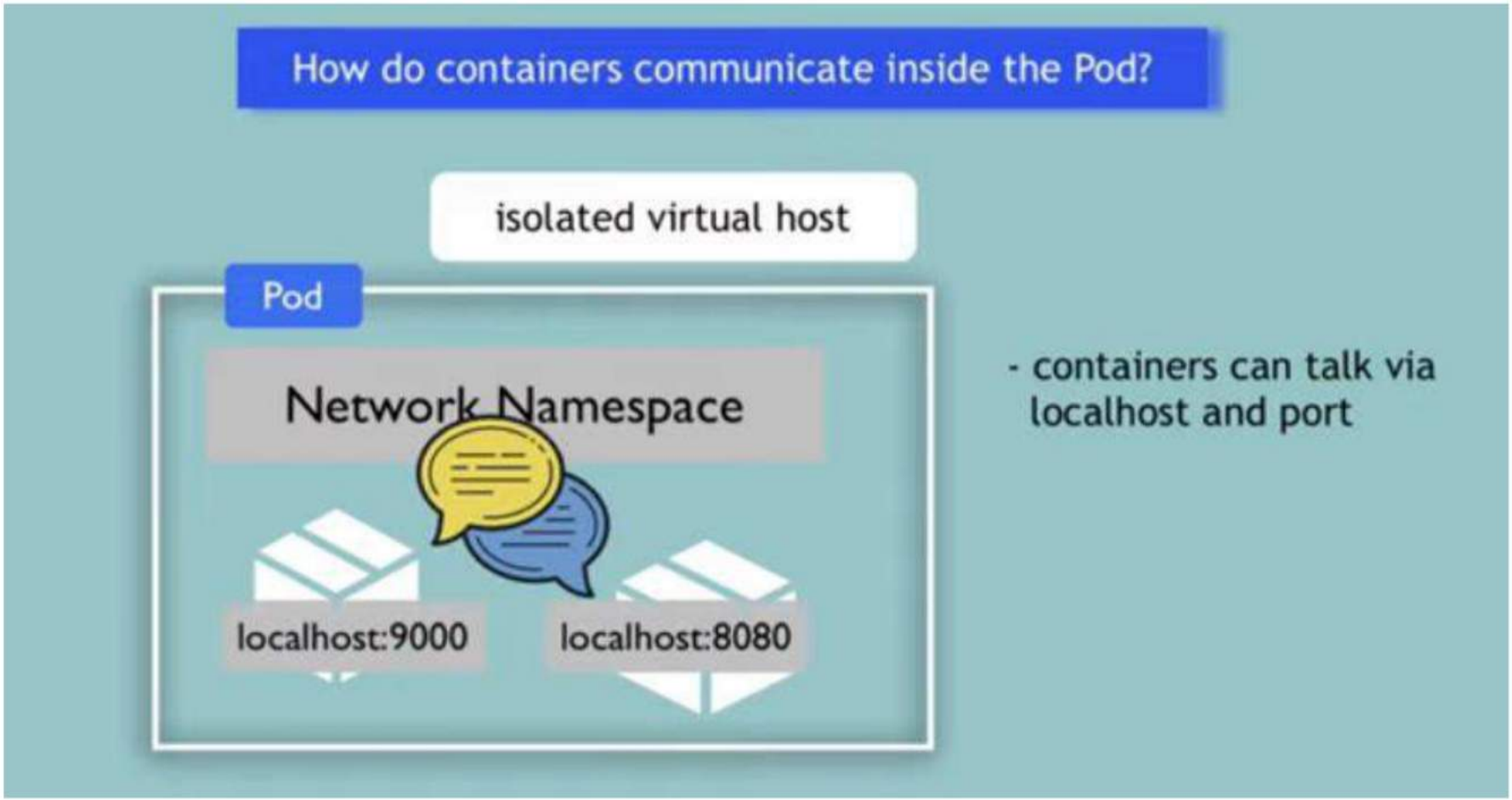

How do containers communicate inside the Pod?
isolated virtual host
Pod
Network Namespace
localhost:9000
localhost:8080
- containers can talk via localhost and port

Default Scheduler

- kube-scheduler is the default scheduler for Kubernetes
- For every newly created pod or other unscheduled pods, kube-scheduler selects a optimal node for them to run on
- kube-scheduler selects a node for the pod in a 2-step operation:
  - Filtering : Filters the set of Nodes where it's feasible to schedule the Pod
  - Scoring : Scheduler ranks the nodes to choose the best fit node

Labels

- key-value pairs that are used to identify, describe and group together related sets of objects or resources.
- **NOT** characteristic of uniqueness.
- Have a strict syntax with a slightly limited character set*.

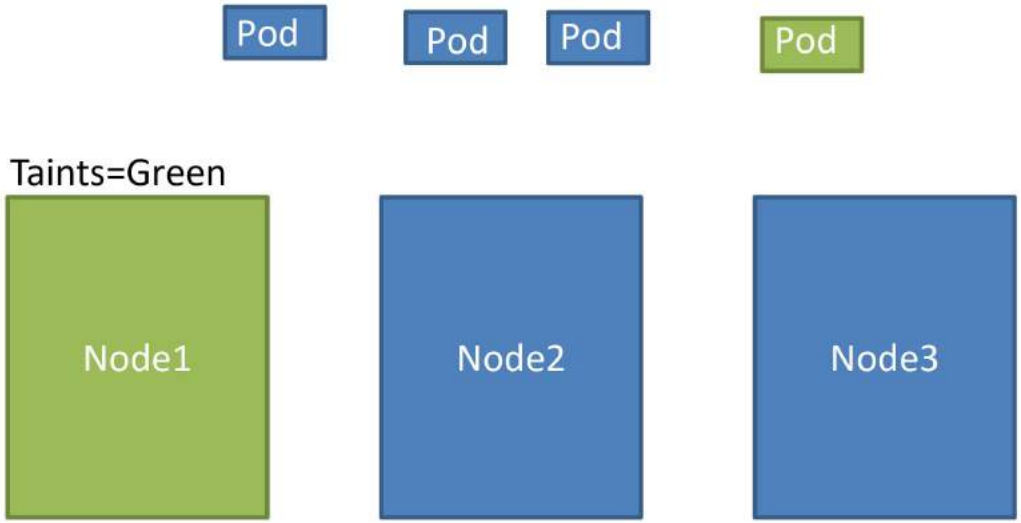
```yaml
apiVersion: v1
kind: Pod
metadata:
   name: nginx
   annotations:
       description: "nginx-test-server"
   labels:
      app: nginx
      env: prod
spec:
   containers:
   - name: mycon123
     image: nginx:latest
     ports:
     - containerPort: 80
~
```

annotations

```yaml
apiVersion: v1
kind: Pod
metadata:
   name: nginx
   annotations:
       description: "nginx-test-server"
   labels:
      app: nginx
      env: prod
spec:
   containers:
   - name: mycon123
     image: nginx:latest
     ports:
     - containerPort: 80
~
```

Taints and Tolerations

Taints are set on Node
Tolerations are set on POD



#kubectl taint nodes node-name key=value:taint effect

NoSchedule , PreferNoSchedule, NoExecute

#kubectl taint nodes node1 app=myapp:NoSchedule

NoSchedule instructs Kubernetes scheduler not to schedule any new pods to the node unless the pod tolerates the taint..
PreferNoSchedule then Kubernetes will try to not schedule the pod onto the node but no guaranty.
NoExecute new pod will not be scheduled and existing pod will be evicted if they do not tolerate the taint.

taints and toleration does not tell the POD to go to a particular node. Instead it tells the node to only accept PODs with certain toleration.
If your requirement is to restrict a POD to certain nodes, it is achieved through node affinity.

Replication Controllers

A **replication controller** ensures that a **specified number of replicas of a pod are running at all times**. If pods exit or are deleted, the replication controller acts to instantiate more up to the defined number. Likewise, if there are more running than desired, it deletes as many as necessary to match the defined amount.

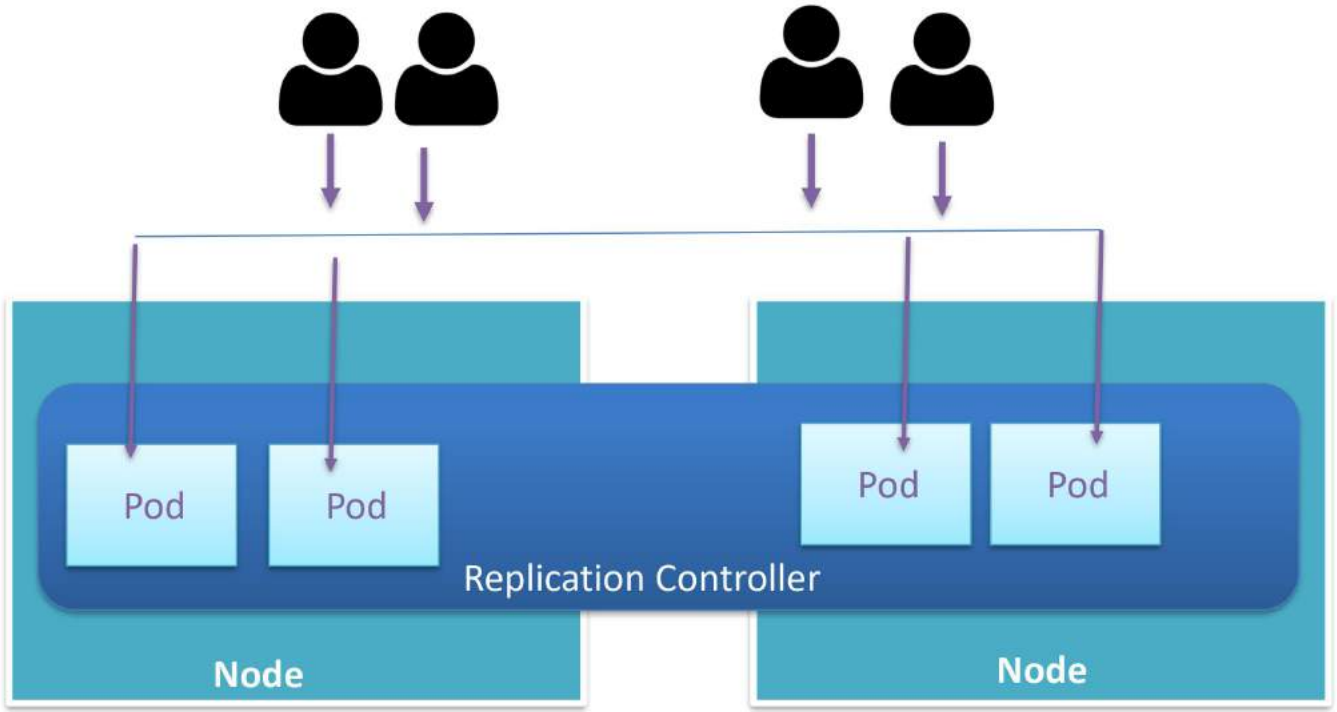
A replication controller configuration consists of:

1.The number of replicas desired (which can be adjusted at runtime).
2.A pod definition to use when creating a replicated pod.
3.A selector for identifying managed pods.

A selector is a set of labels assigned to the pods that are managed by the replication controller. These labels are included in the pod definition that the replication controller instantiates. The replication controller uses the selector to determine how many instances of the pod are already running in order to adjust as needed.

The replication controller does not perform auto-scaling based on load or traffic, as it does not track.

Load Balancing and Auto Scaling



Selector Types

Equality based selectors allow for simple filtering (=,==, or !=).

```
selector:
  matchLabels:
    gpu: nvidia
```

kubectl get pods -l db!=oracle

Set-based selectors are supported on a limited subset of objects. However, they provide a method of filtering on a set of values, and supports multiple operators including: `in`, `notin`, and `exist`.

```
selector:
  matchExpressions:
    - key: gpu
      operator: in
      values: ["nvidia"]
```

kubectl get pods -l 'db in (oracle)'

Replica Set

Similar to a replication controller, a replica set ensures that a specified number of pod replicas are running at any given time. The difference between a replica set and a replication controller is that a replica set supports set-based selector requirements whereas a replication controller only supports equality-based selector requirements.

Only use replica sets if you require custom update orchestration or do not require updates at all, otherwise, use Deployments

A ReplicaSet identifies new Pods to acquire by using its selector. If there is a Pod that match a ReplicaSet's selector, it will be immediately acquired by said ReplicaSet.

```
spec:
   replicas: 3
   selector:
     matchExpressions:
      - {key: app, operator: In, values: [soaktestrs, soaktestrs, soak
      - {key: teir, operator: NotIn, values: [production]}
  template:
     metadata:
```

Deployment

Deployment configuration:

Deployment

ReplicaSet

Pod

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.7.9
        ports:
        - containerPort: 80
```

Service configuration:

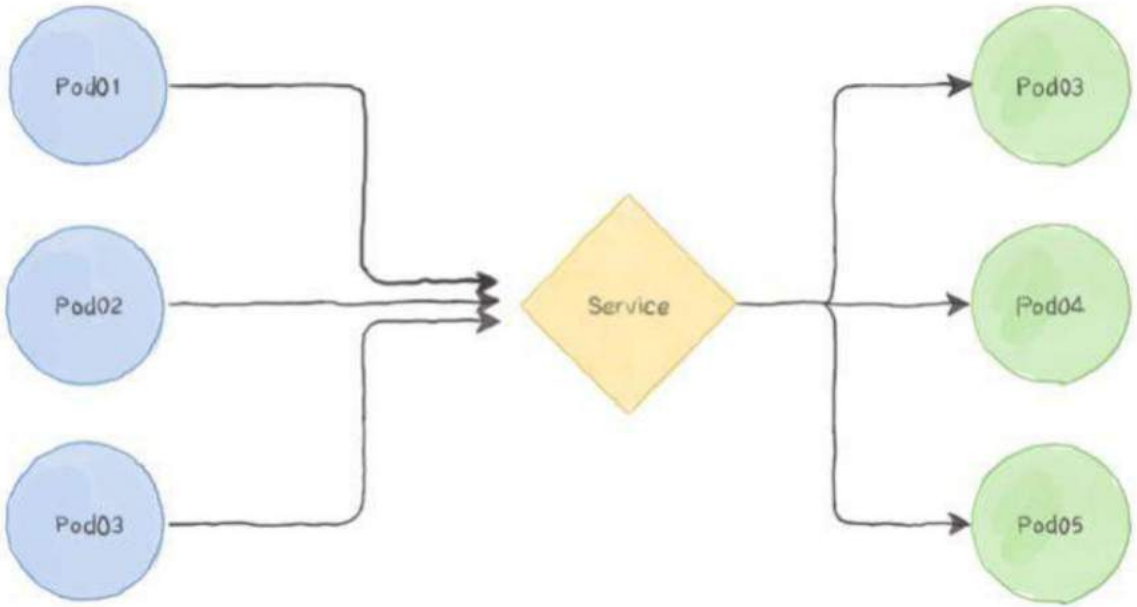
```yaml
kind: Service
apiVersion: v1
metadata:
  name: my-service
spec:
  selector:
    app: nginx
  ports:
  - protocol: TCP
    port: 80
    targetPort: 9376
```

Ingress configuration:

```yaml
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: my-ingress
  annotations:
    kubernetes.io/ingress.class: nginx
spec:
  rules:
  - host: foo.bar.com
    http:
      paths:
      - path: /foo
        backend:
          serviceName: my-service
          servicePort: 80
      - path: /bar
        backend:
          serviceName: my-other-service
          servicePort: 80
```

Service Discovery

Service discovery is the process of figuring out how to connect to a service. While there is a service discovery option based on environment variables available, the DNS-based service discovery is preferable.



Kube-DNS

As noted in the previous section, Kubernetes version 1.11 introduced new software to handle the kube-dns service. The motivation for the change was to increase the performance and security of the service. Let's take a look at the original kube-dns implementation first.
kube-dns

The kube-dns service prior to Kubernetes 1.11 is made up of three containers running in a kube-dns pod in the kube-system namespace. The three containers are:

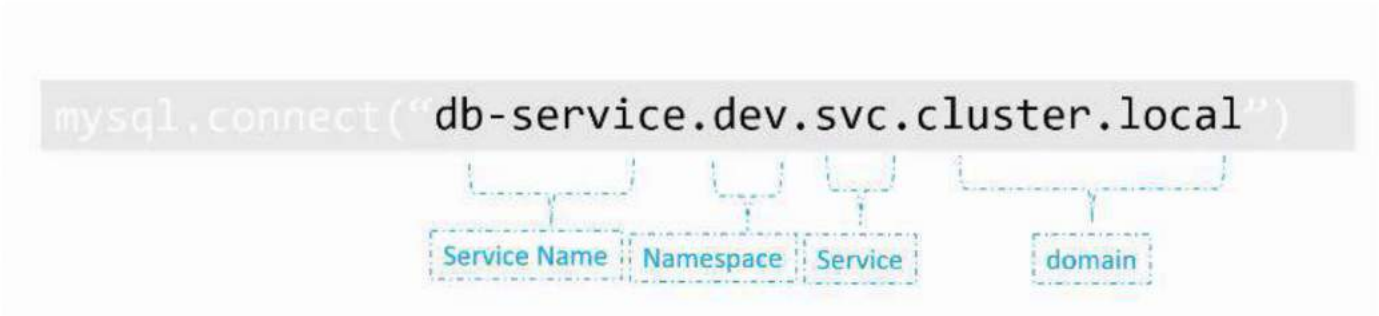
kube-dns: a container that runs SkyDNS, which performs DNS query resolution
dnsmasq: a popular lightweight DNS resolver and cache that caches the responses from SkyDNS
sidecar: a sidecar container that handles metrics reporting and responds to health checks for the service

Security vulnerabilities in Dnsmasq, and scaling performance issues with SkyDNS led to the creation of a replacement system, CoreDNS.

DNS and namespace

Servive Discovery works with coredns

```
mysql.connect("db-service.dev.svc.cluster.local")
```



Namespace and Resource Quota

```yaml
apiVersion: v1
kind: Namespace
metadata:
    name: dev
```

```yaml
apiVersion: v1
kind: ResourceQuota
metadata:
    name: compute-quota
    namespace: dev
spec:
  hard:
    pods: "10"
    requests.cpu: "4"
    requests.memory: 5Gi
    limits.cpu: "10"
    limits.memory: 10Gi
```

List all pods in default NameSpaace

```
Kubectl get pods
```

List all pod in dev NS

```
Kubeclt get pods  - -namespace=dev
```

Change Default NameSpace from Default to dev

```
Kubectl config set-context $(kubectl config current-context) - -namespace=dev
```

Now if want to list pods available in default namespace

```
Kubeclt get pods  - -namespace=default
```

List pods available in all name-spaces

```
Kubeclt get pods  - -all-namespaces
```

Pod Health checks

	Liveliness	Readiness
On failure	Kill container	Stop sending traffic to pod
Check types	Http , exec , tcpSocket	Http , exec , tcpSocket
Declaration example (Pod.yaml)	livenessProbe: failureThreshold: 3 httpGet: path: /healthz port: 8080	readinessProbe: httpGet: path: /status port: 8080

ExecAction - executes a command inside the container.
TCPSocketAction - performs a TCP check against the container's IP address on a specified port.
HTTPGetAction - performs an HTTP GET request on the container's IP.
A handler can then return the following:

Success - the diagnostic passed on the container.
Fail - the container failed the diagnostic and will restart according to its restart policy.
Unknown - the diagnostic failed and no action will be taken.

DaemonSet

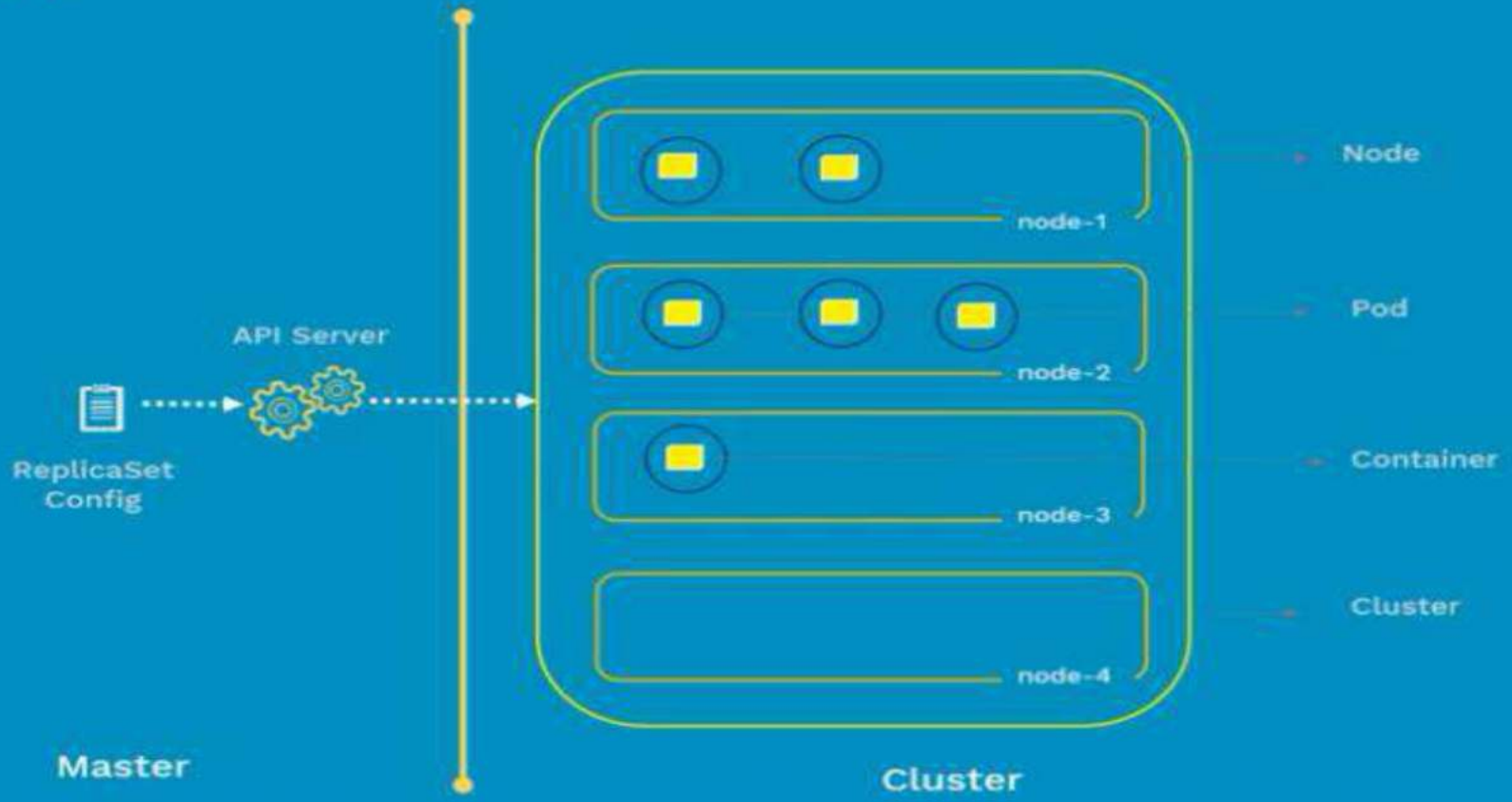
A DaemonSet ensures that all Nodes run a copy of a Pod. As nodes are added to the cluster, Pods are added to them. As nodes are removed from the cluster.

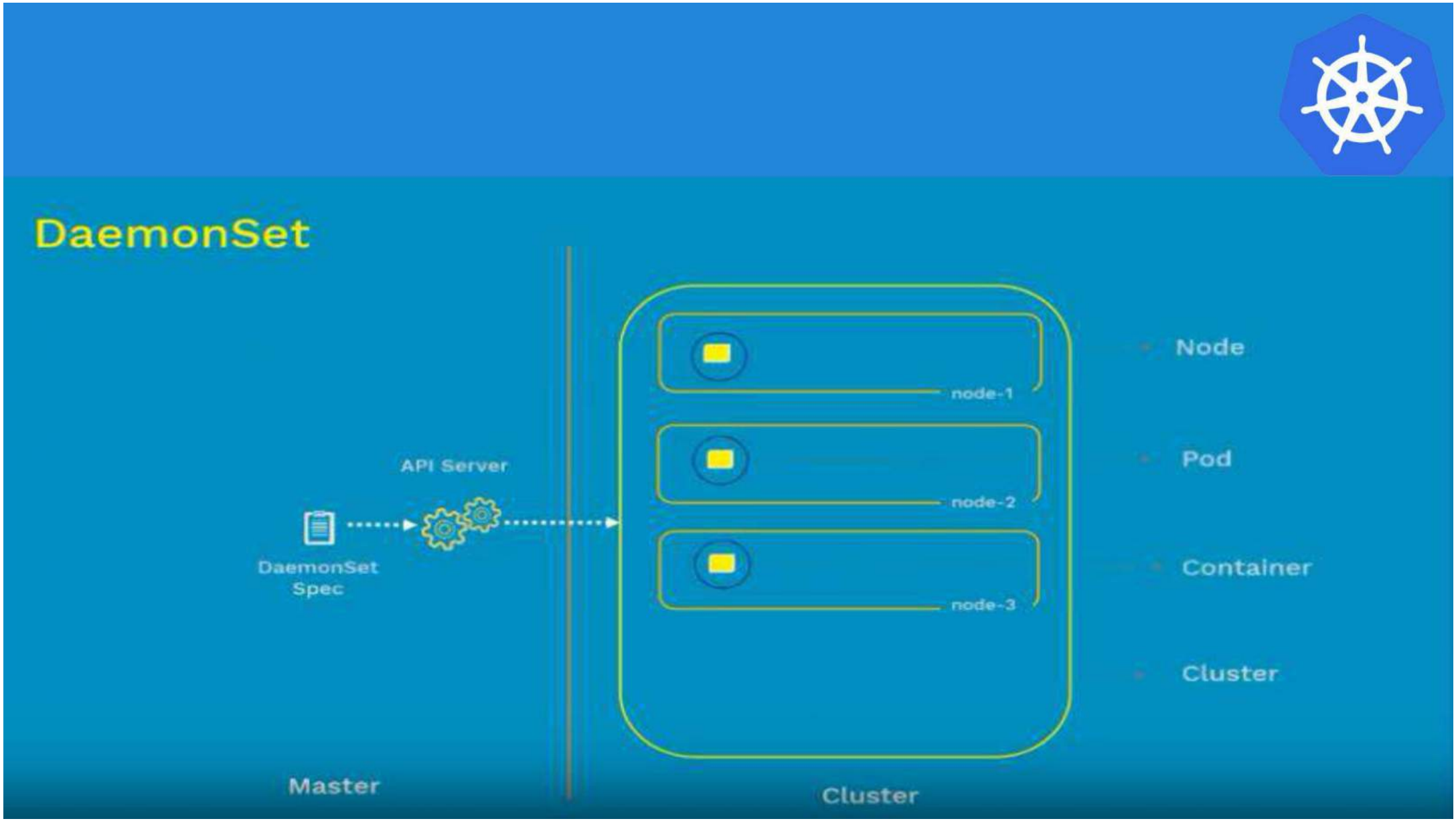
Some typical uses of a DaemonSet are:

running a logs collection daemon on every node
running a node monitoring daemon on every node

In a simple case, one DaemonSet, covering all nodes,

Replica Set / Deployment




DaemonSet
API Server
DaemonSet
Spec
node-1
node-2
node-3
Node
Pod
Container
Cluster
Master
Cluster

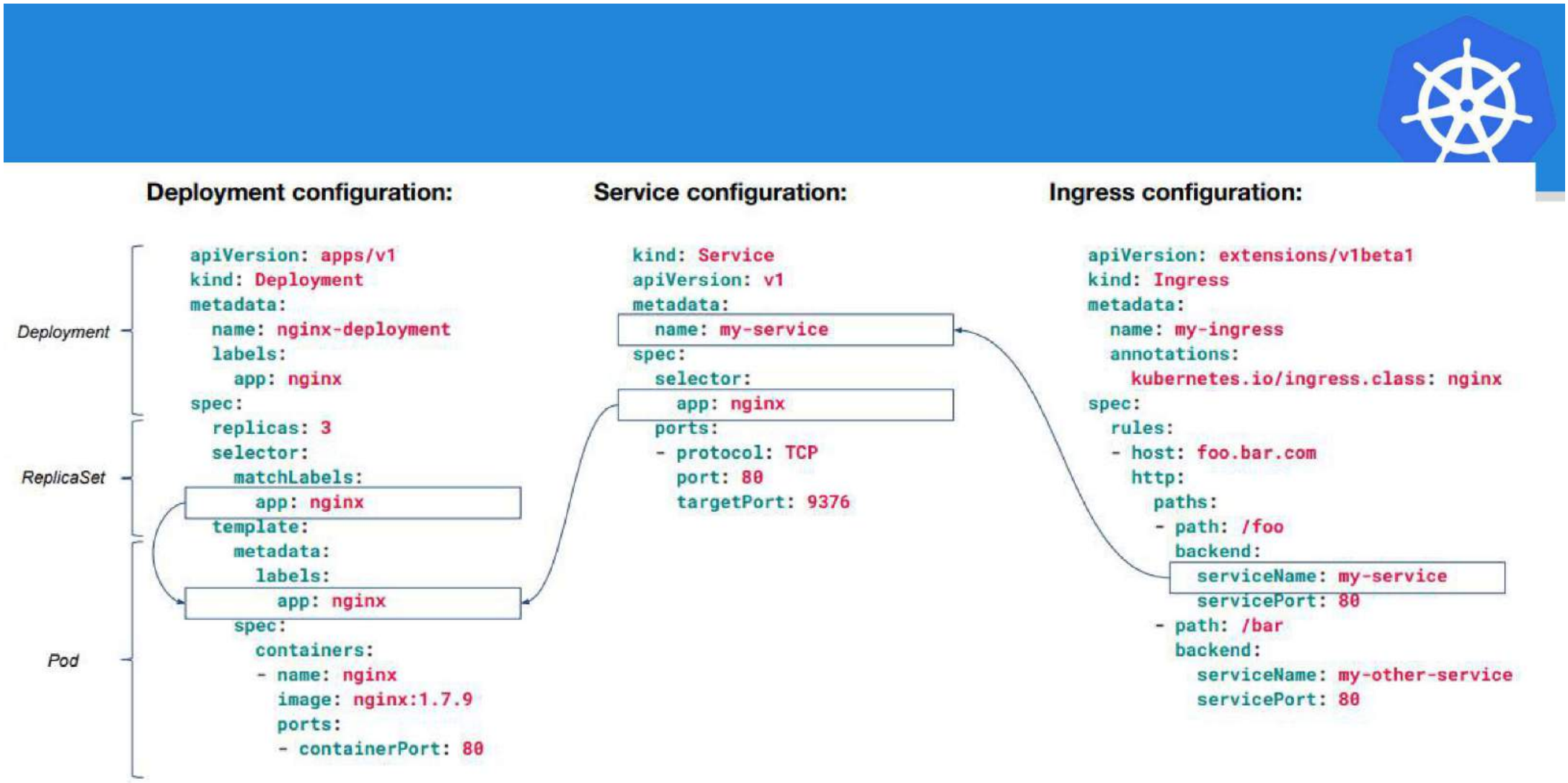
Secret

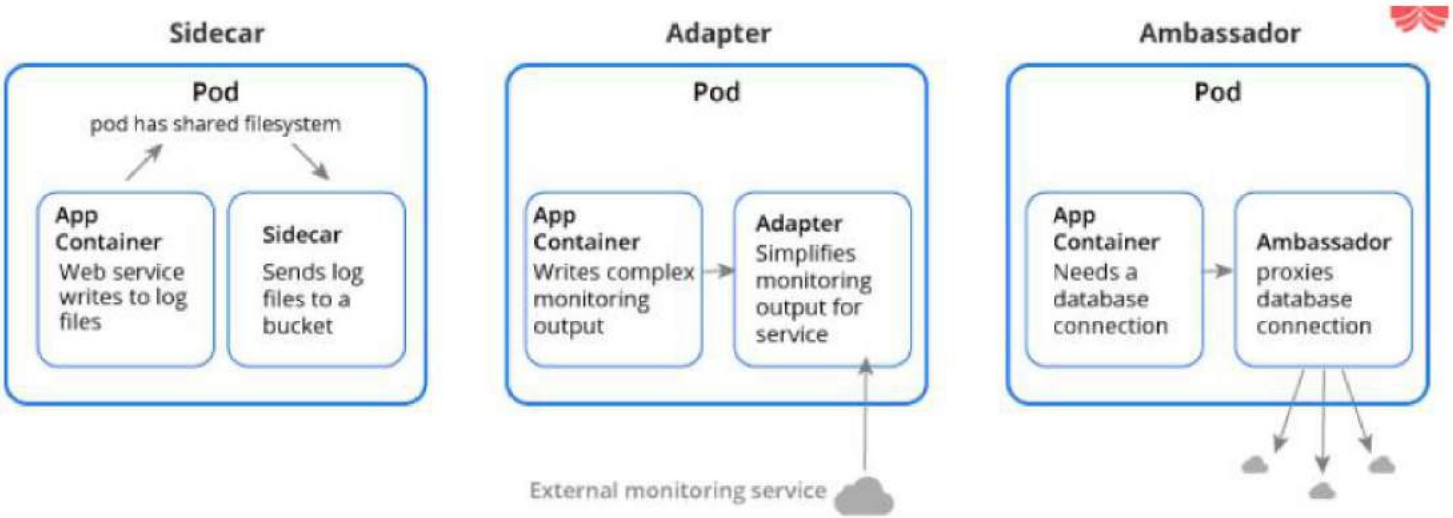
```
root@master:~/secret# kubectl create secret generic mycred --from-file=user.txt --fro
m-file=pass.txt --dry-run -o yaml
W1011 12:20:47.943889  133015 helpers.go:553] --dry-run is deprecated and can be repl
aced with --dry-run=client.
apiVersion: v1
data:
  pass.txt: YWJjMTIzCg==
  user.txt: dXNlcjEKCg==
kind: Secret
metadata:
  creationTimestamp: null
  name: mycred
root@master:~/secret#
```

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: selc-nod
  labels:
    env: test
  annotations:
      imagePulldFrom: http://hub.docker.com
spec:
  containers:
  - name: mypod-123
    image: nginx:latest
    ports:
    - containerPort: 80
    envFrom:
       - secretRef:
            name: demo
```

aws dynamic Volume provisioning

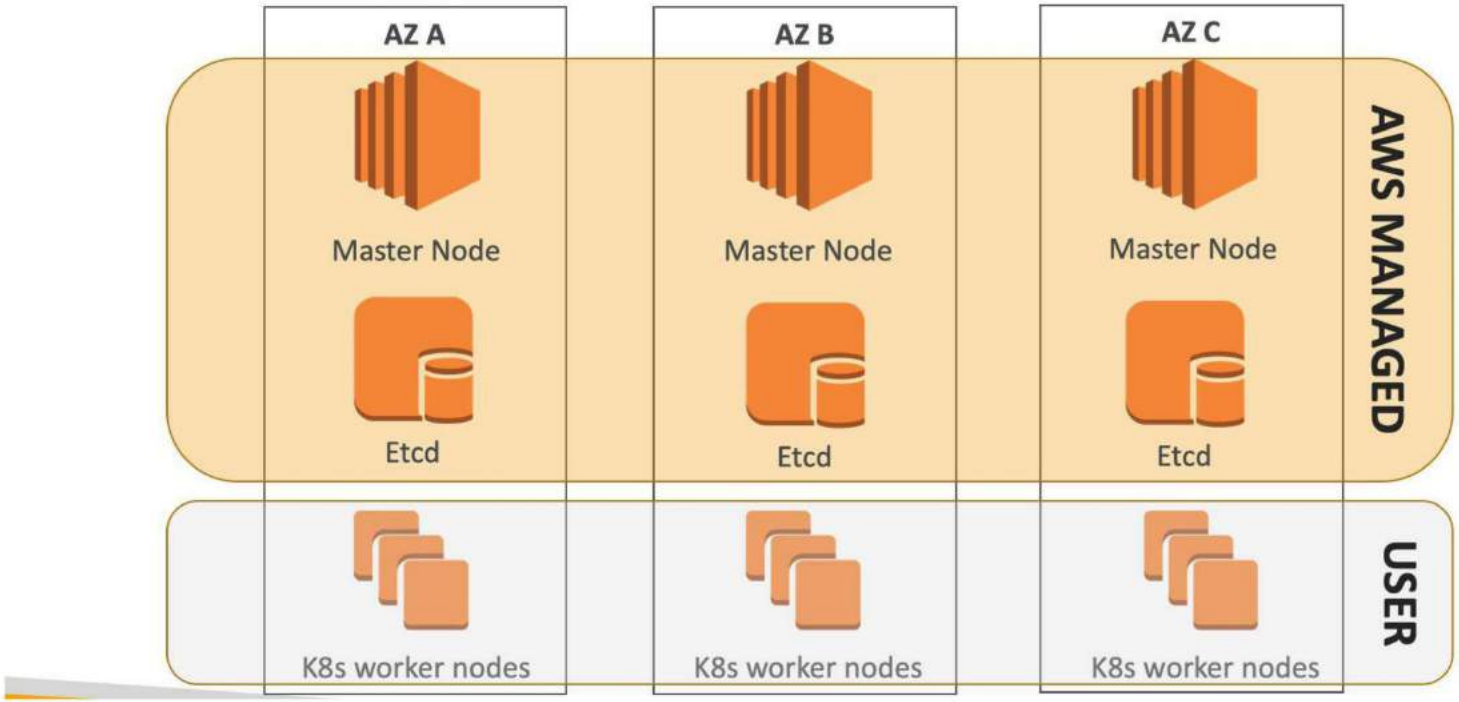
See yaml files ,


Deployment configuration:
Service configuration:
Ingress configuration:
Deployment
ReplicaSet
Pod
apiVersion: apps/v1
kind: Deployment
metadata:
name: nginx-deployment
labels:
app: nginx
spec:
replicas: 3
selector:
matchLabels:
app: nginx
template:
metadata:
labels:
app: nginx
spec:
containers:
- name: nginx
image: nginx:1.7.9
ports:
- containerPort: 80
kind: Service
apiVersion: v1
metadata:
name: my-service
spec:
selector:
app: nginx
ports:
- protocol: TCP
port: 80
targetPort: 9376
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
name: my-ingress
annotations:
kubernetes.io/ingress.class: nginx
spec:
rules:
- host: foo.bar.com
http:
paths:
- path: /foo
backend:
serviceName: my-service
servicePort: 80
- path: /bar
backend:
serviceName: my-other-service
servicePort: 80


Sidecar
Pod
pod has shared filesystem
App Container
Web service writes to log files
Sidecar
Sends log files to a bucket
Adapter
Pod
App Container
Writes complex monitoring output
Adapter
Simplifies monitoring output for service
External monitoring service
Ambassador
Pod
App Container
Needs a database connection
Ambassador
proxies database connection

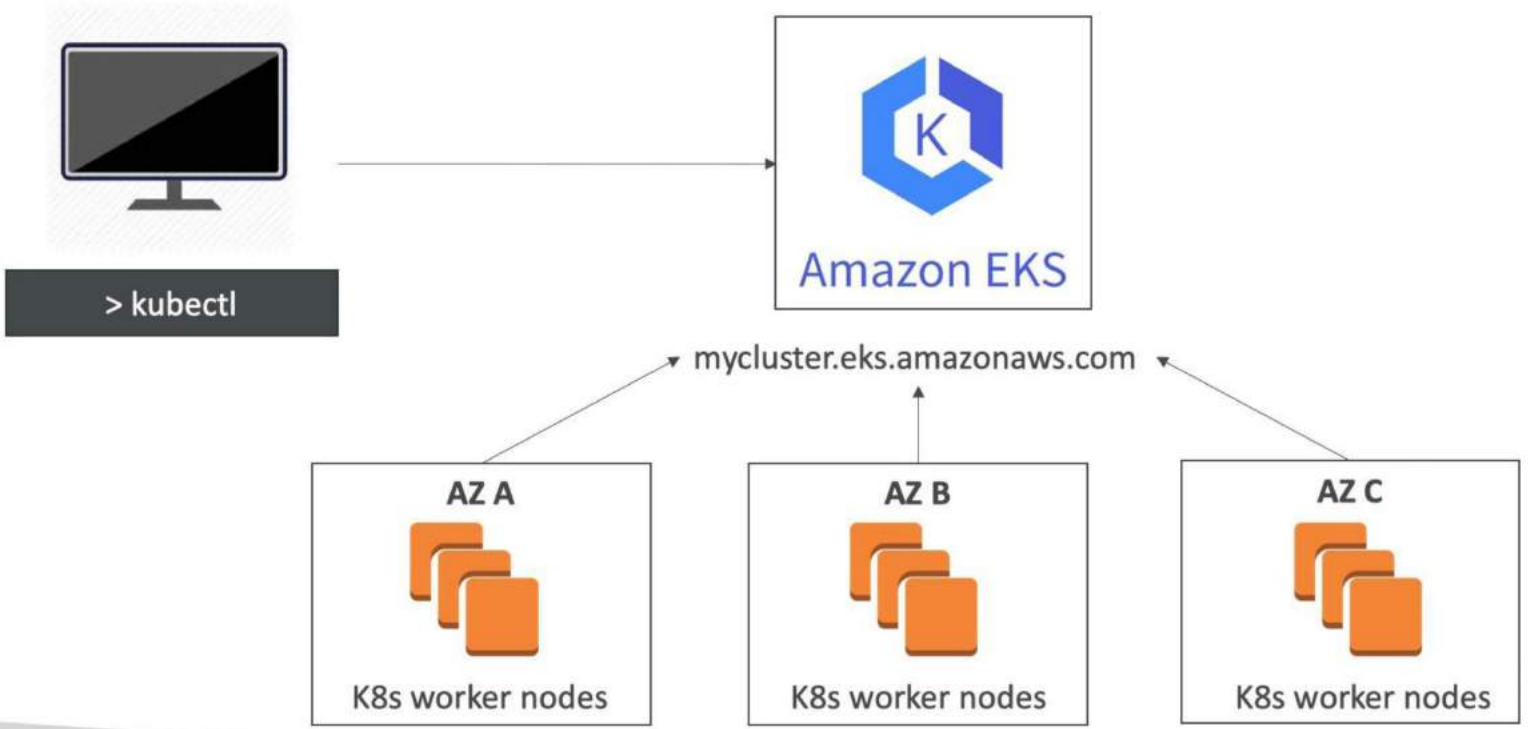
EKS

Kubernetes Control Plane



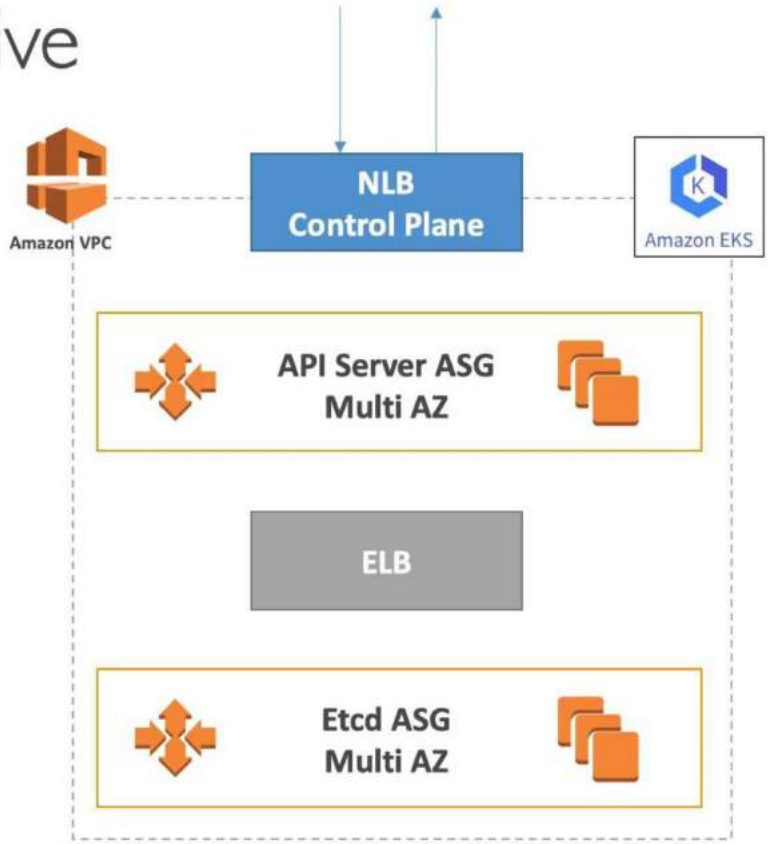
Pricing:
https://aws.amazon.com/eks/pricing/

Kubernetes Control Plane



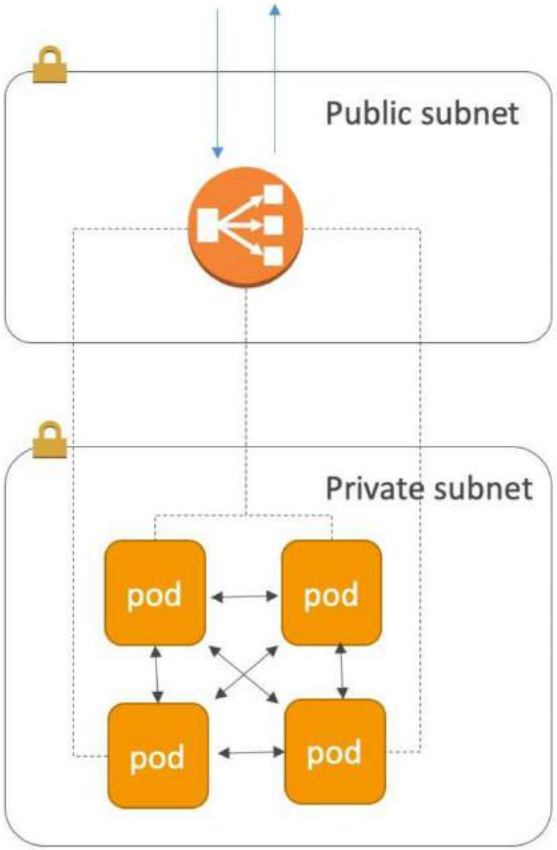
EKS Control Plane Deep Dive

- The EKS Kubernetes Control plane is highly available
- Single tenant (you do not share it with other customers)
- Made of native AWS components (EC2, ELB, ASG, NLB, VPC).
- The whole control plane is fronted by an NLB (provides fixed IP to the control plane)



EKS Networking - VPC

- Recommended to have:
  - **Private subnets:** contains all the worker nodes to have the application deployed. Must be large CIDR
  - **Public subnets:** will contain any internet-facing load balancer to expose the applications.
- Private only means you can't expose your applications
- Public only means your worker nodes are exposed to the internet
- The VPC must have DNS hostname and DNS resolution support, otherwise nodes can't register



EKS Networking – Security Groups

- You control 2 security groups: Control Plane and Worker Nodes
- Read https://github.com/freach/kubernetes-security-best-practice/blob/master/README.md#firewall-ports-fire

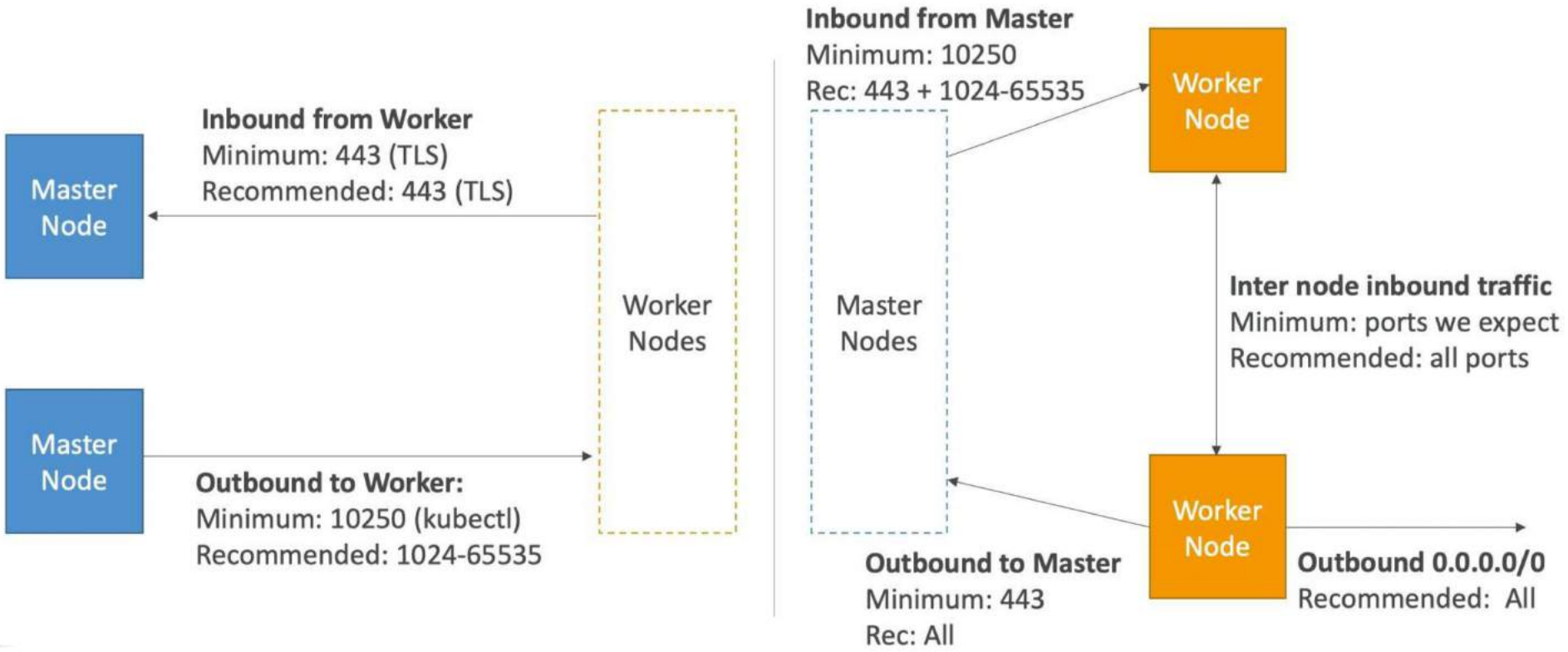
Control Plane Security Group

	Protocol	Port Range	Source	Destination
Minimum inbound traffic	TCP	443	Worker node security group	
Recommended inbound traffic	TCP	443	Worker node security group	
Minimum outbound traffic	TCP	10250		Worker node security group
Recommended outbound traffic	TCP	1025-65535		Worker node security group

Worker Node Security Group

	Protocol	Port Range	Source	Destination
Minimum inbound traffic (from other worker nodes)	Any protocol you expect your worker nodes to use for inter-worker communication	Any ports you expect your worker nodes to use for inter-worker communication	Worker node security group	
Minimum inbound traffic (from control plane)	TCP	10250	Control plane security group	
Recommended inbound traffic	All TCP	All 443, 1025-65535	Worker node security group Control plane security group	
Minimum outbound traffic*	TCP	443		Control plane security group
Recommended outbound traffic	All	All		0.0.0.0/0

Security Groups Rules Visualized



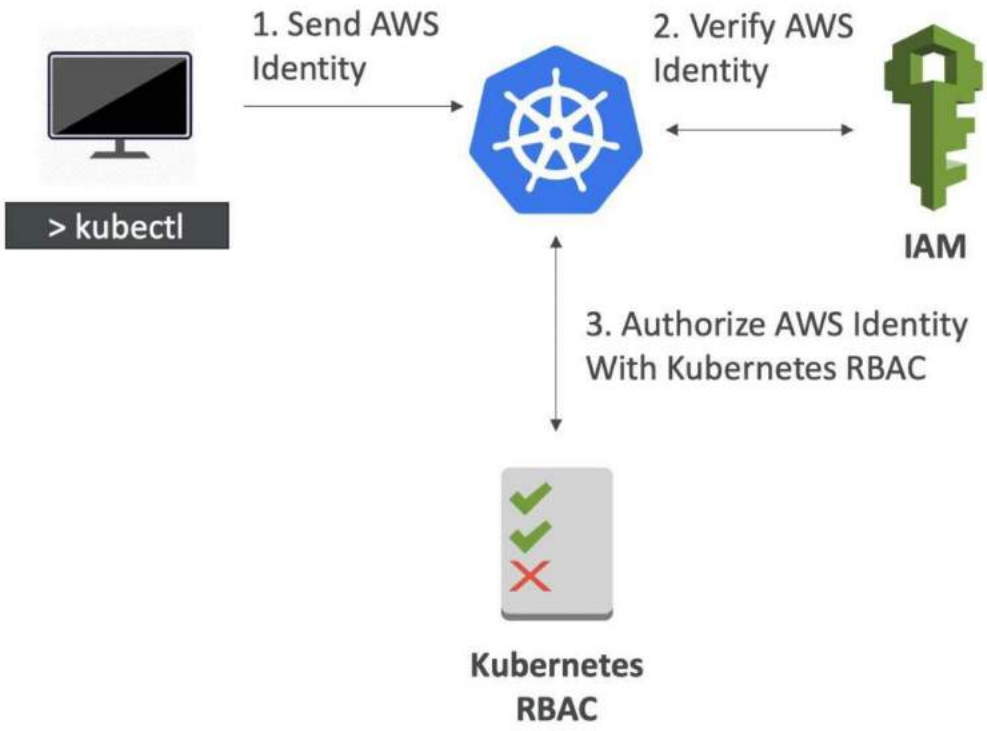
Network security with Calico (optional)

PROJECT CALICO

- Security groups allow all worker nodes to communicate to each other on any ports
- This may be a problem if you want to segment applications, tenants, or environments
- Instead of dealing with AWS Security Groups, we can install the **project Calico** onto EKS
- The network policies are directly assigned to pods (instead of worker nodes)
- We effectively reproduce what security groups but at granular pod level
- See user guide here: https://docs.aws.amazon.com/eks/latest/userguide/calico.html

Kubernetes IAM & RBAC Integration

- Authentication is held by IAM
- Authorization is done by Kubernetes RBAC (native auth for K8s)
- This is done through a collaboration done between AWS and Heptio
- You can assign RBAC directly to IAM entities!
- By default, the role you assign to your K8s cluster has system:master permissions



K8s worker nodes

- When you create a Worker node, assign an IAM role, and authorize that role in RBAC to join system:bootstrappers and system:nodes in your ConfigMap

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: aws-auth
  namespace: kube-system
data:
  mapRoles: |
    - rolearn: <ARN of instance role (not instance profile)>
      username: system:node:{{EC2PrivateDNSName}}
      groups:
        - system:bootstrappers
        - system:nodes
```

LoadBalancer

- Through the service of type `LoadBalancer`, EKS will create a…:
  - Classic Load Balancer by default
  - Network Load Balancer if this is specified:
    service.beta.kubernetes.io/aws-load-balancer-type: nlb
- There's also support for internal load balancers:
  service.beta.kubernetes.io/aws-load-balancer-internal: 0.0.0.0/0
- You can control the configuration of LBs using annotations in your manifest
- All the documentation for LoadBalancer on AWS is directly on the Kubernetes project:
  https://kubernetes.io/docs/concepts/services-networking/service/#loadbalancer