



# Application Vulnerability Scanning

## Key Takeaways

# All rights reserved to nnSoftware GmbH

No part of this publication may be reproduced, copied, transmitted in any form or by any means, electronic, mechanical, photocopying, recording or otherwise, without the prior written permission of nnSoftware GmbH

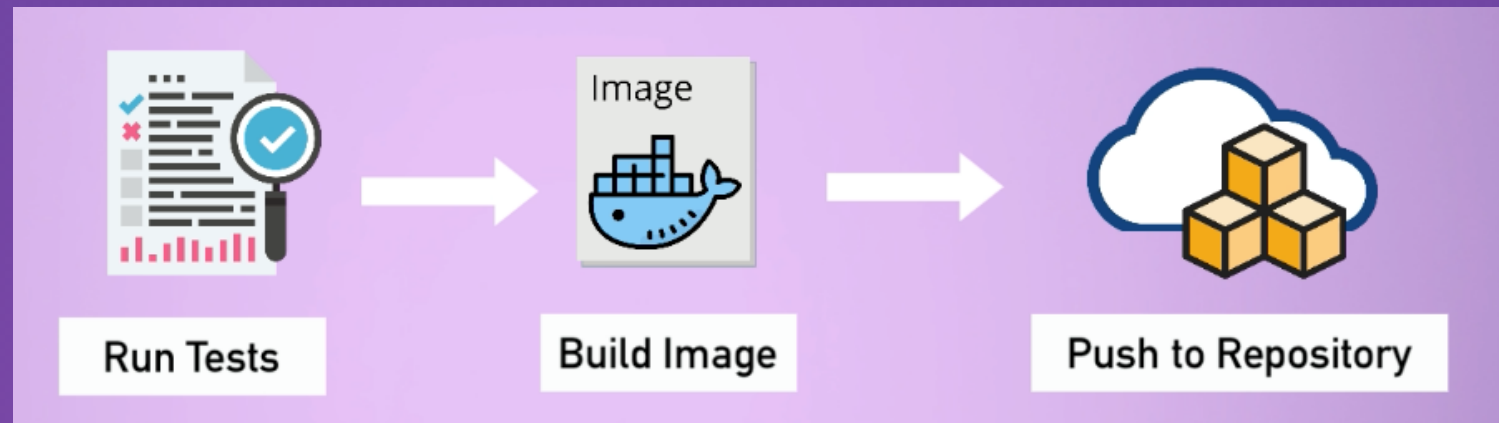
# About TechWorld with Nana

TechWorld with Nana is an established name in the DevOps and Cloud industry, and it stands for the quality trainings helping 1,000s of engineers acquire the most in-demand skills in this field.

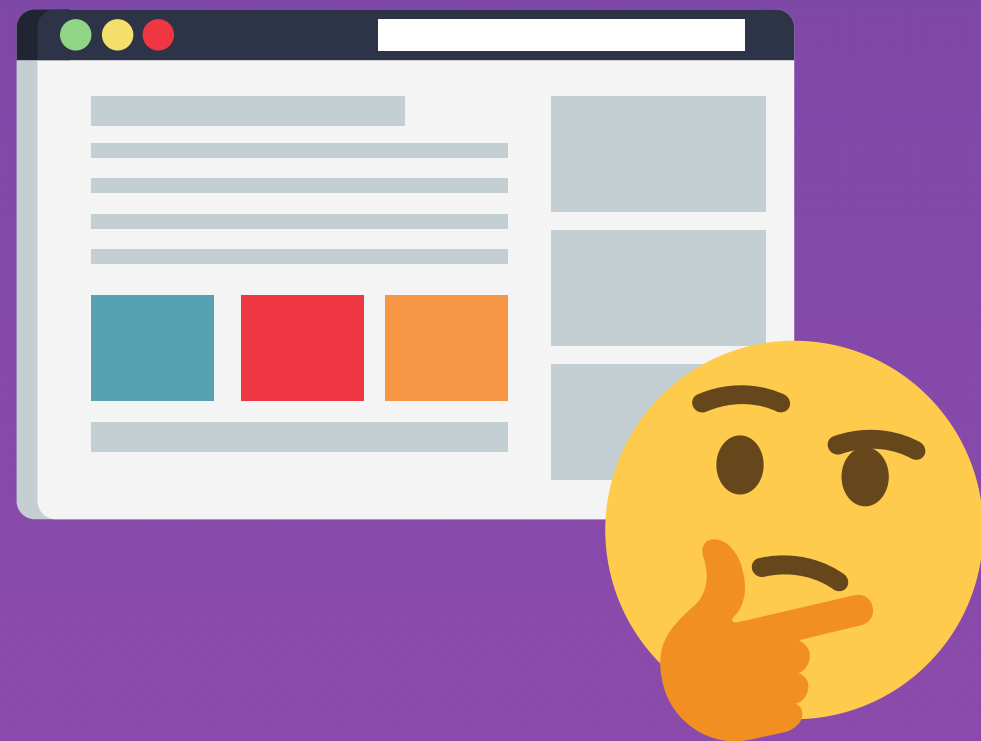


Our mission is enable individual engineers as well as companies to take advantage of the recent developments in Cloud and DevOps fields, to use technologies and concepts in order to create efficient, automated, streamlined DevSecOps processes in organisations.

# Missing Security Scanning Steps in CI Pipeline



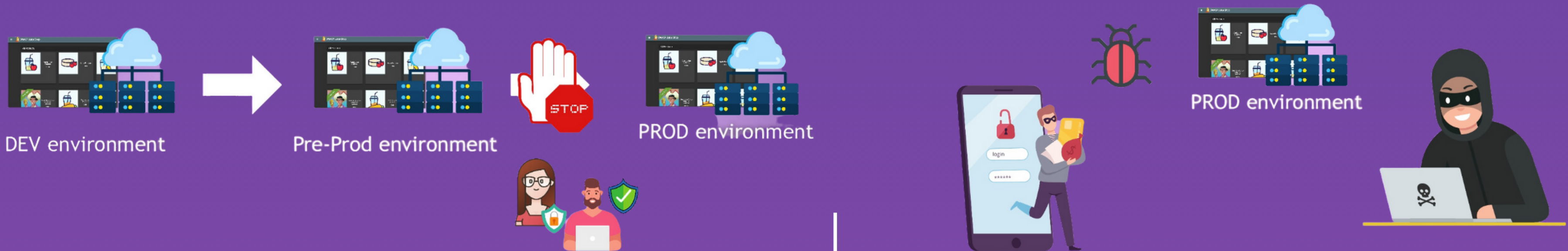
CI Pipeline - No security checks



With a simple, basic CI pipeline with **no security scanning steps**, we don't know:

- ✗ How secure our application is
- ✗ If our code allows SQL injections
- ✗ If there are any vulnerabilities for XSS
- ✗ If there are any hardcoded credentials

# 2 Options, one worse than the other



## Security Team checking before release

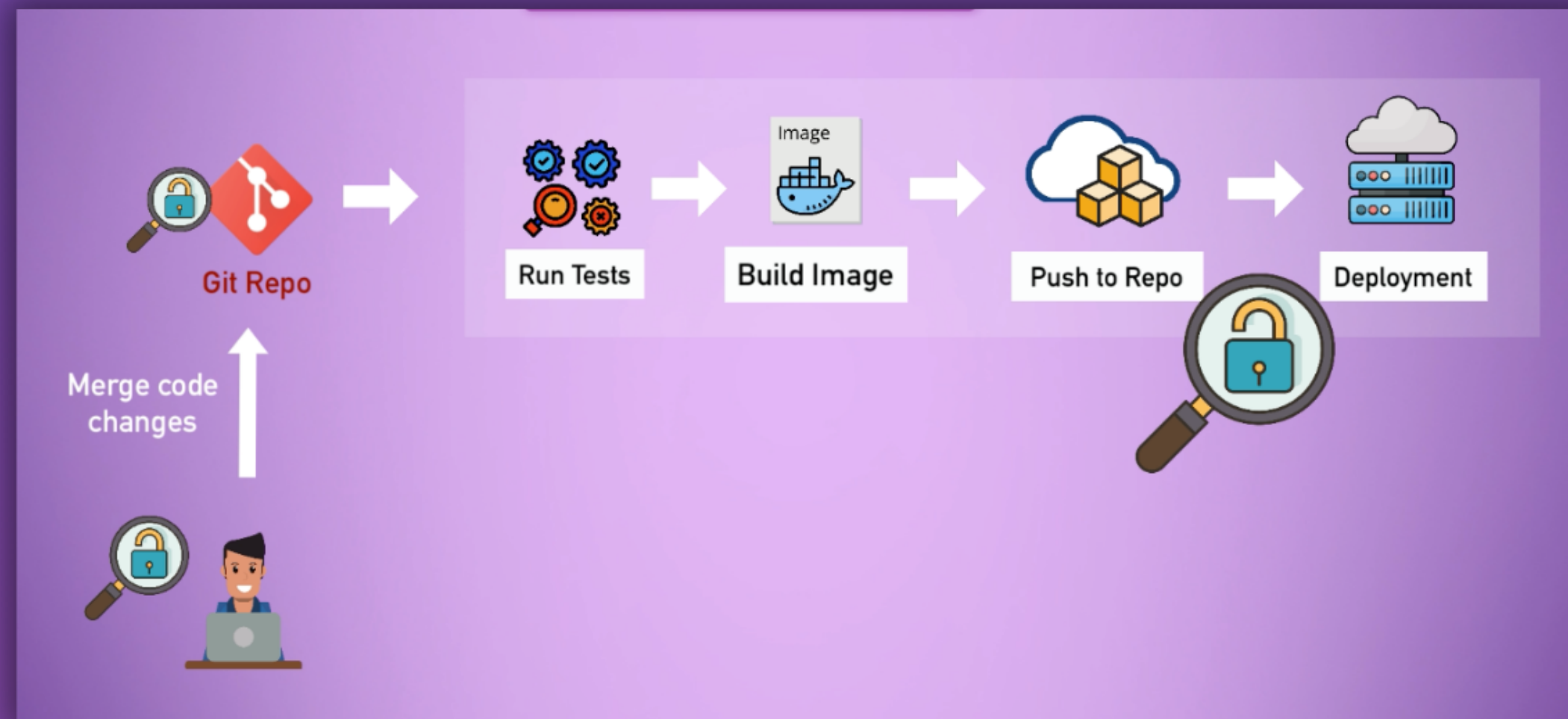
- ❌ Manual security checks, would block right before release
- ❌ Slow feedback cycle, blocking release for weeks or months

## No manual checks - Release to Production

- ❌ If we release it, we would end up with an app full of security vulnerabilities
- ❌ We are blind to what we are deploying



# Solution: Integrate App Vulnerability Scanning Tools



So we want to add the security tests from the beginning, integrating it into the developer workflow instead of a separate isolated step

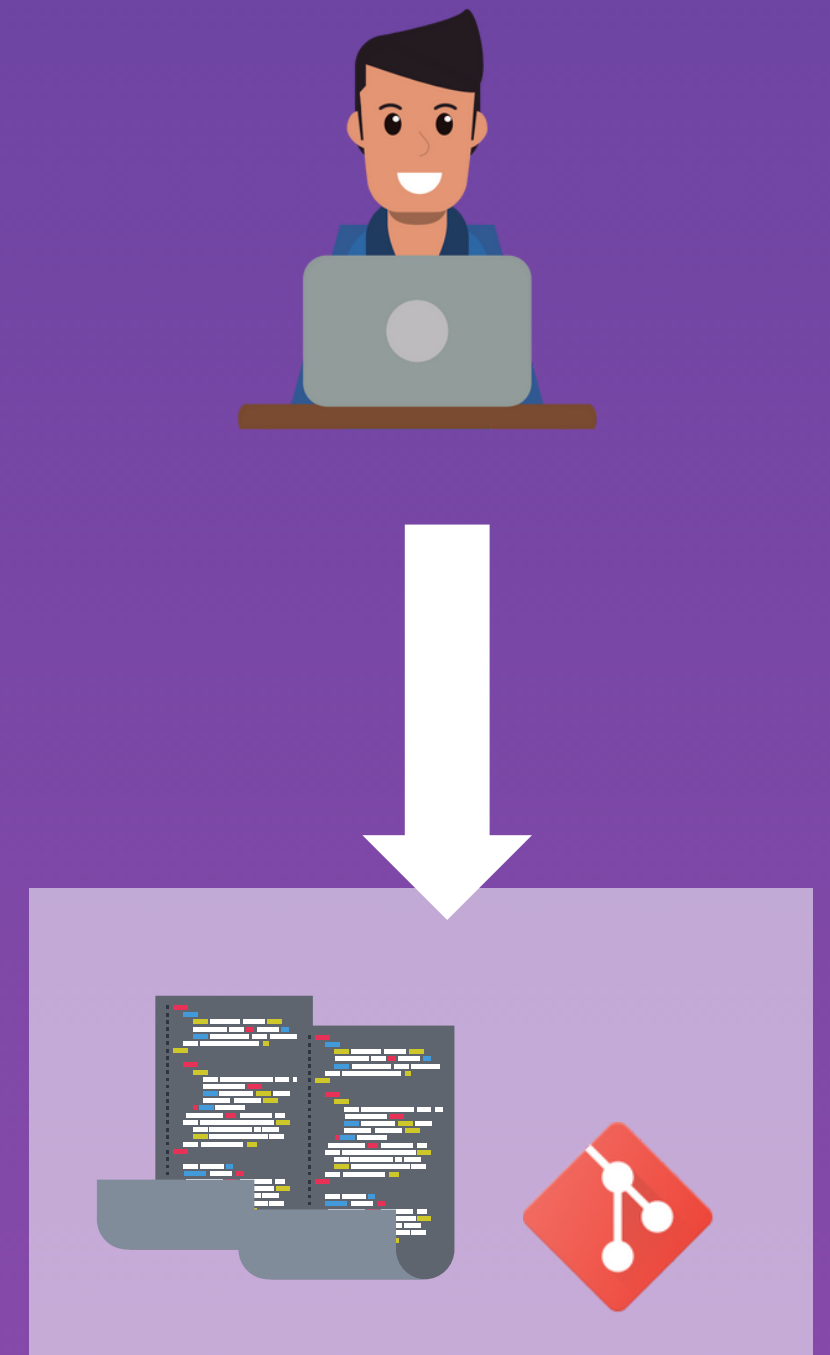
# Secret Scanning

# Prevent Secrets in Git Repositories

- As we learnt, there are different vulnerability scanning types for different purposes
- We start with **scanning our application code for leaked secrets.**

## Why? Even private repositories don't have adequate protection

- ❌ Source code is made to be duplicated and distributed
- ❌ It's a leaky asset, you never know where it is going to end up.  
Could be cloned to a compromised server
- ❌ It just takes one compromised developer account to compromise all the secrets they have access to



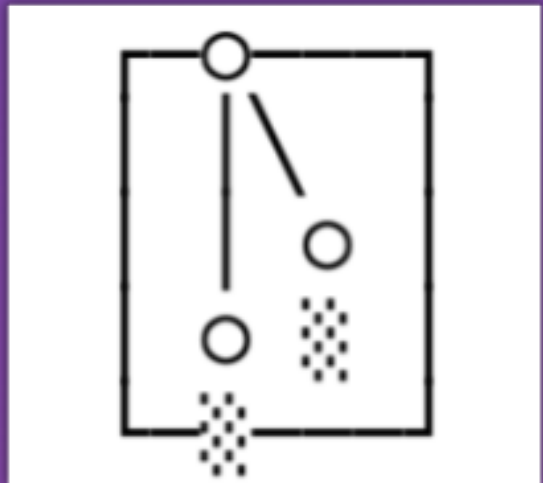


# Secret Scanning Tool - GitLeaks



## What are Secret Scanning Tools?

- Tools that scan the source code and **detect hard-coded secrets**
- There are many different such tools available
- Choose based on what you are trying to achieve and find a widely, validated tool that does exactly that

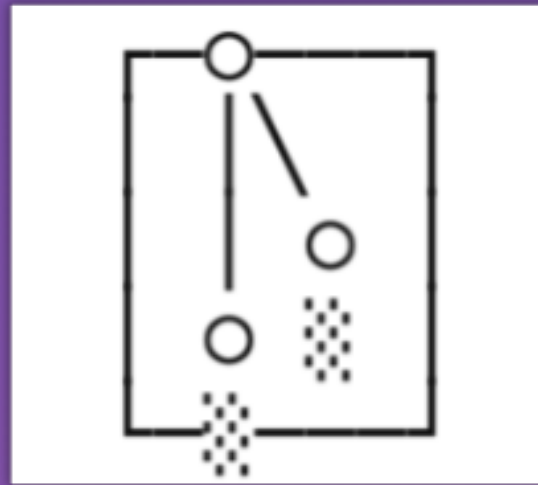


GitLeaks

## GitLeaks Secret Scanning Tool

- One such tool is GitLeaks
- A fast, light-weight and open-source secret scanner for Git repositories
- Detects over 160 secret types, new types added all the time
- GitLeaks parses the output of a `git log -p` command

# How to get started



GitLeaks

1 - Install it locally

2 - Run it against your application

3 - Integrate it into the CI pipeline

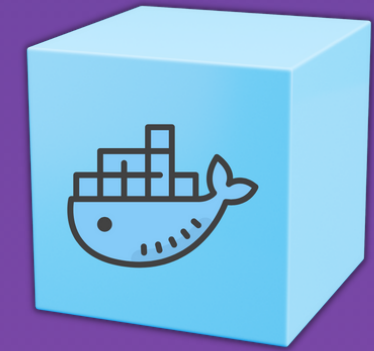
# How to get started

## 1 - Install it locally

Install directly on your OS

- ✗ Different for each operating system
- ✗ May change in the future

Use Docker Image




- ✓ Use a ready GitLeaks image and run as a Docker container
- ✓ Installation steps the same, regardless of your operating system

# How to get started



## 2 - Run it locally against your application

```
% docker run -v ${path_to_host_folder_to_scan}:/path zricethezav/gitleaks:latest detect --source="/path"
```



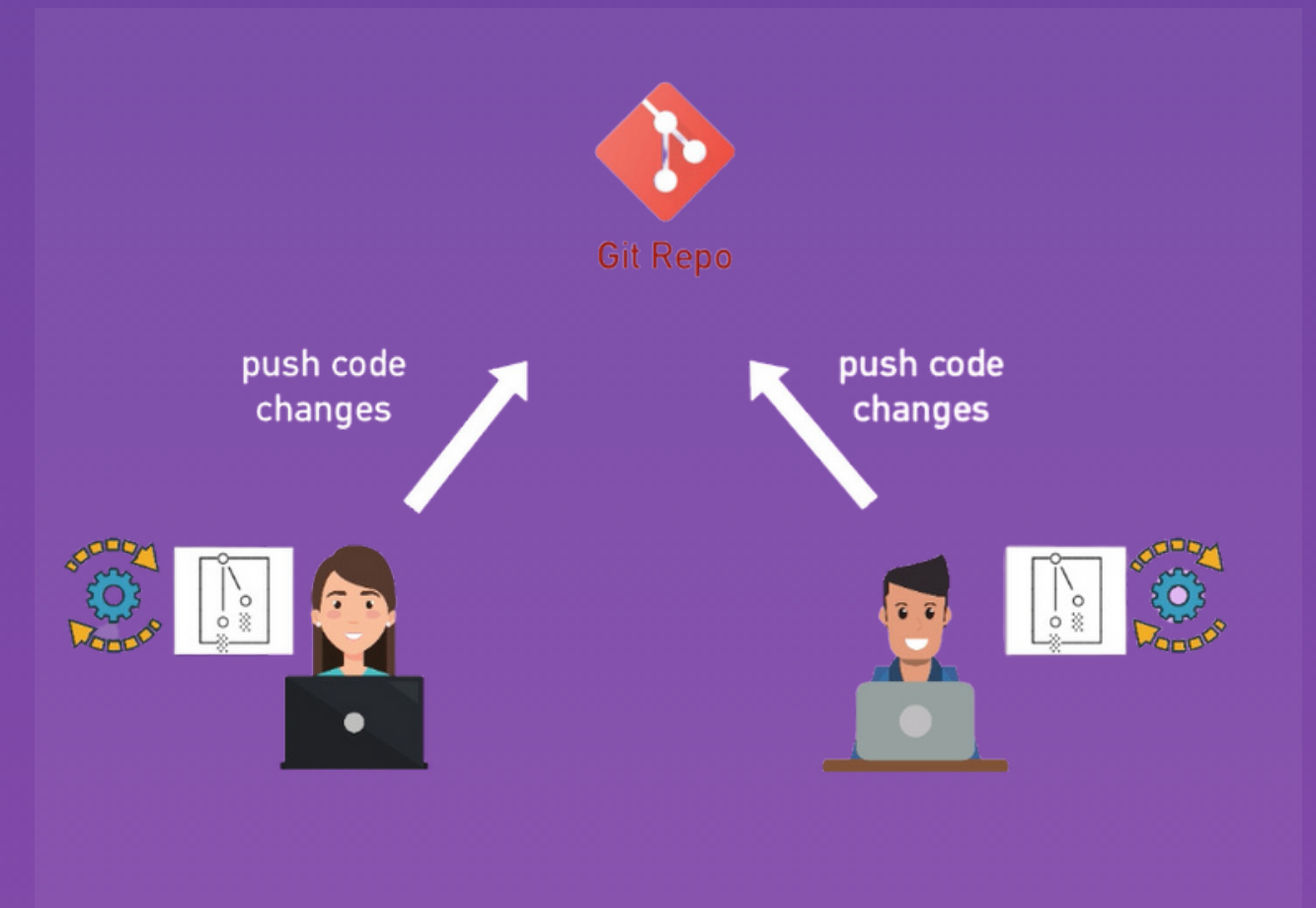
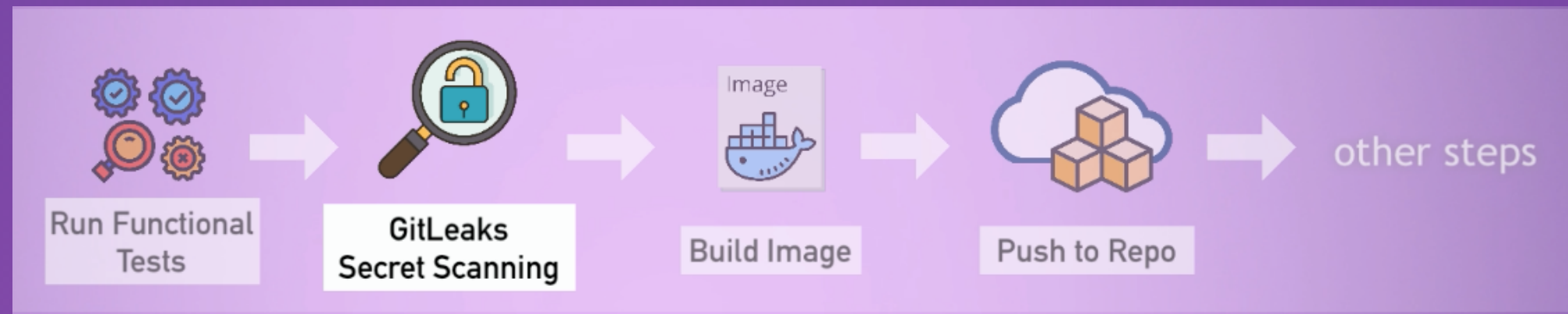
```
gitleaks
```

6:32PM **INF** 1 commits scanned.  
6:32PM **INF** scan completed in 2.96s  
6:32PM **WRN** leaks found: 9

# How to get started



## 3 - Integrate it into the CI pipeline



✓ We want to automate the execution

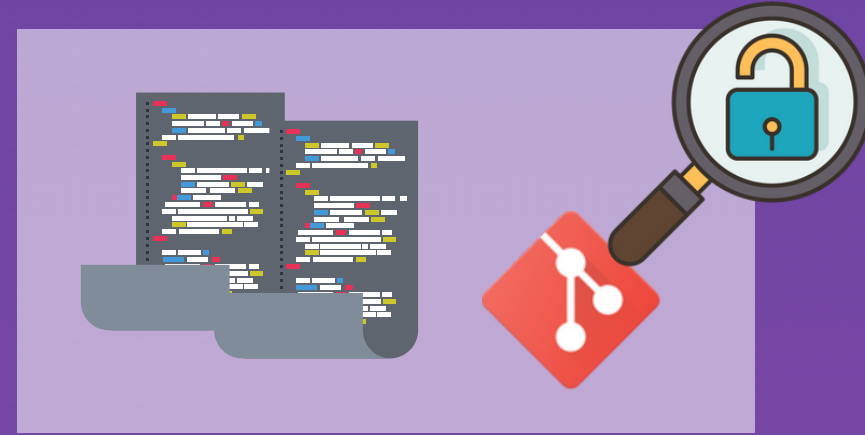
✓ First step in creating a DevSecOps pipeline and shifting security left

✗ Manual execution is not reliable

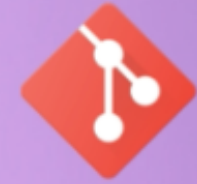
# Pre-Commit Hook



# Why Pre-Commit Hook?



Git repository



Git Repo

commit 238d-3d3ij-214

commit 49dj3-4po2j-19a

commit 39du5-5kf93-99j

commit 5gh7-973hf-9k8



Even though sensitive data is being removed, the **history of the repository remains intact**

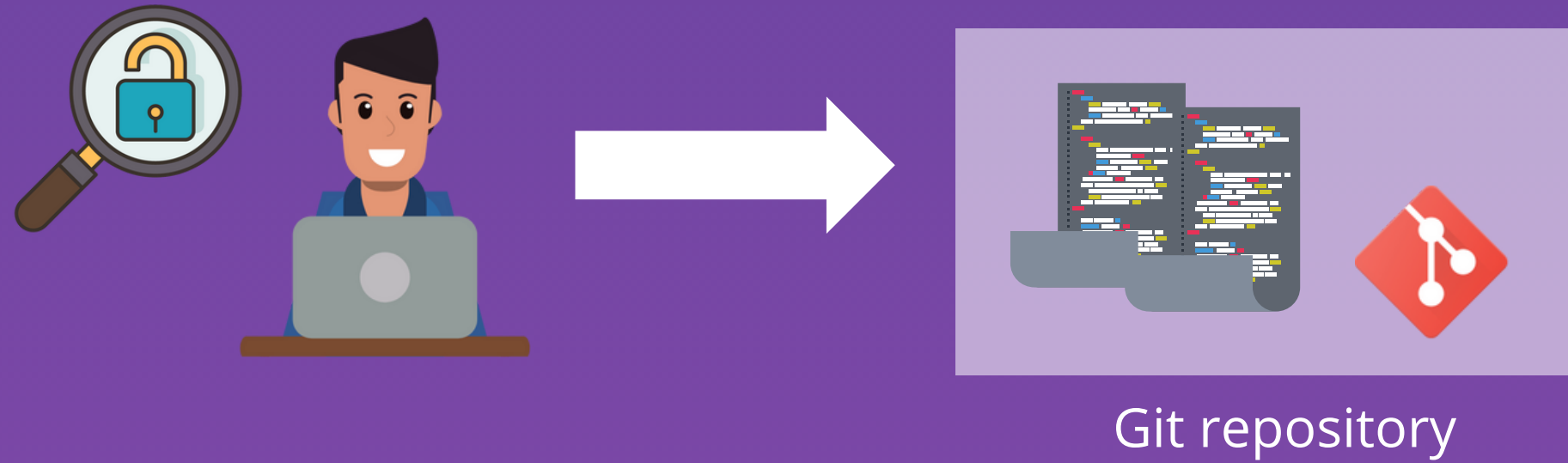


So while GitLeaks checks the code for leaked secrets once a commit is pushed, it's actually too late



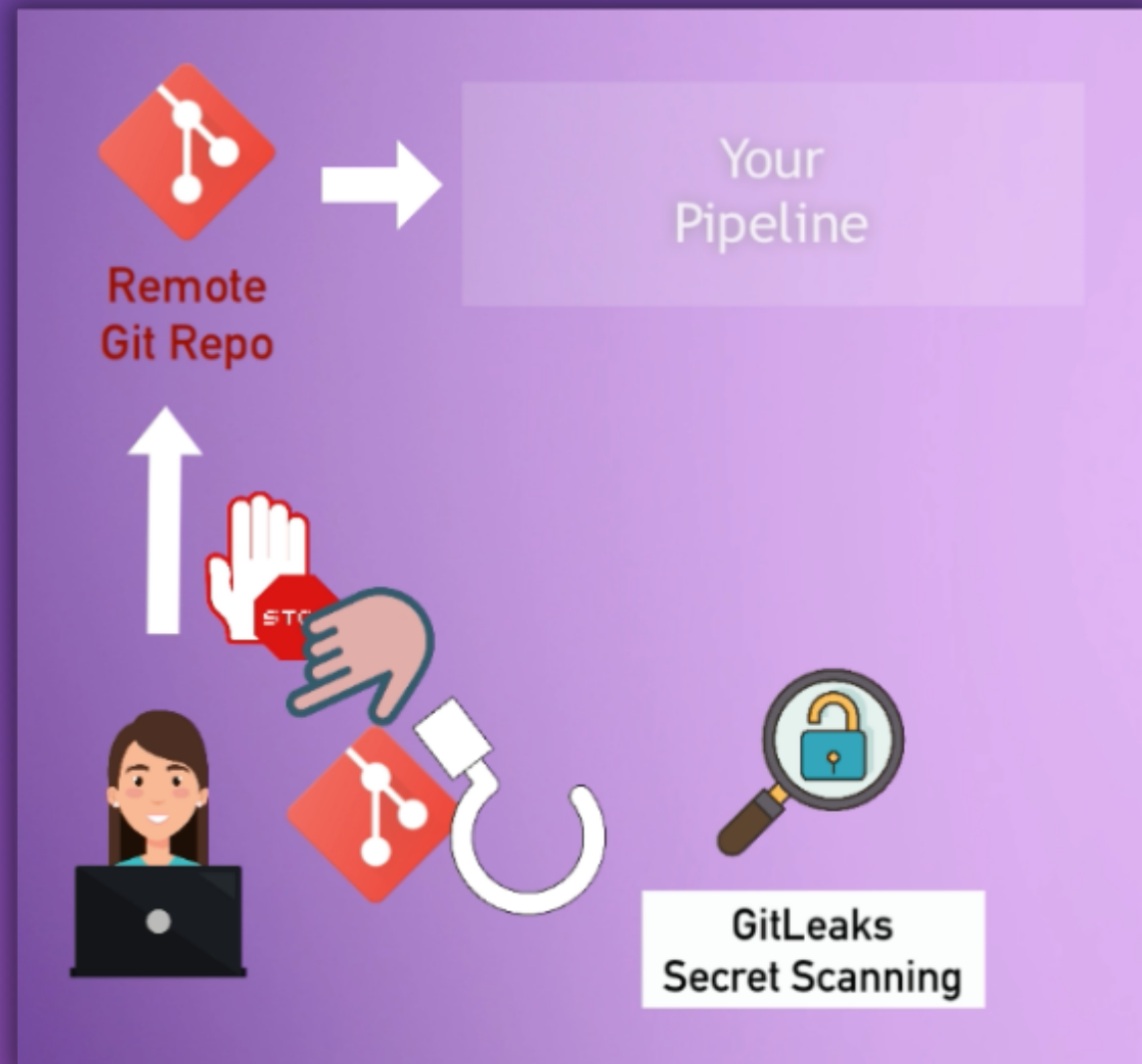
We could change the password or revert Git history, but that is not an ideal solution

# Why Pre-Commit Hook?



- ✓ Automatically run scan **before code** is pushed to remote Git repository
- ✓ With this we prevent any hard-coded secrets in the Git repository

# What is a Pre-Commit Hook?



## Git Hooks

- Git Hooks is a Git functionality
- It's a way to fire off custom scripts when certain important actions occur
- There are different types of hooks:
  - pre-commit
  - pre-push
  - pre-rebase

## "Pre-Commit" Hook

- Fires when you are about to commit your changes

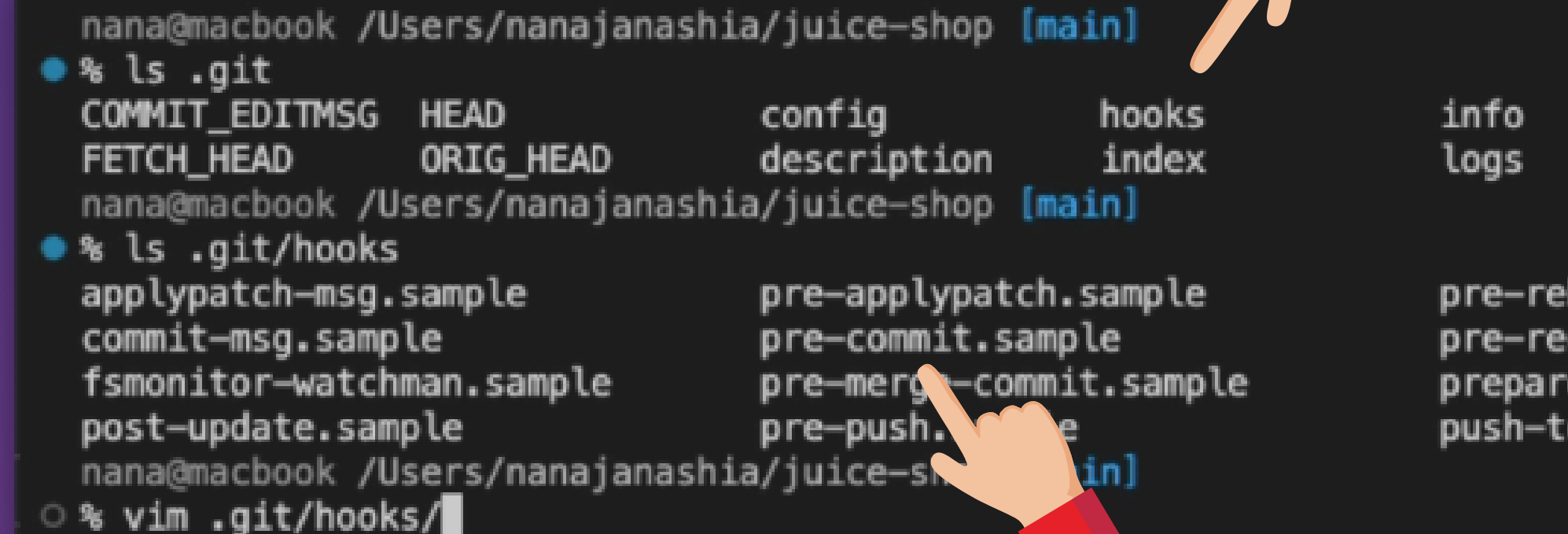
# How to configure a Pre-Commit Hook?

## Git Hooks Folder

- The hooks are all stored in the hooks sub directory of the Git folder: **.git/hooks**
- In this folder you can already view samples for different hooks

## Configure Pre-Commit Hook

- You can turn them into actual scripts that get executed
- File should be called "pre-commit"
- And in this file you just run the same command as executed locally (shell script)
- Pre-commit script must be made executable
- Now, every time you do "*git commit*", it gets executed



A terminal window showing the contents of the .git directory and the .git/hooks directory. The first command is `ls .git`, which lists `COMMIT_EDITMSG`, `HEAD`, `config`, `hooks`, `info`, and `logs`. The second command is `ls .git/hooks`, which lists several sample hook files: `applypatch-msg.sample`, `commit-msg.sample`, `fsmonitor-watchman.sample`, `post-update.sample`, `pre-applypatch.sample`, `pre-commit.sample`, `pre-merge-commit.sample`, `pre-push.sample`, and `pre-receive.sample`. A hand icon points to the `pre-commit.sample` file. The third command is `vim .git/hooks/`.

```
nana@macbook /Users/nanajanashia/juice-shop [main]
• % ls .git
COMMIT_EDITMSG  HEAD          config        hooks         info
FETCH_HEAD     ORIG_HEAD     description   index         logs
nana@macbook /Users/nanajanashia/juice-shop [main]
• % ls .git/hooks
applypatch-msg.sample  pre-applypatch.sample  pre-receive.sample
commit-msg.sample      pre-commit.sample      pre-receive.sample
fsmonitor-watchman.sample  pre-merge-commit.sample  prepare-commit-msg.sample
post-update.sample       pre-push.sample         push-to-checkout.sample
nana@macbook /Users/nanajanashia/juice-shop [main]
○ % vim .git/hooks/
```

```
docker pull zricethezav/gitleaks:latest
export path_to_host_folder_to_scan=/Users/nanajanashia/juice-shop
docker run -v ${path_to_host_folder_to_scan}:/path zricethezav/gitleaks:latest detect --source="/path" --verbose
```

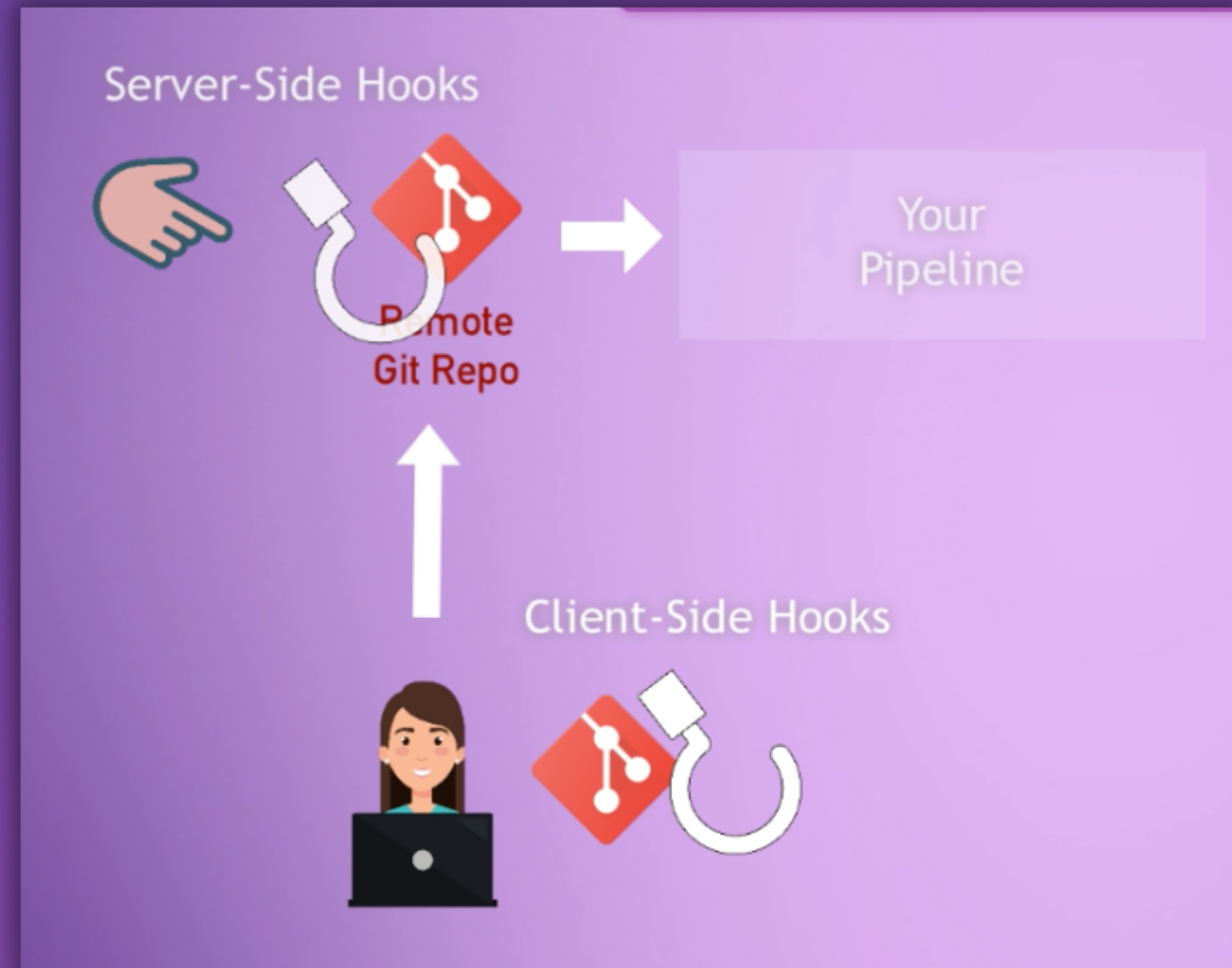
# Shift Security Left



**Most "shift left"**

- We learnt we want to shift security as much left as possible to detect security issues as early as possible
- Besides planning phase and threat modeling before development, pre-commit hook is one of the furthest shifts left possible

# Client-Side vs Server-Side Hooks



## Server-side Hooks

- Scripts that run **before and after pushes to the server**, so you can **enforce this server-side**
- Different server-side hooks available:
  - pre-receive
  - update
  - post-receive



# False Positives and Fixing Security Vulnerabilities

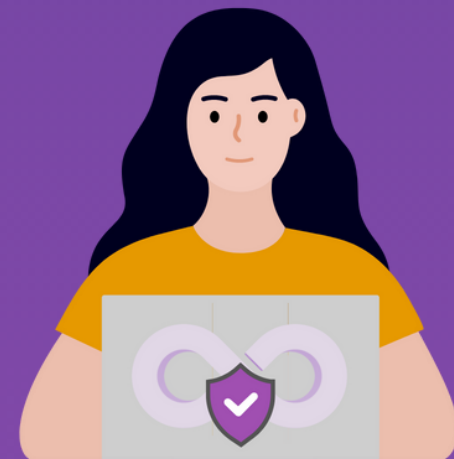


# Fixing Security Issues

Next step would be to fix the detected security vulnerabilities

## Recap of roles and responsibilities

### DevSecOps Engineer



- Help team to identify any issues and bring issues to the attention of the developers

### Developers



- Developers will see failed CI/CD Pipeline and vulnerabilities found by the tool
- Developers and other project team members will fix it



# False Positives

## What is a "false positive"?

- When a scanning tool incorrectly flags a security vulnerability

## What is a "false negative"?

- Opposite error, where the test result incorrectly indicates the absence of a vulnerability when it's actually present

### Many reasons for this

- ✗ Tools are not mature enough
- ✗ Complexity of the system
- ✗ Limited context awareness
- ✗ Overly Strict Policies
- ✗ Outdated vulnerability data



# False Positives

## How to deal with false positives?

- ✓ As part of the set up we need to **tweak our security tools to mitigate false positives**
- ✓ Configure those tools properly
- ✓ Keep them updated with latest rules
- ✓ Regularly review and refine results based on real-world context



## Why it's important

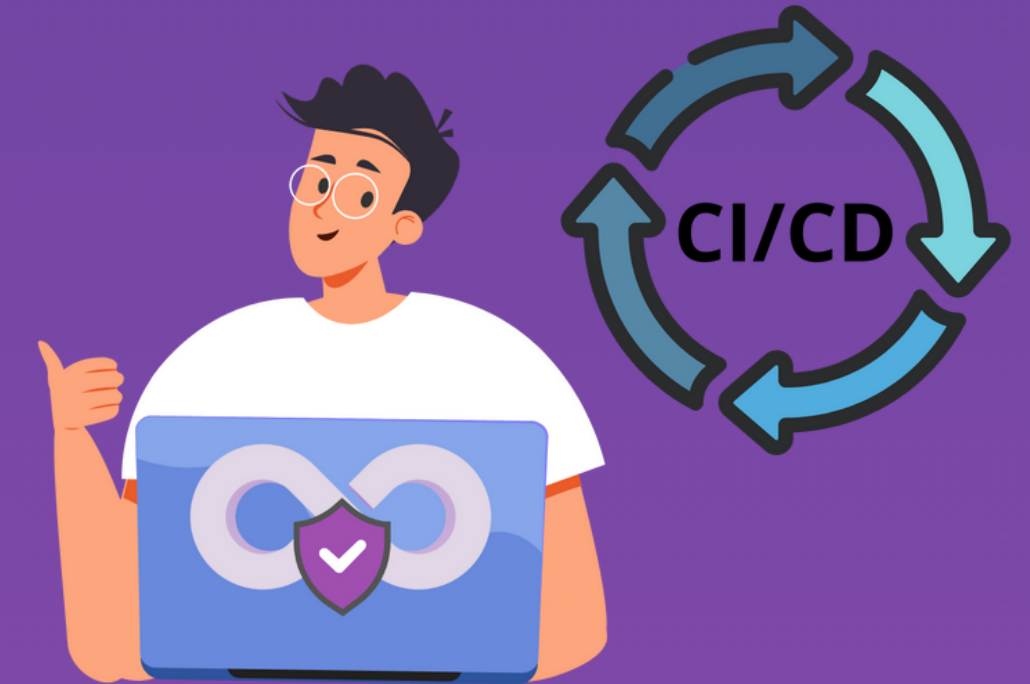
- ✗ Slows down the team
- ✗ Distracts them from real risks
- ✗ Makes the security scan almost useless



# Handling False Positives

## 1 - We don't fail the CI build

- You will almost always face some false positives
- But if there are so many that it distracts the team you need to take steps to reduce them
- We will integrate it into the pipeline to **optimize it - step by step**
- But we don't interrupt the developer workflow until we have matured the tool



## 2 - Adjusting the tool configuration

- Start adjusting tool configuration, custom configuration that is application specific
- With the goal of producing less false positives over time

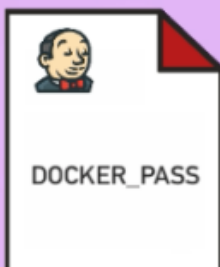
# Example Security Vulnerability Fix

```
build_image:
  stage: build
  image: docker:24
  services:
    - docker:24-dind
  variables:
    DOCKER_PASS: 6HGGhKTzi!v^YSPq
    DOCKER_USER: techworldwithnana
  before_script:
    - echo $DOCKER_PASS | docker login -u $DOCKER_USER --password-stdin
  script:
    - docker build -t $IMAGE_NAME:$IMAGE_TAG .
    - docker push $IMAGE_NAME:$IMAGE_TAG
```

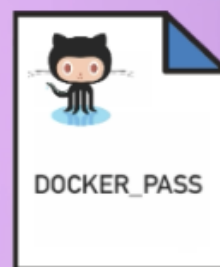


Bad Security Practice: **Using hard-coded credentials in the pipeline configuration**

DOCKER\_PASS:  
s crt-pwd



DOCKER\_PASS:  
s crt-pwd



DOCKER\_PASS:  
s crt-pwd



**No hard-coded secrets in Pipeline Configuration**



We achieve that, for example, by extracting the password and using variables



# Example Security Vulnerability Fix

Define the environment variable in the CI Platform



↑ Key	Value	Attributes	Environments
DOCKER_PASS	*****	<span>Protected</span> <span>Expanded</span>	All (default)
DOCKER_USER	*****	<span>Protected</span> <span>Expanded</span>	All (default)

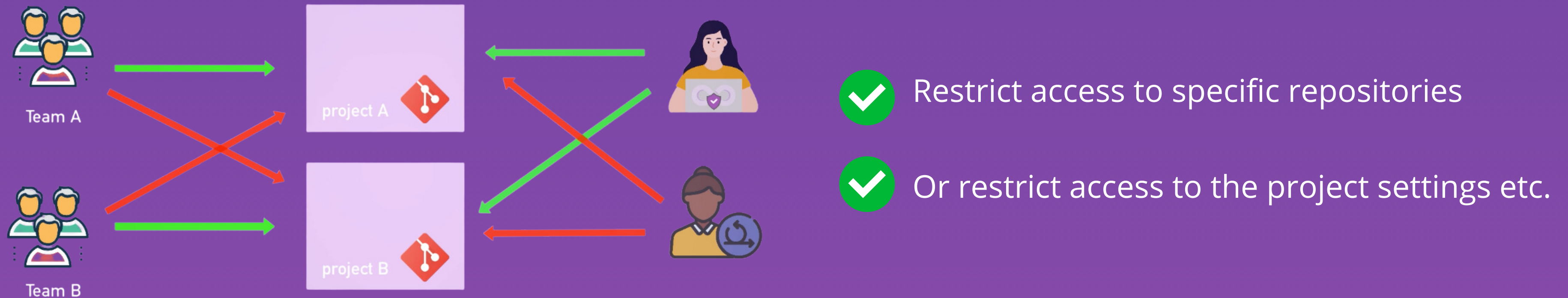
Use the variable in the Pipeline Configuration



```
49
50 build_image:
51   stage: build
52   image: docker:24
53   services:
54   | - docker:24-dind
55   variables:
56   | DOCKER_PASS: $DOCKER_PASS
57   | DOCKER_USER: $DOCKER_USER
58   before_script:
59   | - echo $DOCKER_PASS | docker login -u $DOCKER_USER --password-stdin
60   script:
61   | - docker build -t $IMAGE_NAME:$IMAGE_TAG .
62   | - docker push $IMAGE_NAME:$IMAGE_TAG
63
```

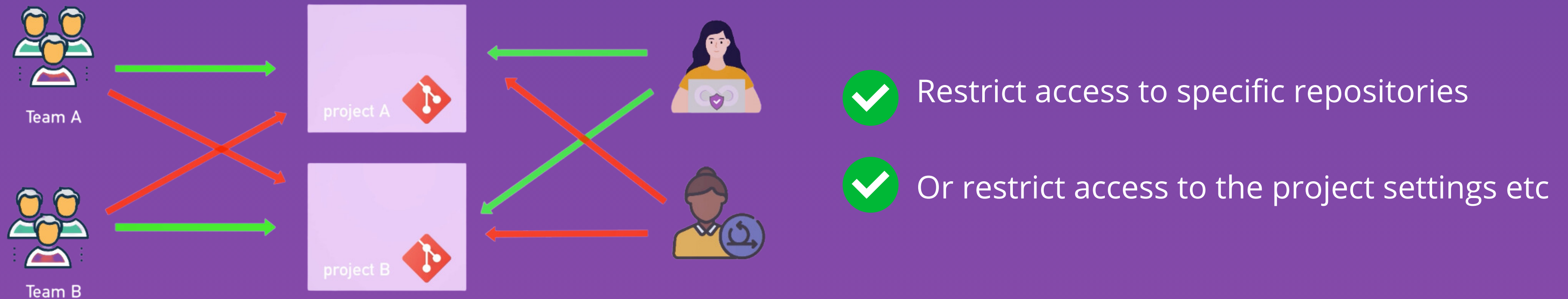
# Access Permissions on Git Repositories

- CI platforms have granular permission systems
- So you can and should give access with least privilege principle



# Access Permissions on Git Repositories

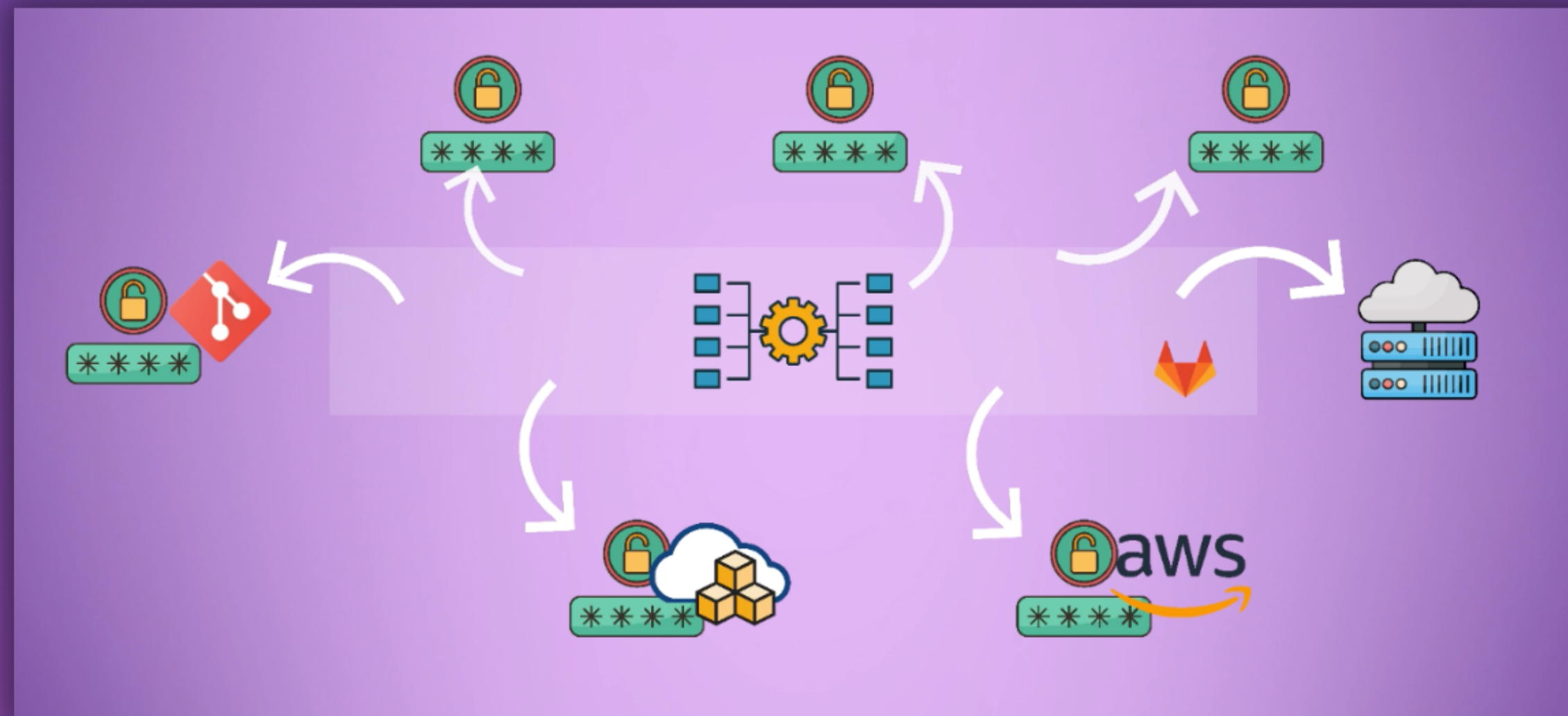
- CI platforms have granular permission system
- So you can and should give access with least privilege principle



# Secrets across whole CI/CD



- CI/CD platform interacts with different external systems, platforms and services, as it is the **orchestrator** of the whole release pipeline
- So you have **secrets throughout the pipeline**





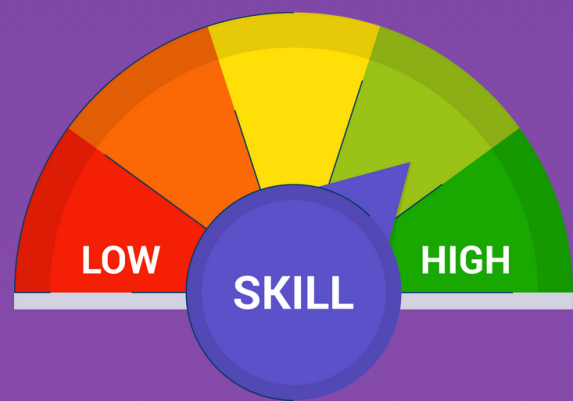
# Integrate **SAST** Scans

in Release Pipeline

# Introduction to SAST



- Detecting leaked secrets is just one part of vulnerability scanning
- The code itself can be written in a way that **allows for exploitation**



- Source code with **good coding practices prevents exploitation**
- It depends on the ability of developers to implement secure code
- And also - a developers' focus is about the speed of developing new features
- **Code quality and security is often secondary**





# Introduction to SAST



- So application developers have to learn writing code with security best practices
- Security tools giving **feedback to developer team helps in that learning process**
- And eventually this will lead to less security findings over time

# Introduction to SAST



DDoS



SQL Injection



Phishing



Weak Passwords



XSS



We learnt about the **different types of security vulnerabilities** the code can have

SAST Tools help in **finding those vulnerabilities automatically**

# What is SAST?



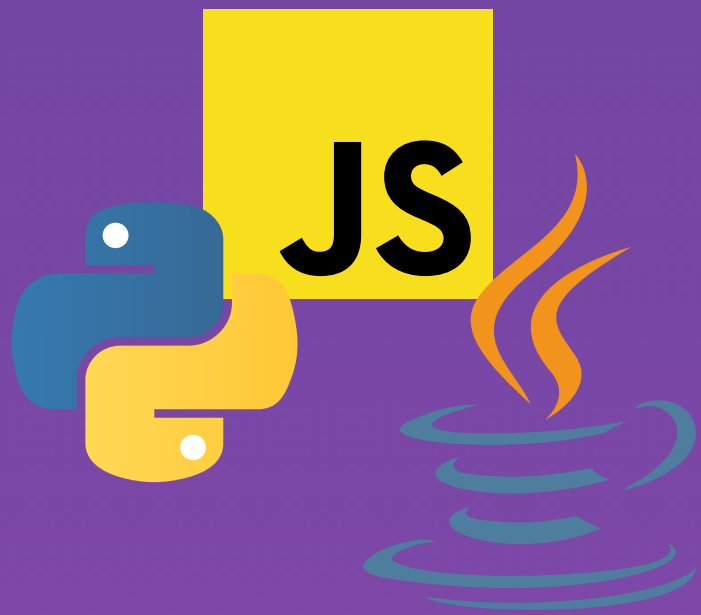
SAST

## Static Application Security Testing

- **Static** code analysis (app is not running)
- Identifies security vulnerabilities in app's source code, configuration files etc. using SAST tools

# SAST Tools

## Different SAST tools **based on programming language**



- They are focused and specialized on finding vulnerabilities in only that programming language
- Each language has its own syntax, semantics and potential security pitfalls, which is why specialized SAST tools are created
- Language-specific vulnerabilities



## SAST Tools that can scan **multiple languages**

- Tools that can understand and test multiple programming languages
- Like Semgrep, Snyk Code, SonarQube etc.

## Open Source and Proprietary Tools available

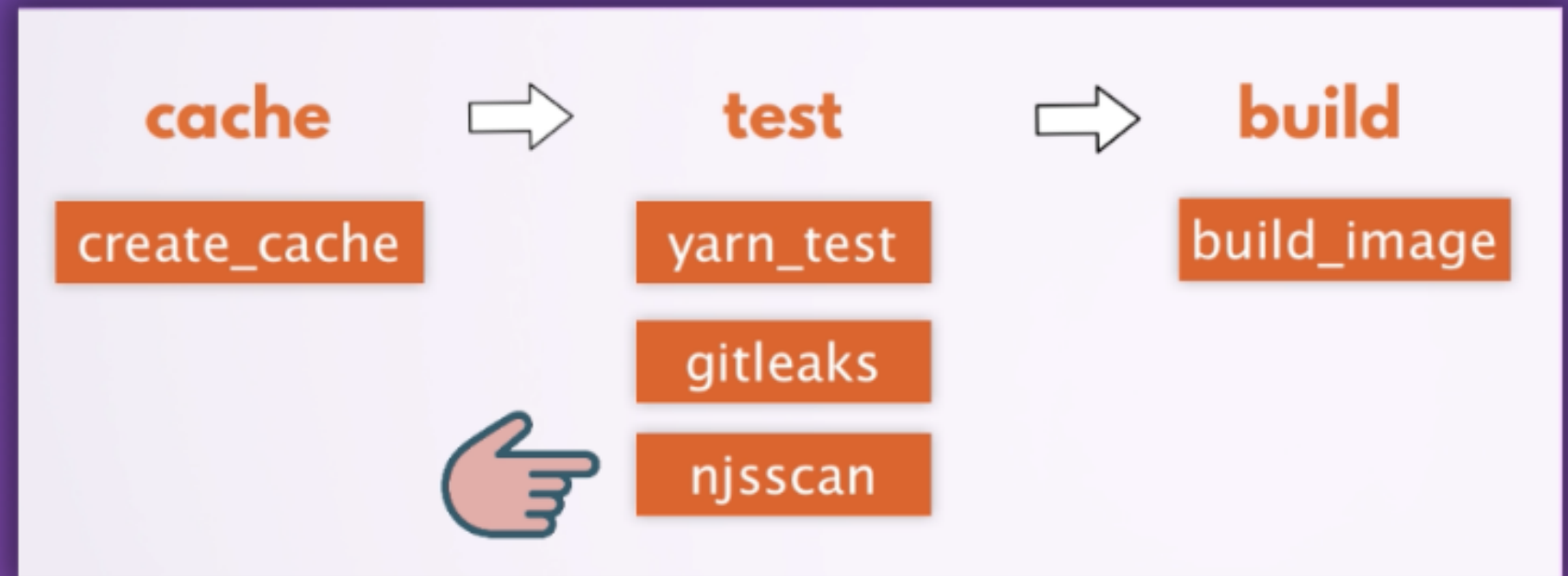
# How to use a SAST Tool

- Pick tool that fits your requirements
- For example you can use **njsscan** for JavaScript applications
- All tests (functional and security) can run in parallel, as there are no dependencies between them
- Use Docker Image for that tool
- Execute command in pipeline configuration



**Always reference latest official docs on how to use it**

- Often there are example snippets, even for different CI/CD tools

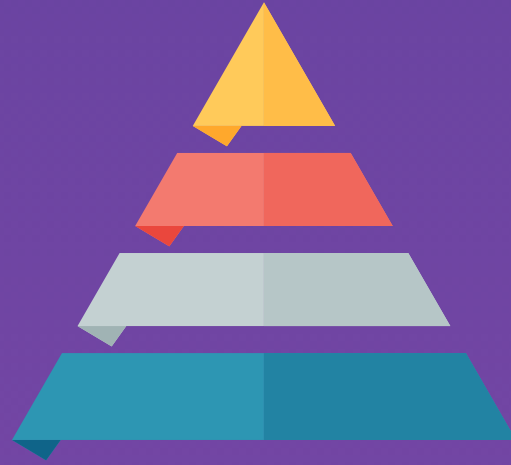


## Gitlab CI/CD

Add the following to the file `.gitlab-ci.yml`.

```
stages:
  - test
njsscan:
  image: python
  before_script:
    - pip3 install --upgrade njsscan
  script:
    - njsscan .
```





# Different levels of Severity

- Security tools' results often contain “severity” information
- Refers to the **degree of impact or potential harm** that a security vulnerability could have on a system, app or data
- Often **categorized** into:
  - Critical or High Severity
  - Medium Severity
  - Low or Warning
  - Informational
- **Helps us understand the level of risk** posed by a particular vulnerability and determine how urgently it needs to be addressed
- Improvement to reduce unnecessary findings: We don't fail the build on Info and Warning Risks



```
154 [notice] To update, run: pip install --upgrade pi
155 $ njsscan .
156 - Pattern Match 
157 - Semantic Grep 20
158 njsscan: v0.3.4 | Ajin Abraham | opensecurity.in
159
160 RULE ID      node_md5
161
162 CWE          CWE-327: Use of a Broken or Risky
163
164 OWASP-WEB    A9: Using Components with Known V
165
166 DESCRIPTION  MD5 is a a weak hash which is kno
167
168 SEVERITY      WARNING
169
170 FILES
171   File      Gruntfile.js
172   Match Position  19 - 43
173   Line Number(s)  73
174   Match String    const md5 = cr
175
176
177
178
179
180
181 Cleaning up project directory and file based vari
182
183 Job succeeded
```

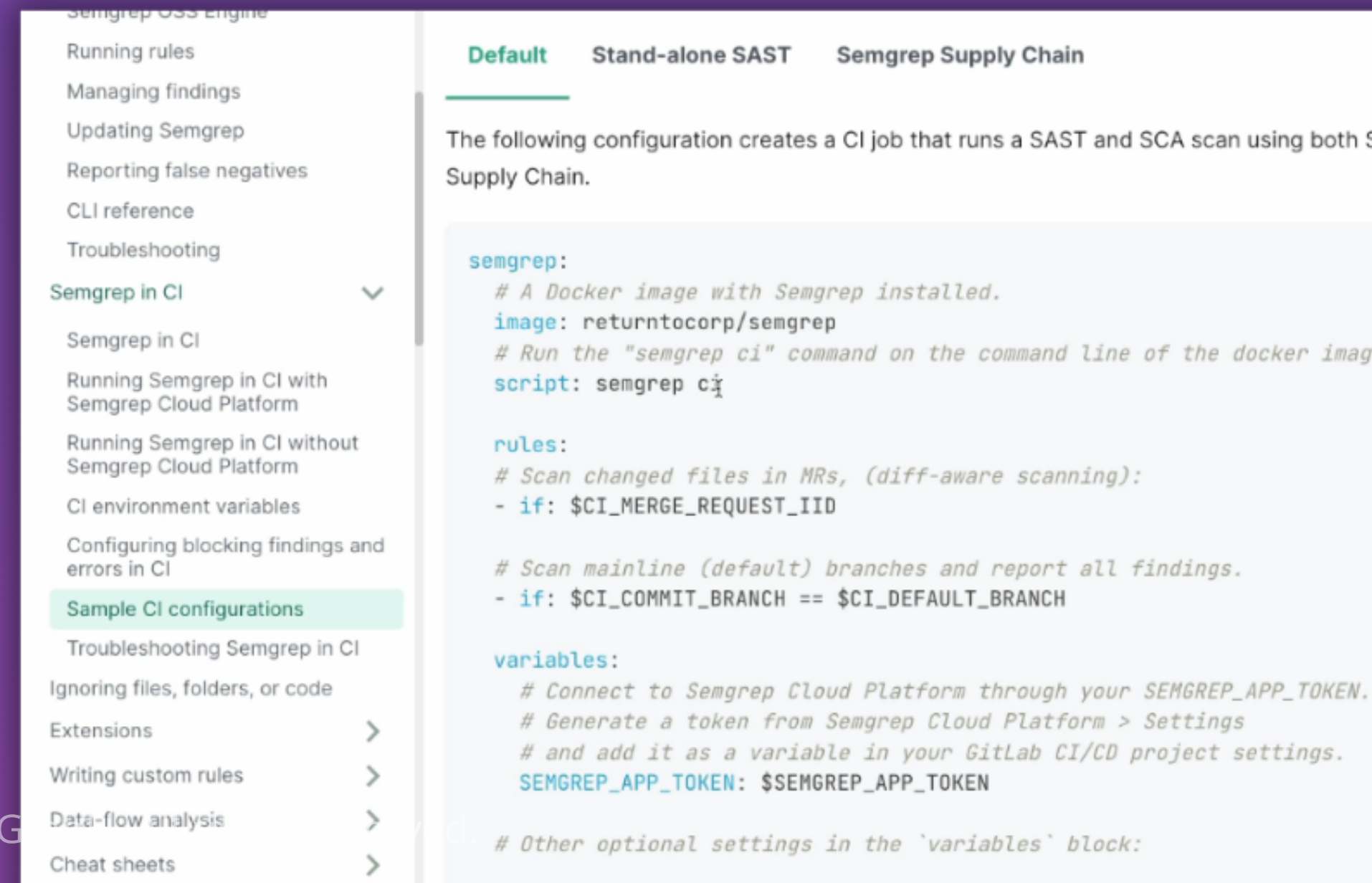
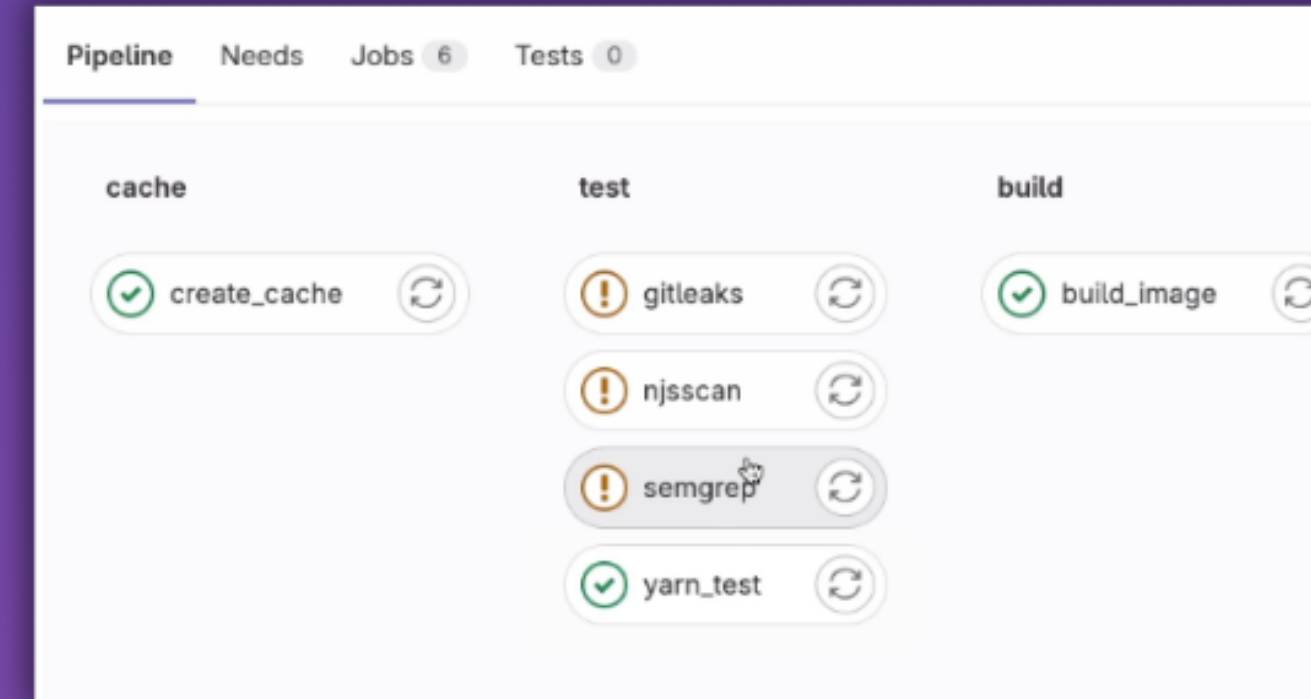


# Combine multiple security tools

- To achieve full coverage, you may need to combine multiple tools



- Free, open-source SAST tool
- It supports multiple languages like C#, Go, Java, JavaScript, Python, PHP, Ruby, Scala
- A powerful tool
- Again - use the official Docker Image
- Use command in pipeline code to execute semgrep scanning



# Our DevSecOps Pipeline until now

- ✓ Get insight about state of application security

