

DevSecOps, day 4



2. Lab: Static Analysis (SAST)

SAST with a regex engine

- We will use Semgrep, a pattern matching SAST.
- This will work on our Dev Workstation,
 - As well as in Azure DevOps!
- <https://github.com/returntocorp/semgrep>

On your Dev Workstation

- We will use the Docker-based solution:

```
$ docker pull returntocorp/semgrep
```

```
$ cd ~/Team1JS
```

```
$ docker run --rm -v "$(pwd):/src" \  
  returntocorp/semgrep \  
  semgrep scan --config auto
```

In Azure DevOps

- With our pipeline we can use Docker,
 - Or just install and run Semgrep with Python
- There's a sample pipeline: *pipeline-step4-semgrep.yml*

Pipeline additions

- First we install Semgrep.

```
- stage: SAST
  jobs:
    - job: semgrep_sast
      steps:
        Settings
        - task: Bash@3
          displayName: install_semgrep
          inputs:
            targetType: 'inline'
            workingDirectory: '$(Build.SourcesDirectory)'
            failOnStderr: true
            script: |
              python3 -m pip install semgrep
```

Pipeline additions

- Then we run the test.
 - And specify a Sarif output file.

```
... - task: Bash@3
...   continueOnError: true ..... # We do not want to "break the build".
...   displayName: run_semgrep
...   inputs:
...     targetType: 'inline'
...     workingDirectory: '$(Build.SourcesDirectory)'
...     failOnStderr: true
...     script: |
...       semgrep --sarif --output semgrep-result.sarif --config auto
```


Pipeline additions

- The Sarif file is published.
 - The name "CodeAnalysisLogs" is needed,
 - For the "scan results" tab. ([source](#))

```
... - task: PublishBuildArtifacts@1
...   inputs:
...     PathToPublish: '$(Build.SourcesDirectory)/semgrep-result.sarif'
...     ArtifactName: 'CodeAnalysisLogs'
...     publishLocation: 'Container'
```


Pipeline results: "scans" tab

#20220830.7 fix sarif export
ITVitae team 1 JuiceShop

Summary **Scans**

Cancel

Filter by keyword

Baseline: New (+2) Level: Error (+1)

semgrep 113

Path Details ↑ Baseline

ajinabraham.njsscan.nosql_find_injection.node_nosqli_injection: ajinabraham.njsscan.nosql_find_injection.node_nosqli_injection 25

! routes/address.ts

Untrusted user input in findOne() function can result in NoSQL Injection.

18 const address = await AddressModel.findOne({ where: { id: req.params.id } })

New

! routes/basket.ts

Untrusted user input in findOne() function can result in NoSQL Injection.

17 const id = req.params.id
18 BasketModel.findOne({ where: { id }, include: [{ model: ProductModel }]
19 .then((basket: BasketModel | null) => {
20 /* jshint eqeqeq:false */
21 challengeUtils.solveIf(challenges.basketAccessChallenge, () => {
22 const user = security.authenticatedUsers.from(req)
23 return user && id && id !== 'undefined' && id !== 'null' &&
24 })
25 })

New

! routes/basket.ts

Untrusted user input in findOne() function can result in NoSQL Injection.

18 BasketModel.findOne({ where: { id }, include: [{ model: ProductModel }]

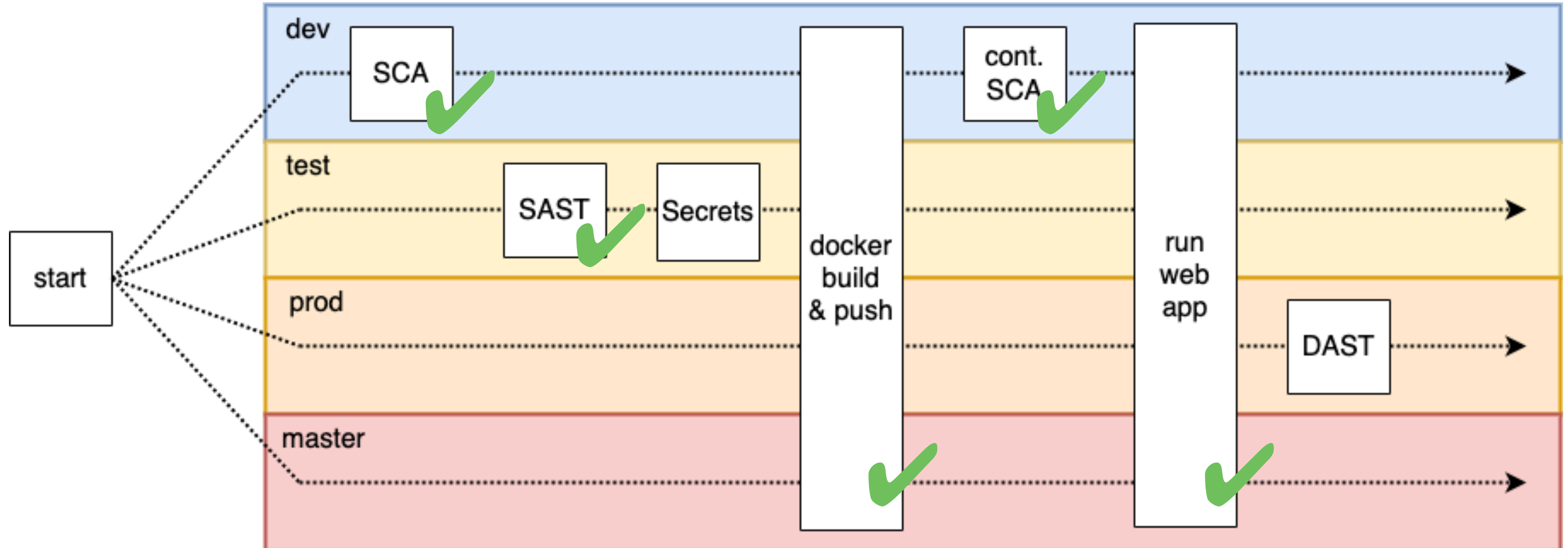
New

Checkpoint!

- Does everyone have:
 - A pipeline on the "dev" branch.
 - Which still builds + runs the webapp,
 - Plus NPM + Trivy + Semgrep?
- Have you tested this?



Our final pipeline goal



2. Lab: Semgrep in your IDE

Clone the JuiceShop Git project

- The Git repo was already on your VM, yes.
 - But for VS:Code you need it on the host OS.
- Clone the usual Juiceshop repo:
<https://github.com/Unixerius/juiceshop15>

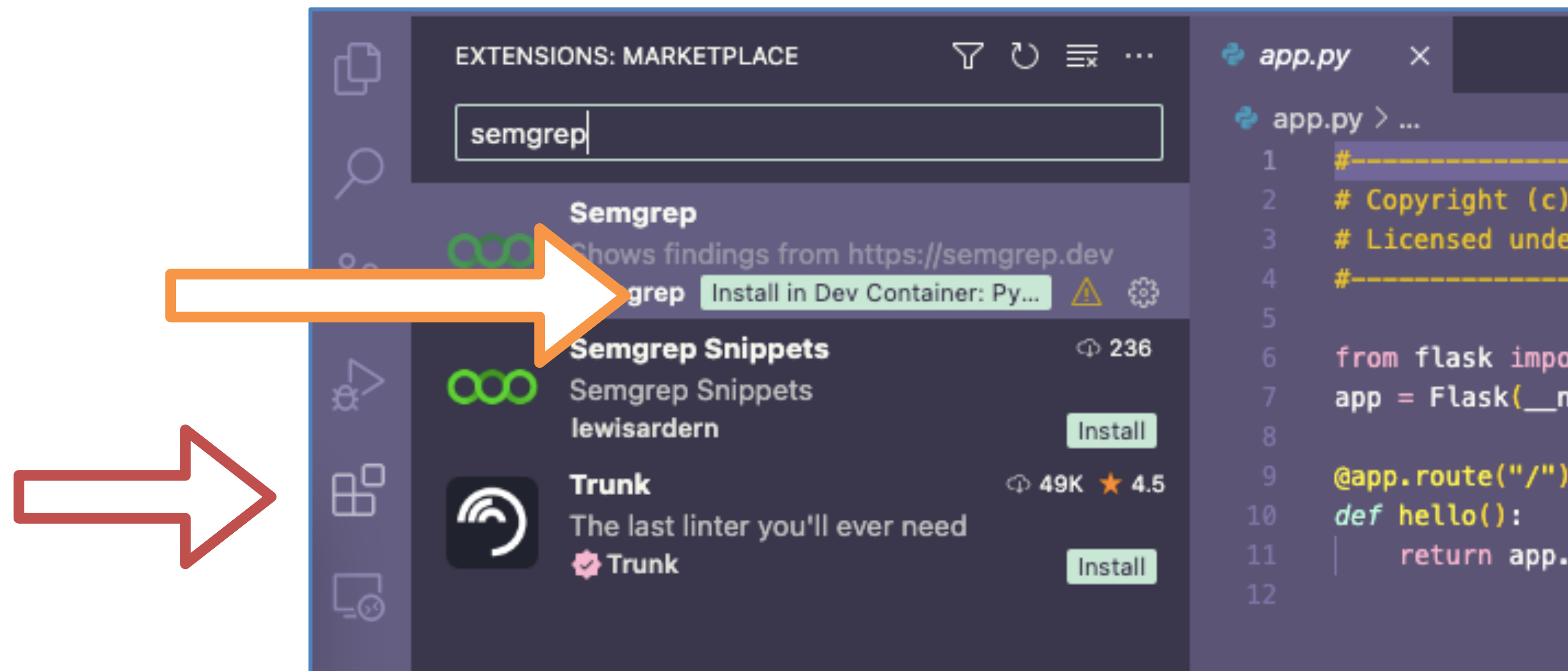
Install VS:Code

- Let's try this!
 - Download Visual Studio Code.
 - And yes, install it.

<https://code.visualstudio.com>

SAST in your IDE

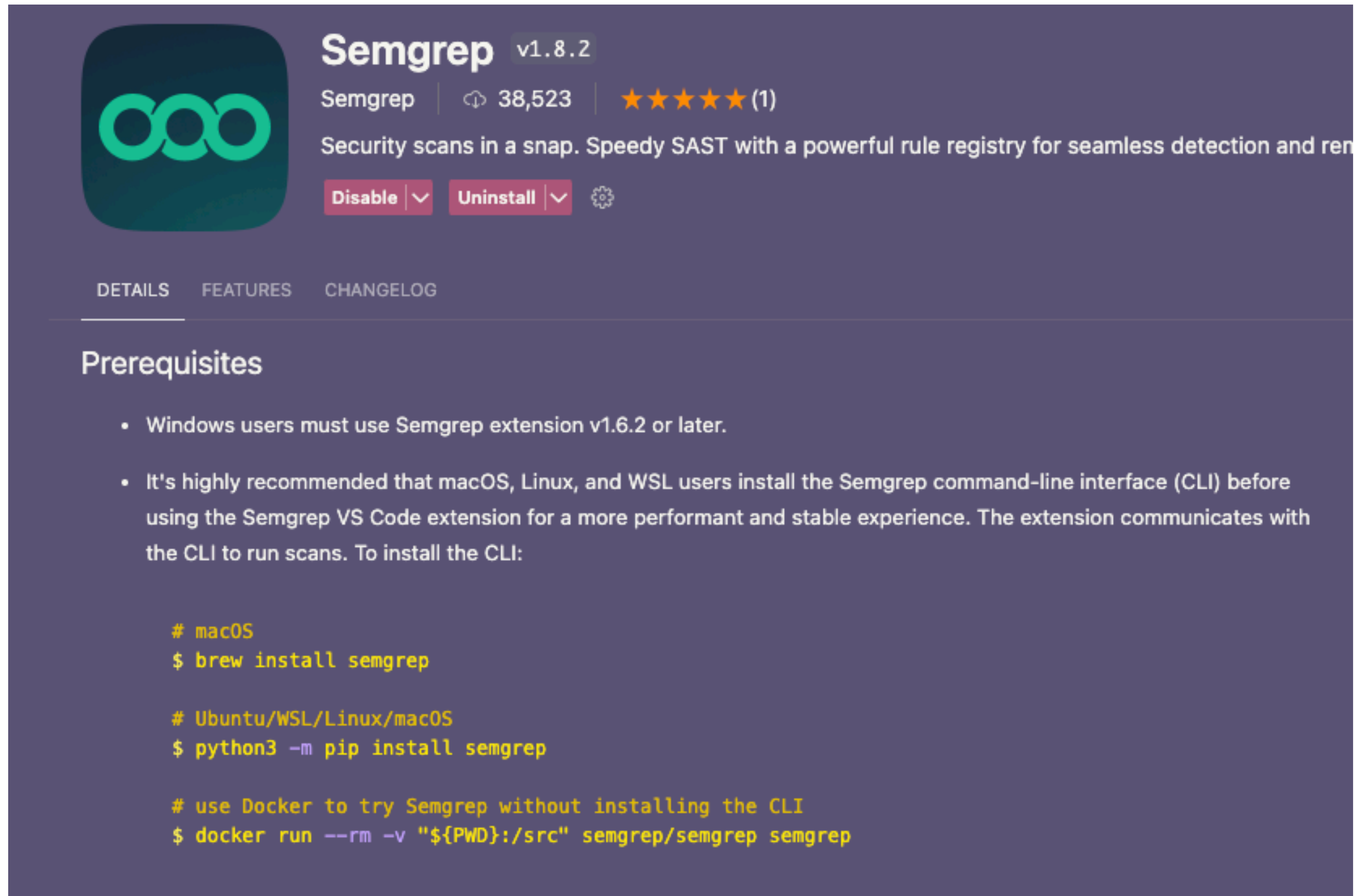
- Search for the extension "Semgrep".
 - Click "*Install*".



Time to make a choice ⚠

- How you proceed depends on your host setup.
 - MacOS with Brew and Python?
 - MacOS with Docker?
 - Windows with Docker?
 - Windows with WSL and Python?
 - Linux with Docker, or Python?

Time to make a choice ⚠



The screenshot shows the Semgrep extension page in the Visual Studio Code marketplace. The extension is version 1.8.2, has 38,523 downloads, and a 5-star rating. The description states: "Security scans in a snap. Speedy SAST with a powerful rule registry for seamless detection and remediation." Below the description are buttons for "Disable", "Uninstall", and a settings gear icon. The "Prerequisites" section lists two main points: Windows users must use Semgrep extension v1.6.2 or later, and macOS, Linux, and WSL users should install the Semgrep CLI before using the VS Code extension. The CLI installation commands are provided for macOS, Ubuntu/WSL/Linux/macOS, and Docker.

Semgrep v1.8.2
Semgrep | 38,523 | ★★★★★ (1)
Security scans in a snap. Speedy SAST with a powerful rule registry for seamless detection and remediation.

Disable | Uninstall | ⚙

DETAILS | FEATURES | CHANGELOG

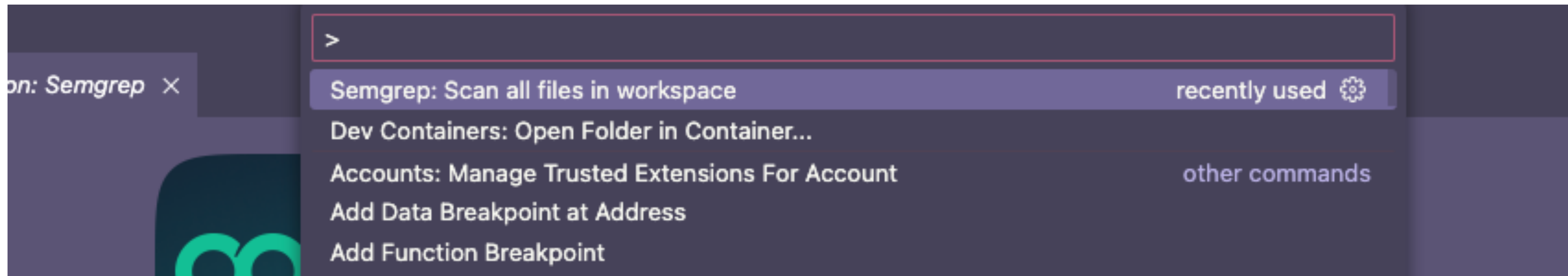
Prerequisites

- Windows users must use Semgrep extension v1.6.2 or later.
- It's highly recommended that macOS, Linux, and WSL users install the Semgrep command-line interface (CLI) before using the Semgrep VS Code extension for a more performant and stable experience. The extension communicates with the CLI to run scans. To install the CLI:

```
# macOS  
$ brew install semgrep  
  
# Ubuntu/WSL/Linux/macOS  
$ python3 -m pip install semgrep  
  
# use Docker to try Semgrep without installing the CLI  
$ docker run --rm -v "${PWD}:/src" semgrep/semgrep semgrep
```

Time to make a choice ⚠

- Do the Semgrep install of your choice.
- Then in VS:Code press F1 and type/select:
 - *Semgrep: scan all files in workspace.*



Semgrep in your IDE

- While we are talking,
 - Semgrep is already going over all the code.
- Every file in our project will get scanned,
 - And will have results (warnings) to show.

SQL injection?!

- Open the following file:
 - *routes/login.ts*
- After Semgrep is done, check the SELECT statements.
 - It should warn about SQL Injection issues.

Our "bingo" card

DevOps	Intro	Work IRL	AzDO	SDLC	Juice!	
DevOps	Agile	Git	Virtual	Contain	CI/CD	
DevSecOps	App test	DSO	Vulns	SCA	TModel	
DevSecOps	SAST	Secrets	PROD			
DevSecOps	VulnScan	Pentest	DAST	WAF		

4. Lab: Secrets detection

We'll go hunting!

- Let's use Gitleaks.
 - An open source tool, with solid support.
- <https://github.com/zricethezav/gitleaks>

On your Dev Workstation

- We will use the Docker-based solution:

```
$ cd ~/Team1JS
```

```
$ docker run --rm -v ${PWD}:/source \
zricethezav/gitleaks detect \
--source="/source" --verbose
```

On your Dev Workstation

- Gitleaks doesn't look at *just* the current state.
 - It also searches your Git history!
- Gitleaks finds quite some secrets.
 - Some tools are better than others.
 - Some tools require more configuration.

In Azure DevOps

- Gitleaks is written in Go.
- With our pipeline we can install Golang,
 - But in this case, Docker's a lot easier!
 - No dependency hell, much faster.
- There's a sample pipeline: *pipeline-step5-gitleaks.yml*

Pipeline additions

```
- job: Gitleaks
  steps:
  - task: Bash@3
    continueOnError: true          # We do not want to "break the build".
    displayName: run_gitleaks
    inputs:
      targetType: 'inline'
      workingDirectory: '$(Build.SourcesDirectory)'
      script: |
        docker run -v "$(pwd):/pwd" zricethezav/gitleaks detect\
        --source="/pwd" \
        --redact \
        --no-banner \
        --report-path gitleaks-result.json

  - publish: '$(Build.SourcesDirectory)/gitleaks-result.json'
    artifact: gitleaks-result.json
```

False positives?

- You can suppress warnings on false positives!
- Research the *".gitleaksignore"* file.
 - You make this on your workstation,
 - And add it to the Git repo.
- If you have extra time, go try this!

Do you have more time?

- Gitleaks works?
 - And you have used .gitleaksignore?
- Bored?
 - There's also TruffleHog to give a try.

We'll go hunting!

- Let's use TruffleHog. 🐷
 - An open source tool, with a commercial version.
- <https://github.com/trufflesecurity/trufflehog>

On your Dev Workstation

- We will use the Docker-based solution:

```
$ cd ~/Team1JS
```

```
$ docker run --rm -it -v "$(pwd):/pwd" \
trufflesecurity/trufflehog filesystem \
/pwd --no-verification
```

On your Dev Workstation

- Oddly, TruffleHog only finds two SSH keys.
 - The hard-coded JWT aren't found.
- Some tools are better than others.
 - Some tools require more configuration.

In Azure DevOps

- With our pipeline again let's use Docker.
- There's a sample pipeline: *pipeline-step5-truffles.yml*

Pipeline additions

- job: TruffleHog
steps:
 - task: Bash@3
continueOnError: true
displayName: run_trufflehog
inputs:
 - targetType: 'inline'
 - workingDirectory: '\$(Build.SourcesDirectory)'
 - script: |
docker run --rm -v "\$(pwd):/pwd" trufflsecurity/trufflehog \
filesystem /pwd --no-verification \
--json --fail | tee trufflehog-result.json
- publish: '\$(Build.SourcesDirectory)/trufflehog-result.json'
artifact: trufflehog-result.json

Checkpoint!

- Does everyone have:
 - A pipeline on the "dev" branch.
 - Which still builds + runs the webapp,
 - Plus NPM + Trivy + Semgrep + Truffles?
- Have you tested this?



4. Lab: Pre-commit hook

No more foot-shooting

- Let's add Gitleaks to our workstation,
 - In a way to block committing secrets.
- This uses a "pre-commit hook".
 - <https://github.com/gitleaks/gitleaks#pre-commit>

On your Dev Workstation

```
$ pip3 install pre-commit  
$ cd ~/Team1JS  
  
$ vi .pre-commit-config.yaml  
# The next slide has the contents!
```

On your Dev Workstation

repos:

- repo: `https://github.com/zricethezav/gitleaks`
rev: `v8.16.0`

hooks:

- id: `gitleaks`

On your Dev Workstation

```
$ PATH="$PATH:~/local/bin"
```

```
$ export PATH
```

```
$ pre-commit autoupdate
```

```
$ pre-commit install
```

Let's test this!

```
$ cp ~/.ssh/id_rsa ~/Team1JS  
$ git add id_rsa  
$ git commit -m "Shooting my own foot!"
```

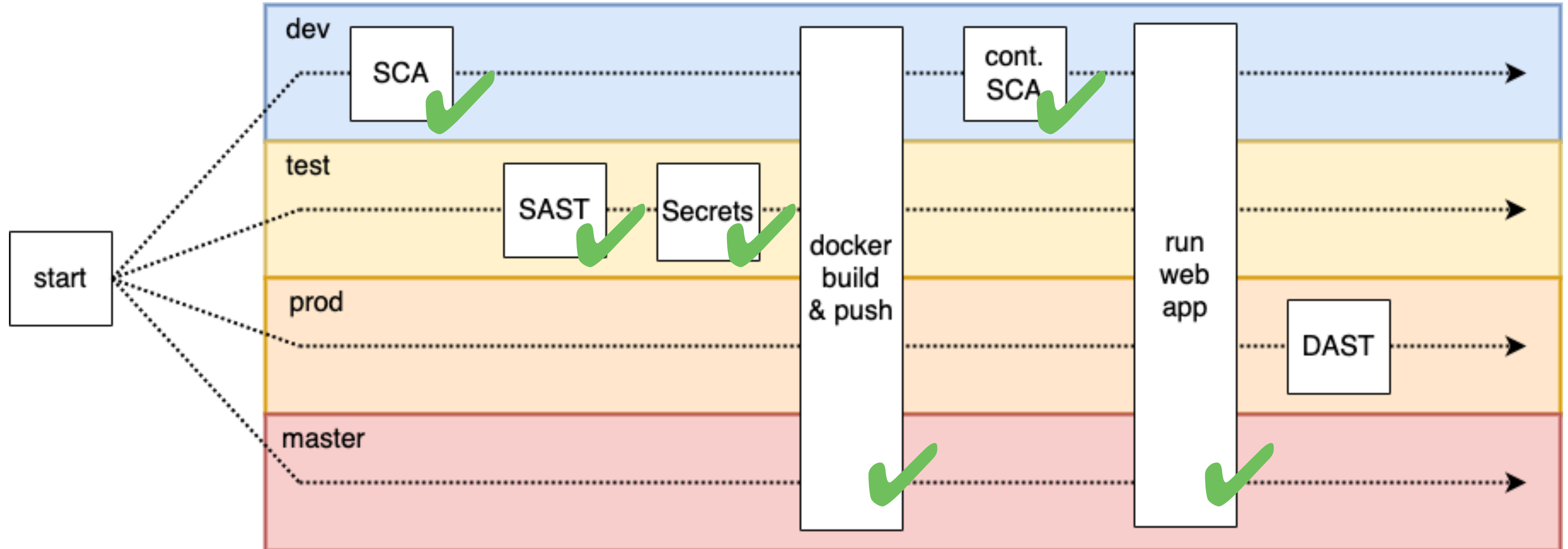
Did Gitleaks block the commit?

Checkpoint!

- Does everyone have:
 - A pipeline on the "dev" branch.
 - Which still builds + runs the webapp,
 - Plus NPM + Trivy + Semgrep + Gitleaks?
- Have you tested this?



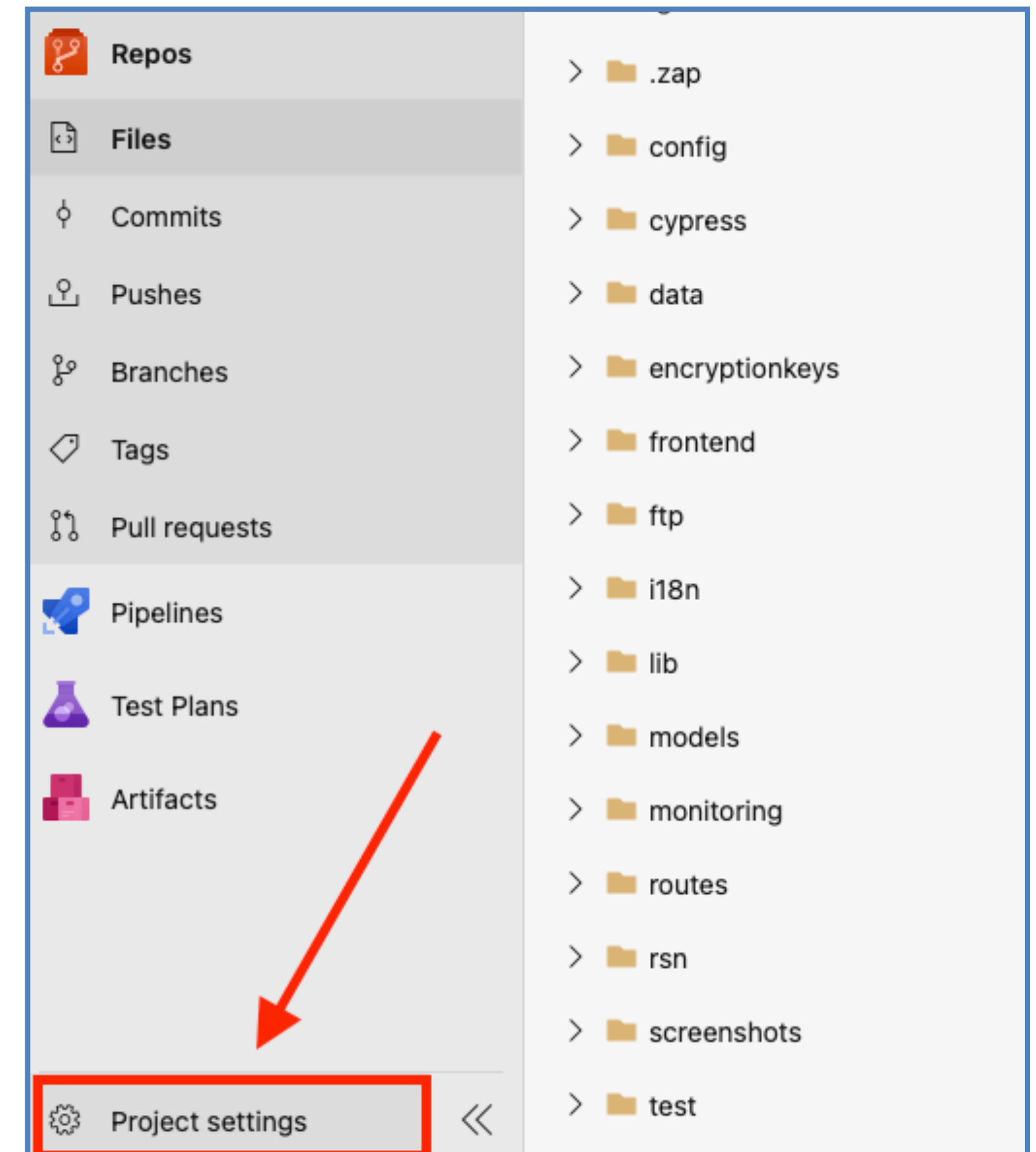
Our final pipeline goal



6. Lab: Branch policies

Configuring branch policies

- In Azure DevOps,
 - To to your **Project Settings**.



Configuring branch policies

- Go to **Repositories**,
 - To the **Policies** tab,
- At the bottom, with **Branch Policies**,
 - Click on the **+** button.
 - Create a policy for the "*prod*" branch.

New policy for prod

Project Settings

- Permissions
- Notifications
- Service hooks
- Dashboards
- Boards**
- Project configuration
- Team configuration
- GitHub connections
- Pipelines**
- Agent pools
- Parallel jobs
- Settings
- Test management
- Release retention
- Service connections
- XAML build services
- Repos**
- Repositories**
- Artifacts
- Storage
- Test
- Retention

All Repositories + Create

Repositories Settings **Policies** Security

Repository Policies

- ☐ Off **Commit author email validation**
Block pushes with a commit author email that does not match the following patterns.
- ☐ Off **File path validation**
Block pushes from introducing file paths that match the following patterns.
- ☐ Off **Case enforcement**
Avoid case-sensitivity conflicts by blocking pushes that change name casing on files, folders, branches, and tags. [Learn more](#)
- ☐ Off **Reserved names**
Block pushes that introduce files, folders, or branch names that include platform reserved names or incompatible characters. [Learn more](#)
- ☐ Off **Maximum path length**
Block pushes that introduce paths that exceed the specified length. [Learn more](#)
- ☐ Off **Maximum file size**
Block pushes that contain new or updated files larger than this limit.

Branch Policies
Protect important branch namespaces across all repositories in this project

🔗 prod

➕

New policy for prod

Add branch protection

×

Branches to protect

☐ Protect the default branch of each repository

☒ Protect current and future branches matching a specified pattern

prod|

Example patterns: "main", "master", "releases/*", "**"

Matches 1 branch in 1 repo

🔗 prod

ITVitae team 1 JuiceShop

Require reviews

- 1 review minimum.
- New changes?
 - Reset votes!

Branch Policies
Note: If any required policy is enabled, this branch cannot be deleted and changes must be made via pull request.

☒ On

Require a minimum number of reviewers
Require approval from a specified number of reviewers on pull requests.

Minimum number of reviewers

☐ Allow requestors to approve their own changes
☐ Prohibit the most recent pusher from approving their own changes
☐ Allow completion even if some reviewers vote to wait or reject
☒ When new changes are pushed:

- ☐ Require at least one approval on the last iteration
- ☒ Reset all approval votes (does not reset votes to reject or wait)
- ☐ Reset all code reviewer votes

Resolve all comments

- All comments must be resolved.

☒ On

Check for comment resolution
Check to see that all comments have been resolved on pull requests.

☒ **Required**
Block pull requests from being completed while any comments are active.

☐ **Optional**
Warn if any comments are active, but allow pull requests to be completed.

Testing the policy

- Go back to your repository.
- One person creates a pull request,
 - From "dev" or "test", to "prod".
- Does the pull request require approvals?
 - Can someone in the team approve?
 - Does the merge work, after approval?

Checkpoint!

- Does everyone have:
 - Three new branches? dev/test/prod
 - A protected prod branch?
- Did you prove they all work?!



Closing



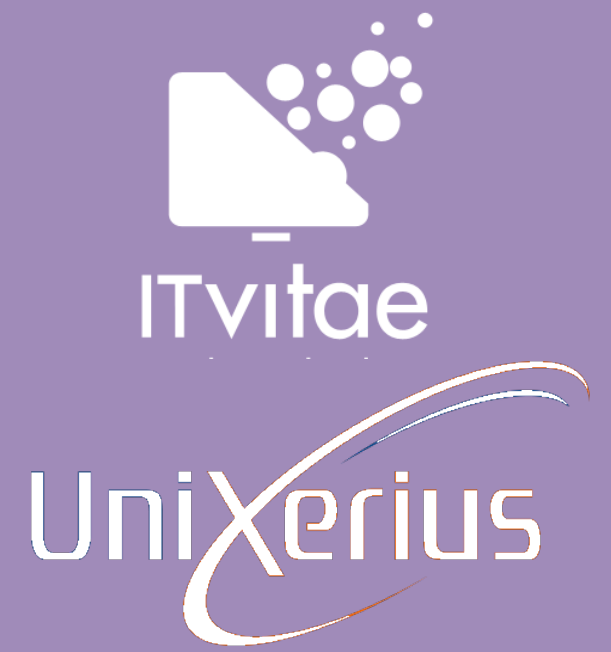
Tomorrow

- Vulnerability scanning
- Pen-testing
- DAST: Dynamic Application Security Testing
- WAF: Web App Firewalls

Homework

- [Request Nessus Essentials activation code](#)
 - Do not use it yet, only request it.
 - Make sure it's for **Nessus Essentials**.

Reference materials



Resources

- [In-depth explanation of SAST](#) (TU Delft)
- [How bad can it Git?](#)
- [What to do after leaking secrets](#)
- [Request Nessus Essentials activation code](#)