* Course Overview
  + Spinning up multiple containers on a single machine is easy
  + In production environment it may need to span multiple nodes so there is enough capacity
* The Magic of Docker Should W..
  + Some point containers will require more resources than a single machine can handle
  + Spin up containers that can talk to each other across multiple nodes(machines)
  + Swarm mode built into docker engine 1.12 and above
* What Are Some of the Concern..
  + Containers are just process that are running software
    - New way of running software
  + Will run out of resources from a single machine eventually
  + In production environment you will want to have containers over multiple machines
    - Fault tolerance: can lose one machine
  + Dont want to manage each machine individually
  + Docker swarm manage the machines for you
  + Containers will run on top of nodes(machines)
  + Docker swarm will also manage where to place containers based on resource requirements
* What You Should Already Know
  + Getting started with docker on windows
  + Containers and Images: The Big Picture
  + Will need multiple machines(VMs)
* What if a Single Container Isn’t …
  + In production environment you will need to handle alot of requests
  + Can run more containers or optimize api to handle more requests
* Scaling Capacity by Scaling Containers
  + Can add more containers on the same machine if it has enough resources
    - Requests will need to be routed to the containers
  + How to route requests to each of these instances
  + Hot to load balance incoming traffic to spread it out to multiple instance of the application
* What About Balancing Load A..
* What Happens When a Container..
  + What if application dies inside of container
  + Concerns:
  + Scaling Capacity
    - Multiple containers
    - Load balancing
  + Container failure
    - Restart, not good for production
  + Restart, type ‘docker run --rm -d -p [host port]:[container port] --restart=[option] [image]
    - Options are
    - ‘always’
    - ‘no’
    - ‘on-faliure’
    - ‘unless-stopped’
* What Happens When A Node Fails
  + If containers fails when can just restart
  + If node fails, all the containers fails with it
    - Reboot machine
  + In production
    - We will need to bring up capacity somewhere else
    - And then load balance
  + Node Failure
    - Redistribute containers
    - Replace node
    - Placement
    - Node Maintenance
* What About Internal Communication
  + Set environment variable connect to other containers
* User Defined Networks to Con…
  + Bound ports to node, set environment variable to connect to other containers
    - Tricky to scale
    - Shares single address space on the node
    - Loss benefit of network namespace even though we are using containers
  + Custom network
    - Create a virtual network and have containers talk to each other through network
  + Type ‘docker network create -d=bridge [network name]’
    - -d: type of network
    - bridge: network you can use locally to connect containers on a single node
  + Type ‘docker run --rm -d --name customer-api --network [network name] swarmgs/customer’
    - --network: specify which network to use
  + Type ‘docker run --rm -d --name balance-api --network company -p 4000:3000 -e MYWEB\_CUSTOMER\_API=customer-api:3000 swarmgs/balance’
    - Can list container as the MYWEB\_CUSTOMER\_API because the custom networks have service discovery enabled
    - Service discovery is setup based on the name of the containers
    - Resolvable via DNS with container names
    - Will get back ip address specific to container
  + User defined networks comes with embedded DNS server for service discovery
  + Need to attach to the same network
* docker-compose simplifies Co..
  + create network and containers has a lot of commands and can get confusing
  + can do all this with docker-compose
  + need a yml file
    - network is automatically created with docker-compose
  + type ‘docker-compose -f company.yml up -d’
    - -f: specify file
    - up: to run
    - -d: run detach in the background
  + docker-compose creates a network specific to a compose file so that containers are automatically attached to their own private network
    - enables service discovery to talk to each other
  + type ‘docker-compose -f [.yml file] stop [service to stop]’ to stop service
  + type ‘docker-compose -f [.yml file] start [service to start]’ to start service
  + docker-compose can be used with a swarm
* What About Scaling Internal A..
  + Face same issues with application dependencies as with single application
    - Scaling each application
    - Running instances across different nodes
    - Fault tolerance
    - High availability
  + Load balancer for incoming requests
  + Load balancer for internal requests(between containers)
  + Then can scale each component separately
  + Docker swarm makes this simple
    - Build out and schedule work
* Preparing a Single Node to Foll..
  + Launching containers with docker swarm is different then with docker engine via docker run or docker compose
  + Use service to start up a container
  + Service can be used to spawn containers across multiple nodes
  + Need docker v17.03.0-ce-rc1and up
  + Go to about docker to find your version
  + Or type ‘docker version’
  + Enable Experimental field
* Enabling Experimental Features
  + Set the configuration flag for experimental to true
  + Click on docker icon
    - Preference
    - Daemon tab
    - Check experimental to true
* Enabling Swarm Mode By Initia..
  + Type ‘docker info’ to see information
  + See ‘Swarm:…’
  + Swarm mode of docker daemon is not enabled by default
  + Management command ‘swarm’
    - Each management commands has a series of sub commands
  + Ex) ‘docker images ls’ does the same thing as ‘docker images’
  + Ex) ‘docker container ls’ does the same thing as ‘docker ps’
  + Type ‘docker swarm init’ to initialize a brand new swarm
    - If you have multiple Ips then use
    - ‘docker swarm init --advertise-addr X.X.X.X’
    - Any ip is fine
* Listing and Inspecting Nodes
  + ‘docker node …’ sub management command to manage nodes that are apart of a swarm
  + A swarm is just a cluster
  + Type ‘docker node ls’ to list nodes in a swarm
    - \*: indicates the node that you are currently one
  + ‘docker node inspect self’ to get information about current node
  + ‘docker inspect [host name/id]’ to get information about current node
* Creating an NGINX Service
  + After creating swarm
  + If you use ‘docker run’ it will create a container on the node but it won’t be apart of the swarm
  + Type ‘docker service create --name [name] -p [host port]:[container port] [image]’
    - Gets a container running
    - Not exact same a docker run but it is similar
    - Can use --publish instead of -p
* A Service is a Definition of an..
  + Service: how we create containers with docker swarm mode
    - A service is a definition describing what you would like to run
* Services Lead to Tasks
  + A service is a definition, it describes a desired for your application
  + Service is not directly mapped into running containers
  + A notion in between is a Task
  + A service leads one or more task
  + Task: slots that are eventually used to run a single container
    - Replicas = tasks
  + Type ‘docker service ls’ to get list of services that are part of the swarm
  + Mode
    - Replicated: create multiple instances that are spread out across the cluster
    - Global: run only one instance of application on each node in the cluster
  + Type ‘docker service inspect [name or id]’ to get in depth information about service
  + A service is definition, a recipe for tasks
  + Tasks ultimately turns into a running container
    - Slot though, not actually running container itself
    - Another declarative concept that describes the container
  + Type ‘docker service ps [name or id]’ to list tasks
    - Similar to docker ps
* Removing a Service
  + Type ‘docker service rm [name or id]’
  + In production you would update rather than rm
* Updating a Service to Scale th..
  + Type ‘docker service update --replicas=2 [name or id]‘ to change definition of service
    - --replicas: updates the replicas(task) definition, will create 2 tasks in this case
    - Each task will run on separate containers
    - There are more update sub commands
  + Type ‘docker service scale [name or id]=2’ does the same as above
  + Service is declarative, describes what we want, not actually instance of container
  + Swarm keeps desired state in mind, service definition
  + Constantly monitoring to make sure we are fulfil constraints we describe in the service
* Swarm managers Ensure the D..
  + The swarm manager accepts your service definition as the desired state of your application
  + Swarm manager always make sure desired state is enforced
    - It reconciles and disparities
  + Once a task is dead, its gone
  + If a containers stops, that task is gone and a new once is used to spin up another container
    - There is some time between bring cluster back to desired state
* The Scheduling Process
  + Docker docs has more information about scheduling
  + Docker services create -> swarm manager api -> orchestrator -> allocator -> dispatcher -> scheduler -> worker(creates containers & reports back to manager node) -> executor
* Creating a Second Service for..
  + Type ‘docker service create --name customer-api --publish 3000:3000 swarmgs/customer’
  + Each application on swarm has separate service definitions for each of them
  + Type ‘docker service scale [name]=1’ to scale replicates according to number
* The Swarm Mode Routing Mesh
  + A port is published on every node in the swarm
  + Mapped to the swarm load balancer
    - When traffic comes in, it is routed to one of the containers with the actually application
  + Part of swarm node routing mesh
  + If containers doesn’t container it will route to node with the container running
  + Automatic load balancer put in front of all instance of your applications
  + Can scale and add more containers without doing much
* Testing Throughput on a Scale