* Course Overview
  + Kubernetes building blocks
  + Role of pods
  + Services
  + Secrets
* Overview
  + Kubernetes from a developer perspective
  + Creating pods
  + Creating deployments
  + Creating services
  + Understanding storage options
  + Creating ConfigMaps and Secrets
* Introduction
  + Can use Kubernetes in non production environment
* Kubernetes Overview
  + Open source system for automating deployment, scaling and management of containerized applications
  + How are you managing containers
    - Load balance to server with containers
  + Can use docker compose to manage containers
    - Not meant for production but can be used
  + Kubernetes is the conductor of a container orchestra
  + Key Kubernetes Features
    - Service discovery, load balancing
    - Storage orchestration/volumes
    - Automate rollouts/rollbacks
    - Self healing
    - Secret and configuration management
    - Horizontal scaling
    - More features
* The Big Picture
  + Container and cluster management
  + Kubernetes open source project
  + Supported by all major cloud platforms
  + Provides a “declarative” way to define cluster’s state
  + One or more master nodes
    - Manage worker nodes
  + Together they create a cluster
  + Master will start a pod on each node
  + Pod is a way to host a container
  + Pod is suit and container is a person in the suit
  + Deployment and replicaset to deploy pods
  + Need service for pods to talk to each other and the outside world
  + Node is like a VM, can run one or more pods
  + Store and controller manager
    - Store is database for master node to track things
    - Controller manager deals with request scheduling
  + kubectl command line tool
  + kubelet registers node with cluster and talks to manager
  + container runtime
  + Kube-Proxy: unique ip address for pods
* Benefits and Use Cases
  + benefits
    - accelerate developer onboarding
    - eliminate app conflicts
    - environment consistency
    - ships software faster
  + Key Kubernetes benefits
    - orchestrate containers
    - zero downtime deployments
    - self healing
    - scale containers
  + develop use cases
    - emulate production locally
    - move from docker compose to kubernetes
    - create an end to end testing environment
    - ensure application scales properly
    - ensure secrets/config are working properly
    - performance testing scenarios
    - workload scenarios(CI/CD and more)
    - learn how to leverage deployment options
    - help devops create resources and solve problems
* Running Kubernetes Locally
  + Minikube
  + docker desktop
  + kind: kubernetes in docker
  + kubeadam: full kubernetes, administrators
  + click on docker desktop icon
    - click on on settings
    - click on kubernetes
    - check enable kubernetes
    - click apply & restart
  + if it doesnt work, click restart docker
* Getting Started with kubecti
  + kubectl command line tool
  + type ‘kubectl version’ to check kubernetes version
  + type ‘kubectl cluster-info’ to get cluster information
  + ‘kubectl get all’ retrieve information about kubernets pods, services, etc
  + ‘kubectl run [container-name] --image=[image-name]’ simple way to create a deployment for a pod
  + ‘kubectl port-forward [pod] [ports]’ to forward a port to allow external access
    - normally a pod has a cluster ip, so only items in cluster can talk to pod
    - port-forward to expose it to external items
  + ‘kubectl expose ..’ expose a port for a deployment/prod
  + ‘kubectl create [resource]’ create a resource
  + ‘kubectl apply [resource’ create or modify a resource
  + ‘kubectl get pods’ to get all the pods
  + ‘kubectl get services’ to get all the services
* Web UI Dashboard
  + Optional
  + Visual dashboard to inspect nodes, etc
  + Steps
    - ‘kubectl apply -f [dashboard-yaml-url]’
    - ‘kubectl describe secret -n kube-sytem’
    - Locate the Kubernetes.io/service-account-token and copy it
    - kubectl proxy
    - visit the dashboard url and login using the token
  + type ‘kubectl apply -f [dashboard yaml link]’ to apply yml to set up for dashboard
  + then type ‘kubectl describe secret -n kube-system’
    - copy the first token
  + then run ‘kubectl proxy’ opens port to connect to dashboard
  + then open the link they specify under the proxy command on the documentation
    - <http://localhost:8001/api/v1/namespaces/kubernetes-dashboard/services/https:kubernetes-dashboard:/proxy/>.
  + select token and paste in token you copied earlier
* Summary
  + Kubernetes provides container orchestration capabilities
  + Used for production, emulating production, testing and more
  + Several options to run Kubernetes locally
  + Interact with Kubernetes using kubectl
* Introduction
  + Pod core concepts
  + Create a pod
  + Kubectl and pods
  + Yaml fundamentals
  + Defining a pod with yaml
    - Declarative approach
  + Pod health
* Pod Core Concepts
  + Pod is the basic execution unit of a kubernetes application- the smallest and simplest unit in the Kubernetes object model that you create or deploy
  + Pods runs containers
    - Can have more than one container
  + Environment for containers
  + Organize application “parts” into pods(server, caching, APIs, database)
  + Pods have IP, memory, volumes, etc. shared across containers
  + Scale horizontally by adding pod replicas
  + Pods live and die but never come back to life
  + Master node schedules pod on a node
  + Pods can be scaled to have replicas
    - Kubernetes load balance between them
    - Kubernetes manages the health of the pods
  + Pods within a node have unique ip address, default is cluster ip
    - Containers within pods have their own unique ports
  + Pods containers share the same network namespace
  + Pods containers have the same loopback network interface
  + Containers processes need to bind to different ports within a pod
  + Ports can be reused by containers in separate pods
  + Pods do not span nodes
* Creating a Pod
  + There are several different ways to schedule a pod
    - kubectl run command
      * imperative way
    - kubectl create/apply command with a yaml file
      * declarative way
  + ‘kubectl run [podname] --image=nginx:alpine’ to run nginx:alpine container in a pod
  + ‘kubectl get pods’ list only pods
  + ‘kubectl get all’ list all resources
  + The ‘kubectl get’ command can be used to retrieve information about pods and many other Kubernetes objects
  + Pods and containers are only accessible within the Kubernetes cluster by default
    - Cluster ip address
  + One way to expose a container port externally: ‘kubectl port-forward’
  + ‘kubectl port-forward [name of pod] [host port/external port]:[container port/internal port]’
    - enables pod container to be called externally
  + ‘kubectl delete pod [name of pod]’ to delete pod
    - Pod will be delete but pod will be recreated
    - Kubernetes wants to maintain state
  + ‘kubectl delete deployment [name of deployment’ delete deployment that manages the pod
    - Pod will not be recreated
  + Running a pod will cause a deployment to be created
  + To delete a pod use kubectl delete pod or find the deployment and use kubectl delete deployment
* Kubectl and Pods
  + Deployment and replicas sets are responsible for actually deploying pods
  + A deployment is responsible for making sure that the current state is maintained
* YAML Fundamentals
  + Imperative: command centric approach to creating pods in Kubernetes cluster
  + Declarative: use language to define pods and other aspects of Kubernetes
  + YAML(yet another markup language)
  + Yaml files are text files composed of maps and lists
  + Similar to json
  + Key, value, sequence of items
  + Indentation matters
  + Always use spaces
  + Indentation determines when section ends and begins
  + Maps(key value pairs)
    - Can contain other maps for more complex data structures
  + Lists: sequence of items
    - Multiple maps can be defined in a list
  + Example
    - key:value
    - complexMap:
    - key1: value
    - key2:
    - subKey: value
    - items:
    - -item1
    - -item2
    - itemsMap:
    - -map1: value
    - map1Prop: value
    - -map2: value
    - map2Prop: value
* Defining a Pod with YAML
  + Run yaml file through kubectl will generate pod running in cluster
  + Ex yml file
    - apiVersion: v1 //kubernetes api version
    - kind: Pod //type of Kubernetes resource
    - metadata: //metadata about the pod
    - name: my-nginx
    - spec: //spec/blueprint for the pod
    - containers: //information about the containers that will run in the Pod
    - - name: my-nginx
    - image: nginx:alpine
  + ‘kubectl create -f [yml file] --dry-run --validate=true’
    - Perform a “trial” create and also validate the YAML
    - -f or --filename: specifies the file to use
    - --dry-run: try the command and see the output
    - --validate=true: default, perform validationa
  + ‘kubectl create -f [yml file]’
    - Create a pod from yaml
    - Will error if pod already exists
  + ‘kubectl apply -f [yml file]’
    - Create or apply changes to a pod using yaml
  + ‘kubectl create -f [yml file] --save-config’
    - Use --save-config when you want to sue kubectl apply in the future for updates
    - Store current properties in resource’s annotations
    - --save-config causes the resource’s configuration settings to be saved in the annotations
    - Having this allows in-place changes to be made to a pod in the future using kubectl apply
  + ‘kubectl delete pod [name of pod]’ to delete pod
  + ‘kubectl delete -f [yml file]’ delete the pod using the yaml file that created it
* Kubectl and YAML
  + Can link resources to each other through labels
  + Ex
    - labels:
    - app: nginx
    - rel: stable
  + Can define container port in yml file
  + Ex
    - …
    - containers:
    - …
    - ports:
    - - containerPort: 80
  + ‘kubectl get pod [pod name] -o yaml’ to output yaml file
    - -o: output flag
    - yaml: specifies output type of yaml
  + ‘kubectl describe pod [pod name]’ to get information about pod
    - Also about events that occur
  + ‘kubectl exec [pod name] -it sh’ to get into container of pod
  + ‘kubectl edit -f [yml file]’ opens editor
  + ‘kubectl delete -f [yml file]’
    - Since there is no deployment it will be deleted
* Pod Health
  + Kubernetes relies on probe to determine the health of a pod container
  + Probe: diagnostic performed periodically by the kubelet on a container
  + Liveness probe: is it alive
    - Used to determine if a pod is healthy and running as expected
  + Readiness probe
    - Can be used to determine if a pod should receive requests
  + failed pod containers are recreated by default(restartPolicy defaults to always)
    - if pod fails with a deployment it can recreated
    - if containers in pod fails health check it can be restarted
  + ExecAction: execute an action inside the container
  + TCPSocketAction: TCP check against the container’s IP address on a specified port
  + HTTPGetAction: Http Get request against containers
  + Probs can have the following results
    - success
    - failure
    - unknown
  + liveness probe example yml
    - apiVersion: v1
    - kind: Pod
    - ….
    - spec:
    - containers:
    - - name: my-nginx
    - image: nginx:alpine
    - livenessProbe: //livenessProbe, readinessProbe
    - httpGet:
    - path: /index.html
    - port: 80 //check /index.html on port 80
    - initialDelaySeconds: 15 //wait 15 seconds before
    - timeoutSeconds: 2 //timeout after 2 seconds
    - periodSeconds: 5 //check every 5 seconds
    - failureThreshold: 1 //allow 1 failure before failing pod
* Pod Health in Action
* Summary
  + pods are the smallest unit of kubernetes
  + containers run within pods
    - share a pod’s memory, IP, volumes and more
  + pods can be started using different kubectl commands
  + YAML can be used to create a pod
  + health checks provide a way to notify kubernetes when a pod has a problem
* Introduction
  + deployment core concepts, creating a deployment, kubectl and deployment, deployment options
* Deployments Core Concepts
  + ReplicaSet is declarative way to manage Pods
    - manages pods
  + Deployment is a declarative way to manage Pods using a ReplicaSet
  + Pods are created and destroyed but are never re-created
  + Deployments and ReplicaSets ensure pods stay running and can be used to scale pods
  + ReplicaSet
    - ReplicaSet -> pod -> container
    - acts as a pod manager
    - self-healing mechanism
    - ensure the requested number of pods are available
    - provide fault-tolerance
    - can be used to scale pods horizontally
    - relies on a pod template
    - no need create directly
    - used by deployments
  + Deployment
    - deployment -> ReplicaSet -> Pod -> Container
    - pods are managed using replicaset
    - scales replicaset, which scale pods
    - supports zero-downtime updates by creating and destroying replicasets
    - provides rollback functionality
    - creates a unique label that is assigned to the replicaset and generated pods
    - YAML is very similar to a ReplicaSet(different kind)
* Creating a Deployment
  + Define a deployment with YAML
  + Deployment yaml + kubectl
  + Ex yaml file
    - apiVersion: apps/v1 //find type in documentation
    - kind: Deployment //Pod, ReplicaSet, Deployment
    - metadata:
    - spec:
    - selector: //select pod template label(s)
    - template:
    - spec: //template used to crate the pods
    - containers:
    - - name: my-nginx
    - Image: nginx:alpine
  + Ex yaml file
    - apiVersion: apps/v1
    - kind: Deployment
    - metadata:
    - name: frontend
    - labels: //can be used to query and ties thing together
    - app: my-nginx
    - tier: frontend
    - spec:
    - selector: //used to select the template to use(based on labels)
    - matchLabels:
    - tier: frontend
    - template: //templated use to create the Pod/Containers
    - metadata:
    - labels:
    - tier: frontend
    - spec:
    - containers:
    - - name: my-nginx
    - image: nginx:alpine
* kubectl and Deployments
  + ‘kubectl create -f [deployment.yml]’ create a deployment
    - -f/--filename: specify file
  + ‘kubectl apply -f [deployment.yml]’ create or apply changes to deployment
  + ‘kubectl get deployments’ to get all deployments
  + ‘kubectl get deployment --show-labels’ list the labels for all deployments
    - Use -I switch to get information about a deployment with a specific label
  + ‘kubectl get deployment -l [name of label=value of label]’ to get all deployments with a specific label
  + ‘kubectl delete deployment [deployment-name]’ to delete a deployment
    - Will delete the deployment and all associated pods/containers
  + ‘kubectl delete -f [yaml file]’ to delete a deployment through using yaml file
  + ‘kubectl scale deployment [deployment-name] --replicas=[# of replicas]’ to scale the deployment pods to # of replicas
    - Can update the YAML file or use the kubectl scale command
  + ‘kubectl scale -f [deployment yaml] --replicas=[# of replicas]’ scale by referencing the yaml file
  + Update the YAML file
  + Ex
    - ….
    - spec:
    - replicas: 3
    - selector:
    - tier: frontend
* kubectl Deployments in Action
  + resources in yaml
  + ex
    - …
    - containers:
    - - name: …
    - ….
    - resources:
    - limits:
    - memory: “128Mi”
    - cpu: “200m”
  + Allows you to constrain what a given container is allowed to run inside of a pod
  + Containers without constraints can cause trouble
  + ‘kubectl scale -f [yml file] --replicas=[#]’ to scale from yml file
  + can also edit yml to scale
    - …
    - spec:
    - replicas: 4
    - selector: ….
  + ‘kubectl create -f [yml] --save.config’
  + ‘kubectl describe [pod | deployment] [pod-name | deployment-name]’
  + ‘kubectl apply -f [yml]’
  + ‘kubectl get deployments --show-lables’
  + ‘kubectl get deployments -l [label=labelVal]’
  + ‘kubectl scale -f [yml] --replicas=[#]’
* Deployments Options
  + Zero downtime deployments allow software updates to be deployed to production without impacting end users
  + Container -> Pod -> ReplicaSet -> Deployment
  + Kubernetes offers zero downtime deployments
  + Update apps pods without impacting end users
  + Several options
    - Rolling updates
    - Blue-green deployments
    - Canary deployments
    - Roll back
  + Blue-green
    - Multiple environments running at the same time
    - When new one is good switch traffic to new one
  + Canary
    - Small amount of traffic goes to new deployment
    - If good then switch all traffic over
  + Rollback
    - New version doesn’t work, go back to previous version
  + Rolling Deployments
    - New version rolls outs one by one but old ones still stays running
    - When new one is ready, then one is deleted
  + Can update a deployment by changing yaml and applying changes to the cluster
    - ‘kubectl apply -f [yml]’
* Zero Downtime Deployments in…
  + minReadySeconds: wait for pod to make sure container hasn’t crashed for certain amount of time
  + ex)
    - …
    - spec:
    - replicas: 3
    - minReadySeconds: 10
    - selector: …
  + Load balancing is based on connection to the server
    - Browsers create a single connection
    - Keep on same pod if using same connection
* Summary
  + Pods are deployed, managed, and scaled using deployments and ReplicaSets
  + Deployments are a higher-level resource that define one or more pod templates
* Introduction
  + Service core concepts
  + Service types
  + Creating a service with kubectl and yaml
* Service Core Concepts
  + A service provides a single point of entry for accessing one or more pods
  + Since pods live and die you cannot rely on their IPS
    - That’s why we need services, Ips change a lot
  + Pods can be horizontally scaled so each pod gets its own IP address
  + A pod gets an IP address after it has been scheduled
    - No way for clients to know IP ahead of time
  + Services abstract Pod IP addresses from consumers
  + Service has a fixed IP
  + It knows how to talk to the pods behind it
  + Labels are important, they used to associate pods with a service
  + Service load balances between pods
  + Relies on labels to associate a service with a pod
  + Node’s kube-proxy creates a virtual IP for Services
  + Layer 4(TCP/UDP over IP)
  + Services stick around
  + Create endpoints which sit between service and pod
* Service Types
  + ClusterIP: Expose the service on cluster-internal IP(default)
    - Service can talk to internal ip addresses and set it up for your pods
    - Service IP is exposed internally within the cluster
    - Only pods within the cluster can talk to the service
  + NodePort: expose the service on each nodes’s IP at a static port
    - Allocates a port from a range be default
    - Each node proxies the allocated port
  + LoadBalancer: provision an external IP to act as a load balancer for the service
  + ExternalName: maps a service to a DNS Name
    - Service acts as an alias for an external service
    - Allows a service to act as the proxy for an external service
    - External service details are hidden from cluster
* Creating a Service with kubectl
  + Port forwarding to access a pod from outside of Kubernetes
    - Forward a local port to a pod port
  + ‘kubectl port-forward pod/[pod-name] [external port]:[internal port]’
  + ‘kubectl port-forward deployment/[deployment-name] [external port]’
  + ‘kubectl port-forward service/[service-name] [external port]’
  + port-forward is useful when you want to expose a single pod
* Creating a Service with YAML
  + Yaml service + kubectl = service -> pod -> container
  + Service is the network to external resources
  + Ex
    - apiVersion: v1
    - kind: Service
    - spec:
    - type: //ClusterIP, NodePort, LoadBalancer, defaults to ClusterIP
    - selector: //select pod templates label(s) that service will apply to
    - ports: //define container target port and port for the service
  + Ex
    - apiVersion: v1
    - kind: Service
    - metadata:
    - name: nginx //name of service(each service gets a DNS entry)
    - labels:
    - app: nginx
    - spec:
    - selector:
    - app: nginx
    - ports:
    - - name: http
    - port: 80
    - targetPort: 80
  + Deployment yaml + service yml
    - Selector in service yml applies to any other pod or deployment with template of same selector
    - Service hooks itself to any pods that have that label
  + Name of service in metadata section, each service gets a DNS entry in the Kubernetes cluster
  + Node port ex)
    - apiVersion: v1
    - kind: Service
    - metadata:
    - …
    - spec:
    - type: NodePort
    - selector:
    - app: nginx
    - ports:
    - - port: 80
    - targetPort: 80
    - nodePort: 31000 //optionally set, defaults between 30000-32767
  + ExternalName service ex
    - apiVersion: v1
    - kind: Service
    - metadata:
    - name: external-service //other pods can use this FQDN to access the external service
    - spec:
    - type: ExternalName
    - externalName: api.acmecor.com //service will proxy to FQDN
    - ports:
    - - port: 9000
  + So the call to this service is ‘external-service:9000’
* Kubectl and Services
  + ‘kubectl create -f [service.yml]’ to create a service
    - Will generate clusterip be default unless otherwise specified
    - Service will then pick up deployed pods with specific labels
  + ‘kubectl apply -f [service.yml]’ to create or update a service
    - Update a service assumes --save-config was used with create
  + ‘kubectl delete service [service]’
  + ‘kubectl delete -f [service.yml]’
  + Testing a service and pod
    - ‘kubectl exec [pod-name] --curl -s [http://podIP](http://podip)’
  + To install curl
    - ‘kubectl exec [pod-name] -it sh’ to get into pod
    - Then type ‘apk add curl’
    - Then ‘curl -s [https://podIP](https://podip)’
  + The role of services
    - Is to give running pods ip addresses
* kubectl Services in Action
  + ‘kubectl get pod [pod name] -o yaml’ to get information about pod in yaml output
    - Get ip to be able to call it
  + ClusterIP
    - Only accessible within cluster
    - Helpful, when pods talks to other pods it is nice to not have to look up the ip of pods
  + Once you set up service, you should be able to access pods picked up by the service by going to the clusterIP:port of the service
  + Since service uses DNS you can also access pods by using [service-name]:[port]
  + NodePort
    - After deploying service can access pods using localhost:nodePort
    - Open a nodeport that can get to pod
  + Loadbalancer
    - Need to be called on top of cloud service
* Summary
  + Pods live and die so their IP address can change
  + Services abstract pod ip addresses from consumers
    - Get to pods without having to know pod ip addresses
  + Labels associate a service with a pod
    - Associate a service with one or more pods using labels
  + Service types
    - clusterIP(internal to cluster - default)
    - NodePort(exposes service on each’s node’s IP)
      * Open port on node to call into deployment or service
    - LoadBalancer(exposes a service externally)
    - ExternalName(proxies to an external service)
* Introduction
  + Storage core concepts
  + Volumes
  + Persistent volumes and persistent volume claims
  + Storage classes
* Storage Core Concepts
  + Can use volumes to store application state/data and exchange it between pods with Kubernetes
  + Volumes can be used to hold data and state for pods and containers
  + A pod can have multiple volumes attached to it
  + Containers rely on a mounthPath to access a volume
  + Kubernetes supports
    - Volumes
    - PersistentVolumes
    - PersistentVolumeCalims
    - StorageClasses
* Volumes
  + A volume references a storage location
  + Mush have unique name
  + Attached to a pod and may or may not be tied to the pod’s lifetime
  + A volume mount references a volume by name and defines a mountPath
  + Volumes types
    - emptyDir: empty directory for storing “transient” data(shares a pods lifetime)
      * useful for sharing files between containers running in a Pod
      * files that pods needs during their lifetime
      * if pods goes down, this would go down as well
    - hostPath: pod mounts into the node’s filesystem
      * easy to setup
      * if node goes down, can lose data
    - nfs: an NFS(Network file system) share mounted into the pod
    - configMap/secret: special types of volumes that provide a pod with access to Kubernetes resources
    - persistenVolumeClaim: provides pods with a more persistent storage option that is abstracted from the details
    - Cloud: cluster wide storage
  + A lot more volume types
  + emptyDir volume ex
    - apiVersion: v1
    - …
    - spec:
    - volumes:
    - - name: html
    - emptyDir: {}
    - containers:
    - - name: nginx
    - image: nginx:alpine
    - volumeMounts:
    - - name: html
    - mountPath: /usr/share/nginx/html
    - readOnly: true
    - - name: html-updater
    - image: alpine
    - command: [“/bin.sh”, “-c”]
    - args:
    - - while true; do date >> /html/index.html;
    - sleep 10; done
    - volumeMounts:
    - - name: html
    - mountPath: /html
  + hostPath volume ex(mount to volume on node)
    - apiVersion: v1
    - kind: Pod
    - spec:
    - volumes:
    - - name: docker-socket
    - hostPath:
    - path: /var/run/docker.sock
    - type: socket
    - containers:
    - - name: docker
    - image: docker
    - command: [“sleep”]
    - args: [“10000”]
    - volumeMounts:
    - - name: docker-socket
    - mountPath: /var/run/docker.sock
  + Cloud Volumes
    - Azure - azure disk and azure file
    - AWS - elastic block store
    - GCP - GCE persistent disk
  + Azure file volume
    - apiVersion: v1
    - kind: Pod
    - metadata:
    - name: my-pod
    - spec:
    - volumes:
    - - name: data
    - azureFile:
    - secretName: <azure-secret>
    - shareName: <share-name>
    - readOnly: false
    - containers:
    - - image: someimage
    - name: my-app
    - volumeMounts:
    - - name: data
    - mountPath: /data/storage
  + ‘kubectl describe pod [pod-name]’ will also show volumes attached to pod
  + ‘kubectl get pod [pod-name] -o yaml’ to show volumes attached to pod
* Volumes in Action
  + EmptyDir
    - Need to share data between containers inside a pod
    - If pod goes away, data is not maintained
  + HostPath
    - Mount to directory on host, on worker node where pod is running
    - Talk to docker daemon on worker node
    - Locked down to work node where pod is running
* PersistentVolumes and Persistent..
  + PersistenVolume(PV) is a cluster wide storage unit provision by an administrator with a lifecycle independent from a pod
    - NAS(network attached storage)
    - Cloud or local machine
    - Normally provisioned by a cluster administrator
    - Available to a pod even if it gets rescheduled to a different node
    - Rely on a storage provide such as NFS, cloud storage, or other options
    - Associated with a pod by using a PersistentVolumeClaim(PVC)
  + PersistentVolumeClaim(PVC): a request for a storage unit(PV)
  + emtpyDir tied to pod, hostPath tied to worker node
  + PersistentVolume workflow
    - 1) create network storage resource(NFS, cloud, etc.)
    - 2) define a persistent volume(PV) and register to the Kubernetes api
    - 3) create a persistent volume claim(PVC)
    - 4) Kubernetes binds the PVC to the PV
    - 5) Pod volume references the PVC
* PersistentVolume and Persisten…
  + Github site for different yaml example
    - https://github.com/kubernetes/examples
  + Example defining a persistent volume for azure
    - apiVersion: v1
    - kind: PersistentVolume
    - metadata:
    - name: my-pv
    - spec:
    - capacity: 10Gi
    - accessModes:
    - - ReadWriteOnce //one client can mount for read/write
    - - ReadOnlyMany //many clients can mount for reading
    - persistentVolumeRelaimPolicy: Retain //retain even after claim is deleted
    - azureFile: //reference storage to use(specific to provider)
    - secretName: <azure-secret>
    - shareName: <name\_from\_azure>
    - readOnly: false
  + Example: defining a PersistentVolumeClaim
    - apiVersion: v1
    - kind: PersistentVolumeClaim
    - metadata:
    - name: pv-dd-account-hdd-5g
    - annotations:
    - Volume.beta.kubernetes.io/storage-class: accounthdd
    - spec:
    - accessMode:
    - - ReadWriteOnce
    - resources:
    - requests:
    - storage: 5Gi
  + Example using a persistentvolumeclaim
    - apiVersion: v1
    - kind: Pod
    - metadata:
    - name: pod-uses-account-hdd-5g
    - labels:
    - name: storage
    - spec:
    - containers:
    - - image: nginx
    - name: az-c-01
    - command:
    - - /bin/sh
    - - -c
    - - while true; do echo $(date) >> /mnt/blobdisk/outfile; sleep 1; done
    - volumeMounts:
    - - name: blobdisk01
    - mountPath: /mnt/blobdisk
    - volumes: //create volume that binds to PersistentVolumeClaim
    - - name: blobdisk01
    - persistentVolumeClaim:
    - claimName: pv-dd-account-hdd-5g
* StorageClasses
  + a type of storage template that can be used to dynamically provision storage
  + used to define different “classes” of storage
  + act as a type of storage template
  + supports dynamic provisioning of PersistentVolumes
    - admin sets up storage class
    - we request it through PVC and it can dynamically set up PersisentVolume(PV)
  + administrators don’t have to create PVs in advance
  + StorageClass Workflow
    - 1) create storage class
    - 2) create PersistentVolumeClaim that references storageclass
    - 3) kubernetes use StorageClass provisioner to provision a PersistentVolume
    - 4) storage provisioned, PersistentVolume created and bound to PersistentVolumeClaim
    - 5) pod volumes references PersistentVolumeClaim
  + local storage StorageClass
    - apiVersion: storage.k8s.io/v1
    - kind: StorageClass
    - metadata:
    - name: local-storage
    - reclaimPolicy: Retain //retrain or delete(default) after PVC is released
    - provisioner: kubernetes.io/no-provisioner //Provisioner(volume plugin) that will be used to create PresistentVolume resource
    - volumeBindingMode: WaitForFirstConsumer //wait to create until Pod making PVC is created. Default is immediate(create once PVC is created)
  + defining a local storage PersistentVolume
    - apiVersion: v1
    - kind: PersistentVolume
    - metadata:
    - name: my-pv
    - spec:
    - capacity:
    - storage: 10Gi
    - volumeMode: Block
    - accessModes:
    - - ReadWriteOnce
    - storageClassName: local-storage
    - local:
    - path: /data/storage
    - nodeAffinity: //select the node where the local storage PV is created
    - required:
    - nodeSelectorTerms:
    - - matchExpressions:
    - -key: kubernetes.io/hostname
    - operator: In
    - values:
    - - <node-name>
  + PersistentVolumeClaim
    - apiVersion: v1
    - kind: PersistentVolumeClaim
    - metadata:
    - name: my-pvc
    - spec:
    - accessModes:
    - - ReadWriteOnce
    - storageClassName: local-storage
    - resources:
    - requests:
    - storage: 1Gi
  + using a PersistentVolume Claim
    - apiVersion: apps/v1
    - kind: [Pod | StatefulSet | Deployment]
    - …
    - spec:
    - volumes:
    - - name: my-volume
    - persistentVolumeClaim:
    - claimName: my-pvc
* PersitentVolumes in Action
  + Kubernetes StatefulSet: manages the deployment and scaling of a set of pods and provides guarantees about the ordering and uniqueness of these pods
  + Dynamically provisioning PersistentVolumes
* Summary
  + Ephemeral storage(emtyDir)
  + Persistent Storage
  + PersistentVolumes, PersistentVolumeClaims, and StorageClasses
  + ConfigMaps(key/value pairs)
  + Secrets
* Introduction
  + ConfigMaps Core Concepts
  + Creating a ConfigMap
  + Using a ConfigMap
  + Secrets Core Concepts
  + Creating a Secret
  + Using a Secret
  + Storage/ConfigMaps/Secrets
* ConfigMaps Core Concepts
  + Provide a way to store configuration information and provide it to containers
  + Provide a way to inject configuration data into a container
  + Can store entire files or provide key/value pairs
    - Key is the filename
    - Values is the file contents
    - Provide on the command-line instead of file
    - ConfigMap manifest(YAML)
  + ConfigMaps can be accessed from a pod using
    - Environment variables(key/value)
    - ConfigMap Volume(access as files)
* Creating a ConfigMap
  + Define values in a ConfigMap Manifest
    - apiVersion: v1
    - kind: ConfigMap
    - metadata:
    - name: app-settings
    - labels:
    - app: app-setting
    - data:
    - enemies: aliens
    - lives: “3”
    - enemies.cheat: “true”
    - enemies.cheat.level=noGoodRotten
  + ‘kubectl create -f [file.configmap.yml]’ create from a ConfigMap
  + Define key/value pairs in a file
    - enemies=aliens
    - lives=3
    - enemies.cheat=true
    - enemies.cheat.level=noGoodRotten
  + ‘kubectl create configMap [cm-name] --from-file=[path-to-file]’ to create a ConfigMap using data from a file
  + ‘kubectl create configMap [cm-name] --from-env=[path-to-file]’ to create a ConfigMap using data from an Env file
  + ‘kubectl create configMap [cm-name] --from-literal=apiUrl=https://my-api --from-literal=otherKey=othervalue’ create a ConfigMap from individual data values
* Using a ConfigMap
  + Kubectl get cm can be used to get a ConfigMap and view its contents
  + ‘kubectl get cm [cm-name] - o yaml’
  + Pods can access ConfigMap values through environment vars environment variable create
  + Ex load specific values
    - apiVersion: apps/v1
    - …
    - spec:
    - template:
    - …
    - spec:
    - containers: …
    - env:
    - - name: ENEMIES
    - valueFrom:
    - configMapKeyRef:
    - name: app-settings
    - key: enemies
  + Ex load all ConfigMap keys/values into environment variables
    - Use envFrom
  + Ex
    - apiVersion: apps/v1
    - …
    - spec:
    - template:
    - ….
    - spec:
    - containers:
    - envFrom:
    - - configMapref:
    - name: app-settings
  + ConfigMap values can be loaded through a volume
  + Each key is converted to a file - value is added into the file
  + Ex
    - apiVersion: apps/v1
    - ….
    - spec:
    - template:
    - …
    - spec:
    - volumes:
    - - name: app-config-vol
    - configMap:
    - name: app-settings
    - containers:
    - volumeMounts:
    - - name: app-config-vol
    - mountPath: /etc/config
* ConfigMaps in Action