Now **when adding a reader is much quicker than if you were to convert a mySQL based RDS deployment from no multi AZ to multi AZ and it's also much quicker than adding a read replica**. **The reason for this is it's using the same, shared cluster storage and so the only thing that Aurora needs to do is provision this new instance and point it at a storage. It doesn't need to provide this new replica with its own isolated storage. So that's another significant enhancement over mySQL running inside RDS.**

**Now, when it comes to the scaling architecture of an Aurora cluster, you've got two different options. If you want to scale your ability to handle increasing write workloads, then you need to increase the size of this primary or writer instance, and that's known as scaling up. But if you want to improve the ability of your cluster to handle read-heavy workloads, then we could scale the cluster out, and that involves adding additional replicas.**

Now the reason why Aurora is much more capable of handling read heavy workload scaling is if I go into this database cluster, so inside the Aurora cluster, now that I've added this additional reader replica then this reader endpoint now points at both the writer and this additional reader. So if you recall in the previous topic when I talked about multi AZ mode, I told you how multi AZ didn't help with read scaling because you couldn't directly address the standby instance. I also told you how if you added additional mySQL read replicas and there was no way of addressing those as a unit. **Well with Aurora you can add additional reader replicas and every time you do that, this reader endpoint is expanded to include those additional replicas. So if we use this the DNS name and we use this for reads, then we get immediately the benefit of these additional reader endpoints, and we can define these in whatever availabilities zones the subnet group that we used to create the cluster allows us to.**

So an **Aurora cluster has a number of different endpoint types. We've got the writer endpoint or also known as a cluster endpoint, which you can use for reads or writes. We've got the reader endpoint, which always addresses any of the database replicas that can tolerate reads and then, in addition to that, each of the database instances has its own endpoint.** And then, finally, **you can create custom endpoints and explicitly allocate individual database instances. So you've got a lot of flexibility and configurability when it comes to how to manage read scaling inside an Aurora cluster.**

Now that its cluster is open, running on to demonstrate how we can utilize it. And that process is exactly the same as using mySQL RDS that I demonstrated in the previous topic of the course. If I go to services and then open up the EC2 console, the CloudFormation template has also preconfigured an instance so that it's already got WordPress installed and the Apache web server installed, and it's ready to go. So I'm going to go ahead and connect to this. So I'll right click and hit "Connect" now because I'm running a Linux compatible operating system, so I'm running Mac OS, I'll need to adjust the configuration on the pem key, so that's what I'll do. First, I'm going to move across to my terminal, going to go into my downloads folder, which is where I've downloaded the pem key and then adjust the key's permissions. Now, if you're running this on Windows, I've included a link in the lesson description, which gives you a step by step guide. But you won't need to follow this process to address these permissions, but you will need to perform the steps slightly differently. Once I've done that, I'm going to go ahead and SSH directly into this instance. So I'll copy the command, move back to my terminal, and then connect to the instance. It's the first time I've done it's only to authenticate. I'll type "yes" and then I'm connected to the instance. Now I'll clear the screen to make it easier to see and just to verify this instance is correctly set up, I'll go /var/www/html. I'll do a file listing and assuming everything's configured correctly, I should have the WordPress files preinstalled and I do, so that's great. What I'm going to go ahead and do is as before, I'll run a sudo copy, and then I want to copy the template configuration file, which is wp-config-sample.php and I'm going to rename that to wp-config.php then next, I'll go ahead and edit that configuration file. So wp-config.php and I forgot to type the nano command for the editor so that command obviously should be sudo nano wp-config.php. So what I need to do is go ahead and update these configuration variables with the details for our Aurora cluster. So first I'll adjust the database name. So that's auroralessondb than the mySQL database users to use, which is admin, the auto generated password, which I noted down earlier. So I'll go ahead and get that so I'll paste that in and then I'll need the database hostname to use and that's going to be the cluster endpoint address. So the one that's capable of reads and writes. So I'll go back to the RDS console and I'll get the writer endpoints. I'll copy that into my clipboard, move back to the terminal, and paste that in. Once I've done that, I'll save that file out and then exit. Now that's done, I've got one more step that I need to do. Remember, this Aurora cluster is secured using a security group, so I need to make sure that my EC2 instance has access to that Aurora security group. So to do that, I'm going to move to the EC2 console, go to security groups, note here that I've got the Aurora security group that I configured when I made the cluster, I'll click on the inbound tab and then edit, going to add a rule, and duplicate one that's already here. But instead of adding an IP address allocation, we're going to type SG for security group and select the Aurora bastion and security group and that's the one that's associated with this EC2 instance. So I'll save that, go to the dashboard, go to running instances, select this instance, copy that DNS name into my clipboard and open it in a new tab. Once I've done that, I'll be greeted with WordPress installation screen. So I'll just follow this process through the blog named Cat Pictures, the username, I'll pick a suitable password to use. It is a simple one just to make this demonstration easier. So I'll confirm the use of a weak password link to my email address and I'll click "Install WordPress." That process is fairly quick. I'll click on "Login" just to verify this is working correctly enter my password and click on login. And that looks good, we've got a fully operational WordPress site using Aurora. Now, what I'm going to do next just to demonstrate some of the advanced functionality is I'm going to go ahead and add a post to this website. So I'm going to add a new post and say, "Hello World. Here is a cat." and then below that I'm going to add an image, click on "Upload" have included a sample image in this lesson's folder. So I'll just go ahead and select that so nl.jpg click on "Open" just a picture of a Dutch cat. One I have done that, I'll go ahead and click on "Publish." Once that's published, I can click on cat pictures and go to visit site just to verify that it's visible on the front page and it is. So we've got that one cat post. Now, the reason why I'm adding this post is to illustrate one of the really cool features of Aurora, which is known as backtrack. Backtrack is a feature that allows you to roll the database back into a previous state. So I've got this database in a position where I've added this Dutch cat picture and what I want to do now, is leave this for a number of minutes and then add another post and we're going to use this scenario that we've posted this cat picture. Somebody else has come along and then added a blog article that we don't want and we can't get access to the WordPress instance to delete this post. It could just as easily be a database corruption or some other feature or piece of data that we'll want to roll back from. So let's go ahead and do that. So I'm going to go to cat pictures again and then dashboard them, going to add another post. So I'll click on, "Add New." This time it's going to be dog post and I'm going to say dogs are better than cats. Which is just clearly not true, dogs are great, but cats are, of course, the best. So I'm going to enter that and then hit "Publish." So now that's published, let's go have a look at how the front page of the blog looks. We've got a cat post and our dog post. Now with other database engines available in RDS, the only option you would have to do any sort of recovery from any sort of data corruption will be to restore from backup. Aurora is capable of doing the same type of back up and restores the other database engines are capable of. So I could pick a specific database instance, go to actions, and then take a snapshot. So I can create manual snapshots of Aurora instances. So I can go ahead and do that. So Aurora snapshot, click on "Take Snapshots," and that will create a manual snapshot in just the same way that RDS does. Aurora also performs the same type of automatic backup that the other RDS engines do. So that's configurable in the same way. So if I click on the cluster and then go to modify, we've got the same backup retention window so we can set this anywhere from one day to 35 days, and this is an automatic backup process. This has the same lifetime as the database instance. We've got a maximum of 35 days of retention and we can restore at any point during that 35 days to a brand new database instance. Remember from previous topics of the course, you can only restore automatic backups and snapshots to a brand new database instance, and I taught you how that requires application reconfiguration. But one of the options that we've got with Aurora is to perform a backtrack. Let's have a look at how that works. Let's say that I wanted to roll this database back. I'm going to go to dashboard and then database instances, select the cluster, click on "Actions," and then go to backtrack. Now, **when you create the cluster, you set the window time that you want backtrack to work to and the maximum is 72 hours.** Now it is currently August the fifth and the current time is 9:16 and I made that blog post about dogs a number of minutes ago. So why don't I try to roll back this database to a previous point in time. So I'm going to select nine and then why don't I roll it back to 10 past nine? So 9:10 and then I'll set it to zero seconds. Now what this is going to attempt to do is to roll the database back to this exact point in time and if it can't, is going to pick the closest available time. So I'm going to go ahead and try this. So 9:10 is selected and I click on backtrack DB Cluster. **Now, when you do backtrack an Aurora cluster, it does cause an outage. The entire cluster is taken off line because what it's actually doing is rolling back this shared cluster storage, and that's really important to know. So whenever you perform a backtrack, you do get the flexibility that you don't have to create a new database cluster, and you don't have to update application configuration but you do get an outage because the entire database storage is being rolled back**. If I go back to this cat pictures blog and attempt to do a refresh, you'll see that the dog post is gone because Aurora has rolled back to its previous state 10 past nine before I posted the dog post. So that's a really amazing piece of functionality that can come in really handy where you backtrack a database to a previous point in time before a corruption or before elements of that database were deleted. So for the exam and for real world production usage, **it's a really important feature to understand. So it's got a maximum window of 72 hours, and you control the entire cluster back to a previous point in time without having to create a new cluster.**

Now, obviously, I mentioned earlier in this lesson how, because **we've created additional endpoints inside this cluster, as well as allowing us to scale read operations. It also does provide a level of high availability, so these replicas are available for both reads, but they can also be used as fail over.** So if I select Aurora lesson two and then go to actions and then failover I can manually initiate a failover process, which is going to change the writer role from being this original instance that created with the cluster to being Aurora lesson two which is the one I added after the fact. **So you're able to initiate a failover operation manually. You can even run some sequel queries inside Aurora to simulate this process, or the process will be performed automatically whenever Aurora detects any issues with this writer instance, this writer node. If you've got any problems at all with the instance that's allocated the writer role then Aurora will automatically roll over to one of the other replicas inside the cluster and that protest also occurs much faster than with other engine types inside RDS because of the shared cluster storage architecture.** So if I just hit refresh already, we can see that the Aurora lesson two instance is now the writer. **So that failover process is also much faster than the other engine types available in RDS.** Now this is everything that I wanted to cover in this core architectural lesson for Aurora. There's going to be a lesson immediately following this one where I'm going to talk about some of the more advanced architectural concepts but I do want to make sure that I have some separation. This is a lot to understand for one product, but it is really important for the exam because Aurora questions do feature quite heavily in some of the questions, so you need to understand exactly how it works. So key takeaways is to understand that **Aurora has this notion of a cluster and a cluster contains shared storage. The instances themselves do not have their own allocated storage, and that comes with massive performance benefits and massive resiliency benefits. The storage is replicated across three or more availability zones inside a region, and you have at least six copies of that data, and it is self healing. Aurora can tolerate two failures without writes being impacted on three failures without impacting reads. So it really is a highly resilient.** So with that being said, that is everything that I wanted to cover in this lesson. Go ahead, mark this lesson as complete, and when you're ready, join me in the next.