

A. Application Configuration Management (Python)

Development (default)

```
$ python config_loader.py
```

```
[CONFIG] Loaded environment: dev
```

```
{'DB_URL': 'mongodb://localhost:27017/devdb', 'DEBUG': True, 'SERVICE_URL':  
'http://localhost:8080'}
```

Staging

```
$ APP_ENV=staging python config_loader.py
```

```
[CONFIG] Loaded environment: staging
```

```
{'DB_URL': 'mongodb://staging-db:27017/stagingdb', 'DEBUG': False, 'SERVICE_URL':  
'https://staging.api.bookandride.com'}
```

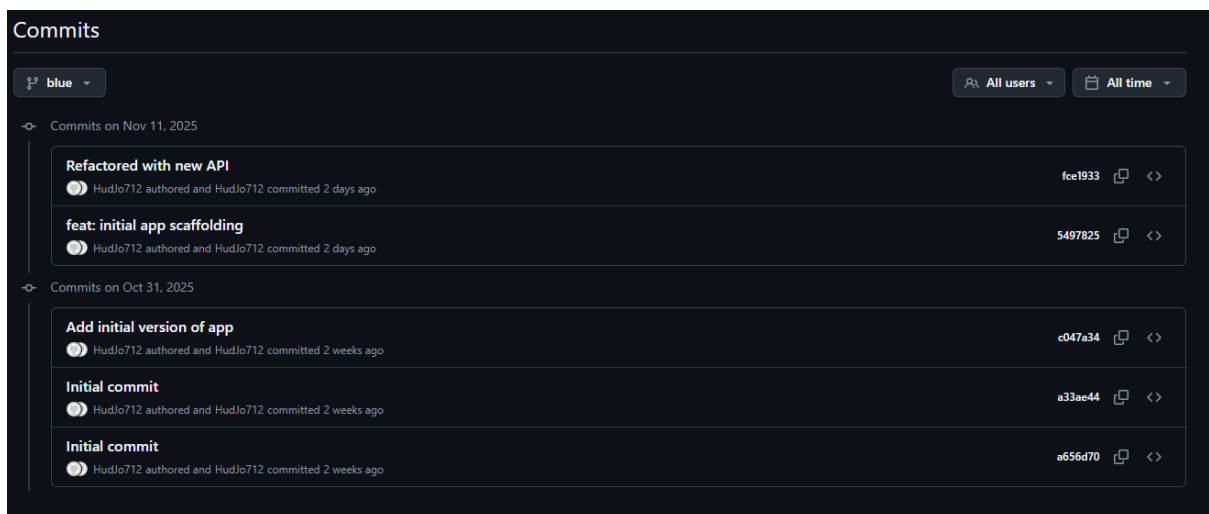
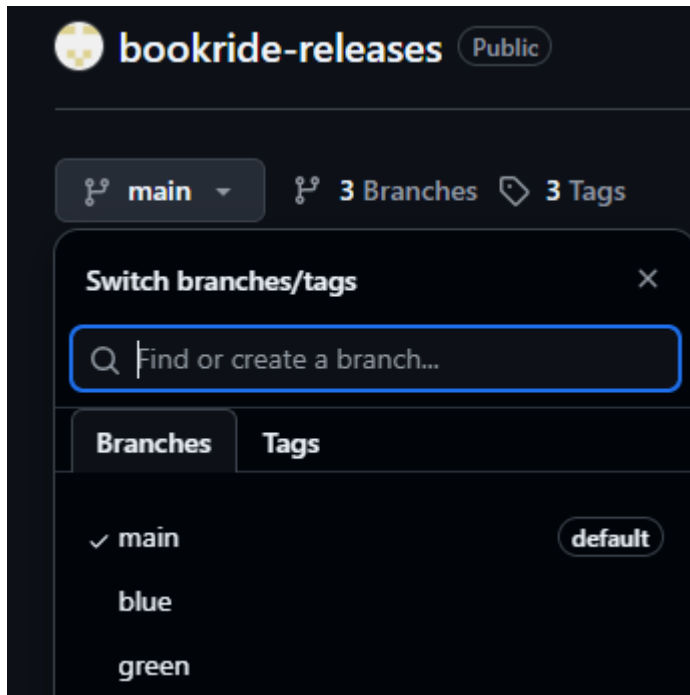
Production

```
$ APP_ENV=prod python config_loader.py
```

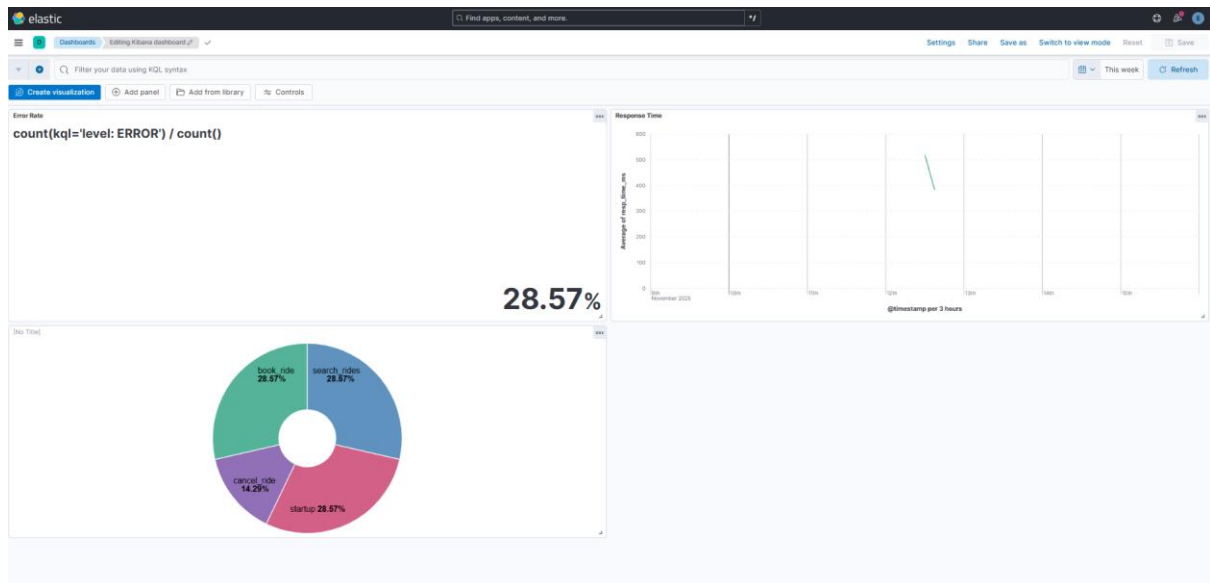
```
[CONFIG] Loaded environment: prod
```

```
{'DB_URL': 'mongodb://prod-db:27017/proddb', 'DEBUG': False, 'SERVICE_URL':  
'https://api.bookandride.com'}
```

B. Release Management & Versioning (Git + Semantic Versioning)



C. Observability and logging



D. Git rollback & Recovery

```
cd ~/DevOp2/bookride-devops
```

```
# make sure your work is clean first
```

```
git status
```

1) Undo local changes (working directory only)

```
# simulate accidental edits
```

```
echo "# temp change" >> app.py
```

```
echo "# debug logging = true" >> config.yaml
```

```
# see them
```

```
git status
```

```
# undo changes just in your working directory:
```

```
git restore app.py config.yaml
```

```
# or selectively:
```

```
git restore app.py
```

```
git restore -> undo uncommitted changes to files.
```

2) Revert a bad production commit (safe on main / prod)

```
Simulate a bad prod commit
```

```
# break something in the API
echo "raise Exception('Boom in prod')" >> api/main.py
# and flip upstream in nginx (just as an example)
echo "# temporary tweak" >> nginx/active-upstream.conf
git status
git add api/main.py nginx/active-upstream.conf
git commit -m "bad: break API and tweak active upstream"
Revert it safely
# find the commit hash
git log --oneline
# copy the hash of "bad: break API and tweak active upstream"
git revert <bad-commit-hash>
```

3) Recover “lost” commits after a reset (reflog + reset)

Create a commit we’ll “lose”

```
echo "v1.2.0 - new pricing rules" >> RELEASE_NOTES.md
git add RELEASE_NOTES.md
git commit -m "Add v1.2.0 notes"
git reset --hard HEAD~1
Use reflog to get it back
git reflog
git reset --hard <hash2>
```

4) Revert vs Reset — when to use which in *your* repo

git revert (for your main / prod branch):

- Adds a **new commit** that undoes another commit.
- Keeps the full history, which is crucial for:
 - main in bookride-devops
 - auditability (who deployed what, when)
 - CI/CD pipelines that rely on linear history.

- Use when:
 - A bad commit is **already pushed** (e.g., broken api/main.py, wrong nginx/*.conf).
 - You want to **roll back production** safely.

git reset (for local cleanups, never on shared history):

- **Rewrites history** by moving your branch pointer.
- Good for:
 - Cleaning up your own work before pushing (reset --soft or --mixed).
 - Throwing away your local commits you haven't shared yet.
- Dangerous on branches others pull from (would require force-push, can confuse everybody).

Quick modes:

- git reset --soft HEAD~1 → keep changes staged.
- git reset HEAD~1 (mixed, default) → keep changes but unstaged.
- git reset --hard HEAD~1 → throw away commit + changes (recoverable only via reflog).

5) Hotfix branch workflow (for production bugs)

1. Branch off from current production (main)

git checkout main

git pull origin main # make sure you're up to date (when working with GitHub)

git checkout -b hotfix/pricing-bug

2. Fix the bug and tests

edit the bug:

nano api/models.py

update/add tests:

nano api/tests/test_pricing.py

optionally update release notes:

nano RELEASE_NOTES.md

run tests

```
python -m pytest api/tests
```

3. Commit the hotfix

```
git add api/models.py api/tests/test_pricing.py RELEASE_NOTES.md
```

```
git commit -m "hotfix: fix pricing calculation edge case"
```

```
git push -u origin hotfix/pricing-bug
```

4. Merge the hotfix into main (no rewriting history)

```
git checkout main
```

```
git pull origin main    # in case main moved
```

```
git merge --no-ff hotfix/pricing-bug -m "Merge hotfix: pricing calculation edge case"
```

5. Tag the fixed release (so deployments know what to deploy)

```
git tag -a v1.0.1 -m "Hotfix v1.0.1: pricing calculation edge case"
```

```
git push origin main
```

```
git push origin v1.0.1
```

Now the commit pointed to by v1.0.1 includes:

- The fix in api/models.py
- Updated tests
- Updated RELEASE_NOTES.md

6. Clean up the hotfix branch

```
git branch -d hotfix/pricing-bug
```

```
git push origin --delete hotfix/pricing-bug # if you want it gone from GitHub too
```