Github Actions Exercice

- Created two repositories on github
  - Action (Stores all the GitHub Actions)
  - Sandbox (Runs the Dummy App)
    - Here is where i test the actions

PS: used the https://docs.github.com/en/repositories/creating-and-managing-repositories/quickstart-for-repositories to create the repositories

- Used the VCC- prefix (Virtual Contact Center)
- Used https://8x8.okta.com/oauth2/v1/authorize?client_id=0oa8oqeem24hM185T297&redirect_uri=https%3A%2F%2F8x8com.cloudflareaccess.com%2Fcdn-cgi%2Faccess%2Fcallback&response_type=code&scope=openid%20groups%20profile%20email&state=2cc70e9a - VCC GitHub Repo Standards to setup and manage the repositories
  - Branch Protection Rules
  - Security and GHAS
  - Github Actions
  - Collaborators
  - Code Owners files

- Creating the GitHub Actions Workflows
  - Ansible Lint Workflow
    - Inspects Ansible files
    - The workflow, named **“ansible-lint”**, runs whenever a pull request is opened, synchronized, or marked as ready for review, but only if the changes affect files in the `playbooks/` directory. It runs a single job called **“Ansible Lint”** on a Linux runner (`build8-linux-x64`). The job checks out the repository code and then executes `ansible-lint` (version 25.6.1) to analyze the Ansible files located in the `playbooks` folder for syntax issues and best practice violations.

```yaml
---
# .github/workflows/ansible-lint.yml
name: ansible-lint

on:
  pull_request:
    types:
      - opened
      - synchronize
      - ready_for_review
    paths:
      - "playbooks/**"

jobs:
  build:
    name: Ansible Lint
    runs-on: build8-linux-x64
    steps:
      - uses: actions/checkout@v4

      - name: Run ansible-lint
        uses: ansible/ansible-lint@v25.6.1
        with:
          working_directory: playbooks
```

    - Este ficheiro deve estar em .github → workflows
  - CI Pipeline Workflow
    - Checkouts the app code

- Runs dummy tests
- Simulate deployments
-

The workflow, called **“CI Pipeline”**, is triggered whenever code is pushed to any branch or a pull request is created, updated, edited, or closed. It runs a single job named **“Run CI Pipeline”** on a Linux runner (`build8-linux-x64`). The job performs four main steps:

1. **Checks out the repository code.**
2. **Sets up the environment** (simulated with an echo command).
3. **Runs dummy tests** to represent automated testing.
4. **Simulates a deployment**, mimicking the process of deploying an application.

```yaml
---
name: CI Pipeline


on:
  push:
    branches:
      - '**'          #qualquer branch
  pull_request:       #qualquer pull request
    types:
      - opened
      - reopened
      - edited
      - synchronize
      - closed
      - ready_for_review

jobs:
  build:
    name: Run CI Pipeline
    runs-on: [build8-linux-x64]

    steps:
      - name: Checkout code
        uses: actions/checkout@v4

      - name: Setup environment
        run: |
          echo "Setting up the environment..."

      - name: Run dummy tests
        run: |
          echo "Running Tests"
          sleep 2
          echo "All tests passed successfully!"

      - name: Simulate Deployment
        run: |
          echo "Deploying the artif.."
          sleep 2
          echo "Deployment simulation complete!"
```

- Implementing Build8
- The workflow, named **“vcc-sandbox-gcunha-repo”**, runs when:
  - It is manually triggered (`workflow_dispatch`),
  - A pull request is opened, updated, or closed, or
  - Code is pushed to the `master` branch.

It uses **concurrency control** to ensure that only one workflow run per branch or reference is active at a time, preventing overlapping executions.

The workflow's single job, `build`, doesn't define steps directly — instead, it **reuses an external workflow** from the repository `8×8/build8` (`vcc_standard.yaml` version `vcc-1.2.1`). It also **inherits repository secrets** and grants **write permissions** to allow actions such as updating statuses or uploading artifacts.

.gitignore file:

- Used to stop constantly upload temporary files to the repo, like for example _DS.Store

Set your identitygit config --global user.name "Your Name"
git config --global user.email "your@email.com"View all configurationsSet default branch nameSet default editor (example: VS Code)Useful Commands

⚙️ Setup & Configuration

- `git config --global user.name "Your Name"` — set your Git username
- `git config --global user.email "you@example.com"` — set your Git email
- `git config --global --list` — show all global settings
- `git config --global init.defaultBranch main` — set default branch name
- `git config --global core.editor "code --wait"` — set VS Code as default editor

📦 Repository Basics

- `git init` — create a new repository
- `git clone <repo-url>` — clone an existing repo
- `git status` — check current file status
- `git add <file>` — stage file for commit
- `git add .` — stage all changes
- `git commit -m "message"` — save staged changes
- `git log --oneline --graph` — compact commit history view

🌿 Branching & Merging

- `git branch` — list branches
- `git branch <name>` — create new branch
- `git switch <name>` — switch branches
- `git checkout -b <name>` — create & switch to new branch
- `git merge <branch>` — merge into current branch
- `git branch -d <name>` — delete branch

🌍 Remote Operations

- `git remote add origin <url>` — add remote repository
- `git remote -v` — view remotes
- `git push -u origin main` — first push to remote

- `git push` — push changes
- `git pull` — fetch and merge updates
- `git fetch` — fetch updates only
- `git remote rename origin upstream` — rename remote
- `git remote remove origin` — remove remote

🔄 Undo & Reset

- `git restore --staged <file>` — unstage file
- `git reset --soft HEAD~1` — undo last commit, keep changes
- `git reset --hard HEAD~1` — undo last commit, discard changes
- `git revert <commit>` — create new commit that undoes changes
- `git checkout -- <file>` — restore file from last commit

🧹 Cleanup & Maintenance

- `git clean -f` — remove untracked files
- `git clean -fd` — remove untracked files & folders
- `git gc --prune=now` — clean unnecessary files
- `git rev-list --objects --all | sort -k 2 > allfiles.txt` — list all repo files

🧠 Inspect & Compare

- `git diff` — show unstaged changes
- `git diff --staged` — show staged changes
- `git diff <branch1>..<branch2>` — compare branches
- `git show <commit>` — show commit details
- `git show --name-only <commit>` — show files changed in a commit

🏷️ Tags & Releases

- `git tag v1.0` — create lightweight tag
- `git tag -a v1.0 -m "Release v1.0"` — create annotated tag
- `git push --tags` — push all tags
- `git tag -d v1.0` — delete local tag
- `git push origin --delete tag v1.0` — delete remote tag

🧪 Useful Tricks

- `git show HEAD --stat` — show details of last commit
- `git stash` — temporarily save uncommitted work
- `git stash list` — show stashed items
- `git stash pop` — restore stashed changes
- `git cherry-pick <commit>` — apply specific commit to current branch
- `git rebase -i HEAD~3` — squash or edit last 3 commits

- `git blame <file>` — see who changed each line