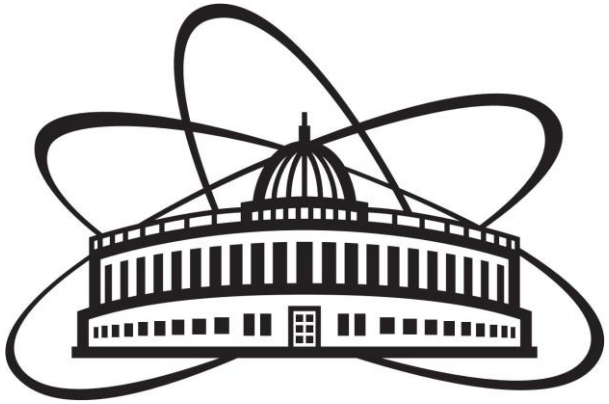


Building, testing and deployment method of SPD application software

Korotkin R.N., Prokoshin F.V., Kozlovsky A.A.

08.07.2025

Context

Necessary actions of a software developer in JINR:

- Setting of virtual operational environment manually;
- System dependencies control;
- Test data preparation;
- Manual building, testing and deployment of software.

Critical tasks:

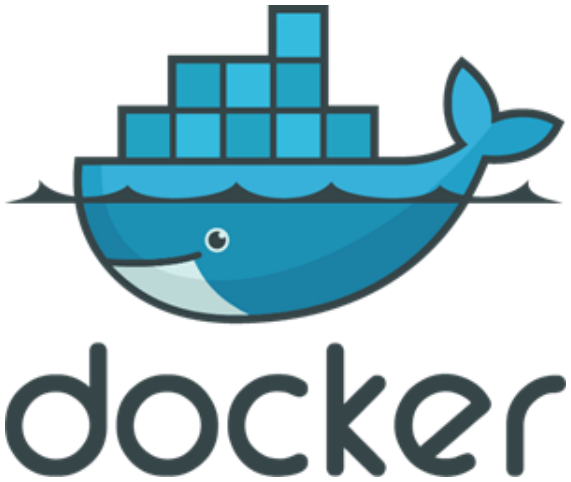
Production System safety from incorrect software, reliability advancement, software deployment automatization, version control.

Solution:

Development of methods for maintaining repositories and automating the processes of building, testing and deployment application software.

Stack

- Gitlab
- Docker
- Bash scripts
- Yaml scripts



DevOps

- CI/CD
- Continuous Testing (WIP)
- Continuous Monitoring (WIP)
- IaC (WIP)
- Microservices

Version control and management practices

- Conventional Commits
- Semantic Versioning
- GitFlow

GitLab

- Gitlab CI/CD — Automating the processes and pipelines management
- Gitlab Runner — Execution of tasks, described in Gitlab CI/CD pipelines
- Container Registry — Images of applications storage
- Gitlab Pages — Documentation web-pages

Version control and management practices

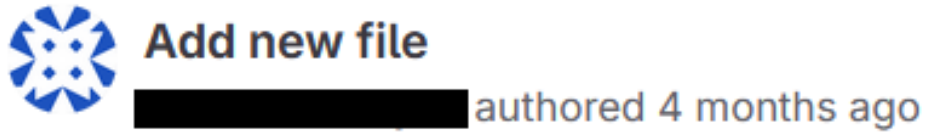
Conventional Commits

<type>[optional scope]:
<description>

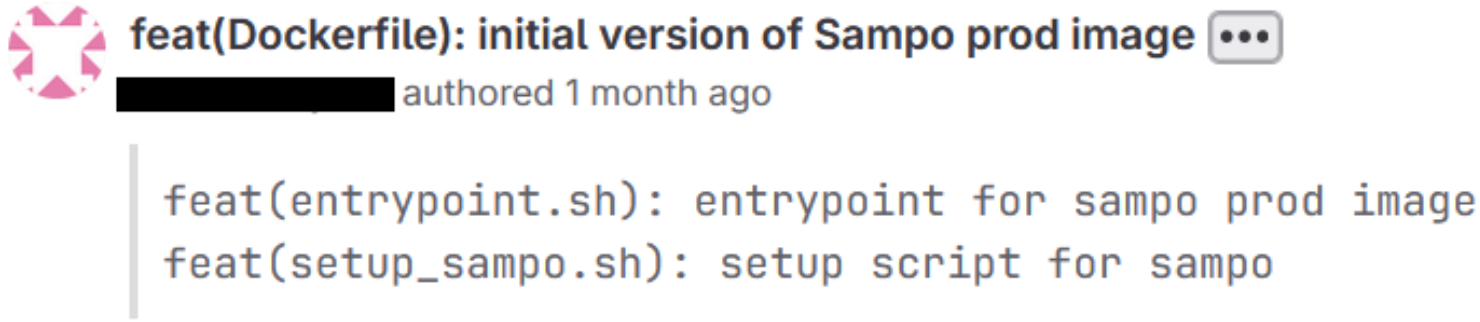
[optional body]

[optional footer(s)]

Instead of such commit description:



Contextual commit description:



Version control and management practices

Conventional Commits

A specification for adding human and machine readable meaning to commit messages

Semantic Versioning

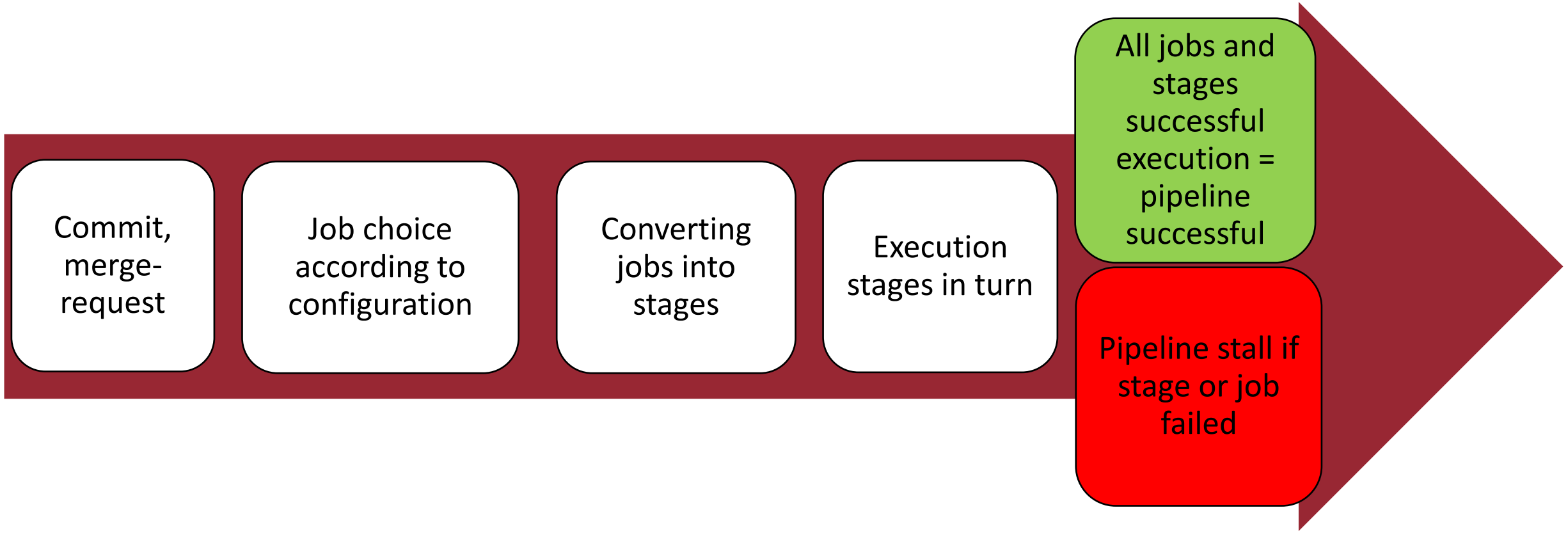
- **fix:** patches a bug in your codebase ⟺ **PATCH** version when you make backward compatible bug fixes
- **feat:** a commit of this type introduces a new feature to the codebase. ⟺ **MINOR** version when you add functionality in a backward compatible manner.
- **BREAKING CHANGE:** a commit that has a footer BREAKING CHANGE:, or appends a **!** after the type/scope, introduces a breaking API change ⟺ **MAJOR** version when you make incompatible API changes.

Software Version Control and Lifecycle Platform

GitLab key features:

- Full integrated CI/CD;
- Auto secrets detecting and security testing;
- Explicit permissions for restriction of merge and code pushing to specific users and groups ;
- Free static web pages with auto-building documentation in Git repositories (Gitlab Pages) and/or Wiki (Gitlab Wiki) for every project.

CI/CD as concept



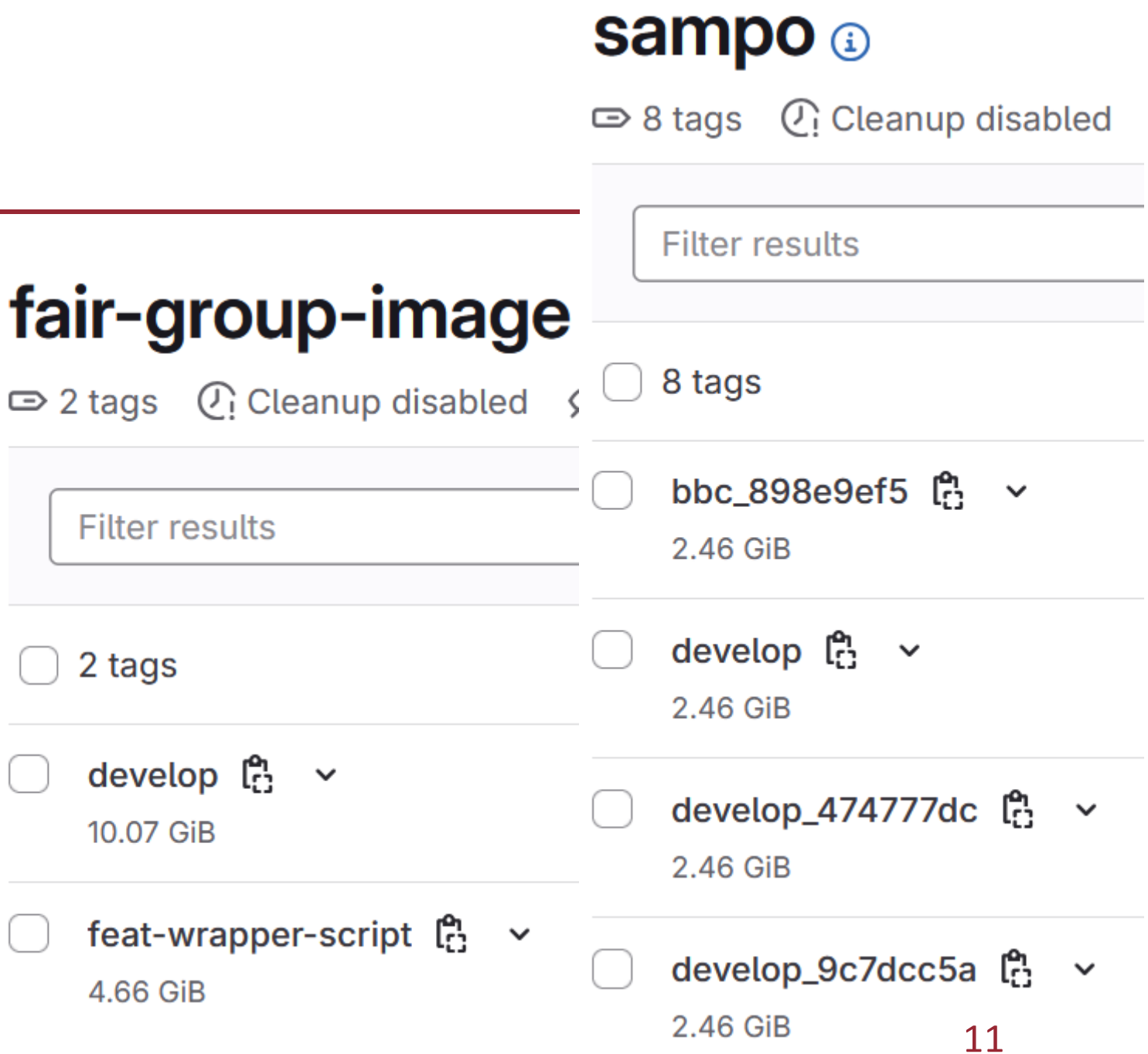
Gitlab Runner

GitLab Runner — the agent that run the GitLab Runner application, to execute GitLab CI/CD jobs in a pipeline defined in a special .gitlab-ci.yml file.

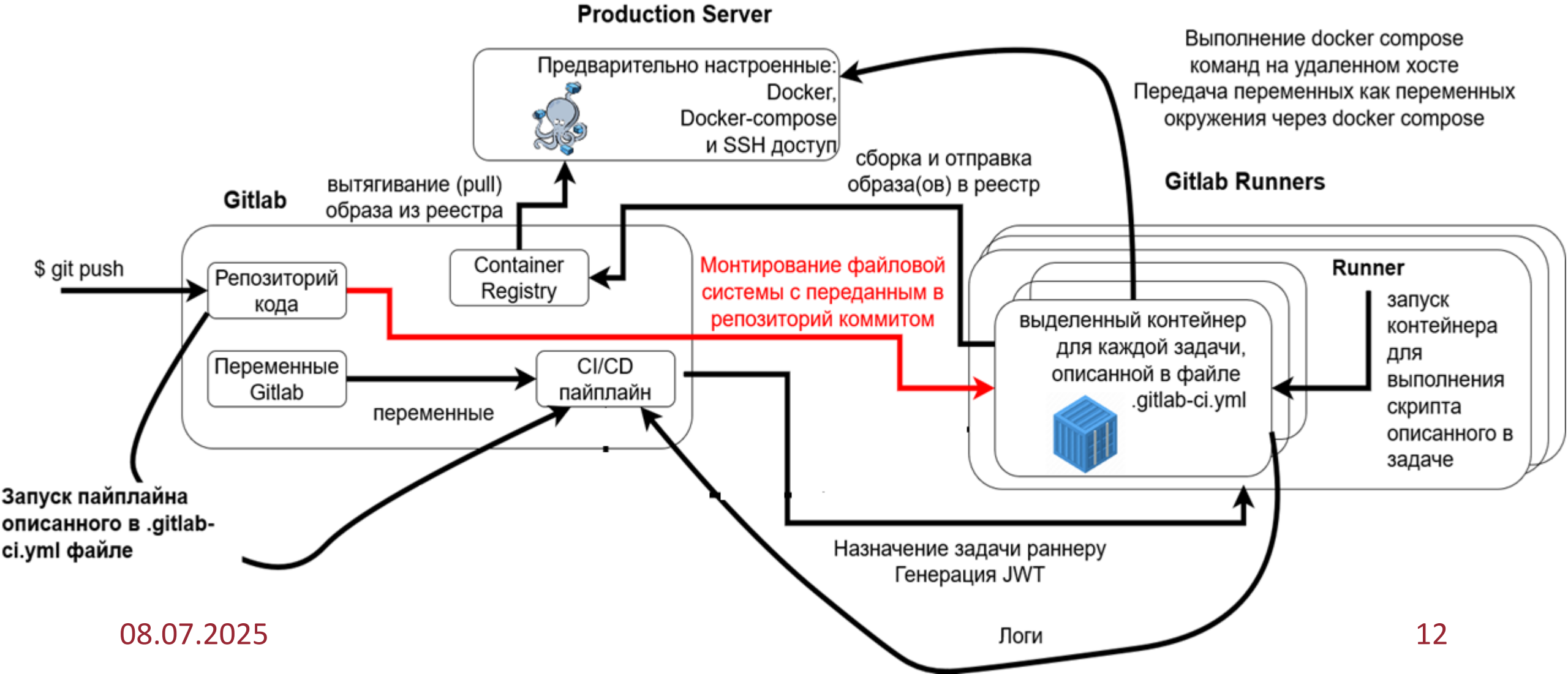
- Execute jobs defined in pipeline;
- Can be installed on any platform;
- Support various executors (including Shell, Docker, and Kubernetes) and environments (local, ssh, clouds);
- Parallel job execution with Runners and data exchange with artifacts (Gitlab artifacts).

Container Registry

GitLab Container Registry is safe and private registry for Docker images. GitLab Container Registry is *fully* integrated into GitLab.



Gitlab: CI/CD, Runner, Container Registry, Pages



Key Features

- Masked project variables;
- Docker-Secrets;
- Unified image tag template: **CONTAINER_REGISTRY:BRANCH_NAME_COMMIT_ID;**
- Retagging job for merge-requests with tag: **CONTAINER_REGISTRY:BRANCH_NAME;**
- Image convertation to .sif (Apptainer format).

FairRoot Framework и SPDRoot Framework

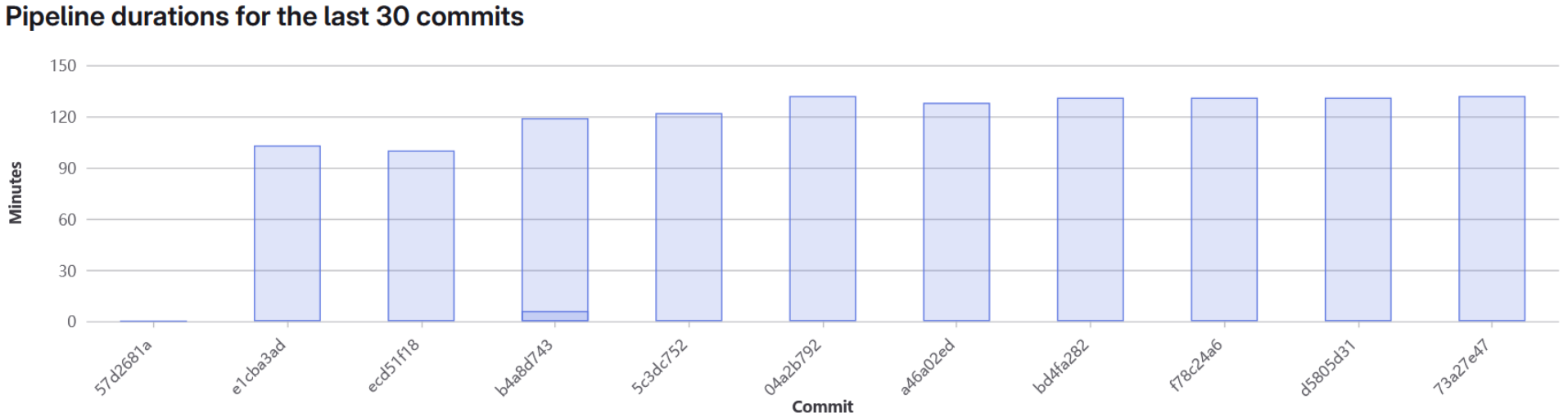
- Separate repositories for the source code of FaitRoot framework and base image designed on it (“fair-group-image”) were created;
- Container Registry images storage was configured;
- CI/CD pipelines are created including automatic:
  - FairRoot framework building inside of a base image “fair-group-image”;
  - Built image pushing into Container Registry storage of “fair-group-image” repository;
  - Building of SPD Root project and image designed on a base image “fair-group-image”.

SPD Gaudi Framework и Sampo Framework

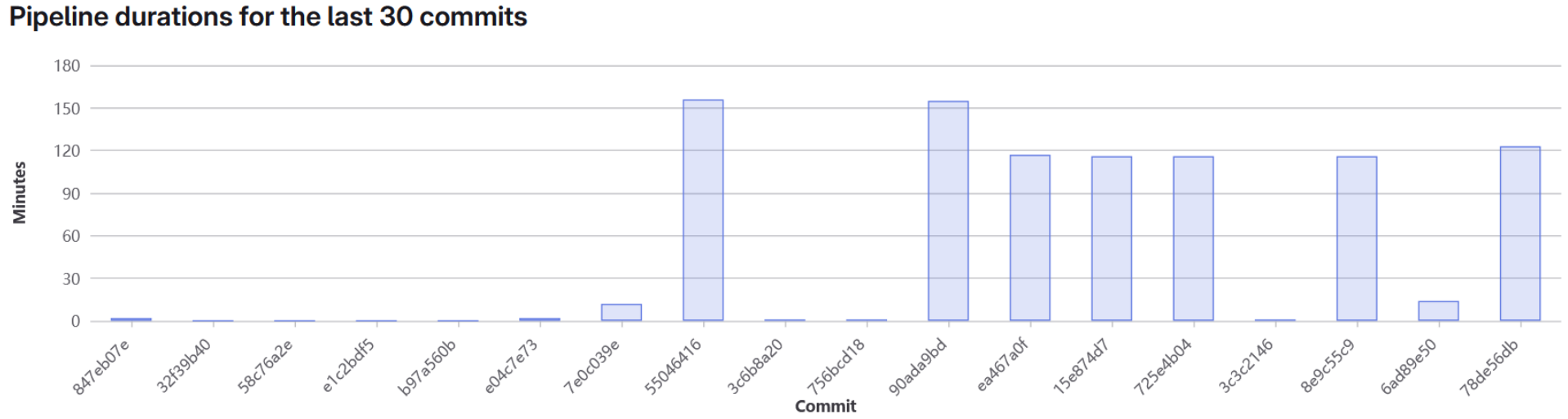
- Separate repositories for the source code of Gaudi framework and base image designed on it (“Gaudi”) were created;
- Code-writing approach described within specified methods is implemented;
- Container Registry images storage was configured;
- CI/CD pipelines are created including automatic:
  - Building of Gaudi framework and base image designed on it;
  - Built image pushing into Container Registry storage of “Gaudi” repository;
  - Building of Sampo project and image designed on a base image “Gaudi”;
  - Built image pushing into Container Registry storage of “Sampo” repository;
- Sampo container deployment with “Apptainer” container engine in CVMFS is being developed.

Results

Gaudi

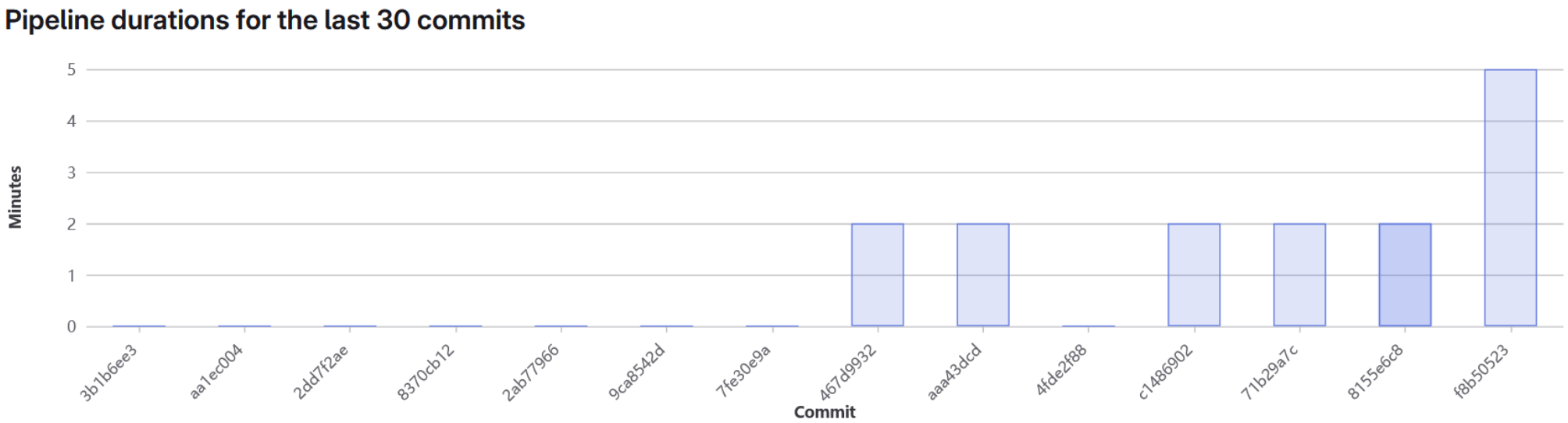


Fair-group-image



Results

Sampo



Base images (Gaudi for Sampo and fairsoft-image for SPDRoot**)** building and deploying ~120 minutes

Sampo image building and deploying ~5 minutes

Conclusion

- The proposed configuration is based on international experience and can be saved as standardized templates for use in other projects.
- The methods remain relevant in a diverse range of projects due to its flexibility.
- The carried out work is subject to further development, and it is important to continue implementing the methods and software tools for SPD collaboration, adjusting them based on the specific tasks, expanding their scope, and training laboratory staff on how to use the methods.
- An instruction manual with recommendations for developers based on the methods is currently being prepared.

Thank you for your attention