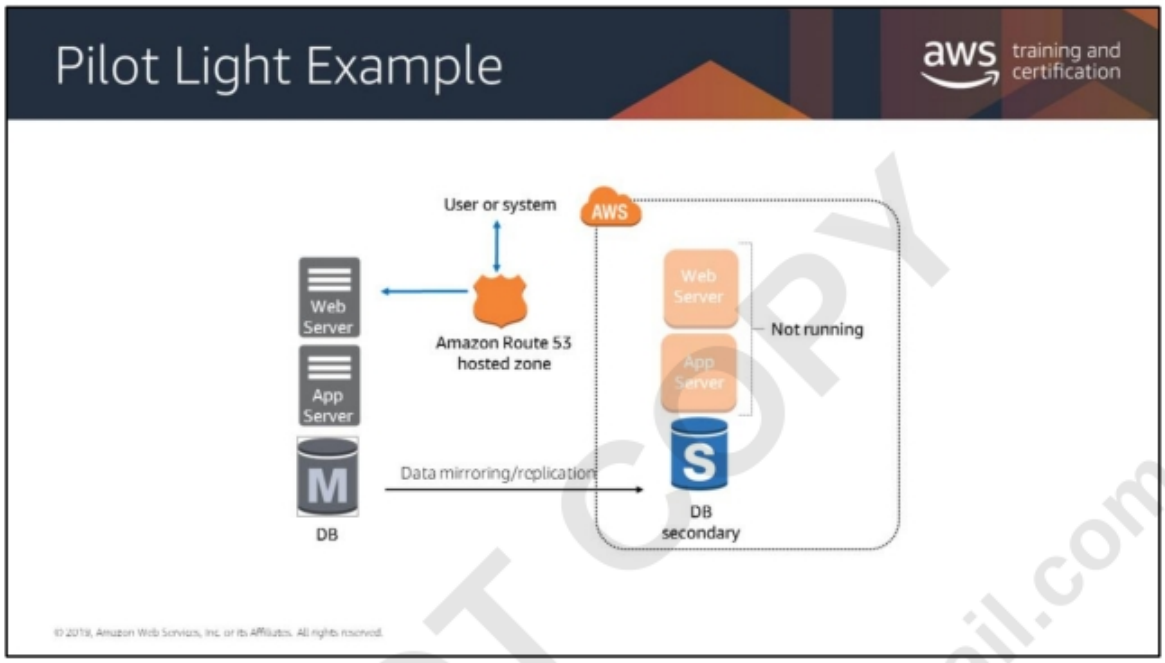
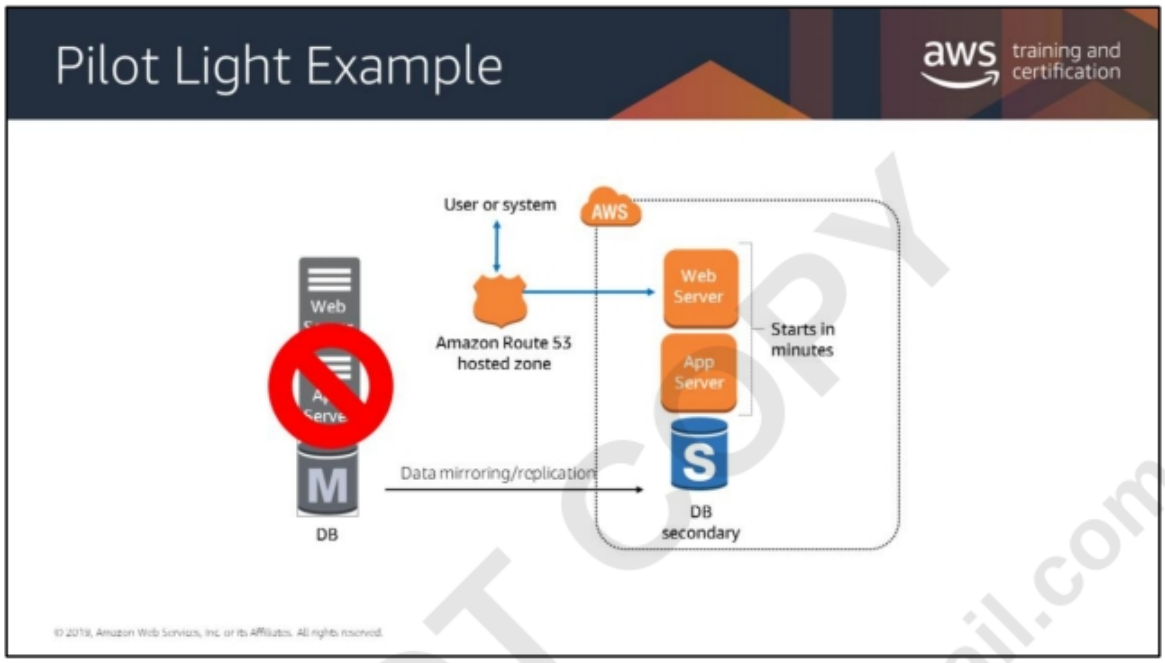


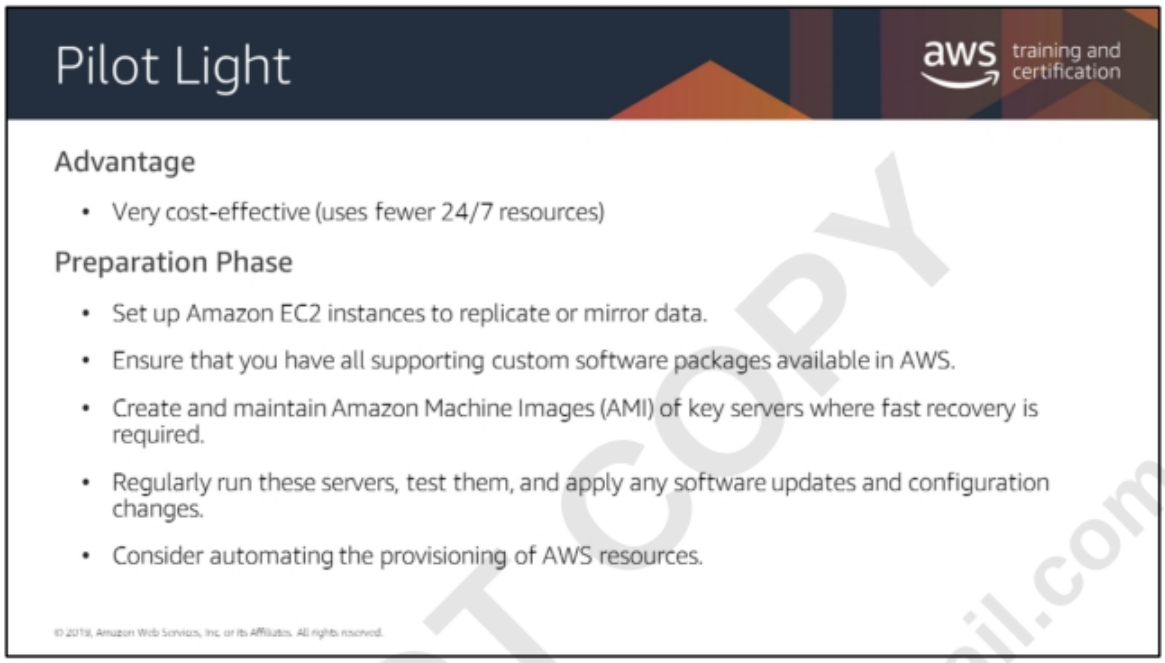
This pattern is relatively inexpensive to implement. In the preparation phase of DR, it is important to consider the use of services and features that support data migration and durable storage, because they enable you to restore backed-up, critical data to AWS when disaster strikes. For some of the scenarios that involve either a scaled-down or a fully scaled deployment of your system in AWS, compute resources will be required as well.

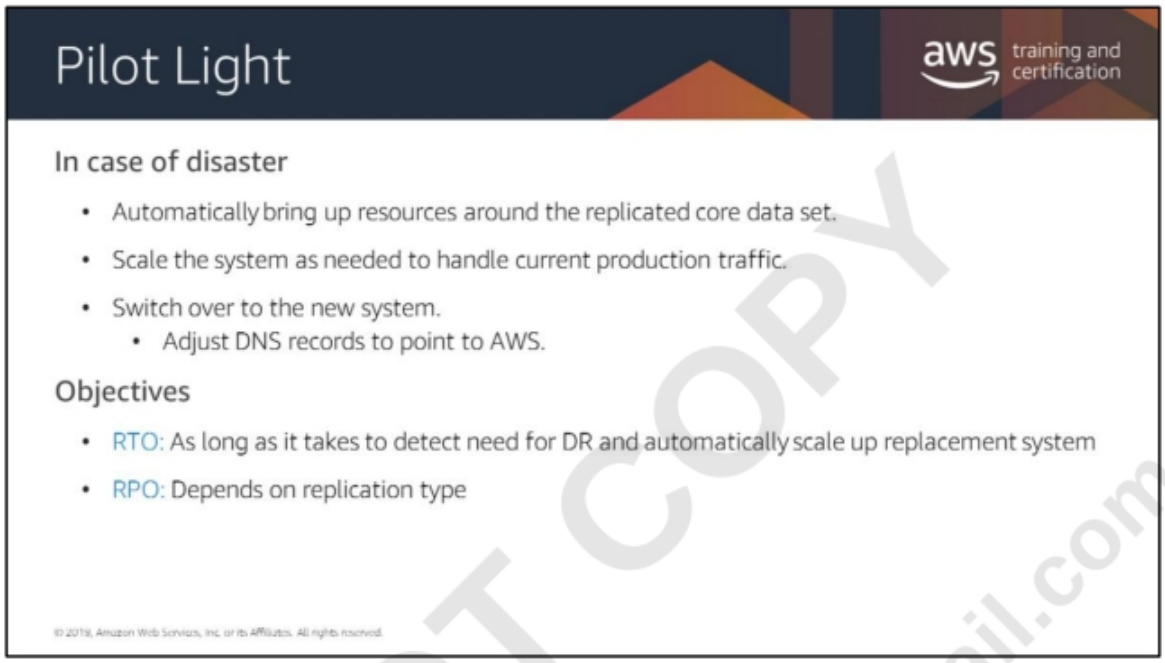
When reacting to a disaster, it is essential to either quickly commission compute resources to run your system in AWS or to orchestrate the failover to already running resources in AWS. The essential infrastructure pieces include DNS, networking features, and various Amazon EC2 features.

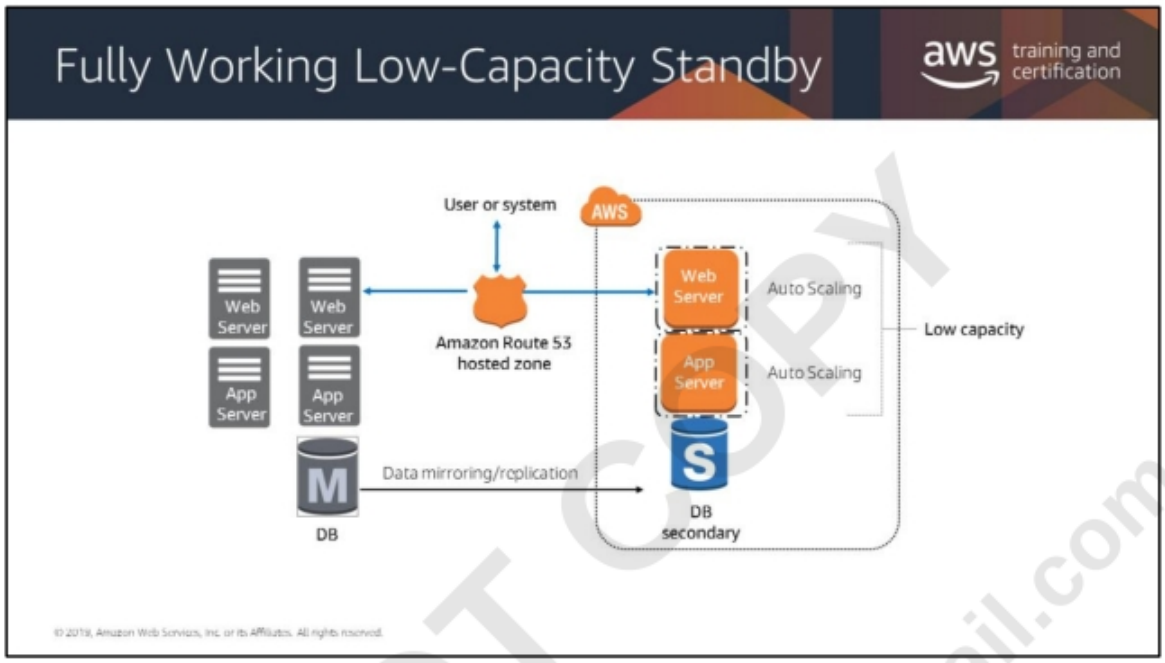
In the preparation phase, in which you need to have your regularly changing data replicated to the pilot light, the small core around which the full environment will be started in the recovery phase. Your less frequently updated data, such as operating systems and applications, can be periodically updated and stored as AMIs.

/







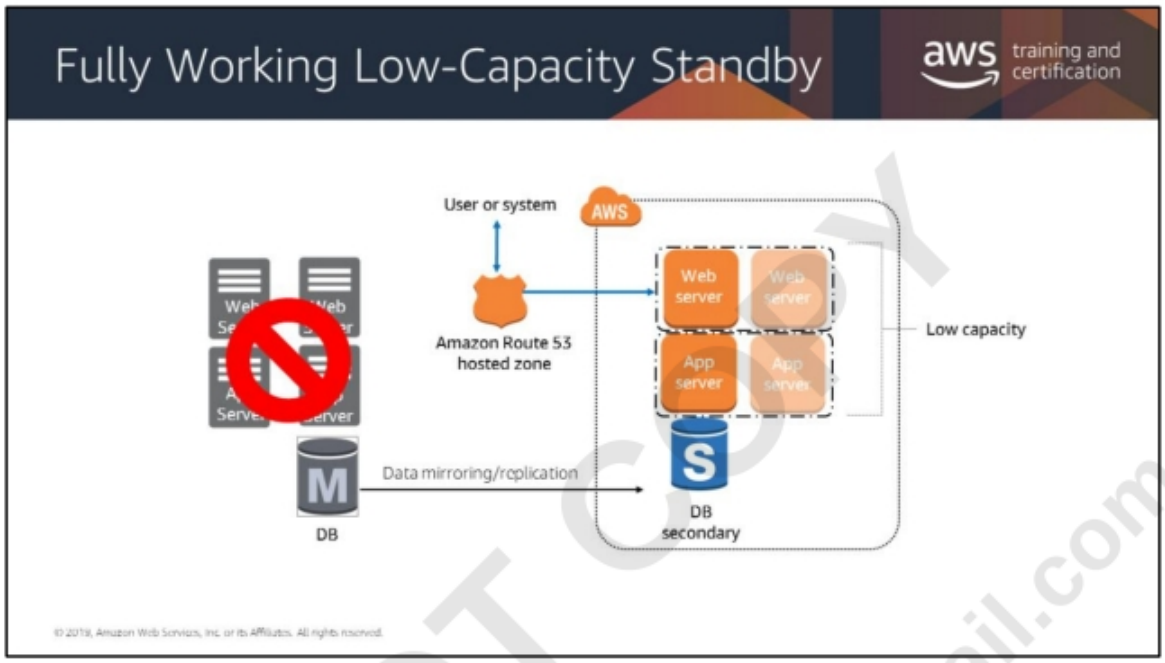


Low capacity standby is like the next level of Pilot Light. The term *warm standby* is used to describe a DR scenario in which a scaled-down version of a fully functional environment is always running in the cloud. A warm standby solution extends the pilot light elements and preparation. It further decreases the recovery time because some services are always running. By identifying your business-critical systems, you can fully duplicate these systems on AWS and have them always on.

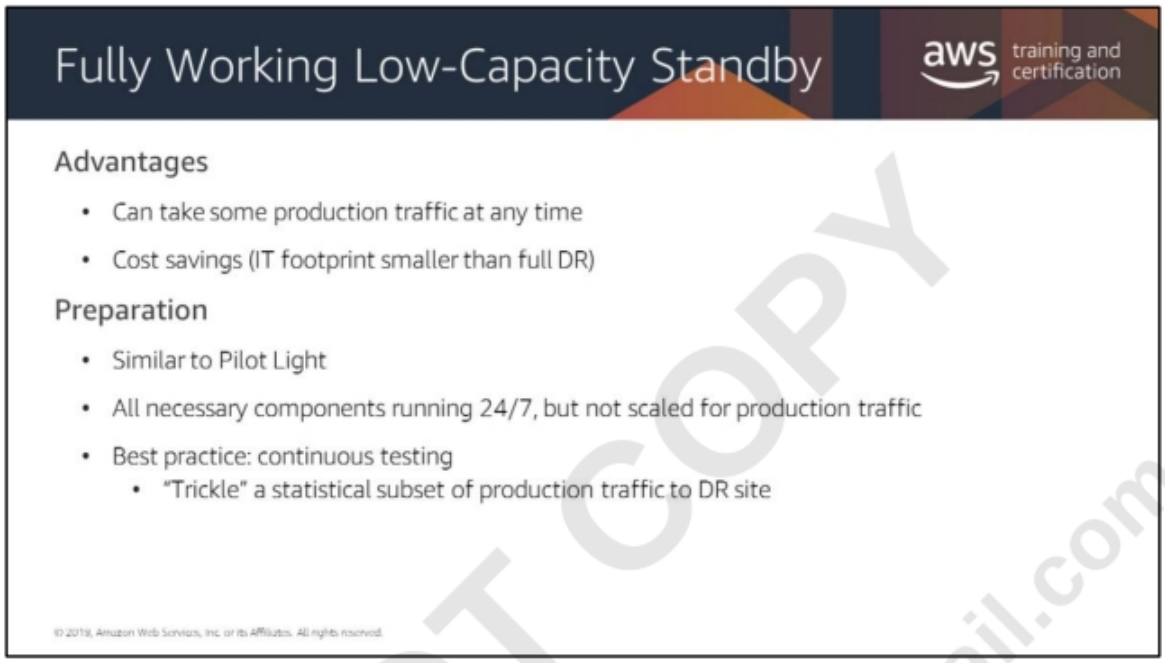
These servers can be running on a minimum-sized fleet of Amazon EC2 instances on the smallest sizes possible. This solution is not scaled to take a full production load, but it is fully functional. It can be used for non-production work, such as testing, quality assurance, and internal use.

In a disaster, the system is scaled up quickly to handle the production load. In AWS, this can be done by adding more instances to the load balancer and by resizing the small capacity servers to run on larger Amazon EC2 instance types. As stated in the preceding section, horizontal scaling is preferred over vertical scaling.

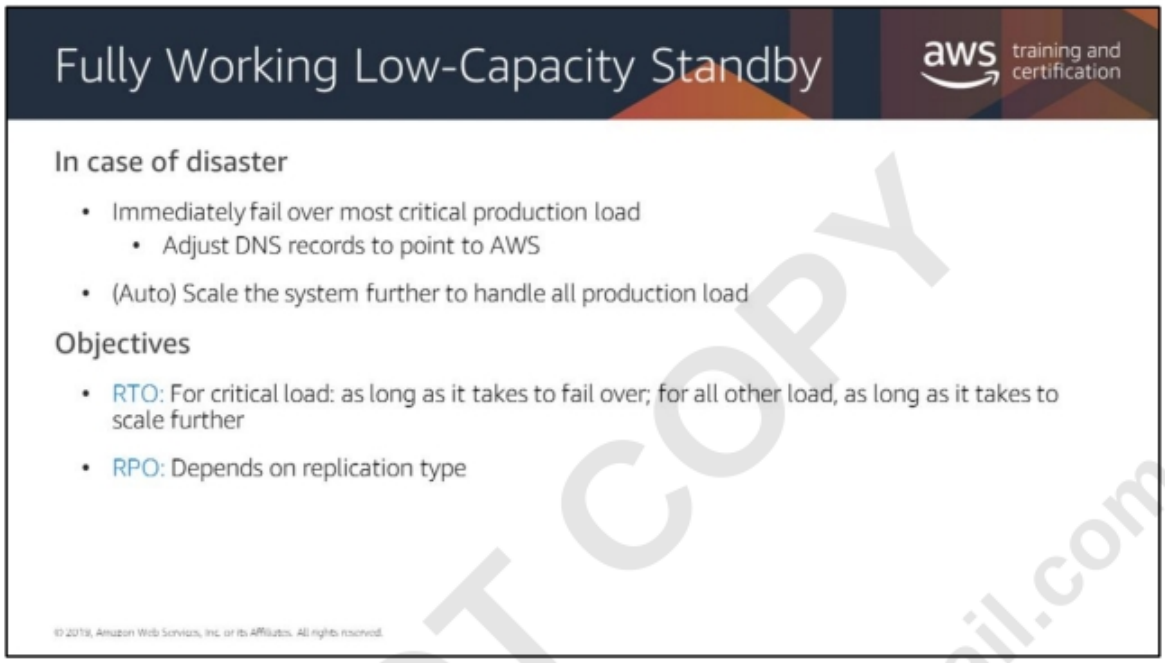
In the diagram above there are two systems running: the main system and a low-capacity system running on AWS. Use Amazon Route 53 to distribute requests between the main system and the cloud system.

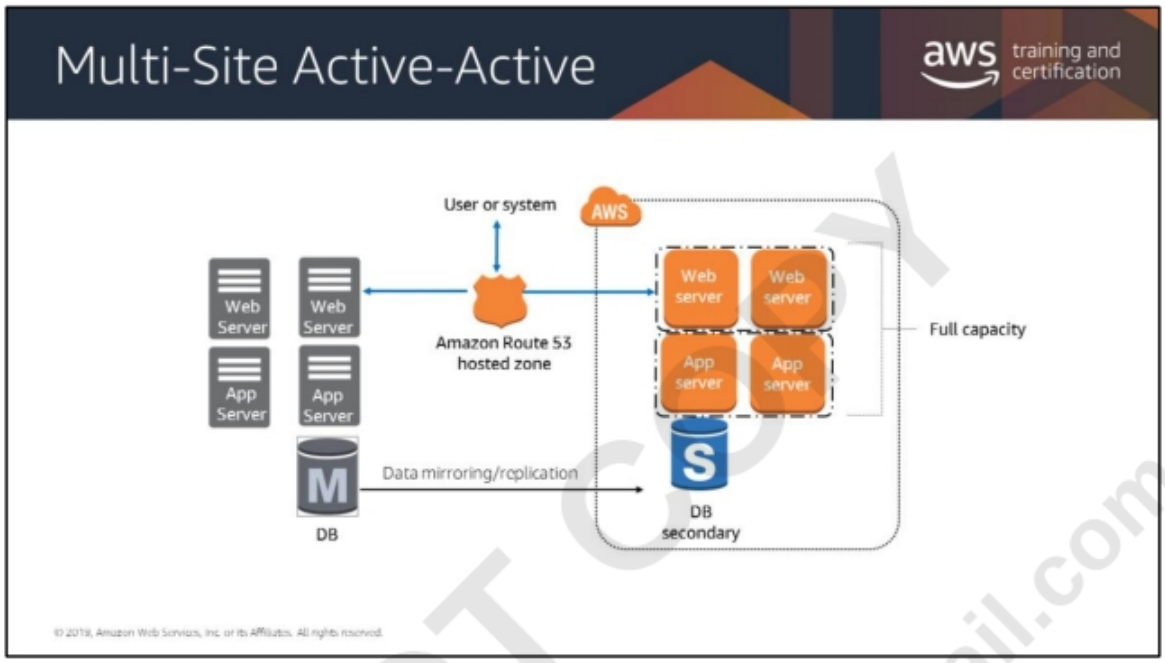


If the primary environment is unavailable, Amazon Route 53 switches over to the secondary system, which is designed to automatically scale its capacity up in the event of a failover from the primary system.



This pattern is more expensive because active systems are running.





The next level of disaster recovery is to have a fully functional system running in AWS at the same time as the on-premises systems.

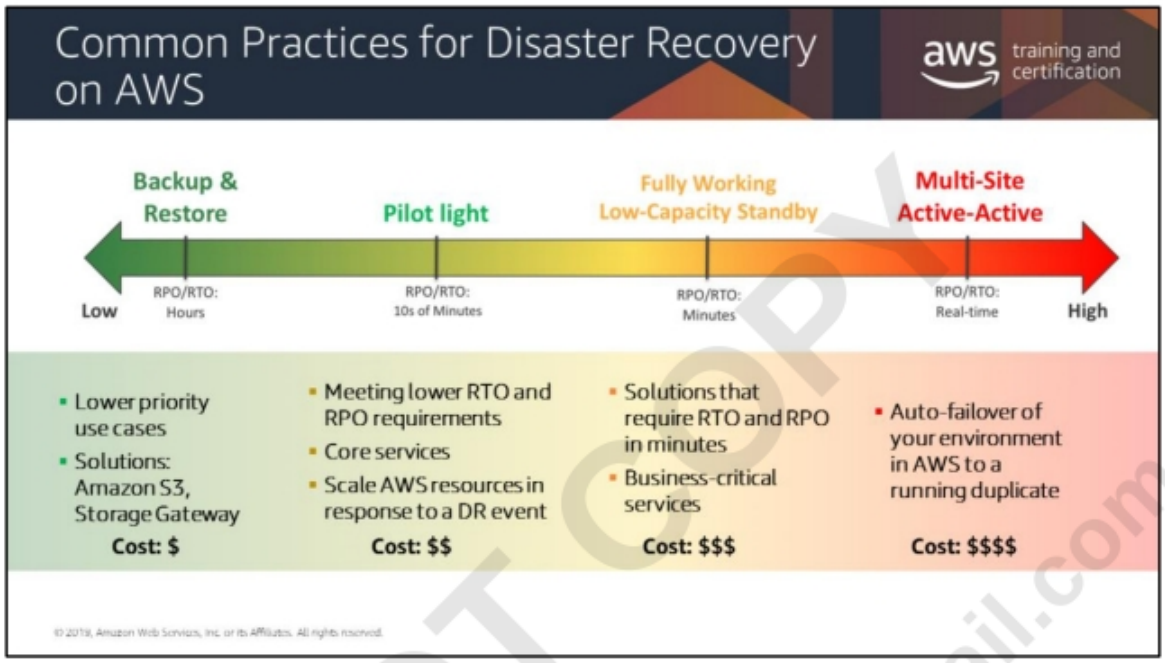
A multi-site solution runs in AWS as well as on your existing on-site infrastructure, in an active-active configuration. The data replication method that you employ will be determined by the recovery point that you choose.

You can use a DNS service that supports weighted routing, such as Amazon Route 53, to route production traffic to different sites that deliver the same application or service. A proportion of traffic will go to your infrastructure in AWS, and the remainder will go to your on-site infrastructure.

In an on-site disaster situation, you can adjust the DNS weighting and send all traffic to the AWS servers. The capacity of the AWS service can be rapidly increased to handle the full production load. You can use Amazon EC2 Auto Scaling to automate this process. You might need some application logic to detect the failure of the primary database services and cut over to the parallel database services running in AWS.

The cost of this scenario is determined by how much production traffic is handled by AWS during normal operation. In the recovery phase, you pay only for what you use for the duration that the DR environment is required at full scale. You can further reduce cost by purchasing Amazon EC2 Reserved Instances for your “always on” AWS servers.

This pattern potentially has the least downtime of all. It does have more costs associated with it, because more systems are running.



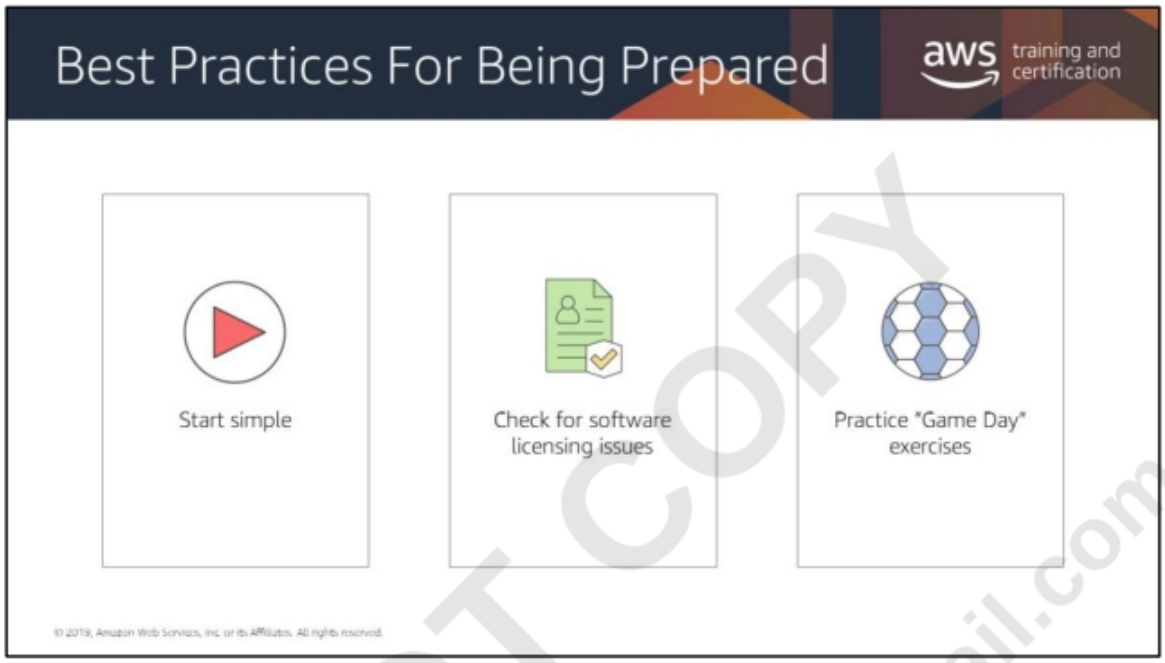
Applications can be placed on a spectrum of complexity. Business continuity ensures that critical business functions continue to operate or recover quickly despite serious disasters.

The next slides outline four DR scenarios that highlight the use of AWS and compare AWS with traditional DR methods (sorted from highest to lowest RTO/RPO), as follows:

- Backup and Restore
- Pilot Light
- Fully Working Low-Capacity Standby
- Multi-Site Active-Active

The figure above shows a spectrum for the four scenarios, arranged by how quickly a system can be available to users after a DR event.

AWS enables you to cost-effectively operate each of these DR strategies. It's important to note that these are just examples of possible approaches, and variations and combinations of these are possible. If your application is already running on AWS, then multiple regions can be employed and the same DR strategies will still apply.



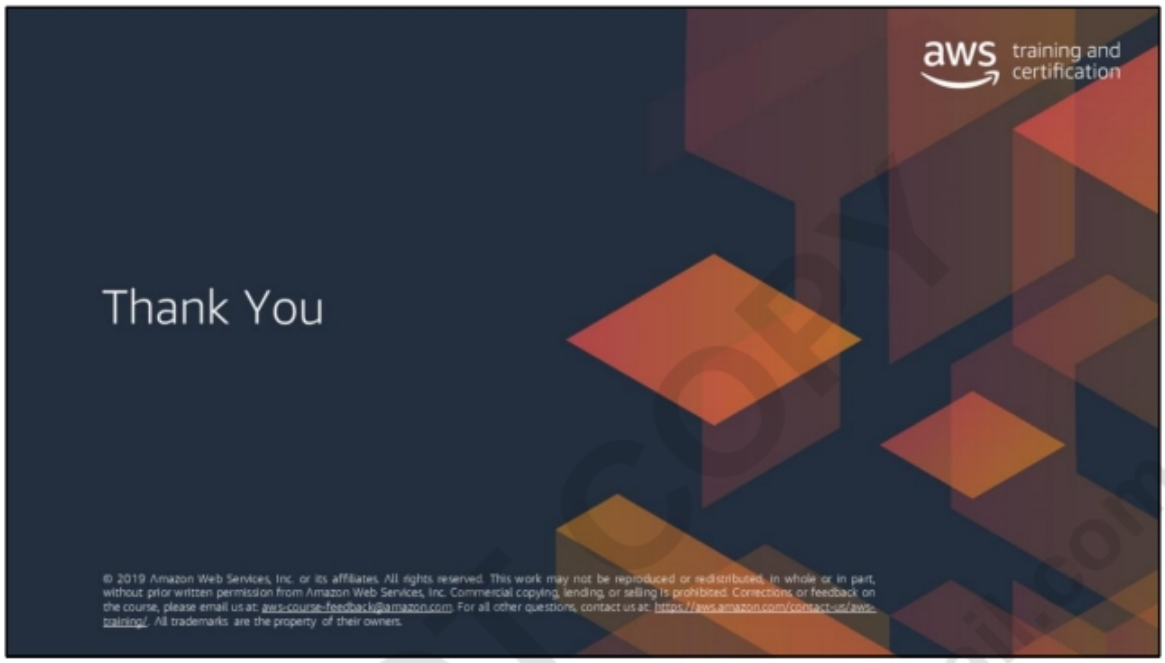
Start simple and work your way up.

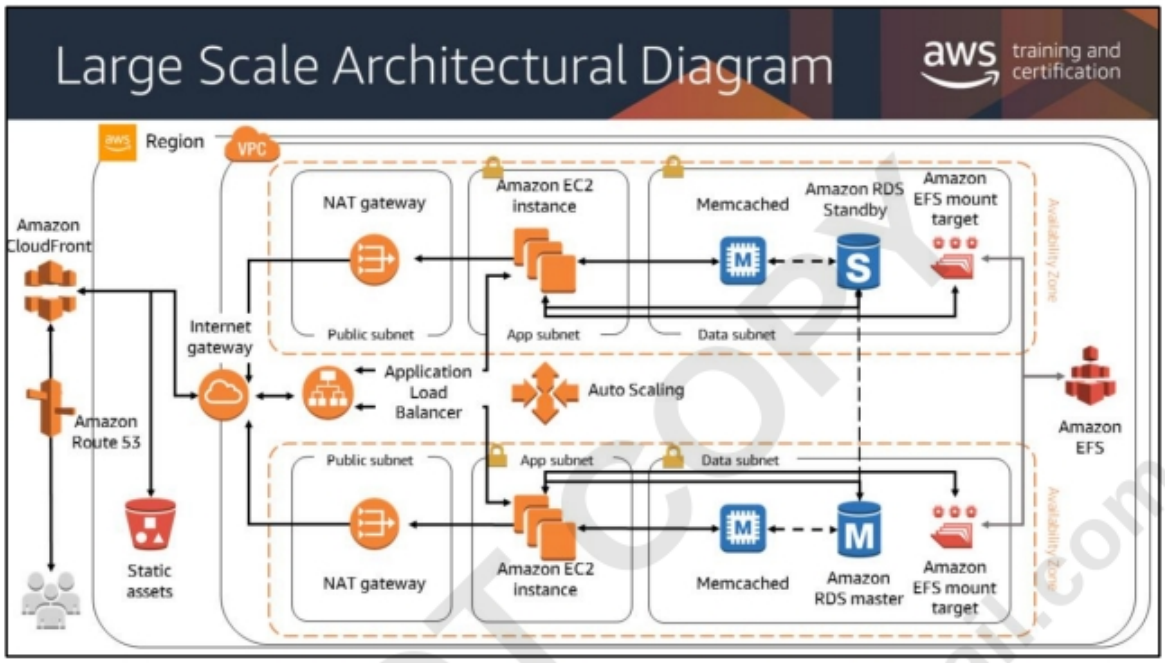
- Backups in AWS are a first step.
- Incrementally improve RTO/RPO as a continuous effort.

Check for any software licensing issues.

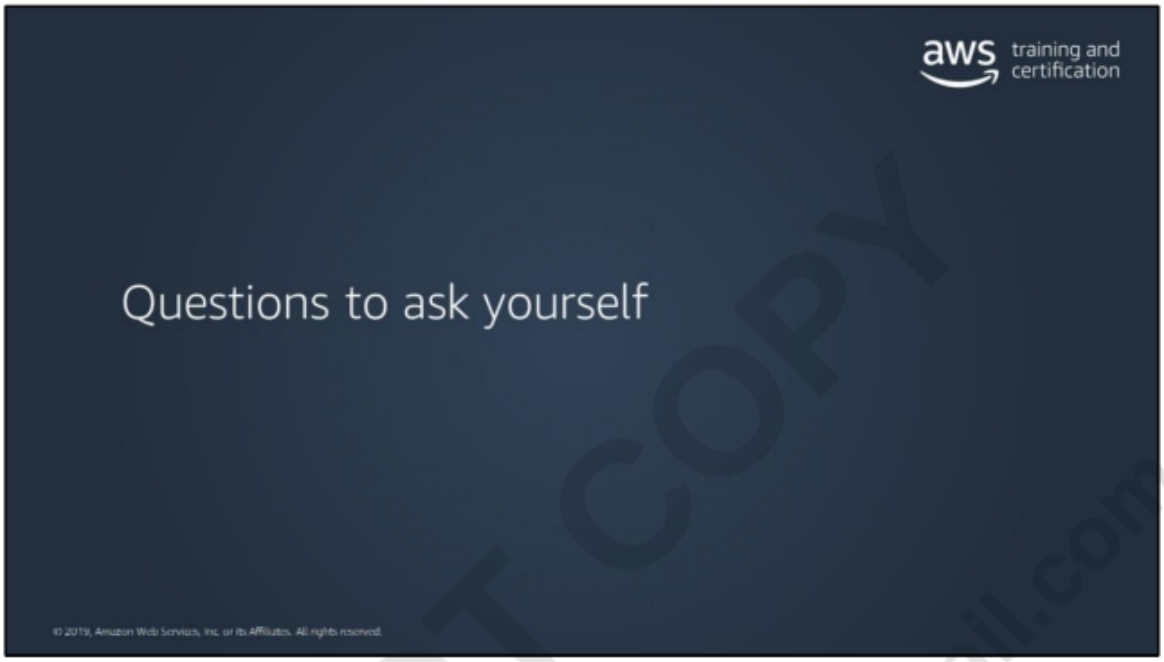
Exercise your DR solution

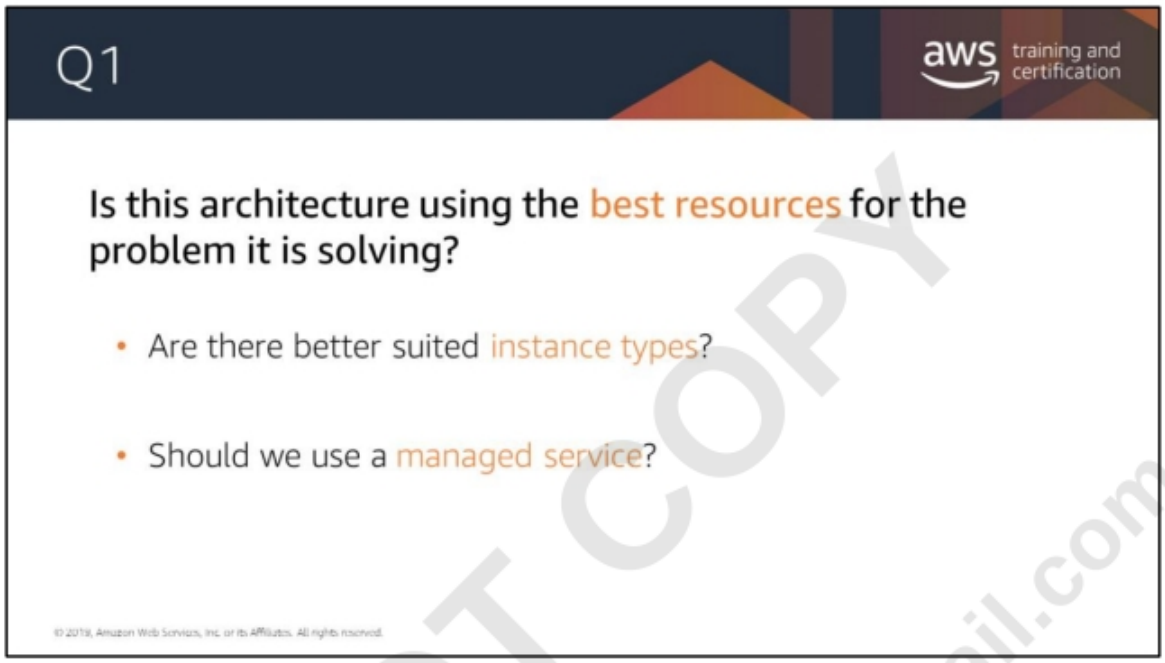
- Practice "Game Day" exercises. These exercises test critical systems going offline or even entire regions. What if an entire fleet were to crash?
- Ensure that backups, snapshots, AMIs, etc. are working.
- Monitor your monitoring system.





By the end of class, you will be able to understand all of the components of this architectural diagram. You will also be able to construct your own architectural solutions that are just as large and robust.





Often, when organizations buy hardware to put in a data center, they're given a budget and buy hardware with an eye to the future. The best hardware today will be obsolete in 18 months (+/-). So, with a one-time purchase, they're forced to buy bigger than they need to stave off future requirements.

So when you're considering building out a data center, ask yourself, "What do I need right now?" If you need to make it larger or smaller, you can.

Another way to think about it is, "Can I really afford to pay for capacity I'm not really using?"

Managed services, such as Amazon RDS, allow you to build resources in AWS without having to figure out all the requirements first. Let's say a developer asks for a Microsoft SQL Server database and you've never installed the software before, it will take considerable time and effort. Instead, find out what kind of database is needed and let AWS build it for you. You have total control of your database, your data, and its access. All AWS does is free you to do the creative work of design and implementation.

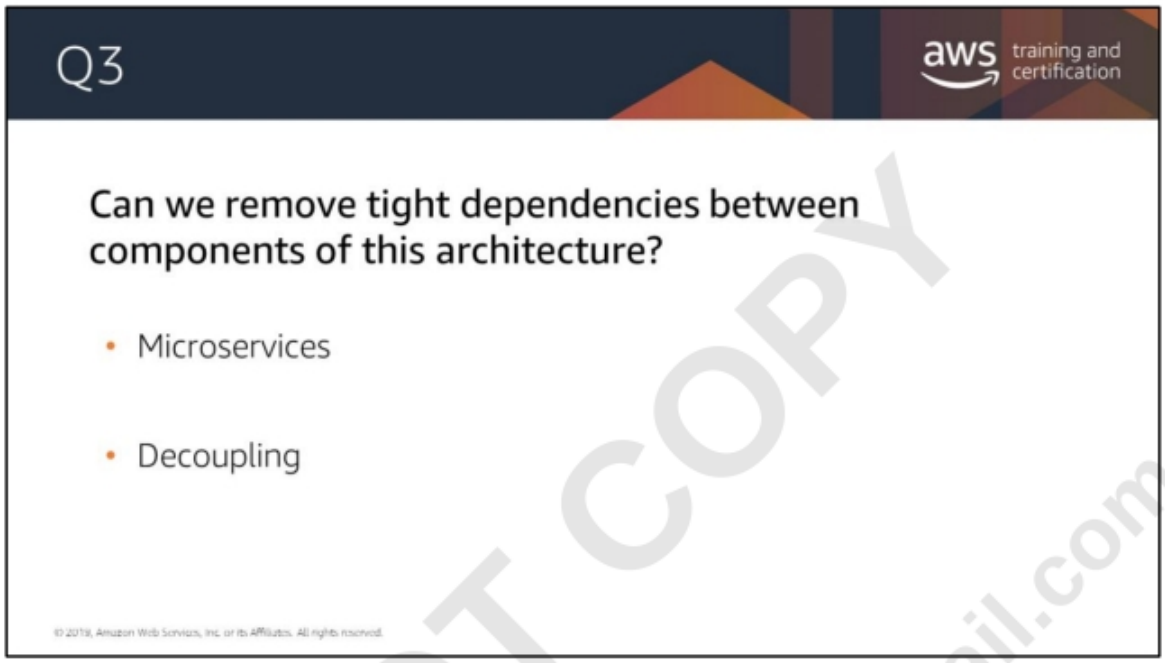
At Amazon, one of our best practices is to start at the end of a process and work backwards. We ask ourselves, What's going to break? If it does, what happens?

Yes, you can fix it, but do you really need to do that? Instead, delete the bits that have gone bad and turn on new ones. You can, and should, do a postmortem report for disasters. However, if an instance, container, or similar thing goes down, replace it first, then figure out why.

In AWS, you can run a full disaster drill without impacting your production environment. Yes, it might cost a couple of dollars, but it's much less expensive than building a second data center.

You've been hired to do creative work—why spend time on manual work that could be automated? Using AWS, you can set up an environment that can monitor itself, delete broken things as needed and then *tell* you what's happened.

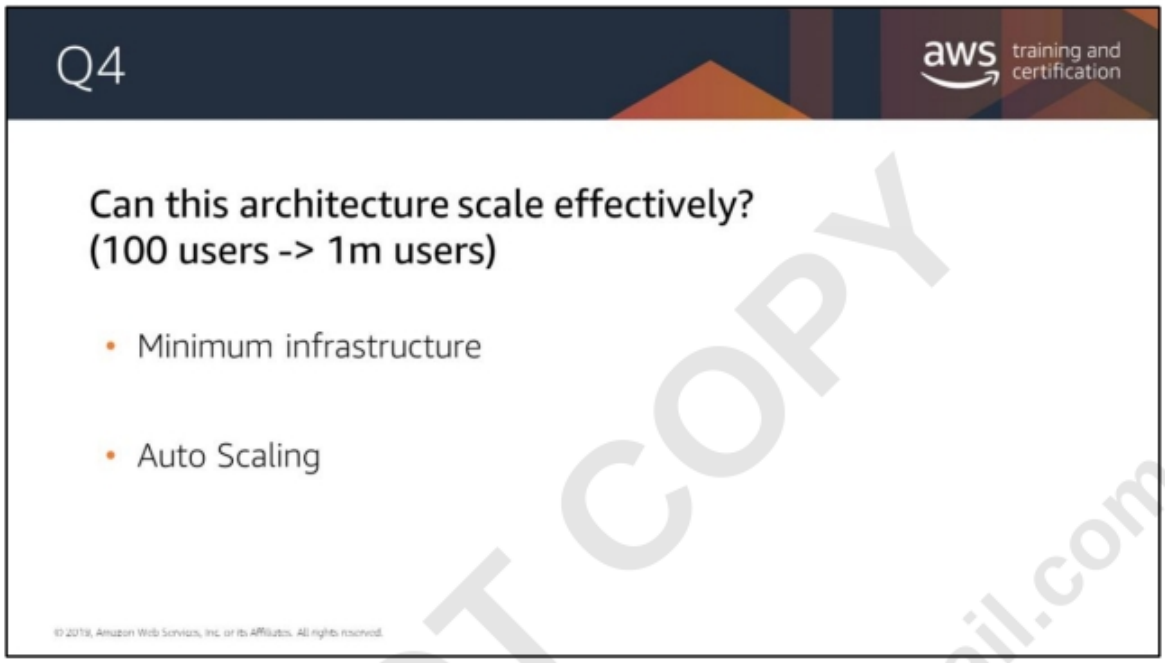
/



Microservices are one of the best ways to increase reliability and availability of your architecture. Using this approach helps to improve your speed of innovation, by reducing your dependencies between application components. Your teams will be able to act asynchronously to deliver feature updates without affecting the whole system.

Taking this kind of approach may require a new way of thinking about how systems work. Monolithic systems have many sunk costs which can be eliminated in the cloud, if you take the time to pull things apart to see how they really work.

/



You may want the biggest, fastest, most powerful machine possible—but if you don't use a machine to its fullest capacity, you're wasting money.

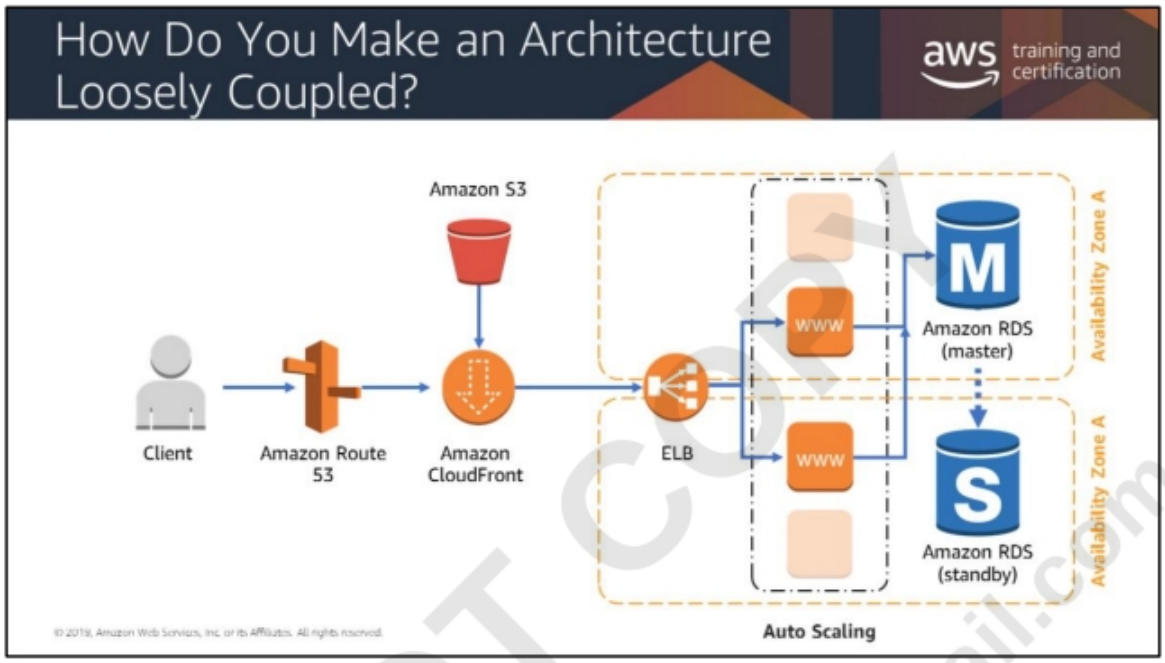
It takes about 3 months to get enough data points to figure out what that number really is. So when you're trying to figure out what capacity you really need, start with a machine that is bigger than you need. Then, once you see what you're using, you can "right-size" your environment.

When you have an on-premises data center, spinning up new resources is absorbed by all the line items involved with the data center itself. When you run a new workload in AWS, there's a cost associated with it. The good news is that in AWS, costs are transparent—and you can see that spinning up a new instance is far less expensive than building out a physical server.

There are two types of failures in the world of IT:

- Something breaks because you asked hardware or software to do something out of its specified tolerances.
- Your product becomes so successful that users ask your hardware and software to exceed its specified tolerances.

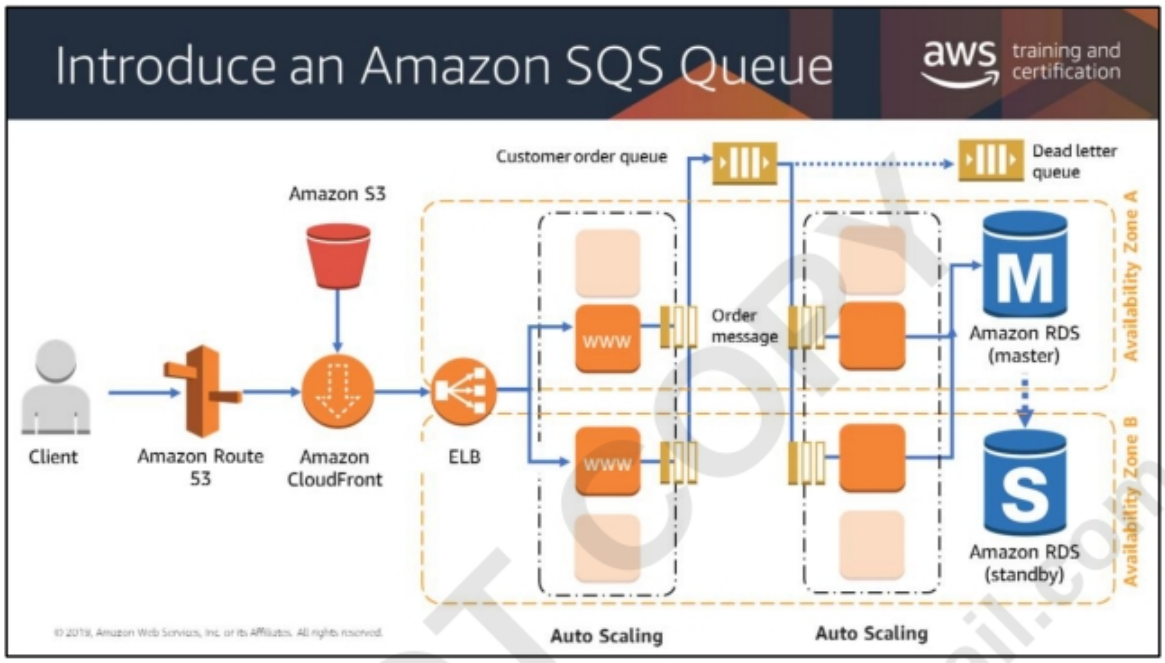
Automatic scaling costs money—but without it, your software will go offline because it can't handle the load. It's worth the investment to keep it running.



In this example, the application is responsible for handling and persisting the order data, as well as dealing with increases in traffic for popular items.

One potential point of vulnerability in the order processing workflow is in saving the order in the database. The business expects that every order has been persisted into the database. However, any potential deadlock, race condition, or network issue could cause the persistence of the order to fail. Then, the order is lost with no recourse to restore the order.

With good logging capability, you may be able to identify when an error occurred and which customer order failed. This wouldn't allow you to "restore" the transaction, and by that stage, your customer is no longer your customer.

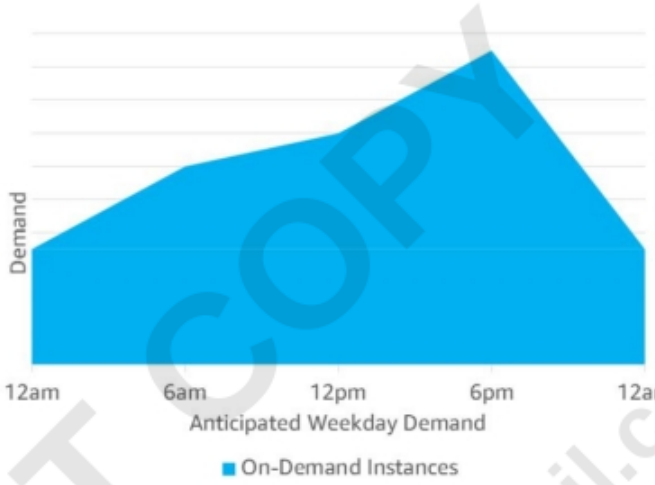


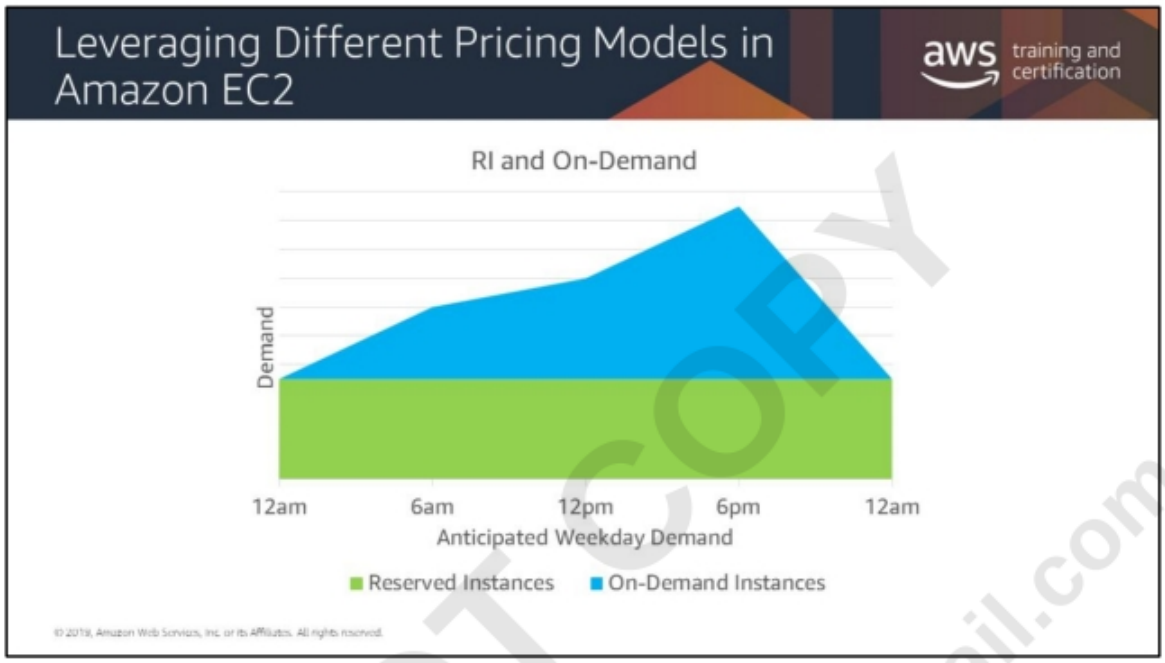
How Can You Reduce Cost in This Scenario?

aws training and certification

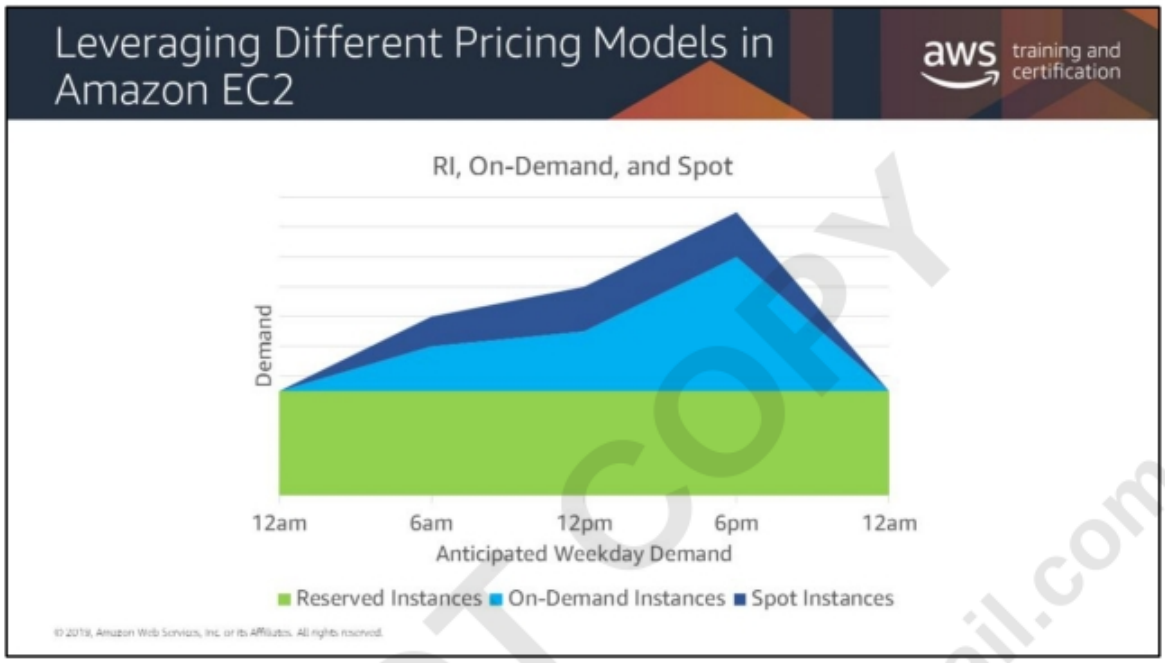
The client has traffic volumes as seen in this graph:

- Spike in traffic during work hours
- Base level of consistent traffic at all times
- Using On-Demand Instances





This chart illustrates a very basic way to use Reserved Instances and On-Demand Instances together to accommodate for fluctuations in demand over time. In this example, multiple Reserved Instances have been purchased and are running, but as the day goes on and demand increases, the need for more instances increases as well. This customer supplements their capacity with On-Demand Instances, which can be shut down when they're not needed later in the evening.



In this second chart, this customer takes a more complex approach, which attempts to leverage all three price models. In this case, some of the supplemental instances come first as On-Demand, but further needs are addressed with Spot Instances. This allows them to save money by using Spot over On-Demand; however it exposes them to unexpected instance termination, because they would lose those instances if they're outbid. This could lead to lost data or insufficient capacity for their customers, which means a model such as this should only be implemented in circumstances where sudden termination of instances is acceptable and handled appropriately.

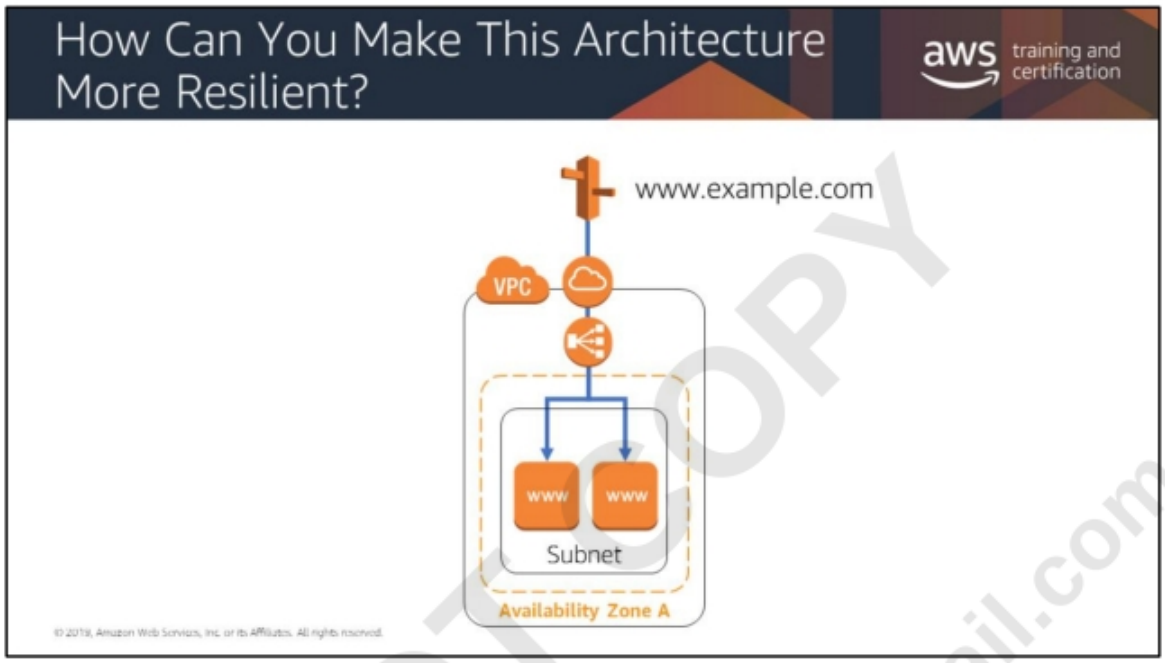
With the new pricing model, introduced in 2017, bidding for spot instances is no longer necessary. Simply pay for the spot price that is in effect for the current hour for the instances launched. Spot prices are more predictable, updated less frequently, and are determined by supply and demand for Amazon EC2 spare capacity. Additionally, to reduce the impact of interruptions and optimize Spot Instances, diversify and run your application across multiple capacity pools. Each instance family, each instance size, in each Availability Zone, in every Region is a separate Spot pool. You can use the RequestSpotFleet API operation to launch thousands of Spot Instances and diversify resources automatically.

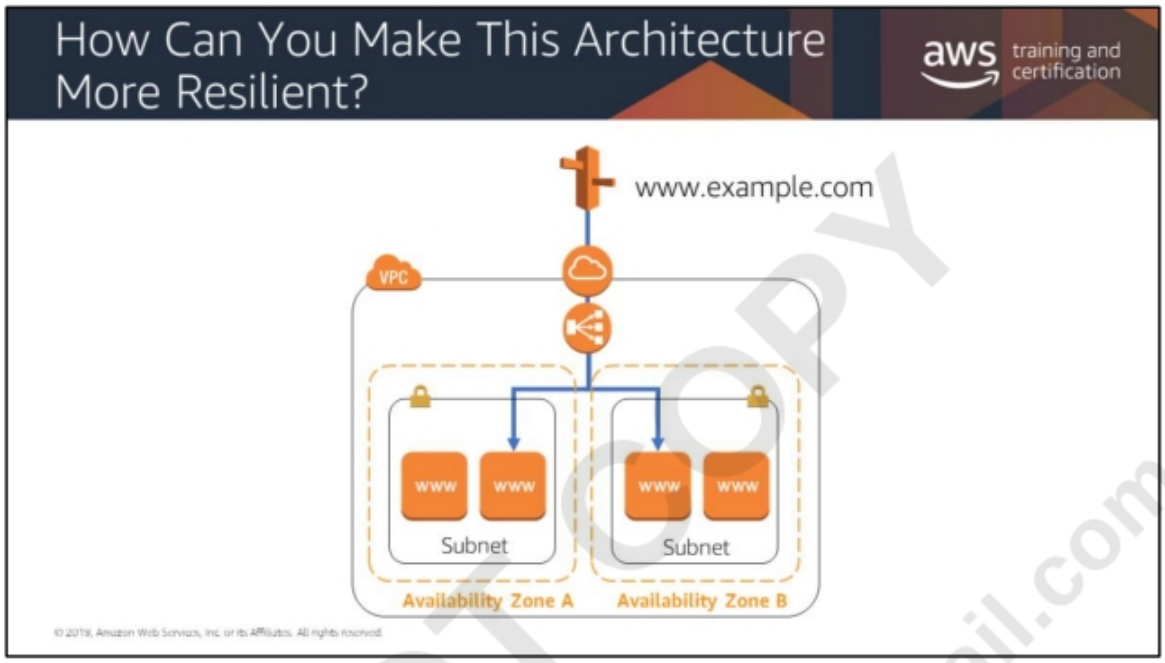
To further reduce the impact of interruptions, you can also set up Spot Instances and Spot Fleets to respond to an interruption notice by stopping or hibernating rather than terminating instances when capacity is no longer available.

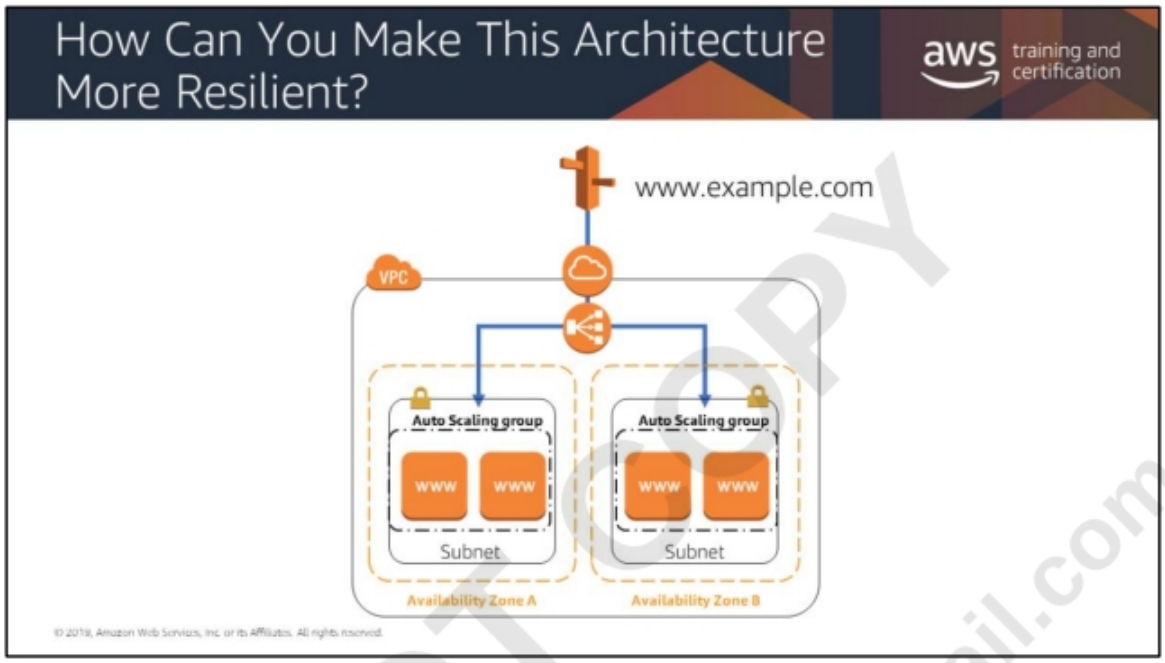
For more information on using Spot Instances, see:
https://aws.amazon.com/ec2/spot/getting-started/
For more information on Spot pricing, see:
https://aws.amazon.com/blogs/compute/new-amazon-ec2-spot-pricing/

One AWS customer who has leveraged all three models together is Pinterest. For more information, see http://www.allthingsdistributed.com/2012/08/tco-and-return-on-agility.html.

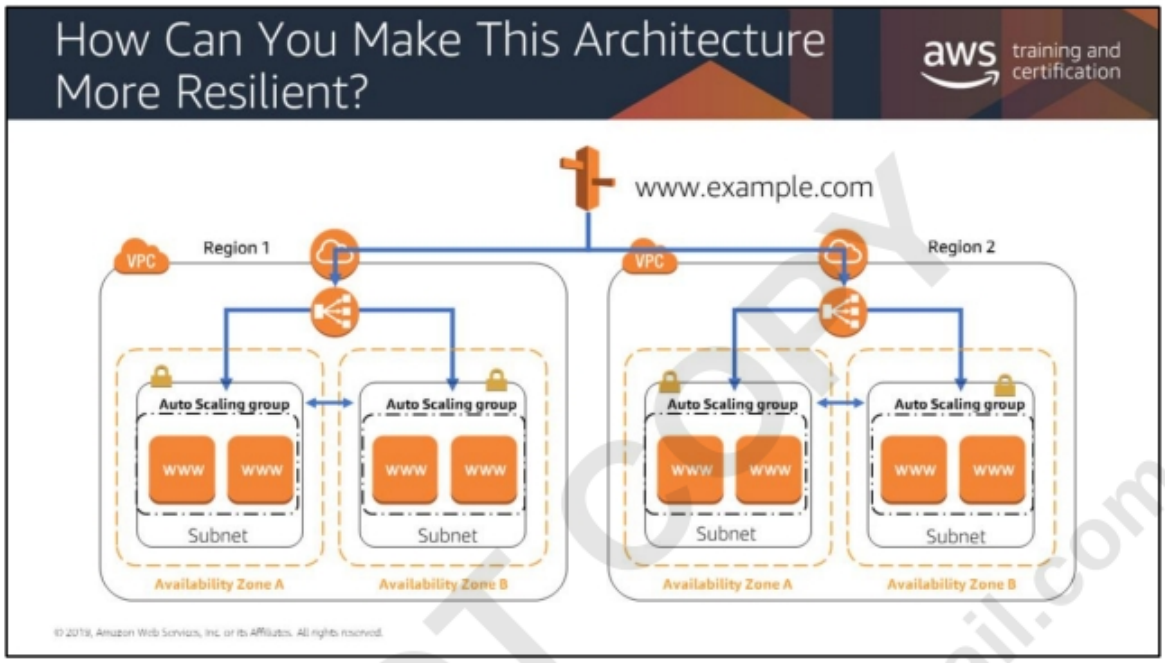
/



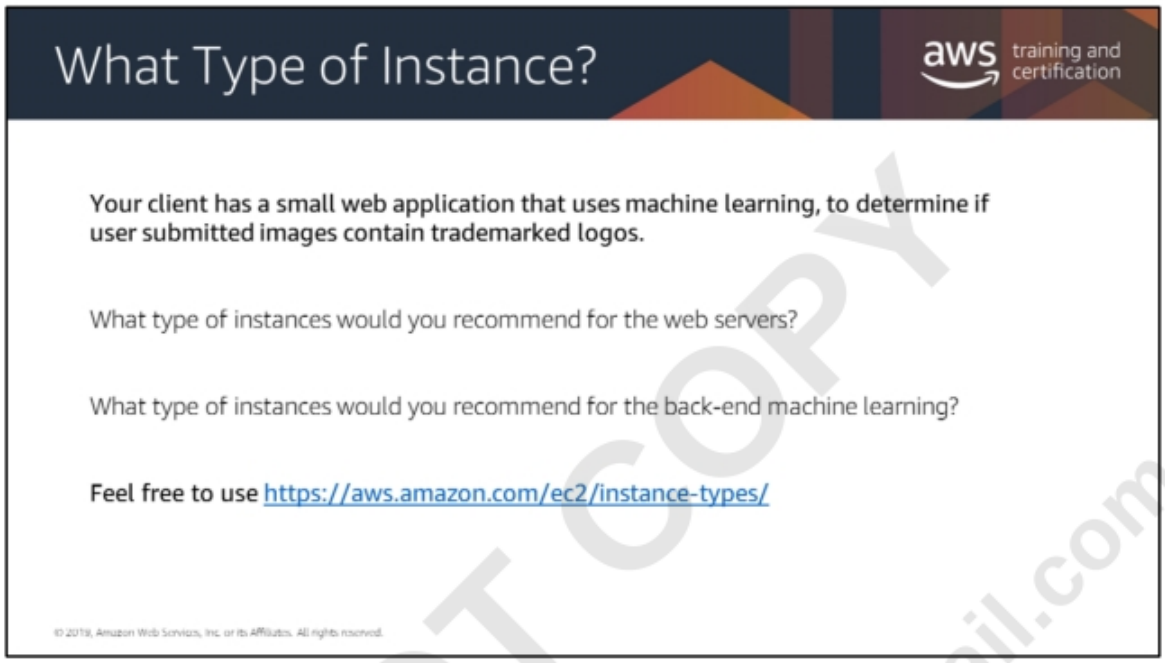




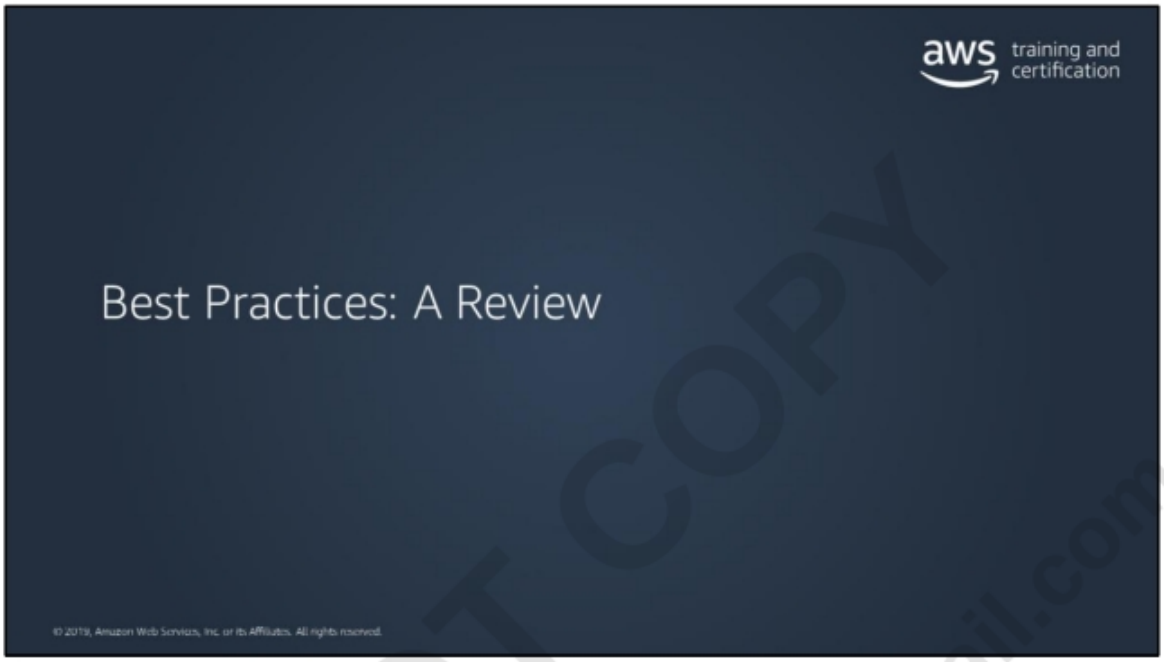
/



Note: Auto scaling groups can span Availability Zones.

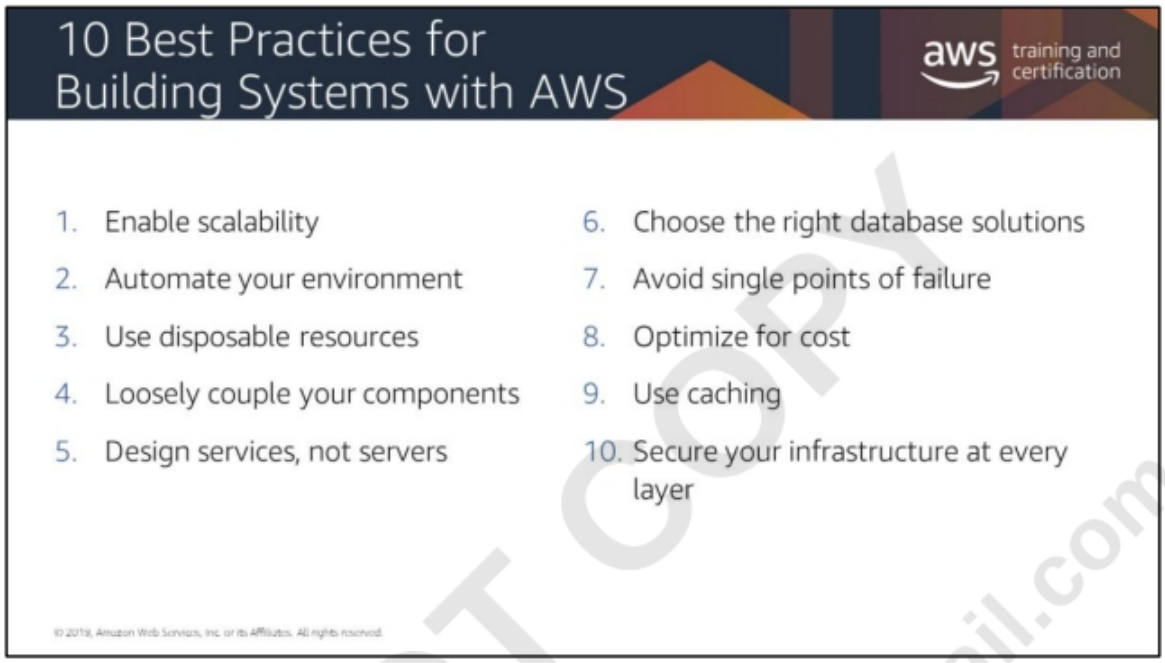


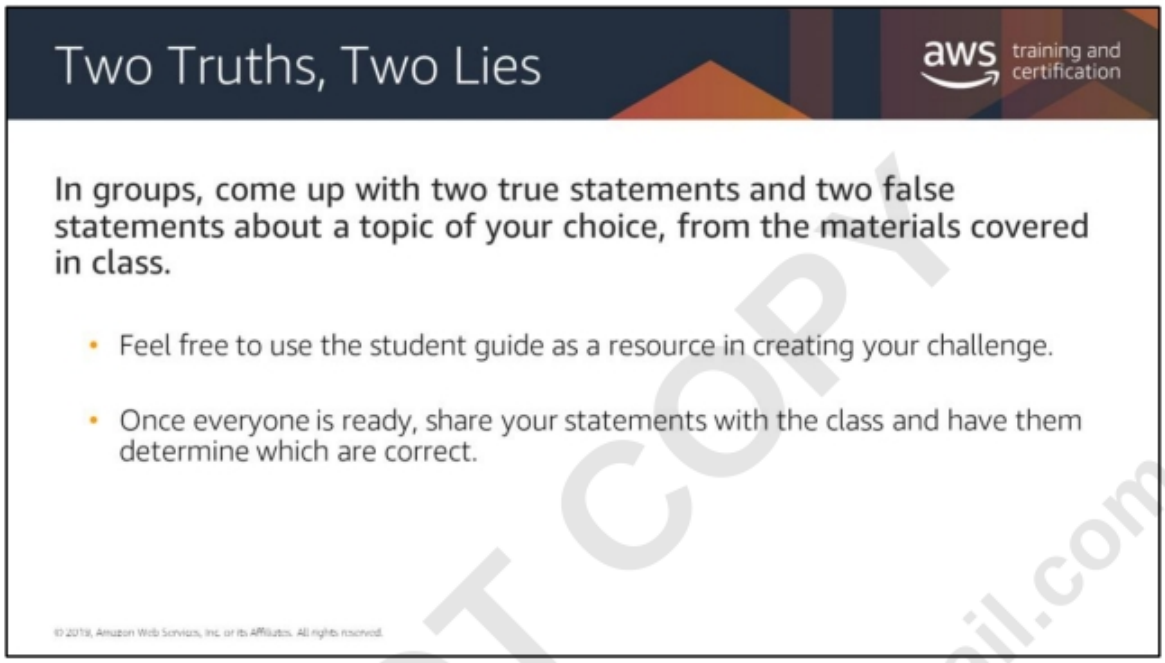
For more information about choosing an instance type, see https://aws.amazon.com/ec2/instance-types/.



The AWS Well Architected Tool is a free tool available in the console for users to help them review and compare their workloads to the latest AWS Architectural best practices. It is based off of the AWS Well-Architected Framework. Has been used in tens of thousands of workload reviews conducted by the AWS solutions architecture team.
https://aws.amazon.com/well-architected-tool/

/



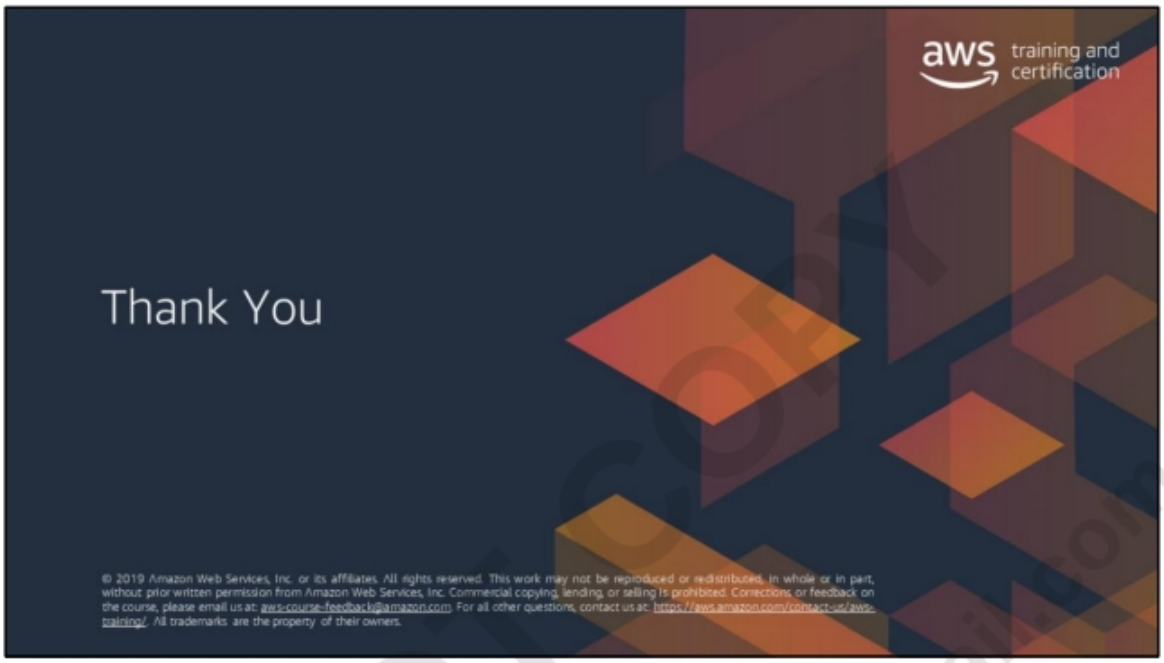


Sample problems:

- Online image resizing app
- Simple online store with order processing
- Video streaming on demand
- Image sharing website with account logins (Facebook/Google/Amazon)
- Online virtual desktop

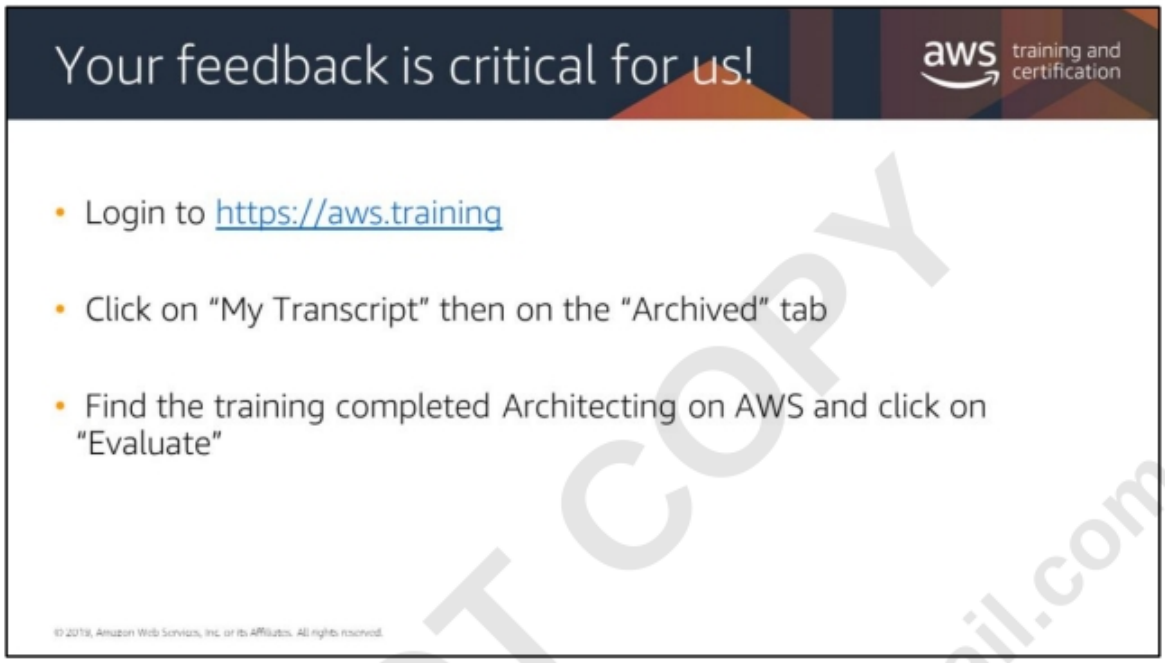
- Feel free to add your own architectural challenges to this exercise.

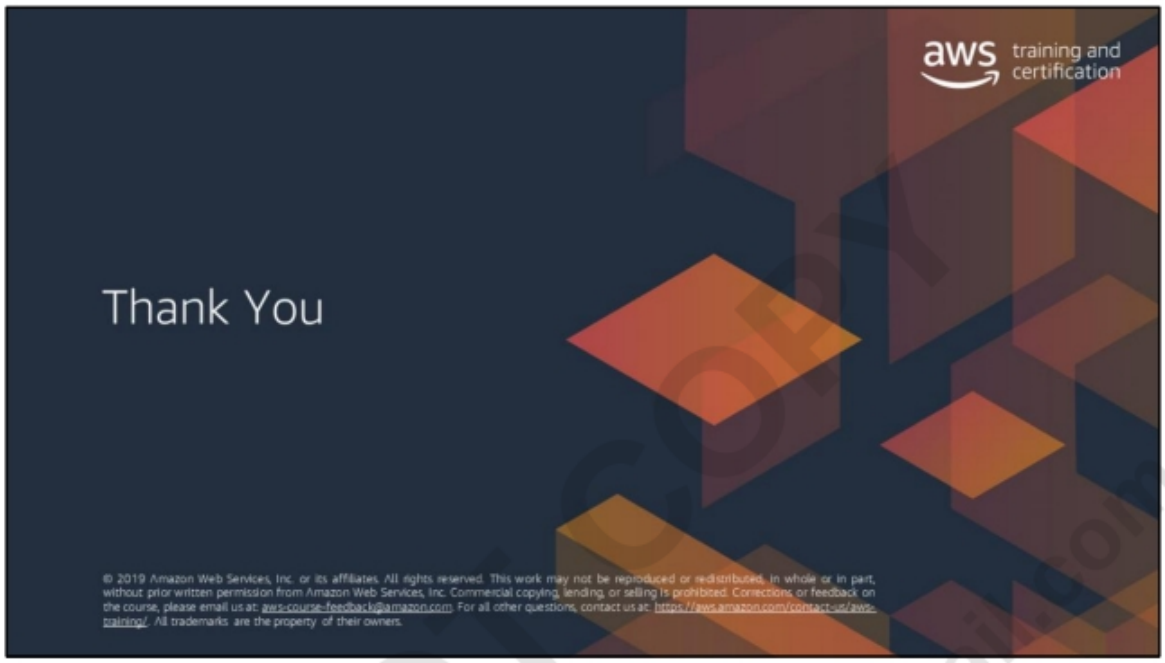
/



/

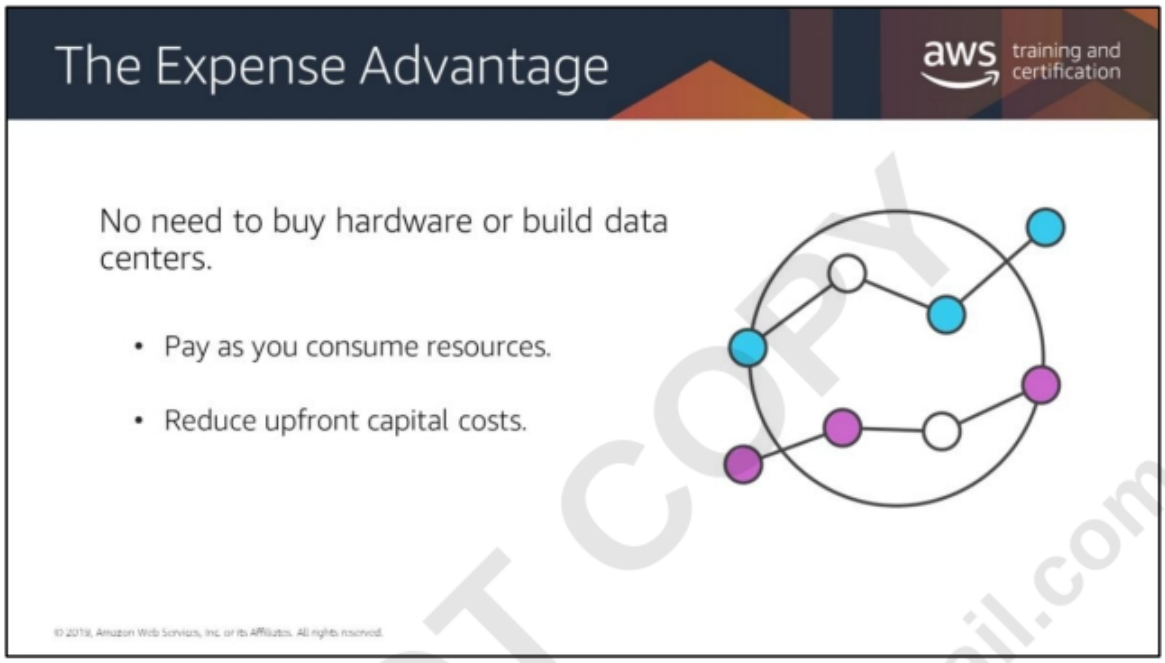
/





/

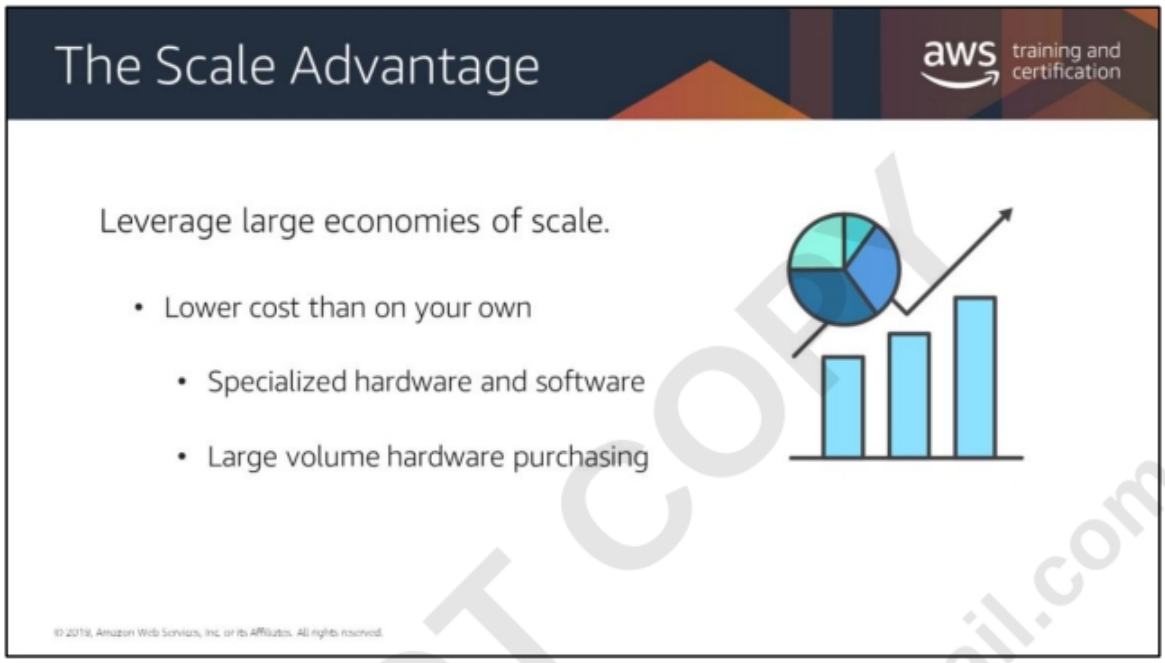
/



Instead of having to invest heavily in data centers and servers before you know how you're going to use them, you can only pay when you consume computing resources, and only pay for how much you consume.

This advantage is particularly good for startups or upfront budget-constrained projects. Being on the forward edge of technology can be risky. Building an on-premises infrastructure yourself might be cost-prohibitive, and it could stall testing, experimentation, and innovation. With the expense advantage, you can quickly get up and running while only paying for what you use.

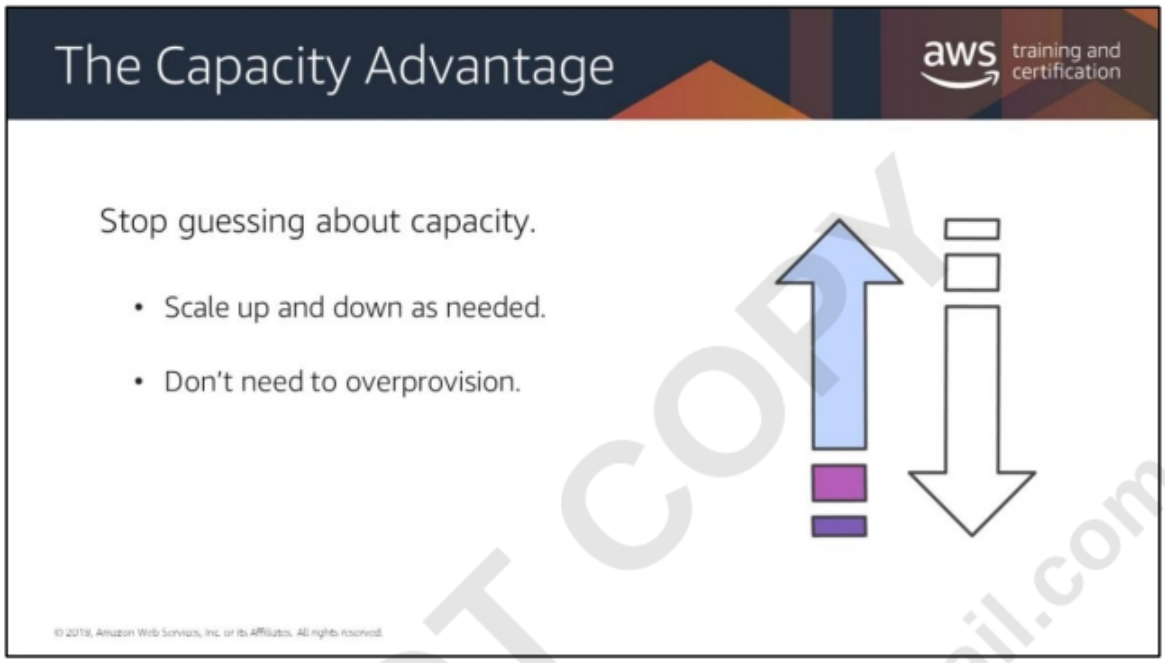
/



By using cloud computing, you can achieve a lower variable cost than you can get on your own. Because usage from hundreds of thousands of customers are aggregated in the cloud, providers like AWS can achieve higher economies of scale, which translates into lower pay-as-you-go prices.

AWS has developed specific hardware and software that is optimized for large-scale clouds. This, combined with the volume of these purchases, allows AWS to support lower costs and high efficiencies than those in most on-premises data centers. These savings are in turn passed on to the customer in order to lower prices and enhance the customer experience.

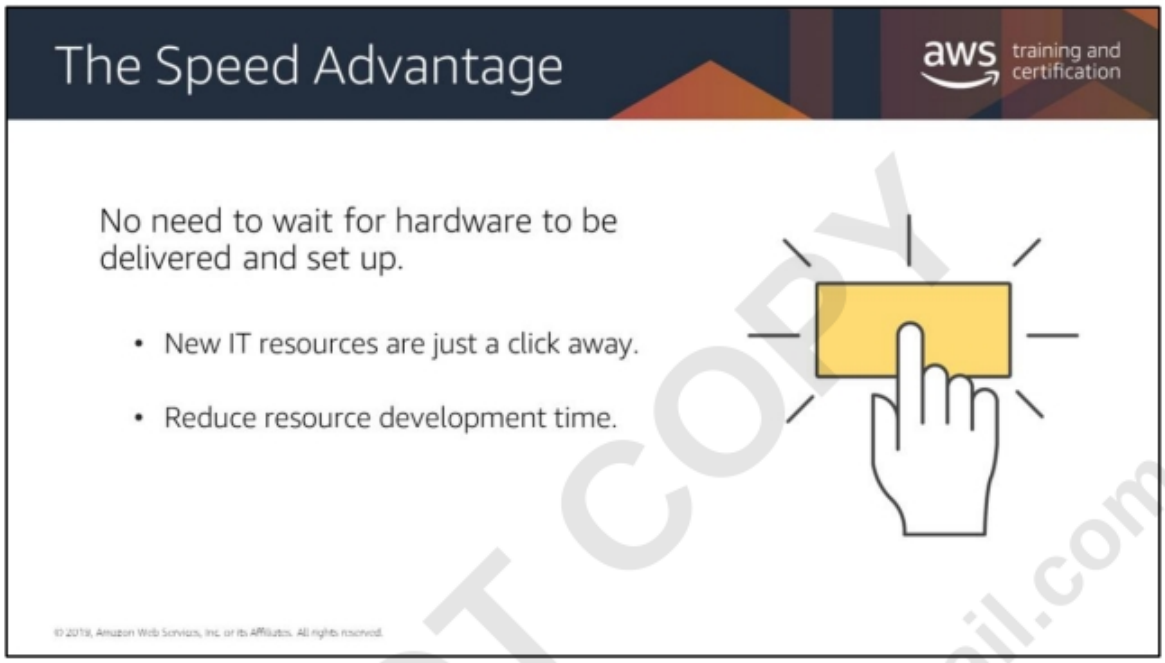
/



Eliminate guessing on your infrastructure capacity needs. When using AWS, you only pay when you consume computing resources, and only pay for how much you consume. You can access as much or as little as you need, and scale up and down, scale in and out as required with only a few minutes' notice.

It's really hard to guess capacity if you don't yet know what the customer response is—for example, if you're launching a new product or service. Scaling your infrastructure as demand shifts and spikes offers a huge advantage over the mostly static on-premises solutions.

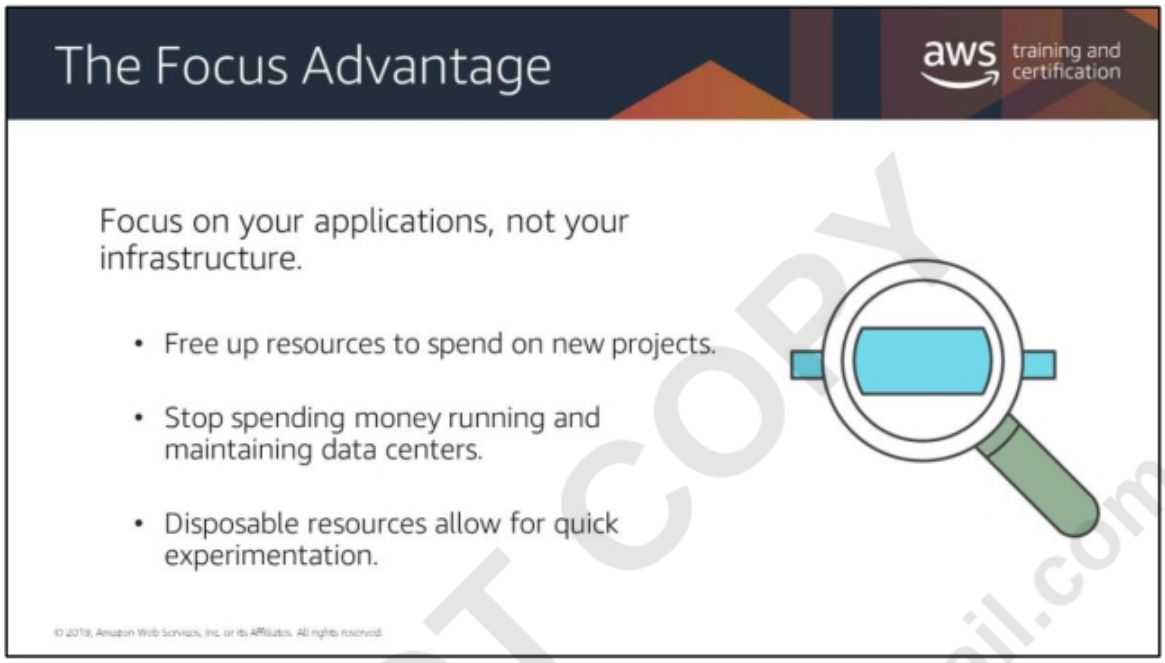
/



In a cloud computing environment, new IT resources are only a click away. You can make resources available to your developers in minutes, instead of weeks. This results in a dramatic increase in agility for the organization, since the cost and time it takes to experiment and develop is significantly lower.

It can take 6 to 20 weeks to provision a server in an on-premises environment. This timeframe can really stifle innovation. With AWS you can provision completely on your own, hundreds or even thousands of services in minutes. This allows you to experiment and create quickly.

/



Focus on projects that differentiate your business, not the infrastructure. Cloud computing lets you focus on your own customers, rather than on the heavy lifting of racking, stacking and powering servers.

The cloud has already done much of the heavy lifting for you. For most companies, their scarcest resource are their software development engineers. Engineering teams have a long list of priorities and tasks that need to be accomplished. It's a significant advantage to focus that resource on projects that move your mission forward, instead of working on underlying infrastructure.

/