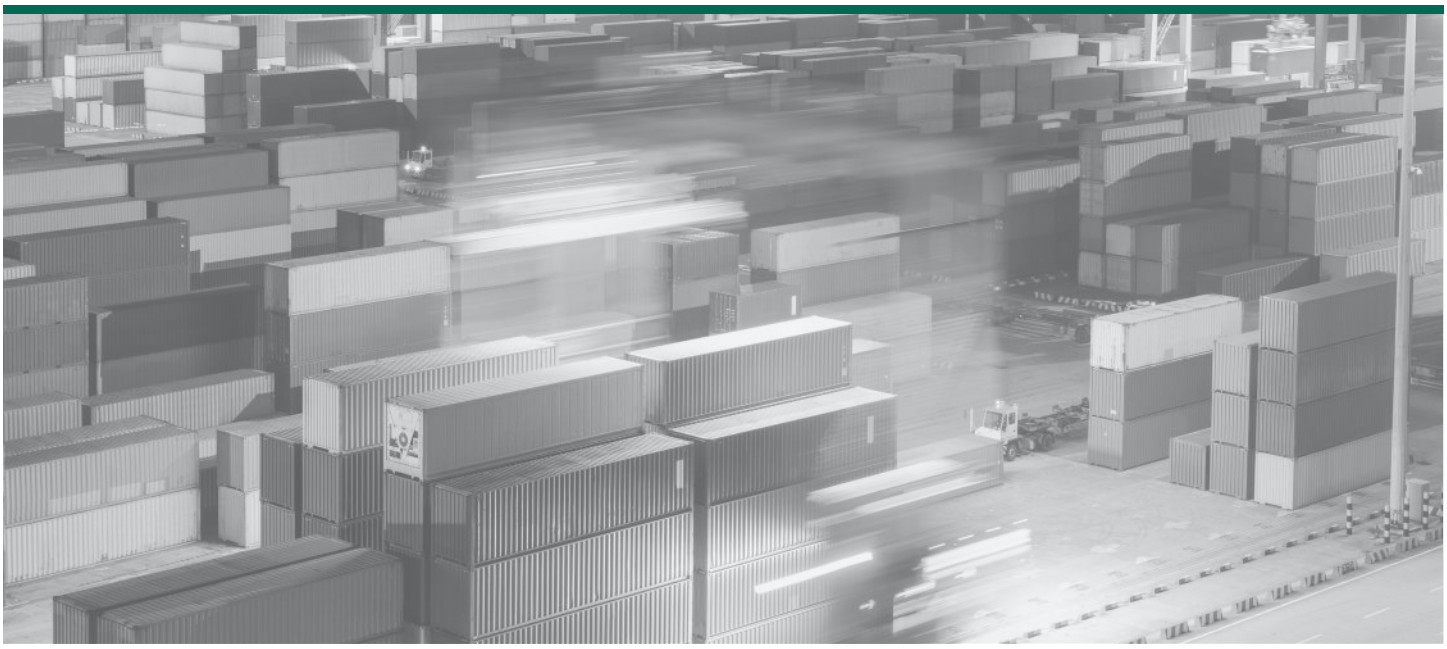
FORTRA

GUIDE

Secrets to a Stronger Container Security Strategy



The Fellowship of Containers, an Emerging Developers' Community

In recent years, container adoption has continued to grow as more organizations transition from virtual machines to microservices-based architectures. Containers provide increased efficiency, portability, and scalability. Today, orchestration platforms like Docker Swarm and Kubernetes are some of the most widely adopted technologies because of the central role they play in developing with containers.

Compared to virtual machines, containers have a smaller footprint which means a smaller attack surface. They also provide an additional layer of security through their ability to isolate applications. However, this doesn't mean that your containerized environment is not susceptible to malicious attacks between containers or within the shared resources of the underlying host.

A strong container security strategy starts with a 360-degree awareness of the container and how it interacts with its environment and ends up with automated governance policies woven into the Continuous Integration/Continuous Delivery (CI/CD) pipeline. When you first plan a container security strategy, it can feel a little like trial-and-error. And while DevOps may be all about iterating to get better, security isn't something you should take a chance on.

Strategy

Much of the work involved in creating a strong security strategy for your container environment versus other virtualized assets comes down to setting permissions, limiting resources, and monitoring systems.

There are other considerations unique to containers that also pertain to general development environments. While guidance for those matters tends to be less prescriptive, they still are vital.

As containers continue to evolve, the decisions you make regarding how you host, run, and secure them also must evolve. Fortra's Alert Logic's container experts have developed, evaluated, and refined our guidance and best practices for security in your environment:

- **Choose your ecosystem carefully:** How will you host and manage your containers? Will you choose a completely do-it-yourself approach or go full serverless?
- **Take care of your host:** Unless you are using a technology like Fargate to manage and host your container applications, you will need to ensure your container hosts are up to date and monitored.
- **Know your containers:** Which approach will you chose to run your containers? You will be responsible for the container itself and the containerized application.

The Container Ecosystem

Key Risk: Insecure configurations or vulnerabilities allow malicious container hijacking. For example, allowing attackers to insert their own cryptomining containers consumes resources.

To deploy and manage containers, organizations typically rely on an orchestrator — a system that automates management activities such as deployment and scaling.

Container management and orchestration layers can be prone to insecure configurations that allow remote management access.

Containerized environments have many more components than traditional server hosting, which poses its own set of security challenges:

- Can you tell which deployments or clusters are affected by a high-severity vulnerability?
- Are any exposed to the internet?
- What is the blast radius if a given vulnerability is exploited?

Orchestrators include components, such as an API server that integrates with other applications seeking to create or distribute containers. They also will include monitoring and management tools that provide visibility into the container farm's health. Orchestrators also can create, store, and manage container images.

These are the control layer for containers. If attackers can compromise it, then so too can all associated containers.

Container management is available in several models. In a similar way to running a server, the choice is based on how much of the container ecosystem you want to manage yourself.

Choose Your Ecosystem Carefully

Run Containers with Server-level Control
e.g., Kubernetes running on AWS EC2

Users must secure the container orchestrator console/dashboard and APIs in the same way that you would any other application.

- Update regularly and scan for vulnerabilities
- Restrict public access to the orchestration platform
- Check configurations continuously

Run Containerized Applications or Build Microservices
e.g., AWS Elastic Container Services

In Amazon EKS, the Kubernetes control plane is under the control and responsibility of AWS. Users are responsible for securing worker nodes and workloads running on them.

- Limit access from root AWS user accounts
- Implement least privilege access when granting IAM permissions
- Enable CloudTrail logging for audit and forensics
- Follow configuration best practices for VPC networking and EC2 resources

Run containers without managing servers with per-application resource management
e.g., AWS Fargate

While AWS Fargate provides container application isolation, users still must secure the AWS account and applications running on the container.

- Follow AWS account best practices
- Use trusted images
- Scan containers for vulnerabilities
- Monitor network and log activity

Take Care of Your Host

Key Risk: Host compromise — or escape to host attacks — allows compromise of all containers running on a host, enabling the attacker to write/overwrite host or other container files, or even inject code and harvest data from running processes.

It's well known that a healthy host is a happy host — containers depend on the host for many system components. Any flaw or vulnerability in those container hosts can impact all your containers, making security of the host paramount.

Sets of containers run on a shared host with common underlying system components. Containers can be isolated from one another, but if the host is compromised, then all containers running on it are at risk.

Each host needs to have its own set of security access controls in place and be continuously monitored for any new vulnerabilities discovered since that host was deployed.

Keeping your main host up to date will greatly improve your efficiency and reduce risks. Do this by automating regular patching using AWS Lambda and governance policies using Docker Bench as part of your continuous integration/continuous delivery pipeline. Alternatively, you can use a managed container service — like AWS Fargate — to take care of the host for you.

How Does Your Host Affect Your Security?

Limit host resources by container — A denial of service (DOS) attack on a container could deplete its host's resources and consequently shut down the other containers it supports. By using container orchestration frameworks like Docker Swarm and Kubernetes, you can limit CPU, RAM, and ulimits for each container, which can help reduce DOS attacks and general resource hogging.

TIP — Amazon Elastic Container Service (ECS) allows you to configure CPU, RAM, and ulimits to help you automate governance.

Restrict kernel capabilities — Set limits on what the kernel can access by task.

TIP — With Amazon ECS, kernel capabilities are limited by the service and can be set per task.

Validate your host before launching — One of the biggest challenges in SecOps is incorporating repeatable container governance policies into your pipeline. You can use an automated service like Docker Bench to validate your container host against security best practices.

TIP — In an AWS environment, you can use AWS Lambda to build Docker Bench into your CI/CD pipeline so it automatically calls the service whenever you launch a new host.

Control file system access — There's no reason for the data in your file systems to be accessible to all your containers and users, but it is unless you specify otherwise. By configuring the file systems directory default to read only, you can make it so that only the host can access the data. SELinux provides a default for Docker that enforces the read/execute to /usr.

TIP — Use the Amazon ECS configuration flag to turn on read-only.

How Well Do You Know Your Containers?

Key Risk: A compromise through an application or component vulnerability providing access to underlying data and systems that may live outside the container ecosystem.

Building apps using containers introduces new security challenges and risks. A single compromised container can threaten all other containers and the underlying host. Securing containers generally includes:

Securing the image — Vulnerabilities can impact container images just like any other piece of code. Complexity occurs in the sheer number of containers running in an application environment. It is critical to scan your containers before they are deployed and operational.

Container registries — A container registry provides a convenient, centralized source for storing and distributing application images. Because the registry is central to the way a containerized environment operates, it is essential to secure it.

TIP — Amazon Elastic Container Registry transfers your container images over HTTPS and automatically encrypts your images at rest. You can configure policies to manage permissions and control access to your images using AWS Identity and Access Management (IAM) users and roles without having to manage credentials directly on your EC2 instances.

Container runtime — The container runtime is one of the most difficult parts of a container stack to secure because traditional security tools were not designed to monitor running containers. Legacy tools typically cannot see inside containers, much less establish a baseline for a secure container environment.

Map out container traffic — It's important to start with a good understanding of the traffic you expect to see traveling North/South (from container to its host) and East/West (between containers) to help you better detect anomalies. You can use your network architecture or a third-party tool to help you map out the network pattern for expected behavior. Based on your map, it's a good idea to disable inter-container communication (icc) that is not specifically needed (see "Are Your Containers Talking Behind Your Back?" for guidance).

Monitor your traffic — Once you know what expected traffic looks like, you need a mechanism to monitor actual traffic so you can spot traffic mishaps. That's where IDS comes in. When evaluating your options for a third-party IDS solution, consider whether you need an integrated or side-car solution. In an integrated solution, you run a container on the host that has access to ETH 0 bridge and can see all the traffic from inside the system. With a side-car solution, the IDS runs outside of the system and gets fed information on the containers. This often comes in the form of log correlation.

Output logs to a centralized location — If your IDS detects a problem, you can use your log data to understand what's happening inside the environment and perform forensics. Of course, logs only help if you can get the data. You'll want to configure containers so they output logs to a centralized location, ideally a separate container, where you can use a log management system to help you make sense of them. For continuous logging on Docker, you may have to configure the default logging driver to write logs to your desired location (/var/log/, /var/ log/docker/).

TIP — AWS S3 is an object storage service that can be used to store logs. AWS CloudWatch Logs is a log service to which most AWS services can output logs. CloudWatch Logs provide a single view over all the different logs from your container instances in one convenient location, allowing you to gain deeper contextual evidence on the findings, manipulate the data, and take action on it.

Are Your Containers Talking Behind Your Back?

Segregate containers — You can lower your overall risk exposure by establishing smaller groups of containers that don't talk to each other. Depending on your environment, you might segregate containers by host, by the role they play (e.g., web server, database, customers), or based on their risk of exposure so those most susceptible are grouped together.

TIP — With AWS Fargate Individual ECS tasks or EKS pods, each run in their own dedicated kernel runtime environment and do not share CPU, memory, storage, or network resources with other tasks and pods. This ensures workload isolation and improved security for each task or pod.

Disable inter-container communication — Another way to cut down on inter-container communication is to disable it.

TIP — Amazon ECS lets you use a link flag to connect containers and control communications between them. You can set it up by marking docker flags "–icc=false" and "–iptables=true".

Restrict network chatter — Security threats also can be transmitted between containers through your network. Be sure to consider your network controls.

TIP — Amazon ECS offers network control through the container control for Elastic Network Interface (ENI). You can use the ENI to customize port configurations for your use case. Furthermore, AWS App Mesh allows you to standardize network communication by giving you visibility into how things are communicating on an application level.

Remove static libraries and binaries — Prepopulate containers ship with libraries and binaries you'll never need to use as these can be a point of entry if you're not careful. By turning on the SECCOMP modular that's built into the Linux kernel, you can limit system calls and enable least privilege access.

TIP — Remove setuid/setgid binaries from images

TIP — Debian 'defanged' image Dockerfile configuration (removes access to those binaries)

Who is Accessing Your Containers?

Set least-privilege access rules — As with any software system you run, it's a good idea to use the lowest privileges possible for your containers, as well as the binaries and libraries within them. This helps prevent privilege escalation and wrongful data access and all kinds of bad habits.

TIP — AWS Identity and Access Management offers a shortcut to implementing least privilege restrictions based on AWS best practices for leveraging roles established within one service to another. IAM policies can be applied automatically based on a task definition you set when launching the container and attached to either users or roles.

Turn off root privileges — We recommend setting up a non-root user and making it the default in your container configuration. For the most part, Docker and Kubernetes subprocesses do not run with root privileges out of the box; however, it never hurts to double check. And when you absolutely must use the root account for container-based actions, be mindful of how you use it.

TIP — Amazon ECS provides a configuration flag that lets you choose the user you want for any task when you launch. Flag your non-

Store secrets separately — As a rule of life and app building, secrets are necessary — and messy. Instead of embedding your secrets in your app configuration directly, which could compromise the security of other apps that use similar secrets, you can store secrets separately.

TIP — AWS Systems Manager Parameter Store will store and encrypt your secrets separately from your app and its container. Systems Manager Parameter Store is fully integrated with AWS which means you can use IAM roles to call the System Manager Parameter Store to retrieve the secret for the system task (e.g., logging into the server) without embedding the secret into the app. At the application level, AWS Secrets Manager allows you to store secrets that manage application and database passwords.

Meet Your Secret Agent, Fortra's Alert Logic

With Alert Logic, you get the industry's leading network intrusion detection solution and log management for containers, all our offerings employ our AWS cloud-native security platform, integrated threat intelligence, and prioritized vulnerability remediation recommendations.

Fortra's Alert Logic MDR® is a fully managed service where our experts monitor your environment 24/7.

Unlike many of the other solutions available today, ours has been developed to work with AWS native architecture as well as other cloud and on-premises systems. We are a certified AWS Container Competency partner.

Detect cyberattacks in real-time by analyzing the signature of data packets as they traverse your containerized environments.

Collect, aggregate, and search container application logs to gain insight into container activity for optimal security and compliance.

Make your security as portable as your containers across cloud, hybrid, and on-premises environments.

Build a better view of security impacts with a graphical representation of the compromised container and its relationships.

Get proactive notifications from our security experts when suspicious activity occurs.

For more information, please visit alertlogic.com

FORTRA

Fortra.com

About Fortra

Fortra is a cybersecurity company like no other. We've created a simpler, stronger, and more straightforward future for our customers. Our trusted experts and best-in-class portfolio of integrated, scalable solutions bring balance and control to organizations around the world. We're the positive changemakers and your relentless ally for cybersecurity that prevails. Learn more at fortra.com.