Q1) Which managed service could provide distribution of incoming traffic across multiple Amazon EC2 instances?

Answer:

C. Elastic Load Balancing

Q2) Which element of AWS Global infrastructure consists of multiple Availability Zones?

Answer:

B. Region

Q3) What feature of a Virtual Private Cloud (VPC) works like a virtual firewall for the network?

Answer:
C. Security Groups

Q4) You are looking to launch a static website with minimal configuration. Which of the following AWS services provides this feature?

Answer:
D. Amazon S3

Q5) Which online tool provides guidance in following AWS best practices by providing personalised recommendations for your cloud workloads?

Answer:
A. AWS Trusted Advisor

Q6) What is the minimum AWS support plan that provides the complete set of AWS Trusted Advisor checks?

Answer:
B. Business

Q7) Your US-based team wants to expand an internal application to your office in Singapore. They are able to take the application and spin up a version in the ap-

southeast-1 region. Which benefit of cloud computing is best illustrated with this scenario?

Answer:
B. Go global in minutes

Q8) Your organisation is looking to test a new web application concept. The team set up an AWS account and infrastructure within a day on which they will build a prototype for this application. Which cloud computing benefit does this illustrate?

Answer:
C. Increase speed and agility

Q9) Which element of the AWS Global Infrastructure is made up of one or more data centres connected by a low latency network with redundant power, networking, and connectivity within an AWS Region?

Answer:
D. Availability Zone

Q10) You want to have email support within business hours from AWS for an application you are building. What is the minimum support plan that provides this?

Answer:
C. Business

Q11) Which of these is NOT true about containers?

Answer:
A. Containers include the operating system as well as code and libraries

Q12) What is a Dockerfile?

Answer:
C. A template used to describe the build of an image

Q13) What command do we use to completely get rid of a Docker Image?

Answer:
B. docker rmi

Q14) How do you manage Docker images?

Answer:
D. Both A & B

Q15) What is Docker?

Answer:
A. an open-source lightweight containerisation technology

Q16) What are the important features of Docker?

Answer:
D. All of the above

Q17) What is a Docker image?

Answer:
A. a read-only set of instructions for creating a container

Q18) What is Docker Engine?

Answer:
A. an open-source containerisation technology for building and containerising your applications

Q19) What is Docker Swarm?

Answer:
A. a container orchestration tool

Q20) What is Docker Hub?

Answer:
A. a service provided by Docker for finding and sharing container images with your team

Q21) Which of the following commands will run a command inside a running container?

Answer:
B. docker exec

Q22) How can you view logs for a running container?

Answer:
A. docker logs <container_id>

Q23) What is the primary purpose of Docker Compose?

Answer:
C. To define and run multi-container Docker applications

Q24) Which file format is used to define services in Docker Compose?

Answer:
C. YAML

Q25) What does the command `docker ps` do?

Answer:
A. Lists all running containers

Q26) Which command is used to create a new Docker container from an image?

Answer:
B. docker run

Q27) Which command is used to build a Docker image from a Dockerfile?

Answer:
A. docker build

Q28) What is the purpose of the `docker volume` command?

Answer:
D. To manage persistent data storage

Q29) What does the `--rm` option do when running a Docker container?

Answer:
B. Automatically removes the container when it exits

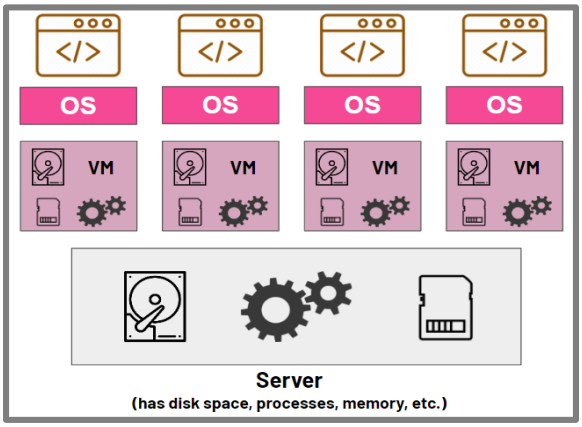
Q30) What is the role of a Docker registry?

Answer:
C. To store and distribute Docker images

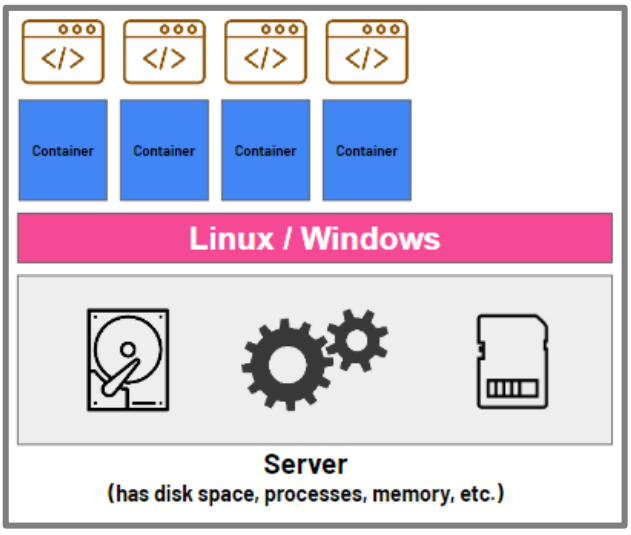
Diagram Description (20 points total)

Given the diagrams below, please describe these two models in detail and compare how their efficiency may differ from one another. You will be awarded 10 points for the correct description of these models and further 10 points for providing the "key points" comparison analysis.

Model 1



Model 2



Answer:

1. Virtual Machines (VMs)

Think of a virtual machine (VM) as a complete computer running inside another computer. Here's how it works:

- **Multiple operating systems**: In the first image, each box labeled "OS" represents a separate operating system (like Windows, Linux, etc.). Each application is running its own operating system inside the server, even though the server already has one.
- **Virtual machines**: Each application is put inside a **VM**, which has its own space, processes, and memory. It's like having different computers inside one bigger computer (the server). Each VM doesn't know the others exist—they are isolated, meaning if one VM crashes or has a problem, it won't affect the others.
- **Hypervisor**: There's something called a **hypervisor** in the background that manages all these VMs. It controls how much memory, CPU, and storage each VM gets from the server.

Example:

Imagine a hotel. Each VM is like a guest in their own hotel room. Each room has its own bathroom, TV, and bed (like having its own operating system and resources), but the hotel staff (the hypervisor) controls what each room gets—how much water, electricity, etc.

2. Containers

Containers are a lighter, more modern way of running applications on a server. Here's how they work:

- **Shared operating system**: In the second image, instead of having a separate operating system (OS) for each application, all the containers share one single OS. This saves space because you don't need to install a full OS for each container.
- **Containers**: Each application runs inside its own container. Containers are like mini-boxes where each app has just what it needs to run. They are isolated in terms of processes and files, but they don't need their own full operating system.
- **Faster and lighter**: Containers are smaller than VMs because they don't carry their own OS. They use the shared OS of the server and only package the necessary files and settings to run the application.

Example:

Imagine a train with carriages. Each container is like a carriage in a train. All the carriages share the same engine (like the shared OS), but each carriage has its own passengers (applications) and is separate from the others.

Comparison of Efficiency:

The two models differ significantly in terms of resource efficiency, scalability, and performance. Below are the **key points** for comparison:

Key Points Comparison:

Aspect	Virtual Machines (VMs)	Containers
Resource Usage	Requires a full OS per VM, making it heavier on CPU and memory.	Shares the OS, making it lightweight with lower resource overhead.
Boot Time	Slower startup as each VM needs to boot a full OS.	Faster startup since only the application needs to start (OS is already running).
Isolation	Strong isolation since each VM has its own OS.	Good isolation, but slightly less than VMs as they share the same OS kernel.
Efficiency	Less efficient due to the duplication of OS resources per VM.	Highly efficient in resource usage since containers share the same OS.
Portability	VMs are portable but depend on the underlying hypervisor.	Containers are highly portable across different environments due to OS-level abstraction.
Overhead	Higher overhead because of running separate OS instances.	Minimal overhead as containers share the same OS.
Density	Lower density: fewer VMs can run on the same hardware due to resource demands.	Higher density: many more containers can run on the same hardware, maximizing utilization.
Scaling	More difficult to scale quickly due to heavier resource use.	Easier to scale up and down as containers are lightweight.

Final Thoughts on Efficiency:

1. **Resource Efficiency**: Containers are much more resource-efficient than VMs because they share the same OS and don't require as much CPU, memory, or storage.
2. **Speed**: Containers start faster and are easier to move around than VMs. They're great for environments where you need things to happen quickly.
3. **Scalability**: Containers are better for scaling up because they use fewer resources, allowing you to run many more applications on the same server.
4. **Isolation and Security**: While VMs offer stronger isolation, containers are still secure enough for most use cases and are preferred for modern cloud applications.

Hosting a Static Website with S3

Diagram Description (20 points total)

Given the diagrams below, please describe these two models in detail and compare how their efficiency may differ from one another. You will be awarded 10 points for the correct description of these models and further 10 points for providing the "key points" comparison analysis.

For this task, I needed to host a simple static website using AWS S3. Below are the detailed steps I followed, from creating the website to configuring the hosting. I didn't have an AWS account at first, so I also included the registration process.

Step 1: Creating an AWS Account

To begin, I first registered for an AWS account:

1. I went to AWS and clicked on **Create an AWS Account**.
2. I followed the steps to fill out my personal details, payment information, and verification process.
3. After the registration was complete, I was able to access the **AWS Management Console**.

Step 2: Setting Up an S3 Bucket

Next, I set up an S3 bucket to store and host the website files:

1. I logged into AWS, searched for **S3** in the search bar, and clicked on it from the list of services.
2. Once in the S3 dashboard, I clicked **Create Bucket**.
    - I named the bucket as something unique (for example, bharani-static-website).
    - I chose a region closest to me for better performance.
    - I unchecked **Block all public access** since I needed the website to be accessible by the public.
    - Finally, I acknowledged the changes and created the bucket.

Step 3: Uploading Website Files

After the bucket was ready, I uploaded my website files:

1. Inside the bucket, I clicked **Upload** and then selected the **Add Files** button.
2. I navigated to the "static-webpage" folder on my computer and uploaded the following files:
    - index.html
    - css folder

- images folder

3. Once all the files were uploaded, I clicked **Upload** to save them in the S3 bucket.

Step 4: Configuring Static Website Hosting

With the files in the S3 bucket, I configured it to host the website:

1. I navigated to the **Properties** tab of the bucket and scrolled down to find **Static website hosting**.
2. I clicked **Edit** and enabled static website hosting.
3. I set index.html as the **Index Document** (this is the homepage).
4. Although I didn't have an error page, I left the **Error Document** field blank for now and clicked **Save**.

Step 5: Making the Files Public

To make sure that the website files were accessible by visitors, I made them public:

1. I went to the **Objects** tab in the S3 bucket.
2. I selected all the files (index.html, css, images) and clicked **Actions → Make public**.

This ensured that everyone could view the webpage when visiting the link.

Step 6: Testing the Website

After completing the steps, I tested my static website by going to the **Static website hosting** section again and copying the website endpoint link, which looked something like this:

http://bharani-static-website.s3-website.region.amazonaws.com

I pasted the link in my browser and successfully viewed the webpage!

Personalizing the Webpage

Before uploading, I made a small change to the **index.html** file to include my name:

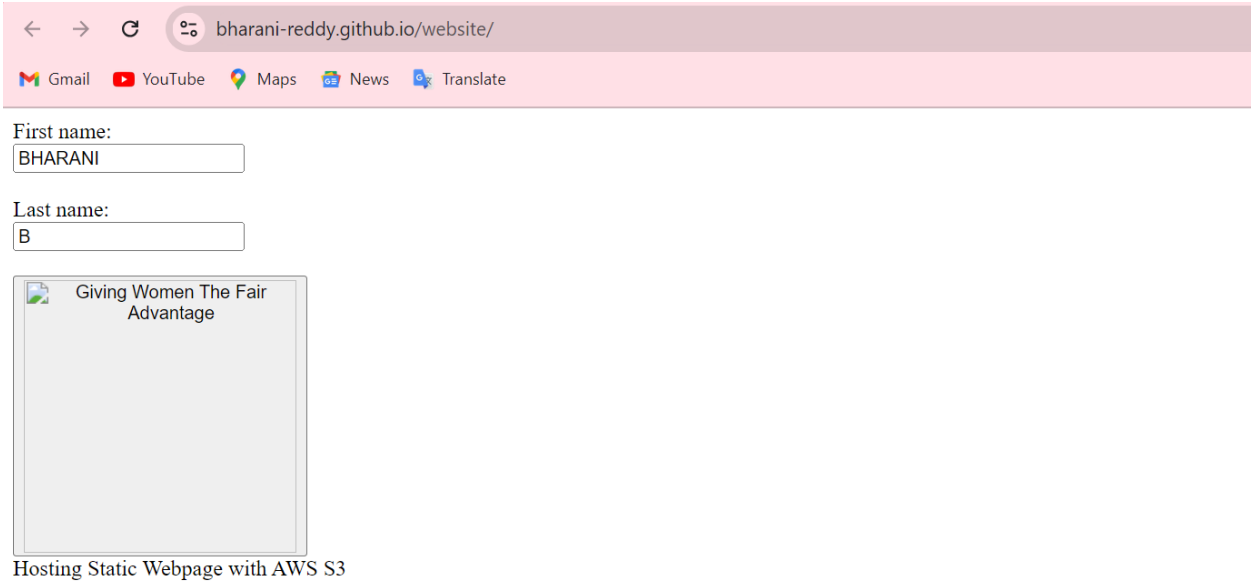
1. I opened the index.html file in a text editor (like Notepad or Visual Studio Code).
2. I found the following lines:

```
html
Copy code
<input type="text" value="<YOUR NAME HERE>">
<input type="text" value="<YOUR SURNAME HERE>">
```

3. I replaced them with:

   html
   Copy code
   <input type="text" value="Bharani">
   <input type="text" value="B">

4. After saving these changes, I uploaded the modified index.html file back to the S3 bucket.



https://bharani-reddy.github.io/website/

Hosting a Static Website with S3 (20 points total)

Please use the contents of the folder called "static-webpage" that have been shared with you at the beginning of this assessment. Open index.html in any browser to see an example of a simple static webpage. The task is to register and host this webpage (and all required contents like images and css) on cloud with the help of AWS S3 service.

Steps for Hosting Docker App on AWS

Step 1: Modifying `app.py` with My Name

Before anything, I first opened the provided `app.py` file to update my personal details:

1. I opened `app.py` in a text editor.
2. I found the part where I needed to insert my name and surname, as mentioned in the task description. I replaced the placeholders with:

```
python
Copy code
name = "Bharani"
surname = "B"
```

3. After saving the changes, I moved on to create the Docker image.

Step 2: Creating a Docker Image

Next, I needed to create a Docker image using the provided `Dockerfile` and `app.py`:

1. I made sure that Docker was installed on my local machine.
    - If not installed, I would download Docker from the official site and install it.
2. I navigated to the folder containing `app.py` and `Dockerfile` using the terminal.
3. Inside the folder, I ran the following command to build the Docker image:

```
bash
Copy code
docker build -t bharani-docker-app .
```

    - Here, `bharani-docker-app` is the name of the Docker image.
4. Once the build completed, I verified the image by listing the Docker images:

```
bash
Copy code
docker images
```

   I could see my `bharani-docker-app` listed here.

Step 3: Pushing Docker Image to AWS ECR

To store my Docker image in AWS Elastic Container Registry (ECR), I followed these steps:

1. I logged into the AWS Management Console.
2. I searched for **ECR** (Elastic Container Registry) and selected it from the list of services.

3. I clicked **Create Repository** and gave it a unique name like `bharani-docker-app-repo`. I then clicked **Create**.
4. AWS provided the commands to log in to the registry, tag the image, and push the image to the repository:
    - First, I logged into the ECR registry using the following command (AWS CLI needs to be configured):

      ```
      bash
      Copy code
      aws ecr get-login-password --region <your-region> | docker login
      --username AWS --password-stdin <aws-account-id>.dkr.ecr.<your-
      region>.amazonaws.com
      ```

    - Then, I tagged the Docker image to match the repository name:

      ```
      bash
      Copy code
      docker tag bharani-docker-app:latest <aws-account-
      id>.dkr.ecr.<your-region>.amazonaws.com/bharani-docker-app-
      repo:latest
      ```

    - Finally, I pushed the image to the ECR repository:

      ```
      bash
      Copy code
      docker push <aws-account-id>.dkr.ecr.<your-
      region>.amazonaws.com/bharani-docker-app-repo:latest
      ```

Now, my Docker image was successfully uploaded to AWS ECR.

Step 4: Hosting the Docker App with AWS ECS

After pushing the image to ECR, I used AWS Elastic Container Service (ECS) to deploy my app:

1. I navigated to **ECS** in the AWS console and clicked on **Create Cluster**.
2. I chose **Networking only** (AWS Fargate) since it simplifies the process of running containers.
3. I created a cluster, then moved on to define a task definition.
    - In the **Task Definition** section, I clicked **Create new Task Definition**.
    - I chose **Fargate** as the launch type and named the task definition `bharani-task`.
    - For the container definition, I added my Docker image URL from ECR (which I tagged and pushed earlier).
    - I specified the memory and CPU requirements for the container.
4. After creating the task definition, I moved on to create a **Service**:
    - I selected **Fargate** as the launch type.
    - In the service configuration, I chose my task definition and set the desired number of tasks to 1 (since I'm only running a simple app).

- I selected the **Load Balancer** and configured it, then added the port on which my app runs (typically port 80 for web apps).

5. After configuring everything, I clicked **Create** and waited for the ECS service to launch my container.

Step 5: Accessing the Docker App

Once my container was running, I could access my app via the URL provided by AWS:

1. I went to the **ECS Service** dashboard and clicked on the running task to find the **Public IP** or **URL** of the app.
2. I copied the URL and opened it in my browser to check that the app was up and running.

Step 6: Backup (Pushing to Docker Hub)

In case I faced any issues hosting the app with AWS ECS, I followed the alternative option to push the Docker image to **Docker Hub**:

1. I logged into Docker Hub on my machine using:

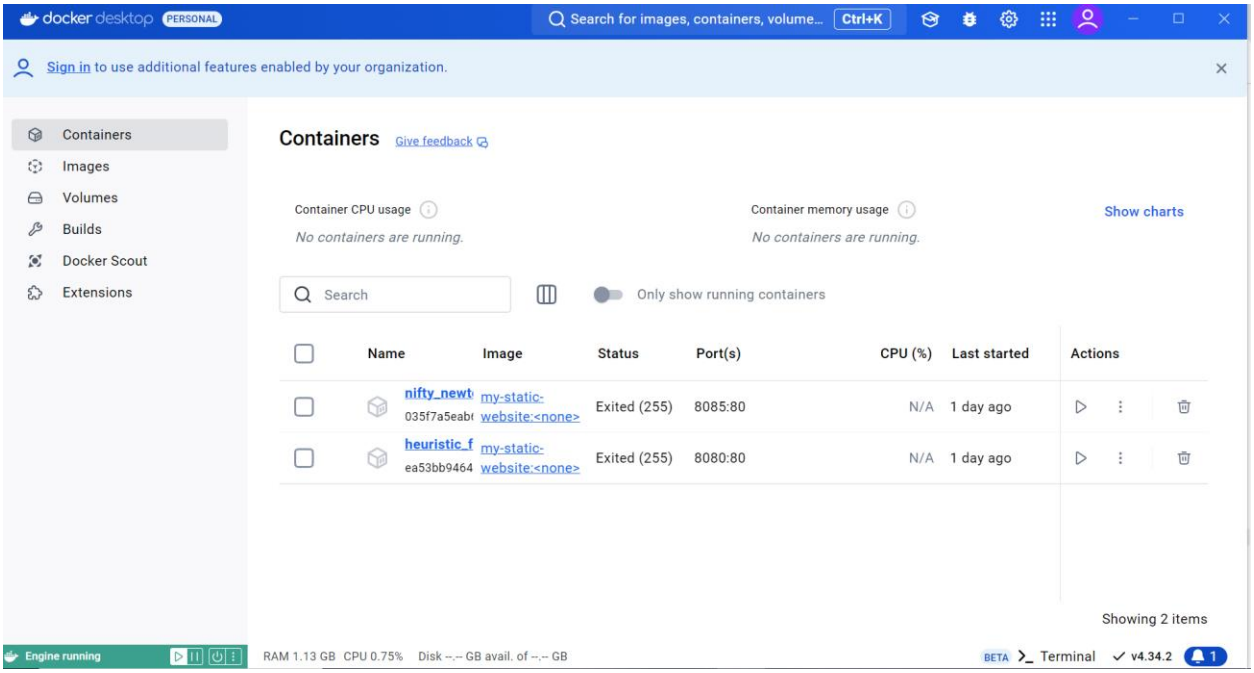
```
bash
Copy code
docker login
```

2. I tagged the image for Docker Hub:

```
bash
Copy code
docker tag bharani-docker-app:latest <dockerhub-username>/bharani-
docker-app:latest
```

3. I pushed the image to my Docker Hub account:

```
bash
Copy code
docker push <dockerhub-username>/bharani-docker-app:latest
```


docker desktop PERSONAL
Search for images, containers, volume... Ctrl+K
Sign in to use additional features enabled by your organization.
Containers
Images
Volumes
Builds
Docker Scout
Extensions
Containers
Give feedback
Container CPU usage
No containers are running.
Container memory usage
No containers are running.
Show charts
Search
Only show running containers
Name
Image
Status
Port(s)
CPU (%)
Last started
Actions
nifty_newt
035f7a5eab
my-static-website:<none>
Exited (255)
8085:80
N/A
1 day ago
heuristic_f
ea53bb9464
my-static-website:<none>
Exited (255)
8080:80
N/A
1 day ago
Showing 2 items
Engine running
RAM 1.13 GB CPU 0.75% Disk --.-- GB avail. of --.-- GB
BETA Terminal
v4.34.2
1