* 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
  + Type ‘ab -n [# requests] -c [# parallel] [address]’ to test performance
  + Can scale up to have better through put
    - Type ‘docker service scale [service name]=[number of replicas]’
* Moving to Multiple Nodes
  + Service can scale across nodes
* Destroying the Single Node Sw..
  + Can reset environment via UI
  + Type ‘docker swarm leave -f’ to remove node from swarm
    - If node is the last manager, we will erase current state of swarm
  + Node needs to be manager to list services
* Creating and Managing VMs wi..
  + Can use ‘docker-machine’ to spin up VM with docker engine
  + Can use ‘vagrant’ to also spin up VM
  + Docker-machine comes with docker on windows
  + Type ‘docker-machine create -d virtualbox [name]’
    - -d: driver, specifies what environment to create the VM in
  + Type ‘docker-machine env [name of VM]’ to set up environment
  + Then type ‘eval $(docker-machine env m1)’
    - Set up environment to point at docker engine
    - Could be different on windows
  + Type ‘docker-machine ssh [vm name]’ to get into machine
  + Type ‘docker-machine rm [vm name]’ to wipe out vm
* Accessing my Lab Setup Vagrant
  + Download Vagrantfile and node.sh file from <https://github.com/g0t4/docker-swarm-mode-getting-started>
    - Its a file to setup VMs
  + Managers nodes: manage state of the cluster(swarm)
  + Workers nodes: carry out task
  + Vargrantfile sets up VMs, the node.sh file sets up docker
* Launching 3 VMs with vagrant
  + Type ‘vagrant up m1 w1 w2’ to create vms
    - up: command to create VMs from vagrant file
    - m1: manager node
    - w1: worker node 1
    - w2: worker node 2
  + Type ‘vagrant status’ to check things out
* Accessing the Docker Engine I..
  + Type ‘vagrant ssh m1’ to ssh into m1 vm
  + On host type ‘export DOCKER\_HOST=192.168.99.201’ to point to vagrant vm
    - The ip address is from the set up file
  + In power shell type ‘$env:DOCKER\_HOST=”192.168.99.201” ‘
  + Type ‘docker info’ to see where you are pointed at
    - Name part
  + Vagrant machines you have to set up TLS(encrypted communications)
  + docker-machine set up encryptions and certificate automatically
* docker swarm init --advertise-a…
  + On cmd type ‘docker info | grep Name’ to get back name of machine
  + On powershell type ‘docker info | sls Name’ to get back name of machine
  + On manager node type ‘docker swarm init --advertise-addr 192.168.99.201’
    - Do it on manager node because which ever machine you type ‘docker swarm init…’ it will become a manager node
    - If VM has multiple interfaces IPs you will need --advertise-a
    - Docker swarm needs to know which IP address we want to bind to
    - The IP address for other nodes to communicate with
  + It will return a token
    - Use token on other nodes to join the cluster
  + A few things happen after ‘swarm init’
    - Current node is switch over to swarm mode
    - Current node is set to be a leader(manager)
    - Current node is also a worker node
    - Node initializes a distributed data store to keep track state of the cluster
    - Only on manager nodes
    - Root certificate authority is generated for swarm
    - Which is used to generate token to allow other nodes to join
* Joining Worker Nodes to the Swarm
  + In another node type ‘docker swarm join --token [token] [manager node ip]’ to join node to swarm as a worker
  + Token is made up of --token SMTKN-version-digest of root CA-randomly generated secret
    - SMTKN: swarm token
    - Digest: used to validate the root CA of the manager
    - Secret: used make sure given node is approved to join cluster
  + If token is leaked, use ‘docker swarm join-token --rotate' to change tokens
    - Existing nodes will not be affected
  + Type ‘docker swarm join-token manager’ to get instruction and the token to add nodes as manager
    - Wont work on a worker node
    - Have to be on a manager node
  + It will return a different token, the secret will be what is different
  + Type ‘docker swarm join-token worker’ to get token to join worker nodes
  + Type ‘docker node ls’ to list out node in the swarm
    - Only works on a manager node
* Creating a Service to Visualize…
  + Download docker-swarm-visualizer
    - ‘docker service create --name viz --publish 8090:8080 --mount=type=bind,src=/var/run/docker.sock,dst=/var/run/docker.sock –constraint=node.role==manger manomarks/visualizer’
    - --mount is similar to --volume
    - Giving service access to docker.sock
    - Constraint: allows use to constrain where application is run
    - 8090 is the cluster ip
    - 8080 is the container ip
  + In powershell use backtick ` to wrap a line
  + In cmd use \ to wrap a line
  + The port will be available to all nodes in the cluster
  + Service needs access to docker engine on manager node
    - Need to map in that in
    - With docker run use the --volume flag
* What Happens When a Node Is..
  + Type ‘vagrant halt [machine name]’ to bring down a machine
  + Type ‘vagrant up [machine name]’ to bring up a machine
  + Type ‘docker node [machine name]’ to remove a node
    - Have to go to node and type ‘docker swarm leave’ first
  + Type ‘docker node rm -f [machine name]’ to force remove a machine
    - May have to do a ‘docker swarm leave’ first if it was part of a swarm before
* Creating and Scaling a Second Service
  + ‘type docker service scale customer-api=2’ to scale up task
* The Spread Strategy and Testing
  + By default docker swarm spreads out the work across all the nodes
  + There are other modes of publishing ports on the cluster where those ports don’t have to be publish on every node
  + Easiest is to publish on every node and uses load balance to redirect to a given container
* Inspecting Nodes and Clustering
  + Type ‘docker node inspect [node name]’ to get info on node
  + To exec into container, you have to be on the node the container is on
  + Ingress network is used for network mess
  + Services with published ports are connected to the ingress network
  + If you do ‘docker network inspect ingress’
    - You will only see containers on the node
  + Do ‘docker network inspect --verbose ingress’
    - Shows all tasks and Ips
* Listing Tasks per Node
  + Type ‘docker node ps [node]’ to list of running tasks on node
    - Must be on manager node
  + Type ‘docker node ps [node1] [node2] ..’ to list running task on the nodes
* Promoting a Worker to a Manager
  + Type ‘docker node promote [node]’ to promote one or more nodes to manager in swarm
  + Type ‘docker node demote [node]’ to demote one ore more nodes from manager in the swarm
  + Have to be on manager node
  + Manager status
    - Leader
    - Reachable: manager node that is not the leader
* Draining a Node to Perform Ma..
  + Type ‘docker node update … ‘ to update a node
  + Type ‘docker node update --availability=drain [node]’
    - drain: means to get work off this node
    - running this command will shutdown the tasks on this node and start up the task on another node
  + type ‘docker node update --availability=active [nope]’ to bring node back up again
    - container wont move work around unless it has to
  + --availability=pause: will allow current task to continue to run but wont allow new task to be assigned to the node
  + Can scale up then down again to move containers to different nods
* One Container per Node with
  + ‘docker service create --mode==global…’
    - --mode is set to replicated by default
    - global: once instance of task(thus one container) onto each node in the cluster
  + each node will contain a task for the service
  + scale can only be used with replicated mode
* Swarm Mode is Incredibly Easy
  + Swarm mode: embedded swarm kit
  + Secure by default with tokens
    - Possible because certificate are create automatically
  + Certificate rotation is also automatically
  + Cluster management integrated with Docker Engine
  + Multi-host networking
  + Scaling
  + Declarative service model
    - Swarm mode enforces desired state of application
  + Rolling updates
  + Load balancing
  + Service discovery
* Routing External Traffic to Clus..
  + How traffic goes into swarm
* Published Ports Provide Extern…
  + Each node has its own IP addresses that is apart of a single network
  + The network has an underlay network that connects to the outside world
  + When you publish a port for a container, each node in the cluster will have the port to the application
* The Ingress Overlay Network
  + Traffic might come into a node that doesn’t have the desired container
  + It has to route to another node to get into the container
  + Another network the swarm load balance is connected to
  + Ingress overlay network
    - Span multiple host
    - Created by default when we enable docker swarm node
    - Routes published ports back to containers that provide the given service
  + Ingress overlay(virtual network) runs on top of the underlay network
  + Underlay network route request from outside to the swarm
  + Load balancer determine which node to send the request to
  + Ingress overlay routes published ports to containers that provide the service
  + Container creates highly compute environments
* Options for Routing External T..
  + Can set up DNS server to route outside traffic to desired node
    - Unreliable
  + External load balance outside of swarm
    - One ip address to load balancer
    - Load balancer will route off to different available nodes
    - Can have health check
* Host Publishing Mode Instead of Swarm
  + Can control which container the request is routed to
  + Global swarm instead of replicated swarm
  + Map a port on node directly to container on node
* Ingress Publish Mode Routes to…
  + Create service with --mode=global
    - Type ‘docker service create --mode=global --name [service name] --publish 8080:8080 [image]
  + Going to a specific node but routing to random host since all nodes contains the application due to the ingress network
* Removing a Published Port on..
  + Type ‘docker service update --publish-rm [target port] [service name]’
    - To remove a published port
    - update: to update definition of service
    - --publish-rm: to remove publish port
* Adding a Host Mode Published..
  + Type ‘docker service update --publish-add mode=host,published=[host port],target=[container port] [service name]’
    - --publish-add: to add port
    - mode=host: use host mode
  + One to one mapping with published port
  + Each node that we access the request will be fulfilled by the container running on that node
  + Be careful not to run multiple containers on a single node for a given service
    - There will be a port collision if you do
  + For global service with one container on each node
  + Ingress, host mode, plus future modes
* Publishing a Random Port
  + When you publish a port you don’t have to specify the published port
  + It is assigned automatically when you just specify the target
  + Type ‘docker service create --name [name] -p target=[target port] [image]’
  + Type ‘docker service inspect [service name] --pretty’ to see info about the service
  + Random port starts at 3000
* Reconciling a Desired State
  + Maintain a desired state is a big feature of docker swarm
  + Instead of explicitly specifying what we want to run
  + We describe it in a declarative style
* Quiz-What Happens When W..
  + Type ‘docker swarm join-token worker’ to get command to join node as a worker
  + If you created service in mode global
  + Every time you join a new node, that node will create a task to spin up a container for the service
* Creating a Pending Service an..
  + Can use constraints to ensure app runs on proper nodes
  + Go to docker/swarmkit
    - Go to read me
    - Look at constraints table
  + Type ‘docker service create --name [name] -p 9000:80 --constraint node.hostname==[desired host name] [image]’
    - --constraint: specifies a constraint for the service
    - node.hostname: specifies that this should run on a specific host
    - If constraint is not met, service will be created but no replicas will be available
    - It will not run, always be in pending state
* Joining a New Node to Fulfill a ..
  + When a node that fulfills the constraint joins the swarm
    - The service will then create a container on the node
    - Swarm is constantly monitoring cluster
    - Task will just sit in pending state until it is possible to assign it to a given node
* What Happens to a Service Wh..
  + Tasks are one way concept
  + Once container for task fails, task is shutdown and a new task created based on definition of service
  + Type ‘docker inspect [task id]’ to get more information about task
* Cleaning up Nodes That Have …
  + Have to forcibly remove node if its gone
  + Otherwise it will sit in swarm as a down node
  + Type ‘docker node rm [node name]’
  + Type ‘docker node rm -f [node name]’ to remove node that is not down
    - Safest way is to remove node from swarm first
    - Type ‘docker swarm leave’
* Remove Vestigial Services
  + If a service can't run, it will just sit and wait to be run
* If Your App Fails Then the Corr…
  + If application terminates the task is terminated and a new task is created with state of pending
    - May switch nodes
  + Type ‘docker service ps [service name]’ to get list of tasks for service
* Scale a Service to Zero to Stop..
  + If you don’t want to get rid of service
  + You can scale it down to 0
    - If node joins cluster that fulfills constraints, then service won’t accidentally run
* Desired State Reconciliation Af..
  + Can segment responsibilities
* How Do We Update an Application
* Updates Seem to Happen All a..
  + Scaling is no destructive, doesn’t destroy existing ones
  + Changing versions, you will need to destroy existing ones and create new ones
  + Tasks are destroy all at once, updates are rolled out incrementally
* Updates Are Incremental
  + One container and task is shut down and new container and task is created
  + Then subsequent task is updated
  + Spread strategy is not always sequential, rolling updates
  + Only one task is updated at a time
* Running a Docker Command o…
  + Can make script to remove images from all the nodes
    - .sh file
    - set -x
    - for host in [machine ips range]; do
    - export DOCKER\_HOST=192.168.99.$host
    - eval $command
    - done
    - The ip range will be take the place of $host
  + Type ‘./[file name].sh [command]’
  + Iterate through vm ip address and run command you pass into it
  + Manager resolves tag from digest
    - Digest: a pointer to a specific image
    - A tag can change but a digest can’t
  + Worker only sees the digest, not the tag
  + If content trust is enabled, the client actually resolves the image’s tag
  + Called pinning, important for production environment
* Specifying an Image Tag When..
  + Type ‘docker service create --name pay -p 3000:3000 swarmgs/payroll:1’
    - :1 uses version 1 of the image
* Adding Delay Between Task U…
  + Type ‘docker service update --image [image name]:[tag] [service to apply to]’
    - To update version of image
    - --image: flag to change image
  + Can adjust default so the roll out is slower
  + Type ‘docker service update --image [image name]:[tag] --update-delay=[time] [service]’
    - --update-delay: amount of time between update each of task in service
      * Time is s, m: ex) 20s
    - --update-parellelism: maximum number of task to update simultaneously
  + Slots stay around, so you get the history of the task in that slot
  + Update delay puts a weight in between updating batches of tasks
  + Parallelism flag controls how big the batches sizes, how many task we do at once
* Updating Multiple Tasks Concu…
  + Type ‘docker service update --image [image name]:[tag] --update-delay=[time] --update-parallelism=[number of concurrent task update] [service name]’
  + Parallelism allow you to control number of instances of application you want to update at a given time
* Cleaning up Task History When..
  + Can scale task down 0 then up again to clear history
* Quiz Recap on Update Delay a..
* Using watch to Visualize Updat…
  + When you update a task
    - Original task is remove then its container
    - A new task is started then its container is spun up
    - New task is created at same time old task is removed
    - New task is run when old task and container is shutdown
  + There are tool you can use to watch status from command line to see order of shutting down task and starting a new one
  + Watch command
    - Allows us to run a command over and over again
    - Type ‘watch [command]’
    - Ex) type ‘watch docker service ls’
    - -d flag: flashes differences
    - -n time: run frequency
* Slowing Down the Update Proc…
  + Can write out a script to split screens and run watch for you
  + Don’t want to start up new container while old one is being shutdown
    - Because there could be multiple instances of application running
* RollOut Mode and Other Updat..
  + UpdateConfig file
* Inspecting the UpdateConfig P…
  + UpdateConfig represents an update policy
  + Policy holds all configurations for how updates are peformed
  + Type ‘docker service inspect [service]’ to see UpdateConfig option
    - Pay attention to previous spec and current spec
  + Its parts of service definition
* Watching the UpdateConfig Po..
  + Make changes to the service definition to update the UpdateConfig option
  + Type ‘docker service update --update-delay=5s [service name]’
  + Updating the update policy doesn’t require to update task
  + Version index: indexes version of our application
    - Service definitions are indexed
    - Index is global counter across all services in our cluster
* Inspecting Task Errors
  + Type ‘docker inspect [task id]’ to get details of task
    - Look at status section
  + Version index also applies to task
* Rolling Back to the Previous Se..
  + Since we have previous spec for services, it means we can roll back to that previous spec
  + Sometimes docker on github has more in depth documentation
  + Type ‘docker service update --rollback [service name]’
    - Will use previous spec
    - Current spec then become the new previous spec
  + Constantly rolling back will just toggle between previous spec and current spec
  + Just one version back
* Configuring Rollback Policies
  + Configuration parameters --rollback-delay, --rollback-failure-action, etc
  + You can control how the rollback takes place
  + Can also setup up service configuration for rollback
    - --update-failure-actin=rollback
    - --update-max-failure-ratio: ratio of failed task before rollback happens
* Use --force to Test Changes to..
  + Can update without changing any values but do want to roll out new task for service
    - Type ‘docker service update --force [service]’
    - Kill of all tasks and create new ones
  + Can also be used to test update policy configuration settings
* Watching UpdateStatus During..
* Simulating and Monitoring an..
  + If failure occurs while updating, the update is paused if FailureAction is set to pause
* Resuming a Paused Update
  + Type ‘docker service update [service]’ to resume paused update
* Internal Container to Container..
  + Container may need to talk to each other
  + External and internal networking
* Ingress Network Is Special Pur..
  + Ingress network is special purpose network
    - Meant to route incoming traffic on published port
    - Created to properly route requests outside the cluster to service which are publishing ports in the swarm
    - Like user request to api
    - Get request into swarm and into container
    - Not meant for traffic between containers
  + Create custom network for container to container communication
  + Type ‘docker network create ..’
* Our New Network Topology
  + Outside -> underlay network -> nodes -> ingress overlay -> published ports -> custom network
    - Underlay network route external traffic to nodes
    - Then ingress routes to published ports
    - Custom network then routes between containers in swarm
  + Separating of public facing services and internal supporting services
* Creating an Overlay Network
  + Type ‘docker network create -d overlay --subnet=[subnet] [name]’
    - -d: driver
    - subnet: subnet in CIDR format that represents a network segment
    - ex subnet: 10.0.9.0/24
  + don’t have to specify subnet
    - don’t use subnet that you have on an existing network in your organization
    - if you do there could be problems with communication and routing
  + docker will specify 10.0.1.0/24 which could collide
* Inspecting Overlay Networks
* Attaching a New Service to Ou..
  + Type ‘docker service create --name [name] -p [host port]:[container port] --network [network name] [image]’
    - --network: specify network attachment
  + Type ‘open [ip address]’ to open browser to ip address
* Adding a Second Service to Ou..
  + Type ‘docker service create --name --network [network] [image]’
    - Leave out -p because we want this not to be accessible publicly
  + Adding two services on the same network doesn’t mean they can automatically talk to each other
  + Have to do other configurations
* Viewing Service Logs
  + Have to tell applications in the swarms what host name/ip address to use to communicate with another application
  + For the example application balance
    - It uses the process environment setting to talk
  + Type ‘docker service logs [service]‘
    - Stream logs for service
    - Compare the logs for the right task because old task logs will also be listed
  + We have service discovery available based on name of service when services are hooked up on to user defined overlay networks
  + Type ‘docker service update --env-add ..’
    - Set environment variable to point at customer
* Adding an Environment Variable…
  + Type ‘docker service update --env-add MYWEB\_CUSTOMER\_API=customer:3000 balance’
    - MYWEB\_CUSTOMER\_API is used in balanced app code to connect
    - --env-add: flag to set environment variable
    - This is cause old task to shutdown and new task to run
  + To get rid of old logs, scale down to 0 then up again
* docker exec to Check Service…
  + go to node with the container
  + type ‘docker exec -it [container id] bash’ to interactively connect to container
  + in container type ‘dig [service]’
    - get ip address of the service
* Spelunking Service Discovery a..
  + In container type ‘curl [ip address]’ to make call to api
  + The dns resolves customer to the assigned virtual ip
    - Load balancer ip address
    - Virtual ip is assigned to service
    - You hit the virtual ip and request is load balanced over backend containers
  + Type ‘dig tasks.[service]’ get ip address of containers for the service
  + IPVS is transport layer load balancing inside the linux kernel
  + Overlay network for windows
  + External load balancer and internal load balancer
  + All services automatically have load balancers in front of them
  + Dynamic load balancer hooked in via DNS into docker swarm mode
* Using Curl to Validate Internal …
  + In side of container type ‘curl [ip address]:[container port]..’
* Validating External Load Balancer
  + Browsers tends to keep connection alive
* docker service inspect for Find…
  + to find virtual ip address without exec into containers
  + type ‘docker service inspect [service]’
    - look at virtual ips section
  + each overlay network a service is connected will have its own ip address
* Use DNS Round Robing Instead
  + Service discovery is configurable
  + Uses IPVS behind the scene
  + Known as VIP endpoint mode
  + Type ‘docker service inspect [service]’ look at the endpointspec section
  + To change mode type ‘docker service update --endpoint-mode=[options] [service]’
    - Options include dnsrr
    - dnsrr: dns round robin
  + DNS RR
    - Use round robin via DNS
    - Removed load balancer
    - Client picks on of the ip address to communicate
    - Great if you want to remove load balancer or use your own load balancer
* Networks Are Lazily Extended…
  + When you create a network docker swarm will not extend network to any nodes that don’t need it
  + Only on nodes needs it
    - When running container is connected to network
  + All ip address for containers that are needed for service discovery is gossiped over to additional node in process of extending network to node
    - Why it is efficient node to extend network to nodes automatically
* A Few Last Things
  + Ability to have overlay network with service discovery traditional meant some external key value store
  + Now in swarm mode key value store is embedded inside of docker engine
    - Kept in memory
  + Internal services do not need ports
* Enough with All the Flags Alre…
  + Can put all flags into .yml config file
* Remove All Services
  + Type ‘docker service inspect [service] > [filename].inspect’
    - To dump configuration of service to a file
  + Type ‘docker service rm [service]’ to get rid of service
  + Type ‘docker network rm [network name]’ to get rid of network
* The New Compose Version 3 F..
  + Make a directory
    - Type ‘take [directory name]’ creates and switches into directory
  + Type ‘vi [file name]’ to create file
* Creatine a Compose File for O.
  + Create a .yml with by type ‘vi [service].yml’
    - version: ‘3.1’ //yaml version
    - services: //define services
    - viz: //name of service
    - image: [image name] //name of image, tag options
    - volumes: //specify volumes
    - - “[absolute path of host]:[absolute path to container]”
    - deploy:
    - placement:
    - constraints:
    - - node.role==manager
  + Compose file is like typing up all arguments that you would pass to docker service create or docker run
    - Like flags
    - Much easier approach to passing arguments
  + ‘docker service create --name viz --publish 8090:8080 --mount=type=bind,src=/var/run/docker.sock,dst=/var/run/docker.sock --constraint=node.role==manager manomarks/visualizer’
* Deploying a Stack with a Com..
  + Type ‘cat [filename]’ to display contents file on screen
  + Type ‘docker stack…’
    - Stack is management command
  + Stack has services
    - Services has tasks
    - Tasks correspond to containers that has applications
  + Type ‘docker stack deploy -c [path to compose file] [name of stack]’
    - A bundle file is json equivalent of a compose file
  + When you deploy a stack each stack gets its own network automatically
    - Services in stack are automatically hooked up to that network
  + All artifacts created from stack are prefixed by name of stack
  + Type ‘docker stack ls’ to list stack
  + Type ‘docker stack services [stack name]’ to list services stack contains
  + Type ‘docker stack ps [stack name]’ to list task for service
* Updating a Service with a Stack
  + To update a stack, edit viz.yml file and edit
  + Then deploy stack again ‘docker stack deploy -c [filename].yml [stack name]’
  + Docker swarm is smart enough to read file and see what needs to change
* Removing a Stack
  + Type ‘docker stack rm [stackname]’
* Creating and Deploying a Multi..
  + Application that are related they should be together a stack
  + A stack is a means managing multiple services
  + Create .yml file for balance and customer
    - version: ‘3.1’
    - services:
    - customer:
    - image: swarmgs/customer
    - balance:
    - image: swarmgs/balance
    - ports:
    - - “5000:3000”
    - environment:
    - MYWEB\_CUSTOMER\_API: “customer:3000”
  + Type ‘docker stack deploy -c [file] [stack name]’
  + Type ‘docker stack services [stack name]’ to look at services that are part of stack
  + Type ‘docker stack ps [stack name]’ to look at task associated with services for stack
* Specifying Replicas in Compos…
  + Can specify replicates with .yml file
  + In deploy section
    - customer:
    - image: swarmgs/customer
    - deploy:
    - replicas: 5
  + Then deploy the .yml file again
  + Load balancing applies to stack
  + Stack is just organization concept to quickly deploy services
* Quiz and Key Course Takeaway
  + docker run, docker service create(for swarm)
  + docker-compose, docker stack deploy(for swarm)
* The Trifecta Services - Network..
  + Look at the compose-file documentation on docker docs
  + Volume configuration
  + Network configuration
* A Running Process Is Not Nece..
  + Healthy check: if instance of service is healthy, advance checks
* Deploying a Cowsay Stack
  + Use cowsay.yml to launch stack
* What Happens if We Break a C…
  + Type ‘docker exec -it $(docker ps -f name=cow -q) bash’ to connect to container
    - $(docker ps -f name=cow -q) returns id for container
    - This is convenient since when you shut down container the id changes
  + If app doesn’t work app is still running, meaning docker thinks it is healthy
  + If app crash, where it is no longer running then a new task and new container will be created
  + Can tell docker what it means for app to be unhealthy
    - This information can be used as a condition to create new task and container
* Automatic Service Recovery wi..
  + Can add safety checks to cause docker to restart unhealthy tasks
    - Health checks
  + Build health check into image
* Manually Forcing a Corrupted ..
  + Type ‘docker service update --force [service name]’ to shut down task and create a new one
* Options for Adding Health Che…
  + Option 1: Update stack file
    - Add section ‘healthcheck’’
  + Option 2: call docker service create/update with health flags
  + Option 3: bake health check into image
  + Option 4: docker run reference health check
* Adding a Health Check to a Sta..
  + In .yml
    - version: ‘3.1’
    - services:
    - calc:
    - image: swarmgs/cacl
    - ports:
    - - “7000:80”
    - healthcheck:
    - test: curl -f -s -S [http://localhost/[endpoint](http://localhost/%5Bendpoint)] || exit 1
  + Health check options
    - test: command or probe that will check if application is healthy
      * Can be any command that is baked into container
      * Exec format, shell format
      * -f: silently fail, dont print output of website
      * -s: silent
      * -S: show eros
      * Exit returns 22, how docker know if health check succeed or not
      * Per docs its better to return exit 1
      * If curl fails then exit 1 is run
* Configuring Interval and Timeo..
  + There are default health check
  + Health check options
    - interval: amount of time between health check
    - timeout: how long to respond
    - retrys: how many retries but mark the container as unhealthy
* Deploying Health Checks and I..
  + Type ‘docker inspect [container id] | jq ‘.[].State’ to return just the state section of inspect
  + If you update with health check then will need new task
  + Have to do inspect on container and not stack to get health information
* Monitoring When Testing Healt..
* Monitoring Service Auto Recov..
  + With the specified health check, after 3rd failure docker will shutdown that task then create a new one
* Adjust the Health Check Interval
* Health Checks Prevent Traffic t..
  + Dns is responsive based on healthy status of container
  + Wont route traffic to containers while they are starting up and pass health checks
* Health Check Prevent Traffic t..
  + If application fails, the entry is gone from DNS
  + Once it restarts it gets added to DNS
* Health Checking with docker run
  + Type ‘docker run --rm -p 7000:80 --health-cmd “curl -f -s -S <http://localhost/calc/iseverythingok> || exit 1” --health-interval=5s swarmgs/cacl’
* Adding a Health Check to a Do..
  + What indicates failure for application and what is best way to check that
  + Can use curl for websites
* Disable Health Checking
  + Use flag --no-healthcheck when starting stacks, containers, etc
  + In .yml file
    - healthcheck:
    - disable: true
* Environment Variables Can Lea..
  + Want to keep sensitive information is protected is important
  + By using environment variable, it can leak sensitive information
  + ‘docker inspect [container id]’ too see environment variables
  + Secrets allow us to protect sensitive information
  + Alternative means as opposed to environment variables to inject sensitive information
  + Example traditional environment mysql.yml
    - version: ‘3.1’
    - services:
    - mysql:
    - image: mysql
    - environment:
    - MYSQL\_USER: wordpress
    - MYSQL\_DATABASE: wordpress
    - MYSQL\_ROOT\_PASSWORD: root
    - deploy:
    - placement:
    - constraints:
    - - node.role==manager
* Creating a Secret for a MySql
  + Docker has a management command ‘secret’ with sub commands to manage secret
  + Type ‘docker secret ls’ to list secrets
  + Type ‘echo [password] | docker secret create [name of file] -‘
    - Echo to product STDIN
    - -: to create a secret file from STDIN as content
* Granting a Service Access to a..
  + In .yml file you can then specify which secrets to use
  + Have to have a top level ‘secret’ section that defines all of the secrets
  + Then another ‘secret’ section under each service that specifies which secrets the service should receive
  + Can have external secrets and files secrets
  + Ex
    - …
    - services:
    - environment:
    - …..
    - secrets:
    - - mysql\_root\_pass
    - - my\_file\_secret
    - …
    - secrets:
    - mysql\_root\_pass:
    - external: true
    - my\_file\_secret:
    - file: my\_file\_secret.txt
  + Then deploy with .yml file
  + Inside stack or compose file, the project/stack is prepended to the beginning of whatever element it is
  + Inside of container, secrets are in the /run/secrets folder
  + Cd to /run/secrets
    - Type ls to get list of secrets
    - Type ‘cat [secret file name]’ to get contents
* Troubleshooting a Failing Service
  + Type ‘docker service logs [service]’ to see errors
  + Mysql will give errors
  + In .yml file add allow empty passwords for the time being
    - …
    - environment:
    - …
    - MYSQL\_ALLOW\_EMPTY\_PASSWORD: “yes”
* Accessing Secrets in a Container
  + Containers have access to secrets
  + Type ‘docker exec -it [container id] bash’ to get into container
  + Cd into run/secrets folder, where secrets is located
  + Secrets are available via the file system
  + File system is unique to the container
  + Have to pass value in file into mysql
* Using a Secret to Provide a My..
  + In .yml file add mysql root password file
    - …
    - environment:
    - ….
    - MYSQL\_ROOT\_PASSWORD\_FILE: “/run/secrets/mysql\_root\_pass”
  + Specific to each application
* Steps to Use Secrets
  + 1. Create secret
    - Password, ssh key, certfificates
    - <= 500 KB
    - Swarm only
    - Storage: encrypted, replicated raft log
    - Raft log: shared by managers
    - STDIN or file
    - Define in compose stack file
  + 2. Grant access to service
    - docker service create -secret X
    - stack compose file - secrets section
  + 3. App reads secret from /run/secrets/X
    - In-memory filesystem
    - Secrets will not be included in snap shot if you commit image
* \_FILE Image Secrets Convention
  + [environment variables]\_FILE convention
* Removing Secrets
  + Type ‘docker secret rm [secret name]’ to remove secret
    - Will get error if secret is in use
* A Convention for Updating a S…
  + Create and version secrets
    - Type ‘echo “passv1” | docker secret [secret name]\_v1 -‘
  + In .yml file
    - …
    - environment:
    - ….
    - secrets:
    - - source: mysql\_root\_pass\_v1
    - target: mysql\_root\_pass
  + Target is what is named in the file system
  + If target is not specified source is used and you have to update file location
  + Alias source name as target name
  + To update password just update .yml with the password version
  + Some update functionality maybe coming later
* The End
  + Keep an eye on swarmkit git hub