This study guide demonstrates the lesson from *Introduction to AWS*.
My full AWS Architect Associate course can be found here:
https://www.udemy.com/course/ultimateaws/?referralCode=7ED214B795C444141361

Introduction to Amazon Web Services Study Guide

Defining Amazon Web Services (AWS)

- AWS is a cloud service provider that offers a range of computing, storage, and networking services.
- It allows users to access these services on-demand without the need to invest in physical infrastructure.

Section 2: Cloud Computing Simplified

The Utility Provider Analogy

- Comparing AWS to traditional infrastructure is akin to the difference between generating your own power and using a utility provider.
- Using AWS is cost-efficient, eliminating the need for upfront hardware investments, physical space, and infrastructure management.

Challenges with Traditional Data Centers

- Traditional data centers require extensive physical security, power, cooling, redundancy, and floor space.
- Ongoing expenses include hardware maintenance, support agreements, and hardware replacement.

Section 3: Pay-As-You-Go Model

Scalability and Efficiency

- AWS adopts a pay-as-you-go model, where you only pay for the resources you consume when you need them.
- This eliminates overprovisioning and ensures cost efficiency.

Restaurant Analogy

- Using a restaurant analogy, building for peak demand requires significant overbuilding, which is costly and inefficient.
- Elasticity in AWS allows resources to expand and contract as needed, saving costs and ensuring peak performance.

Section 4: Focus on Core Business Goals

Aligning with Organizational Goals

- The primary objective of most organizations is not managing data centers but achieving their core business goals.
- AWS allows businesses to offload data center management to professionals, freeing up resources to focus on their primary mission.

Shifting the Focus

- When the complexities of data center management are delegated to experts, businesses can work on optimizing applications and services.
- Rapid responses to changes, new features, and software enhancements become more achievable.

Chapter Review and Key Takeaways

- AWS is a leading cloud service provider that offers a range of computing, storage, and networking services.
- Cloud computing simplifies IT operations by adopting a utility provider model, eliminating the need for extensive physical infrastructure.
- AWS's pay-as-you-go model ensures cost efficiency and scalability, eliminating overprovisioning.
- The focus on core business goals becomes easier with AWS as data center management is handled by professionals.
- AWS enables businesses to respond rapidly to changing demands and achieve their core business objectives.

In this lesson, we've explored the fundamental concepts of AWS and its benefits. AWS simplifies IT operations, provides cost efficiency, and allows organizations to focus on their core business goals. Understanding the paradigm shift from traditional data centers to cloud computing is essential for harnessing the full potential of AWS.

See slides below:

What is AWS?

Rick Crisci

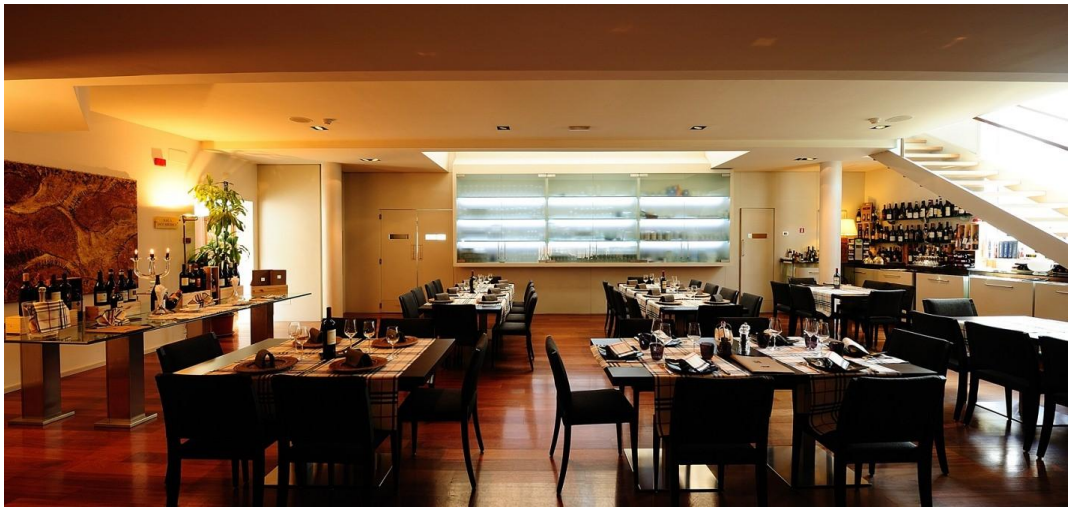
Amazon Web Services

- Cloud service provider
- Move workloads away from a physical datacenter to the cloud
- Pay for the services you need
- Compare to a utility provider

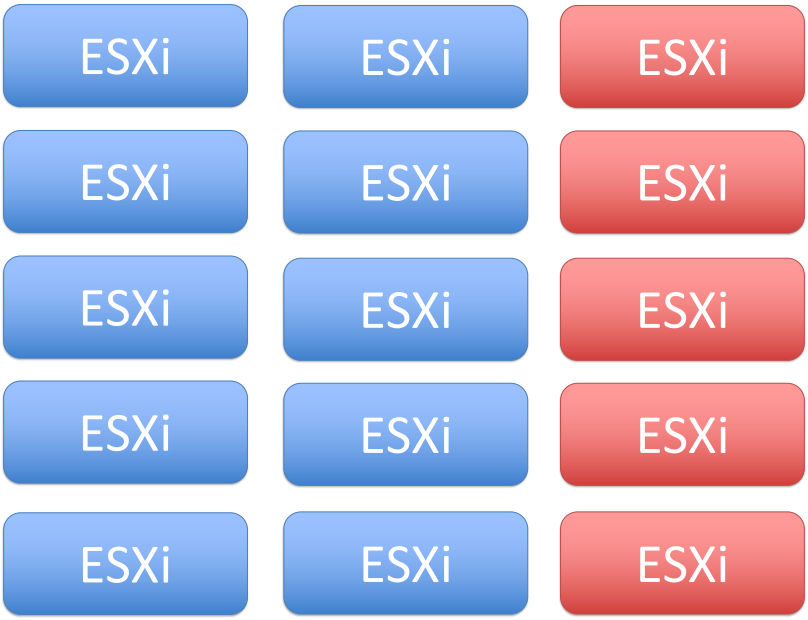
Costs of Physical Datacenters

Security	Cooling	Compute	Storage
Power	Floor Space	Maintenance Contracts	Networking

OverProvisioning



Elasticity in a Physical Datacenter



Elasticity with AWS

- No need to overprovision hardware
- Plenty of resources available at a moment's notice
- No need to rack and cable
- The cost contracts automatically with your usage

The True Information System Goal

Expertise and Systems

For more details see my full AWS Architect Associate course:

https://www.udemy.com/course/ultimateaws/?referralCode=7ED214B795C444141361

This study guide demonstrates the steps from *Sign up for an AWS Account.*
My full AWS Architect Associate course can be found here:
https://www.udemy.com/course/ultimateaws/?referralCode=7ED214B795C444141361

Here is a step-by-step guide to sign up for the AWS Free Tier:

1. Open your web browser and navigate to the AWS Free Tier page.
2. Click on the "Create a Free Account" at the center of the page.
3. Provide your email address and create a username, then create a password.
4. Enter your personal information, including your name, address, and phone number.
5. Provide your payment information. You will not be charged unless you exceed the free tier limits.
6. Confirm your identity by entering the verification code sent to your phone number.
7. Choose a support plan. You can choose the free basic plan or a paid plan.
8. Review your account details and click on "Create Account and Continue".

Once you have created your account, you can start using the AWS Free Tier. The AWS Free Tier provides limited free usage of AWS services. These limits vary from one service to another and are subject to change. Once you exceed these limits, standard service charges apply. For instance, if you exceed 750 hours of EC2 t2.micro instances or the 5GB of S3 storage, you'll be billed for the additional usage.

It's important to note that not all AWS services are free, and some services have usage limits. You will be charged at standard rates if you exceed those limits. The AWS Free Tier expires 12 months from the date you sign up.

For more details see my full AWS Architect Associate course:
https://www.udemy.com/course/ultimateaws/?referralCode=7ED214B795C444141361

This lab demonstrates the steps from *Demo: Learn to Navigate the AWS Console.*
My full AWS Architect Associate course can be found here*:*
https://www.udemy.com/course/ultimateaws/?referralCode=7ED214B795C444141361

Lab 1 – *Working with the AWS Console*

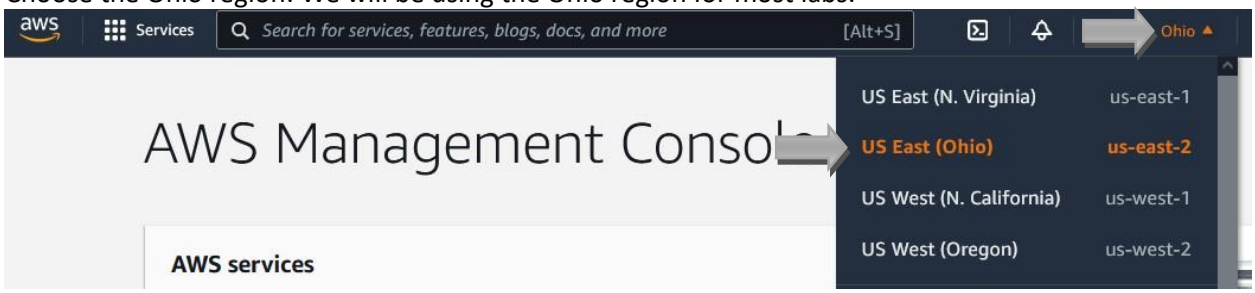
You will need SSH open between your machine and AWS. If you are on a VPN or secure network you should verify that the SSH port is open, or connect from an open network

Log In

1. Go to aws.amazon.com
2. Sign in with your credentials
3. If you have not already configured Multi-Factor Authentication on your root account, this is a good time to do that.
    a. You will need an MFA device. I suggest installing Google Authenticator on your smart phone.
    b. Click on your account name on the top right

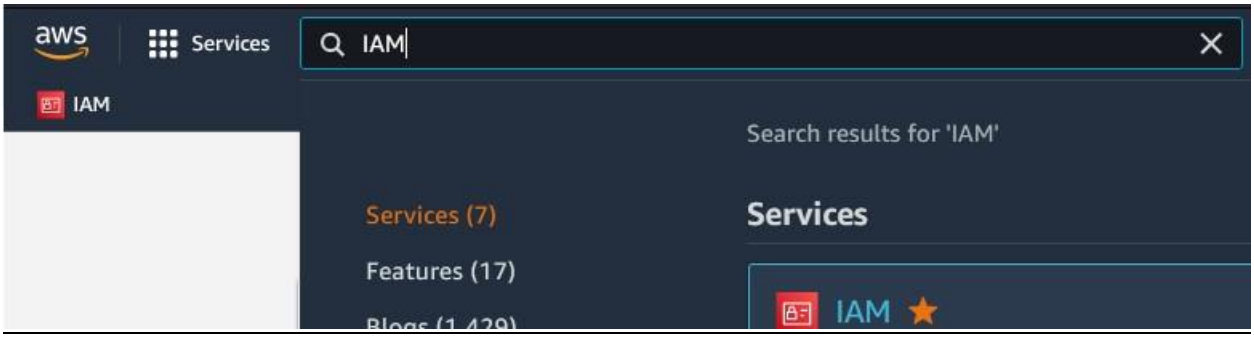
    c. Select **Security Credentials**
    d. Scroll down to **Multi-factor authentication (MFA)**
    e. Click **Assign MFA device**
    f. Give the device a simple name **(Ex. – MyCellPhone)**
    g. Choose your MFA device type and click **Next**
    h. On the next screen you will have to scan the QR code shown into your MFA app to associate the app with your AWS root account.
4. Click on the AWS icon at the top left to return to the home screen.

5. Choose the Ohio region. We will be using the Ohio region for most labs.



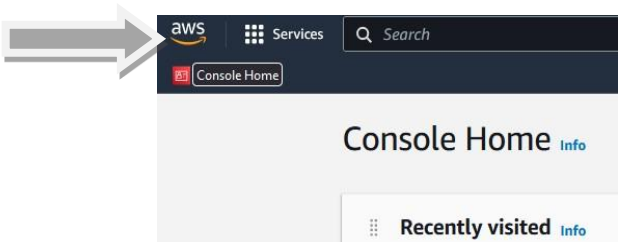
Note: Some services are regional, some are not.
For example, IAM is a global service that spans all regions.
When you change to a global service, the region will automatically change to global.

6. Click in the search bar at the top of the screen and type IAM

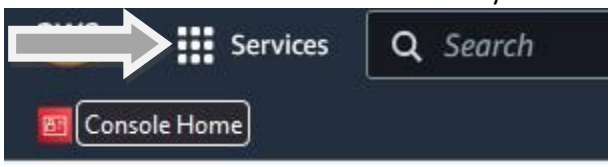


7. Click on **IAM** to navigate to the IAM Dashboard.
8. Notice that the region changes to Global.
   IAM Configuration changes are global to your AWS account and are not local to any specific region.
9. Notice the navigation menu on the left. As you move to different areas of the AWS console this menu will change.
10. On the left, Click **Users**
    This is where you will create IAM users. You should not continue to use the root account to manage your AWS resources.
    Every unique resource created in an AWS account is assigned an identifier called an ARN.

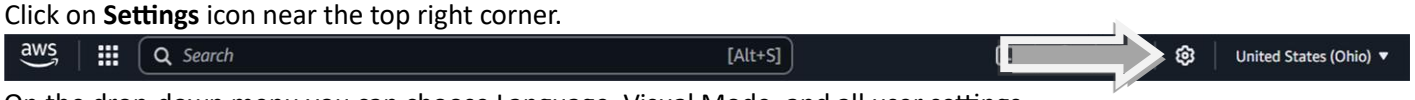
11. Click on the AWS Logo at the top left to go back to the home screen.



12. Click on the **Services** button. Notice that you can browse all the available services here.



13. Click on **Settings** icon near the top right corner.



14. On the drop-down menu you can choose Language, Visual Mode, and all user settings.
    You can change the display to dark mode here if desired.

15. Click on the AWS Logo at the top left to go back to the home screen.

For more details see my full AWS Architect Associate course:
https://www.udemy.com/course/ultimateaws/?referralCode=7ED214B795C444141361

This study guide demonstrates the lesson from *A Quick Overview of Common AWS Use Cases.*
My full AWS Architect Associate course can be found here:
https://www.udemy.com/course/ultimateaws/?referralCode=7ED214B795C444141361

Common Use Cases for AWS Study Guide

In this lesson, we will explore some common use cases for Amazon Web Services (AWS). Understanding how AWS can be employed for various scenarios is essential for organizations looking to harness the power of cloud computing. We'll discuss how AWS addresses data center reduction, backup and disaster recovery, data storage, and the benefits of managed services.

Section 1: Hybrid Cloud and Data Center Reduction

Understanding Elasticity

- Traditional data centers lack elasticity, making it challenging to accommodate fluctuating workloads.
- AWS enables organizations to scale resources on-demand, reducing the need for long-term investments in physical hardware.

Hybrid Cloud Approach

- Organizations can avoid overprovisioning by using AWS resources to handle temporary workload increases.
- Hybrid cloud combines on-premises data centers with cloud resources, optimizing cost and flexibility.

Section 2: Backup and Disaster Recovery

Legacy Backup Methods

- Traditional backup methods often involve writing data to physical tapes, which can be time-consuming and costly.
- Data backups are essential for business continuity and disaster recovery.

AWS Storage Gateway

- AWS provides a solution called Storage Gateway, a virtual appliance that mimics a tape library but sends data directly to AWS.
- This approach ensures backups are instantly shipped off-site to AWS, enhancing data protection.

Disaster Recovery with AWS

- AWS facilitates disaster recovery planning through solutions like pilot light models.
- Image-based backups and synchronization allow rapid recovery in the event of disasters, without the need for extensive physical facilities.

Section 3: Data Storage Simplified with AWS S3

Storage Challenges in Traditional Data Centers

- On-premises data centers require significant investments in storage technology, licenses, and maintenance.
- High-performance storage, such as SSDs, comes at a premium cost.

Amazon S3 - The Simple Storage Service

- AWS offers Amazon S3 (Simple Storage Service), a virtually unlimited storage solution with various storage classes and flexible pricing.
- Organizations pay only for the storage capacity and data transfer they actually use, making it cost effective.

Managed Services in AWS

- AWS provides managed services like S3 that abstract the underlying infrastructure, eliminating the need for hardware management.
- These services are similar to autopilot mode, allowing organizations to focus on their core objectives without worrying about maintenance.

Section 4: Leveraging Managed Services

Managed Services in AWS

- AWS offers a wide range of managed services, such as Amazon Redshift for data warehousing and DynamoDB for NoSQL databases.
- These services are pre-configured and maintained by AWS, reducing the complexity of building custom solutions.

Benefits of Managed Services

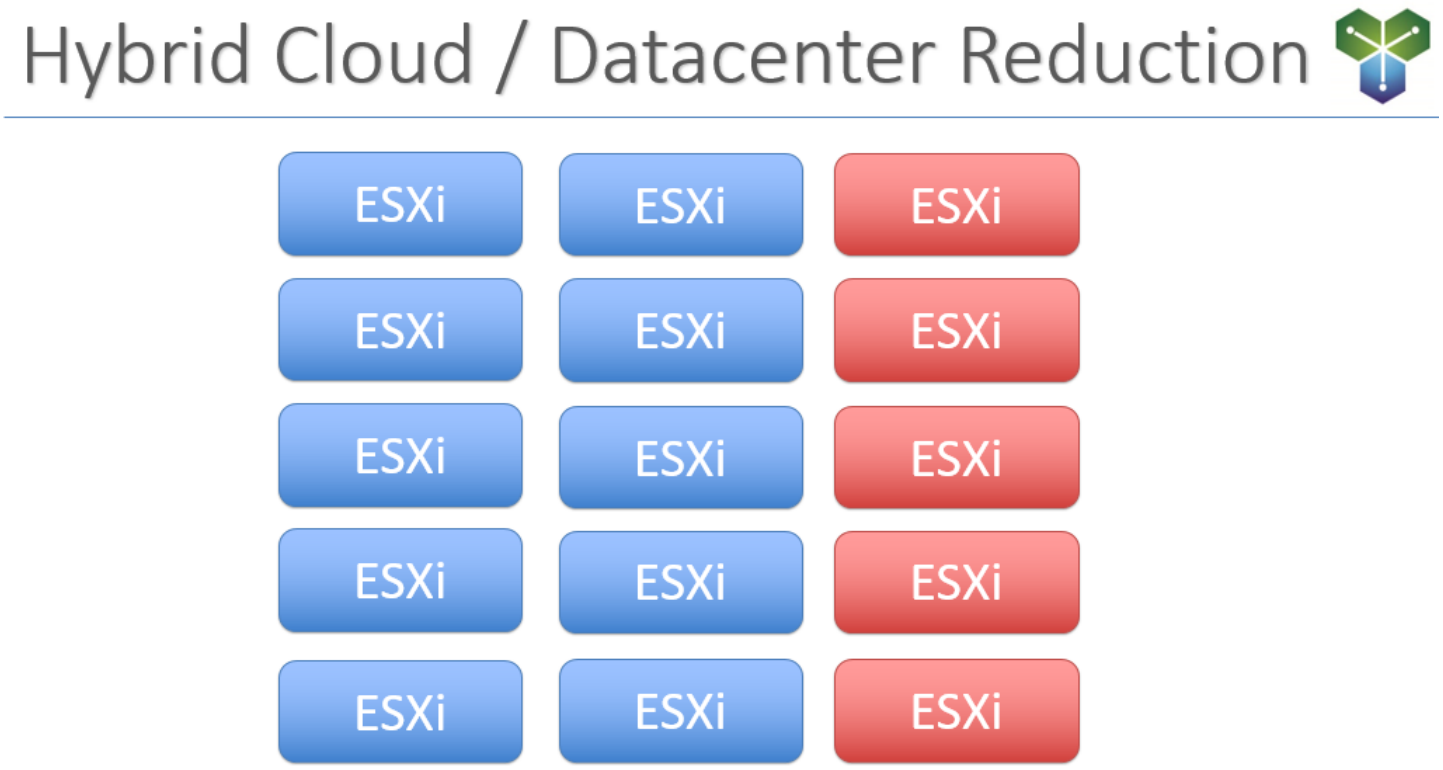
- Organizations can leverage managed services to avoid the time-consuming process of building, maintaining, and patching their own solutions.

- Managed services in AWS provide a simplified, efficient, and cost-effective way to address various business needs.

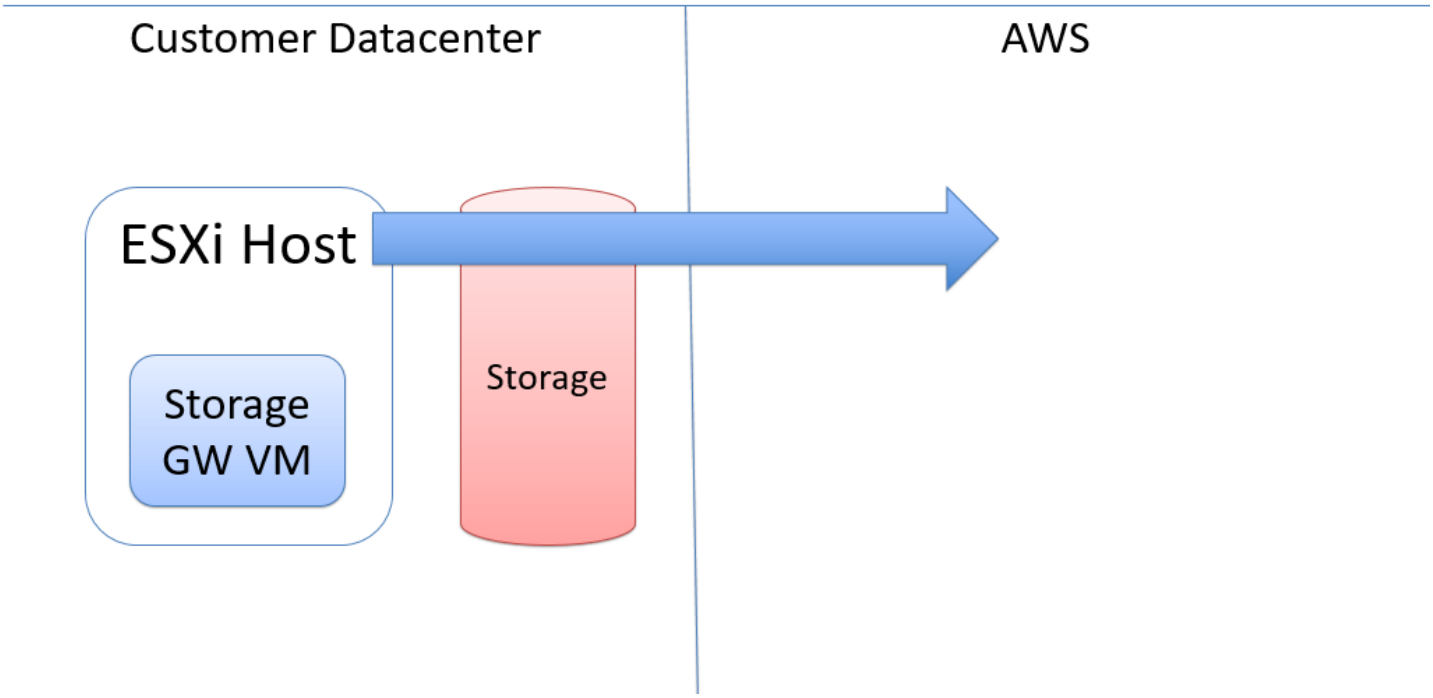
Lesson Review and Key Takeaways

- AWS offers practical solutions for addressing common IT challenges, such as hybrid cloud and data center reduction.
- Backup and disaster recovery are simplified with AWS's Storage Gateway and image-based backup solutions.
- Amazon S3 provides scalable and cost-effective data storage with various storage classes.
- Managed services in AWS allow organizations to access pre-configured solutions, eliminating the need for extensive infrastructure management.
- Understanding how AWS can be applied to specific use cases is crucial for organizations aiming to optimize their IT operations and reduce costs.

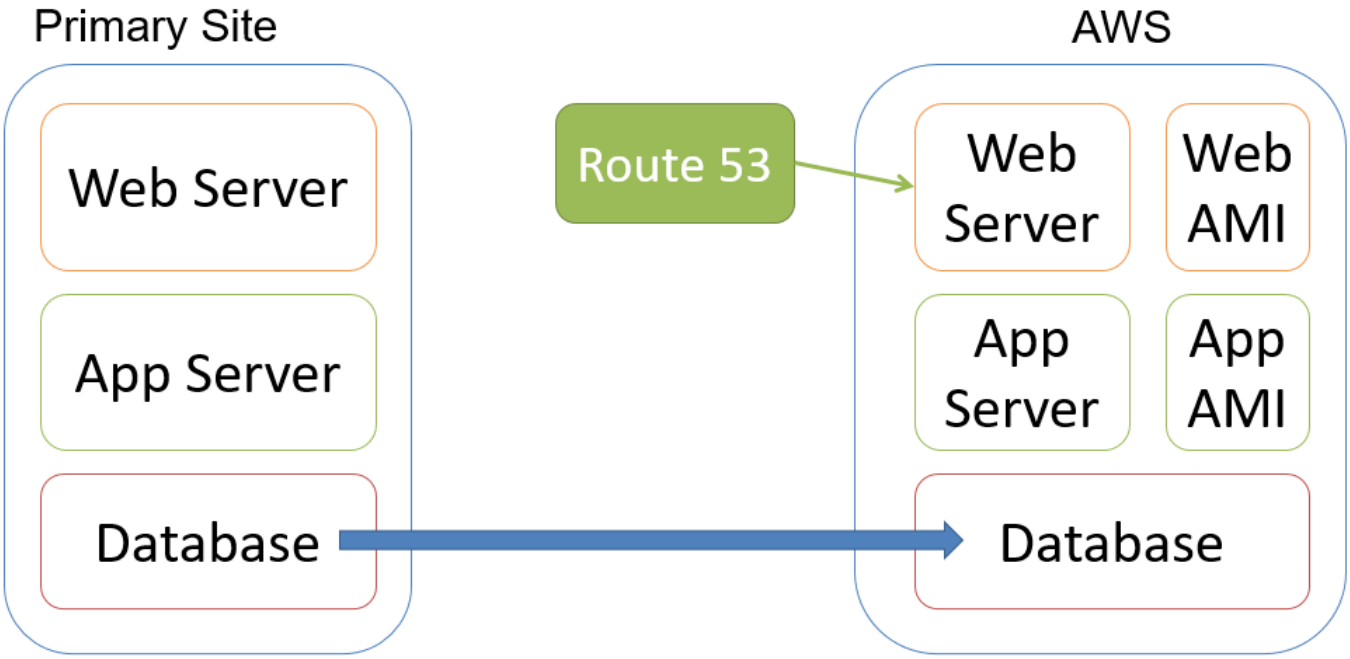
See slides below:



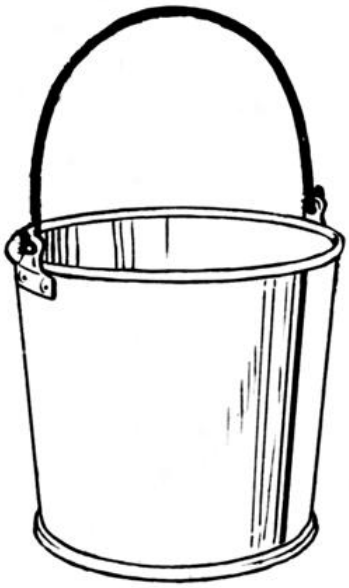
Backup



Disaster Recovery



Data Storage



Managed Services

For more details see my full AWS Architect Associate course:
https://www.udemy.com/course/ultimateaws/?referralCode=7ED214B795C444141361

This study guide demonstrates the lesson from *Even More AWS Use Cases!*
My full AWS Architect Associate course can be found here:
https://www.udemy.com/course/ultimateaws/?referralCode=7ED214B795C444141361

More AWS Use Cases Study Guide

In this lesson, we will explore how Amazon Web Services (AWS) can support high-performance applications and provide global scalability for your business needs. We will delve into a fictitious scenario involving financial data analysis, where the computational power, hardware, and storage requirements are substantial. We'll examine the cost-efficiency of using AWS for this purpose and explore the ability to scale globally with AWS services like Amazon CloudFront.

Section 1: High-Performance Data Analysis

Scenario Overview

- Imagine a scenario in which complex financial data is gathered continuously, and extensive data analysis is needed each night for risk assessment.
- This scenario requires significant computational power, high-performance hardware, and efficient data storage for timely analysis.

On-Premises Challenges

- Meeting the computational demands may require expensive, powerful servers with substantial compute resources.
- Maintaining idle equipment during non-analytical hours can result in wasted resources and increased costs.

AWS Solution

- AWS offers a scalable solution to analyze data quickly, leveraging massive computing power.
- By using hundreds or thousands of virtual servers, organizations can process data efficiently and cost effectively.
- Data can be stored in Amazon S3, and costs are reduced immediately after task completion.

Section 2: Managed Services for High-Performance Applications

Cost-Efficient Computing

- AWS's pay-as-you-go model allows organizations to allocate computing resources as needed, reducing costs during idle periods.

Managed Services

- AWS provides managed services for high-performance tasks, eliminating the need for complex hardware and software setups.
- Managed services simplify resource provisioning and offer efficient solutions for various high-performance requirements.

Section 3: Global Expansion with AWS CloudFront

Global Content Delivery

- Organizations often need to distribute content or services to users worldwide, requiring a global presence.

AWS CloudFront for Global Content Delivery

- Amazon CloudFront is an AWS service designed to distribute content globally and reduce latency for users.
- It replicates content across multiple edge locations, ensuring fast and responsive access for geographically dispersed users.

Global Expansion with AWS Infrastructure

- AWS's extensive global infrastructure, including data centers, regions, and availability zones, enables organizations to scale globally without managing multiple data centers.

Lesson Review and Key Takeaways

- AWS provides a robust solution for high-performance data analysis, offering scalable and cost-effective resources.
- Managed services within AWS simplify resource provisioning and eliminate the need for extensive hardware and software setups.
- AWS CloudFront facilitates global content delivery, reducing latency for geographically dispersed users and expanding an organization's global reach.
- Understanding how AWS can meet high-performance application requirements and enable global expansion is essential for organizations looking to address modern business demands efficiently and effectively.

See slides below:

High Performance Applications

- Financial data is gathered all day
- Massive amount of data that needs to be analyzed each night
- What hardware is required?
- What software is required? (Managed Services)

Go Global in Minutes

For more details see my full AWS Architect Associate course:
https://www.udemy.com/course/ultimateaws/?referralCode=7ED214B795C444141361

This study guide demonstrates the lesson from *Summary of Key AWS Services.*
My full AWS Architect Associate course can be found here:
https://www.udemy.com/course/ultimateaws/?referralCode=7ED214B795C444141361

AWS Services Overview Study Guide

In this lesson, we will provide an overview of some key AWS services that are commonly used. AWS offers a vast array of services catering to a wide range of use cases. We will discuss the fundamental services, and you can explore more detailed information in the AWS documentation.

Section 1: AWS Services Page

Introduction to AWS Services

- AWS provides a comprehensive set of cloud services designed to help you build and deploy applications, manage infrastructure, and more.
- AWS services cover a wide spectrum of use cases, and we'll explore some of the fundamental ones in this chapter.

Section 2: Virtual Private Cloud (VPC)

Understanding VPC

- VPC, or Virtual Private Cloud, is the networking backbone of your AWS environment.
- It allows you to create and manage your private network within AWS, complete with subnets, route tables, security groups, and more.

Section 3: Amazon EC2 (Elastic Compute Cloud)

Introduction to EC2

- Amazon EC2 is a fundamental AWS service that allows you to run virtual servers (EC2 instances) in the cloud.
- EC2 instances can be used for various purposes, such as web hosting, application development, or data analysis.

Section 4: Amazon RDS (Relational Database Service)

About Amazon RDS

- Amazon RDS is a managed database service that simplifies database administration tasks.
- It supports various database engines like SQL, Oracle, MySQL, and PostgreSQL.
- RDS instances are designed to provide high availability, automated backups, and easy scalability.

Section 5: Amazon Route 53

Understanding Route 53

- Amazon Route 53 is a managed Domain Name System (DNS) service.
- It routes incoming traffic to your AWS resources or performs health checks for resource availability.
- Route 53 plays a critical role in connecting users to your applications hosted on AWS.

Section 6: AWS Lambda

Introduction to AWS Lambda

- AWS Lambda is a serverless computing service that enables you to run code without provisioning or managing servers.
- Lambda functions are event-driven and execute only when triggered.
- It's an efficient way to perform tasks, process data, or run applications in a serverless environment.

Section 7: Amazon S3 (Simple Storage Service)

About Amazon S3

- Amazon S3 is a scalable object storage service designed to store and retrieve data.
- It provides unlimited storage capacity for various data types, including media files, backups, and more.
- Amazon S3 offers high durability and availability for your data.

Section 8: Amazon Glacier

Understanding Amazon Glacier

- Amazon Glacier is a low-cost storage service for long-term data archiving.
- It is ideal for storing data for compliance or backup purposes for extended periods.
- Glacier offers cost-effective, secure data storage.

Section 9: AWS Identity and Access Management (IAM)

Introduction to IAM

- AWS Identity and Access Management (IAM) is the service that enables you to control access to your AWS resources.
- You can create users, groups, and roles, and define permissions for them.
- IAM is vital for maintaining security and managing user access.

Section 10: Amazon CloudWatch

About Amazon CloudWatch

- Amazon CloudWatch is a monitoring service for AWS resources and applications.
- It provides real-time data and insight into resource utilization, performance, and operational health.
- You can set alarms and automate actions based on CloudWatch metrics.

Lesson Review and Key Takeaways

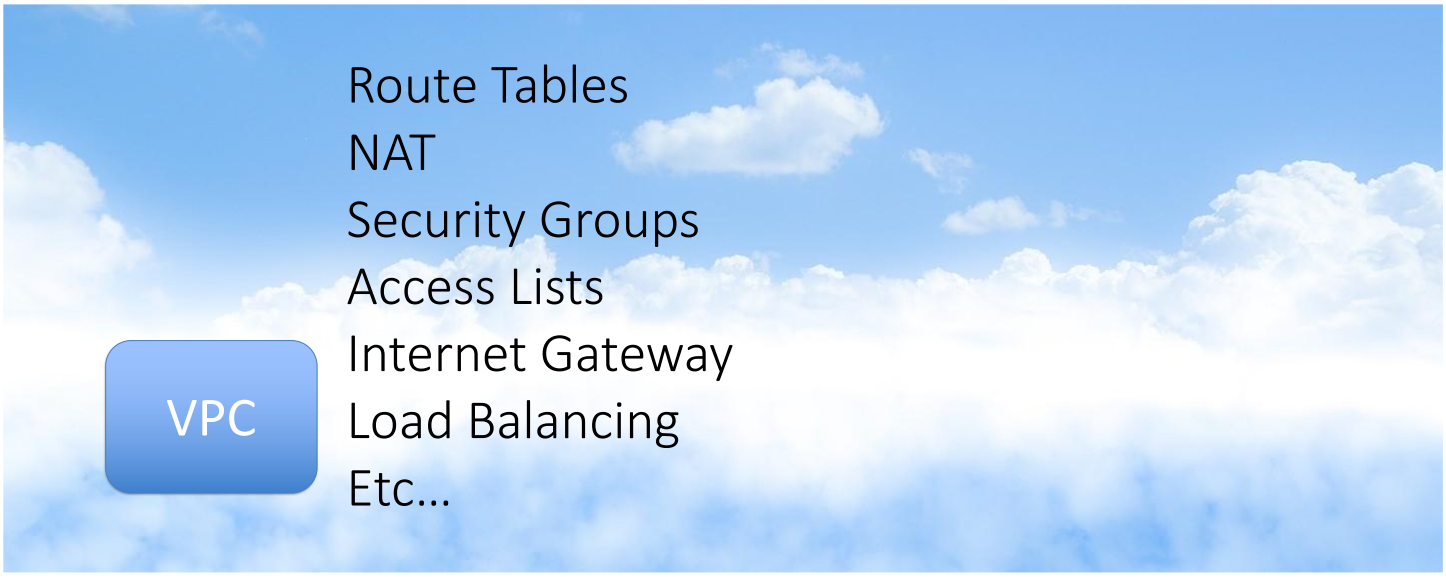
- AWS offers a vast and diverse collection of services for different use cases.
- Understanding the core services is essential for building and managing your cloud infrastructure.
- These services lay the foundation for various cloud computing applications and solutions.
- For more in-depth knowledge and certifications, consider pursuing AWS Certified Solutions Architect Associate or other AWS certifications.

See slides below:

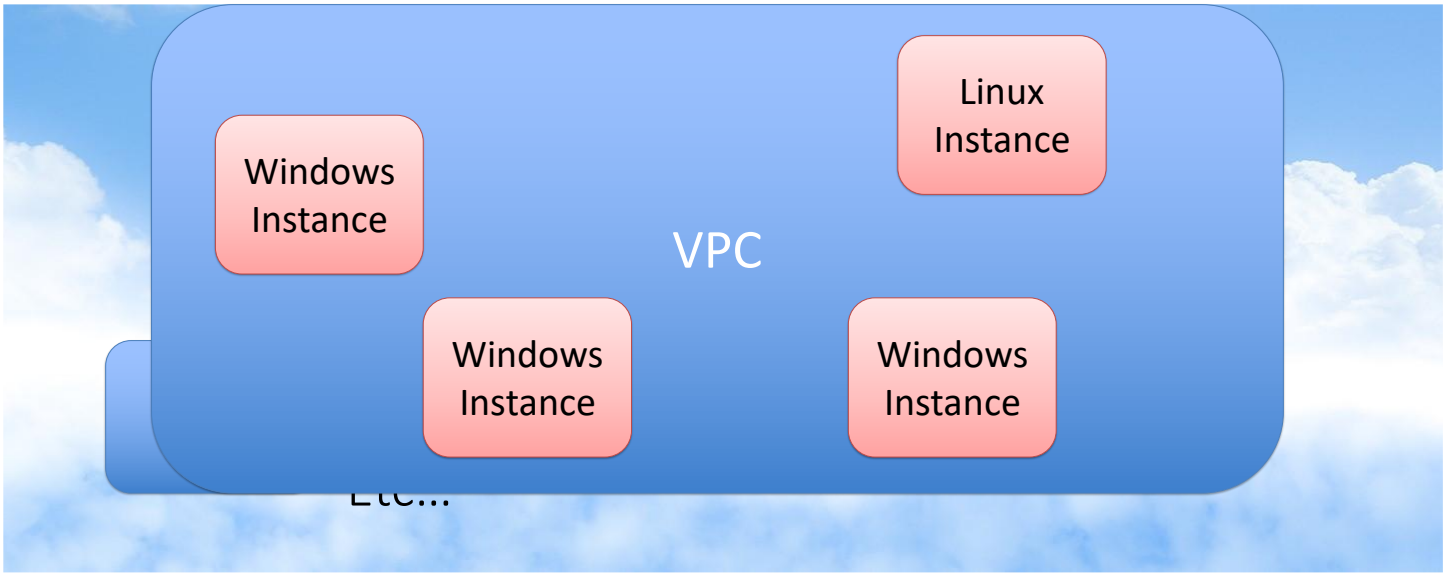
AWS Services

Rick Crisci

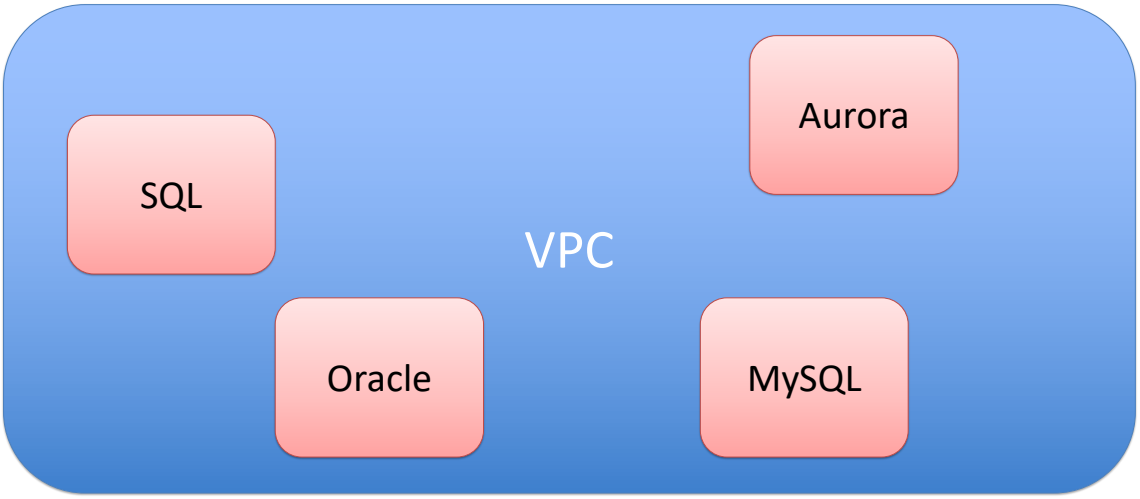
Virtual Private Cloud (VPC)



Elastic Compute Cloud (EC2)



Relational Database Service (RDS)



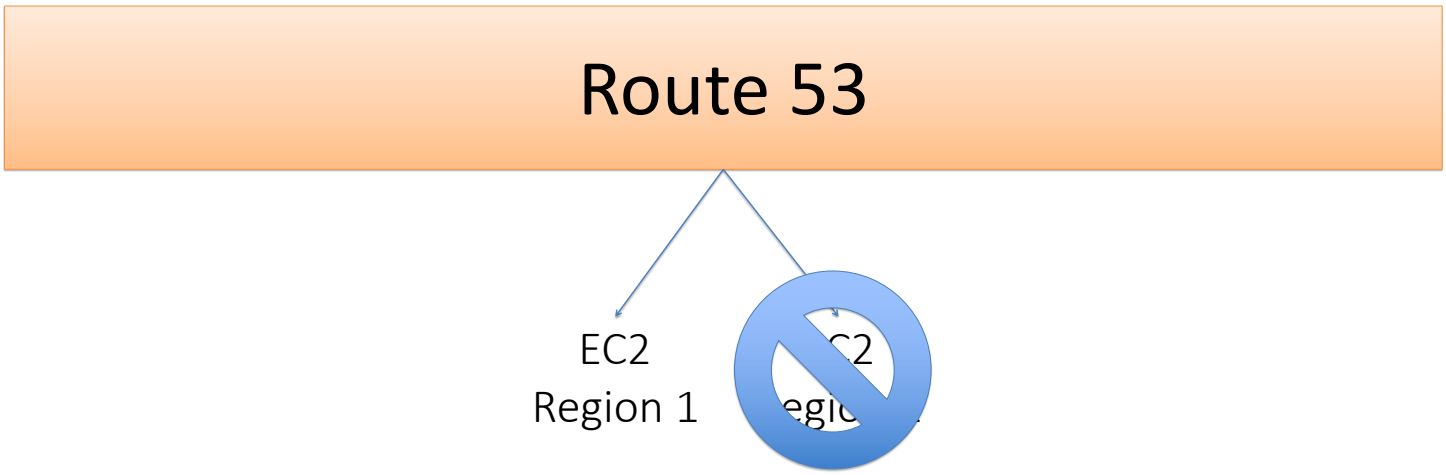
Route 53

www.trainertests.com

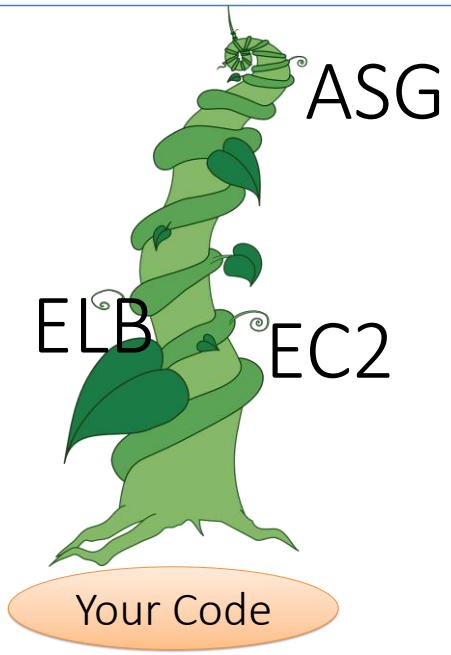
162.241.225.36

On-Premise Web Server EC2

Route 53 Health Check

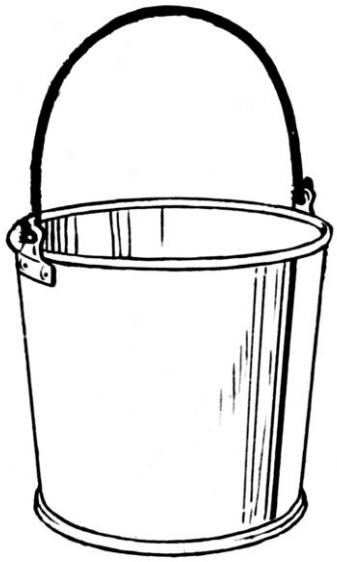


Elastic Beanstalk



Lambda

S3 (Simple Storage Service)



Glacier

EFS (Elastic File System)

- Like NAS in the cloud
- EC2 instances can mount EFS storage
- EFS provides a common data source for multiple instances
- Physical servers can mount EFS

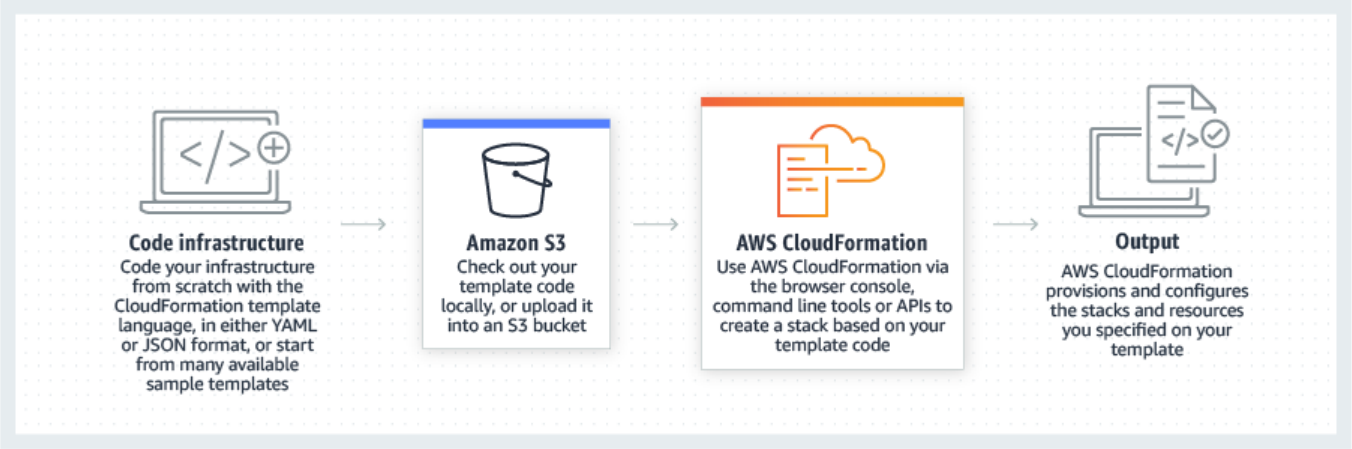
SnowBall

- Secure Appliance
- Migrate mass data to AWS
- Request from AWS Console

Identity and Access Mgt. (IAM)

CloudWatch

CloudFormation



CloudTrail

- Logging for API calls
- View identity, time, source IP, request parameters, and response
- Useful for security analysis, change tracking, and compliance auditing
- Multiple solutions are supported

For more details see my full AWS Architect Associate course:
https://www.udemy.com/course/ultimateaws/?referralCode=7ED214B795C444141361

This study guide demonstrates the lessons from *Introduction to S3 – Unlimited Object Storage in S3.*
My full AWS Architect Associate course can be found here:
https://www.udemy.com/course/ultimateaws/?referralCode=7ED214B795C444141361

Study Guide: S3 Basics

This lesson provides a comprehensive overview of fundamental concepts related to Amazon S3 (Simple Storage Service), a core component of AWS cloud storage.

1. **Object Storage vs. Block Storage:**
    - Object Storage (S3): S3 is introduced as a classic example of object storage, akin to services like Dropbox. It is ideal for storing files and objects such as photos, HTML files, and videos. Unlike block storage, object storage is not used for installing operating systems or databases.
    - Block Storage: The lesson explains that block storage, unlike S3, is used for booting computers and installing applications. It includes hard drives and LUNs on storage arrays.
2. **S3 as Hosted Object Storage:**
    - S3 Overview: S3 is presented as a hosted object storage solution within the AWS cloud, offering exceptional durability. The AWS documentation reveals that S3 provides 11 nines of durability, making data loss highly unlikely.
    - Use Cases: The lesson discusses various use cases for S3, including storing backups, videos, images, and website content. The instructor emphasizes S3's unlimited storage capacity for files and objects.
3. **S3 Characteristics:**
    - Object Size: Each individual object in S3 can be a maximum size of five terabytes, offering flexibility for handling large files.
    - Unlimited Storage: S3 is described as providing virtually unlimited storage capacity, making it suitable for diverse storage needs.
4. **Creating a Static Website with S3:**
    - Unique Naming: S3 buckets must have unique names, ensuring that each bucket has a unique DNS name on the internet.
    - Internet Accessibility: While S3 buckets are not internet-accessible by default, the lesson details how to make contents accessible by configuring DNS records.

- Use Case Example: A practical use case is explored, demonstrating how to create a static website using S3 by storing HTML files, images, and other necessary objects in an S3 bucket.

5. **Summary and Analogy:**
    - S3 as Dropbox of AWS: The lesson concludes by summarizing S3 as an unlimited storage capacity solution for files, likening it to the functionality of Dropbox within the AWS ecosystem.

Static Website Hosting

Amazon S3 serves as a powerful and cost-effective solution for hosting static websites within the AWS cloud environment. To host a static website using S3, users can follow a straightforward process. First, an S3 bucket needs to be created with a unique name, adhering to the DNS naming conventions. This bucket acts as a container for storing the various components of the static website, including HTML files, images, videos, scripts, and other related objects.

Once the necessary files are uploaded to the S3 bucket, users configure the bucket's permissions and make it accessible over the internet. By default, S3 buckets are not internet-accessible, providing a layer of security. To enable public access, users can set up the appropriate permissions and configure DNS records. The unique DNS name assigned to the S3 bucket ensures that each bucket is distinguishable on the internet, allowing users to create a custom web address for their static website. This process makes the contents of the S3 bucket reachable globally, providing a scalable and reliable infrastructure for hosting static web content.

In summary, Amazon S3 simplifies the hosting of static websites by offering a scalable and durable storage solution. Users can leverage the unique DNS names assigned to S3 buckets, ensuring accessibility over the internet, and configure the necessary permissions for public access. The ability to store and serve web content directly from S3 makes it an efficient choice for hosting static websites with the added benefit of seamless integration with other AWS services for enhanced functionality and performance.

See slides below:

Object vs. Block Storage

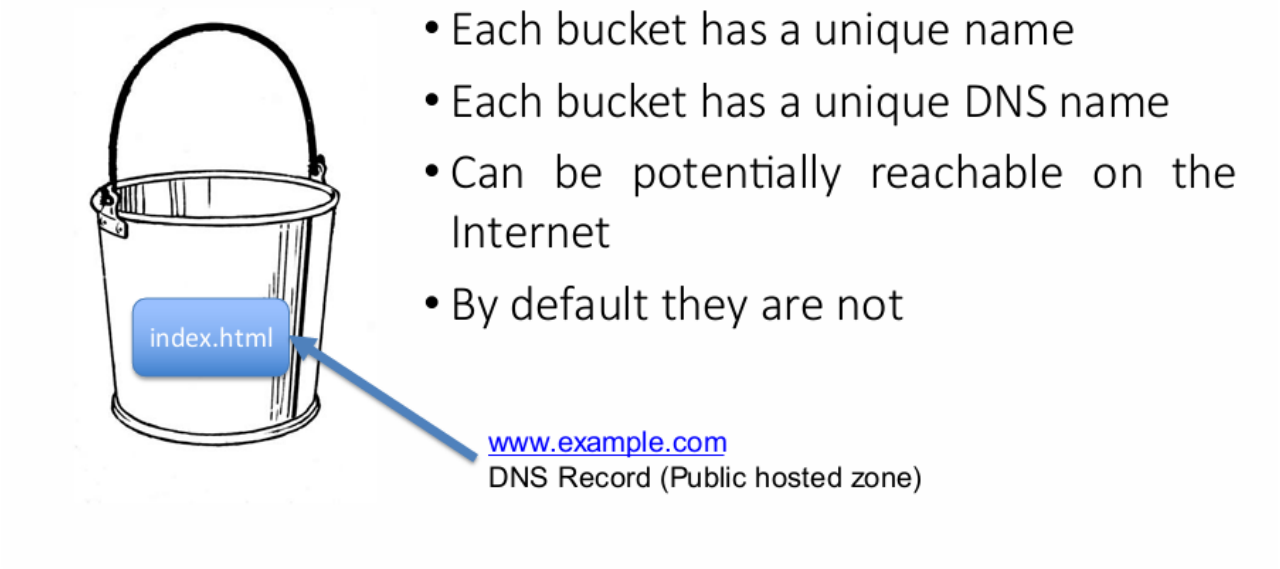
What is S3?

- This is a key part of the AWS Certified Solutions Architect Associate exam.
- Simple file storage
- Designed to deliver 99.999999999% durability
- Many use cases

S3 Object Based Storage

- This is a place to store files
- You can't install your O.S. or databases here
- Maximum file size is 5 TB
- Unlimited Capacity
- Tolerates the failure of two facilities without losing access to objects

Use Case – Static Website



For more details see my full AWS Architect Associate course:
https://www.udemy.com/course/ultimateaws/?referralCode=7ED214B795C444141361

This lab demonstrates the steps from *Demo: Create an S3 Bucket*
My full AWS Architect Associate course can be found here:
https://www.udemy.com/course/ultimateaws/?referralCode=7ED214B795C444141361

Lab Guide: Creating and Managing Amazon S3 Buckets

Amazon S3 (Simple Storage Service) is a fundamental AWS service that provides scalable and reliable storage for various data types and use cases. In this chapter, we will explore how to create an S3 bucket, manage its properties, and understand the concepts of permissions and public access. Additionally, we'll briefly cover uploading objects and making them accessible.

Section 1: Introduction to Amazon S3 Buckets

Understanding Amazon S3

- Amazon S3 is a globally distributed object storage service offered by AWS.
- S3 is commonly used to store and manage a wide range of data, including backups, media files, and static website assets.

Creating an Amazon S3 Bucket

- AWS provides a straightforward process for creating an S3 bucket through the AWS Management Console.
- Bucket names must be unique across all of AWS, as they serve as part of the bucket's URL.

Section 2: Step-by-Step Guide to Creating an S3 Bucket

Launching the AWS Management Console

- Access the AWS Management Console by navigating to the AWS login page and signing in with your credentials.

Accessing the Amazon S3 Console

- After logging in, you can locate the S3 service by either selecting it from the AWS dashboard or by typing "S3" in the search bar and clicking on the Amazon S3 option.

Selecting the Region

- When accessing the S3 console, the region is set to "Global" because S3 buckets are global resources.
- However, when creating a new bucket, you can choose the specific region where you want to place it.

Creating a New Bucket

- Provide a unique name for your S3 bucket, ensuring it adheres to AWS naming conventions.
- Select your preferred region for the bucket.

Section 3: Understanding Amazon S3 Bucket Settings

Default Settings

- By default, most of the bucket settings are set to their standard values.
- These settings include permissions, logging, events, and tags.

Making a Bucket Public

- You can choose to make your S3 bucket public, allowing anyone to access its contents.
- It's essential to be cautious when configuring public access to prevent unauthorized access.

Section 4: Uploading Objects to an S3 Bucket

Uploading Files

- AWS S3 allows you to upload files to your S3 bucket from your local device or an existing storage location.
- Files can be uploaded individually or in bulk.

Configuring Object Properties

- When uploading objects, you can configure settings such as access control and storage class.
- These settings can influence the cost and security of storing objects in the S3 bucket.

Making Objects Public

- You have the option to make uploaded objects accessible to the public or restrict access.

Review and Key Takeaways

- Amazon S3 is a versatile and scalable storage service for various types of data.
- Creating an S3 bucket involves selecting a unique bucket name and choosing the region for its storage.
- Default settings, permissions, and object properties can be configured to match your specific use case and security requirements.
- Uploading objects to an S3 bucket allows you to store, manage, and share data efficiently.
- It's crucial to understand the implications of public access settings to maintain data security and privacy.

This study guide demonstrates the lesson from *AWS vs. other Cloud Providers.*
My full AWS Architect Associate course can be found here:
https://www.udemy.com/course/ultimateaws/?referralCode=7ED214B795C444141361

Comparing AWS, Microsoft Azure, and Google Cloud Study Guide

Choosing the right cloud service provider is a crucial decision when it comes to building and managing your cloud infrastructure. In this lesson, we will compare the three major cloud service providers: Amazon Web Services (AWS), Microsoft Azure, and Google Cloud, to help you understand their respective strengths, market positions, and benefits.

Section 1: Amazon Web Services (AWS)

Introduction to AWS

- AWS is the largest and most established cloud service provider, boasting a significant market share and vast resources.
- AWS offers a wide range of services, making it suitable for various applications and workloads.
- Competitive pricing and options are available, but understanding AWS pricing models is essential for cost management.

Advantages of AWS

- Market leader with a 31% market share as of the second quarter of 2020.
- Broad and mature service offerings with scalability and reliability.
- Cost-effective due to the economy of scale.

Hybrid Cloud Capabilities

- AWS provides hybrid cloud solutions with AWS Outposts and VMware Cloud on AWS.
- Allows the seamless transition of workloads from on-premises environments to the AWS cloud.

Section 2: Microsoft Azure

Introduction to Microsoft Azure

- Microsoft Azure is the closest competitor to AWS, holding a 20% market share as of the second quarter of 2020.
- Provides a full range of cloud services, including Azure, Office 365, and Teams.

Azure's Benefits

- Offers a comprehensive ecosystem where Azure services integrate seamlessly with on-premises Microsoft infrastructure.
- Ideal for hybrid cloud environments as it supports the same operating systems on-premises and in the cloud.

License Cost Savings

- You can leverage existing Microsoft licenses, reducing costs when running Windows-based workloads.

Section 3: Google Cloud

Introduction to Google Cloud

- Google Cloud has made significant strides and holds a 6% market share as of the second quarter of 2020.
- Deeply rooted in open-source technologies and a pioneer in Kubernetes development.

Google Cloud's Strengths

- Well-suited for high-performance computing tasks due to its connection with Kubernetes.
- Aggressive growth and expansion, positioning itself for the future.

Lesson Review and Key Takeaways

- AWS is the leading cloud service provider, known for its vast service offerings and competitive pricing.
- Microsoft Azure is a strong competitor and benefits from seamless integration with Microsoft products and existing licenses.
- Google Cloud, while a smaller player, focuses on high-performance computing and is a significant contributor to open-source technologies.
- Selecting the right cloud service provider depends on your specific needs, workloads, and existing infrastructure.

See slides below:

AWS Competitive Advantages

- Market Share
- Maturity
- Number of Services
- Pricing *
- Outposts and VMC on AWS are great for hybrid

Microsoft

- AWS: 31%, Azure: 20%, Google: 6%
- One vendor for Cloud and On-Premise
- Azure, Office 365, and Teams
- Azure stack is ideal for hybrid
- Cost benefits when running Windows O.S.

Google

- Deep open-source roots
- Central to development of Kubernetes
- Anthos is ideal for hybrid Kubernetes deployments

This study guide demonstrates the lesson from *Understand AWS Regions and Availability Zones.*
My full AWS Architect Associate course can be found here:
https://www.udemy.com/course/ultimateaws/?referralCode=7ED214B795C444141361

AWS Regions and Availability Zones Study Guide

In this lesson, we will explore the fundamental concepts of AWS regions and availability zones. Understanding these concepts is essential for designing highly available and scalable AWS architectures. Let's dive into the world of AWS's geographical and infrastructure layout.

Section 1: Introduction to AWS Regions

What Are AWS Regions?

- AWS regions are geographic areas where AWS has established data centers.
- Each region is identified by a name, such as North Virginia or California.
- Regions are distributed around the world, allowing customers to choose the most suitable region for their workloads.

Section 2: Key Region Characteristics

Dynamic Nature of Regions

- AWS regions are dynamic and may change over time.
- While existing regions may not go away, AWS continues to add new regions to expand its global presence.

Regional Service Availability

- Not all AWS services are available in every region.
- AWS often rolls out new services initially in one or a few regions before making them accessible in other regions.
- When selecting a region, consider the services offered and their pricing, as it may vary between regions.

Section 3: Introduction to Availability Zones

Understanding Availability Zones (AZs)

- Availability zones are isolated data centers or clusters of data centers within an AWS region.
- Each region is composed of multiple availability zones, typically starting with at least two.
- The primary purpose of availability zones is to enhance redundancy and fault tolerance within a region.

Section 4: Redundancy and High Availability

Redundancy Within Availability Zones

- Availability zones provide redundant infrastructure.
- Each availability zone has separate power grids, networking, and facilities, reducing the risk of simultaneous failures.

Building Highly Available Architectures

- AWS customers can design highly available architectures by distributing resources across multiple availability zones.
- For instance, applications can be load-balanced across EC2 instances in different availability zones, ensuring that an application remains available even if one availability zone experiences issues.

Section 5: Visualizing Regions and Availability Zones

Diagrams of AWS Regions and Availability Zones

- A visual representation of the AWS US East North Virginia region with six availability zones.
- Each availability zone contains one or more data centers with distinct infrastructure and resources.
- This architectural design minimizes the likelihood of widespread failures, improving the resilience of AWS services.

Lesson Review and Key Takeaways

- AWS regions are geographic areas where AWS has data centers, and they are dynamic, with new regions being added over time.
- Not all AWS services are available in all regions, and service pricing can vary between regions.
- Availability zones are isolated data centers within an AWS region, designed to enhance redundancy and fault tolerance.
- Distributing resources across multiple availability zones enables the creation of highly available and resilient AWS architectures.
- AWS's geographic layout and architectural design are critical for ensuring high availability and disaster recovery in the cloud.

See slides below:

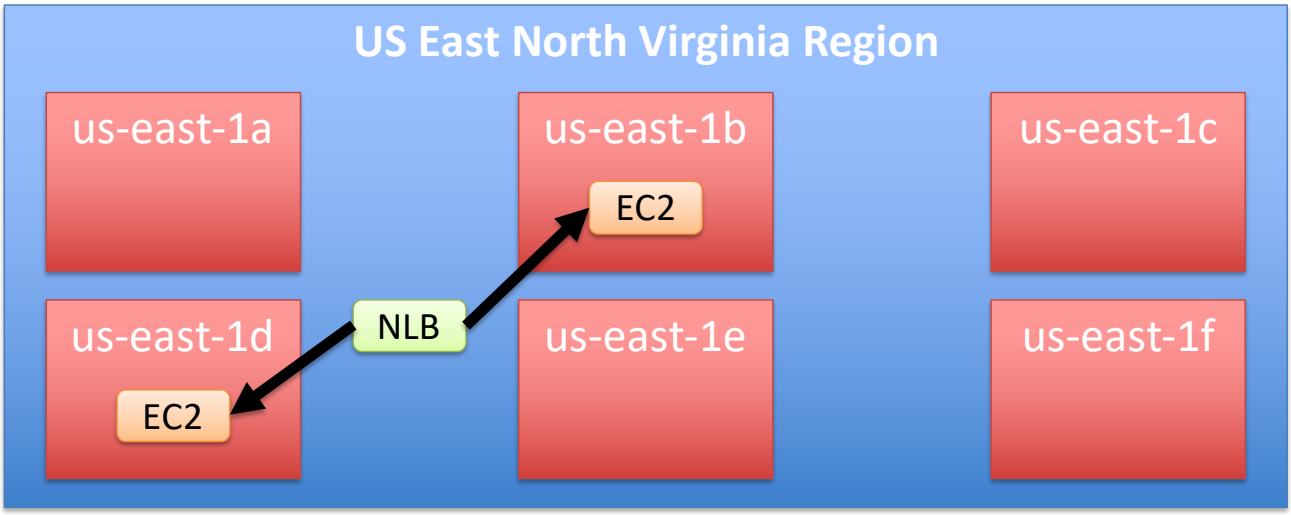
Regions

- Simply a geographic area
- Regions are always being added
- May not provide all services
- Each region has different prices

Availability Zones

- Datacenters or sets of datacenters within a region
- Two or more per region
- Built to provide redundancy within a region

Availability Zones



Edge Locations

- Edge locations are spread across the globe
- Websites are delivered from the nearest edge location
- Improves performance
- Supports S3, EC2, ELB, Route53 and even on-premise
- Lambda@Edge

This study guide demonstrates the lesson from *Getting Started with AWS Certifications.*
My full AWS Architect Associate course can be found here:
https://www.udemy.com/course/ultimateaws/?referralCode=7ED214B795C444141361

Understanding AWS Certifications Study Guide

Introduction

AWS certifications are a gateway to validating your knowledge and expertise in the Amazon Web Services ecosystem. In this lesson, we will focus on two key certifications: the **AWS Certified Cloud Practitioner** and the **AWS Certified Solutions Architect - Associate**. Both certifications cater to different skill levels and objectives, making them essential for individuals seeking to establish or advance their careers in cloud computing.

AWS Certified Cloud Practitioner

The AWS Certified Cloud Practitioner certification is designed for individuals seeking a foundational understanding of AWS. It is ideal for beginners and non-technical professionals who want to grasp the basics of the AWS platform. Below, we'll break down the essential details, requirements, and preparation strategies for this certification.

Exam Overview

- **Difficulty Level**: Entry-level
- **Exam Length**: 90 minutes
- **Number of Questions**: 65
- **Cost**: $100
- **Delivery Options**: Test center or remote (at-home) proctoring

Key Concepts to Master

To pass this exam, a high-level understanding of AWS services, concepts, and practices is required. Key areas include:

1. **Value of AWS Cloud**: Understanding the benefits of cloud computing, such as scalability, elasticity, cost savings, and global reach.
2. **AWS Shared Responsibility Model**: Knowledge of the division of security responsibilities between AWS and the user.
3. **Cloud Economics and Cost Management**: Familiarity with AWS's pricing models, cost management tools, and budgeting practices.
4. **AWS Services Overview**: Basic understanding of core AWS services such as:
    - Compute (e.g., EC2, Lambda)
    - Storage (e.g., S3, EBS)
    - Networking (e.g., VPC, Route 53)
    - Security and Identity (e.g., IAM)

Preparation Tips

- **Study Materials**: Use comprehensive online courses and official AWS documentation.
- **Practice Exams**: Practice tests simulate the real exam experience and familiarize you with question formats.
- **Hands-On Experience**: Even for a beginner's certification, exploring AWS services via the Free Tier is valuable.

Who Should Take This Exam?

This certification is ideal for:

- Individuals new to cloud computing
- Non-technical professionals such as sales, marketing, or management roles who interact with AWS solutions

---

AWS Certified Solutions Architect - Associate

The AWS Certified Solutions Architect - Associate certification is more advanced and caters to individuals seeking a deeper understanding of AWS architecture, design principles, and service integrations.

Exam Overview

- **Difficulty Level**: Intermediate to Advanced
- **Exam Length**: 130 minutes
- **Number of Questions**: 65
- **Cost**: $150
- **Delivery Options**: Test center or remote (at-home) proctoring

Key Concepts to Master

This certification requires significant technical knowledge and hands-on experience. Core topics include:

1. **AWS Basics**:
    - Understanding the AWS Free Tier and account management.
    - Concepts like consolidated billing and AWS Organizations.
2. **Core Services**:
    - Compute: EC2, Auto Scaling, Elastic Load Balancers.
    - Storage: S3, EBS, Glacier.
    - Databases: DynamoDB, RDS, Aurora.
    - Serverless: AWS Lambda and its integrations.
3. **Architectural Best Practices**:
    - Designing secure, scalable, and cost-optimized architectures.
    - Using the AWS Well-Architected Framework.
4. **Networking**:
    - VPC design, subnetting, NAT gateways, and Route 53.
5. **Security**:
    - Identity and Access Management (IAM), Key Management Service (KMS).
6. **Advanced Topics**:
    - High availability and disaster recovery.
    - Monitoring and optimization using tools like CloudWatch and Trusted Advisor.

Preparation Tips

- **Comprehensive Courses**: Look for courses with hands-on labs and real-world scenarios.
- **Practice Exams**: Ensure practice questions mirror the complexity and format of the real exam.
- **Hands-On Projects**: Build sample architectures using AWS services to solidify your understanding.

Who Should Take This Exam?

This certification is ideal for:

- Cloud practitioners looking to advance to technical architect roles.
- Professionals tasked with designing, implementing, and maintaining AWS solutions.
- Individuals preparing for roles like Cloud Architect, Solutions Architect, or DevOps Engineer.

Comparison of Cloud Practitioner vs. Solutions Architect - Associate

Feature	Cloud Practitioner	Solutions Architect - Associate
Difficulty	Beginner	Intermediate to Advanced
Exam Length	90 minutes	130 minutes
Cost	$100	$150
Focus	Broad overview of AWS services	In-depth architecture and design
Ideal Candidate	Non-technical roles, beginners	Technical professionals

---

Conclusion

AWS certifications provide a structured path to validate your skills and open doors to exciting cloud computing opportunities. The **Cloud Practitioner** certification is an excellent starting point for beginners, while the

Solutions Architect - Associate certification is perfect for those ready to tackle more technical and in-depth cloud challenges. With the right preparation, including study materials, hands-on practice, and practice exams, achieving these certifications is entirely within reach.

For more details see my full AWS Architect Associate course:
https://www.udemy.com/course/ultimateaws/?referralCode=7ED214B795C444141361

This study guide demonstrates the lesson from *AWS Support Plans.*
My full AWS Architect Associate course can be found here:
https://www.udemy.com/course/ultimateaws/?referralCode=7ED214B795C444141361

AWS Support Plans Study Guide

In this lesson, we will delve into the various AWS support plans available, ranging from basic developer support to comprehensive enterprise support. Understanding these support options is crucial for AWS users to choose the appropriate level of assistance based on their business needs and budget.

Section 1: Introduction to AWS Support Plans

Why AWS Support Plans Matter

- AWS offers a range of support plans tailored to meet the diverse needs of its customers.
- The right support plan can provide timely assistance, access to AWS experts, and additional benefits to optimize AWS usage.

Section 2: Developer Support Plan

Overview of Developer Support

- Developer support is the most cost-effective support plan, priced at $29 per month or 3% of monthly usage.
- With this plan, customers can open cases, receive general guidance within 24 hours, and get responses within 12 hours if their system is impaired.
- Architectural guidance and other advanced support services are limited.

Section 3: Business Support Plan

Key Features of Business Support

- The Business Support Plan is recommended for users with production workloads.
- It provides faster support response times and access to AWS Managed Services for an additional fee.
- Pricing starts at $100 per month or 10% of monthly AWS usage, with the percentage decreasing as spending increases.

Section 4: Enterprise On-Ramp Support Plan

Understanding Enterprise On-Ramp Support

- The Enterprise On-Ramp Support Plan offers a faster response to system down situations, with response times of less than 30 minutes.
- It includes consultative architectural guidance and provides access to AWS experts to assist in designing the AWS environment.
- Minimum spending requirements are $5,500 per month or 10% of monthly AWS usage.

Section 5: Enterprise Support Plan

Comprehensive Enterprise Support

- The Enterprise Support Plan is the most comprehensive and expensive option.
- It includes all benefits from Business Support and Enterprise On-Ramp Support.
- Response times are as short as 15 minutes, and it provides access to AWS incident detection and response.
- Customers receive additional access to AWS Managed Services and are assigned a dedicated technical account manager.

This study guide demonstrates the lesson from *Understand AWS Managed vs. Unmanaged Services*
My full AWS Architect Associate course can be found here:
https://www.udemy.com/course/ultimateaws/?referralCode=7ED214B795C444141361

Managed vs. Unmanaged Services in AWS Study Guide

In this lesson, we will explore the fundamental concepts of managed and unmanaged services in the context of AWS. Understanding the key differences between these two service types is essential for making informed decisions when choosing the right AWS services to meet your specific needs.

Section 1: Introduction to Managed and Unmanaged Services

Defining Managed and Unmanaged Services

- Managed services are AWS offerings in which AWS takes care of most operational and maintenance tasks, allowing users to focus on their applications and data.
- Unmanaged services, on the other hand, require users to take full responsibility for configuring, maintaining, and securing the service themselves.

Section 2: Unmanaged Services: The Analogy of Cooking Your Own Meal

The Analogy of Cooking Your Own Meal

- Unmanaged services are akin to preparing your own meal from scratch.
- Users have complete control over the process, from ingredient selection to cooking to serving and cleanup.
- EC2 is a prime example of an unmanaged service where users have 100% control over the virtual server, operating system, and security configurations.

Section 3: Managed Services: The Analogy of Dining in a Restaurant

The Analogy of Dining in a Restaurant

- Managed services can be compared to dining in a restaurant, where everything is prepared and served by experts.
- Users relinquish some control but enjoy the convenience of not having to manage the intricate details.
- AWS's S3 is an illustration of a managed service where AWS handles maintenance, security, and scaling, allowing users to focus on their data and applications.

Section 4: Implications of Managed and Unmanaged Services

Implications for Users

- Managed services are ideal for users who want to delegate operational tasks, reduce maintenance work, and concentrate on their core activities.
- Unmanaged services are suitable for users who require full control over configurations, security, and scalability but need to invest time and effort in management.

Lesson Review and Key Takeaways

- Managed services in AWS are akin to dining in a restaurant, where experts handle all service aspects, offering convenience and time-saving benefits.
- Unmanaged services, represented by EC2, are similar to cooking your own meal, providing total control but requiring more effort.
- Users should choose between managed and unmanaged services based on their specific requirements, considering factors like control, maintenance, and scaling.

See slides below:

Unmanaged Service Analogy

Managed Service Analogy

For more details see my full AWS Architect Associate course:
https://www.udemy.com/course/ultimateaws/?referralCode=7ED214B795C444141361

This study guide demonstrates the lesson from *Introduction to EC2.*
My full AWS Architect Associate course can be found here:
https://www.udemy.com/course/ultimateaws/?referralCode=7ED214B795C444141361

Introduction to Amazon EC2 Study Guide

Amazon Elastic Compute Cloud (EC2) is a core service within the Amazon Web Services (AWS) ecosystem, offering scalable virtual servers in the cloud. This lesson will guide you through the fundamental concepts of EC2, including its architecture, the process of creating virtual servers, and the essential considerations for managing these resources effectively.

What is EC2?

Amazon EC2 provides **Infrastructure as a Service (IaaS)**, enabling users to deploy and manage virtual servers in the AWS Cloud. These servers, called **EC2 instances**, run on **physical hosts** managed by AWS in their global data centers. Through virtualization technology, AWS allows multiple virtual servers to share the same physical infrastructure.

Key Characteristics of EC2:

1. **Virtual Servers**: Each EC2 instance functions like a standalone computer, running its own operating system and applications.
2. **User Responsibility**: While AWS provides the physical infrastructure and virtualization layer, users are responsible for tasks such as:
    - Managing the operating system (e.g., updates, patches).
    - Installing and maintaining software.
    - Configuring security settings.

EC2 Instances and Hypervisors

EC2 instances operate on physical servers equipped with **hypervisors**. A hypervisor is software that enables multiple virtual machines to share the physical resources of a single server.

How It Works:

- A single physical host in an AWS data center can run multiple EC2 instances simultaneously.
- The hypervisor manages the allocation of CPU, memory, and storage resources, ensuring isolation between instances.
- Instances from different customers may run on the same physical host, but AWS implements strong isolation to maintain security.

Storage for EC2: Elastic Block Store (EBS)

Every EC2 instance requires storage to function. AWS uses **Elastic Block Store (EBS)** to provide persistent block storage for instances.

Features of EBS:

1. **Virtual Disks**: EBS volumes act as virtual hard drives for EC2 instances, storing operating systems, applications, and data.
2. **Persistent Storage**: Data on an EBS volume persists independently of the instance, allowing recovery in case of instance termination.
3. **Flexible Sizing**: Users can customize the size, performance, and type of EBS volume based on application needs.

Creating an EC2 Instance

Step 1: Select an Amazon Machine Image (AMI)

An **AMI** serves as a template for your EC2 instance, defining the operating system and pre-installed software. AWS offers:

- **Pre-configured AMIs**: Include popular operating systems like Linux and Windows.
- **Custom AMIs**: Users can create their own AMIs tailored to specific requirements.

Step 2: Choose an Instance Type

The instance type determines the hardware configuration of the EC2 instance, including:

- **CPU**: Number and speed of virtual processors.
- **Memory**: Amount of RAM.
- **Network Performance**: Bandwidth and throughput.
- **Storage Performance**: Read/write speeds for attached EBS volumes.

Step 3: Configure the Instance

After selecting an AMI and instance type, you configure additional settings such as:

- Security groups (firewall rules).

- Key pairs for SSH access.
- Placement within a specific **Availability Zone** (a distinct data center within a region).

Availability Zones and Instance Placement

Each EC2 instance is deployed within a single **Availability Zone (AZ)**. An AZ is a physically separate location within an AWS region, designed for high availability.

Key Points:

- **Instance Locality**: Instances are tied to the AZ where they were created and cannot be moved while running.
- **Workaround for Movement**: To relocate an instance, create an image of the existing instance and launch it in a new AZ.

Cost and Pricing

The cost of running an EC2 instance varies based on:

- **Instance Type**: More powerful instances with higher CPU and memory cost more.
- **Billing Model**: AWS offers several pricing options, including:
  - **On-Demand**: Pay by the hour or second without long-term commitments.
  - **Reserved Instances**: Save costs with upfront payment for long-term usage.
  - **Spot Instances**: Bid for unused capacity at lower prices.

Security and Multi-Tenancy

AWS employs robust security mechanisms to ensure isolation between instances, even when they share the same physical host:

- **Network Isolation**: Virtual networks ensure instances cannot communicate unless explicitly configured.
- **Instance Isolation**: The hypervisor prevents unauthorized access between instances.

Diagram of EC2 Architecture

A typical EC2 setup includes:

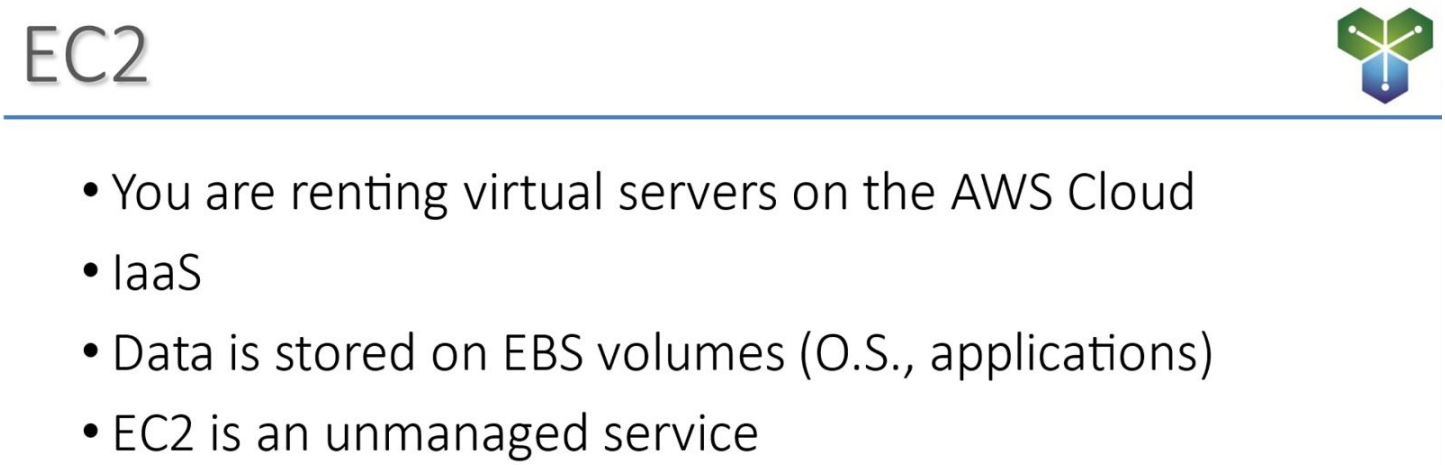
- **Physical Host**: Runs multiple EC2 instances.
- **Hypervisor**: Manages resource allocation and isolation.
- **EC2 Instances**: Virtual servers for customers, running operating systems and applications.

Summary

1. EC2 provides IaaS, enabling users to deploy virtual servers in the AWS Cloud.
2. Users are responsible for managing their EC2 instances, including the operating system and software.
3. Elastic Block Store (EBS) offers persistent block storage for EC2 instances.
4. Instances are tied to specific Availability Zones and cannot be moved directly.
5. Pricing varies by instance type, size, and billing model.
6. AWS ensures strong security and isolation between instances.

By understanding these foundational concepts, you can confidently deploy and manage EC2 instances to meet your application's needs.

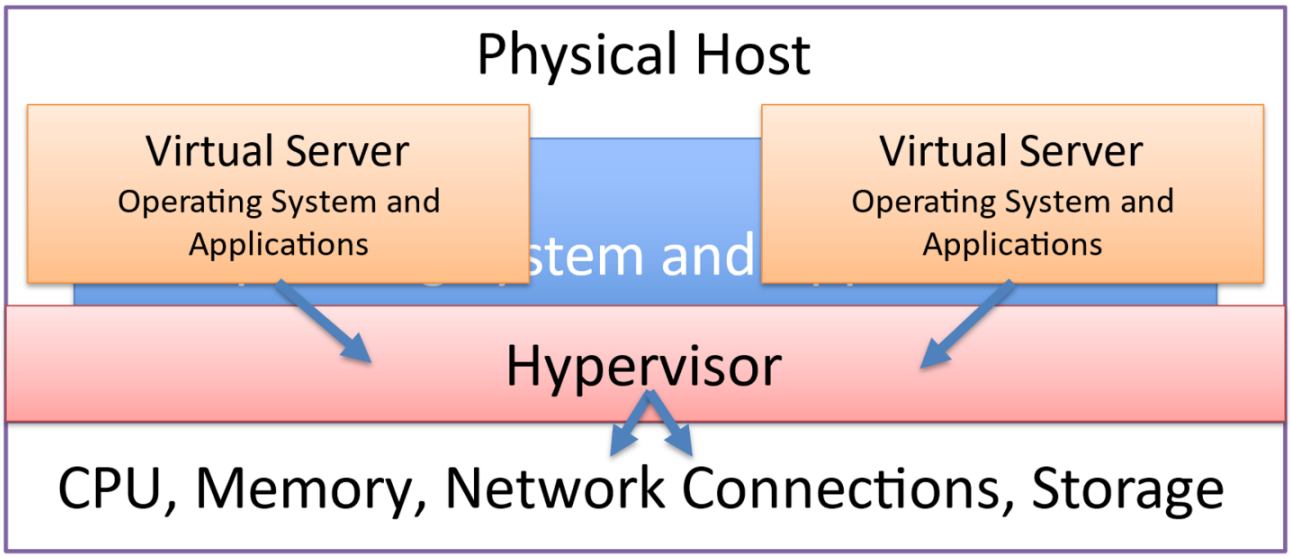
See slides below:



EC2 Instances

- You choose Operating System (AMI)
- You choose resources (CPU, memory, network)
- Different prices for different instance types
- Local to an AZ

Hypervisors and Virtual Machines



For more details see my full AWS Architect Associate course:
https://www.udemy.com/course/ultimateaws/?referralCode=7ED214B795C444141361

This lab demonstrates the steps from *Lab Guide – Create an EC2 Instance.*
My full AWS Architect Associate course can be found here:
https://www.udemy.com/course/ultimateaws/?referralCode=7ED214B795C444141361

Lab Guide: Creating an EC2 Instance in AWS

Objective

This lab will guide you through the step-by-step process of **creating an EC2 instance** in AWS. You will learn key concepts such as **Amazon Machine Images (AMI), instance types, key pairs, security groups, and termination protection** while launching your own virtual server in AWS.

Prerequisites

- An **AWS account** with access to the **AWS Management Console**.
- Basic knowledge of **EC2** and **networking concepts** is helpful but not required.
- Familiarity with AWS **free tier** usage if applicable.

Step 1: Navigate to the EC2 Dashboard

1. **Log into the AWS Management Console**.
2. **Select your AWS Region** in the top-right corner (e.g., **Ohio (us-east-2)**).
3. **Search for "EC2"** in the AWS console's search bar.
4. Click on **EC2** to enter the **EC2 Dashboard**.

Step 2: Launch an EC2 Instance

1. On the **EC2 Dashboard**, scroll down and click on **Launch Instance**.
2. This process will create a new virtual server.

Step 3: Configure the Instance Settings

3.1 Naming Your Instance

- Enter a **name** for your instance, e.g., `RickCrisciWeb1`.

3.2 Choose an Amazon Machine Image (AMI)

- An **Amazon Machine Image (AMI)** is a pre-configured OS image.
- Select the **default Amazon Linux AMI**.

  Key Concept:

  - The AMI determines the **operating system** and **pre-installed software**.
  - AWS offers multiple AMI options, including **Windows, Ubuntu, Red Hat**, and more.

Step 4: Choose an Instance Type

- Choose **t2.micro** (which is eligible for AWS Free Tier).
- **Instance Type Explanation**:
  - **t2.micro** has **1 vCPU** and **1 GB RAM**, sufficient for basic tasks.
  - AWS offers **many instance types** with varying CPU, memory, and network capabilities.

Step 5: Configure Key Pair for Secure Access

1. Click **"Create a new key pair"**.
2. Name it **RickCrisciWeb1** (or any name of your choice).
3. **Download the key pair file** (`.pem`) and **store it securely**.
4. **Key Pair Explanation**:
   - AWS uses **public-key cryptography** to secure SSH access.
   - The `.pem` file is required for **SSH authentication**.

Step 6: Configure Network Settings

1. Expand the **Network Settings** section.
2. Click **Edit** to manually select a **specific subnet**.
3. Choose a **public subnet** in **Availability Zone us-east-2A**.
4. **VPC & Subnet Explanation**:
   - Every EC2 instance **must be inside a VPC**.
   - A **subnet** determines **which Availability Zone** the instance runs in.
   - A **public subnet** allows the instance to have a **public IP address**.

Step 7: Configure Security Group (Firewall Rules)

1. **Create a new security group**:
    - Name it **RickCrisciWebServers**.
    - Use the same name for the **description**.
2. **Add inbound rules**:
    - **Allow HTTP (port 80) from anywhere**.
    - **Allow HTTPS (port 443) from anywhere**.
    - **Allow SSH (port 22) from anywhere** (for management access).

Security Group Explanation:

- Security groups act as **firewalls**, controlling inbound/outbound traffic.
- HTTP and HTTPS are needed for **web traffic**.
- SSH allows **secure remote access** to the instance.

Step 8: Configure Storage (EBS Volume)

- Keep the default **GP2 General Purpose SSD**.
- **Elastic Block Store (EBS) Explanation**:
    - **EBS** is **persistent block storage** for EC2.
    - The default **8GB SSD** is sufficient for most use cases.

Step 9: Enable Termination Protection

1. Expand **Advanced Details**.
2. Enable **Termination Protection**.

Termination Protection Explanation:

- Prevents accidental deletion of the instance.
- Adds an extra step before **permanently terminating** the EC2 instance.

Step 10: Launch the Instance

1. Review your settings in the summary section.
2. Click **"Launch Instance"**.
3. AWS will begin provisioning your virtual machine.

Step 11: Viewing and Managing Your Instance

1. Click on the **instance ID** to open its details.
2. Observe its **initial status as "Pending"**.
3. After a few moments, the status will change to **"Running"**.

Instance Details

- Check the **public and private IP addresses**.
- Verify the **instance type, AMI, and VPC** settings.

Step 12: Monitor the Instance

1. Click the **Status Checks** tab.
2. Observe the **System Status Check** and **Instance Status Check**.
3. Wait until both checks pass.

Status Checks Explanation:

- **System Status Check** ensures AWS infrastructure is working correctly.
- **Instance Status Check** verifies the OS is running properly.

Summary

In this lab, you successfully:

- **Created an EC2 instance** using the AWS console.
- **Configured an AMI, instance type, key pair, security group, and storage**.
- **Enabled termination protection** to prevent accidental deletion.
- **Verified the instance status** and monitoring details.

This EC2 instance can now be used for **web hosting, database management, or other applications**.

Next Steps

- **Connect to the EC2 instance via SSH** (if using Linux).
- **Deploy a web application** on the instance.
- **Explore additional configurations** like **Elastic Load Balancers (ELB)** and **Auto Scaling**.

This lab demonstrates the steps from *Demo: Back up an EC2 Instance Using Snapshots*
My full AWS Architect Associate course can be found here:
https://www.udemy.com/course/ultimateaws/?referralCode=7ED214B795C444141361

Lab Guide: Backing Up an EC2 Instance Using Snapshots

This lab guide walks you through the steps to back up an Amazon EC2 instance using snapshots. You will learn key concepts such as EBS volumes, snapshots, and creating AMIs based on snapshots. Follow the steps carefully to gain hands-on experience.

Lab Objectives

1. Learn how to identify and back up an EC2 instance's EBS volume.
2. Understand how snapshots store data and their use cases.
3. Create a new volume or AMI from a snapshot.
4. Launch a new EC2 instance using a snapshot or AMI.

Step 1: Access the AWS Management Console

1. Log in to your AWS account and navigate to the **EC2 Dashboard**.
2. Under the **Instances** tab, identify the EC2 instance you wish to back up.

Step 2: Locate and Rename the EBS Volume

1. Click on the EC2 instance to open its details.
2. Select the **Storage** tab to view the attached EBS volume(s).

3. Click the **Volume ID** to open the volume's details.
4. Rename the volume (optional but recommended for clarity):
    - Click the **Edit name** button.
    - Enter a meaningful name (e.g., "Kali-VM-Volume").
    - Click **Save**.

Step 3: Take a Snapshot of the EBS Volume

1. In the EBS volume details, click **Actions → Create Snapshot**.
2. Provide a description for the snapshot (e.g., "Backup of Kali VM").
3. Click **Create Snapshot** to start the process.
4. Navigate to the **Snapshots** section in the left-hand menu to monitor progress:
    - The snapshot will initially show as **Pending** while AWS copies the volume's data.
    - Once finished, it will be marked **Completed**.

Step 4: Verify the Snapshot

1. Confirm that the snapshot appears in the **Snapshots** section.
2. Review the details of the snapshot, such as size, volume ID, and creation time.

Step 5: Create a New Volume from the Snapshot

1. Select the snapshot in the **Snapshots** section.
2. Click **Actions → Create Volume**.
3. Configure the volume:
    - Volume Type: General Purpose SSD (gp2).
    - Size: Should match or exceed the original volume size.
    - Availability Zone: Ensure it matches the zone of the intended EC2 instance (e.g., `us-east-2a`).
4. Click **Create Volume** to generate the new volume.
5. Verify the new volume under the **Volumes** section.

Step 6: Create a New AMI from the Snapshot

1. Select the snapshot in the **Snapshots** section.
2. Click **Actions → Create Image**.
3. Provide details for the new AMI:
    - Name: (e.g., "KaliFromBackup").
    - Description: Same as the snapshot or customized.
4. Click **Create Image**.
5. Navigate to the **AMIs** section in the left-hand menu to monitor progress.
6. Once the AMI is marked **Available**, it is ready for use.

Step 7: Launch a New EC2 Instance Using the AMI

1. In the **AMIs** section, select the newly created AMI.
2. Click **Launch Instance**.
3. Review the instance:
    - Choose an instance type (e.g., t2.micro).
    - Select the appropriate key pair for SSH access.
    - Configure storage:
        - The root volume will be pre-configured based on the snapshot.
        - Adjust the storage size if necessary.
4. Launch the instance.

Step 8: Verify the New Instance

1. Go to the **Instances** section and confirm the new instance is running.
2. SSH into the instance to verify its functionality.
3. Confirm that the data and applications from the original instance are intact.

Key Concepts Recap

1. **EBS Volumes**: Act as virtual disks for EC2 instances, storing OS, applications, and data.
2. **Snapshots**: Point-in-time backups of EBS volumes stored in Amazon S3.
3. **Creating Volumes**: Snapshots can be used to create new volumes for recovery or expansion.
4. **AMIs**: Snapshots can also be converted into Amazon Machine Images, enabling rapid deployment of identical EC2 instances.

Lab Summary

In this lab, you backed up an EC2 instance by creating a snapshot of its EBS volume. You learned how to use snapshots to create new volumes and AMIs, which can be leveraged for disaster recovery or scaling operations. By completing this exercise, you now have the skills to secure your EC2 data and deploy backups efficiently.

This study guide demonstrates the lesson from *Block Storage (EBS and Instance Store).*
My full AWS Architect Associate course can be found here:
https://www.udemy.com/course/ultimateaws/?referralCode=7ED214B795C444141361

Understanding and Configuring Amazon EBS Volumes Study Guide

In this lesson, we will explore Amazon Elastic Block Store (EBS), the primary block storage option used with Amazon EC2 instances. EBS volumes play a crucial role in storing operating systems, applications, and data for virtual machines in the AWS cloud. This lesson will also cover the various EBS volume types, their use cases, performance characteristics, and how to configure them effectively.

What is an EBS Volume?

Amazon Elastic Block Store (EBS) is a scalable, high-performance block storage service designed for use with EC2 instances. EBS volumes provide persistent storage, meaning they retain data even after an EC2 instance is stopped or terminated.

Key Characteristics of EBS:

1. **Block Storage**: Functions like a physical disk (e.g., SSD or HDD) that your operating system interacts with.
2. **Direct Access**: EBS volumes are attached to EC2 instances, allowing fast and low-latency access to the data.
3. **Persistence**: Unlike ephemeral storage, EBS volumes are durable and retain data independently of the EC2 instance lifecycle.

Configuring Storage for an EC2 Instance

When launching an EC2 instance, the **Configure Storage** section allows you to set up the EBS volume for that instance. You can specify:

- **Size**: Define the storage capacity required for your workloads.
- **Volume Type**: Choose from different storage performance levels (e.g., General Purpose SSD or Provisioned IOPS).

EBS Volume Types

EBS offers several volume types to meet varying performance and cost needs. Below are the key options:

1. General Purpose SSD (gp3 and gp2)

- **Purpose**: Ideal for a wide range of workloads, such as boot volumes, medium-sized databases, and development/test environments.
- **Characteristics**:
  - **gp3**: Latest generation offering better performance at a lower cost.
    - Price: $0.08 per GB per month.
    - Performance: 3,000 IOPS baseline and scalable up to 16,000 IOPS.
  - **gp2**: Older generation, more expensive ($0.10 per GB per month).
    - Performance scales with size, up to 16,000 IOPS.
- **Recommendation**: Use **gp3** for most general-purpose workloads.

2. Provisioned IOPS SSD (io1 and io2)

- **Purpose**: Designed for latency-sensitive and high-performance applications like large databases or analytics workloads.
- **Characteristics**:
  - Provision predictable IOPS levels.
  - High throughput, but significantly more expensive than gp3 or gp2.
- **When to Use**: Select this option for critical workloads requiring consistent high performance.

3. Hard Disk Drive (HDD) Options

- **Purpose**: Best for infrequent access or massive datasets where cost is a primary concern.
- **Types**:
  - **Cold HDD (sc1)**: For data accessed rarely, such as backups or archives.
  - **Throughput Optimized HDD (st1)**: For streaming workloads like big data or log processing.
- **Limitations**:
  - HDD volumes cannot be used as root volumes in most cases.
  - Lower performance compared to SSD options.
- **Recommendation**: Use HDD only for large, cold data that doesn’t require high IOPS.

Comparing EBS with Other AWS Storage Options

While EBS is the default choice for EC2 instances, it is not always the best option for every workload. Consider alternatives:

1. **Amazon S3**: For object storage of large datasets, backups, or archives.
2. **Amazon EFS**: For shared file storage across multiple instances, suitable for applications requiring POSIX compliance.

Key Considerations When Configuring EBS

When selecting and configuring an EBS volume, consider the following factors:

1. **Performance Needs**:
   - Use gp3 for general-purpose workloads.
   - Choose io1/io2 for high-performance applications.
2. **Cost Optimization**:
   - Avoid over-provisioning storage to save costs.
   - Opt for gp3 instead of gp2 to reduce expenses.
3. **Volume Size**:
   - Ensure the volume is large enough for the operating system, applications, and expected data growth.
4. **Durability**:
   - Use snapshots to back up EBS volumes regularly for disaster recovery.

Practical Example: Configuring an EBS Volume

Step 1: Launch an EC2 Instance

1. Log into the AWS Console and navigate to the EC2 dashboard.
2. Click **Launch Instance** and choose an appropriate AMI (e.g., Amazon Linux 2).
3. Name your instance (e.g., "EBS Demo").

Step 2: Configure Storage

1. Scroll to the **Configure Storage** section.
2. Set the **Size** of the EBS volume (e.g., 40 GB).
3. Select a **Volume Type**:
   - Choose **gp3** for general workloads.
   - Select **io2** for high-performance needs.

Step 3: Launch the Instance

1. Complete the rest of the configuration (e.g., network, key pairs).
2. Click **Launch** to deploy your EC2 instance.

Monitoring and Managing EBS Volumes

After deployment, monitor and manage EBS volumes using the following tools:

1. **Amazon CloudWatch**:
    - Track metrics like IOPS, throughput, and latency.
2. **Snapshots**:
    - Regularly back up volumes to Amazon S3 for recovery.
3. **Elastic Volumes**:
    - Resize, change volume types, or adjust performance settings without downtime.

Summary

1. **EBS Volumes**: Essential block storage for EC2 instances, used to store operating systems, applications, and data.
2. **Volume Types**: General Purpose SSD (gp3, gp2), Provisioned IOPS SSD (io1, io2), and HDD options (sc1, st1).
3. **Cost and Performance**: Optimize by selecting the right volume type based on workload requirements.
4. **Best Practices**: Use snapshots for backup and elastic volumes for scaling without downtime.

This study guide demonstrates the lesson from *Introduction to AWS VPCs – Networking in the Cloud.*
My full AWS Architect Associate course can be found here:
https://www.udemy.com/course/ultimateaws/?referralCode=7ED214B795C444141361

AWS Virtual Private Cloud (VPC) Study Guide

Overview of VPC

An AWS Virtual Private Cloud (VPC) is a virtual network dedicated to your AWS account, providing a logically isolated section of the AWS cloud where you can launch AWS resources. By default, a VPC is isolated from other virtual networks and the internet, ensuring no traffic can enter or exit without explicit configuration.

Key Components and Concepts

- **Region and Availability Zones**: A VPC is created within a specific AWS region and can span multiple Availability Zones (AZs), allowing for the creation of highly available and resilient architectures.
- **CIDR Range**: When creating a VPC, you must define a Classless Inter-Domain Routing (CIDR) block, determining the IP address range available within the VPC. The largest range you can choose is /16, offering a substantial number of private IP addresses.
- **Subnets**: Within a VPC, you can create subnets, which are segments of the VPC's IP address range located within a specific Availability Zone. Subnets can be public (accessible from the internet) or private (isolated from the internet), depending on their intended use, such as hosting web servers or databases.
- **Internet Gateway (IGW)**: An internet gateway is a VPC component that allows communication between resources in your VPC and the internet. It's necessary for enabling internet access in public subnets.
- **VPN and AWS Direct Connect**: For secure connections between your on-premises data center and your VPC, AWS provides VPN connections and AWS Direct Connect, offering encrypted internet-based connections and dedicated physical connections, respectively.
- **Network ACLs and Security Groups**: Network Access Control Lists (ACLs) and Security Groups act as firewalls for controlling traffic into and out of subnets and EC2 instances, respectively. While Network ACLs apply at the subnet level affecting all resources within, Security Groups apply to individual instances, offering granular control.

Security and Network Traffic Control

- **Network ACLs**: Are associated with subnets, controlling all inbound and outbound traffic passing through the subnet. They offer another layer of security, supplementing Security Groups.

- **Security Groups**: Act on individual instances, allowing you to specify permissible traffic for each instance, providing customizable security at a more granular level.

Practical Applications

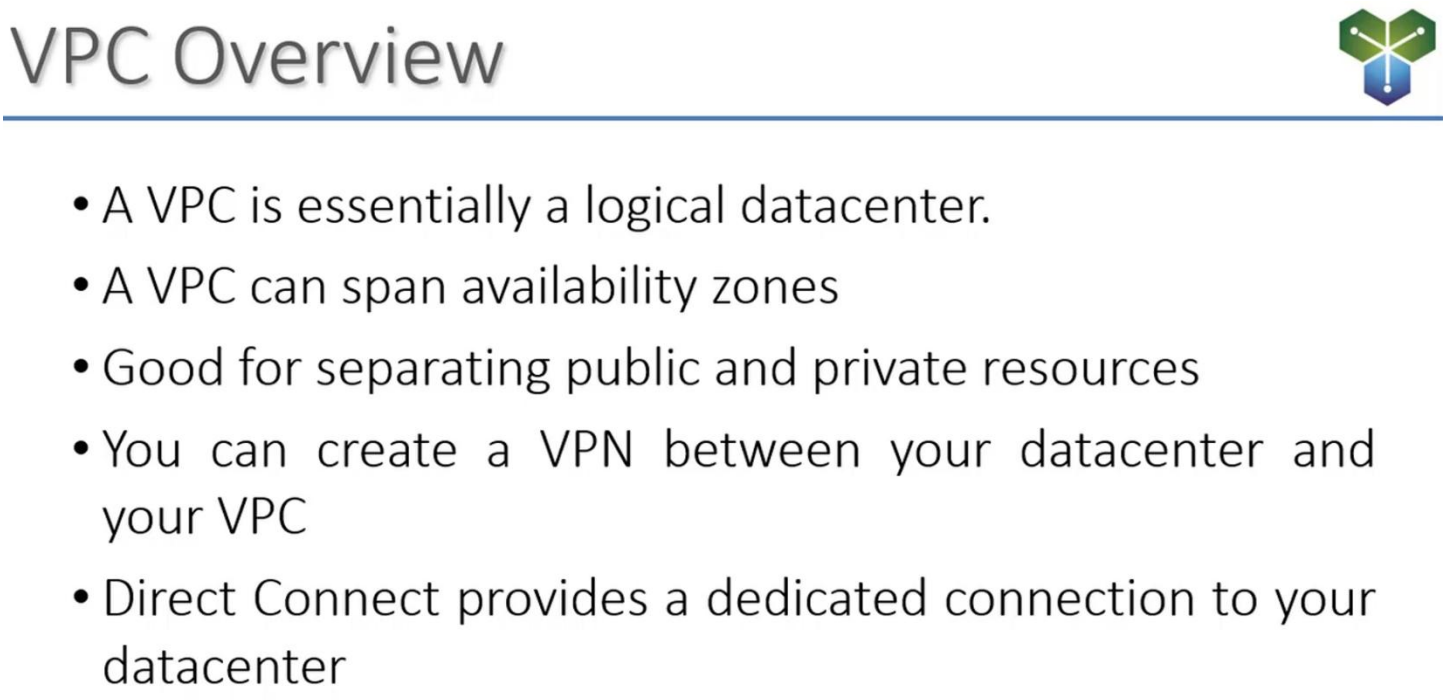
AWS VPC enables you to build a network architecture within AWS that mimics a traditional network you might operate in your own data center but with the benefits of AWS infrastructure. It's instrumental in:

- **Creating Isolated Networks**: For sensitive applications or data, ensuring they are not accessible from the internet.
- **Hosting Public-facing Services**: Like websites, by configuring public subnets and internet gateways.
- **Interconnecting with On-premises Data Centers**: Securely, using VPN or AWS Direct Connect, for hybrid cloud scenarios.

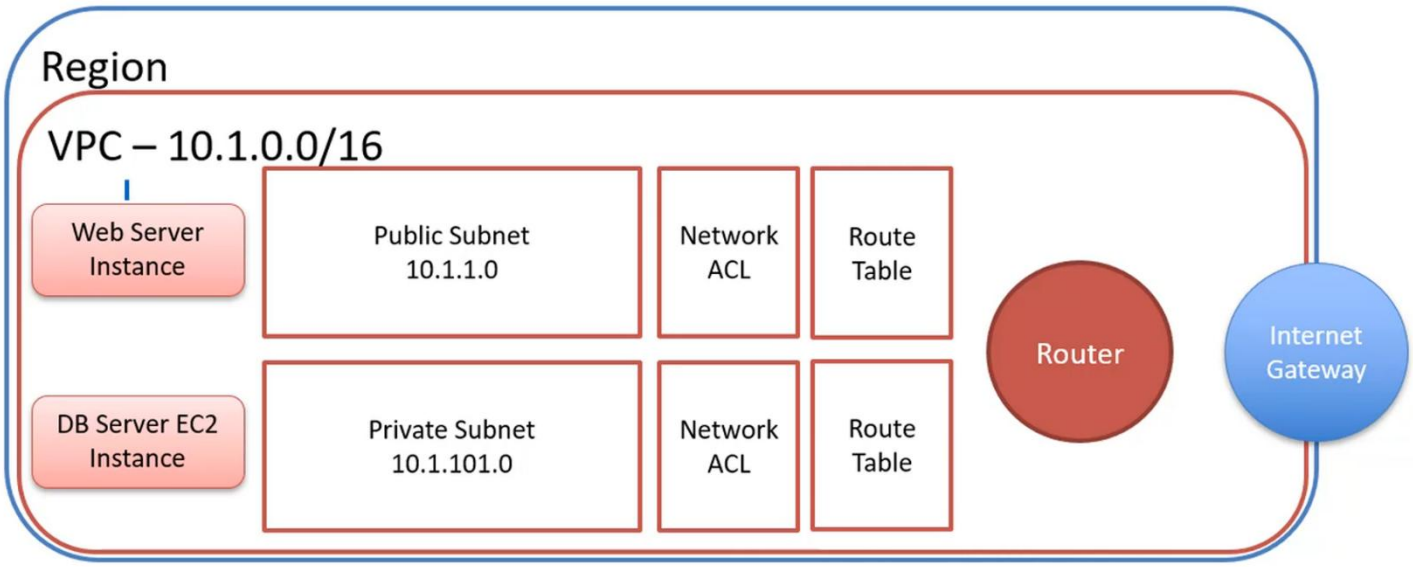
Conclusion

Understanding and effectively utilizing AWS VPC components like subnets, internet gateways, VPNs, Network ACLs, and Security Groups are fundamental for architecting secure, scalable, and highly available environments on AWS. By controlling network traffic at both the subnet and instance levels, AWS VPC provides the tools necessary to design complex networks tailored to an organization's specific needs.

See slides below:



VPC Diagram



For more details see my full AWS Architect Associate course:
https://www.udemy.com/course/ultimateaws/?referralCode=7ED214B795C444141361

This lab demonstrates the steps from *Demo: Create a VPC.*
My full AWS Architect Associate course can be found here:
https://www.udemy.com/course/ultimateaws/?referralCode=7ED214B795C444141361

Lab Guide: Creating a Virtual Private Cloud (VPC) in AWS

This lab guide walks you through the basics of creating a Virtual Private Cloud (VPC) in the AWS Console. It follows the steps outlined in the material and explains key concepts along the way. By the end of this lab, you will understand how to create a secure and functional VPC for hosting resources in AWS.

Lab Objectives

1. Learn about the Default VPC and why it may not be suitable for secure workloads.
2. Create a custom VPC using AWS's "VPC and More" wizard.
3. Understand key VPC components, including CIDR ranges, subnets, NAT gateways, and VPC endpoints.

Step 1: Access the VPC Console

1. **Log into the AWS Console.**
2. Ensure you are in the **North Virginia (us-east-1)** region (or your desired region).
3. In the search bar at the top, type **VPC** and select the **VPC Console**.

Step 2: Understand the Default VPC

1. **Default VPC Overview**:
    - AWS creates a **Default VPC** in each region of your account.

    - This VPC includes default subnets, route tables, and an internet gateway.
2. **Security Consideration**:
    - The Default VPC is not highly secure by design and is intended for testing and basic use cases.
    - If not in use, consider deleting the Default VPC (only if no resources like EC2 instances or network interfaces are attached).

Step 3: Create a Custom VPC

1. Click on **Create VPC** in the VPC Console.
2. Select the **"VPC and More"** option for a simplified setup.

Step 4: Configure the Custom VPC

1. **Name Your VPC**:
    - Enter a name for your VPC (e.g., **DemoVPC**).
2. **Set the CIDR Range**:
    - Use the default range (e.g., `10.0.0.0/16`), which allows for 65,536 private IP addresses.
    - **Important**: Ensure this CIDR range does not overlap with any other VPCs or your on-premises network to avoid conflicts.
3. **Understand CIDR Ranges**:
    - CIDR ranges define the IP address space for the VPC.
    - Private IP ranges (e.g., `10.0.0.0/8`, `172.16.0.0/12`, `192.168.0.0/16`) are used for internal networking.

Step 5: Configure Subnets

1. **Availability Zones**:
    - Subnets will be automatically created in **two availability zones** for high availability.
    - Each subnet receives a portion of the VPC's CIDR range.
2. **Subnet Types**:
    - **Public Subnets**:
        - Designed for resources like web servers that need internet access.
        - Associated with an internet gateway.
    - **Private Subnets**:
        - For resources like databases and application servers that do not require direct internet access.
        - Use NAT gateways for outbound internet traffic.
        - The vast majority of workloads you create should be in Private Subnets.

Step 6: Add a NAT Gateway

1. **Purpose**:
    - Allows instances in private subnets to access the internet securely.
    - Prevents inbound traffic from reaching private instances.
2. **Create NAT Gateway**:
    - In the **NAT gateways ($)** section, select **In 1 AZ** in the setup wizard.
    - AWS will allocate an Elastic IP automatically.

Step 7: Add a VPC Endpoint for S3

1. **Purpose**:
    - Allows resources in your VPC to communicate with Amazon S3 directly, without sending traffic over the internet.
2. **Configuration**:
    - Enable the **S3 VPC Endpoint** option during the setup.

Step 8: Review and Create

1. **Review Configuration**:
    - Verify all settings: VPC name, CIDR range, subnets, NAT gateway, and VPC endpoint.
2. **Click Create VPC**:
    - AWS will handle the creation of associated resources, including:
        - Internet Gateway
        - Route Tables
        - Subnets
        - NAT Gateway
        - Elastic IP
3. **Monitor Progress**:
    - The process may take a few minutes as AWS sets up all resources.

Step 9: Verify Your VPC

1. Go to the **VPC Dashboard** and select your newly created VPC.
2. Review the components:
    - CIDR range
    - Subnets
    - Route tables
    - Internet gateway
    - NAT gateway
    - VPC endpoints

Key Concepts Explained

1. **VPC**:
    - Your isolated network in AWS where you can launch resources securely.
    - Provides control over IP address ranges, subnets, routing, and security.
2. **CIDR Range**:
    - Defines the pool of private IP addresses available within the VPC.
3. **Subnets**:
    - **Public Subnets**: Resources can communicate directly with the internet.
    - **Private Subnets**: Resources are isolated from the internet and rely on NAT gateways for outbound traffic.
4. **NAT Gateway**:
    - Allows private instances to securely access the internet.
5. **VPC Endpoint**:
    - Optimizes access to AWS services like S3 without internet exposure.

Step 10: Cleanup (Optional)

If you no longer need the VPC, you can delete it:

1. Terminate any resources (e.g., EC2 instances) using the VPC.
2. Delete subnets, NAT gateways, and route tables.
3. Finally, delete the VPC itself.

Summary

In this lab, you created a custom VPC using AWS's simplified "VPC and More" wizard. You learned about key VPC components, including CIDR ranges, subnets, NAT gateways, and VPC endpoints, and configured a secure environment for deploying AWS resources. By following this process, you can design VPCs tailored to your application's networking and security needs.

This study guide demonstrates the lesson from *Create Highly Available Applications with Load Balancers*
My full AWS Architect Associate course can be found here:
https://www.udemy.com/course/ultimateaws/?referralCode=7ED214B795C444141361

Understanding Elastic Load Balancers (ELB) and Network Load Balancing in AWS Study Guide

In this lesson, we will explore the basic concepts of Elastic Load Balancers (ELBs) in AWS, focusing specifically on **Network Load Balancers (NLBs)**. Load balancing is an essential practice in cloud computing, enabling traffic distribution across multiple instances for scalability, availability, and fault tolerance. While this lesson focuses on NLBs, the concepts also apply broadly to other AWS load balancers, such as Application Load Balancers (ALBs) and Gateway Load Balancers (GLBs).

Introduction to Load Balancers

A **load balancer** acts as a traffic manager, distributing incoming requests across multiple backend targets, such as EC2 instances, containers, or Lambda functions. Load balancers improve:

1. **Scalability**: Supporting more traffic by distributing workloads across multiple instances.
2. **Availability**: Ensuring high uptime through redundancy and health checks.

AWS Elastic Load Balancing provides three types of load balancers:

- **Network Load Balancer (NLB)**: Optimized for high performance and low latency, suitable for TCP and UDP traffic.
- **Application Load Balancer (ALB)**: Operates at the application layer (Layer 7) and is used for HTTP/HTTPS traffic.
- **Gateway Load Balancer (GLB)**: Facilitates transparent deployment of third-party virtual appliances.

Network Load Balancer Basics

Definition:

A **Network Load Balancer (NLB)** operates at Layer 4 (the transport layer) of the OSI model, using IP addresses and ports to manage traffic. It is ideal for high-performance workloads and scenarios requiring low latency.

Key Features:

1. **Traffic Distribution**: Routes incoming traffic to backend instances across multiple **Availability Zones (AZs)**.
2. **Health Checks**: Continuously monitors the health of targets and ensures traffic is only routed to healthy instances.
3. **Scalability**: Handles millions of requests per second, making it suitable for high-traffic applications.
4. **Static IPs**: Each NLB provides a static IP address per AZ, simplifying DNS management.

Benefits of Using a Load Balancer

1. Horizontal Scaling (Scaling Out)

- Instead of relying on a single large instance, multiple smaller instances are deployed to handle traffic.
- Horizontal scaling increases fault tolerance: if one instance fails, others can continue serving traffic.

2. Improved Availability

- Load balancers detect unhealthy instances using periodic **health checks**.
- When an instance or AZ becomes unavailable, the load balancer automatically stops routing traffic to it.

How a Network Load Balancer Works

Scenario Overview:

- Imagine a website hosted on multiple EC2 instances, distributed across two availability zones (AZs).
- The NLB evenly distributes traffic across all healthy instances in both AZs.

Target Groups:

- Backend instances connected to the NLB are organized into **target groups**.
- A target group specifies:
    - Protocol and port for traffic (e.g., TCP on port 80).
    - Health check configurations (e.g., ping frequency, response timeout).

Traffic Flow:

1. Incoming traffic reaches the NLB.
2. The NLB forwards requests to EC2 instances in the target group.
3. Traffic is distributed across healthy targets within multiple AZs.

Health Checks

A **health check** is a periodic test performed by the load balancer to ensure targets are functioning correctly. If an instance fails a health check:

- The NLB stops routing traffic to it.
- Traffic is redirected to healthy instances.

Example:

- If an entire AZ fails, the NLB identifies unhealthy instances and routes all traffic to the remaining healthy AZ.

Handling Failures

Example Scenario:

1. **Normal Operation**:
    - Six EC2 instances are active: three in AZ1 and three in AZ2.
    - The NLB distributes traffic evenly across all instances.
2. **Failure Scenario**:
    - AZ2 fails, taking its three instances offline.
    - The NLB detects this via health checks and stops sending traffic to AZ2.
    - Traffic is routed only to the three healthy instances in AZ1.
3. **Auto Scaling**:
    - If **Auto Scaling** is enabled, additional instances can be launched in AZ1 to handle the increased load.

Horizontal Scaling vs. Vertical Scaling

Horizontal Scaling:

- Adds more instances to handle increased traffic.
- Advantages:
    - Higher fault tolerance: failure of one instance doesn’t affect others.
    - Easier to adjust to varying workloads.

Vertical Scaling:

- Increases the size (e.g., CPU, memory) of a single instance.
- Disadvantages:
    - Single point of failure.
    - Limited scaling capacity.

Horizontal scaling, supported by load balancers, is preferred in modern cloud architectures.

Configuring a Network Load Balancer

Steps to Set Up an NLB:

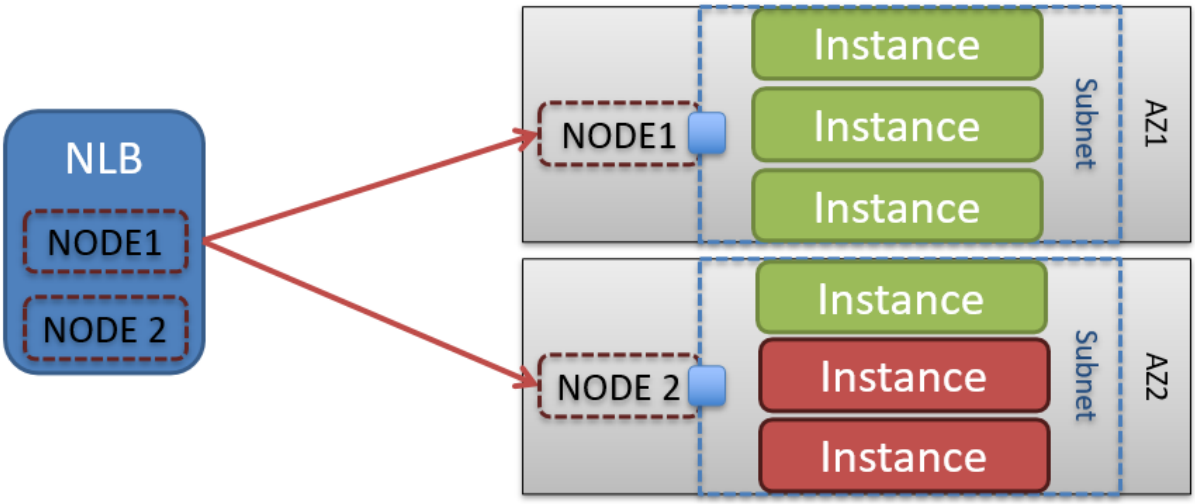
1. **Access the AWS Console**:
    - Navigate to the **EC2 Dashboard** and select **Load Balancers**.
2. **Create a Network Load Balancer**:
    - Select the NLB option.
    - Assign a name (e.g., "WebApp-NLB").
    - Choose the VPC and availability zones for your targets.
3. **Configure Listeners**:
    - Specify the protocol and port (e.g., TCP on port 80).
4. **Create Target Groups**:
    - Add backend EC2 instances to the target group.
    - Define health check parameters.
5. **Test**:
    - Use the NLB's DNS name to send traffic and verify distribution across instances.

Summary

- **Elastic Load Balancers** are critical for distributing traffic, ensuring high availability, and enabling horizontal scaling.
- **Network Load Balancers** are optimized for high-performance, low-latency workloads.
- Key features of NLBs include static IPs, health checks, and multi-AZ traffic distribution.
- Horizontal scaling with load balancers provides fault tolerance and better resource utilization compared to vertical scaling.

See slide below:

Network Load Balancer Example



For more details see my full AWS Architect Associate course:
https://www.udemy.com/course/ultimateaws/?referralCode=7ED214B795C444141361

This lab demonstrates the steps from *Demo: Create a Load Balancer.*
My full AWS Architect Associate course can be found here:
https://www.udemy.com/course/ultimateaws/?referralCode=7ED214B795C444141361

Lab Guide: Setting Up a Network Load Balancer in AWS

This lab guide walks you through the process of setting up a **Network Load Balancer (NLB)** in AWS to distribute traffic across multiple EC2 instances in different Availability Zones. By following these steps, you will gain hands-on experience configuring and testing an NLB while understanding its key concepts.

Lab Objectives

1. Launch two EC2 instances in different Availability Zones.
2. Configure a target group to include the EC2 instances.
3. Create and associate a Network Load Balancer with the target group.
4. Test the load balancer's functionality and observe how it handles instance failures.

Step 1: Launch Two EC2 Instances

1.1 Log into the AWS Console

1. Log into your AWS Console and ensure you're in the **Ohio region (us-east-2)**.
2. Navigate to the **EC2 Dashboard**.

1.2 Launch the First EC2 Instance

1. Click **Launch Instance** and follow these settings:
   - **Name**: `Webserver01a`.
   - **AMI**: Amazon Linux 2 AMI.
   - **Instance Type**: t2.micro (or another free-tier eligible type).

    - **Availability Zone**: Ensure it is in **us-east-2b**.
    - **Key Pair**: Select or create a key pair for SSH access.
    - **Security Group**:
        - Create a new security group checking **Allow HTTPS traffic from the internet** and **Allow HTTP traffic from the internet**.
    - Click **Advanced details** and scroll down to **User data – *optional***.
    - Run the **User Data Script** included in the course resources
2. Launch the instance.

1.3 Launch the Second EC2 Instance

1. Repeat the steps above with the following changes:
    - **Name**: `Webserver02a`.
    - **Availability Zone**: Ensure it is in **us-east-2a**.
    - Use the **same security group** and **key pair** as the first instance.
    - Use the same user data script.
2. Launch the instance.

Step 2: Create a Target Group

2.1 Navigate to Target Groups

1. In the **EC2 Dashboard**, scroll down the left menu and select **Target Groups** under **Load Balancing**.

2.2 Create the Target Group

1. Click **Create Target Group** and configure as follows:
    - **Target Type**: EC2 Instances.
    - **Name**: `DemoTargetGroup`.
    - **Protocol**: HTTP.
    - **Port**: 80.
    - **Health Check Path**: `/`.
    - Leave other settings as default.
2. Click **Next**.

2.3 Register Targets

1. Select both EC2 instances (`Webserver01a` and `Webserver02a`) and include them in the target group.
2. Click **Create Target Group**.

Step 3: Create a Network Load Balancer

3.1 Navigate to Load Balancers

1. In the **EC2 Dashboard**, scroll to **Load Balancers** under **Load Balancing**.

3.2 Configure the Load Balancer

1. Click **Create Load Balancer** and select **Network Load Balancer**.
2. Set the following:
    - **Name**: `DemoNLB`.
    - **Scheme**: Internet-facing.
    - **Availability Zones**: Select all AZs and ensure `us-east-2a` and `us-east-2b` are included.
    - **Security Group**: `WebServers` (not default). It must allow HTTP traffic.
    - **Listeners**:
        - Protocol: HTTP, Port: 80.
3. Click **Next**.

3.3 Associate the Target Group

1. Under **Listeners and Routing**, select `DemoTargetGroup` as the target group.
2. Click **Create Load Balancer**.

Step 4: Test the Network Load Balancer

4.1 Retrieve the Load Balancer's DNS

1. Go to **Load Balancers** in the EC2 Dashboard.
2. Select `DemoNLB` and copy its **DNS Name**.

4.2 Test the Load Balancer

1. Open a web browser and paste the DNS name.
2. Refresh the page multiple times to observe the traffic being distributed between `Webserver01a` and `Webserver02a`.
    - The browser should alternate between displaying "Webserver01a" and "Webserver02a."

Step 5: Simulate an Instance Failure

5.1 Terminate One EC2 Instance

1. Go to **Instances** in the EC2 Dashboard.
2. Select `Webserver02a` and terminate it.

5.2 Verify Target Group Health

1. Return to **Target Groups** and select `DemoTargetGroup`.
2. Go to the **Targets** tab and observe the health status.
    - The terminated instance will show as **unhealthy** and eventually be removed from the target group.
3. Test the load balancer's DNS again. It should now only serve traffic from `Webserver01a`.

Key Concepts

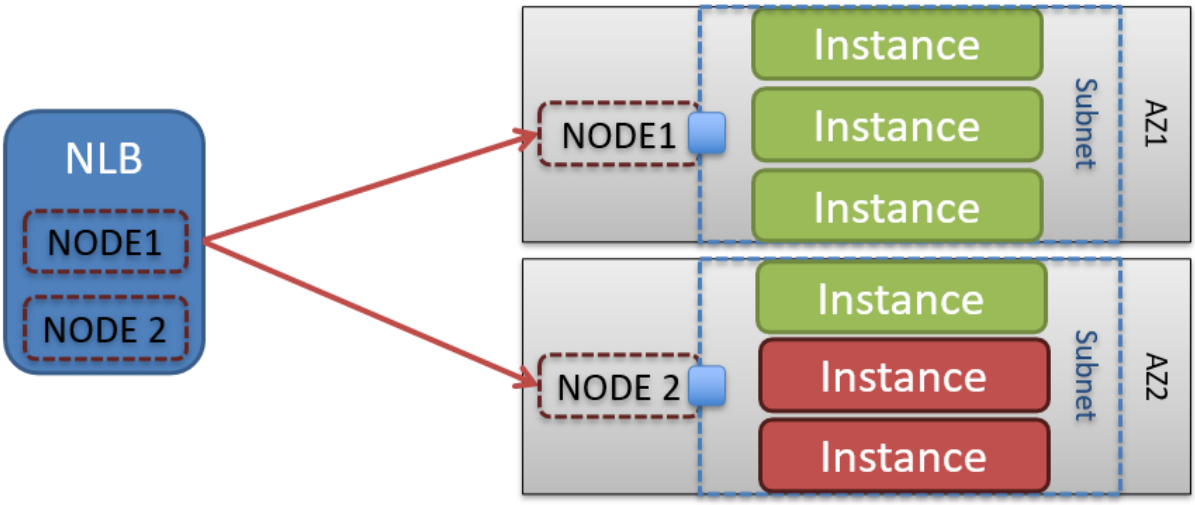
1. **Network Load Balancer (NLB)**:
    - Distributes traffic across multiple targets, such as EC2 instances, for high availability and scalability.
    - Operates at the transport layer (Layer 4).
2. **Target Groups**:
    - A collection of resources (EC2 instances, IPs, or Lambda functions) that receive traffic from the load balancer.
3. **Health Checks**:
    - Periodic checks to determine the availability of targets. Traffic is only routed to healthy targets.
4. **Multi-AZ Deployment**:
    - Distributing targets across multiple Availability Zones ensures high availability even if one AZ fails.
5. **User Data Scripts**:
    - Automates instance setup tasks such as installing software and configuring web servers.

Summary

In this lab, you set up a Network Load Balancer to distribute traffic across EC2 instances in multiple Availability Zones. You also learned how the load balancer detects and handles instance failures using health checks, ensuring continued application availability. By completing this lab, you now understand the fundamentals of configuring and testing an NLB in AWS.

See slide below:

Network Load Balancer Example



For more details see my full AWS Architect Associate course:
https://www.udemy.com/course/ultimateaws/?referralCode=7ED214B795C444141361

This study guide demonstrates the lesson from *Introduction to Serverless Using Lambda – Eliminate Virtual Servers!*
My full AWS Architect Associate course can be found here:
https://www.udemy.com/course/ultimateaws/?referralCode=7ED214B795C444141361

Serverless and Lambda Study Guide

Introduction

The landscape of application development has been revolutionized by the advent of serverless computing, a paradigm shift from traditional server-based approaches. This lesson introduces the fundamental concepts of serverless applications, focusing on AWS Lambda, Amazon Web Services' flagship serverless compute service. By contrasting Lambda with traditional server services like EC2 (Elastic Compute Cloud), we will explore the key advantages of serverless computing, including cost efficiency, scalability, and ease of management.

Understanding Serverless Computing

Serverless computing is a cloud-computing execution model in which the cloud provider manages the execution environment, freeing developers from the complexities of server management.

- **Abstraction of Servers:** Serverless computing doesn't mean the absence of servers, but rather the abstraction of server management from the developer.
- **Event-Driven Architecture:** Serverless applications are typically event-driven, automatically triggering and executing code in response to specific events.

AWS Lambda: A Deep Dive

AWS Lambda is a prime example of serverless computing. It allows you to run code in response to events without the need to provision or manage servers.

- **Function-as-a-Service (FaaS):** Lambda operates on a FaaS model, where you upload your code, and AWS takes care of everything required to run and scale the code execution.
- **Event-Driven Triggers:** Lambda functions can be triggered by various AWS services, such as S3 (Simple Storage Service) and DynamoDB, or by HTTP requests through API Gateway.

Contrasting Lambda with EC2

To appreciate the benefits of AWS Lambda, it's essential to understand how it differs from traditional services like EC2.

- **EC2 (Traditional Server Model):** In EC2, you provision and manage server instances. You are responsible for server health, scaling, and ensuring the server software is up to date.
- **Lambda (Serverless Model):** With Lambda, there are no servers to manage. You are only responsible for your code. AWS handles the scaling, high availability, and maintenance of the underlying infrastructure.

Advantages of Serverless Computing

Serverless computing, as exemplified by AWS Lambda, offers several key benefits:

Cost Efficiency

- **Pay-Per-Use Model:** You only pay for the compute time you consume with Lambda, as opposed to paying for reserved compute capacity with EC2.
- **Reduced Overhead:** The absence of server management reduces the cost associated with infrastructure maintenance.

Scalability

- **Automatic Scaling:** Lambda automatically scales your application by running code in response to each trigger. This can be a single request or thousands at once.
- **Handling Traffic Spikes:** Serverless applications can handle sudden spikes in traffic more gracefully, as the cloud provider manages the scaling.

Ease of Management

- **No Server Management:** The elimination of server management tasks simplifies deployment and operational management.
- **Rapid Deployment:** You can quickly deploy new versions of applications without worrying about the underlying infrastructure.

Use Cases for AWS Lambda

AWS Lambda is suited for various use cases, such as:

- **Web Applications:** Building serverless backends for web applications.
- **Data Processing:** Real-time file processing, stream processing, and batch processing.
- **Automation:** Automating AWS services and backend tasks.

Conclusion

Serverless computing, particularly through AWS Lambda, represents a significant shift in application development and deployment. By abstracting the complexities of server management, Lambda offers a streamlined, cost-effective, and scalable solution for running code in the cloud. This model allows developers to focus on writing code and creating value, rather than managing and operating servers, making serverless computing an increasingly attractive option in a wide range of applications.

See slides below:

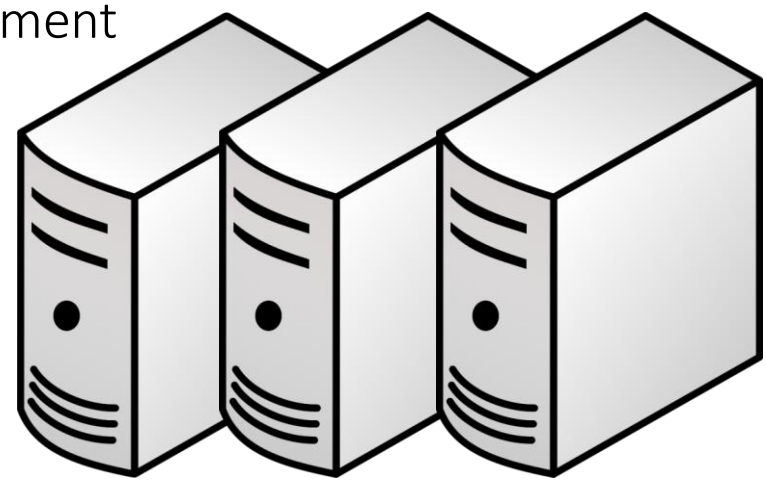
What is Serverless?

Amazon DynamoDB

Amazon S3

What is Serverless?

- EC2 Charges
- OS Management
- ELB

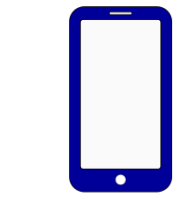


What is Serverless?

Eliminate Management Requirements

- Operating Systems
- Scalability
- Availability
- Fault Tolerance
- No need to pay for idle resources
- Stateless

Invoking Lambda



AWS Services that Invoke Lambda

API Gateway

Amazon Kinesis

Amazon DynamoDB

Amazon S3

Simple Notification Service SNS

Simple Queue Service SQS

For more details see my full AWS Architect Associate course:
https://www.udemy.com/course/ultimateaws/?referralCode=7ED214B795C444141361

This lab demonstrates the steps from *Demo: Lambda in the AWS Console.*
My full AWS Architect Associate course can be found here:
https://www.udemy.com/course/ultimateaws/?referralCode=7ED214B795C444141361

Lab Guide: Creating and Testing Your First AWS Lambda Function

This lab guide will walk you through the process of creating, configuring, and testing your first AWS Lambda function. It explains the key concepts behind AWS Lambda and provides a hands-on demonstration to help you understand how Lambda operates and integrates with other AWS services.

Lab Objectives

1. Understand the basics of the AWS Lambda service.
2. Create a Lambda function from scratch using the AWS Management Console.
3. Modify and deploy code for the Lambda function.
4. Test the function using a custom test event.
5. Optimize function performance by configuring memory and analyzing execution metrics.

Step 1: Accessing the AWS Lambda Console

1. **Log into the AWS Console** and ensure you're in a specific region (e.g., **Ohio (us-east-2)**).
    - **Note**: AWS Lambda is regional, meaning Lambda functions are tied to the region in which they are created.
2. Use the search bar at the top of the AWS Console and search for **Lambda**.
3. Click on **Lambda** to open the Lambda Dashboard.

Step 2: Creating Your First Lambda Function

2.1 Start the Function Creation Process

1. On the Lambda Dashboard, click **Create Function**.
2. Choose **Author from scratch**.

2.2 Configure the Function

1. **Function Name**: Enter a name for your function (e.g., `MyFirstLambda`).
2. **Runtime**: Select the programming language for your function. For this lab, choose **Python 3.9**.
3. Leave the **Permissions** as default to create a new execution role.
4. Click **Create Function**.

Step 3: Writing and Deploying Code

3.1 Edit the Default Code

1. In the Lambda function editor, you'll see some default code preloaded by AWS.
2. Replace the default code with the following simple script:

```
def lambda_handler(event, context):
    print("Hello from Lambda!")
    return {
        'statusCode': 200,
        'body': 'Lambda executed successfully!'
    }
```

3.2 Deploy the Code

1. After editing the code, click the **Deploy** button.
    - Deploying saves the updated code to the Lambda function, making it ready for execution.

Step 4: Testing the Lambda Function

4.1 Configure a Test Event

1. Click the **Test** button at the top of the function editor.
2. Create a new test event:
    - **Event Name**: Enter `MyTestEvent`.
    - **Template**: Select **CloudWatch Logs** or leave the default JSON template.
3. Click **Save**.

4.2 Run the Test

1. Click the **Test** button again to execute the function.

2. Review the output:
    - Expand the **Execution Results** section to see the function's response and logs.
    - The print statement (`"Hello from Lambda!"`) should appear in the logs.

Step 5: Analyzing Function Performance

5.1 View Execution Metrics

1. After testing, review the **Details** section in the execution results:
    - **Duration**: The time taken to execute the function.
    - **Billed Duration**: The time rounded up to the nearest millisecond for billing.
    - **Max Memory Used**: The amount of memory consumed during execution.

5.2 Adjust Memory Allocation

1. Scroll to the **Configuration** tab and select **General Configuration**.
2. Increase the memory from the default **128 MB** to **512 MB**.
3. Save the changes.

5.3 Retest the Function

1. Click **Test** again to execute the function with the updated memory allocation.
2. Compare the new execution metrics to the previous test:
    - Check whether the increased memory improved execution time or had no significant effect.

Key Concepts

AWS Lambda

- **Serverless Computing**: Lambda allows you to run code without provisioning or managing servers.
- **Event-Driven**: Lambda functions are triggered by events from AWS services like S3, DynamoDB, and CloudWatch.
- **Pay-per-Use**: Billing is based on the number of requests and execution time (rounded to the nearest millisecond).

Test Events

- **Purpose**: Simulate triggers for Lambda functions during development.
- **Templates**: AWS provides sample JSON templates for common triggers (e.g., S3, CloudWatch).

Performance Tuning

- **Memory Allocation**: Adjusting memory can impact performance and cost.
- **Metrics**:
    - **Duration**: How long the function takes to run.
    - **Max Memory Used**: Indicates whether the allocated memory is sufficient for the workload.

Step 6: Clean Up

1. To avoid incurring charges:
    - Delete the Lambda function:
        - Go to the **Functions** page, select your function, and click **Delete**.
    - Delete any associated resources (e.g., IAM roles).

Summary

In this lab, you created and tested your first AWS Lambda function, gaining insights into its configuration, execution, and performance. You also learned how to analyze metrics and optimize memory settings to improve efficiency. This hands-on experience provides a foundation for understanding serverless computing in AWS.

This study guide demonstrates the lesson from *Elasticity – Scaling Your Architecture to Meet Demand and Reduce Cost.*
My full AWS Architect Associate course can be found here:
https://www.udemy.com/course/ultimateaws/?referralCode=7ED214B795C444141361

Elasticity in AWS Cloud Study Guide

In this lesson, we dive into a pivotal concept that is the backbone of cloud computing – Elasticity. Elasticity is a concept that underpins the dynamic scalability of cloud resources based on demand. This crucial lesson will empower you to understand how AWS resources can effortlessly stretch or shrink to accommodate changing requirements.

Understanding Elasticity

> **The Rubber Band Analogy** Elasticity, when applied to the AWS cloud, is best grasped through a simple analogy. Visualize it as a rubber band – just like you can stretch and expand a rubber band when needed, AWS resources can adapt their capacity to match your current demands. When the demand wanes, they contract, mirroring the rubber band's ability to return to its original state.

> **Real-World Application** This elasticity concept is not abstract; it finds practical applications across AWS services, making it an essential principle to master for cloud professionals.

Scaling Out: Meeting Increased Demand

> **Scenario: Web Server Hosting** Imagine you have an EC2 instance hosting your web server. During periods of surging demand, when resource utilization soars, the EC2 instance's performance might begin to degrade. This decline manifests as extended response times and a less efficient operation. How can you address this situation?

> **Scaling Out in AWS** Scaling out is the answer. To mitigate the impact of heightened demand, AWS allows you to create additional EC2 instances that can distribute the workload. As these new EC2 instances come

online, they collectively shoulder the increased demand, effectively reducing the strain on any individual EC2 instance. In this section, we delve into the details of this crucial concept.

> **Scaling Out vs. Scaling Up** It's vital to distinguish between scaling out and scaling up. Scaling up involves upgrading the existing EC2 instance's size and power. However, this approach results in a single, more potent instance. In contrast, scaling out involves creating multiple resources that work collaboratively to fulfill the required tasks. This collaborative effort offers improved availability, making it an attractive option for many cloud scenarios.

Scaling In: Optimization and Cost Reduction

> **Managing Costs and Resource Efficiency** While scaling out efficiently meets high demand, it's equally important to consider scaling in when demand recedes. For example, if you have four running EC2 instances, you are billed for all four instances. If the demand decreases, keeping all instances operational could lead to unnecessary costs.

> **Scaling In with AWS** Scaling in comes to the rescue. It involves terminating unnecessary resources, ensuring you only pay for what you genuinely require. For example, if two instances can handle the workload effectively, AWS can automatically terminate the surplus instances, providing cost savings.

Beyond EC2: Elasticity in AWS

> **Auto Scaling Groups** In the next lesson, we explore how Auto Scaling Groups, a fundamental AWS feature, can provide elasticity, specifically in the context of EC2 instances. This feature plays a pivotal role in managing resource scalability and optimizing performance.

> **Diverse AWS Services with Elasticity** Elasticity extends beyond EC2 and is a prevalent characteristic of numerous AWS services. Understanding how to harness this elasticity across various services is vital for constructing resilient and cost-effective AWS architectures.

See slides below:

Key AWS Concept: Elasticity

Elasticity

- Grow or shrink infrastructure resources dynamically
- Responds to changes in demand
- Scales out to meet performance requirements
- Scales in to reduce costs during times of low usage

This study guide demonstrates the lesson from *Elasticity with EC2 Auto Scaling Groups.*
My full AWS Architect Associate course can be found here:
https://www.udemy.com/course/ultimateaws/?referralCode=7ED214B795C444141361

Understanding Auto Scaling in AWS Study Guide

Auto Scaling is a critical feature in AWS that dynamically adjusts the number of EC2 instances in response to changing demand. This ensures that applications remain highly available and cost-efficient by scaling out when demand increases and scaling in during periods of low demand. In this lesson, we'll explore the fundamental components of Auto Scaling, including Launch Templates, Launch Configurations, and Auto Scaling Groups (ASGs), and how they work together.

What is Auto Scaling?

Auto Scaling automatically manages the number of EC2 instances in a defined group to match current demand. This elasticity allows workloads to scale efficiently, improving application availability while optimizing costs.

Key Features of Auto Scaling:

1. **Dynamic Scaling**: Automatically adjusts resources in response to predefined metrics or events (e.g., CPU usage, network traffic).
2. **Elasticity**: Scales out (adds instances) during high demand and scales in (removes instances) during low demand.
3. **Integration with Load Balancers**: Ensures new instances are automatically added to target groups and begin handling traffic immediately.
4. **Fault Tolerance**: Detects and replaces unhealthy instances.

Key Components of Auto Scaling

1. Launch Templates

A **Launch Template** is a blueprint for creating EC2 instances. It defines the configuration of the instances that the Auto Scaling Group will manage.

Key Attributes of a Launch Template:

- **AMI (Amazon Machine Image)**: Determines the operating system (e.g., Amazon Linux, Ubuntu, Windows) and pre-installed software.
- **Instance Type**: Specifies the hardware configuration (e.g., t2.micro, m5.large).
- **User Data Scripts**: Automates instance setup by running commands during boot (e.g., installing web servers).
- **Storage Configuration**: Defines EBS volume size and type.
- **Network Settings**: Specifies the VPC, subnets, and security groups.
- **Key Pair**: Enables SSH access to instances.

Launch Templates provide flexibility because they can be used to:

- Launch individual EC2 instances.
- Serve as a template for Auto Scaling Groups.

---

2. Launch Configurations

A **Launch Configuration** is similar to a Launch Template but is exclusively used with Auto Scaling Groups. Unlike Launch Templates:

- It **cannot** be used to launch individual EC2 instances.
- Once created, it **cannot be modified** (you must create a new one for changes).

3. Auto Scaling Groups (ASG)

An **Auto Scaling Group** is a logical grouping of EC2 instances managed by Auto Scaling. It uses the configuration defined in a Launch Template or Launch Configuration to scale resources dynamically.

Key Attributes of an ASG:

- **Minimum, Maximum, and Desired Capacity**:
  - **Minimum**: The least number of instances the ASG should maintain.
  - **Maximum**: The upper limit of instances the ASG can scale out to.
  - **Desired Capacity**: The number of instances the ASG tries to maintain under normal conditions.
- **Scaling Policies**:
  - **Dynamic Scaling**: Adds or removes instances based on performance metrics (e.g., CPU utilization).
  - **Scheduled Scaling**: Adjusts capacity at specific times (e.g., increasing capacity during business hours).
- **Health Checks**: Automatically replaces unhealthy instances.
- **Integration with Load Balancers**: Ensures traffic is distributed to instances efficiently.

How Auto Scaling Works

Step 1: Create a Launch Template

The Launch Template defines the EC2 instances' specifications:

1. Choose an **AMI** for the instance.
2. Specify the **instance type** (e.g., t2.micro).
3. Add **User Data** for automated setup:
    - Example: Install Apache on Amazon Linux:

```
#!/bin/bash
sudo yum update -y
sudo yum install -y httpd
sudo systemctl start httpd
sudo systemctl enable httpd
echo "Hello from Auto Scaling!" > /var/www/html/index.html
```

4. Configure **network settings**: Assign subnets and security groups.
5. Save the Launch Template.

Step 2: Configure the Auto Scaling Group

1. Define the ASG's **capacity settings**:
    - Minimum: 1
    - Desired: 2
    - Maximum: 5
2. Attach the ASG to a **Load Balancer Target Group**:
    - Ensures new instances are automatically registered and begin receiving traffic.
3. Set **scaling policies**:
    - Example: Add an instance if CPU utilization exceeds 70%.
    - Example: Remove an instance if CPU utilization drops below 30%.

Step 3: Monitor and Adjust

1. Auto Scaling launches instances when demand increases and terminates them when demand decreases.
2. The ASG continuously checks instance health and replaces any failing instances.

Elastic Scaling: Scale In and Scale Out

Scale Out (Add Instances)

- Triggered when demand increases, such as:
    - High CPU utilization.
    - Spikes in network traffic.
- Example: During a flash sale, Auto Scaling might launch 10 additional instances to handle the load.

Scale In (Remove Instances)

- Triggered during periods of low demand.
- Reduces operational costs by shutting down unnecessary instances.

Elasticity Analogy:

Think of Auto Scaling like a rubber band that stretches (scales out), or contracts (scales in) based on demand. It ensures you have just the right amount of resources at any given time.

Benefits of Auto Scaling

1. **Cost Efficiency**:
   - Scale out during peak demand to ensure availability.
   - Scale in during low demand to reduce costs.
2. **High Availability**:
   - Maintains sufficient capacity to meet demand at all times.
3. **Fault Tolerance**:
   - Automatically replaces unhealthy instances.
4. **Seamless Integration**:
   - Works with Elastic Load Balancers to distribute traffic efficiently.

Launch Template vs. Launch Configuration

Feature	Launch Template	Launch Configuration
Usage	Launch instances or ASGs	Only for ASGs
Modifiable	Yes	No
Advanced Features	Supports newer features	Limited capabilities
Reusability	Can be reused for multiple purposes	Single-purpose

---

Summary

In this lesson, you learned how Auto Scaling in AWS dynamically adjusts EC2 instance capacity based on demand. By creating a Launch Template as a blueprint and configuring an Auto Scaling Group, you can ensure your applications are highly available and cost-efficient. Auto Scaling's ability to elastically scale resources provides flexibility, fault tolerance, and operational efficiency, making it an essential tool in cloud computing.

See slides below:

Auto Scaling

BLUEPRINT

Launch Template or
Launch Configuration

"What should the EC2 instances look like? Blueprint!"

AMI
User Data
Instance Type (T3? C5?)
EBS volume?
Subnet, VPC

ASG

Auto Scaling Group

"How many instances should be created and when?

ELB?

Auto Scaling

- Right number EC2 instances to handle the workload
- Configurable minimum and maximum number of instances
- Auto Scaling expands or contracts based on demand
- This is called “Scaling In” or “Scaling Out”

Launch Template

- Provides a template for EC2 Instances
- Specifies the source AMI
- You can create boot scripts to customize each instance

Launch Configuration

- Basically the same as a Launch Template
- Launch Templates can be used to create individual EC2 instances or Auto Scaling Groups
- Launch Configuration can only be used with Auto Scaling

This lab demonstrates the steps from *Demo: Auto Scaling*.
My full AWS Architect Associate course can be found here:
https://www.udemy.com/course/ultimateaws/?referralCode=7ED214B795C444141361

Lab Guide: Setting Up an Auto Scaling Group for EC2 in AWS

This lab guide walks you through the process of setting up an **Auto Scaling Group (ASG)** for EC2 instances in AWS. By following along, you will understand how to create a **Launch Template**, configure the **Auto Scaling Group**, and test the automatic scaling of EC2 instances.

Lab Objectives

1. Create a **Launch Template** as a blueprint for EC2 instances.
2. Configure an **Auto Scaling Group** to manage instances dynamically.
3. Understand how scaling policies adjust the number of instances based on demand.

Step 1: Access the AWS Console

1. Log in to the **AWS Management Console** and ensure you are in the **North Virginia (us-east-1)** region.
2. Navigate to the **EC2 Dashboard** by clicking on **EC2** from the AWS home screen.

Step 2: Create a Launch Template

2.1 Start the Launch Template Creation

1. From the left-hand menu, select **Launch Templates** under the "Instances" section.
2. Click **Create Launch Template**.

2.2 Configure the Launch Template

1. **Name**: Enter a name for the template, such as `DemoLT`.
2. **Template Version**: Leave as version 1 (default).
3. **AMI (Amazon Machine Image)**: Choose **Amazon Linux 2 AMI**.
4. **Instance Type**: Select **t2.micro** (free tier eligible).
5. **Key Pair**: Choose an existing key pair or create a new one to allow SSH access.
6. **Network Settings**:
    - Do not specify a subnet (the ASG will handle this).
    - Select an existing **security group**. For this example, choose a security group that allows:
        - **SSH** (port 22) for access.
        - **HTTP** (port 80) for web traffic.
7. **User Data (Optional)**: Add a script to automatically configure instances:

```
#!/bin/bash
sudo yum update -y
sudo yum install -y httpd
sudo systemctl start httpd
sudo systemctl enable httpd
echo "Welcome to Auto Scaling!" > /var/www/html/index.html
```

8. Scroll to the bottom and click **Create Launch Template**.

Step 3: Create an Auto Scaling Group

3.1 Start the Auto Scaling Group Setup

1. Go back to the **EC2 Dashboard**.
2. From the left-hand menu, click on **Auto Scaling Groups** under the "Auto Scaling" section.
3. Click **Create Auto Scaling Group**.

3.2 Configure the Auto Scaling Group

1. **Name**: Enter a name, such as `DemoASG`.
2. **Launch Template**: Select the Launch Template you just created (`DemoLT`).
3. Click **Next**.

3.3 Define Network Settings

1. **VPC**: Choose the default VPC or another VPC if configured.
2. **Availability Zones**:
    - Select multiple Availability Zones (e.g., us-east-1a and us-east-1b) to ensure high availability.
3. Click **Next**.

3.4 Configure Load Balancing (Optional)

1. **Load Balancer**: For simplicity, do not associate a load balancer in this demo. In production, you would typically use an Elastic Load Balancer (ELB) to distribute traffic across instances.
2. Click **Next**.

3.5 Set Desired Capacity and Scaling Policies

1. **Group Size**:
    - **Desired Capacity**: 2 (start with 2 instances).
    - **Minimum Capacity**: 1 (ensure at least 1 instance is always running).
    - **Maximum Capacity**: 5 (allow up to 5 instances).
2. **Scaling Policy**:
    - Select **Target Tracking Scaling Policy**.
    - Configure the target value for **CPU Utilization**:
        - Example: Maintain an average of **70% CPU utilization** across instances.
    - Auto Scaling will add instances when CPU utilization exceeds 70% and remove instances when it falls below 70%.
3. Click **Next**.

3.6 Review and Create

1. Review all configurations, then click **Create Auto Scaling Group**.

Step 4: Monitor the Auto Scaling Group

4.1 View Instances

1. In the Auto Scaling Group dashboard, click on your newly created ASG (`DemoASG`).
2. Go to the **Instance Management** tab.
3. You will see two EC2 instances being created (as per the desired capacity).

4.2 Test Automatic Scaling

1. Simulate high CPU usage to trigger scaling out (launching more instances).
    - For simplicity, manual testing or more advanced workload generators can be used in production setups.
2. Observe how Auto Scaling adds or removes instances based on the scaling policy.

Step 5: Key Concepts Explained

5.1 Launch Template

- A **Launch Template** is a reusable blueprint defining how EC2 instances should be configured.
- Key elements include:
    - AMI (operating system)
    - Instance type (e.g., t2.micro)
    - Key pair, security groups, and user data.

5.2 Auto Scaling Group

- An **Auto Scaling Group (ASG)** ensures the correct number of EC2 instances are running to handle workload demands.
- It uses the Launch Template to create instances and adjusts their number dynamically based on scaling policies.

5.3 Scaling Policies

- **Target Tracking Scaling**: Automatically adjusts capacity to maintain a target metric (e.g., 70% CPU utilization).
- **Scheduled Scaling**: Adjusts capacity based on pre-defined schedules.
- **Dynamic Scaling**: Adds or removes instances based on custom policies triggered by CloudWatch alarms.

Step 6: Clean Up

1. To avoid incurring costs, delete resources created during the lab:
    - Terminate all EC2 instances in the ASG.
    - Delete the Auto Scaling Group.
    - Delete the Launch Template.

Summary

In this lab, you learned how to configure an Auto Scaling Group to dynamically manage EC2 instances based on demand. You created a Launch Template as a blueprint for the instances and used a Target Tracking Scaling Policy to maintain optimal performance and cost efficiency. By automating scaling, AWS Auto Scaling ensures that your applications remain highly available while minimizing resource usage and costs.

This study guide demonstrates the lesson from *The Shared Responsibility Model – What are My Responsibilities?*
My full AWS Architect Associate course can be found here:
https://www.udemy.com/course/ultimateaws/?referralCode=7ED214B795C444141361

AWS Shared Responsibility Model Study Guide

In this lesson, we delve into the core concept of the AWS Shared Responsibility Model. This model outlines the distribution of security responsibilities between AWS, the cloud service provider, and you, the customer, when you create an account on AWS.

1. Security of the Cloud vs. Security in the Cloud:

- AWS provides "Security of the Cloud," encompassing the physical security of data centers, secure hardware retirement, and network-level security, including border firewalls.
- As the customer, you are responsible for "Security in the Cloud." This includes tasks like patching and hardening guest operating systems, creating security groups to manage network traffic, and effectively managing Identity and Access Management (IAM) credentials. Multi-factor authentication (MFA) is also recommended to enhance security.

2. Managed Services:

- AWS takes administrative responsibility for "Managed Services," such as patching the operating system and minimizing the attack surface. However, customers retain the responsibility for user access and account management.

3. Data Destruction:

- AWS handles the secure destruction of data on decommissioned storage systems, relieving customers of this task.

4. Virtual Private Cloud (VPC):

- Customers are responsible for creating their own Virtual Private Cloud (VPC) rather than relying on the default VPC automatically provided when signing up for an AWS account. The VPC acts as an isolated environment, enabling control over resources with features like security groups and access lists.

5. Unmanaged Services:

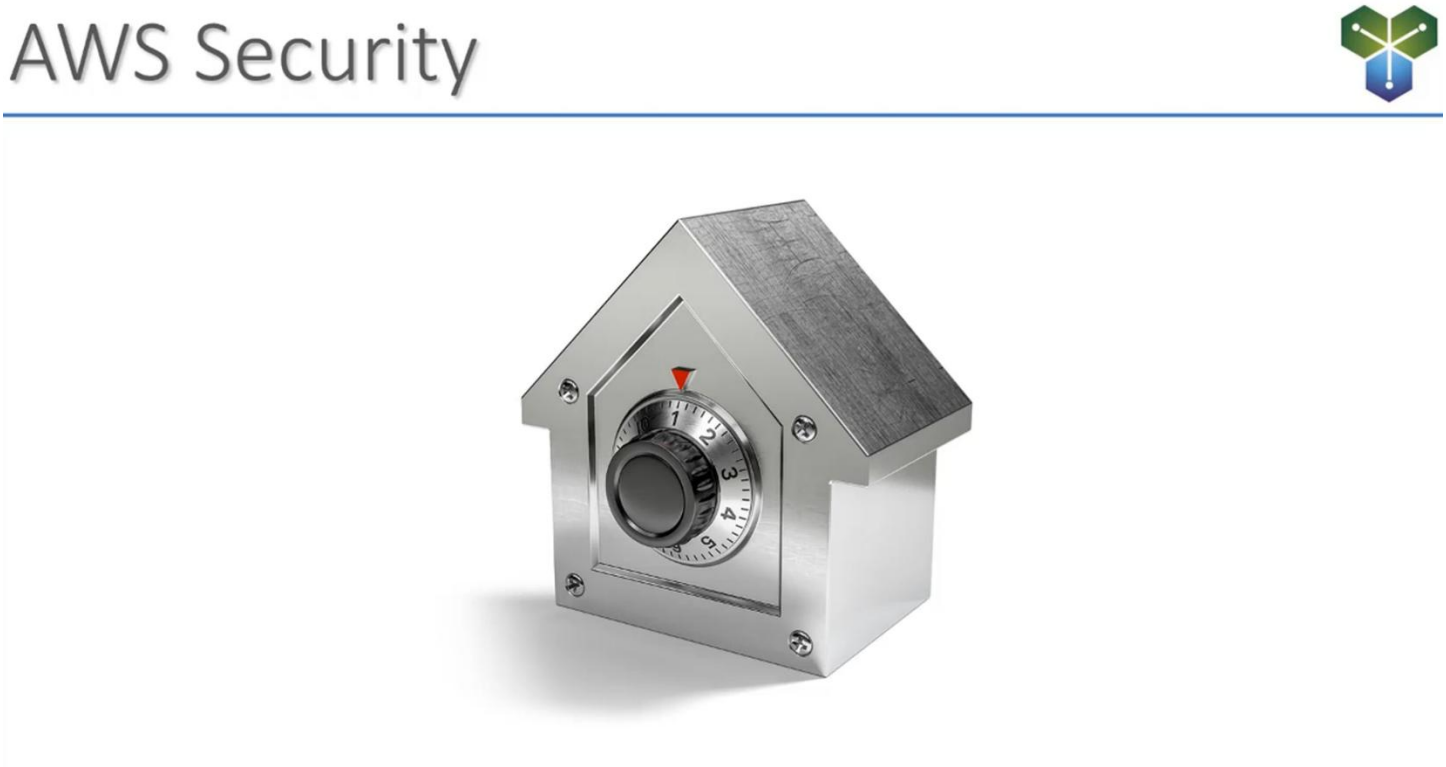
- For unmanaged services like Amazon EC2, where customers have full control over the operating system, it is your responsibility to keep the OS patched, install antivirus, and configure necessary security measures like firewalls.

6. Managed Service Security:

- Even on managed services like Amazon S3 for storage, customers must take charge of securing their content. This includes managing the public or private status of S3 buckets and configuring roles and permissions to control access to bucket contents.

Understanding the AWS Shared Responsibility Model is essential for ensuring a secure and compliant AWS environment. AWS handles certain security aspects, but it's crucial for customers to be aware of their responsibilities and actively manage them to maintain a robust security posture.

See slides below:



Shared Responsibility Model

AWS:

- Physical Security, Hardware Retirement, Border Firewall

Customer:

- Guest O.S Patching, Security Groups, IAM

Additional AWS Responsibilities

- Updates and Anti-virus on managed services
- The customer is still responsible for user access and account management
- Destruction of retired storage systems

Additional Customer Responsibilities

- VPC
- EC2 instances
- S3

For more details see my full AWS Architect Associate course:
https://www.udemy.com/course/ultimateaws/?referralCode=7ED214B795C444141361

This lab demonstrates the steps from *Demo: Navigating the IAM Dashboard.*
My full AWS Architect Associate course can be found here:
https://www.udemy.com/course/ultimateaws/?referralCode=7ED214B795C444141361

Lab Guide: Exploring the IAM Dashboard and Managing the Root Account in AWS

Objective:

- Understand the basic elements of the AWS Identity and Access Management (IAM) dashboard.
- Learn about root account security configurations and key IAM concepts.

---

1. Accessing the AWS Console and Region Settings

Steps:

1. **Log in to the AWS Management Console.**
   - Upon logging in, you will see the AWS home screen.
   - The top right corner displays the current region (e.g., *Ohio*).
   - AWS regions represent different geographic locations where AWS data centers operate.
2. **Understand Regional vs. Global Services:**
   - Certain AWS services, such as **VPCs** (Virtual Private Clouds) and **EC2 instances**, are regional. Resources created in one region are not automatically available in another.
   - IAM, however, is a **global service**. Changes made in IAM apply across all AWS regions.

Key Concept:

- **Global Services**: Services that operate across all AWS regions (e.g., IAM, Route 53).
- **Regional Services**: Services limited to a specific region (e.g., EC2, S3).

---

2. Navigating to the IAM Dashboard

Steps:

1. **Search for IAM:**
    - Use the **search bar** at the top of the AWS console and type "IAM."
    - Select the **IAM dashboard** from the dropdown menu.
2. **Observe the Dashboard Layout:**
    - The IAM dashboard provides an overview of security recommendations and configuration status.
    - Key recommendations include:
        - Enabling **multi-factor authentication (MFA)** for the root account.
        - Deleting **root account access keys** (if any exist).

---

3. Securing the Root Account

Steps:

1. **Access Root Account Security Credentials:**
    - Click on your **account name** in the top right corner.
    - Select **"Security Credentials"** from the dropdown menu.
2. **Review Root Account Configurations:**
    - **Password Management**: You can update the root account password here.
    - **Multi-Factor Authentication (MFA)**:
        - MFA adds an extra layer of security by requiring a second authentication factor (e.g., a smartphone).
        - If MFA is not enabled, click **"Manage"** to set up an MFA device.
3. **Enabling MFA for the Root Account:**
    - Select **"Add MFA Device."**
    - Follow the prompts to scan a QR code with your MFA app (e.g., Google Authenticator).
    - Test and confirm the MFA setup.
4. **Access Key Management:**
    - Verify that there are **no access keys** for the root account. Access keys are credentials used for programmatic access to AWS services.
    - If access keys exist, delete them to enhance security.

Key Concept:

- **Root Account**: The master account with full administrative privileges. It should only be used for critical tasks and secured with strong security measures.
- **MFA**: Enhances account security by requiring both a password and a time-based one-time code.

---

4. IAM Sign-In URL and Account Alias

Steps:

1. **Locate the IAM User Sign-In URL:**
    - On the IAM dashboard, locate the **Sign-In URL** in the right-hand panel.
    - This URL is specific to your AWS account and can be used by IAM users to log in.
2. **Test the Sign-In URL:**

    - Copy the URL and open it in an **incognito/private browsing window**.
    - Paste the URL into the address bar.
    - Notice that it directs you to a sign-in page specific to your AWS account.
3. **Sign-In Options:**
    - You can log in as an **IAM user** with a username and password.
    - Alternatively, log in as the **root user** using the email address associated with the account.
4. **Customize the Account Alias:**
    - In the IAM dashboard, locate the **Account Alias** setting.
    - The alias customizes the sign-in URL to make it more user-friendly (e.g., `your-company.signin.aws.amazon.com`).
    - To edit the alias:
        - Click **"Edit"** next to the current alias.
        - Enter a new alias (e.g., `YourOrg-Security`).
        - Save the changes.
5. **Verify the Updated URL:**
    - After updating the alias, notice that the sign-in URL now reflects the new alias.

Key Concepts:

- **IAM Sign-In URL**: A unique URL for IAM user login, tied to your AWS account.
- **Account Alias**: A customizable identifier that makes the sign-in URL easier to remember.

---

5. Best Practices for the IAM Dashboard

- **Enable MFA** on all accounts, especially the root account.
- **Delete root access keys** to minimize security risks.
- Use **IAM users** and **groups** with appropriate permissions instead of logging in as the root user.
- Regularly review and update IAM policies and permissions.

Conclusion:

In this lab, you explored the IAM dashboard, secured the root account, and learned how IAM configurations apply globally across AWS regions. You also customized the IAM sign-in URL to enhance user access. These foundational steps are critical for maintaining security and efficient account management in AWS.

This lab demonstrates the steps from *Demo: Assign IAM Permissions and Group and Users.*
My full AWS Architect Associate course can be found here:
https://www.udemy.com/course/ultimateaws/?referralCode=7ED214B795C444141361

Lab Guide: IAM Users, Groups, and Policies in AWS

Objective:

- Understand IAM users, groups, and policies.
- Learn how to create and manage users, attach policies, and utilize groups for efficient permission management.

---

1. Accessing the IAM Dashboard

Steps:

1. **Log in to the AWS Console.**
   - Navigate to the **IAM Dashboard** by searching for "IAM" in the AWS Console search bar.
2. **Observe Global Scope:**
   - Note that the region automatically changes to **Global** when accessing IAM.
   - IAM configurations apply across all AWS regions.

Key Concept:

- **IAM Global Service:** IAM settings are global and not tied to specific regions.

---

2. Understanding IAM Users

Definition:

- **IAM User**: A digital identity in AWS associated with one individual in real life. Each user has a unique set of credentials for accessing AWS resources.

Steps to Create a User:

1. In the IAM Dashboard, click **"Users"** in the left-hand menu, then click **"Create user."**
2. Enter a username (e.g., "Rick").
3. Choose **"AWS Management Console Access"** to allow the user to log in to the console.
4. Create a password:
    - Optionally, enable **"Require Password Reset"** for first-time login.
5. **Access Keys**:
    - Do **not** create access keys unless the user requires programmatic access via the AWS CLI, SDK, or API.
    - Access keys should only be generated when necessary for enhanced security.
6. Click **"Next"** to proceed to permissions.

Key Concepts:

- **Console Access**: Grants the user the ability to log in via the AWS Management Console.
- **Access Keys**: Credentials for programmatic access; use them sparingly to reduce security risks.

---

3. Attaching Policies to Users

Definition:

- **Policy**: A document defining permissions (e.g., access to specific services and actions). Policies control what a user can and cannot do in AWS.

Steps to Attach a Policy:

1. Choose **"Attach existing policies directly"** for the user.
2. Search for a policy (e.g., type "S3").
    - Example: **"AmazonS3FullAccess"** allows full access to S3.
    - Alternatively, select **"AmazonS3ReadOnlyAccess"** for read-only permissions.
3. Select the desired policy and attach it to the user.

Key Concepts:

- **AWS Managed Policies**: Predefined policies created and maintained by AWS.
- **Customer Managed Policies**: Custom policies created by you to meet specific needs.

---

4. Using IAM Groups for Efficient Permission Management

Definition:

- **IAM Group**: A collection of users sharing the same permissions. Permissions are assigned to the group, and all users in the group inherit them.

Steps to Create a Group:

1. In the IAM Dashboard, click **"User groups"** in the left-hand menu, then click **"Create group."**
2. Enter a group name (e.g., "Admins").
3. Assign permissions to the group:
    - Search for the policy **"AdministratorAccess."**
    - Select this policy to grant full administrative privileges to the group.
4. Click **"Create user group."**

Adding Users to the Group:

1. Go to the **"Users"** section and select the user you want to add (e.g., "Rick").
2. Click **"Add to Groups."**
3. Select the appropriate group (e.g., "Admins") and confirm.

Key Concept:

- **Group Benefits**: Managing permissions at the group level simplifies administration and reduces the risk of inconsistencies when dealing with multiple users.

---

5. Exploring Policies in Detail

Types of Policies:

1. **AWS Managed Policies**:
    - Maintained by AWS and regularly updated to follow best practices.
    - Example: **"AmazonS3FullAccess"**.
2. **Customer Managed Policies**:
    - Created and customized by the account owner for specific use cases.
    - Example: A policy restricting access to specific S3 buckets.

Steps to Create a Custom Policy:

1. In the IAM Dashboard, click **"Policies"** and select **"Create Policy."**
2. Use the **Visual Editor** or write the policy in JSON format.
3. Define actions, resources, and conditions for the policy.
4. Save the policy and attach it to a user, group, or role as needed.

Key Concept:

- **Policy Attachment**: Policies can be attached directly to users or indirectly via groups. Group-level management is preferred for better scalability.

---

6. Best Practices for IAM

1. **Use Groups**:
    - Always assign permissions at the group level when possible.

    - Avoid attaching policies directly to individual users.
2. **Enable MFA**:
    - Require multi-factor authentication for all users, especially those with elevated privileges.
3. **Follow the Principle of Least Privilege**:
    - Grant only the permissions a user or group needs to perform their job.
4. **Avoid Using Root Account for Daily Tasks**:
    - Reserve the root account for account and billing management.
5. **Regularly Review IAM Policies**:
    - Audit permissions to ensure they align with current business needs.

Summary:

In this lab, you:

- Learned about IAM users, groups, and policies.
- Created an IAM user and assigned permissions directly via policies.
- Created an IAM group, assigned permissions to the group, and added users to it.
- Explored AWS-managed and customer-managed policies.

By following these steps, you can effectively manage access and permissions in AWS, ensuring a secure and scalable environment.

This study guide demonstrates the lesson from *Introduction to CloudWatch – Create Alarms and Notifications.*
My full AWS Architect Associate course can be found here:
https://www.udemy.com/course/ultimateaws/?referralCode=7ED214B795C444141361

AWS CloudWatch - Monitoring and Automation in the AWS Cloud Study Guide

---

1. Introduction to AWS CloudWatch

AWS CloudWatch is a monitoring and management service within the AWS ecosystem. It provides real-time data and insights into AWS services, applications, and infrastructure. By using CloudWatch, administrators can monitor metrics, collect and aggregate log files, and set up alarms to respond to predefined conditions.

---

2. Key Features of AWS CloudWatch

2.1 Monitoring Metrics

CloudWatch provides detailed monitoring for a variety of AWS services. Common examples include:

- **EC2 Instances**: Monitor metrics such as CPU utilization, disk I/O, and network traffic.
- **EBS Volumes**: Track disk performance metrics like read/write operations.
- **Elastic Load Balancers**: Analyze latency and request metrics to ensure traffic is being handled properly.

Metrics are organized by namespaces (e.g., `AWS/EC2` for EC2 metrics), and each metric consists of:

- **Namespace**: A container for related metrics.
- **Dimensions**: Metadata for filtering data, such as instance IDs.
- **Statistics**: Values derived from raw data, such as average, minimum, or maximum.

2.2 Aggregating Log Files

- CloudWatch Logs allows you to **collect, monitor, and store logs** from AWS services and custom applications.

- Logs from EC2 instances can be pushed to CloudWatch Logs for centralized analysis and troubleshooting.

2.3 Setting Alarms

- CloudWatch Alarms monitor metrics and take action when a threshold is breached.
  - Example: An alarm can be triggered if **CPU utilization** on an EC2 instance exceeds 90%.
- Alarms can integrate with **SNS (Simple Notification Service)** to send alerts via email, SMS, or other endpoints.

2.4 Automation with CloudWatch

- CloudWatch is not limited to monitoring—it can **trigger automated actions**:
  - Invoke AWS **Lambda** functions to execute code in response to alarms.
  - Work with **Auto Scaling** to adjust the number of instances based on workload.
  - Integrate with other AWS services for a fully automated response.

---

3. How CloudWatch Differs from CloudTrail and CloudFormation

- **CloudWatch**: Focuses on monitoring AWS services and applications by collecting metrics and generating alarms.
- **CloudTrail**: Tracks user activity and API usage, providing an **audit trail** of who did what and when in your AWS environment.
- **CloudFormation**: Used for provisioning AWS infrastructure using **templates** that define resources like EC2 instances, S3 buckets, or VPCs.

Quick Tip to Remember:

- **CloudWatch = Watching services** (monitoring and alerting).
- **CloudTrail = Tracking actions** (audit trails).
- **CloudFormation = Forming resources** (template-driven automation).

---

4. Working with CloudWatch Alarms and SNS

4.1 CloudWatch Alarms

Alarms monitor a specific metric and compare it against a threshold. If the threshold is breached, the alarm enters a triggered state.

Example: Monitoring CPU Utilization on an EC2 Instance

1. Set up a CloudWatch Alarm to monitor the `CPU Utilization` metric.
2. Define the threshold:
   - Example: Trigger the alarm if CPU utilization exceeds 90% for more than 5 minutes.
3. Specify the action:
   - Send a notification via SNS.
   - Trigger a Lambda function to handle the issue programmatically.
   - Initiate an Auto Scaling event.

4.2 Simple Notification Service (SNS)

SNS is a messaging service used by CloudWatch Alarms to notify stakeholders or trigger actions.

Components of SNS:

- **SNS Topic**: A logical access point and communication channel (e.g., a mailing list).
- **Subscribers**: Endpoints that receive notifications (e.g., email, SMS, Lambda function).

Example Workflow:

1. Create an SNS Topic (e.g., `HighCPUAlarmTopic`).
2. Subscribe endpoints (e.g., your email or phone number) to the topic.
3. Link the CloudWatch Alarm to the SNS Topic.
4. When the alarm is triggered, all subscribers to the topic receive notifications.

---

5. Automating Responses with CloudWatch and Auto Scaling

5.1 Auto Scaling Overview

Auto Scaling ensures that your applications have the right amount of resources to handle current traffic. By integrating Auto Scaling with CloudWatch, you can dynamically scale your infrastructure.

Example Scenario: High CPU Utilization

1. CloudWatch monitors CPU utilization across a fleet of web servers.
2. If the `CPU Utilization` metric exceeds 80% for a sustained period:
    - A CloudWatch Alarm is triggered.
    - The alarm invokes Auto Scaling to **add more EC2 instances** to the group.
3. New instances are automatically:
    - Attached to the Auto Scaling group.
    - Registered with the load balancer.
    - Begin receiving traffic without manual intervention.

5.2 Other Automation Use Cases

- **Lambda Integration**: Trigger a Lambda function to execute custom logic when an alarm is triggered.
- **Log Analysis**: Automatically analyze logs for errors or anomalies and take corrective action.

---

6. Best Practices for Using CloudWatch

1. **Leverage Tags for Metrics Filtering**: Use tags to organize and filter metrics across resources for better visibility.
2. **Set Up Alerts for Critical Metrics**: Ensure alarms are configured for vital system components like CPU, memory, and disk utilization.
3. **Implement Retention Policies**: Define how long metrics and logs should be retained to balance cost and compliance.

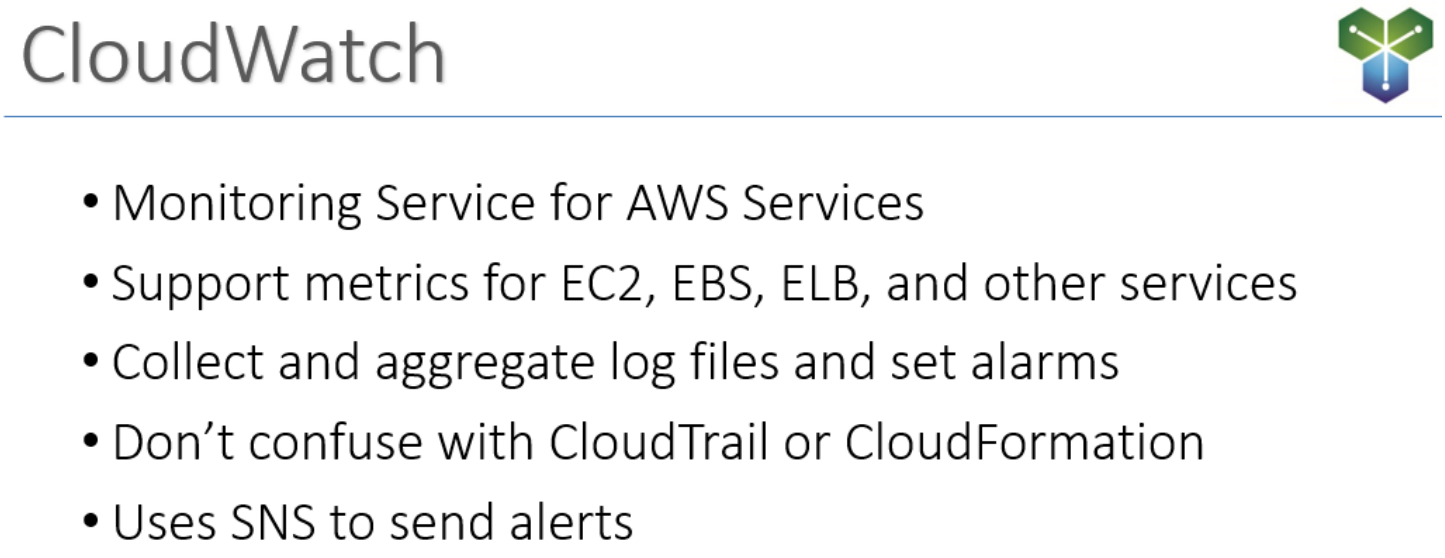
4. **Integrate with Other Services**: Combine CloudWatch with SNS, Lambda, and Auto Scaling to enable robust monitoring and automation.

---

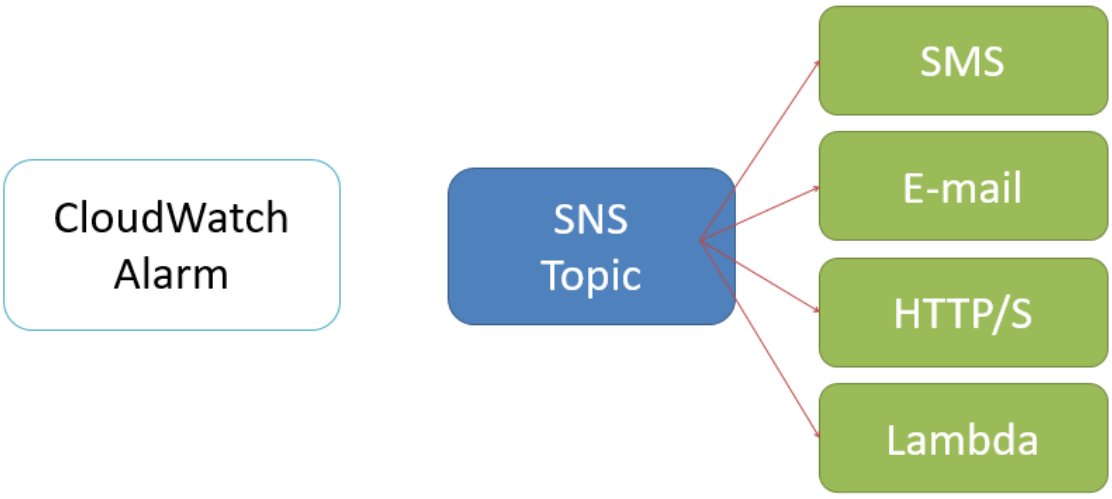
7. Summary

AWS CloudWatch is a powerful monitoring service that provides visibility into your AWS environment. It enables you to track performance metrics, collect logs, set alarms, and automate responses. By integrating with SNS, Lambda, and Auto Scaling, CloudWatch extends beyond monitoring to become a cornerstone of automation within the AWS ecosystem.

See slides below:



CloudWatch and SNS



CloudWatch and Autoscaling Groups

For more details see my full AWS Architect Associate course:
https://www.udemy.com/course/ultimateaws/?referralCode=7ED214B795C444141361

This study guide demonstrates the lesson from *Introduction to RDS*.
My full AWS Architect Associate course can be found here:
https://www.udemy.com/course/ultimateaws/?referralCode=7ED214B795C444141361

Understanding AWS Relational Database Service (RDS) Study Guide

---

1. Introduction to AWS RDS

The AWS Relational Database Service (RDS) is a **managed database service** that simplifies the setup, operation, and scaling of relational databases in the cloud. Unlike managing databases on EC2 instances, where you handle the underlying infrastructure and software manually, RDS automates many common administrative tasks such as backups, software patching, and failure recovery.

---

2. EC2 vs. RDS: A Comparison

2.1 EC2: A Self-Managed Service

- **Flexibility**: EC2 provides complete control over the instance, including:
  - Choice of operating system (Linux, Windows, etc.).
  - Installing and managing any software, including database engines.
- **Responsibilities**:
  - Manual patching of the operating system and database engine.
  - Configuring backups and disaster recovery mechanisms.
  - Monitoring system health and performing upgrades.

2.2 RDS: A Managed Service

- **Ease of Use**: RDS abstracts the underlying infrastructure and automates most administrative tasks.
- **Responsibilities Handled by AWS**:
  - Database engine updates and patches.
  - Automated backups and snapshots.
  - High availability through Multi-AZ deployments.
  - Failure detection and recovery.

Key Concept:
RDS is ideal for those who want the power of relational databases without the overhead of managing the infrastructure.

---

3. Database Engines Supported by RDS

RDS supports a variety of popular relational database engines, including:

- **MySQL**: Open-source, widely used for web applications.
- **MariaDB**: A fork of MySQL with additional features and improvements.
- **PostgreSQL**: Known for its advanced features and standards compliance.
- **Oracle**: Suitable for enterprise-grade applications.
- **Microsoft SQL Server**: Popular for Windows-based applications.
- **Amazon Aurora**: AWS's proprietary database engine, offering high performance and compatibility with MySQL and PostgreSQL.

Note: Aurora is designed to provide high availability, scalability, and durability, making it a preferred choice for many modern applications.

---

4. Automated and Manual Backups

4.1 Automated Backups

- RDS automatically performs daily backups of your database and retains them for **7 days by default** (configurable up to 35 days).
- Backups include a snapshot of the underlying **EBS volume**.
- When a database is terminated, these backups can be retained if needed.

4.2 Manual Snapshots

- A manual snapshot is a user-initiated backup that is retained until explicitly deleted.
- Useful for creating point-in-time backups before major changes or migrations.
- **Persistence**: Manual snapshots are not automatically deleted and provide long-term retention.

Key Concept:
Automated backups provide continuous protection for recent data, while manual snapshots offer flexibility for specific use cases.

---

5. Database Migration with AWS DMS

The **AWS Database Migration Service (DMS)** is a tool designed to simplify database migrations.

Key Features:

- **Migrate Existing Databases**: Move production databases to RDS with minimal downtime.

- **Cross-Engine Migrations**: Convert schemas and data from one database engine to another using the **Schema Conversion Tool**.
    - Example: Migrate an Oracle database to Amazon Aurora.
- **Continuous Replication**: Keep source and destination databases synchronized during migration.

Key Concept:
AWS DMS is particularly valuable when transitioning on-premises or EC2-hosted databases to RDS.

---

6. High Availability with Multi-AZ RDS

Multi-AZ (Availability Zone) deployment enhances the reliability and availability of your RDS instance.

How Multi-AZ Works:

- AWS creates a **primary RDS instance** in one availability zone and a **standby replica** in another.
- The standby replica is automatically updated with changes from the primary instance.
- If the primary instance fails, **DNS automatically redirects traffic** to the standby replica, ensuring minimal downtime.

Key Benefits:

- Automatic failover in case of hardware or network issues.
- Increased resilience against regional failures.

---

7. Scaling with Read Replicas

Read replicas allow you to scale **read-heavy workloads** by creating read-only copies of your database.

Key Characteristics:

- **Read-Only Copies**: Replicas cannot be used for write operations.
- **Location Options**:
    - Same availability zone.
    - Different availability zone.
    - Different AWS region (cross-region replication).
- **Workload Offloading**: Direct read queries to replicas while keeping write queries on the primary database.

Limitations:

- Read replicas do not provide high availability (no automatic failover).
- Primarily used for **performance improvement**, not disaster recovery.

Key Concept:
Use read replicas to handle increased read traffic, reducing the load on the primary database.

---

8. RDS Features Summary

Feature	Description
Managed Service	Automates backups, updates, patching, and failure recovery.
Backup Options	Includes automated daily backups and user-initiated manual snapshots.
Multi-AZ	Provides high availability by maintaining a standby replica in a separate AZ.
Read Replicas	Scales read-heavy workloads by creating read-only copies of the database.
Supported Engines	MySQL, MariaDB, PostgreSQL, Oracle, Microsoft SQL Server, and Amazon Aurora.
Database Migration	AWS DMS enables seamless migrations with minimal downtime.

---

9. Use Cases for RDS

1. **Web Applications**: Host relational databases for dynamic websites and content management systems.
2. **Enterprise Applications**: Use Oracle or SQL Server for ERP or CRM systems.
3. **Data Migration**: Migrate on-premises databases to RDS using DMS for scalability and reduced maintenance.
4. **Scaling**: Use read replicas to handle increased workloads without modifying the primary database.

---

10. Best Practices for Using RDS

- **Enable Multi-AZ** for production databases to ensure high availability and disaster recovery.
- Use **automated backups** to maintain data integrity and **manual snapshots** for critical events.
- **Monitor performance** using Amazon CloudWatch metrics for RDS, such as CPU utilization and read/write IOPS.
- **Optimize costs** by selecting the right instance type and enabling **Reserved Instances** for predictable workloads.
- **Use IAM roles and security groups** to restrict database access and enhance security.

---

11. Summary

AWS RDS is a robust and scalable solution for relational database management in the cloud. By automating administrative tasks, providing high availability options, and offering seamless integration with other AWS services, RDS allows developers and administrators to focus on their applications instead of managing the underlying infrastructure. Whether you need to migrate existing databases, ensure high availability, or scale out read-heavy workloads, RDS provides the tools and flexibility to meet your needs.

See slides below:

Relational Database Service (RDS)

EC2 Instance
T3.small

O.S.
Applications

RDS Instance
db.T3.small

Database Engine
~~O.S.~~

RDS Automated Backups

- Volume level snapshot
- You can specify the retention policy (default is 7 days)
- When you terminate an instance the backups can be retained

Relational Database Service (RDS)

- Industry standard relational database
- Common database administration tasks are performed automatically
- Supports MySQL, MariaDB, PostgresQL, Oracle, SQL, and Aurora.
- Shell access unavailable because it's a managed service
- AWS provides backups, software patching, failure detection, and recovery

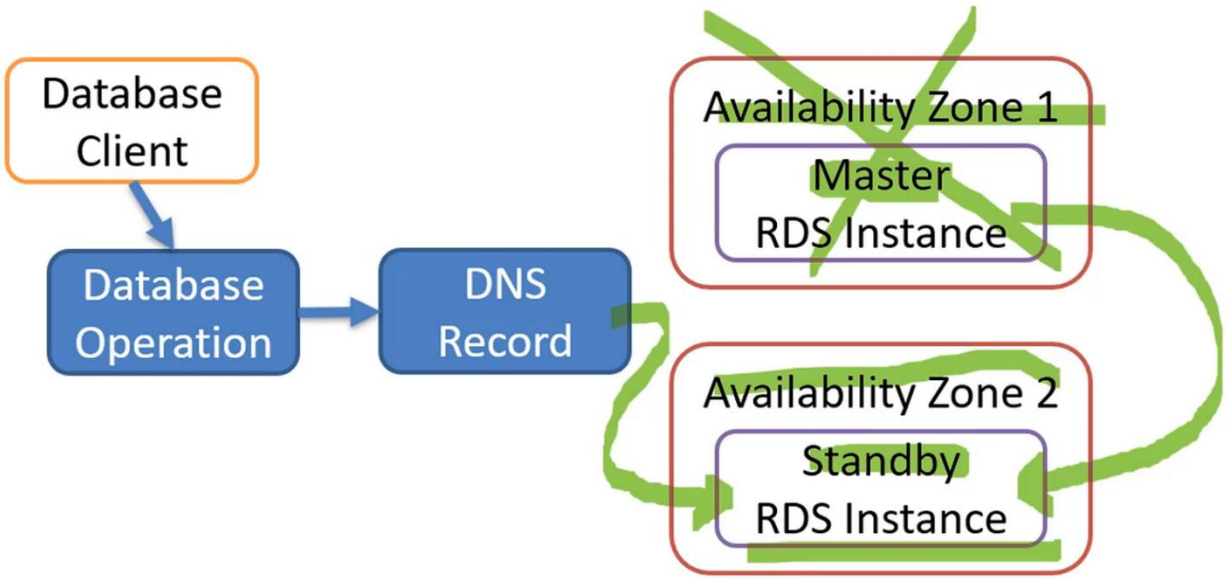
RDS Snapshots

- Can be used to take a manual snapshot at any time
- Backs up the entire instance
- Kept until you manually delete them

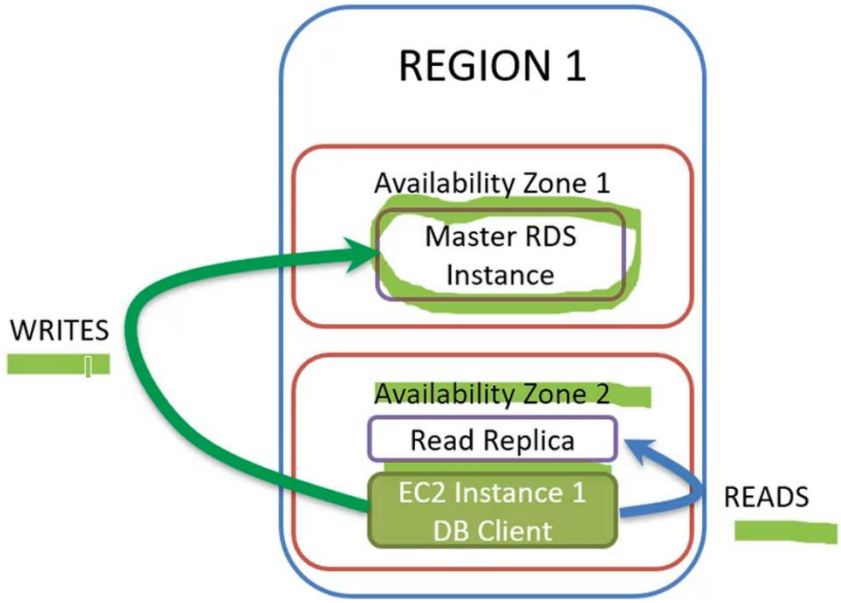
Database Migration Service

- Migrate a production DB onto AWS
- Ensures that any changes while data is in-flight are consistent
- Allows you to convert the source database schema automatically
- Example – Oracle database can be converted to an open source database like Aurora

Multi-AZ RDS Diagram



RDS Read Replica Diagram



For more details see my full AWS Architect Associate course:
https://www.udemy.com/course/ultimateaws/?referralCode=7ED214B795C444141361

This lab demonstrates the steps from *RDS Demo.*
My full AWS Architect Associate course can be found here:
https://www.udemy.com/course/ultimateaws/?referralCode=7ED214B795C444141361

Lab Guide: Creating an RDS Instance Using Easy Create in AWS

Objective

This lab will guide you through the basics of creating an Amazon RDS instance using the **Easy Create** option in the AWS Management Console. You will also learn key concepts about RDS as a managed relational database service.

---

1. Accessing RDS in the AWS Console

1. **Log in to the AWS Management Console.**
2. In the **search bar** at the top of the page, type "RDS" and select **RDS** from the results.
3. On the RDS Dashboard, click **"DB Instances"** in the left-hand menu.
4. Click the **"Create Database"** button to start the process.

---

2. Understanding RDS and Managed Databases

Key Concepts

- **Relational Database Service (RDS)**:
  - AWS RDS is a managed database service that automates administrative tasks like backups, patching, and failure recovery.
  - Unlike EC2, where you manage the OS and database engine, RDS is a fully managed solution where AWS handles these tasks for you.
- **Managed Service Benefits**:
  - No access to the underlying operating system.
  - Automated maintenance and backups.
  - Simplified setup with preconfigured options.

---

3. Using the Easy Create Option

Steps to Create an RDS Instance:

1. On the **Create Database** page, select **Easy create** for simplicity.
2. **Choose a Database Engine**:
    - Select **MySQL Community Edition** for this lab.
    - Note: Open-source engines like MySQL and PostgreSQL are more cost-effective than commercial engines like Oracle or Microsoft SQL (due to licensing fees).
    - Alternatively, AWS Aurora (a high-performance, proprietary engine) is available for advanced use cases.
3. **Select a Database Type**:
    - **Free Tier**: For demonstration purposes, select Free Tier to minimize costs. This is suitable for low-resource, non-production environments.
4. **Database Identifier**:
    - Enter a name for your database instance (e.g., `RickDemoRDS`).
5. **Set Master Credentials**:
    - Specify a **Master Username** (e.g., `admin`).
    - Choose **"Auto-generate password"** to let AWS create a strong password for you.
    - Note: The password will be available for viewing after the RDS instance is created.

---

4. Configuring Connectivity

1. **Select EC2 Instance Connectivity**:
    - If you plan to connect this RDS instance to an EC2 instance:
        - Select the EC2 instance from the dropdown to allow AWS to configure the necessary **security group** and **firewall rules** automatically.
    - For this lab, choose **"Don't connect to an EC2 compute resource"**.
2. **Key Settings (Managed by AWS with Easy Create)**:
    - **VPC and Subnet Group**: RDS automatically selects the VPC and subnets.
    - **Encryption**: Enabled by default for data security.
    - **Backups**: Automated backups are enabled, providing point-in-time recovery.
    - **Security Groups**: AWS configures rules to control inbound/outbound traffic.

Key Concept:
Using **Easy Create** simplifies setup by preconfiguring settings. For more control, the **Standard Create** option can be used.

---

5. Creating the RDS Instance

1. Review the settings and click **"Create Database"**.
2. AWS will begin provisioning the RDS instance. This process may take several minutes.
3. Once the instance is ready, click on the **database identifier** (e.g., `RickDemoRDS`) to view its details.

---

6. Viewing and Using the Database Endpoint

1. After the RDS instance is created, locate the **Endpoint** in the instance details. The endpoint is the address used by database clients (e.g., application servers or web servers) to connect to the database.
2. Copy the endpoint address and store it for later use when configuring your database clients.

---

7. Key Features of RDS Instances

- **No OS Access**:
  - Unlike EC2 instances, you cannot access the operating system of an RDS instance. This restriction ensures AWS handles OS-level management.
- **Automated Backups**:
  - RDS automatically creates backups and retains them for the default period (7 days for Free Tier).
  - Backups include a snapshot of the EBS volume to allow point-in-time recovery.
- **Maintenance and Updates**:
  - AWS handles database engine upgrades, patches, and security updates automatically.
- **Connectivity and Security**:
  - Security groups control access to the RDS instance.
  - Only authorized clients (e.g., EC2 instances or on-premises applications) can connect.

---

8. Practical Use Case: Connecting to the RDS Instance

Steps to Connect:

1. Use any database client tool (e.g., MySQL Workbench, pgAdmin) to connect to the RDS instance.
2. Configure the client with:
   - **Endpoint**: The RDS instance's endpoint address.
   - **Port**: Default port for the selected database engine (e.g., 3306 for MySQL).
   - **Master Username and Password**: Credentials created during setup.

---

9. Advantages of Managed RDS

- **Simplified Setup**: Easy Create reduces the complexity of configuring a database.
- **Reduced Maintenance**: AWS automates backups, updates, and security patches.
- **High Availability**: Features like Multi-AZ deployments ensure redundancy and failover support.
- **Scalability**: Easily adjust resources to meet growing workload demands.

---

10. Summary

In this lab, you learned how to:

1. Navigate the AWS console and access the RDS dashboard.
2. Create a managed RDS instance using the **Easy Create** option.

3. Configure basic settings, such as selecting a database engine, connectivity options, and instance type.
4. Understand the key benefits and features of RDS as a managed service.

RDS simplifies the management of relational databases by automating many traditional tasks, allowing you to focus on building applications instead of maintaining infrastructure.

This study guide demonstrates the lesson from *Introduction to ECS – Run Containers in the AWS Cloud.*

My full AWS Architect Associate course can be found here:

https://www.udemy.com/course/ultimateaws/?referralCode=7ED214B795C444141361

Running Docker Containers in AWS Environment Study Guide

Introduction to Docker Containers in AWS

Running Docker containers within the AWS environment introduces an efficient, scalable way to deploy applications using microservices. Containers provide a lightweight alternative to traditional VMs, encapsulating applications with their dependencies and configurations in self-contained environments.

Container Basics

- **Container Images**: These are blueprints for containers, containing the application code, dependencies, and necessary configurations. Container images are stored in a container registry, a library where images are housed until needed for deployment.
- **Container Hosts**: Containers run on container hosts, which can be either physical servers, virtual machines, or cloud instances like AWS EC2 instances. AWS also offers Fargate, a serverless option for running containers without managing the underlying EC2 instances.

Container Orchestration with ECS

Container orchestration is crucial for managing multiple containers' deployment, scaling, and failure handling. AWS provides the Elastic Container Service (ECS) as a container orchestration tool to automate these tasks, ensuring efficient container management across EC2 instances or with Fargate.

Elastic Container Registry (ECR)

AWS's ECR is a managed Docker container registry service that allows developers to store, manage, and deploy Docker container images. It is integrated with ECS, simplifying the development to production workflow for containerized applications.

ECS Clusters and Auto Scaling

- **ECS Cluster Setup**: An ECS cluster is a collection of container hosts where containers are deployed. In EC2 mode, you're responsible for the operating system and instance management, including patching and scaling.
- **Auto Scaling Groups**: ECS can integrate with AWS Auto Scaling to automatically adjust the number of EC2 instances in the cluster based on demand, optimizing resource utilization and cost.

Managing EC2 Instances for ECS

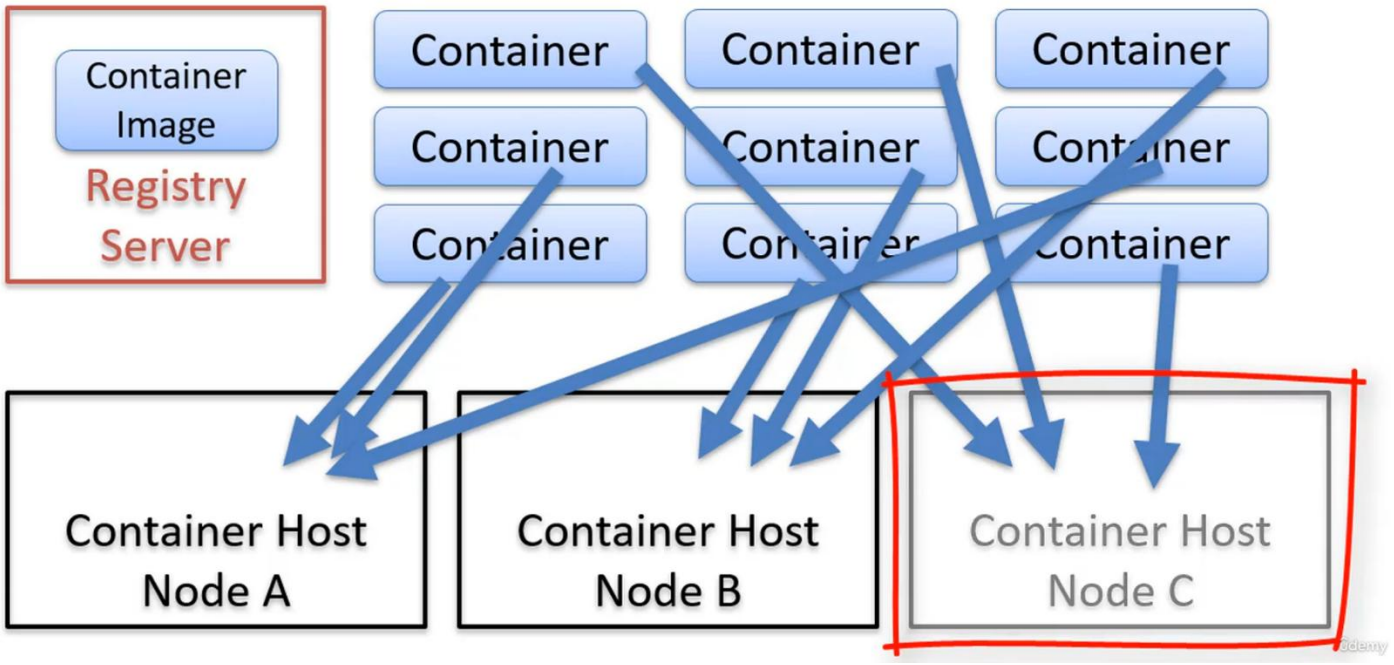
While ECS handles container orchestration, managing the underlying EC2 instances involves overseeing the operating system, applying patches, and updating AMIs. AWS offers options to automate these tasks, reducing the operational burden on developers.

Key Takeaways

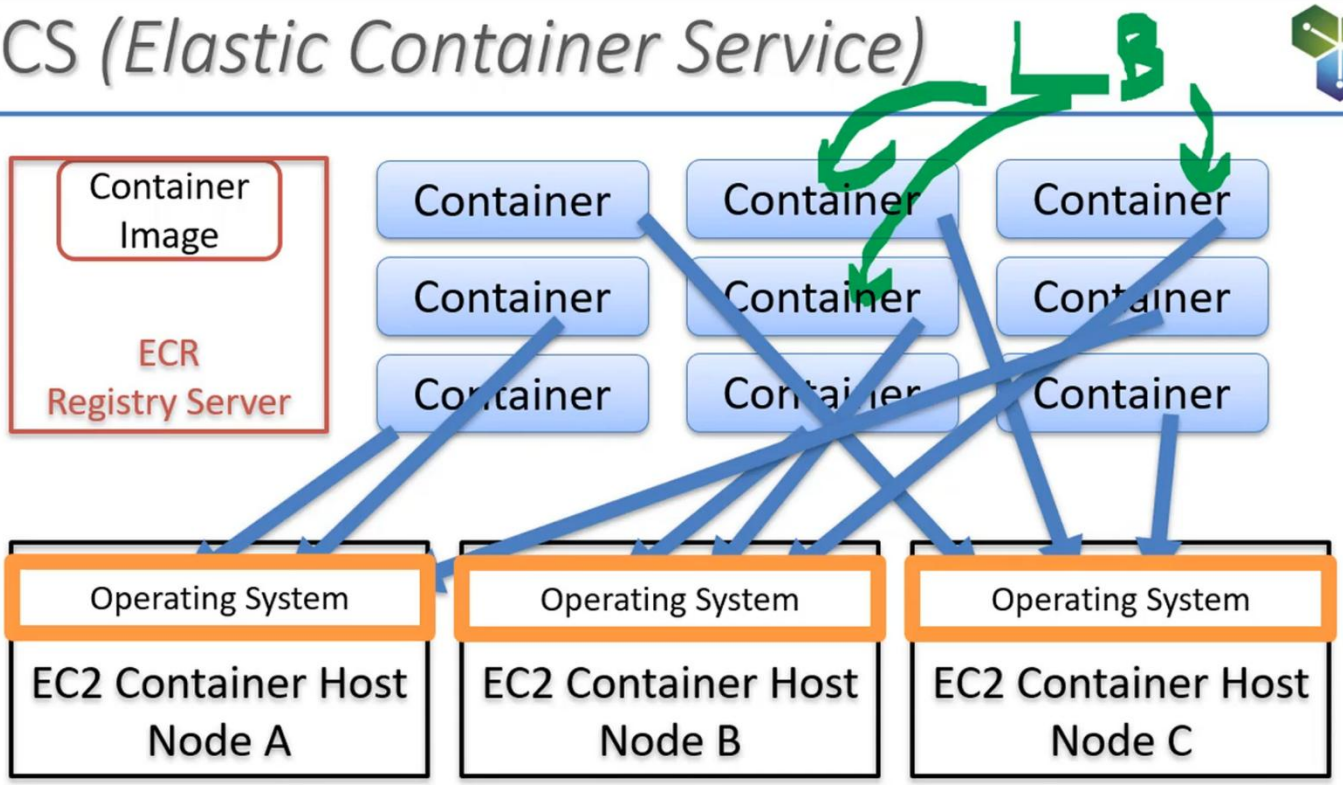
- Docker containers offer a scalable and efficient way to deploy applications in AWS.
- ECS provides robust orchestration capabilities, simplifying container deployment, scaling, and management.
- ECR serves as a centralized repository for container images, facilitating easy storage and retrieval.
- Auto Scaling ensures that the ECS cluster scales according to demand, improving resource efficiency.
- Managing EC2 instances for ECS requires attention to operating system maintenance and security, although AWS provides tools and services to alleviate some of these responsibilities.

See slides below:

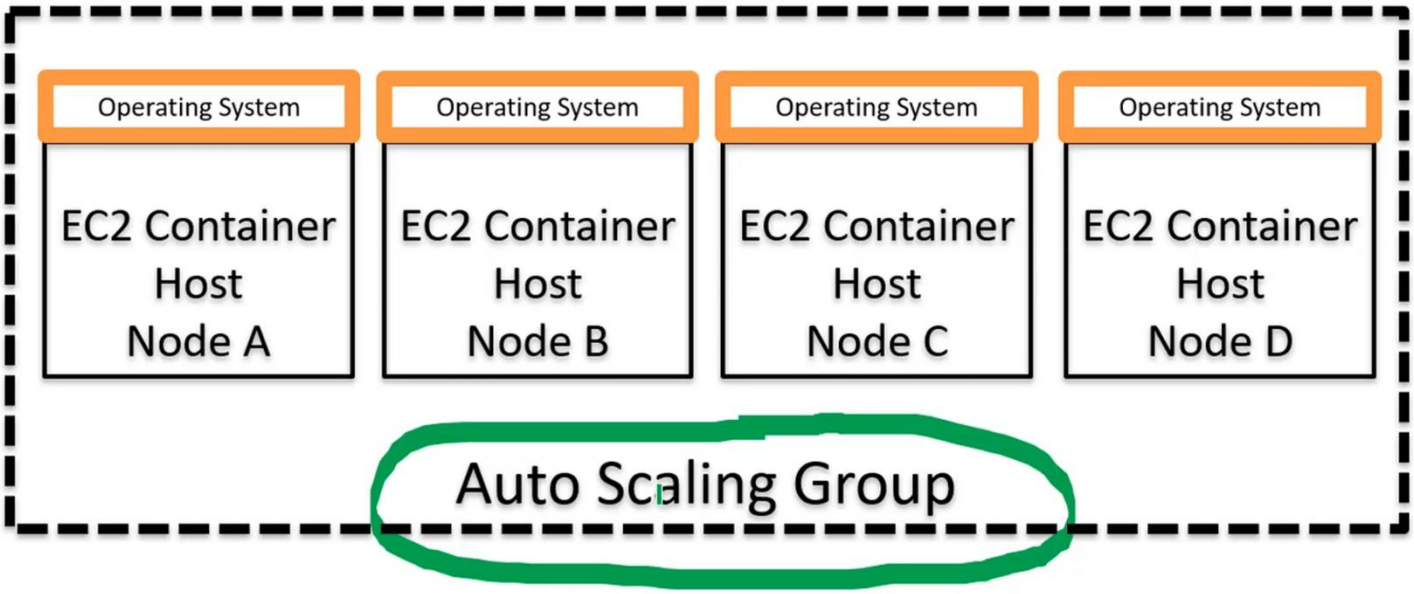
Container Orchestration



ECS *(Elastic Container Service)*



ECS Cluster



For more details see my full AWS Architect Associate course:
https://www.udemy.com/course/ultimateaws/?referralCode=7ED214B795C444141361