KubeCon

CloudNativeCon

THE LINUX FOUNDATION

OPEN SOURCE SUMMIT

AI_dev

Open Source GenAI & ML Summit

China 2024

KubeCon | CloudNativeCon

THE LINUX FOUNDATION OPEN SOURCE SUMMIT

AI_dev Open Source GenAI & ML Summit

China 2024

Securing the Supply Chain
A Practical Guide to SLSA Compliance from Build to Runtime

August 21 2024 - Enguerrand Allamel, Ledger

About Me

Enguerrand **Allamel**

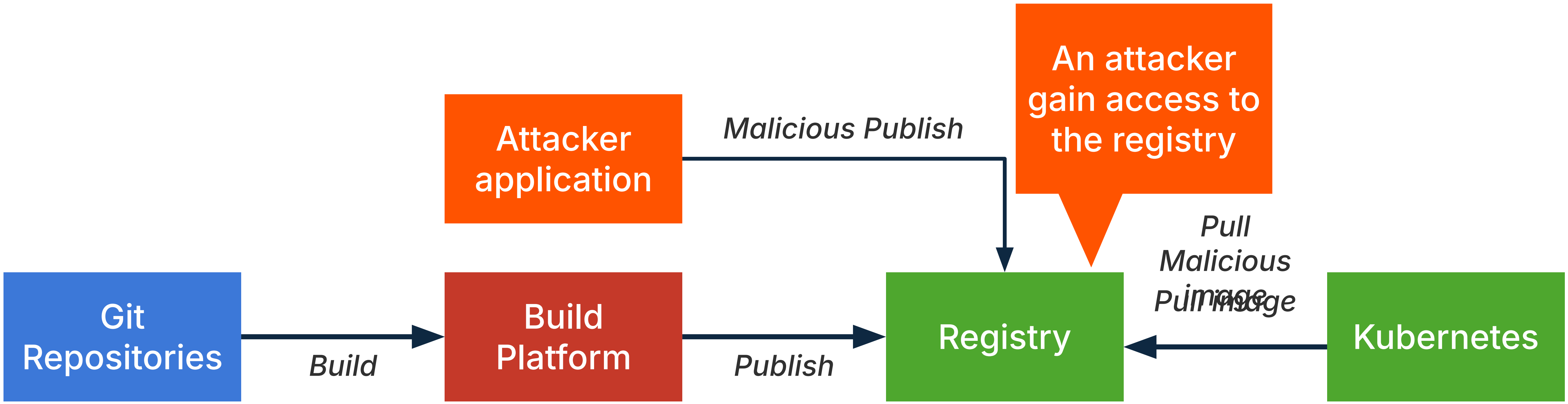
- **Academic Experience:** Studied for one year at **Tsinghua University**
- **Current Role:** Senior Cloud Security Engineer at **Ledger**
- **Company Overview: Ledger** specializes in secure hardware wallets and cutting-edge security products

LEDGER

Agenda

1. Why is **Supply Chain Security Important?**
2. What is **SLSA** (Supply-chain Levels for Software Artifacts)?
3. Possible **Milestones** for **Supply Chain Security Defense**
4. Example **Implementations**
   4.1. On the **Build** Side
   4.2. On the **Runtime** Side
5. Going Further with **HSM** (Hardware Security Module)

Example of a Supply Chain Attack



Why is Supply Chain Security Important ?

"*Gartner predicts* that by **2025**, **45% of organizations** will have experienced a **software supply chain attack**"*

Type attack	Known example
Submit unauthorized change to source git repository	**SushiSwap:** Contractor with repository access pushed a malicious commit redirecting cryptocurrency to itself More than $3 millions of users funds impacted
Compromise build process	**SolarWinds:** Attacker compromised the build platform and installed an implant that injected malicious behavior during each build Massive data breach Around 18 000 organisations impacted SolarWinds stock price drop by 40%

Source: Gartner report

What is SLSA?

- **SLSA:** Security Levels for Software Artifacts
- **Backing:** Sponsored by the OpenSSF (Open Source Security Foundation), associated with the Linux Foundation
- **Collaborative Framework:** Developed through cross-industry collaboration
- **Purpose:** Establishes standards and guidelines for securing software supply chains
- **Core Components:**
  - SLSA Requirements
  - SLSA Provenance (similar to attestation)
- **Audience:** Tailored for software producers, consumers, and infrastructure providers

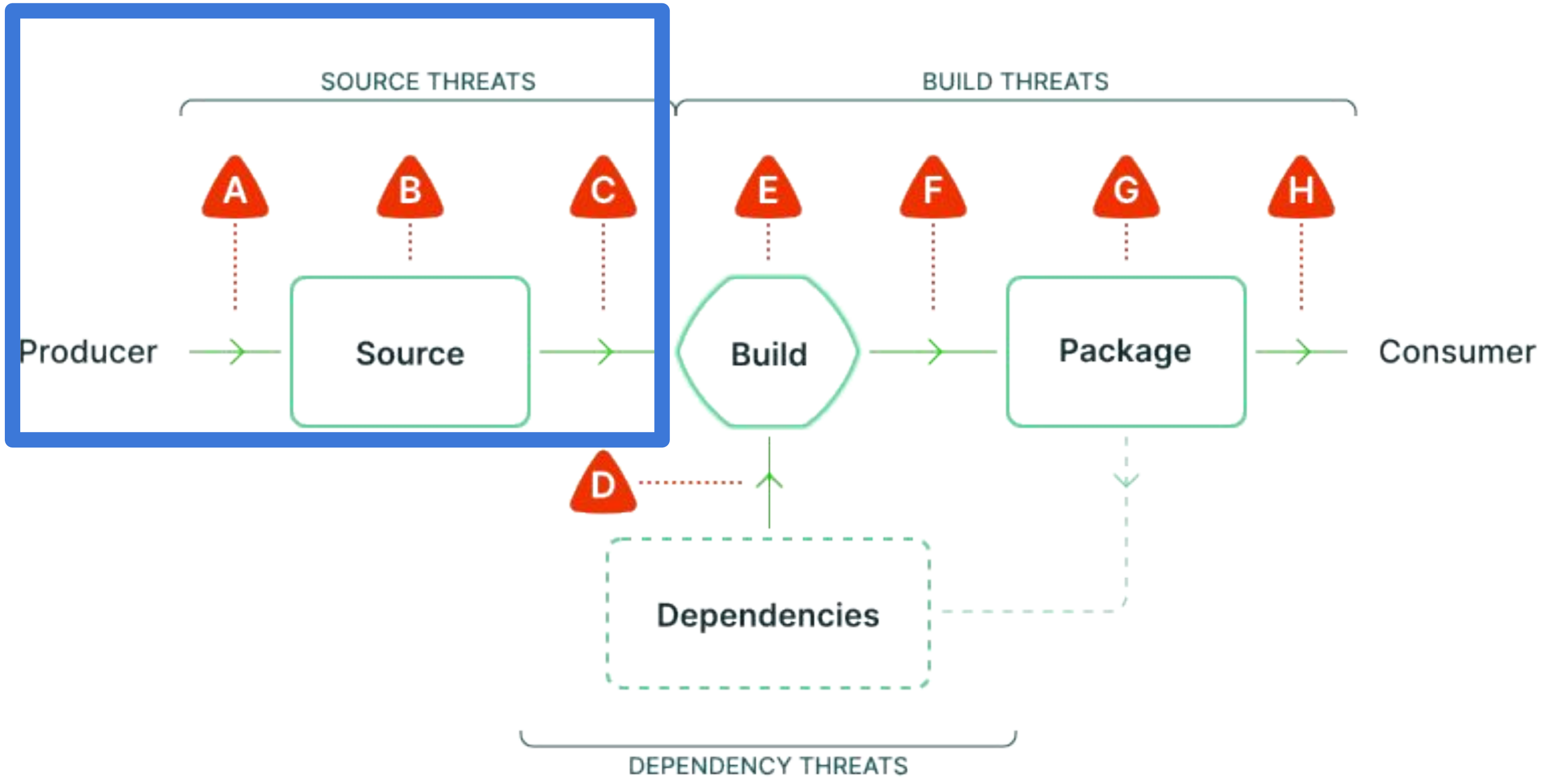
Website:

[*https://slsa.dev/*](https://slsa.dev/)

Github Repository:

https://github.com/slsa-framework/slsa

Scope of Threats and Attack in SLSA: Source



SOURCE THREATS

- A Submit unauthorized change
- B Compromise source repo
- C Build from modified source

DEPENDENCY THREATS

- D Use compromised dependency

BUILD THREATS

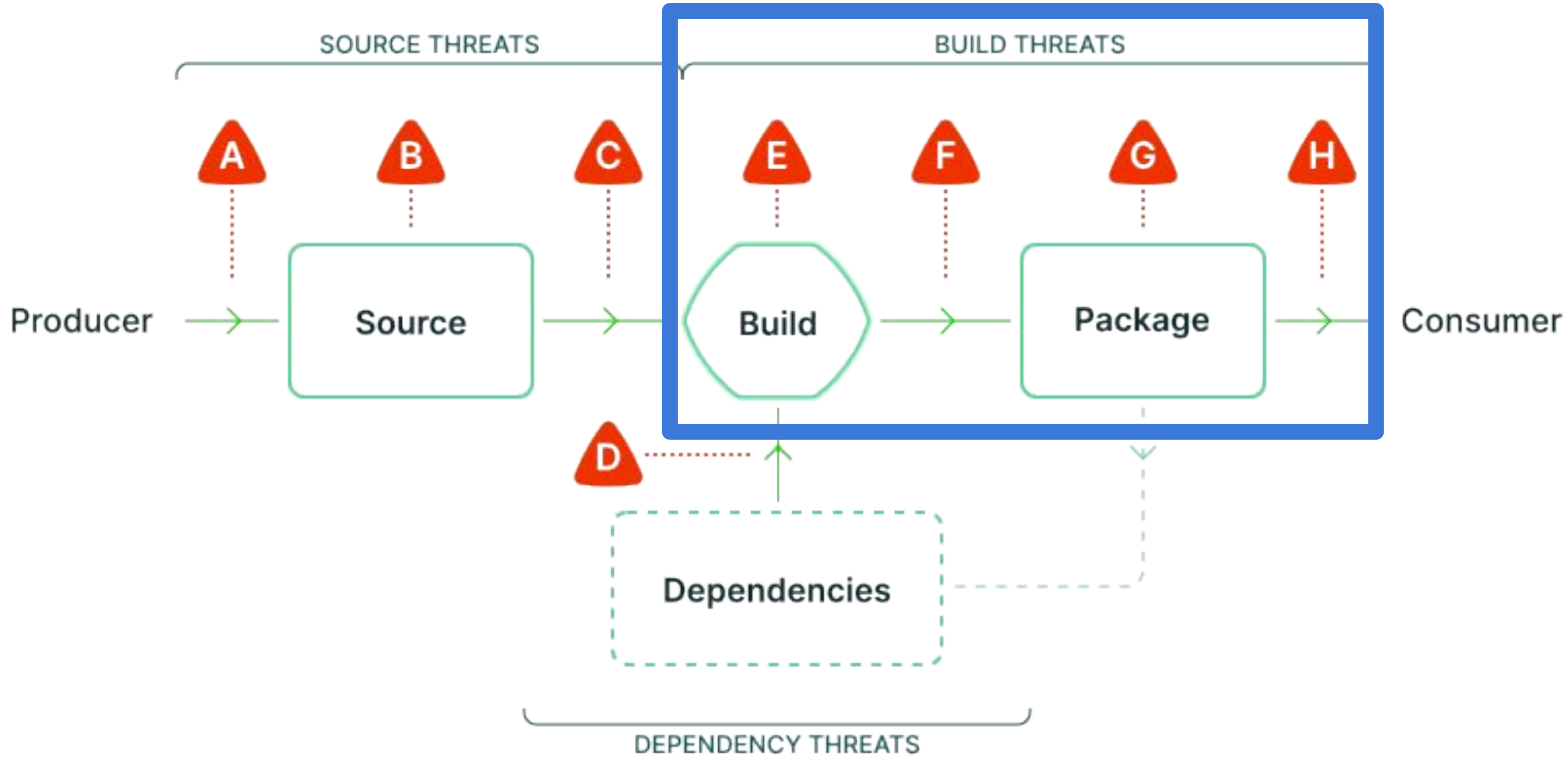
- E Compromise build process
- F Upload modified package
- G Compromise package registry
- H Use compromised package

Example:

- Code modification within a Git repository
- Permission bypass on a Git repository hosting platform (e.g., GitLab, GitHub, Gitea)

Diagram based from SLSA docs

Scope of Threats and Attack in SLSA: Build



Example:

- CI/CD or build platform compromised
- Package registry compromised

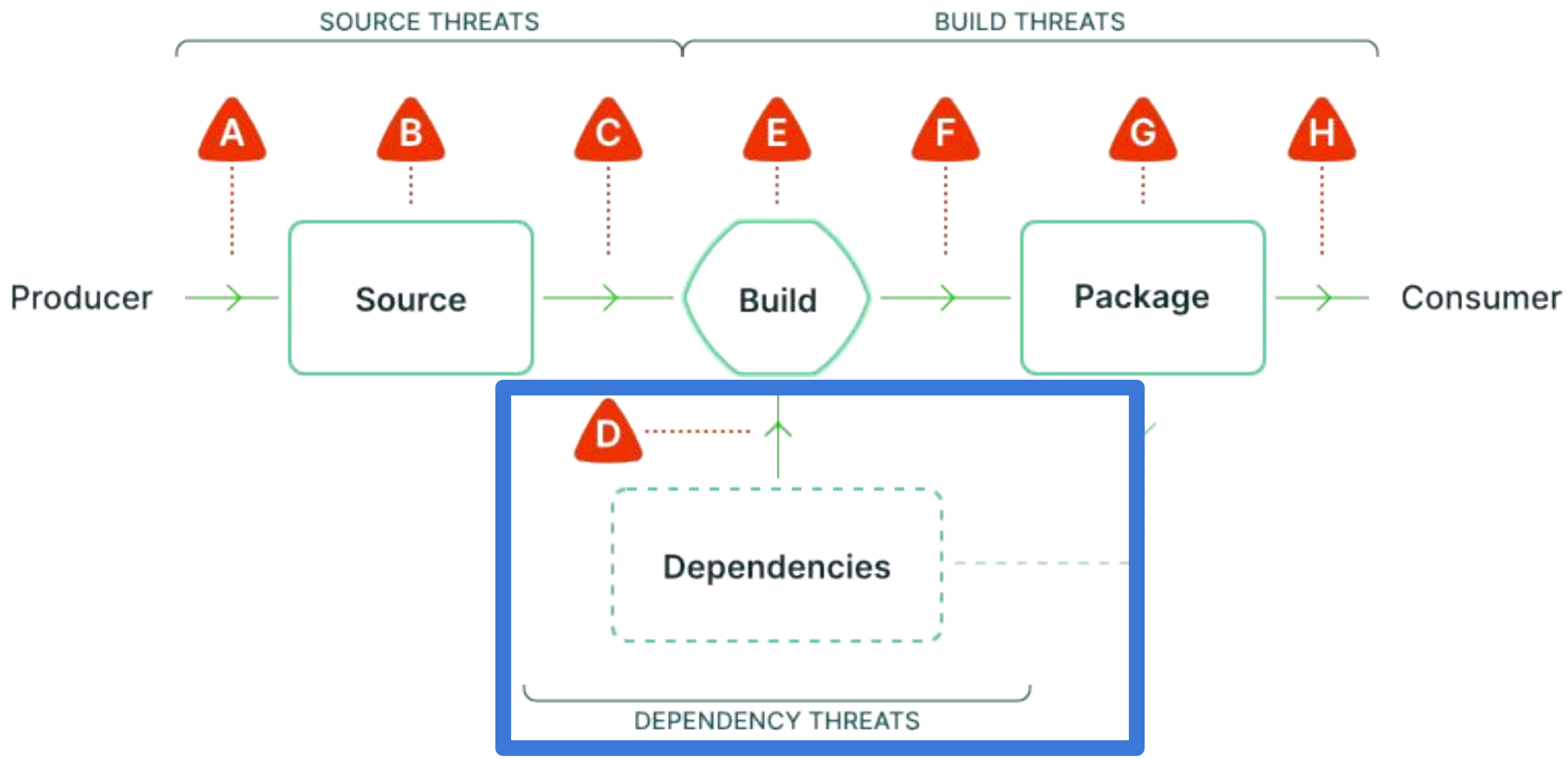
SOURCE THREATS
A Submit unauthorized change
B Compromise source repo
C Build from modified source

DEPENDENCY THREATS
D Use compromised dependency

BUILD THREATS
E Compromise build process
F Upload modified package
G Compromise package registry
H Use compromised package

Diagram based from SLSA docs

Scope of Threats and Attack in SLSA: Dependencies



SOURCE THREATS

A Submit unauthorized change

B Compromise source repo

C Build from modified source

DEPENDENCY THREATS

D Use compromised dependency

BUILD THREATS

E Compromise build process

F Upload modified package

G Compromise package registry

H Use compromised package

Example:

- Typosquatting of a package in dependencies hosted on platforms like PyPI.org, npmjs.com, etc.
- Malicious code embedded within dependencies

Diagram based from SLSA docs

Possible milestone for Supply Chain Security defense: On SLSA

Definition of Security Level link to Build Thread of SLSA

Target complexity	Level	Requirements	Focus
By default	**Build L0**	*(none)*	*(n/a)*
Easy	**Build L1**	Provenance showing how the package was built	Mistakes, documentation
Easy-Medium	**Build L2**	Signed provenance, generated by a hosted build platform	Tampering after the build
Hard	**Build L3**	Hardened build platform	Tampering during the build

The SLSA framework in version 1.0 defines levels only for build threats/tracks.

Table based from https://slsa.dev/spec/v1.0/levels

Possible Milestone for Supply Chain Security Defense

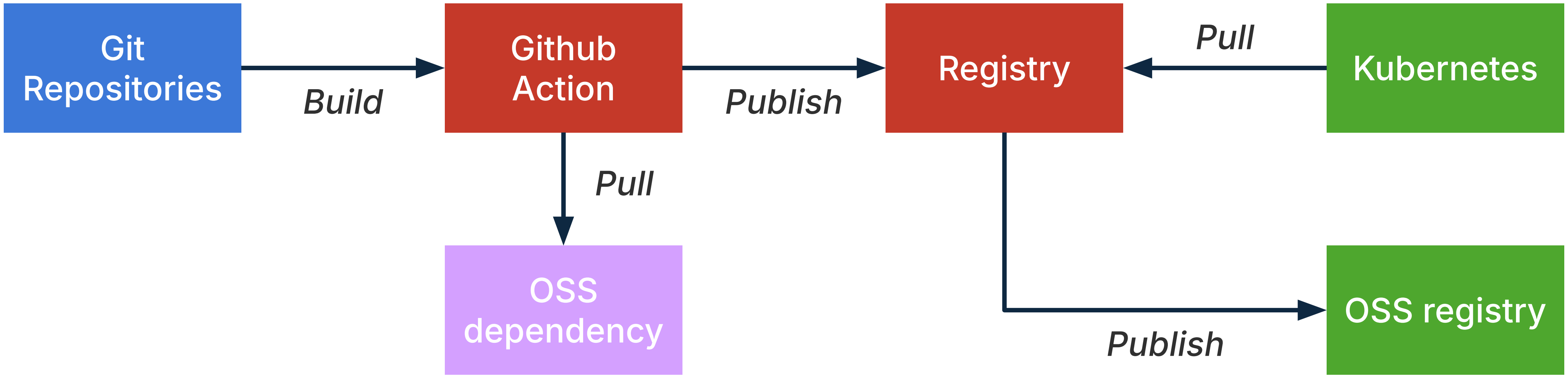
Possible Milestones of Supply Chain Security defense

Order	Example of practices	Focus
0: Default	*(none)*	*(n/a)*
1: First testing	First artifact/attestation signature locally, initial monitoring of image usage on a Kubernetes cluster, auditing current OSS usage/distribution	**Testing defense mechanisms and tooling**
2: Initial Defense Implementation	Build inside a CI/CD pipeline (not locally), signed artifact/attestation within the build platform, SBOM at runtime	Establishing a **basic level of Supply Chain Security defense**
3: Advanced defense	Hardened build platform, proxy for OSS registry, rebuilding OSS artifacts, HSM with CA signature key, exclusively keyless artifact signatures, etc.	Implementing **defense in depth** and **protection against advanced scenarios**

Example Implementation: Context

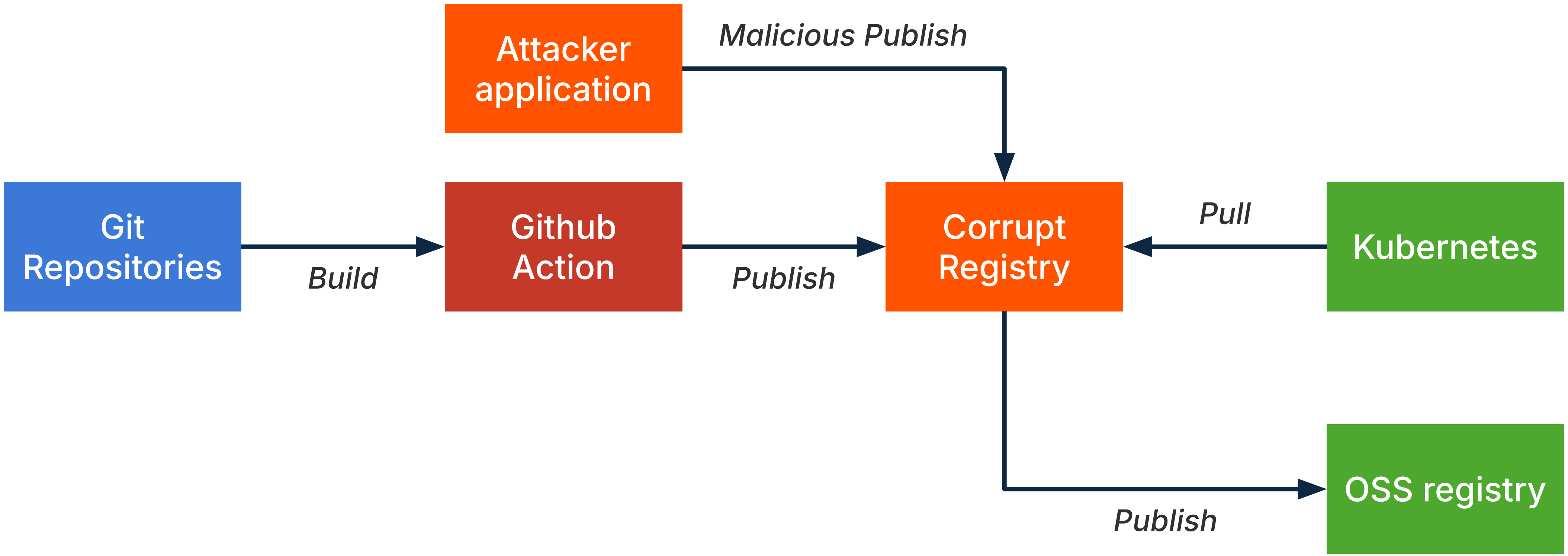
For this scenario, we assume the following setup:

- Applications are running on Kubernetes
- Build platform is Github Action
- Open Source (OSS) dependency is used
- Open Source (OSS) applications are built and deployed to public registry



Build Side: Possible Defense

- How to ensure that the software deployed on Kubernetes was built within GitHub Actions?
  - **Signature:** Use tools like Cosign, Notary, etc.
  - **Provenance:** Implement SLSA Provenance, In-Toto Attestation, etc.



Build Side: Sigstore

- **Sigstore**: Open source project for Software Supply Chain Security
- **Backing**: Sponsored by the OpenSSF (Open Source Security Foundation)
- **Purpose**: Provides a simple and secure way to sign software artifacts
- **Motto**: "Sign, Verify, Protect"
- **Core Functions:** Signature of Artifacts, Verification and Monitoring of Signatures
- **Supported Formats:** Works with blobs, container images, etc.
- **Tooling Provided: Cosign**: Command-line interface (CLI) for signing, **Fulcio**: Keyless signature authority, **Rekor**: Transparent metadata logging, etc

Documentation:

https://docs.sigstore.dev/

Github Organisation:

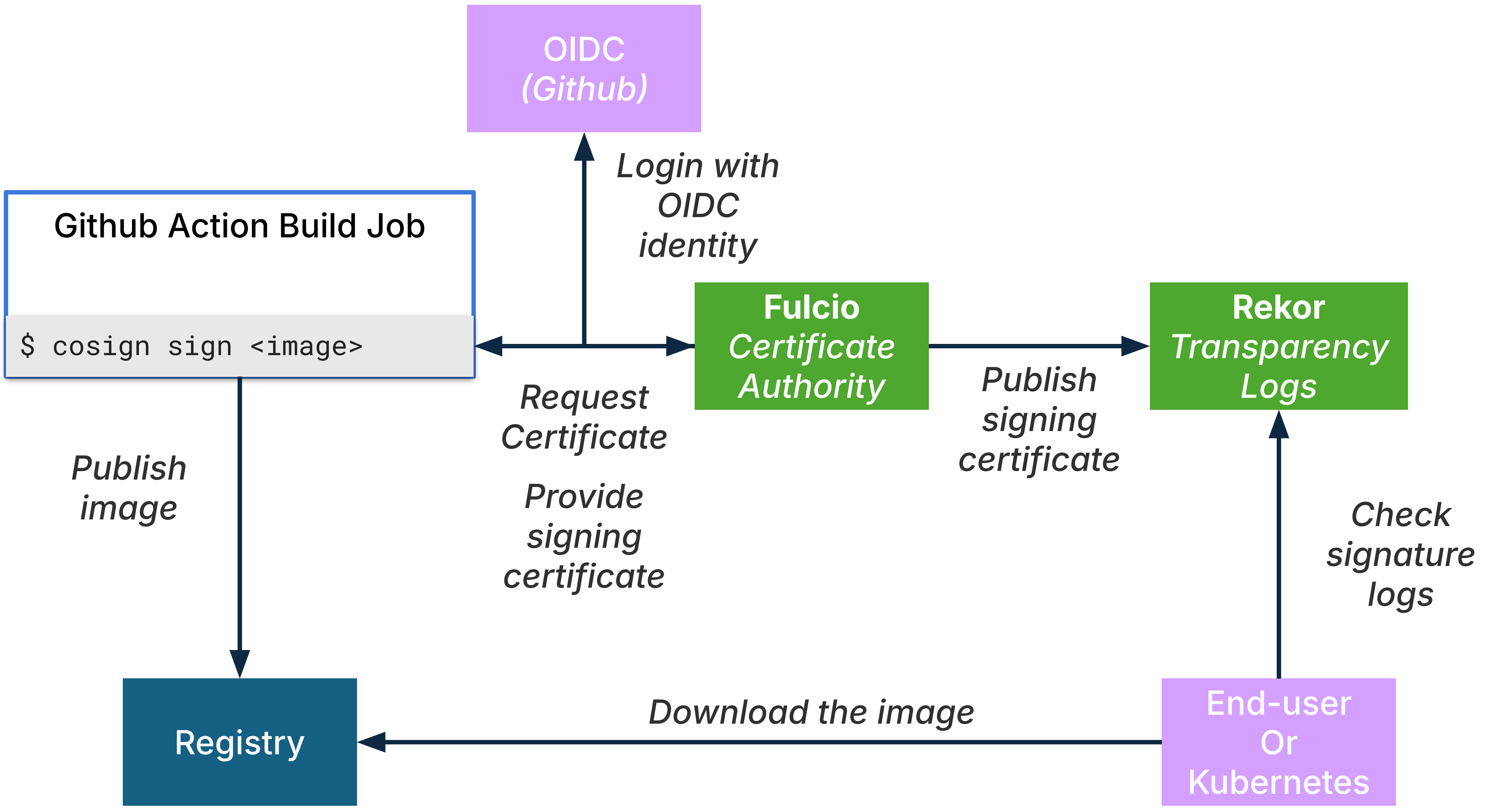
https://github.com/sigstore

Build Side: Signature Keyless vs Static

Type	Descriptions	Cosign CLI command
Static	Generated private/public keys are used, self-managed and not based on OIDC	`$ cosign sign --key cosign.key myimage:v1`
Keyless	Ephemeral keys are used, based on OIDC identity (e.g., GitHub, Google, Microsoft)	`$ cosign sign myimage:v1`

Default Recommendation: Keyless signatures are used by default and recommended for enhanced security and transparency

Build Side: Signature Keyless



Build Side: Signature Inside Github Action

- **CI/CD Integration:** Within the CI/CD pipeline, specifically during the build job in GitHub Actions, the container image is signed
- **Beyond Signature:** A signature alone isn't sufficient, attestation provides additional information to enhance security

```yaml
...
jobs:
  build-and-push:
    steps:
      - name: Install Cosign
        uses: sigstore/cosign-installer@v3
        ...
      - name: Load Docker metadata
        uses: docker/metadata-action@v5
        ...
      - name: Build and Push container images
        uses: docker/build-push-action@v6
        id: build-and-push
        ...
      - name: Sign the images with GitHub OIDC Token
        env:
          DIGEST: ${{ steps.build-and-push.outputs.digest }}
          TAGS: ${{ steps.docker_metadate.outputs.tags }}
        run: |
          images=""
          for tag in ${TAGS}; do
            images+="${tag}@${DIGEST} "
          done
          cosign sign --yes ${images}
```

Build Side: In-Toto Attestation

- **In-Toto:** Open source framework for protecting supply chain integrity
- **Backing:** Sponsored by the CNCF
- **Purpose:** Enhances transparency and security in the software supply chain
- **Global Scope:** Focused on supply chain security with integration in multiple languages, primarily Python
- **SLSA Integration:** Can incorporate SLSA Provenance specifications
- **Detailed Attestations:** Provides critical supply chain information, such as code testing results or code review attestations

in-toto

Website:

https://in-toto.io/

Demo (global project):

https://github.com/in-toto/demo

Attestation spec:

https://github.com/in-toto/attestation/tree/v1.0/

Build Side: In-Toto Attestation: Example

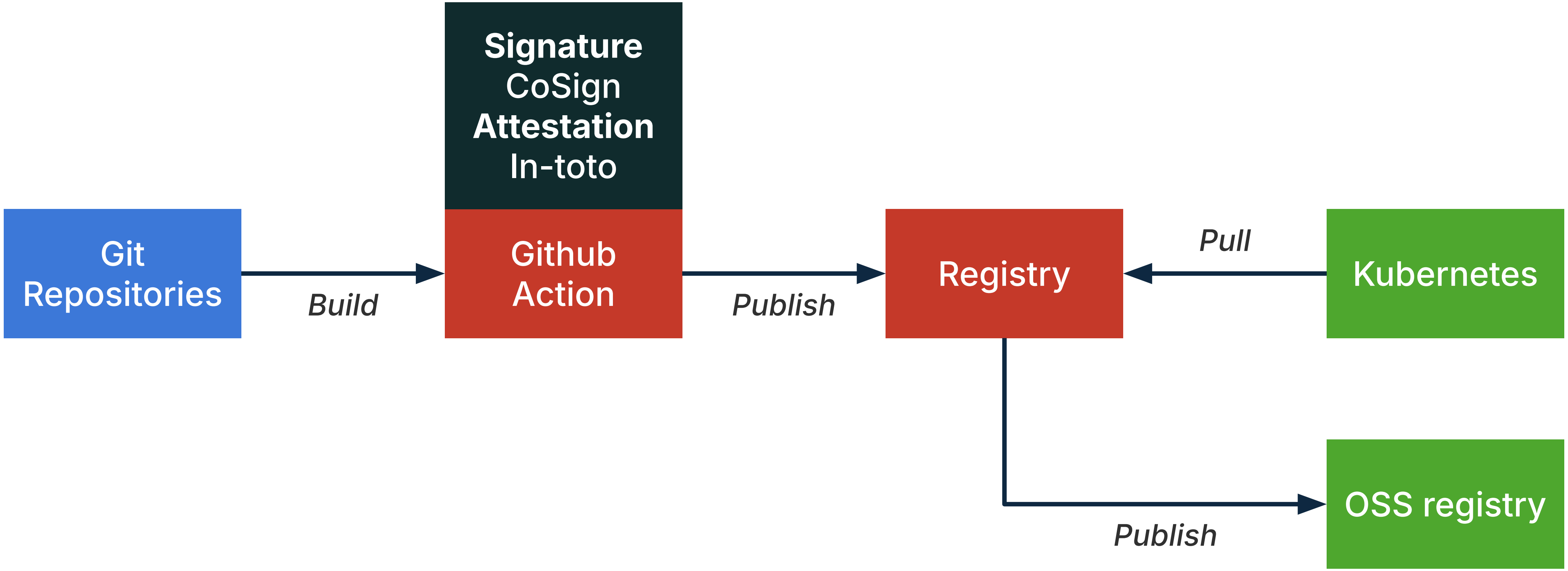
- **Predicate File:** Metadata or information embedded in the attestation
  - **Example:** Test results, runner details, build environment, etc.
- **Cosign Integration:** Cosign can create and sign predicate files, similar to how it handles containers or blobs
- **Enhanced Security:** Provides trusted information to software consumers
  - **Example:** Prove that tests have passed or that code has been reviewed

```
{
  "_type": "https://in-toto.io/Statement/v0.1",
  "predicateType":
"https://cosign.sigstore.dev/attestation/v1",
  "subject": [
    {
      "name": "ghcr.io/ledgerhq/signed-image",
      "digest": {
        "sha256": "<image-sha256>"
      }
    }
  ],
  "predicate": {
    "<my-data>": "<my-value>",
    "Timestamp": "2021-08-11T14:51:09Z"
  }
}
```

```
$ cosign attest --predicate <file> <image>
$ cosign verify-attestation <image>
```

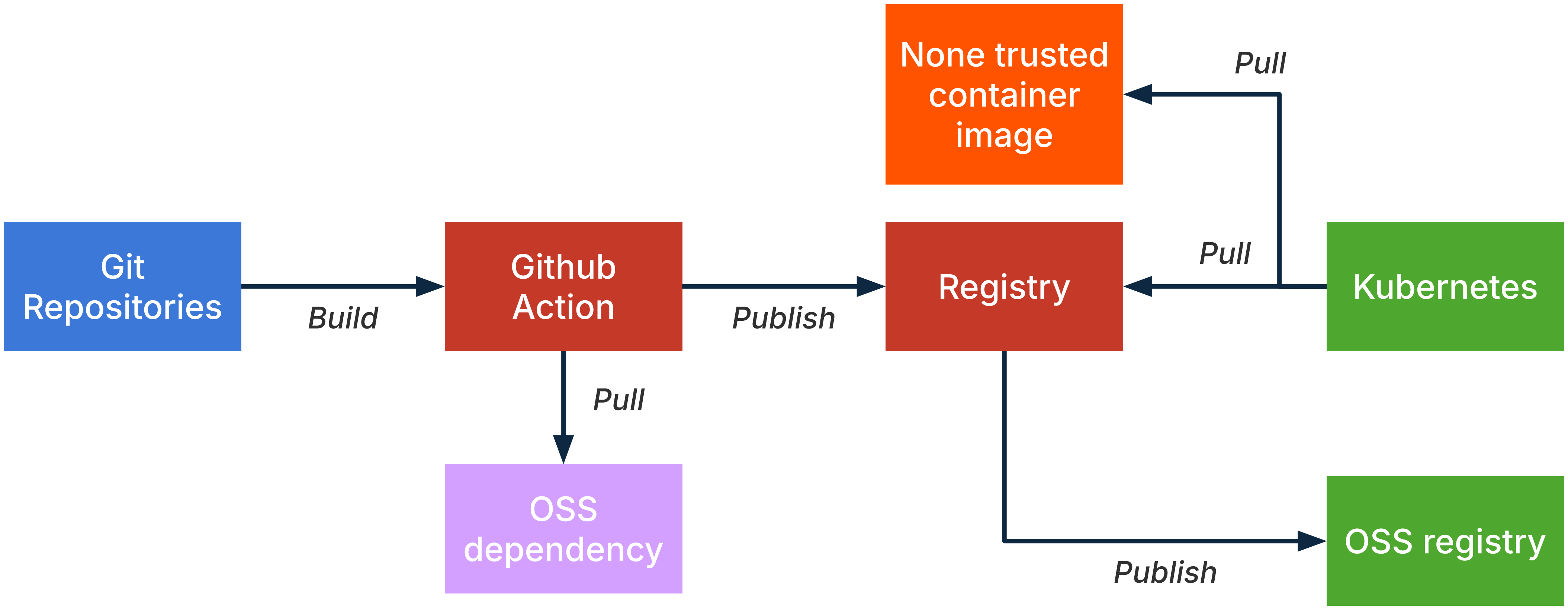
Build Side: Overview

- **Signature:** Executed within the build runner using Cosign
- **Attestation:** Performed within the build runner using Cosign in combination with In-Toto
- **Trusted Information:** Ensures the integrity of the build and artifact by providing verifiable details



Runtime Side: Possible Defense

- How to verify the application provenance running in production?
  - **Kubernetes Admission Controller:** Cosign Policy Controller, Kyverno, etc.
  - **Audit & Detection:** Kubescape, etc.



Runtime Side: Kyverno

- **Kyverno**: Open source policy engine for Kubernetes
- **Backing**: Sponsored by the CNCF
- **Purpose**: Enforces security, compliance, and operational policies in Kubernetes
- **Features**: Validates and cleans up Kubernetes resources, audits and reports policies, etc
- **Policy Format:** Policies are written as Kubernetes resources
- **Supply Chain Security Enforcement:** Ensures only properly signed and attested images are deployed

Kyverno

Website:

https://kyverno.io/

Github Organisation:

https://github.com/kyverno

Example policies link to Supply Chain Security:

https://kyverno.io/policies/?policytypes=Software%2520Supply%2520Chain%2520Security

Runtime Side: Verification

- **Signature Verification**: Policy to check the signature of the container image
- **Attestation Verification**: Policy to validate the content of the attestation
- **Example**: Ensure that all images matching a regex pattern are signed with the correct key

```yaml
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
  name: verify-image
spec:
  validationFailureAction: Enforce
  rules:
    - name: verify-image
      match:
        any:
        - resources:
            kinds:
              - Pod
      verifyImages:
      - imageReferences:
        - "ghcr.io/ledgerhq/signed-*"
        attestors:
        - entries:
          - keyless:
              subject: "https://<url-to-the-workflow>@<refs>"
              issuer: "https://token.actions.githubusercontent.com"
              rekor:
                url: https://rekor.sigstore.dev
```

Runtime Side: Kubescape

- **Kubescape**: Open source Kubernetes security platform
- **Backing**: Sponsored by CNCF (Cloud Native Computing Foundation) and linked to the Linux Foundation
- **Global Scope**: Focused on enhancing security in Kubernetes, CI/CD pipelines, and source code
- **Features**: Includes a Kubernetes scanner, CI/CD integrations, and more

Kubescape

Website:

https://kubescape.io/

Github Organisation:

https://github.com/kubescape

Runtime Side: Analyse your Kubernetes Cluster

- **Scan Execution:** Run scans based on predefined controls
- **Modes:** Scans can be executed one-time or in continuous mode
- **Registry Usage:**
  - Identify usage of trusted image registries
  - Detect usage of unsafe image registries (a good first step)
- **Image Signature Verification:**
  - Check if image signatures exist (a good first step)
  - Verify image signatures for authenticity

Additional Controls:

More controls available in the documentation: https://hub.armosec.io/docs/controls

Runtime Side: Analyse your Kubernetes Cluster

Control: **C-0237**: Check if signature exists *(https://hub.armosec.io/docs/c-0237)*

```
$ kubescape scan control "C-0237" -v
...
################################################################################
ApiVersion: apps/v1
Kind: Deployment
Name: my-hello-ledger
Namespace: default

Controls: 1 (Failed: 1, action required: 0)
```

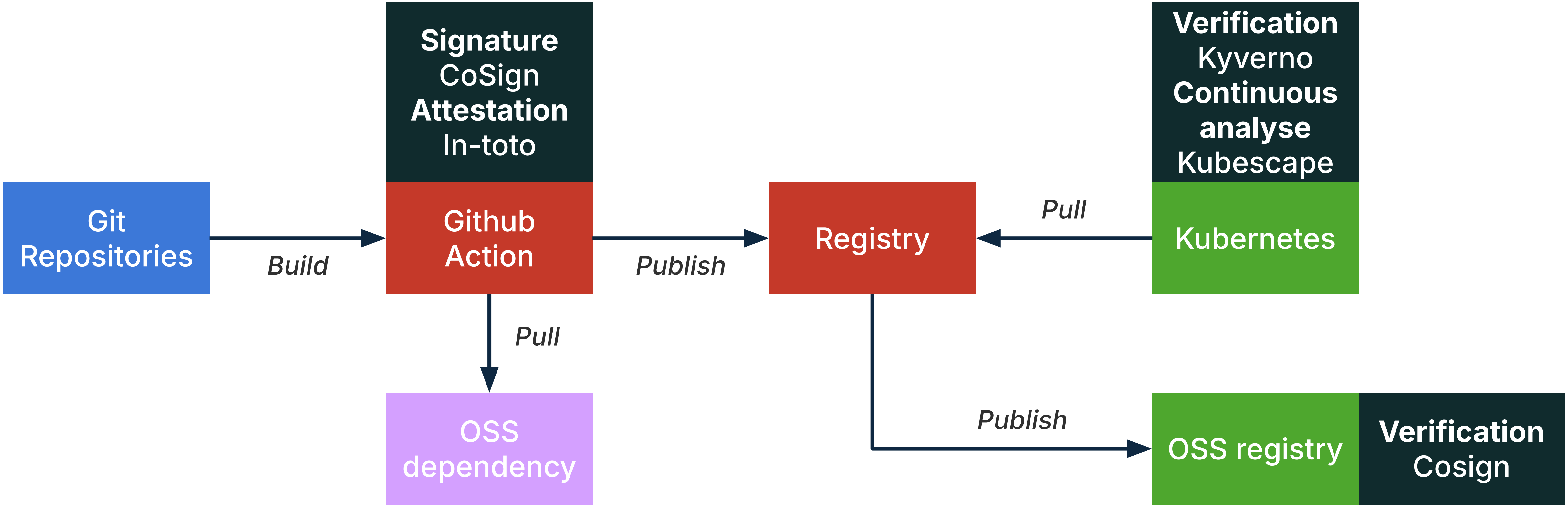
Severity	Control name	Docs	Assisted remediation
High	Check if signature exists	https://hub.armosec.io/docs/c-0237	spec.template.spec.containers[0].image

...

Severity	Control name	Failed resources	All Resources	Compliance score
High	Check if signature exists	8	10	20%
	Resource Summary	8	10	20.00%

Overview of this Implementation

- **Signature:** Executed within the build runner using Cosign
- **Attestation:** Performed within the build runner using Cosign in combination with In-Toto
- **Verification:** Conducted within the Kubernetes cluster to validate container image integrity, or locally using Cosign



Going Further with HSM (Hardware Security Module)

KubeCon CloudNativeCon THE LINUX FOUNDATION OPEN SOURCE SUMMIT AI_dev China 2024

- **HSM (Hardware Security Module):** A physical device designed for cryptographic operations
- **Private Certificate Protection:** HSM provides a high level of physical security to protect private certificates
- **Certificate Authority (CA):** Holds the root certificate, which is used through Fulcio
- **Fulcio:** Acts as the link between your build system and the Certificate Authority
- **Sigstore Stack:** The full stack, including Fulcio and Rekor, can be hosted on Kubernetes for a self-contained solution
- **Privacy Considerations:** When signing private artifacts, using public Fulcio and Rekor services may expose information about your signature

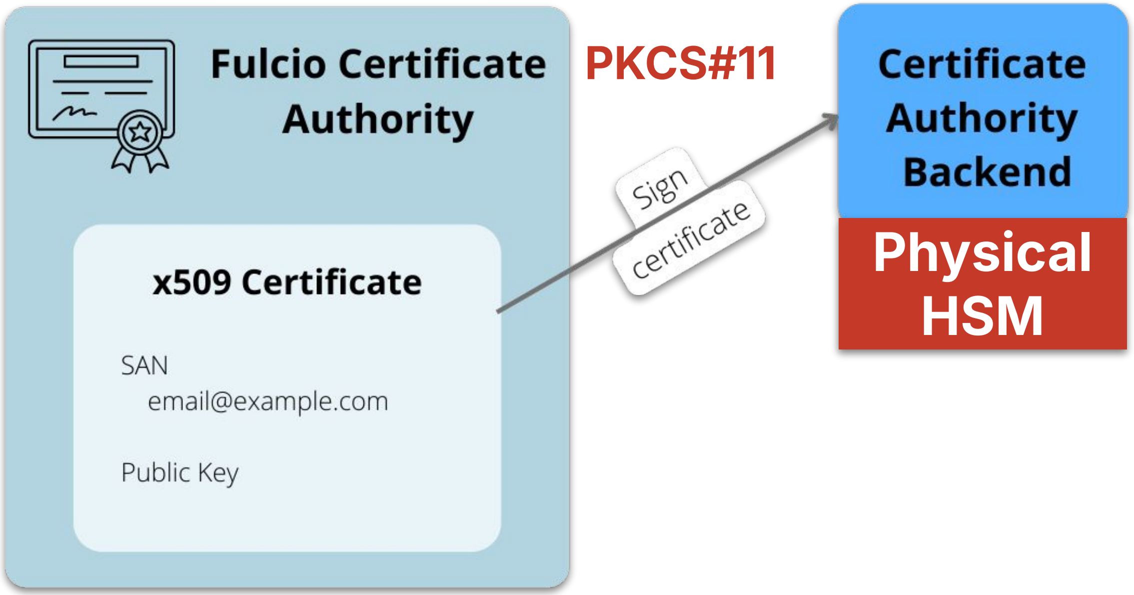


Diagram based from SigStore Docs

Questions?

- Do you have any **questions** or **remarks?**
- **Additionals resources:**
  - CNCF Tag Security Whitepaper: https://project.linuxfoundation.org/hubfs/CNCF_SSCP_v1.pdf