Audit Network Orchestration Platform

Orchestrate your audits across networks and unify the results in one place

Development Team

Augsburger Kénan
Baud Tristan
Nicolet Victor
Rocha Ferreira Mário André

Summer Project

August 2025

Table of Contents

Introduction

Problem statement

Penetration tests and audits today often span across complex environments: VLANs, cloud VPCs, branch offices.

Security engineers must rely on many tools (nmap, gobuster, ffuf, etc.), each producing its own reporting format.

This creates several issues :

- High manual effort to correlate and analyze
- Risk of errors and missing findings
- Valuable time lost that could be used for deeper testing

Proposed solution

Pentulz is a platform that orchestrates the execution of pentest tool on distributed agents and centralizes the findings.

- Agents are deployed on the target machines (on-premises hosts, cloud instances, etc.) and run the tools locally within those environments.
- The backend plane collects and normalizes the results into a consistent schema
- The frontend provides a clear web interface to explore and export results

As a result, the impact is accelerated audits, improved accuracy, and increased value from security engagements.

Requirements

Functional requirements

1. Ability to launch scans from the web interface (tool selection, target specification, parameters).
2. Support deployment and management of multiple remote agents.
3. Monitor scan execution and endpoints status, including logs and status updates.
4. Normalize raw tool outputs into a unified JSON schema.
5. Centralize and persist findings in a relational database.
6. Provide a frontend to display results (tables, graphs, filters, search).
7. Ensure secure communication between agents and backend via authentication tokens/keys.
8. Allow exporting of results in common formats (CSV, JSON).

Non-functional requirements

1. **Performance**
   - Frontend must completely load in less than **500 ms**
2. **Security**
   - All communications must be encrypted via **TLS 1.2+**
3. **Auditability**
   - All user actions (job creation, modification, deletion) must be logged with timestamp, user ID, job ID.
4. **Portability**
   - Agents must run on **Linux x86_64**, **Windows 64 bits** and containerized environments (Docker).
   - Deployment should be possible with Docker Compose
5. **CI/CD**
   - Build pipelines must complete in $\leq$ **10 minutes**
6. **Interface**
   - Frontend should be responsive
   - Frontend should have a good UI and UX design
7. **Simplicity**
   - Instead of having websockets communication between the API server and the agents, a simple manifest file is used to indicate what tasks should be run on each machine (agents perform pooling)
8. **Modularity**
   - Adding a new command to run on the agents should not require modifying more than 5 files (on the API server and the agents source code)

The project

Architecture

The project uses a 3 tier architecture.

Agents are probes which run on the tested network. They can be deployed on physical hosts on premise, virtual machines in the cloud or even containerized. The containerized approach is useful for both simpler deployment since the image features all the available tools, as well as the possibility of testing container networks.

The backend is where the results are centralized, pre-analyzed and made available to the security engineers. It can be scaled since it's stateless as all the data is stored in a PostgreSQL database. All the communications with the backend from the agents and front-end go through its API.

Finally, the third tier is the front-end which provides an easy to use graphical interface to access the scan results as well displays the pre-analyzed results in a way that's easy to understand and parse.

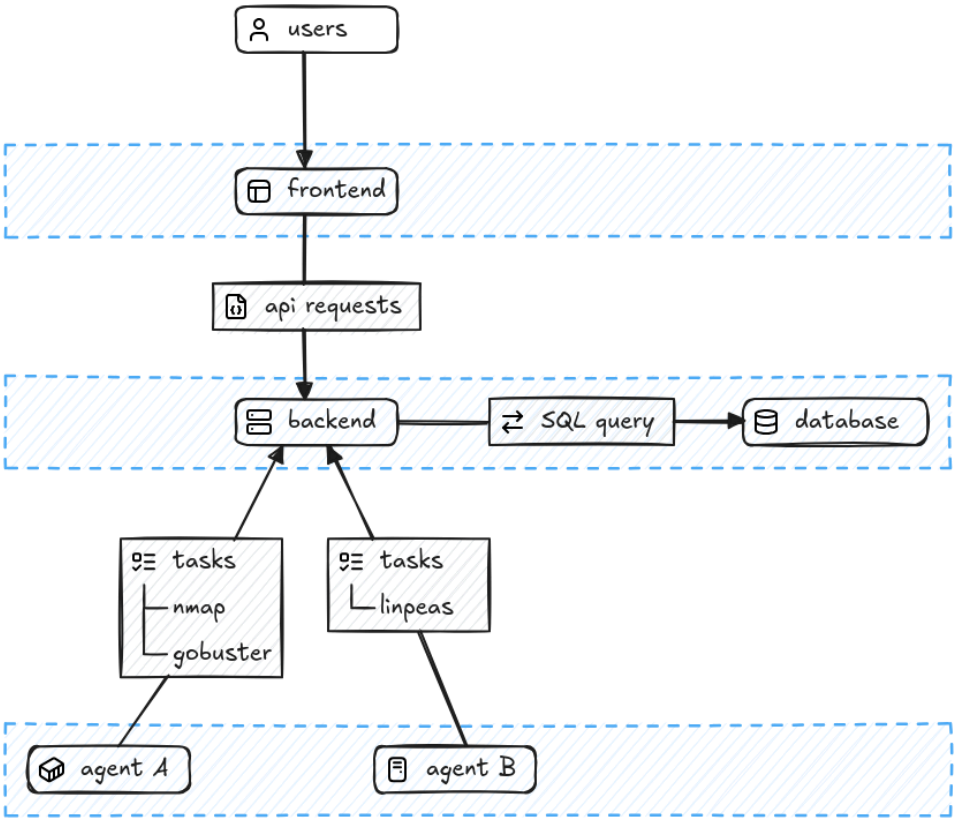

Figure 1: Architecture

Workflow

Project management and planning

The project planning follows the agile methodology. Due to the time constraint, the project will span three sprints of one week and rely on a Kanban approach to ease the load on the planning side.

All the planning is done using GitHub Projects inside the organization.

As stated during the kickoff, the first week is dedicated to specifications, planning, mockups and setting up the environment with the pipeline. The tasks for each sprints will be defined during our meetings each Monday.

Collaboration Framework

All code changes require peer review approval. Architecture decisions involve the full team, with each lead responsible for their domain's technical quality and documentation.

Tooling

There's no assigned editor or tools directly. The only requirements to merge pull requests is:

- The checks must pass (unit tests, linter and build)
- A member of the organization must approve the pull request after reviewing the changes.

CI/CD Pipeline

Each repository automatically builds and releases on `main` branch pushes. The `main` branch is protected and requires successful workflows before merging.

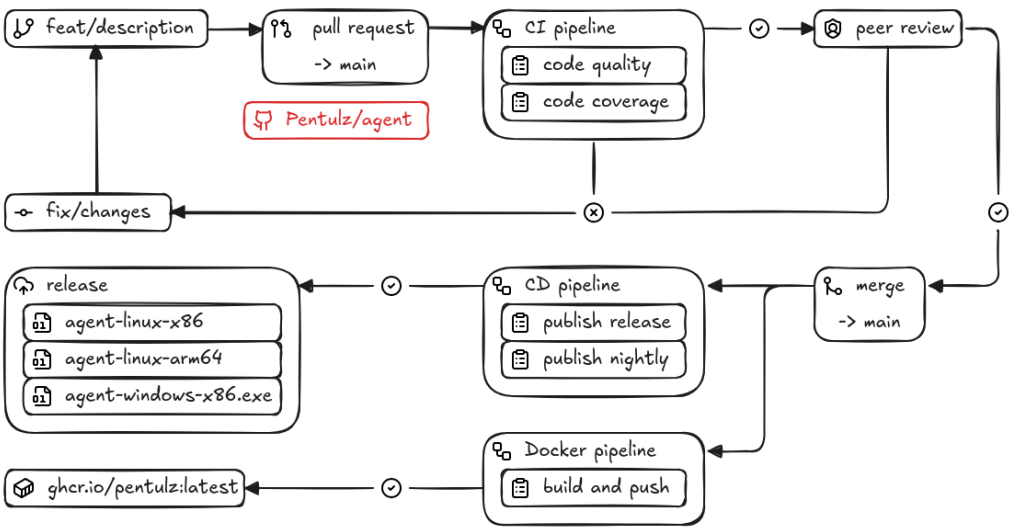

Figure 2: Agent pipeline

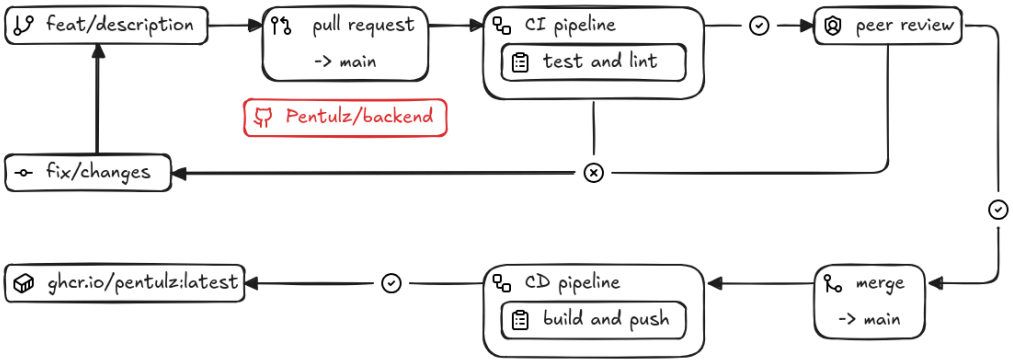

Figure 3: Backend pipeline

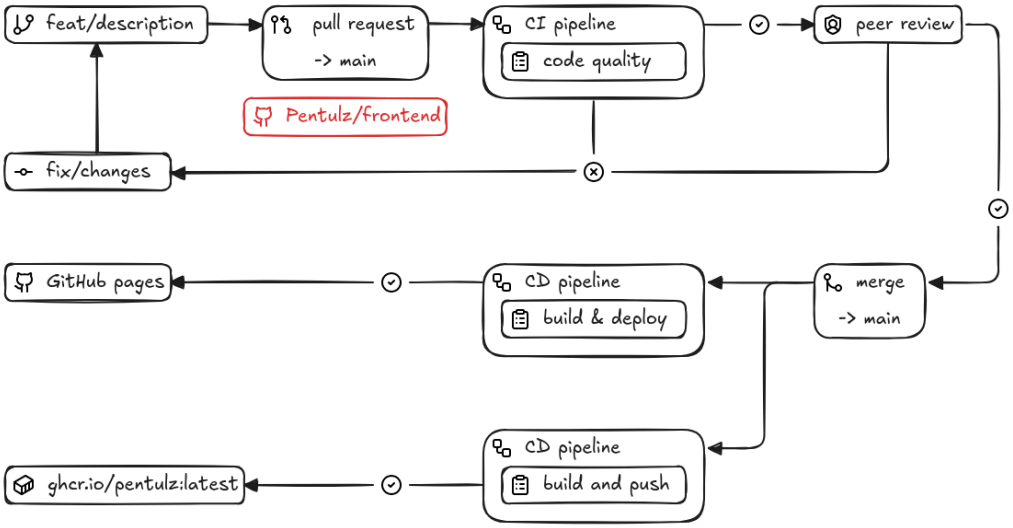

Figure 4: Frontend pipeline

Team Organization and Roles

Role Assignment

Role	Team Member	Key Responsibilities
Scrum Master/ Developer	Augsburger Kénan	Sprint coordination, cross-component development suppor
Backend Lead	Tristan Baud	FastAPI development, database design, API security
Frontend Lead	Nicolet Victor	Nuxt/Vue interface, UX design, performance optimization
Agent/DevOps Lead	Rocha Ferreira Mário André	Rust agent development, CI/CD pipelines, deployment

Responsibilities Overview

Scrum Master

- Facilitate agile ceremonies and weekly Monday meetings
- Maintain GitHub Projects board and track sprint progress
- Provide development support across backend, frontend, and agent components

Backend Lead

- Design PostgreSQL schema and implement FastAPI endpoints
- Handle data normalization and authentication mechanisms

Frontend Lead

- Develop responsive web interface
- Implement data visualization components and export functionality
- Integrate with backend API and optimize user experience

Agent/DevOps Lead

- Buid Rust agents
- Configure Docker containerization and deployment automation
- Integrate pentesting tools and secure communication protocols

Technical choices

While the mix of technologies in this project might not be the best idea in a real situation, we believe that it still makes sense for each individual components and is a great learning opportunity.

We chose rust for the agent because we were interested in learning more about the language and being a compiled language we'll get a single binary executable. Without going too in depth, the main dependencies we'll be relying on will be:

- `clap`: To create a CLI app with argument parsing and documentation
- `reqwest`: To communicate with the back-end over HTTP
- `serde/serde_json`: To serialize/de-serialize the messages sent/received from the API
- `tokio`: To provide a good async runtime as well as a good support to run commands on the system

For the back-end, we chose python and FastAPI since it makes it easy to create and document a REST API. This allows to focus on the business logic rather than writing a lot of boilerplate and repeating ourselves. Since we're using python we have the issue of requiring a python interpreter as well as handling issues with dependencies in different environments. This actually isn't an issue since the back-end should be deployed through containers since it is stateless and can be scaled horizontally. We also don't want to spend too much time writing SQL unless necessary, which is why we'll be using an ORM.

Finally, for the front-end and the landing page, we chose to use Nuxt since it uses Vue and can be easily deployed to GitHub pages or served with any web server.

Conclusion