Mattia Danese
Homework #3
CS118 - Cloud Computing
Professor Sambasivan

Part One:

Question 1

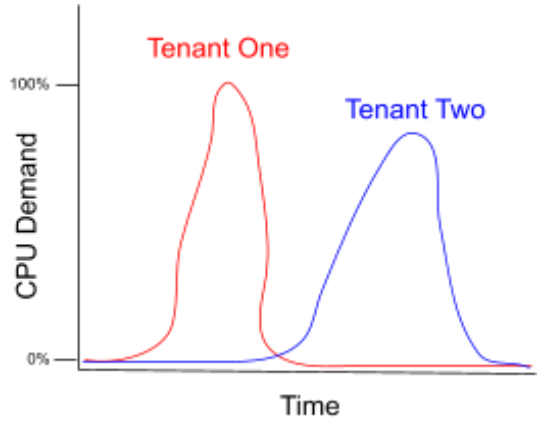
By definition, virtualization is the partitioning of underlying resources (of a host machine) while giving entities (the guest instances) the illusion that they have the same dedicated access and resources as if they were the only instance running on the physical (host) machine. As such, virtual machines are a key enabler of cloud computing because they allow for various applications to run on one physical machine which, in turn, greatly increases the practicality of remote computation on "the cloud". Being able to run multiple instances and/or applications on the same piece of hardware minimizes client-side functionality otherwise required and allows for more scalable/diverse server-side functionality (i.e. client no longers needs to worry about doing *x* and *y* since VM is running server *a* that can do function *x* and server *b* that can do function *y*).

Question 2

The difference between an *Infrastructure as a Service* (IaaS) cloud and a *Platform as a Service* (PaaS) cloud is that an IaaS cloud regards building and hosting the framework of a cloud-system whereas a PaaS cloud regards the functionality and client-end software (i.e. an API) of a cloud-system. In other words, IaaS cloud allows for a user (in most cases an admin) to configure the resources of their platform while PaaS cloud takes care of this configuration for users and therefore allows users to focus more on the API functionality of their platform.

Two *Infrastructure as a Service* cloud technologies are: AWS and Microsoft Azure.
Two *Platform as a Service* cloud technologies are: Heroku and Google App Engine.

Question 3



Question 4

Two pieces of information about the applications running on each VM that would help the scheduler decide which VMs should and which should not be co-located on the same host machine are: their functionality (computation-intensive or storage-intensive) and their traffic. Knowing the functionality of the applications can be helpful because this could indicate if the two applications are generally very computation-intensive (they frequently require the CPU for extended periods of time) or are generally very storage-intensive (they frequently need to store data or store a relatively large amount of data). In either case, co-locating two applications on the same host that are both computation-intensive or that are both storage-intensive is most likely not a good idea because it is reasonable to assume that there will be a high chance of either high host-CPU competition, in the former case, or not having enough physical storage for both applications, in the latter case. Knowing the traffic of the applications can also be helpful because an application with high traffic will most likely require much of the host CPU quite frequently. Thus, co-locating two applications on the same host that both have a great amount of traffic is most likely not a good idea since it is reasonable to assume there would be high host-CPU competition.

Question 5

(c) No.

No, it is not possible for log *1.2* to be committed in the future because, in RAFT, logs can only be committed when majority node consensus is satisfied. So for log *1.2* to be committed, a node with the log *1.2* must be elected leader, it then must propose log *1.2* to all participants, and the majority of participants must agree with log *1.2* (i.e. have log *1.2* in their own logs). Since Node 3 is the only node with log *1.2*, even if it is elected as the new leader and proposes log *1.2*, there cannot be a majority consensus because no other node contains log *1.2* in their logs; thus log *1.2* cannot be committed. If any other node is elected as the new leader, then log *1.2* will not even be proposed (since it will not be in the leader's logs). Therefore, it is not possible for log *1.2* to be committed in the future.

Question 6

Yes, a coordinator can still commit a transaction even if one or more participants fail after responding affirmatively. Since the coordinator received all participant responses before any participants failed (it is assumed participant failures only occurred after they sent their response to the coordinator), the coordinator will either abort or commit (depending on if it received all successful responses or not). Once the participants that failed recover, they will identify that the prepare message they received (the one that they responded affirmatively to) does not have a corresponding abort/commit message (the final message from the coordinator which informs the participants of the end result of the current transaction) in their logs. As such, these participants can query the coordinator for the end result and computation can continue as normal.

Part Two:

Question 1

Docker (a container) on Mac or Windows running Linux containers is possible because Docker runs a virtual machine and *that* virtual machine runs the Linux containers. To the user, it appears as if the Linux containers are running directly on the host OS (i.e. Mac or Windows); in reality, though, the Linux containers are being run by a virtual machine which is being run by Docker which is being run by the host OS.

Question 2

A problem that may arise without containers is that an application will not be able to run in an environment (i.e. physical machine, container, or virtual machine) if the correct versions of its required libraries and packages are not installed. Thus, Container 1's application cannot run in the same environment as Container 2's application (i.e. they cannot run in the same container or on the same machine) since they depend on different versions of the same library. Containers can solve this problem by being able to abstract the libraries and packages "installed" such that each application is only aware of its own versions of libraries and packages, and is not aware of those pertaining to other concurrent applications. Therefore, the application in Container 1 and the application in Container 2 can both run concurrently on the same host machine, even though they require different versions of libraries and packages (or different libraries and packages allogether), because Container 1 and Container 2 will only have the libraries and versions needed for the respective application running in them.

Question 3

Running an application in a container should have lower overhead in terms of resources consumed than running it in a virtual machine. Since the container abstracts the operating system, it uses the same kernel as the host machine while the virtual machine abstracts memory and needs its own kernel (separate from the host machine kernel). Thus, running an application in a container deals with just one kernel (the host kernel) while running it in a virtual machine deals with two different kernels (the host kernel and the VM kernel). Therefore, it is clear that less resources are consumed and less overhead is present when running an application in a container rather than a virtual machine.