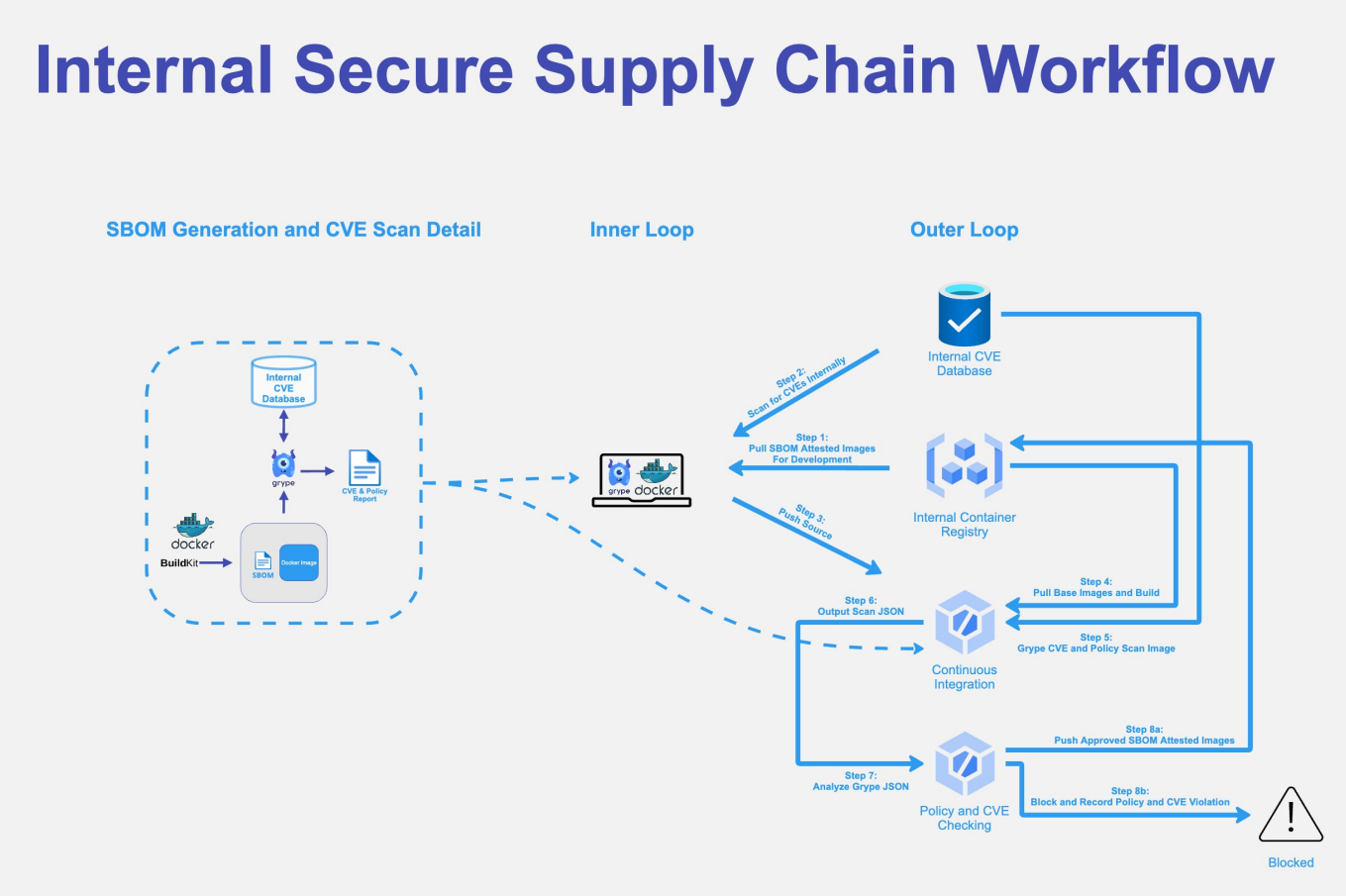
Using Buildx to generate SBOM and CVE scanning

This walkthrough is intended for anyone unable to leverage SaaS products for SBOM generation and CVE detection. For everyone else we highly recommend our more comprehensive solution, Docker Scout.

Overview

SBOM generation, CVE detection, and Policy enforcement are core components of Supply Chain Security. This overview will show you how to incorporate Docker Desktop's buildx command to leverage BuildKit SBOM generation and how to consume a generated SBOM, or image with an SBOM, with open-source toolings like grype to detect CVEs and policy violations.



Generating an SBOM

Generating an SBOM at Container Build Time

The following command will build the Dockerfile in the current directory and create an out directory with a SPDX based JSON file representing your SBOM. It will also generate an attestation that proves the provenance of the image.

```
docker buildx build --sbom=true --output type=local,dest=out .
```

Once you've verified your SBOM output locally, you can build, attest, generate an SBOM, and push it to your registry with the following command.

```
docker buildx build --tag <namespace>/<image>:<version> --attest type=sbom
--push .
```

Generating an SBOM from an Image

If you need to generate an SBOM from an image that has already been built, you can do so with the following command.

```
docker buildx imagetools inspect <namespace>/<image>:<version> --format "
{{ json .SBOM.SPDX }}"
```

Scanning Images for CVEs

You can scan almost any Docker image, but the scan will be even more effective if you build your image with an attestation and SBOM, as shown in Generating an SBOM at Container Build Time.

Scan an image with the default grype CVE providers.

```
docker run -it anchore/grype:latest "<namespace>/<image>:<version>" "-o
json"
```

Creating a Custom Grype Database for Offline and Proprietary CVE Database Use

- Step 1: Create a custom vunnel provider by following the steps outlined in the Vunnel Example Provider README.md
- Step 2: Create a .vunnel.yaml with any overrides

For example (visit https://github.com/anchore/vunnel for more details):

```
# .vunnel.yaml
root: ./processed-data

log:
  level: trace

providers:
  <your_provider>:
    request_timeout: 125
    runtime:
      existing_input: keep
      existing_results: delete-before-write
```

```
      on_error:
        action: fail
        input: keep
        results: keep
        retry_count: 3
        retry_delay: 10
```

- Step 3: Run vunnel against your newly created provider

```
docker run --rm -it -v $(pwd)/data:/data -v
$(pwd)/.vunnel.yaml:/.vunnel.yaml ghcr.io/anchore/vunnel:latest run <your
provider>
```

- Step 4: Create a grype-db config.yaml file
- Step 5: Run grype-db against your vunnel data output and config

```
docker run --rm -it -v $(pwd):/config -w /config ubuntu:latest bash -c
"apt-get -y update;apt-get -y install curl;curl -sSfL
https://raw.githubusercontent.com/anchore/grype-db/main/install.sh | sh -s
-- -b /usr/local/bin; pwd; ls -lh; grype-db -v build""
```

- Step 6: Set the grype `GRYPE_DB_CACHE_DIR` to your grype-db output directory

For example:

```
export GRYPE_DB_CACHE_DIR=/data
```

Citations and Helpful Information

- Build Attestations and SBOMs
- BuildKit and buildx
- Grype
- Grype-db
- SPDX
- Vunnel