* Course Overview
  + Installing docker(docker for windows vs docker on windows)
  + Command line
  + Host static web site
  + Build images
  + Databases in containers
  + Web application and databases with docker compose
* Installing Software Is Often Pai..
  + Docker simplifies software interaction
    - Installing, starting, stoping, building, etc
* Challenges Discovering Software
  + Find correct version
  + Then download it
  + Extract it and run installer
  + Run it
* Challenges with Software Installation
  + OS/Build compatibility
  + Format
    - Source, executable, executable + libraries, executable + libraries + runtime
  + Installers, package managers, manual install
* Challenges with Running Software
  + Documentation
  + Path of software
  + Staring, stoping
  + Service registration
  + Licenses
  + Installing and running dependencies
  + Security and sandboxing
  + Breaking changes updates
* Containers Are About Software
  + Docker and containers are about running software
  + Image: packaged up application
    - Kinda like a zip file
* Installing and Running MongoDB
  + Traditional
    - Search for mongodb
    - Download right version
    - Run installer
    - Set up mongodb environment
    - Make data directory
    - Then run mongod.exe
    - Run mono.exe
* Finding Software on Docker Hub
  + Search for mongodb docker
  + Click on ‘library/mongo’
  + Pull the image(docker pull)
    - Contains application and dependency
* Downloading Software with docker
  + Type ‘docker pull mongo:windowsservercore’
  + Instead of installing the software we create container
    - A stopped container is like installed software
  + Take image and run it in a container
  + Container: like installed software
    - Install step
  + Running container
    - Run step
* Running Software in a Container
  + ‘docker run’ does all the steps above
    - It will find and pull image
    - It will spin up container
    - Then run container
  + Type ‘docker run [image]:windowsservercore’
    - Windowsservercore specifies a window container
  + Command prompt will be attached to container
    - Locked up with the output of the running process
  + Type ‘docker exec -it 27 mongo’
    - To connect to mongo server
  + ‘docker create’ if you just want to create a container
* A Running Container is Just A
  + Find the software on docker hub
  + Download the software with docker pull, comes down as an image
    - Much like a zip file or msi installer
    - An image is an application packaging format
  + Instead of installing the software, we create a container
    - A stopped container is like installed software
  + Can use ‘docker create’ to just create a container
  + Then we run the container
    - Running the container is exactly like running an exe
  + We often use ‘docker run’ to orchestrate all these steps with on command
  + ‘docker exec’ can be thought of as running another copy of our installed software
    - After container is create and running, we can use docker exec to run multiple applications or multiple copies of the same app inside the container
  + Type ‘docker exec -it 27 powershell’ to run powershell inside of hte container
  + A container is like process namespace
  + Application launched via container is the same as if we had launched the application normally
  + Containers are special ways to work with them software/applications
  + A running container contains one or more running applications
* Stopping Apps in a Containers
  + Type ‘docker ps’ to look at running processes
  + Type ‘docker stop [container id]’ to stops the running software in the container
* Why Docker and Containers
  + Docker hub has all the software we can use
  + Repository hold images
  + Stats about software
  + Can be trusted, official images
  + Can scan through tags for vulnerabilities
  + Only have to worry about linux vs windows container
    - Don't have to worry about os
  + Docker file shows commands to create image
  + Docker is a way to manage software
* Installing Docker for windows
  + Download ‘Docker for Windows’
    - Mostly for desktop or workstation environment
    - Not for server environment
  + Needs windows 10
  + Hyper-v package must be enabled
  + Docker uses hyper-v to run containers
  + Need hardware virtualization
  + Go to task manager
    - Performance tab
    - Lower right ‘virtualization’ should be enabled
  + If virtualization not enabled then you will have to enable in the bios
    - Google instructions on how to do so
  + Update windows
  + Download the beta for windows containers
* What Docker for Windows Install
  + Open Hyper v Manager
    - Will be installed along with docker
  + It will set up hyper v vm that contains linux so we can run linux containers
  + ‘docker info’: information about docker server
  + ‘docker version’ information about docker client
  + Docker client will use windows os
  + Docker server will use linux os
    - Runs on VM
  + Docker for windows is a wrapper around client and server
    - Manages everything for our windows environment
* Running Linux Containers on Windows
  + ‘docker ps’: show running processes inside of docker
    - ps: process status
  + Containers are just running applications
  + ‘docker run hello-world’
    - Hello-world is the image we will download and take and create container and run it
    - Simple application that runs then exits
  + ‘docker run -p [host port]:[container port] nginx’
    - Since its a web server, will need to specify port number
  + Takes downloaded image and extracted locally
  + Images are layered
* Docker Inverts Learning
  + Quickly launch software with one command
    - No need to know how to set it up
    - Inverted learning
  + Inverted learning: use software without knowing how to set it up
* Stopping Containers Means St
  + Use “ctrl + p + q” to detach from container
    - Application still running, just go to cmd
  + When you create a container, each is given a random name unless you give it a name
  + Each container is also assigned a randomly generated id
  + Name or id can be used to interact with and control the container
  + Type ‘docker stop [name or id]’ to stop container
    - Only need to use enough of id to be unique
* Restarting Containers and doc
  + Type ‘docker start [name or id]’ to restart container
  + Type ‘docker ps -a’ to list all containers, even stopped ones
  + Starting and stopping containers does not destroy container
  + Type ‘docker run [image]’ will create a new container each time
* Removing Containers is Akin to..
  + Removing containers is like uninstalling software
  + Each container is like one installed software
    - 3 containers, 3 installed software
  + Type ‘docker images’ to list images we downloaded
  + Images are application packages, like zip files
  + Containers are the installed application
  + Removing containers does not remove the image
  + Have to stop containers before we can remove it
  + Type ‘docker rm [name of container or id]’ to remove the container
  + Type ‘docker rm [name of container or id] [name of container or id] ..’ to remove multiple containers
* Removing Images is Akin to De..
  + Type ‘docker’ or ‘docker --help’ to show list of commands and what they do
  + Type ‘docker rmi [image name/repository or id]’ to remove image
    - Removes the layers of images
* Running the Docker Docks in a ..
  + Type ‘docker help search’ for instructions on how to search the docker hub
  + Type ‘docker search docs’ to get back list of repositories
  + Can do same thing on docker hub website
  + Repository can have multiple tags that link to multiple images
  + Type ‘docker run [images]:[tags]’ to download specific version of image
* Docker run --it and --name
  + ‘docker run -p [host port]:[container port] -it --name [name] [image name]’
    - -it: flag opens an interactive terminal into the container
    - --name: flag sets the name of the container
  + Add the -it flag if you want to run containers in the foreground and be able to stop the container
    - Commands are sent to the running process inside of the container
  + Type ‘ctrl+p+q’ if you want to detach but not kill the process
* Switching to Windows Containers
  + Right click on docker icon
  + Choose ‘Switch to Windows containers’
    - Docker for windows will set up windows containers on your computer
  + Will need a new containers features
    - Need windows 10 anniversary, pro, enterprise
* Running IIS in a Windows Container
  + Windows containers are completely isolated from linux containers
  + IIS is windows equivalent of nginx on linux
  + ‘docker run -p 80:80 -d microsoft/iss:nanoserver’
    - -d: tag to detach from container
    - Container will run in the background
  + Type ‘ipconfig’ to get info about ip
  + Grab ipv4 address
  + Use another computer on the network and connect to the ipv4
* Containers Have Their Own Isolation
  + Can go directly to container
  + Process are isolated, so it looks like it is running on a completely isolated computer
  + Container has its own network adapter
  + Can ping network adapter via its associated ip address, can get into container that way
  + ‘docker inspect [container id or name]’ gives information about container
  + Can use http:// + ip address : + container port to access the container
  + ex) <http://127.17.0.1:4000>
  + Windows containers have trouble accessing with localhost but linux containers are abe to just fine
* Key Takeaways
  + Containers and docker is all about helping you run software
  + Find software = docker hub
  + Download software = pull an image = docker pull
  + Install software = create a container from image = docker create
  + Start software = run the container = docker start
  + Stop software = stop the container = docker stop
  + Uninstall software = remove container = docker rm
  + docker run does all with one command(not stop, or uninstall)
  + docker exec to run another instance of software or a different instance in the same container
  + Docker for windows is not meant for server environment
  + Run windows containers for window servers
  + Run linux containers for linux servers
* How Docker for Windows Sup..
  + Docker windows enable to you use windows and linux containers on your workstation
  + Can set up docker on windows server just to run windows containers
  + Docker for windows
    - Lanuchs a docker proxy service
    - Creates mobylinuxVM
    - Docker commands, docker CLI talks through docker proxy, which then forwards commands to mobylinuxVM which spins up the containers
    - For windows containers, docker daemon(dockerd.exe) is start up
    - Docker CLI talks through proxy, which is routed through dockerd engine which spins up windows container
* Docker on Windows Architecture
  + Windows server environment will only have dockerd engine
  + Use docker CLI to send message directly to dockerd engine to spin up containers
  + Only have docker exe and dockerd exe on windows server environment
  + Docker for windows is separate from docker engine that runs on windows
* Using the MSDN Windows Containers
  + Use <https://git.io/vPj49> for a tutorial on how to set up docker in the window environment
* Installing Docker on Windows(windows server environment only, not desktop)
  + Run windows updates
  + Run powershell as administrator
  + Type ‘Install-Module -Name DockerMsftProvider -Repository PSGallery -Force’
    - To install OneGet module
  + Then type ‘Install-Package -Name docker -ProviderName DockerMsftProvider’
    - To use OneGet to install the latest version of docker
* Running the Microsoft .NET Con..
  + ‘docker version’ to get info on docker client and server
  + Docker client and point to different docker server
  + Docker is a front end for running containers
  + ‘docker run microsoft/dotnet:nanoserver’
    - Runs then exits
    - Tries to open command prompt
    - Need to attach to it
  + ‘docker run --it microsoft/dotnet:nanoserver’
    - To run container and attach to command prompt inside of container
  + ‘docker ps -a --no-trunc’ shows all process with all information
* Running a Command Prompt I…
  + dotnet CLI available
  + Access to dotnet core
* User space and kernel space
  + Command prompt is a process running in container
  + Images are application packages, like zip file or MSI installer
    - Application code
  + Software layers
  + Hardware: cpu, memory, usb ports etc
  + On top of hardware is Kernel, Drivers, HAL(hardware abstraction layer)
    - Kernel takes control with full access of hardware
    - Abstract away hardware
    - Creates API for software to interact with hardware
  + All 3 below resides in user space
  + On top of kernel is OS that contains libraries
    - Win32 for windows
    - Posic API
  + Then “OS” apps
    - Applications that come with OS
    - CMD, powershell, IIS
  + The install our own software
    - Mongo, mssql, .net core, etc
  + Division between kernel and everything not in kernel
  + User space: everything not in kernel
    - Execute at lower privilege levels
  + Kernel space: everything apart of the kernel
    - Execute at higher privilege
  + In order for application in user space cannot directly access hardware
    - Need to go through kernel
    - Use series of systems calls to kernel
  + System calls: made from libraries that are apart of OS
    - Application call win32 API
    - Win32 API forwards calls to kernel via system calls
  + In past, only install top most layer of application
  + Relies on OS components already installed
* Images Contain User Space
  + To avoid problem of not having components on system you need for application you need to run
  + Containers and images can how we distribute software
  + Images containers everything inside of user space
    - Apps, OS Apps & Libs
* You can run any app in the I..
  + Pull image
  + Create container to run one of the application in the image
  + Multiple containers share the same kernel
  + More efficient
  + VMs use separate kernels inside of each VM running
    - Extra and expensive layer
* Layers and Shared and Immutable..
  + Docker avoids having multiple copies of entire images on computer
  + Images a layered to avoid this
    - Base OS layer is common amongst many different images
    - Only need one copy of base layer locally
    - Can be shared amongst all images and running containers
  + Layers content always need to be the same
  + Docker does this by pulling down them as read only
* Controlling the Process That R..
  + Type ‘docker run -it [image]:[version] [command to run]’
* Defining a Container Host
  + Host/Container Host: the machine that you run containers upon
* Containers Have Isolated File S…
  + Image delivers the file system that will available inside of the container
  + The files system for container is isolated from the file system for the rest of the machine
  + Go to ‘Computer Management’
  + Then to ‘Disk Management’
  + Disk will be created for container
  + Container has its own isolated file system
    - Does not share host file system
* Namespaces Provide Isolation
  + A lot of other things are isolated as well
    - Processes, file system, network
    - For windows, registry is isolated
    - For linux, namespace isolated
* Processes Are Isolated
  + Containers cannot see other process on the host computer
    - Good for security
  + Host computer can see processes inside of container
  + Windows containers will have multiple process
  + Linux containers will most likely have on process
* Network Stacks are Isolated
  + Type ‘ipconfig’ to look at network adaptors
    - Ethernet adapter section shows info about connectivity from computer to outside
  + Container ethernet is completely different then that of host machine
  + Host machine has virtual adapter that share ip address as the gateway for the container
    - Container and host machine can thus talk to each other
  + Containers has completely isolated network stack inside of container
* Environment Variables and Con..
  + Type ‘ls env:\’ list all environment variables
  + Environment variables are different as well
    - Host name
    - Etc
* The Registry Is Isolated
  + To see registry, best way is to enumerate it in powershell
  + Type ‘ls hkcu:\’
    - ‘ls hklm:\’
  + Container environment is isolated with its own registry
* Users and Groups Are Isolated
  + Type ‘Get-LocalUser’ to get local users
  + Users and groups are isolated, changes to once won’t affect the other
  + They have different SID’s(identifiers)
* There Are Two Types of Windows..
  + Special type of isolation for windows but not for linux
  + Windows server containers
    - Use isolation like linux
    - Kernel “lies” to processes
    - Process isolation
  + Hyper-V containers
    - Isolation by the hypervisor
    - VM isolation
  + Windows 10 can not use windows server container but can use Hyper-V container
* Running a Hyper-V Container
  + In power shell type ‘Install-WindowsFeature hyper-v’ to install hyper v
  + ‘docker run --rm …’
    - --rm: when container exit, it will deleted automatically
  + Use hyper v for strong isolation for containers
* Key Takeaways
  + Containers
    - Simplify software management
    - They also provide isolation
  + Spinning up VMs takes a long time
  + Containers are lightning fast
    - Linux containers are faster than window containers
* Exporting Images with docker
  + Type ‘docker save [images] -o [output file]’
    - Export image and archive it in a TAR archive
    - -o: output
    - [output file]: name of the output file
* Switching Docker for Windows
  + Needs software to pull apart tar files
* Is This Image Safe to Use? - Off…
  + Search for alpine on docker hub
  + Things could break out of container
    - Make sure repository is marked as ‘official’
  + Logging in to docker hub
    - Tags provides security scanning
  + Can also check popularity of image
  + Check for word ‘automated build’
    - It was built on docker hub
    - Not always required
* Running a Shell with the Alpine
  + Type ‘docker run -it alpine sh’
    - Might not need sh, its the default command
    - -it: opens shell inside container
  + While in shell
    - ‘ps’ command will show processes
    - ‘sh’ opens a nested shell
    - ‘exit’ exits shell
    - ‘clear’ to clear the screen
  + ‘tar’ command to interact with tar files
  + ‘ls’ to show list of files
* Using Volumes to Share the Ho..
  + To get files into container, will need to mount drive into container
    - Like plug usb stick into computer
    - Usb stick mounts as an additional drive
  + Volumes does this
  + Can mount additional drives via idea of volumes
    - Add another hard drive
  + Right click on ‘docker icon’
    - Click on ‘Shared Drives
    - Check C drive
      * Makes c drive available to linux vm
      * Then take c drive and mount it
  + Type ‘docker run --rm -v c:/Users:/data [image] ls /data’
    - To mount Users sub directory of c drive
    - Map it into container as /data
    - And run ls command to list contents of data folder
* Manipulating Host Files with To..
  + What volume mounts you have, the process in the containers now has access to it
  + Type ‘tar -tf [tar file]’ to list contents of tar file
  + Each nested layer is one of the individual layer of the image
  + Type ‘tar -xf [tar file] -C [extract folder]’ to extract contents of tar file to folder
    - Extract folder has to already exist
  + In alpine linux, type ‘apk add --no-cache [package]’ to install package
  + Can get files out of container
* Getting Files out of a Container
  + Type ‘docker run --rm -it -v [path to host directory]:[container directory] alpine tar -xf [tar file] -C[host output directory]’
    - Runs container, mount host directory to container directory, run the tar command to extract files, output files to specified host directory
  + Can run commands that you don't have on your computer
* Extracting the File System from..
  + Open tar file for layer, extract content, then run process
  + Exactly what docker does for you
* Running nmap in a Container
  + Run nmap
    - Type ‘docker run --rm weshigbee/nmap -v 192.168.0.0/24
    - Does a port scan across network
    - -v: verbose
* Converting Videos with ffmpeg..
  + Type ‘docker run --rm --volume ${pwd}:/output jrottenberg/ffmpeg -i <https://bit.ly/2fcrRK2> /output/Turkey.gif’
* Key Takeaways
  + Containers makes it easy to run software with a consistent command
  + Share host files with container processes
    - Be default there is no connection from container file system to host file system
    - Have to selectively enable connections using volumes
  + A volume is like attaching a extra drive
  + Can add references back to host file system from container
    - Map a folder from host into the container
* Building Images to Host Web sites
  + Host services inside of containers
  + Host a website
  + Building images
* Mapping Static Web Site Files i…
  + Web site static files, multiple ways
    - 1) Pulling existing server(nginx, iss) image, create container, use volume mount to map files for static website from host into container
    - 2) copy files from host system into the container file system
    - 3) bake files into an image
* Volume Mount Web Site Files
  + Download course folder solitaire
  + Copy the materials/solitaire folder to somewhere you can access
  + Cmd line into that folder
  + Type ‘docker run --rm -it -p [host port]:[container port] nginx’
  + If you need to find info on what port to run
    - Go to image on docker hub
    - There should be info on there
  + Type ‘docker run --rm -it -p [host port]:[container port] -v [path to file,lingux]:[container folder] [image]’
    - Find info on container folder to map to on docker hub
    - Might need to enclose path to file in “ “
* Modifying Files in a Running Container
  + Type ‘docker run -d -p 8080:80 --name nginx nginx’
    - -d: detach, container runs in background
    - --rm does not work with detached containers
    - --name so you can easily refer to it
  + Type ‘docker exec -it [name/id] [command]’‘
    - exec: run another process inside container
    - Use command ‘bash’ to open bash shell(linux)
  + Using bash shell, go into folder ‘/usr/share/nginx/html
  + Can make changes to file system in container
* Copying Files into a Running Container
  + Type ‘docker cp [source path]’ [container]:[destination path]’
    - cp: copies files/folder between a container and the local filesystem
    - Source path should start with .\
* Baking Files into an Image fro..
  + Setup container, copy in files into running container, take snapshot of container to create image
  + Type ‘docker exec [image name/id] [command]’ to run command within container
  + Type ‘docker commit [container name/id] [name:tag]’
    - commit: create a new image from a container’s changes
    - Has state at the moment you took the snapshot
  + Images are created are from containers
  + Containers are created from images
  + Created image is stored locally
    - Use ‘docker images’ to see
* Running a Container from a Cu..
  + Run the image
* Looking at Image Layers
  + Images contains
    - Operating System
    - Framework(nginx)
  + Type ‘docker history [image id]’, prints out history of how image was created
    - Bottom two lines form the base Operating System layer
    - Rest of lines form the nginx image
  + Type ‘docker history nginx --no-trunc’ to get full history
  + Can find history on docker hub as well
    - The docker file
  + Docker file is way to automate creation of image
  + Container contains
    - Operating System
    - Framework(nginx)
    - Container/Run
  + Each container has top layer where it can read and write files to
* Union File System
  + Union file system
    - Union of different layers
    - Aggregates them to a single view the container sees
    - View contains the read/write layer
  + Union Mount Point
  + Each layer can contribute files
  + Files in layer above trumps files in layer below
  + A layer only has files that were changed,created, or deleted when that layer was created
  + Only contains what was added for that layer
* How Running Containers Turn i…
  + To create image, take snapshot of container
* Docker history and docker diff
  + Type ‘docker history’ to get history of image
  + Type ‘docker diff [container]’ to see what changes container layers has in it
  + Image could contain files that were modified by running nginx instance
    - May not want to take a snapshot of this when in production
* From Commands to docker file
  + ‘docker run -d -p 8080:80 --name nginx nginx’
  + ‘docker cp .\app\.nginx:/usr/share/nginx/html’
  + ‘docker commit nginx solitaire:nginx’
  + Typing manual is error prone
  + Docker file is special script file to do all this
  + ‘FROM nginx’
    - Specify image we want to build on top of
  + ‘COPY app /usr/share/nginx/html’
    - Copy in app folder to container path
  + Save the script file next to the app folder, so path to app is correct
  + Type ‘docker build -f [image name]:[tag].’
    - To build image from docker files
    - The ‘.’ at the end, is the path to the build context(current directory)
* Using docker build to Create a
  + Create file with the from and copy command
  + Save it next to the app folder
  + Type ‘move .\Dockerfile.txt .\Dockerfile’ to remove the .txt extension
  + Type ‘docker build -t [image name]:[tag] .
    - -t: specify tag of the image
    - .: is the build context, current folder
  + In production is is best to linux images on linux docker host
    - Avoid line feed issues and permission issues
* Creating an Image to Host a Windows
  + Copy app folder to solitaire folder on windows server machine
  + Create docker file
    - FROM microsoft/iis:nanoserver
    - COPY app C:/inetpub/wwwroot
  + Running docker build on subsequent runs will use cache if docker file has not changed
* Pushing Images to Docker Hub
  + To share to docker hub, need to retag images in way that conforms with docker hub recongizes the slug of the image
  + Type ‘docker tag [image]:[tag] [new tag]/[name of repository]:[tag]
  + Type ‘docker push [new tag]’
    - Can push multiple images
    - Will need to login first
  + Type ‘docker login’ to login
  + Better to set up automated build on docker hub
    - Set git repository
    - Then link to git repository docker file via docker hub
    - Each time you check in changes, you can have new build of docker image created for you
  + Can make repository private
* Key Takeaways
  + Container comes from images and images come from containers
  + Container layer: read and write
    - Can make modifications to file system
  + Can commit layer to form a new image
  + Use Dockerfile: script to build an image
* Running MSSQL 2016 Express i…
  + Find microsoft/mssql-server-2016-express-windows on docker hub
  + On windows server instance
  + Pull 2014 and 2016 edition
  + Type ‘docker run -d -p 1433:1433 --env sa\_password=<[password] microsoft/mssq-lserver-2016-express-windows’
    - -d : detach, run in background
    - -p: set host and container port, 1433 is standard sql server port
    - --env: allows us to specify an environment variable
    - sa\_password: allows use to specify our own password
  + Type ‘docker logs [id]’ to what is going on in detached container
* Connecting SSMS to MSSQL in..
  + Use ‘ipconfig’ to get ip of container host
  + Hop over to machine with Microsoft SQL Server Management Studio
  + Use ip of container host to login with password you set up with the run command
* Running MySQL in a Container
  + Search for mysql on docker hub
  + In a linux container type ‘docker run --name some-mysql -e MYSQL\_ROOT\_PASSWORD=my-secret-pw -d mysql’
    - --name: name container
    - -e: shortcut for --env, specify environment variable
    - MYSQL\_ROOT\_PASSWORD: environment variable to set password for mysql server instance
    - -d: detach
  + Type ‘docker exec -it [container id] mysql --user=root --password=my-secret-pw’
    - To start up mysql client inside of container
  + Inside of mysql client
    - Type ‘show databases;’ to show database
    - Type ‘use [database];’ to use once of the databases
    - Type ‘show tables’ to show tables in database
    - Type ‘select \* from users’
* Docker Managed Volumes
  + Type ‘docker volume ls’
    - See volumes that have been with docker
  + Database have special consideration for storing their data
  + Pretty common for volumes to be created to house data that a database write to disc
  + Volumes are also used to map files from host into running container
  + Volumes can also be used to performantly store data for a database
  + Volumes created for us, like for mysql server, persists whether or not container is running
    - Data has to be there so we can start and stop database and not lose the data
  + Read write layer for container also persists
  + In docker file for mysql
    - There is a volume instruction
    - Marks a path inside image to be created as an external volume outside of the file system for the container
  + Volume is created on the host and mounted in the container
  + /var/lib/mysql is the default data location of mysql image
  + Removing the container, the volume remains
  + The lifespan of the volume is independent of the lifespan of the container
* Using Managed Volumes to Persist Data
  + Type ‘docker run --name some-mysql -e MYSQL\_ROOT\_PASSWORD=my-secret-pw -d -v [name of volume]:[path] mysql’
    - -v: set the name of volume
    - Path: is /var/lib/mysql
  + A volume is created per run of mysql
  + In mysql container
    - Type ‘CREATE DATABASE pets;’ to create database
  + Cannot remove containers while they are running
    - Have to stop them first
    - Or do a force stop
  + Type ‘docker rm -f [container id/names]’
  + After volume is created
  + Typing ‘docker run --name some-mysql -e MYSQL\_ROOT\_PASSWORD=my-secret-pw -d -v [name of volume]:[path] mysql’
    - It will reuse the existing volume if already present else it will create a new one
* Cleanup - Stopping All Running..
  + Power shell only
  + Type ‘docker ps -q’
    - -q: quiet output mode, containers id is the only that comes back
  + Pipe that command into ‘docker stop’ to stop all running containers
  + Type ‘docker stop $(docker ps -q)’ to stop all running containers
    - $(): is how you pipe in another docker command to another
    - Only for powershell etc
* Clean up - Removing All Containers
  + Power shell only
  + Type ‘docker rm -f $(docker ps -aq)’ to remove all containers
    - -f: stop running containers
    - -a: lists all containers
    - -q: quiet mode, only run container ids
  + Type ‘docker container prune’ to remove only stopped containers
* Cleanup -Removing Volumes
  + Power shell only
  + Type ‘docker volume ls -q’ to return only id of volumes
  + Type ‘docker volume rm $(docker volume ls -q)’ to remove all volumes
  + Type ‘docker volume prune’ delete unused volumes
  + If you have running container with volume
  + Type ‘docker rm -fv [container id]’
    - -f: force running container to stop
    - v: remove associated volume, only unnamed volumes
* Cleanup-Removing Dangling …
  + Dangling volume: volume that is not associated with a container
    - Container it was created with is destroyed
  + Type ‘docker inspect [container id]’ to get information about container
    - Look at “mounts” section to see the volume name
  + “Source” is where volume is created at on container host
    - Managed volumes are created on host file system
  + Type ‘docker volume ls -f dangling=true’
    - To get back all volumes that are dangling
  + Type ‘docker volume rm $(docker volume ls -qf dangling=true)’
    - To remove all dangling volumes
* Cleanup - Removing All Images
  + Type ‘docker rmi [repository]’ to remove images
    - Images can’t be in use, need image to have container running from it
  + Type ‘docker images -q’ to return just the image ids
  + Type ‘docker rmi $(docker images -q)’ to remove all images
  + Type ‘docker image prune’ to delete unused images
* Cleanup - Removing Dangling Images
  + Dangling Image: image that is untagged
  + Type ‘docker images -f dangling=true’ to return dangling images
  + Typically end up with dangling images when you rebuild your own images
    - The old image becomes dangling because new image has the tag
  + Type ‘docker build -t [new image name]:[tag] .’ to build image from docker file
  + Type ‘docker tag solitaire:nginx-df weshigbee/solitaire:nginx’ to tag image
  + As you build images over and over again, you will amass dangling images
  + Type ‘docker rmi $(docker images -qf dangling=true)’ to delete all dangling images
* Key Takeaways
  + Two volume types
    - From host file system(bind-mount)
    - Managed volume: managed by ‘docker volume…’
  + Volumes
    - Bypass union FS
  + All volumes bypass file system
    - Volumes lives beyond container
    - Independent lifespan
    - Not part of images
  + Put read heavy data in images
  + Put write heavy data in volumes(DB)
* Module Introduction
* Why docker-compose Exists
  + Can combine frontend and backend into one container, but it more work
  + Better to separate one container for front end and one for backend
  + Type ‘docker run --name db -d -p 3306:3306 -e MYSQL\_ROOT\_PASSWORD=mysecret-pw -v db:/var/lib/mysql mysql’
    - --name: name the container
    - -d: detach
    - -p: host and container port
    - -e: environment variables
    - MYSQL\_ROOT\_PASSWORD=mysecret-pw: set up password for database
    - -v [name]:[path]: create volume independent of container
  + ‘docker inspect db #’ extract ip address of db to pass to web server
  + ‘docker run --name web -d -p 8080:80 -e MY\_DB\_PORT=3306 -e MY\_DB\_HOST=? -v [path] nginx’
    - MY\_DB\_PORT: where database port is
    - MY\_DB\_HOST: ip address of the database
    - -v [path]: map in website files
  + Dockerfile: automating process of creating images
  + Docker compose file: automate process of creating containers
    - Yaml format of commands to run containers
    - Ex
    - version: ‘2’
    - services:
    - db:
    - image: mysql
    - ports:
    - -3306:3306
    - environment:
    - -MYSQL\_ROOT\_PASSWORD=my-secret-pw
    - volumes:
    - -db:/var/lib/mysql
  + docker-compose.yml
  + Feed yaml file to ‘docker compose..’
  + version: version of file format of docker comp
  + services: containers we want to run
  + db: name of container
  + White space and indentation is important in yaml file
* A TeamCity docker-compose.yaml
  + Go to class github page
    - https://gist.github.com/g0t4/0d97a9595c87736a8a72a2bd21afc0d9
  + Create team city folder
  + Inside folder type ‘curl <https://gist.githubusercontent.com/g0t4/fc6a7b62be5a14c5af9df42af2aa2b2b/raw/b9d2475d0cf96d576a355917cb8d6c7f42079539/docker-compose.yml> -o docker-compose.yml’
  + Type ‘cat .\docker-compose.yml’ to see contents of files
  + Docker runs containers highly isolated
  + When you spin up containers via docker compose file, docker creates isolated network for you
    - Will place containers on network
  + Embedded DNS server that will resolve the rest of the containers via their container name
  + Ex: server container name is teamcity
  + Then can access server with [http://teamcity](http://teamcity:8111):[container port]
    - Containers talk to each other over container ip addresses and container port
    - The host port is so you can get to server from container host
  + It will resolve to whatever ip address server container is created with
  + Teamcity: is server with web UI
  + Teamcity-agent: does builds
  + Postgres: is the database
* Spinning Up Complex Apps with…
  + ‘docker-compose’ command
    - Way of putting arguments for docker run into a files
  + Type ‘docker-compose up ‘
    - Will use ‘docker-compose.yml’ by default
    - up: run command for docker-compose
  + Go to the teamcity page(http://localhost:8111’)
    - Set up the database connection setup via the UI
    - Database type is ‘PostgreSQL’
    - Database host is ‘postgres’
    - Database name is ‘teamcity’
    - User name is ‘postgres’
    - Password is blank
  + Teamcity creates its own database in the container
  + Create administrator account
* What You Created with docker…
  + Container host
    - Docker engine on linux VM
  + Docker compose file
    - Docker engine spins up containers
  + Virtual NIC
    - Virtual network adapter
  + Docker engine creates a private network
    - Hooked up all containers to network
    - They can talk to each other
* docker-compose Creates Isolat..
  + ‘docker network ls’ list out network on computer
  + Docker can manage
    - process/containers
    - Volumes
    - Networks
  + ‘docker network inspect [network name]’ to get info about network
  + Name of containers are [project prefix]\_[name of service]\_[increment number]
  + Project name is the folder you are inside of that contains docker-compose.yml file
  + If you don’t specify network, default bridge network is used
  + ‘docker network…’ to manage networks
* Service Discovery via an Embe..
  + Type ‘docker-compose exec [name of service] bash’ shortcut to shell into one of the containers
  + Can ping other containers because of Embedded DNS
* Connecting Another Container ..
  + User define network all have embedded DNS
    - Not in bridge network
  + ‘docker run --name alpine --rm --net [network name] -it alpine sh’
    - --net: to setup network
* Restarting Containers with doc…
  + ‘docker-compose ps’ get status about processes just inside of containers associated with yml file
  + ‘docker-compose start [service name]’ to start up a single service from yml file
  + ‘docker-compose logs [service name’ to get logs
* Using psql in the postgress Containers
  + Bash shell into postgress containers
  + Type ‘docker-compose exec postgress psql -U postgres’
    - psql: psql client tool to browse database
    - -U: username
  + In psql
    - \l: to list out databases
    - \c [database] list out tables of database
* Tearing Down Infrastructure Cr…
  + Type ‘docker-compose stop’ will stop all containers in yml file
  + Type ‘docker-compose rm -v’ to remove containers and volumes
    - The networks remains
  + Type ‘docker-compose down’ will tear everything down from yml files
  + Type ‘docker-compose up’ will build everything
* Running the ASP.NET Core MVC..
  + Yml file can also configure networks
  + services: for containers
  + networks: for networks
* Key Takeaways
  + Dockerfile:images
  + docker-compose: containers(and more)
  + User defined networks
* What to Learn Next
  + Docs for Docker
    - Docker run reference doc
    - Command line reference doc
    - Dockerfile reference doc
    - Docker-compose doc
  + Docker Swarm
    - Swarm mode: easily spin up a cluster of docker resources