KodeKloud

Visit www.kodekloud.com to learn more.

Compliance Frameworks

In this section we talk about security compliance frameworks. Now as developers or engineers, I would understand that these may not be the most exciting of topics but I must emphasize how critical they are to our work.

Compliance Frameworks

Personal Info

Health Records

Payment Details

Security compliance frameworks provide the guidelines and standards that help us protect sensitive data such as Personal Info, health records and payment details, to ensure system integrity, and meet legal obligations.

Compliance Frameworks

HIPAA medical records settlement [edit]

In 2017, Mount Sinai West entered into settlement concerning the improper disclosure of patient medical records which was settled as the payment of a levied fine of approximately half-a-million dollars as reported in the medical journal Becker Hospital Review stating: "New York City-based St. Luke's-Roosevelt Hospital Center (Mount Sinai West) will pay $387,200 and implement a corrective action plan as part of a HIPAA settlement to resolve allegations it inappropriately handled a patient's sensitive health information."[20]

FTC charges of privacy violations [edit]

On February 18, 2009, CVS Caremark agreed to settle Federal Trade Commission charges that it failed to take reasonable and appropriate security measures to protect the sensitive financial and medical information of its customers and employees, in violation of federal law. In a separate but related agreement, the company's pharmacy chain also has agreed to pay $2.25 million to resolve Department of Health and Human Services allegations that it violated the Health Insurance Portability and Accountability Act (HIPAA).[69]

In 2019, one year after Allscripts' acquisition of Practice Fusion,[25] a San Francisco-based electronic health records company, the acquisition of which was intended to expand Allscripts' business with independent physicians, Allscripts agreed to settle with the Department of Justice over "potential issues with the EHR vendor's health IT certification and Practice Fusion's compliance with the Anti-Kickback Statute and HIPAA."[26] As part of the settlement, Practice Fusion paid $145 million.

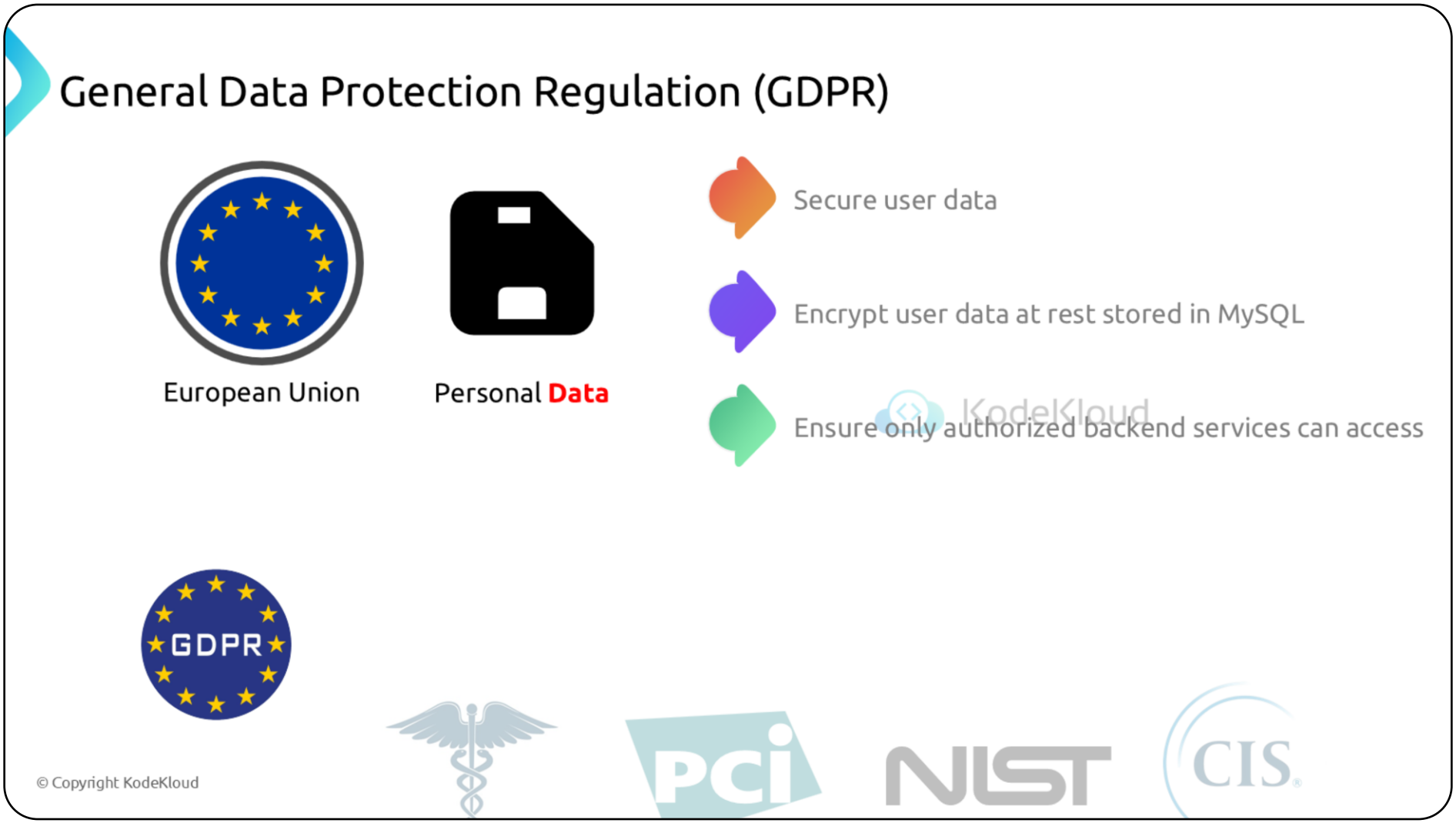
Ignoring them isn't just risky—it can lead to severe consequences like data breaches, legal penalties, and loss of customer trust. So, even if they aren't the most thrilling part of our jobs, understanding these frameworks is essential for building robust and secure systems. Let's dive in and see how they impact our development processes and what we can do to effectively implement them.

For instance, consider the case of these healthcare providers who suffered a data breach because they did not follow HIPAA guidelines. The breach exposed patient records, leading to significant fines and loss of trust.

Compliance Frameworks

To prevent such incidents, organizations need to follow structured guidelines known as compliance frameworks. In this lesson, we will explore various compliance frameworks and understand their importance in securing Kubernetes environments.

Compliance frameworks are guidelines and best practices designed to help organizations meet regulatory requirements and ensure data security and privacy. Adhering to these frameworks not only helps achieve regulatory compliance but also enhances overall security. Key compliance frameworks relevant to Kubernetes environments include GDPR, HIPAA, PCI DSS, NIST, and CIS Benchmarks.

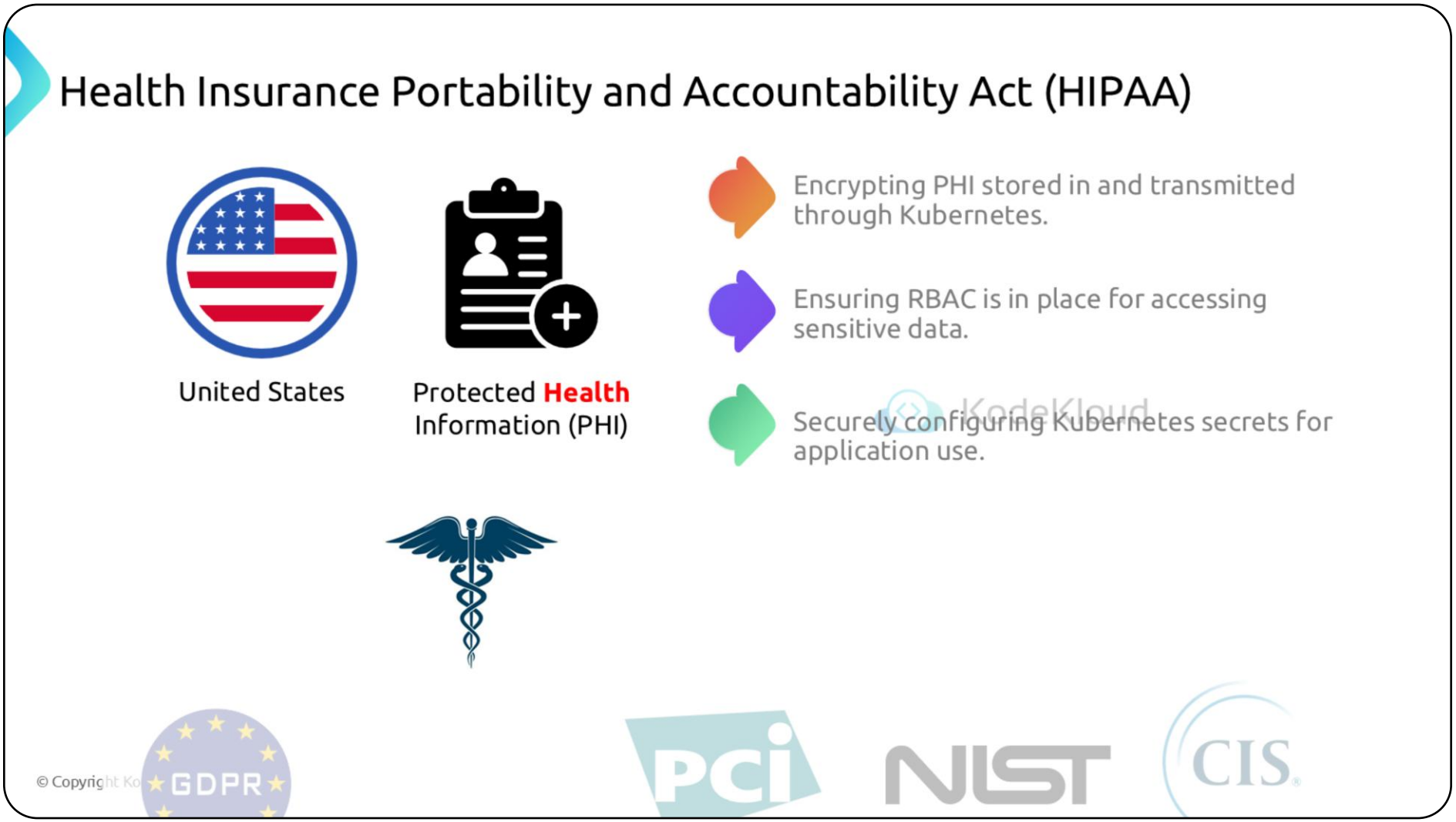


Let's review each of these compliance frameworks at a high level to gain a basic understanding to prepare for the exam.

Starting with, General Data Protection Regulation (GDPR).

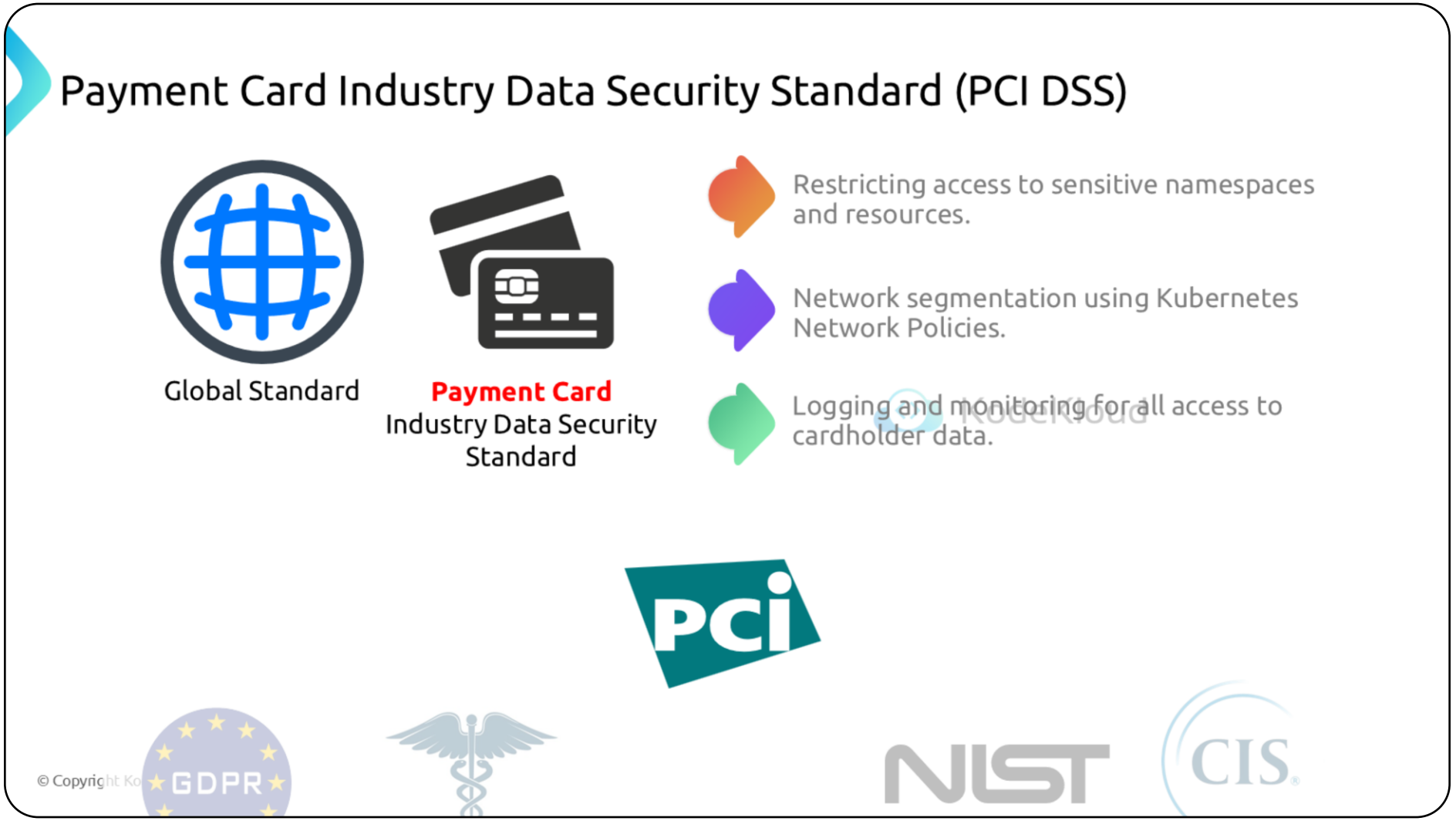
The GDPR is a comprehensive data protection law enacted by the European Union to safeguard individuals' personal data and uphold their privacy rights.

In our scenario, For our web application, ensuring GDPR compliance means securing user data collected through the frontend. This includes encrypting user data stored in the MySQL database and making sure only authorized backend services (Node.js microservices) can access it.



Next, we'll look at Health Insurance Portability and Accountability Act (HIPAA)

HIPAA is a U.S. regulation that protects sensitive patient health information. If our web application handles patient data, we must ensure that all data transfers between the frontend, backend, and database are encrypted using TLS. We also need to implement access controls to restrict unauthorized access to health data. And we want to make sure we securely configuring Kubernetes secrets for application use.



Another compliance framework is Payment Card Industry Data Security Standard (PCI DSS).

PCI DSS protects cardholder data.

If our web application processes payment information, we need to encrypt cardholder data both in transit and at rest. Strong access controls must be in place, and we should regularly monitor and audit access to payment data to comply with PCI DSS.

National Institute of Standards and Technology (NIST)

Originated in the United States ,but is recognized Globally

Cybersecurity

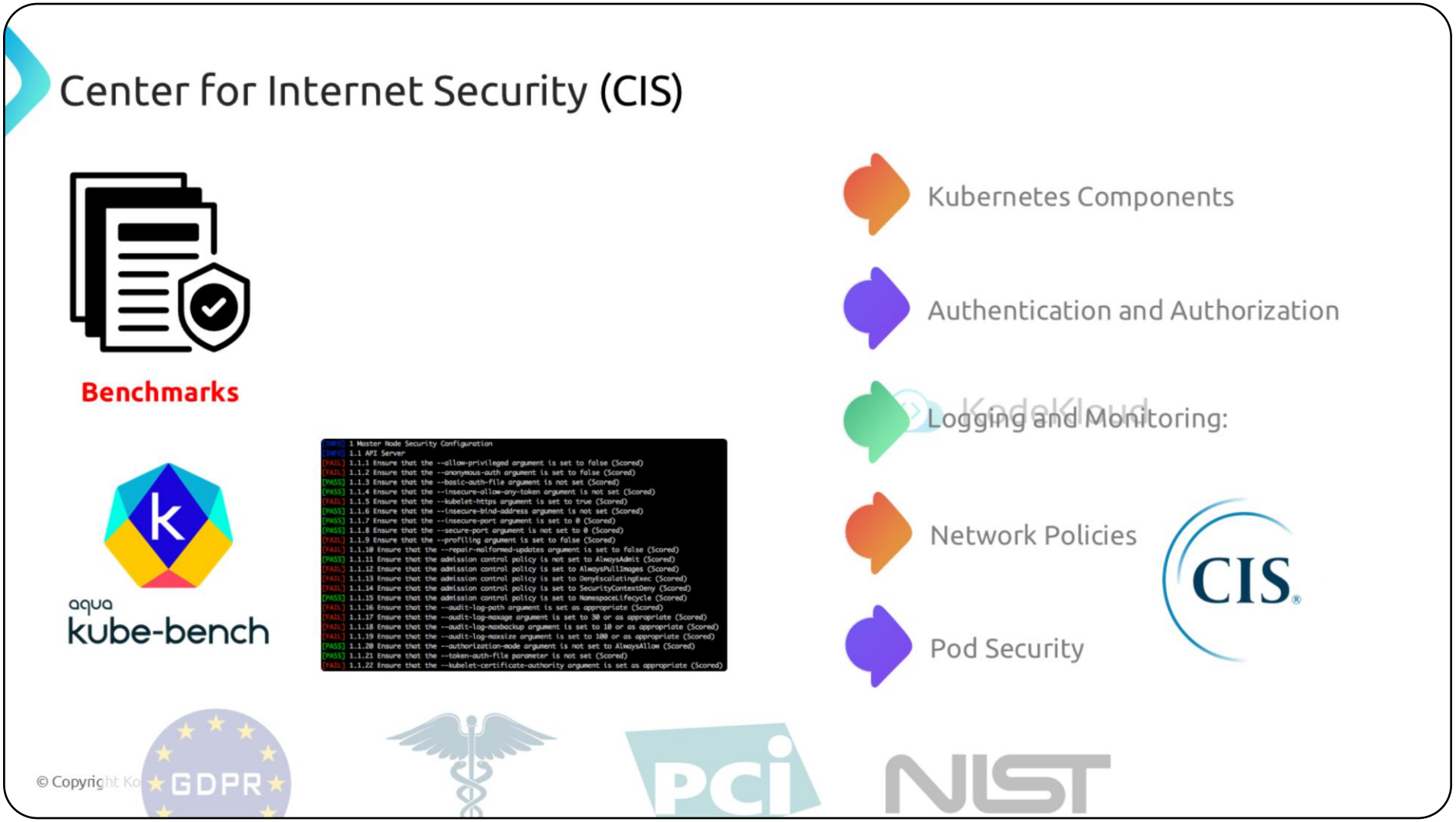
- Conducting regular risk assessments to identify potential vulnerabilities
- Implementing security controls like firewalls, intrusion detection systems
- Regular security audits helps mitigate these risks.

NIST

Next in the list is , National Institute of Standards and Technology (NIST)

NIST provides a framework for improving the security and resilience of information systems.

For our web application, following NIST guidelines means conducting regular risk assessments to identify potential vulnerabilities. Implementing security controls like firewalls, intrusion detection systems, and regular security audits helps mitigate these risks.



Final one in the list, Center for Internet Security (CIS) Benchmarks.

CIS Benchmarks offer best practices for securing IT systems and data, including Kubernetes environments.

Applying CIS Benchmarks to our Kubernetes cluster involves

1.Kubernetes Components:

    1. Secure configuration of the API server, etcd, kubelet, controller manager, and scheduler.

1.Authentication and Authorization:

    1. Enforcing RBAC and other access controls.

2.Logging and Monitoring:

    1. Ensuring auditing and logging are enabled to track cluster activities.

3.Network Policies:

    1. Enforcing network segmentation between workloads.

4.Pod Security:

    1. Using Pod Security Admission or equivalent tools to enforce pod-level security settings.

For example tools like Kube-bench by Acqua security can help check whether Kubernetes is deployed securely by running the checks documented in the CIS Kubernetes Benchmark.

Center for Internet Security (CIS)

Benchmarks

aqua
kube-bench

```
[INFO] 1 Master Node Security Configuration
[INFO] 1.1 API Server
[FAIL] 1.1.1 Ensure that the --allow-privileged argument is set to false (Scored)
[FAIL] 1.1.2 Ensure that the --anonymous-auth argument is set to false (Scored)
[PASS] 1.1.3 Ensure that the --basic-auth-file argument is not set (Scored)
[PASS] 1.1.4 Ensure that the --insecure-allow-any-token argument is not set (Scored)
[FAIL] 1.1.5 Ensure that the --kubelet-https argument is set to true (Scored)
[PASS] 1.1.6 Ensure that the --insecure-bind-address argument is not set (Scored)
[PASS] 1.1.7 Ensure that the --insecure-port argument is set to 0 (Scored)
[PASS] 1.1.8 Ensure that the --secure-port argument is not set to 0 (Scored)
[FAIL] 1.1.9 Ensure that the --profiling argument is set to false (Scored)
[FAIL] 1.1.10 Ensure that the --repair-malformed-updates argument is set to false (Scored)
[PASS] 1.1.11 Ensure that the admission control policy is not set to AlwaysAdmit (Scored)
[FAIL] 1.1.12 Ensure that the admission control policy is set to AlwaysPullImages (Scored)
[FAIL] 1.1.13 Ensure that the admission control policy is set to DenyEscalatingExec (Scored)
[FAIL] 1.1.14 Ensure that the admission control policy is set to SecurityContextDeny (Scored)
[PASS] 1.1.15 Ensure that the admission control policy is set to NamespaceLifecycle (Scored)
[FAIL] 1.1.16 Ensure that the --audit-log-path argument is set as appropriate (Scored)
[FAIL] 1.1.17 Ensure that the --audit-log-maxage argument is set to 30 or as appropriate (Scored)
[FAIL] 1.1.18 Ensure that the --audit-log-maxbackup argument is set to 10 or as appropriate (Scored)
[FAIL] 1.1.19 Ensure that the --audit-log-maxsize argument is set to 100 or as appropriate (Scored)
[PASS] 1.1.20 Ensure that the --authorization-mode argument is not set to AlwaysAllow (Scored)
[PASS] 1.1.21 Ensure that the --token-auth-file parameter is not set (Scored)
[FAIL] 1.1.22 Ensure that the --kubelet-certificate-authority argument is set as appropriate (Scored)
```

Final one in the list, Center for Internet Security (CIS) Benchmarks.

CIS Benchmarks offer best practices for securing IT systems and data, including Kubernetes environments.

Applying CIS Benchmarks to our Kubernetes cluster involves

1.Kubernetes Components:

    1. Secure configuration of the API server, etcd, kubelet, controller manager, and scheduler.

1.Authentication and Authorization:

    1. Enforcing RBAC and other access controls.

2.Logging and Monitoring:

    1. Ensuring auditing and logging are enabled to track cluster activities.

3.Network Policies:

    1. Enforcing network segmentation between workloads.

4.Pod Security:

    1. Using Pod Security Admission or equivalent tools to enforce pod-level security settings.

For example tools like Kube-bench by Acqua security can help check whether Kubernetes is deployed securely by running the checks documented in the CIS Kubernetes Benchmark.

Compliance Frameworks

Framework	GDPR	HIPAA	PCI DSS	NIST	CIS
Purpose	Protects **personal data** of **EU citizens,** emphasizing privacy rights and data governance.	Focuses on safeguarding **patient health information (PHI) through strict data security and access controls.**	Ensures the protection of cardholder data by setting security standards for **payment systems.**	Provides risk-based guidelines and controls to secure **information systems,** reduce threats like **cyberattacks** or natural disasters, and protect privacy.	Helps set secure baselines for systems by providing detailed configuration standards, making it easier to avoid vulnerabilities.
When to Apply	Must be applied to all processes involving personal data to meet regulatory requirements.	Should be applied across development, testing, and production phases to secure **sensitive healthcare data.**	Applicable throughout development, testing, and production to **ensure compliance with payment security standards.**	Most effective during **vulnerability** assessments, risk analysis, and before implementing **security** controls.	Best used during pre-production to validate that system configurations meet security requirements.
Key Focus Areas	Emphasizes minimizing data collection, ensuring lawful processing, and securing stored and transmitted personal data.	Centers on encryption, access management, and incident response to protect healthcare data.	Covers access controls, encryption, vulnerability management, and continuous monitoring of payment systems.	Aims to reduce security risks through risk assessments, secure baselines, and continuous monitoring.	Focuses on configuration management by ensuring systems are aligned with security benchmarks.
How to Use	Regularly review data processing workflows, encrypt sensitive data, and implement access controls to meet GDPR requirements.	Conduct regular risk assessments and enforce strong encryption and access controls to comply with HIPAA.	Follow PCI DSS guidelines to create secure policies for handling payment data and use tokenization.	Use the Risk Management Framework (RMF) to establish secure baselines and address risks. Adopt the Cybersecurity Framework for IT workflows.	Audit your system against **CIS benchmarks** to identify gaps and fix misconfigurations.
Recommended Tools	Tools like **OneTrust** or **TrustArc** help streamline GDPR compliance and manage privacy programs.	Tools like **Compliancy Group** or **Paubox** help achieve HIPAA compliance.	Use **Prisma Cloud** (formerly Twistlock) to manage container security and maintain PCI DSS compliance.	Utilize tools like **NIST SRE Toolkit** for security posture assessments and RMF-based workflows.	Use tools like **Kube-bench** to audit configurations and align with CIS standards.

Summary

01 Compliance frameworks ensure data security and regulatory compliance adherence

02 GDPR, HIPAA, and PCI DSS require encryption and strict access controls

03 NIST and CIS Benchmarks recommend regular security audits and risk assessments

04 Conduct compliance assessments to identify relevant frameworks and security gaps

05 Use monitoring tools like Prometheus and Grafana for continuous compliance

Threat Modelling Frameworks

In this lesson we look at Threat modelling frameworks.

Compliance Frameworks

We discussed about compliance frameworks such as GDPR and these define what needs to be done to meet legal, regulatory or industry standards.

Compliance Frameworks

Compliance Frameworks

Defines what to do

KodeKloud

GDPR mandates that personal data must be secured against unauthorized access, but it doesn't specify how to achieve that.

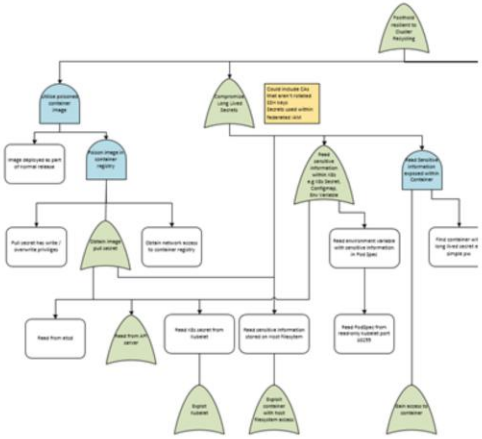
GDPR

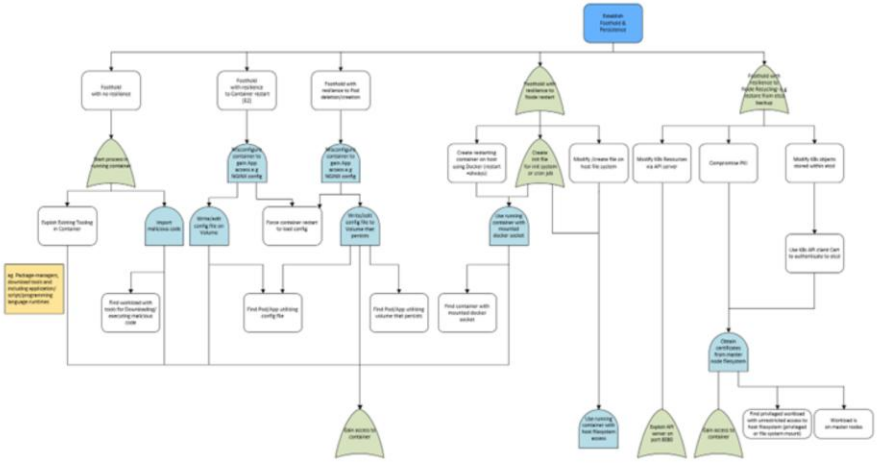
For Example: GDPR mandates that personal data must be secured against unauthorized access, but it doesn't specify how to achieve that.

Threat Modeling Frameworks

Uncover vulnerabilities

Appropriate defenses





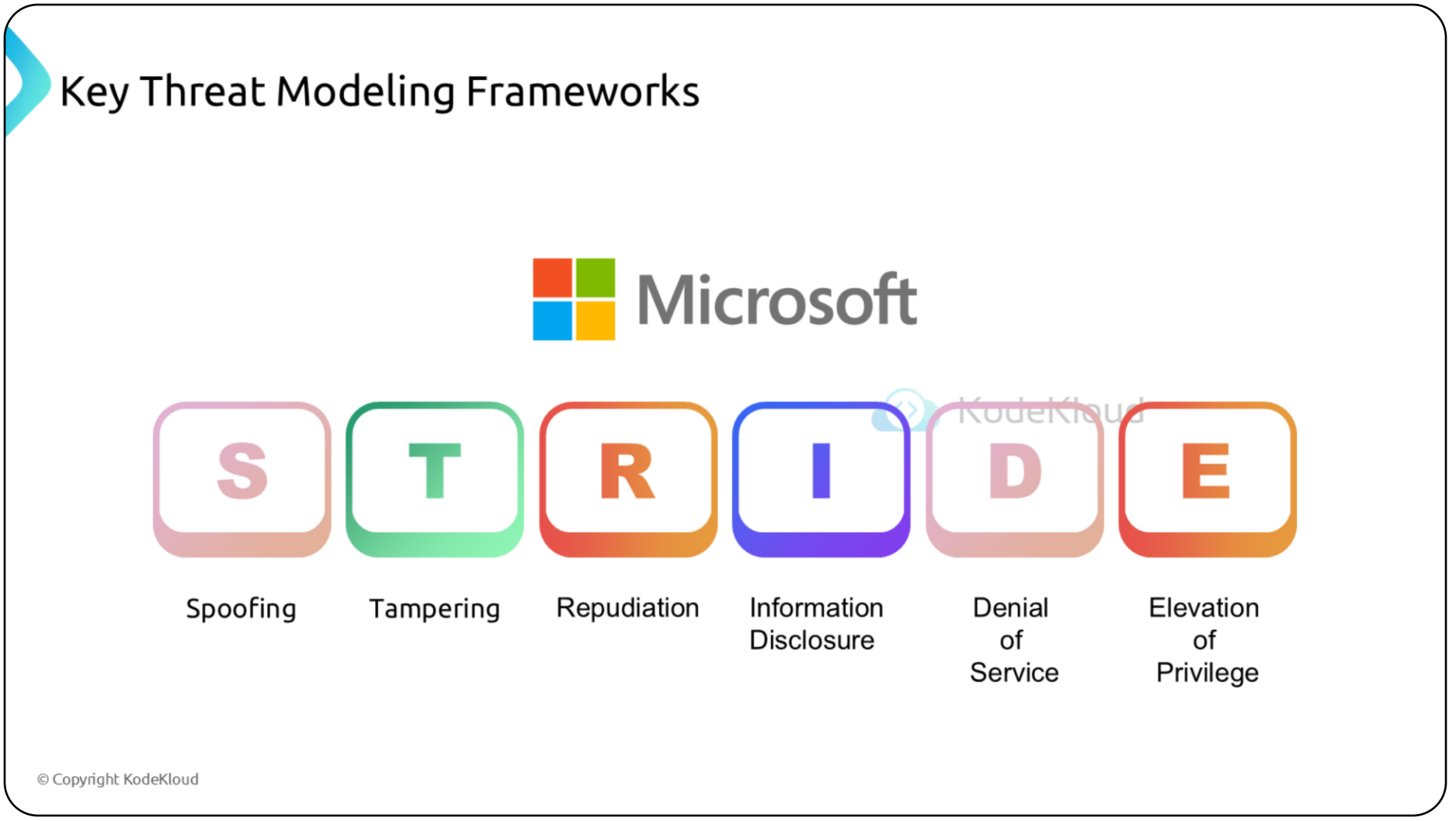
Source: https://github.com/cncf/financial-user-group/blob/main/projects/k8s-threat-model/AttackTrees/EstablishPersistence.md

Threat Modeling Frameworks: Define how to do it by identifying specific threats and suggesting mitigations to secure the system. The attack trees that we discussed in previous section can be used to visualize and analyze different attack paths and scenarios as well!

Threat Modeling Frameworks

Two of the very popular threat modelling frameworks at STRIDE and MITRE ATT&CK that can help you determine how attackers might access sensitive data and what defenses you can implement. We will look at these two frameworks in this lesson.

While stride is a model developed by Microsoft, the MITRE ATT&CK® is a global knowledge base of real-world adversary tactics and techniques, used to build threat models and methodologies across the private sector, government, and cybersecurity industries.



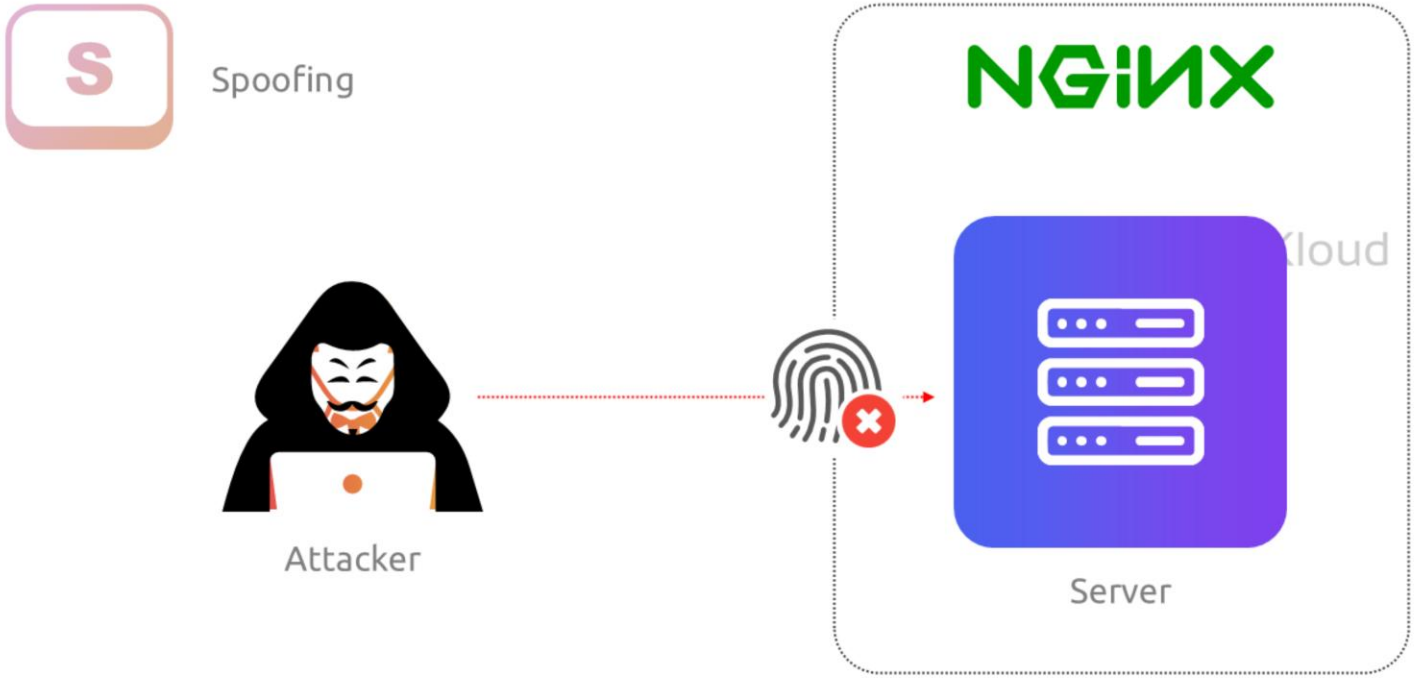
Let's start by looking at STRIDE.

STRIDE

STRIDE is a widely-used threat modeling framework developed by Microsoft that helps identify six categories of threats:

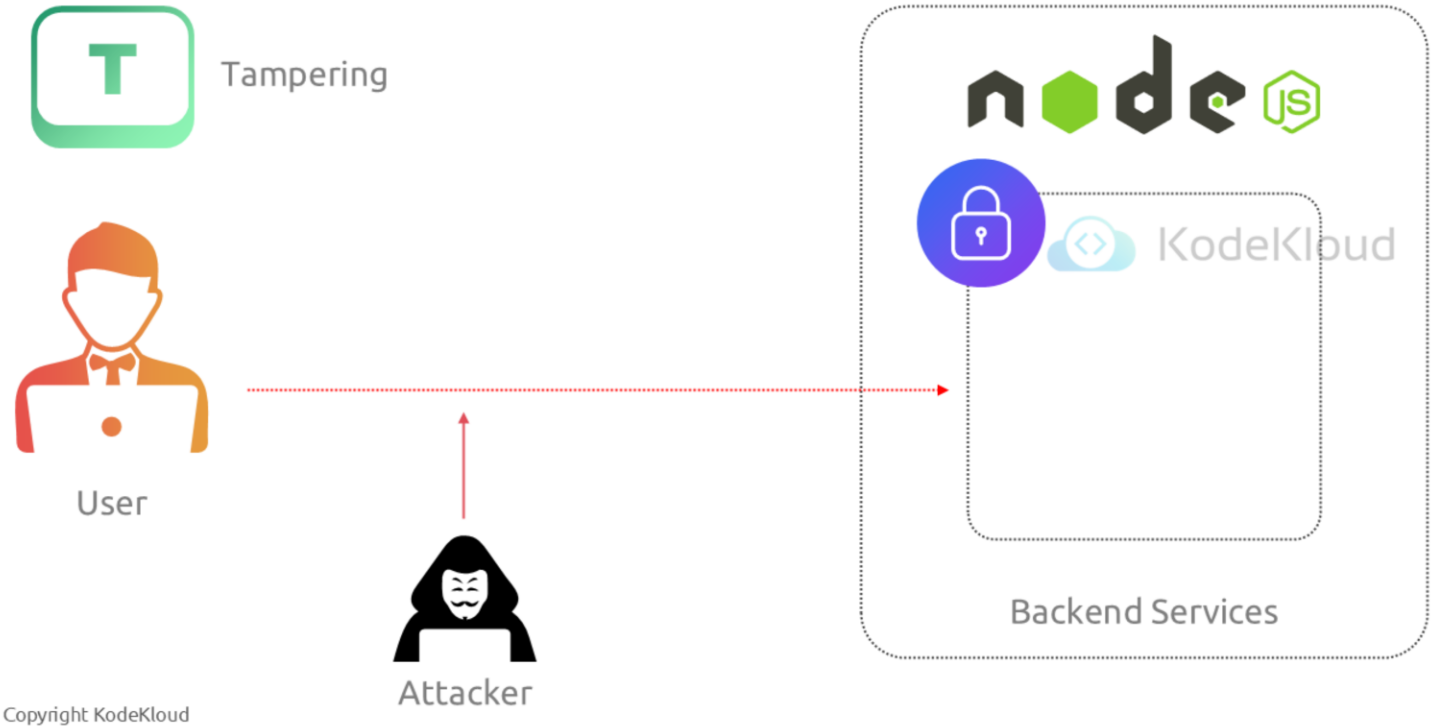
Spoofing, Tampering, Repudiation – I'll tell you what that is in a bit, Information Disclosure, Denial of Service, and Elevation of Privilege. Let's break down STRIDE using our web application as an example.

Key Threat Modeling Frameworks



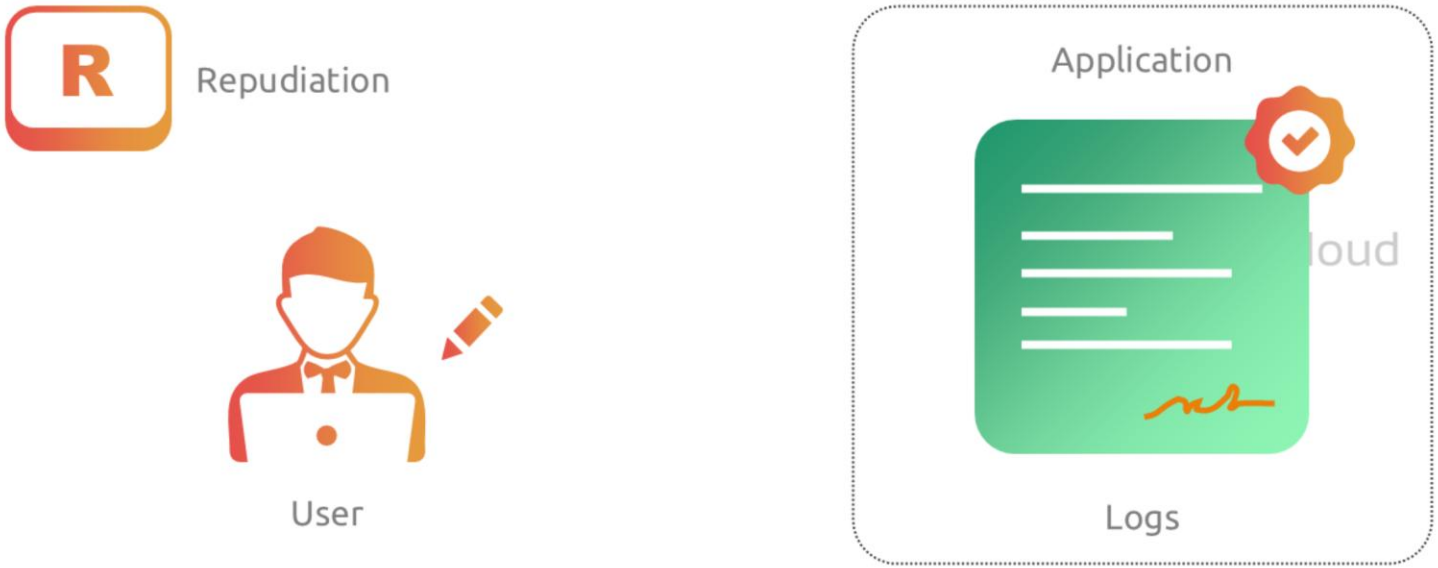
S for Spoofing. An attacker might try to impersonate a legitimate user to access our frontend Nginx server. We can mitigate this by implementing strong authentication mechanisms, such as multi-factor authentication.

Key Threat Modeling Frameworks



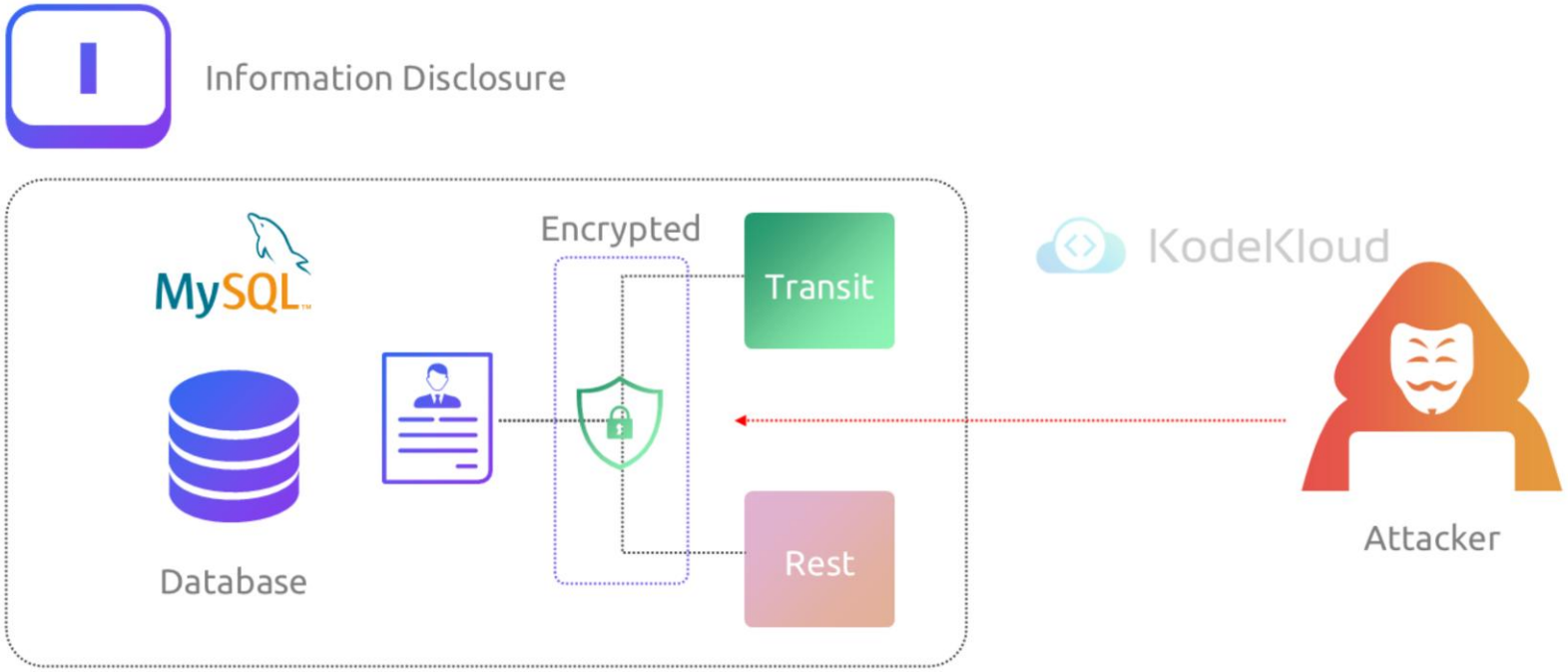
Next T for Tampering. An attacker could attempt to alter data being processed by our Node.js backend services. We can prevent this by ensuring data integrity with encryption and digital signatures.

Key Threat Modeling Frameworks



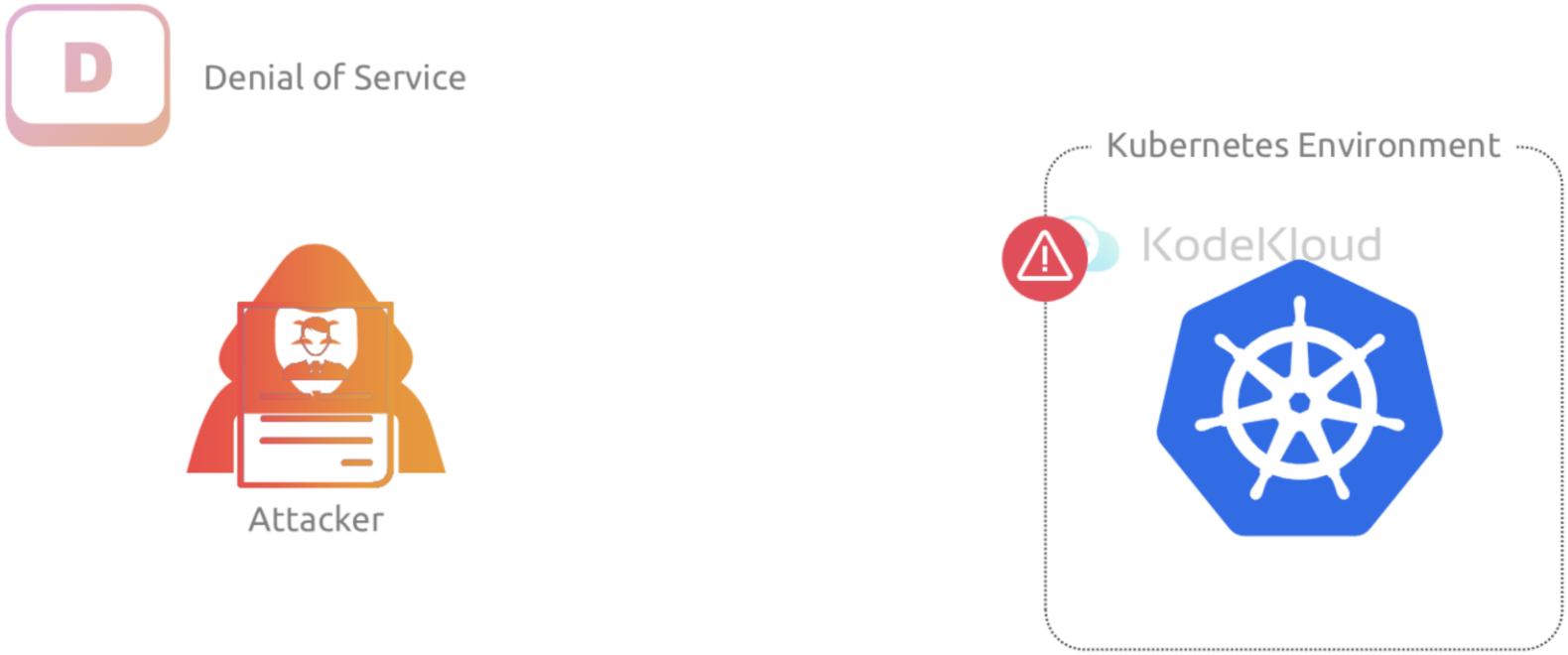
Then Repudiation. Repudiation is Claiming that you didn't do something or were not responsible; can be honest or false. At times users might deny performing certain actions in our application. For example, a user might deny making a transaction or changing their account details. We can address this by keeping comprehensive logs that record user actions and using non-repudiation techniques, such as digital signatures, to verify that an action was performed by the user.

Key Threat Modeling Frameworks



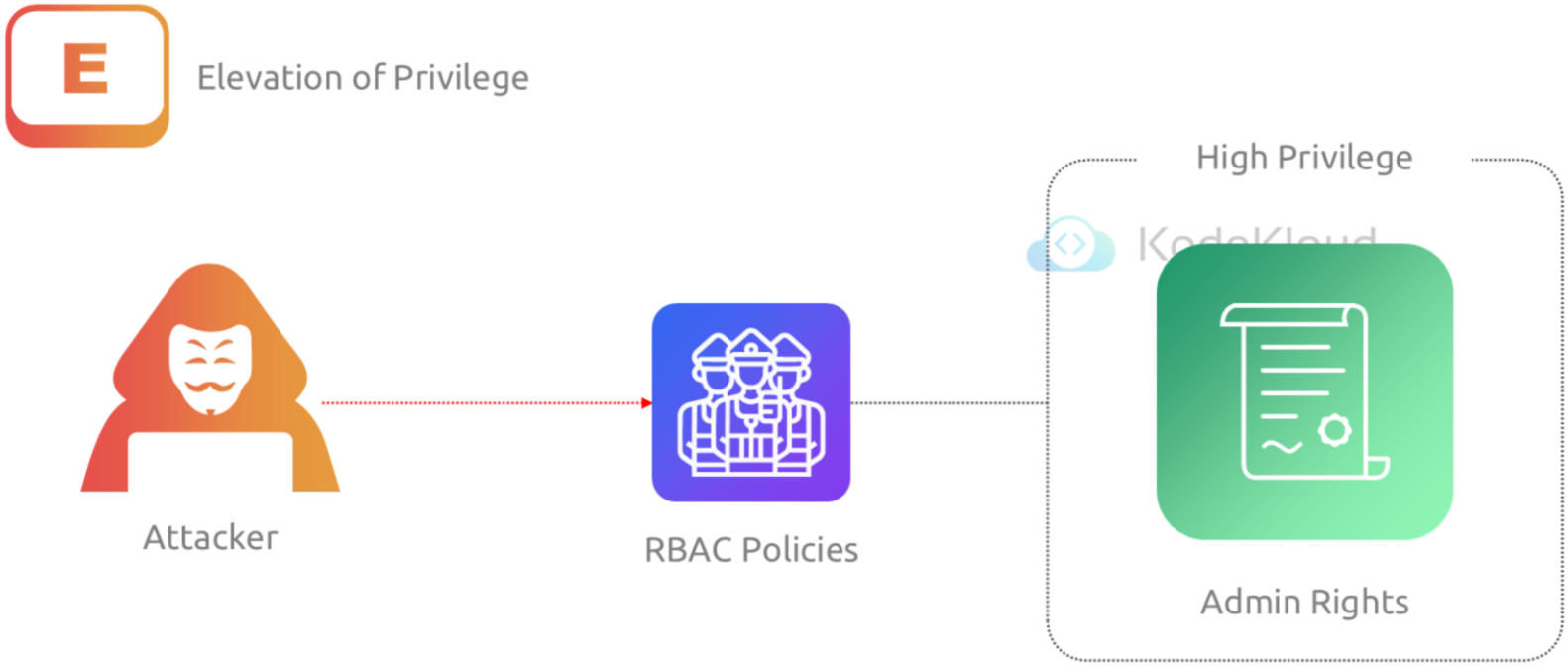
'I' stands for Information Disclosure. This is Someone obtaining information they are not authorized to access. For example in our case, sensitive data in our MySQL database could be exposed. We can prevent this by encrypting data in transit and at rest to ensure that unauthorized users cannot access it.

Key Threat Modeling Frameworks



D for Denial of Service. An attacker might try to overload our system, causing it to crash. We have discussed this in detail in previous sections on threat modeling. We can mitigate this by setting up rate limiting and resource quotas in our Kubernetes environment.

Key Threat Modeling Frameworks



At last ‘E’ for Elevation of Privilege: An attacker might gain unauthorized access to higher privilege levels, such as gaining admin rights to our backend services. This was also covered in previous sections on threat modeling. We can prevent this by implementing strict RBAC policies to limit what each user can do.

MITRE ATT&CK Framework

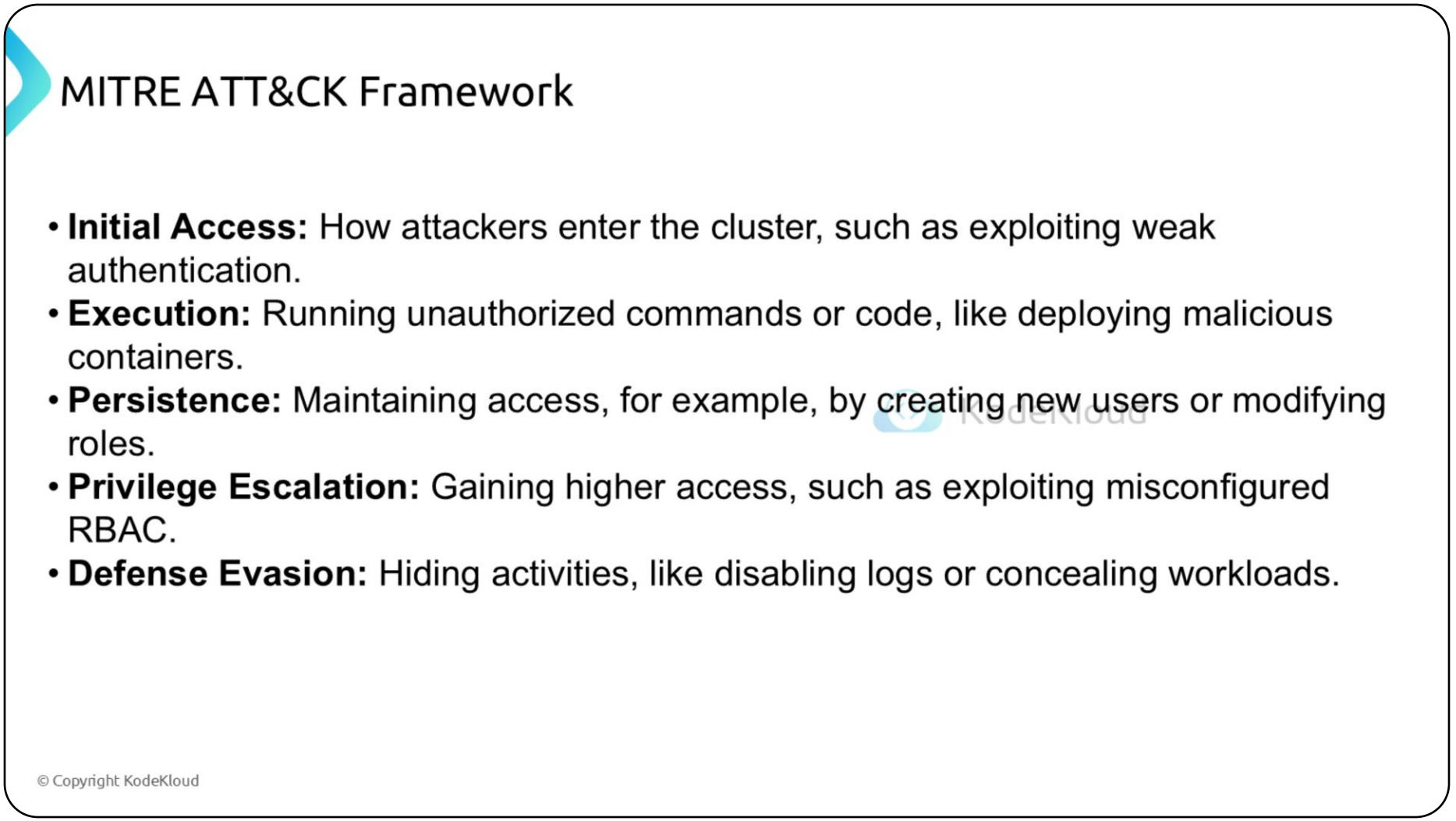
MITRE ATT&CK™

What Attackers aim to do(Tactics)

How they do it (Techniques)

Another very popular framework is the MITRE ATT&CK® Framework, a globally accessible resource that documents real-world adversary tactics and techniques.

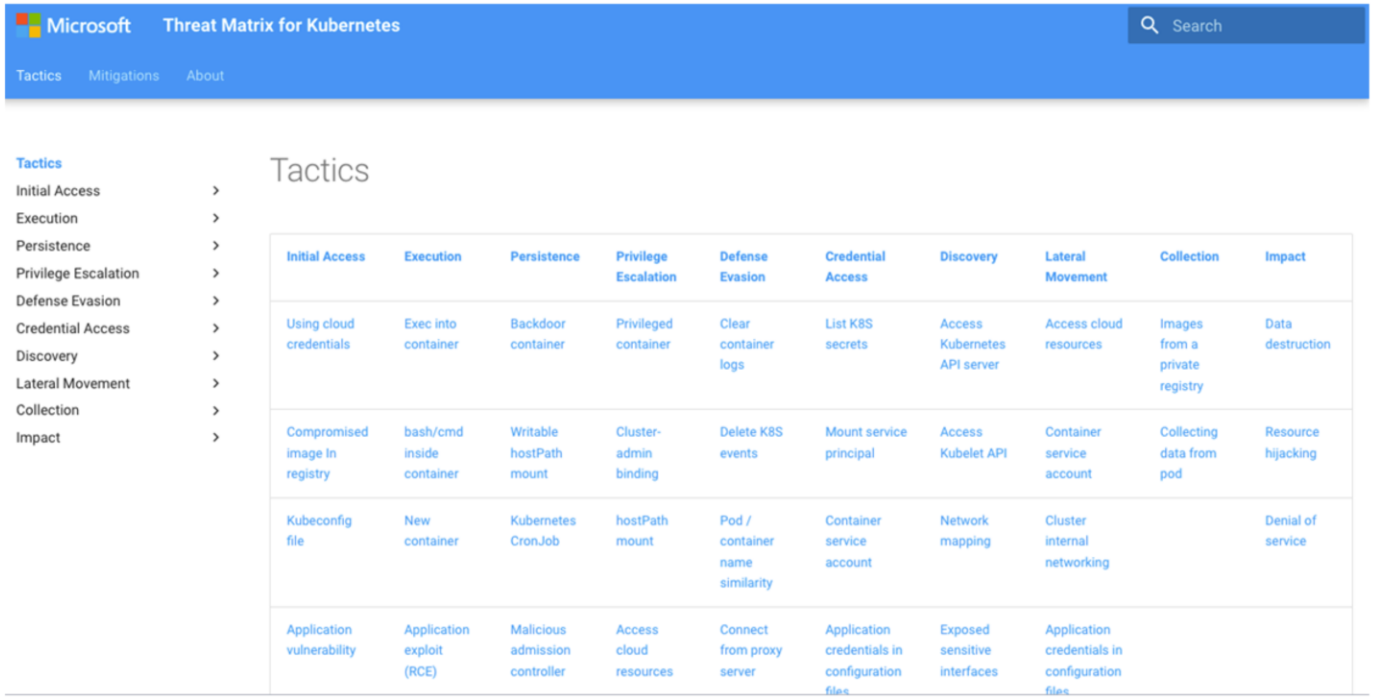
It is used to help organizations build stronger defenses by understanding different tactics (what attackers aim to do) and techniques (how they do it).



The MITRE ATT&CK Framework for Kubernetes focuses on how attackers target Kubernetes clusters. It organizes their tactics (goals) and techniques (methods) into categories:

- Initial Access: How attackers enter the cluster, such as exploiting weak authentication.
- Execution: Running unauthorized commands or code, like deploying malicious containers.
- Persistence: Maintaining access, for example, by creating new users or modifying roles.
- Privilege Escalation: Gaining higher access, such as exploiting misconfigured RBAC.
- Defense Evasion: Hiding activities, like disabling logs or concealing workloads.

MITRE ATT&CK Framework



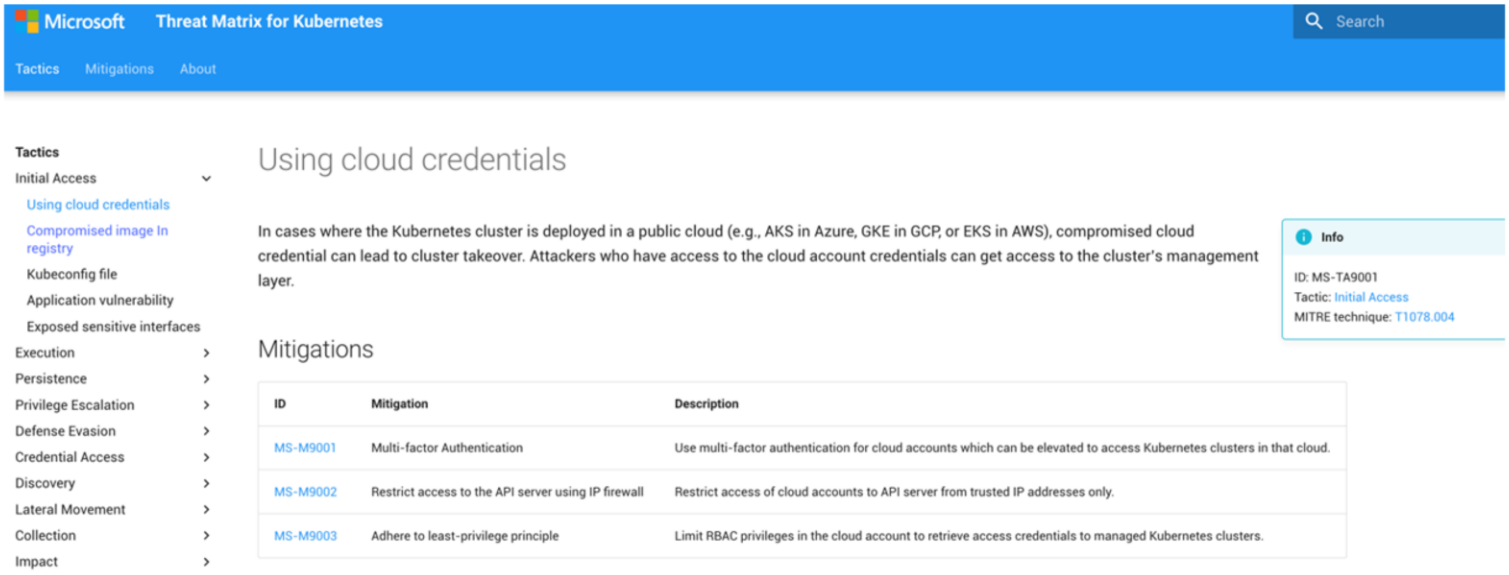
Source: https://microsoft.github.io/Threat-Matrix-for-Kubernetes/

MITRE created this framework to help everyone—from businesses to governments—work together and improve cybersecurity. It's free and open for anyone to use.

For more Kubernetes-specific examples, you can check out Microsoft's Threat Matrix for Kubernetes. This framework is a powerful tool to understand and stop attacks on Kubernetes clusters and it is inspired from MITRE.
https://microsoft.github.io/Threat-Matrix-for-Kubernetes/

This is Microsoft's Threat Matrix for Kubernetes, which maps out potential adversary tactics and techniques specifically targeting Kubernetes environments. It categorizes attacks into stages such as Initial Access, Execution, Persistence, and more, providing specific examples under each (e.g., "Using cloud credentials" under Initial Access or "Privileged container" under Privilege Escalation). The goal is to help security teams understand and mitigate threats in Kubernetes ecosystems by visualizing possible attack paths.

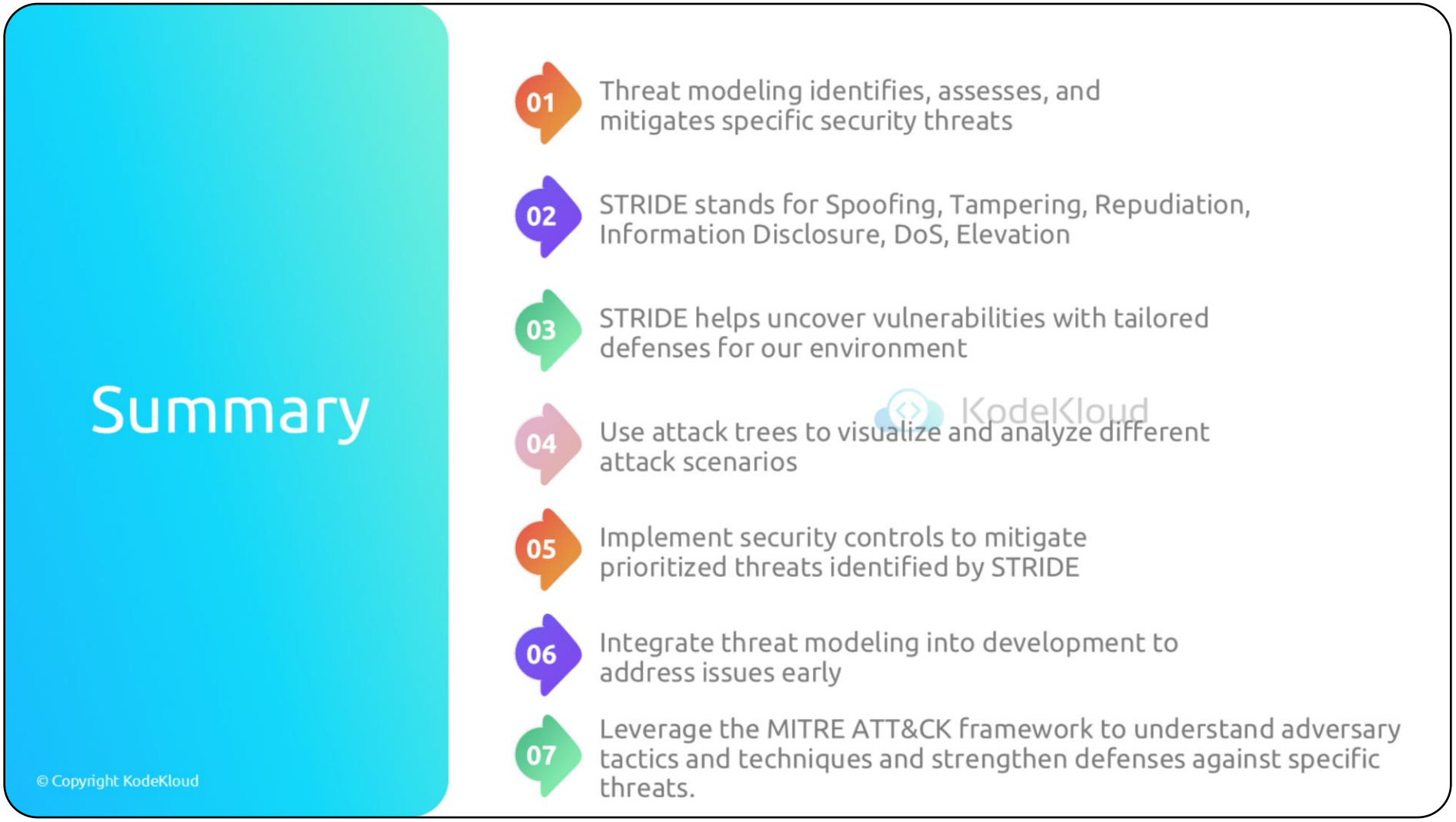
MITRE ATT&CK Framework



Source: https://microsoft.github.io/Threat-Matrix-for-Kubernetes/

And you can click on each item and drill down into it to know further. For example, under using cloud credentials you see a description of what the tactic is and also a set of mitigation criterias such as multi-factor authentication, restricting access to the API server using IP Firewall and adhering to least-privilege principle.

I recommend spending some time going through this page and familiarizing the different tactics by yourself.



Summary:

* Threat modeling identifies, assesses, and mitigates specific security threats.
* STRIDE stands for Spoofing, Tampering, Repudiation, Information Disclosure, DoS, Elevation.
* STRIDE helps uncover vulnerabilities with tailored defenses for our environment.
* Use attack trees to visualize and analyze different attack scenarios.
* Implement security controls to mitigate prioritized threats identified by STRIDE.

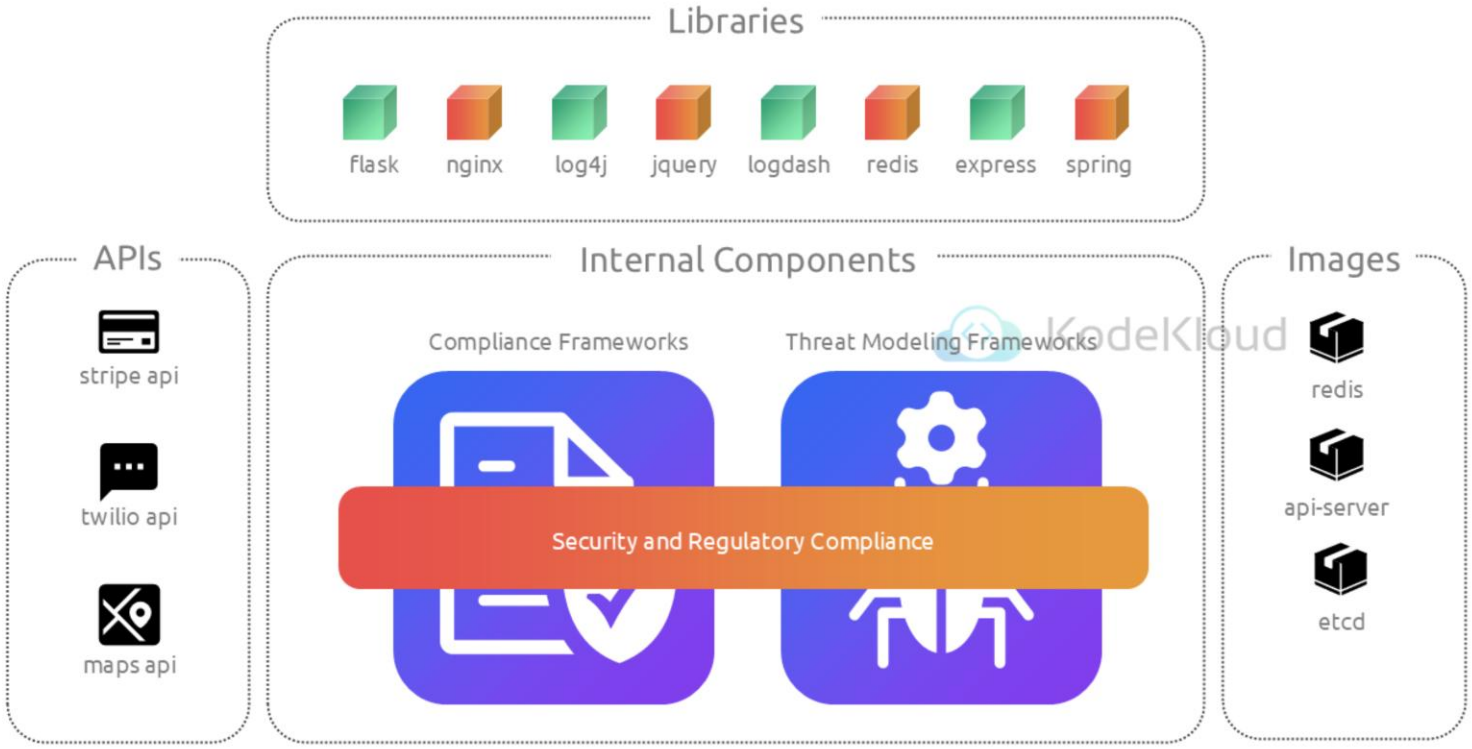
* Integrate threat modeling into development to address issues early.
* Leverage the MITRE ATT&CK framework to understand adversary tactics and techniques and strengthen defenses against specific threats.

Supply Chain Compliance

Supply Chain Compliance

In our previous lessons, we explored how compliance frameworks help meet security and regulatory requirements, while threat modeling frameworks help identify potential risks to an application.
Both focus on securing the internal components of an application.

Supply Chain Compliance



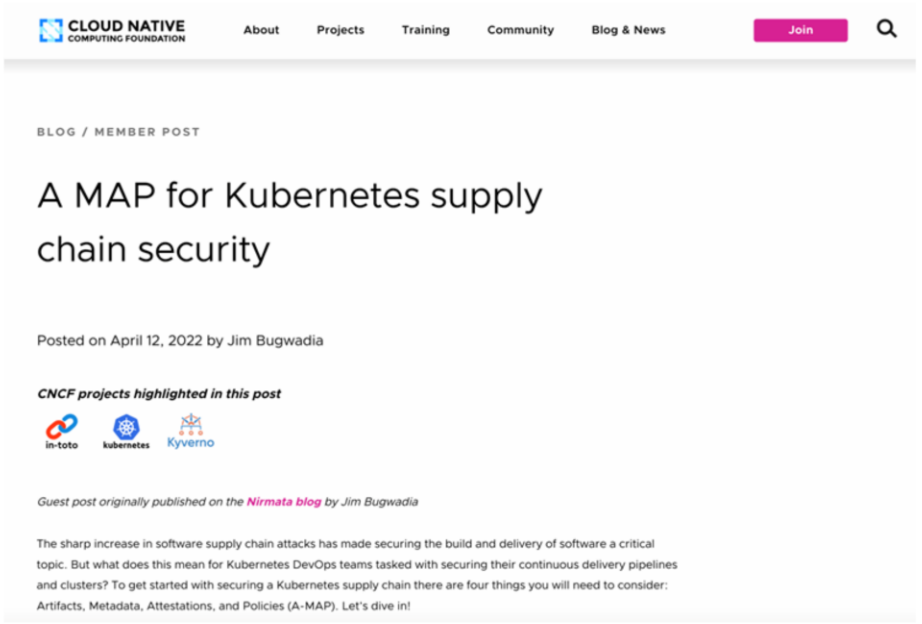
However, what about the external components your application depends on, such as libraries, APIs, and third-party services? These elements also need to be secure and compliant to ensure the overall safety of your application. This is the essence of supply chain compliance.

These elements also need to be secure and compliant to ensure the overall safety of your application. This is the essence of supply chain compliance.

Supply chain compliance involves verifying that all the external components and services you integrate into your application meet security and compliance standards.

Key Components of Supply Chain Security

- Artifact
- Metadata
- Attestations
- Policies



CLOUD NATIVE COMPUTING FOUNDATION About Projects Training Community Blog & News Join

BLOG / MEMBER POST

A MAP for Kubernetes supply chain security

Posted on April 12, 2022 by Jim Bugwadia

CNCF projects highlighted in this post

in-toto kubernetes Kyverno

*Guest post originally published on the **Nirmata blog** by Jim Bugwadia*

The sharp increase in software supply chain attacks has made securing the build and delivery of software a critical topic. But what does this mean for Kubernetes DevOps teams tasked with securing their continuous delivery pipelines and clusters? To get started with securing a Kubernetes supply chain there are four things you will need to consider: Artifacts, Metadata, Attestations, and Policies (A-MAP). Let's dive in!

As per this blog named a map of Kubernetes supply chain security posted on the CNCF blogs by Jim Bugwadia – the founder and CEO of KeyVerno - securing the supply chain focuses on four core areas: Artifacts, Metadata, Attestations, and Policies.

Artifacts: Verifying What You Deploy

sigstore cosign

Artifact

Metadata

Attestations

Policies

Artifacts
(Binaries, Container Images)

```
cosign sign $IMAGE

Generating ephemeral keys...
Retrieving signed certificate...

By typing 'y', you attest that you grant (or have permission to grant)
Are you sure you would like to continue? [y/N] y
Your browser will now be opened to:
https://oauth2.sigstore.dev/auth/auth?
access_type=online&client_id=sigstore&code_challenge=sdfsdfsdf
Successfully verified SCT...
tlog entry created with index: 12086900
Pushing signature to: $IMAGE
```

```
cosign verify-blob "$BINARY" \
  --signature "$BINARY".sig \
  --certificate "$BINARY".cert \
  --certificate-identity krel-staging@k8s-releng-prod.iam.gserviceaccount.com \
  --certificate-oidc-issuer https://accounts.google.com
```

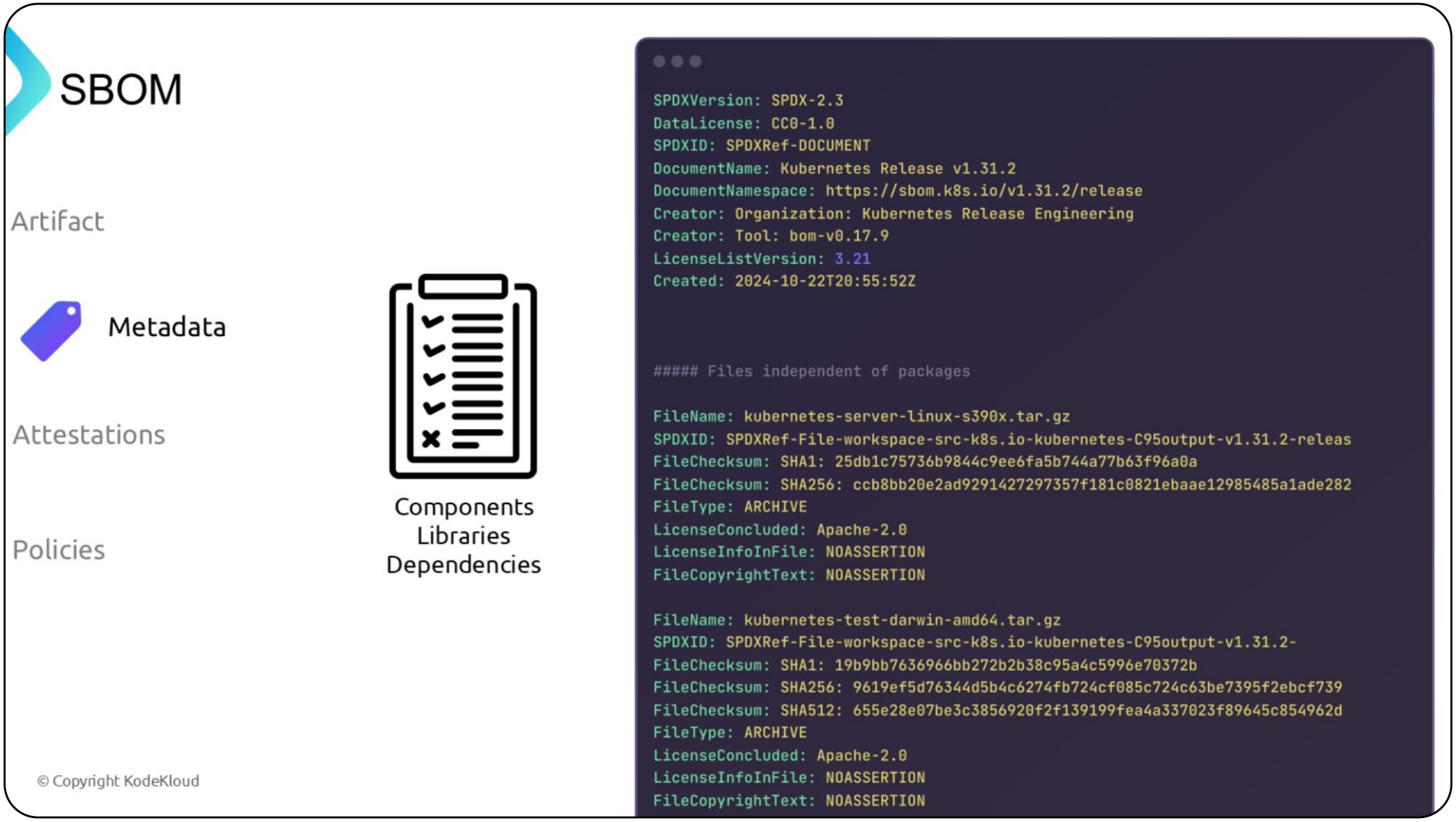

Let's start with Artifacts - Verifying What You Deploy

Artifacts are the outputs of your build process—things like binaries, container images, or tarballs. These are the components that get deployed into production, so verifying their integrity is critical.

In Kubernetes, artifacts are signed during the release process using keyless signing tools like Cosign. This ensures the artifacts haven’t been tampered with.

To verify an artifact, you simply download the binary along with its signature and certificate, and then use a cosign verify-blob command to confirm its authenticity.
This ensures the artifact comes from a trusted source and hasn't been altered.

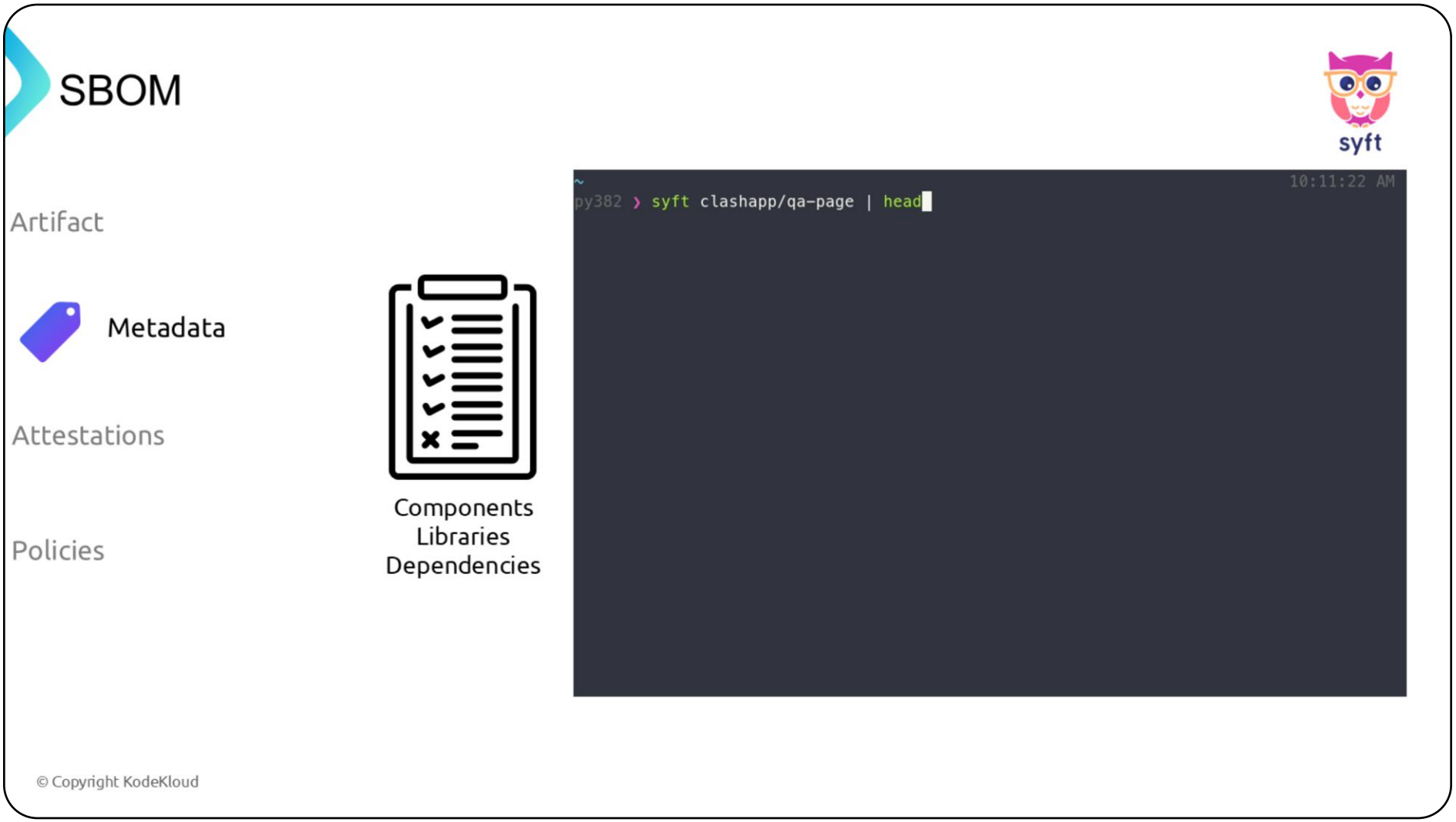
You can go ahead and checkout detailed version in K8s Docs: https://kubernetes.io/docs/tasks/administer-cluster/verify-signed-artifacts/



Metadata describes what's in your artifacts and where they came from. A critical type of metadata is the Software Bill of Materials (SBOM). For example here is the SBOM of kubernetes version 1.31.2. And it's in what is known as an SPDX format.

Think of an SBOM as the "ingredients list" for your application. It details all the components, libraries, and dependencies, along with their versions and sources. For example in this case you see the different components of kubernetes such as the controller manager, kubelet api server etc. .

In Kubernetes, SBOMs are generated for each release and signed for authenticity.



But how do you generate an SBOM? Here is where tools like Syft comes in handy. Syft is a CLI tool and go library that can be used for generating a software bill of materials from container images and filesystems.

Metadata: Understanding What's Inside

Artifact

Metadata

Attestations

Policies

Components
Libraries
Dependencies

```
# Retrieve the latest available Kubernetes release version
VERSION=$(curl -Ls https://dl.k8s.io/release/stable.txt)

# Verify the SHA512 sum
curl -Ls "https://sbom.k8s.io/$VERSION/release" -o "$VERSION.spdx"
echo "$(curl -Ls "https://sbom.k8s.io/$VERSION/release.sha512") $VERSION.spdx" |
sha512sum --check

# Verify the SHA256 sum
echo "$(curl -Ls "https://sbom.k8s.io/$VERSION/release.sha256") $VERSION.spdx" |
sha256sum --check

# Retrieve sigstore signature and certificate
curl -Ls "https://sbom.k8s.io/$VERSION/release.sig" -o "$VERSION.spdx.sig"
curl -Ls "https://sbom.k8s.io/$VERSION/release.cert" -o "$VERSION.spdx.cert"

# Verify the sigstore signature
cosign verify-blob \
    --certificate "$VERSION.spdx.cert" \
    --signature "$VERSION.spdx.sig" \
    --certificate-identity krel-staging@k8s-releng-prod.iam.gserviceaccount.com \
    --certificate-oidc-issuer https://accounts.google.com \
    "$VERSION.spdx"
```


To verify an SBOM, you retrieve it along with its signature and validate it using Cosign. This helps confirm that the SBOM is accurate and hasn't been modified.

In this example script shown here we are first retrieving the latest available kubernetes release version. We then retrieve the sbom for this version available at sbom.k8s.io. It then fetches the SHA512 checksum for the SBOM file from the Kubernetes release repository.Verifies that the downloaded SBOM matches the checksum using sha512sum

It then Downloads the Sigstore signature (.sig) and certificate (.cert) files for the SBOM.These files will be used to cryptographically verify the integrity and authenticity of the SBOM.

And finally, Uses the cosign tool (part of Sigstore) to verify the SBOM file. The certificate and signature are checked against a trusted identity (krel-staging@k8s-releng-prod.iam.gserviceaccount.com) and the OIDC issuer (accounts.google.com).

Now remember this step verifies that the downloaded SBOM itself was not tampered with or altered during transit.

Attestations: Building Trust

Artifact

Metadata

Attestations

Components
Libraries
Dependencies

Policies

```
cosign sign --key <PRIVATE_KEY> sbom.k8s.io/v1.27.4/release.spdx > sbom.attestation
```

```
cosign verify-attestation \
  --key <PUBLIC_KEY> \
  --certificate-identity krel-staging@k8s-releng-prod.iam.gserviceaccount.com \
  --certificate-oidc-issuer https://accounts.google.com \
  sbom.k8s.io/v1.27.4/release.spdx
```


Metadata is useful, but how do you ensure it's trustworthy? This is where attestations come in.

Attestations are signed statements that verify metadata like SBOMs, provenance data, or vulnerability reports. For example, an SBOM signed with Cosign assures you that the information it contains is authentic and comes from a trusted source.

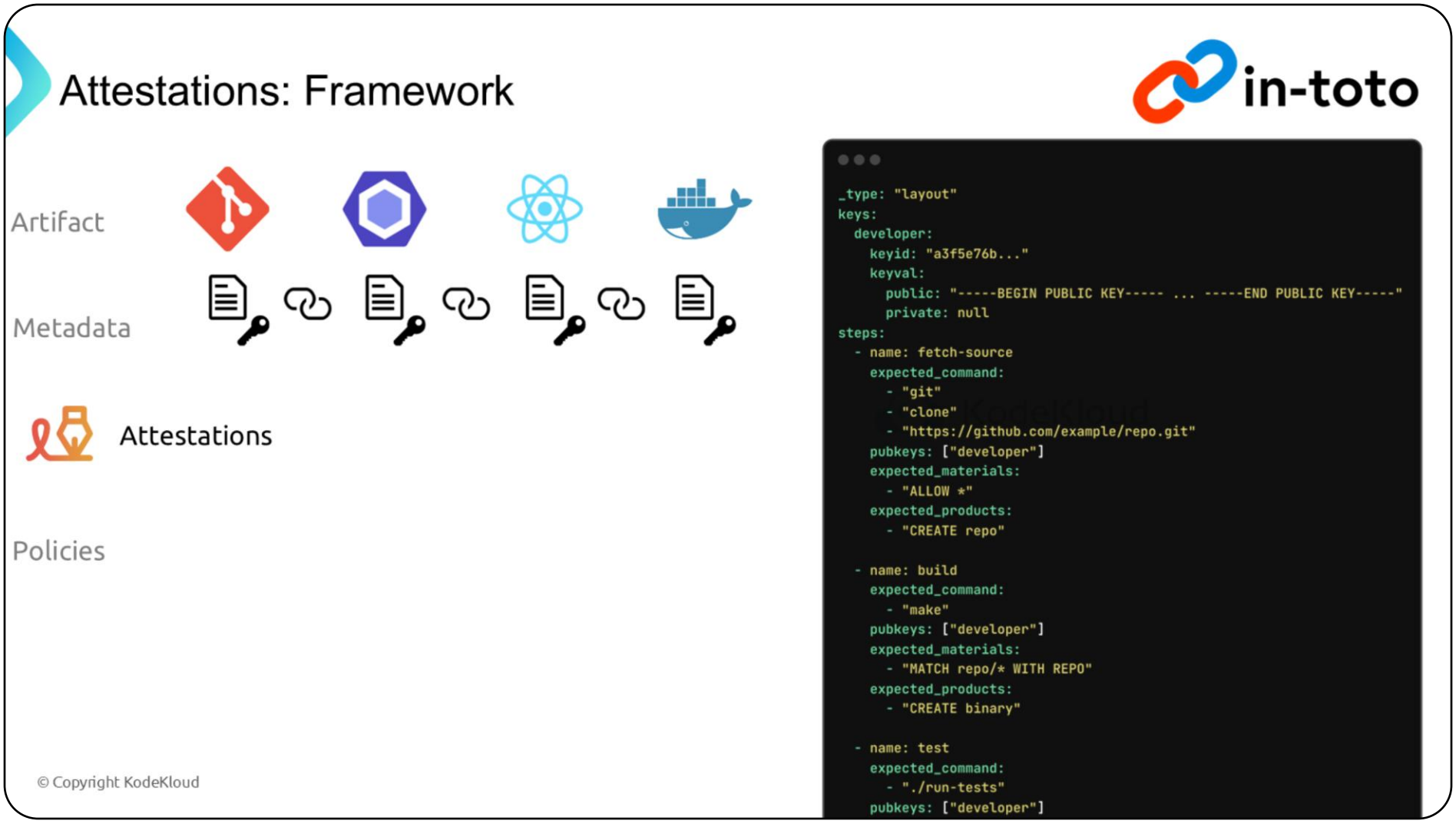
So the trusted party (e.g., Kubernetes release team) generates the SBOM and signs it using a private key (creating an attestation).

You verify the attestation using their public certificate (as shown in the script with Cosign).

The signature proves both the file's integrity (it hasn't changed) and its authenticity (it comes from the trusted Kubernetes release team).

If you are wondering how this is different from the previous script we saw that verifies SHA checksum – think of this analogy.

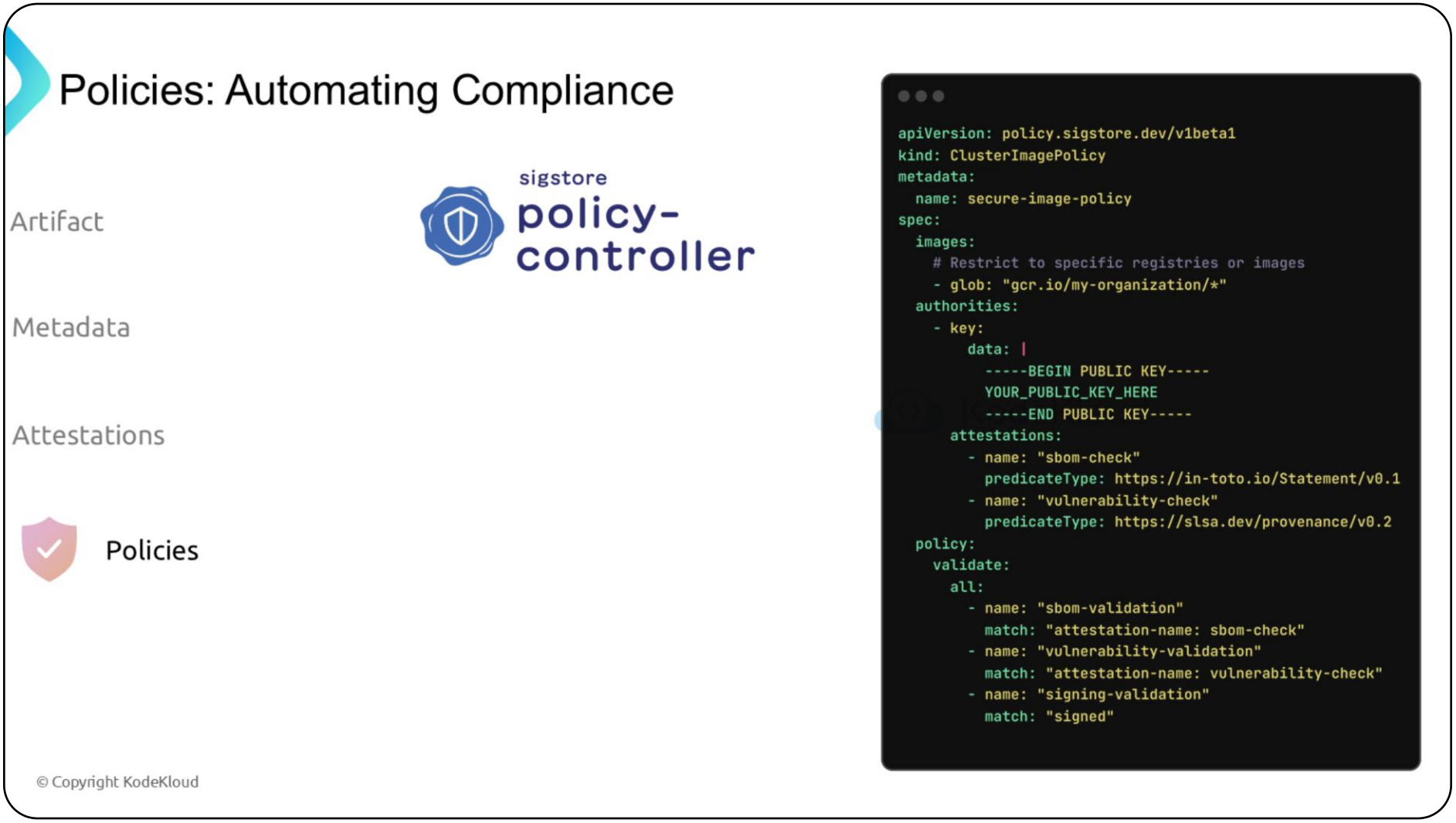
SHA512/SHA256: Ensures the downloaded SBOM file matches what the server published, guaranteeing file integrity during transit. Attestation: Adds an extra layer of trust by proving the SBOM is signed by a trusted source and hasn't been tampered with since it was signed. A good analogy would be Think of the SHA checksum as checking that a sealed package hasn't been damaged in shipping.The attestation is like verifying the package's seal and sender's signature to confirm it's from a trusted vendor and contains what they claim.



Tools like in-toto standardize how these attestations are generated, making it easier for teams to trust their supply chain data. By incorporating attestations, you ensure the information guiding your security decisions is reliable. in-toto is a not just a tool rather a framework for defining and verifying the entire supply chain process, focusing on provenance and attestation. in-toto defines how attestations (like SBOMs or provenance data) should be generated, structured, and verified. It creates a "chain of trust" for all steps in the software supply chain.

You can checkout more details around this topic at in-toto official website. https://in-toto.io/

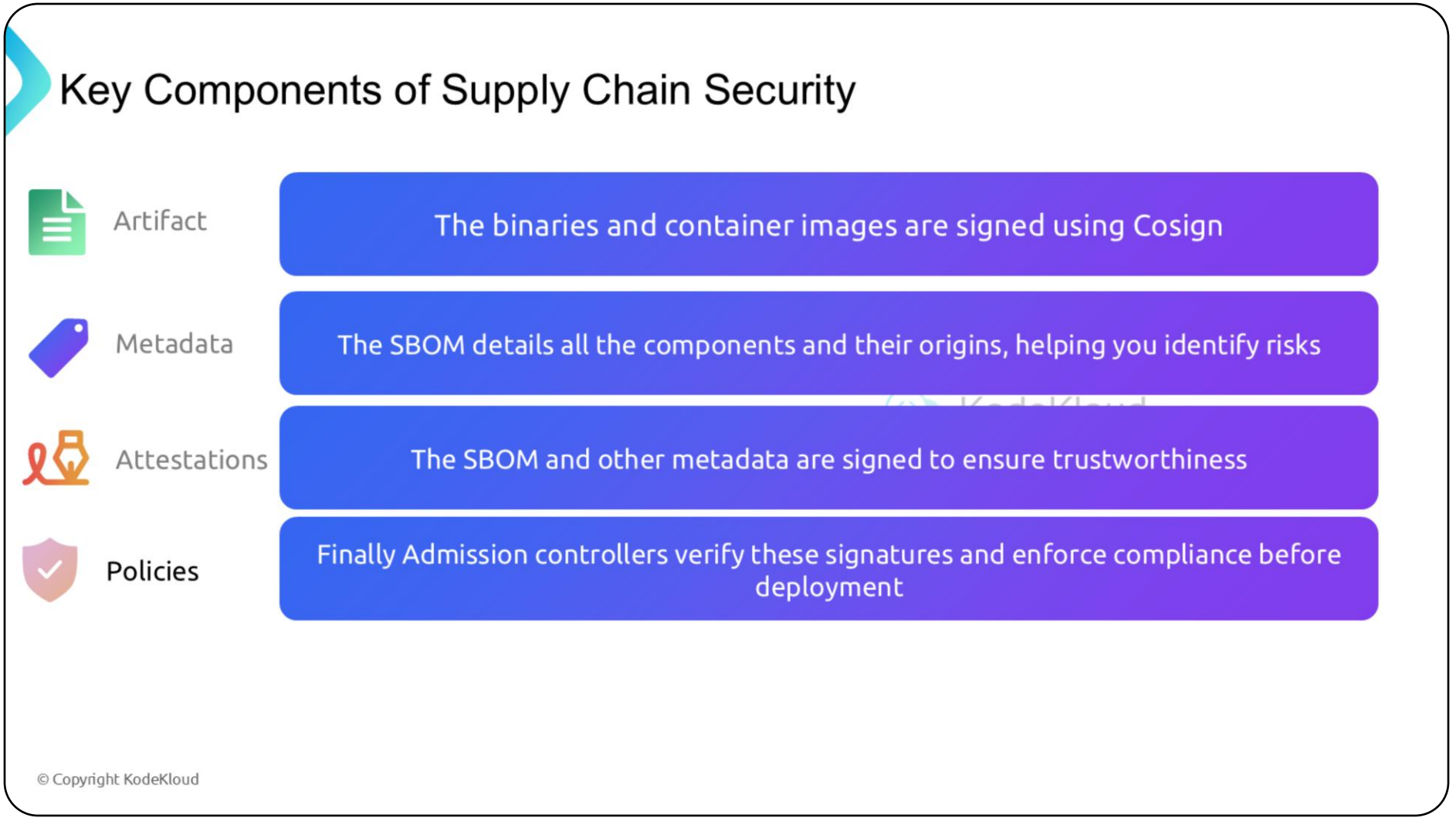
So now we have a framework and set of tools that can automate artifact generation, metadata creation and attestation at different stages of the development cycle. Now how do you automate the deployment and verification of these components on Kubernetes? When these artifacts are deployed how do we make sure we are only deploying verified trustworthy artifacts.



This is where Policies and policy controller comes. Policies are the rules that ensure compliance and security standards are enforced automatically. They help prevent insecure or non-compliant components from being deployed.

For instance, a policy might enforce that all images include a valid SBOM, have no critical vulnerabilities, or meet specific signing requirements. Kubernetes tools like sigstore's policy-controller can integrate with admission controllers to enforce these rules at deploy time, blocking anything that doesn't meet your standards.

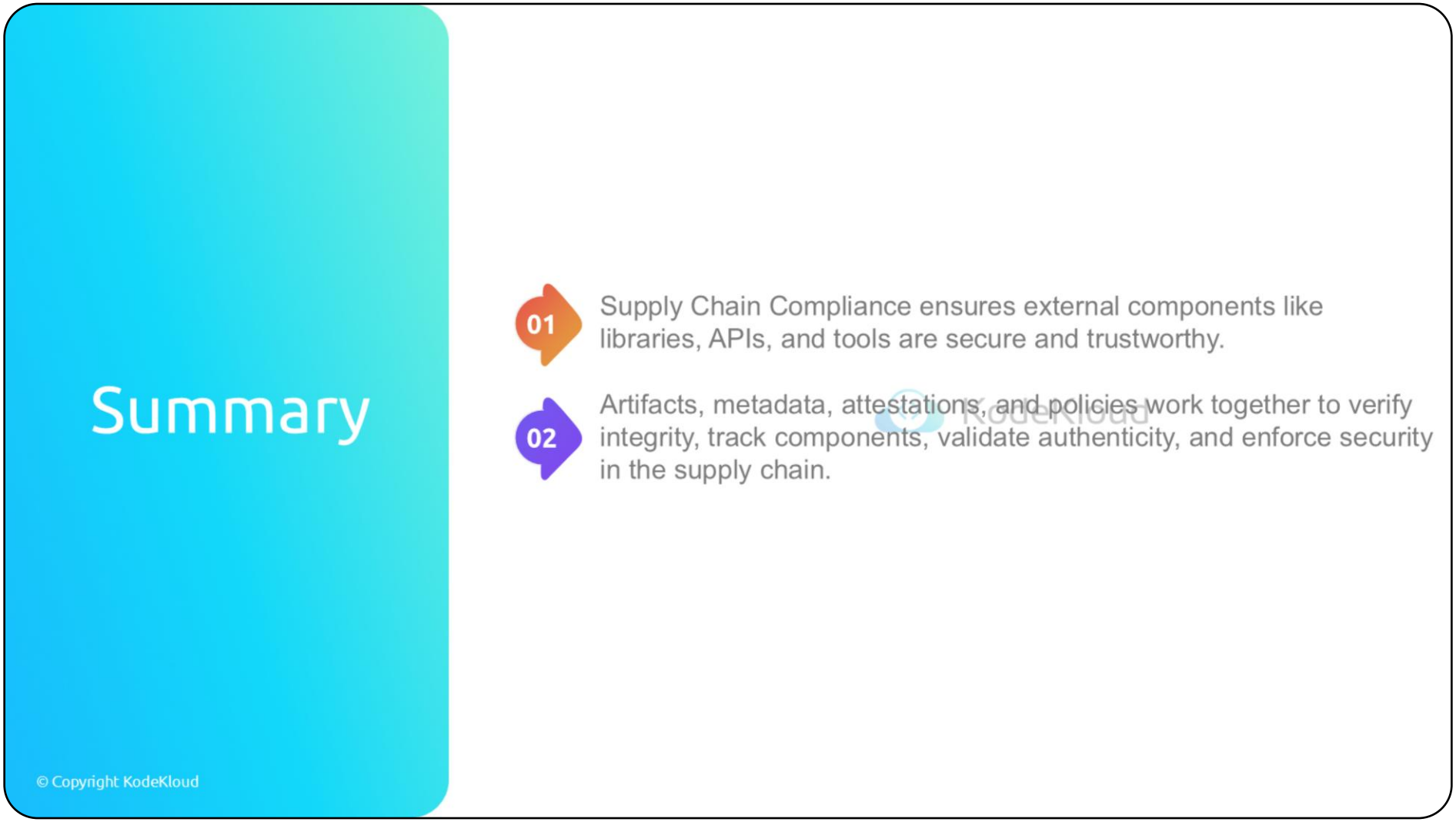
The details can be found on https://docs.sigstore.dev/policy-controller/overview/



Let’s see how these principles work together. Consider a Kubernetes release:

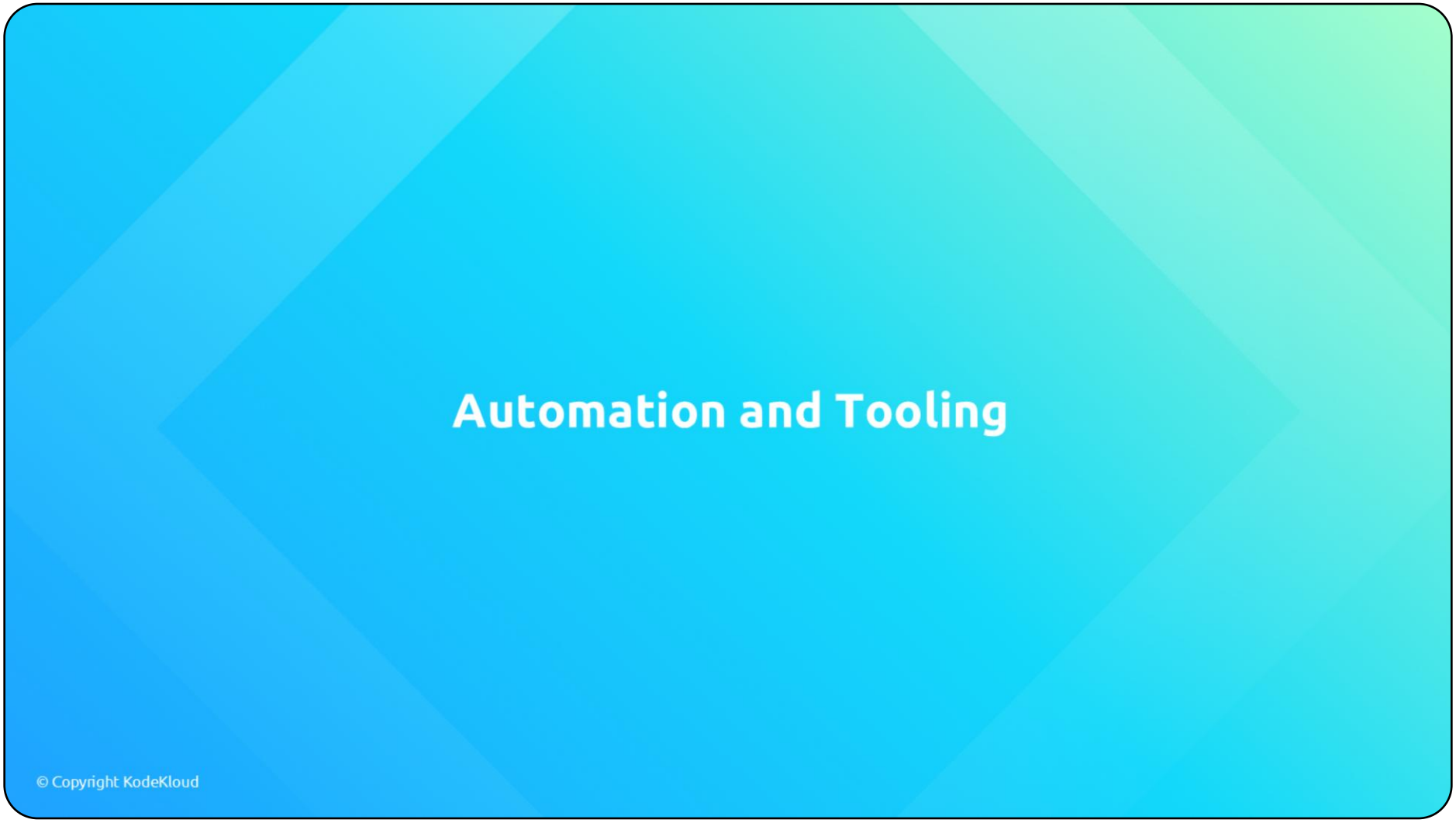
First The binaries and container images are signed using Cosign.(Artifacts)
Then: The SBOM details all the components and their origins, helping you identify risks.(Metadata)
Next, the SBOM and other metadata are signed to ensure trustworthiness.(Attestations)
Finally Admission controllers verify these signatures and enforce compliance before deployment.(Policies)

This is a high-level guide to help you build the skills and confidence needed to tackle the KCSA exam. In a real-world environment, additional complexities and internal processes are integrated into this workflow.



Summary

- Supply Chain Compliance ensures external components like libraries, APIs, and tools are secure and trustworthy.
- Artifacts, metadata, attestations, and policies work together to verify integrity, track components, validate authenticity, and enforce security in the supply chain.



In this lesson we will go through automation and tooling. We have looked at a number of tools throughout this course so there isn't much new here, we will summarize all of that here and also take some time to look at some alternatives.

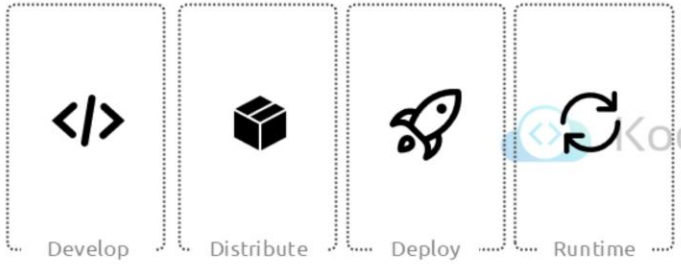
Cloud Native Security Whitepaper

The Cloud Native Security Whitepaper by SIG-Security is one resource that we have referred to extensively during this course. It is built by experts from a variety of industries and serves as one of the top resources to gain an understanding of security in the cloud native world. So I highly recommend reading this along with this course.

Cloud Native Security Whitepaper

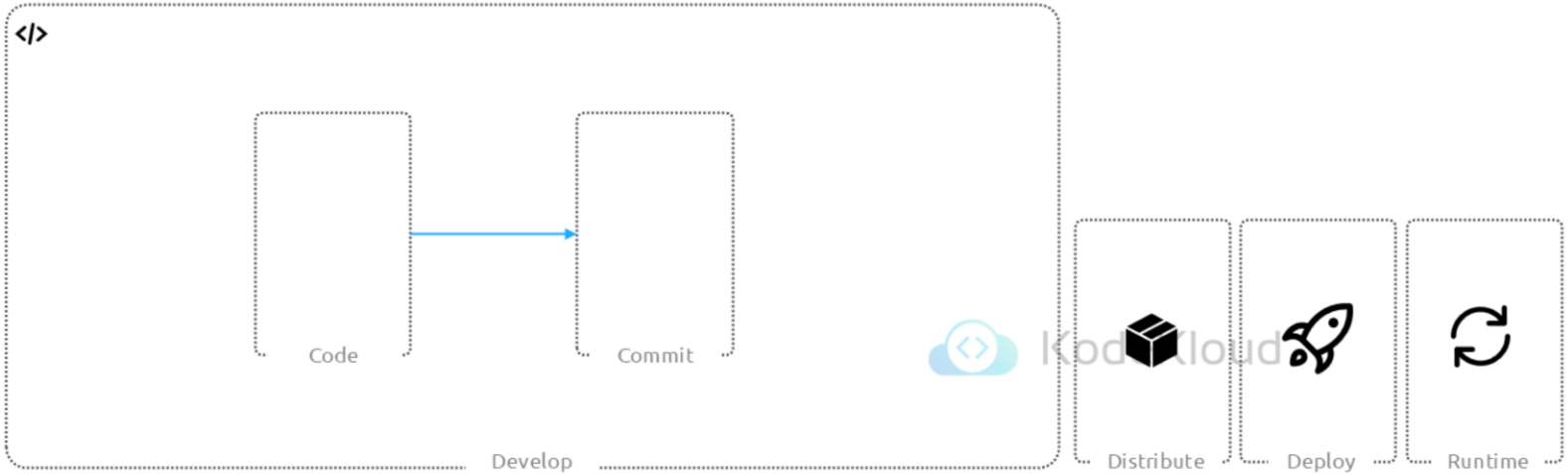
The Cloud Native Security map builds on top of the Cloud Native Security Whitepaper by SIG-Security, by providing an additional practitioner's perspective as well as an interactive mode of consumption. In this video I'll walk you through some of the tools referred here.

Lifecycle

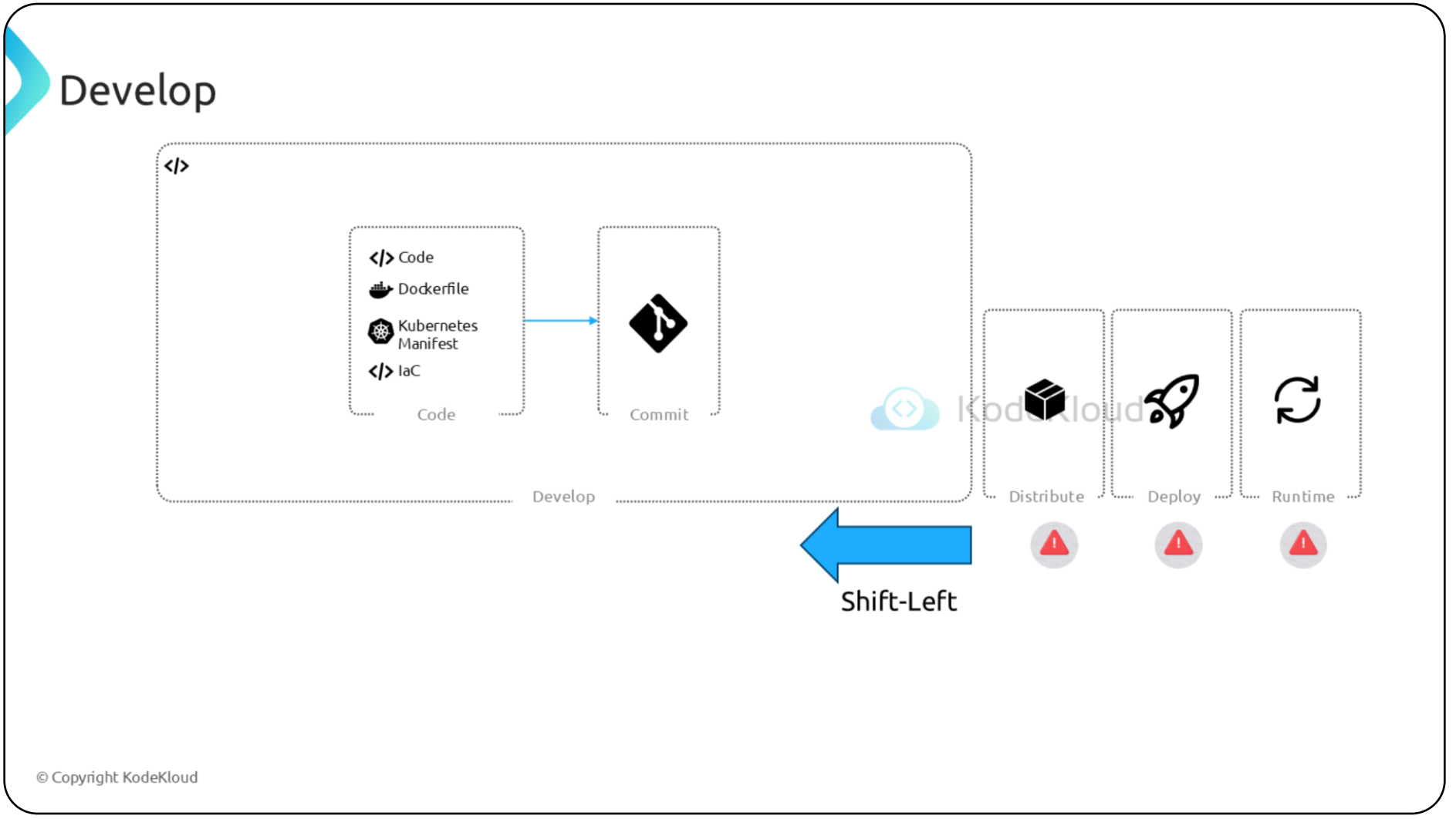


The cloud native security map discusses security in various phases of the application lifecycle – namely develop, distribute, deploy and runtime.

Lifecycle



The Develop phase is all about building your application code and committing it to Source Code Management (SCM) systems like GitHub or GitLab.

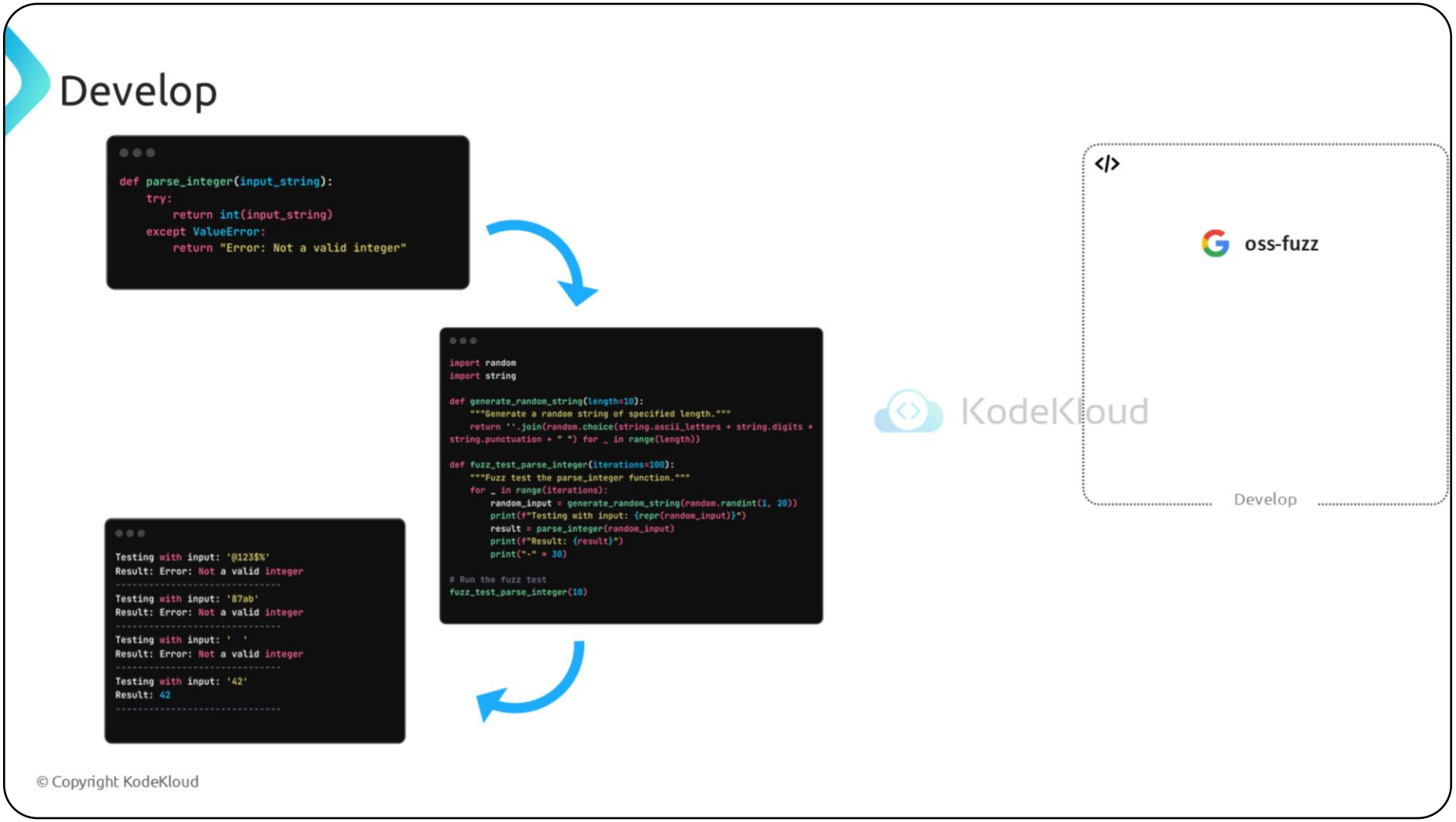


While coding applications you generate your code and also produce Dockerfile, IaC code, workload manifest files etc. And then you commit your code to SCM like Github or Gitlab.

The goal here is to identify security issues early in the SDLC, aligning with the shift-left approach. Meaning the more security related issues you identify towards the later part of the cycle, the more costly and time-consuming they become to fix. Addressing vulnerabilities early reduces risk, streamlines development, and ensures a smoother delivery process. By embedding security practices into the early stages of development, teams can proactively prevent issues rather than reacting

to them post-deployment.

For this we want to use tools that provide developers with security insights and actionable fixes directly within their workflows. Integrate IDE plugins, CLI tools, and automate processes like git pre-commit hooks. Implement automated security scans in source code management systems to test PRs. Establish clear policies that developers can follow, such as disallowing high-severity vulnerabilities with available fixes, enforcing non-root container images, and using only approved base images.



One of the tools to be used during the develop phase is Google's OSS-Fuzz, which helps identify bugs and vulnerabilities in open-source software by performing extensive automated fuzz testing.

Fuzzing is a technique used to test programs by feeding them random or unexpected inputs to uncover crashes, undefined behavior, or security vulnerabilities. It works by exploring unusual or invalid inputs that can cause software to behave incorrectly.

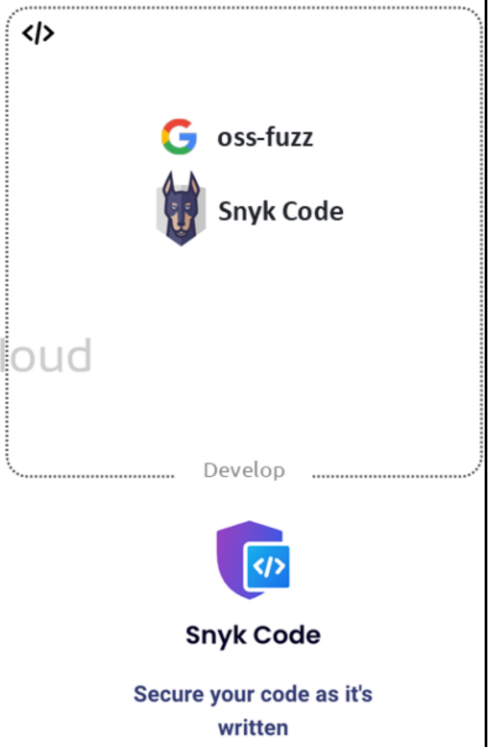
Let's say we have a simple function that parses a string into an integer:
We want to fuzz test this function to ensure it handles unexpected inputs gracefully.
So this code has a function that creates random strings containing letters, numbers, puncuation, and spaces.

The fuzz_test_parse_integer function repeatedly feeds random strings into parse_integer.

Each test checks how the function handles unexpected inputs, and the results are printed.

This example shows how fuzz testing can expose edge cases or unexpected behaviors in code.

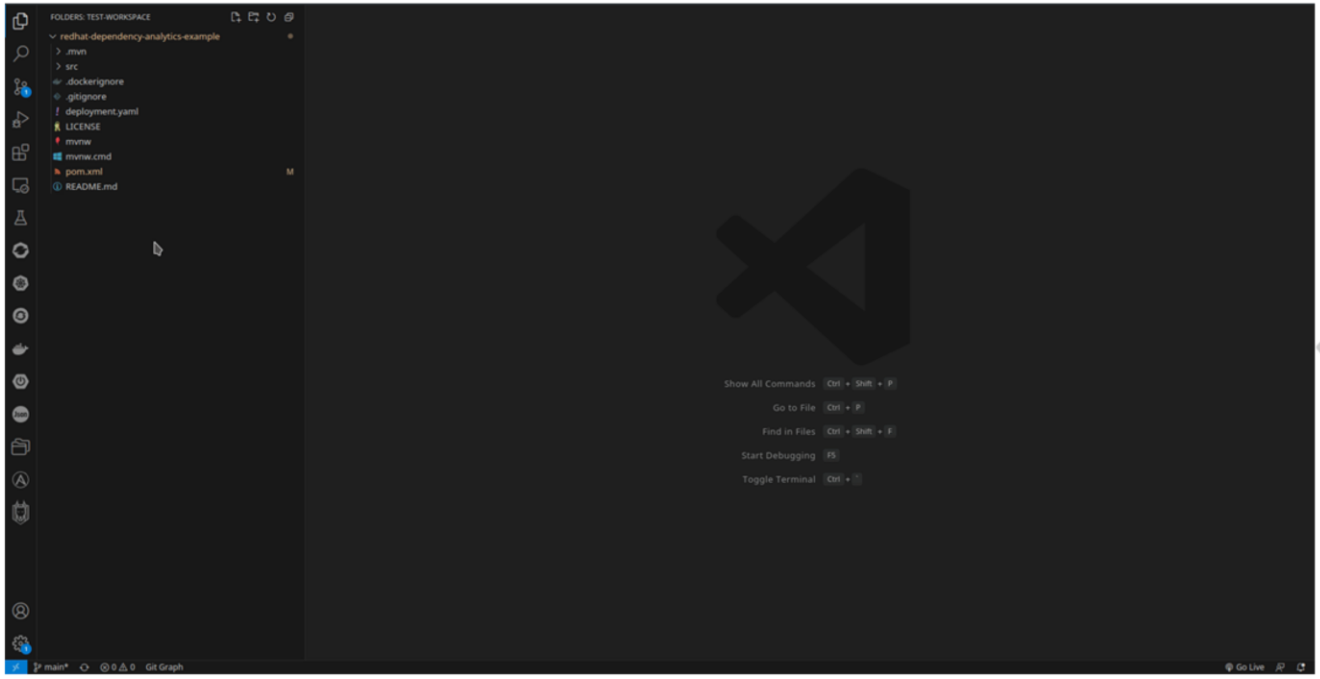
Develop



https://marketplace.visualstudio.com/items?itemName=snyk-security.snyk-vulnerability-scanner

Tools like the Snyk Visual Studio Code extension allows you to analyze your code, open-source dependencies, and Infrastructure as Code (IaC) configurations. With actionable insights directly in your IDE, you can address issues as they arise.

Develop



</>

oss-fuzz

Snyk Code

fabric8

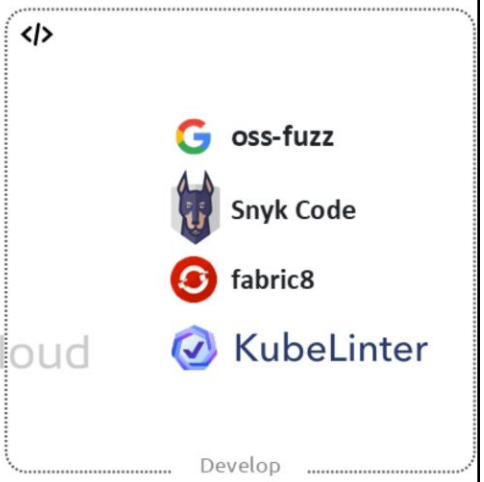
Develop

Similarly another tool is facric8 by redhat. It integrates seamlessly with VSCode to help enforce security and compliance right from your development environment. It provides actionable insights, making it easier to fix issues before they escalate.

Develop

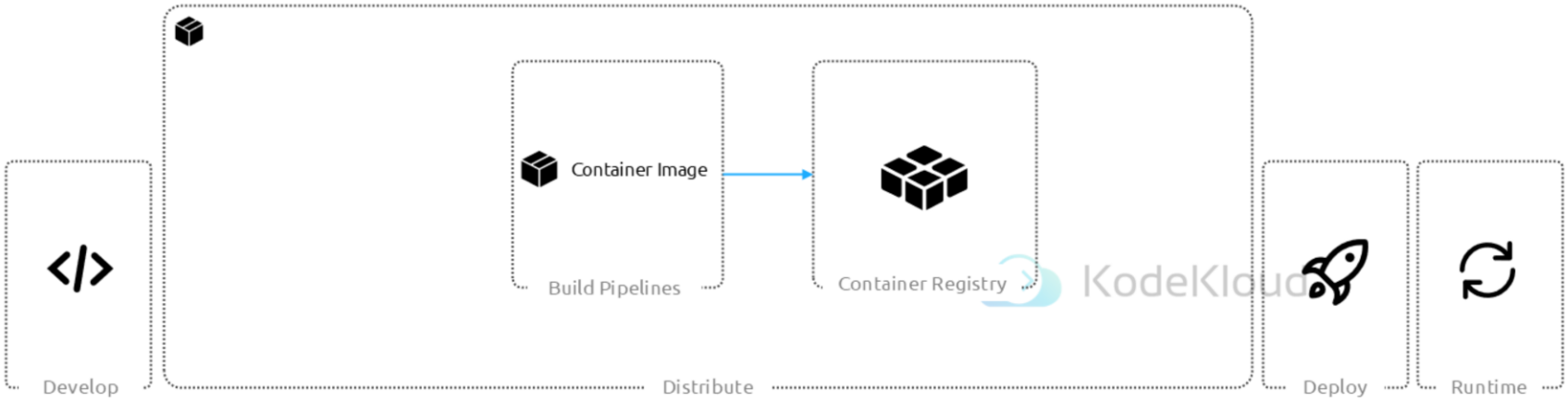
```
kube-linter lint pod.yaml
```

```
pod.yaml: (object: <no namespace>/security-context-demo /v1, Kind=Pod) container "sec-ctx-demo" does not have a read-only root file system (check: no-read-only-root-fs, remediation: Set readOnlyRootFilesystem to true in your container's securityContext.)

pod.yaml: (object: <no namespace>/security-context-demo /v1, Kind=Pod) container "sec-ctx-demo" has cpu limit 0 (check: unset-cpu-requirements, remediation: Set your container's CPU requests and limits depending on its requirements. See https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/#requests-and-limits for more details.)

pod.yaml: (object: <no namespace>/security-context-demo /v1, Kind=Pod) container "sec-ctx-demo" has memory limit 0 (check: unset-memory-requirements, remediation: Set your container's memory requests and limits depending on its requirements.    See https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/#requests-and-limits for more details.)

Error: found 3 lint errors
```



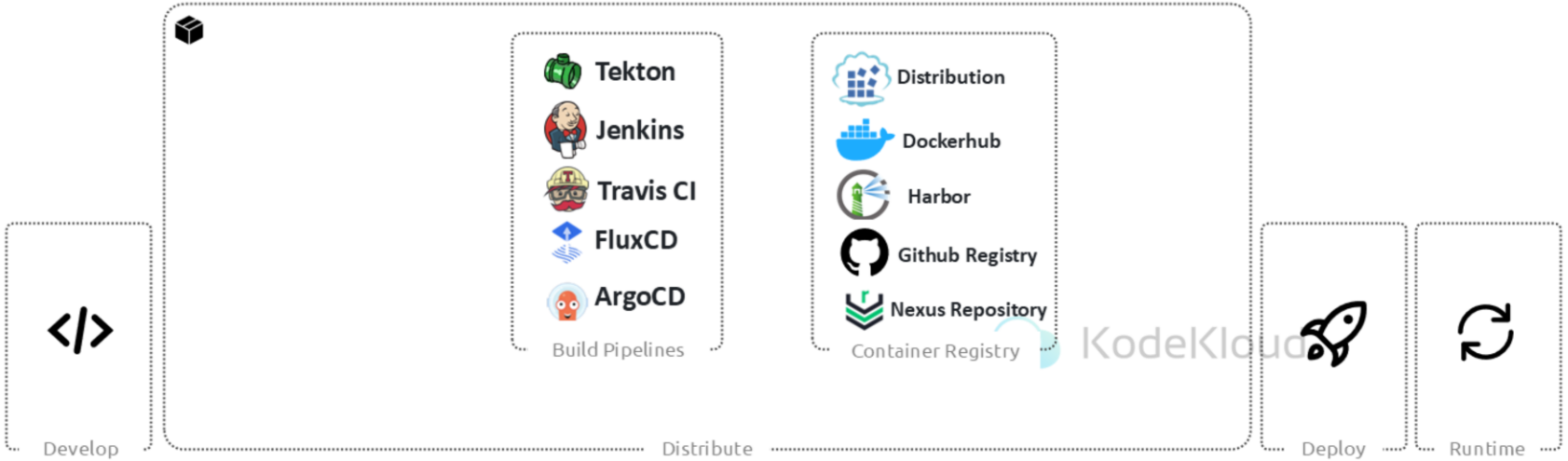
Kubelinter: This tool helps you catch issues in Kubernetes YAML files. It ensures best practices, like enforcing container limits, disallowing privileged containers, and verifying image tags etc.

Distribute



In the distribute phase you build container images and push them into container registries.

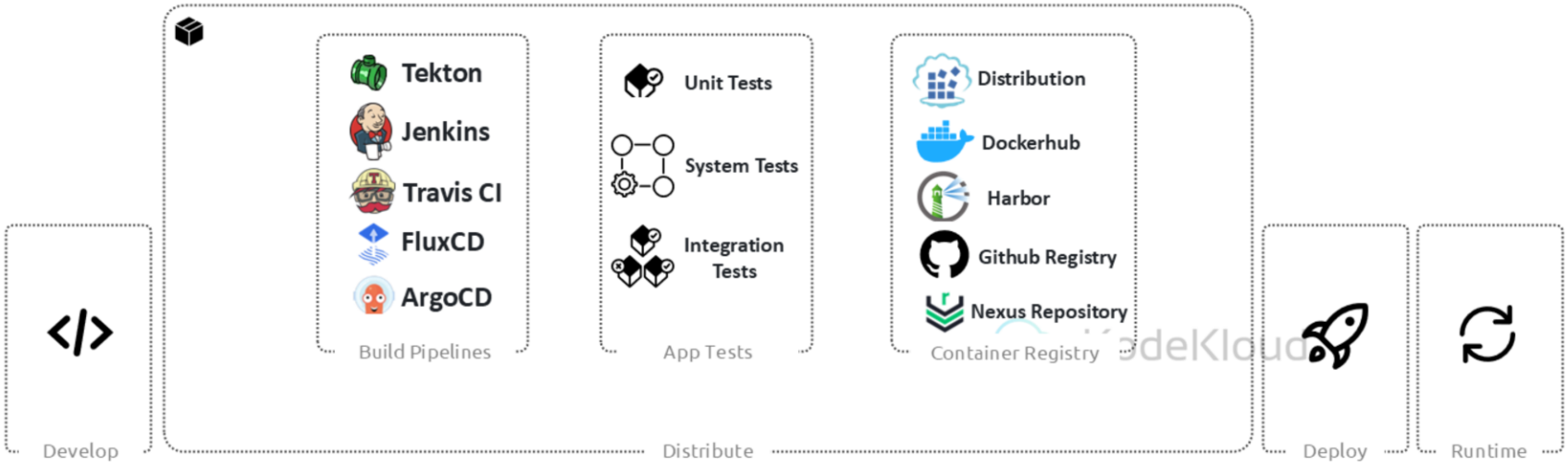
Distribute



In the build phase you have all the CI/CD tools used to automate build pipelines. These are the traditional ones like Tekton, Jenkins, Travis CI, CircleCI etc. And then we also have the new ones built for cloud native like FluxCD and ArgoCD.

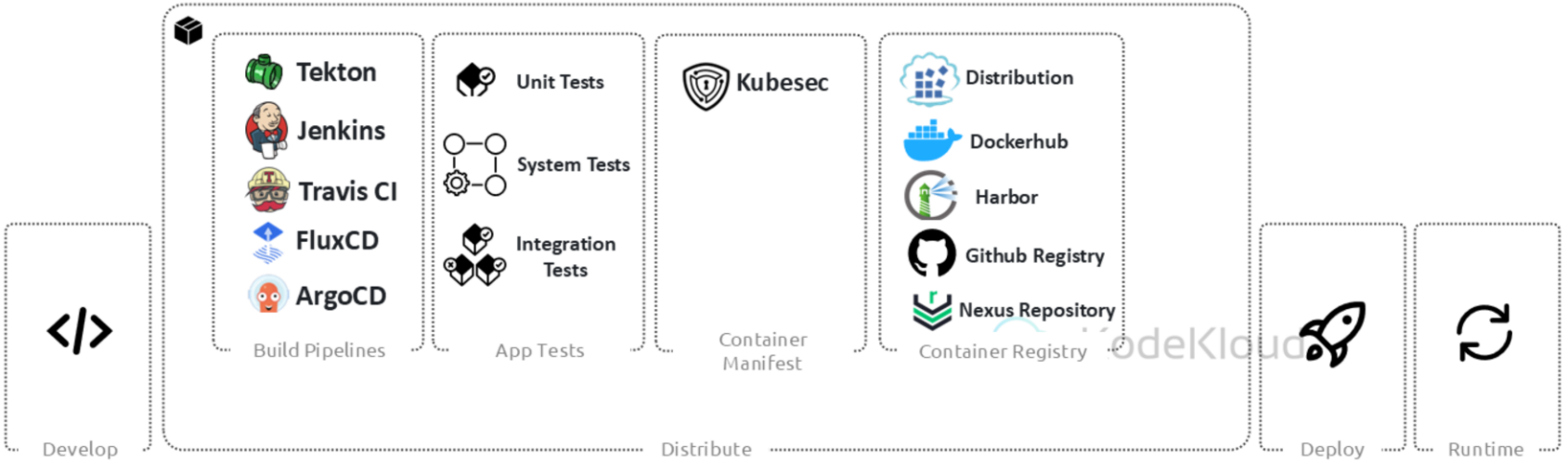
And then as container registries we have the Docker Distribution project. This is a set of tools used to pack, ship, store and deliver container images. This is the open source registry implementation that is used by many of the other registry operators like Docker hub, the CNCF and VMWare harbor projects, Github Gitlab container registry and the Nexus Repository by sonatype.

Distribute



Now before pushing images to the registry we want to make sure the application goes through the standard quality testing phase. These include unit tests, system tests and integration tests. There are many tools that fall into this category that are specific to different programming languages. Not getting into those for now.

Distribute



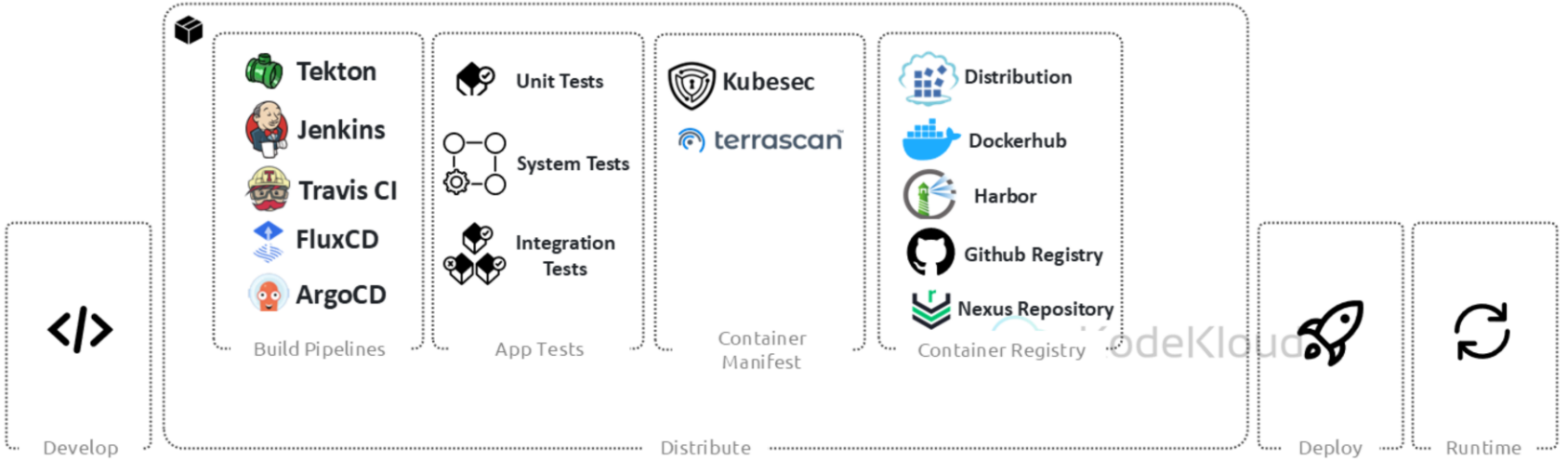

```
kubesec scan k8s-deployment.yaml
```

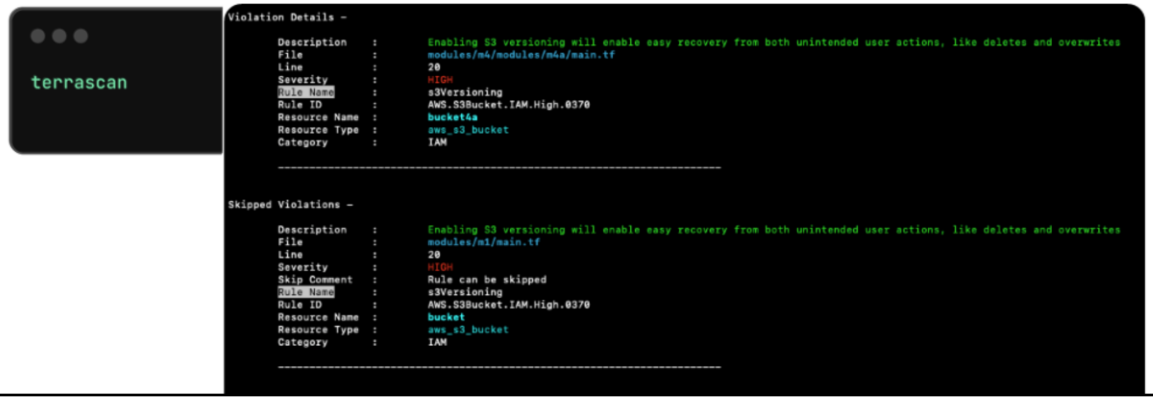

```
[
  {
    "object": "Pod/security-context-demo.default",
    "valid": true,
    "message": "Failed with a score of -30 points",
    "score": -30,
    "scoring": {
      "critical": [
        {
          "selector": "containers[] .securityContext .capabilities .add == SYS_ADMIN",
          "reason": "CAP_SYS_ADMIN is the most privileged capability and should always be
avoided"
        }
      ],
      "advise": [
```


Once those tests are complete and before the images are built we want to make sure the manifest files created are secure and compliant to our policies. This is where tools like kubesec, and terrascan come in.

Kubesec helps enforce policies like disallowing containers with privileged access or enforcing secure defaults in Kubernetes YAMLs.

Distribute





- 500+ Policies for security best practices
- Scanning of Terraform (HCL2)
- Scanning of AWS CloudFormation Templates (
  - Scanning of Azure Resource Mana
- Scanning of Kubernetes (JSON/YAML), Helm v
- Scanning of Dockerfiles
- Support for AWS, Azure, GCP, Kubernetes, Do

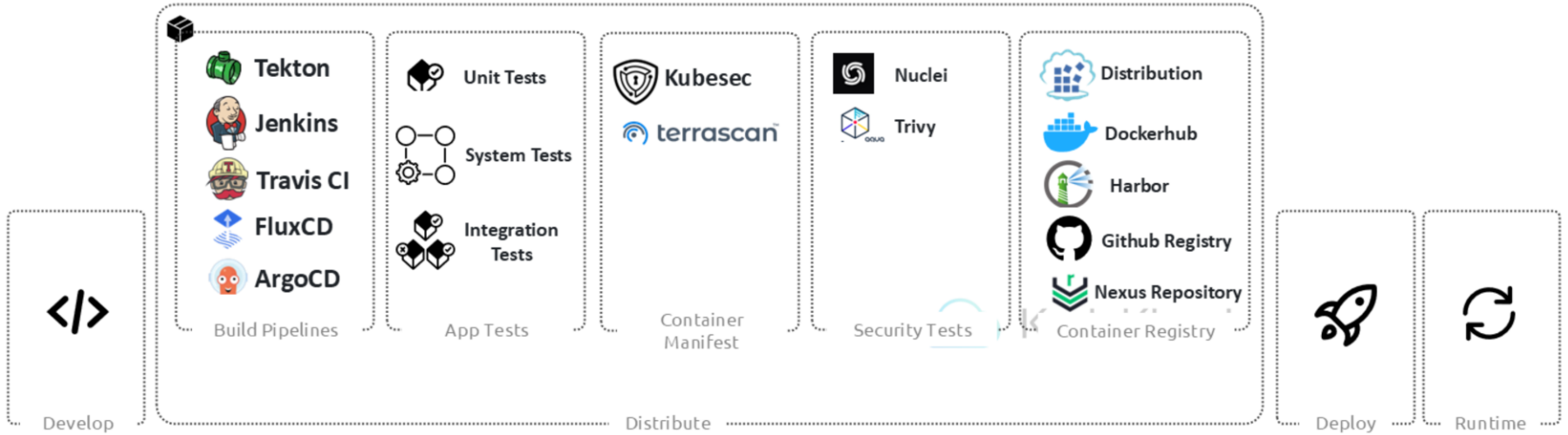
- Detects compliance violations across multiple
NIST, GDPR, HIPAA)

https://github.com/tenable/terrascan

While kubesec is specifically designed for Kubernetes manifests, another tool – Terrascan covers A broader security scanning tool for Infrastructure as Code (IaC) like Terraform, Dockerfiles, Kubernetes YAML, Helm charts, AWS CloudFormation Templates and others.

It also, detects compliance violations across multiple frameworks (e.g., CIS, NIST, GDPR, HIPAA).

Distribute




```
trivy image python:3.4-alpine
```

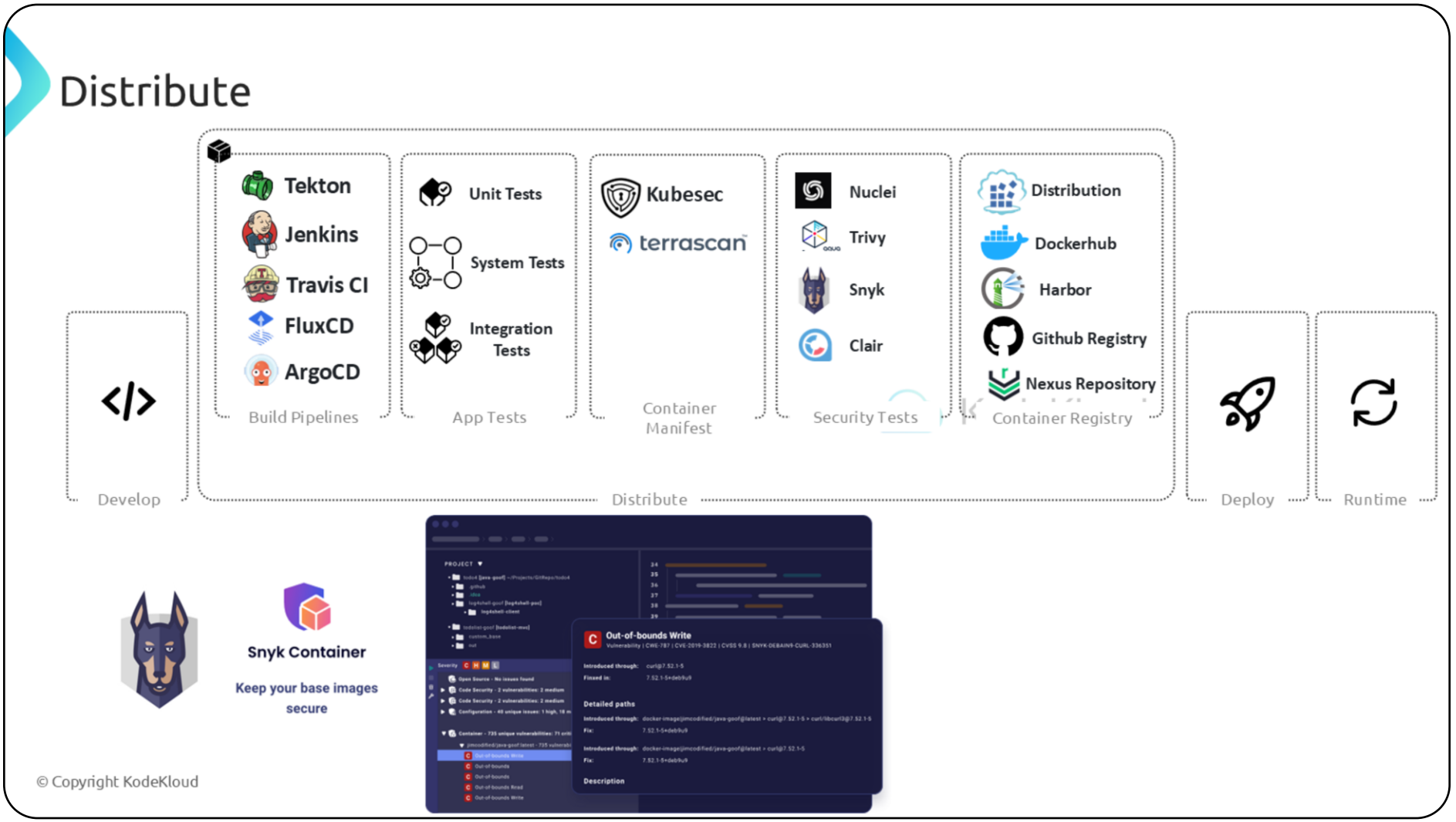
```
2022-05-30T15:47:04.096+0300    INFO    Detected OS: alpine
2022-05-30T15:47:04.096+0300    INFO    Detecting Alpine vulnerabilities...
2022-05-30T15:47:04.098+0300    INFO    Number of language-specific files: 1
2022-05-30T15:47:04.098+0300    INFO    Detecting python-pkg vulnerabilities...
2022-05-30T15:47:04.099+0300    WARN    This OS version is no longer supported by the distribution: alpine 3.9.2
2022-05-30T15:47:04.099+0300    WARN    The vulnerability detection may be insufficient because security updates are not provided

python:3.4-alpine (alpine 3.9.2)

Total: 37 (UNKNOWN: 0, LOW: 4, MEDIUM: 16, HIGH: 13, CRITICAL: 4)
```

Library	Vulnerability	Severity	Installed Version	Fixed Version	Title
expat	CVE-2018-20843	HIGH	2.2.6-r0	2.2.7-r0	expat: large number of colons in input makes parser consume high amount... https://avd.aquasec.com/nvd/cve-2018-20843
expat	CVE-2019-15903	HIGH	2.2.6-r0	2.2.7-r1	expat: heap-based buffer over-read via crafted XML input https://avd.aquasec.com/nvd/cve-2019-15903
libbz2	CVE-2019-12900	CRITICAL	1.0.6-r6	1.0.6-r7	bzip2: out-of-bounds write in function BZ2_decompress https://avd.aquasec.com/nvd/cve-2019-12900

And before we push the images to container registries, we want to make sure they are scanned for vulnerabilities. So tools like Nuclei, kube-hunter and kube-bench help here. Nuclei is a cutting-edge, high-performance vulnerability scanner that uses straightforward YAML-based templates. It enables you to create custom vulnerability detection scenarios that replicate real-world conditions, ensuring zero false positives.

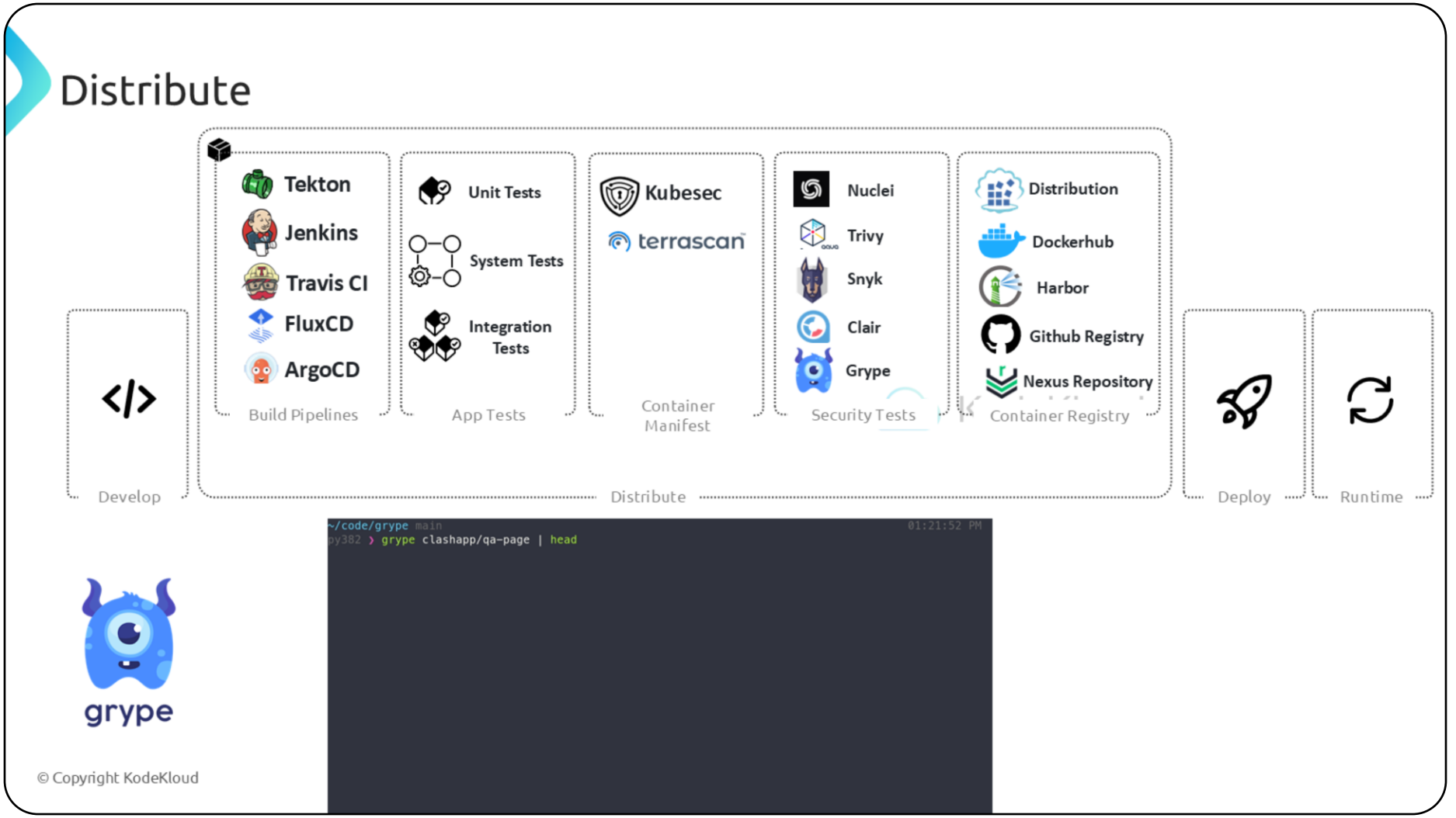
Another tool is Trivy which is a comprehensive and versatile security scanner. Trivy has *scanners* that look for security issues, and *targets* where it can find those issues.

It can analyze:
•Container images
•Filesystems
•Git repositories
•VM images
•Even Kubernetes clusters
It’s lightweight, fast, and supports YAML-based templates for custom scenarios.

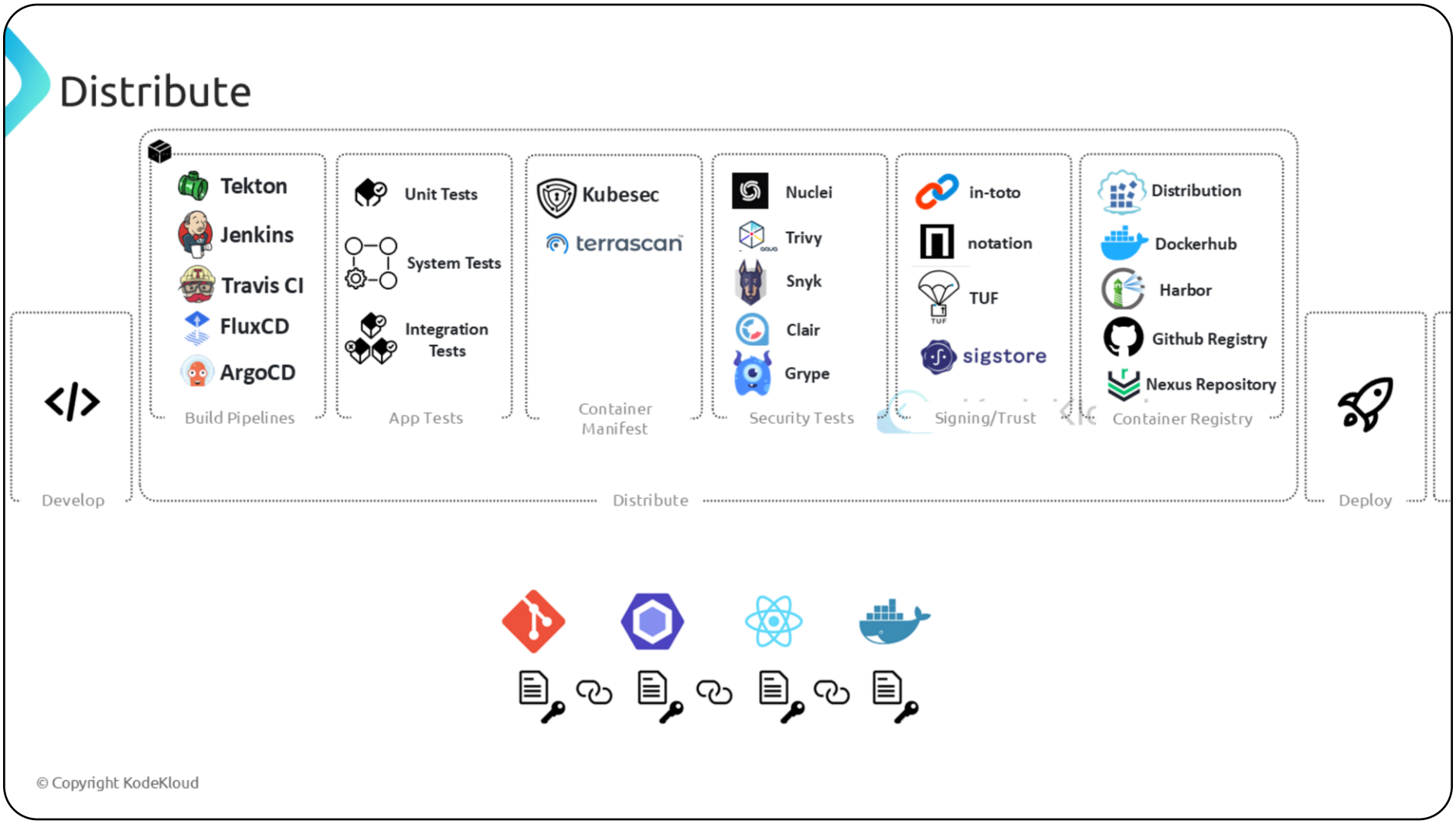


Snyk Container helps Find and automatically fix container and workload vulnerabilities. The Integrated IDE checks Detect vulnerabilities in base image dependencies, Dockerfile commands, and Kubernetes workloads while coding to fix issues early and save development time.

Another tool in this category is clair. Clair is an open source project for the static analysis of vulnerabilities in application container. You may use the Clair API to index their container images and can then match it against known vulnerabilities.



A lastly another tool that would fall into this category is Grype. I is a vulnerability scanner for container images and filesystems.



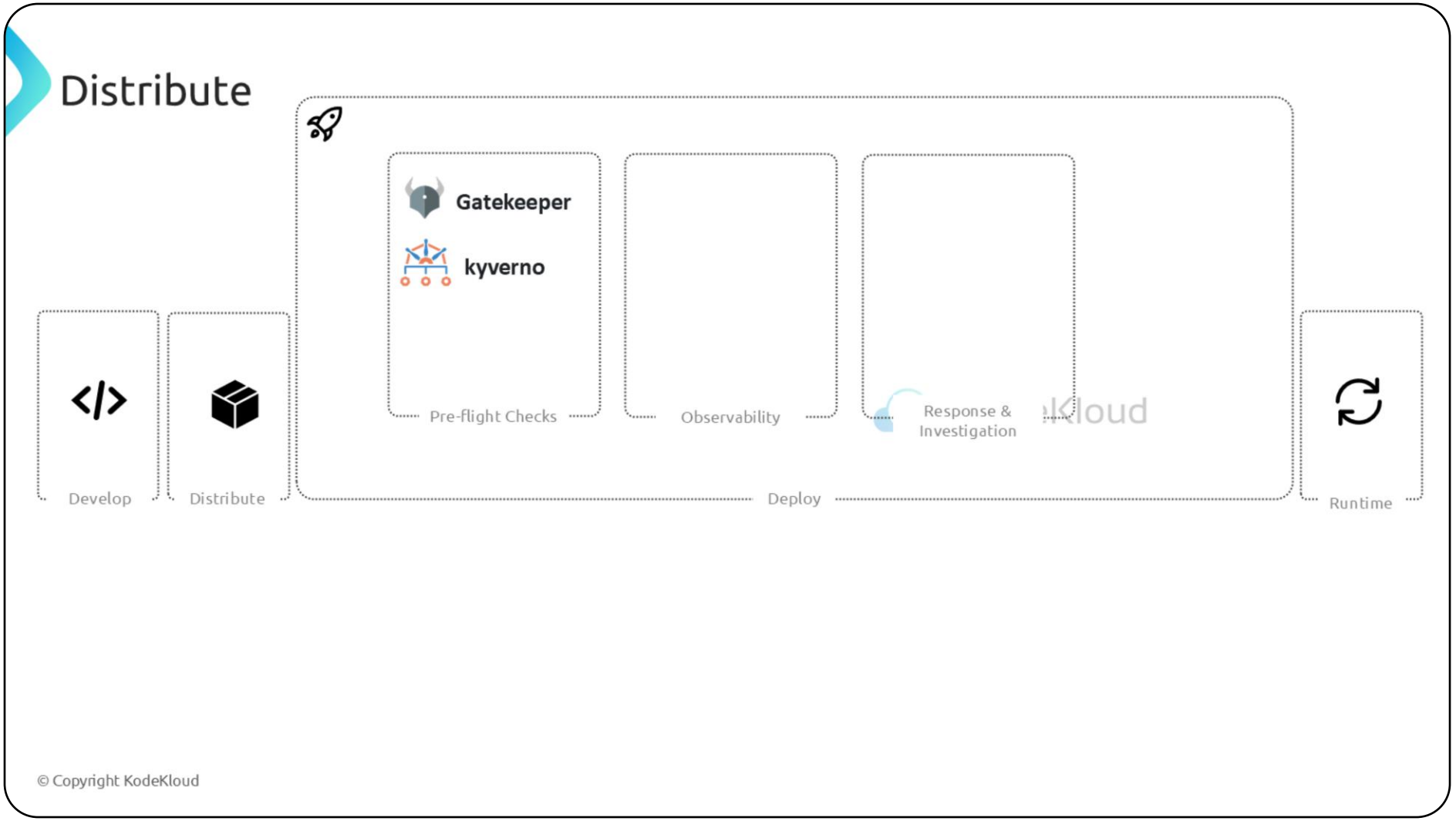
And another key area is the signing, trust and integrity. We discussed this thoroughly at different parts of the course.

Tools like in-toto that is A framework to secure the integrity of software supply chains.

Notation is a CLI tool for adding and verifying signatures in the OCI registry ecosystem, akin to checking Git commit signatures, but designed for broader use cases.

The Update Framework (TUF) maintains the security of software update systems, providing protection even against attackers that compromise the repository or signing keys. TUF provides a flexible framework and specification that developers can adopt into any software update system.

And we have Sigstore which is an open-source project that simplifies signing, verifying, and protecting software supply chains. It provides tools and services like transparency logs, keyless signing, and certificate-based authentication, ensuring secure provenance for software artifacts without the need for managing long-term keys.

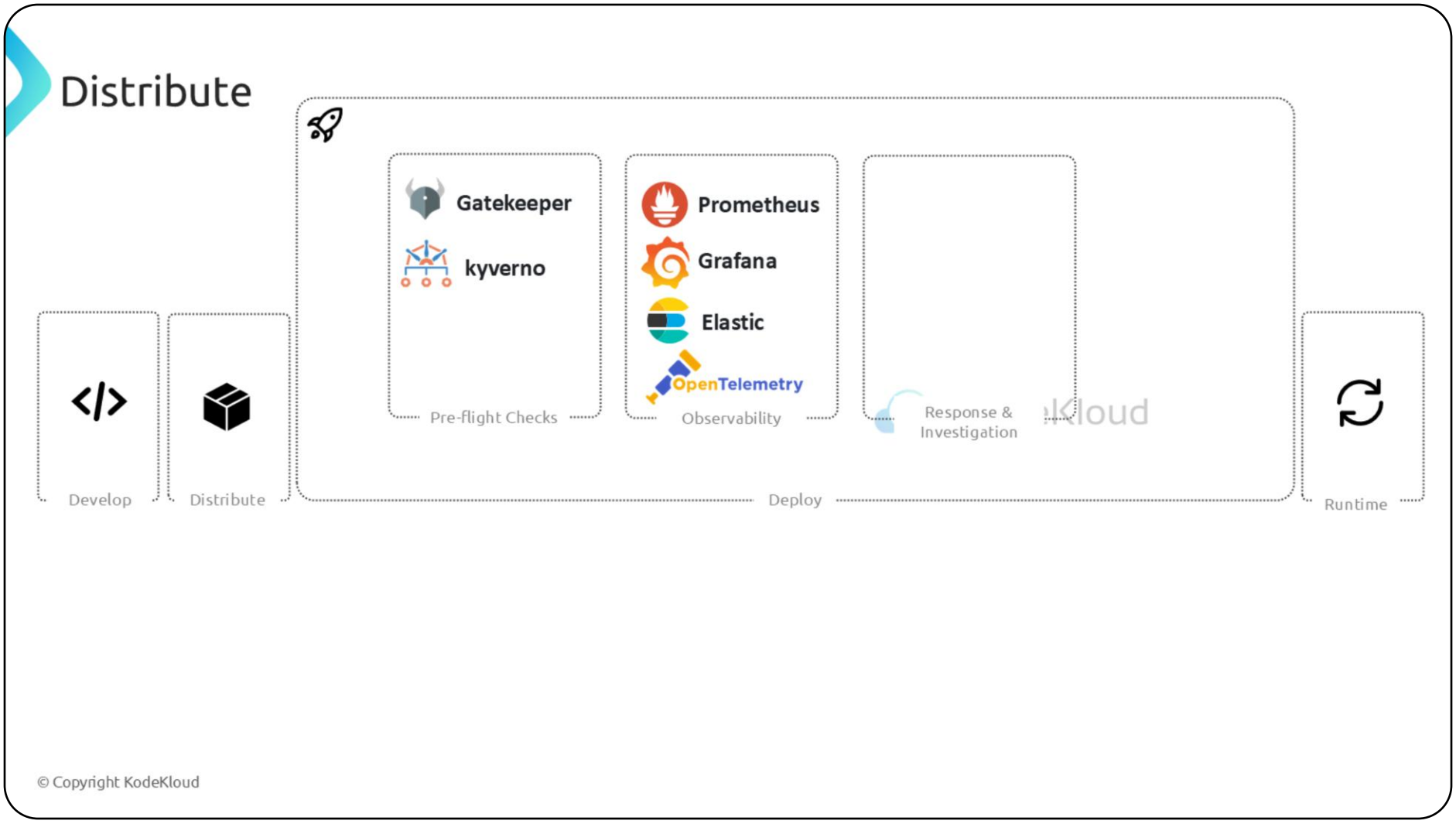


So let's move into the deploy phase. Your app is going to take off and before it does there are some pre-flight checks it must pass. So here we'll look into some pre-flight checks, observability and response & investigation requirements.

OPA Gatekeeper enforces policies in Kubernetes by evaluating configurations against defined rules written in the Rego policy language. It helps ensure best practices, such as requiring resource limits on containers or blocking privileged containers.

Kyverno simplifies policy management in Kubernetes by allowing policies to be written in YAML. It not only validates

configurations but can also automatically modify or "mutate" resources to align with predefined policies. For example, it can add missing labels or enforce image signing requirements.



Once your workloads are deployed, observability is essential to understand how your applications and infrastructure are behaving. It's about collecting, monitoring, and analyzing metrics, logs, and traces. Let's talk about the key tools:

Prometheus
Prometheus is a powerful open-source tool for collecting metrics. It's widely used in Kubernetes environments to monitor things like CPU, memory usage, or application performance. Prometheus uses a pull-based model, scraping data from endpoints, and you can set up alerts for specific thresholds or anomalies.

Grafana

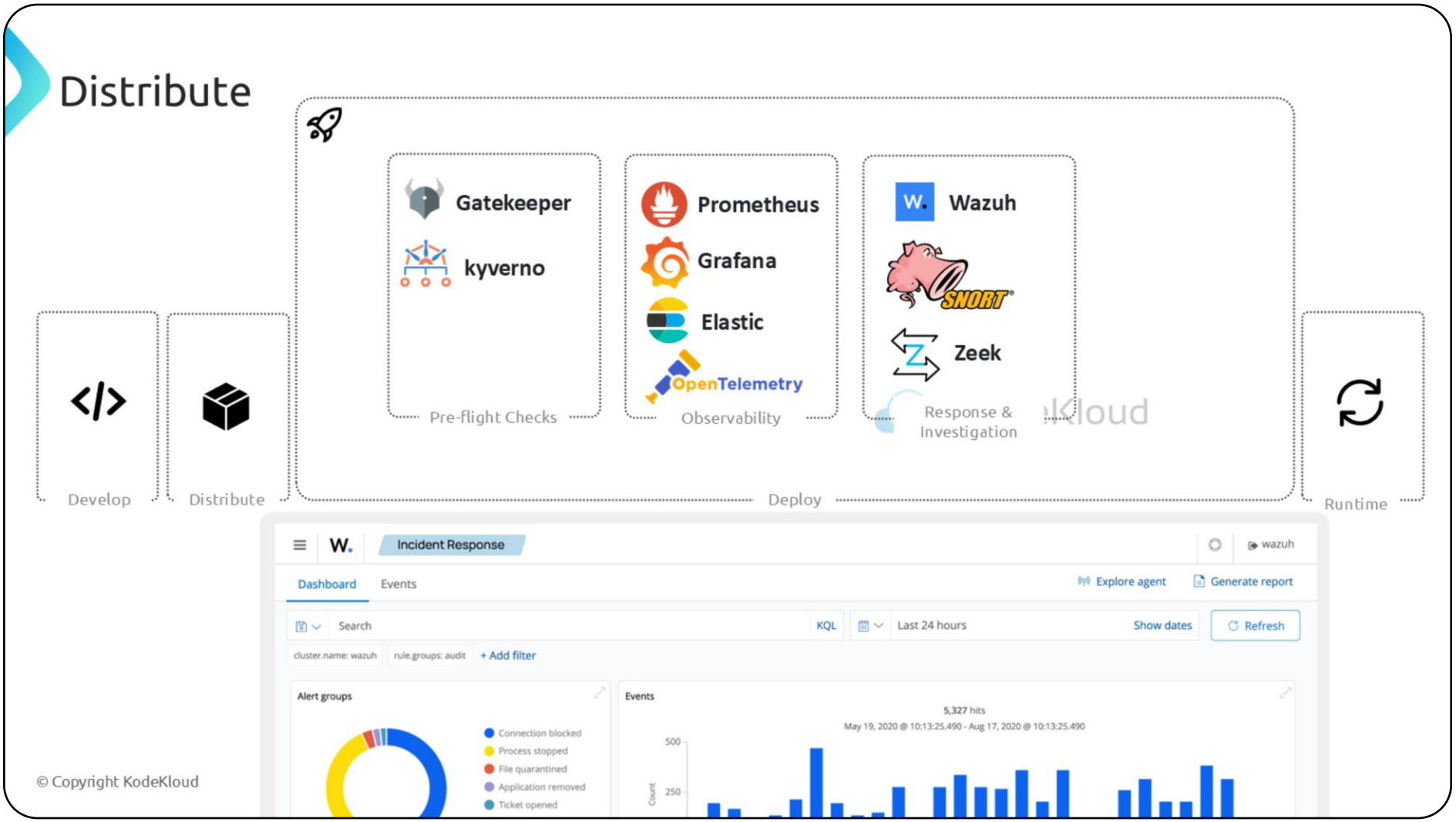
Grafana complements Prometheus by providing visualization. It lets you create custom dashboards to display metrics from Prometheus (or other sources). Instead of poring over raw data, you get clear graphs, making it easier to spot trends or issues at a glance.

Elasticsearch

Elasticsearch focuses on logs. It’s a highly scalable search engine that indexes and stores log data. Combined with Kibana, you can search and visualize logs to troubleshoot issues or investigate patterns.

OpenTelemetry

OpenTelemetry is the standard for collecting distributed traces, logs, and metrics. It helps provide end-to-end observability across microservices, ensuring you can trace the path of a request through multiple services and spot where delays or failures occur.



After monitoring and detecting an issue, the next step is to respond and investigate. For this, tools like Wazuh, Snort, and Zeek are commonly used.

Wazuh

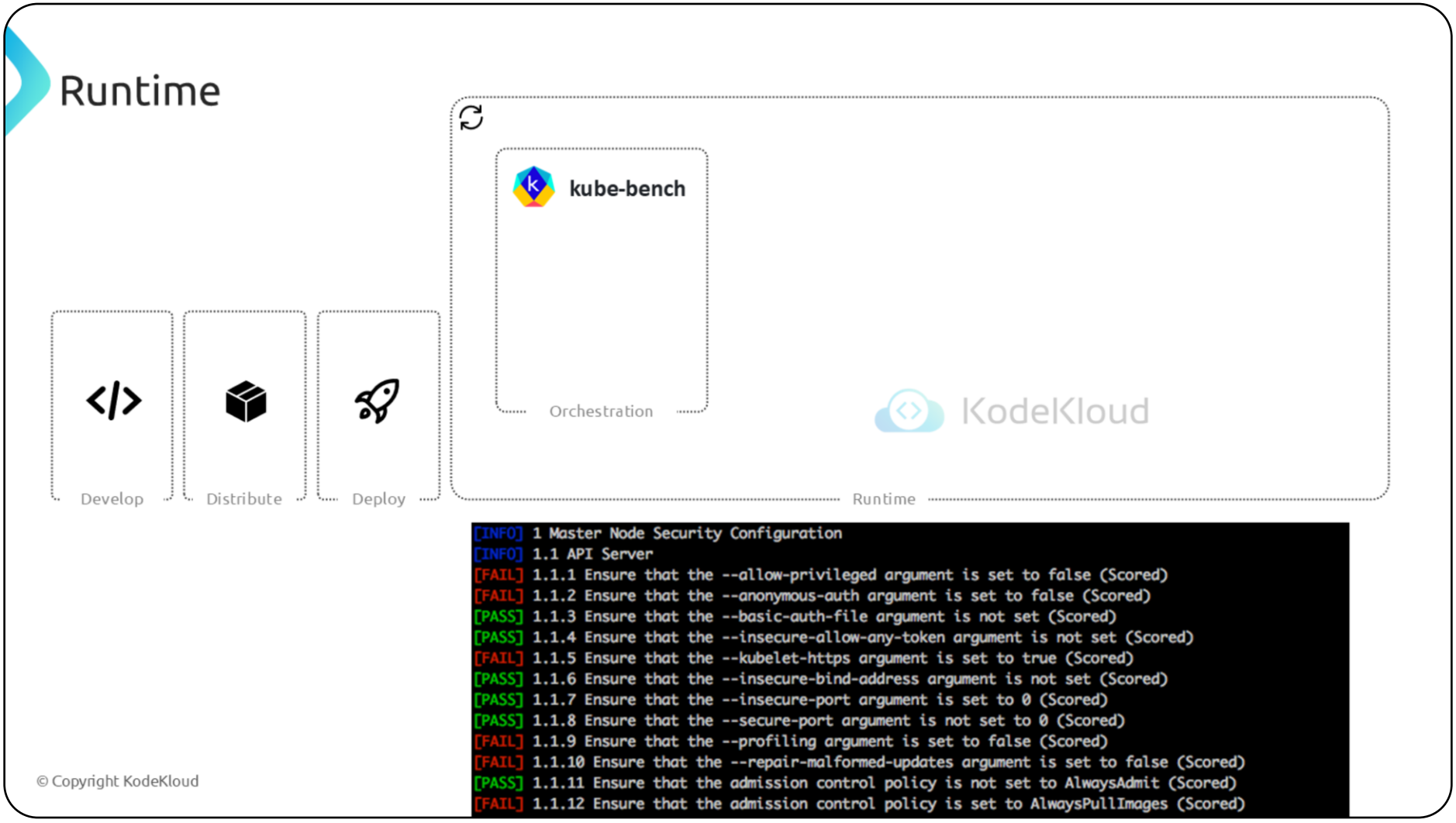
Wazuh is a comprehensive security monitoring and intrusion detection system (IDS). It helps detect vulnerabilities, policy violations, or unusual activity in your infrastructure. It also integrates with other tools to centralize logs, correlate events, and trigger alerts.

Snort

Snort is a lightweight intrusion detection and prevention system (IDPS). It monitors network traffic in real-time, identifying threats like suspicious payloads, port scans, or brute-force attacks. It’s rule-based, meaning you can define conditions for detecting specific types of attacks.

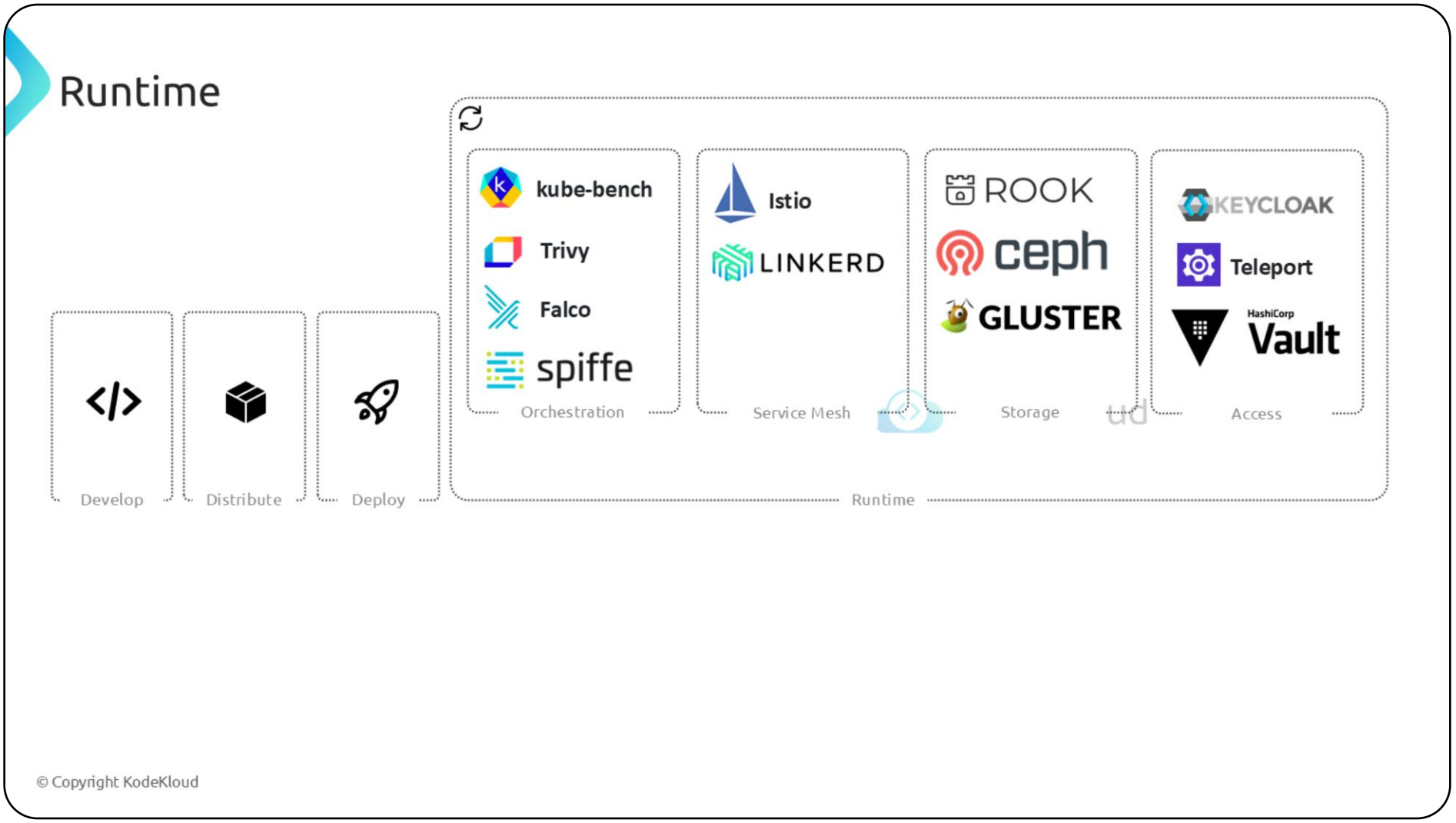
Zeek

Zeek (formerly Bro) is a network security monitoring tool. Unlike Snort, which focuses on packet-level analysis, Zeek works at a higher level, analyzing network flows and protocols to uncover anomalies or complex attack patterns. It’s particularly useful for detailed forensic investigations after an incident.



In the Runtime phase, the application is live and running, so the focus is on securing operations, managing access, ensuring storage reliability, and maintaining service connectivity. Let’s break this down:

At the orchestration layer we want to make sure the underlying infrastructure is benchmarked against industry standard best practices. Kube-bench checks your Kubernetes clusters against the CIS (Center for Internet Security) benchmarks. It ensures that configurations follow security best practices, such as securing API servers and limiting privileged access.



Trivy: A versatile security scanner that can analyze container images, Kubernetes clusters, and more. In the runtime context, Trivy monitors for vulnerabilities and misconfigurations within running workloads.

Falco: An open-source runtime security tool that monitors Kubernetes clusters for suspicious activity. It watches system calls and triggers alerts for policy violations, such as containers spawning unexpected processes or writing to restricted directories.

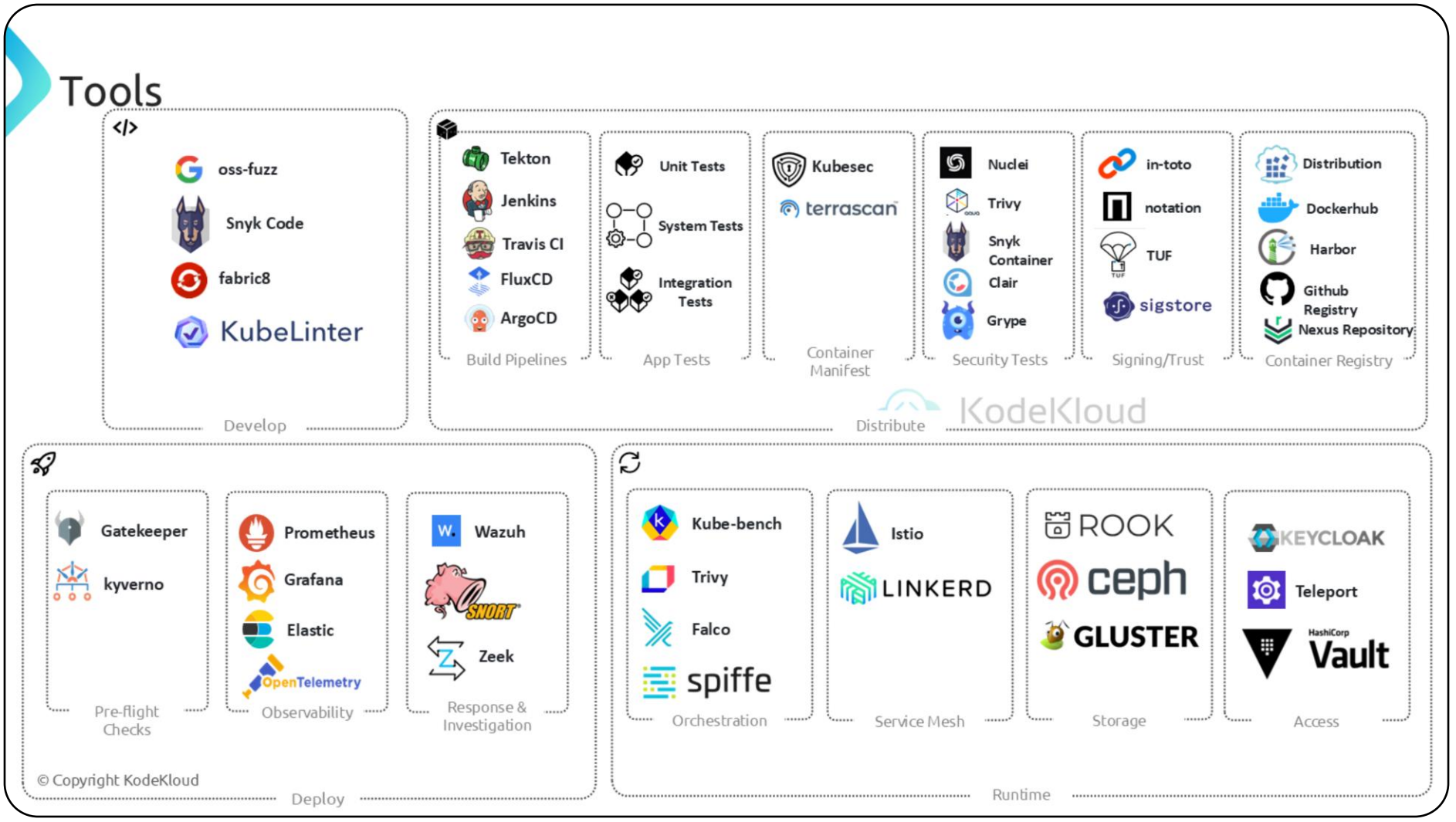
SPIFFE/SPIRE: Focused on workload identity, SPIFFE (Secure Production Identity Framework for Everyone) and its

implementation SPIRE provide secure identity management for microservices. It establishes trust between workloads using certificates and cryptographic methods.

Service meshes simplify and secure communication between microservices. The most popular options are:
•Istio: A powerful service mesh that provides traffic management, observability, and security for microservices. It enables features like service-to-service encryption, mutual TLS (mTLS), and granular access policies.
•Linkerd: A lightweight service mesh designed for simplicity and performance. It focuses on reliability and security while being easier to set up compared to Istio. Linkerd also supports mTLS for secure communication.

Reliable and secure storage is critical for running applications that require persistent data. Tools in this category include:
•Rook: A storage orchestrator that makes it easy to deploy and manage storage backends in Kubernetes. It supports multiple storage engines like Ceph and GlusterFS.
•Ceph: A scalable, distributed storage system that handles block, object, and file storage. It's commonly used for applications requiring high availability and reliability.
•GlusterFS: Another distributed storage solution, GlusterFS is known for its scalability and flexibility in handling large volumes of data. It's simpler than Ceph but effective for specific use cases.

Managing secure access to applications and infrastructure is critical in runtime. Here are the tools that help:
•Keycloak: An open-source identity and access management tool that provides authentication and authorization services. It supports single sign-on (SSO), user federation, and social login integration.
•Teleport: A secure access solution for infrastructure. It allows engineers to access servers, Kubernetes clusters, databases, and applications using short-lived certificates instead of passwords or static keys, ensuring zero-trust principles.
•HashiCorp Vault: A robust tool for managing secrets, credentials, and encryption keys. Vault secures sensitive data through access policies, encryption as a service, and automated secret rotation.



So that's a high level overview of all the tools in different phases of the lifecycle. I know it can be a bit too much but having a high level understanding of what tools fall in what category and what they do can help you make informed decisions when designing, implementing, or securing your cloud-native environments. It ensures you're using the right tools at the right stage, streamlining your workflows and improving overall security and efficiency.

Just to summarize in the develop phase you have tools that help developers catch issues early – the shift left approach we spoke about. So using tools that integrate with developer IDEs like Snyk Code, fabric8 or kube-linter helps identify potential

issues as you code.

In the distribute phase tools that you are already familiar with in building pipelines would be jenkins, travis CI, FluxCD and ArgoCD. Tools like kubesec and terrascan help identify potential issues in manifest files. Post which the images are built and they are scanned for security vulnerabilities using tools like trivy, snyk container, clair or grype. Before pushing the images to container registries like Docker Distribution, docker hub, harbor, github registry and nexus repository we use frameworks like in-toto, notation, tuf and sigstore to sign the images for integrity.

Moving on to the deploy phase, pre-flight checks help ensure only trusted images and valid configurations are deployed on the cluster. Admission controllers with OPA Gatekeeper and keyverno help here. For observability, prometheus, grafana, elastic and open telemetry come to the rescue. To detect, investigate and respond to attacks tools like wazuh, snort and zeek are good examples.

Finally in the runtime phase we have tools like kube-bench and trivy that helps ensure the underlying cluster is built following security benchmarks and best practices. Tools like Falco help watches system calls and triggers alerts for policy violations. Frameworks like SPIFFE help in bootstrapping security into clusters when they are built.

Istio and LinkerD are good solutions for Service Mesh and Rook, ceph and gluster are popular solutions for storage. Keycloak, teleport, and vault are examples of tools that help secure access within the cluster .

Well that's all for now.

KodeKloud

Visit www.kodekloud.com to learn more.