**Compute Services Design**

**EC2 overview**

AWS compute services with a heavy emphasis on Amazon EC2. Here are some key points that might help clarify the concepts:

* **Amazon EC2 Overview**: Amazon EC2 (Elastic Compute Cloud) provides scalable computing capacity in the AWS cloud. It allows you to launch virtual servers (instances) as needed.
* **Instance Types**: EC2 offers various instance types optimized for different use cases, such as compute-optimized, memory-optimized, and storage-optimized instances.
* **Pricing**: EC2 pricing can vary based on factors like instance type, region, and usage. Options include On-Demand, Reserved Instances, and Spot Instances.
* **EBS and EC2**: Amazon Elastic Block Store (EBS) provides persistent block storage for EC2 instances, allowing data to persist even after the instance is terminated.
* **AWS Compute Optimizer**: This tool helps you choose the optimal EC2 instance types based on your usage patterns to improve performance and reduce costs.

**EC2 instance types**

1. **General Purpose Instances**:
   * **Types**: T2, M5, M4, M3
   * **Use**: Balanced resources for general tasks like web hosting or small databases.
   * **Special Note**: T2 instances offer burst performance for occasional high loads.
2. **Compute Optimized Instances**:
   * **Types**: C5, C4, C3
   * **Use**: CPU-intensive tasks like media encoding, batch processing, and gaming servers.
3. **Memory Optimized Instances**:
   * **Types**: X1e, X1, R4, R3
   * **Use**: High memory requirements such as large data sets, in-memory databases, and big data processing.
4. **Storage Optimized Instances**:
   * **Types**: H1, I3, D2
   * **Use**: High sequential read/write operations, like relational databases and data warehousing.
5. **Advanced Computing Instances**:
   * **Types**: P3, P2, G3, F1
   * **Use**: Specialized hardware needs, such as GPU for graphics processing or FPGA for custom hardware configurations.

Each instance type is designed to meet specific needs, so choosing the right one depends on your workload requirements. Remember, you can always change the instance type later if your needs evolve.

**EC2 pricing**

* **On-Demand Pricing**: You pay for compute capacity by the second with no long-term commitments. This is ideal for short-term, irregular workloads that cannot be interrupted.
* **Reserved Instances**: You commit to using EC2 for a one or three-year term and receive a significant discount compared to on-demand pricing. This is best for steady-state or predictable usage.
* **Spot Instances**: You bid for unused EC2 capacity, which can save you up to 90% off the on-demand price. However, these instances can be terminated by AWS with little notice, so they're suitable for flexible, fault-tolerant applications.
* **Additional Costs**: Remember to factor in costs for storage, data transfer, and any additional services like databases or networking.

**EBS and EC2**

* **Elastic Block Store (EBS)**: Think of EBS as a hard drive for your EC2 instance. It provides persistent storage, meaning any changes you make are saved even if you shut down the instance. This is useful for storing operating systems and data that need to be retained.
* **Persistent Block Storage**: This means that the data on the EBS remains intact and unchanged even when the instance is stopped or restarted. It's like saving your work on a computer so you can pick up where you left off.
* **EBS-Optimized Instances**: These are specific types of EC2 instances that are designed to provide the best performance when using EBS. They ensure that the storage performs well, especially if you're using SSD (Solid State Drives) for faster data access.
* **Instance Types and IOPS**: Different instance types support different levels of IOPS (Input/Output Operations Per Second). Higher IOPS means better performance for tasks requiring fast data access. For example, a c4.large instance supports up to 4,000 IOPS, while a c4.8xlarge can go up to 32,000 IOPS.
* **Non-EBS Storage**: If you use storage like S3 (Simple Storage Service) instead of EBS, changes are not saved when the instance is stopped. This might be suitable for scenarios where you don't need to retain changes, like certain network appliances.
* **Free Tier Instances**: The T instances, which are part of the free tier, do not support EBS optimization. This means if you need EBS optimization, you'll have to choose a paid instance type.

Top of Form

**AWS Compute Optimizer**

* **AWS Compute Optimizer**: This tool helps you identify the best AWS compute resources for your workloads. It makes recommendations to improve performance and, in some cases, reduce costs.
* **Setup Requirements**:
  + You need to enable CloudWatch to collect utilization data.
  + At least 30 consecutive hours of CloudWatch data is required for EC2, EC2 Auto Scaling Groups, and EBS to make recommendations.
  + For Lambda functions, it needs at least 50 invocations in the last 14 days.
* **Supported Resources**: Compute Optimizer provides recommendations for:
  + EC2 instances
  + EC2 Auto Scaling Groups
  + Elastic Block Store (EBS)
  + Lambda functions
* **Recommendations**: These are stored in S3 and integrated with Cost Explorer and Systems Manager to help you make informed decisions about optimizing your AWS resources.
* **Performance vs. Cost**: Recommendations may not always be cheaper but are aimed at improving performance, such as better configuration for EBS to handle high IOPS (Input/Output Operations Per Second).

**Compute Services ImplementationTop of Form**

**Launching an EC2 Linux instance lab**

1. **AWS Management Console**: You start by navigating to the EC2 service in the AWS Management Console.
2. **Choosing an AMI (Amazon Machine Image)**: An AMI is a pre-configured template for your instance, including the operating system and other software. For this lab, you select a Linux AMI.
3. **Instance Type**: You choose the type of instance based on your needs. The instance type determines the hardware of the host computer used for your instance.
4. **Configuration**:
   * **Number of Instances**: You can launch multiple instances at once.
   * **Network Settings**: Choose the network and subnet where your instance will run.
   * **Auto-assign Public IP**: Decide if your instance should have a public IP address.
   * **Shutdown Behavior**: Choose what happens when the instance is shut down (stop or terminate).
5. **Storage**: Specify the storage type and size for your instance. You can choose between SSD and magnetic storage.
6. **Tags**: Add tags to your instance for easier management and identification.
7. **Security Groups**: Configure security groups to control the traffic to and from your instance. By default, SSH access is allowed for Linux instances.
8. **Key Pair**: Create or select a key pair to securely connect to your instance via SSH.

Once these steps are completed, you launch the instance, and it will be up and running shortly. The next step involves connecting to the instance to configure it further.

Top of Form

**Configuring an EC2 Linux instance lab**

1. **Connecting to the Instance**:
   * Use the EC2 dashboard to find your running instance.
   * Note the public DNS or IP address for connectivity.
   * Use the key pair (PEM file) you downloaded when launching the instance.
2. **Using SSH to Connect**:
   * On Linux: Use the PEM file directly with the SSH client.
   * On Windows: Convert the PEM file to a PPK file using PuTTYgen, then use PuTTY to connect.
3. **Logging In**:
   * Use the default user EC2-user to log in.
   * Once logged in, you can execute standard Linux commands.
4. **Managing the Instance**:
   * You can install software, update the system, and manage services as you would on any Linux machine.

These steps ensure you can connect to and manage your EC2 Linux instance effectively.

Top of Form

Bottom of Form

**Setting up an EC2 Windows instance lab**

1. **Launching a Windows Instance**:
   * You start by selecting "Launch Instance" in the EC2 dashboard.
   * Choose a Windows Server AMI (Amazon Machine Image) from the available options (e.g., Windows Server 2012 R2).
   * Select the instance type, typically sticking with the free tier for basic setups.
2. **Configuring Instance Settings**:
   * Configure parameters like VPC (Virtual Private Cloud), subnet, and IP address. The video suggests accepting the default settings for simplicity.
   * Add storage, choosing General Purpose SSD for better performance with Windows.
3. **Adding Tags and Security Groups**:
   * Add tags to categorize your instance (e.g., Department: Production).
   * Configure security groups to allow RDP (Remote Desktop Protocol) access, which is essential for managing the Windows instance.
4. **Review and Launch**:
   * Review the configuration and launch the instance.
   * Use an existing key pair for authentication. This key pair is crucial for accessing the instance.
5. **Post-Launch Configuration**:
   * Once the instance is running, you can view and manage it through the EC2 Management Console.
   * Configure security groups and other settings as needed.

This process allows you to quickly set up and manage a Windows Server in AWS, making it accessible for remote management and configuration.

**Shared tenancy**

* **Shared Tenancy**: This is the default setting when you launch an EC2 instance. Multiple customers share the same physical hardware, which helps reduce costs and simplifies deployment.
* **Pros of Shared Tenancy**:
  + **Reduced Cost**: It's cheaper than dedicated hosts or instances.
  + **Simpler Deployment**: Amazon takes care of the physical deployment, so you only need to choose the availability zone.
* **Cons of Shared Tenancy**:
  + **Lower Performance**: Since you share resources with others, performance might not be as high as with dedicated options.
  + **Less Control**: You have less control over the physical host, which might be an issue for regulatory compliance or specific performance needs.
* **Default Setting**: Shared tenancy is the default setting for EC2 instances because it offers a good balance of cost and performance for most users.
* **Regulatory and Performance Considerations**: If you have specific regulatory or performance requirements, you might need to opt for dedicated hosts or instances instead of shared tenancy.

**Dedicated hosts**

* **Dedicated Hosts**: These are physical servers dedicated to a single customer. Unlike shared tenancy, you have control over the physical server your instances run on.
* **Benefits**:
  + **Licensing Management**: Easier to manage software licenses that are tied to specific hardware.
  + **Compliance**: Helps meet regulatory requirements that demand control over physical hardware.
  + **Consistent Performance**: Ensures your instances always run on the same physical server.
* **Drawbacks**:
  + **Higher Cost**: More expensive than shared tenancy because you are paying for exclusive use of the hardware.
* **Use Cases**:
  + **Bring Your Own License (BYOL)**: Allows you to use existing software licenses, reducing costs.
  + **Compliance Needs**: Necessary for industries with strict regulatory requirements.
* **Configuration**: Dedicated hosts must be explicitly configured and are not available in the free tier.

This video emphasizes the importance of dedicated hosts for specific use cases where control, compliance, and licensing management are critical.

**Dedicated instances**

* **Dedicated Instances**: These run on hardware dedicated to a single customer, ensuring no other AWS account shares the same physical machine.
* **Differences from Dedicated Hosts**:
  + **Instance Placement**: Dedicated instances can move between physical machines on restart, whereas dedicated hosts remain on the same physical machine.
  + **Cost**: Both require explicit configuration and are not available in the free tier, but dedicated instances are generally less expensive than dedicated hosts.
* **Pros**:
  + **Performance**: Improved performance as the hardware is dedicated to you, preventing other customers from consuming resources.
  + **Compliance**: Helps meet certain regulatory requirements without the higher cost of dedicated hosts.
* **Cons**:
  + **Licensing Management**: Less control over licensing compared to dedicated hosts since instances can move between physical machines.
  + **Placement Control**: You cannot determine the exact physical machine your instance will run on after a restart.
* **Use Cases**:
  + **Regulatory Compliance**: Useful for meeting specific compliance needs without the higher cost of dedicated hosts.
  + **Performance Needs**: Suitable for applications requiring consistent performance without interference from other customers.

Top of Form

**AMI virtualization**

* **Amazon Machine Image (AMI)**: An AMI is essentially a blueprint for your server. It includes the configuration details and the operating system needed to launch an instance. Think of it as a template that you can use to create new servers quickly.
* **Instance**: When you launch an instance, you're creating a virtual server based on an AMI. It's like making a copy of the blueprint and bringing it to life in the AWS cloud.
* **Sources of AMIs**:
  + **Amazon**: Provides free and paid AMIs.
  + **Amazon Marketplace**: Offers AMIs from various vendors, some free and some paid.
  + **Community AMIs**: Created and shared by other users, usually free.
* **Types of AMIs**:
  + **Hardware Virtual Machine (HVM)**: Fully virtualizes the hardware and requires hardware-assisted virtualization. This type offers better performance.
  + **Paravirtual AMIs (PV)**: Do not require specific hardware support but may not perform as well as HVM.
* **Root Volume**: This is the main storage volume containing the boot sector and boot loader, essential for starting the instance. It can be stored in:
  + **Instance Store**: Temporary storage that is lost if the instance fails.
  + **Elastic Block Store (EBS)**: Persistent storage that retains data even if the instance fails.
* **Launching and Managing AMIs**: You can create your own AMIs by customizing existing ones or importing virtual machines. You can also control who can use your AMIs by setting permissions (public, explicit, or owner-only).

**Compute Services ManagementTop of Form**

**Instance management**

* **Bootstrapping**: This means adding extra code to run when an instance starts. For example, you can automatically update software or install applications when the instance launches, saving you from doing it manually each time.
* **Virtual Machine Import and Export**: You can import an existing virtual machine into AWS EC2 and use it as an instance. This helps you quickly get an instance up and running without starting from scratch.
* **Metadata**: This is extra information about your instance, like its ID, type, and security groups. You can also create custom metadata using tags to help organize and manage your instances better.
* **Tags**: Tags are labels you add to instances to help identify and manage them. For example, you can tag an instance with "SQL Server" to indicate its purpose. Tags are useful for organization and can be customized as needed.
* **Changing Instance Type**: You can change the type of an instance (e.g., from a T2 to a C4) by stopping it, changing the type, and then starting it again. This allows you to adjust the instance's capabilities as needed.
* **Security Groups**: These act like firewalls for your instances, controlling the traffic that can reach them. You can change security groups on the fly to adjust security settings as needed.
* **Termination Protection**: This feature prevents accidental termination of instances. If enabled, it requires you to disable termination protection before you can terminate the instance, adding an extra layer of safety.

These concepts help you manage your instances more efficiently and ensure they meet your needs without manual intervention each time.

**Connecting to instance labs**

1. **Connecting to a Windows Instance using RDP**:
   * **RDP (Remote Desktop Protocol)**: This is a protocol used to connect to Windows instances in AWS, allowing you to manage the instance as if you were physically present at the server.
   * **Steps to Connect**:
     + Go to the EC2 dashboard and select your Windows instance.
     + Right-click on the instance and choose "Connect" to get the connection information.
     + Download the Remote Desktop file and save it to your system.
     + Retrieve the administrator password using the PEM file you created when launching the instance.
     + Use the downloaded RDP file and the retrieved password to connect to the instance.
2. **Importance of Protecting PEM Files**:
   * **PEM File**: This file contains your private key, which is crucial for accessing your AWS instances.
   * **Security**: Ensure that your PEM file is stored securely and not left on your desktop. If someone gains access to this file, they can potentially access your AWS instances.
3. **Naming Conventions**:
   * Naming your instances meaningfully (e.g., Windows 13, Linux 45) helps in easily identifying them in the EC2 dashboard.

By following these steps and understanding the importance of securing your PEM file, you can effectively manage and connect to your AWS instances.

**Working with security groups**

1. **Security Groups Basics**:
   * Security groups act like a firewall for your instances, controlling both inbound and outbound traffic.
   * They are stateful, meaning if you allow an incoming request, the response is automatically allowed back out.
2. **Rules and Constraints**:
   * Security groups only support "allow" rules, not "deny" rules.
   * You can attach up to five security groups to an instance.
   * Each security group can have separate rules for inbound and outbound traffic.
3. **Differences Between Security Groups and Network ACLs**:
   * Security groups operate at the instance level, while Network ACLs operate at the subnet level.
   * Network ACLs are stateless, meaning both inbound and outbound rules must be explicitly defined.
4. **Practical Use**:
   * You can layer security groups to manage different types of traffic (e.g., one for web servers, another for database servers).
   * Instances receive a default security group, which can be changed or removed.
5. **Important Exam Points**:
   * Remember that security groups are stateful and only allow rules.
   * Network ACLs allow both "allow" and "deny" rules and are stateless.

**Working with security group labs**

* **Security Groups Overview**: Security groups act like a firewall for your instances, controlling inbound and outbound traffic. Each security group can have multiple rules that allow specific types of traffic.
* **Default Security Groups**: When you create a Virtual Private Cloud (VPC), it comes with a default security group. Each instance in the VPC is automatically associated with this default security group unless you specify otherwise.
* **Creating a Security Group**: You can create a new security group from scratch by specifying a name, description, and rules. For example, you might create a security group named "WEBSEC" to allow HTTP, HTTPS, and SMTP traffic.
* **Inbound and Outbound Rules**: Inbound rules specify the traffic allowed to enter the instance, while outbound rules specify the traffic allowed to leave. By default, all outbound traffic is allowed, but you can customize these rules.
* **Attaching Security Groups to Instances**: Once a security group is created, you can attach it to an instance. This allows the instance to follow the rules defined in the security group.
* **Implicit Deny**: Any traffic not explicitly allowed by the security group's rules is denied by default. This ensures that only the specified traffic can reach your instances.
* **Managing Multiple Security Groups**: You can attach up to five security groups to a single instance. The rules from all attached security groups are combined, allowing traffic that any of the groups permit.

These concepts are crucial for effectively managing the security of your AWS instances.

**Advanced EC2 management**

* **Resource Optimization Recommendations**: AWS provides recommendations to help you optimize your EC2 instances for cost and performance. By following these recommendations, you can potentially save money and improve efficiency. For example, AWS might suggest resizing instances or using reserved instances to reduce costs.
* **Host Recovery**: This feature automatically restarts your EC2 instances if a failure is detected. AWS uses tools like CloudWatch to monitor the health of your instances. If an instance fails, Host Recovery can launch a replacement instance with the same configuration, ensuring minimal downtime.
* **Traffic Mirroring**: This allows you to copy network traffic from an EC2 instance's network interface and send it to another destination for analysis. This is useful for tasks like content inspection, threat monitoring, and troubleshooting network issues. It helps you understand and secure your network traffic by analyzing the data flow in and out of your instances.

These concepts are essential for managing and optimizing your EC2 instances effectively.

Top of Form

**AWS Batch**

* **AWS Batch Overview**: AWS Batch is a service that enables you to run batch computing workloads on AWS. It allows you to efficiently run a collection of processes or scripts as a unit.
* **Batch Jobs**: These are the individual tasks or scripts you want to run. They can be anything from Linux Bash shell scripts to Python scripts.
* **Job Queues**: Jobs are placed in job queues, which manage the order and timing of job execution. You can configure these queues based on your needs.
* **Compute Environment**: This is the environment where your jobs run. AWS Batch uses a managed compute environment, meaning you don't have to manually set up EC2 instances. You can choose between on-demand or spot instances based on cost and performance requirements.
* **Job Execution**: You define jobs, configure the compute environment, and set up job queues. AWS Batch handles the execution, scaling, and management of the jobs.
* **Use Cases**: AWS Batch can be used for automating management tasks, running custom application scripts, or executing AWS CLI commands to manage AWS resources.

These concepts should help you understand how AWS Batch works and how it can be used to manage and automate batch processing tasks in AWS.

**Elastic Container Service (ECS)**

* **Elastic Container Service (ECS)**: ECS is a service provided by AWS to run Docker containers. It simplifies the deployment and management of containerized applications.
* **Docker Containers**: These are standardized units of software that package up code and all its dependencies, making it easy to transfer and run applications across different environments.
* **Amazon Fargate**: This is a serverless compute engine for containers that works with ECS. It allows you to run containers without having to manage the underlying infrastructure.
* **Multi-Tier Applications**: ECS supports running multi-tier applications by allowing you to break your application into different containers (e.g., web server, application server, database) and manage them separately.
* **Scalability**: ECS makes it easy to scale your applications by distributing containers across multiple instances and managing the load.
* **ECS Management Console**: This is the interface where you can manage your ECS resources, including clusters, task definitions, and services.

These concepts are crucial for understanding how to deploy and manage containerized applications using ECS in AWS.

**Elastic Beanstalk environment**

* **Elastic Beanstalk**: This is a service provided by AWS that allows you to deploy and manage applications in the AWS cloud without worrying about the infrastructure. The name "Beanstalk" comes from the idea that, like Jack's beanstalk, a small effort (a bean) can grow into something big (a beanstalk).
* **Elasticity**: This refers to the ability of the service to automatically scale your application up or down based on demand. This means your application can handle varying levels of traffic without manual intervention.
* **Ease of Use**: With Elastic Beanstalk, you can quickly deploy applications using various programming languages and platforms like PHP, .NET, Java, Node.js, Ruby, Python, and Tomcat. You simply upload your code, and Elastic Beanstalk handles the deployment, from capacity provisioning, load balancing, and auto-scaling to application health monitoring.
* **Infrastructure Management**: Elastic Beanstalk creates an environment for your application, which includes EC2 instances, security groups, load balancers, and more. It automates the setup, making it easier for you to focus on writing code rather than managing infrastructure.
* **Customizability**: While Elastic Beanstalk manages the infrastructure, you still have the flexibility to customize your environment. You can choose instance types, configure more options, and even upload your own application code or use sample applications to get started.

This service is particularly useful for developers who want to deploy web applications quickly and efficiently without getting bogged down by the complexities of infrastructure management.Top of Form

Bottom of Form

Top of Form

Bottom of Form

Bottom of Form

Top of Form

Bottom of Form

Top of Form

Bottom of Form

Top of Form

Bottom of Form

Bottom of Form

Bottom of Form

Top of Form

Bottom of Form

Top of Form

Bottom of Form

Top of Form

Bottom of Form

Bottom of Form

**Bottom of Form**

**Bottom of Form**

**Top of Form**

**Bottom of Form**

**Top of Form**

**Bottom of Form**