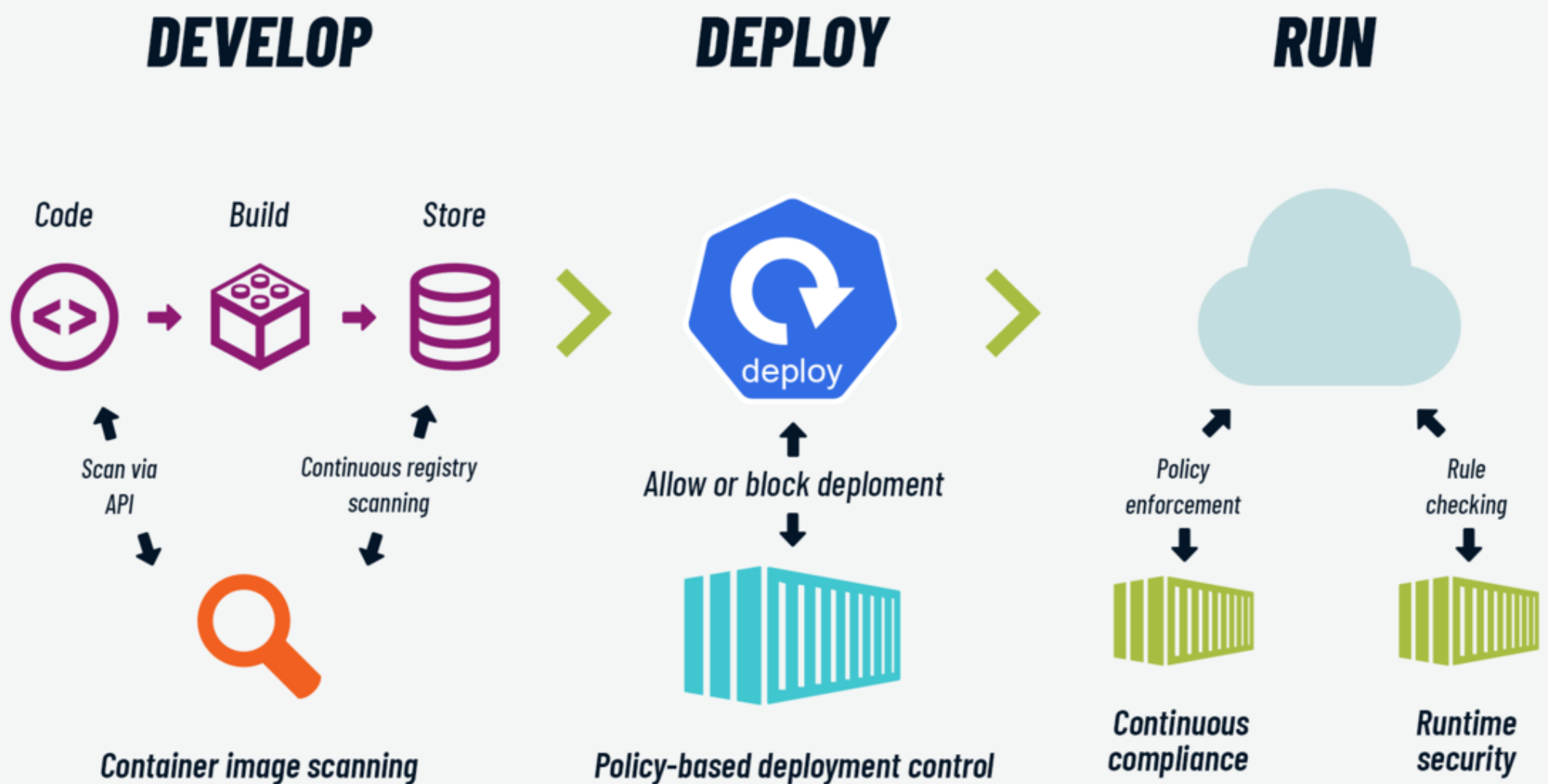


Container Security:



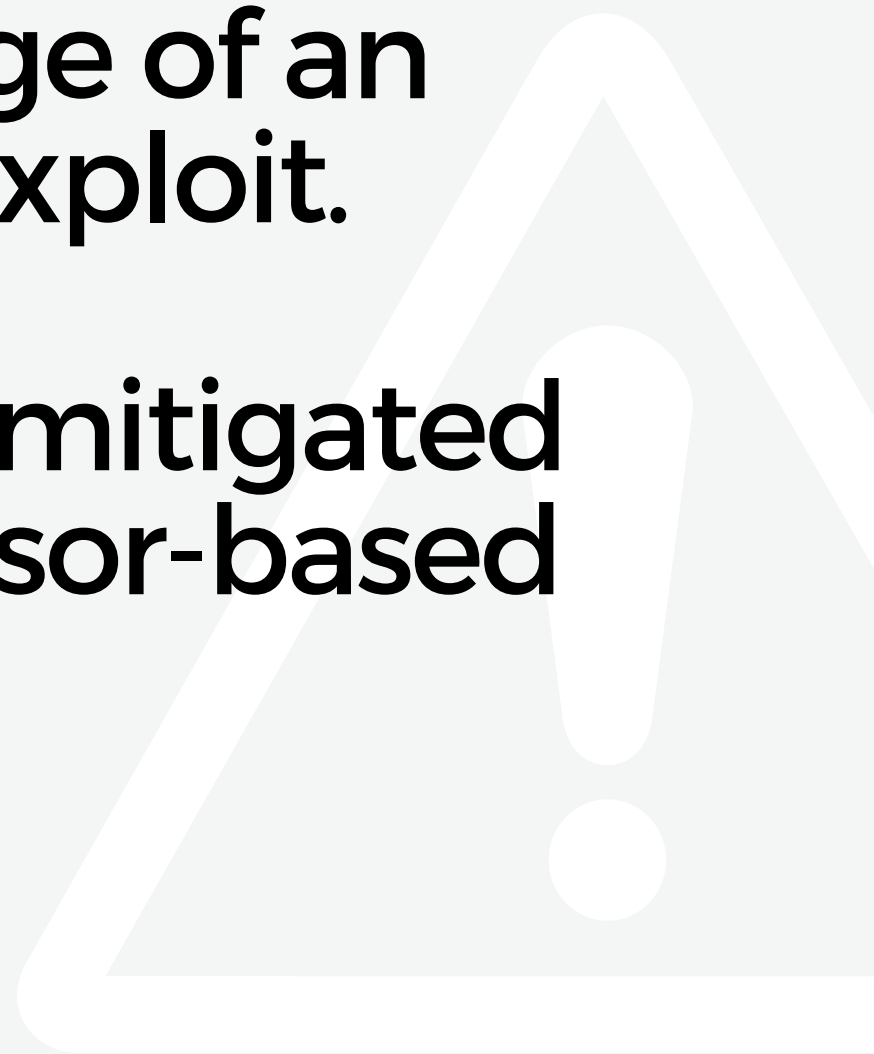
How to Implement It



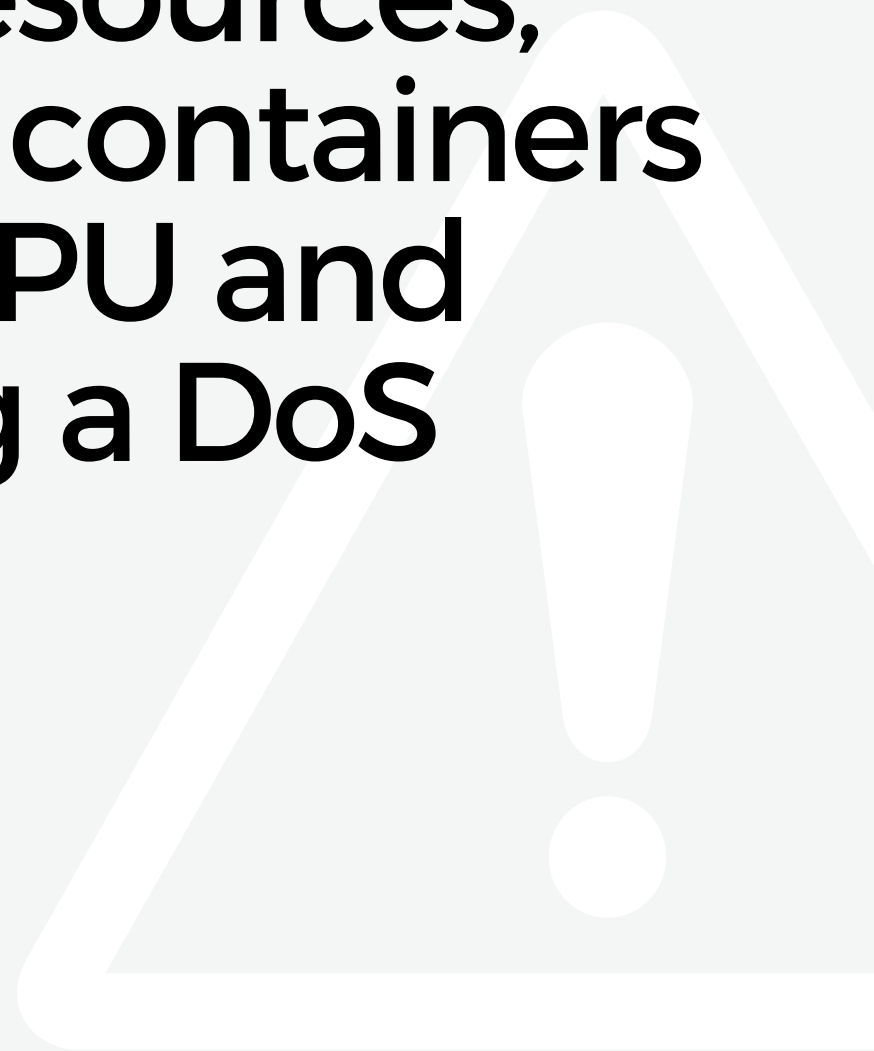
What are the main security issues related to containers?

KERNEL EXPLOIT: Sharing the kernel of the host machine exposes all containers running on it in the event of an attack that takes advantage of an operating system exploit.

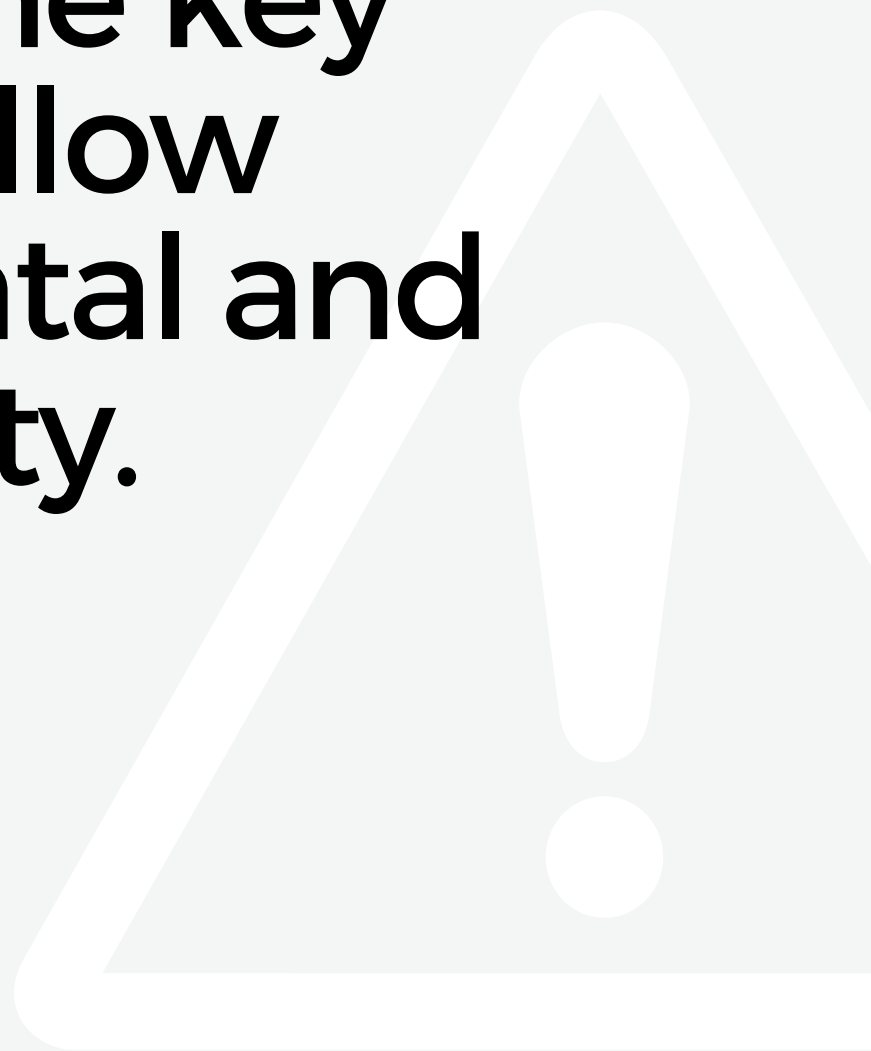
This risk is strongly mitigated by classical hypervisor-based virtualization.



DENIAL OF SERVICE (DOS) ATTACKS: Exploiting an application vulnerability or a programming error could monopolize available computational resources, preventing other containers from accessing CPU and RAM and causing a DoS attack.

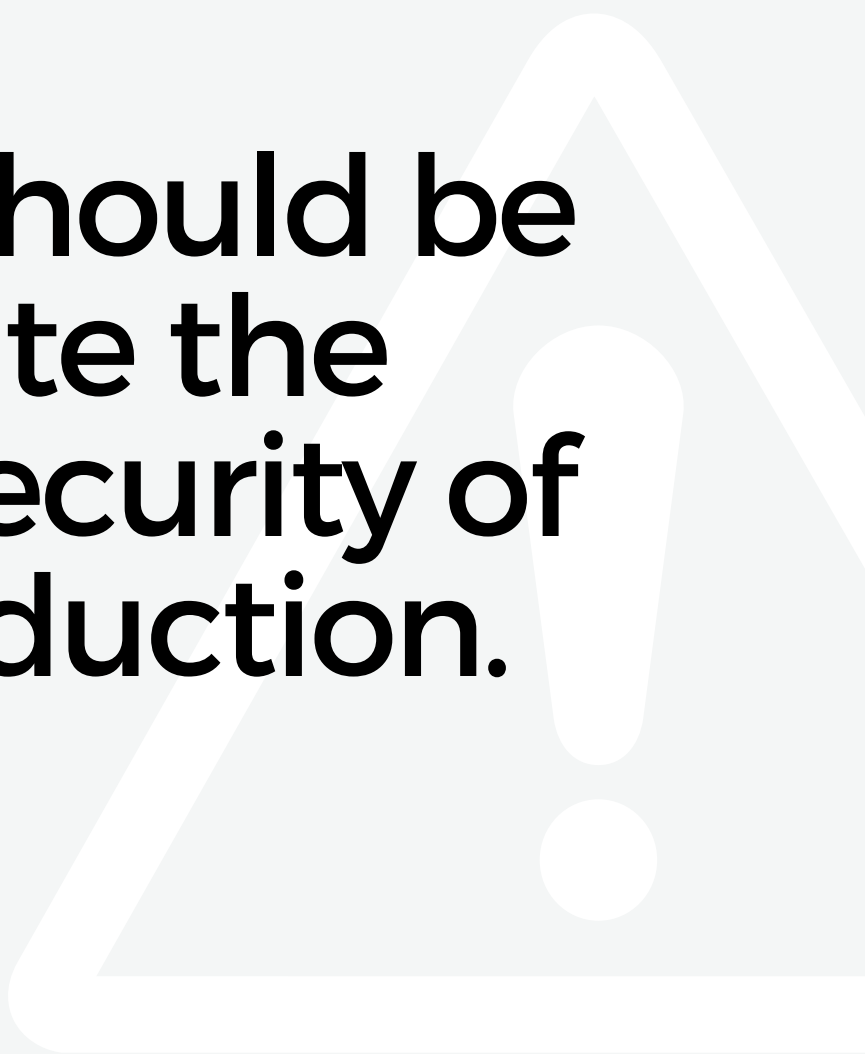


You can prevent this by appropriately configuring the resource allocation dedicated to each container and the key parameters to allow efficient horizontal and vertical scalability.



CONTAINER BREAKOUT: In case of a bug present within an application, a user could climb the privileges pyramid up to the root level and gain access to the host.

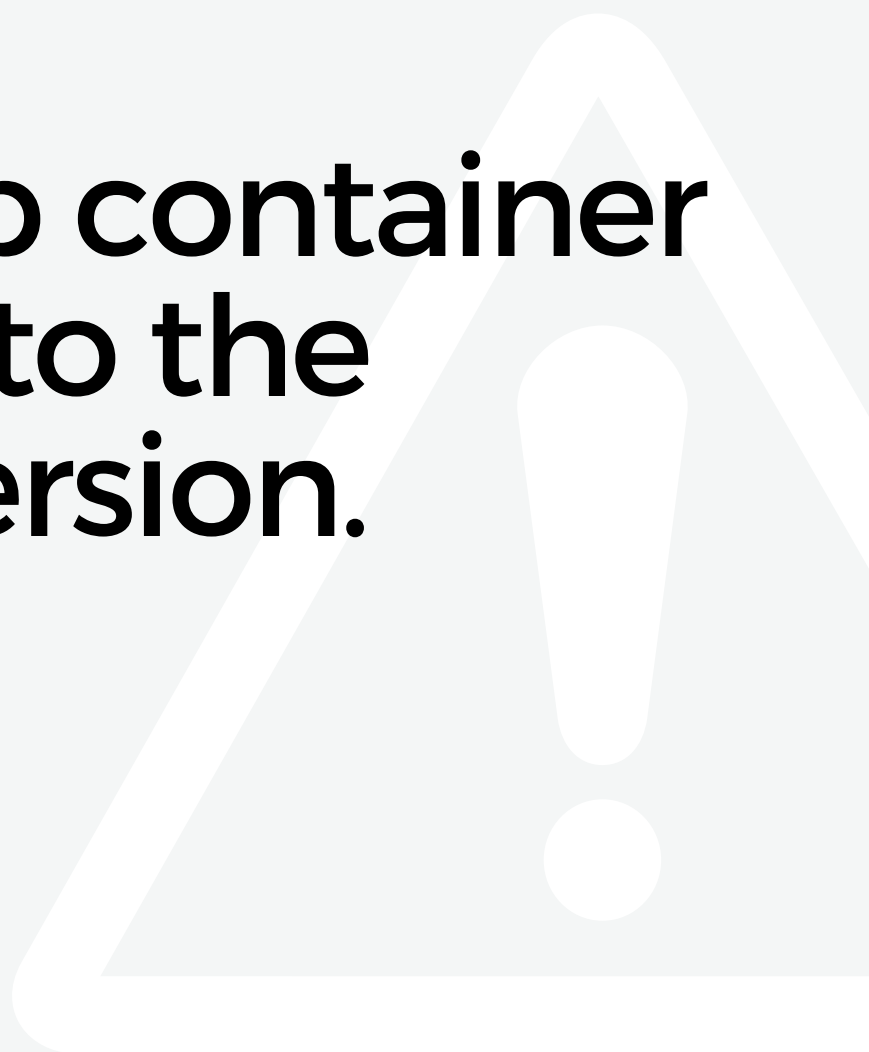
Periodic checks should be planned to validate the robustness and security of containers in production.



INFECTED IMAGES:

Standard container images (builds) can be infected with malware or have known vulnerabilities.

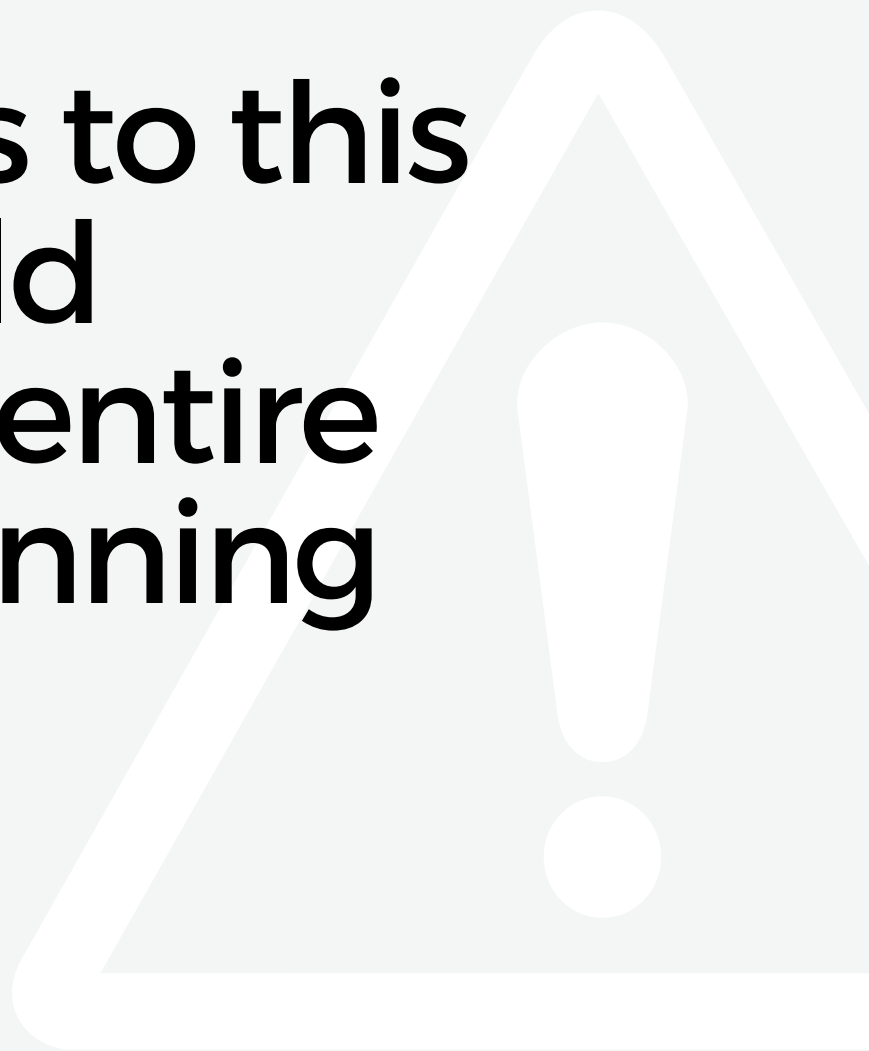
It's critical to keep container images updated to the latest available version.



COMPROMISED SECRETS:

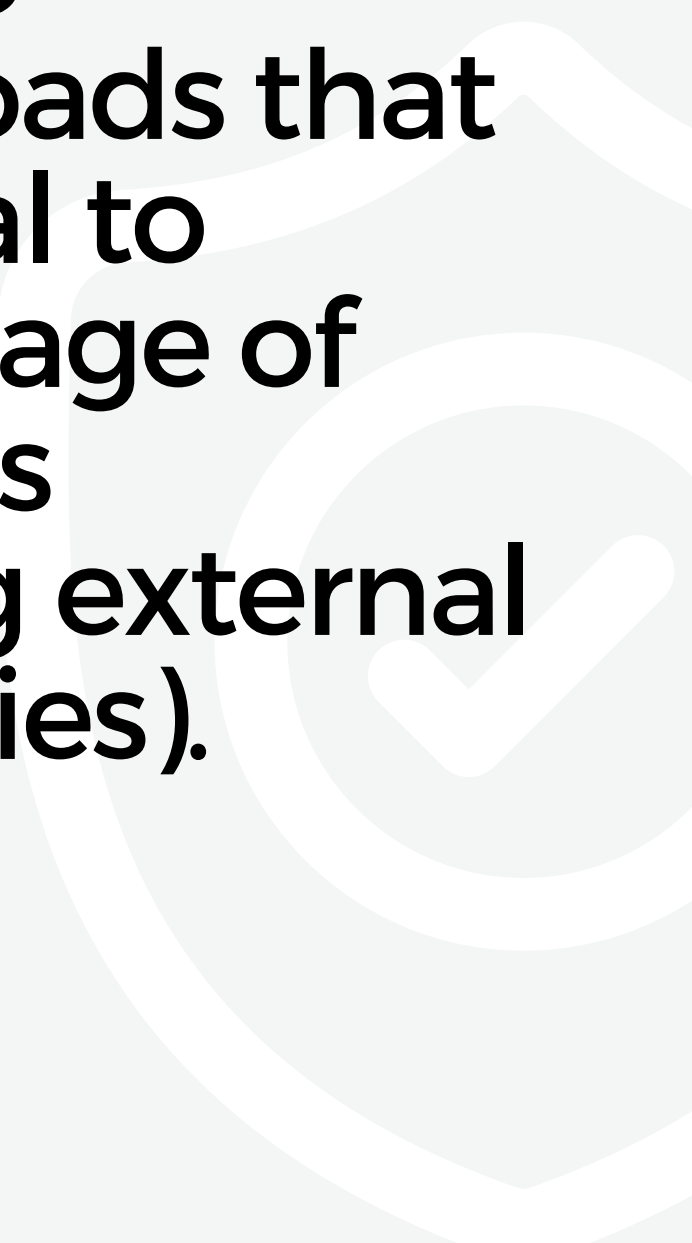
During the execution of applications, the container periodically accesses sensitive information, API keys and credentials.

Unwanted access to this information would compromise the entire security of the running services.



6 best practices for container security

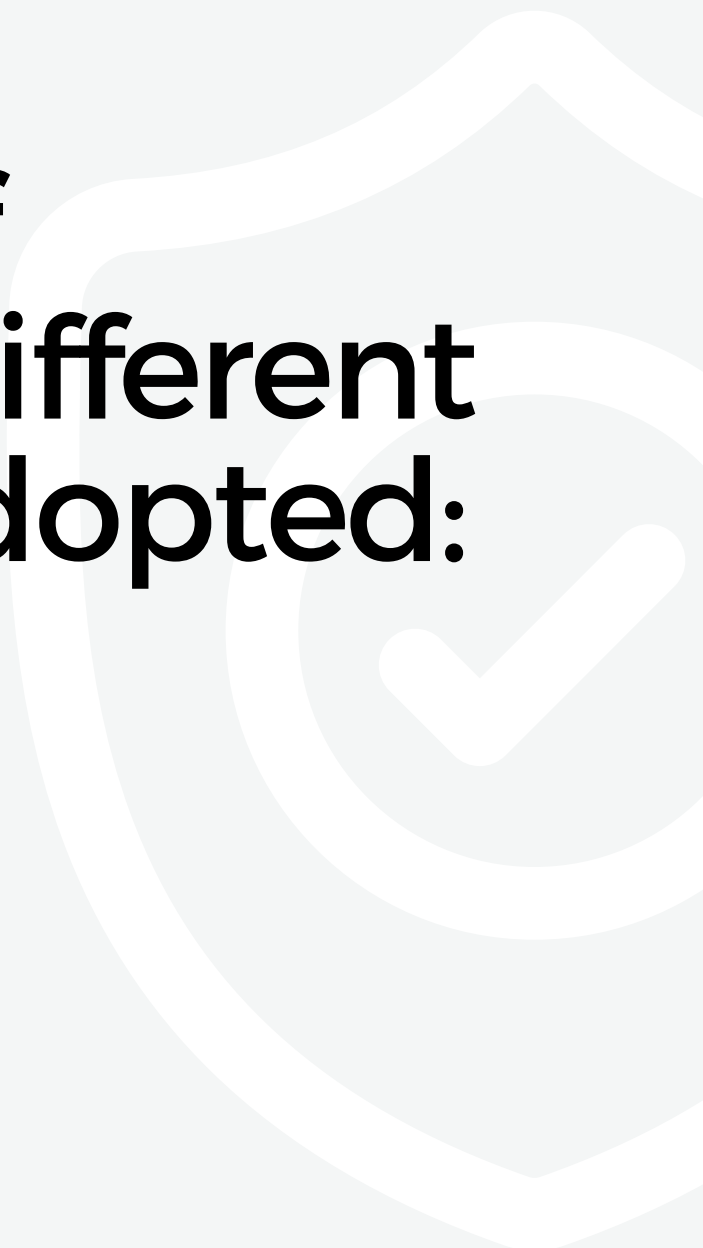
1. IMAGE SECURITY: A vulnerable or compromised base image can lead to problems for all workloads that use it as a base. It's ideal to customize the base image of the container as little as possible (e.g. by adding external packages or SDK libraries).



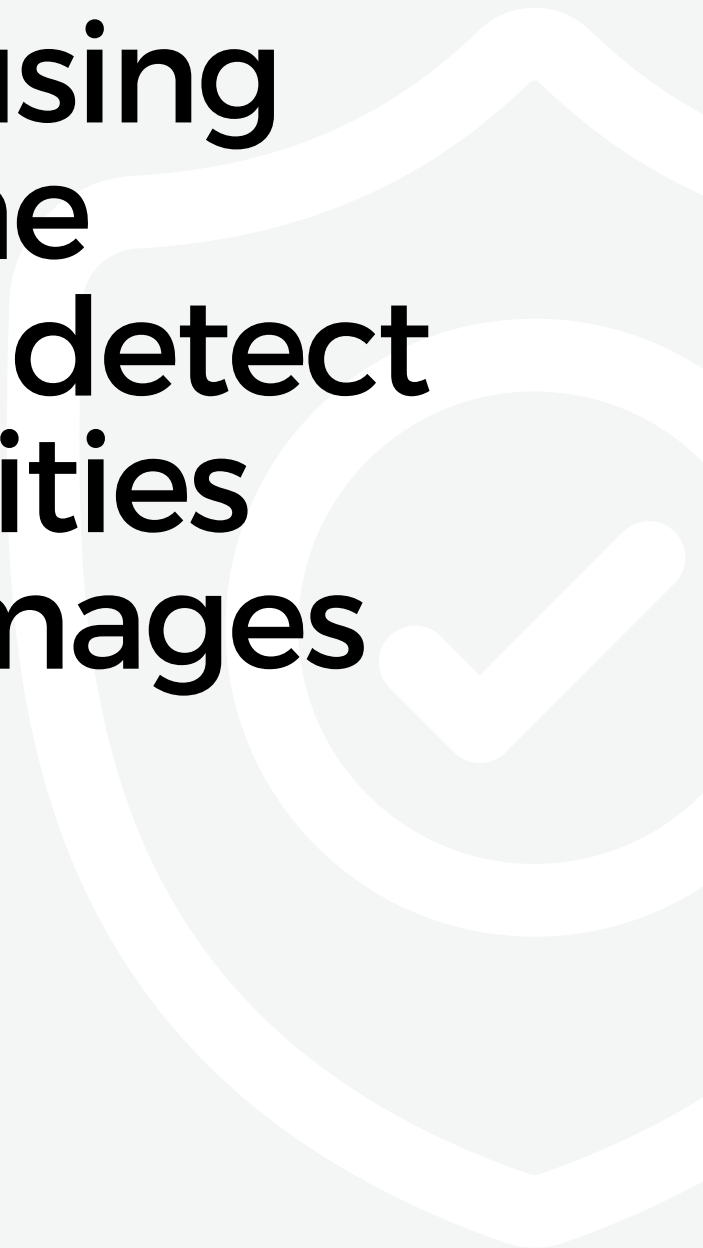
2. IMAGE REGISTRY

SECURITY: Registries are the repositories of containerized software images.

To keep this type of repository secure, different strategies can be adopted:



- Using access control policies to limit access/actions.
- Using container image signing.
- Making use of continuous image scanning using plugins built-in the registry service to detect known vulnerabilities within the base images used.



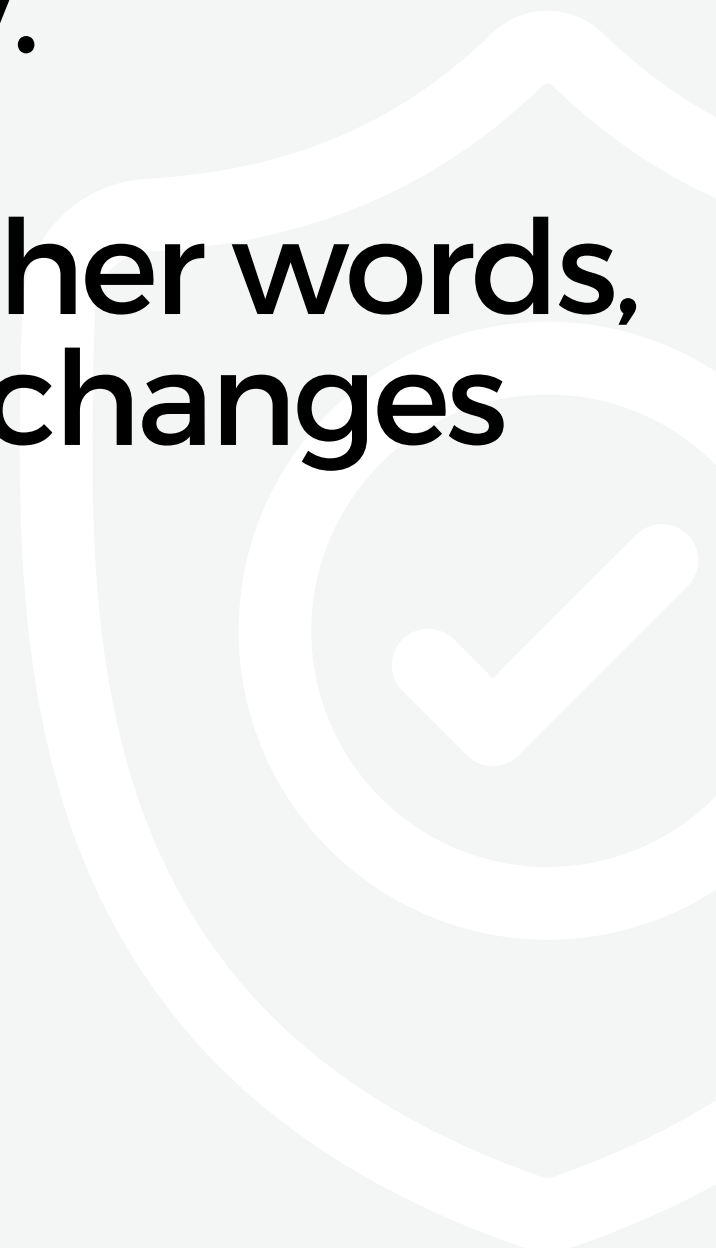
3. SECURING THE DEPLOYMENT:

Considering a production environment as the target of our deployment process, this phase can be secured in several ways:

- Physically protecting the environment with dedicated firewalls or network security rules/groups.



- Using an orchestration platform that typically provides secure API endpoints and that supports role-based security.
- With "immutable" deployments, in other words, reinitialized when changes occur.



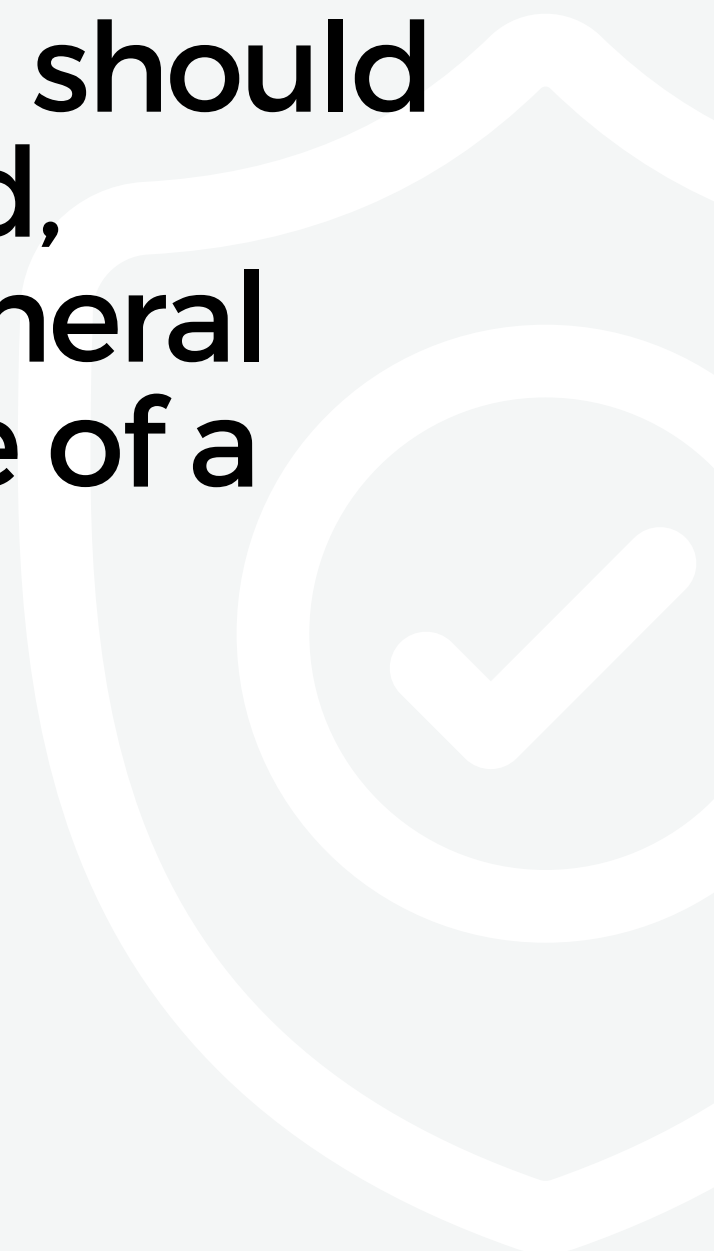
4. SECURING THE APPLICATION

RUNTIME: It's ideal to follow traditional best practices in the context of Cloud Native architectures, such as:

- Creating separate virtual networks for workloads that are different in nature and disconnected from each other.
- Applying the least privilege principle on the communication channels/ports that each container can use, thus only exposing strictly necessary ports.

5. USING MINIMAL AND SHORT-LIVED CONTAINERS: Containers are not meant to be used as "servers".

Their customization and the files contained within them should therefore be minimized, maintaining the ephemeral nature of each instance of a container.



6. MONITORING CONTAINER

ACTIVITY: This last objective is achieved by implementing mechanisms that provide a deep level of observability on the following components of the containerized infrastructure:

- Master nodes (for Kubernetes-based orchestration);
- Container engines;
- Workloads running in containers;
- Containerized middleware and networking.