PLAGIARISM DECLARATION

THE UNIVERSITY OF THE WEST INDIES
The Office of the Board for Graduate Studies
INDIVIDUAL PLAGIARISM DECLARATION

STUDENT ID: 816003022

COURSE TITLE: CLOUD TECHNOLOGIES

COURSE CODE: COMP 6905

TITLE OF ASSIGNMENT: ASSIGNMENT I

This declaration is being made in accordance with the **University Regulations on Plagiarism (First Degrees, Diplomas and Certificates**) and must be attached to all work, submitted by a student to be assessed in partial or complete fulfilment of the course requirement(s), other than work submitted in an invigilated examination.

STATEMENT

1. I have read the Plagiarism Regulations as set out in the Faculty or Open Campus Student Handbook and on University websites related to the submission of coursework for assessment.

2. I declare that I understand that plagiarism is a serious academic offence for which the University may impose severe penalties.

3. I declare that the submitted work indicated above is my own work, except where duly acknowledged and referenced and does not contain any plagiarized material.

4. I also declare that this work has not been previously submitted for credit either in its entirety or in part within the UWI or elsewhere. Where work was previously submitted, permission has been granted by my Supervisor/Lecturer/Instructor as reflected by the attached Accountability Statement.

5. I understand that I may be required to submit the work In electronic form and accept that the University may subject the work to a computer-based similarity detention service.

NAME: AADIDEV SOOKNANAN

SIGNATURE: [illegible]

DATE: 2021-SEP-19

Question I

a. **What is cloud computing** (3)

Cloud computing is the act of running workloads within computing environments that abstract, pool and share scalable resources across a network (RedHat blogs). Depending on the level of abstraction being utilized/provided by a given cloud, a cloud environment may or may not be aware of the specific hardware configuration used to serve varying cloud functions. Currently, cloud encompasses a broad range of services across many levels of abstraction, where the main feature touted for cloud is the on-demand delivery of IT resources over the internet with pay-as-you-go pricing. To expand upon this, cloud-computing leverages the ability of *hyperscaling*, whereby a cloud environment can expand or contract depending on the computing needs, without the need to purchase physical, local servers (in the case of a public, over-the-internet cloud service such as GCP, AWS, Azure, etc). Computing services use *virtualization* in many forms to provide pooled resources based on runtime and forecasted load and feature requirements. These computing services are typically grouped into three types of as-a-Service solutions, namely: IaaS (Infrastructure), PaaS (Platform) and SaaS (System), depending on the level of services offered by the cloud providers, ranging from bare-metal servers to fully-fledged virtual operating-systems.

b. **What's the difference between multi-cloud and hybrid cloud** (4)

A multicloud includes more than 1 cloud (whether private or public), that may or may not be networked together. A hybrid cloud is a computing environment which may seem to be a single, uniform entity to the end user/developer. However, a hybrid cloud is built from multiple environments connected through a variety of networks and/or VPNs. For example, a hybrid cloud may connect a public cloud environment to a private cloud environment by means of tunneling or an authenticated API. This may be necessary due to privacy, computing or legal requirements. Therefore every hybrid cloud is a multicloud (according to RedHat developer blogs), but not every multicloud is a hybrid cloud. A multi-cloud may exist on purpose (for redundancy, privacy, performance by creating striped RAID arrays) or by accident. A hybrid cloud is built to allow full portability of applications (hence a near unified user/developer experience), and has recently shifted away from the API-connection approach to focus on cross-environment applications (those which work on all components of the hybrid environment, taking advantage of, and being restricted by the idiosyncrasies of each).

Question II

a. Describe what are IaaS, PaaS and SaaS and the relationships among them (12)

IaaS, PaaS, SaaS and FaaS are used to describe differing levels of abstraction in cloud computing. 'As-a-service' refers to the fundamental difference between the architecture behind cloud-based and traditional development. In the conventional tradition, an organization consumes all assets (hardware, development tools, applications) by means of rote-purchase. It is up to the organization/end-user to install, manage and maintain all IT-related resources in an on-premises data center. In cloud-computing, a cloud provider owns, manages and maintains the assets (IBM Cloud). The customer consumes these resources via an Internet connection, and typically pays for them using a rolling subscription or on a pay-as-you-go basis. According to AWS Developer Blogs, the chief advantage of *as-a-Service* architecture is economic; a customer may access and scale their required resources as it needs without the expense of purchasing and maintaining everything in its own data center.

Following a tree of abstraction, SaaS is the highest level, followed by PaaS, IaaS and having on-site infrastructure, managed fully by internal teams would be the lowest, most cost-intensive, least adaptable model. IaaS provides an infrastructure environment, PaaS provides platform tools (OS, etc), SaaS provides complete applications
SaaS, PaaS and IaaS not mutually exclusive, and is chosen depending on management ease offered and level of flexibility required. Technically speaking, the exact same things can be done with all three, with differing levels of management granularity

i. **IaaS** - Infrastructure-as-a-Service is the lowest level of abstraction (when compared to PaaS and SaaS). An IaaS contains basic building blocks for cloud operations by providing access to networking features, computers via virtualization and data storage space, whilst users handle applications, data, operating system, middleware and runtimes. Virtualization is the creation of multiple, simulated environments or dedicated resources from a single hardware system. A hypervisor such as Oracle VM, or the KVM connects directly to hardware and distributes the physical resources amongst multiple, independent environments. There are three main categories of virtualization, namely network functions virtualization (or the isolation of guest virtual networks from a host network), server virtualization (where a single server can be made to function like a more than a single server) and OS virtualization (1 computer running multiple operating systems). The IaaS provider implements software-defined networking which abstracts networking resources into a virtualized system. This separates network forwarding functions from network control functions. Users don't have to manage on-premise resources

An IaaS user has complete control of infrastructure through an *application-programming-interface* (API). IaaS provides highest level of flexibility and management, whilst performing load balancing, etc and other bare-metal functions. IaaS users can choose between Virtual Machines hosted on shared physical hardware, where customers provision, configure and operate the servers. It is the most similar to having in-house resources.

Few common examples of IaaS are RedHat OpenStack, GCP Compute Engine and Linode

ii. **PaaS** - *Platform-as-a-Service* is one level of abstraction above IaaS. Hardware and an application-software platform are provided /managed by a given service provider in addition to the services offered through an IaaS. The user handles actual applications and data via a GUI. This makes the PaaS the most popular for application development, as an OS (or other runtime such as the JRE, Oracle VM, etc) is provided unlike in the case of the IaaS, and more importantly, maintained by an external service provider. This allows developers to focus on application development, without needing to manage bare-metal resources. As such, this level of as-a-Service requires less management than an IaaS.
Common examples of PaaS are: GCP App Engine, AWS Elastic Beanstalk and RedHat OpenShift on IBM Cloud

iii. **SaaS** - *Software-as-a-Service* is the highest level of abstraction, and typically delivers a web application. This eliminates the need to have an application installed on each individual machine. SaaS users do not carry the responsibility of maintaining infrastructure, platforms and software. SaaS applications use multi-tenant architecture, where a provider handles bug fixes, updates and general app maintenance. This reduces upfront costs by eliminating the need to permanently purchase software
Common examples of SaaS are Office 365, Google Docs and Adobe CC.

If Cloud were a Formula One racing team, an IaaS would be the equivalent of having the car chassis alone. You cannot go against regulations and modify the chassis (analogous of not having access to the cloud provider's hardware), however you must provide your engine and driver (or operating system), racing team (software applications) and must ensure that your car pits to change tires. One level above this is

PaaS, where you receive the entire car and driver, all need be done is supply fuel and racing team. Above this is SaaS, where an entire Formula 1 team is purchased, car, track and all. All this user may be concerned about is strategy, and which Grand Prix to enter, or which drivers to use, with little to no concern regarding driver training, or type of tires to use based on track conditions, as the team automatically knows how to scale according to weather and track.

b. What is Function-as-a-Service (FaaS) (3)

IaaS, PaaS and SaaS are resource-driven, where an end-user pay for resources directly. Function-as-a-Service is an event-driven execution model, where an end user pays for events (functions triggered by predefined things happening on the application-end). An FaaS runs in a stateless container, with no stored knowledge of or reference to past transactions (each transaction made as if from scratch).

Functions manage server-side logic, and is a way of implementing serverless computing, where developers write/implement procedures guided by organizational needs which are then executed in Linux containers.

If a function is a piece of software running business logic on an OS, then an application is made from many functions. Applications may automatically scale up and down with demand. If several requests for a function occurs simultaneously, the FaaS typically abstracts and automatically creates as many copies of the function as needed. Additionally, an FaaS gives abstraction for running web applications in response to events without managing servers. It does not require server process to run constantly in the background (as in PaaS) and can be triggered from SaaS applications. For example, an AWS Lambda function can be executed by means of an API call from an AWS Application. Other examples are IBM Cloud Functions, GCP Cloud Functions and OpenFaaS

c. **Why use containers** (3)

Containers provide a standard way to package application’s code, configuration and dependencies into a single object. A container provides a consistent, portable software environment, since the entire runtime is packaged. An OS is typically shared and installed on a server, and a container is then run from a container *image*. A container typically uses less resources than a traditional hardware VM since an OS is not included in the container. A container can be easily deployed to multiple OS’s and hardware platforms. Additionally, the *separation of concerns* is satisfied: developers focus on applications and the operations team focuses on infrastructure. Containers can also be made highly secure by means of strongly isolating containerised environments from each other. Containers can be built, tested and deployed with Docker, whilst Kubernetes handles container orchestration (scheduling and running containers on clusters of physical or virtual machines). Containers can also be used for microservices, batch processing (containers can be duplicated on-demand to match number of requests,) PaaS, etc

Question III

a. What is virtualization ? (5)

Virtualization allows the creation of multiple, simulated and possibly isolated software environments from a single, physical hardware system. A hypervisor connects directly to the hardware and handles the splitting of one system into separate environments known as virtual machines. The hypervisor (also known as the *Virtual Machine Monitor VMM)* is responsible for the distribution and allocation of system resources by treating the host machine's physical hardware as a pool of redistributable resources. The host system's hardware performs actual computation, whilst the hypervisor manages the schedule and splitting of varying tasks in terms of time and resources. For example, Xeon W, Threadripper Pro and EPYC Rome have upwards of 32 physical cores with hyperthreading, and can handle in certain instances, *terra*bytes of system memory. Additionally, server-level CPUs typically have far more expansion slots (in the form of PCI-E lanes), which has traditionally handled high-speed networking, video-compute resources (GPUs and other specialised hardware such as RED's dedicated compute card) and of recent, high-speed NVME storage. To compound this, modern, real-time operating systems employ complex scheduling schemes. This host of abundant resources is used by the hypervisor, which scales storage, memory, etc based on user needs. A single server may host multiple users concurrently requiring high amounts of comput resource. Additionally, a hypervisor allows multiple operating systems to be run alongside each other . There are two main classes of hypervisor:

i. **Type I -** *bare-metal* runs directly on the host machine's hardware to manage guest Operating Systems. It completely replaces the host's OS and Virtual Machine resources are scheduled directly to hardware. This is typically used in an enterprise setting or other server-based environments, where a fully-featured host operating system is not required. *Examples of these are Microsoft's Hyper-V and the KVM.*

ii. **Type II** - *hosted* runs on a conventional operating system as a software layer application. Thi abstracts guest operating systems from the host, and is better for individual users who which to run multiple operating systems on a personal computer. *Examples of these are Oracle VirtualBox and VMware Workstation*

b. Describe the types of virtualization (12)

There are four main types of virtualization

i. **Network Functions Virtualization** - this is the virtualization of traditionally hardware-based networking services, such as routers, firewalls and load balances. These are wrapped into virtual machines and run on traditional server hardware, which de-necessitates traditional proprietary networking hardware. Dedicated hardware is not required for each network function. According to the European Telecommunications Standards Institute (ETSI), network functions virtualization consists virtualized network functions (software applications which deliver network functions such as file sharing, directory services, etc), Network Functions Virtualization Infrastructure (computer, storage, networking) and management, automation and network orchestration (framework for managing the entire virtualization networking infrastructure and provisioning news virtualized network functions). Network Functions Virtualization allows a service provider to run network functions on standard hardware, and allows a single server to take the place of what was traditionally multiple, dedicated hardware devices. This benefits both the up-front costs of purchasing hardware, resulting in an overall reduction in physical space required, power and overall cost

ii. **Operating System Virtualization -** this is a server-virtualization method where the kernel of an operating system allows for multiple, independent user-facing environments on a single hardware implementation. This is typically used in virtual hosting environments, where the need to securely isolate multiple users on the finite hardware is required. OS-level virtualization reduces bulk hardware costs, increases security (since all individual environments can be isolated and monitored) and limits the time needed to manage individual environments, since this can typically be scheduled and batched by the host operating system.

iii. **Server Virtualization -** As aforementioned, the latest Intel Xeon W-3375 has 38 cores and supports up to four Terabytes of system memory, whilst the latest AMD Epyc-7763 has 64 cores and also supports up to 4TB of system memory. With this incredible amount of raw compute, modern server applications typically use mere fractions of the available hardware capability. Additionally, size and power requirements for a data-center are typically at a premium. Server Virtualization is where a single physical server into multiple virtual servers, capable of running its own operating system. This saves space through consolidation and allows running of the same application on multiple virtualized servers, which may minimize the interruption in service, if, for example, a single virtualized server fails. It additionally offers developers isolated, independent systems to test new applications or operating systems, without the need to purchase additional hardware. Owing to the isolated nature of virtualized servers, this additionally means that other applications are not affected by these new, potentially unstable software-in-testing. This isolation additionally allows the continued support of legacy applications on ever more modern hardware, where a a legacy environment can be simulated on newer equipment. This thereby reduces the need to completely rewrite software to run on updated hardware and saves the cost of needing to keep legacy, potentially inefficient, purpose-built hardware at a data center

iv. **Data Virtualization** – this is the consolidation of data into a single (apparent) source. Data virtualization delivers data from multiple sources to users at the right time to any application or user as required. It reduces the need to copy data from multiple sources into a central data store (such as a data lake or warehouse), which is costly and prone to error, since no data need be manually siloed for any particular application. Certain data virtualization providers additionally implement implicit load-balancing and encryption (such as IBM's Cloud Pak solution). Data nodes are automatically self-organized into a collaborative network depending on storage and privacy requirements, and can be automatically integrated into a previous Cloud Pak environment.

v. **Desktop Virtualization** – this allows a central administrator to deploy simulated desktop environments to multiple physical machines at once. This allows the mass update, configuration, and security machines at once. Additionally, this allows users to work from virtually anywhere in within a network, without regard for the device or operating system installed on individual machines. Desktop virtualization runs via a client-server model, where the provider's chosen OS and applications run on a server in a data center or in the cloud. This allows potentially automatic redistribution of physical resources based on user needs, enhanced security by removal of the entire operating system environment from the user devices, and enabling of a remote workforce, since new user desktops can be provisioned quickly and become instantly available for new users to access.

c. What is virtualization management (3)

Virtualization management is software which is responsible for provisioning and administering physical resources to virtual environments. Virtualization management software typically provide a simple interface for creating, monitoring and setting up reports and rules for virtual environments. According to RedHat blogs, virtualization management consists four main tasks:

Provisioning - processing resource request, creating and configuring Virtual machines. This function is listed separately from “operating” as defined by RedHat, however I believe that provisioning and operating are both sides of the same proverbial coin, as both deal with on-demand reallocation of physical resources.

Operating - reclaiming unused resources, investigating bugs and projecting future needs

Compliance - setting up isolated, secure environments, identifying issues and validating user access. This function is essential for many purported advantages of virtualization. Without proper isolation, risky software may interfere with lower-level software, and can pose safety risks.

Hybrid unification - mapping provisioning, compliance and operations across virtual, public, private and containerised environments. In other terms, an umbrella term which encompasses higher-level management of the aforementioned functions.

d. Are virtual machines just containers?

In traditional virtualization, a hypervisor provisions physical hardware, where each guest machine may contain a guest OS, copy of hardware and associated libraries and dependencies. Multiple virtualized operating systems can be run on the same physical server. Containers virtualize only the operating system, as opposed to the underlying hardware. Each individual container contains only the application and its libraries and dependencies. Containers do not need to include a guest OS in every instance and can leverage the resources and features of its host OS. Containers allow a finer detail of granularization of resource-provisioning by means of the microservices architecture. A full-blown VM is typically significantly larger than a containerised application, which allows a VM to perform multiple resource-intensive tasks at once. A container on the other hand is purpose-built to be lightweight and highly portable. This allows workloads to be broken down into the smallest possible serviceable units, packaged as containers, in support of upcoming practices such as CI/CD and DevOps. Containers should be used when building cloud-native applications, packaging microservices, implementing CI/CD and when scalable projects must be distributed across a wide range of user-environments which share an OS. VMs on the other hand, can run far more simultaneously than a single container, and are best suited for housing traditional monolithic legacy applications, isolating risky development cycles, provision infrastructural resources or where another OS need be run inside a host OS.