Kubernetes

Saturday, December 5, 2020 8:44 PM

Key Terminology

Thursday, July 30, 2020 8:40 PM

- **General Terms**
  - Virtual Ethernet Device (veth)
    - Consists of two virtual interfaces that can be spread across multiple namespaces allowing traffic to flow between the two namespaces
  - Linux Ethernet Bridge
    - Virtual layer 2 networking device used to connect two or more network segments (namespaces) using ARP to discover link-layer MAC addresses associated with an IP address
  - netfilter / iptables
    - Netfilter is a firewall framework for Linux which consists of many modules including iptables which is the primary module
  - IP Virtual Server (IPVS)
    - Built on top of netfilter
    - Acts as a Network Load Balancer (Layer 4 LAN Switching) as part of Linux Kernal
    - Runs on a host and acts as a load balancer in front of a cluster of real servers and makes services on real services appear as virtual services on a single IP
- **Kubernetes Terminology**
  - Init container
    - One or more initialization containers that must run to completion before any app containers
    - In the pod spec use the *initContainers* heading
  - headless service
    - Used when you do not need load balancing and a single IP and want to achieve the discovery process outside of Kubernetes
  - node affinity
    - Property of pods that attracts them to a specific set of nodes either as a preference or a hard requirement
  - Binding
    - Process by which the scheduler notifies the API server about the decision as to where to schedule a pod
  - Eviction
    - Process by which pods on a node are deleted
    - If a node becomes unavailable for more than 5 minutes pods are evicted from the node
    - Also happens when node runs out of resources
  - Runtime class
    - Property of a pod that can allows you to run different pods with the same container runtime but with different settings
    - Use case is may a highly secure workload running on a machine that supports

hardware isolation

- Static Pod
  - Managed directly by kubelet daemon on a specific node without the API server observing them
  - Mirror object created in api-server so it can be displayed with kubectl
  - Used primarily for boostrapping Kubernetes itself such as with kubeadm

- Pod Disruption Budget (PDB)
  - Limits number of Pod of a replicated application that are down simultaneously from voluntary disruptions
  - Prevents a cluster administrator from potentially draining too many nodes causes an application to have an insufficient number of pods running

- Ephemeral Containers
  - Lack guarantees for resources and executions and never automatically restarted, no ports (so no probes), no setting resources
  - Useful for interactive troubleshooting when kubectl exec is insufficient

- Proporational Scaling
  - Supported by RollingUpdate Deployments to run multiple versions of an application at the same time
  - If scaling request comes in during rollout, the replicas are spread among the two replica sets

- Garbage Collection
  - Control whose responsibility is to remove cleanup objects that once had an owner but do not anymore

- Service Discovery
  - Can be provided by Kubernetes API Server which will provide a dynamic listing of endpoints available for a particular service which is kept up to date as Pods are brought up and down

- sidecar
  - Another container within a pod that is used to run another specific task such as collecting and aggregating logs

- Container Storage Interface (CSI)
  - Standard to expose block and file storage systems to containerized workloads on Kubernetes

- Container Network Interface (CNI)

- Persistent Volume
  - Allows abstraction of details of how storage is provided from the users of a cluster
  - Lifecycle is independent of a Pod unlike a Volume
  - Clusterwide storage and created by the administrator

- Persistent Volume Claims
  - Every persistent volume claim is bound to a single persistent volume and each volume can only be bound to a single persistent claim
  - Claims are bound to persistent volumes based upon the properties of the request

        - Uses properties such as capacity, access modes, volume modes, storage class, and standard selectors
        - Persistent Volumes are retained (default), deleted, or recycled after a PVC is deleted

    - Storage Classes
        - Defines a dynamic provisioner which provisions storage based upon apps request them
        - Persistent volume claims can directly reference a storage class negating the need for a persistent volume

General Concepts

Thursday, July 30, 2020 8:49 PM

- **General Facts**
  - Default pull policy for images in Kubernetes is IfNotPresent but be modified in the container spec by setting imagePullPolicy to Always
  - CoreDNS has a TTL for cached responses of 30 seconds

- **Cluster Resources vs Namespace Resources (Common examples)**
  - Namespace Resources
    - Pods
    - Replicasets
    - Jobs
    - Deployments
    - Services
    - Secrets
    - Roles
    - Rolebindings
    - Configmaps
    - Persistent Volume Claim
  - Cluster Resources
    - Nodes
    - Persistent Volumes
    - Clusterroles
    - Clusterrolebindings
    - Certificatesigningrequests
    - Namespace

- **kubeadm vs non-kubeadm Deployment**
  - In kubeadm deplolyment,kube-apiserver is deployed as a pod
  - In kubeadm deployment kube-controller-manager is deployed as pod
  - In kubeadm deployment kube-scheduler is deployed as pod
  - In kubeadm deployment kube-proxy deployed as a pod via daemon sets on each worker
  - In non-kubeadmn deployment kube-apiserver is deployed as a service
  - In non-kubeadmn deployment kube-controller-manager deployed as a service
  - In non-kubeadmn deployment kube-scheduler deployed as a service
  - In non-kubeadmn deployment kube-proxy is deployed as a service

- **Kubernetes API Structure**
  - **Core API - /api/v1**
    - All standard and long-lived resources
    - Examples/
    -

  - **Named API - /apis/<API_NAME>**
    - This will be the API all new features are added to
    - Examples
      - /apps/v1
        - Deployments

        - Replicasets
        - statefulsets
    - /networking.k8s.io
        - networkpolicies
    - /storage.k8s.io
    - /authentication.k8s.io
    - /rbac.authorization.k8s.io/v1
        - Role
    - /certificates.k8s.io
        - certificatesigningrequests

Management Plane

Monday, August 10, 2020 2:38 PM

- ***Control Plane***
  - **Key Facts**
    - Make global decisions about the cluster such as scheduling and responding to cluster events such as starting new pods

  - **Worker Components**
    - **kubelet**
      - Runs as a service
      - Agent that runs on each node in the cluster and makes sure containers are running in a pod
      - Always manually installed on worker nodes
    - **kube-proxy**
      - Maintains network rules on nodes which allow network communication to your pods from network sessions outside your cluster
      - Uses OS packet filtering layer if available other forwards traffic itself
    - **Container runtime**
      - Software responsible for running the containers
      - Examples: Docker, containerd, CRI-O

  - **Manager Components**
    - **kube-apiserver**
      - Exposes Kubernetes API which is the front-end of the control plane
      - Scales horizontally
      - Responsibilities
        - Authenticate User
        - Validate Request
        - Retrieve Data
        - Update ETCD
        - Communicate with Scheduler
        - Communicate with Kubelet

    - **etcd**
      - Runs as service
      - Consistent and highly available key value store used as Kubernetes backing store for all cluster data

    - **kube-scheduler**
      - Watches for newly created pods with no assigned node and selects a node for them to run on

    - **kube-controller-manager**
      - Runs controller processes
      - Consists of separate controllers wrapped into a single binary named *kube-controller-manager*
      - Node controller
        - Responsible for noticing and responding when nodes go down
          - *Monitors health via kube-apiserver every 5 seconds*
          - *Grace period of 40s before marked as unreachable*

        - *Given 5 minutes before Pods are evicated*
      - Assigning CIDR block to node when registered (if CIDR assignment on)
      - Keeps node controller's internal list of nodes up to date with cloud provider's list of available machines
    - Replication controller
      - Responsible for maintaining correct number of pods for every replication controller object in the system
    - Endpoints controller
      - Populates the endpoints objects
    - Service account and token controllers
      - Create default accounts and API access tokens for new namespaces

  - **Cloud-controller-manager**
    - Embeds cloud-specific logic and links cluster to cloud provider's API
    - Separates out components that interact with the cloud platform from components that interact with your cluster
    - Only runs controllers specific to cloud provider
    - Not present if running on-premises
    - Wrapped into a single binary and scales horizontally same as kube-controller-manager
- ***etcd***
  - **Key Facts**
    - etcd is a distributed key value store that provides a reliable way to store data across a cluster of machines
    - Used by Kubernetes to store all of its internal data about cluster state and synchronize across all controller nodes
    - Installed on each controller node to create a etcd cluster
    - Runs as a service on each controller node
  - **Configuration**
    - Etcd requires root CA public-key cert used to issue certificates to Kubernetes cluster and both public and private key certificates for API in etc/etcd directory

Storage

Monday, December 7, 2020 7:03 PM

- ***Pod Storage***
  - By default storage used by containers within a Pod is ephemeral
  - Volumes can be used to provide persistent storage of data to containers within a Pod

- Persistent Volumes
  - Key Facts
    - Access mode for a Persistent Volume can be
      - ReadOnlyMany
      - ReadWriteOnce
      - ReadWriteMany
    - An overall capacity is defined for the Persistent Volume
    - It also has a type which could be hostPath (local host) or another storage class such as public cloud provider

Networking

Sunday, September 27, 2020 8:31 PM

- ***Key Terminology***
  - **Virtual Ethernet Device (veth)**
    - Consists of two virtual interfaces that can be spread across multiple namespaces allowing traffic to flow between the two namespaces
  - **Linux Ethernet Bridge**
    - Virtual layer 2 netw**or**king device used to connect two or more network segments (namespaces) using ARP to discover link-layer MAC addresses associated with an IP address
  - **Services**
    - Expose an application running on a set of Pods as a network service
  - **Ingress**
    - Object that manages external access to services in a cluster and may provide load balancing, SSL termination, and name-based virtual routing
  - **Cluster CIDR**
    - IP range used to assign IPs to Pods in the cluster
  - **Service Cluster IP Range**
    - IP range for services in the cluster
    - Must not overlap with cluster CIDR range
  - **Pod CIDR**
    - IP range for Pods on a specifc worker node
    - Should fall within the cluster CIDR but not overlap with the Pod CIDR of other worker nodes
  - **Headless Services**
    - Use to interface with other service discovery mechanisms without being tied to Kubernetes implementation

- ***Key Facts***
  - One virtual network for all hosts within a cluster
  - Each Pod has a unique IP within the cluster
  - Each Service has a unique IP that is in a different range than Pod IPs
  - All Pod can communicate with other Pods within a clsuter and can reach each other's ports on localhost
  - All Pods can see each other without NAT

- ***Standard Concepts***
  - **DNS**
    - *Key Facts*
      - Service Structure: <SERVICE_NAME>.<NAMESPACE>.svc.cluster-domain.example

        - Pod Stucture: <POD_IP_ADDRESS>.<NAMESPACE>.pod.clusster-domain.example
        - kubeadm deploys CoreDNS for DNS resolution within the cluster by default
    - *CoreDNS*
        - Configuration file provided to the CoreDNS containers in the CoreDNS deployment is a configmap
        - The service fronting the CoreDNS deployment is name kube-dns

- **CNI Detection Process**
    - Kubelet uses the following parameters to identify to use a CNI and to set up each Pod's network
        - *--network-plugin=cni*
            - Indicates that a CNI is to be used
        - *--cni-confi-dir=<DIRECTORY>*
            - Indicates the directory of the CNI configuration file to use
            - By default /etc/cni/net.d
        - *--cni-bin-dir=<DIRECTORY>*
            - Indicates the directory of the CNI plugins
            - By default /opt/cni/bin

- **IPVS vs iptables**
    - iptables runs into performance issues with >5,000 services because iptables rules are evaluated sequentially
    - IPVS scales more effectively for >5,000 services because it uses a hash table managed by the kernal to determine the destination of a packet

- **kubenet**
    - Very basic network plugin available on linux only and no support for advanced features like Network policy
    - Run with --network-plugin=kubenet
    - Pods assigned network through --pod-cidr kubelet command line option or --allocate-node-cidrs=true through controller-manager

- ***Control Plane to Node***
    - **API server to kubelet**
        - API server communicates with kubelet running on nodes to fetch logs for pods, attach running pods, and provide port-forwarding functionality
        - Connections terminate at kubelet's HTTPS endpoint
        - By default API server doesn't validate the kubelets server certificate but this behavior can be modified using the --kubelet-certificate-authority flag

    - **API Server to nodes, pods, and services**
        - By default communicate to node, pod, or service is done via HTTP so no encryption in transit or authentication of the node, pod, service
        - Can specify HTTPS but server certificate will not be verified
        - Do not do this type of communication over untrusted/public networks

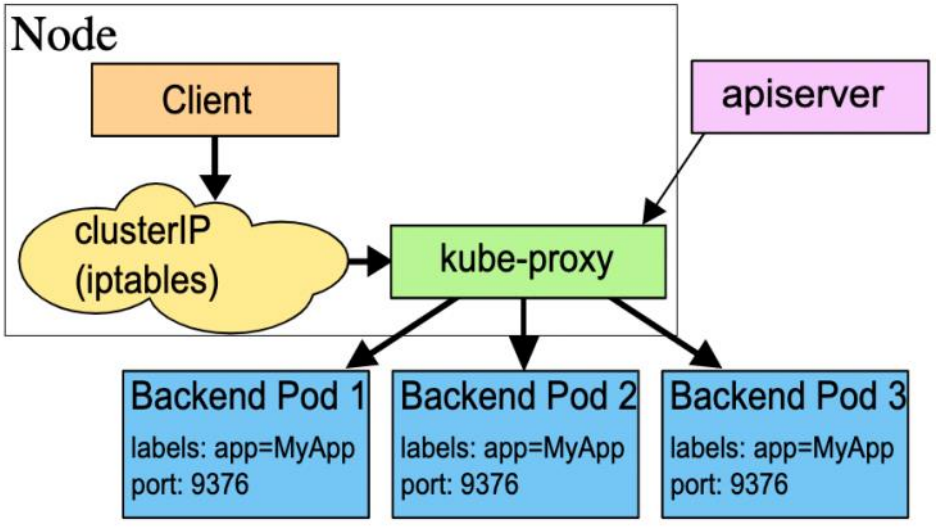
    - **SSH tunnels**
        - SSH tunnels to protect control plane to node communication is supported but is deprecated

    - **Connectivity service**
        - New replacement for SSH tunnels for control plane to node communication

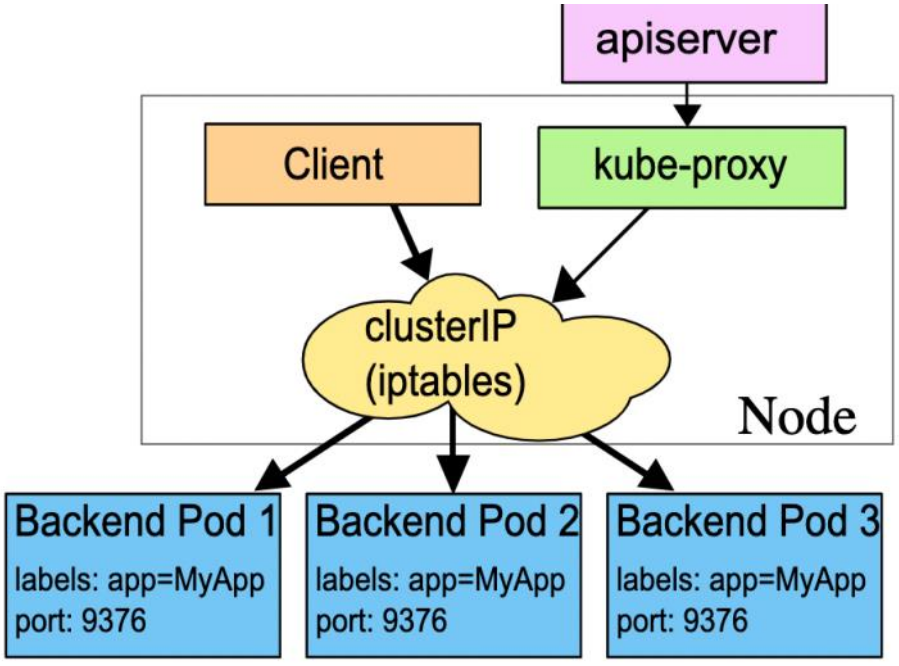
        - TCP level proxy for the control plane to cluster communication
        - Konnectivity server runs on control plane network and Konnectivity agent runs on node network
        - Agents initiate connections to server and all control plane to node traffic goes through the connection

- ***Pod to Control Plane Communication***
    - Pods communicate with API server using service account
    - When pod uses service account Kubernetes automatically injects cluster public root certificate and valid bearer token into pod when it is instantiated
    - Pods communicate with API server using a VIP that is redirected via kube-proxy to HTTPS endpoint on the API server

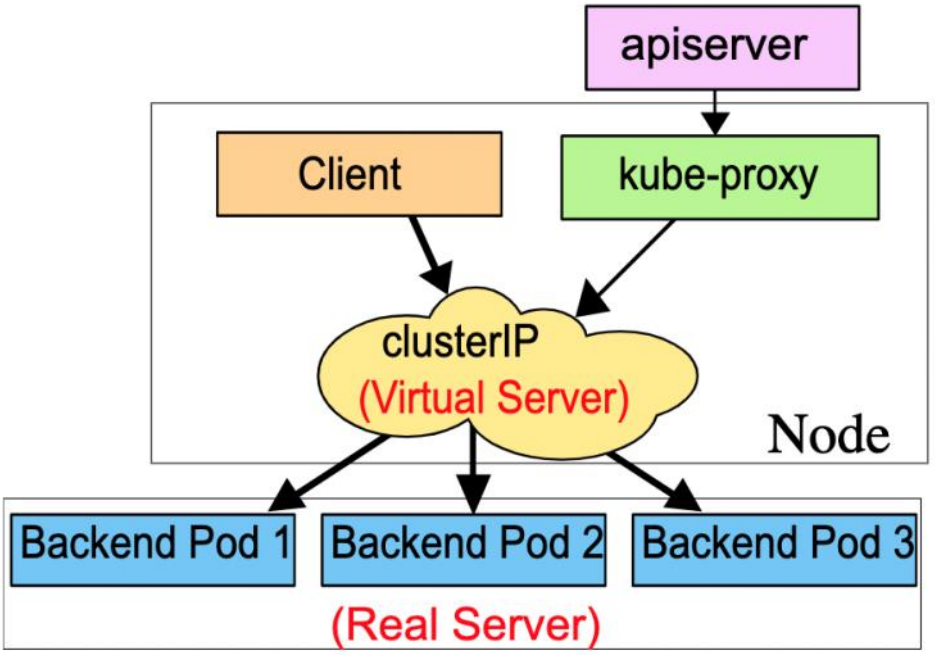
- **kube-proxy Modes**
    - ***User space proxy mode***
        - kube-proxy watches K8 master for addition and removal of Service and Endpoint objects
        - For each service it opens port randomly on the local node and connection to proxy port are proxied to one of Services's backend pods (as reported by Endpoints)
        - Honors SessionAffinity
        - Installs iptables rules which capture traffic to Service's clusterIP and Port which redirect traffic to proxy port which proxies to backend Pod
        - Backend chosen by round-robin by default and if backend connection to first Pod fails automatically try with a different backend Pod
        -



    - ***iptables Proxy Mode***
        - kube-proxy watches K8 control plane for addition and removal of Service and Endpoint objects
        - Nodes see traffic from original ciient IP
        - Chooses backend at random
        - Lower system overhead because traffic is handled by Linux netfilter
        - If first Pod that's selected does not respond then the connection fails so should use Readiness probes



- ***IPVS Proxy Mode***
  - Kube-proxy watches K8 service for Services and Endpoints and calls netlink interface to open IPVS rules and synchronizes rules with K8 services and endpoints periodically
  - IPVS directs traffic to one of backend Pods
  - Offers additional options for load balancing traffic to Pods such as round robin, least connection, destination hashing, source hashing, shortest expected delay, never queue
  - Best option for large deployments (>10,000 services)



- ***Kubernetes Resources***
  - **Services**
    - *Key Facts*
      - Have a DNS label of <SERVICE_NAME>.<NAMESPACE>.svc.cluster.local

        - Assigned an IP address from the --service-cluster-ip-range argument set for the kube-apiserver process
        - Can be used to abstract resources besides Pod by not using a selector
        - NodePort types choose a random port between 30,000 to 32,767
    - *Types*
        - ClusterIP
            - Exposes service on a cluster-internal IP; reachable only within the cluster
            - USE CASE: Microservice-based application
        - NodePort
            - Exposes the service on each Node's IP at a static port
            - NodePort Service will route to an automatically created ClusterIP Service
        - LoadBalancer
            - Exposes the service externally using cloud provider's load balancer
            - NodePort Service and ClusterIP Service which loadbalancer routes to are automatically created
        - ExternalName
            - Maps Service to contents of externalName field (such as myservice.hello.com) by returning CNAME record with its value
            - No proxying occurs
            - No selector used
- **Ingress**
    - *Key Facts*
        - Kubernetes does not come with an ingress controller by default and you must add one if you want this functionality
        - GCP and NGINX ingress controllers are maintained by Kubernetes
    - *Components*
        - Ingress Controller
            - With nginx, there is a spec file that can be used to deploy the controller
            - Additionally needs a Service, Service Account, ClusterRole, and RoleBinding
        - Ingress Resource
            - Set of rules and configurations applied on Ingress Controller
            - Additional rules for each domain name and additional paths for each set of Pods (service)
            - Remember to set a default rule as a catchall

- **Resources**
    - Great comparison of iptables vs IPVS - https://www.objectif-libre.com/en/blog/2018/03/19/kubernetes-ipvs/#:~:text=The%20IPVS%20implementation%20differs%20from,massive%20packet%20processing%2C%20performances%20collapse.

Security and Administration Concepts

Friday, July 31, 2020 12:28 PM

- ***Concepts***
  - ***Roles and Role Bindings***
    - **Key Facts**
      - Scoped to a namespace
      - In the definition for a role the apiGroups property can be left blank if referencing a resource in the core/v1 API
      - Roles are associated with a user using a RoleBinding
  - ***Cluster Roles and Cluster Role Bindings***
    - **Key Facts**
      - Scoped to the entire cluster
      - Can be used to grant access to namespace-scoped resources across an entire cluster
      - ClusterRoles are associated with a user using a ClusterRoleBinding
  - ***Network Policies***
    - ***Key Facts***
      - Applied to Pods to control ingress and egress traffic
      - Not supported by all CNIs
  - ***Kubeconfig***
    - **Key Facts**
      - kubectl by default references config file in $HOME/kube/config
      - Designate a specific kubeconfig file using the --kubeconfig <CONFIG_FILE> option of kubectl
      - Configuration file that stores information about clusters, users, namespaces, and authentication mechanisms
      - Contains data needed to connect to and interact with one or more Kubernetes clusters
    - **Structure**
      - Clusters - location of the cluster (hostname or IP address)
      - Contexts - marry the clusters and users together to determine which user will be used to access a cluster
      - Users - user accounts of which you have access to
  - ***Run a Pod or Container with a Security Context***
    - **Key Facts**
      - Security context can be configured at the Pod or individual container
      - Capabilties can only be added at the container level
- ***PKI Dependences***
    - **Key Facts**
      - Clusters setup with kubeadm make the first master node the CA for the cluster
      - Kubernetes supports a certificate API where CSRs can be sent to and processed and is managed by controller-manager
      - Certificate requests can be expressed in YAML using the CertificateSigningRequest spec and value of request must be base64 encoded

        - **Server components**
            - kube-apiserver certificate
                - Secures access to kube-apiserver
                - Communicates to etcd server and kubelet server on individual nodes
            - etcd server certificate
                - Secures access to etcd which is accessed by kube-apiserver
                - If deploying in HA, must generate certificate for each member of the etcd cluster
            - kubelet server certificate
                - Distributed to workers and controls access to HTTP endpoint on workers which is accessed by kube-apiserver

        - **Client components**
            - kube-scheduler certificate
                - Looks for Pods that need scheduling and communicates with kube-apiserver
            - kube-controller-manager certificate
                - Communicates with kube-apiserver
            - kube-proxy certifocate

- ***Authentication***
    - **Human**
        - Kubernetes does not support human-being identities and authentication out of the box, this must be provided by a third-party integration
        - kube-apiserver handles authentication of human users
            - Static password file
                - Pass as an option to kube-apiserver service using --basic-auth-file=FILE.CSV
                - This file has three columns, username, password, (optionally group)
            - Static token file
                - Pass as an option to kube-apiserver service using --token-auth-file=FILE.CXSV
                - This file has three columns, token, username, (optionally group)
                - Pass the token as an authorization header
            - Certificates
            - Identity services

    - **Non-Human**
        - Kubernetes does support non-human identities and authentication via service accounts
        - Kubernetes control nodes and worker nodes authenticate with each other using certificate

- ***Authorization***
    - **RBAC**
        - **Key Facts**
            - In the definition for an RBAC role the apiGroups property can be left blank if referencing a resource in the core/v1 API
            - Roles are associated with a user using a RoleBinding

- ***Best Practices for Administration***

    - Cluster Manager and Application should be thought of as distinct roles with limited knowledge of each other

- ***Kubernetes Data Encryption Config***
    - **Key Facts**
        - Supports the ability to encrypt data at rest
        - Requires an encryption key it will encrypt and decrypt the data

Scheduling

Monday, August 10, 2020 2:00 PM

- ***Key Facts***
  - If NodeName in Pod spec is not set, then scheduler will determine which node to schedule the Pod to

- ***Manually Scheduling Pod***
  - At creation Pods can be manually scheduled by setting the NodeName property of the Pod spec
  - If already deployed, you must create a Binding object and POST it to the API

- ***Resource Requests***
  - **Key Facts**
    - Request and Limits defaults are configured on per namespace
    - CPU, Memory Disk
    - By default Kubernetes limits each container in the Pod to 1vCPU and 512Mi
    - If Pod tries to use more than it's limit of memory consistently it will be terminated but a Pod is throttled at the CPU limit and cannot use more than that limit
    - Request are the minimum required by the container and the limit is the maximum amount of resources the container in a Pod can consume
    - Set the request and the limits in the Pod spec for each container in the Pod
    - Express CPU by full vCPU (1) or tenths of a CPU (100m)
    - Express Memory in G, Gi (Gibityes - 1024MB), M, Mi (Mebibyte)
  - **Example**
    - Resources:
      requests:
      memory: "1Gi"
      cpu: 1
      limits:
      memory: "2Gi"
      cpu: 2

- ***Taints and Tolerations***
  - **Key Facts**
    - Taints are set on nodes to restrict which Pods can be scheduled to them
    - By default Pods have no tolerations to any taints
    - Does not guarantee a Pod will be placed on a Node if it has a toleration, but it will restrict others without the toleration to the taint from being placed on it
  - **Taint Types**
    - NoSchedule
      - Pods cannot be scheduled
    - PreferNoSchedule
      - Scheduler will try not to schedule on the Node
    - NoExecute
      - New Pods will not be scheduled on the Node and existing Pods will be evicted if they don't tolerate the taint

- ***Annotations***
  - **Key Facts**

        - Used to record other informational metadata for some other purpose potentially by a third party tool
- ***Labels***
    - **Facts**
        - Key value pairs attached to objects such as pods
        - Specify identifying attributes of objects and organize/select subsets of objects
        - Differ from annotations which can't be used to select the objects and are used for another identifying purpose such a 3rd party tool
        - Services use label selectors to specify a set of pods using syntax:
            selector:
            <KEY>:<VALUE>
    - **Label Selectors**
        - Allow a client to identify a set of objects
        - Equality-based (=, !=) and Set-based (in, notin, exists (ex: !accounting, accounting)

    - **Node Selector**
        - Allows you to constrain a pod to a specific node(s)
        - Does not support conditional language like AND or OR
        - Field of the PodSec
            nodeSelector:
            <KEY>:<VALUE>

    - **Node Affinity**
        - **Facts**
            - Limit Pod placement using advanced conditionals
            - Detailed in Pod spec
            - Use in combination with taints and tolerations to dedicate nodes to specific Pods
        - **Types**
            - requiredDuringSchedulingIgnoredDuringExecution
                - Pod will not be scheduled if affinity rules don't match any Nodes
                - Pods running on a Node where affinity rules no longer match will continue to run
            - preferredDuringScheudlingIgnoreDuringExecution
                - Pod will scheduled on Node with matching affinity if possible
                - Pods running on a Node where affinity rules no longer match will continue to run
            - requiredDuringSchedulingRequiredDuringExecution
                - Pod will not be scheduled if affinity rules don't match any Nodes
                - Pods running on a Node where affinity rules no longer match will be evicted
        - **Examples**
            - affinity:
              nodeAffinity:
                requiredDuringSchedulingIgnoredDuringExecution:
                  nodeSelectorTerms:
                  - matchExpressions:
                    - key: <KEY>
                      operator: In/NotIn/Exists/NotExists
                      values:
                      - <VALUE1>
                        <VALUE2> (Processed as OR)
- ***Field Selectors***
    - **Facts**
        - Let you seelct Kubernetes resources based on value of one or more resource fields

        - Example: metadata.name=<SOME_VALUE>

- ***Multiple Schedulers***
    - **Facts**
        - Kubernetes supports custom schedules and multiple schedulers running in the same cluster
        - Add new field of *schedulerName* in Pod spec to designate a custom scheduler
    - **kube-scheduler Facts**
        - kube-scheduler assumes name of default-scheduler so to add a different scheduler it must be named something unique
        - If multiple copies of kube-scheduler running on multiple master nodes in HA configuration --*leader-elect* option must be set to true to ensure that one instance of the scheduler is handling all scheduling

Logging and Monitoring

Thursday, August 6, 2020 7:26 PM

- ***Monitoring the Cluster Components with Metrics Server***
  - Allow you to monitor node, pod, and container runtime
  - One per cluster
  - kubelet runs cAdvisor which retrieves performance metrics from Pods and exposes over API
  - Uses Metrics Server queries node kubelet on all nodes in the cluster for CPU and memory usage
  - kubectl top node -> Memory, CPU from nodes
  - kubectl top pod -> Memory, CPU from Pods

- ***Other 3rd Party Monitoring Solutions***
  - Prometheus
  - Elastic Stack
  - Datadog
  - Dynatrace

- ***Probes***
  - **Facts**
    - Diagnostic performed periodically by the kubelet on a Container
  - **Handlers**
    - ExecAction
      - Specified command run in container and successful if status code 0
    - TCPSocketAction
      - Performs a TCP check against the Pod's IP address on a specified port
    - HTTPGetAction
      - Performs HTTP GET request on Pod's IP address on a specified port and path which is successful if > or = to 200 and <400
  - **Types**
    - livenessProbe
      - Indicatese whether container is running and if probe fails kubelet kills the container and subject to restart policy
      - USE CASE - Use if your application may fail without crashing the container
    - readinessProbe
      - Indicates whether container ready to respond to requests
      - If fails endpoint controller removes Pod's IP address from endpoints of all services that match the pod
      - USE CASE - If you'd like to send traffic only after the probe succeeds
    - startupProbe
      - Indicates whether the application within the container has started
      - All other probes are disabled if startup probe is provided and succeeds
      - USE CASE - Useful for containers that take a long time to start

- ***Cluster Component Logs***
  - Logs from containers stout are written to /var/log/containers

Cluster Maintenance

Monday, September 14, 2020 4:08 PM

- **Taking down a node in the cluster**
  - ***Key Facts***
    - Pods that are part of a replica set that are on a node that goes offline are not recreated until the "pod eviction timeout"
    - Pod Eviction Timeout is defined on controller manager'
    - Draining or just cordoning requires the node to be uncordoned after maintenace is compelte
  - **Methods**
    1. Draining the node gracefully terminates Pod on one node and starts them on another and the node is marked as "cordoned" which is prevents the scheduler from using the node
    2. Cordon the node to prevent new Pods from being scheduled on it but not evicting existing Pod

- **Cluster Upgrades**
  - ***Key Facts***
    - Management components can be different versions but none should be higher than kube-apiserver
    - Controller-manager and kube-scheduler can be one version lower than kube-apiserver while kubelet and kube-proxy can be two versions lower than kube-apiserver
    - kubectl can be one version higher or one version lower than kube-apiserver
    - Recommended approach for upgrades is upgrading one minor version at a time
  - ***Upgrade Options***
    - Cloud providers typically have seamless upgrades
    - Kubeadm simplifies upgrade process
      - Only updates master nodes and worker nodes must be updated separately
    - Deployment from scratch requires manual upgrade of each component
  - ***Upgrade Process***
    - Master nodes are updated first so controller-manager functions are unavailable at that time unless multiple master nodes
    - Nodes should be upgraded on a per node basis to avoid downtime of workloads if upgrading existing nodes
    - Adding new nodes is another option then decomming old nodes

Workload Concepts

Thursday, July 30, 2020 8:52 PM

- ***Nodes***
  - **Key Facts**
    - Run on every node maintaining pods and providing Kubernetes runtime environment
    - Sends heartbeats to node controller to inform controller it is running

- ***Controller***
  - **Key Facts**
    - Control loops that watch state of cluster and make or change requests where needed to achieve desired state
    - Tracks at least one resource type
    - Sometimes carry out actions themselves but more commonly send message to API server to initiate an action

- ***Cluster Add-Ons***
  - **Key Facts**
    - Implement cluster features
    - Cluster DNS add-on should be installed on server
    - Other examples include Web UI, Container Resource Monitoring, and Cluster-level logging

- ***Object***
  - **Facts**
    - Persistent entity in the Kubernetes system which are used to represent the state of the cluster
    - Record of intent which Kubernetes will constantly work to ensure exists (desired state)
    - Defining YAML files require the apiVersion, kind, metadata, and spec
    - Names are unique for a given resource within a namespace
    - Every object created over the entire lifetime of a cluster is assigned a unique UID (a UUID)

  - **Key Terms**
    - spec
      - Set at the creation of the object and provides a description of characteristics (desired state)
    - Status
      - Describes current state of the object and Kubernetes
  - **Management**
    - Imperative commands operate on live objects
      - Example: kubectl create deployment <DEPLOYMENT_NAME> --image <CONTAINER_IMAGE>
    - Imperative object configuration operates on individual files
      - Example: kubectl create -f <FILENAME>.yaml
    - Declarative object configuration operates on directory of files
      - Processes all all objects across a directory or set of directories and the operations to be performed are detected automatically by kubectl
      - Example
        - kubectl diff -f configs/

        - kubectl apply -f configs/
- ***Namespaces***
    - **Facts**
        - Means to divide cluster resources between multiple users via resource quotas
        - Creating service creates corresponding DNS entry in format <SERVICE_NAME>.<NAMESPACE_NAME>.svc.cluster.local
        - Not all objects exists in a namespace such as namespaces, nodes, persistent volumes
    - **Initial Namespaces**
        - default
            - The default namespace for objects with no other namespace
        - kube-system
            - Namespace for objects created by the Kubernetes system
        - kube-public
            - Namespace is created automatically and readable by all users (included non-authenticated users)
            - Reserved for cluster usage
        - kube-node-lease
            - Namespace for lease objects associated with each node to improve performance of node heartbeats as cluster scales

- ***Container Hooks***
    - **Facts**
        - Enable container to be aware of events in management lifecycle and run code implemented by handler when lifecycle hook is executed
    - **Hooks**
        - PostStart
            - Executes immediately after a container is created
        - PreStop
            - Executes immediately before is container is terminated due toa n API request or management event such as liveness probe failure, preemption, resource contention, etc
    - **Hook Handler Types**
        - Exec
            - Executes a specific command inside cgroups and namespaces of the container
        - HTTP
            - Executes an HTTP request against a specific endpoint on the container

- ***Pods***
    - **Facts**
        - Group of one or more containers that share the same storage/network resources and can refer to each other as localhost
        - Co-located and run in shared context
        - Mean to be ephemeral and disposable
        - Pods are never updated directly and instead if a template is changed are deleted and recreated using the new template
        - readinessGates allow you to introduce extra feedback or signals to PodStatus which until the condition is met the pod is not deemed ready
        - Pods that are already running can only have the following items edited:
            - spec.containers[*].image
            - spec.initContainers[*].image
            - spec.activeDeadlineSeconds
            - spec.tolerations

- **initContainers**
  - ***Facts***
    - Containers that run before the normal containers come up and meant to run operations like staging data or pulling code
    - InitContainers are run in order from top down

- **Lifecycle**
  - Pending
  - Running
  - Succeeded - All containers in pod have terminated in success
  - Failed - All containers in Pod have terminated but at least one container terminated in failure
  - Unknown

- **Container States**
  - Waiting
  - Running
  - Terminated

- **Container Restart Policy**
  - Applies to all containers in the pod and restart of containers by kubelet on a single node
  - Capped at 5 minutes
  - **Phases**
    - Always (Default)
    - OnFailure
    - Never

- **Pod Status**
  - PodScheduled
    - Pod has been scheduled to a node
  - ContainersReady
    - All containers in pod are ready
  - Initiatlized
    - All init containers have started successfully
  - Ready
    - Pod is able to serve requests and should be added to the load balancing pools of all matching Service

- **Disruptions**
  - Involuntary
    - Hardware failure of physical VM backing a node
    - Cluster admin deletes VM
    - Cloud provider or hypervisor makes VM disappear
    - Kernal panic
    - Node disappears from cluster to due networking issue
    - Evocation of pod due to out of resources
  - Voluntary
    - Deleting the deployment or other controller that manages the pod
    - Updating a deployment's pod templates causing a restart
    - Directly deleting a pod
    - Draining a node for repair or upgrade

        - Draining a node from a cluster to scale the cluster down
        - Removing a pod from a node to permit something else to fit on that node

- ***Static Pod***
    - **Key Facts**
        - Pods created and managed by the kubelet process on each node without interaction with API Server or Scheduler
        - Definition files are placed in directory and Kubelet regularly checks this directory and will create and update Pods as per the manifest file
        - The directory is designated in the kubelet etcd config file using the --pod-manifest-path option
        - Clusters configured with kubeadm sets the directory by using the --config option of the kubelet.service file which points to a config file where the directory is configured
        - Static Pods are reported to the kube-apiserver because the kubelet creates a stub object to represent the Pod and are appended with "-NODENAME"
    - **Use Case**
        - kubeadmn tool uses Static Pods to create the Pods on the Master nodes for the controller-manager, apiserver, and etcd services.

- ***ReplicaSet***
    - Key Facts
        - ReplicaSet Controller is replacing Replication Controller and is the preferred method moving forward
        - Controller that has a main purpose of maintaining a stable set of replica Pods at any given time
        - Defined with fields that include a selector to identify which pods to acquire, number of replicas, and a Pod template
        - Ownership of a Pod by a ReplicaSet is indicated in the metadata.ownerReferences field of the Pod
        - Deployments manage a ReplicaSet and should be used instead of directly using a ReplicaSet

- ***Deployments***
    - **Key Facts**
        - Controller that changes the actual state of a deployment to meet the desired state
        - By default Deployment ensure at least 75% of the desired number of Pods are up
        - Rollouts can be paused and resumed
        - Deployments can fail due to insufficient quota, readiness probe failures, image pull errors, insufficient permissions, limit ranges, and application runtime misconfig
        - Control rollout with Max Unavailable and Max Surge (how many pods over max can be created)
    - **Features**
        - Rolling updates
        - Rollback updates
    - **Deployment Strategies**
        - Recreate
            - All older Pods deleted before new version of deployment added
        - Rolling Update
            - Default
            - Gracefully removes Pods to ensure application does not become unavailable
    - **Use Cases**

        - Create a deployment to rollout a replica set
        - Declare new state of the Pods
        - Rolback to an earlier deployment revision
        - Scale up a deployment to handle a bigger load
        - Pause the deployment to apply fixes to PodTemplateSpec
        - Use status of deployment as indicator rollout is stuck
        - Clean up old replica sets you don't need anymore

- ***Stateful Sets***
    - **Key Facts**
        - Updates have two strategies
            - OnDelete
                - pods are not replaced when you apply manifest and instead you have to manually delete existing pods before new one will be created
            - RollingUpdate
                - same as deployment
        - Each pod has a unique network identifier <STATEFULSETNAME>-<ORDINAL>.<SERVICE_NAME>.<NAMESPACE>.svc.cluster.local

    - **Use Cases**
        - Care about order of pod deployment
        - Persistent storage
        - Unique and stable network identifier

- ***Daemon Set***
    - **Key Facts**
        - Ensures that all or some nodes run a copy of a Pod
        - Pods for DaemonSets are scheduled by the DaemonController not the K8 Scheduler and will be installed on all nodes in the cluster
        - DefaultScheduler can be used to limit the nodes a pod is deployed to if you use the nodeAffinity field of the DaemonSet spec
    - **Use Cases**
        - Running a cluster storage daemon on every node
        - Running a logs collection daemon on every node
        - Running a node monitoring daemon on every node

- ***Job / Cronjob***
    - **Key Facts**
        - Jobs creates one or more Pods and ensures that the specified number of them terminate successfully and tracks successful completions
        - Cronjobs can be used to create jobs on a repeating schedule

Key Commands

Sunday, September 27, 2020 8:59 PM

- ***Non Kubernetes***
  - **Get a listing of network interfaces**
    - ip link
  - **Get a listing of network interfaces and the IP configuration**
    - ip addr
  - **Get a listing of routes configured on a machine**
    - ip route
  - **Get status of a service**
    - service <SERVICE_NAME> status
  - Get the logs of a kubelet process
    - sudo journalctl -u kubelet
  - Location of kubelet process file for systemd
    - /etc/systemd/system/kubelet.service.d

- **Kubernetes**
  - **Check the logs of a previously running Pod in case of failure**
    - kubectl logs web -f --previous

How-To Key Activities

Sunday, September 27, 2020 9:03 PM

- ***Installing kubeadm (***https://kubernetes.io/docs/setup/production-environment/tools/)
  1. Install a container runtime
  2. Install kubelet, kubeadmin, and kubectl
  3. Initialize the control plane node
     i. kubeadm init
     ii. Follow the instructions to copy the kubeconfig file to your home directory
  4. Install a networking CNI
  5. Get a token to join a worker node
     - kubeadm token list
  6. Add the work node
     - Kubeadm join --token <TOKEN> <CONTROL_PLANE_HOST>:<CONTROL_PLANE_PORT> --discovery-token-ca-cert-hash sha256:<CA_CERT_HASH>

- ***Upgrade Kubernetes Cluster using kubeadm***
  1. Upgrade kubeadm tool
     - apt update
     - apt-cache madison kubeadm
     - apt-get upgrade -y kubeadm=<VERSION>
     - kubeadm version
  2. Upgrade the control plane
     - kubeadm upgrade apply <VERSION>
  3. Upgrade the kubelets (master first)
     - apt-get upgrade -y kubelet=<VERSION>
  4. Restart kubelet service
     - systemctl restart kubelet
  5. Repeat on worker nodes
     i. kubectl drain <NODE_NAME>
     ii. apt-get upgrade -y kubeadm=<VERSION>
     iii. kubeadm upgrade node
     iv. apt-get upgrade -y kubelet=<VERSION>
     v. systemctl restart kubelet
     vi. kubectl uncordon <NODE_NAME>

Object Definitions

Sunday, September 27, 2020 8:38 PM

- ***Networking***
  - **Services**

```
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  type: ClusterIP
  selector:
    app: myApp
  ports:
    - name: http
      protocol: TCP
      port: 80
      targetPort: 9376
```

  - **Ingress Resource (w/ multiple domains and multiple backends)**

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: my-ingress-resource
spec:
  rules:
  - host: wear.my-online-store.com
    http:
      paths:
      - backend:
          serviceName: wear-service
          servicePort: 80
  - host: watch.my-online-store.com
    http:
      paths:
      - path: /watch
        - backend:
            serviceName: watch-service
            servicePort: 80
      - path: /contact
        - backend:
          serviceName: contact-service
          servicePort: 80
```

JQ examples

Wednesday, August 5, 2020 2:56 PM

- Get listing of pod names and ReplicaSet owners
    - Kubectl get po | jq '[.items[] | {name: .metadata.name, .metadata.ownerReerences}]

Application Patterns and Terminology

Wednesday, October 28, 2020 8:32 AM

- Circuit Breaker
  - Pattern to account for failures with a micro-services architecture
  - Main goal is to fail fast and avoid lengthy timeouts and large queues
  - Products
    - Istio - Blackbox approach with sidecar dropped
    - Hystrix / Resilience4J - Whitebox approach where library is integrated into code

- Resources
  - https://www.exoscale.com/syslog/istio-vs-hystrix-circuit-breaker/

3rd Party Products

Wednesday, October 28, 2020 8:34 AM

- Istio
  - Functions as a Service Mesh
  - Runs as a sidecar and functions as a proxy receiving traffic before passed to other container
  - Capabilities
    - Automatic load balancing of HTTP, gRPC, WebSocket, and TCP Traffic
    - Fine-grained control of traffic behavior with routing rules, retries, failovers, and fault injection
    - Pluggable policy layer and configuration API supporting access controls, rate limits, quotas
    - Automatic metrics, logs, traces for all traffic within a cluster which includes cluster ingress and egress
    - Secure service-to-service communication in a cluster with identity-based authentication and authorization

- Hystrix / Reslience4J
  - Latency and fauilt tolerance library provided by Netflix (Resilience4J is successor)
  - Can be used to implement circuit breaker patterns