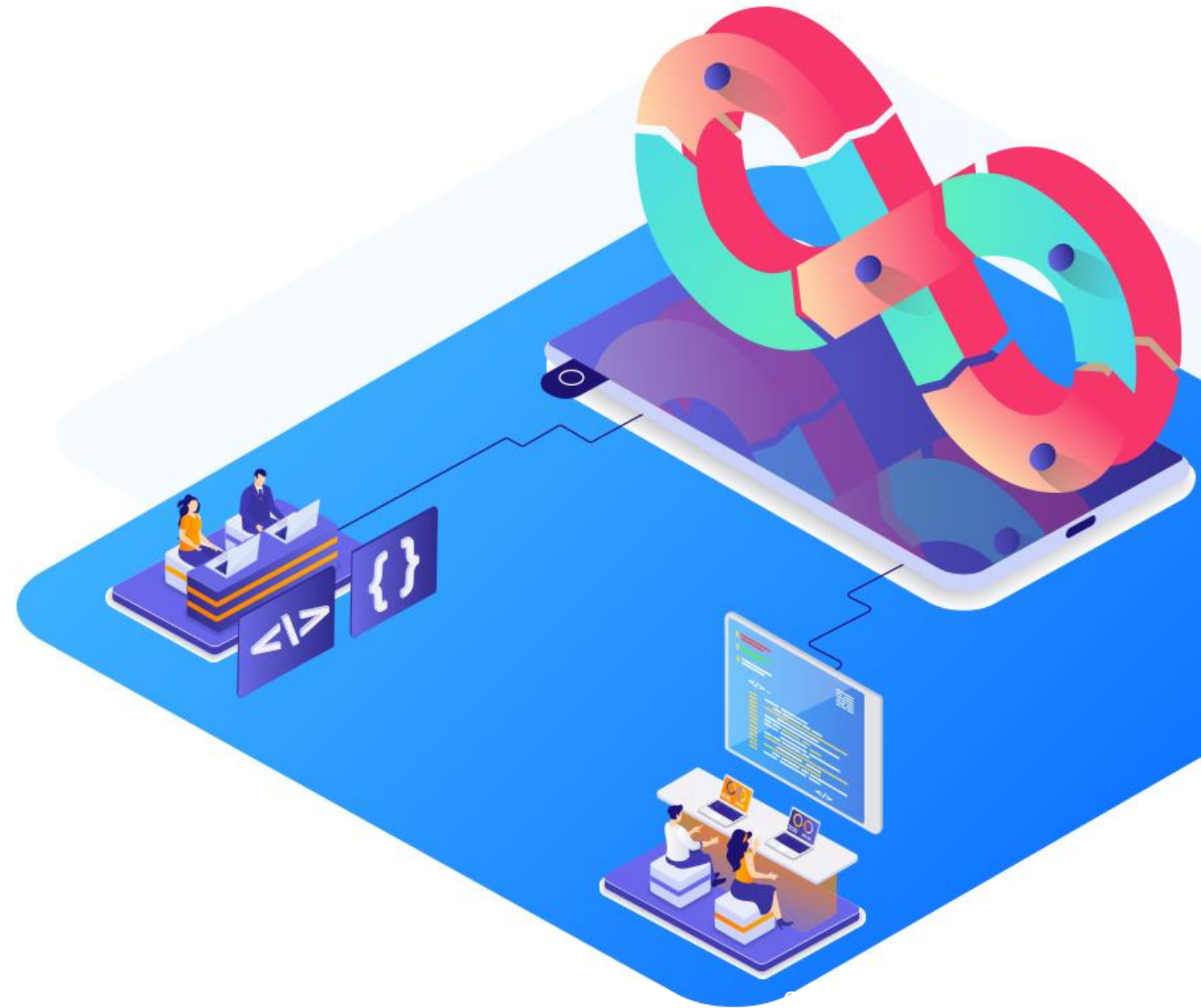


## Containerization with Docker



# Security in Docker



# Learning Objectives

By the end of this lesson, you will be able to:

- 🔗 Implement the use of non-root users in Docker containers to enhance security and mitigate risks associated with running processes as the root user
- 🔗 Perform vulnerability scan on container images using Trivy to identify and mitigate security risks in containerized applications
- 🔗 Utilize Snyk as a SAST tool for assessing Docker images for early vulnerability detection within the CI/CD pipeline
- 🔗 Set up a DAST scan on a running Docker container using OWASP ZAP to assess the security of dockerized applications dynamically





# Docker Security

# Docker Security

Docker security is crucial for managing containerized applications, particularly in production environments where vulnerabilities can have serious impacts.

The intrinsic security of kernel and its support for **namespaces** and **cgroups**

The vulnerable surface on the Docker daemon

Things to consider when reviewing Docker security:

The loopholes in the container configuration profile

The kernel's **hardening** security features and how they interact with containers

# Kernel Namespace

Docker uses kernel namespaces to isolate containers from each other and the host system, ensuring that processes in one container cannot interfere with those in another.

## Types of namespaces

- **The pid namespace:** Isolates the process ID
- **The net namespace:** Manages network interfaces
- **The ipc namespace:** Manages access to IPC resources
- **The mnt namespace:** Manages filesystem mount points
- **The uts namespace:** Isolates kernel and version identifiers

# Kernel Namespace: Security Implications

The following are the ways by which the kernel namespace provides Docker security:

Process  
isolation

Reduces the risk of process-level attacks, such as privilege escalation or unauthorized process manipulation

Network  
isolation

Mitigates the risk of network-based attacks

File system  
isolation

Prevents file system attacks, such as unauthorized data access or modification

User  
isolation

Maps the root user in a container to a non-root user on the host, keeping attackers non-privileged even with root access

# Control Groups

Control Groups, or cgroups, are a fundamental feature of Linux containers that manage and allocate system resources like CPU, memory, and disk I/O among containers.

## Key features of control groups:

- Isolate resources to prevent one container from affecting others
- Track and report resource usage for monitoring and optimization
- Assign different resource priorities to containers based on importance
- Prevent resource exhaustion by limiting container resources
- Customize resource groups and policies for optimal resource management



# Control Groups: Security Implications

The following are the ways by which control groups enhance Docker security:

## Mitigate DoS attack

They limit a container's resource usage (CPU, memory, disk I/O), preventing any single container from exhausting system resources and causing a denial-of-service (DoS) attack on the host system.

## Provide controlled environment

They allow administrators to run untrusted or unstable applications in a controlled environment, limiting their ability to harm the host system by consuming excessive resources.

# Control Groups: Security Implications

## Enhance shared environment security

They ensure that one container cannot negatively impact others by overusing shared resources, thus enhancing security and fairness across users.

## Reduce attack surface

If a container is compromised, cgroups limit its impact by restricting resource usage, reducing the overall attack surface.

In summary, cgroups secure Docker environments by managing and limiting resource usage, protecting against DoS attacks, ensuring system stability, and providing a safer container execution environment.

# Docker Daemon

It helps the Docker to allow the user to share a directory between the Docker host and a guest container without limiting the access rights of the container.

## Key features of Docker daemon:

- It is responsible for creating, managing, and monitoring Docker containers.
- It manages Docker images, including pulling images from a registry and building new images.
- It configures and manages container networking.
- It manages storage for containers, including creating and managing volumes, bind mounts, and tmpfs mounts.

# Docker Daemon Attack Surface

It refers to the potential vulnerabilities and points of exposure that could be exploited by attackers.

## Main aspects of Docker daemon attack surface

Compromised root privileges

Unauthorized socket access

Compromised remote REST API access

Incorrect container configurations

Malicious third-party integrations

Improper monitoring and logging

Enforcing strict access controls, securing communication channels, regularly updating Docker, and carefully managing configurations and container privileges can prevent these vulnerabilities.

# Linux Kernel Capabilities

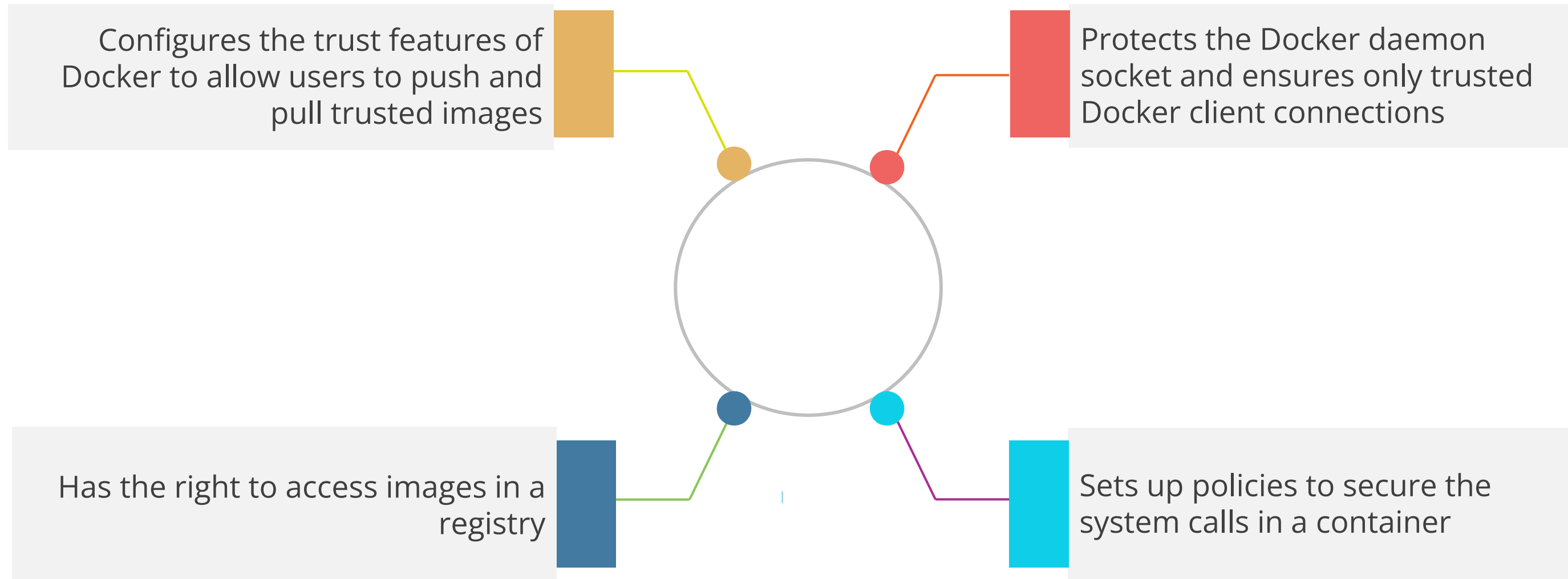
Docker runs the containers with certain restricted capabilities by default. This means the root capabilities are not provided to all processes operating inside a container.

For instance, it is possible to:

- deny all **mount** operations
- deny access to raw sockets
- deny access to some filesystem operations, like creating new device nodes, changing the owner of files, or altering attributes
- deny module loading

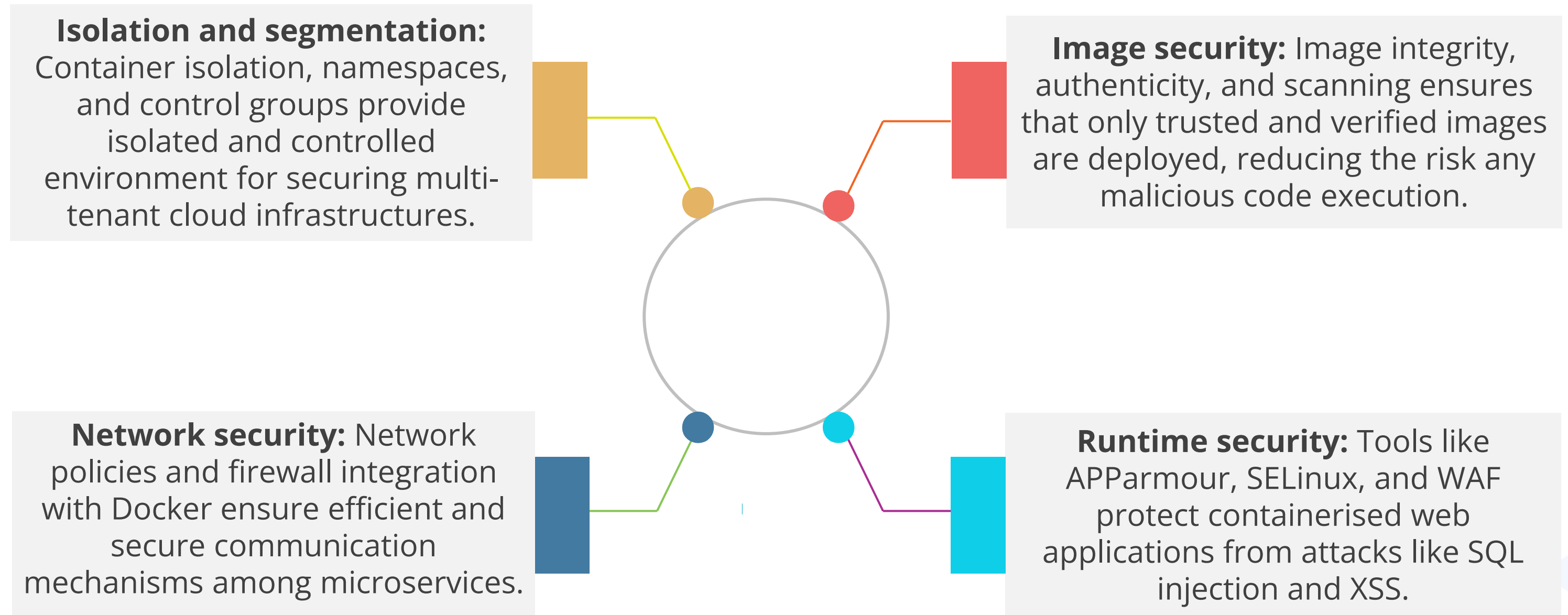
# Default Engine Security

The following are the key security features of Docker's default engine security:



# Additional Security Measures

Here are the ways Docker security is implemented in a cloud infrastructure:



## Assisted Practice



### Building a secure Docker container

Duration: 10 Min.

#### Problem statement:

You have been assigned a task to secure Docker containers by granting access to a non-root user within the container to mitigate the risks associated with running processes as the root user.

#### Outcome:

By completing this demo, you will be able to build and run a secure Docker container by creating a Dockerfile with a non-root user, building the image, and verifying that the container runs with restricted user privileges, enhancing security.

**Note:** Refer to the demo document for detailed steps:  
01\_Building\_a\_Secure\_Docker\_Container



# Assisted Practice: Guidelines



Steps to be followed:

1. Create a Dockerfile with a non-root user
2. Build the Docker image
3. Create and run a Docker container

## Quick Check



You are tasked with securing a Docker container that runs a critical application in your organization. To enhance the security of a Docker container, which action best reduces the risk of privilege escalation and limits resource consumption?

- A. Running the container with the **--privileged** flag
- B. Configuring the container to run as a non-root user and applying control groups
- C. Running the container as the root user for easier permissions management
- D. Disabling Docker Content Trust to speed up deployment



# **Docker Content Trust**

# Docker Content Trust

Docker content trust (DCT) uses digital signatures to verify the integrity and authenticity of images from remote Docker registries, enabling client-side or runtime verification of image tags.

## Docker image tags and DCT

- In Docker, an image is identified by a combination of a repository name and a tag, formatted as **[REGISTRY\_HOST[:REGISTRY\_PORT]/]REPOSITORY[:TAG]**
- When DCT is enabled, image publishers can sign specific tags within a repository using a set of cryptographic keys.
- These signatures verify the integrity and authenticity of the image associated with that tag, ensuring that the image has not been tampered with.

# Signing Docker Images with DCT

Docker content trust (DCT) allows users to sign and verify Docker images using the **\$docker trust** command.

The process uses the Notary server to generate and manage keys that sign image tags. Once a tag is signed, DCT ensures that only trusted images can be pushed, pulled, or run.

To sign an image:

Generate a  
delegation  
key pair

Add the private  
key to the local  
Docker trust store

Sign the image  
tag using the  
**docker trust sign**  
command

Push the signed  
image to the  
registry

# Docker Content Trust Signature Verification

It is a security feature within Docker that ensures the authenticity and integrity of Docker images by requiring that only signed images can be run by the Docker Engine.

It is implemented by:

- Integrating directly into the Docker Engine **dockerd** binary

- Configuring Docker's **daemon.json** file to enforce trust pinning

# Docker Content Trust Signature Verification

The security implications of Docker content trust (DCT) signature verification are significant in enhancing the overall security of Docker environments.

Protection against tampering

DCT signature verification ensures that Docker images remain unaltered after being signed by the publisher.

Verification of source

It confirms the identity of the image publisher, ensuring that images are pulled from trusted sources only.

Reduction of attack surface

It reduces potential entry points for attackers attempting to inject malicious code through unauthorized images.

## Quick Check



You are tasked with ensuring that only trusted Docker images are deployed in your organization's production environment. Which of the following actions best achieves this goal while verifying the integrity and authenticity of the images?

- A. Disabling DCT to simplify the image push process
- B. Using DCT to sign images and enforce signature verification before deployment
- C. Pushing unsigned images directly to the production environment for faster deployment
- D. Running Docker containers in privileged mode to avoid permission issues during deployment





## **Sign a Docker Image**

## Sign an Image

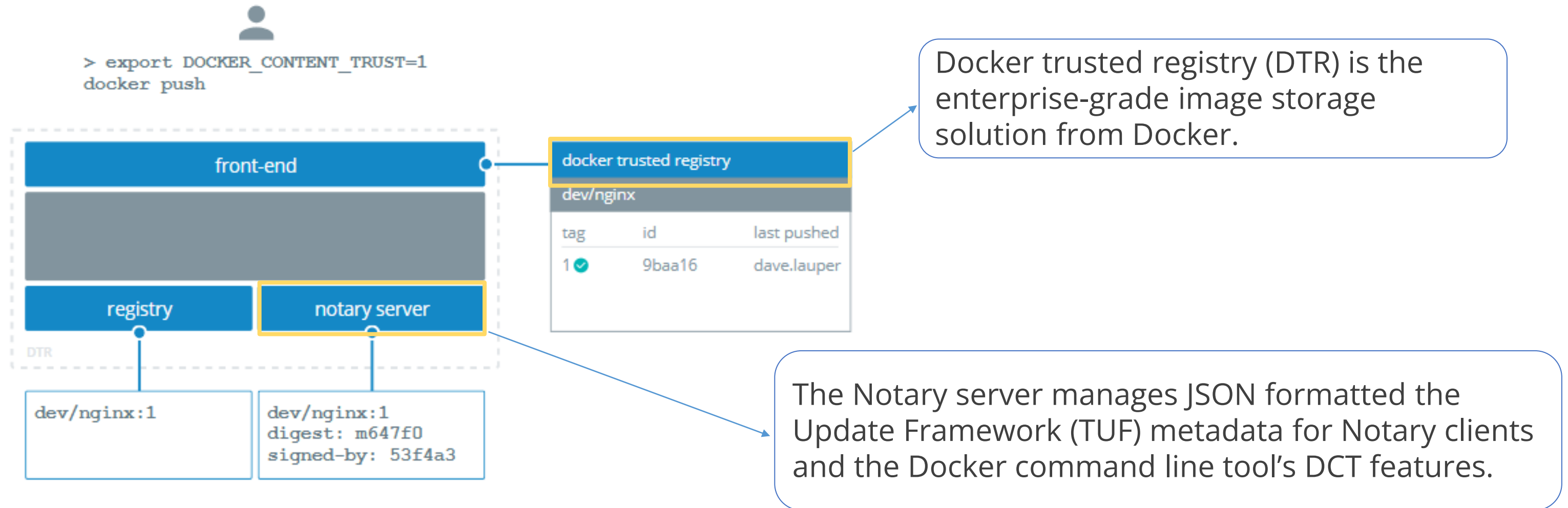
The user can configure the Docker CLI to sign images pushed to the Docker trusted registry (DTR), enabling anyone who pulls the image to verify its authenticity.

To sign an image, the user can run:

```
export DOCKER_CONTENT_TRUST=1  
docker push <dtr-  
domain>/<repository>/<image>:<tag>
```

The command pushes the image to DTR and generates trust metadata, including public and private key pairs. These keys are used to sign the trust metadata, which is then sent to the Notary server integrated within DTR for secure storage and verification.

# Sign an Image



# Image Security Best Practices

Prefer minimal base images

Sign and verify images to mitigate MITM attacks

Find, fix, and monitor for open-source vulnerabilities

Don't leak sensitive information to docker images

Create a dedicated user with minimal permissions

Use fixed tags for immutability

Use **COPY** instead of **ADD**

Use labels for metadata

Use multi-stage builds for small secure images

Use a linter

## Quick Check



You are responsible for ensuring the security of Docker images in your organization's DTR. Which action best practices secure your images before deployment?

- A. Sign Docker images with DCT and push them to DTR
- B. Use the **--privileged** flag to run the images with elevated permissions
- C. Skip image signing and push images directly to DTR for faster access
- D. Rely solely on the base image's security without additional measures.



## **DevSecOps Implementation in Docker**

# DevSecOps in Docker: Introduction

DevSecOps is a combination of Development, Security, and Operations. It is an approach that integrates security practices within the DevOps process.



DevSecOps in Docker is about embedding security into every stage of the container lifecycle, from development to deployment.

# Why Is DevSecOps Important in Docker?

The following are the reasons for implementing DevSecOps in Docker:

## Managing risks

Companies can proactively manage risks early by integrating security checks, such as SAST and DAST scans, into the CI/CD pipeline rather than reacting to them after deployment.

## Streamlining security solutions

Teams can avoid delays caused by last-minute security reviews and rework by integrating security into the development workflow..

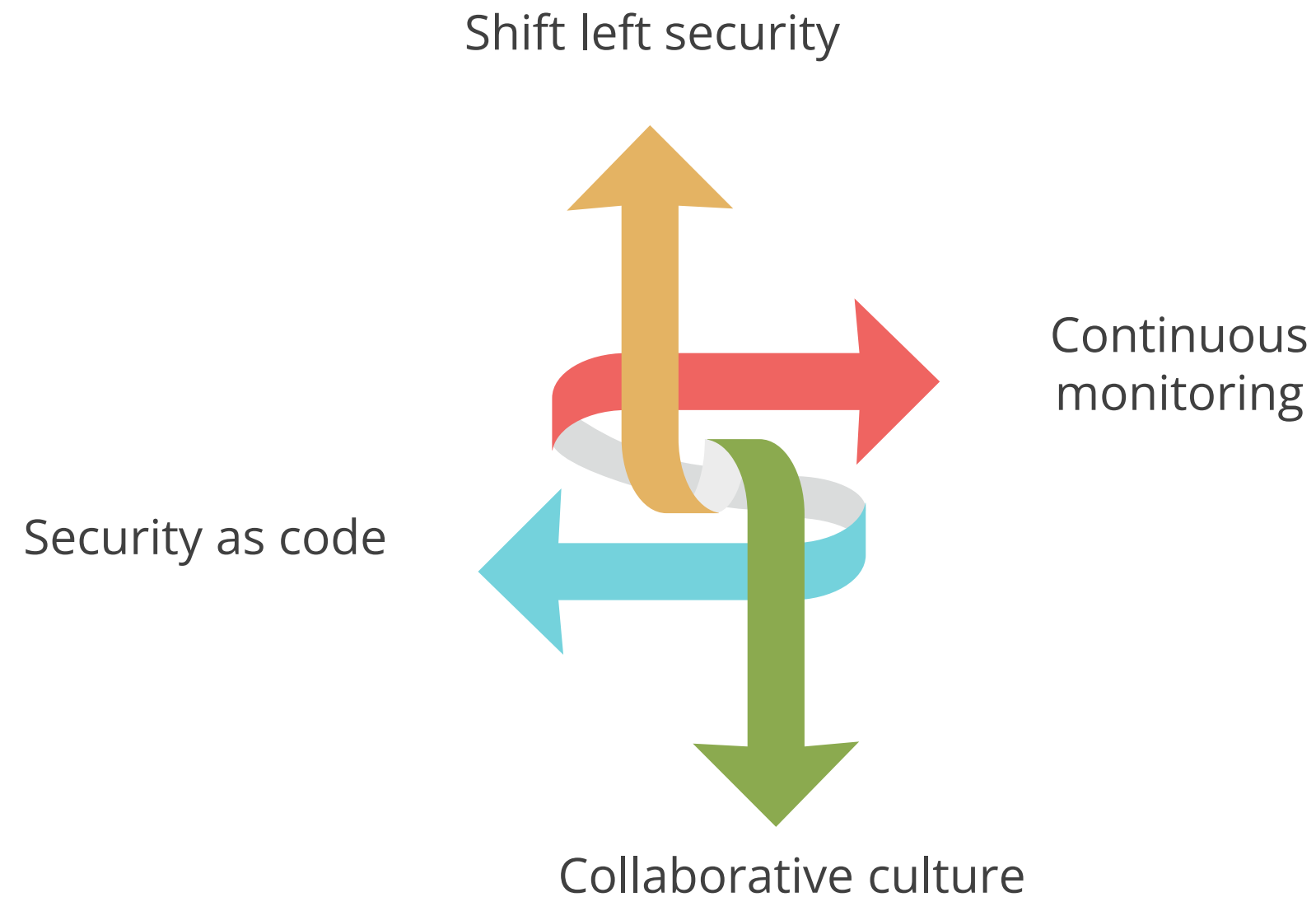
## Adhering to compliance

Organizations can ensure compliance with security standards and regulations by consistently applying security practices throughout the development lifecycle.



# Applying DevSecOps Principles in Docker

The following DevSecOps principles are applied to Docker to ensure security:



# DevSecOps Security Check Stages

DevSecOps integrates security throughout the entire DevOps workflow. This proactive approach helps to identify and fix vulnerabilities early, reducing the risk of security breaches.

Following are the stages to implement DevSecOps to ensure security throughout the development process:

Secrets  
analysis

Static  
application  
security  
testing  
(SAST)

Dynamic  
application  
security  
testing  
(DAST)

Runtime  
application  
self-protection  
(RASP)

# Trivy: Image Vulnerability Scanner

It is an open-source tool designed to scan Docker containers for vulnerabilities. It can identify security flaws in Docker images, enabling shift left, security as code, and continuous monitoring.



It leverages various vulnerability databases, including CVE, NVD, and Red Hat's Security Data API, to detect and report potential risks, providing actionable guidance to help mitigate these issues.

# Trivy: Image Vulnerability Scanner

Some key features of Trivy are:



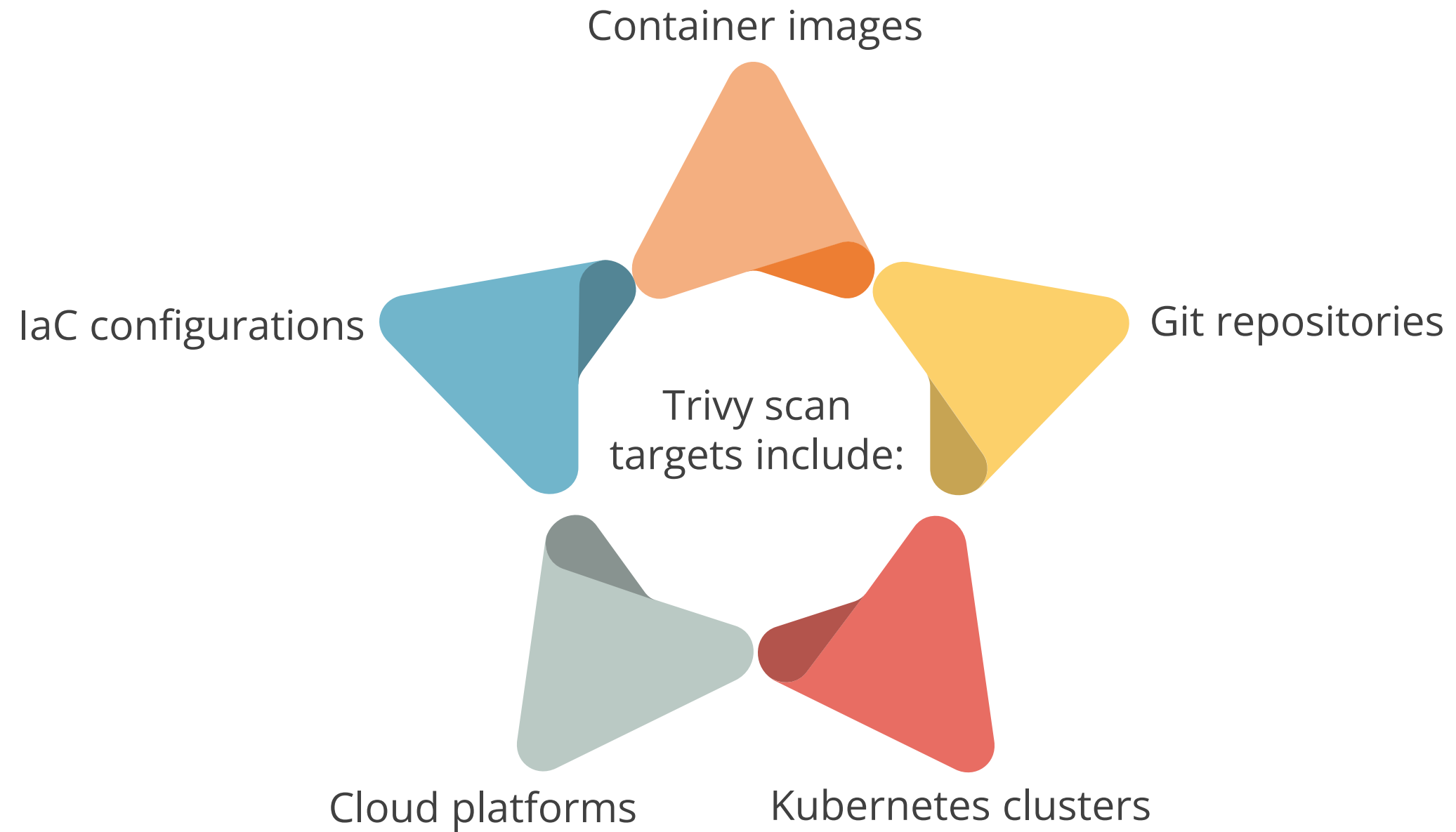
It can support Infrastructure as Code file scanning, helping to secure the infrastructure configuration early in the development lifecycle.

It can generate a software bill of materials (SBOM), offering detailed insights into software components, which is essential for supply chain security.

It can identify hard-coded secrets in code, containers, and repositories, such as passwords and API keys, which are common security risks.

It can scan in diverse environments, including public and private registries, local filesystems, and multiple container formats.

# Trivy: Image Vulnerability Scanner



## Assisted Practice



### Checking vulnerabilities using Trivy

Duration: 10 Min.

#### Problem statement:

You have been assigned a task to scan container images for vulnerabilities using Trivy for identifying and mitigating security risks and ensuring that containerized applications are secure.

#### Outcome:

By completing this demo, you will be able to install Trivy and scan container images for vulnerabilities, identifying potential security risks to ensure the safety of containerized applications.

**Note:** Refer to the demo document for detailed steps:  
02\_Checking\_Vulnerabilities\_Using\_Trivy

# Assisted Practice: Guidelines



Steps to be followed:

1. Install Trivy
2. Scan the vulnerabilities using Trivy

# SAST Scan for Docker Containers

Static application security testing (SAST) is one of the stages of DevSecOps implementation, focusing on scanning the source code, binaries, and configuration files of the Docker containers for security risks.

SAST tools can be integrated with Docker to perform the following:

**Early vulnerability detection:** Detects security vulnerabilities within the application's source code before it is containerized

**CI/CD pipeline integration:** Scans for vulnerabilities in code every time before it is built into a Docker image

**Compliance and reporting:** Provides a detailed report that helps in maintaining compliance with security standards and regulations



## Snyk: SAST Tool

Snyk is a developer security platform that helps software development teams find and fix vulnerabilities across various aspects of their applications.



It can proactively help find and fix vulnerabilities for **NPM, Maven, NuGet, RubyGems, containers, and infrastructure as code.**

## Assisted Practice



### Performing SAST for a Docker image using Snyk CLI

Duration: 10 Min.

#### Problem statement:

You have been assigned a task to download, install, and configure the Snyk command line interface (CLI) to perform SAST scan for a Docker image, enabling automatic vulnerability detection for enhanced project security.

#### Outcome:

By completing this demo, you will be able to install and configure the Snyk CLI to perform a SAST scan on a Docker image, enabling automated vulnerability detection for enhanced security.

**Note:** Refer to the demo document for detailed steps:  
03\_Performing\_SAST\_for\_a\_Docker\_Image\_Using\_Snyk\_CLI

# Assisted Practice: Guidelines



Steps to be followed:

1. Download and install the Snyk CLI
2. Authenticate the Snyk CLI
3. Scan a Docker image

# DAST Scan for Docker Containers

As a part of DevSecOps implementation, dynamic application security testing (DAST) involves scanning the running containers to identify vulnerabilities that could be exploited in a live environment.

DAST tools can be integrated with Docker to perform the following:

**Real-time analysis:** Identifies runtime vulnerabilities that static analysis might not detect

**Attack simulation:** Simulates real-world attacks to find security weaknesses, making it critical for security testing strategy

**Automation:** Ensures continuous security testing and helps maintain a secure production environment within CI/CD pipelines

# OWASP ZAP: DAST Tool

OWASP ZAP is a free and open-source web application security scanner developed and actively maintained by a dedicated international team of developers.



**OWASP**  
Zed Attack Proxy

It is used to find vulnerabilities in web applications, identify misconfigurations, and test web functionality. It supports various protocols such as HTTP, HTTPS, SOAP, REST, FTP, and FTPS.

## Assisted Practice



### Performing DAST for a Docker Container Using OWASP ZAP

Duration: 10 Min.

#### Problem statement:

You have been assigned a task to demonstrate the deployment and utilization of a dynamic application security testing (DAST) tool using Docker.

#### Outcome:

By completing this demo, you will be able to perform dynamic application security testing (DAST) on a Docker container using OWASP ZAP for identifying vulnerabilities and generating a security report.

**Note:** Refer to the demo document for detailed steps:  
04\_Performing\_DAST\_for\_a\_Docker\_Container\_Using\_OWASP\_ZAP

# Assisted Practice: Guidelines




Steps to be followed:


1. Set up and start the ZAP Docker container
2. Run the security scan and retrieve the report

# Real-Life Impacts of Docker Security


The security features of Docker are crucial for businesses using containerized environments, as they enhance application security and ensure efficient deployments in the following ways:



Docker's default security engine provides robust isolation and resource management through kernel namespaces and cgroups, which helps businesses safeguard sensitive data, maintain service availability, and reduce the risk of costly security breaches.




Kernel namespaces isolate containers from each other and the host system, minimizing the risk of security breaches and ensuring that vulnerabilities in one container do not compromise the entire system.




Control Groups enforce resource limits and isolation for containers, preventing any single container from consuming excessive resources and potentially causing a denial-of-service (DoS) attack.




# Real-Life Impacts of Docker Security



Linux kernel capabilities allow fine-grained control over container privileges, enabling you to grant only the necessary capabilities to each container, thereby reducing the potential attack surface.



Signing Docker images with DCT guarantees that only authorized and unaltered images are used, adding a critical layer of security to the software supply chain.



The vulnerability scan of Docker images is a part of DevSecOps that identifies and mitigates security risks early, ensuring that only secure images are deployed.



The SAST scan identifies vulnerabilities in source code before deployment, allowing developers to fix issues early in the development lifecycle.



The DAST scan tests running applications for security vulnerabilities, uncovering potential exploits that could be targeted in a live environment.

## Quick Check



As part of implementing DevSecOps in your Docker environment, you need to ensure that your containerized applications are secure from vulnerabilities. Which sequence of actions best aligns with DevSecOps principles to achieve this?

- A. Run a DAST scan with OWASP ZAP, then sign the images before pushing them to production
- B. Skip SAST scanning to save time, then run Trivy and DAST scans just before deployment
- C. Perform SAST scanning with Snyk, then run Trivy for vulnerability scanning, followed by a DAST scan with OWASP ZAP
- D. Deploy the container images to production directly after building to meet release deadlines

# Key Takeaways

- Docker uses kernel namespaces to isolate containers from each other and the host system, ensuring that processes in one container cannot interfere with those in another.
- The cgroups prevent scenarios where one container's resource-intensive processes destabilize the host or other containers, thus maintaining overall system stability.
- Docker content trust (DCT) allows users to sign and verify the Docker images using the **\$docker trust** command.
- Docker content trust (DCT) uses digital signatures to verify the integrity and authenticity of images from remote Docker registries.



# Key Takeaways

- Trivy is an open-source tool to identify security flaws in Docker images, enabling shift left, security as code, and continuous monitoring.
- Snyc is a security tool that can be used with Docker to proactively find and fix vulnerabilities for NPM, Maven, NuGet, RubyGems, containers, and infrastructure as code.
- ZAP is an open-source security web application used to find vulnerabilities in web applications, identify misconfigurations, and test web functionality.





**Thank You**