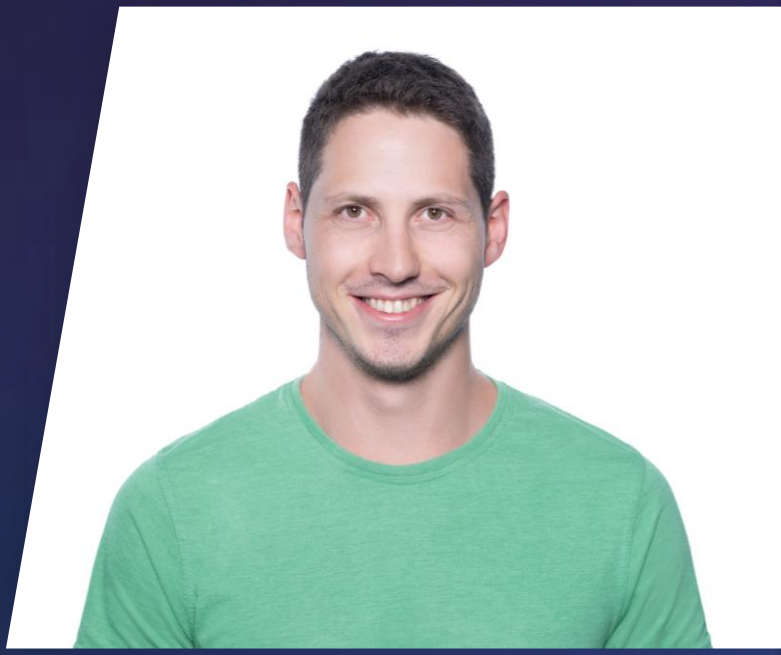
Conquering The Jungle of Kubernetes Compliance

IT-SECX
07.10.2022



Markus Gierlinger

Cloud Native Security Researcher

dynatrace

Kubernetes Security in 2022

93 %

experienced 1+ security incidents in their Kubernetes environments in the last 12 months

Source: Red Hat: 2022 state of Kubernetes security report

Agenda

Cloud Security

What is Kubernetes?

Security Concepts in Kubernetes

Compliance in Kubernetes

Introducing *KALM benchmark*

This talk is and is NOT about

- Focus only on
  - Kubernetes core resources

- Not about:
  - Plugins and 3rd-Party applications in Kubernetes
    - E.g. Service Meshes, CNI-Plugins
  - Not about compliance frameworks (HIPPA, PCI, PHI, and SOC2, etc.) in general
    - Instead it’s about compliance w.r.t. to Kubernetes specific security controls
  - Covering peculiarities for certain cloud providers

4C’s of Cloud Native Security

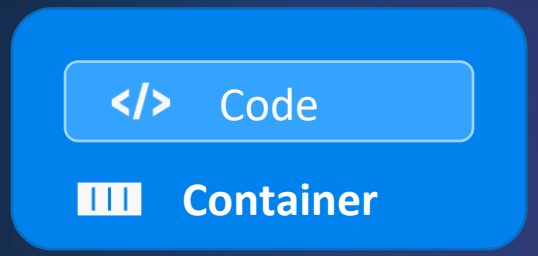
- Defense-in-depth approach
- Every “C” represents a layer for Dev/DevOps

4C’s of Cloud Native Security - Code

- Primary attack surface
- Shift to micro-services
- Key issues:
  - Insecure code
  - Supply chain
  - Logic flaws

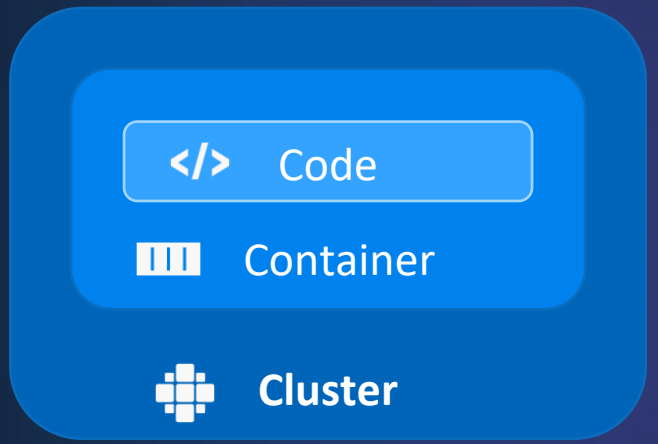
4C’s of Cloud Native Security - Container

- Clusters use Container Runtime engines
- Packaged apps and micro-services
- Key issues:
  - Vulnerabilities
  - Supply chain
  - Misconfiguration



4C’s of Cloud Native Security - Cluster

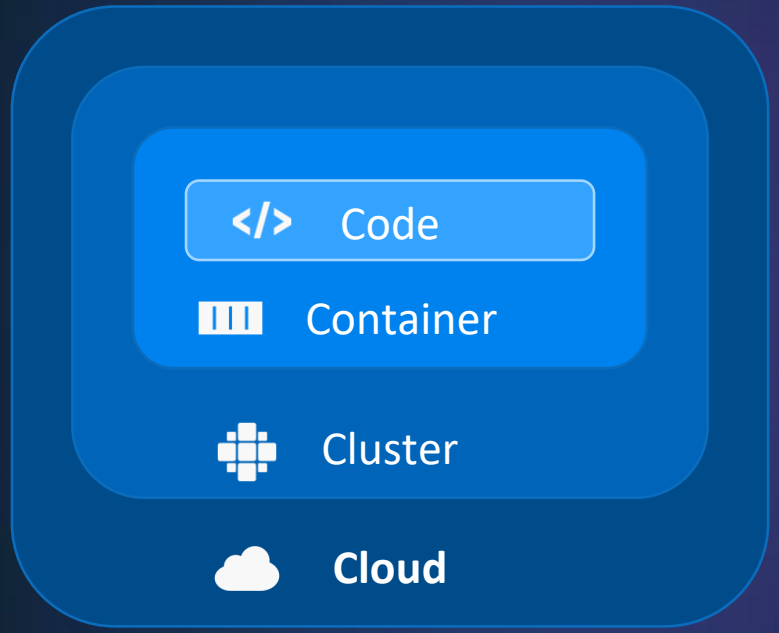
- Usually Kubernetes
- Focus areas: [1]
  - Securing cluster components
  - Securing workloads in the cluster



[1] Kubernetes: Overview of Cloud Native Security - Cluster

4C’s of Cloud Native Security - Cloud

- Physical infrastructure that runs server
- Typically managed by Cloud Service Providers (CSP)
- Shared Responsibility Model
- Key Issues: [1]
  - Misconfigurations
  - Automation loopholes



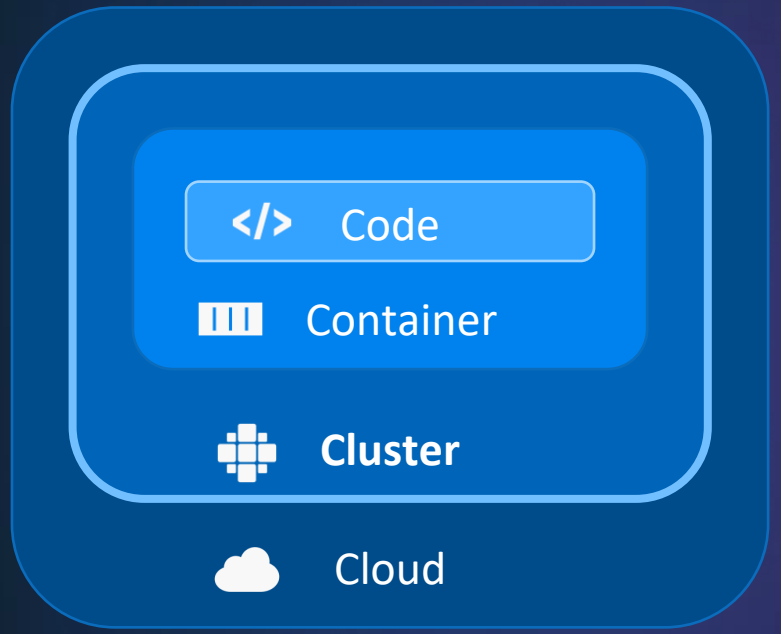
[1] Container Journal - The Four Cs of Cloud-Native Security

Cloud - Shared Responsibility

Source: [CIS: Shared Responsibility for Cloud Security: What You Need to Know]

4C’s of Cloud Native Security

- Defense-in-depth approach
- Every “C” represents a layer for Dev/DevOps



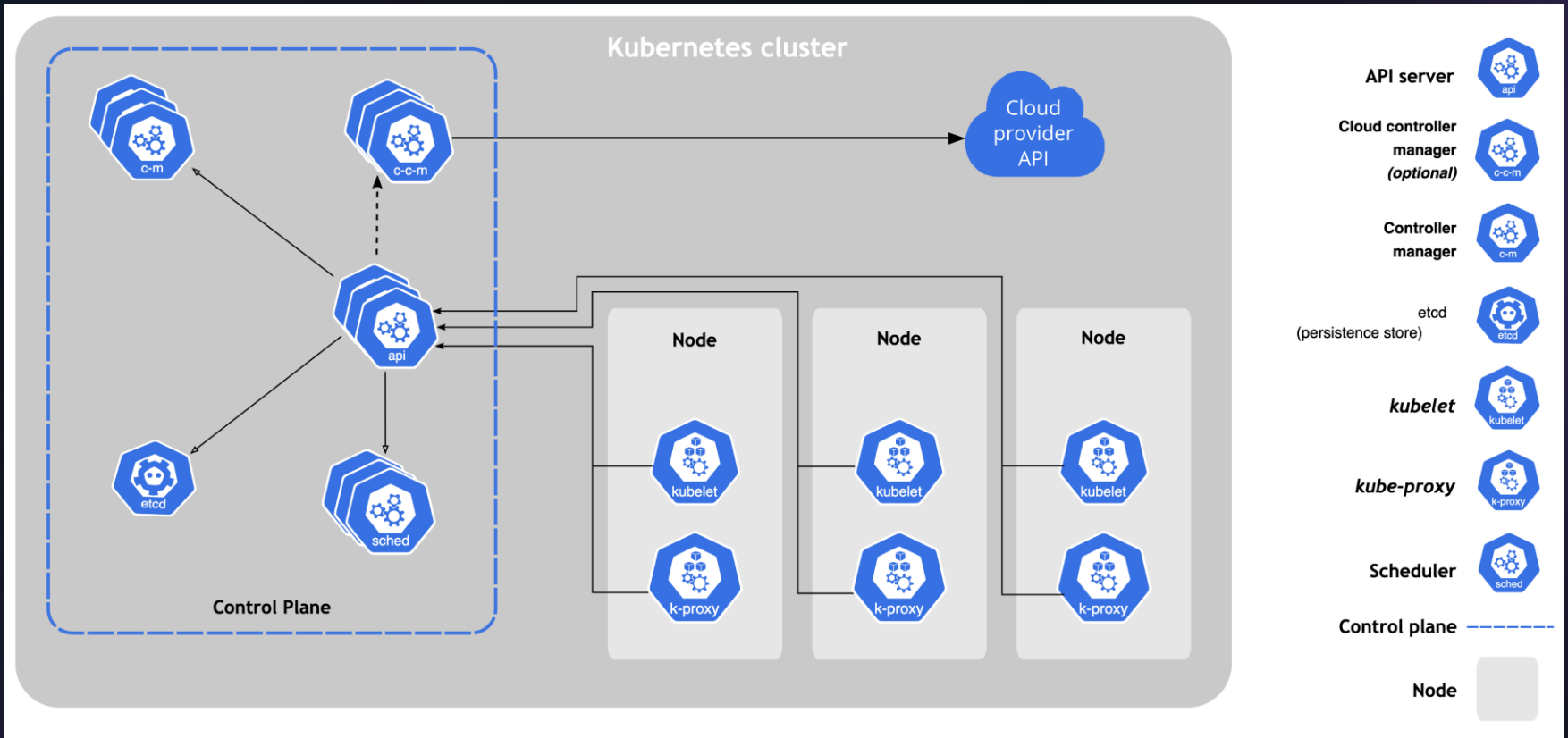
Kubernetes

What is Kubernetes?

- Platform for managing containerized apps and services
- Open sourced by Google in 2014
- “OS” of the cloud
  - Storage orchestration
  - Automatic bin packing
  - Self-healing
  - Service discovery and load balancing
  - Automated rollouts and rollbacks
  - Secret and configuration management

Kubernetes Architecture

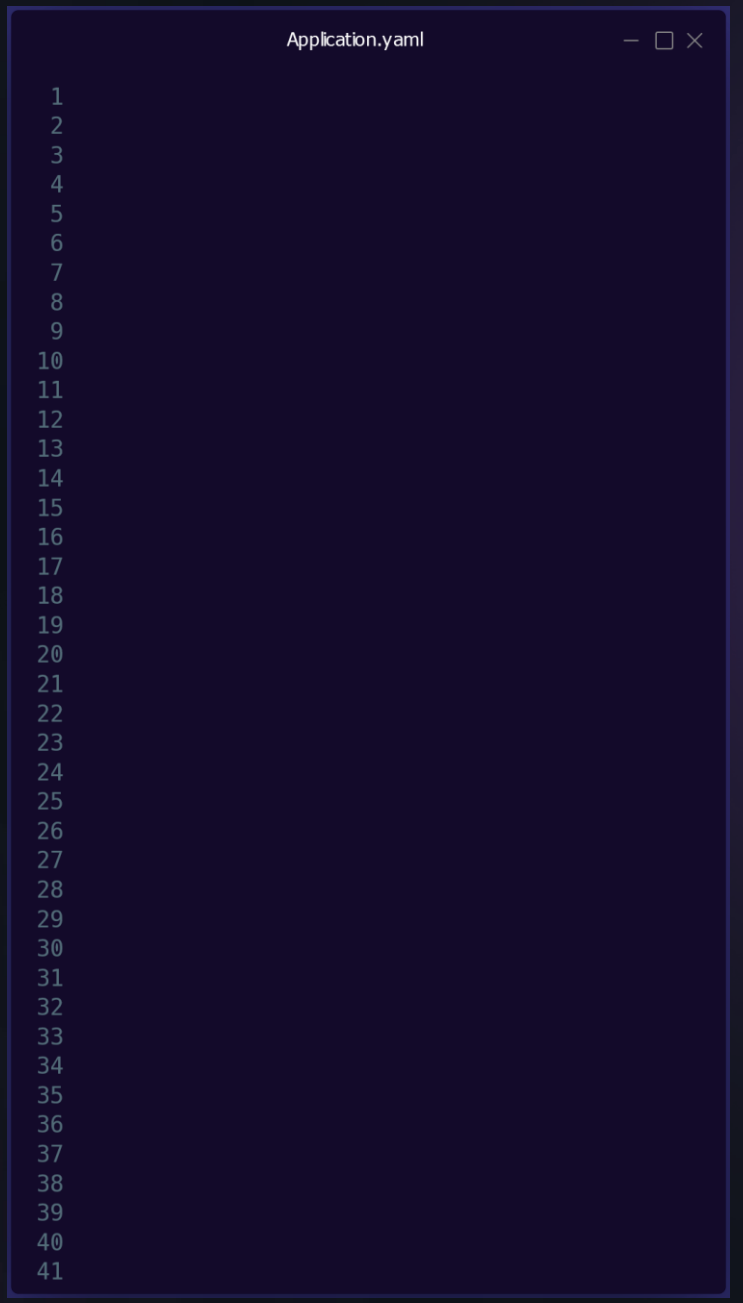
- Control Plane
  - API Server
  - ETCD
  - Controllers
- Data Plane
  - Kubelet
  - Kube-proxy



Source: [Kubernetes Components]

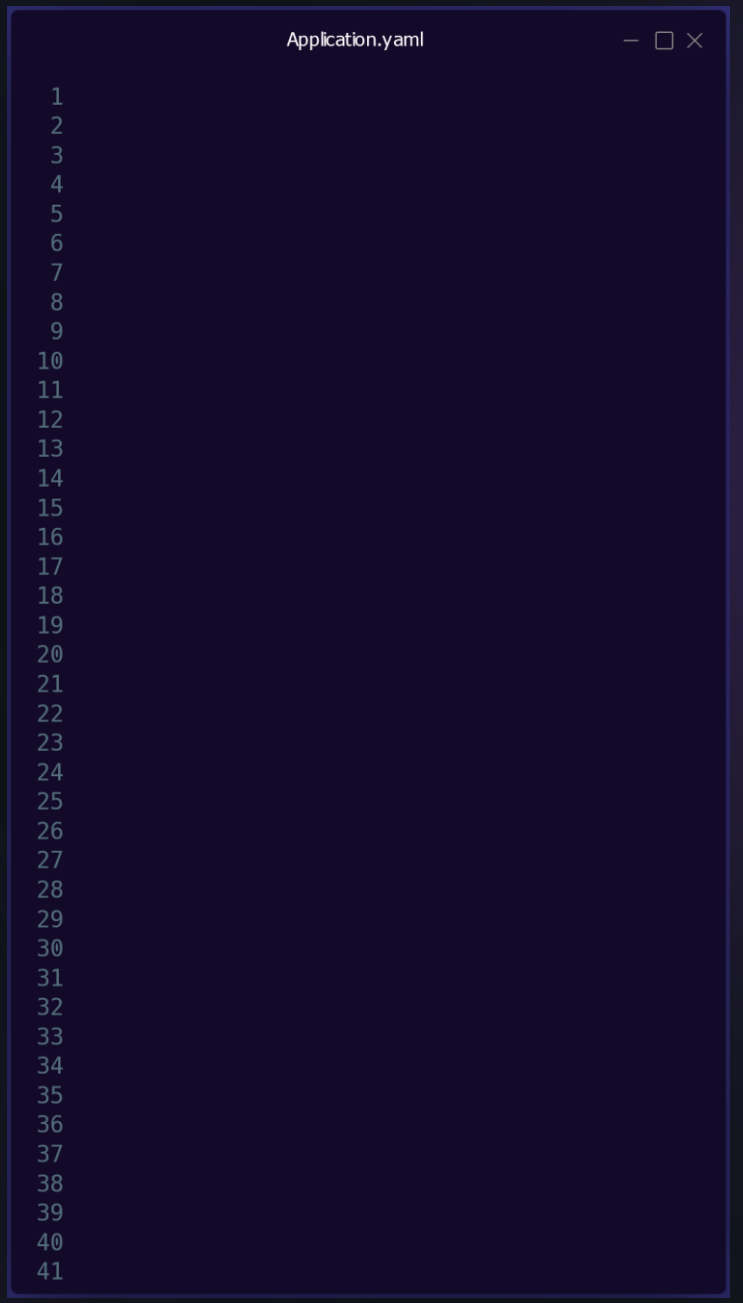
Working with Kubernetes

- Infrastructure as Code (IaC)



Working with Kubernetes

- Infrastructure as Code (IaC)
- All resources are Kubernetes objects



Working with Kubernetes

- Infrastructure as Code (IaC)
- All resources are Kubernetes objects

Application.yaml

```yaml
apiVersion: v1
kind: Namespace
metadata:
  name: my-namespace
  labels:
    pod-security.kubernetes.io/warn: restricted
```

Working with Kubernetes

- Infrastructure as Code (IaC)
- All resources are Kubernetes objects

Application.yaml

```yaml
apiVersion: v1
kind: Namespace
metadata:
  name: my-namespace
  labels:
    pod-security.kubernetes.io/warn: restricted
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-nginx
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.14.2
        ports:
        - containerPort: 80
```

Working with Kubernetes

- Infrastructure as Code (IaC)
- All resources are Kubernetes objects

Application.yaml

```yaml
apiVersion: v1
kind: Namespace
metadata:
  name: my-namespace
  labels:
    pod-security.kubernetes.io/warn: restricted
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-nginx
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.14.2
        ports:
        - containerPort: 80
```

Pod template

Working with Kubernetes

- Infrastructure as Code (IaC)
- All resources are Kubernetes objects

Application.yaml

```yaml
apiVersion: v1
kind: Namespace
metadata:
  name: my-namespace
  labels:
    pod-security.kubernetes.io/warn: restricted
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-nginx
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.14.2
        ports:
        - containerPort: 80
---
apiVersion: v1
kind: Service
metadata:
  name: my-nginx-svc
  labels:
    app: nginx
spec:
  type: LoadBalancer
  ports:
  - port: 80
  selector:
    app: nginx
```

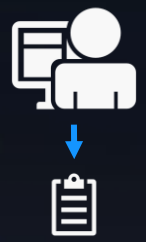
Working with Kubernetes

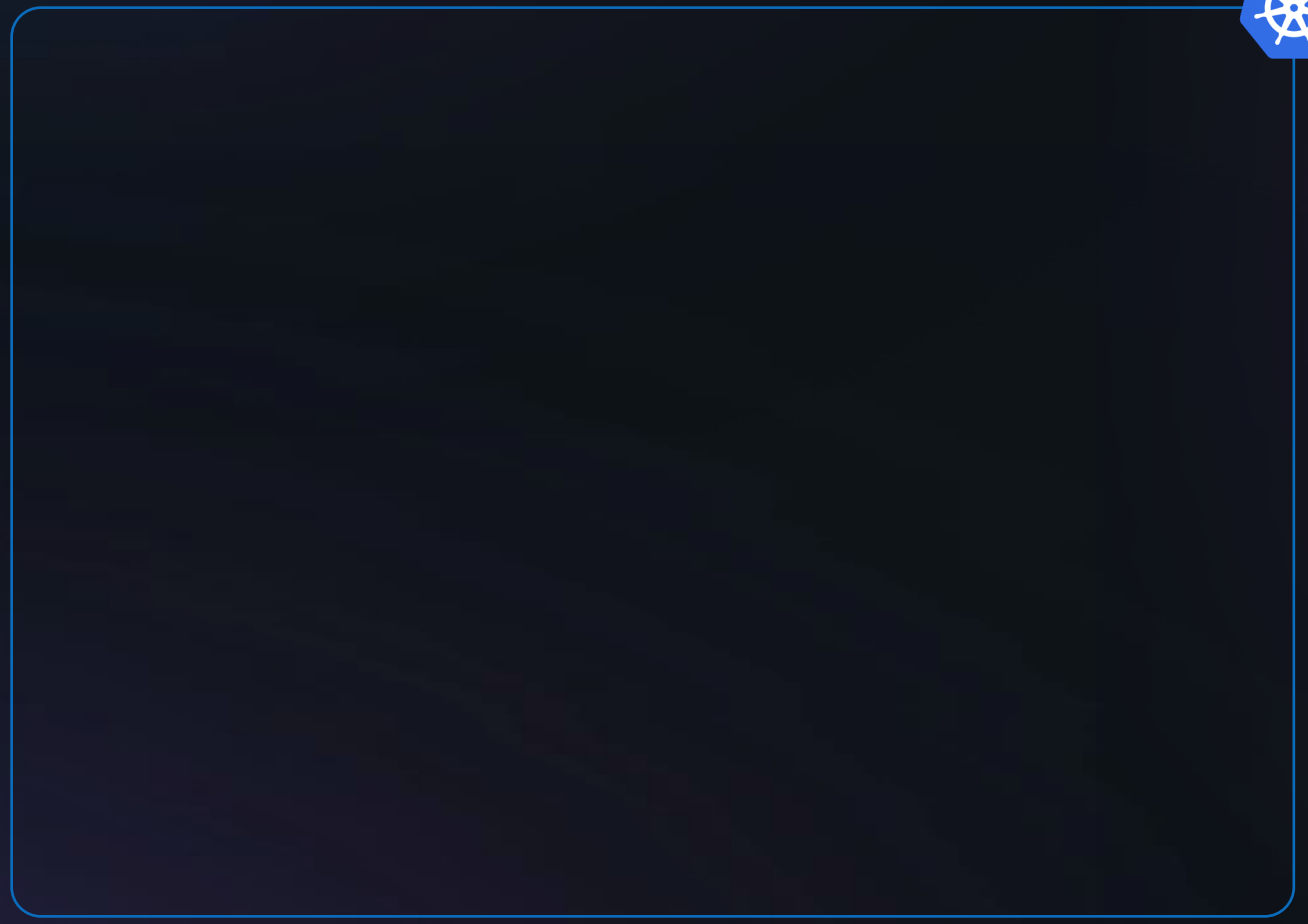
- Infrastructure as Code (IaC)
- All resources are Kubernetes objects
- Kubernetes takes care of creating and maintaining the desired state

Application.yaml

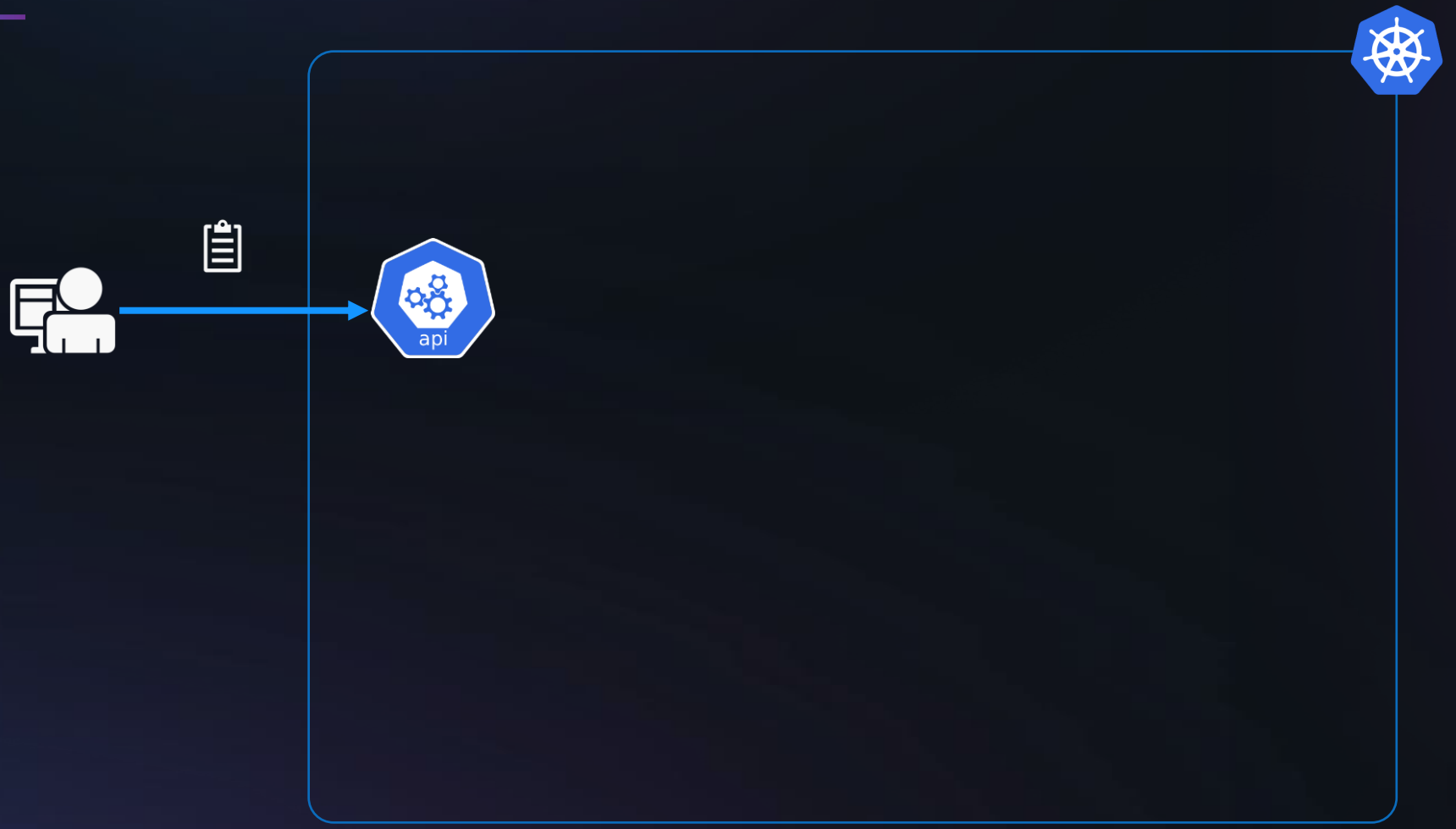
```yaml
apiVersion: v1
kind: Namespace
metadata:
  name: my-namespace
  labels:
    pod-security.kubernetes.io/warn: restricted
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-nginx
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.14.2
        ports:
        - containerPort: 80
---
apiVersion: v1
kind: Service
metadata:
  name: my-nginx-svc
  labels:
    app: nginx
spec:
  type: LoadBalancer
  ports:
  - port: 80
  selector:
    app: nginx
```

Deploying Workloads

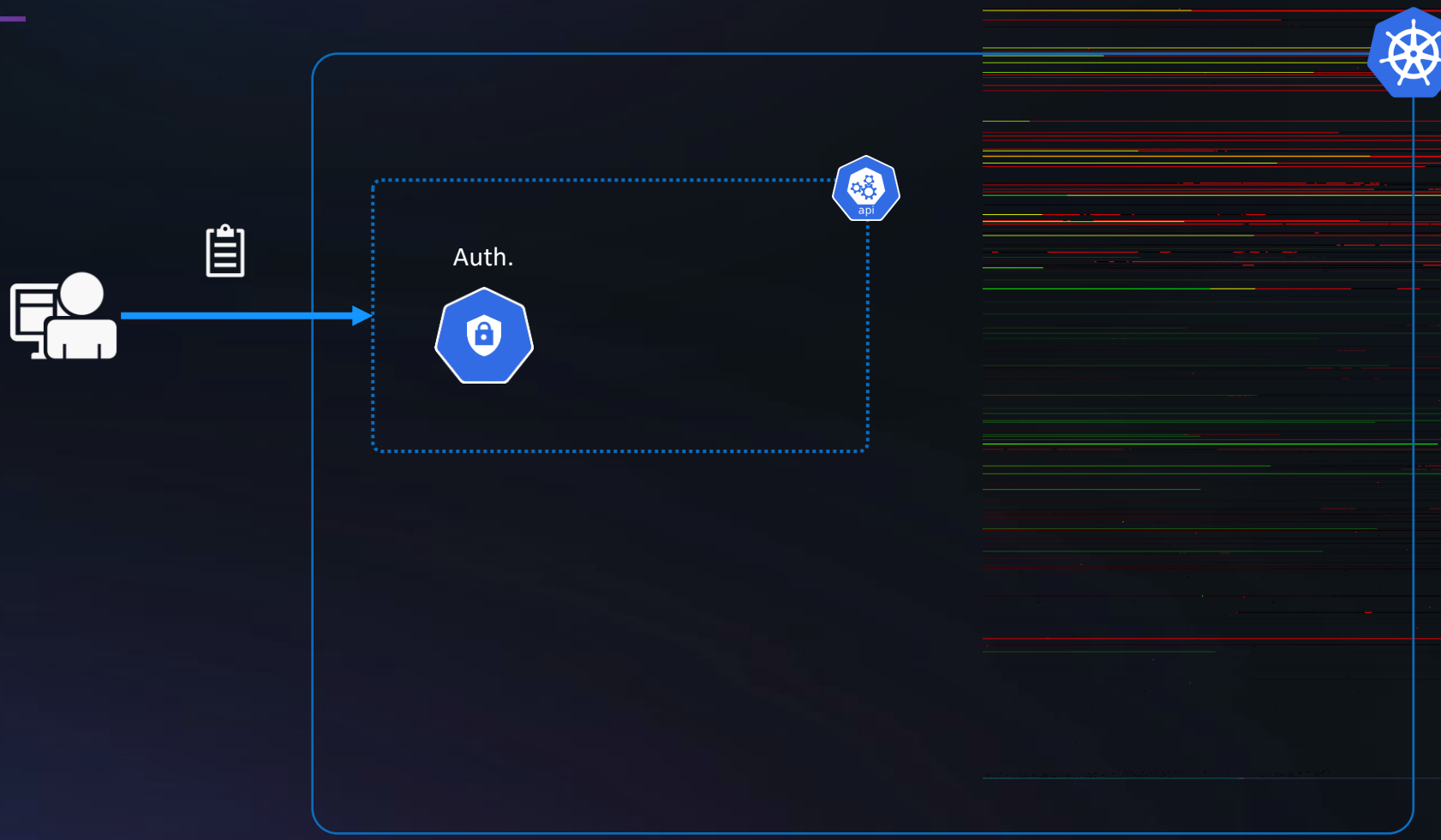




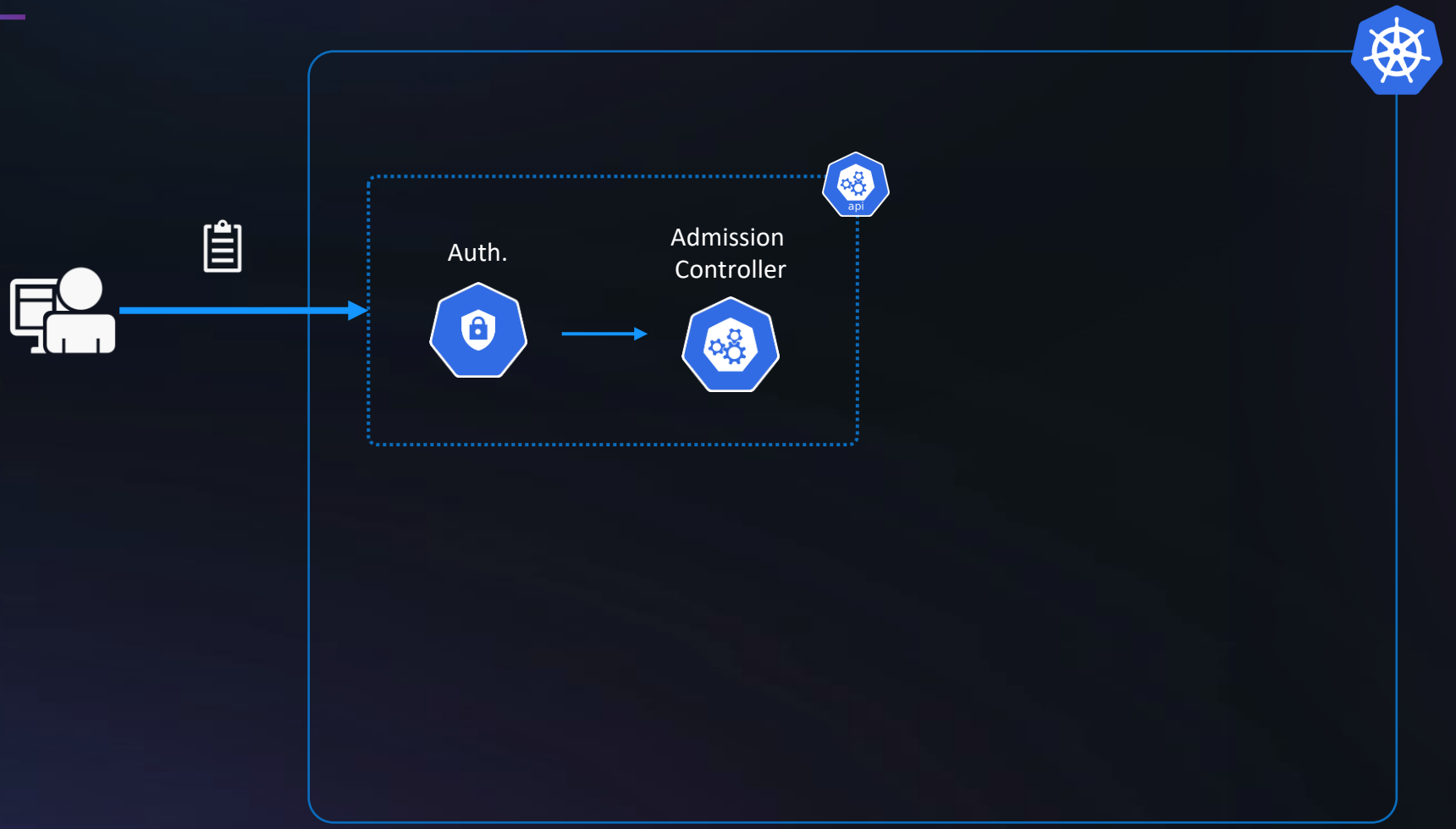
Deploying Workloads



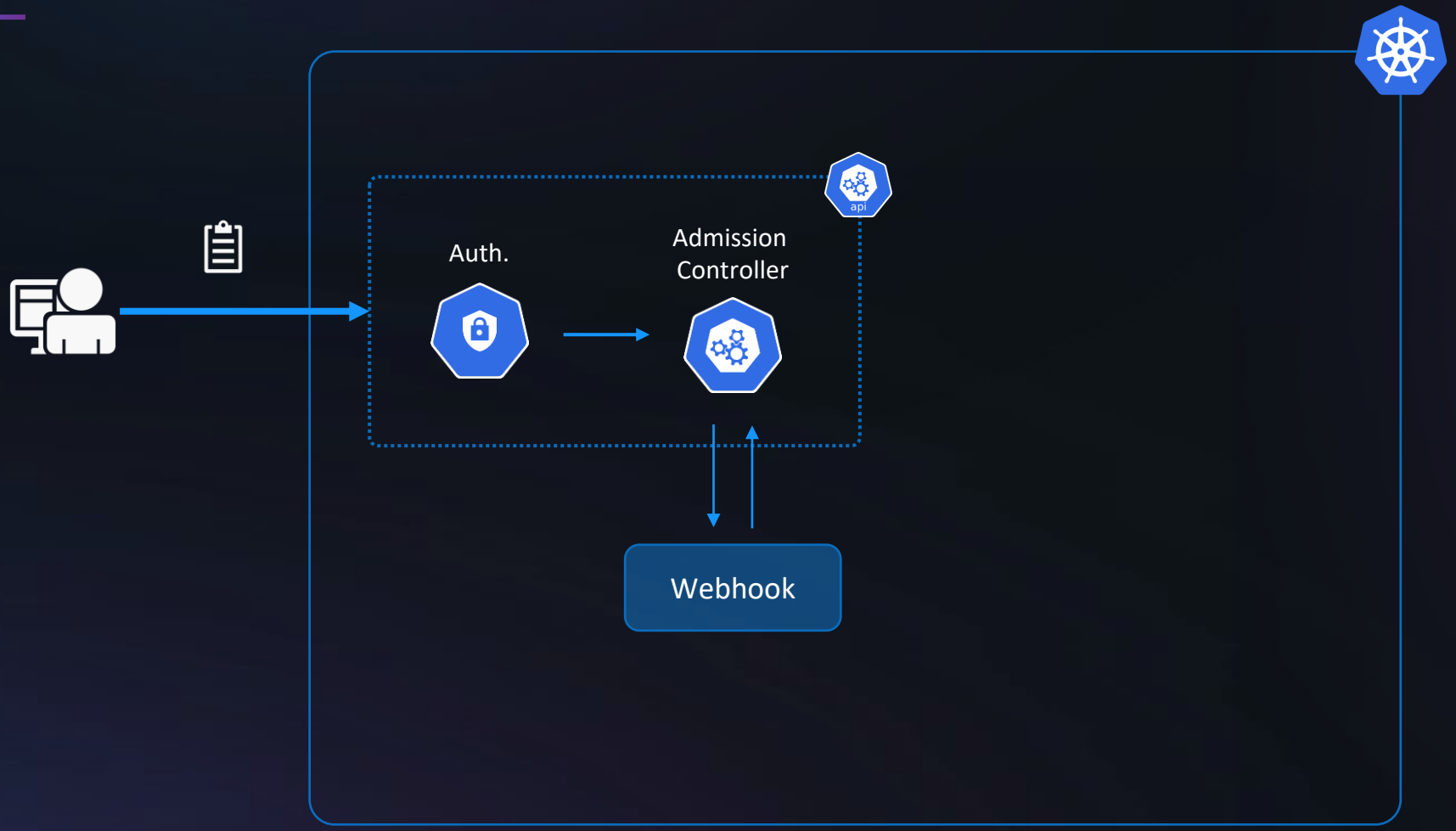
Deploying Workloads



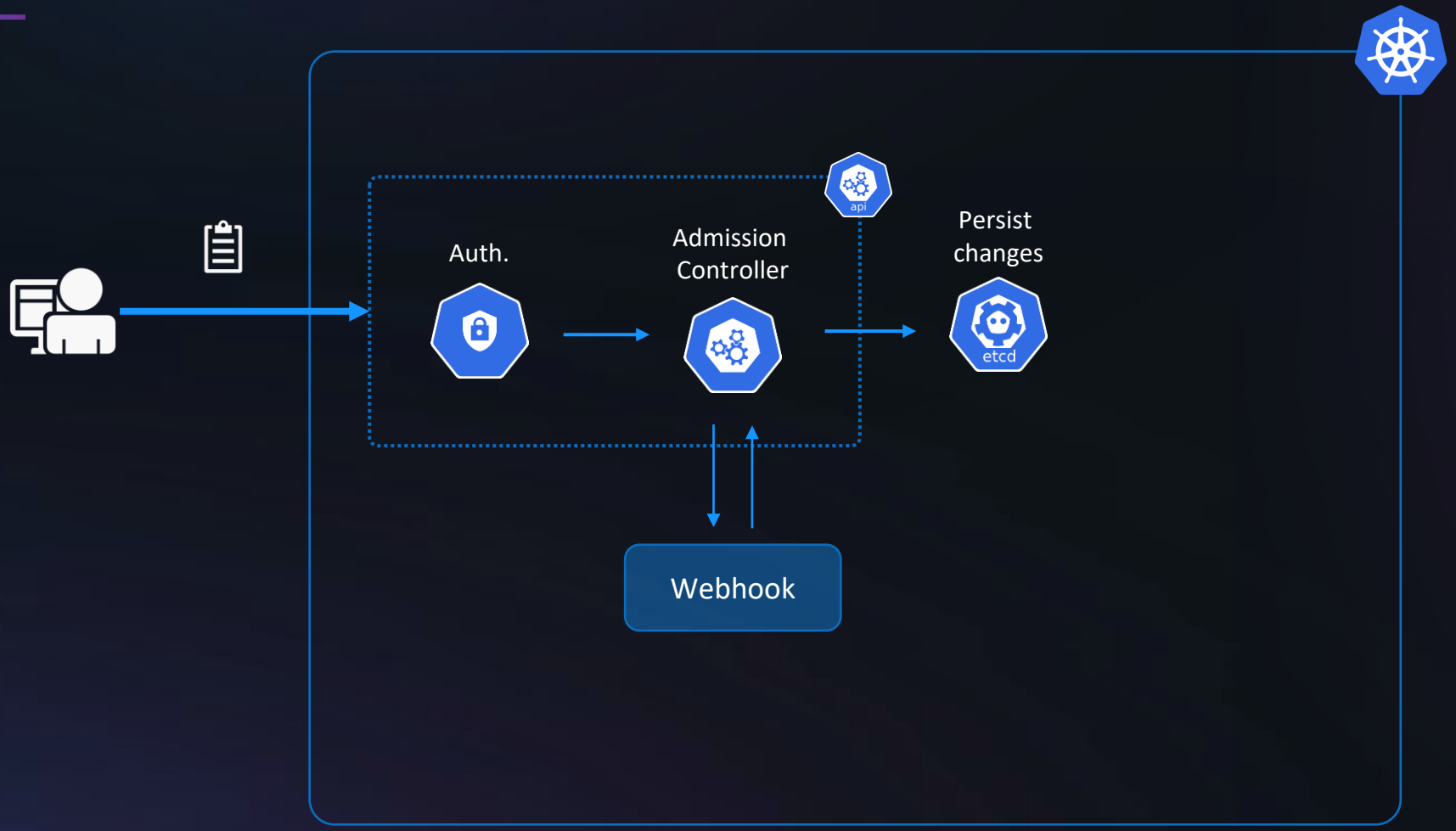
Deploying Workloads



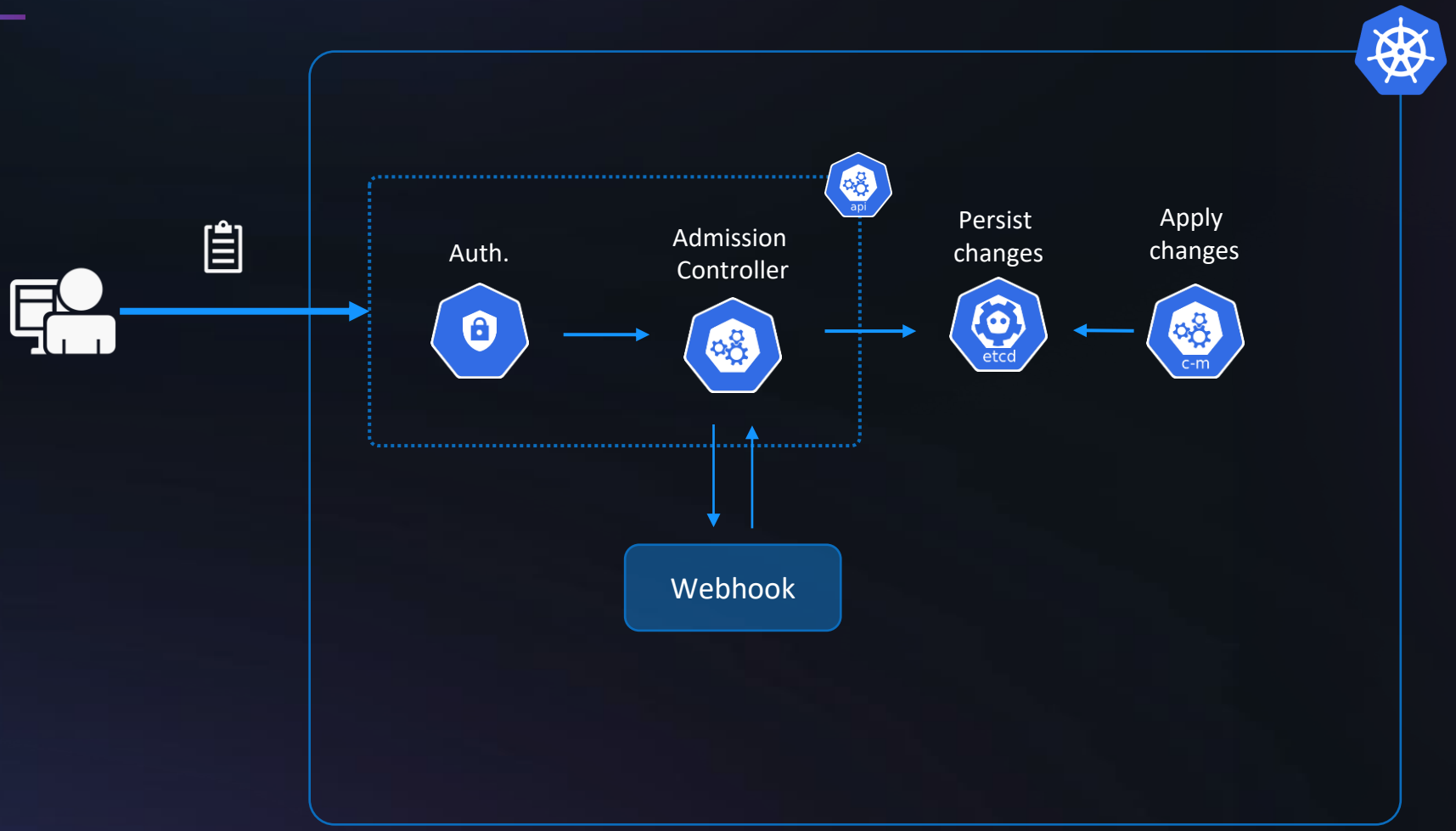
Deploying Workloads



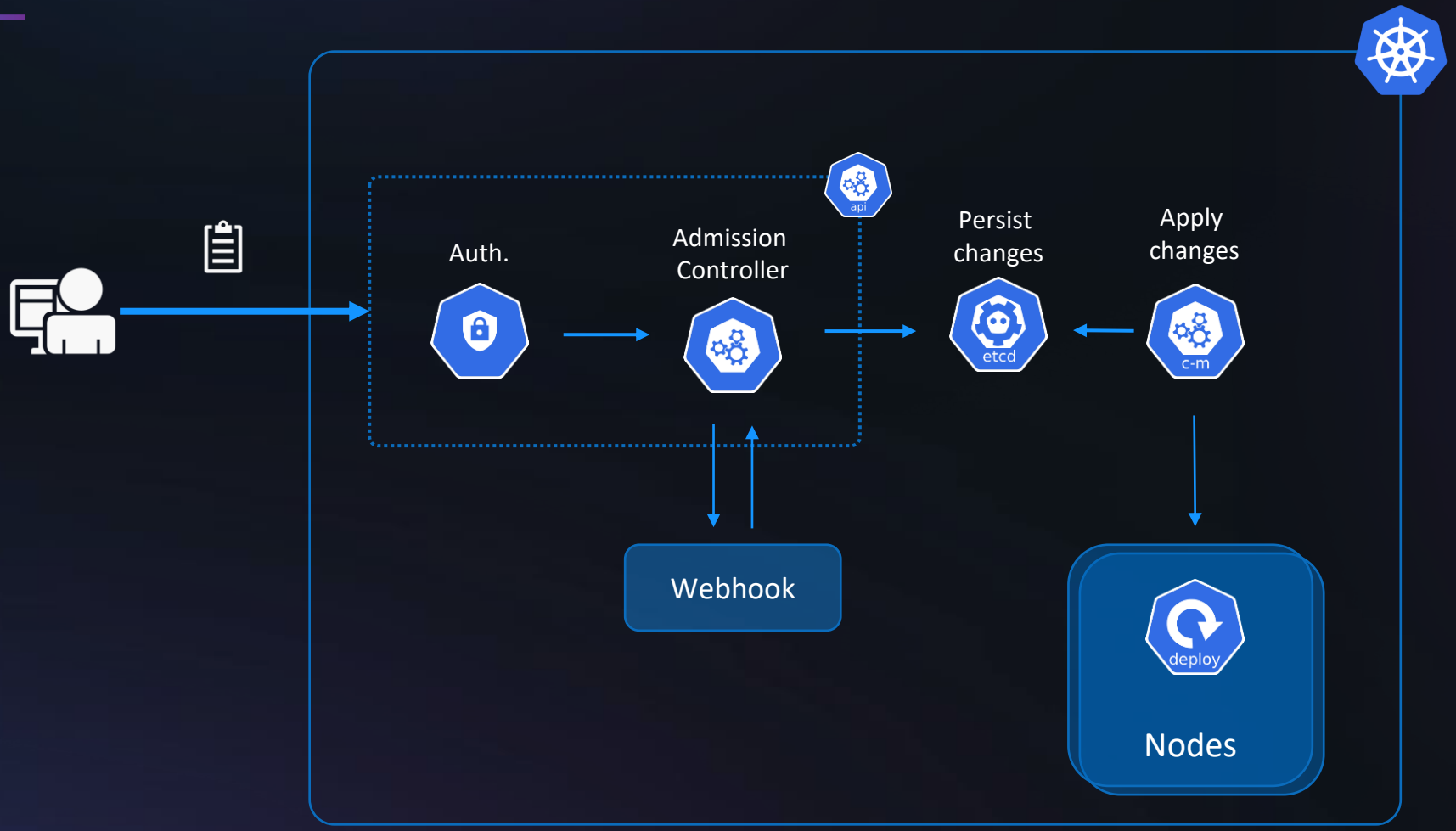
Deploying Workloads



Deploying Workloads



Deploying Workloads



Kubernetes Security

OWASP Kubernetes Top 10

1. Insecure Workload Configuration
2. Supply Chain Vulnerabilities
3. Overly Permissive RBAC Configurations
4. Lack of Centralized Policy Enforcement
5. Inadequate Logging and Monitoring
6. Broken Authentication Mechanisms
7. Missing Network Segmentation Controls
8. Secrets Management Failures
9. Misconfigured Cluster Components
10. Outdated and Vulnerable Kubernetes Components

Source: https://github.com/OWASP/www-project-kubernetes-top-ten (09.2022)

Areas of Kubernetes Security

- Authentication & Authorization
- Pod Security
- Networking
- Workload Configuration
- Supply Chain

Authentication & Authorization

- Role-based access control (RBAC)
- Apply Principle of Least Privilege
- 3 types of subjects
  - User, Group (external)
  - ServiceAccount (internal)

Authentication & Authorization

- Role-based access control (RBAC)
- Apply Principle of Least Privilege
- 3 types of subjects
  - User, Group (external)
  - ServiceAccount (internal)

Pod.yaml

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: read-secrets
  namespace: development
subjects:
- kind: User
  name: dave
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: ClusterRole
  name: secret-reader
  apiGroup: rbac.authorization.k8s.io
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: secret-reader
rules:
- apiGroups: [""]
  resources: ["secrets"]
  verbs: ["get", "watch", "list"]

```

Authentication & Authorization

- Role-based access control (RBAC)
- Apply Principle of Least Privilege
- 3 types of subjects
  - User, Group (external)
  - ServiceAccount (internal)

Pod.yaml

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: read-secrets
  namespace: development
subjects:
- kind: User
  name: dave
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: ClusterRole
  name: secret-reader
  apiGroup: rbac.authorization.k8s.io
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: secret-reader
rules:
- apiGroups: [""]
  resources: ["secrets"]
  verbs: ["get", "watch", "list"]

```

Authentication & Authorization

- Role-based access control (RBAC)
- Apply Principle of Least Privilege
- 3 types of subjects
  - User, Group (external)
  - ServiceAccount (internal)
- Authorization based on
  - API groups
  - Resources
  - Verbs

Pod.yaml

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: read-secrets
  namespace: development
subjects:
- kind: User
  name: dave
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: ClusterRole
  name: secret-reader
  apiGroup: rbac.authorization.k8s.io
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: secret-reader
rules:
- apiGroups: [""]
  resources: ["secrets"]
  verbs: ["get", "watch", "list"]

```

Pod Security Context

- Configuration for privileges and capabilities of pods/ containers

Pod.yaml

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
spec:
  securityContext:
    runAsUser: 1000
    readOnlyRootFilesystem: true
  containers:
  - name:
    image: nginx:latest
    securityContext:
      runAsUser: 2000
      allowPrivilegeEscalation: false
  hostIPC: false
  hostNetwork: false
  hostPID: false
```

Pod Security Context

- Configuration for privileges and capabilities of pods/ containers

Pod.yaml

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
spec:
  securityContext:
    runAsUser: 1000
    readOnlyRootFilesystem: true
  containers:
  - name:
    image: nginx:latest
    securityContext:
      runAsUser: 2000
      allowPrivilegeEscalation: false
  hostIPC: false
  hostNetwork: false
  hostPID: false
```

Pod Security Context

- Configuration for privileges and capabilities of pods/ containers

Pod.yaml

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
spec:
  securityContext:
    runAsUser: 1000
    readOnlyRootFilesystem: true
  containers:
  - name:
    image: nginx:latest
    securityContext:
      runAsUser: 2000
      allowPrivilegeEscalation: false
  hostIPC: false
  hostNetwork: false
  hostPID: false
```

Pod Security Context

- Configuration for privileges and capabilities of pods/ containers

Pod.yaml

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
spec:
  securityContext:
    runAsUser: 1000
    readOnlyRootFilesystem: true
  containers:
  - name:
    image: nginx:latest
    securityContext:
      runAsUser: 2000
      allowPrivilegeEscalation: false
  hostIPC: false
  hostNetwork: false
  hostPID: false
```

Pod Security Context

- Configuration for privileges and capabilities of pods/ containers

Pod.yaml

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
spec:
  securityContext:
    runAsUser: 1000
    readOnlyRootFilesystem: true
  containers:
  - name:
    image: nginx:latest
    securityContext:
      runAsUser: 2000
      allowPrivilegeEscalation: false
  hostIPC: false
  hostNetwork: false
  hostPID: false
```

- Pod Security Policy (PSP) replaced by Pod Security Standards (PSS)
  - Are a set of best-practice profiles for running pods securely [1]
  - are applied at the *Namespace* level when pods are created/modified

Application.yaml

```yaml
apiVersion: v1
kind: Namespace
metadata:
  name: my-namespace
  labels:
    pod-security.kubernetes.io/warn: restricted

```

Network Security

- Kubernetes uses Container Network Interface (CNI) to create virtual networks

Network Security

- Kubernetes uses Container Network Interface (CNI) to create virtual networks
- Default: all pods can communicate with each other

Network Security

- Kubernetes uses Container Network Interface (CNI) to create virtual networks
- Default: all pods can communicate with each other
- Namespaces are **not** a separation mechanism!

Network Security

- Kubernetes uses Container Network Interface (CNI) to create virtual networks
- Default: all pods can communicate with each other
- Namespaces are **not** a separation mechanism!
- Network Policies
  - Define rules how pods can communicate
  - Require support by CNI plugin

Calico

Network Security

- Kubernetes uses Container Network Interface (CNI) to create virtual networks
- Default: all pods can communicate with each other
- Namespaces are **not** a separation mechanism!
- Network Policies
  - Define rules how pods can communicate
  - Require support by CNI plugin

Calico

- Service Meshes

Istio

Workload Security

- Generic configurations

Pod.yaml

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: my-app
spec:
  automountServiceAccountToken: false
  serviceAccountName: dedicated-service-account
  containers:
    - image: nginx:latest
      name: app
      ports:
        - containerPort: 8080
```

[1] Kubernetes - Recommended Labels

Workload Security

- Generic configurations
- Secrets Management
  - Not in environment variables!
  - ConfigMaps are not secure

Pod.yaml

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: my-app
spec:
  automountServiceAccountToken: false
  serviceAccountName: dedicated-service-account
  env:
  - name: SECRET_PASSWORD
    valueFrom:
      secretKeyRef:
        name: mysecret
        key: password
  containers:
    - image: nginx:latest
      name: app
      ports:
        - containerPort: 8080
```

[1] Kubernetes - Recommended Labels

Workload Security

- Generic configurations
- Secrets Management
  - Not in environment variables!
  - ConfigMaps are not secure
- Use of labels/annotations [1]
  - Configuration of some features
  - Useful semantic information

Pod.yaml

```yaml
apiVersion: v1
kind: Pod
metadata:
  labels:
    app.kubernetes.io/part-of: webshop
    app.kubernetes.io/managed-by: owner
  name: my-app
spec:
  automountServiceAccountToken: false
  serviceAccountName: dedicated-service-account
  env:
  - name: SECRET_PASSWORD
    valueFrom:
      secretKeyRef:
        name: mysecret
        key: password
  containers:
    - image: nginx:latest
      name: app
      ports:
        - containerPort: 8080
```

[1] Kubernetes - Recommended Labels

Workload Security

- Generic configurations
- Secrets Management
  - Not in environment variables!
  - ConfigMaps are not secure
- Use of labels/annotations [1]
  - Configuration of some features
  - Useful semantic information
- Reliability
  - Resource Requests/ Limits
  - Readiness-/ Liveness-Probe

Pod.yaml

```yaml
apiVersion: v1
kind: Pod
metadata:
  labels:
    app.kubernetes.io/part-of: webshop
    app.kubernetes.io/managed-by: owner
  name: my-app
spec:
  automountServiceAccountToken: false
  serviceAccountName: dedicated-service-account
  env:
  - name: SECRET_PASSWORD
    valueFrom:
      secretKeyRef:
        name: mysecret
        key: password
  containers:
    - image: nginx:latest
      name: app
      ports:
        - containerPort: 8080
      readinessProbe:
        httpGet:
          path: /ready
          port: 8080
      resources:
        limits:
          cpu: 1m
          ephemeral-storage: 1Mi
          memory: 1Mi
        requests:
          cpu: 1m
          ephemeral-storage: 1Mi
          memory: 1Mi
```

[1] Kubernetes - Recommended Labels

Supply Chain

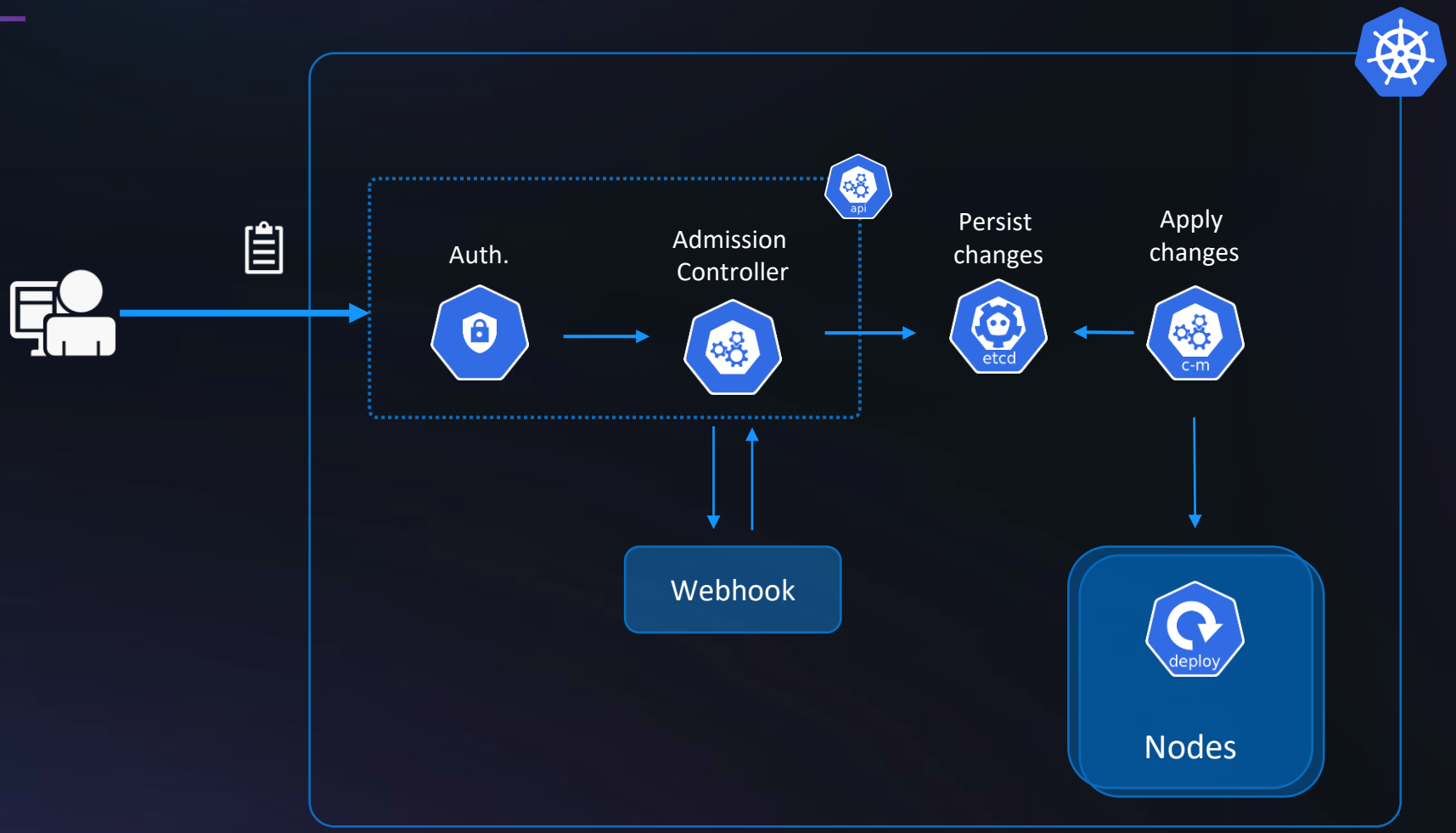
- Primarily about containers
  - Software composition (SBOM)
  - Vulnerabilities
- Use admission controller to check:
  - Limit to trusted registries
  - Image Verification (3rd Party)

cosign

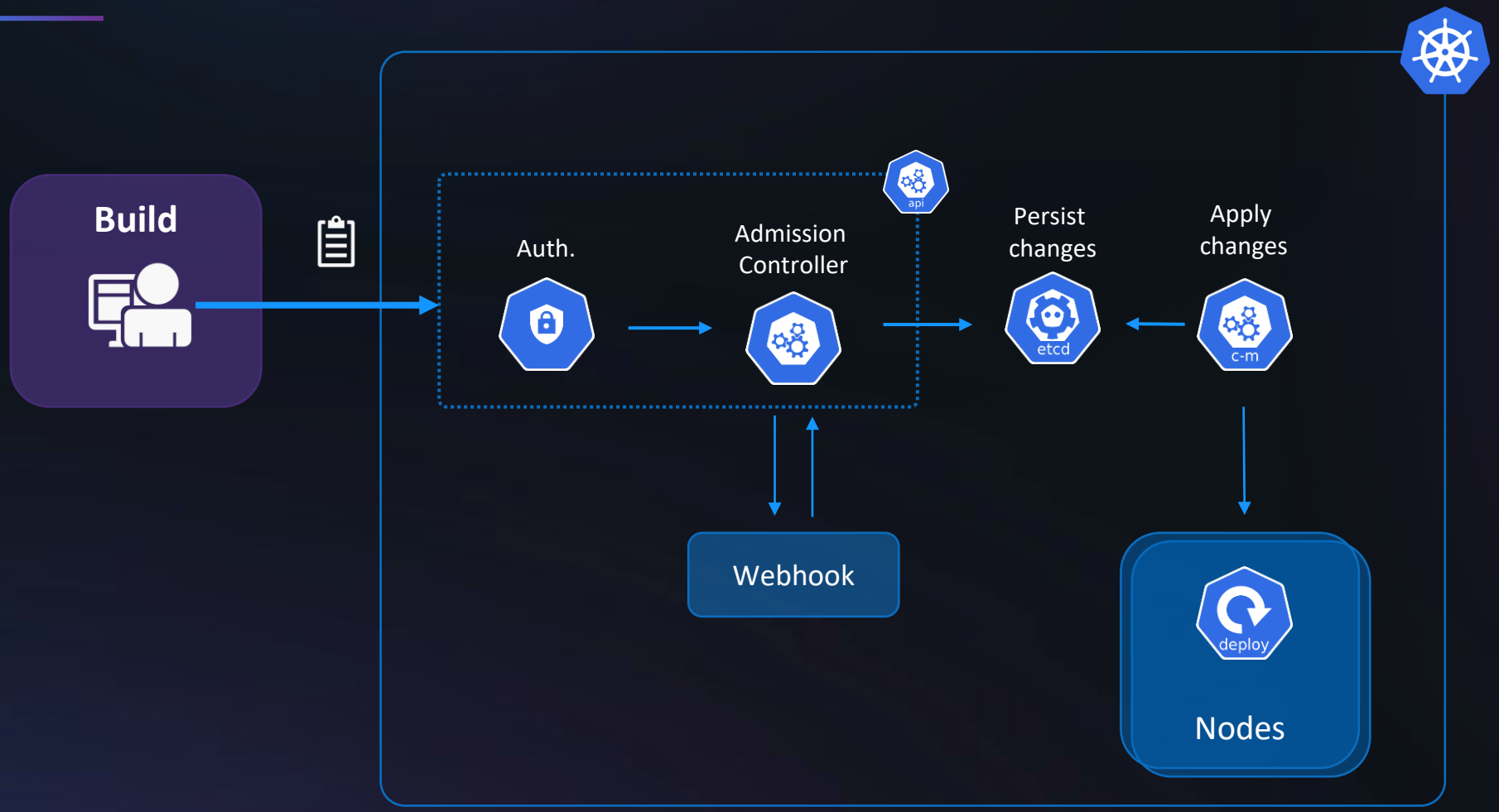
Kritis

Source: CNCF Supply Chain Security Paper (09.2022)

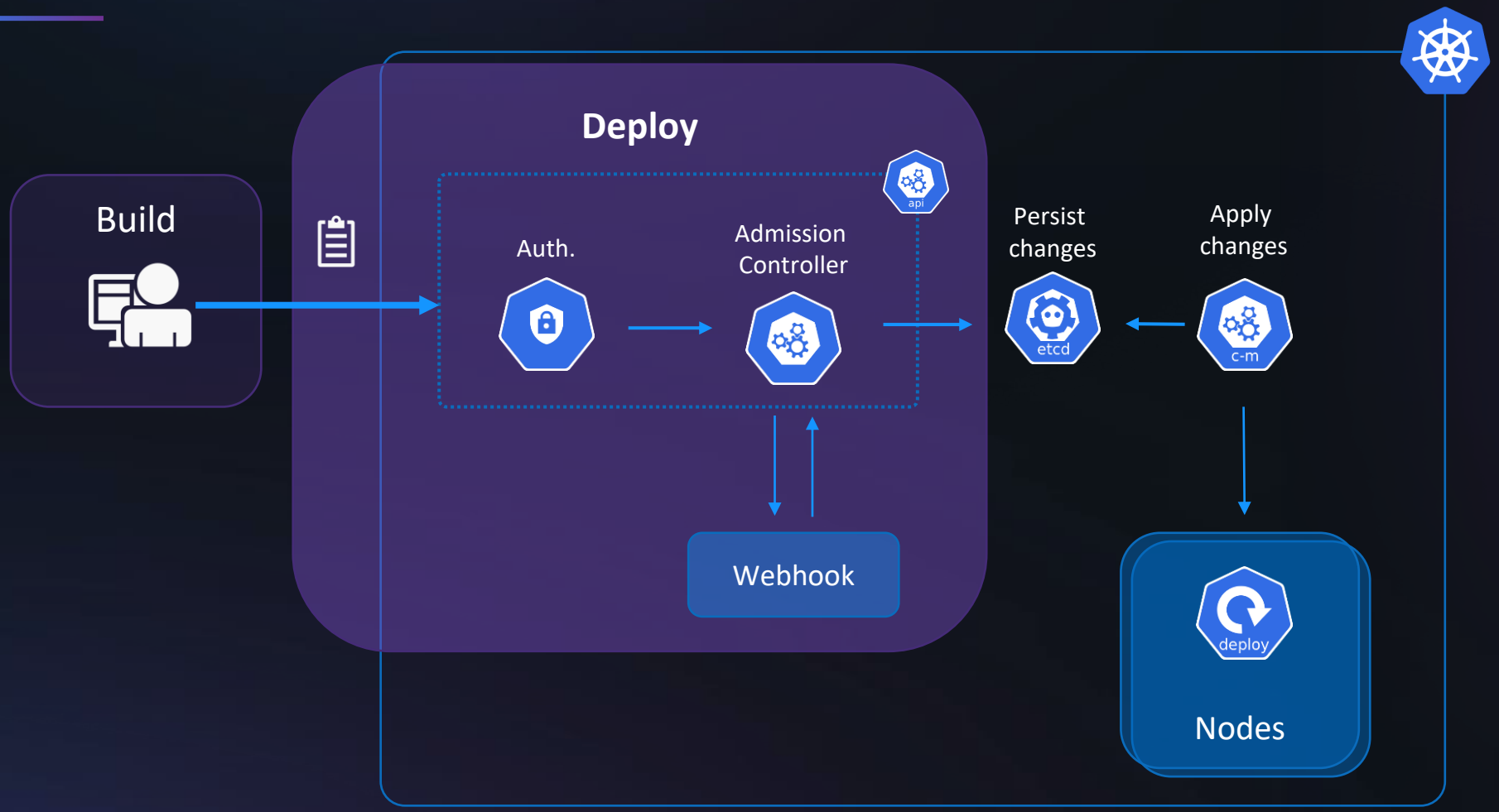
Where can security be enforced?



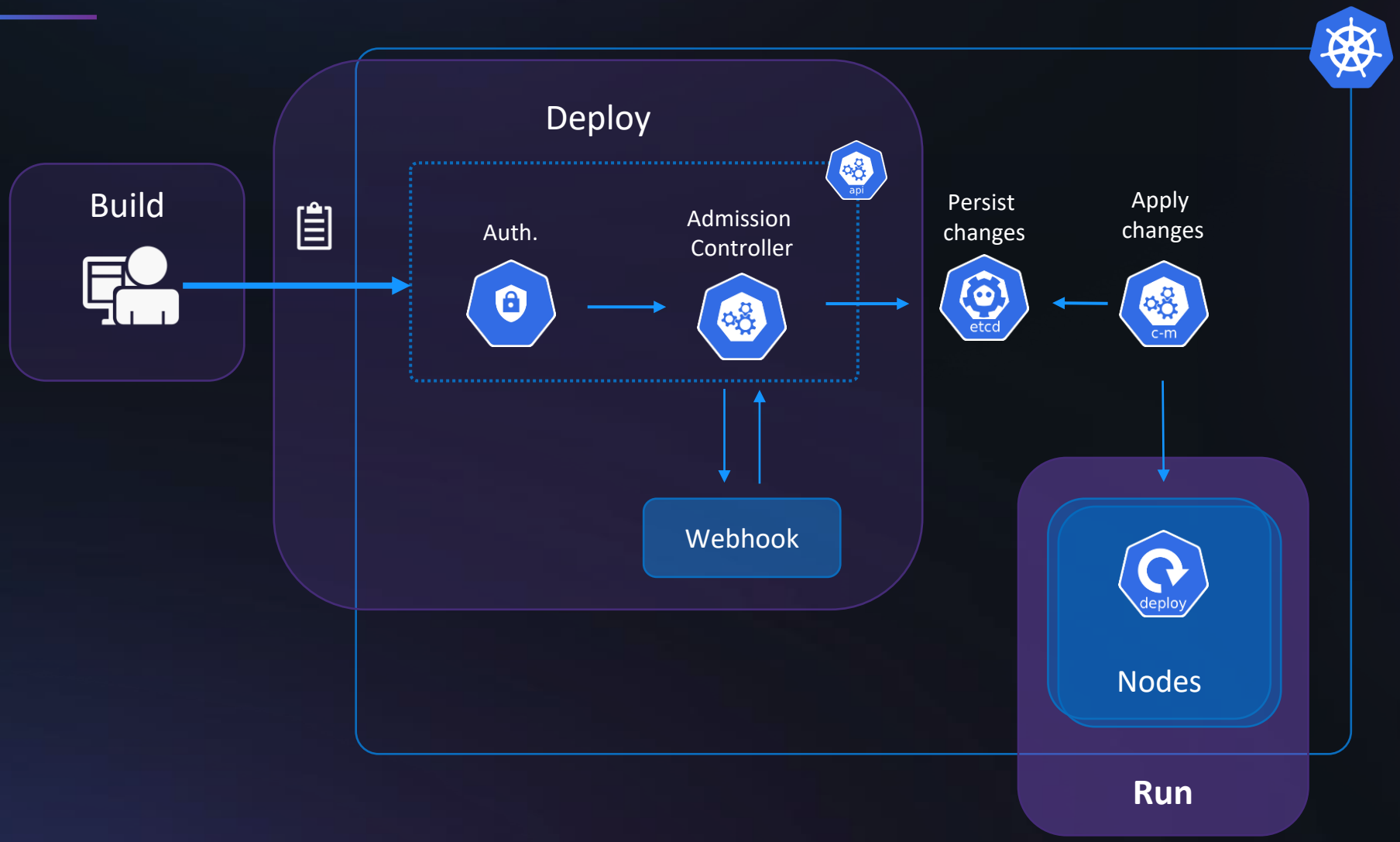
Where can security be enforced?



Where can security be enforced?



Where can security be enforced?



Problems

- Cloud environments move very fast
- Kubernetes is designed to just work
  - implies security defaults are very lenient
- Kubernetes is evolving quickly
  - 3 release per year
  - Hard to keep track of attack vectors [1]

[1] OWASP Kubernetes Security Cheat Sheet

Compliance

Compliance Frameworks

- Regulatory standards
  - NIST
  - GDPR
  - SOC2
  - PCI DSS
  - HIPAA
  - …

Compliance Frameworks

- Regulatory standards
  - NIST
  - GDPR
  - SOC2
  - PCI DSS
  - HIPAA
  - …
- Guides and Benchmarks
  - CIS Benchmarks
  - NSA / CISA Kubernetes Hardening Guide
  - Kubernetes STIG – DISA
  - Kubernetes Security Checklist

CIS Benchmarks

- Center for Internet Security
- Are configuration baselines and best practices for securely configuring a system
- Controls map to many established standards and regulatory frameworks
- Dedicated for various technologies, e.g.
  - Docker
  - Kubernetes (for various clouds)

Source: [CIS: Shared Responsibility for Cloud Security: What You Need to Know]

CIS Kubernetes Benchmark

- **1. Control Plane Components**
  - Master Node Configuration Files
  - API Server
  - Controller Manager
  - Scheduler
- **2. etcd**
- **3. Control Plane Configuration**
  - Authentication and Authorization
  - Logging
- **4. Worker Nodes**
  - Worker Node Configuration Files
  - Kubelet
- **5. Policies**
  - RBAC and Service Accounts
  - Pod Security Standards
  - Network Policies and CNI
  - Secrets Management
  - Extensible Admission Control
  - General Policies

CIS Kubernetes Benchmark

- **1. Control Plane Components**
  - Master Node Configuration Files
  - API Server
  - Controller Manager
  - Scheduler
- **2. etcd**
- **3. Control Plane Configuration**
  - Authentication and Authorization
  - Logging
- **4. Worker Nodes**
  - Worker Node Configuration Files
  - Kubelet

Infrastructure

- **5. Policies**
  - RBAC and Service Accounts
  - Pod Security Standards
  - Network Policies and CNI
  - Secrets Management
  - Extensible Admission Control
  - General Policies

Workload

Tools

Tools

Tools

kics. by Checkmarx

checkov by bridgecrew

KubeLinter

Kubescape

aqua kube-bench

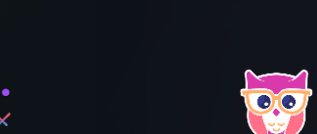
snyk

aqua trivy

Kyverno

polaris by Fairwinds

Tools



rbac-police

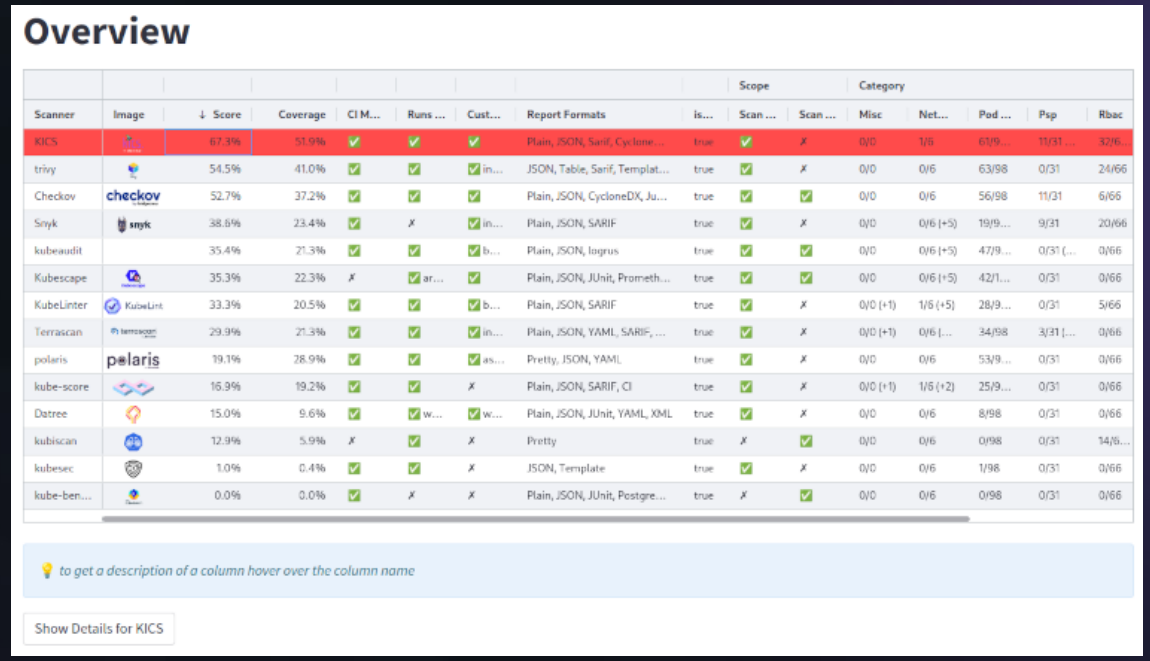
KubeLinter

Krane

KALM Benchmark

KALM Benchmark

- Open-Source project (on Github)
- Focus on workload misconfiguration
- Goal:
  - Have a benchmark for comparing tools/scanners
  - Help decision makers find the right fit
  - Help with the development/adjustment of the rule-set
- 2 parts
  - Benchmark manifests itself
  - Tool to analyze results



KALM Benchmark

- Normalized checks across benchmarks and tools
- Design
  - Best practice manifest per default
  - Only 1 explicit misconfiguration per check
- Multiple categories
- Contains > 250 manifests

- *Note: not 100% coverage*

Demo

Takeaways

Automation is indispensable for cloud security

Lots of **tools** available to achieve compliance

KALM Benchmark can help to pick the right tools

dynatrace.com