

# 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**

Filter by keyword Baseline: New (+2) Level: Error (+1) X

semgrep 113

| Path   | Details ↑   | Baseline |
|--|---|----------|
| ajinabraham.njsscan.nosql_find_injection.node_nosqli_injection: ajinabraham.njsscan.nosql_find_injection.node_nosqli_injection | Untrusted user input in findOne() function can result in NoSQL Injection.<br>18     const address = await AddressModel.findOne({ where: { id: req.params.id } })  | 25 New   |
| routes/address.ts  |   |          |
| routes/basket.ts   | Untrusted user input in findOne() function can result in NoSQL Injection.<br>17     const id = req.params.id<br>18     BasketModel.findOne({ where: { id }, include: [{ model: ProductModel } ] })<br>19     .then((basket: BasketModel   null) => {<br>20         /* jshint eqeqeq:false */<br>21         challengeUtils.solveIf(challenges.basketAccessChallenge, () => {<br>22             const user = security.authenticatedUsers.from(req)<br>23             return user && id && id !== 'undefined' && id !== 'null' && user.id === id<br>24         })<br>25     }) | New      |
| routes/basket.ts   | Untrusted user input in findOne() function can result in NoSQL Injection.<br>18     BasketModel.findOne({ where: { id }, include: [{ model: ProductModel } ] })   | New      |

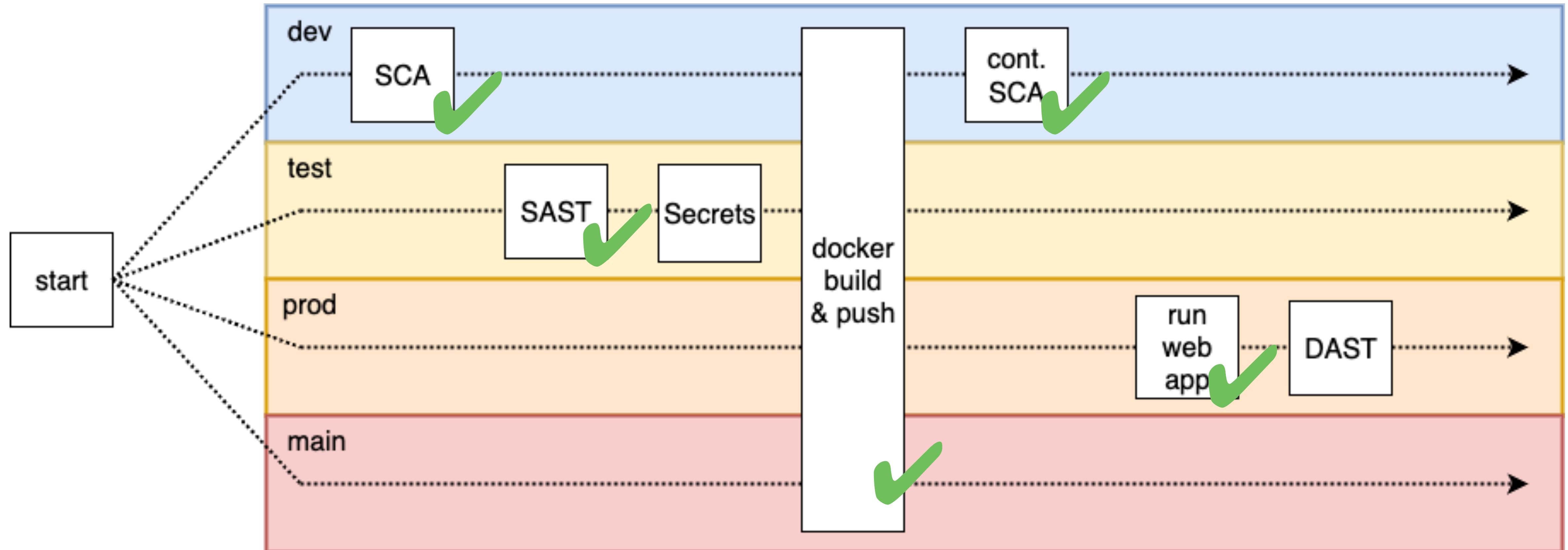
# Checkpoint!

- Does everyone have:
  - A pipeline on the "test" branch.
  - With NPM + Trivy + Semgrep?
- Have you tested this?



SmartSign.com • 800-952-1457 • K2-4293

# 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:

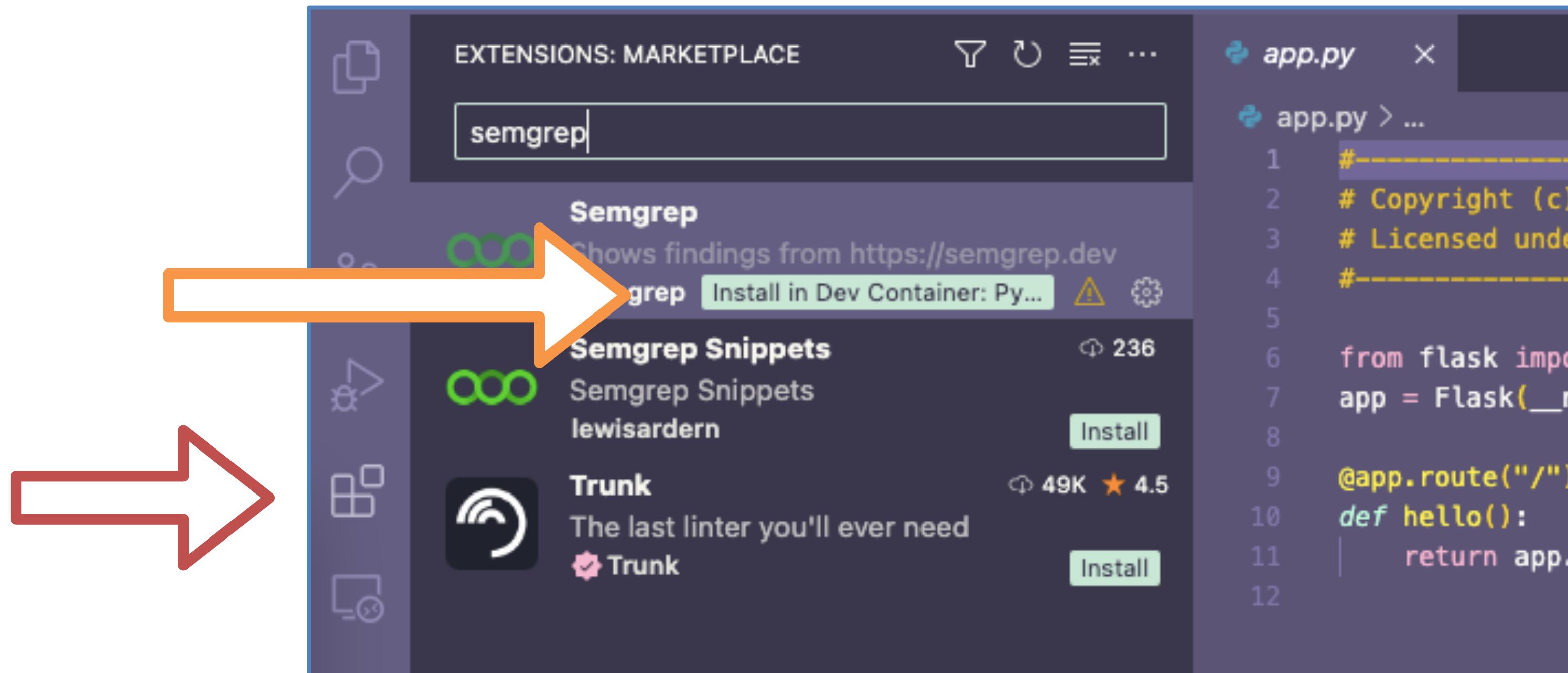
<https://github.com/Unixerius/juiceshop15>

# Install VS:Code

- Let's try this!
  - Download [Visual Studio Code](#).
  - And yes, install it.
- Already use IntelliJ?
  - Skip this lab and install [Spotbugs](#).

# SAST in your IDE

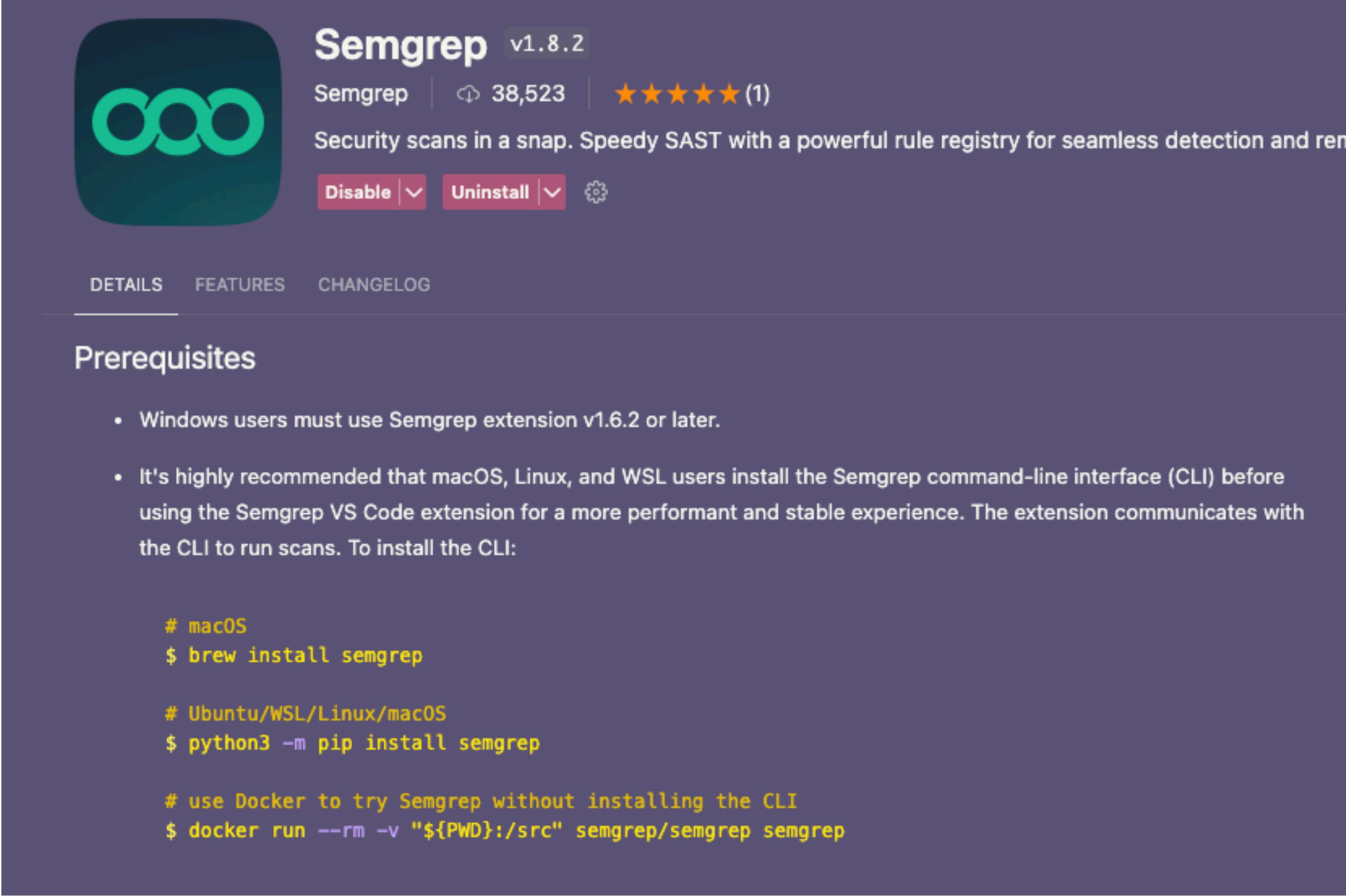
- Search for the extension "Semgrep".
  - Click "*Install*".



# 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 Microsoft Store. The extension is version v1.8.2, has 38,523 installs, and a 5-star rating from 1 review. The description states: "Security scans in a snap. Speedy SAST with a powerful rule registry for seamless detection and ren...". Below the description are "Disable" and "Uninstall" buttons. At the bottom of the page, there is a "Prerequisites" section with a bulleted list and some command-line installation instructions.

**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:

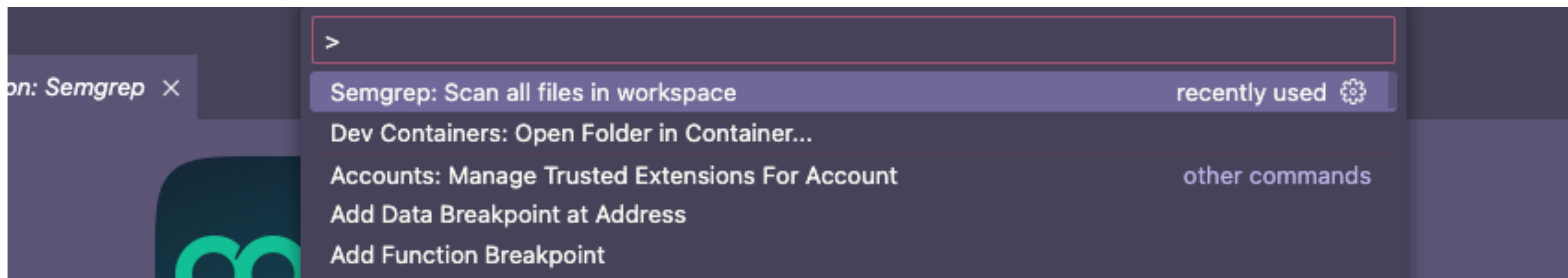
```
# macOS
$ brew install semgrep

# 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
```

# Scanning with Semgrep

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



# Crash?

- The Semgrep extension may crash or fail.
  - Try installing an older version, like 1.9.x

# 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.

# 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}:/src \  
zricethezav/gitleaks detect \  
--source="/src" --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
```

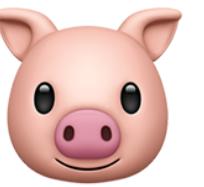
# False positives?

- You can suppress warnings on false positives!
- Research the `".gitleakignore"` 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):/src" \  
trufflesecurity/trufflehog filesystem \  
/src --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) :/src" trufflesecurity/trufflehog \
filesystem /src --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 "test" branch.
  - With NPM + Trivy + Semgrep + Truffles?
- Have you tested this?



SmartSign.com • 800-952-1457 • K2-4293

# 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

- Instead of installing the binaries, let's use Docker.

```
$ docker pull docker.io/zricethezav/  
gitleaks:latest
```

# On your Dev Workstation

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

# On your Dev Workstation

```
repos:
  - repo: local
    hooks:
      - id: gitleaks
        name: Detect hardcoded secrets
        description: Detect hardcoded secrets with Gitleaks
        entry: bash -c 'docker run --rm -v "${PWD}:/src"
docker.io/zricethezav/gitleaks:latest protect --verbose
--redact --staged --source="/src"'
        language: system
```

# 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?

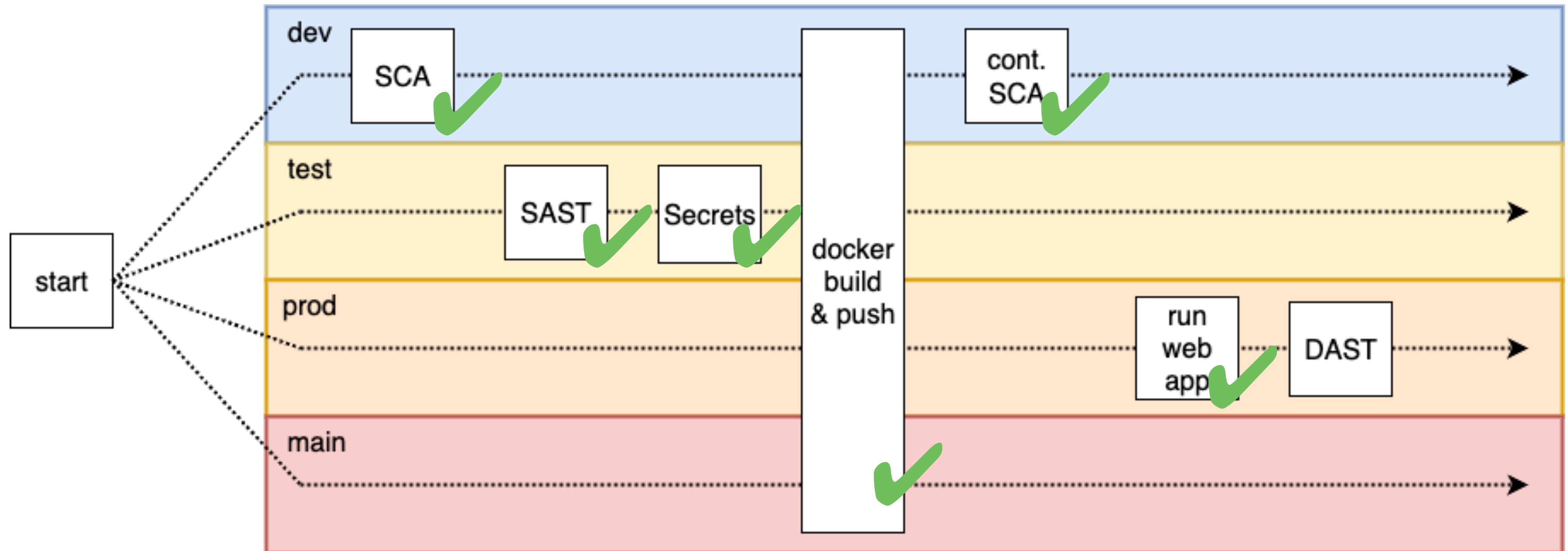
# Checkpoint!

- Does everyone have:
  - Pre-commit blocking new secrets?
- Have you tested this?



SmartSign.com • 800-952-1457 • K2-4293

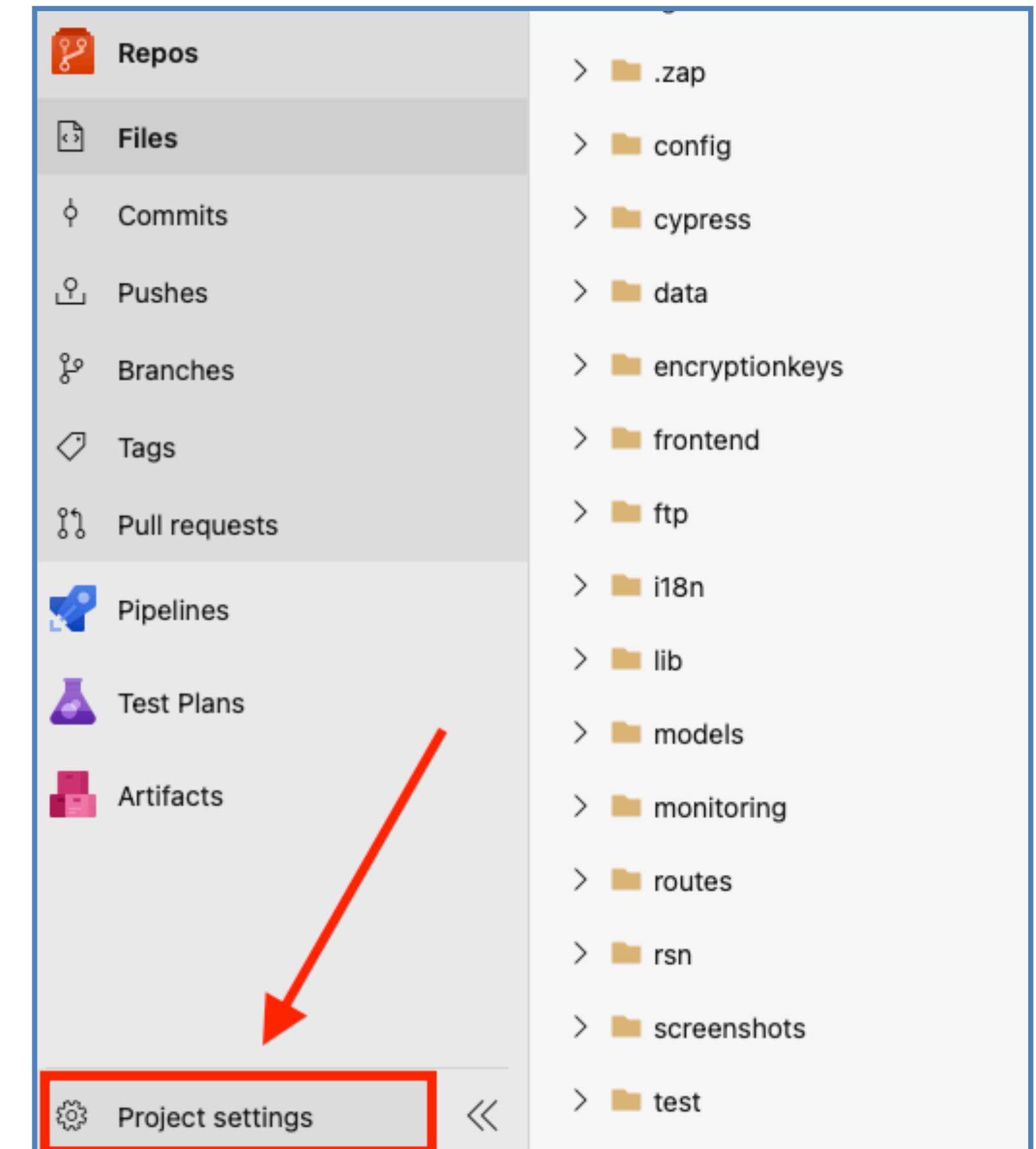
# Our final pipeline goal



# 6. Lab: Branch policies

# Configuring branch policies

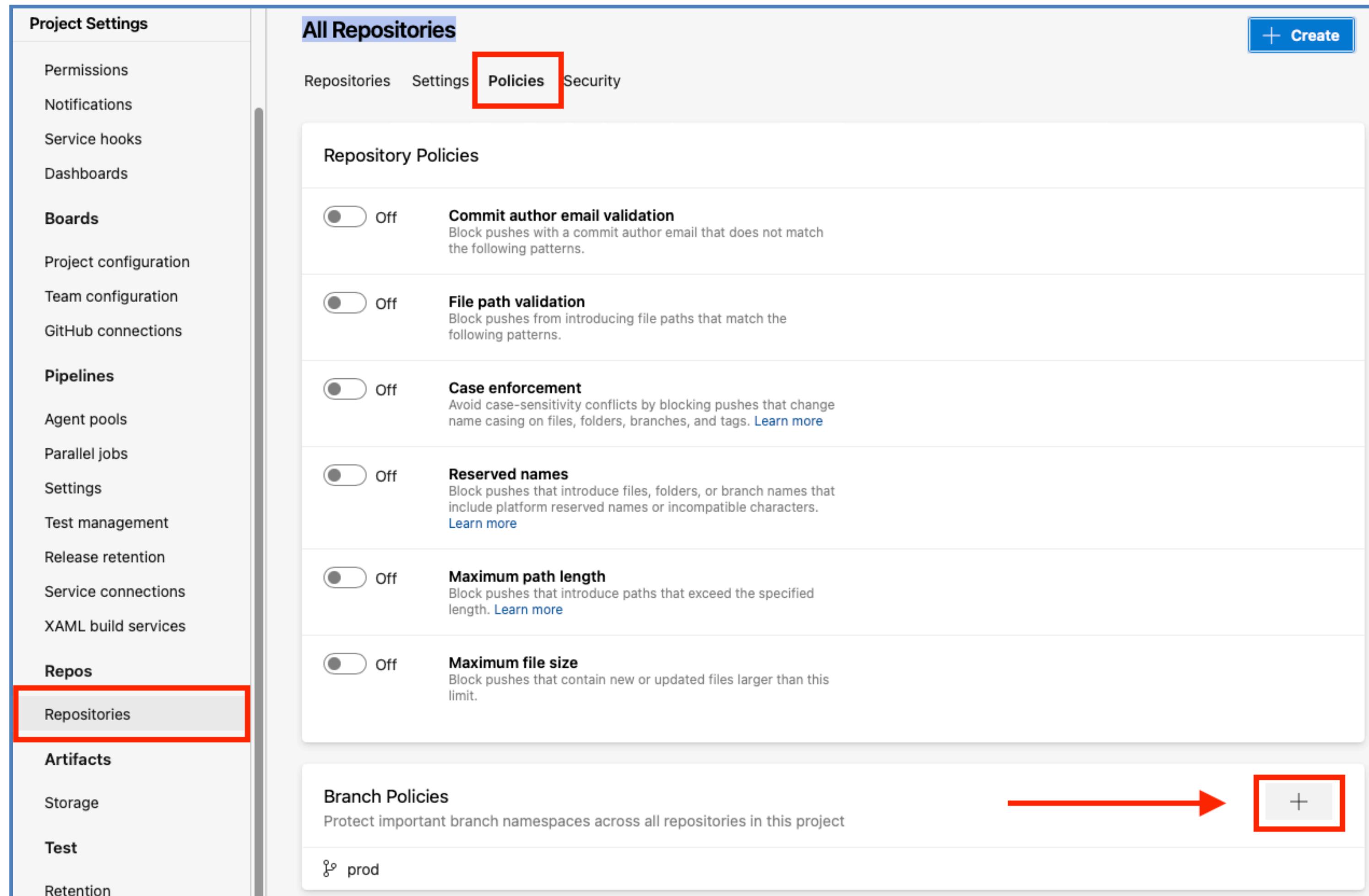
- In Azure DevOps,
  - To to your **Project Settings**.



# 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



The screenshot shows the GitHub Project Settings interface for a project named 'All Repositories'. The left sidebar lists various project settings categories: 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, and Repos. The 'Repositories' item under the 'Repos' section is highlighted with a red box. The main content area is titled 'All Repositories' and shows tabs for Repositories, Settings, Policies (which is selected and highlighted with a red box), and Security. A 'Create' button is in the top right. The 'Repository Policies' section lists six policies, all currently off: Commit author email validation, File path validation, Case enforcement, Reserved names, Maximum path length, and Maximum file size. Below this is a 'Branch Policies' section for protecting the 'prod' branch. A red arrow points to a '+' button in the bottom right corner of the 'Branch Policies' section, indicating where a new policy can be created.

# 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  
1

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:
  - A protected prod branch?
- Did the pull request work?!



SmartSign.com • 800-952-1457 • K2-4293

# Homework: lab prep

# Homework

- Request Nessus Essentials activation code
  - Make sure it's for **Nessus Essentials**.

# Closing

# Tomorrow

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

# Relevant reading

| Topic                   | Book  |
|-------------------------|-------|
| Operations and security | Ch 13 |
| Compliance              | Ch 14 |
| Agile security          | Ch 16 |
|                         |       |
|                         |       |

# 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](#)