Birla Institute of Technology & Science, Pilani
Work Integrated Learning Programmes Division
First Semester 2024-2025

Mid-Semester Test
(EC-2 Make-up)

Course No.	: CSIZG527/SEZG527
Course Title	: Cloud Computing
Nature of Exam	: Closed Book
Weightage	: 35% (As per Course Handout)
Duration	: 2 Hours
Date of Exam	: 05/10/2024 (FN/AN)

No. of Pages = 02

Note to Students:
1. Please follow all the *Instructions to Candidates* given on the cover page of the answer book.
2. All parts of a question should be answered consecutively. Each answer should start from a fresh page.
3. Assumptions made if any, should be stated clearly at the beginning of your answer.

Q.1 An international retail company is worried about potential data loss and downtime due to disasters such as natural calamities or cyber-attacks. The company is considering using cloud-based disaster recovery solutions. What type of cloud deployment model and disaster recovery strategy would you recommend? How would you ensure minimal downtime and data loss? Provide a detailed justification for your recommendations.**6 Marks**

Q.2 A financial institution is considering using virtualization to enhance its disaster recovery plan. By implementing virtualization, they aim to achieve quicker recovery times and reduce downtime. What virtualization techniques should they adopt, and how would these techniques improve their disaster recovery capabilities? Discuss any limitations or challenges they might face. **6 Marks**

Q.3 Consider the QEMU/KVM VMM that implements hardware-assisted virtualization.
(a) A QEMU process has memory-mapped a guest OS image into its address space, and is scheduled by the host OS scheduler on a certain CPU core. Once the guest VM has been suitably configured using the KVM APIs, which system call(s) does QEMU use to begin the execution of the guest VM code on the CPU core? Be Brief
(b) When QEMU invokes the system call(s) in part (a) above, which CPU instruction(s) are used by KVM to begin execution of the guest VM on the CPU core? Be Brief **3 + 3(6 Marks)**

Q.4 Given the need to optimize performance and resource utilization in a cloud-based data center, analyze the differences between Full Virtualization and Para-Virtualization. How would the choice between these two approaches impact the overall efficiency, performance, and management of the data center? Consider aspects such as hypervisor overhead, guest operating system compatibility, and system security in your analysis. Provide a well-reasoned argument for which virtualization method you would recommend in this scenario, supported by examples and justification. **6 Marks**

Q.5 Explain the relationship between Docker images and Docker containers. How does a Docker image become a running container? Provide an example to illustrate your explanation(**6 Marks)**

Q.6 A) Describe the primary differences between Amazon EC2 (Elastic Compute Cloud), Amazon RDS (Relational Database Service), and Amazon S3 (Simple Storage Service). What are the main use cases for each of these services?
B) Explain the difference between an AWS Region and an Availability Zone. How does AWS use these concepts to enhance the reliability and fault tolerance of its services? **3+2 (5 Marks)**

Answer Key

Q.1 An international retail company is worried about potential data loss and downtime due to disasters such as natural calamities or cyber-attacks. The company is considering using cloud-based disaster recovery solutions. What type of cloud deployment model and disaster recovery strategy would you recommend? How would you ensure minimal downtime and data loss? Provide a detailed justification for your recommendations. **6 Marks**

Answer:

A **Hybrid Cloud** deployment model is ideal for an international retail company concerned with data loss and downtime due to disasters. This model combines the advantages of both public and private clouds. Critical and sensitive business data can be stored in the private cloud, while less sensitive data or recovery environments can be hosted in the public cloud, providing flexibility, cost savings, and scalability.

Recommended Disaster Recovery Strategy: Disaster Recovery as a Service (DRaaS)

Disaster Recovery as a Service (DRaaS) is a cloud-based solution that replicates and hosts physical or virtual servers in a secondary data center, offering failover in the event of a disaster. DRaaS is efficient for organizations that do not want to maintain a fully redundant on-premise data center but need robust recovery options.

Key Components of the Strategy:

1. **Cloud Backup and Replication**:
    - Regular backups and real-time replication of critical data to the cloud minimize the risk of data loss.
    - Replication ensures that data is always up-to-date, even during a disaster.
2. **Geographically Dispersed Data Centers**:
    - Cloud providers often maintain data centers in various regions. Replicating data to geographically separated data centers ensures availability even if one location is affected by a disaster.
3. **Failover and Failback**:
    - In the event of an outage, **automatic failover** to a secondary cloud environment ensures minimal downtime.

- Once the primary systems are restored, **failback** to the original environment can be performed without data loss.

4. **Business Continuity Planning**:
   - A detailed **business continuity plan (BCP)** is crucial to ensure that both cloud and on-premises systems work in tandem.
   - The hybrid approach allows the company to operate critical services via private cloud infrastructure while leveraging public cloud resources for rapid recovery in the event of downtime.

Justification for Recommendations:

1. **Minimal Downtime**:
   - DRaaS and hybrid cloud ensure fast failover and quick recovery. This can bring downtime to near-zero, ensuring the company continues operations even during a disaster.
2. **Scalability and Cost-Effectiveness**:
   - Using public cloud services for disaster recovery allows the company to pay for resources only when needed, avoiding the high costs of maintaining a secondary physical data center.
3. **Global Availability**:
   - As an international company, the retail chain would benefit from the global footprint of major cloud providers, ensuring that data and recovery environments are always close to operational regions, minimizing latency and downtime.
4. **Security**:
   - Hybrid cloud ensures that sensitive data remains under the company's control in a private cloud, reducing the risk of cyber-attacks, while using public cloud resources for disaster recovery to minimize costs.
5. **Data Integrity and Redundancy**:
   - Regular backups, coupled with real-time replication to the cloud, ensure that no data is lost in the event of a failure. Data integrity is maintained across different regions and environments.

Q.2 A financial institution is considering using virtualization to enhance its disaster recovery plan. By implementing virtualization, they aim to achieve quicker recovery times and reduce downtime. What virtualization techniques should they adopt, and how would these techniques improve their disaster recovery capabilities? Discuss any limitations or challenges they might face. **6 Marks**

Answer:

1. **Virtual Machine Replication (VM Replication)**:
   - VM replication involves creating real-time or scheduled copies of virtual machines (VMs) and storing them in a separate physical location or data center. This ensures that the institution has ready-to-use copies of its critical systems in the event of a disaster.
   - **How it improves DR**: In the case of an outage, VMs can be instantly switched to the replicated versions, ensuring minimal downtime and data loss. The recovery time

objective (RTO) is reduced as the secondary VMs can be activated almost immediately.

2. **Snapshot Technology**:
    - Snapshots capture the state of a VM at specific points in time, enabling quick restoration to a previous version in case of data corruption or system failure.
    - **How it improves DR**: If a disaster or a failure occurs, the institution can revert to the last known good state of the VM quickly, reducing downtime and providing greater recovery flexibility.
3. **Live Migration**:
    - Live migration allows the institution to move VMs between physical hosts without shutting them down. This feature is typically used for load balancing or maintenance but can also be utilized for disaster recovery purposes.
    - **How it improves DR**: During a potential failure or scheduled maintenance, VMs can be migrated to other hosts, avoiding service interruptions and improving availability.
4. **Disaster Recovery as a Service (DRaaS) with Virtualization**:
    - DRaaS often relies on virtualized infrastructure to back up and restore VMs to a cloud environment. This eliminates the need for an on-premise disaster recovery site.
    - **How it improves DR**: DRaaS leverages virtualized resources in the cloud for quick failover, allowing the financial institution to recover services almost instantly from a remote location, improving recovery time objectives (RTO) and recovery point objectives (RPO).
5. **High Availability (HA) Clusters**:
    - HA clusters ensure that if one physical host fails, VMs are automatically restarted on another host in the cluster without manual intervention.
    - **How it improves DR**: This reduces downtime because the VMs can recover from hardware failure automatically, providing continuous availability for critical financial services.

How These Techniques Improve Disaster Recovery:

- **Faster Recovery Times**: Virtual machines can be replicated and restored much faster than physical servers. This means the institution can recover critical systems quickly, reducing both downtime and the risk of financial loss.
- **Reduced Costs**: Virtualized environments eliminate the need for maintaining separate physical servers for disaster recovery, leading to cost savings. Resources can be dynamically allocated and used only when needed, especially when paired with DRaaS.
- **Scalability**: Virtualization allows the financial institution to scale its disaster recovery plan easily. As workloads grow, more VMs can be added, and replication policies can be adjusted accordingly.
- **Flexibility**: Virtualization provides the flexibility to restore services in different locations, including in the cloud, which ensures a higher level of resilience for financial operations.

Limitations and Challenges:

1. **Network Bandwidth**:
    - High-speed network connections are critical for continuous VM replication and live migration. Insufficient bandwidth may result in slow replication or migration times, affecting recovery performance.

- **Challenge**: A financial institution would need to invest in high-quality network infrastructure to support efficient replication and disaster recovery, particularly for real-time replication.

2. **Data Consistency**:
   - Virtualized environments must ensure that replicated VMs or snapshots are consistent, especially for transactional systems like databases used in financial services.
   - **Challenge**: Maintaining data consistency across replicated VMs is complex, particularly in environments where transactions occur rapidly. Snapshot or replication failures can lead to data corruption.
3. **Resource Overhead**:
   - Virtualization platforms may introduce resource overheads, such as CPU and memory consumption. These overheads can affect the performance of both production and recovery environments.
   - **Challenge**: The financial institution must carefully allocate resources to balance both performance and disaster recovery capabilities, which might increase costs or complexity.
4. **Licensing and Compliance**:
   - Virtualized environments may require additional licenses for DR software, virtualization platforms, or cloud services.
   - **Challenge**: Financial institutions must ensure that their disaster recovery solutions comply with stringent regulatory standards (e.g., PCI DSS, GDPR) while managing the additional licensing costs for virtualization tools.
5. **Complexity of Configuration**:
   - Configuring virtualized disaster recovery solutions, such as live migration and VM replication, can be complex and may require specialized skills.
   - **Challenge**: The institution might face challenges related to the configuration and ongoing management of virtualized disaster recovery environments, potentially requiring skilled IT personnel and significant planning.

Q.3 Consider the QEMU/KVM VMM that implements hardware-assisted virtualization. (a) A QEMU process has memory-mapped a guest OS image into its address space, and is scheduled by the host OS scheduler on a certain CPU core. Once the guest VM has been suitably configured using the KVM APIs, which system call(s) does QEMU use to begin the execution of the guest VM code on the CPU core? Be Brief (b) When QEMU invokes the system call(s) in part (a) above, which CPU instruction(s) are used by KVM to begin execution of the guest VM on the CPU core? Be Brief **3 + 3 (6 Marks)**

Answer:

(a) System Call(s) Used by QEMU to Begin Guest VM Execution (3 Marks):
After QEMU has set up the guest VM using KVM APIs, it invokes the **ioctl** system call to interact with the KVM module. Specifically, QEMU uses the **ioctl(KVM_RUN)** command to start the execution of the guest VM. This system call triggers the transfer of control from the QEMU process in user space to the KVM kernel module, which runs the guest VM on the designated CPU core.

- **System Call**: ioctl(KVM_RUN)

Explanation:

- **ioctl(KVM_RUN)**: This is the key system call used by QEMU to start running the guest VM on the CPU core. It transitions the execution from QEMU to the guest virtual machine, using the KVM hypervisor's capabilities.

(b) CPU Instruction(s) Used by KVM to Start Guest VM Execution (3 Marks):

Once QEMU invokes **ioctl(KVM_RUN)**, the KVM kernel module takes over and uses CPU instructions that support hardware-assisted virtualization to execute the guest VM. The specific instructions used depend on the processor architecture:

- On **Intel processors** (with Intel VT-x), KVM uses the **VMLAUNCH** or **VMRESUME** instructions to start or resume the guest VM execution.
- On **AMD processors** (with AMD-V), the **VMRUN** instruction is used.
- **For Intel processors**: VMLAUNCH (to start the VM) or VMRESUME (to resume a paused VM).
- **For AMD processors**: VMRUN (to start/resume the VM).

Explanation:

- These instructions transition the CPU from executing host code to running the guest VM in a virtualized environment with hardware support. The CPU operates in a special mode where it can execute the guest OS directly, while the KVM module handles transitions between host and guest contexts.

4.Q. Given the need to optimize performance and resource utilization in a cloud-based data center, analyze the differences between Full Virtualization and Para-Virtualization. How would the choice between these two approaches impact the overall efficiency, performance, and management of the data center? Consider aspects such as hypervisor overhead, guest operating system compatibility, and system security in your analysis. Provide a well-reasoned argument for which virtualization method you would recommend in this scenario, supported by examples and justification. (6 Marks)

Answer:

In the context of cloud-based data centers, virtualization enables multiple operating systems to run on a single physical server by abstracting the hardware resources. The two prominent approaches to virtualization are Full Virtualization and Para-Virtualization. Each has specific advantages and challenges when it comes to performance, efficiency, and management. Below, we analyze their differences and implications for cloud data center operations.

Full Virtualization: It creates a complete virtual environment where guest operating systems run as if they were on dedicated hardware. A hypervisor, or Virtual Machine Monitor (VMM), emulates the entire hardware architecture.

Hypervisor Overhead: Full virtualization typically incurs more overhead because the hypervisor must simulate the hardware environment for each VM. This emulation slows down performance since every request from the guest OS goes through the hypervisor.

Guest OS Compatibility: One of the strengths of full virtualization is that the guest OS requires no modifications. This makes it suitable for a broad range of operating systems, providing flexibility in multi-tenant cloud environments.

Performance: Due to the hypervisor's emulation layer, full virtualization can experience degraded performance, especially during I/O-intensive operations. However, modern hardware-assisted virtualization (e.g., Intel VT-x, AMD-V) can mitigate some of this performance overhead.

System Security: Full virtualization offers strong isolation between VMs, making it highly secure. Since the VMs operate independently, they do not interfere with each other, which is ideal for multi-tenant cloud systems.

Para-Virtualization: It modifies the guest operating system to make it aware that it is running in a virtualized environment. The hypervisor interacts with the guest OS more directly, bypassing certain hardware emulation processes.

Hypervisor Overhead: Para-virtualization reduces the overhead significantly compared to full virtualization. Since the guest OS cooperates with the hypervisor, tasks such as memory management and I/O processing are more efficient.

Guest OS Compatibility: The main drawback of para-virtualization is that the guest OS must be modified, which restricts the number of compatible operating systems. Only those specifically designed or adapted for para-virtualization (e.g., certain Linux distributions) can be used.

Performance: Due to the reduced overhead, para-virtualization generally offers better performance than full virtualization, especially in environments where resource-intensive operations occur. I/O operations, in particular, benefit from this approach.

System Security: While para-virtualization can also provide isolation between VMs, the direct interaction between the guest OS and the hypervisor presents potential security vulnerabilities. Additional security measures are required to prevent hypervisor attacks or breaches.

Impact on Cloud Data Centers

1. Efficiency: Para-virtualization offers reduced overhead and improved resource utilization, making it more efficient in environments with demanding computational workloads. Its ability to directly interact with the hardware (with some assistance from the hypervisor) minimizes the overhead seen in traditional virtualization techniques. This makes it ideal for cloud data centers where performance is prioritized. In contrast, full virtualization tends to introduce more overhead as it relies on emulating hardware completely, making it slightly less efficient but more versatile. This versatility can be beneficial for cloud data centers that support a wide range of operating systems and applications, where flexibility and compatibility are key concerns.

2. Performance: For cloud data centers dealing with high-frequency I/O operations or compute-heavy applications, para-virtualization is the optimal choice because it introduces less latency. Para-virtualized environments have closer-to-native performance since there is no need to emulate the

entire hardware stack. Full virtualization, by comparison, adds layers of abstraction and emulation, which results in higher latency, making it less suitable for performance-critical tasks.

3. Management: Full virtualization excels in ease of management, largely due to its support for a wide variety of guest operating systems. This flexibility makes it easier for cloud administrators to scale and manage diverse workloads. As new clients or applications are introduced, full virtualization allows seamless integration without requiring extensive system changes. On the other hand, para-virtualization may require more specialized configurations and management, particularly because it often necessitates guest OS modifications. This increases the complexity of managing a cloud environment, but it can also offer greater opportunities for fine-tuning and performance optimization.

4. Security: In cloud data centers, security is a paramount concern, particularly for providers handling sensitive or critical data. Full virtualization generally provides better isolation between virtual machines (VMs), which strengthens security by ensuring that breaches or compromises in one VM don't affect others. Para-virtualization, while offering superior performance, may not provide the same level of isolation out of the box. Additional security configurations, such as more advanced access controls and monitoring, may be required to match the isolation provided by full virtualization.

Therefore, for a cloud data center focused on maximizing performance and resource efficiency, para-virtualization is the recommended approach. Its low overhead and optimized hardware utilization can significantly improve performance in high-demand environments, such as those with intensive computational or I/O tasks. However, if the flexibility to run a wide variety of guest operating systems and the need for robust security are more important, then full virtualization is a better fit. Full virtualization offers better isolation between VMs, making it ideal for environments where data sensitivity or compliance with strict security regulations is crucial.

Example: Major cloud providers, such as Amazon Web Services (AWS), often adopt a hybrid virtualization model. They use para-virtualization for workloads that require higher performance, such as high-throughput computing or large-scale data processing. At the same time, full virtualization is employed where OS flexibility or enhanced security isolation is required. This approach allows cloud providers to optimize both resource utilization and security, adapting to the specific needs of different applications and workloads.

5.Q. Explain the relationship between Docker images and Docker containers. How does a Docker image become a running container? Provide an example to illustrate your explanation. (6 Marks)

Answer:

Docker images and Docker containers are fundamental components in the Docker ecosystem, each serving distinct but closely related purposes. To fully grasp their relationship, let's break it down step by step, focusing on their roles, how they interact, and the process by which an image becomes a running container.

Docker Images: A Docker image is essentially a blueprint or template used to create Docker containers. It is a read-only package that contains everything needed to run an application such as the application code, libraries, dependencies, environment variables, configuration files, and other essential settings. Docker images are built in layers, with each layer representing a change or addition to the previous one. These layers ensure that the image is lightweight and that changes are efficient, as only modified layers need to be updated rather than the entire image.

The characteristics of Docker images are as follows:
Static - A Docker image is unchangeable (read-only). It remains the same after being created.
Versioned - Images can be tagged to represent different versions of the same application, making it easy to roll back to a previous version if needed.
Portable - Docker images can be shared across environments, ensuring consistency whether they are deployed on a developer's local machine or in a production environment.

In simple terms, a Docker image is akin to a snapshot of an application's environment, capturing all the necessary components for that application to run correctly.

Docker Containers: A Docker container, on the other hand, is a runtime instance of a Docker image. It is an active, running environment based on the image, where the application code is executed. When a container is created from an image, Docker adds a writable layer on top of the read-only image, allowing the container to make changes (like creating files, logs, or modifying configurations) without altering the underlying image.

In this sense, a container is like a living, breathing entity built from the static image. It includes all the components from the image but also has the capability to run processes, interact with the outside world, and persist data.

The characteristics of Docker containers are as follows:
Dynamic - Containers are running instances that can be started, stopped, or destroyed. They are temporary but can persist as long as they are needed.
Isolated - Containers are isolated from each other and the host machine, ensuring that changes made within one container do not affect others. This isolation is one of the core benefits of Docker.
Lightweight - Unlike virtual machines, which require entire operating systems, containers share the host OS kernel, making them much more lightweight and faster to deploy.

The relationship between Docker images and Docker containers can be summarized as follows:
A Docker image is the blueprint from which Docker containers are created, and a Docker container is the running instance of that image. When you run a Docker image, Docker converts that static image into a running container by adding a writable layer and initiating the processes defined in the image (such as starting the application, running commands, etc.). In other words, containers are spawned from images. The image serves as the immutable foundation, while the container is the mutable environment where the application lives and operates.

Explaining how a Docker Image becomes a Running Container: The process of turning a Docker image into a running container involves several steps.

1. Pull the Image : When you attempt to run a Docker container, Docker first checks whether the image already exists on the local machine. If the image is not present, Docker will pull the image from a Docker registry like Docker Hub.

2. Create a Container: Once the image is available, Docker creates a container by adding a writable layer on top of the read-only image. This writable layer allows the container to handle temporary changes (such as logs, temporary files, or changes to configurations) while keeping the original image intact.

3. Start the Container: Docker then starts the container by launching the entry point defined in the image. This entry point is often a command or script that starts the main process of the application inside the container. And at this point, the Docker image has become a live, running container.

4. Execution and Interaction: While the container is running, users can interact with it, view logs, or even execute commands inside the container. However, when the container is stopped or destroyed, any changes made inside the container (in its writable layer) will be lost unless they were saved to a persistent storage volume.

Example: Let's walk through a simple example to explain the relationship between Docker images and containers, Imagine you are baking a cake.

1. Docker Image: The Recipe (Blueprint)
- A Docker image is like a cake recipe. It contains all the instructions and ingredients (software, libraries, dependencies, etc.) needed to make the cake (the container).
- The recipe tells you exactly how to make the cake, but by itself, it's not edible. Similarly, a Docker image contains everything needed to run an application, but it is not in a running state.
- Docker images are stored in a repository (like Docker Hub), much like recipes are stored in a cookbook or a recipe website. To use the recipe, you need to pull it out from this storage.

Command in Docker (Getting the Recipe): docker pull ubuntu
This command downloads the "recipe" (in this case, the Ubuntu image) from Docker Hub, just like copying a recipe from a cookbook.

2. Creating a Docker Container: Baking the Cake
- When you want to create a cake, you follow the instructions in the recipe and gather the ingredients (run the Docker image).
- A container is like the finished cake that you can interact with. It is the actual running instance of the image, just like the baked cake is a product of following the recipe.
- You can make as many cakes (containers) as you like from the same recipe (image), and each cake can be slightly different (depending on the environment or additional customizations), but all are based on the same recipe.

Command in Docker (Baking the Cake): docker run -it ubuntu
`docker run` creates and runs a new container from the Ubuntu image (just like following the recipe and baking a cake). The `-it` flag allows you to interact with the container (baked cake) through the terminal.

3. Multiple Containers: Baking Multiple Cakes

- With the same recipe (Docker image), you can bake multiple cakes (containers). Each cake is an independent entity, but they all come from the same recipe.
- For instance, you might bake one cake with vanilla frosting and another with chocolate frosting. Similarly, you can run multiple containers from the same image, and each container can be modified or customized differently.

Command in Docker (Multiple Cakes):

```
docker run --name cake1 -d ubuntu
docker run --name cake2 -d ubuntu
```

These commands create two containers (`cake1` and `cake2`) from the same Ubuntu image, similar to baking two cakes from one recipe. Each container runs independently, just like each cake can be served or decorated separately.

4. Container Lifecycle: Eating, Freezing, or Storing the Cake

- After baking the cake (creating a container), you can decide what to do with it: eat it right away, freeze it for later, or even throw it away once you're done.
- Similarly, Docker allows you to start, stop, or remove containers. You can also make changes to a running container, just as you could add frosting or decorations to a baked cake.

Commands in Docker (Managing the Cake):

-Start/Stop (Eating the Cake/Putting it in the Freezer):

```
docker stop cake1
docker start cake1
```

These commands stop and start a container, much like eating part of a cake and then storing it for later use.

Remove the Container (Throwing Away the Cake): docker rm cake1
This command removes the container, similar to discarding a cake when it's no longer needed.

5. Docker Layers: Adding Frosting and Toppings

- A container has layers, just like a cake can have layers (base cake, frosting, toppings). In Docker, when a container is created from an image, a writable layer is added on top of the image, allowing changes to be made (like adding frosting).
- The original image (recipe) remains unchanged, just like you can bake multiple cakes with different toppings while the base recipe stays the same.

Example of Adding Frosting (Modifying the Cake):

- If you want to install software or change configurations in the running container, you can do it without affecting the original image:

```
docker exec -it cake1 bash
apt-get install curl
```

- Here, you are adding “frosting” to the container (i.e., making changes inside it) by installing additional software like `curl`. This modification only affects the container (cake) and not the image (recipe).

6. Saving Changes: Creating a New Recipe from a Modified Cake
- If you’ve baked a cake and added some amazing decorations, you might want to save this version of the cake as a new recipe for future use. In Docker, if you modify a running container and want to save it as a new image, you can “commit” the changes.
- This is like writing down your changes to the original recipe so you can bake the modified version in the future.

Command in Docker (Saving the Modified Cake as a New Recipe):

```
docker commit cake1 my_custom_cake_image
```

This command saves the current state of the `cake1` container as a new image called `my_custom_cake_image`, just like creating a new recipe based on your modifications to the original cake.

Using this cake analogy, we can see that Docker images and containers have a clear relationship:
- A Docker image is the immutable recipe that defines how to create a container (cake).
- A Docker container is the running instance of that image, like a freshly baked cake, which you can interact with, modify, or remove.

Just as you can bake multiple cakes from a single recipe, you can create multiple containers from a single image. Containers are lightweight and efficient, sharing resources while remaining isolated from one another, just like multiple cakes baked from the same recipe can be served and decorated differently.

In conclusion, the relationship between Docker images and Docker containers is foundational in containerized environments. Docker images provide a static definition of the environment, while Docker containers represent dynamic, running instances of these images. A Docker image becomes a running container through the docker run command, which sets up the containerized environment, executes the application, and provides isolation and efficiency.

6.Q. (A) Describe the primary differences between Amazon EC2 (Elastic Compute Cloud), Amazon RDS (Relational Database Service), and Amazon S3 (Simple Storage Service). What are the main use cases for each of these services?
(B) Explain the difference between an AWS Region and an Availability Zone. How does AWS use these concepts to enhance the reliability and fault tolerance of its services? (3+2= 5 Marks)

Answer:

(A) Primary Differences Between Amazon EC2, Amazon RDS, and Amazon S3
Amazon EC2 (Elastic Compute Cloud): It provides scalable computing capacity in the cloud. It allows users to run virtual servers, known as instances, to deploy and manage applications without needing to invest in physical hardware. Users have full control over the operating system, network

configuration, and storage associated with each instance, offering flexibility for a wide range of applications.
- Use Case: Hosting web servers, running applications, batch processing, high-performance computing, and deploying microservices architectures.

Amazon RDS (Relational Database Service): It is a managed relational database service that simplifies database administration tasks like backups, patching, scaling, and replication. RDS supports various database engines such as MySQL, PostgreSQL, Oracle, and SQL Server, allowing users to focus on optimizing their application instead of managing the underlying database infrastructure.
- Use Case: Running and managing relational databases for transactional workloads, CRM systems, and ERP systems, without needing to manage the database hardware and software directly.

Amazon S3 (Simple Storage Service): It is an object storage service designed to store and retrieve any amount of data from anywhere on the web. It provides highly durable and scalable storage with easy integration for backup, disaster recovery, data archiving, and big data analytics.
- Use Case: Storing large amounts of unstructured data such as files, images, videos, and backups. S3 is also widely used for hosting static websites and providing storage for big data applications.

Differences:
Functionality: EC2 is focused on compute resources, RDS on managed databases, and S3 on object storage.
User Control: EC2 offers maximum control with infrastructure management, RDS simplifies database management, while S3 provides a fully managed storage solution.
Use Case: EC2 is for running applications, RDS is for database management, and S3 is for storage and retrieval of large data sets.

(B) Difference between an AWS Region and an Availability Zone
AWS Region: It is a geographic area that contains multiple isolated Availability Zones (AZs). Each region is completely independent and provides low-latency, secure cloud computing services to customers in that specific area. AWS currently has numerous regions worldwide, and data is not automatically replicated across regions unless explicitly configured.

Availability Zone (AZ): It is a physical data center located within a region. Each region typically has multiple AZs, which are isolated from one another but connected with low-latency, high-bandwidth networks. This ensures redundancy and enables applications to remain available even if one AZ experiences an issue.

Enhancing Reliability and Fault Tolerance: AWS leverages the concept of multiple regions and AZs to enhance reliability and fault tolerance. By deploying resources across different AZs or regions, AWS customers can create highly available and resilient architectures. If one AZ goes down, applications can failover to another AZ, ensuring minimal downtime. Similarly, by replicating data across multiple regions, organizations can safeguard against regional outages, ensuring data and services remain available even in the event of major failures in a specific geographic area.