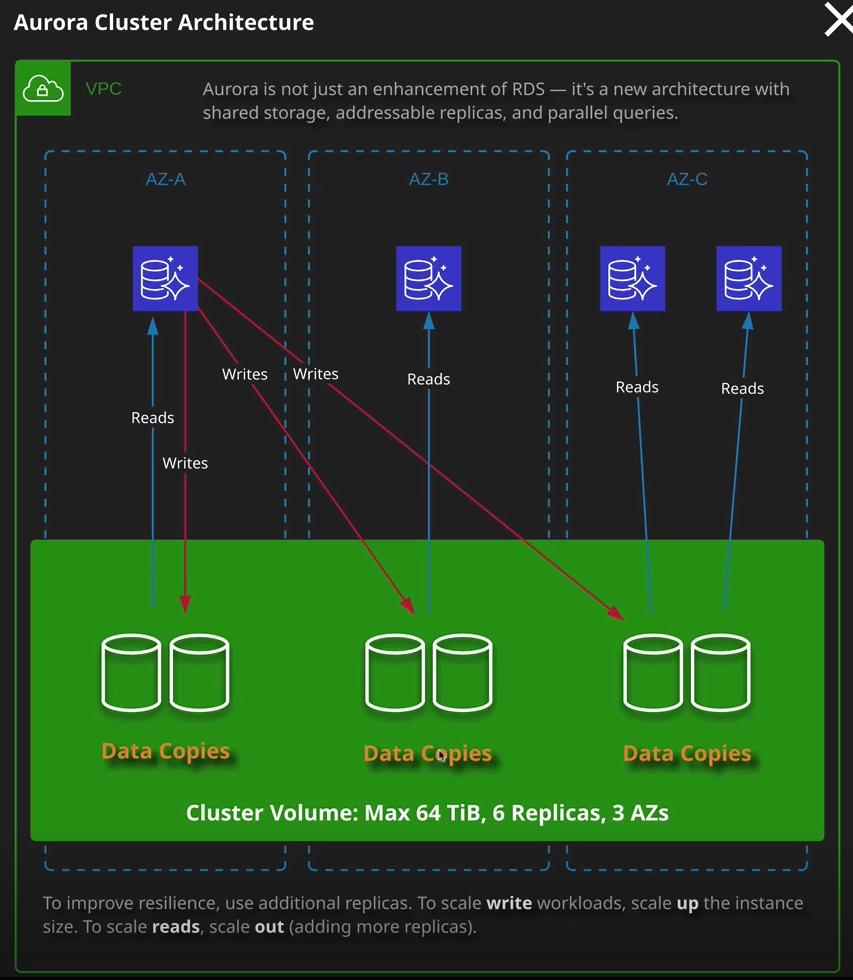
Welcome back, and in this lesson, I want to introduce Amazon Aurora.

**Now Aurora is a custom designed database engine delivered as part of the RDS product.** Most people think that Aurora is a simple evolution of the other database engines available in RDS. But over the next few lessons in this topic, I want to illustrate that **Aurora comes with a large number of significant performance and feature enhancements over the other database engines.** Now for the Solutions Architect Associate exam, you don't need to be aware of how Aurora works in detail, but I do want to give you a demo over the next few lessons just to give you some practical exposure as I talk about the architecture.



So before I do that, I just want to prep a few things to be ready for the demonstration later in this lesson. I'm going to go to the EC2 console as I've have done in previous lessons and create a key pair. I'm going to go to key pairs, create key pair, and then call this one Aurora. I'll hit "Create." It'll download it to my local machine, then I'll move to the CloudFormation console. I'll create a stack and I'll upload the VPC document file that's in this lesson's folder. So I'll click on "Choose File." Then in the dropdown, I'll go to databases 03\_Aurora and then the Aurora folder and in there, vpc.yaml select that, click "Open." Now, as with the RDS lessons, **this is just a simple CloudFormation template that creates a VPC, some public, private, and database subnets, and in addition, it creates a bastion and web server host**. I'll click "Next." I'm going to go ahead and call it Aurora and then just let the key pair that I created a second ago. So I'll click "Next," "Next" again, accept that it will create some IAM resources and then hit "Create Stack."

Now while that's creating, let's talk about the architecture of Aurora, exactly what makes Aurora special versus the other RDS database engines**. So Aurora was developed by AWS and it's designed to be compatible with both mySQL and PostgreSQL. Aurora operates with a radically different architecture, as opposed to the other RDS database engine. So Aurora uses a base configuration known as a cluster, and a cluster contains at least a single primary instance together with zero or more replicas.** Now, in the previous topic of the course, when I covered RDS, I told you how you could enable multi AZ mode on an RDS instance and that created a standby node that was synchronously replicating with the primary or master node but crucially, both of those used their own isolated storage.



Well Aurora does not operate in that way, If I click on this "Architecture" button, this is actually how Aurora operates. **So you define a cluster and inside that cluster you've actually got cluster shared storage or a cluster volume. Now it's got a maximum capacity of 64 TiB, and it has six replicas across three availability zones. So constantly, all this storage is replicating between all of these copies of the data and Aurora's capable of self healing any data problems that exist on the shared storage. Now that's a critical difference between how Aurora works versus the other RDS database engines. So straight away you've got this cluster shared volume. It could be up to 64 TiB in size compared with, for example, 16 TiB for mySQL and another difference with Aurora compared with other database engines is they are actually built for the storage that you use, not for the storage that you're allocated. So with Aurora, you actually don't get to set the storage. You start off with zero allocation and you use whatever you use. It's based on a high watermark system. So if you consume 10 TiB of storage and then delete five TiB of that, you're still using 10 TiB. So you build for whatever the high watermark point is the only way to reduce that high watermark. It's to take it back up and make a new cluster with just that data but it does offer much more efficient billing than RDS.**

So straight away, the storage component of Aurora is radically different. Now, **inside the cluster, as I mentioned, you've also got the concept of these different node types. Now one node is what is always the primary node and that performs the writes to all of these different data copies. So the writes come from a single node. In addition to that, though, you can have replicas and replicas are capable of operating reads from the shared data copies,** and I'll talk more about that in a moment. The important thing to understand, though, **is you can have lots of these replicas inside an Aurora cluster**.

Well, now that this is finished creating, I can go ahead and create my Aurora deployment as part of this lesson's demonstration and I can highlight each of the key concepts as I go through that process. I'm going to move across to the RDS console and just like I did with the demonstrations in the previous topic of the course, I'll need to define a subnet group. **Aurora still uses subnet groups** and if you recall from the previous topic, **that's essentially a group which specifies which subnets to deploy a database instance into**. So I'm going to go ahead and create a subnet group and I'm going to call it Aurora. I use the same for the description. I'll make sure that I've got VPC 1 selected, which is the VPC that was created by the CloudFormation template. I'm going to pick availability zone 1A and I'm going to pick the database subnet, which is 10.0.21.0. So I'm going to go ahead and add that then I'll select US East 1b, and I'm going to pick the database subnet in this one, which is 10.0.22.0. So I'll go ahead and add that and then hit "Create." So that's a subnet group. Next, I'll go to dashboard, create database, and this time I'll leave it selected as the Amazon Aurora database engine. **Now Aurora is compatible with one of two existing database engines either mySQL or PostgreSQL, and you'd select between these if you've got existing tooling or existing data that you want to import into this new Aurora cluster**. Most of the time when I'm deploying this into production I generally select **Amazon Aurora with mySQL compatibility because I find most tooling is more compatible with this option but if you do have any specific requirements for Postgres then you can go ahead and select that**. As with the other RDS database engines, you're also able to select the **database version.** **This is actually really important for Aurora because not all features and functionality are supported in all of these different database engines**. So I normally leave this at whatever the default is, which is currently 5.6.10a. Notice how we got all of these options around regional and global database location. If I was to go ahead and change this maybe to Aurora 5.7 a lot of this would disappear. **So you need to be sure that you're picking the most appropriate database engine version.** So if you're in doubt, just make sure you go ahead and select the default. I'm going to be talking about a lot of the advanced functionality in the next lesson. In this lesson, I want to focus on the core architecture of Aurora. So I'm going to leave this selected as regional for now, and I'll be covering exactly what global is in the next lesson. I'll also be covering parallel query in the next lesson and then Aurora serverless in the lesson following. So I won't spend any time discussing those now.

**Just like with RDS, you're able to select between two different template types production in Dev test, and these just preselect some of these following configuration options**. For now, I'm going to hit Dev test, and I'll talk about exactly what that changes once I get this cluster provisioning. Okay, so the first **difference between Aurora and the previous RDS engine types is that you do need to identify the cluster. So the cluster is a configuration entity that the other RDS engines didn't have. So I need to specify an identifier for the Aurora cluster.** So I'm going to go ahead and do that now, and I'm going to call it Aurora Lesson Cluster. I'll need to specify admin credentials and just like I did with the RDS lessons in the previous topic, I'm going to allow **RSS to automatically generate this**. So I'm going to tick this box. As with the previous engines, you're also able to select the different types of instances that you want to use for this Aurora cluster. **The selection that you have access to is slightly different than the other RDS engines but you do get to pick between the memory optimized classes and the burstable classes**. For this demonstration, I'm going to select the burstable classes and select db.t3.medium. I'll be talking about exactly how the replica architecture is different in Aurora. For now, I won't be creating a replica, so I'm just going to carry on scrolling down.

Now**, Aurora still is a VPC based product**. So you do need to select a VPC going to go ahead and expand the additional connectivity configuration and note how you do have to select a subnet group just like with the previous database engines. You do have the ability to select between whether you want it to be publicly accessible or not. I'm going to select "not" at this stage. **Aurora clusters are still secured from a network perspective using a security group**. So I'm going to go ahead and create a new security group and I'm going to call it Aurora\_SG. I'm going to leave both the availability zone on the database port at the defaults. I have explained those in previous lessons, and I don't need to make any changes. And then I'm going to expand the additional configuration at the bottom, and I'll need to specify a DB instance identifier together with initial database name. So in this first box, I'm going to call it Aurora Lesson. I'll keep it consistent with the previous lessons in the last topic and then for the initial database name, I'm going to call it Aurora Lesson DB and then scrolling down, you've got some familiar options. So a lot of this is shared with the other database engines that RDS supports but we do have some additional ones. So we've got **fail over priority**, which we'll talk about later in this lesson and we've also got **backtrack**. So backtrack is a really useful feature that I want to cover towards the end of this lesson, which adds ***additional backup and restore capability***. So I'm going to tick to enable backtrack. This is something that you need to enable when you create an Aurora cluster. So I need to enable this now, even if I'm not going to use the functionality later on. So it's always best practice to enable this. So I'm going to tick that box. You do get the option to pick a backtrack window and for now, I'm going to put the maximum, which is 72. Again, I'll talk about this more when I cover backtrack later in this lesson. Now, at this point, I believe everything else is default and I'm going to click on "Create Database" because I've decided to autogenerate the password. It is going to pop up a view credentials detail box, and I do need to make sure that I note this down because it's the only time that I'll be able to get access to these credentials. I can change them later, but it is always best practice, obviously, to click on "View Credentials" and note them down at this point. No, as with RDS this is going to take some time to provision. So while it's doing that I want to talk about the other key architectural elements of an Aurora cluster. So we've already covered the cluster itself together with shared storage. That's one of the key benefits between Aurora on the other engine types available in RDS. I've covered the fact that, just like with RDS you get to pick the sizes for these different instances inside the Aurora cluster. **That size selection is slightly different than the previous RDS engines. Now just as with RDS, you are able to purchase instance reservations for an Aurora provisioned cluster. So if you do have some consistent database requirements, you can go ahead and purchase some instance reservations to reduce your ongoing costs. You got the same option as you do with EC2 on the other engines available inside RDS.** So keep that in mind.

**Now one of the most important ways that Aurora differs from the other RDS engine types is that it has various different types of instances and you're able to configure those with much more granularity than the other RDS engines. So every cluster begins with one instance and that's known as the writer instance. That's the primary instance or primary node of an Aurora cluster. Now, this is the only instance that's allowed to perform writes to the cluster shared storage.** So **much like with RDS with its primary instance, this is the instance that you would use to do any writes to the cluster.**

Now, if we open up the cluster, just as with the other engine types inside RDS, we've got these endpoints. **So every cluster starts off with a writer endpoint and a reader endpoint. And to start with, if you only have one instance, which is this primary instance, then both of these point at the primary instance. Where Aurora differs from RDS though, and this is where it starts to add significantly more capability and features is that it's really easy to add an additional instance to an Aurora cluster**. So we can just go to actions and then add reader and a reader is a specific type of instance that operates inside an Aurora cluster, which is only available for reads. So in this particular case, I'm adding a reader. I select no preference. I don't want to pick the availability zone that it goes into. Just like when I defined the cluster and defined the primary instance, **I've got the ability to specify whether this is publicly accessible or not.** I can pick the instance specifications to use. So I'm going to keep this the same as the primary instance. I specify a replica source, which is going to be the primary instance inside the cluster and I need to give this a DB instance identifier. So I'm going to go ahead and do this. I'm going to give it an identifier. I'm going to call it Aurora lesson two.

Now **an Aurora class that can have up to 15 replicas, and these could be placed in as many or as few availability zones as you need. These replicas used the same cluster shared storage as the primary instance and all of this storage is replicating synchronously, so, by definition, these replicas, because they use the same storage, they also replicate synchronously. Unlike previous engines in RDS though, you can actually target these replicas for reads. So they operate somewhere between the standby instance and read replica functionality levels of mySQL in RDS. You actually get the best of both worlds. You get to define as many as you need up to 15. They can operate as fail over targets in case you have any issues with the primary instance, and you also get to dictate exactly where you want them to go, and they help you scale the database cluster from a read perspective. So the more of these replicas you add, the more read performance that you can get.** Now I'm going to go ahead and at this replica, but before I do, I want to talk about **the fail over priority.** So when you do define these replicas, when you add them, you can actually specify the **fail over tier that it uses anywhere from tier zero to tier. 15. Now tier zero has the highest priority and tier 15 has the lowest. When Aurora attempts to perform a fail over, it's going to look for any replicas that are specified as tier zero first and then it will go to tier one and then it will go through the entire list all the way up to tier 15 which is the lowest priority. So this just gives you a hierarchical fail over order that you can use to control how fail over actually works. Now you can either specify a tier explicitly or you can specify no preference** and that's what I'm going to do. Under most circumstances, you don't need to worry about specifying this fail over preference. So I'm going to leave it as no preference. I'm going to scroll down, leave everything else is default, and click on "Add Reader."