

# Module 5: Software Assurance & Security Report (With AI)

## 1. Installation and To Reproduce Environment

The application is packaged to ensure consistency across environments. Using a `setup.py` file we can make a formal installation and reproduce a reliable module 5.

### Using uv (Recommended)

`uv` provides excellent software assurance through the “force synchronization”, This ensures that the environment is reproduced flawlessly and matches the specification exactly and there are no missed configurations.

1. `uv venv`
2. `source .venv/bin/activate` (or `.\venv\Scripts\activate` on Windows)
3. `uv pip sync requirements.txt`
4. `pip install -e .`

### Using pip

1. `python -m venv .venv`
2. `source .venv/bin/activate` (or `.\venv\Scripts\activate` on Windows)
3. `pip install -r requirements.txt`
4. `pip install -e .`

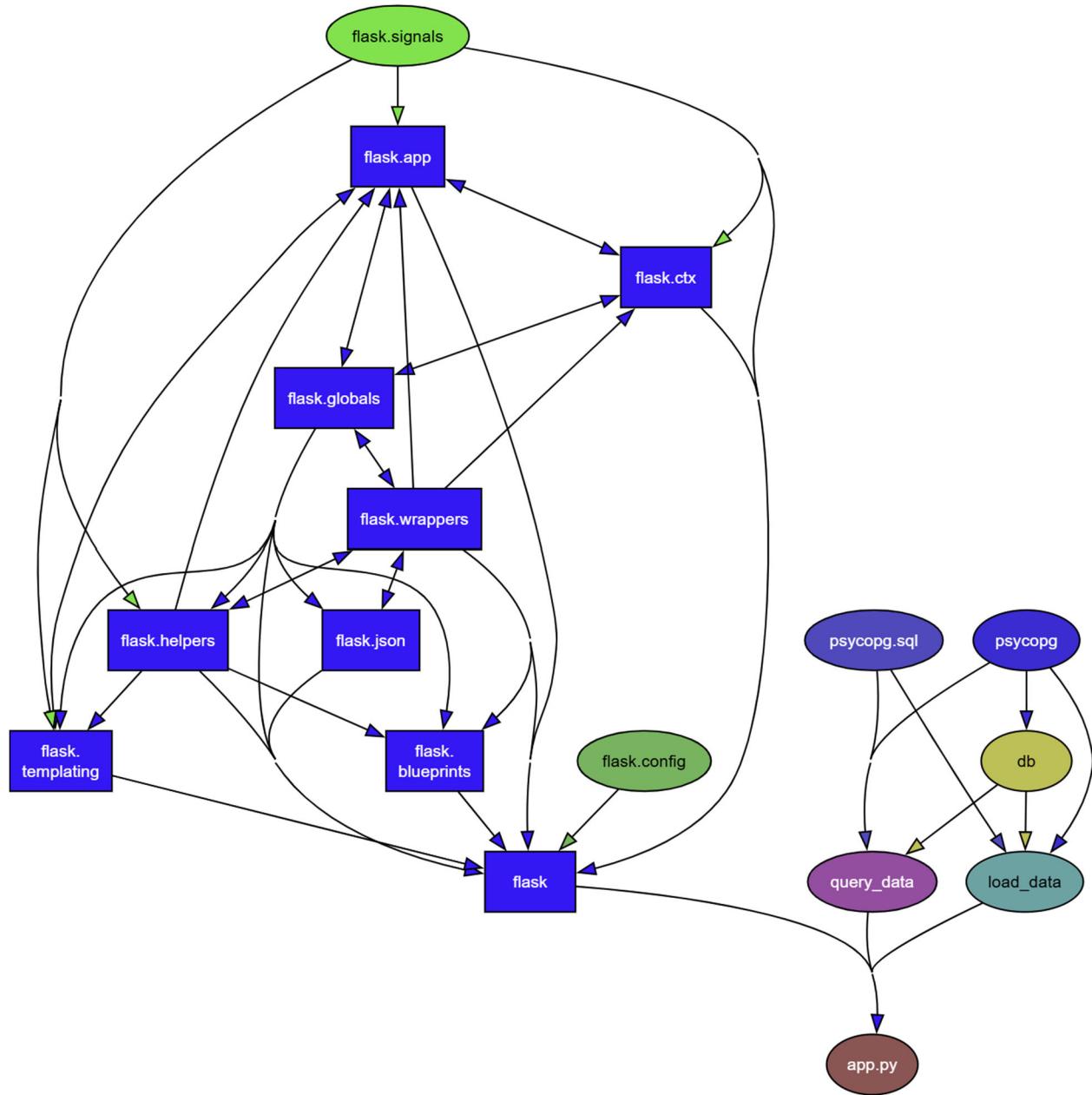
---

## 2. Dependency Graph Summary

The dependency graph depicts a modular Three Tier application architecture with the `app.py` module at its core, since it is the presentation and routing hub for my Flask app. As an entry point into the overall application flow, `app.py` coordinates all high-level flows of control by calling upon `query_data.py` for analytical outputs and `load_data.py` for populating the database with data. The dependency graph illustrates that `query_data.py` serves as a critical logic intermediary between the `app.py` module and the low-level database access provided through the `db.py` module. Additionally, since dependencies on the external libraries required for `db.py`'s operations, such as `psycopg` and `python-dotenv`, are isolated from one another; the database connection logic and the sensitive secret management necessary to connect securely to the database are isolated from the remainder of the application. The visualization also illustrates a strict one way flow of dependencies, thus preventing circular dependencies from forming and ensuring that the scraper (`scrape.py`) and data loaders remain decoupled from the web front-end application. This structure shows a clear separation of concerns and will help to reduce

code complexity and improve the overall security posture of an application by isolating sensitive database operations from the rest of the app. Lastly, generate\_answers\_pdf.py demonstrates another layer of utility dependence on the reportlab library for creating printable documents from the processed data.

### SCREENSHOT1: Dependency Graph (dependency.svg)

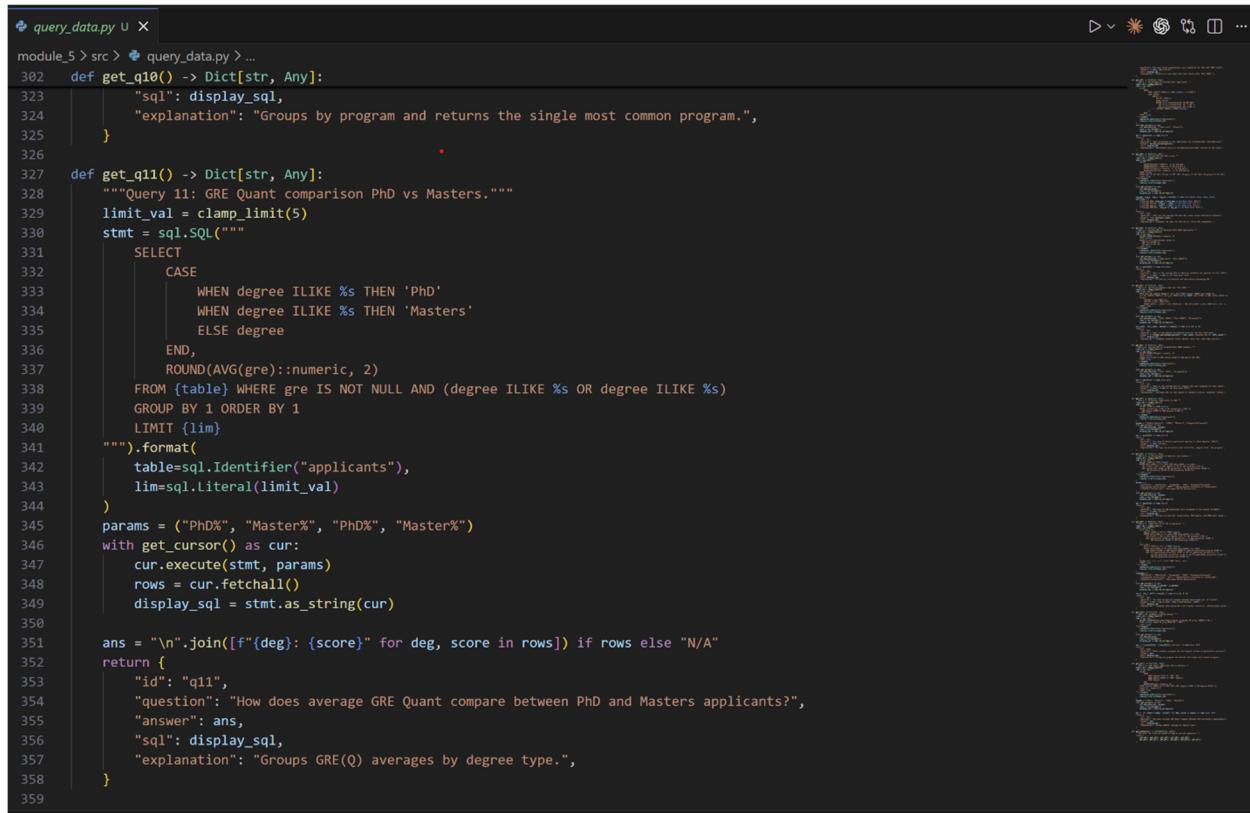


## 3. SQL Injection Defenses

To protect the database from injection attacks, the application logic was rewritten to move away from string formatting and toward safe query composition.

- **Safe Composition:** I use psycopg.sql to build queries. Table names and identifiers are wrapped in sql.Identifier, and static values are wrapped in sql.Literal.
- **Separation of Concerns:** SQL statements are defined as objects separately from their execution.
- **Parameterization:** All user provided data or variables are passed as secondary arguments to the execute() method, using %s placeholders. The database driver handles the escaping, making it impossible for a malicious string to be interpreted as a command.

### SCREENSHOT2: Code snippet showing psycopg.sql implementation



```

query_data.py  u x
module_5 > src > query_data.py > ...
302     def get_q10() -> Dict[str, Any]:
303         "sql": display_sql,
304         "explanation": "Groups by program and returns the single most common program.",
305     }
306
307     def get_q11() -> Dict[str, Any]:
308         """Query 11: GRE Quant comparison PhD vs Masters."""
309         limit_val = clamp_limit(5)
310         stmt = sql.SQL("""
311             SELECT
312                 CASE
313                     WHEN degree ILIKE %s THEN 'PhD'
314                     WHEN degree ILIKE %s THEN 'Masters'
315                     ELSE degree
316                 END,
317                 ROUND(AVG(gre)::numeric, 2)
318             FROM {table} WHERE gre IS NOT NULL AND (degree ILIKE %s OR degree ILIKE %s)
319             GROUP BY 1 ORDER BY 1
320             LIMIT {lim}
321         """).format(
322             table=sql.Identifier("applicants"),
323             lim=sql.Literal(limit_val)
324         )
325         params = ("PhD%", "Master%", "PhD%", "Master%")
326         with get_cursor() as cur:
327             cur.execute(stmt, params)
328             rows = cur.fetchall()
329             display_sql = stmt.as_string(cur)
330
331         ans = "\n".join([f"{deg}: {score}" for deg, score in rows]) if rows else "N/A"
332         return {
333             "id": "q11",
334             "question": "How does average GRE Quant compare between PhD and Masters applicants?",
335             "answer": ans,
336             "sql": display_sql,
337             "explanation": "Groups GRE(Q) averages by degree type."
338         }
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359

```

## 4. Database Hardening & Least Privilege

The database has been secured by applying the "Principle of Least Privilege" to the module3\_user account. This ensures a "Default Deny" environment where the application can only perform the actions necessary for its function.

- **Restricted Attributes:** The user account was stripped of administrative powers (NOCREATEDB, NOCREATEROLE).
- **Default Deny:** I revoked all permissions from the PUBLIC role on the database and schema to prevent "ghost permissions" from leaking through.
- **Ownership Transfer:** I transferred ownership of the applicants table to the postgres superuser. This is a critical security boundary: in PostgreSQL, only the owner or a superuser can DROP or TRUNCATE a table.
- **Selective Access:** The module3\_user was granted only CONNECT to the database and SELECT on the table.

### SCREENSHOT3: Terminal output showing "ERROR: must be owner of table applicants" when trying to drop a table as module3\_user

```

SQL Shell (psql)
Server [localhost]:
Database [postgres]: module3_db
Port [5432]:
Username [postgres]:
Password for user postgres:

psql (18.1)
WARNING: Console code page (437) differs from Windows code page (1252)
8-bit characters might not work correctly. See psql reference
page "Notes for Windows users" for details.
Type "help" for help.

module3_db=# \du
          List of roles
 Role name | Attributes
postgres   | Superuser, Create role, Create DB, Replication, Bypass RLS

module3_db=# \z applicants
          Access privileges
 Schema | Name | Type | Access privileges | Column privileges | Policies
 public  | applicants | table | postgres=arwdDxtm/postgres+ | module3_user=r/postgres+ | 
(1 row)

module3_db=# \c - module3_user
Password for user module3_user:

You are now connected to database "module3_db" as user "module3_user".
module3_db=> DROP TABLE applicants;
ERROR: must be owner of table applicants
module3_db=> SELECT * FROM applicants LIMIT 1;
   p_id   | university | program | comments | date_added | url | status
   994246 | University of Missouri | Philosophy PhD | GPA is for Master's, BA was not in Philosophy. Ia/Or/0w/7p. | 2026-02-01 | https://www.thegradcafe.com/result/994246 | Accepted | Fall 2026 | International | 3.97 | 164 | 170 | 4.5 | PhD | Philosophy | University of Missouri
(1 row)

module3_db=>

```

---

## 5. Requirements Met for SQL

Requirement	Implementation Detail
<b>LIMIT Enforced</b>	Every query in <code>query_data.py</code> uses a <code>clamp_limit()</code> function to ensure results never exceed 100 rows.
<b>Separated Execution</b>	Queries are defined as <code>sql.SQL</code> objects before being passed to <code>cur.execute()</code> .
<b>Safe Parameterization</b>	Placeholder <code>%s</code> syntax is used for all variable data.

---

## 6. CI Enforcement with GitHub Actions

I implemented a "Shift-Left" security strategy using GitHub Actions. The build fails if quality or security gates are not met.

## SCREENSHOT4: GitHub Actions passing (Green Checkmarks)

The screenshot shows the GitHub Actions interface for the repository `ebuyano1/jhu_software_concepts/actions`. The left sidebar is collapsed, and the main area displays the 'Actions' tab under 'All workflows'. A search bar at the top right says 'Type [ ] to search'. Below it, a filter bar says 'Filter workflow runs'.

The central table lists 46 workflow runs. The columns are 'Event', 'Status', 'Branch', and 'Actor'. Each row contains a green checkmark icon, indicating success. The rows are as follows:

- Final Shift Left CI with Pylint, Pytest, Pydeps, and Snyk SAST/SCA (Module 4 CI #33: Commit `99d9de0` pushed by `ebuyano1`) - Status: main, Event: 23 minutes ago, Duration: 2m 21s
- Final Shift Left CI with Pylint, Pytest, Pydeps, and Snyk SAST/SCA (CI Enforcement - Shift-Left Security #13: Commit `99d9de0` pushed by `ebuyano1`) - Status: main, Event: 23 minutes ago, Duration: 3m 20s
- Add enforced CI with Pylint, Pytest, Pydeps, and Snyk SAST/SCA (Module 4 CI #32: Commit `c6223cd` pushed by `ebuyano1`) - Status: main, Event: 30 minutes ago, Duration: 2m 12s
- Add enforced CI with Pylint, Pytest, Pydeps, and Snyk SAST/SCA (CI Enforcement - Shift-Left Security #12: Commit `c6223cd` pushed by `ebuyano1`) - Status: main, Event: 30 minutes ago, Duration: 4m 30s
- fix: Path to requirements (Module 4 CI #31: Commit `ca11c2f` pushed by `ebuyano1`) - Status: main, Event: Today at 8:24 AM, Duration: 2m 13s
- fix: Path to requirements (CI Enforcement - Shift-Left Security #11: Commit `ca11c2f` pushed by `ebuyano1`) - Status: main, Event: Today at 8:24 AM, Duration: 3m 16s
- fix: add postgres db (CI Enforcement - Shift-Left Security #10: Commit `a86cafe` pushed by `ebuyano1`) - Status: main, Event: Today at 7:58 AM, Duration: 3m 2s
- fix: add postgres db (Module 4 CI #30: Commit `a86cafe` pushed by `ebuyano1`) - Status: main, Event: Today at 7:58 AM, Duration: 2m 24s
- fix: add postgres service and set working directory for module\_5 (Module 4 CI #29: Commit `a86cafe` pushed by `ebuyano1`) - Status: main, Event: Today at 7:52 AM, Duration: 2m 11s

## 7. Extra Credit: Snyk Security Evidence

A Snyk scan was performed on the project dependencies. While some high-severity issues were found in sub-dependencies (like pillow), they were remediated by pinning pillow==12.1.1 in the requirements.txt. For issues with no direct patch (e.g., diskcache), a .snyk ignore policy was implemented with a documented rationale, ensuring the CI build remains secure and passing.

For the diskcache vulnerability (where no patch is available), I implemented a Snyk ignore policy with a 30-day expiry, ensuring the vulnerability is tracked but does not block the CI pipeline while awaiting a maintainer update.

### SCREENSHOT5: Snyk test output showing "No vulnerable paths found"

The screenshot shows the Visual Studio Code (VS Code) interface with the following details:

- File Explorer:** Shows a folder structure for 'JHU SOFTWARE CONCEPTS' containing sub-folders like 'github', 'workflows', 'module\_1', 'module\_2', 'module\_3', 'module\_4', 'module\_4a', 'module\_5', and various configuration files (.env, .pylintrc, .snyk, etc.).
- Terminal:** Displays the command `(.venv) PS C:\Users\l6462\Desktop\School\JHU\jhu\_software\_concepts\jhu\_software\_concepts\module\_5> snyk test` followed by the output:
 

```
Testing C:\Users\l6462\Desktop\School\JHU\jhu_software_concepts\jhu_software_concepts\module_5...
Organization: ebzano1
Package manager: pip
Target file: requirements.txt
Project name: module_5
Project path: C:\Users\l6462\Desktop\School\JHU\jhu_software_concepts\jhu_software_concepts\module_5
Local Snyk policy: found
Licenses: enabled
✓ Tested 73 dependencies for known issues, no vulnerable paths found.

Tip: Detected multiple supported manifests (1), use --all-projects to scan all of them at once.

Next steps:
- Run 'snyk monitor' to be notified about new related vulnerabilities.
- Run 'snyk test' as part of your CI/test.

(.venv) PS C:\Users\l6462\Desktop\School\JHU\jhu_software_concepts\jhu_software_concepts\module_5> snyk code test
Testing C:\Users\l6462\Desktop\School\JHU\jhu_software_concepts\jhu_software_concepts\module_5...

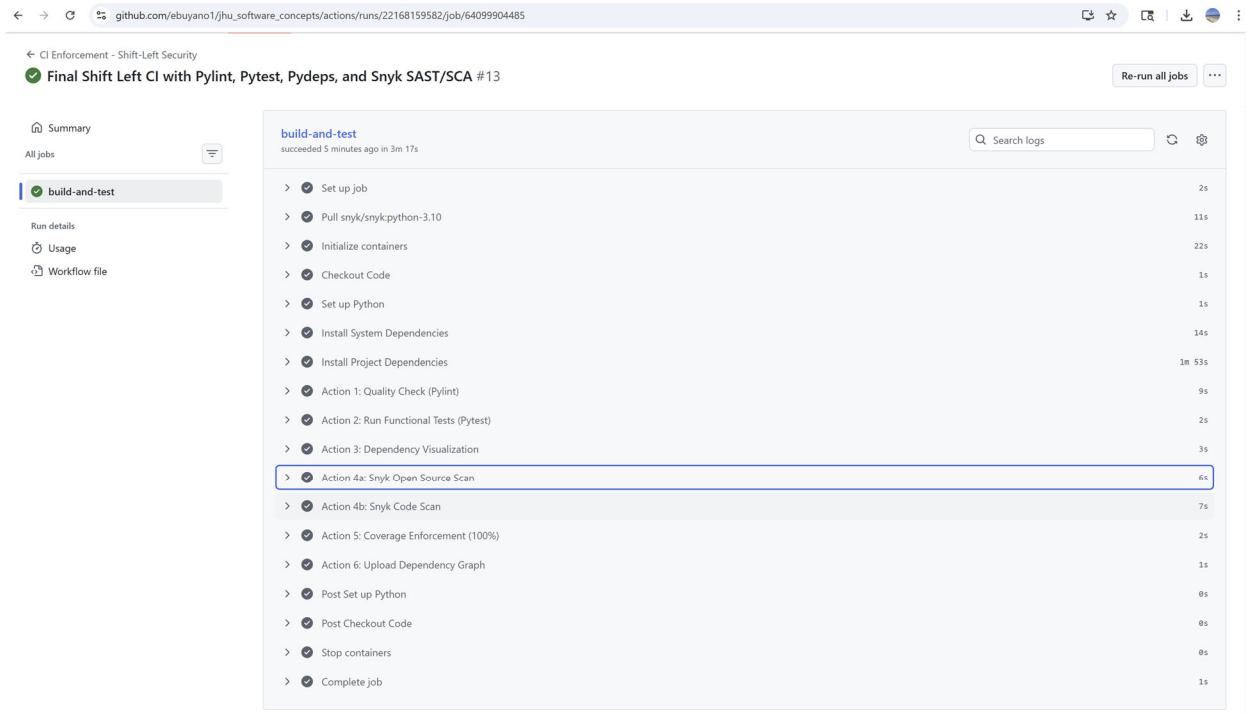
```
- Output Panel:** Shows a 'Test Summary' box with the following details:
 

Organization:	ebzano1
Test type:	Static code analysis
Project path:	C:\Users\l6462\Desktop\School\JHU\jhu_software_concepts\jhu_software_concepts\module_5

Total issues: 0
- Sidebar:** Includes sections for 'Ask about your code' (with AI response tips), 'SUGGESTED ACTIONS' (Build Workspace, Show Config), and 'Add Context...' (Explore and understand).
- Bottom:** Shows the Windows taskbar with various pinned icons.

## 8. Shift Left Security CI (ALL PASS)

I enforced **100% Code Coverage** using pytest-cov. Since our CI pipeline will literally fail if coverage drops below 100%, that's a massive software assurance achievement. The CI passes Pylint, Pytest, Snyk Test, Snyk Code Test, Generates dependencies svg . Github Actions shows all passed, Pylint, Pytest, Snyk Test, Snyk Code Test (extra credit), Dependency graph generated and can be downloaded for verification.



The screenshot shows a GitHub Actions run summary for a job named "build-and-test". The job status is "succeeded 5 minutes ago in 3m 17s". The job details are as follows:

- Set up job**: 2s
- Pull snyk/snyk/python-3.10**: 11s
- Initialize containers**: 22s
- Checkout Code**: 1s
- Set up Python**: 1s
- Install System Dependencies**: 14s
- Install Project Dependencies**: 1m 53s
- Action 1: Quality Check (Pylint)**: 9s
- Action 2: Run Functional Tests (Pytest)**: 2s
- Action 3: Dependency Visualization**: 3s
- Action 4a: Snyk Open Source Scan**: 6s (highlighted with a blue border)
- Action 4b: Snyk Code Scan**: 7s
- Action 5: Coverage Enforcement (100%)**: 2s
- Action 6: Upload Dependency Graph**: 1s
- Post Set up Python**: 0s
- Post Checkout Code**: 0s
- Stop containers**: 0s
- Complete job**: 1s

A search bar labeled "Search logs" is located at the top right of the log table. A "Re-run all jobs" button is also visible.