How to secure your Kubernetes control plane *And node components*

A Series on Kubernetes Security

www.logiq.ai

Introduction

Kubernetes is an open-source container orchestration platform designed to run distributed services and applications at scale. A K8s or Kubernetes cluster contains several components that are a part of either the Kubernetes control plane or Kubernetes nodes. Over the years, Kubernetes has emerged as the hot topic in the DevOps space and is among the most wanted platforms for developers. As they are widely adopted to support varied use cases, cluster security is becoming a big concern for many organizations. The rising adoption of the platform has resulted in attackers targeting and abusing different parts of the cluster. Since Kubernetes is not immune to potential threats, such as misconfiguration, we need to take robust measures to improve overall cluster security.

$4.3 billion: The projected market for application container technologies in 2022, according to 451 Research. That's more than double the $2.126 billion the firm has predicted will be spent in 2019, and it also represents a 30% compound annual growth rate from 2017 through 2022.

It is vital to keep your Kubernetes control plane and node components secure, especially when you deploy it for compliance or business-related reasons. With that said, in this post, we will discuss some of the critical steps to secure your Kubernetes deployments.

Let's jump right into it.

Fundamentals of a Kubernetes Design

There are three primary aspects to a robust Kubernetes network policy. That is, a Kubernetes cluster should be:

- **Extendable**: The cluster should be customizable and should not exclusively favor a single provider.
- **Easy to Use:** Design the cluster to be operable with a few simple commands.
- **Secure**: The cluster should follow the latest pod security policies and best practices.

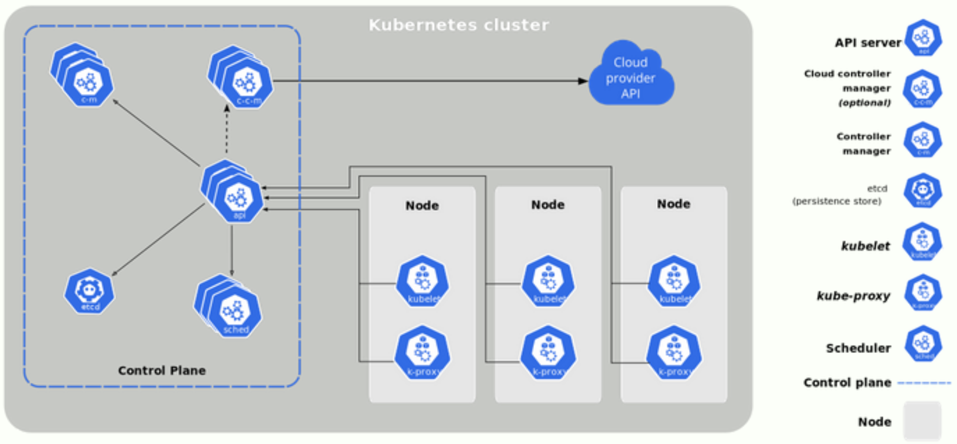
Understanding the Components of a Kubernetes Cluster and How to Secure Them

As said, you can categorize Kubernetes clusters into two parts – the control plane and the nodes, or the computing machines.

The control plane is responsible for controlling the cluster. Acting as the nerve center of a Kubernetes cluster, the control plane manages the cluster state and configuration data. The control plane components ensure that your containers run in adequate numbers along with all the necessary resources.

However, the control plane is highly susceptible to abuse from attackers because it is not easy to configure. While many businesses configure their clusters to run a certain way, some loopholes allow attackers to compromise the control plane.

To secure your Kubernetes control plane, check the following components and make sure you perform the listed operations.



etcd:

Configuring information and data about the cluster's state lives in etcd. It is a key-value store database designed to be fault-tolerant and distributed to be the source of truth about your cluster. To check if etcd is secure, type the following command:

- **ps -ef | grep etcd**

Then, modify the etcd pod specification file located at

/etc/kubernetes/manifests/etcd.yaml

Finally, check if encryption is already enabled by running: **–encryption-provider-config**

kube-controller-manager:

The **kube-controller-manager** is responsible for running the cluster and controlling its functions. It ensures that the correct number of pods are running at all times. In case a pod goes down, the controller immediately notices it and takes the necessary action.

To check if your **kube-controller-manager** is configured securely, first run the following command on your master node:

- **ps -ef | grep kube-controller-manager**

In the output, check and modify the following arguments:

- Set **--use-service-account-credentials argument** to **true**. This setting, along with RBAC ensures that the control loops run with the minimum required permissions.
- Set **--service-account-private-key-file argument** such that it uses a separate public/private key pair for signing service account tokens.
- Check if the **--root-ca-file** argument exists. Set **--root-ca-file** to a cert file containing the root cert for the API Server's serving cert, which will allow pods to verify the API Server's serving cert before making a connection.
- Set **RotateKubeletServerCertificate** to **true**. Ensure the argument applies when the kubelets get their certificates from the API Server only.
- Set **--address** to **127.0.0.1**, ensuring that the controller manager service is not bound to non-loopback insecure addresses.

kube-scheduler:

The kube-scheduler takes care of the cluster's health. It determines whether your cluster needs new containers and provisions them accordingly, thereby ensuring that your pod's resource needs are met at all times.

To check if kube-scheduler is configured securely, run the following command on your master node:

- **ps -ef | grep kube-scheduler**

In the output, check and modify the following arguments wherever necessary:

- Set **--profiling** to **false** to reduce the attack surface.
- Set **--address** to **127.0.0.1**, ensuring that the controller manager service is not bound to non-loopback insecure addresses.

kube-apiserver:

The kube-apiserver is the front-end of the control plane that handles external and internal requests. The kube-apiserver determines the validates requests and sends them for processing. Try hitting the kube-apiserver from an external IP to see if it's secure. If the kube-apiserver responds, it means other machines can also do the same unless you have restricted the API server to your IP. To ensure that your kube-apiserver is not public and exposed to the world, limit access to your Kubernetes cluster on GKE through the master authorized network settings by running the following commands:

- **gcloud container clusters create --enable-master-authorized-networks --master-authorized-networks=CIDR**

- **ps -ef | grep kube-apiserver**

How to configure RBAC in your K8S cluster?

Configuring RBAC correctly goes a long way in keeping your cluster from being compromised. With RBAC, you can restrict pods and users to the CRUD operations they can perform within a cluster. It allows for a granular view of what cluster components a user or pod can access and can help limit the attack surface of your cluster.

You can enable RBAC in your Kubernetes cluster via the **kube-apiserver** at startup with the flag: –**authorization-mode=RBAC**

A malicious user can easily disable RBAC. To see if RBAC is disabled, run this command:

ps -ef | grep kube-apiserver

Another way to ensure RBAC is securely configured is to manually impersonate users or user groups via user impersonation. User impersonation allows admins to debug and validate an authorization policy by temporarily simulating another user. It is also good practice to document unrecognized users and track their activity within the cluster.

Securing node components

If the control plane is considered the nerve center, the node components are the Kubernetes cluster's muscle. These nodes are responsible for running and controlling all the containers an pods in a cluster. While there is no necessity to have worker nodes, it's best to have at least one worker node running and controlling all the containers and pods on the same node as the control plane make things complex.

The primary components of a node are the container runtime engine, the **kubelet**, and **kube-proxy.**

- The container runtime engine runs containers in your cluster.
- The **kubelet** is a tiny agent in charge of making sure your nodes and containers are running.
- The **kube-proxy** facilitates Kubernetes networking services, handling network communications outside and inside the cluster.

To secure your node components, make sure you check these components for what's listed below.

Container runtime engine
Each compute node in a Kubernetes cluster has a container runtime engine. Most Kubernetes applications use Docker as a container runtime engine, although Kubernetes supports other runtime engines such as containerd and CRI-O. Securely configuring your container runtime engine is down to following security best practices issues by the engine you use. Configuring your engine as advised by your provider makes sure that your Kubernetes containers are safe from attacks.

kubelet
The kubelet is an agent that runs on each of your cluster's nodes, ensuring that all containers are running. It is also in charge of making configuration changes whenever necessary. A misconfigured kubelet can open a backdoor for attackers to your cluster.

You can either use a kubelet configuration file or use arguments on the running kubelet executable to configure your kubelet. You can find your kubelet configuration file by running the following command and looking for the **--config** argument:

- **ps -ef | grep kubelet | grep config**

Next, run the following command on each node:
- **ps -ef | grep kubelet**

In the output, check and modify the following arguments wherever necessary:
- **Set --anonymous-auth argument** to **false** to disallow unauthenticated requests.

- Set **--rotate-certificates** to **true**. If you're using a kubelet configuration file, set **RotateKubeletServerCertificate** to **true**. If you aren't doing so already, ensure that your kubelets get their certificates from the API Server.
- Set **--tls-cert-file** and **--tls-private-key-file** appropriately. If you're using a kubelet configuration file, set **tlsCertFile** and **tlsPrivateKeyFile** appropriately so that all connections on the kubelets happen over TLS.

kube-proxy
kube-proxy is a network proxy that runs on each node in your cluster, maintaining network rules on these nodes. It ensures that your nodes can communicate with internal and external resources as required and allowed.

Securing **kube-proxy** is down to ensuring the security and integrity of its kubeconfig file. To make sure your **kube-proxy** file is secure, first find the kubeconfig file in use by running:

- **ps -ef | grep kube-proxy**

Check **--kubeconfig** to find the **kube-proxy** config file location and then run the following command on each worker node:

- **stat -c %a <kube-proxy config file>**

From the output, ensure that permissions are set to 644 or stricter to help maintain the file's integrity.

Next, run the following command on each worker node using the same kube-proxy config file location:

- **stat -c %U:%G <kube-proxy config file>**

In the output, set permissions to root:root to prevent any unauthorized access to the file.

Final Words

Keeping your Kubernetes control plane and node components secure is finally down to following the basics. While configuring every single component in your cluster may seem daunting, doing so will ensure that you can safeguard your applications and deployments from unwanted threats and attacks. You can significantly reduce your attack surface and avoid common misconfigurations by following these configuration recommendations.