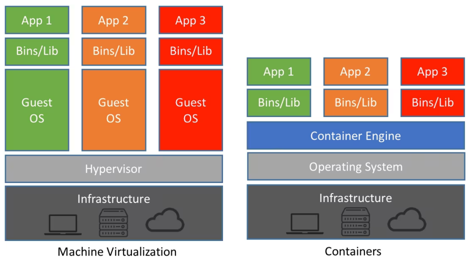
Challenge Question: Write a short essay (at least 500 words) comparing virtual machines and containers. Make sure to discuss if there are any security implications with using either one of these options. Make sure to cite any references that you use and feel free to use a diagram similar to Figure 3.8 or 3.9 in the textbook.

Virtual machines and containers are very different things, though can appear in a very similar way to the end user. Both typically have an IP address that can be used to access them, and various services can be installed on both.

Virtual machines tend to be bulkier, as they include their own kernel separate from the host operating system. This allows more flexibility due to fewer dependencies on the host operating system when compared to containers. In order to run the guest operating system/virtual machine, there needs to be a virtual machine manager. These can be layer one or layer two hypervisors, with the main difference being whether or not there is a host OS or if the hypervisor runs directly on bare metal. In general, the hypervisor for a virtual machine will translate the system calls made inside the VM to whatever the host OS can understand or needs. Virtual machines typically need to be allocated resources, and depending on how the virtual machine manager works, *surplus* resources (such as disk space, memory, processing power, or acceleration cards like GPUs) may not always be shared with other guest operating systems. Sometimes it is even difficult to share a GPU for processing power. Other things, like RAM tend to be easier to share, but again, it is not uncommon to allocate 8 gigabytes of memory to a VM and for that VM to keep all 8, even while it is idling at 3 gigabytes.

There are two main types/use cases of containers: containerized services and linux containers (LXC). Docker and podman are common runtimes used to manage and run containerized services. For this use case, containers would be pre-packaged with all dependencies and libraries required to run a particular application. This gives the ability to simply install docker or podman, and just download and run the service container on top of that. It is all packaged up and ready to go as is. LXC containers are more commonly used for manual installs of operating system environments where resources are restricted or more fluidity is desired. The last major difference that sets containers apart from virtual machines is the fact that they not only need to run on top of a guest OS but that they share the kernel with their guest operating system. Because of this, resources can be shared between containers and resource management is much more fluid. You typically do not allocate RAM for a docker container; it just uses memory similar to a process.

Virtual machines are generally thought to be more secure than containers. This is because virtual machines typically have what is basically emulated hardware. The hypervisor emulates a BIOS for the guest OS, and it is difficult to escape from this low level environment (assuming an attacker is able to make it to this low level code). A container, on the other hand, shares resources with the guest OS. This makes a system of containers more lightweight to run (see the figure below) when compared to virtual machines, but due to the shared kernel, can open the door to a compromised container potentially having the ability to compromise the host operating system.



(https://www.netapp.com/blog/containers-vs-vms/)

- https://www.atlassian.com/microservices/cloud-computing/containers-vs-vms
- https://www.ibm.com/cloud/blog/containers-vs-vms
- https://www.netapp.com/blog/containers-vs-vms/

Short Answer Questions

1) Now that we have secure communication, if student A has the server program running on machine P, could student B use his client program running on machine Q to connect to A's server program if student B knows the port on which student A's server is running?
    a) If your answer to question (1) is yes, then explain how to prevent this from happening.

        The only vulnerability on my server regarding session hijacking would be for a client server to attempt to establish the secondary connection directly to a port after it has been assigned to another user but before the other user's client establishes the new connection. This would be very unlikely as the port number sent during the initial transaction is encrypted, but technically not impossible.

    b) If your answer to question (1) is no, then explain why this is not possible and what is necessary to support this option.

        This is largely not possible because the server only accepts one client for each unique connection (after the primary port assignment server). It would not make sense to allow additional users to connect to an existing client's server and it would require a complete overhaul of the system. It may be possible to send unwarranted data to the port, though now that encryption is being used, there would be no way for the server to use/decrypt the information in a meaningful manner.

2) If we were to modify the server program such that we could execute the server program on multiple computers instead of one computer, what changes would be required to the server program and the client program? Write down the changes required, if any, for both server and client programs (no need to implement any of these changes, just describe this in the report).

    The client program would work as is, other than the fact that it would need to run on a machine that has the root CA installed. Currently, it trusts the self-signed root CA certificate PEM file each time at runtime.

    The server would need a large overhaul- the current method of communicating between the controller thread and each client_connection thread is through python multithreading/thread-safe queues. I essentially built a basic publisher-subscriber using these, and because threads can share memory, this is how they communicate. If each client_connection was to be moved to a new container or machine, there would need to be a pub-sub framework that works over sockets. This could be done using something like redis or by overcomplicating the primary controller. The LIST command would also require some simple changes, such as the controller telling the client_connection process the list of users instead of reading it directly from memory.

Feedback Questions

1) Was this homework too difficult, or too easy?

   It was alright. There were less changes to be made when compared to homework 1 which required building the entire application. The essay question above was frustrating though as I was focussing on the actual assignment, and right when I finally get it and think I'm done, I realize that I have a lot more work to do.

2) Was the assignment fun or challenging?

   It was fun but annoying to troubleshoot as I made an important mistake when generating the root CA's common name and I spent more time trying to troubleshoot code when I should've been looking at the CA chain.

3) Was there something that was unclear?

   The short answer question 1 was confusing. I think I know what was being asked (similar to a homework 1 question) but it was worded weirdly this time.

4) Was the homework too long for the given amount of time?

   It was alright, though the extra 500 word essay was a considerable chunk considering I was focussing on the code and getting the server to work, and then I go to the project spec document and see that I have to do a lot of additional unrelated work.

5) What did you learn from this homework?

   I learned how to convert standard Python sockets to use SSL/TLS. I didn't realize how efficient it ends up being, and looking back on it now, it makes a lot of sense with regards to setting up the ssl context and why those parameters are important (self-signed certificate chain of trust).