Andersen DevOps course

Exam task

Maxim Chepukov

Languages

1. **Golang for first application**
    a. Gin framework
    b. At first implementation it should return “Hello world 1” by http request
    c. https://gitlab.com/MaximChepukov/hellogolangapp
2. **Python for second**
    a. Flask framework
    b. As in the first case but it should return “Hello world 2”
    c. https://gitlab.com/MaximChepukov/pythonhelloapp

Tools

CI/CD - GitLab pipelines

SCM/Control Version - GitLab / git

Registry - Docker Hub

Infrastructure - Amazon ECS (Elastic Container Service)

Notification - Slack

Infrastructure

Amazon Elastic Container Service (ECS)

1. It was chosen because I never used it before
2. Two clusters ECS - EC2 Linux + Networking
3. Only one container in each cluster
4. Without any load balancer but better to use it in production
5. Created manually, but in real life need to describe it in code and store in git

Infrastructure - Network bindings

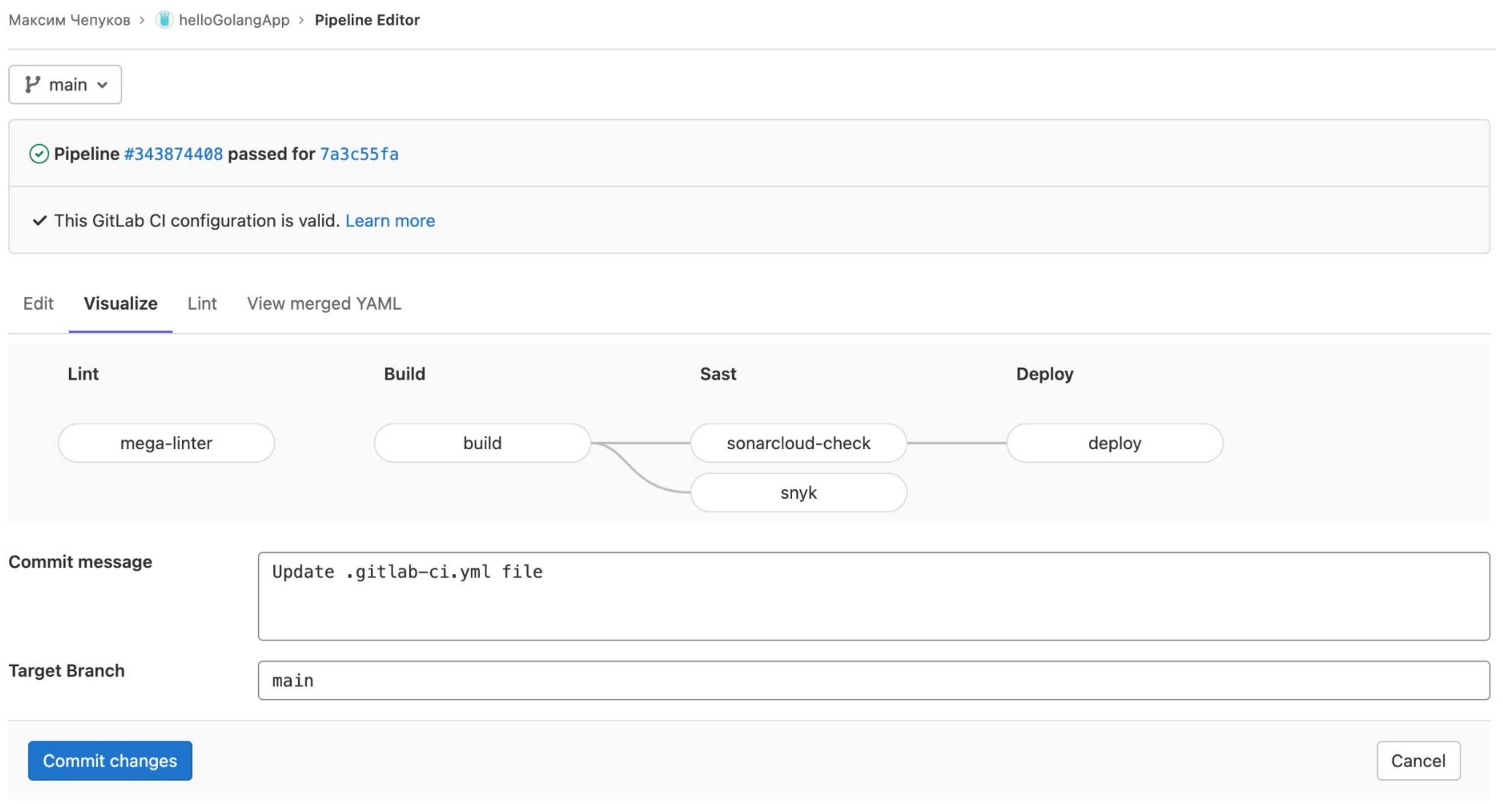
Port Forwarding

1. Golang App
   a. Host Port 80 -> Container Port 8080 (protocol tcp)
2. Python App
   a. Host Port 80 -> Container Port 5000 (protocol tcp)

CI/CD

1. Can’t push to master directly - only from merge request
2. Checkout branch from master, commit and that merge request
3. After commit it will be automatically checked by megalinter
    a. golang app linting by:
        golangci-lint, hadolint, dockerfilelint and markdownlint
    b. python app linting by:
        pylint, isort, flake8, black, bandit, hadolint, dockerfilelint and markdownlint
4. After successful merge request it will be go through next stages:
    a. GoLang App - build, sast (snyk, sonar-qube), deploy (automatically)
    b. Python App - build, sast (snyk, sonar-qube), deploy (manual mode)
5. Errors in sast stage is ignored (don’t do it in production)
6. Notifications about commits, merge requests, fails, etc. send to Slack

Golang app CI/CD process


Максим Чепуков > helloGolangApp > Pipeline Editor

main

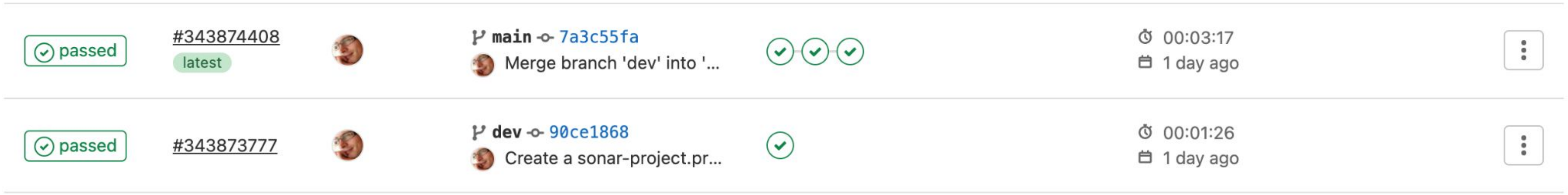
Pipeline #343874408 passed for 7a3c55fa

✓ This GitLab CI configuration is valid. Learn more

Commit message: Update .gitlab-ci.yml file

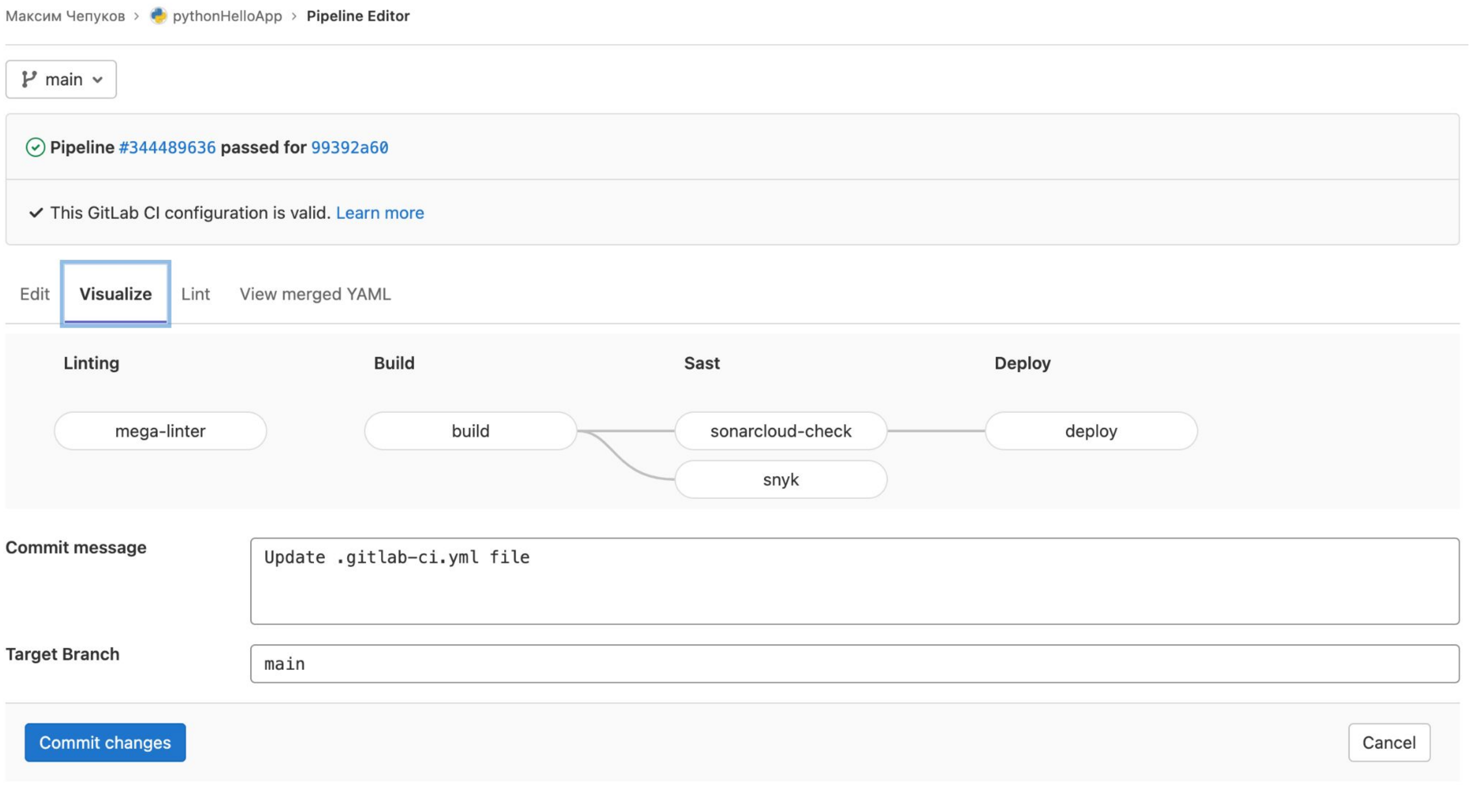
Target Branch: main

Golang app CI/CD process (automatically deployment)



✓ passed	#343874408 latest		main 7a3c55fa Merge branch 'dev' into '...	✓ ✓ ✓	00:03:17 1 day ago	
✓ passed	#343873777		dev 90ce1868 Create a sonar-project.pr...	✓	00:01:26 1 day ago	

Python app CI/CD process



Максим Чепуков > pythonHelloApp > **Pipeline Editor**

main

Pipeline #344489636 passed for 99392a60

✓ This GitLab CI configuration is valid. Learn more

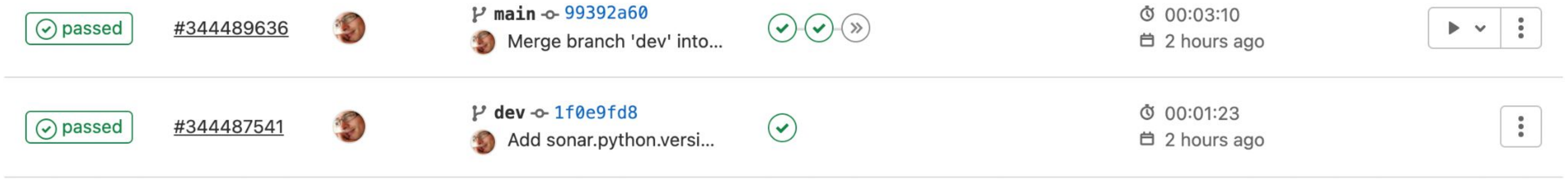
Commit message

```
Update .gitlab-ci.yml file
```

Target Branch

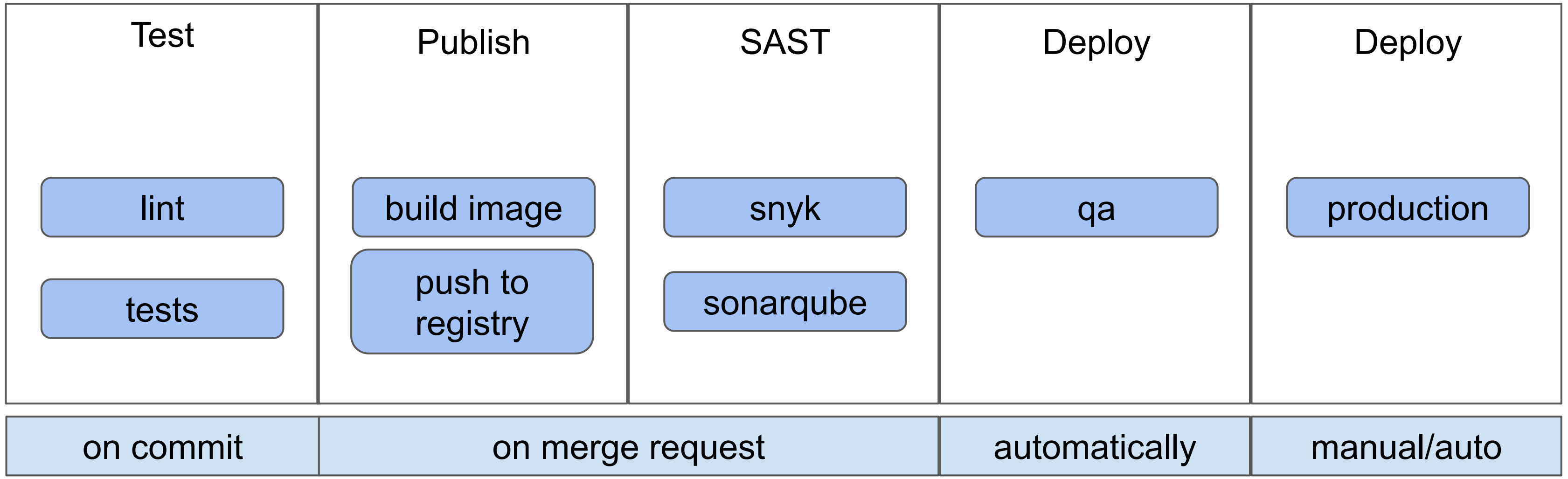
```
main
```

Python app CI/CD process (manual deployment)



✓ passed	#344489636	main -o- 99392a60 Merge branch 'dev' into...	✓ ✓ »	00:03:10 2 hours ago	▶ ⌄ ⋮
✓ passed	#344487541	dev -o- 1f0e9fd8 Add sonar.python.versi...	✓	00:01:23 2 hours ago	⋮

Example another CI/CD process



Differences between CD and CD

Continuous delivery is an extension of continuous integration since it automatically deploys all code changes to a testing and/or production environment after the build stage.

However, if you truly want to get the benefits of continuous delivery, you should deploy to production as early as possible to make sure that you release small batches that are easy to troubleshoot in case of a problem.

Continuous deployment goes one step further than continuous delivery. With this practice, every change that passes all stages of your production pipeline is released to your customers. There's no human intervention, and only a failed test will prevent a new change to be deployed to production.

https://www.atlassian.com/continuous-delivery/principles/continuous-integration-vs-delivery-vs-deployment