Project Report – Group 10

Michael Chan, Rémi Lejeune, Jan van der Meulen, Shayan Ramezani

1 RELEASE PIPELINE DOCUMENTATION

RELEASE PIPELINE DOCUMENTATION FOR `LIB-ML` PYTHON PACKAGE

This documentation provides a detailed description of the release pipeline used for publishing the `lib-ml` Python package to PyPI. The goal is to help new team members understand the pipeline steps, the tools used, and the flow of data and artifacts throughout the process. For a illustrative overview, see Figure 1

1.1 Pipeline Overview

The release pipeline is triggered when a pull request (PR) is closed. It consists of two main jobs:

(1) **Test**: Runs tests on multiple environments to ensure the code is stable and functional.
(2) **Bump Version and Publish**: Bumps the package version, updates files, and publishes the package to PyPI.

1.2 Pipeline Steps

1.2.1 *Testing (`test` job).*

Purpose. Ensure the package works correctly in different environments and Python versions.

Implementation.

- **Triggered**: When a pull request is merged.
- **Runs on**: Multiple OS and Python versions specified in a matrix.
- **Timeout**: 10 minutes.

Steps.

(1) **Checkout code**
  - Uses `actions/checkout@v4`.
  - Fetches the code from the merged pull request.
(2) **Set up Python**
  - Uses `actions/setup-python@v5`.
  - Sets up the specified Python version from the matrix.
(3) **Install Poetry**
  - Installs Poetry using `pipx install poetry` or `pip install poetry`.
(4) **Install dependencies**
  - Runs `poetry install -with dev` to install development dependencies.
(5) **Run tests**
  - Executes `poetry run pytest` to run the test suite.

1.2.2 *Bump Version and Publish (`bump_version_and_publish` job).*

Purpose. Bump the package version, update the version in relevant files, and publish the package to PyPI.

Implementation.

- **Triggered**: After the successful completion of the `test` job.
- **Runs on**: `ubuntu-latest`.

Steps. Steps (1) to (4) are repeated before moving on to the next steps, which are:

(1) **Bump version and push tag**
  - Uses `anothrNick/github-tag-action@1.67.0`.
  - Bumps the version and pushes a new tag to the repository.
  - Environment variables used:
    – `GITHUB_TOKEN`: Token for accessing GitHub.
    – `DEFAULT_BUMP`: Default version bump type (`patch`).
    – `TAG_CONTEXT`: Context for tagging (`branch`).
    – `WITH_V`: Whether to include 'v' in the version tag (false).
    – `PRERELEASE`: Pre-release flag (false).
(2) **Update files with new version**
  - Runs `poetry run bump-my-version replace -config-file pyproject.toml -new-version $(git describe -tags -abbrev=0)`.
  - Updates the version in the `pyproject.toml` file.
(3) **Build and publish to PyPI**
  - Runs `poetry publish -build -u __token__ -p $ secrets.PYPI_API_KEY` to build and publish the package to PyPI.
  - Uses `PYPI_API_KEY` secret for authentication.

1.3 Artifacts and Data Flow

(1) **Source Code**: Checked out from the merged pull request.
(2) **Python Environment**: Set up using specified versions from the matrix.
(3) **Dependencies**: Installed using Poetry.
(4) **Test Results**: Determines if the publish job should proceed.
(5) **Version Tag**: Created and pushed to the repository.
(6) **Updated `pyproject.toml`**: Contains the new version.
(7) **Published Package**: The final artifact, published to PyPI.

1.4 Tools Used

- **GitHub Actions**: CI/CD platform for running workflows.
- **Poetry**: Dependency management and packaging tool.
- **pytest**: Testing framework.
- **anothrNick/github-tag-action**: Action for tagging releases.

RELEASE PIPELINE DOCUMENTATION FOR `MODEL-SERVICE` CONTAINER IMAGE

This documentation provides a detailed description of the release pipeline used for publishing the `model-service` container image. Also here, the goal is to help new team members understand the pipeline steps, the tools used, and the flow of data and artifacts throughout the process. For a illustrative overview, see Figure 2.

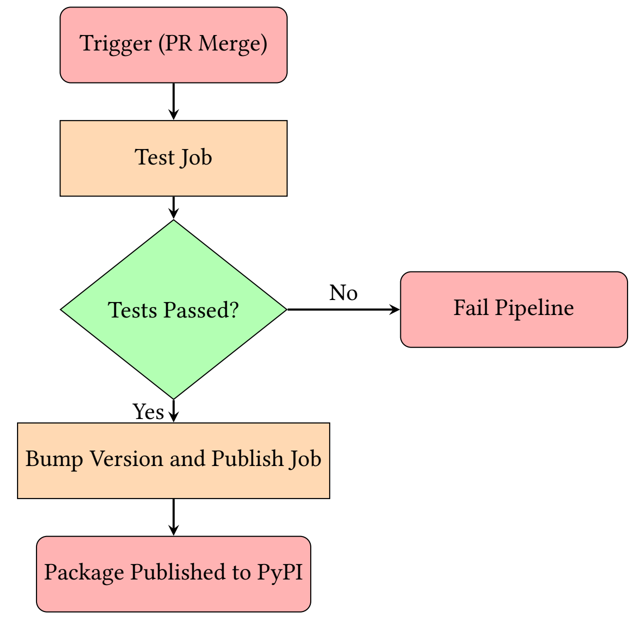


Figure 1: Flowchart of `lib-ml` Python Package Release Pipeline

1.5 Pipeline Overview

The release pipeline is triggered when a new tag matching the pattern `v[0-9]+.[0-9]+.[0-9]+` is pushed to the repository. It consists of a single job:

(1) **Build**: Builds the Docker image and pushes it to the GitHub Container Registry (GHCR).

1.6 Pipeline Steps

1.6.1 *Build (`build` job).*

Purpose. Build the Docker image for the `model-service` and push it to the GitHub Container Registry (GHCR).

Implementation.

- **Triggered**: When a tag matching the pattern `v[0-9]+.[0-9]+.[0-9]+` is pushed.
- **Runs on**: `ubuntu-22.04`.

Steps.

(1) **Checkout code**
   - Uses `actions/checkout@v4`.
   - Checks out the code from the repository.

(2) **Parse version info from tag**
   - Runs a shell script to parse the version information from the tag.
   - Extracts the major, minor, and patch version numbers.
   - Sets the parsed version numbers as environment variables.

(3) **Registry Login (ghcr.io)**
   - Logs into the GitHub Container Registry (GHCR) using the `GH_TOKEN` secret.
   - Uses the GitHub Actions context for authentication.

(4) **Build and Push Docker Image**
   - Builds the Docker image using the Dockerfile in the repository.
   - Tags the image with:
     – Full version (e.g., `v1.2.3`).
     – Major and minor version with `.latest` suffix (e.g., `1.2.latest`).
     – Major version with `.latest` suffix (e.g., `1.latest`).
     – `latest` tag.
   - Pushes all tagged images to the GitHub Container Registry.

1.7 Artifacts and Data Flow

(1) **Source Code**: Checked out from the repository.
(2) **Docker Image**: Built from the source code.
(3) **Version Tags**: Parsed from the pushed tag and used to tag the Docker image.
(4) **Published Image**: The final artifact, pushed to the GitHub Container Registry.

1.8 Tools Used

- **GitHub Actions**: CI/CD platform for running workflows.
- **Docker**: Containerization platform for building and pushing images.
- **GitHub Container Registry (GHCR)**: Registry for storing and managing Docker container images.

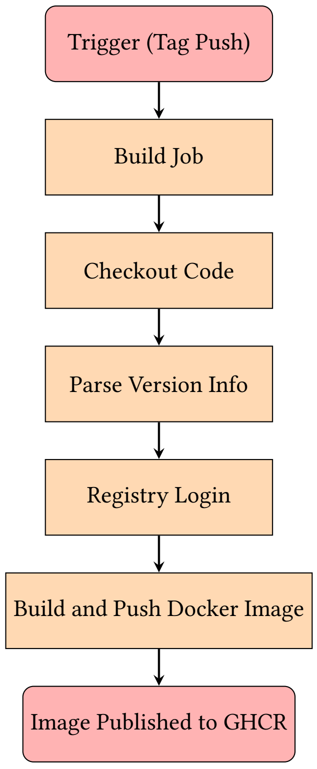


Figure 2: Flowchart of `model-service` Container Image Release Pipeline

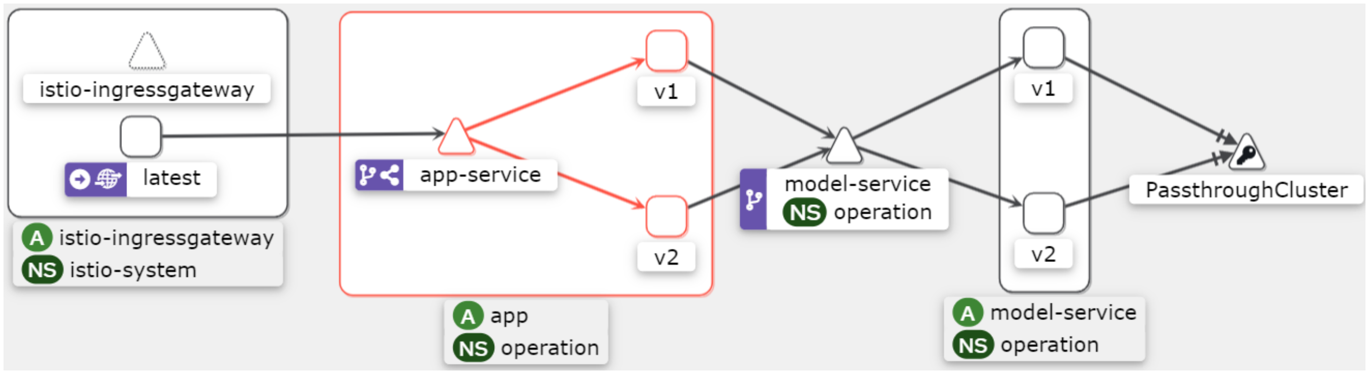


Figure 3: Deployment of operation

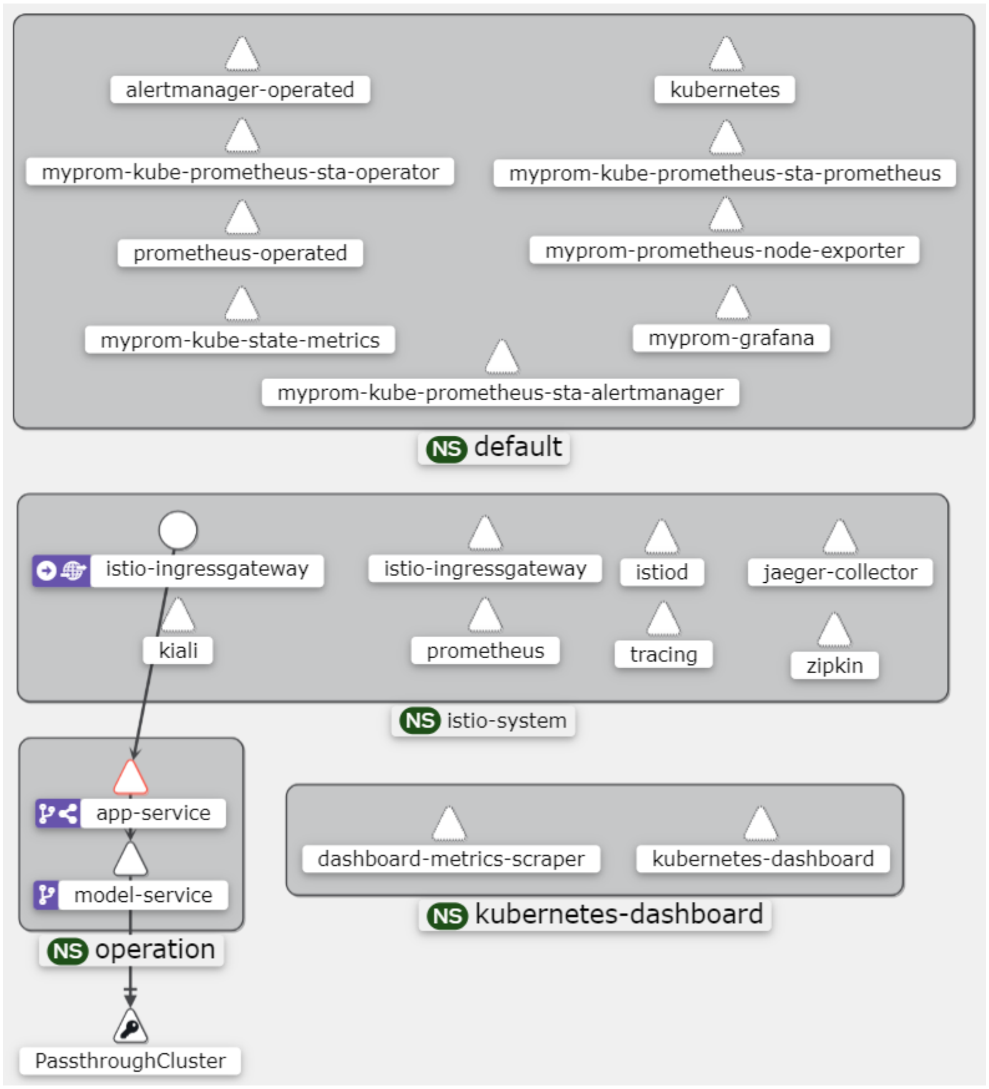


Figure 4: Entire deployment cluster

2 DEPLOYMENT DOCUMENTATION

This section covers the deployment and data flow of the project. The project is deployed on a minikube cluster, this allows for scaling of the project as well as stability through backup deployment replicas.

2.1 Deployment structure

The deployment structure is visualized in Figure 3. It consists of two main services with each having two deployment versions for a potential canary release:

(1) **app-service**: App-service handles the front-end of the application and serves the page that the user directly interacts with. This is done through two flask deployments each running a different version. Each of these deployments consist of only 1 replica and therefore do not provide redundancy out of the box. This can be scaled up to improve availability.
(2) **model-service**: Model-service handles the back-end of the application and provides the "predict" endpoint for the app to interact with, which returns the prediction result given an url. This is done through two flask deployments hosting the model each running a different version. Each of these deployments also consist of only 1 replica.

2.2 Data flow

The Ingress gateway is provided by Istio and serves as the entry point to the application. The request is then handled by the app-service which forwards the request to one of the two deployment versions with equal probability (50/50). The user is then able to make a prediction on the app. The app then sends a post request to the model-service which forwards it to the corresponding model-service deployment version. The prediction is then directly returned through the passthrough cluster.

2.3 Full cluster deployment

The project furthermore employs various monitoring tools on the cluster, these include Prometheus and Grafana. Additionally various other dashboards can be added to the cluster such as the Minikube, Jaeger and Kiali dashboard. The full deployment of all clusters can be found in Figure 4. Prometheus scrapes the "/metrics" endpoint of the model-service deployments to keep track of various metrics. Grafana can then be connected to prometheus to provide a intuitive dashboard of the various metrics.

3 EXTENSION PROPOSAL

During the project, we experienced issues with setting up the software environment using Ansible and Vagrant. These issues became apparent as we had to do a more complex task by connecting the virtual machines and setting up the Kubernetes cluster for the virtual machines. The goal of Ansible is to provision a local software environment together with Vagrant, which enables the software to run locally on virtual machines. This should remove the *it runs on my machine* argument by providing a stable and reproducible environment. As a result, software development should speed up. However, during the development process, we noticed that using Ansible instead slowed down development due to several issues we encountered. Following our discussion of the issues, we will explore two development approaches that could be used to address them.

Issues. In this paragraph, we will discuss some issues that we experienced personally. During development, Ansible sometimes breaks despite no apparent changes in the local environment. Different inventory.cfg files might be required depending on the developer's operating system, which hurts reproducibility. Command outputs have limited verbosity or the verbosity can be needlessly complex, and playbook execution can have very long wait times[4]. Additionally, some commands that run successfully when executed directly via SSH do not work in the playbook. Configuration setup options are also limited and sometimes difficult to implement, due to Ansible using a different thread for every command. As discussed in this blog by Eric Hu, Ansible seems better for setting up applications than for configuration management (e.g. setting up a Kubernetes cluster)[3]. This was something we experienced ourselves as well, as Ansible creates a new shell when executing a command. As a result, the inexperienced user can lose environment and/or configuration settings when executing commands sequentially[5]. Even though it is certainly possible to configure a complex environment with Ansible, it might not be the best tool for our specific job.

3.1 Deployment of an In-House Cluster

One of the use cases for this setup could be to run an in-house application on a local company network. Vagrant and Chef can be used to provision this server to allow for easy scaling. Docker-compose will be used for local development. Especially the continuous deployment features of Chef can be useful in this case[2]. Even though Ansible has some strong features, especially the good security using SSH and its ease of use Chef is likely a better option here. Because Chef is a more mature technology[6], has better configuration management and better deployment features. Furthermore, because Chef works well with our current CI/CD pipelines, if a new version of the software is created it can immediately be used in the company network.

4 ADDITIONAL USE CASE

In addition to the canary release the Istio service mesh is used to limit the rate of local requests. For this an EnvoyFilter is deployed which limits the number of requests allowed for each instance of app-deployment. It defines a bucket with a maximum capacity of 20 requests. This bucket is completely refilled every minute. This ensures that the services will not be overrun and that they will be able to keep up with the workload. However since it is random which deployment service will be assigned to a user it is possible that a user will be rate limited on one deployment while not on the other which results in inconsistent page availability as the user is not rerouted to the available service.

5 EXPERIMENTAL SETUP

The current experiment is still being discussed, however the infrastructure allows us to test two different phishing detection models and then utilize metrics such as the average rate of phishing detected and the average phishing probability returned by the model. The current model-service endpoints does not yet support predictions with known feature-label pairs so it is not yet possible to

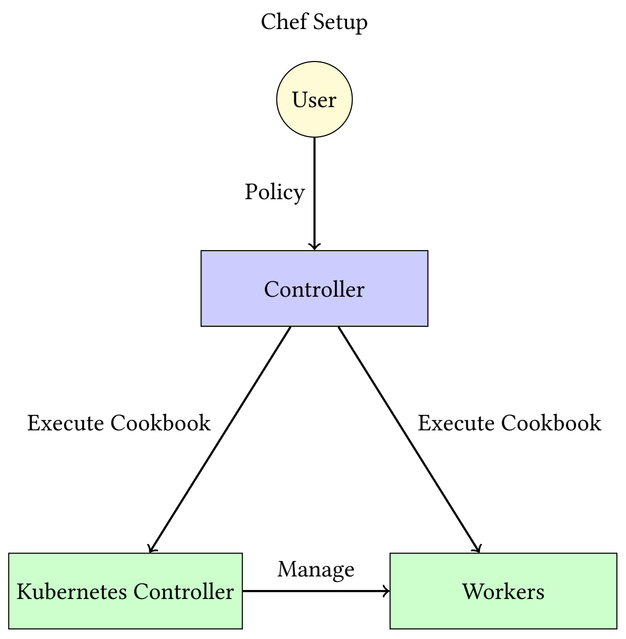


Figure 5: Example Chef Infrastructure

determine metrics such as accuracy. An example hypothesis that can currently be tested could be: *A model trained on fewer epochs is more likely to predict "legitimate" and therefore has a lower average phishing rate.*

6 ML-PIPELINE

This section covers the pipeline for training the phishing detection model as well as the different tools used. Furthermore it explains some of the shortcomings of the pipeline.

6.1 Pipeline

The pipeline is managed by DVC and consists of following three stages: preprocessing, training, and testing. The pipeline was designed such that after each of these stages intermediate output files could be created which can serve as checkpoints for the next stage. An overview of the pipeline can be found in Figure 6.

6.2 Tools

The project relies on DVC for both pipeline management and data version control. This allows for great reproducibility as well as efficient data management. Furthermore google drive was used for remote storage. One google drive folder manages all the DVC artifacts, while a seperate google drive folder contains the models and related files specifically for deployment. These files are downloaded during runtime of model-service. This allows for swapping of the models without creating a new image.

6.3 Shortcomings

A major shortcoming of the pipeline is that it is only partially automated as some artifacts still need to be manually uploaded to the separate drive folder that was created specifically for deployment. This can be very error prone as this can not only introduce human error but it also means that the manually uploaded version for deployment is not versioned. This makes it not immediately clear which version of the model is currently deployed.

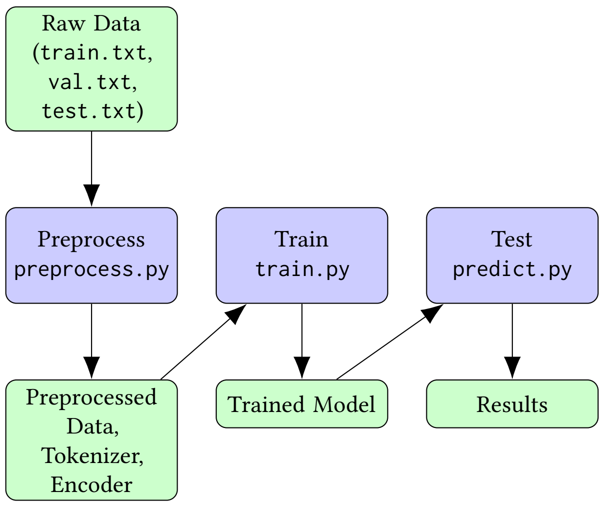

Raw Data (train.txt, val.txt, test.txt)
Preprocess preprocess.py
Train train.py
Test predict.py
Preprocessed Data, Tokenizer, Encoder
Trained Model
Results

Figure 6: ML Pipeline

7 ML TESTING DESIGN

This section will first describe and explain the tests implemented, it will then talk about the limitations and which additional tests could be added. Finally, it will explain what was done for continuous training.

7.1 Automated Tests

The tests created are there to ensure the functionality of the training pipeline, therefore they were implemented in the model-training repository.

Here's a list of the different tests that were done:

- Data quality tests: These tests are here to ensure the quality of the data, this is done by verifying the uniqueness of data samples, the data should at least be 99% unique.
- Integration tests: These tests ensure that the complete pipeline can be run together without causing any issues, they follow the following pattern: preprocess the data, build the model, train the model, test the model, and plot
- Model definition test: Verify that the model is defined properly
- Model development tests:
  - Capability test: Ensure that if the data (URL) starts with HTTP or HTTPS it yields similar results.
  - Non-determinism test: Ensure that the training phase is nondeterministic and that the difference accuracy between 2 trained models should be minimal
- Test monitoring: Ensure that it doesn't use too much RAM <4GB

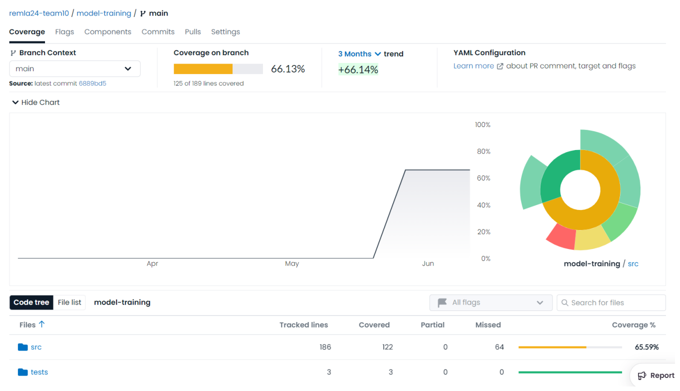
- Test train: Ensure that the training phase returns a trained model
- Test Preprocess: Ensure data exists and that it has the right shape

To improve the testing multiple things can be done, first, add more unit tests to increase the test coverage. Currently, none of the tests are using the DVC data, therefore it would be useful to add some. Finally, some metamorphic tests could be added.

7.2 Continuous training

To ensure the quality of our code, a testing pipeline was set up using GitHub workflows.

This pipeline will run the tests in each OS (Windows, MacOS, Linux) to ensure no dependency issues. Then, it will upload the results to Codecov as seen in Figure 7.



remla24-team10 / model-training / main

Branch Context: main
Source: latest commit 6889bd5

Coverage on branch: 66.13%
125 of 189 lines covered

3 Months trend: +66.14%

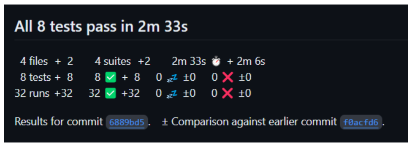
YAML Configuration
Learn more about PR comment, target and flags

model-training

Files	Tracked lines	Covered	Partial	Missed	Coverage %
src	186	122	0	64	65.59%
tests	3	3	0	0	

Figure 7: Screenshot of Codecov dashboard of model-training

Then a test report will be created as shown in Figure 8, it will show up as a comment in each merge request.



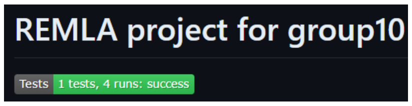
```
All 8 tests pass in 2m 33s

 4 files  +  2    4 suites  +2    2m 33s ⏱️ + 2m 6s
 8 tests +  8    8 ✅ +  8    0 💤 ±0    0 ❌ ±0
32 runs  +32   32 ✅ +32    0 💤 ±0    0 ❌ ±0

Results for commit 6889bd5.   ± Comparison against earlier commit f0acfd6.
```

Figure 8: Screenshot of a test report from the workflow

Finally, a badge is displayed as shown in Figure 9 on the README with the results, this badge is updated every time we merge to the main branch.



REMLA project for group10

Tests | 1 tests, 4 runs: success

Figure 9: Screenshot of the badge

You can cite papers, e.g., [1]. To make the references appear, make sure to compile the latex sources, then bibtex, and then latex twice.

REFERENCES

[1] First Author, Second Author, and Third Author. 2018. An Examplary Paper For The References. In *International Conference on Silly Walks*.

[2] Shannon Flynn. 2022. *Ansible vs Chef: Compare DevOps Tools*. https://www.techrepublic.com/article/ansible-vs-chef/ Accessed: 2024-06-10.

[3] Eric Hu. 2024. *Chef vs. Puppet vs. Ansible: a side-by-side comparison for 2024*. https://betterstack.com/community/comparisons/chef-vs-puppet-vs-ansible/#7-configuration-management-chef-and-puppet-wins Accessed: 2024-06-10.

[4] HubertNNN. 2022. *Why is ansible slow with simple tasks*. https://stackoverflow.com/questions/71565392/why-is-ansible-slow-with-simple-tasks Accessed: 2024-06-10.

[5] Pldimitrov. 2023. *Not possible to source .bashrc with Ansible*. https://stackoverflow.com/questions/22256884/not-possible-to-source-bashrc-with-ansible Accessed: 2024-06-10.

[6] Kaushik Sen. 2024. *Ansible vs Chef Updated for 2024*. https://www.upguard.com/blog/ansible-vs-chef#toc-4 Accessed: 2024-06-10.