Lecture scripts (draft)

1. Concepts of containerization
   1. Why containerization? Tradeoffs? Security?
      1. A container is a standard unit of software that packages up code and all its dependencies so the application runs quickly and reliably from one computing environment to another. Show images illustrating isolation. Show image [here]( https://www.docker.com/resources/what-container)
      2. Security is defined by confidentiality, integrity, and accessibility. This is commonly referred to as the CIA triad in software security. Containerization isolates resources into an environment of relevance. Separation of concern is a large part of this aspect of relevance.
         1. Confidentiality: an application should be clearly and separately defined f rom infrastructure. This is how ops secures sensitive secrets necessary to running an application.
         2. Integrity: isolation from VM host. Prevents users from exploiting security holes in OS, database manipulation.
         3. Isolation from other moving parts of an application, minimizes privileges of each part of application.
   2. Layers of abstraction
      1. Layers of abstraction are present in docker images. As we discussed in the previous video/earlier, we discussed how these images are built with the least privileges necessary. Concepts of secrets, dependent applications, HTTP servers, databases, etc. are abstracted away from an application.
      2. Containerization abstracts all of these other necessities for shipping to production. It allows the developer to focus solely on developing the application. Even the operating system is abstracted away from the application, making it possible for developers to maintain consistent development environments across Windows, Mac OS, Linux, etc.
   3. vs virtual machines (case study ASG vs ECS)
      1. Containers and virtual machines have similar resource isolation and allocation benefits, but function differently because containers virtualize the operating system instead of hardware. Containers are more portable and efficient. Show [image]( https://www.docker.com/resources/what-container).
      2. ASG - https://docs.aws.amazon.com/autoscaling/ec2/userguide/AutoScalingGroup.html
      3. ECS - https://docs.aws.amazon.com/AmazonECS/latest/developerguide/Welcome.html
2. Building images
   1. What’s a Docker image
      1. Install docker on your machine
      2. Go over Docker images as stated in the [docs]( https://docs.docker.com/v17.09/engine/userguide/storagedriver/imagesandcontainers/). Discuss layers of images
      3. Briefly introduce Dockerfile’s. To be further described later
   2. Image size, hosting images
      1. Toolset https://docs.docker.com/engine/reference/commandline/images/. Go over image size and importance. Show hosting options here on Docker hub. Go over pushing/pulling
   3. Dockerfile, using docker images
      1. Example of simple docker file. Reference [docs]( https://docs.docker.com/engine/reference/builder/). Build image
3. Using containers
   1. Images vs containers, ship-ability
      1. Discuss differences between images and containers. Go over containers as discussed in the [docs]( https://docs.docker.com/v17.09/engine/userguide/storagedriver/imagesandcontainers/#container-and-layers). Reference back to docker hub and pulling/running a container in one command
   2. Starting containers from images
      1. Dive deeper into `docker run` and [docs]( https://docs.docker.com/engine/reference/run/).
   3. Using containers
      1. Go over `docker ps` and [docs]( https://docs.docker.com/engine/reference/commandline/ps/).
      2. Starting, stopping, committing, and executing commands on existing containers. Container management/clean up.
4. Volumes
   1. Docker volumes
      1. Go over docker volumes conceptually, reference [docs]( https://docs.docker.com/storage/volumes/).
   2. Permissions, mount types
      1. Filesystem permissions (brief)
      2. [Bind mounts]( https://docs.docker.com/storage/bind-mounts/)
      3. [Tmpfs mounts]( https://docs.docker.com/storage/tmpfs/)
   3. Using docker volumes
      1. Go over toolset with `docker run` and flags for mounting volumes. Use case of sharing mount with host and docker container
5. Networking
   1. Docker networks
      1. Go over Docker networks conceptually, describe how they are virtual networks, reference [docs]( https://docs.docker.com/network/)
   2. Types of networks
      1. [Bridge]( https://docs.docker.com/network/bridge/)
      2. [Overlay]( https://docs.docker.com/network/overlay/)
      3. [Host]( https://docs.docker.com/network/host/)
      4. [Macvlan]( https://docs.docker.com/network/macvlan/)
   3. Using docker networks
      1. Go over toolset in [docs]( https://docs.docker.com/engine/reference/commandline/network/)
6. Services
   1. Concepts
      1. Microservices vs monolithic applications, reference docker [docs]( https://www.docker.com/solutions/microservices).
      2. Illustrate advantages and concepts, reference useful [medium article]( https://timber.io/blog/docker-and-the-rise-of-microservices/)
   2. Orchestration – docker-compose
      1. Introduce docker compose, reference [docs]( https://docs.docker.com/compose/)
      2. Introduce application composed of redis, nginx, server, static client (S3)
   3. Deploying to production – ECS
      1. Discuss (conceptually) implementation of docker-compose file to ECS