WHITE PAPER–JULY 2020

3 Common Kubernetes Security Mistakes and How to Avoid Them

vmware® Carbon Black

Table of contents

Introduction

Kubernetes greatly simplifies cloud native infrastructure for developers. With just a few lines of code and one Kubernetes command, it will deliver a replicated service, a load balancer, and you're ready to start serving internet users. However, Kubernetes also introduces new security gaps and blind spots within the code and configuration when applications are pushed to production. It can be easy to overlook the full consequences of what a single line of configuration is capable of. One simple mistake in a configuration file, or one erroneous copy/paste from a bad source, and you've suddenly poked a hole in the security of your clusters that could be exploited.

We've heard far too many cautionary tales from DevOps teams that have identified issues stemming from the misconfiguration of Kubernetes. These mistakes were caught after they were brought into production, which is far too late. Here are three common mistakes teams make when configuring Kubernetes workloads and how to correct them.

1. Accidentally exposing internal services to the internet

Applications are often made of edge services, which are designed to be exposed to the internet. These require strong authentication, access control and internal services designed to be accessed from other trusted services only. Unfortunately, there are many ways to accidentally expose these internal services to the Internet, potentially giving away access to sensitive data.

Include a load balancer, node port or ingress controller

A developer copies and pastes a workload configuration from a website, or simply uses a helm chart found online. Unbeknown to them, the configuration includes a definition of a *load balancer* that then exposes the service to the Internet. A mistake like this can be difficult to catch because it is a best practice to create load balancers to expose public services, as long as it is done on purpose.

Binding to the node IP

Workloads listen on a private IP address and receive network traffic from the rest of the network by default. To be reachable outside of the cluster, you must explicitly add an ingress controller or a load balancer service, then route traffic to the workload. But there are two dangerous settings to be wary of that result from a single line of code in a long configuration definition and binds the private IP to the underlying IP: *hostPort* and *hostNetwork*.

With the hostPort defined or hostNetwork set to true, the workloads become exposed as the node, but without the firewall rules and access control attached to the host. It is critical that you not expose production workloads directly to outside networks or to the Internet using the host IP. You should always forbid the use of these two settings.

kubectl port-forward

When someone wants to debug a service that's not exposed outside, and does not want to (or cannot) change the container or cluster configuration, there's a stealthy way to poke a hole in the cluster and expose any port to the outside: the *kubectl port-forward* <pod> <port> command. This maps a local pod port to a pod on the post, similarly to hostPort, without any kind of authentication or access control, directly to a workload.

This command does not create or modify Kubernetes configurations, making it much more difficult to detect and address. The port is then left available until the command is killed, which will never happen on its own if the command is sent to the background. This command should always be banned on production clusters.

No network policy, no firewall

In the case of accidental exposure of a workload, unauthorized network access can easily be stopped by a simple *Kubernetes network policy*. Authorize only other workloads to connect to it—never external IP addresses.

Unfortunately, very few companies have a good network policy in place if any. It would be unimaginable to set up an old-fashioned monolithic application server without a firewall, but this continues to be common with Kubernetes. One of the reasons for this is that organizations are now creating and updating network policies for hundreds, or even thousands, of dynamic workloads at any given time. Fortunately, there are solutions to manage these policies automatically and to allow for best practices (CIS Benchmark for Kubernetes, PCI, NIST 800-53, etc.) to be applied to network isolation and ingress/egress traffic management.

2. Overprivileged containers

Another common type of mistake is to give too many privileges to containers just to make things work. This means flaws in the application code. Malicious third-party libraries now have the power to get access to the entire cluster and do a lot more damage.

Privileged containers

Some containers may need special access to the underlying host or the network. The easiest way to enable these additional privileges, which is also the most unsafe, is to set a container as *privileged*. But there are other options available for applying only the specific abilities needed.

One of these is within the container's capabilities. Linux defined approximately 30 unique *capabilities* for audit log access, file ownership, kill signal, network and more. You can add only the capabilities required and keep the container as unprivileged.

It is also important that you not *allow privilege escalation*, meaning that the container can request more capabilities during runtime. Once you've come up with the list of required capabilities, it should never be changed.

There are more advanced ways to restrict the level of access of privileged containers or containers with too many capabilities, including *AppArmor* (Debian, Ubuntu) and *SELinux*. While these policies can become complex, simple policies for file access can be much easier to manage and yet still powerful to make the containers safer.

CAP_SYS_ADMIN

Using granular capabilities to give just the right amount of privileges is great, but one capability in particular should always be avoided: *CAP_SYS_ADMIN*.

This is the catchall of capabilities, with more than 25 privileged actions included. Giving CAP_SYS_ ADMIN capabilities to a container is essentially the same as giving it full privileges.

Sharing sensitive host folders

It can be tempting at times to share the *host file system* with containers. But this means that containers will be able to compromise the entire host by replacing binaries with malicious versions, or messing up the Kubernetes or Docker configuration. You should never share /etc/kubernetes, /bin, /var/run, docker.sock and the like.

This can be made worse if the container is running as root as it allows the container to override any shared file.

LEARN MORE

To set up a personalized demo or try it free in your organization, visit *carbonblack.com/demo*

For more information or to purchase VMware Carbon Black Products please call: (855) 525-2489 in the US, (44) 118 908 2374 in EMEA

For more information, email *Contact@CarbonBlack.com* or visit *CarbonBlack.com/epp-cloud*

3. Not guiding teams toward secure development from the start

Kubernetes is powerful because it gives teams that might not have experience deploying applications securely to production control over the infrastructure that they didn't have before. It has shifted business-critical operations, such as security, to the development teams.

Organizations moving to Kubernetes need to put boundaries for development teams into place from the start by way of configuration policies to avoid the previously mentioned mistakes, along with many others we haven't listed, and ensure that critical changes to infrastructure after deployment are flagged for review. These policies must protect the complete development and deployment process without impacting development agility and speed to market.

VMware Carbon Black Cloud™ enables policy-based reporting and enforcement of your security posture across all workloads deployed in Kubernetes clusters. It takes a proactive approach to container security, automatically detecting and responding to security risks before they become an issue. Octarine technology, now part of VMware Carbon Black Cloud, further expands VMware's intrinsic security strategy to containers and Kubernetes environments. For more information on VMware Carbon Black Cloud, visit *www.carbonblack.com*.

VMware, Inc. 3401 Hillview Avenue Palo Alto CA 94304 USA Tel 877-486-9273 Fax 650-427-5001 vmware.com