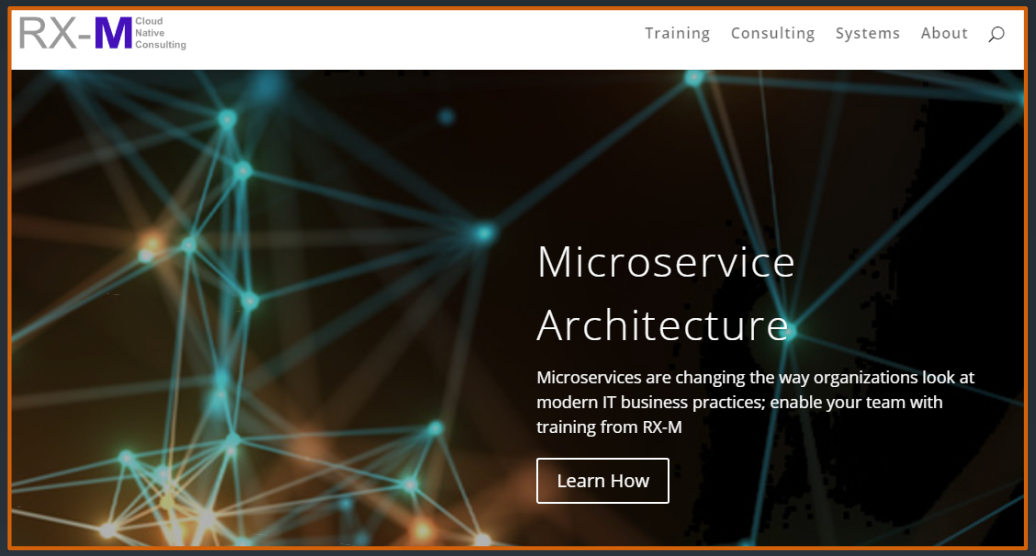
RX-M Cloud Native Consulting

Advanced Kubernetes

Orchestration of Containers in depth

RX-M Cloud Native Advisory, Consulting and Training

- Microservice Oriented
  - Microservices Foundation [3 Day]
  - Building Microservices on AWS [3 Day]
  - Building Microservices with Go [3 Day]
  - Building Microservices with Thrift [2 Day]
  - Building Microservices with gRPC [2 Day]
- Container Packaged
  - Docker Foundation [3 Day]
  - Docker Advanced [2 Day]
  - OCI [2 Day]
  - CNI [2 Day]
  - Containerd [2 Day]
  - Rocket [2 Day]
- Dynamically Managed
  - Cloud Native Container Networking and Cisco ACI [3 Day]
  - Docker Orchestration (Compose/Swarm) [2 Day]
  - Kubernetes Foundation [2 Day]
  - Kubernetes Advanced [3 Day]
  - Mesos Foundation [2 Day]
  - MANTL [2 Day]
  - Nomad [2 Day]



kubernetes CERTIFIED SERVICE PROVIDER

RX-M Cloud Native Consulting

Day 2

4. Services and Networking
5. DNS and Service Discovery
6. API & Security

4. Services and Networking

Objectives

- Explore the nature of Kubernetes services
- Understand the roll of kube-proxy and load balancers
- Describe the IPTables manipulations made by the proxy
- Examine the various Kubernetes Networking options

Service Configs

- kind: Service
- spec:
  - type:
    - ClusterIP – [Default] uses kube-proxy and cluster based virtual IP (available only inside the cluster)
    - NodePort – exposes the service on the same port on each node of the cluster (for external access)
    - LoadBalancer – uses an external load balancer, typically with a public IP and port (AWS ELB, etc.)
    - ExternalName – forwards traffic to the specified external DNS name
  - port:
    - Service listening port
    - Need not be the same as the port of the actual service
  - selector:
    - tells the Service proxy which pods support the service

```
apiVersion: v1
kind: Service
metadata:
  name: testweb
  labels:
    name: testweb
spec:
  type: ClusterIP
  ports:
  - port: 80
    targetPort: 8080
  selector:
    name: testweb
```

```
$ kubectl get endpoints
NAME         ENDPOINTS                AGE
kubernetes   192.168.131.134:6443     5d
testweb      172.17.0.9:80            1h
```

http://kubernetes.io/docs/api-reference/v1/definitions/#_v1_service

Service Config spec

- Spec:
  - ports - The list of ports that are exposed by this service
  - selector - This service will route traffic to pods having labels matching this selector
    - If empty, all pods are selected (!)
  - clusterIP - ClusterIP is usually assigned by the master and is the IP address of the service
    - If specified, it will be allocated to the service if it is unused or else creation of the service will fail
    - Valid values are None, empty string (""), or a valid IP address
    - Cannot be updated
  - type - Type of exposed service
    - ClusterIP (default)
    - NodePort
    - LoadBalancer
  - externalIPs - externalIPs is a list of IP addresses for which nodes in the cluster will also accept traffic for this service
    - These IPs are not managed by Kubernetes
    - The user is responsible for ensuring that traffic arrives at a node with this IP
    - A common example is external load-balancers that are not part of the Kubernetes system
  - sessionAffinity - Supports "ClientIP" and "None“
    - Used to maintain session affinity
    - Defaults to None
  - loadBalancerIP - Only applies to Service Type: LoadBalancer
    - LoadBalancer will get created with the IP specified in this field
    - This feature depends on whether the underlying cloud-provider supports specifying the loadBalancerIP when a load balancer is created
    - This field will be ignored if the cloud-provider does not support the feature

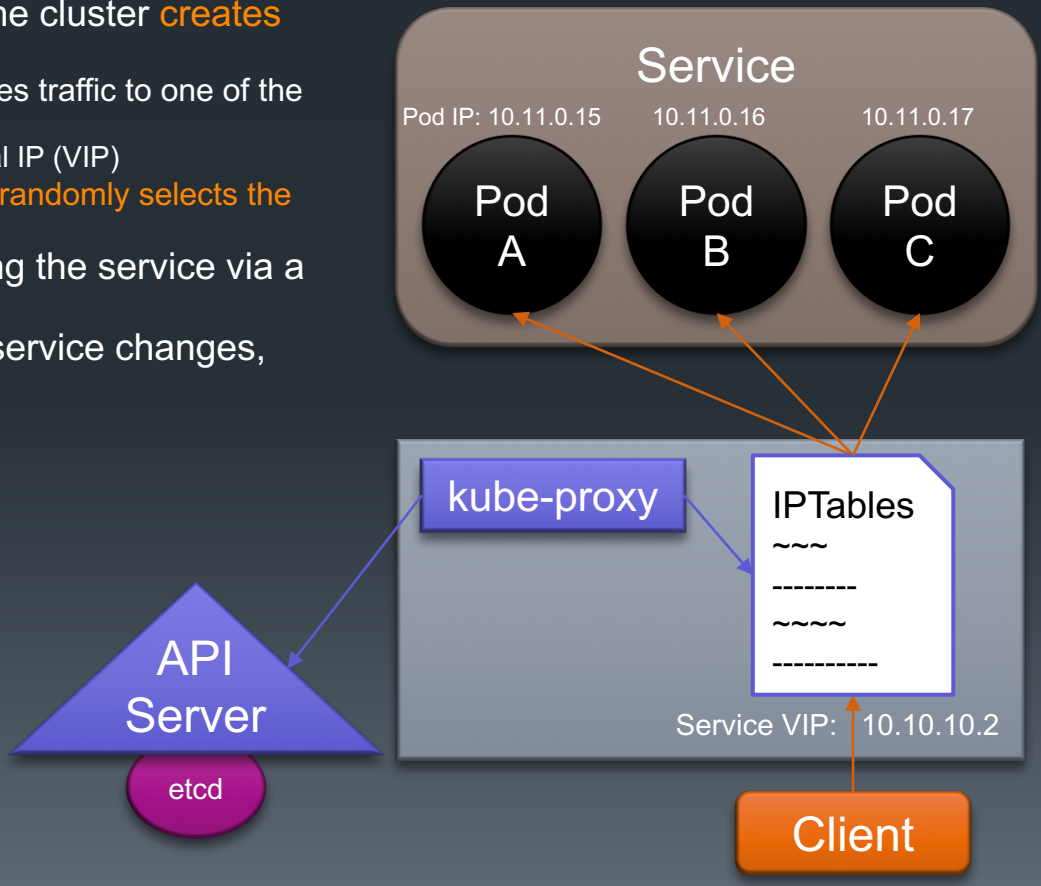
Node Port Details

- The Kubernetes master allocates a port from a flag-configured range
  - Default: 30000-32767
- Each Node will proxy that port (the same port number on every Node) into your Service
  - That port will be reported in your Service's spec.ports[*].nodePort field
- If you want a specific port number, you can specify a value in the nodePort field
  - The system will allocate you that port or else the API transaction will fail
  - The value you specify must be in the configured range for node ports
  - The port may not conflict with existing ports
- This gives developers the freedom to set up their own load balancers, to configure cloud environments that are not fully supported by Kubernetes
- Node Port Services also have all of the features of Cluster IP
  - Visible as both:
    - NodeIP: spec.ports[*].nodePort
    - ClusterIP: spec.clusterIp:spec.ports[*].port

```
user@ubuntu:~/svc$ cat nodeport.yaml

apiVersion: v1
kind: Service
metadata:
   name: testweb-np
spec:
   type: NodePort
   ports:
     - port: 80
       protocol: TCP
  selector:
     run: testweb

user@ubuntu:~/svc$ kubectl create -f nodeport.yaml

user@ubuntu:~/svc$ kubectl get service
NAME         CLUSTER-IP       EXTERNAL-IP   PORT(S)         AGE
kubernetes   10.96.0.1        <none>        443/TCP         1d
testweb-np   10.111.101.166   <nodes>       80:30143/TCP    32s

user@ubuntu:~/svc$ kubectl describe service testweb-np
Name:                   testweb-np
Namespace:              default
Labels:                 <none>
Annotations:            <none>
Selector:               run=testweb
Type:                   NodePort
IP:                     10.111.101.166
Port:                   <unset>      80/TCP
NodePort:               <unset>      30143/TCP
Endpoints:              10.32.0.4:80,10.32.0.5:80,10.32.0.6:80
Session Affinity:       None
Events:                 <none>
```

Kube-Proxy

- Microservices
  - Most sophisticated K8s applications are microservice oriented
    - Networks of small services
  - Each logical service is implemented by a set of pod replicas
  - Services frequently need to communicate with each other
- Routing Mesh
  - A service client needs a way to connect to one of the pods implementing the service
  - The Kube-Proxy running on each node in the cluster creates iptables rules for every service
    - These rules produce a virtual endpoint that routes traffic to one of the service's target pods
      - This end point is typically a ClusterIP, aka Virtual IP (VIP)
    - Load balancing is performed by iptables, which randomly selects the target pod to forward to
  - Kube-Proxy identifies the pods implementing the service via a selector
  - All Kube-Proxies watch the API Server for service changes, updating their iptables as needed
- Service Discovery
  - Clients need a way to discover service IPs
  - Kubernetes has an integrated DNS service to help with service discovery
    - kube-dns acts like a normal DNS service but knows the addresses of all services
    - DNS A records (address records) provide resolution of domain names to IP addresses
    - DNS SRV records (service records) can be used to provide IP addresses and ports



```
user@ubuntu:~/svc$ sudo iptables -nL -t nat
Chain PREROUTING (policy ACCEPT)
target                          prot opt source             destination
KUBE-SERVICES                   all  --  0.0.0.0/0          0.0.0.0/0       /* kubernetes service portals */
DOCKER                          all  --  0.0.0.0/0          0.0.0.0/0       ADDRTYPE match dst-type LOCAL

Chain INPUT (policy ACCEPT)
target                          prot opt source             destination

Chain OUTPUT (policy ACCEPT)
target                          prot opt source             destination
KUBE-SERVICES                   all  --  0.0.0.0/0          0.0.0.0/0       /* kubernetes service portals */
DOCKER                          all  --  0.0.0.0/0          !127.0.0.0/8    ADDRTYPE match dst-type LOCAL

Chain KUBE-MARK-DROP (0 references)
target                          prot opt source              destination
MARK                            all  --  0.0.0.0/0           0.0.0.0/0            MARK or 0x8000

Chain KUBE-MARK-MASQ (4 references)
target                          prot opt source              destination
MARK                            all  --  0.0.0.0/0           0.0.0.0/0            MARK or 0x4000

Chain KUBE-NODEPORTS (1 references)
target                          prot opt source              destination

Chain POSTROUTING (policy ACCEPT)
target                          prot opt source             destination
KUBE-POSTROUTING                all  --  0.0.0.0/0          0.0.0.0/0       /* kubernetes postrouting rules */
MASQUERADE                      all  --  172.17.0.0/16      0.0.0.0/0

Chain KUBE-POSTROUTING (1 references)
target                          prot opt source              destination
MASQUERADE                      all  --  0.0.0.0/0           0.0.0.0/0       /* kubernetes service traffic requiring SNAT */ mark match 0x4000/0x4000

Chain KUBE-SERVICES (2 references)
target                          prot opt source         destination
KUBE-SVC-XGLOHA7QRQ3V22RZ       tcp  --  0.0.0.0/0      10.106.125.172  /* kube-system/kubernetes-dashboard: cluster IP */ tcp dpt:80
KUBE-SVC-NPX46M4PTMTKRN6Y       tcp  --  0.0.0.0/0      10.96.0.1       /* default/kubernetes:https cluster IP */ tcp dpt:443
KUBE-SVC-TCOU7JCQXEZGVUNU       udp  --  0.0.0.0/0      10.96.0.10      /* kube-system/kube-dns:dns cluster IP */ udp dpt:53
KUBE-SVC-ERIFXISQEP7F7OF4       tcp  --  0.0.0.0/0      10.96.0.10      /* kube-system/kube-dns:dns-tcp cluster IP */ tcp dpt:53
KUBE-NODEPORTS                  all  --  0.0.0.0/0      0.0.0.0/0       /* kubernetes service nodeports; NOTE: this must be the last rule in this chain */
ADDRTYPE match dst-type LOCAL

Chain KUBE-SVC-XGLOHA7QRQ3V22RZ (1 references)
target                          prot opt source              destination
KUBE-SEP-WIZ23EZGADKT3IPN       all  --  0.0.0.0/0           0.0.0.0/0            /* kube-system/kubernetes-dashboard: */

...

Chain KUBE-SEP-WIZ23EZGADKT3IPN (1 references)
target                          prot opt source              destination
KUBE-MARK-MASQ                  all  --  10.32.0.3           0.0.0.0/0            /* kube-system/kubernetes-dashboard: */
DNAT                            tcp  --  0.0.0.0/0           0.0.0.0/0            /* kube-system/kubernetes-dashboard: */ tcp to:10.32.0.3:9090
```

KubeProxy and iptables

ClusterIP Service

```
user@ubuntu:~/svc$ kubectl run testweb --image=nginx
deployment "testweb" created
user@ubuntu:~/svc$ vim svc.yaml
apiVersion: v1
kind: Service
metadata:
  name: testweb
  labels:
    svc: testweb
spec:
  type: ClusterIP
  ports:
  - port: 80
  selector:
    run: testweb
user@ubuntu:~/svc$ kubectl create -f svc.yaml
service "testweb" created
user@ubuntu:~/svc$ kubectl scale deployment --replicas=3 testweb
deployment "testweb" scaled
user@ubuntu:~/svc$ kubectl get deploy,po,rs
NAME             DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
deploy/testweb   1         1         1            1           5m

NAME                           READY     STATUS    RESTARTS   AGE
po/testweb-3248343053-d74h5    1/1       Running   0          4m
po/testweb-3248343053-ftfz8    1/1       Running   0          4m
po/testweb-3248343053-m8htj    1/1       Running   0          5m

NAME                     DESIRED   CURRENT   READY     AGE
rs/testweb-3248343053    1         1         1         5m

user@ubuntu:~/svc$ kubectl get service
NAME         CLUSTER_IP    EXTERNAL_IP   PORT(S)   AGE
kubernetes   10.96.0.1     <none>        443/TCP   1d
testweb      10.96.0.149   <none>        80/TCP    5m
user@ubuntu:~/svc$ curl -s http://10.96.0.149 | head -4
...
<title>Welcome to nginx!</title>
user@ubuntu:~/svc$ curl -s http://10.32.0.4 | head -4
...
<title>Welcome to nginx!</title>
```

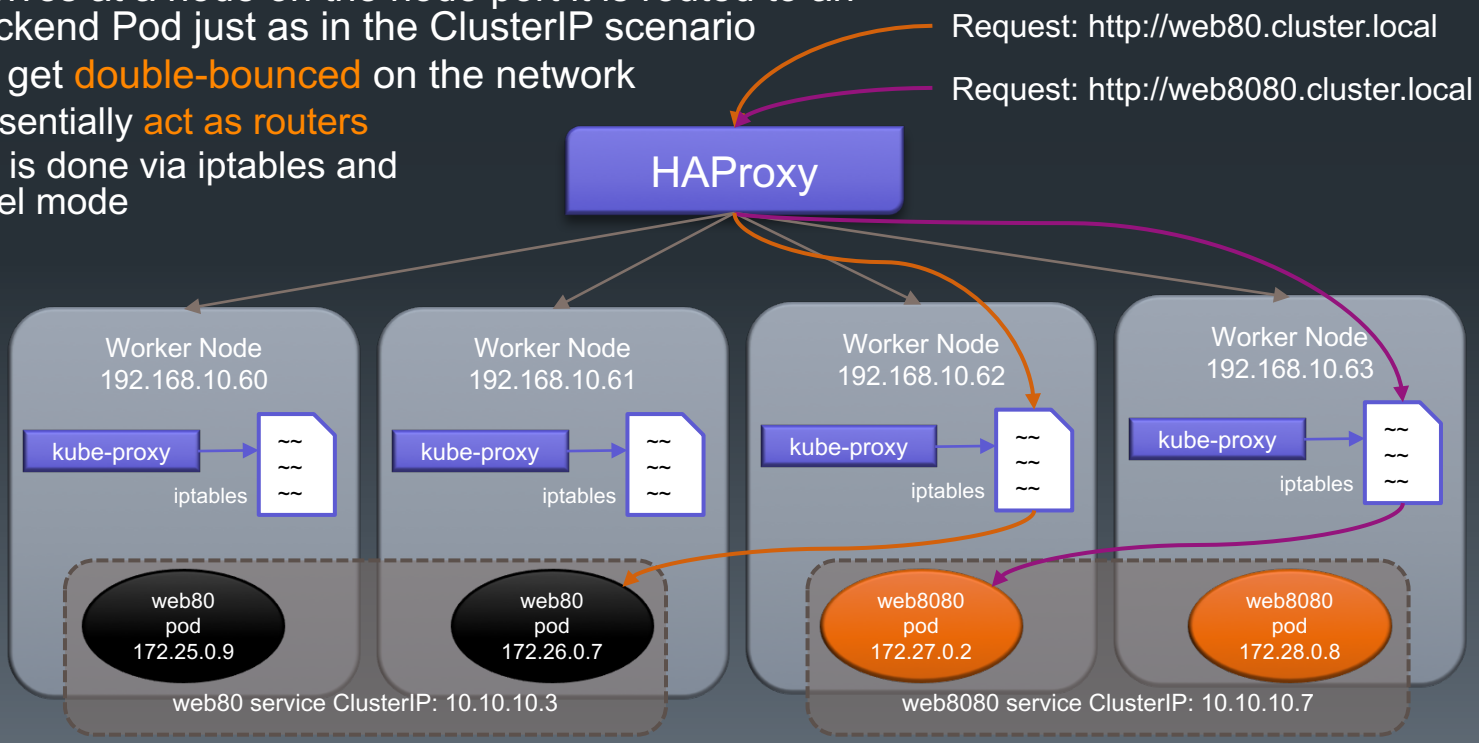
```
user@ubuntu:~/svc$ kubectl describe service testweb

Name:                   testweb
Namespace:              default
Labels:                 svc=testweb
Selector:               run=testweb
Type:                   ClusterIP
IP:                     10.96.0.149
Port:                   <unset>     80/TCP
Endpoints:              10.32.0.4:80,10.32.0.5:80,10.32.0.6:80
Session Affinity:       None
Events:                 <none>
```

ClusterIP Service Nat

```
user@ubuntu:~/svc$ sudo iptables -nL -t nat
Chain PREROUTING (policy ACCEPT)
target          prot opt source               destination
KUBE-SERVICES   all  --  0.0.0.0/0            0.0.0.0/0       /* kubernetes service portals */
DOCKER          all  --  0.0.0.0/0            0.0.0.0/0       ADDRTYPE match dst-type LOCAL

Chain INPUT (policy ACCEPT)
target          prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target          prot opt source               destination
KUBE-SERVICES   all  --  0.0.0.0/0            0.0.0.0/0       /* kubernetes service portals */
DOCKER          all  --  0.0.0.0/0            !127.0.0.0/8    ADDRTYPE match dst-type LOCAL

Chain POSTROUTING (policy ACCEPT)
target          prot opt source               destination
KUBE-POSTROUTING all  --  0.0.0.0/0            0.0.0.0/0       /* kubernetes postrouting rules */
MASQUERADE      all  --  172.17.0.0/16        0.0.0.0/0

Chain KUBE-POSTROUTING (1 references)
target          prot opt source               destination
MASQUERADE      all  --  0.0.0.0/0            0.0.0.0/0       /* kubernetes service traffic requiring SNAT */ mark match 0x4000/0x4000

Chain KUBE-SERVICES (2 references)
target                     prot opt source         destination
KUBE-SVC-XGLOHA7QRQ3V22RZ  tcp  --  0.0.0.0/0      10.106.125.172  /* kube-system/kubernetes-dashboard: cluster IP */ tcp dpt:80
KUBE-SVC-NPX46M4PTMTKRN6Y  tcp  --  0.0.0.0/0      10.96.0.1       /* default/kubernetes:https cluster IP */ tcp dpt:443
KUBE-SVC-TCOU7JCQXEZGVUNU  udp  --  0.0.0.0/0      10.96.0.10      /* kube-system/kube-dns:dns cluster IP */ udp dpt:53
KUBE-SVC-ERIFXISQEP7F7OF4  tcp  --  0.0.0.0/0      10.96.0.10      /* kube-system/kube-dns:dns-tcp cluster IP */ tcp dpt:53
KUBE-SVC-EG66NKXDU7X2QZTA  tcp  --  0.0.0.0/0      10.96.0.149     /* default/testweb: cluster IP */ tcp dpt:80
KUBE-NODEPORTS             all  --  0.0.0.0/0      0.0.0.0/0       /* kubernetes service nodeports; NOTE: this must be the last rule in this chain */
ADDRTYPE match dst-type LOCAL

Chain KUBE-SVC-EG66NKXDU7X2QZTA (1 references)
target                     prot opt source         destination
KUBE-SEP-7KX2LIG5LIB5UVVO  all  --  0.0.0.0/0      0.0.0.0/0       /* default/testweb: */ statistic mode random probability 0.33332999982
KUBE-SEP-TSJSFJG2GUXROSTI  all  --  0.0.0.0/0      0.0.0.0/0       /* default/testweb: */ statistic mode random probability 0.50000000000
KUBE-SEP-RY3VYSR3AYX2BBO2  all  --  0.0.0.0/0      0.0.0.0/0       /* default/testweb: */

Chain KUBE-SEP-7KX2LIG5LIB5UVVO (1 references)
target          prot opt source               destination
KUBE-MARK-MASQ  all  --  10.32.0.4            0.0.0.0/0       /* default/testweb: */
DNAT            tcp  --  0.0.0.0/0            0.0.0.0/0       /* default/testweb: */ tcp to:10.32.0.4:80

Chain KUBE-SEP-TSJSFJG2GUXROSTI (1 references)
target          prot opt source               destination
KUBE-MARK-MASQ  all  --  10.32.0.5            0.0.0.0/0       /* default/testweb: */
DNAT            tcp  --  0.0.0.0/0            0.0.0.0/0       /* default/testweb: */ tcp to:10.32.0.5:80

Chain KUBE-SEP-RY3VYSR3AYX2BBO2 (1 references)
target          prot opt source               destination
KUBE-MARK-MASQ  all  --  10.32.0.6            0.0.0.0/0       /* default/testweb: */
DNAT            tcp  --  0.0.0.0/0            0.0.0.0/0       /* default/testweb: */ tcp to:10.32.0.6:80
```

External Load Balancers

- Access to a pod or service from within the cluster is clean and simple, using the Pod IP or the kube-proxy established service ClusterIP (VIP)
- Accessing a pod from outside the cluster requires an alternate approach
- LoadBalancer services wire up cloud load balancer
  - GCE's ForwardingRules
  - AWS's ELB
- NodePort services expose the service on a single port on every node's external interface
  - When traffic arrives at a node on the node port it is routed to an appropriate backend Pod just as in the ClusterIP scenario
  - Most traffic will get double-bounced on the network
    - The nodes essentially act as routers
    - All forwarding is done via iptables and purely in kernel mode



Ingress (beta)

- Collections of rules that allow inbound connections to reach cluster services
- An Ingress resource supports:
  - Exposing services:
    - Via custom URLs
      - ex: service A at the URL /serviceA and service B at the URL /serviceB
    - Via multiple host names
      - ex: groupa.example.com for one group of services and groupb.example.com for another group
  - Configuring SSL termination for each exposed host name
- Ingress spec contains a list of rules matched against all incoming requests
  - Currently the Ingress resource only supports http rules, each rule contains:
    - Host
    - List of paths
    - Backends associated with the paths
      - Includes a service:port combination
- Ingress controller is required to satisfy an Ingress
  - Simply creating the Ingress resource will have no effect
  - Using annotation it is possible to run multiple Ingress controllers at the same time

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: nginx-ingress
  annotations:
    kubernetes.io/ingress.class: "nginx"
spec:
  rules:
  - host: host.example.com
    http:
      paths:
      - path: /somepath
        backend:
          serviceName: somesvc
          servicePort: 80
```

Ingress Controllers

- Monitor Ingress resources via the Kubernetes API and updates the configuration of a load balancer in case of any changes
  - Acts as reverse proxy for the Ingress rules found at api-server's /ingresses endpoint
- Types:
  - GLBC – is a GCE L7 load balancer controller that manages external loadbalancers configured through the Kubernetes Ingress API
  - nginx – deployed as a Deployment/Pod & uses a ConfigMap to store its configuration
  - Non-official: Træfik, others…

https://github.com/kubernetes/ingress/tree/master/examples contains a catalog of examples on how to run, configure and scale Ingress

```yaml
apiVersion: apps/v1beta1
kind: Deployment
metadata:
  name: nginx-ingress-dep
  namespace: kube-system
  labels:
    app: nginx-ingress
spec:
  replicas: 1
  template:
    metadata:
      labels:
        app: nginx-ingress
    spec:
      containers:
      - image: gcr.io/google_containers/nginx-ingress-controller:0.8.3
        name: nginx-ingress
        ports:
        - containerPort: 80
          hostPort: 80
        - containerPort: 443
          hostPort: 443
```

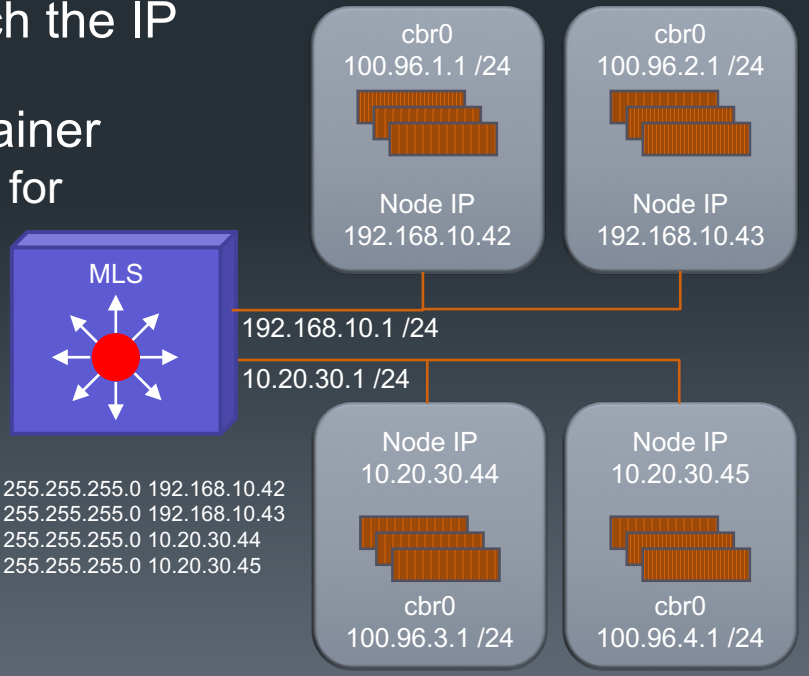
```yaml
apiVersion: v1
kind: Service
metadata:
  name: nginx-ingress-svc
  namespace: kube-system
  labels:
    app: nginx-ingress
spec:
  ports:
  - name: http
    port: 80
    targetPort: 80
  - hame: https
    port: 443
    targetPort: 443
  selector:
    app: nginx-ingress
```

Networking Options

- No K8s networking
  - `--network-plugin=noop`
  - Host based or Docker supported SDN
- K8s kubenet
  - `--network-plugin=kubenet`
  - Only works on Linux
  - Like docker0 bridge but uses a linux bridge called cbr0
- K8s CNI plugin
  - `--network-plugin=cni`
  - Pod network setup and teardown handled by a CNI plugin
  - Uses `--cni-bin-dir` and `--cni-conf-dir`

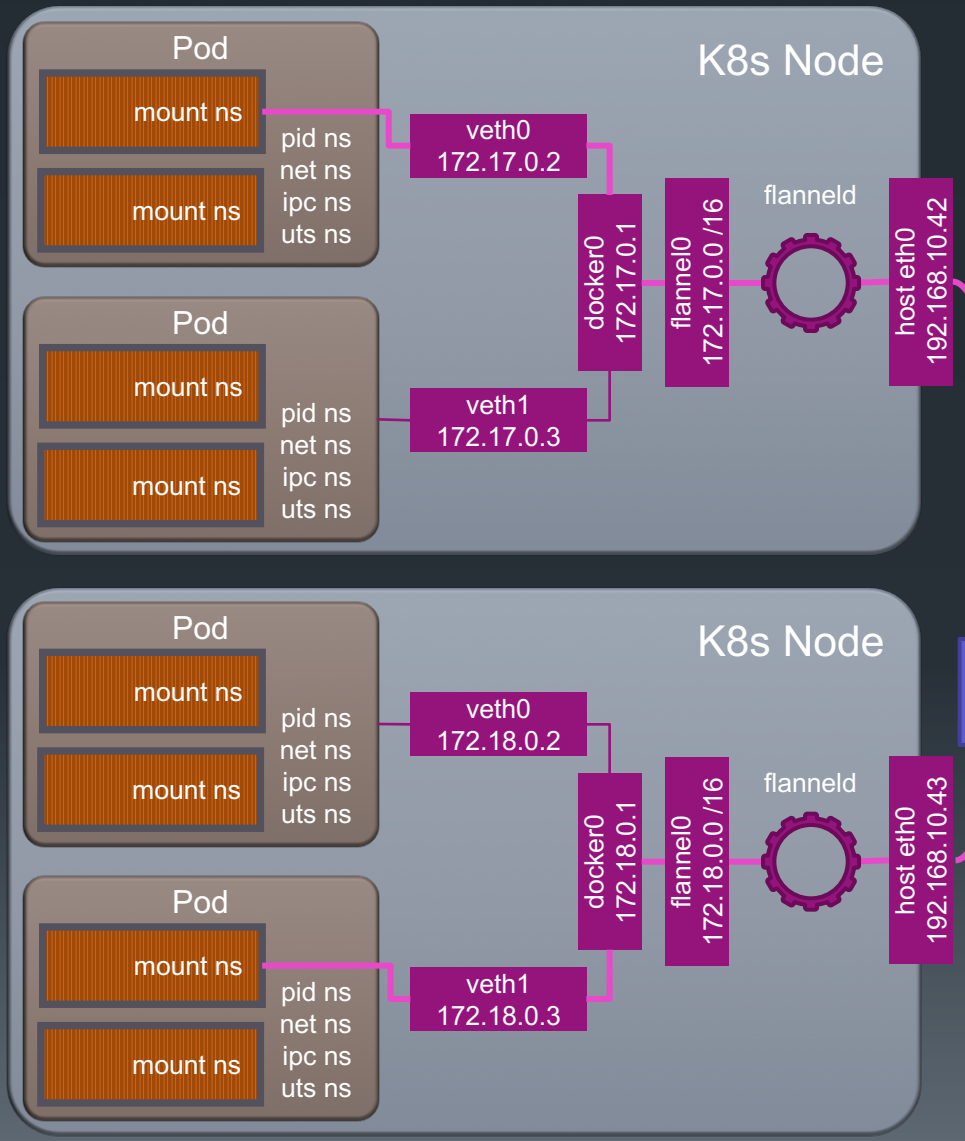
Node Side Networking

- Each pod has its own IP address
  - Host IPs are problematic as container counts grow [too few IPs]
    - Ports must be used to expose services on containers and allow external communication
    - The complexity of running multiple services that may or may not know about each other (and their custom ports), and managing the port space becomes challenging
  - Container IPs can be overkill [too many IPs]
  - Pod IPs represent a happy medium [just right IPs]
- Kubernetes does not allow the use of Network Address Translation (NAT) for container-to-container or for container-to-node (minion) traffic
- The internal container IP address must match the IP address that is used to communicate with it
- Each pod receives a pod infrastructure container
  - This container reserves the network resources for the application containers that will be started later (even if they fail)
- Kubernetes can be configured to use overlay networks if desired but often the service model suffices and keeps things simple




```
ip route 100.96.1.1 255.255.255.0 192.168.10.42
ip route 100.96.2.1 255.255.255.0 192.168.10.43
ip route 100.96.3.1 255.255.255.0 10.20.30.44
ip route 100.96.4.1 255.255.255.0 10.20.30.45
```

SDN Networking Options



- Calico – uses BGP to enable real container IPs
- Cilium – generates Berkeley Packet Filter (BPF) programs for container to provide networking, security, and loadbalancing
- Contiv – supports routed (L3/BGP), bridged (L2/VLAN), overlay (VXLAN) and Cisco ACI models
- Flannel – designed to automatically assign subnets to Docker Daemon bridges and tunnel between nodes via UDP (VXLAN)
  - Canal is a solution that combines Flannel with Calico policy-based network management features
- OpenVSwitch – somewhat more mature but complicated way to build an overlay network endorsed by several key networking vendors
- Romana – layer 3 networking solution for pod networks
- Weave – provides networking (VXLAN) and network policy; does not require an external database

- CNI-Genie – addon enables orchestrators to connect to choice of CNI plugins (Calico, Canal, Romana, Weave) configured on a Node
  - Allows for multiple CNI plugins being available to kubelet simultaneously

Network Policies

- Specification of how groups of pods are allowed to communicate with each other and other network endpoints
  - Use labels to select pods and define rules which specify what traffic is allowed to the selected pods
  - A pod will reject any connections that are not allowed by a NetworkPolicy
- Implemented by a network plugin
  - Must use a provider which supports NetworkPolicy
  - Simply creating the resource without a controller to implement it will have no effect
- Spec
  - `podSelector` – currently only supports defining ingress rules
    - Defines the destination pods for the policy
      - e.g. pods with the label `role=db`
  - `ingress` – whitelist allows traffic which matches **both** the `from` and `ports` sections
  - `ipBlock` – describes a particular CIDR that is allowed
    - `except` – slice of CIDRs that should not be included within an IP Block
- Create a "default" *isolation policy* for a Namespace by creating a NetworkPolicy that selects all pods but does not allow any traffic
  - By default, pods accept traffic from any source

providers that support NetworkPolicy:
Calico
Cilium
Kube-router addon
Romana
Weave Net

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: mynetpolicy
  namespace: default
spec:
  podSelector:
    matchLabels:
      role: db
  ingress:
  - from:
    - ipBlock:
        cidr: 172.17.0.0/16
        except:
        - 172.17.1.0/24
    - namespaceSelector:
        matchLabels:
          project: myproject
    - podSelector:
        matchLabels:
          role: frontend
    ports:
    - protocol: TCP
      port: 6379
```

CNI integration

- Kubernetes kubelets create Pods, which requires the kubelet to configure pod networking
  - The CNI plugin is selected by passing Kubelet the `--network-plugin=cni` command-line option
  - Kubelet reads a file from `--cni-conf-dir` (default /etc/cni/net.d) and uses the CNI configuration from that file to set up each pod's network
    - The CNI configuration file must match the CNI specification
  - Any required CNI plugins referenced by the configuration must be present in `--cni-bin-dir` (default /opt/cni/bin)
    - Kubernetes requires the standard CNI lo plugin, at minimum version 0.2.0
  - HostPort does not presently work with CNI networking plugins
    - Pod hostPort attributes are ignored presently when using CNI
    - https://github.com/kubernetes/kubernetes/issues/31307

CNI

- Container Network Interface - https://github.com/containernetworking/cni
- Open Source project consisting of:
  - A specification
  - Libraries in Go for writing plugins to configure network interfaces in Linux containers
  - Sample supported plugins
- CNI concerns itself only with
  - Network connectivity of containers
  - Removing allocated resources when the container is deleted
- A CNCF project as of Spring 2017
- Diverges from the more complex CNM (Container Networking Model) used by Docker
  - Likely containerd will add CNI support making CNM defunct

Container runtimes

- rkt - container engine
- Kurma - container runtime
- Kubernetes - a system to simplify container operations
- OpenShift - Kubernetes with additional enterprise features
- Cloud Foundry - a platform for cloud applications
- Mesos - a distributed systems kernel

3rd party plugins

- Project Calico - a layer 3 virtual network
- Weave - a multi-host Docker network
- Contiv Networking - policy networking for various use cases
- SR-IOV
- Cilium - BPF & XDP for containers
- Infoblox - enterprise IP address management for containers
- Multus - a Multi plugin
- Romana - Layer 3 CNI plugin supporting network policy for Kubernetes
- CNI-Genie - generic CNI network plugin
- Nuage CNI - Nuage Networks SDN plugin for network policy kubernetes support
- Silk - a CNI plugin designed for Cloud Foundry
- Linen - a CNI plugin designed for overlay networks with Open vSwitch and fit in SDN

CNI Operations

- Network Add and Delete methods with supporting JSON
  - `cniVersion` (string) - ensures backwards/forwards compatibility
  - `name` (string) - “network” name
  - `type` (string) - plugin name
  - `args` (dictionary) - optional
  - `ipam` (dictionary) - specify the IPAM plugin, if any
  - `dns` - static DNS settings
    - Nameservers
    - Domain
    - Search
    - options

```
{
  "cniVersion": "0.3.1",
  "name": "dbnet",
  "type": "bridge",
  "bridge": "cni0",
  "ipam": {
    "type": "host-local",
    "subnet": "10.1.0.0/16",
    "gateway": "10.1.0.1"
  },
  "dns": {
    "nameservers": [ "10.1.0.1" ]
  }
}
```

https://github.com/containernetworking/cni/blob/master/SPEC.md#network-configuration

Summary

- Kubernetes implements a simple Linux native networking model using standards driven technologies
  - Virtual IPs
  - IPTables
  - DNS
  - Etc.
- Third parties provide a range of Container networking functions that (properly integrated) the kube-proxy can work with

Lab 4

- Services and Networking

5. DNS and Service Discovery

Objectives

- Understand the benefits of service discovery
- Learn how DNS is used in K8s
- Learn how to use kube-dns
- Explain the addon system used in some versions of Kubernetes

Add-ons

- Cluster add-ons are Services and Deployments that extend Kubernetes function
  - Common but optional parts of Kubernetes clusters
- Addons are usually created in the kube-system namespace

Networking and Network Policy

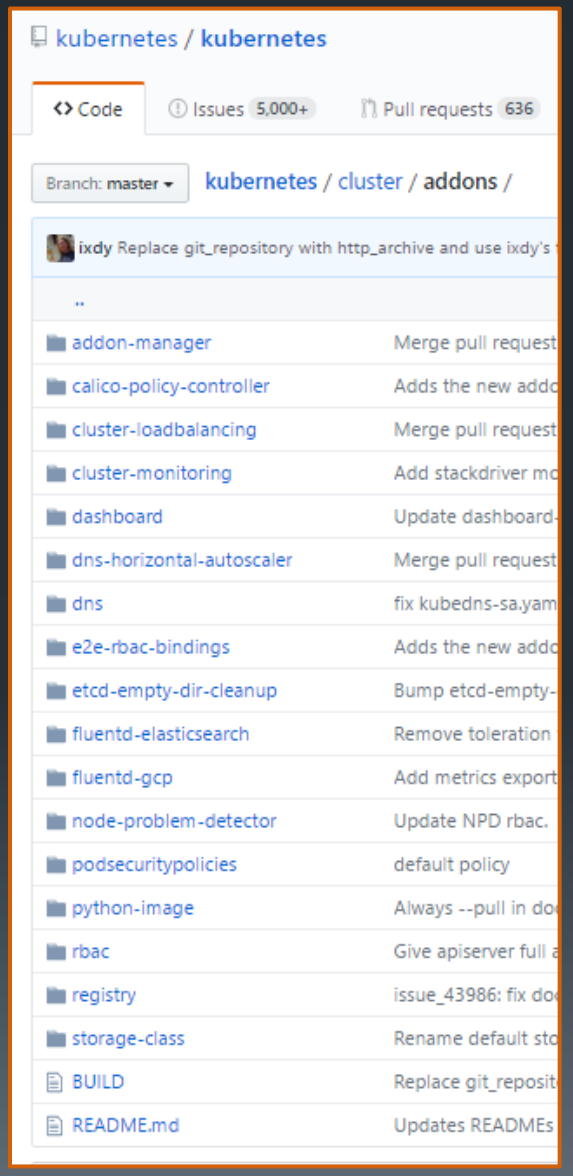
- Calico is a secure L3 networking and network policy provider.
- Canal unites Flannel and Calico, providing networking and network policy.
- Cilium is a L3 network and network policy plugin that can enforce HTTP/API/L7 policies transparently. Both routing and overlay/encapsulation mode are supported.
- Contiv provides configurable networking (native L3 using BGP, overlay using vxlan, classic L2, and Cisco-SDN/ACI) for various use cases and a rich policy framework. Contiv project is fully open sourced. The installer provides both kubeadm and non-kubeadm based installation options.
- Flannel is an overlay network provider that can be used with Kubernetes.
- Romana is a Layer 3 networking solution for pod networks that also supports the NetworkPolicy API. Kubeadm add-on installation details available here.
- Weave Net provides networking and network policy, will carry on working on both sides of a network partition, and does not require an external database.

Visualization & Control

- Dashboard is a dashboard web interface for Kubernetes.
- Weave Scope is a tool for graphically visualizing your containers, pods, services etc. Use it in conjunction with a Weave Cloud account or host the UI yourself.

Legacy Add-ons

- Early versions of Kubernetes used a cluster add-on manager
  - A simple bash script applying configs found in
    - /etc/kubernetes/addons
  - In the master branch at: kubernetes/cluster/addons
  - This feature is now little used and may go away
- Several important addons are still housed here for now
  - DNS in particular
- These are still easy to use
  - Just run the provided configs
  - Some fields may need to be customized



Legacy Addon Classes

- There are two classes of addons with given template files
  - Addons with label `addonmanager.kubernetes.io/mode=Reconcile` will be periodically reconciled
    - Direct manipulation to these addons through apiserver is discouraged because addon-manager will bring them back to the original state
      - Addon will be re-created if it is deleted
      - Addon will be reconfigured to the state given by the supplied fields in the template file periodically
      - Addon will be deleted when its manifest file is deleted
  - Addons with label `addonmanager.kubernetes.io/mode=EnsureExists` will be checked for existence only
    - Users can edit these addons
    - Addon will only be created/re-created with the given template file when there is no instance of the resource with that name
    - Addon will not be deleted when the manifest file is deleted

Service Discovery

- The automatic detection of services on a computer network
- Service discovery protocols (SDP) are network protocols that accomplish service discovery
- Service discovery requires a common language to allow software agents to make use of one another's services without the need for continuous user intervention
- There are many service discovery protocols, including:
  - Bluetooth Service Discovery Protocol (SDP)
  - DNS Service Discovery (DNS-SD), a component of Zero Configuration Networking
  - Dynamic Host Configuration Protocol (DHCP)
  - Internet Storage Name Service (iSNS)
  - Jini for Java objects.
  - Service Location Protocol (SLP)
  - Session Announcement Protocol (SAP) used to discover RTP sessions
  - Simple Service Discovery Protocol (SSDP) a component of Universal Plug and Play (UPnP)
  - Universal Description Discovery and Integration (UDDI) for web services
  - Web Proxy Autodiscovery Protocol (WPAD)
  - WS-Discovery (Web Services Dynamic Discovery)
  - XMPP Service Discovery (XEP-0030)
  - XRDS (eXtensible Resource Descriptor Sequence) used by XRI, OpenID, OAuth, etc.

Kubedns

- Kubernetes DNS is the Kubernetes Cluster service discovery system
- Pods in a K8s cluster are typically configured to use the KubeDNS Service to resolve DNS names
- KubeDNS can resolve:
  - Kubernetes Service Names to VIPs
  - Pod names to Pod IPs
  - Perform reverse lookups and more

Installation DNS

- Kubernetes 1.3
  - DNS became a built-in service launched automatically by most installers using the addon manager cluster add-on
- Kubernetes 1.4
  - The kubeadm tool automatically installs kubedns
- Kubernetes 1.5
  - Add-on solution fades a bit and some deploy DNS with a simple config
- A DNS Pod and Service scheduled on the cluster
  - kubelets will be configured to tell individual containers to use the DNS Service's ClusterIP to resolve DNS names
  - `$ kubelet --kubeconfig=nodea.kubeconfig --require-kubeconfig --allow-privileged=true --cluster-dns=10.0.0.10 --cluster_domain=cluster.local`
  - The kubelet configures the `/etc/resolv.conf` in containers (through Docker) using these settings

```
# cat /etc/resolv.conf
search default.svc.cluster.local.
nameserver 10.0.0.10 10.96.0.10
options ndots:5
```

How it works

- The running Kubernetes DNS pod holds 3 containers:
  - kubedns – watcher
  - dnsmasq – provides network infrastructure for networks:
    - DNS
    - DHCP
    - Router advertisement
    - Network boot
  - healthz – health check endpoint
- The kubedns process watches the Kubernetes master for changes in Services and Endpoints, and maintains in-memory lookup structures to service DNS requests
- The dnsmasq container adds DNS caching to improve performance
- The healthz container provides a single health check endpoint while performing dual healthchecks (for dnsmasq and kubedns)

DNS Records

- The Kubernetes cluster DNS server supports:
  - Forward lookups (A records)
    - "Normal": my-svc.my-namespace.svc.cluster.local
      - Returns clusterIP
    - "Headless": my-svc.my-namespace.svc.cluster.local
      - Returns array of IP address (aka your pods for a service)
  - Service lookups (SRV records)
    - Named ports: _my-port-name._my-port-protocol.my-svc.my-namespace.svc.cluster.local
      - Example, SRV RR: _http._tcp.example.com. SRV 10 0 8080 host1.example.com. (notice http is 8080)
  - Reverse IP address lookups (PTR records)
    - Ex. FQDN
    - Example w.x.y.z resolves to myserver.ns.svc.cluster.local.

```
user@nodea:~$ nslookup 10.0.0.10 172.18.0.3
Server:         172.18.0.3
Address:        172.18.0.3#53

Non-authoritative answer:
10.0.0.10.in-addr.arpa  name = kube-dns.kube-system.svc.cluster.local.

Authoritative answers can be found from:

user@nodea:~$
```

Who Can Use It?

- The only objects to which we are assigning DNS names are Services
- Every Kubernetes Service is assigned a virtual IP address which is stable as long as the Service exists (as compared to Pod IPs which can change over time due to crashes or scheduling changes)
- This maps well to DNS, which has a long history of clients that, on purpose or on accident, do not respect DNS TTLs (see previous remark about Pod IPs changing)

- The DNS server itself runs as a Kubernetes Service
  - All entries are discovered dynamically via the K8s API Master (which stores all of the info in etcd)

- *Anyone who can reach the DNS service can use it for lookups*

Summary

- Kube-dns provides service dns entries to pods
- Kube-dns updates entries by watching service endpoint on api-server

Lab 5

- Enabling service discovery in the cluster with the kube-dns

6. Kubernetes API and Resources

Objectives

- Describe the key components of the K8s API
- Survey the range of resource types offered by K8s
- Break down the latest K8s features by release

K8s Resources

```
user@nodea:~$ kubectl get componentstatuses
NAME                 STATUS    MESSAGE              ERROR
scheduler            Healthy   ok
controller-manager   Healthy   ok
etcd-0               Healthy   {"health": "true"}
```

- K8s resource types (kubelet shorthand)
  - Certificatesigningrequests (csr) - CSR object (unsigned cert destined for a CA)
  - Clusters - valid only for federation apiservers
    - clusterrolebindings
    - clusterroles
  - Componentstatuses (cs) - stat cluster elements
  - Configmaps (cm) - configurations that can be bound to pods
  - Daemonsets (ds) - ensures that all (or some) nodes run a copy of a pod
  - Deployments - declarative updates for Pods and Replica Sets
  - Events (ev) - cluster activity
  - Endpoints (ep) - service targets
  - Horizontalpodautoscalers (hpa) - K8s autoscaling to number of pods in rs/deployment
  - Ingresses (ing) - rules allowing inbound connections to reach services
  - Jobs - ensures successful pod completion
  - Limitranges (limits) - configures resource limits
  - Nodes (no) - worker machine in K8s
  - Namespaces (ns) - virtual clusters backed by the same physical cluster
  - NetworkPolicy - how pods are allowed to communicate with each other
  - StatefulSet - group of stateful pods
  - Pods (po) - group of one or more containers
  - Persistentvolumes (pv) - provisioned networked storage in the cluster
  - Persistentvolumeclaims (pvc) - reserved cluster storage
  - Resourcequotas (quota) - tool to address fair share of resources
  - ReplicaSet (rs) - ensures pod “replicas” are running at any given time
  - Replicationcontrollers (rc) - see rs, replaced by ReplicaSet
  - Secrets - holds sensitive information
  - Serviceaccounts - provides an identity for processes that run in a Pod
  - Services - a logical set of Pods and a policy to access them
  - Statefulsets - Pods with trackable identity and associated storage
  - Storageclasses - Cloud provider storage classes

K8s 1.7 API Resource versions

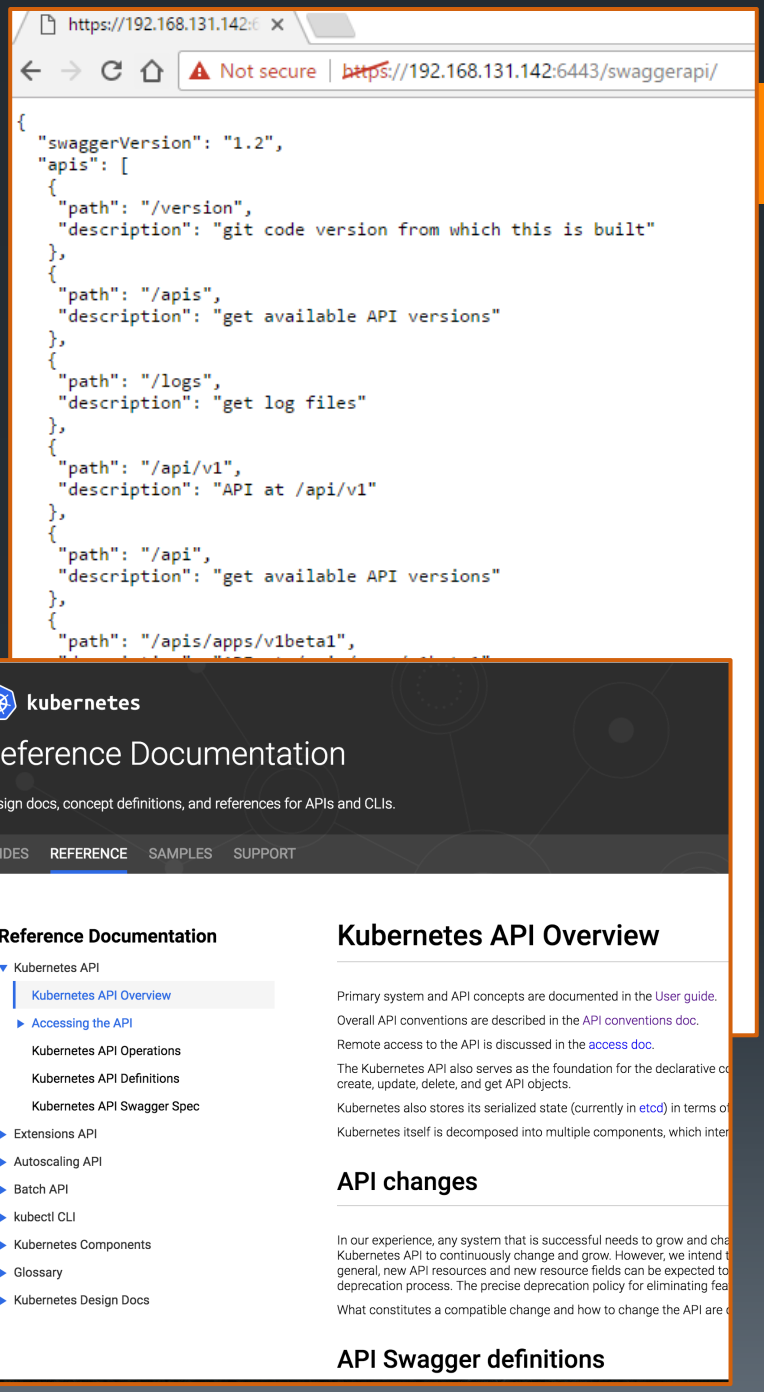
- **v1**
  - WORKLOADS
    - Container
    - Job [batch]
    - Pod
    - ReplicationController
  - DISCOVERY & LOAD BALANCING
    - Endpoints
    - Service
  - CONFIG & STORAGE
    - ConfigMap
    - Secret
    - PersistentVolumeClaim
    - StorageClass
    - Volume
  - METADATA
    - Event
    - LimitRange
    - HorizontalPodAutoscaler
    - PodTemplate
  - CLUSTER
    - Binding
    - ComponentStatus
    - LocalSubjectAccessReview
    - Namespace
    - Node
    - PersistentVolume
    - ResourceQuota
    - SelfSubjectAccessReview
    - ServiceAccount
    - SubjectAccessReview
    - TokenReview
    - NetworkPolicy

- **v1beta1**
  - WORKLOADS
    - DaemonSet
    - Deployment [apps]
    - ReplicaSet
    - StatefulSet [apps]
  - DISCOVERY & LOAD BALANCING
    - Ingress
  - METADATA
    - ControllerRevision
    - PodDisruptionBudget
    - ThirdPartyResource
    - PodSecurityPolicy
  - CLUSTER
    - APIService
    - CertificateSigningRequest
    - ClusterRole
    - ClusterRoleBinding
    - Role
    - RoleBinding
    - NetworkPolicy

- **v2alpha1**
  - WORKLOAD
    - CronJob [batch]
  - METADATA
    - ExternalAdmissionHookConfiguration
    - InitializerConfiguration
    - PodPreset

APIs

- Kubernetes API - Core endpoints
  - Ex. *curl localhost:8080/api/v1/componentstatuses*
- Extension API - Non-core components endpoints
  - Ex. *curl localhost:8080/apis/extensions/v1beta1/deployments*
- Autoscaling API - Horizontal Pod Autoscaling endpoints
  - Ex. *curl localhost:8080/apis/autoscaling/v1/*
- Batch API - Job control endpoints
  - Ex. *curl localhost:8080/apis/batch/v1*
- The API serves as the foundation for the declarative configuration schema for the system
- API stores its serialized state (currently in etcd) in terms of the API resources
- kubectl command-line tool can be used to create, update, delete, and get API objects
- Swagger documented
  - http://swagger.io/
  - The goal of Swagger is to define a standard, language-agnostic interface to REST APIs which allows both humans and computers to discover and understand the capabilities of the service without access to source code, documentation, or through network traffic inspection



Kubernetes API

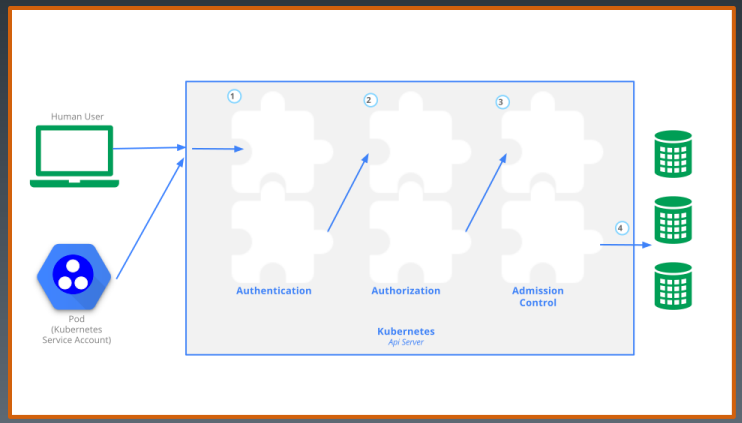
- /api & /apis
- Users access the API using kubectl, client libraries, or by making REST requests
- Both human users and Kubernetes service accounts can be authorized for API access
- API requests go through a security pipeline (authentication, authorization, admission control)
  - Once TLS is established, the HTTP request moves to the Authentication step

  - Authentication
    - If the request cannot be authenticated, it is rejected with HTTP status code 401; otherwise, the user is authenticated as a specific username, and the user name is available to subsequent steps to use in their decisions
    - Multiple authentication modules can be specified, in which case each one is tried in sequence, until one of them succeeds
    - Authentication modules include Client Certificates, Password, and Plain Tokens, and JWT Tokens (used for service accounts)
    - Once the request is authenticated as coming from a specific user, it moves to a generic authorization step

  - Authorization
    - The input to the Authorization step are attributes of the REST request, including: - the username determined by the Authentication step. - a verb associated with the API request. Most object support these common operations: list, watch, create, update, patch, delete.
    - There are multiple supported Authorization Modules including AlwaysDeny, AlwaysAllow, ABAC, RBAC, and Webhook
      - When multiple Authorization Modules are configured, each is checked in sequence, and if any Module authorizes the request, then the request can proceed
        - If all deny the request, then the request is denied (HTTP status code 403)
    - Access controls and policies that depend on specific fields of specific Kinds of objects are handled by Admission Controllers

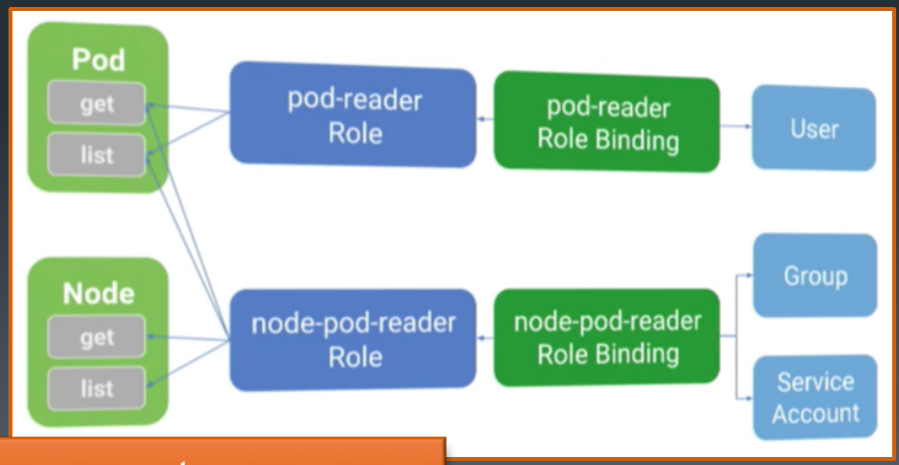
  - Admission Control
    - Admission Control Modules are software modules that can modify or reject requests
    - Multiple admission controllers can be configured and each is called in order; they include AlwaysAdmit, AlwaysPullImages, AlwaysDeny, DenyEscalatingExec, ImagePolicyWebhook, ServiceAccount, PersistentVolumeLabel, SecurityContextDeny, ResourceQuota, LimitRanger, NamespaceLifecycle, DefaultStorageClass, DefaultTolerationSeconds, PodSecurityPolicy
    - Current (K8s >= 1.6.0) recommendation for Admission Controller is:
      - `--admission-control=NamespaceLifecycle,LimitRanger, ServiceAccount,PersistentVolumeLabel,DefaultStorageClass, ResourceQuota,DefaultTolerationSeconds`
    - Unlike Authentication and Authorization Modules, if any admission controller module rejects, then the request is immediately rejected
    - Once a request passes all admission controllers, it is validated using the validation routines for the corresponding API object, and then written to the object store (shown as step **4**)

- https://kubernetes.io/docs/api-reference/v1.7/



RBAC in K8s (beta)

- As of 1.6 RBAC mode is in beta
- RBAC permission policies are configured using kubectl or the Kubernetes API
- Users can be authorized to make authorization policy changes using RBAC
  - Delegate resource management without giving away ssh access to the cluster master (req. w/ ABAC)
- Connection between user and resources is defined in RBAC using two objects:
  - **Roles**
  - **Role Bindings**



To smooth the transition from ABAC to RBAC, you can create Kubernetes 1.6 clusters with both ABAC and RBAC authorizers enabled

Using RBAC Authorization

- RBAC API declares four top-level types
  - Role and ClusterRole
    - Role contains rules that represent a set of permissions
      - Purely additive, no “deny” rules
    - Defined within a namespace (Role) or cluster wide (ClusterRole)
  - RoleBinding and ClusterRoleBinding
    - Grants permissions defined in a role to a user or set of users
    - RoleBinding may reference a Role in the same namespace
      - May also reference a ClusterRole to grant the permissions to namespaced resources
    - ClusterRoleBinding may be used to grant permission at cluster level and in all namespaces
  - Users can interact with these resources as they would with any other API resource

```
kind: Role
apiVersion: rbac.authorization.k8s.io/v1beta1
metadata:
  namespace: default
  name: svc-reader
rules:
  - apiGroups: ["*"]
    resources: ["services"]
    verbs: ["get", "watch", "list"]

---

kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1beta1
metadata:
  name: read-svc
  namespace: default
subjects:
  - kind: User
    name: usera
roleRef:
  kind: Role
  name: svc-reader
  apiGroup: rbac.authorization.k8s.io
```

Service Accounts

- A service account provides an identity for processes that run in a Pod
- When you create a pod, a service account is automatically assigned
  - The default service account of the same namespace
  - When processes in containers inside pods contact the API server, they are authenticated as a particular Service Account (e.g. default)
- In version 1.6+, you can opt out of automounting API credentials for a service account by setting `automountServiceAccountToken: false` on the service account object or a given Pod spec
- Creating additional ServiceAccount objects will automatically create tokens used by the named service account
- Set the `serviceAccountName` field of a pod to the name of the service account to use other than the default
  - The service account has to exist at the time the pod is created, or it will be rejected
- You cannot update the service account of an already created pod

```
$ cat special-sa.yaml

apiVersion: v1
kind: ServiceAccount
metadata:
  name: special-sa

$ kubectl create -f special-sa.yaml
serviceaccount "special-sa" created

$ kubectl get serviceaccounts/special-sa -o yaml

apiVersion:v1
kind: ServiceAccount
metadata:
  creationTimestamp: 2017-06-16T00:12:59Z
  name: special-sa
  namespace: default
  resourceVersion: "272500"
  selfLink: /api/v1/namespaces/default/serviceaccounts/special-sa
  uid: 721ab723-13bc-11e5-aec2-42010af0021e
secrets:
- name: special-sa-token-bvbk5
```

Extension API

- /apis/extensions/v1beta1
- This holds types which will probably move to another API group eventually
- Extensions resources can be enabled by setting runtime-config on apiserver
  - For example to disable deployments and jobs, set the `--runtime-config` setting on the kube-apiserver:
    - `--runtime-config=extensions/v1beta1/deployments=false,extensions/v1beta1/jobs=false`
- To list available extensions:
  - `curl -s http://localhost:8080/apis/extensions/v1beta1 | jq ".resources[].name"`

```
$ curl -s http://localhost:8080/apis/extensions/v1beta1 | jq ".resources[].name"
"daemonsets"
"daemonsets/status"
"deployments"
"deployments/rollback"
"deployments/scale"
"deployments/status"
"ingresses"
"ingresses/status"
"networkpolicies"
"podsecuritypolicies"
"replicasets"
"replicasets/scale"
"replicasets/status"
"replicationcontrollers"
"replicationcontrollers/scale"
"thirdpartyresources"
```

API and Release Versioning

- API versioning is done at the path level, ex.
  - */api/v1*
  - */apis/extensions/v1beta1*
- API Version follows SemVer (semver.org)
  - **Kube X.Y.Z** refers to the version (git tag) of Kubernetes that is released. This versions all components: apiserver, kubelet, kubectl, etc. (**X** is the major version, **Y** is the minor version, and **Z** is the patch version.)
  - **API vX[betaY]** refers to the version of the HTTP API
- Versioning rules
  - Kube X.Y.0-alpha.W, W > 0 (Branch: master) - Alpha releases are released roughly every two weeks directly from the master branch
  - Kube X.Y.Z-beta.W (Branch: release-X.Y) - When master is feature-complete for Kube X.Y, we will cut the release-X.Y branch 2 weeks prior to the desired X.Y.0 date and cherrypick only PRs essential to X.Y
  - Kube X.Y.0 (Branch: release-X.Y) - Final release, cut from the release-X.Y branch cut two weeks prior
  - Kube X.Y.Z, Z > 0 (Branch: release-X.Y) - Patch releases are released as we cherrypick commits into the release-X.Y branch, (which is at X.Y.Z-beta.W,) as needed
  - Kube X.Y.Z, Z > 0 (Branch: release-X.Y.Z) - These are special and different in that the X.Y.Z tag is branched to isolate the emergency/critical fix from all other changes that have landed on the release branch since the previous tag
- Additional details here
  - https://github.com/kubernetes/community/blob/master/contributors/design-proposals/versioning.md

Watching API Resources

- Sometimes you want to monitor a K8s resource for changes
- One way to do that is via the watch API configuration
- Via API
  - `curl localhost:8080/api/v1/nodes?watch=true`
- Via kubectl
  - `kubectl get -w nodes`

```
user@nodea:~$ kubectl get nodes
NAME      STATUS    AGE
nodea     Ready     1d
nodeb     Ready     22h
user@nodea:~$ kubectl get nodes -w
NAME      STATUS    AGE
nodea     Ready     1d
nodeb     Ready     22h
NAME      STATUS                        AGE
nodeb     Ready,SchedulingDisabled      22h
nodeb     Ready,SchedulingDisabled      22h
nodea     Ready     1d
nodeb     Ready     22h
nodeb     Ready     22h
nodea     Ready     1d
odea:~$
```

```
user@nodea:~/temp$ kubectl cordon nodeb
node "nodeb" cordoned
user@nodea:~/temp$ kubectl uncordon nodeb
node "nodeb" uncordoned
```

```
user@nodea:~$ curl -s localhost:8080/api/v1/nodes?watch=true | jq .type
"ADDED"
"ADDED"
"MODIFIED"
"MODIFIED"
"MODIFIED"
"MODIFIED"
"MODIFIED"
^C
user@nodea:~$
```

Security Context

- Defines privilege and access control settings for a Pod or Container
  - Security settings that you specify for a Container override settings made at the Pod level when there is overlap
- Security context settings include:
  - Discretionary Access Control – permission to access an object (like a file) is based on user ID (UID) and group ID (GID)
  - Security Enhanced Linux (SELinux) – objects are assigned security labels
    - SELinux security module must be loaded on the host OS
  - Running as privileged or unprivileged
  - Linux Capabilities – give a process some privileges, but not all the privileges of the root user
  - runAsNonRoot – set containers to only run as non root user
  - readOnlyRootFilesystem – set container filesystem as read only
- Annotations:
  - AppArmor (beta) – use program profiles to restrict the capabilities of individual programs
  - Seccomp (alpha) – define what syscalls are allowed or denied
    - Docker's default is a whitelist

```
apiVersion: v1
kind: Pod
metadata:
  name: hello-nginx
  annotations:
    container.apparmor.security.beta.kubernetes.io/nginx: runtime/default
    container.seccomp.security.alpha.kubernetes.io/nginx: docker/default
spec:
 securityContext:         # pod level security configs
    runAsUser: 1000
    fsGroup: 2000
  containers:
  - name: nginx
    image: nginx
    command: ["nginx", "-g", "daemon off;"]
    securityContext:      # container level security configs
      capabilities:
        add:
          - NET_ADMIN
      seLinuxOptions:
        level: "s0:c123,c456"
      readOnlyRootFilesystem: true
      runAsNonRoot: true
```

New Features in v1.6

- Kubernetes now supports up to 5,000 nodes when using etcd v3
  - etcd3 is now the default
- Role-based access control (RBAC) goes beta
  - defines secure default roles for control plane, node, and controller components
  - Plans to drop ABAC
  - All communication is now over TLS
- kubeadm goes beta
  - Defaults to RBAC
- kubefed goes beta
- Interaction with container runtimes is now through the CRI
  - Docker remains the default runtime via Docker-CRI shim
- Scheduling updates
  - Support for multiple schedulers
  - Nodes and pods now support affinity and anti-affinity
  - Advanced scheduling can be performed with taints and tolerations
  - Can now specify (per pod) how long a pod should stay bound to a node when there is a node problem
- StorageClass pre-installed and set as default on Azure, AWS, GCE, OpenStack, and vSphere
- DaemonSets can now be updated by a rolling update

New Features in v1.5

- StatefulSet (beta)
  - allows workloads that require persistent identity and/or per-instance storage to be created, scaled, deleted and repaired on Kubernetes
- PodDisruptionBudget (beta)
  - API object that specifies the minimum number or minimum percentage of replicas of a collection of pods that must be up at any time
- Kubefed
  - command line tool to help you manage federated clusters
- Windows container support (alpha)
  - Windows Server 2016 nodes and scheduling Windows Server Containers
- Container Runtime Interface (alpha)
  - CRI API to allow pluggable container runtimes; an experimental docker-CRI integration is ready for testing and feedback

New Features in v1.4

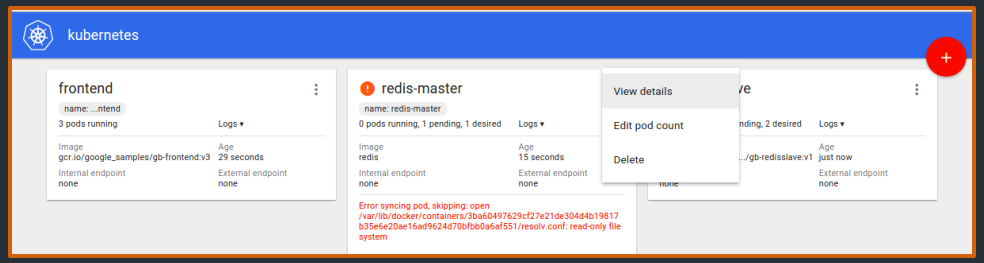
- kubeadm installer (TLS bootstrap)
- Enable extensions/v1beta1/NetworkPolicy by default

New Features in v1.3

- Authorization:
  - **Alpha** RBAC authorization API group
- Federation
  - federation api group is now **beta**
  - Services from all federated clusters are now registered in Cloud DNS (AWS and GCP).
- Stateful Apps:
  - **alpha** PetSets manage stateful apps
  - **alpha** Init containers provide one-time setup for stateful containers
- Updating:
  - Retry Pod/RC updates in kubectl rolling-update.
  - Stop 'kubectl drain' deleting pods with local storage.
  - Add kubectl rollout status
- Security/Auth
  - L7 LB controller and disk attach controllers run on master, so nodes do not need those privileges.
  - Setting TLS1.2 minimum
  - kubectl create secret tls command
  - Webhook Token Authenticator
  - **beta** PodSecurityPolicy objects limits use of security-sensitive features by pods.
- Kubectl
  - Display line number on JSON errors
  - Add flag -t as shorthand for --tty
- Resources
  - Improved node stability by *optionally* evicting pods upon memory pressure - Design Doc
  - **alpha**: NVIDIA GPU support (#24836, @therc)
  - Adding loadBalancer services and nodeports services to quota system

New Features in v1.2

- Significant scale improvements. Increased cluster scale by 400% to 1000 nodes with 30,000 pods per cluster. Kubelet supports 100 pods per node with 4x reduced system overhead.
- Simplified application deployment and management.
- Automated cluster management
- New GUI (dashboard)
- Jobs ga
- HorizontalPodAutoscaler ga
- Kube-Proxy now defaults to an iptables-based proxy
- ReplicaSet API (Beta)
- Dynamic Provisioning of PersistentVolumes (Experimental)
- Run multiple schedulers in parallel (Experimental)



New Features in v1.1

- Daemon Sets
  - Ensures that all nodes run a copy of a pod
  - Good for running:
    - storage daemons (e.g. glusterd, ceph, …)
    - logs collectors (e.g. fluentd, logstash, …)
    - node monitoring (Prometheus Node Exporter, collectd, New Relic agent, Ganglia gmond, …)
- Deployments
  - Declarative update for Pods and ReplicationControllers
  - Allows RC pod template to be changed, extended, etc.
- Ingress
  - New resource type to expose cluster resources outside the cluster
    - Uses an Ingress Controller usually implementing/using a load balancer
    - Ingress controller can be cloud provided or run as a POD (e.g. nginx example provided)
  - Mapping inbound traffic to cluster services
    - Floating Public IPs
    - SSL termination
    - Load balancing
    - …
- Horizontal Pod Autoscaler
  - Allows the number of pods in a replication controller or deployment to scale automatically based on observed CPU utilization
- Jobs
  - creates one or more pods and ensures that a specified number of them successfully terminate

Summary

- Primary API endpoints:
  - Kubernetes API
  - Extension API
  - Autoscaling API
  - Batch API
- Access to apiserver & resources performed in several steps:
  - Authentication, authorization, admission control
- The go-forward authorization mechanism is RBAC
- Workloads can use service accounts to access cluster resources

Lab 6

- RBAC

The End

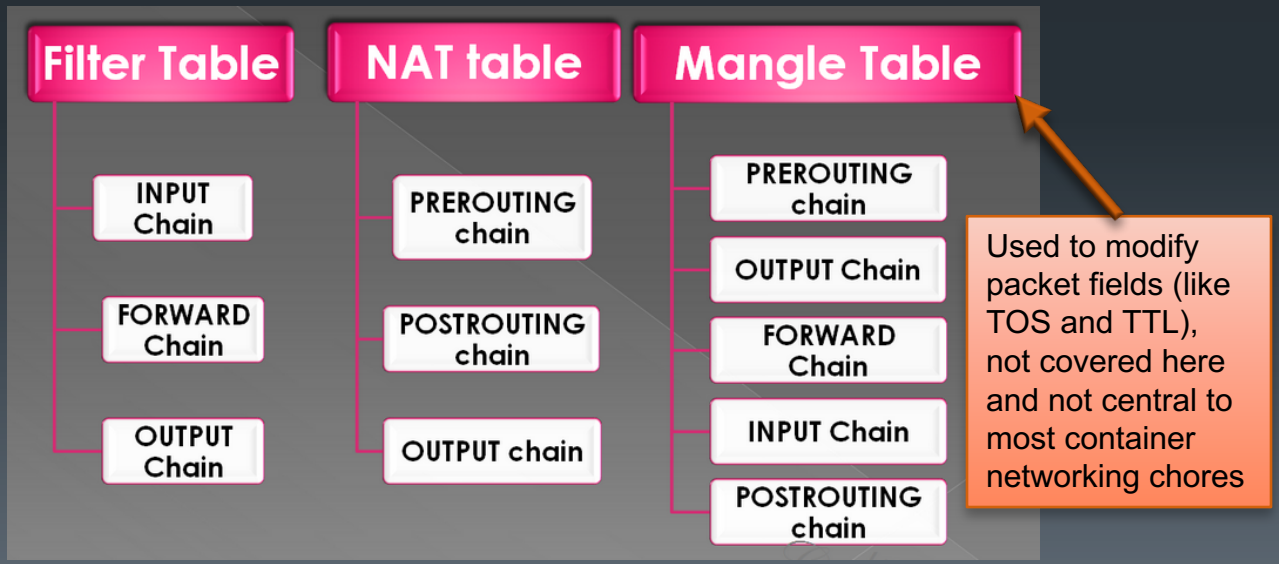
Many thanks for attending!

Appendix

iptables

http://www.iptables.info/en/

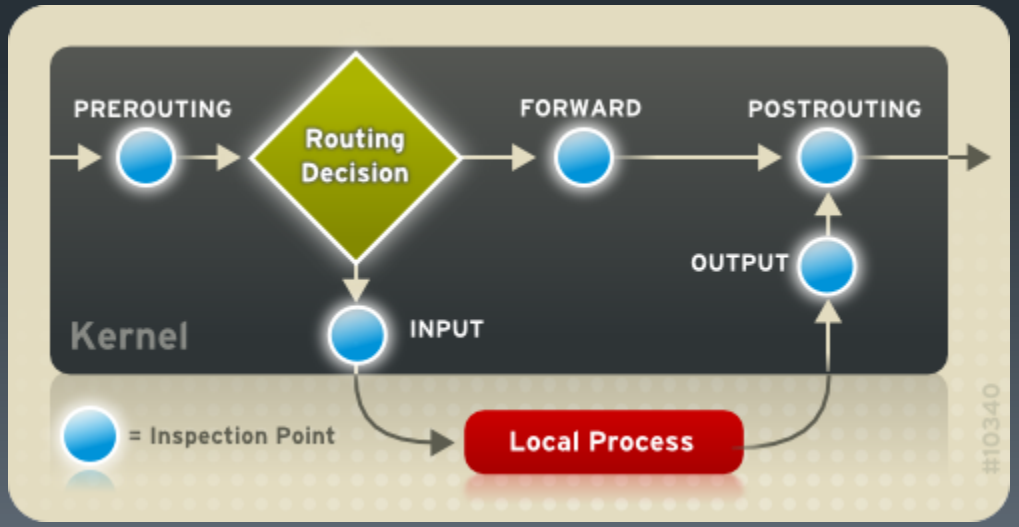
- iptables is a program that configures rules housed in tables used by the Linux kernel when processing network packets
- The term iptables is also commonly used to refer to the kernel packet processing implementation, Netfilter
  - Different Netfilter kernel modules are currently used for different protocols
    - iptables applies to IPv4
    - ip6tables to IPv6
    - arptables to ARP
    - ebtables to Ethernet frames
  - x_tables is the kernel module providing the shared code used by all four modules
  - The successor of iptables is nftables
    - Backward compatible with iptables
    - Merged into Linux kernel version 3.13 (19 January 2014)
- Running the iptables command requires elevated privileges (root)



- Definitions
  - Tables house collections of related packet processing rules
    - **Filter** table
    - **Nat** table
    - **Mangle** table
  - Rules have a matching part and an action part (target)
    - Rules are organized into chains
  - Targets describe what to do with a packet when the rule matches
  - Chains are sequences of rules applied at different phases of packet processing
    - **Prerouting**
    - **Input**
    - **Forward**
    - **Output**
    - **Postrouting**

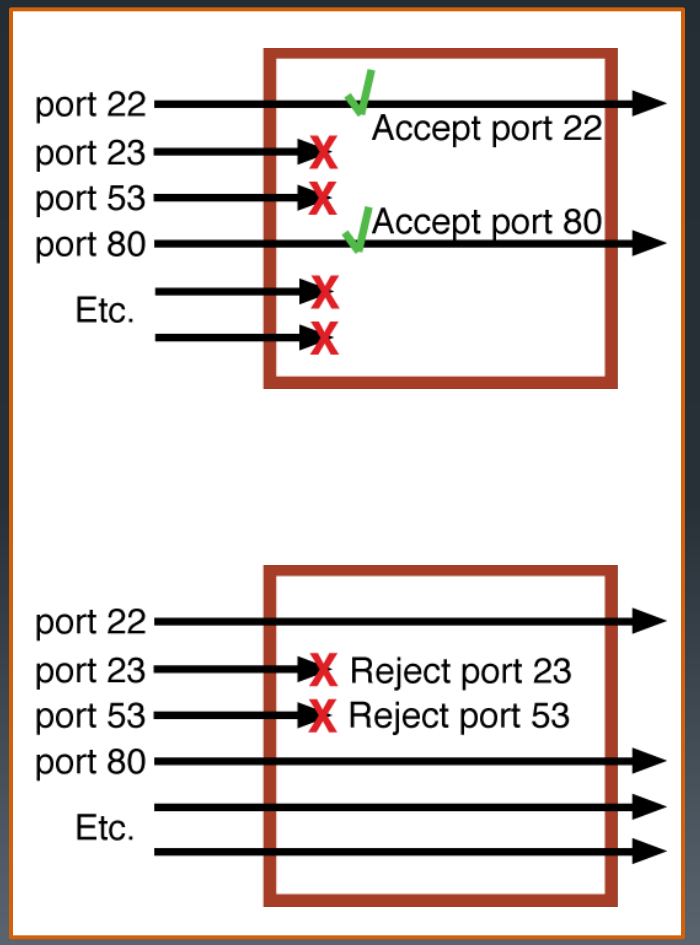
Chains

- *xtables* allows the definition of tables containing chains of rules for the treatment of packets
  - Each table is associated with a different kind of packet processing
  - Packets are processed by sequentially traversing the rules in chains
  - A rule in a chain can cause a goto or jump to another chain, and this can be repeated to whatever level of nesting is desired
    - `-j` Jump – like a "call", i.e. the point that was jumped from is remembered and returned to
    - `-g` Goto – like, well a "goto", i.e. the point of departure is forgotten, returns will resume at the last jump point
  - Every network packet arriving at or leaving from the computer traverses at least one chain
- Packet flow paths
  - Packets start at a given box and will flow along a certain path, depending on the circumstances
  - The origin of the packet determines which chain it traverses initially
  - There are five predefined chains (mapping to the five available Netfilter hooks), though a table need not have all chains
  - Predefined chains have a policy, for example DROP, which is applied to the packet if it reaches the end of the chain
  - The system administrator can create as many chains as desired
    - These chains have no policy; if a packet reaches the end of a user defined chain it is returned to the prior table
  - A chain may be empty
- Chains
  - `PREROUTING`: Packets enter this chain before a routing decision is made
  - `INPUT`: Packet is going to be locally delivered
  - `FORWARD`: All packets that have been routed and were not for local delivery will traverse this chain
    - Inbound traffic bound for containers takes this path
  - `OUTPUT`: Packets sent from the machine itself visit this chain
  - `POSTROUTING`: Routing decision has been made, packets enter this chain just before handing them off to the hardware



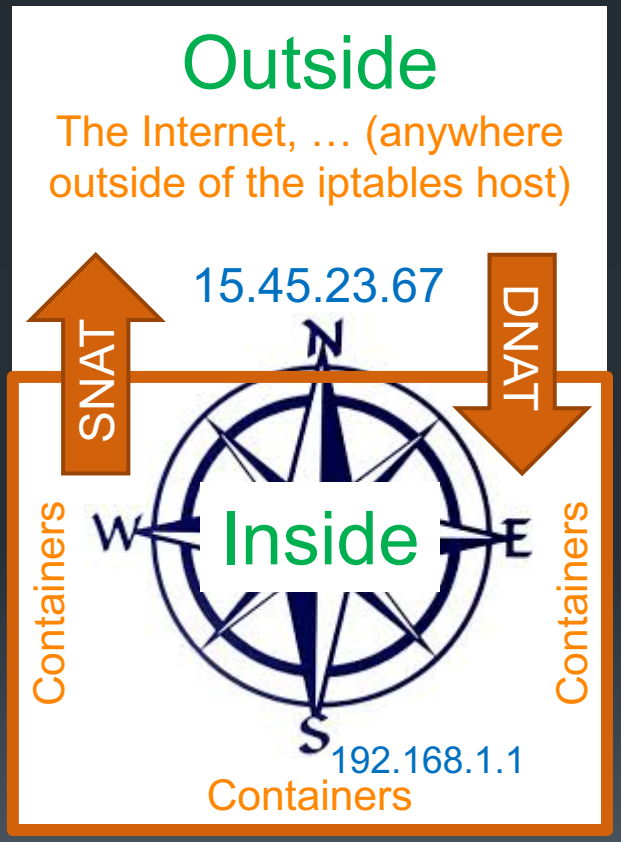
Filter table

- Used to allow or disallow packet transmission
  - Packets can be matched and filtered in many ways
  - Provides basic firewalling
  - The default table
    - No need for the `-t` switch, but if you must: `-t filter`
- Most targets are legal in this table:
  - ACCEPT
    - `iptables -A INPUT -p tcp --dport 22 -j ACCEPT`
    - Packets matching this rule are transmitted on immediately and will not continue traversing the current chain or any other chains in the same table
    - A packet accepted in one chain might still travel through chains within other tables which may drop it
  - DROP
    - `iptables -A INPUT -p tcp --dport 22 -j DROP`
      - Change source of outbound traffic to 15.45.23.67
    - Discards packets with no further processing
    - This action may leave sockets pending response on hosts
    - If a packet is DROPed in a sub chain it will not be processed further in any of the main chains or in any other table
    - The target will not send any kind of information in either direction
  - REJECT
    - `iptables -A FORWARD -p tcp --dport 22 -j REJECT --reject-with tcp-reset`
    - The same as the DROP target but sends back an error message to the source host
    - Only valid in the INPUT, FORWARD and OUTPUT chains or their sub chains
    - All chains that use the REJECT target may only be called by the INPUT, FORWARD, and OUTPUT chains
    - Contrary to popular belief, DROP does not give better security than REJECT, rather it inconveniences legitimate users without additional protection from malicious ones
      - http://www.chiark.greenend.org.uk/~peterb/network/drop-vs-reject



NAT Table

- Used to translate a packet's source IP field or destination IP field
- Compass Model
  - Traffic from inside to outside: S -> N SNAT/MASQ
  - Traffic from outside to inside: N -> S DNAT
  - Traffic between containers: E -> W (or if you prefer W -> E)
- There are 4 NAT Targets:
  - DNAT – Destination NAT : North to South
    - `iptables -t nat -A PREROUTING -p tcp -d 15.45.23.67 --dport 80 -j DNAT --to-destination 192.168.1.1:8089`
      - Change destination of inbound traffic hitting 15.45.23.67:80 to 192.168.1.1:8089
    - Changes the destination address (and optionally port) of the packet and reroutes it
    - Allows an interface with a public IP to redirect traffic to an inside host
      - To a machine on a physical (e.g. DMZ) network behind the firewall
      - To a container on a virtual (e.g. Docker) network behind the firewall
  - SNAT – Source NAT : South to North
    - `iptables -t nat -A POSTROUTING -p tcp -o eth0 -j SNAT --to-source 15.45.23.67`
      - Change source of outbound traffic to 15.45.23.67
    - Changes the source address of packets from inside destined for outside using a defined outside IP address
    - Makes it possible to connect to the Internet from inside
  - MASQUERADE – South to North
    - `iptables -t nat -A POSTROUTING -p tcp -o eth0 -j MASQUERADE`
    - Like SNAT but translates based on an interface not a preconfigured IP
    - The IP of the outside interface is used allowing this to work when the interface receives its address dynamically (DHCP, etc.)
  - REDIRECT
    - `iptables -t nat -A PREROUTING -p tcp --dport 80 -j REDIRECT --to-ports 8080`
    - Used to redirect packets to the machine itself
    - Only valid within the PREROUTING and OUTPUT chains of the nat table or within user-defined chains that are only called from those chains
- Only the first packet in a stream will hit this table
  - After initial inspection all other packets sent over the connection will automatically have the same action taken on them



- Runs as a Deployment
- Uses a ConfigMap for its config file
- Exposes grafana & prometheus dashboards as service endpoints

```
apiVersion: apps/v1beta1
kind: Deployment
metadata:
  labels:
    app: prometheus
  name: prom-dep
spec:
  replicas: 1
  template:
    metadata:
      labels:
        app: prometheus
    spec:
      nodeSelector:
        node-role.kubernetes.io/master:
      tolerations:
      - key: "node-role.kubernetes.io/master"
        operator: "Equal"
        value: ""
        effect: "NoSchedule"
      containers:
      - image: quay.io/prometheus/prometheus:v1.6.2
        name: prometheus
        command:
        - "/bin/prometheus"
        args:
        - "-config.file=/etc/prometheus/prometheus.yml"
        - "-storage.local.path=/prometheus"
        - "-storage.local.retention=12h"
        ports:
        - containerPort: 9090
          protocol: TCP
        volumeMounts:
        - mountPath: "/prometheus"
          name: data
        - mountPath: "/etc/prometheus"
          name: config-volume
...
```

```
...
        resources:
          requests:
            cpu: 100m
           memory: 100Mi
          limits:
            cpu: 500m
            memory: 2500Mi
      - image: grafana/grafana
        name: grafana
        ports:
        - containerPort: 3000
        env:
        - name: GF_SECURITY_ADMIN_PASSWORD
          value: "grafpswd"
        volumeMounts:
        - mountPath: "/var/lib/grafana"
          name: awsvol
      volumes:
      - emptyDir: {}
        name: data
      - configMap:
          name: prom-config
        name: config-volume
      - name: awsvol
        awsElasticBlockStore:
          volumeID: vol-066c9034983ed7fdd
          fsType: ext4
```

Prometheus Example

VMware NSX-T

- Integrating Kubernetes with NSX-T involves three components
  - NSX Container Plugin (NCP)
    - A container image running as an infrastructure Pod in the Kubernetes cluster
    - Watches for changes on Kubernetes Objects (namespaces, network policies, services etc.)
    - Creates networking constructs in NSX-T based on the object add/change events
  - NSX CNI Plugin
    - A small executable installed on all Kubernetes Nodes
    - Kubelet will instantiate/call the CNI Plugin to handle the Pod network attachment
  - NSX Kube-Proxy
    - A daemon running on the Kubernetes Nodes in a container image
    - Run as a Kubernetes Daemon-Set
    - Uses OpenVSwitch (OVS) load-balancing features
    - Replaces the native Kube-Proxy, which uses IPTables
- Integration with underlying VMware IaaS NSX eliminates double encapsulation

Taken from a Spring 2017 blog by Yves Fauser, a Technical Product Manager in VMware's Network and Security Business Unit