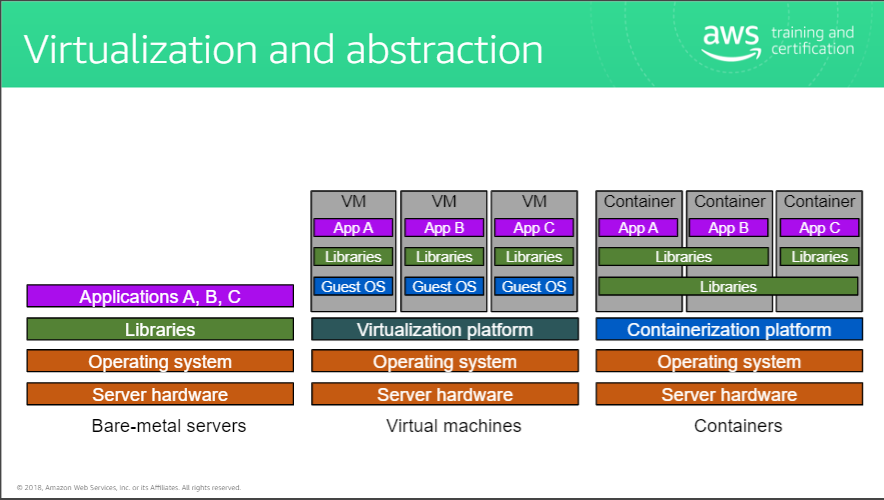
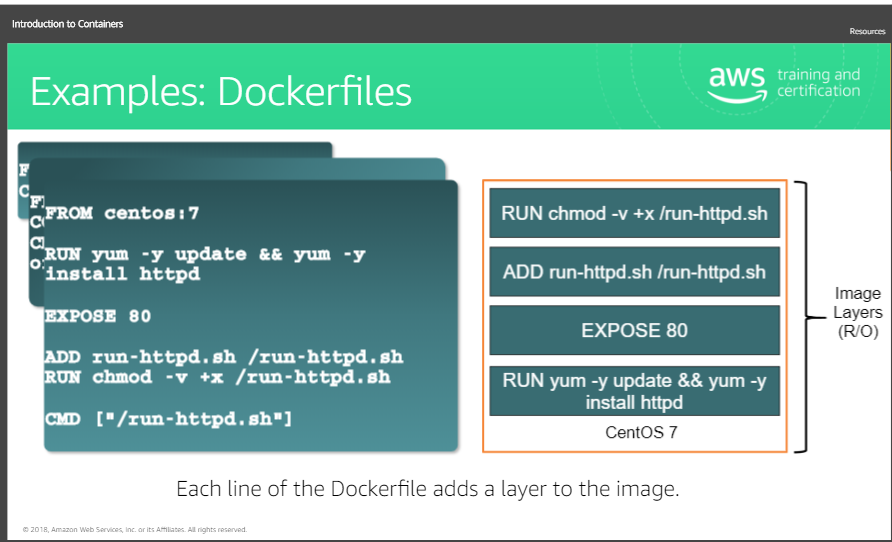
AWS

Container

* Shipping with containers as a parallel
  + Oddly shaped
  + Could not preplan for cargo
  + Standardization of containers started hundreds of years ago
  + There is a current standardized international shipping standard container
  + By focusing on the container instead of individual cargo we improved efficiency and price
* Virtual world
  + Standardized unit of software designed to run quickly and reliably
  + Containers are a form of virtualization run at the os level
  + Containers contain everything needed to run everything including server, os, application, libraries etc
* How do containers differ from other forms of virtualization
* 
* Abstraction – current servers (issues with application sharing libraries and shared resources i.e what if one app requires updated library)
  + Application ABC
  + Libraries
  + Operating system
  + Server hardware
  + Bare-metal servers
* Abstraction – VMs (no longer an issue with shared libraries)
  + VM (3 apps running in separate VM, this does carry some overhead of 4 operating systems vs one which means more patching, updating, redundancy and space taken)
    - Application A
    - Libraries
    - Guest OS
    - Application B
    - Libraries
    - Guest OS
    - Application C
    - Libraries
  + Guest OS
  + Virtualization platform
  + Operating System
  + Server Hardware
  + Virtual Machines
* Abstraction – Containers
  + Container (3 apps running in spate containers, lightweight efficient and fast)
    - App A
    - App B
    - App C
    - Shared Libraries (can share and can be isolated as needed)
  + Containerization platform
  + Operating system
  + Server hardware
  + containers
* Docker
  + Lightweight container virtualization platform
  + Provides tools to create, store, manage and run containers
  + Easily integrates with automated build, test, and deployment pipelines
  + Benefits
    - Portable runtime application environment
    - Package application and dependencies in a single immutable artefact
    - Ability to run different application versions with different dependencies simultaneously
    - Faster dev and deploy schedule
    - Better resource utilization and efficiency
  + Image
    - Definition
      * Read only template with instruction for creating a container
    - Can build from scratch or can use template
    - Image is usually based on another image with customization
    - Ie. Can build ubuntu image but the template with build instructions will build out web server and application in addition to library to make it run
    - To create your own image you create a Docker file
      * 
    - Image Layers
      * Each line of the Docker file adds a layer to the image
      * If Docker File is changed and container already built then only the layer that is changed will be updated
        + This is what makes docker lightweight
  + Images vs. containers
    - Container Image is read only immutable template that is highly portable
      * Can port to any environment that supports docker
    - Container is an instance of that image
      * You can spin up one or many of that image
      * Every container has a thin r/w layer
      * Most of container is read only because of the layers
      * The underlying image remains unchanged
      * Its why multiple containers can share an image
      * Upon destroying the container, the read/write also gets destroyed
* Microservices
  + What are they?
    - Architectures and organizational approach that fosters innovation and ownership of software applications
  + Traditional vs. microservice architectures
    - Microservices are designed to speed up deployment cycles
    - Improves maintainability and scalability of applications
    - Monolithic architectures
      * Processes tightly coupled and run as a single server
      * If one spike happens entire architecture will require enhancement and scaled
      * Adding features gets more complex
      * Add risk for application availability
      * Many tightly coupled and interconnected dependent processes increase the likelihood and impact of process failure
    - Microservice architecture
      * Decentralized evolutionary design
        + Does not require users to use specific technology
      * Components are updated separately
      * Smart endpoints, dumb pipes
      * Independent products, not projects
        + Against waterfall
        + Separate product
        + No assumptions are made about run time environment
        + Package all dependencies in single immutable object
      * Designed for failure
        + Services are made to be redundant
      * Disposability
        + Able to fail fast and start faster
        + Like a circuit breaker
        + Cloud is transitory
        + Resources are temporary
      * Development and production parity
        + Facilitates dev ops
        + Simplifies need for multiple environments for staging, qa, prod