Security Analysis

Alessandro Bocci

name.surname@unipi.it

Advanced Software Engineering (Lab)

14/11/2025

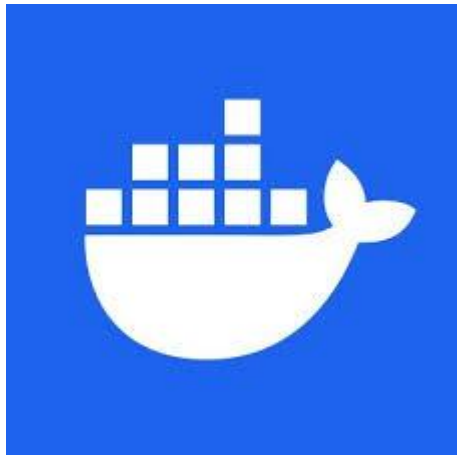
What will you do?

You will perform analysis on:

- The codebase.
- The codebase dependencies.
- The Docker images.

Software Prerequisites

- Bandit (`pip install bandit`).
- pip-audit (`pip install pip-audit`).
- Docker account.
- GitHub account.
- trivy

Secure coding practices

- Complexity of source code leads to more security vulnerabilities.
- Exponential increase of defects as number of lines of code increases.
- Functional and security testing is utterly important.

- Two types of testing and anlyses:
  - Dynamic
  - Static

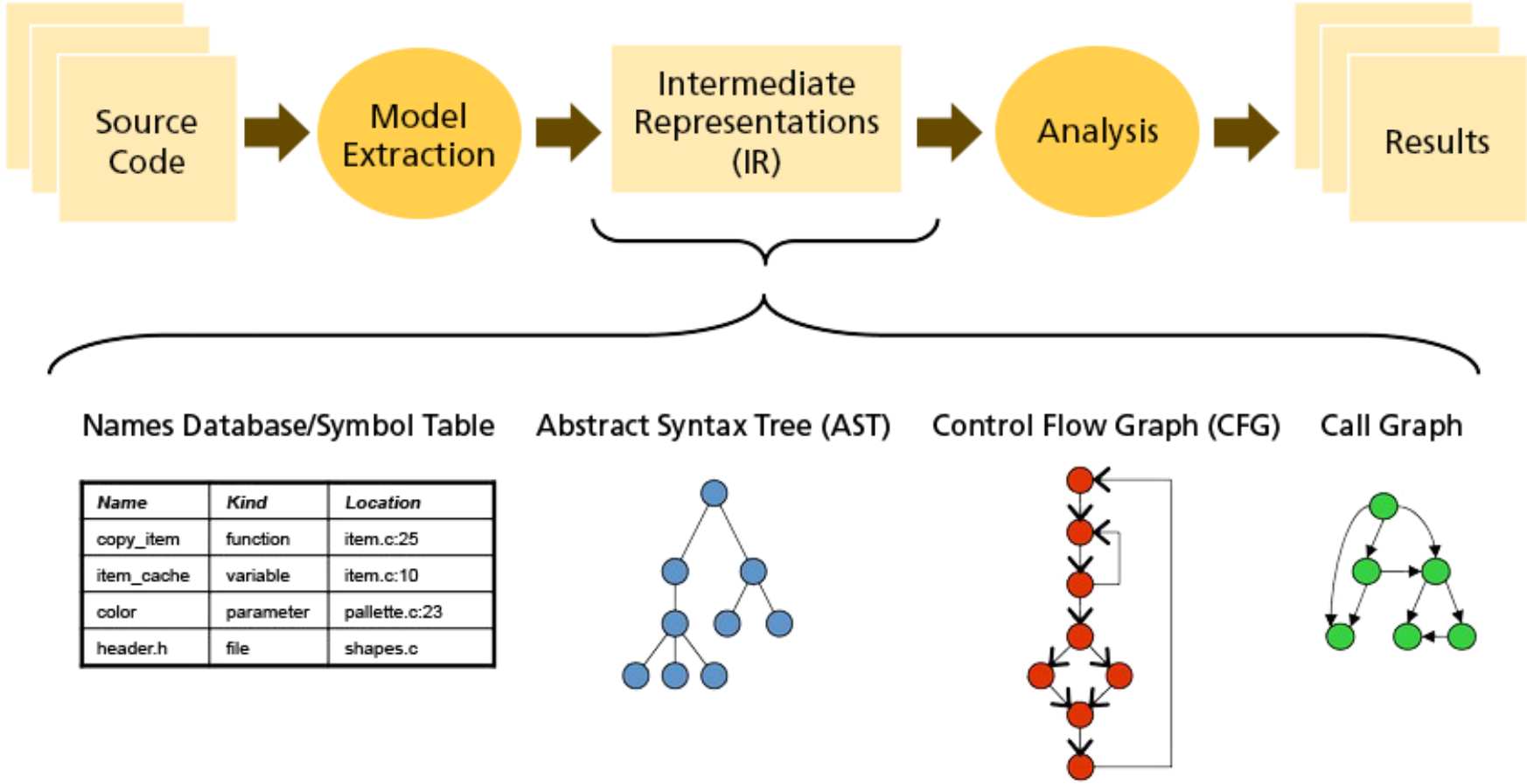
Dynamic Analysis

Dynamic Analysis involves running applications in a controlled environment to observe their behavior and interactions in real-time.

- Identifies runtime vulnerabilities like memory leaks, data exposure, and authentication issues.
- Essential for uncovering zero-day vulnerabilities and unforeseen behaviors due to external factors.

Unfortunately, there are few tools for our purposes that are free and easy to use.

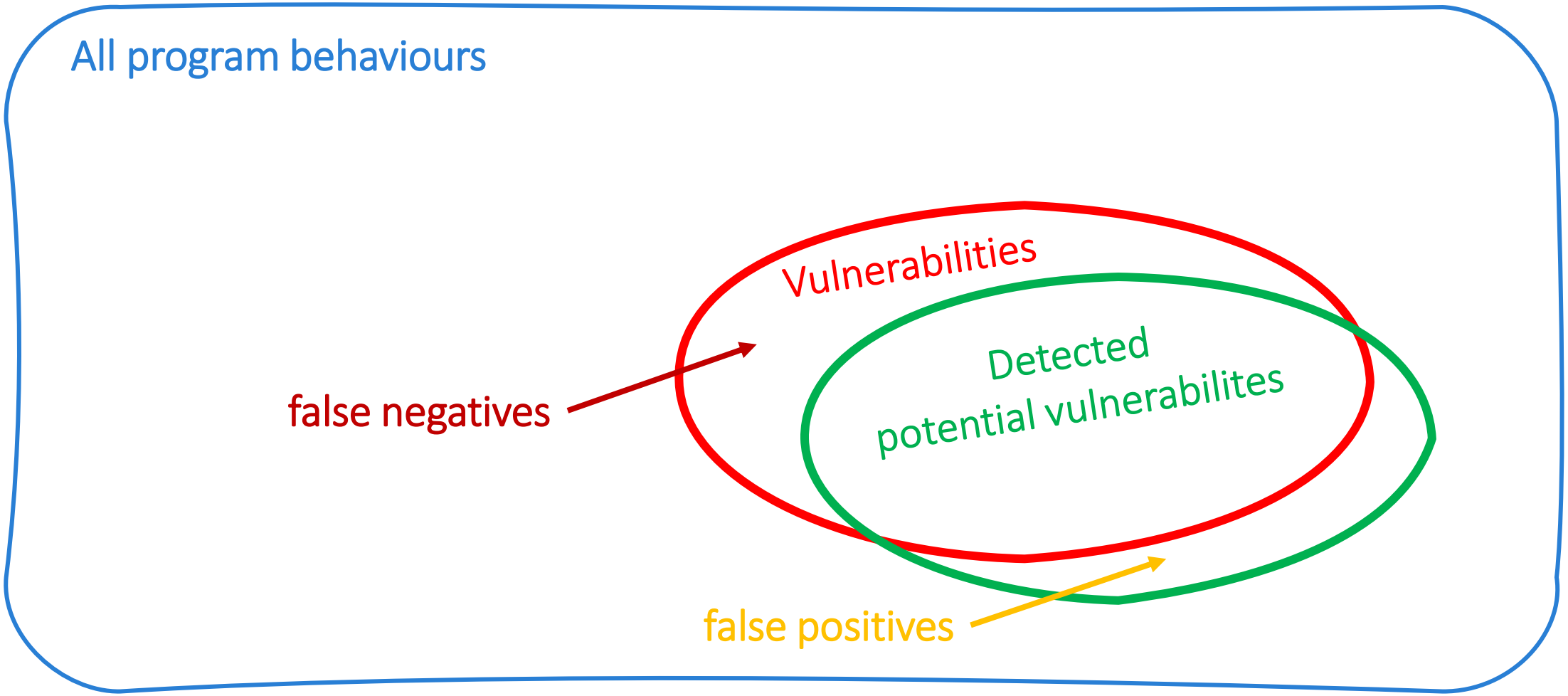
Vulnerability Avoidance with Static Analysis



Name	*Kind*	*Location*
copy_item	function	item.c:25
item_cache	variable	item.c:10
color	parameter	pallette.c:23
header.h	file	shapes.c

Static Analysis = Analyze the system (source code or its representation) to check some property without running it.

A note on Static Analysis



In this lab…

We focus on static analysis:

- Source code analysis with Bandit.
- Dependency analysis with pip-audit and Dependabot.
- Docker image analysis with Docker scout and trivy.

Note: Bandit and pip-audit are for Python only.

Source Code: Bandit

- Bandit is a static analysis tool designed to find common security issues in Python code, by exploiting known patterns (plugins).
- Bandit was originally developed within the OpenStack Security Project and later re-homed to PyCQA.
- It recognizes 70 vulnerabilities out-of-the-box.

Docs @ https://bandit.readthedocs.io/en/latest/


```
pip install bandit
```

```
bandit -r <path to code>
```

Example – B324: hashlib, use of weak MD5

```
>> Issue: [B324:hashlib] Use of weak MD5 hash for security. Consider usedforsecurity=False
   Severity: High   Confidence: High
   CWE: CWE-327 (https://cwe.mitre.org/data/definitions/327.html)
   More Info: https://bandit.readthedocs.io/en/1.7.5/plugins/b324_hashlib.html
   Location: ASE/lab6/test.py:2:9
1       import hashlib
2       result = hashlib.md5(b'ASE ASE ASE')
3       print("The byte equivalent of hash is : ", end ="")
```

Example – B324: hashlib, use of weak MD5

```
>> Issue: [B324:hashlib] Use of weak MD5 hash for security. Consider usedforsecurity=False
   Severity: High   Confidence: High
   CWE: CWE-327 (https://cwe.mitre.org/data/definitions/327.html)
   More Info: https://bandit.readthedocs.io/en/1.7.5/plugins/b324_hashlib.html
   Location: ASE/lab6/test.py:2:9
1          import hashlib
2          result = hashlib.md5(b'ASE ASE ASE')
3          print("The byte equivalent of hash is : ", end ="")
```

Issue -> Use of MD5 (cryptographic hash function).

Example – B324: hashlib, use of weak MD5

```
>> Issue: [B324:hashlib] Use of weak MD5 hash for security. Consider usedforsecurity=False
   Severity: High   Confidence: High
   CWE: CWE-327 (https://cwe.mitre.org/data/definitions/327.html)
   More Info: https://bandit.readthedocs.io/en/1.7.5/plugins/b324_hashlib.html
   Location: ASE/lab6/test.py:2:9
1          import hashlib
2          result = hashlib.md5(b'ASE ASE ASE')
3          print("The byte equivalent of hash is : ", end ="")
```

Severity of the Issue (how much the Issue is ‘dangerous’):

- Bandit classifies issues in Low, Medium and High.
- It gives a level of confidence (how much Bandit is confident about it) for every issue, also Low, Medium and High.

Example – B324: hashlib, use of weak MD5

```
>> Issue: [B324:hashlib] Use of weak MD5 hash for security. Consider usedforsecurity=False
   Severity: High   Confidence: High
   CWE: CWE-327 (https://cwe.mitre.org/data/definitions/327.html)
   More Info: https://bandit.readthedocs.io/en/1.7.5/plugins/b324_hashlib.html
   Location: ASE/lab6/test.py:2:9
1          import hashlib
2          result = hashlib.md5(b'ASE ASE ASE')
3          print("The byte equivalent of hash is : ", end ="")
```

Definition of *why* it is an issue.

You can use it for understand how to resolve it.

Example – B324: hashlib, use of weak MD5

```
>> Issue: [B324:hashlib] Use of weak MD5 hash for security. Consider usedforsecurity=False
   Severity: High   Confidence: High
   CWE: CWE-327 (https://cwe.mitre.org/data/definitions/327.html)
   More Info: https://bandit.readthedocs.io/en/1.7.5/plugins/b324_hashlib.html
   Location: ASE/lab6/test.py:2:9
1          import hashlib
2          result = hashlib.md5(b'ASE ASE ASE')
3          print("The byte equivalent of hash is : ", end ="")
```

Bandit’s info about the issue.

Example – B324: hashlib, use of weak MD5

```
>> Issue: [B324:hashlib] Use of weak MD5 hash for security. Consider usedforsecurity=False
   Severity: High   Confidence: High
   CWE: CWE-327 (https://cwe.mitre.org/data/definitions/327.html)
   More Info: https://bandit.readthedocs.io/en/1.7.5/plugins/b324_hashlib.html
   Location: ASE/lab6/test.py:2:9
1          import hashlib
2          result = hashlib.md5(b'ASE ASE ASE')
3          print("The byte equivalent of hash is : ", end ="")
```

Location of the issue. In this example:

- The path for the file having the issue is `ASE/lab6/`
- The file with the issue is `test.py`

Example – B324: hashlib, use of weak MD5

```
>> Issue: [B324:hashlib] Use of weak MD5 hash for security. Consider usedforsecurity=False
   Severity: High   Confidence: High
   CWE: CWE-327 (https://cwe.mitre.org/data/definitions/327.html)
   More Info: https://bandit.readthedocs.io/en/1.7.5/plugins/b324_hashlib.html
   Location: ASE/lab6/test.py:2:9
1          import hashlib
2          result = hashlib.md5(b'ASE ASE ASE')
3          print("The byte equivalent of hash is : ", end ="")
```

Location of the issue. In this example:

- The path for the file having the issue is `ASE/lab6/`
- The file with the issue is `test.py`
- The line of the issue is 2 (and column 9).

Example – B324: hashlib, use of weak MD5

How to resolve it?

Change the hash function with a secure one.

From:

```
result = hashlib.md5(b'ASE ASE ASE’)
```

To:

```
result = hashlib.sha256(b'ASE ASE ASE')
```

Example – B324: hashlib, use of weak MD5

Remember:

- Do not change the behaviour of the code!

  (in the example the hash of the string is done but with SHA256).

- Do not follow Bandit's suggestions blindly!

  (in the example the suggestion was to put `usedforsecurity=False`, ask yourself when it is correct).

Dependencies: pip-audit

- pip-audit is a tool for scanning Python environments for packages with known vulnerabilities.
- It uses the Python Packaging Advisory Database via the PyPI JSON API as a source of vulnerability reports.
- It scan all the packages installed with pip for vulnerabilities.
- Be careful, it can break backward compatibility.
- It has an official GitHub Action.

Install	`pip install pip-audit`
Run	`pip-audit`
Fix vulnerabilities	`pip-audit --fix`

pip-audit

It scans the pip environment of the system where is executed.

We install Python packages inside the Docker containers.

There are several ways to use it:

- Include the command in the build of the image.
- Build an image that execute it before building the other images.
- ….

Choose what you think is fair for your purposes.

Dependencies: GitHub Dependabot

Dependabot is a tool integrated in GitHub to check vulnerabilities in repository dependecies.

- Go in the Settings of your repo.
- Advanced Security section.
- Enable **Dependabot security updates** and **Grouped security updates.**
- Go in the Security tab or your repo and look if Dependabot find something.
- Probably you are ok, our code use relatively recent things.
- If you want to see Dependabot at work, play with its Demo.

Docker images: Docker scout

Docker tool to scan images for vulnerabilities.

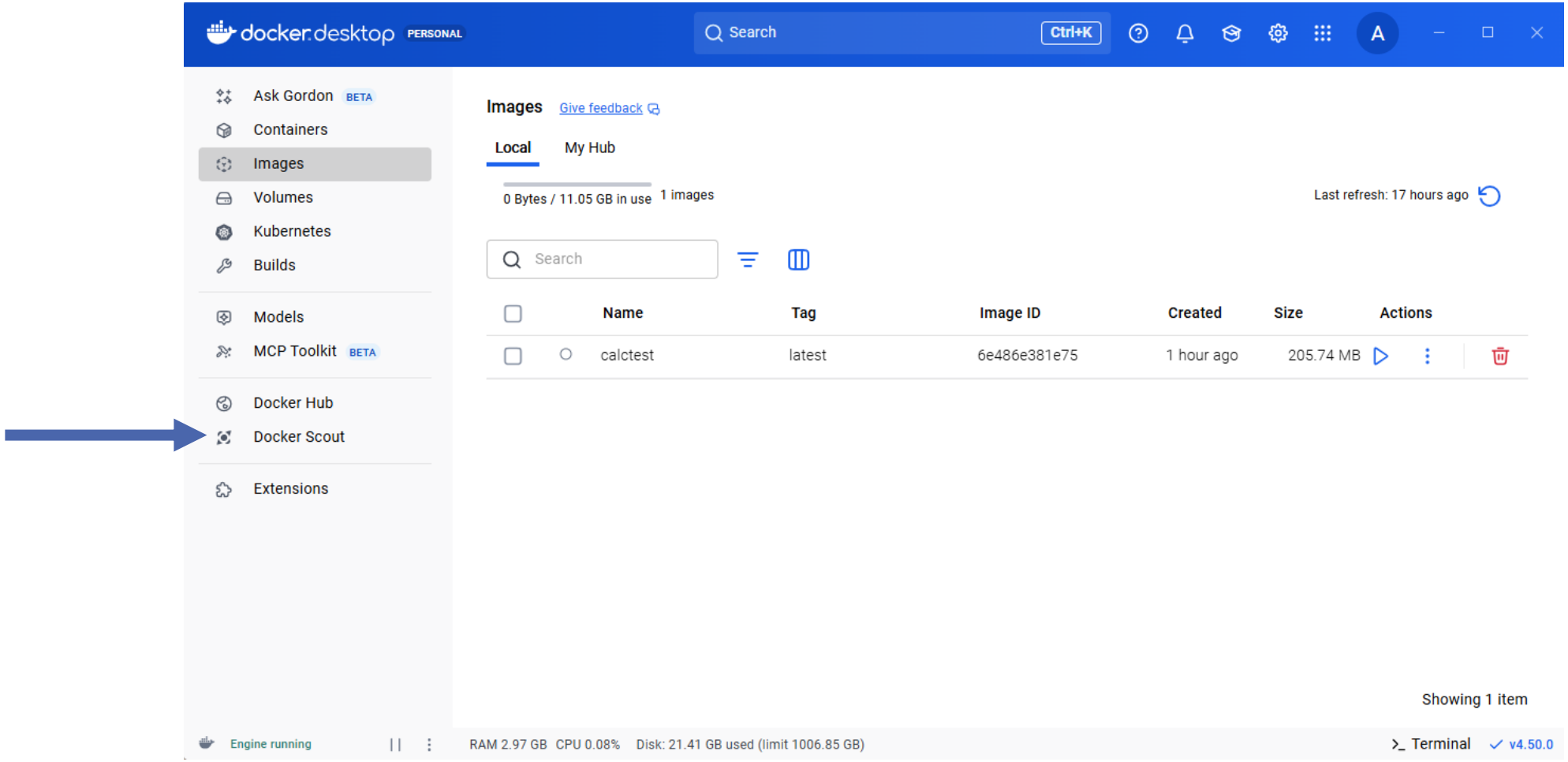
You need a Docker account for use it easily.

- Local images
- Remote images

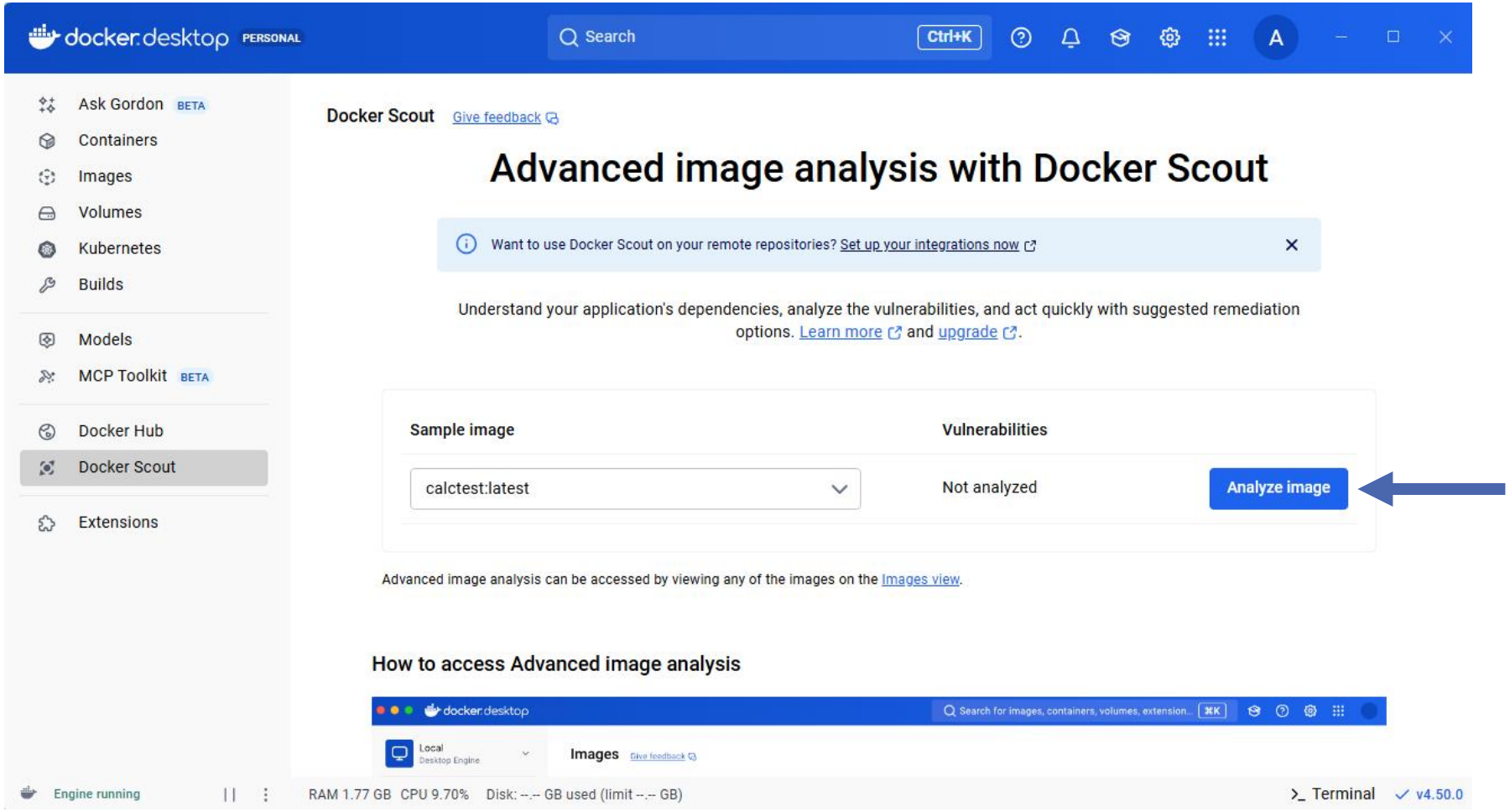
Docs @ https://docs.docker.com/scout/

We focus on the local ones.

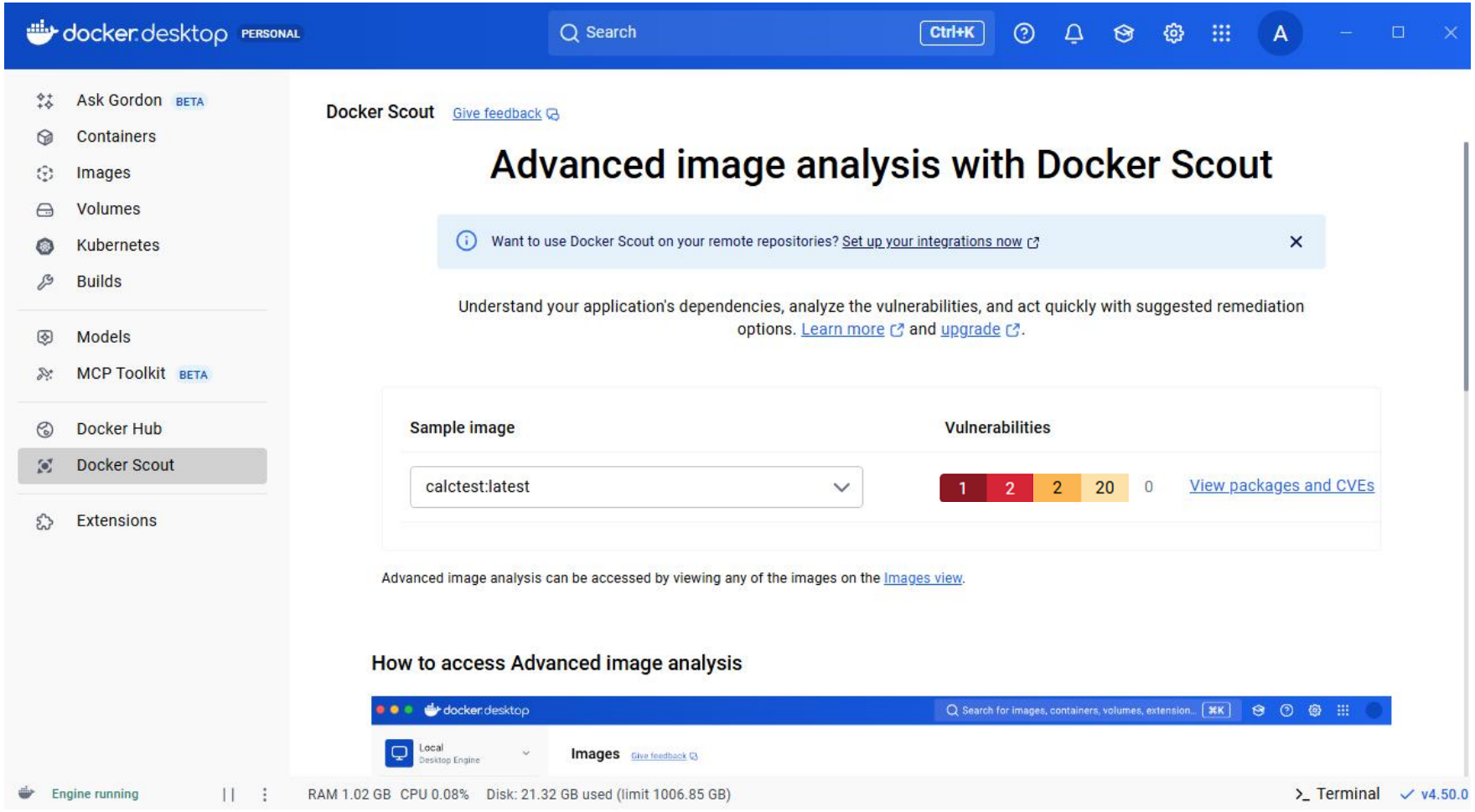
Docker scout in Docker Desktop



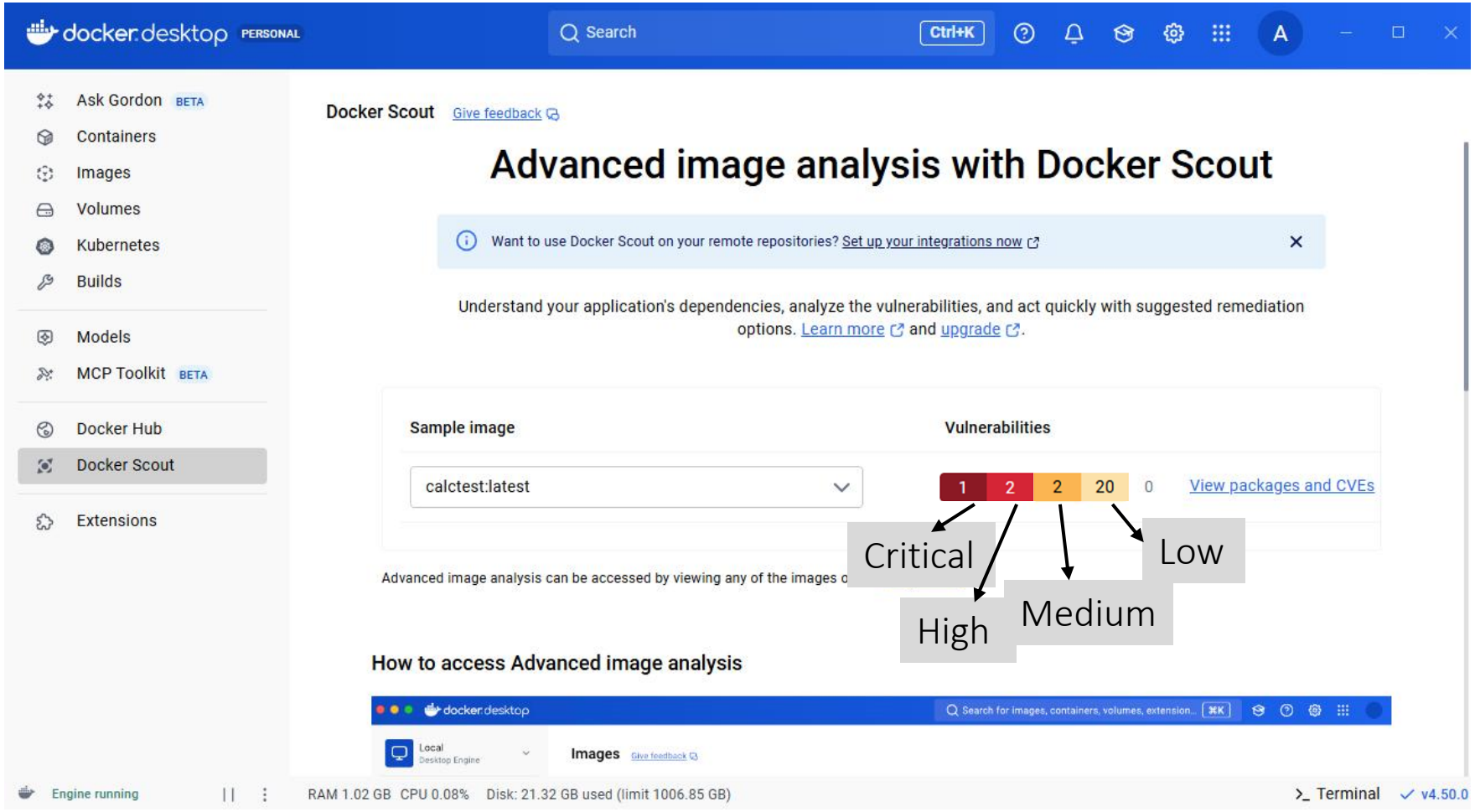
Docker scout in Docker Desktop



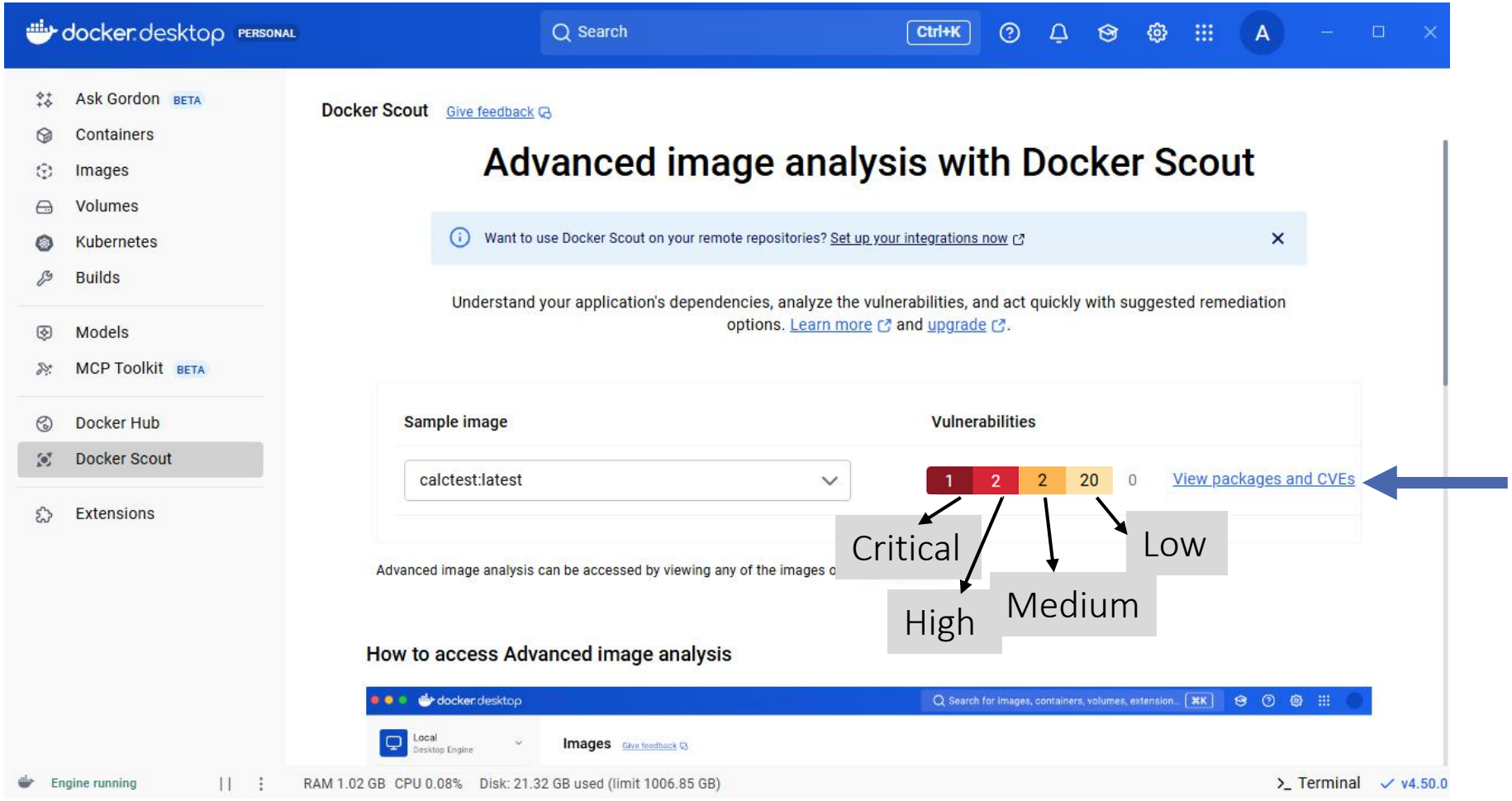
Docker scout in Docker Desktop



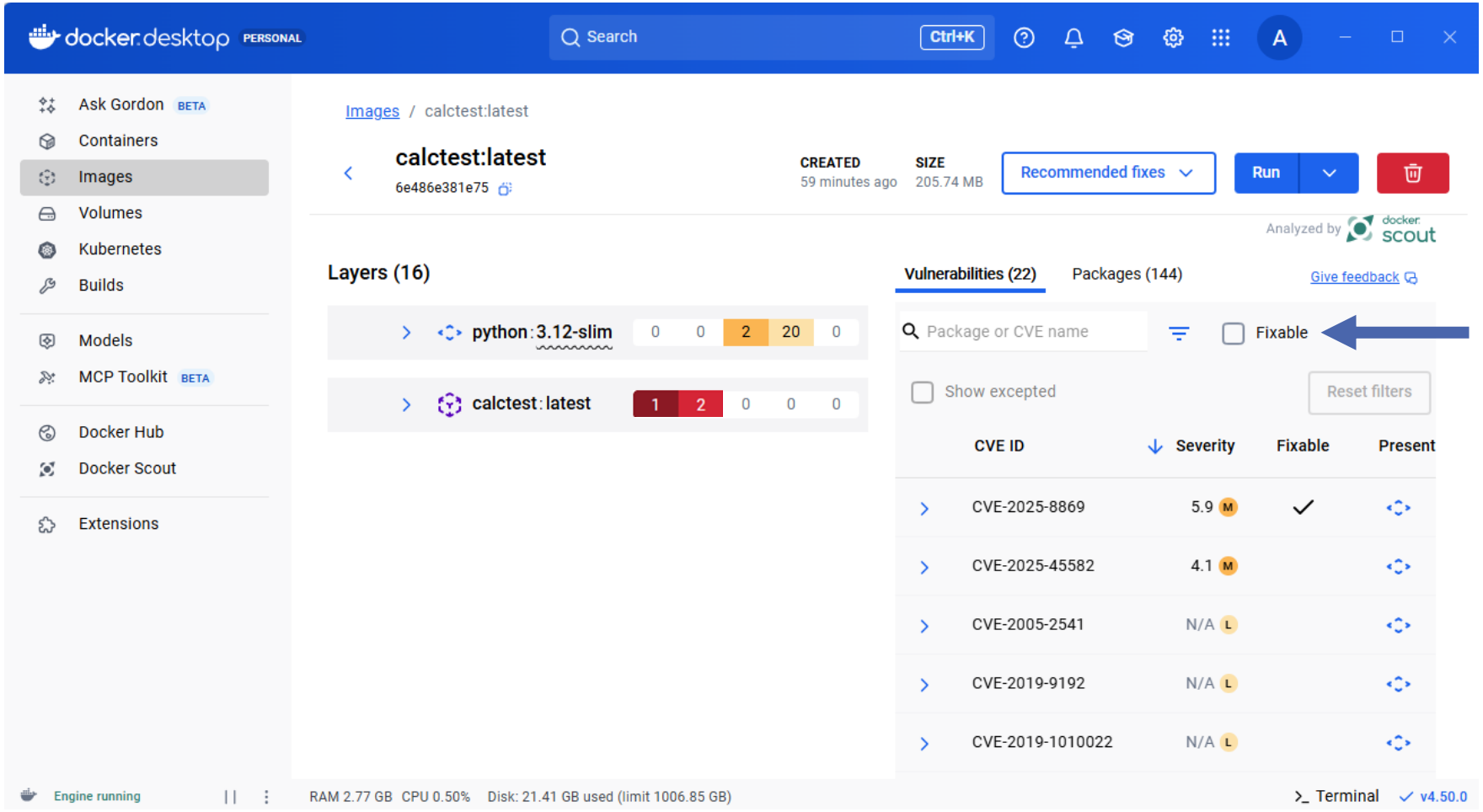
Docker scout in Docker Desktop



Docker scout in Docker Desktop

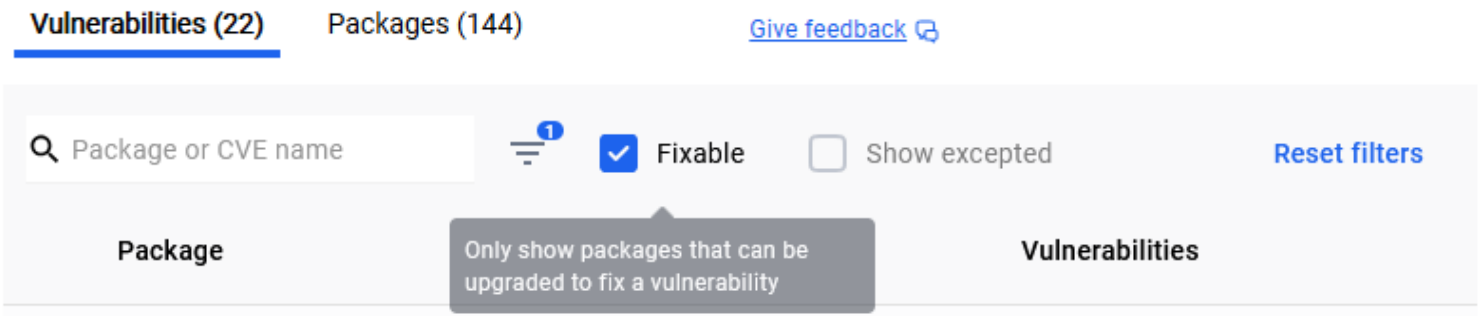


Docker scout in Docker Desktop



Docker scout in Docker Desktop

- Resolve any (fixable) critical and high vulnerability.



- Some vulnerabilities are not fixable in this moment, you can filter by fixable vulnerabilities in the details.
- Most of the time you need only to update base images (e.g. from python:3.12-slim to another one)
- Google and AI can help you ☺
- After every fix you have to build the image again.

Docker scout in CLI

- Login in Docker Hub and insert the password when prompted:

```
docker login –u <username>
```

- Scan the image with:

```
docker scout cves <image-name>
```

- You can use filters on the vulnerabilities:

```
docker scout cves --only-severity critical,high
docker scout cves --only-fixed
```

Docker images: Trivy

Trivy is an all-in-one, open-source security scanner for cloud-native apps. It can scan:

- Container images.
- Local filesystems.
- Git repositories.
- Kubernetes clusters.
- …

Docs @ https://trivy.dev/docs/latest/

Trivy

Our focus is on container (Docker) images.

Trivy supports two targets for container images.

- Files inside container images.
- Container image metadata.

Looking for vulnerabilities, misconfigurations, secrets, licenses.

Trivy

Scan the image with:

```
trivy image <image-name>
```

By default, it scans for vulnerabilities and secrets.

You can specify what scan with the option

```
 --scanners
```

Adding `vulns`, `misconfig`, or `license`. Secrets are always there.

Trivy

Secrets are not Docker secrets.

Are rules to find if are exposed:

- AWS access key
- GCP service account
- GitHub personal access token
- GitLab personal access token
- Slack access token
- etc.

Today’s Lab

Use `microase` code or the project to familiarise with the tools.

LAB TODO

- Run Bandit to check codebase vulnerabilities.
- Run pip-audit to check vulnerabilities in Python dependencies.
- Use GitHub’s Dependabot to do another dependency check.
- Use Docker Scout and trivy to check vunerabilities in your docker images.
- Try to resolve all the vulnerabilites.
- Check the docs of the tools to interpret reports and vulns.

Lab take away

- ❑ Familiarise with static analysis tools for security.
- ❑ Fix vulnerabilities in codebase, dependencies and docker images.

Project take away

- ❑ Use a static analysis tool to check for code vulnerabilities (only if your programming language(s) have one free tool for it).
- ❑ Your project should be free of dependencies vulnearbilities.
- ❑ Your docker images must be free of fixable docker image vulnerabilities with critical and high severity.