Assignment 1 - 2025W-CSD-4553-OTT01-Cloud Computing

Student Name: Premsingh Padya

ID: C0924501

Definitions:

Virtual Machines

Virtual Machines (VMs) are creating isolated environments that emulate physical computers, allowing them to run their own operating systems and applications. Managed by a hypervisor, VMs are enabling multiple operating systems to run on a single physical machine by allocating dedicated resources to each instance. This flexibility is especially useful for developers who need to work across different OS configurations without interference.

The main advantage of VMs is optimizing resource utilization through server consolidation. Developers can run multiple VMs on a single, powerful machine, reducing hardware costs and energy consumption, while still maintaining control over each VM's environment. This makes VMs ideal for testing enterprise applications, running legacy software, and supporting development environments.

VMs are commonly used in cloud computing environments like Amazon EC2, Microsoft Azure Virtual Machines, and Google Compute Engine. These platforms are widely adopted for hosting enterprise applications, disaster recovery solutions, and development environments. However, VMs are less efficient compared to alternatives like containers due to the overhead of running a full operating system for each VM.

Containers

Containers are lightweight, portable units that package an application along with all its dependencies, making it possible to run consistently across different computing environments. Unlike VMs, containers share the host operating system's kernel while isolating user spaces, making them more efficient in terms of resource utilization. For developers, this means faster deployment and scaling, along with greater flexibility when working in cloud-native development.

Containers are particularly beneficial for supporting microservices architecture. Developers can break down an application into smaller, independent services, each running in its own container. This architecture simplifies updates, scaling, and fault isolation. Since containers are platform-agnostic, developers can seamlessly run the same application across different environments, whether it's development, testing, or production.

Tools like Kubernetes and Docker Swarm are helping manage large-scale containerized applications efficiently. Companies like Netflix, Spotify, and Google use containers to build scalable, high-performance applications. As part of modern DevOps workflows, containers are crucial for continuous integration and deployment (CI/CD), enabling fast and reliable software updates.

Serverless Computing

Serverless computing is a cloud computing model that allows developers to build and deploy applications without managing the infrastructure. Cloud providers handle provisioning and scaling automatically based on application demands, meaning that developers only need to focus on coding and defining triggers for specific events. This setup eliminates the need to provision, scale, and maintain servers, significantly reducing operational complexity and costs for developers.

The main benefit of serverless computing is its ability to handle dynamic workloads. Since resources scale automatically in response to usage, serverless functions are ideal for applications with unpredictable or fluctuating traffic. Developers only pay for actual execution time rather than maintaining constantly running servers, leading to cost efficiency.

Serverless platforms like AWS Lambda, Google Cloud Functions, and Azure Functions support a range of applications, including real-time data processing, event-driven automation, and IoT solutions. For example, an e-commerce platform can use serverless functions to process transactions, send notifications, and analyze user behavior in real time. However, serverless computing introduces challenges such as cold start latency and execution limitations for long-running tasks.

Comparison Table:

Perspective	**Virtual Machines**	**Containers**	**Serverless**
Cost	**Generally higher due to resource overhead of running a full OS.**	**More cost-effective as they share the host OS kernel.**	**Very cost-effective as you only pay for actual execution time.**

Perspective	**Virtual Machines**	**Containers**	**Serverless**
Pros	Full isolation and control over the environment.	More affordable due to shared resources.	Pay-per-use model, only pay for execution time, no idle costs.
Cons	High resource usage and overhead, leading to increased costs.	May depend on the security of the host OS and runtime.	Expensive for long-running tasks; potential cold start latency.
Time to Deploy	**Longer due to setting up a full operating system.**	**Faster deployment with smaller, pre-configured images.**	**Instant deployment with no server management needed.**
Pros	Gives developers full control over the setup and environment.	Pre-configured containers speed up deployment.	No infrastructure management; automatic scaling.
Cons	Time-consuming to set up and configure a full OS.	Requires knowledge of container orchestration (e.g., Kubernetes).	Cold starts may cause delays; limited control over infrastructure.
Size of Deployment	**Ranges from several GBs to tens of GBs (e.g., 5 GB to 50 GB).**	**Typically, a few MBs to several hundred MBs (e.g., 50 MB to 1 GB).**	**A few KBs to a couple of MBs (typically ranging from 10 KB to 50 MB).**
Pros	Can support large, complex applications and environments.	Smaller and quicker to deploy, reducing storage requirements.	Extremely lightweight, fast deployment, and minimal resources.
Cons	Requires more storage space and resources.	May not handle large or complex applications effectively.	Best suited for small tasks; not for large workloads.
Security	**VMs have high isolation but requires manual management of security patches.**	**Lower overhead but relies on the host OS for security.**	**Limited by platform-specific security features, but isolation is strong within individual functions.**

Perspective	**Virtual Machines**	**Containers**	**Serverless**
Pros	Strong isolation and control, more secure for sensitive data.	Faster deployment and execution, lightweight overhead.	Strong isolation within each function, good security within the cloud platform.
Cons	Security management can be complex and manual.	Security depends on the host OS and container runtime security.	Security managed by cloud providers; some restrictions on control.
Resource Efficiency	**Less efficient due to the need for a full OS per VM.**	**More efficient, as they share the host OS and isolate user spaces.**	**Highly efficient as only the execution time is billed.**
Pros	Offers complete resource control per VM.	Containers are optimized for performance with shared resources.	Only pay for the exact resources used during execution.
Cons	High resource usage and inefficiency due to full OS setup.	Needs container orchestration tools like Kubernetes.	May not handle stateful or long-running tasks well.
Scaling	**Manual scaling required; typically requires creating and managing more VMs.**	**Automatic scaling with tools like Kubernetes but still requires managing container clusters.**	**Auto-scaling based on demand with minimal management.**
Pros	Can scale based on specific needs with individual VMs.	Scaling is easier with container orchestration tools like Kubernetes.	Automatically scales based on load without manual intervention.
Cons	Scaling requires resource management and can be inefficient.	Requires tools and expertise for scaling and managing clusters.	Cold start latency during scaling; may not be optimal for large-scale apps.

Perspective	**Virtual Machines**	**Containers**	**Serverless**
Maintenance	**Requires regular OS updates and maintenance for each VM.**	**Less maintenance than VMs but still requires monitoring and updating of containers.**	**No maintenance of infrastructure required by developers.**
Pros	Easier to manage from an individual VM perspective.	Less maintenance compared to VMs, with automatic container health checks.	No infrastructure management is required.
Cons	High maintenance overhead; requires updates for each VM.	Still requires updates for security patches and monitoring.	Developers don't control underlying infrastructure, so may face platform limitations.
Portability	**Less portable due to reliance on specific hypervisor setups.**	**Highly portable, can run on any platform that supports the container runtime.**	**Highly portable, as long as the serverless platform is supported.**
Pros	Full control over the environment, ideal for legacy systems.	Can run on any platform supporting containers (e.g., Docker).	Works on any supported cloud provider, with no setup required.
Cons	Tied to specific hypervisor environments, limiting portability.	Requires the underlying system to support the container runtime.	Some degree of platform lock-in; not as portable across providers.
Performance	**More overhead due to running full OS and virtualization layers.**	**Lightweight with lower overhead, offering better performance than VMs.**	**Potential latency due to cold starts, but high scalability.**

Perspective	**Virtual Machines**	**Containers**	**Serverless**
Pros	Full control, stable environment for running complex apps.	Better performance due to lower overhead compared to VMs.	Scalable, handles high-volume traffic with ease.
Cons	High overhead reduces performance efficiency.	May not be suitable for legacy systems that require a full OS.	Cold starts cause delays; performance may vary based on provider.
Use Case	**Ideal for running legacy applications or complex enterprise systems.**	**Best for microservices, modern web apps, and cloud-native applications.**	**Ideal for event-driven applications, APIs, and short-lived tasks.**
Pros	Best for legacy apps or large, complex enterprise environments.	Best for modern applications that require quick scaling.	Best for short tasks, event-based triggers, and APIs.
Cons	Not ideal for microservices or lightweight apps.	Not suitable for legacy applications that need full OS features.	Not good for long-running or stateful applications.

Thank you. 😊