AISA

Securing Your Software Development Lifecycle

Supply chain Levels for Software Artifacts (SLSA)

Stefan Avgoustakis - GCP Security Lead AuNZ
Yazan Mughrabi - GCP Principal Architect AuNZ

AUSTRALIAN CYBER CONFERENCE 2022

Agenda

01
State of Secure Software Supply Chain

02
How to eat an elephant ?

03
Introducing SLSA

04
Practical application of SLSA

05
Demo

06
Q&A

2022 ACCELERATE

State of DevOps Report

DORA
DEVOPS RESEARCH & ASSESSMENT

01 **Adoption has already begun:** Software supply chain security practices embodied in SLSA and SSDF already see modest adoption, but there is ample room for more.

02 **Healthier cultures have a head start:** Organizational culture is a primary driver of software development security practices, with higher trust, "blameless" cultures are more likely to establish SLSA and SSDF practices than lower-trust organizational cultures.

03 **There's a key integration point:** Adoption of the technical aspects of software supply chain security appears to hinge on the use of CI/CD, which often provides the integration platform for many supply chain security practices.

04 **It provides unexpected benefits:** Besides a reduction in security risks, better security practices carry additional advantages, such as reduced burnout.

Increasingly, the software development lifecycle (SDLC) itself has become a vector for attacks.

SolarWinds, Kaseya, and **Codecov** hacks highlight vulnerable surface areas exposed in the SDLC.

650% - 920 > 12K

Sonatype

Surge in OSS supply chain attacks

84%

Venturebeat

Commercial code bases have OSS vulnerabilities

70%

Gartner

of Docker Hub images contain high- or medium severity vulnerabilities

Consider the number of things that are **managed as code**


Dev Team

.go

Application
(Containers, biz logic)


Platforms Team

.yaml

Platform
(Config, Service & policy mgt, CI/CD)


Cloud Team

.tf

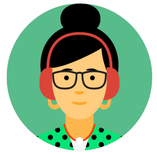
Infrastructure
(Cloud, Clusters, VPCs)

Governance
(D&R, Audit, Compliance)

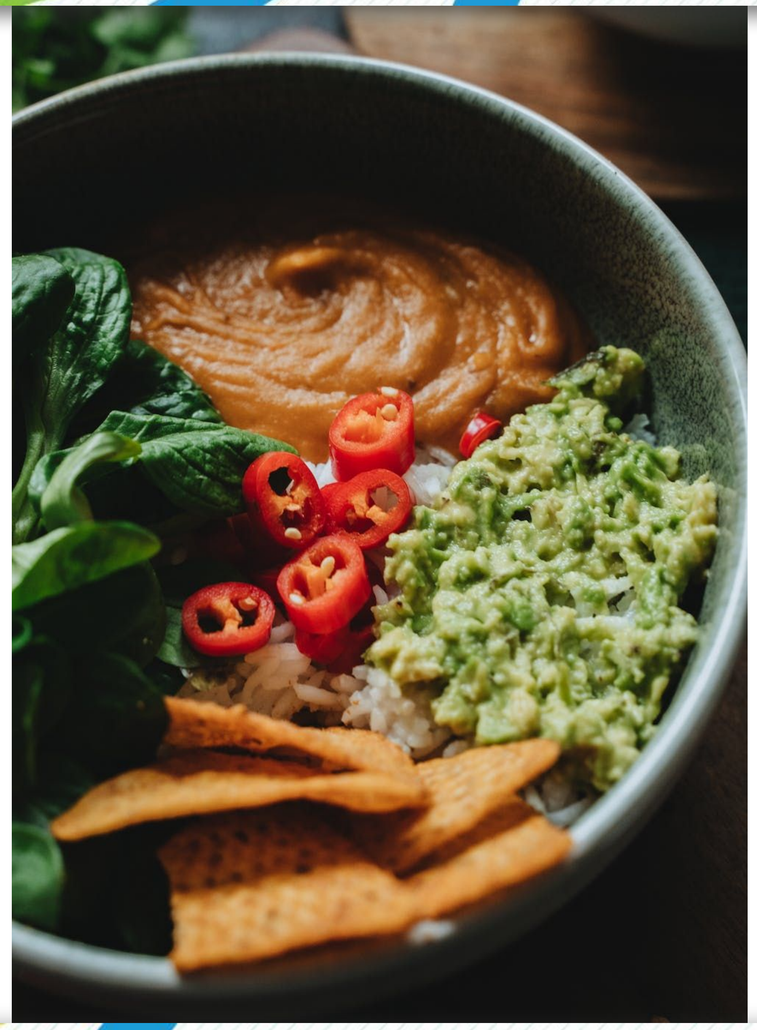
.rego
YARA


Security Team


Audit Team


Compliance Team

Strong supply chain security controls can have a very wide impact



SLSA

Safeguarding artifact integrity across any software supply chain

Know your ingredients

Scorecards

github.com/ossf/scorecard

Open Source Vuln DB

osv.dev

Allstar

github.com/ossf/allstar

Open Source Insights

deps.dev

It’s all about the base

Wolfi
Linux (un)distro

Google Distroless
images

Alpine Linux
Docker image

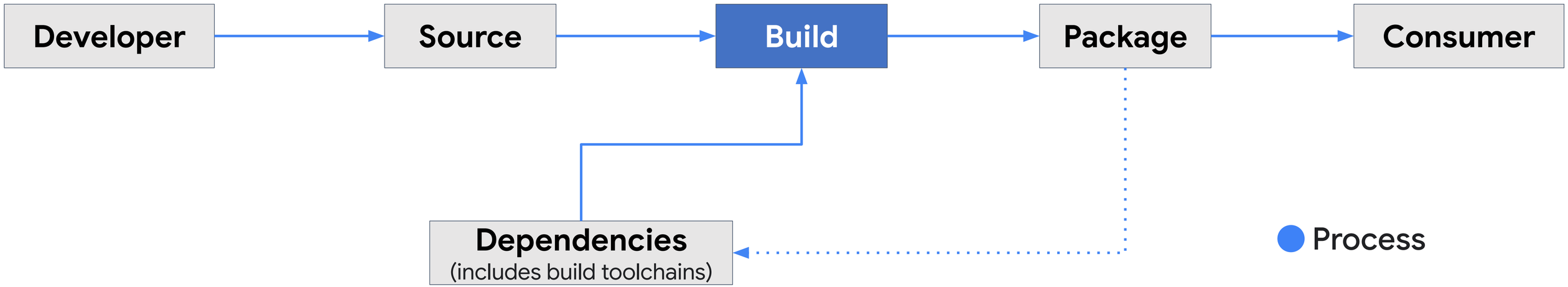
Prepackaged Software

SBOM

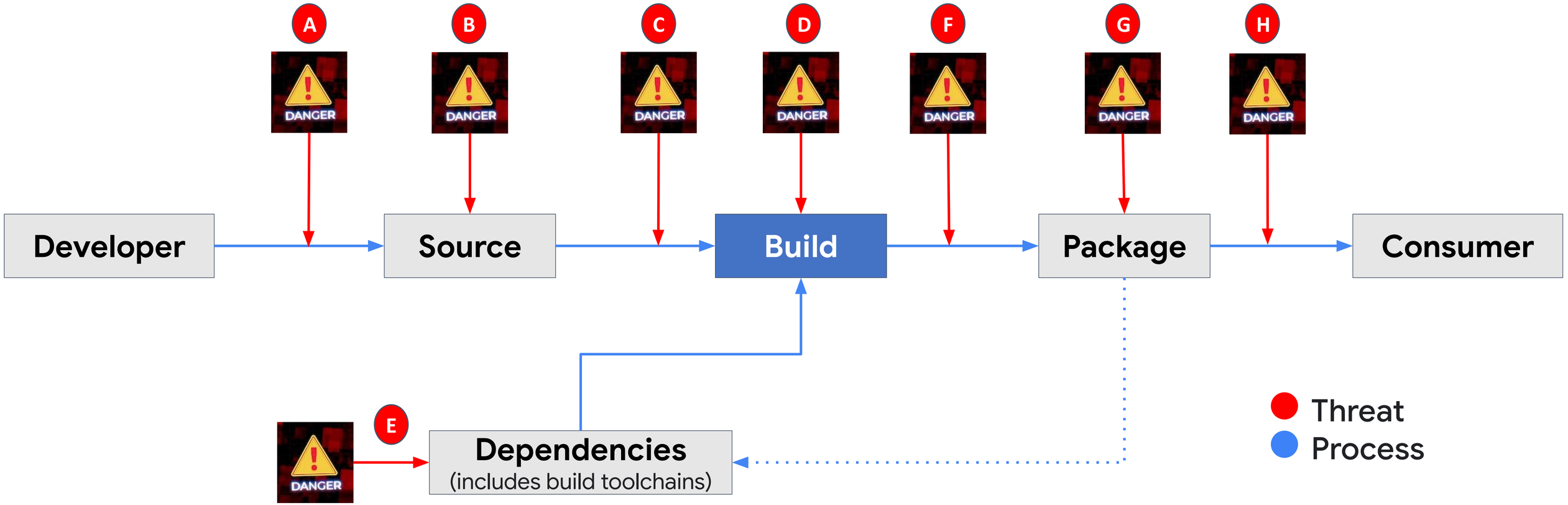
Asset Inventory

Baseline Security

Supply Chain



Supply Chain Threat Model



	Threat	Known example
A	Submit bad code to the source repository	**Linux hypocrite commits**: Researcher attempted to intentionally introduce vulnerabilities into the Linux kernel via patches on the mailing list.
B	Compromise source control platform	**PHP:** Attacker compromised PHP's self-hosted git server and injected two malicious commits.
C	Build with official process but from code not matching source control	**Webmin:** Attacker modified the build infrastructure to use source files not matching source control.
D	Compromise build platform	**SolarWinds:** Attacker compromised the build platform and installed an implant to inject malicious behavior during each build.
E	Use bad dependency (i.e., A-H, recursively)	**Event-stream:** Attacker added an innocuous dependency and then updated the dependency to add malicious behavior. The update did not match the code submitted to GitHub (i.e., attack F).
F	Upload an artifact that was not built by the CI/CD system	**CodeCov:** Attacker used leaked credentials to upload a malicious artifact to a GCS bucket, from which users download directly.
G	Compromise package repository	**Attacks on Package Mirrors:** Researcher ran mirrors for several popular package repositories, which could have been used to serve malicious packages.
H	Trick consumer into using bad package	**Browserify typosquatting:** Attacker uploaded a malicious package with a similar name as the original.

How to eat an elephant ?

NIST SSDF (SP800-218)

Set of practices that are meant to be implemented in the Software Development Lifecycle (SDLC), organized in four groups :

Prepare the Organization (PO)
Protect the Software (PS)
Produce Well-Secured Software (PW)
Respond to Vulnerabilities (RV)

NSA - CISA - ODNI Software Supply Chain Guidance

Provide guidance in line with industry best practices and principles which software developers are strongly encouraged to reference. These principles include :

Security Requirements planning
Designing secure software architecture
Adding security features
Maintaining the security of SW and the underlying infrastructure

Improving artifact integrity across the supply chain:

SLSA (”salsa”) is Supply-chain Levels for Software Artifacts.

Currently a set of ***incrementally adoptable security guidelines / framework*** from source to service, giving anyone working with software a common language for increasing levels of software security and supply chain integrity.

Goal is to support the ***automatic creation of auditable metadata that can be fed into policy engines*** to give “SLSA certification” to a particular package or build platform.

slsa.dev

SLSA Security Levels

Basic protection

Provenance checks to help evaluate risks and security

Medium protection

Further checks against the origin of the software

Advanced protection

Extra resistance to specific classes of threats

Maximum protection

Strict auditability and reliability checks

SLSA Requirements

Source

Version Controlled

Verified History

Retained Indefinitely

2-Person Reviewed

Build

Scripted Build

Build Service

Build as code

Ephemeral env

Isolated

Parameterless

Hermetic

Reproducible

Provenance

Available

Authenticated

Service generated

Non-falsifiable

Content requirements

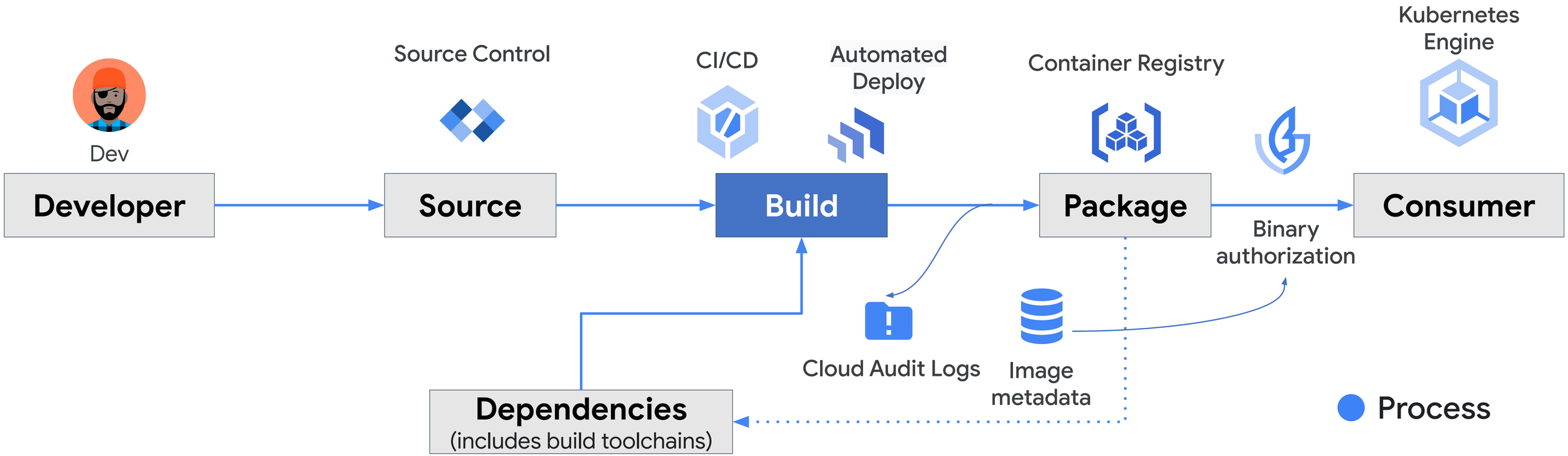
Common

Security baseline

Access control

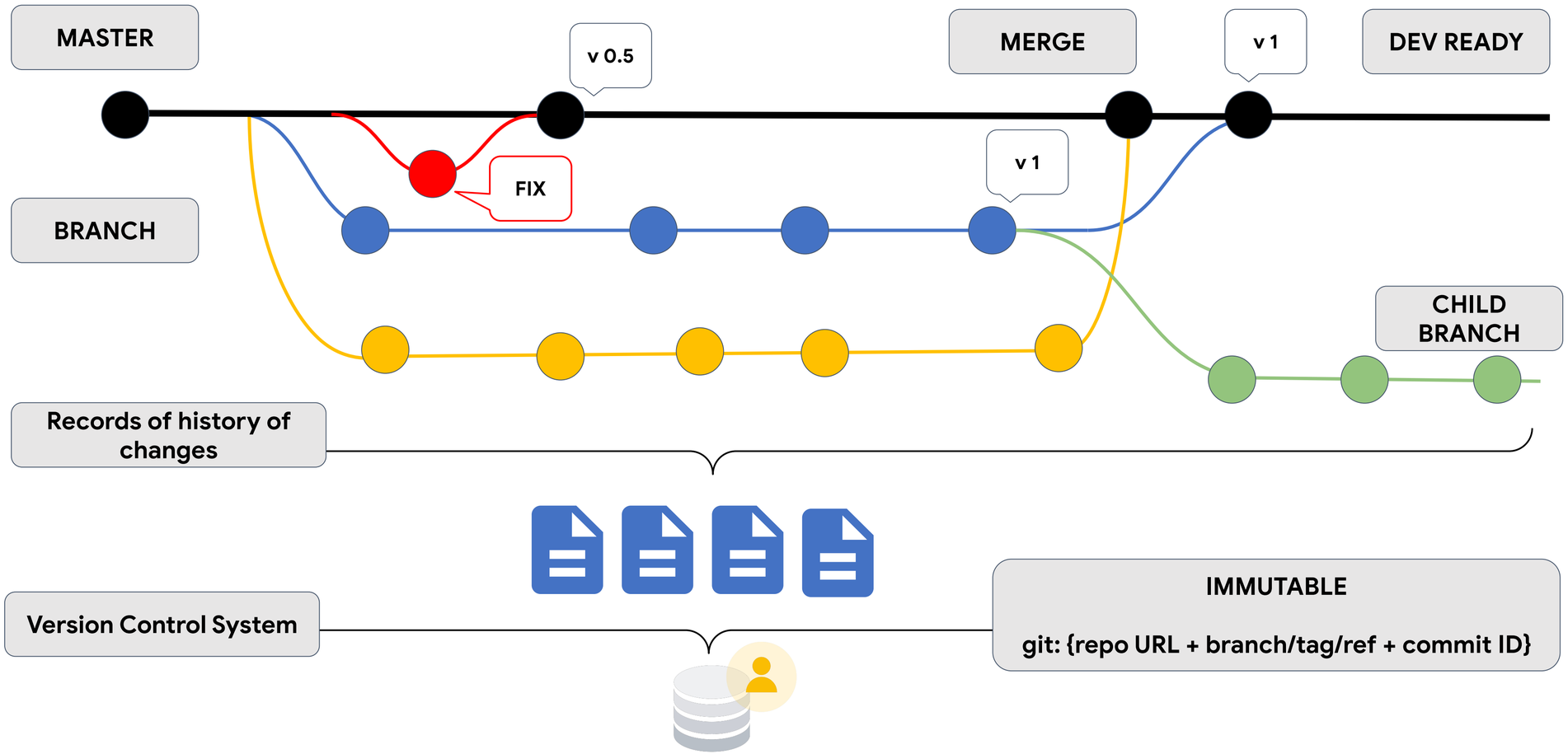
Superuser config

Supply Chain



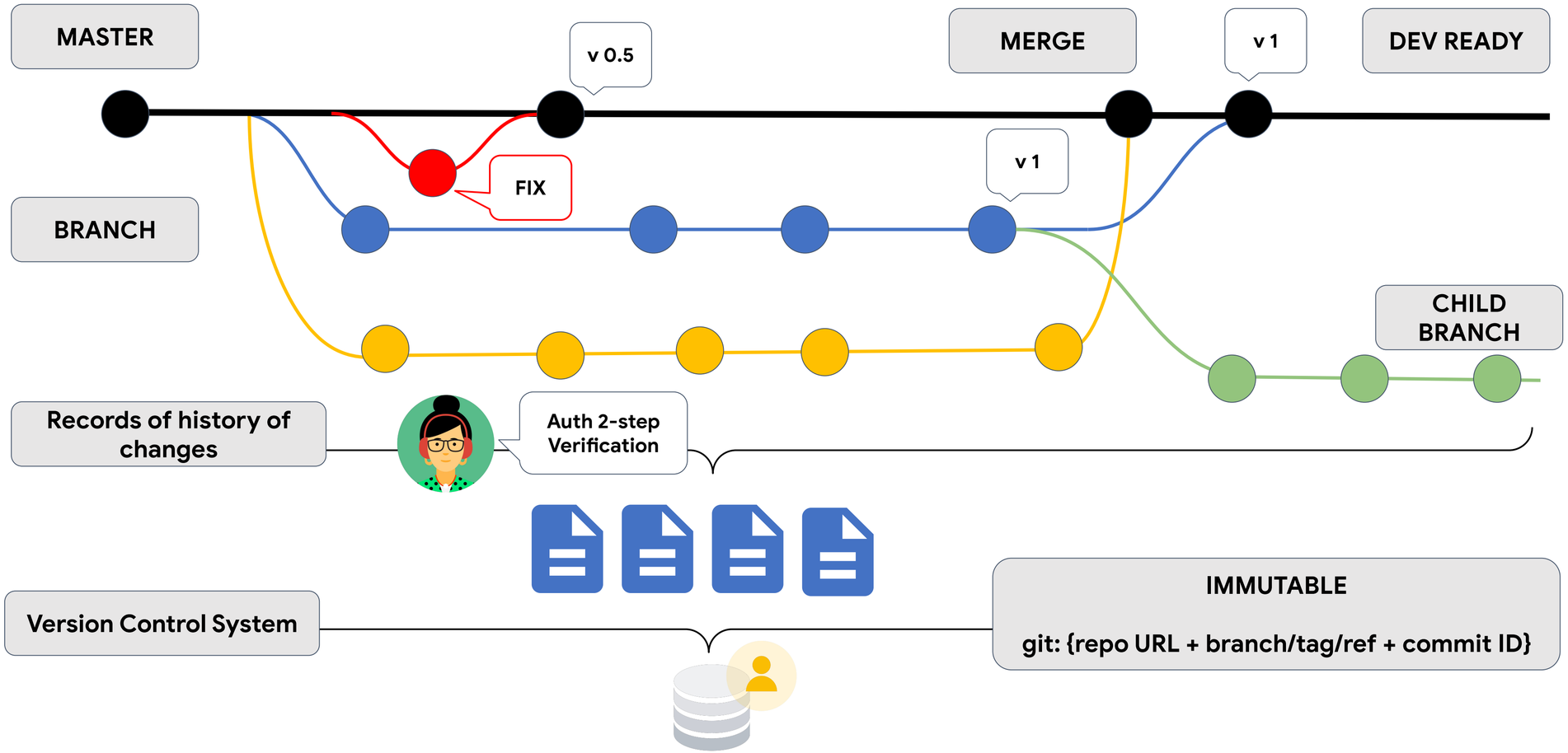
Source

1


MASTER
v 0.5
MERGE
v 1
DEV READY
FIX
v 1
BRANCH
CHILD BRANCH
Records of history of changes
Version Control System
IMMUTABLE
git: {repo URL + branch/tag/ref + commit ID}

Source

2


MASTER
v 0.5
MERGE
v 1
DEV READY
FIX
v 1
BRANCH
CHILD BRANCH
Records of history of changes
Auth 2-step Verification
Version Control System
IMMUTABLE
git: {repo URL + branch/tag/ref + commit ID}

Source

4

MASTER

v 0.5

MERGE

v 1

DEV READY

FIX

BRANCH

v 1

CHILD BRANCH

Records of history of changes

Auth 2-step Verification

Version Control System

IMMUTABLE

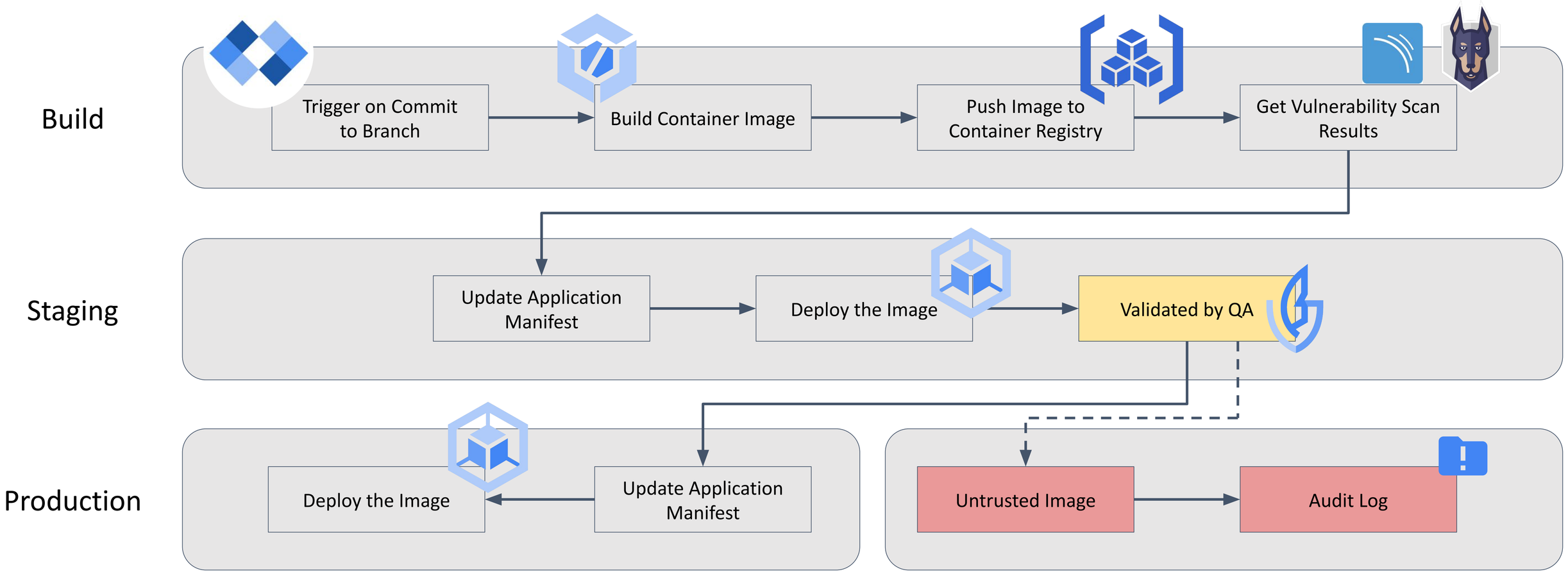
git: {repo URL + branch/tag/ref + commit ID}

3

Retained 18 months

Retained Indefinitely

Demo: Push Changes


Build
Trigger on Commit to Branch
Build Container Image
Push Image to Container Registry
Get Vulnerability Scan Results
Staging
Update Application Manifest
Deploy the Image
Validated by QA
Production
Deploy the Image
Update Application Manifest
Untrusted Image
Audit Log

Build

1

BUILD SCRIPT

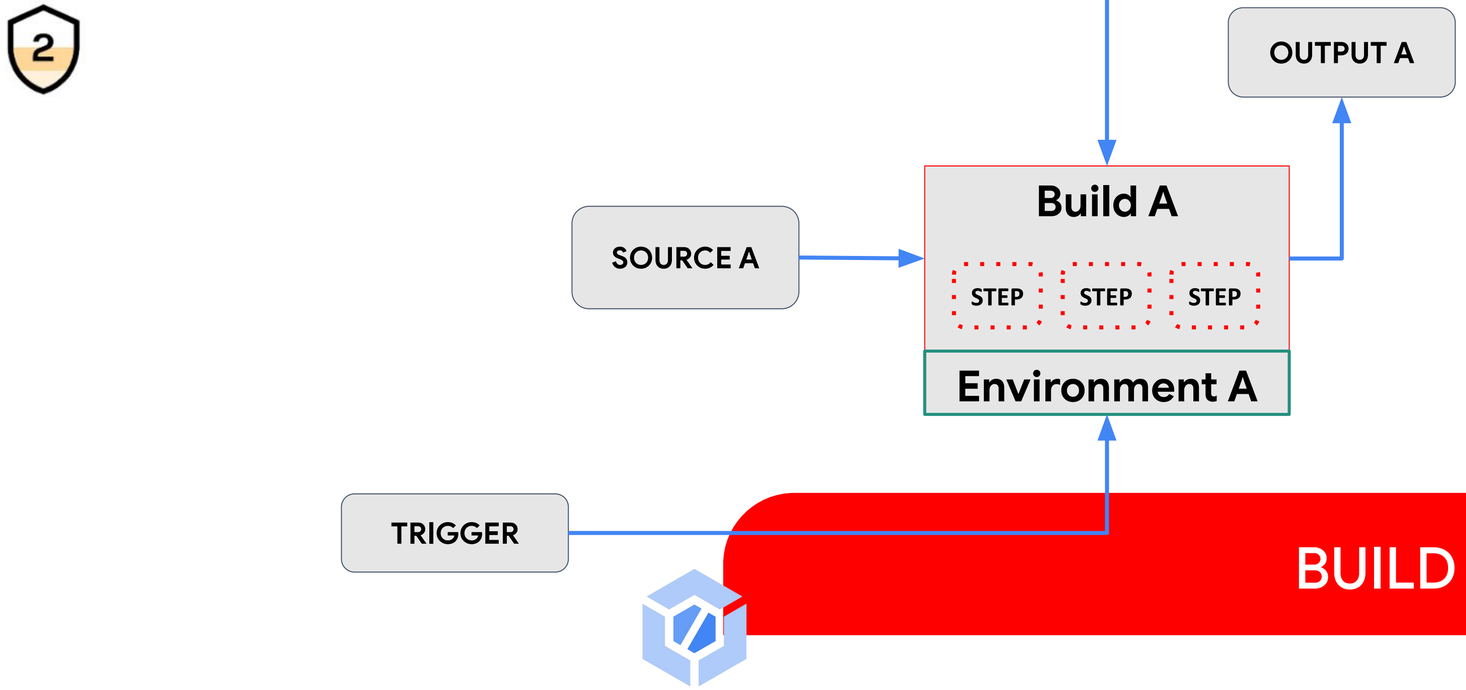
```
# syntax=docker/dockerfile:1
FROM node:12-alpine
RUN apk add --no-cache python3 g++ make
WORKDIR /app
COPY . .
RUN yarn install --production
CMD ["node", "src/index.js"]
```

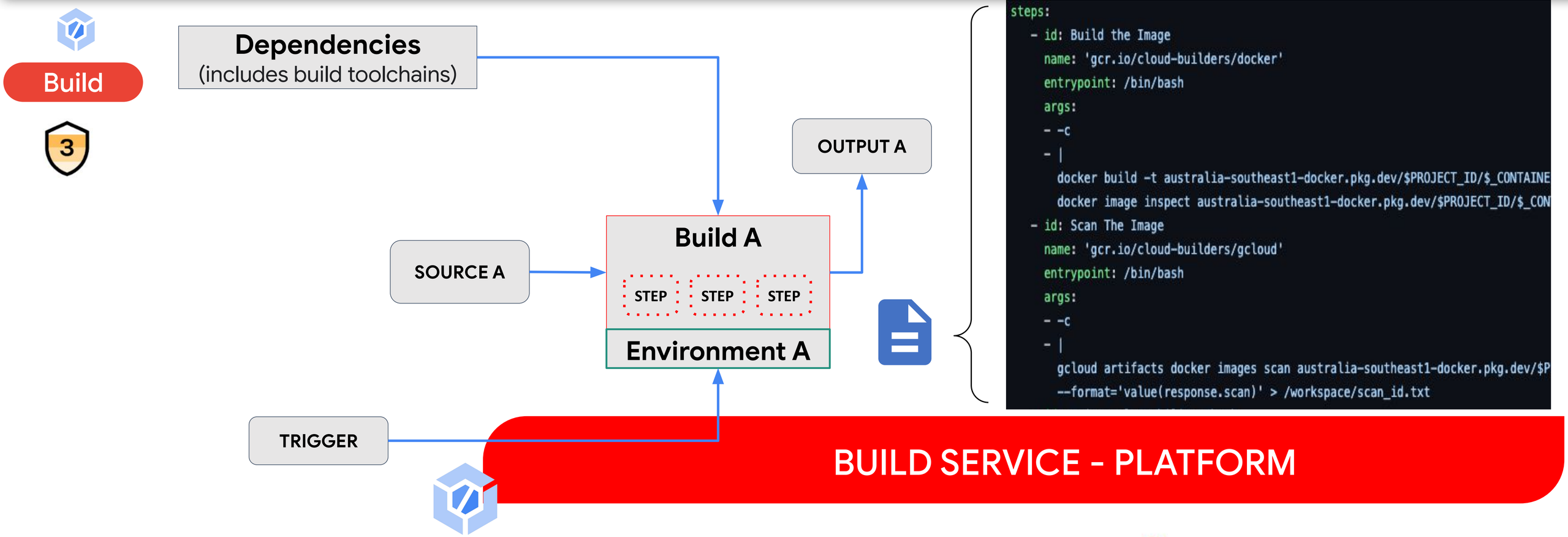
INVOKE BUILD SCRIPT

```
docker build -t my_container_image
```

LOCAL BUILD

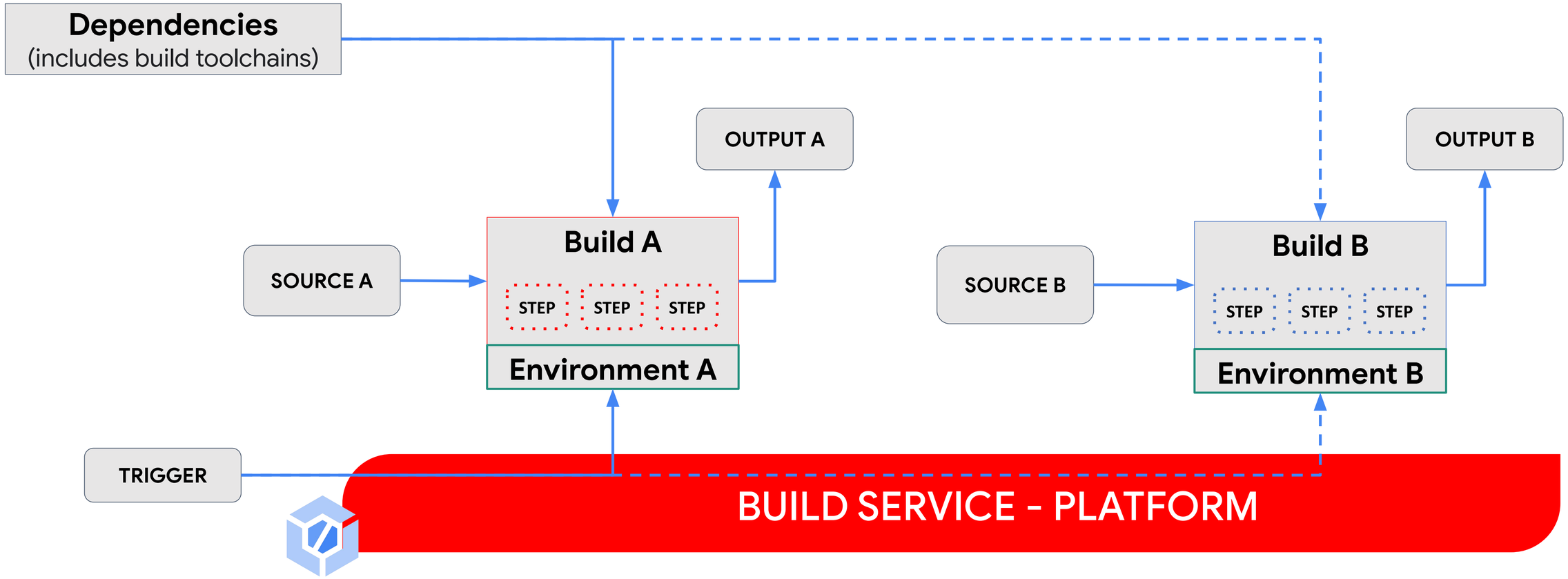
Build


Dependencies
(includes build toolchains)
OUTPUT A
SOURCE A
Build A
STEP
STEP
STEP
Environment A
TRIGGER
BUILD SERVICE - PLATFORM
TEKTON
Jenkins
GitHub Actions


Build
3
Dependencies
(includes build toolchains)
OUTPUT A
SOURCE A
Build A
STEP
STEP
STEP
Environment A
TRIGGER
steps:
- id: Build the Image
name: 'gcr.io/cloud-builders/docker'
entrypoint: /bin/bash
args:
- -c
- |
docker build -t australia-southeast1-docker.pkg.dev/$PROJECT_ID/$_CONTAINE
docker image inspect australia-southeast1-docker.pkg.dev/$PROJECT_ID/$_CON
- id: Scan The Image
name: 'gcr.io/cloud-builders/gcloud'
entrypoint: /bin/bash
args:
- -c
- |
gcloud artifacts docker images scan australia-southeast1-docker.pkg.dev/$P
--format='value(response.scan)' > /workspace/scan_id.txt
BUILD SERVICE - PLATFORM
TEKTON
Jenkins
GitHub Actions

Build

3

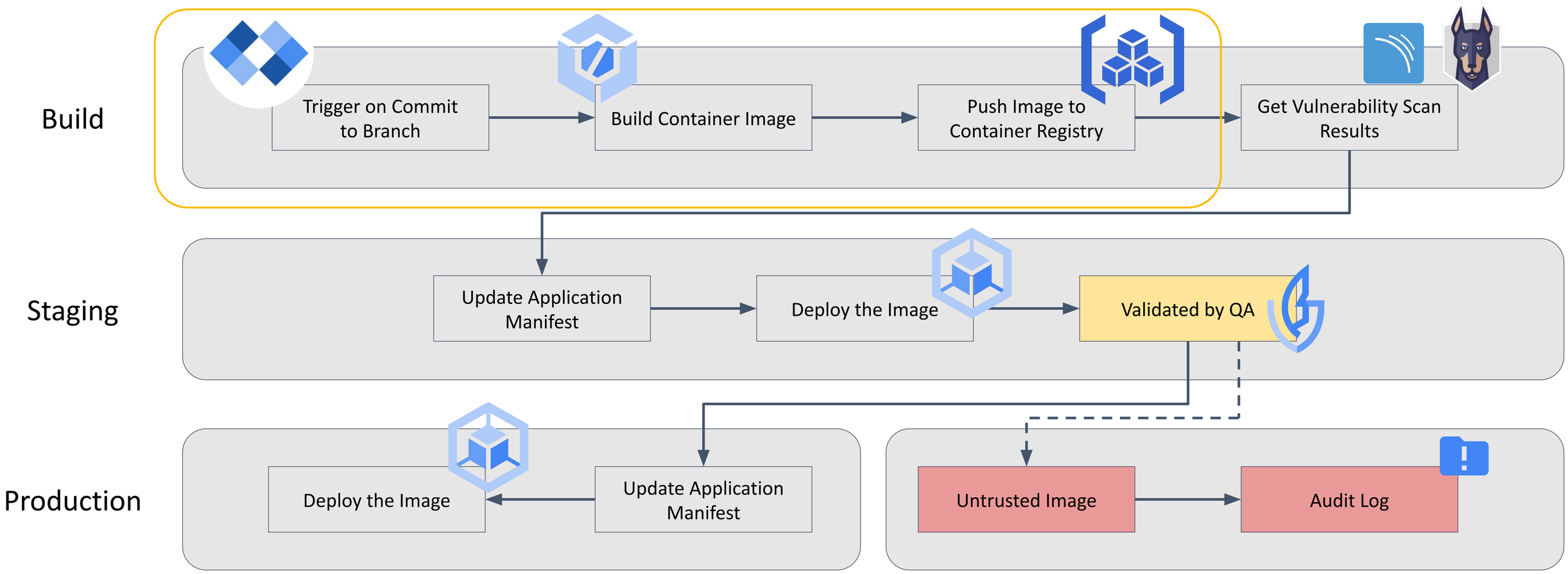


TEKTON

Jenkins

GitHub Actions

Demo: Build App


Build
Trigger on Commit to Branch
Build Container Image
Push Image to Container Registry
Get Vulnerability Scan Results
Staging
Update Application Manifest
Deploy the Image
Validated by QA
Production
Deploy the Image
Update Application Manifest
Untrusted Image
Audit Log

Provenance

Provenance

To ***trace software back to the source*** and define the moving parts in a complex supply chain, provenance needs to be there from the very beginning. It's the ***verifiable information about software artifacts describing where, when and how something was produced.***

The primary intended use case is to feed into automated policy engines, such as in-toto and Binary Authorization.

Provenance

1

ARTIFACT A

SOFTWARE ATTESTATION (snippet)

in-toto format

```
provenance_summary:
  provenance:
  - build:
      intotoStatement:
        _type: https://in-toto.io/Statement/v0.1
        predicateType: https://slsa.dev/provenance/v0.1
        slsaProvenance:
          builder:
            id: https://cloudbuild.googleapis.com/GoogleHostedWorker@v0.3
          materials:
          - digest:
              sha1: cbf58cbb59db47e3187768158d1b30154f7af1ee
            uri: https://github.com/savgoustakis/dev-sec-ops-demo
          metadata:
            buildFinishedOn: '2022-09-26T11:37:54.761900Z'
            buildInvocationId: 13d0bde6-ef64-4f89-9dc8-dbb3a2631e69
            buildStartedOn: '2022-09-26T11:33:14.469170320Z'
```

Provenance

2

ARTIFACT A

SOFTWARE ATTESTATION (snippet)

```
envelope:
  payload:
  eyJfdHlwZSI6Imh0dHBzOi8vaW4tdG90by5pby9TdGF0ZW1lbnQvdjAuMSIsInByZWRpY2F0ZSI6eyJidWlsZGVyIjp7ImlkIjoiaHR0
  cHM6Ly9jbG91ZGJ1aWxkLmdvb2dsZWFwaXMuY29tL0dvb2dsZUhvc3RlZFdvcmtlckB2MC4zIn0sIm1hdGVyaWFscyI6W3siZGlnZXN0
payloadType: application/vnd.in-toto+json
signatures:
- keyid: projects/verified-builder/locations/australia-southeast1/keyRings/attestor/cryptoKeys/
builtByGCB/cryptoKeyVersions/1
  sig: MEUCID5UJIkGMcCRiM2kMuA23rD44oDiFUcBXr_9keEJK9gDAiEA1ju4hrmIvkdCFEbpYqE9vQxUITcQ5qZEcvT2IA_Ip-w=
recipe:
  arguments:
    '@type': type.googleapis.com/google.devtools.cloudbuild.v1.Build
    id: 13d0bde6-ef64-4f89-9dc8-dbb3a2631e69
    name: projects/171810778018/locations/australia-southeast1/builds/
    13d0bde6-ef64-4f89-9dc8-dbb3a2631e69
```

Digital Signature

Service Generated

Provenance

3

ARTIFACT A

SOFTWARE ATTESTATION (snippet)

```
envelope:
  payload:
  eyJfdHlwZSI6Imh0dHBzOi8vaW4tdG90by5pby9TdGF0ZW1lbnQvdjAuMSIsInByZWRpY2F0ZSI6eyJidWlsZGVyIjp7ImlkIjoiaHR0
  cHM6Ly9jbG91ZGJ1aWxkLmdvb2dsZWFwaXMuY29tL0dvb2dsZUhvc3RlZFdvcmtlckB2MC4zIn0sIm1hdGVyaWFscyI6W3siZGlnZXN0
payloadType: application/vnd.in-toto+json
s
-
b
  sig: MEUCID5UJIKGMCCKIMZKMUAZ3rD446DIFUCBXr_9KeEJK9gDAIEAIju4nrmIvKdCrEbpYqE9VQXUITCQ5qZECvIZIA_Ip-w=
recipe:
  a
```

Digital Signature must be stored in a secure management system (HSM)

Trusted control plane - no user alterations

Provenance

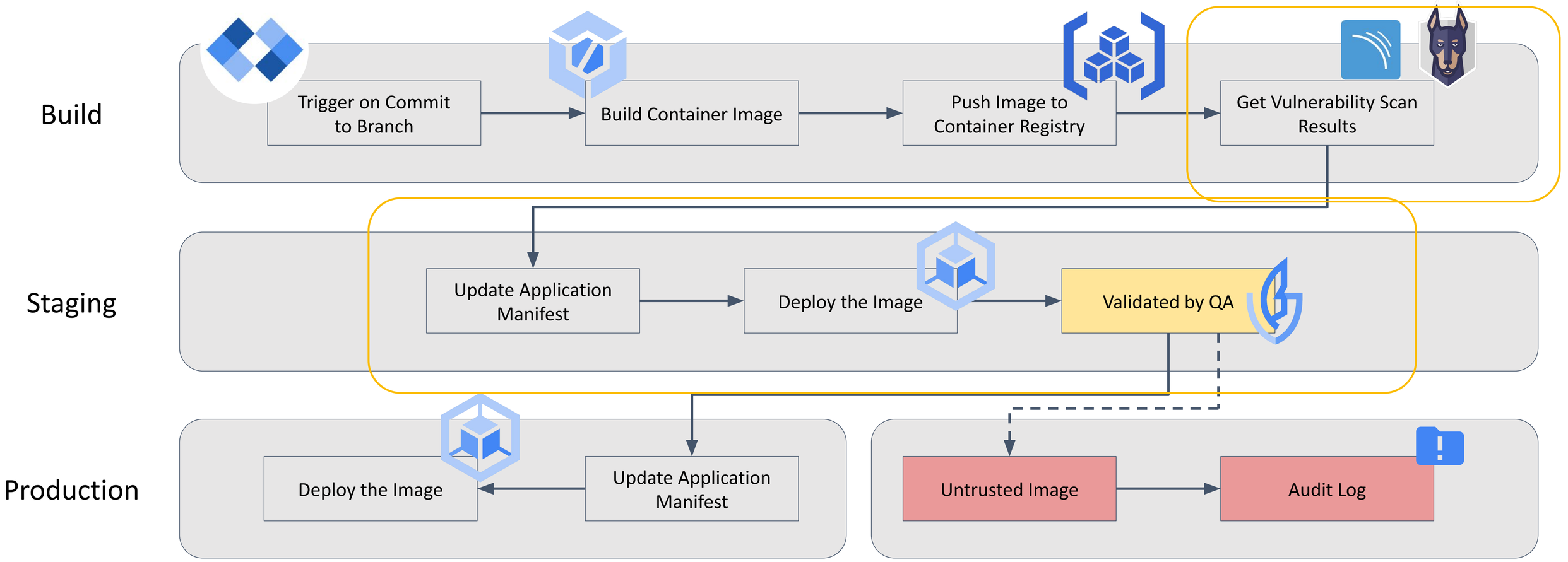
4

ARTIFACT A

```
sourceProvenance: {}
steps:
- args:
  - -c
  - |
    docker build -t australia-southeast1-docker.pkg.dev/slsa-workshop/slsa-repo/
    cd:cbf58cbb59db47e3187768158d1b30154f7af1ee -f ./Dockerfile . &&
    docker image inspect australia-southeast1-docker.pkg.dev/slsa-workshop/slsa-repo/
    cd:cbf58cbb59db47e3187768158d1b30154f7af1ee
  entrypoint: /bin/bash
  id: Build the Image
  name: gcr.io/cloud-builders/docker
  pullTiming:
    endTime: '2022-09-26T11:33:19.454265092Z'
    startTime: '2022-09-26T11:33:19.450955160Z'
  status: SUCCESS
  timing:
    endTime: '2022-09-26T11:34:03.047992261Z'
    startTime: '2022-09-26T11:33:19.450955160Z'
- args:
  - -c
  - |
    gcloud artifacts docker images scan australia-southeast1-docker.pkg.dev/slsa-workshop/
    slsa-repo/cd:cbf58cbb59db47e3187768158d1b30154f7af1ee \
    --format='value(response.scan)' > /workspace/scan_id.txt
  entrypoint: /bin/bash
  id: Scan The Image
  name: gcr.io/cloud-builders/gcloud
  pullTiming:
    endTime: '2022-09-26T11:34:03.050882703Z'
    startTime: '2022-09-26T11:34:03.048084719Z'
  status: SUCCESS
  timing:
    endTime: '2022-09-26T11:34:27.394476495Z'
    startTime: '2022-09-26T11:34:03.048084719Z'
```

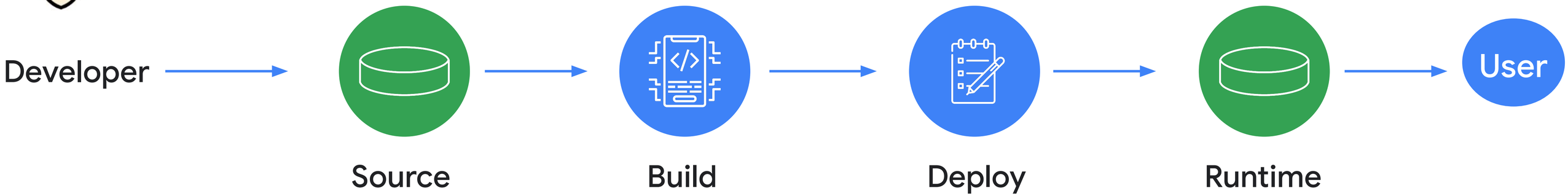
Dependencies complete

Demo: Deploy App


Build
Trigger on Commit to Branch
Build Container Image
Push Image to Container Registry
Get Vulnerability Scan Results
Staging
Update Application Manifest
Deploy the Image
Validated by QA
Production
Deploy the Image
Update Application Manifest
Untrusted Image
Audit Log

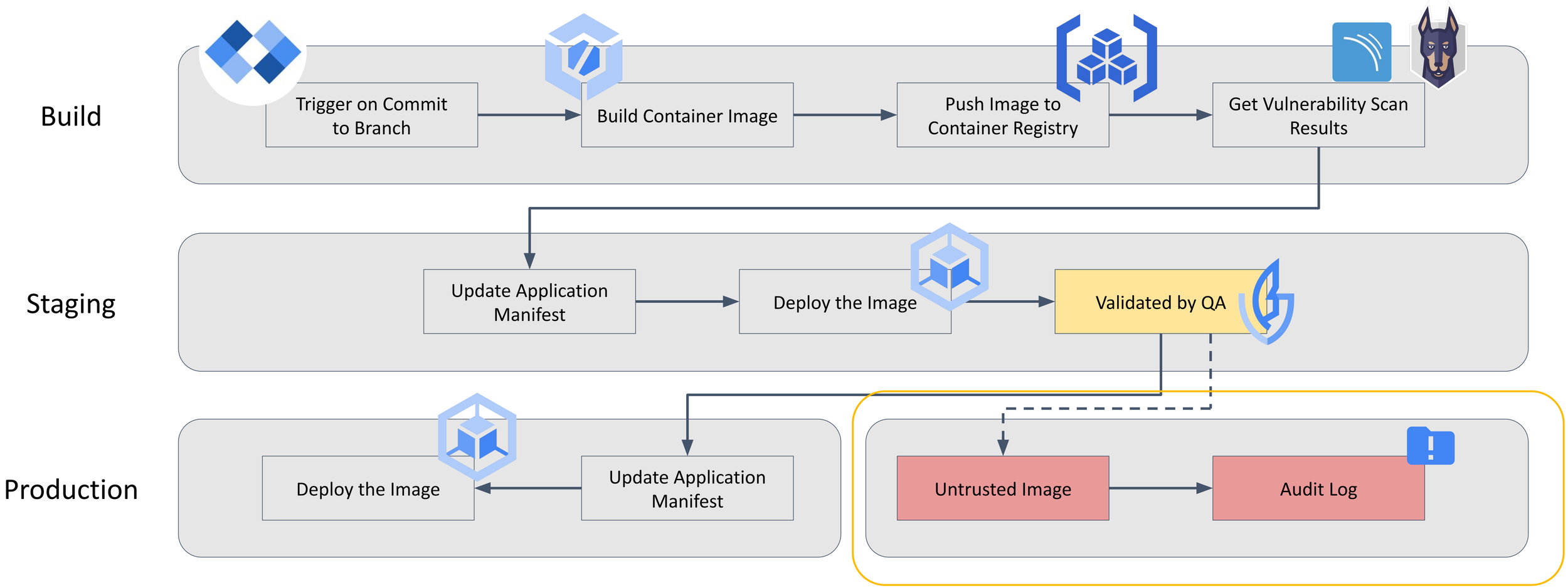
Common

4



Break Glass Process	Vulnerability Scanning and Patching
Rare Remote Access	Secure Boot / Machine Identity
Baseline Security Standard	User Isolation / Transport Security

Demo: Rouge Employee


Build
Trigger on Commit to Branch
Build Container Image
Push Image to Container Registry
Get Vulnerability Scan Results
Staging
Update Application Manifest
Deploy the Image
Validated by QA
Production
Deploy the Image
Update Application Manifest
Untrusted Image
Audit Log

Google
Cybersecurity Action
Team
gcat.google.com

Thank You!

SLSA Site:
slsa.dev
Blog:
https://security.googleblog.com/2021/06/introducing-slsa-end-to-end-framework.html

Source:
https://cloud.google.com/architecture/binary-auth-with-cloud-build-and-gke
https://github.com/yukozuna/slsa

Contact:
- Stefan: savgoust@google.com
- Yazan: mughrabi@google.com

Twitter:
- Stefan: @savgoust

Linkedin:
- Stefan: linkedin.com/in/stefanavgoustakis/
- Yazan: linkedin.com/in/mughrabi

AISA

Thank You

AUSTRALIAN CYBER CONFERENCE 2022