KUBERNETES

Kubernetes hardening guidance

Three common sources of compromise in Kubernetes are supply chain risks, malicious threat actors, and insider threats.

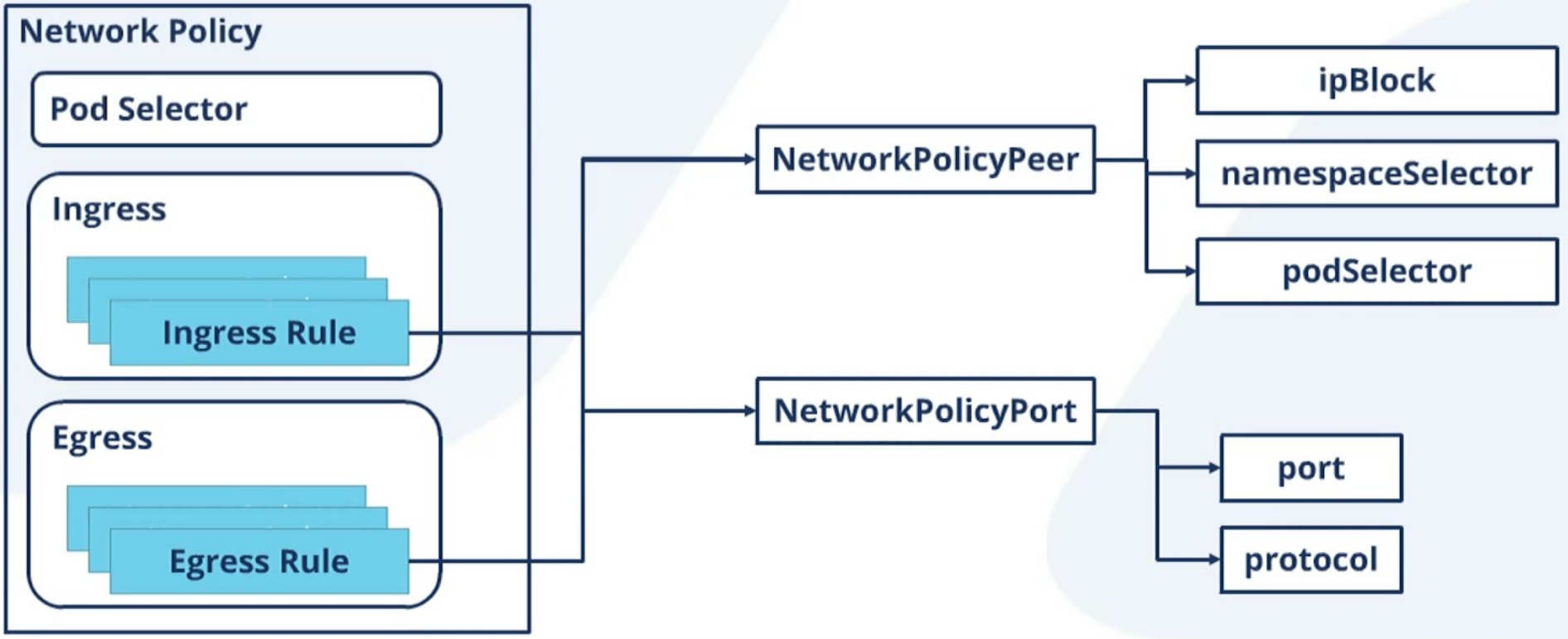
Supply chain risks are often challenging to mitigate and can arise in the container build cycle or infrastructure acquisition. **Malicious threat actors** can exploit vulnerabilities and misconfigurations in components of the Kubernetes architecture, such as the control plane, worker nodes, or containerized applications. **Insider threats** can be administrators, users, or cloud service providers. Insiders with special access to an organization's Kubernetes infrastructure may be able to abuse these privileges.

- Scan containers and Pods for vulnerabilities or misconfigurations.
- Sun containers and Pods with the least privileges possible.
- Use network separation to control the amount of damage a compromise can cause.
- Use firewalls to limit unneeded network connectivity and encryption to protect confidentiality.
- Use strong authentication and authorization to limit user and administrator access as well as to limit the attack surface.
- Use log auditing so that administrators can monitor activity and be alerted to potential malicious activity.
- Periodically review all Kubernetes settings and use vulnerability scans to help ensure risks are appropriately accounted for and security patches are applied.

NETWORK POLICIES

Network Policy Definitions

- Rules contain peers (the other side of the connection) and ports



NETWORK POLICIES

Network policies are implemented by the network plugin. To use network policies, you must be using a networking solution which supports NetworkPolicy.

The entities that a Pod can communicate with are identified through a combination of the following 3 identifiers:

- Other pods that are allowed (exception: a pod cannot block access to itself)
- Namespaces that are allowed
- IP blocks (exception: traffic to and from the node where a Pod is running is always allowed, regardless of the IP address of the Pod or the node)

There are two sorts of isolation for a pod: isolation for egress, and isolation for ingress. They concern what connections may be established. "Isolation" here is not absolute, rather it means "some restrictions apply". The alternative, "non-isolated for $direction", means that no restrictions apply in the stated direction. The two sorts of isolation (or not) are declared independently, and are both relevant for a connection from one pod to another.

By default, a pod is non-isolated for egress; all outbound connections are allowed. A pod is isolated for egress if there is any NetworkPolicy that both selects the pod and has "Egress" in its policyTypes; we say that such a policy applies to the pod for egress. When a pod is isolated for egress, the only allowed connections from the pod are those allowed by the egress list of some NetworkPolicy that applies to the pod for egress. The effects of those egress lists combine additively.

K8S OBJECTS - SECURITY

POD SECURITY CONTEXT: A security context defines privilege and access control settings for a Pod or Container. Security context settings include Discretionary Access Control (Permission to access an object, like a file, is based on user UID and group GID), Security Enhanced Linux (SELinux), Running as privileged or unprivileged, Linux Capabilities (Give a process some privileges, but not all the privileges of the root user)

```
apiVersion: v1
kind: Pod
metadata:
  name: security-context-demo
spec:
  securityContext:
    RunAsUser: 1000      → set a userid, gid or groups
    runAsGroup: 3000
    fsGroup: 2000
  volumes:
  - name: sec-ctx-vol
    emptyDir: {}
  containers:
  - name: sec-ctx-demo
    image: busybox:1.28
    command: [ "sh", "-c", "sleep 1h" ]
    volumeMounts:
    - name: sec-ctx-vol
      mountPath: /data/demo
    securityContext:
      allowPrivilegeEscalation: false      → Controls whether a process can gain more privileges than its parent process
      readOnlyRootFilesystem: true    →  Mounts the container's root filesystem as read-only
```

K8S OBJECTS - SECURITY

SERVICEACCOUNT: A service account is a type of non-human account that, in Kubernetes, provides a distinct identity in a Kubernetes cluster. Application Pods, system components, and entities inside and outside the cluster can use a specific ServiceAccount's credentials to identify as that ServiceAccount. This identity is useful in various situations, including authenticating to the API server or implementing identity-based security policies.

RBAC: The RBAC API declares four kinds of Kubernetes object: Role, ClusterRole, RoleBinding and ClusterRoleBinding. An RBAC Role or ClusterRole contains rules that represent a set of permissions. Permissions are purely additive (there are no "deny" rules).
A Role always sets permissions within a particular namespace; when you create a Role, you have to specify the namespace it belongs in.
ClusterRole, by contrast, is a non-namespaced resource. The resources have different names (Role and ClusterRole) because a Kubernetes object always has to be either namespaced or not namespaced; it can't be both.

Service Account can be binded to Kubernetes Cluster Role

KUBE API SERVER LOGIN

kube-apiserver is running as a Docker container on your master node, it is the front end for the Kubernetes control plane.

All Kubernetes clusters have two categories of users: **service accounts** managed by Kubernetes, and **normal users**. Kubernetes does not have objects which represent normal user accounts. Normal users cannot be added to a cluster through an API call.

Even though a normal user cannot be added via an API call, any user that presents a valid certificate signed by the cluster's certificate authority (CA) is considered authenticated. In this configuration, Kubernetes determines the username from the common name field in the 'subject' of the cert (e.g., "/CN=bob"). From there, the role based access control (RBAC) sub-system would determine whether the user is authorized to perform a specific operation on a resource. For more details, refer to the normal users topic in certificate request for more details about this.

Service accounts are tied to a set of credentials stored as Secrets, which are mounted into pods allowing in-cluster processes to talk to the Kubernetes API. API requests are tied to either a normal user or a service account, or are treated as anonymous requests. This means every process inside or outside the cluster, from a human user typing kubectl on a workstation, to kubelets on nodes, to members of the control plane, must authenticate when making requests to the API server, or be treated as an anonymous user.

CHAOS ENGINEERING

Chaos engineering is the discipline of experimenting on a system in order to build confidence in the system's capability to withstand turbulent conditions in production.

In software development, a given software system's ability to tolerate failures while still ensuring adequate quality of service—often generalized as resilience—is typically specified as a requirement. However, development teams often fail to meet this requirement due to factors such as short deadlines or lack of knowledge of the field. Chaos engineering is a technique to meet the resilience requirement.

Chaos engineering can be used to achieve resilience against infrastructure failures, network failures, and application failures.

DDoS

A distributed denial-of-service (DDoS) attack occurs when multiple systems flood the bandwidth or resources of a targeted system, usually one or more web servers.

Multiple machines can generate more attack traffic than one machine, multiple attack machines are harder to turn off than one attack machine, and the behavior of each attack machine can be stealthier, making it harder to track and shut down. Since the incoming traffic flooding the victim originates from different sources, it may be impossible to stop the attack simply by using ingress filtering. It also makes it difficult to distinguish legitimate user traffic from attack traffic when spread across multiple points of origin.

As an alternative or augmentation of a DDoS, attacks may involve forging of IP sender addresses (IP address spoofing) further complicating identifying and defeating the attack. These attacker advantages cause challenges for defense mechanisms. For example, merely purchasing more incoming bandwidth than the current volume of the attack might not help, because the attacker might be able to simply add more attack machines.

FALCO

Falco is a cloud-native security tool designed for Linux systems. It employs custom rules on kernel events, which are enriched with container and Kubernetes metadata, to provide real-time alerts. Falco helps you gain visibility into abnormal behavior, potential security threats, and compliance violations, contributing to comprehensive runtime security.

Falco is purpose-built for cloud-native architectures and integrates with container orchestrators like Kubernetes.

Falco adapts to the dynamic nature of containers, ensuring continuous compliance. With a comprehensive library of predefined rules based on security best practices and compliance standards like PCI DSS and NIST, Falco covers a wide range of security events, including unauthorized access attempts, privilege escalation, data exfiltration attempts

https://falco.org/docs/reference/rules/default-rules/

POLARIS

Polaris is an open source policy engine for Kubernetes that validates and remediates resource configuration. It includes 30+ built in configuration policies, as well as the ability to build custom policies with JSON Schema. When run on the command line or as a mutating webhook, Polaris can automatically remediate issues based on policy criteria.

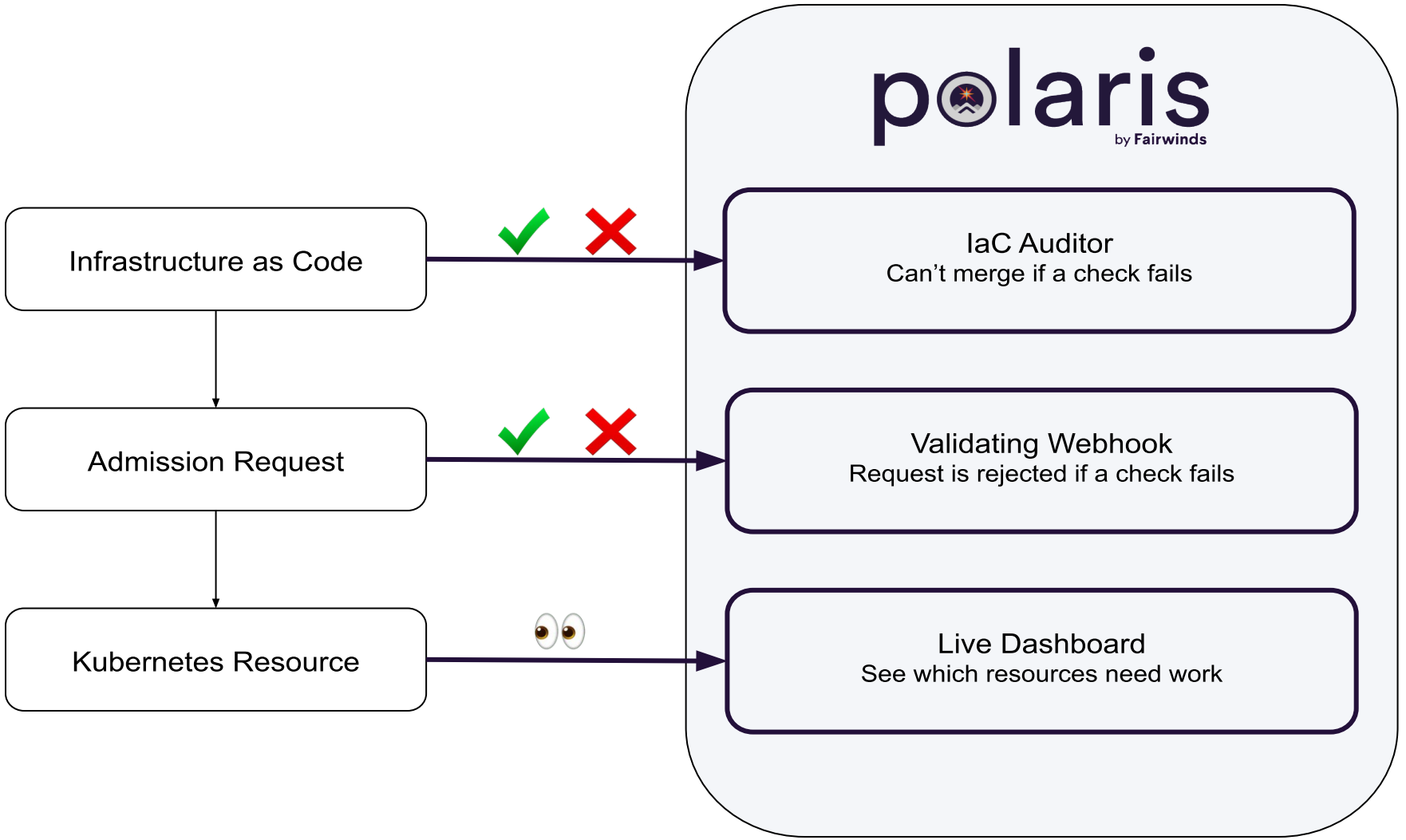
Polaris can be run in three different modes:

As a dashboard - Validate Kubernetes resources against policy-as-code.

As an admission controller - Automatically reject or modify workloads that don't adhere to your organization's policies.

As a command-line tool - Incorporate policy-as-code into the CI/CD process to test local YAML files.

POLARIS


polaris
by Fairwinds
Infrastructure as Code
Admission Request
Kubernetes Resource
IaC Auditor
Can't merge if a check fails
Validating Webhook
Request is rejected if a check fails
Live Dashboard
See which resources need work