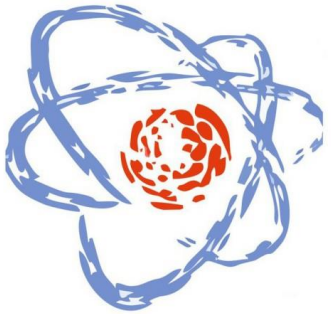
JINR
JOINT INSTITUTE
FOR NUCLEAR RESEARCH

Method for automating the processes of building, testing, and deploying application software within the distributed data processing system of the SPD experiment

Korotkin R., Prokoshin F., Simbiriatin L., Kozlovsky A.



29th International Scientific Conference of Young Scientists and Specialists (AYSS-2025)

29.10.2025

Context

Necessary actions of a software developer:

- 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.

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

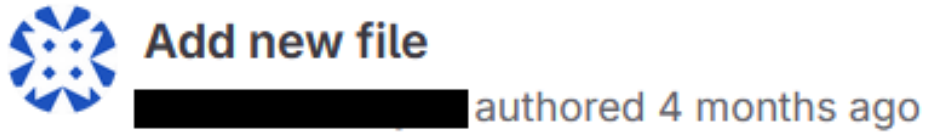
Conventional Commits

<type>[optional scope]:
<description>

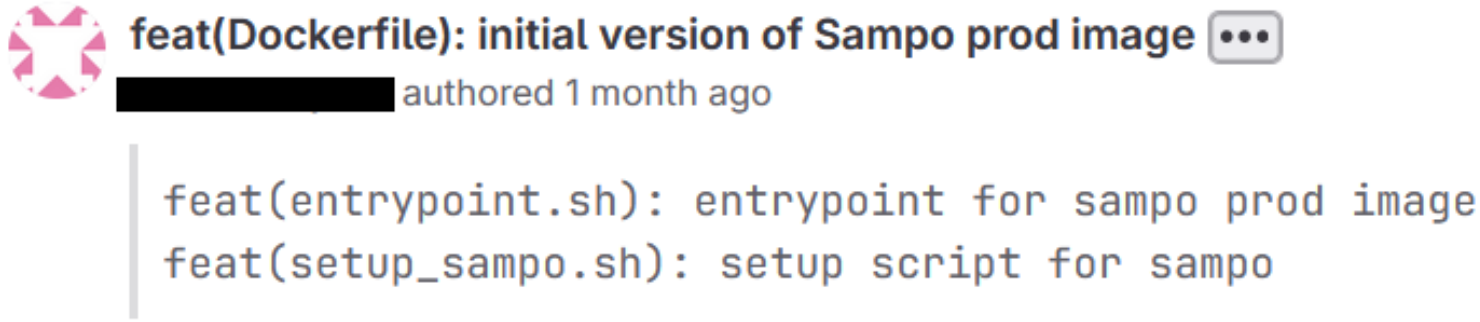
[optional body]

[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

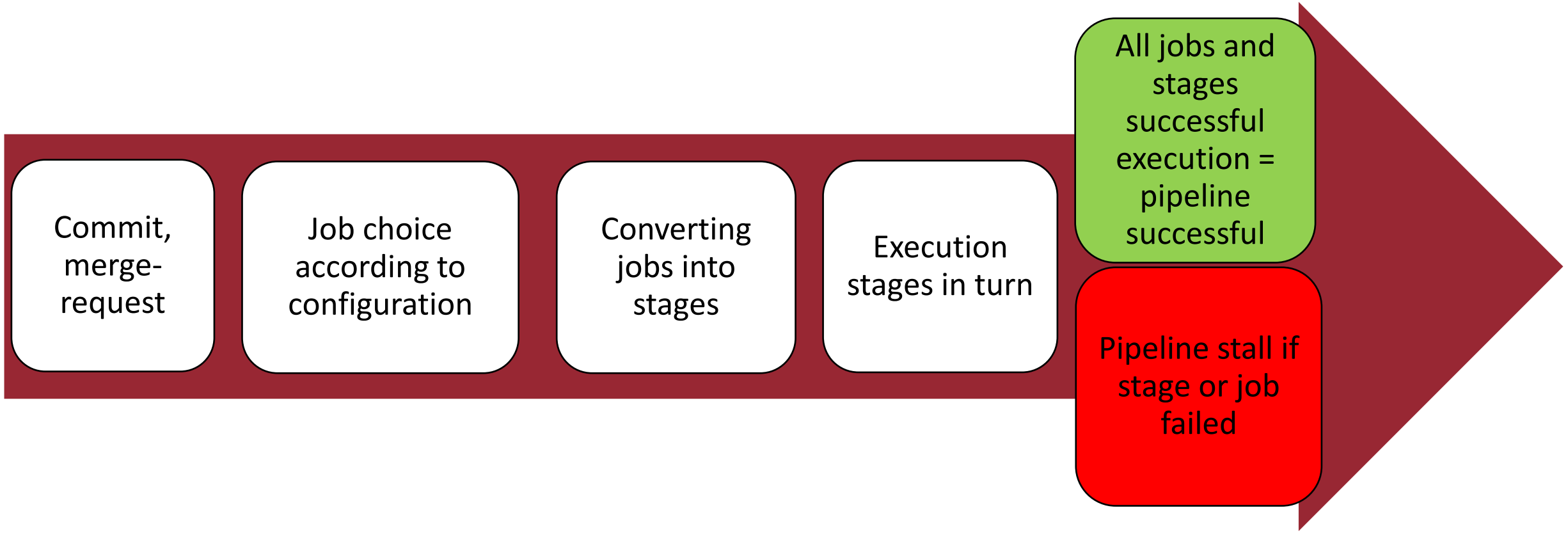
- **fix:** patches a bug in your codebase ⟺ **PATCH** version when you make backward compatible bug fixes
- **feat:** a commit of this type introduces a new feature to the codebase. ⟺ **MINOR** version when you add functionality in a backward compatible manner.
- **BREAKING CHANGE:** a commit that has 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



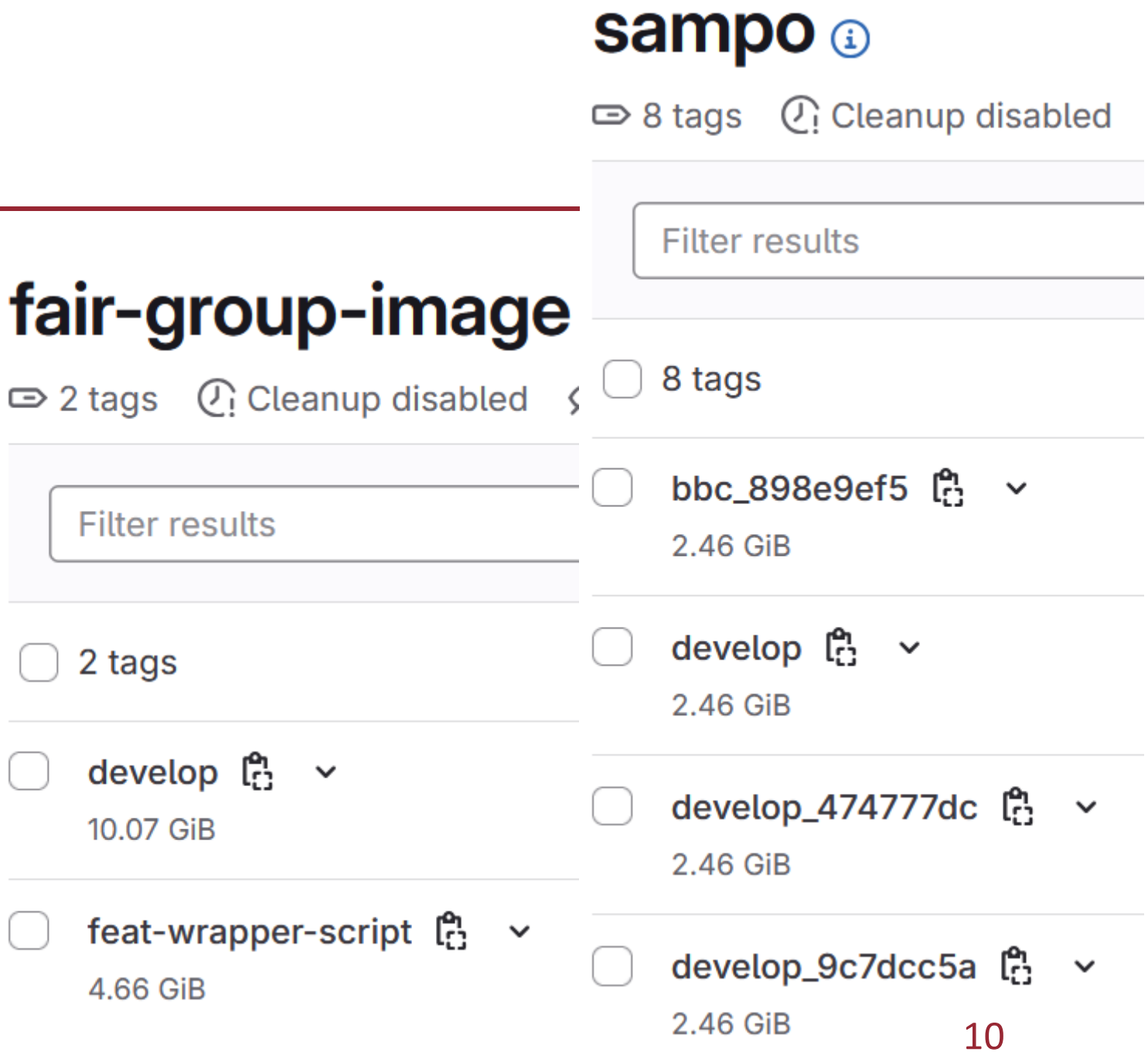
Gitlab Runner

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.

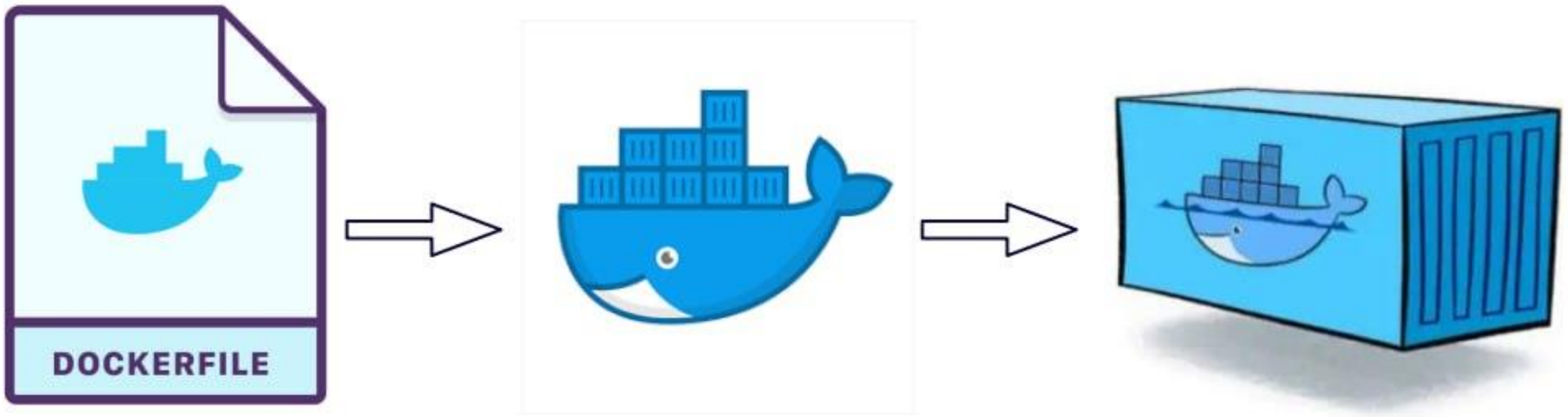
- Execute jobs defined in pipeline;
- Can be installed on any platform;
- Support various executors (including Shell, Docker, and Kubernetes) and environments (local, ssh, clouds);
- Parallel job execution with Runners and data exchange with artifacts (Gitlab artifacts).

Container Registry

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



Basic concepts of containerization

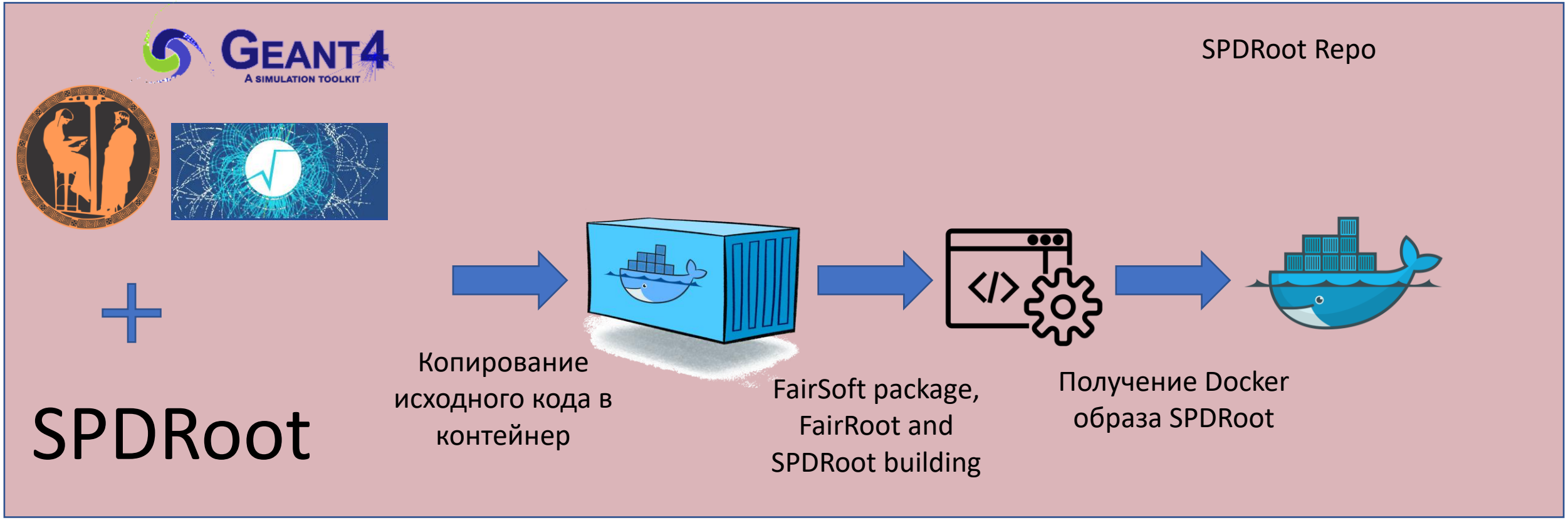


Docker file

Docker Image

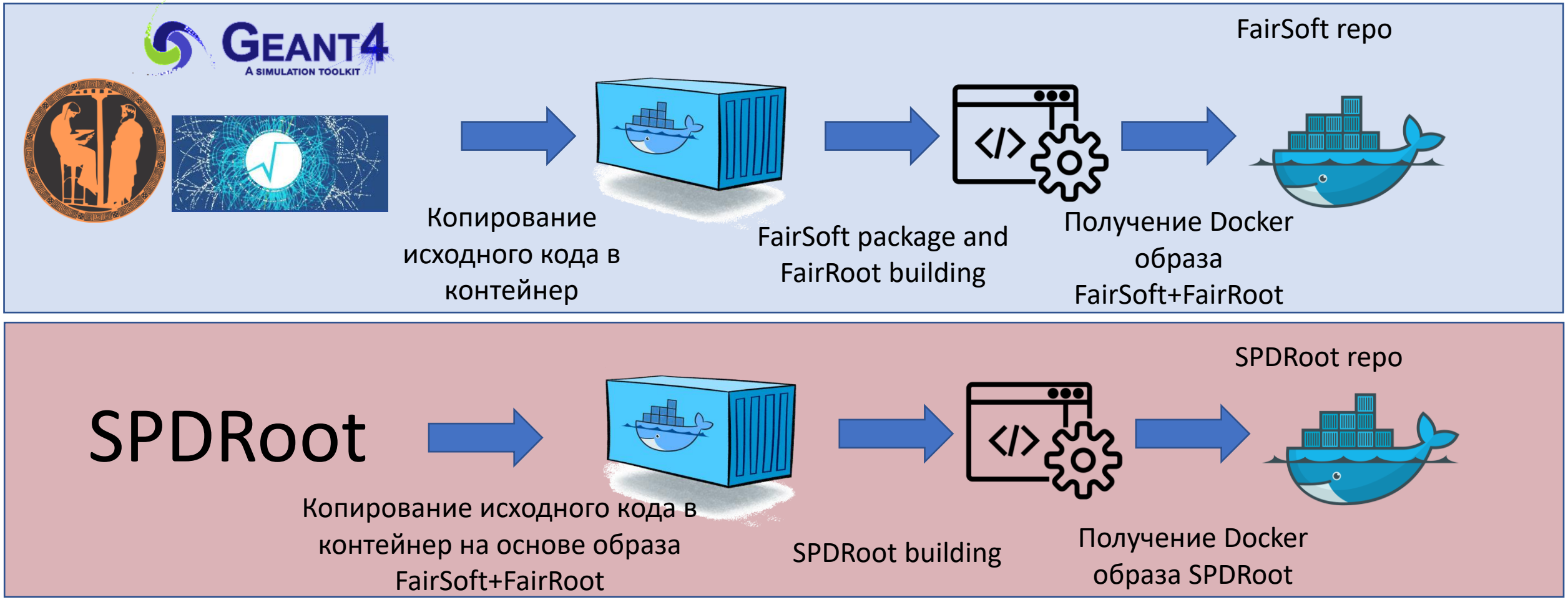
Docker Container

Repositories separation (SPDRoot example)



~ 2.5h every time

Repositories separation (SPDRoot example)

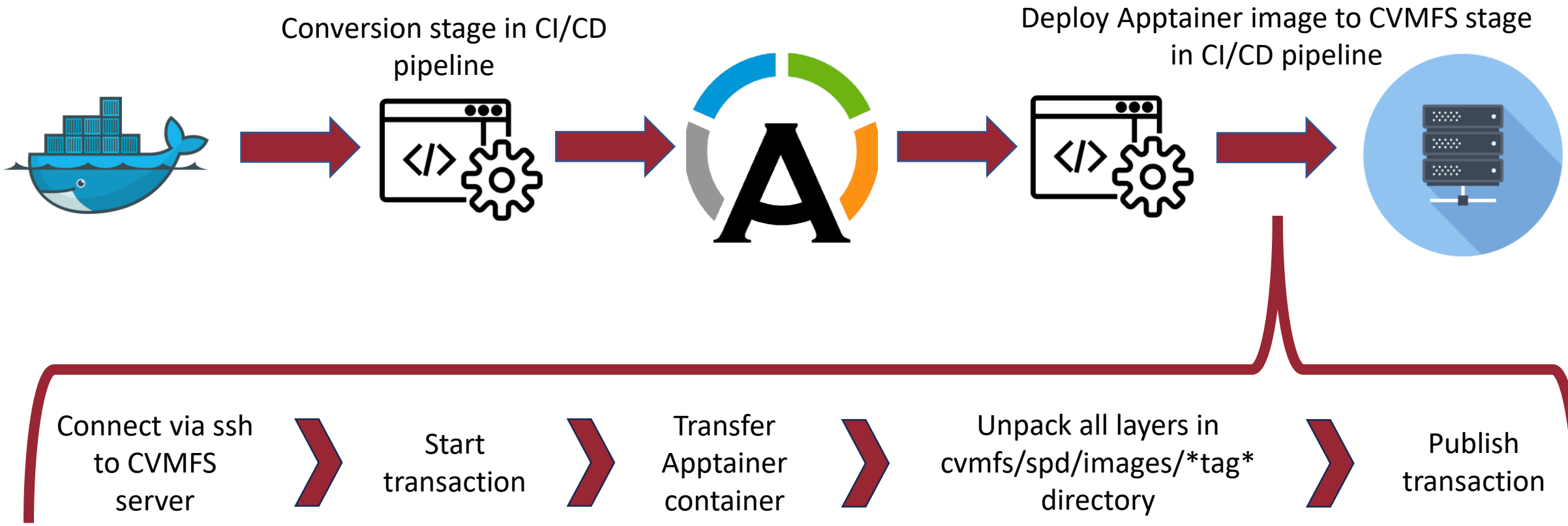


~2h once in base image repo and ~20 mins in prod repo

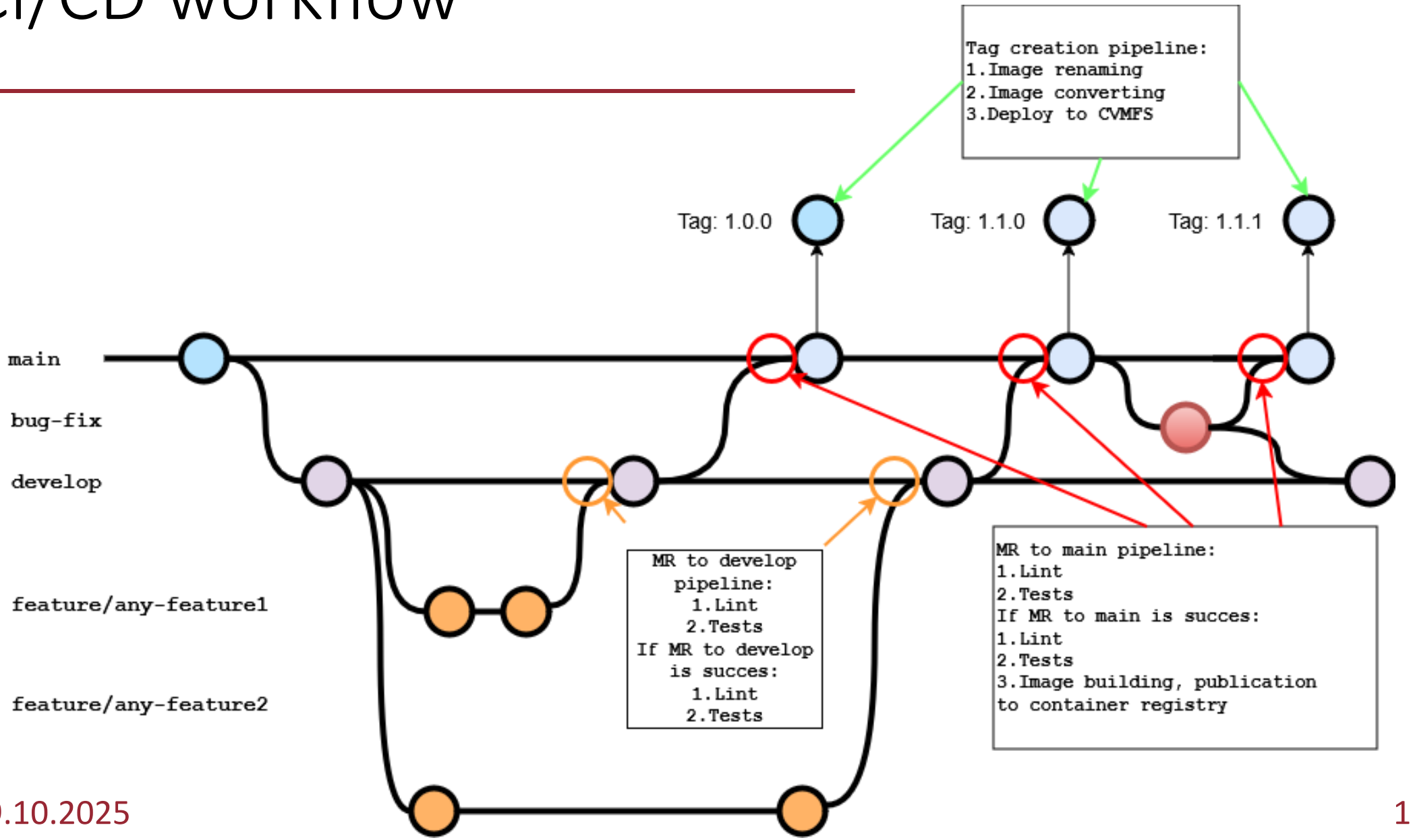
CVMFS

- Scalable, reliable and low- maintenance software distribution service;
- Developed to assist HEP collaborations to deploy software on the worldwide- distributed computing infrastructure used to run data processing applications;
- The containerization standard in this case is Apptainer (Singularity), which was developed specifically for HEP and CVMFS;
- Apptainer format allows to unpack container into individual layers;
- Unpacking allows to reduce the load on the network and production system resources

Deploying to CVMFS



CI/CD workflow



Key Features

- Masked project variables instead of .env files in repositories;
- Docker-Secrets to transfer sensitive data while building stage;
- Base images with its own repositories separated with production code;
- Image building only after successful Merge Request in main/master branch with unified tag: **CONTAINER_REGISTRY:BRANCH_NAME**;
- Already built image retagging automatically after creating tag in repository;

Key Features

- Image conversion to .sif (Apptainer/Singularity format);
- Docker images for personal, debug and development use while Apptainer images for production;
- Unified version control, containerization practices and repository management.

FairRoot Framework and SPDRoot Framework

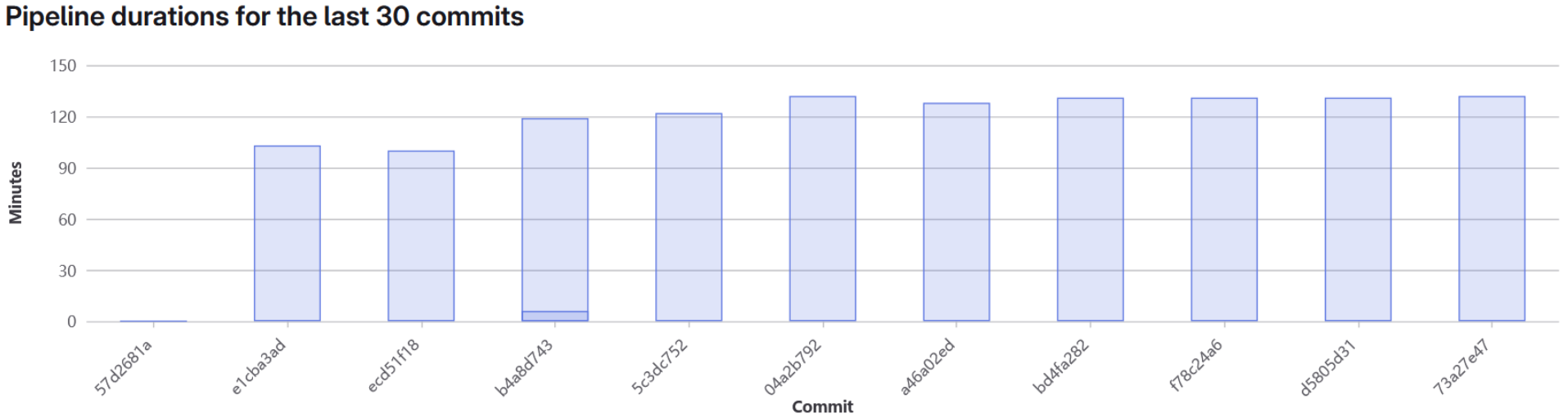
- Separate repositories for the source code of FaitRoot framework and base image designed on it (“fair-group-image”) were created;
- Container Registry images storage was configured;
- CI/CD pipelines are created including automatic:
  - FairRoot framework building inside of a base image “fair-group-image”;
  - 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”;
  - Converting of SPD Root image from Docker to Apptainer format.

Sampo Framework

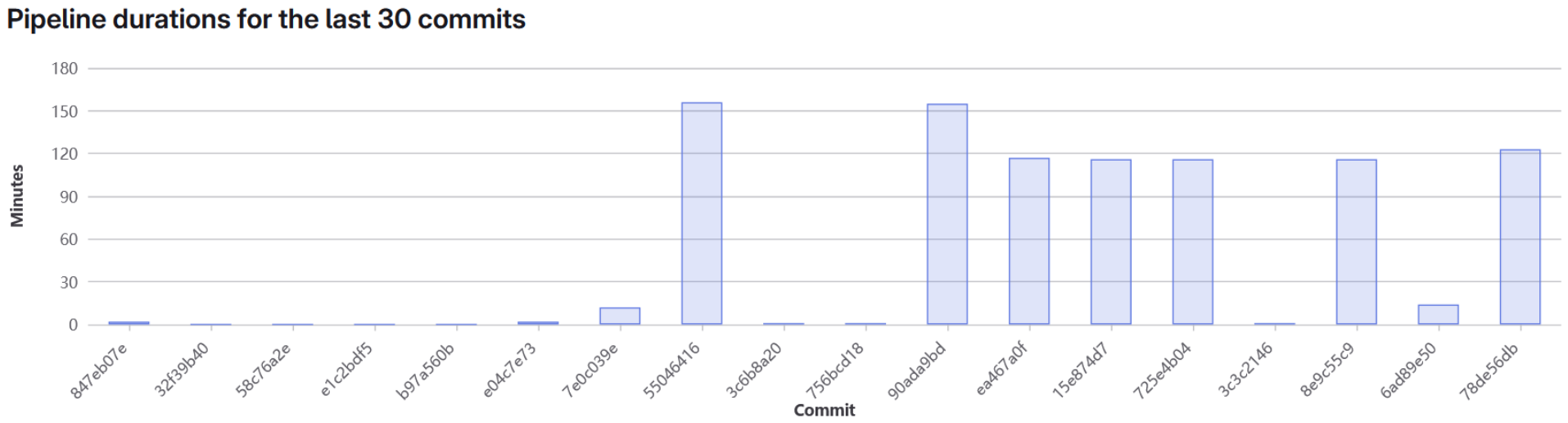
- Separate repository “BISGM” for Sampo’s dependencies and base image containing all of them was created;
- Code-writing approach described within specified methods is implemented;
- Container Registry images storage was configured;
- CI/CD pipelines are created including automatic:
  - Building of Sampo’s dependencies and base image containing all of them with publication to Container Registry of “BISGM” repo;
  - Building of Sampo project and image designed on a base image with publication into Container Registry of “Sampo” repository;
- Sampo container deployment with Apptainer container engine in CVMFS is being developed.

Results

Sampo's dependencies



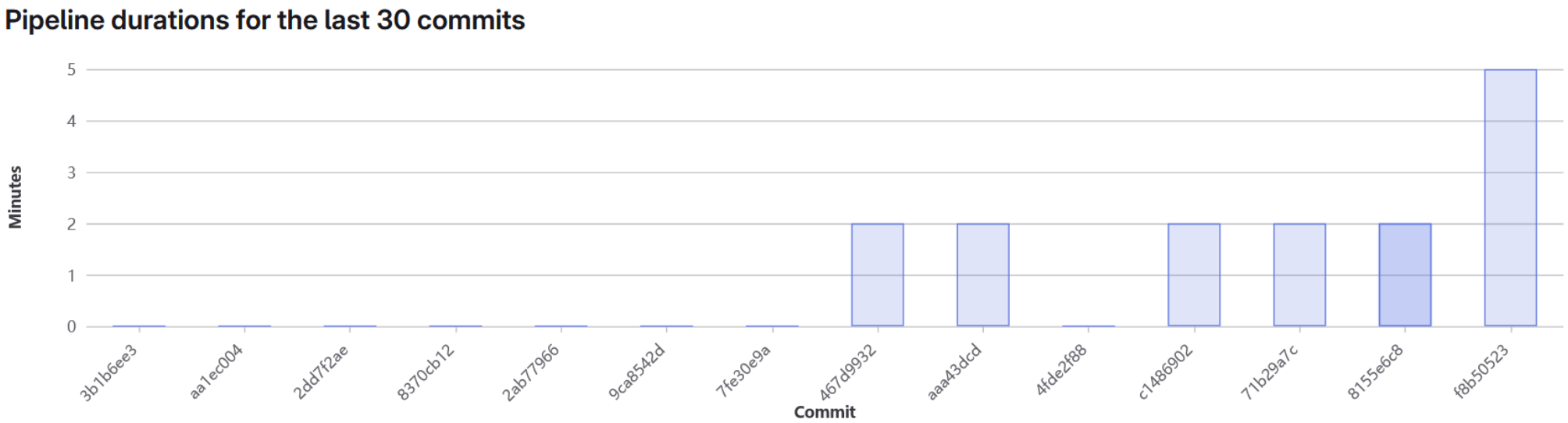
Fair-group-image



29.10.2025

Results

Sampo



Base images (Sampo-base for Sampo and fairsoft-image for SPDRoot**)** building and publication to Container Registry ~120 minutes

Sampo image building and publication to Container Registry ~5 minutes

Conclusion

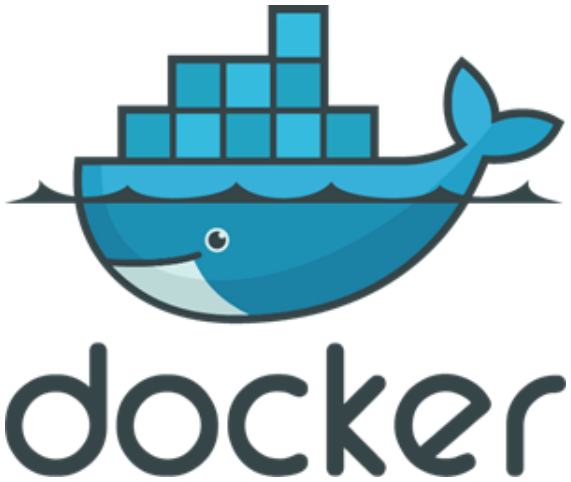
- The proposed configuration is based on international experience and can be saved as standardized templates for use in other projects.
- The methods remain relevant in a diverse range of projects due to its flexibility.
- The carried out work is subject to further development, and it is important to 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

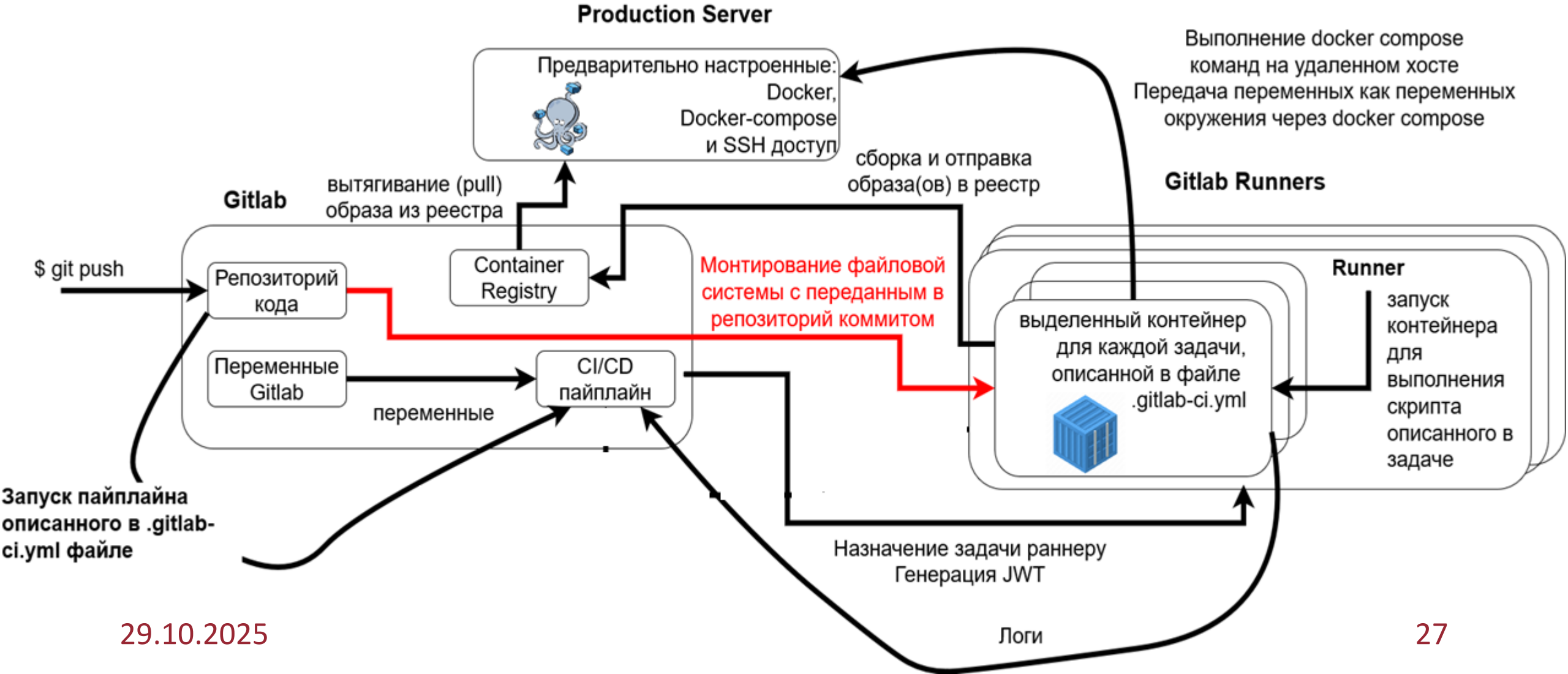
Backup slides

Stack

- Gitlab
- Docker
- Bash scripts
- Yaml scripts



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



CVMFS

- The CernVM File System (CernVM-FS) provides a scalable, reliable and low- maintenance software distribution service. It was developed to assist High Energy Physics (HEP) collaborations to deploy software on the worldwide- distributed computing infrastructure used to run data processing applications.
- The containerization standard in this case is Apptainer (Singularity), which was developed specifically for HEP. After the new image is built and tagged in the Gitlab repository, the Docker image is converted to .sif format, sent to CVMFS, and decompressed into individual layers.
- Unpacking allows you to reduce the load on the network and production system resources, as instead of running a full container with all modules and libraries, you will only run the set of dependencies required for a specific task.

Основные понятия контейнеризации

Docker-образ (Docker Image, Образ) – это неизменяемый файл, содержащий исходный код, библиотеки, зависимости, инструменты и другие файлы, необходимые для запуска приложения. Образ — это шаблон, на основе которого создается контейнер, существует отдельно и не может быть изменен. При запуске контейнерной среды внутри контейнера создается копия файловой системы (docker образа) для чтения и записи.

Контейнер Docker (Docker Container, Контейнер) – это виртуализированная среда выполнения, в которой пользователи могут изолировать приложения от хост-системы. Эти контейнеры представляют собой компактные портативные хосты, в которых можно быстро и легко запустить приложение.

Основные понятия контейнеризации

Базовый Docker-образ – это образ, на основе которого будет создан Docker-образ приложения.

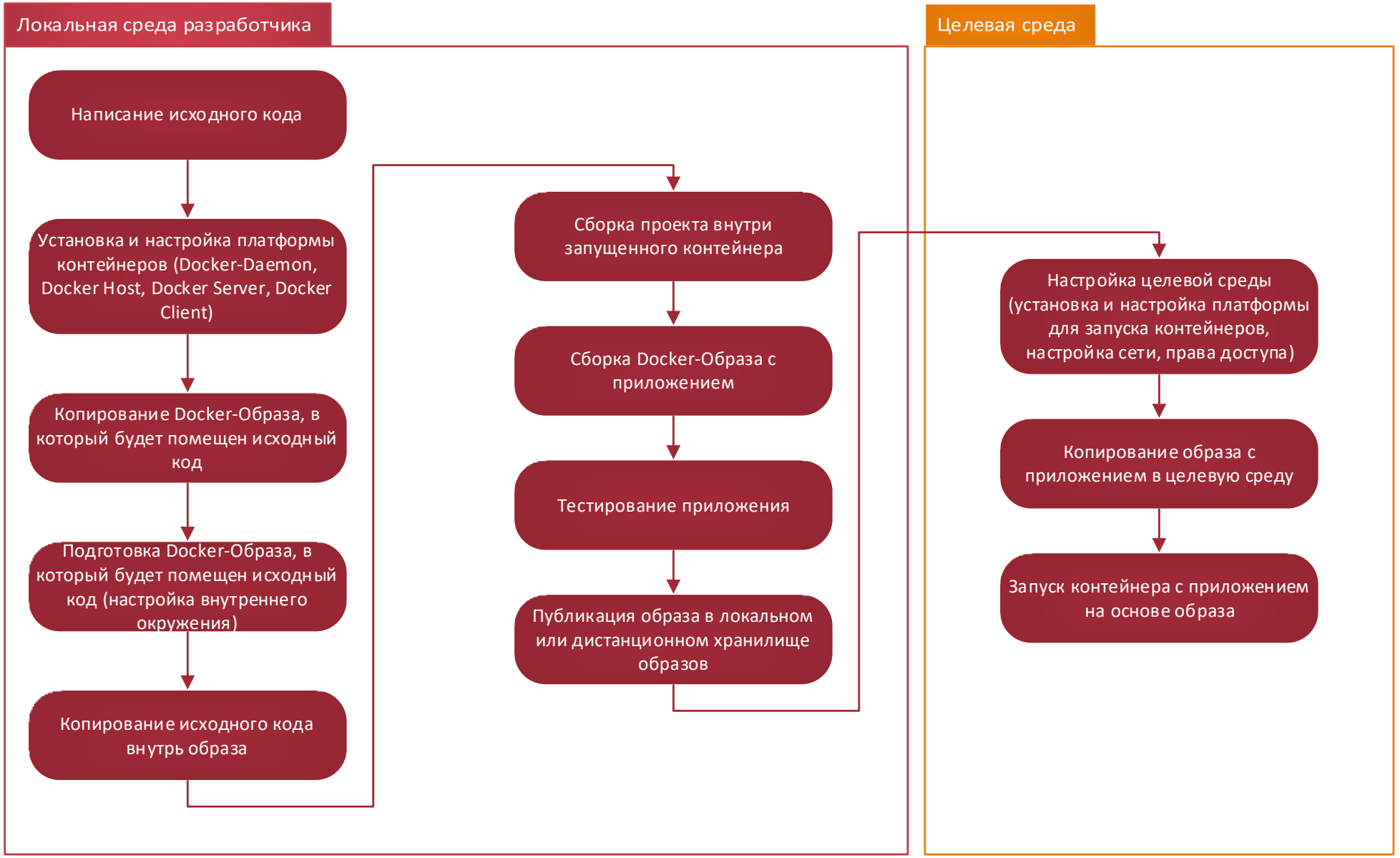
Docker-образ приложения – это образ, основанный на базовом Docker-образе в котором уже собрано приложение, на основе этого образа можно разворачивать контейнер с приложением.

Dockerfile – это текстовый документ формата .yaml, содержащий инструкции для генерации образа Docker. Он указывает Docker, как построить образ, который будет использоваться при создании контейнера.

Разделение кодовых баз

Когда речь касается разработки, например, фреймворка, то подразумевается наличие большого числа установленных библиотек, зависимостей, компиляторов и другого различного ПО, благодаря которому фреймворк может выполнять свои задачи или собираться. Например, фреймворк SPDRoot требует наличие фреймворка FairRoot и пакета ПО FairSoft, что в сумме дает не менее 28 ГБайт. Подготовка окружения занимает порядочное число времени. При сборке Docker-образа существующими на данный момент мощностями выделенными в git.jinr.ru это занимает не менее 2.5 часов. Тогда почему бы не поступить иначе — собранный FairRoot и FairSoft, где уже зафиксированы все версии и не требуются никакие изменения, поместить в контейнеризованную среду, собрать один раз и использовать этот шаблон с собранным софтом на борту при работе с SPDRoot. Таким образом не потребутся каждый раз собирать FairRoot и FairSoft при каждой сборке SPDRoot. Сам же SPDRoot также контейнеризировать, написав образ, базовым для которого станет образ с FairRoot и FairSoft на борту. Время в таком случае сократится до 20 минут.

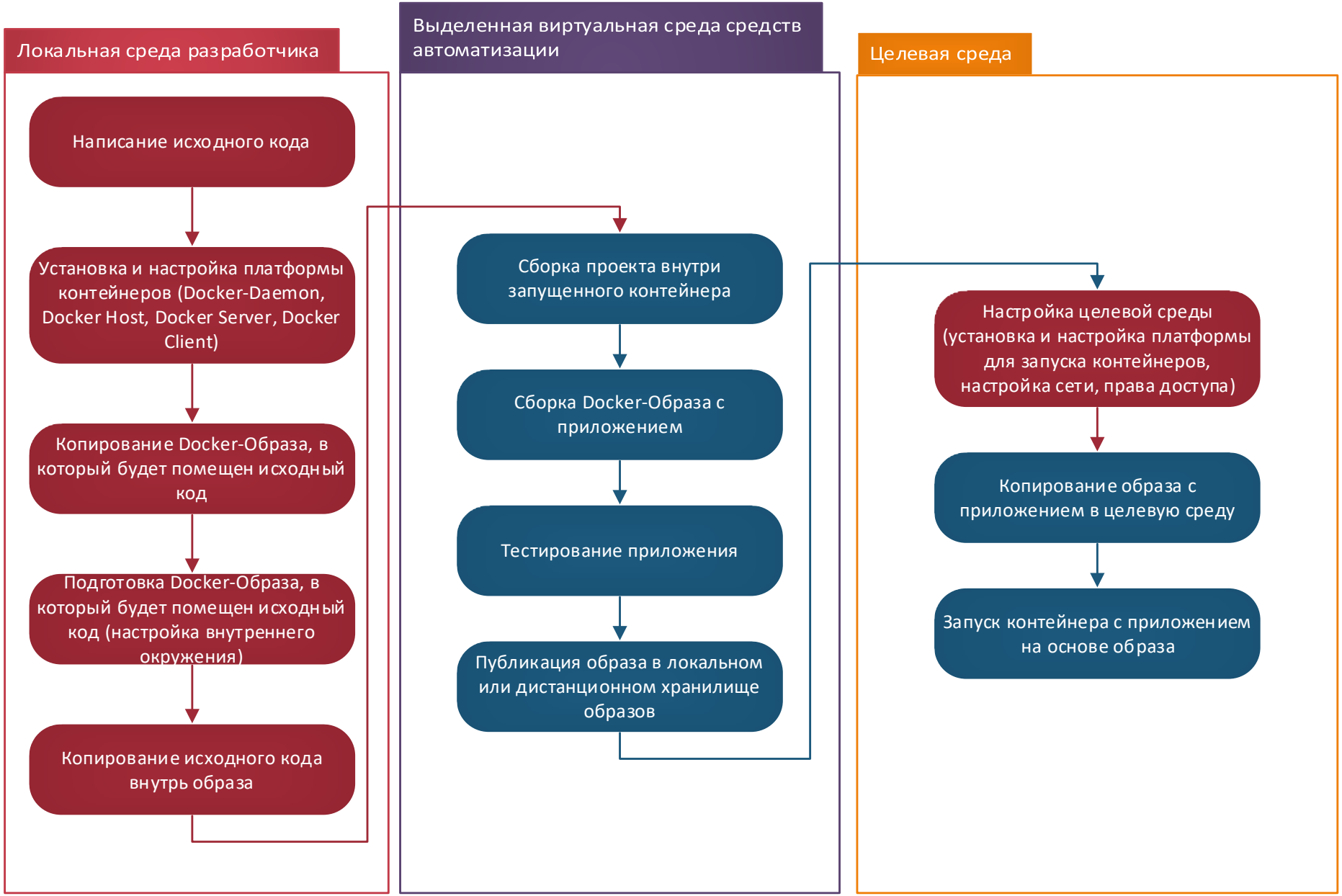
AS-IS



Ручное выполнение

Автоматическое выполнение

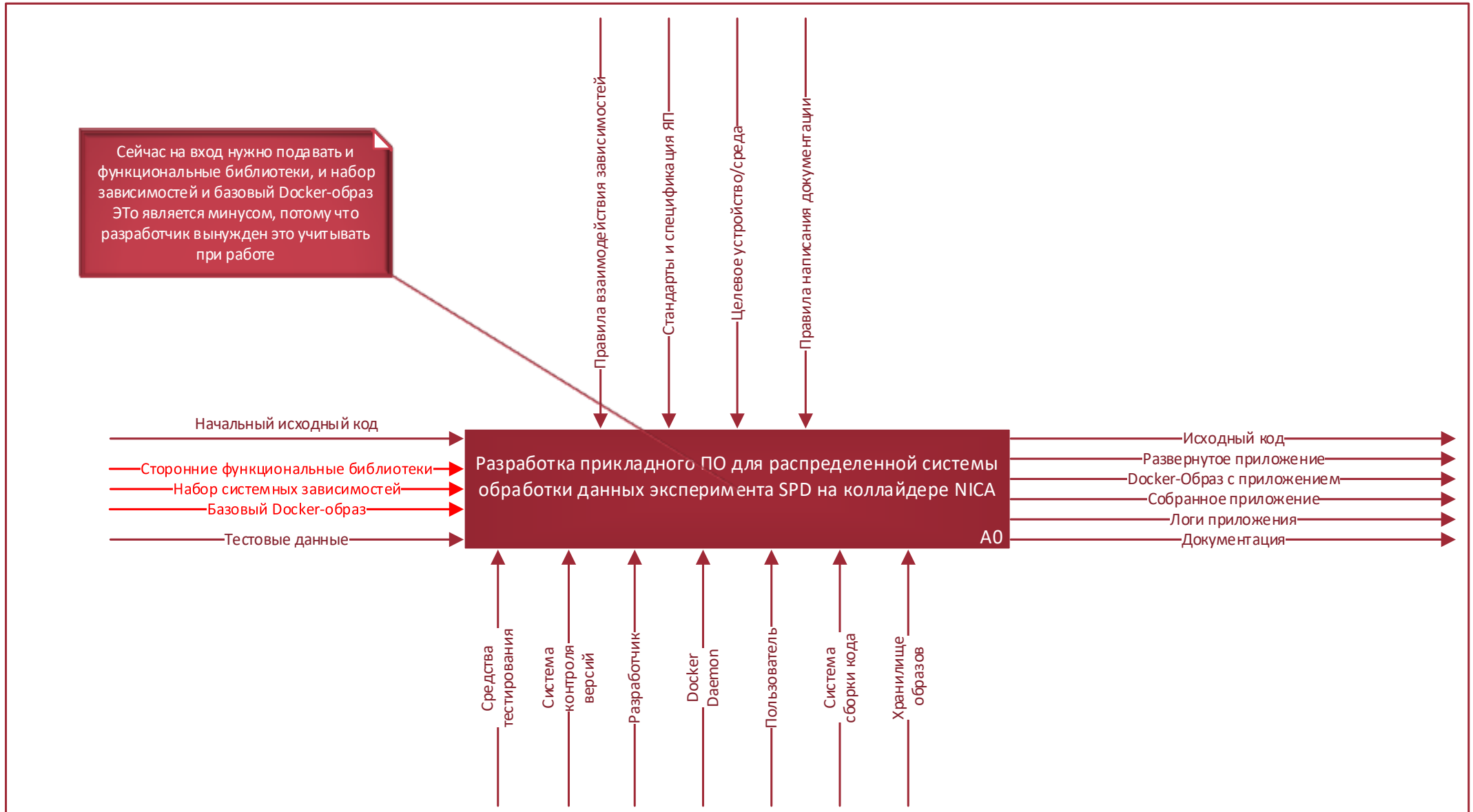
TO-BE



Ручное выполнение

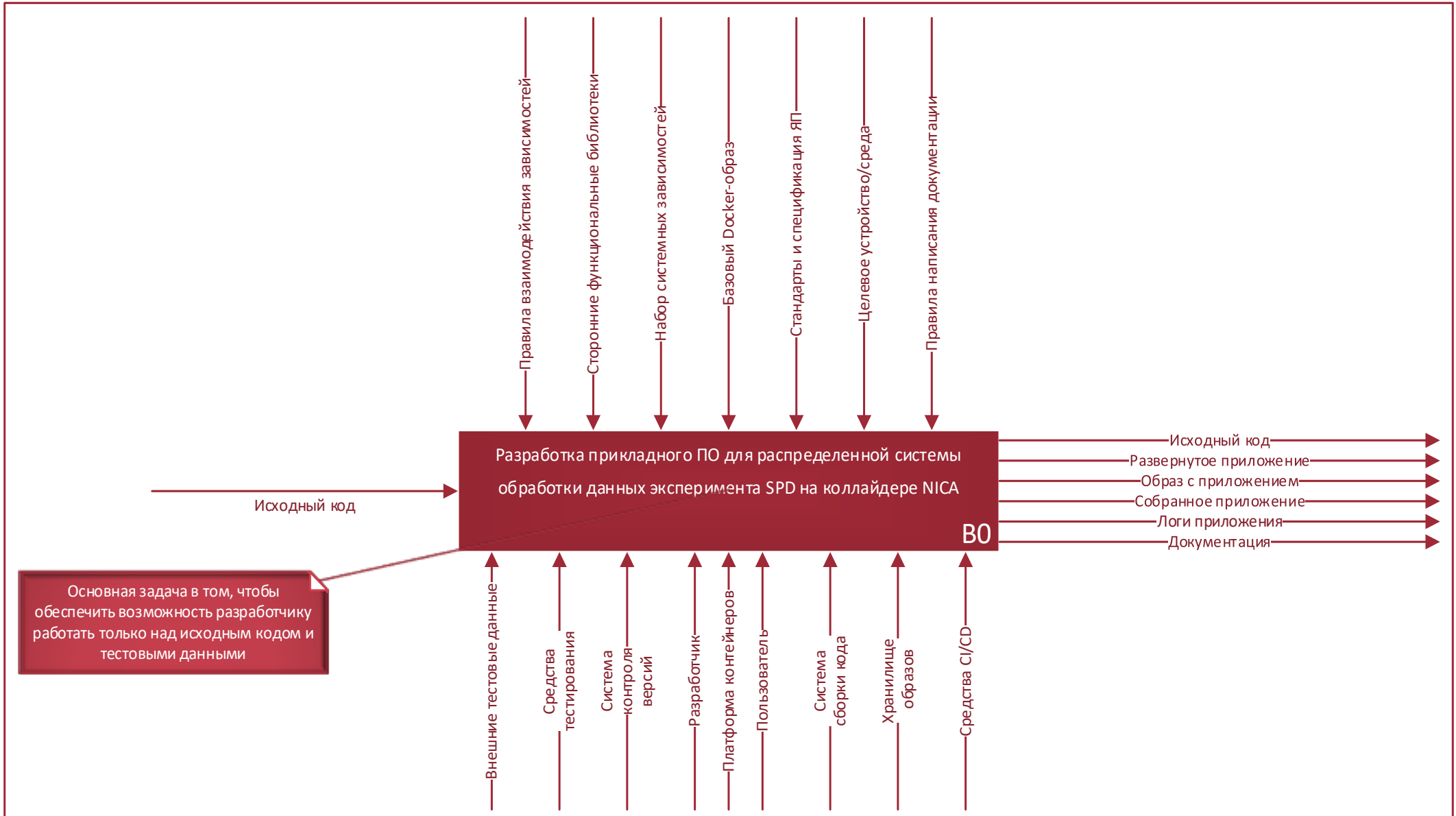
Автоматическое выполнение

AS-IS



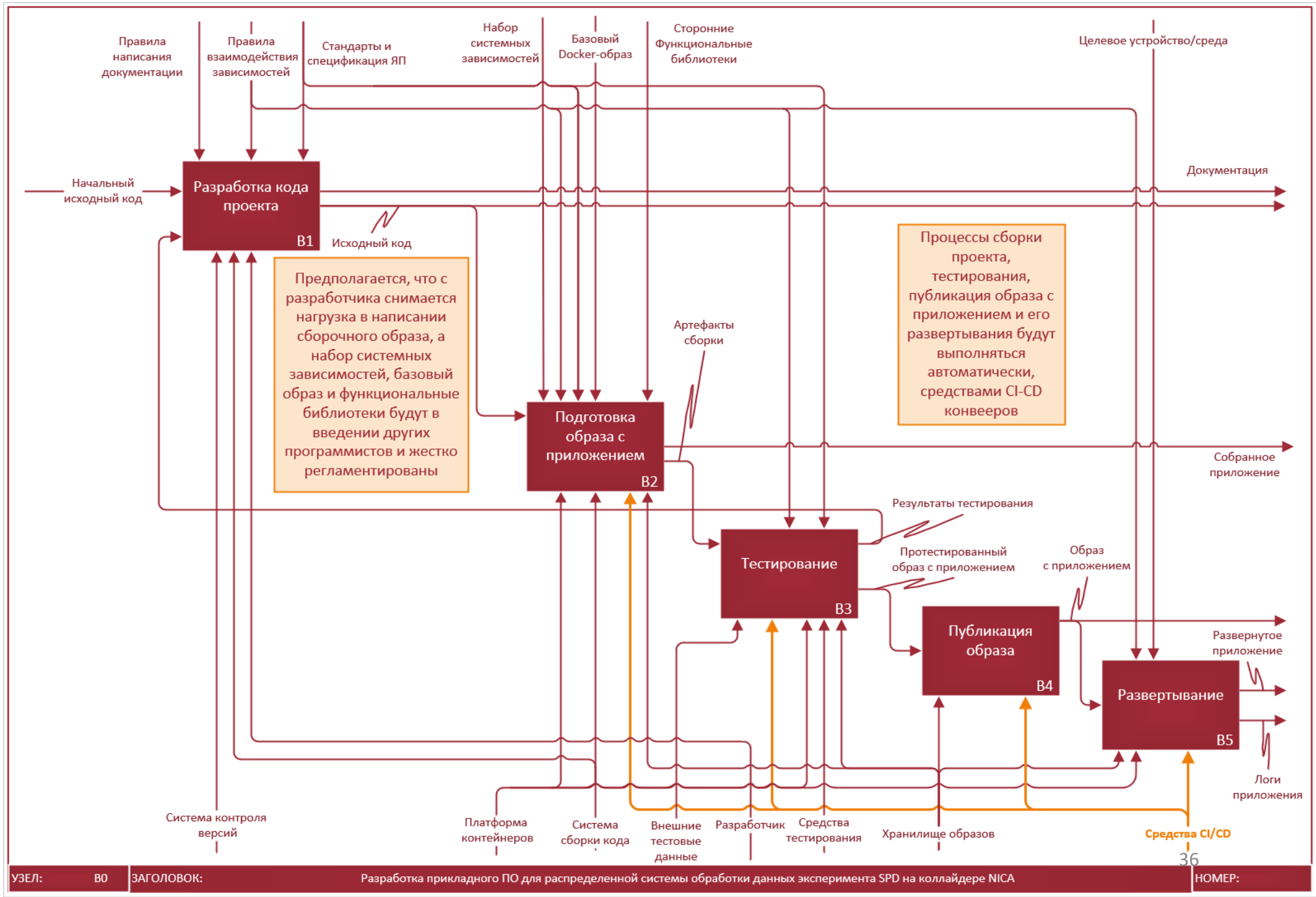
УЗЕЛ:	A0	ЗАГОЛОВОК:	Разработка прикладного ПО для распределенной системы обработки данных эксперимента SPD на коллайдере NICA	НОМЕР:

TO-BE

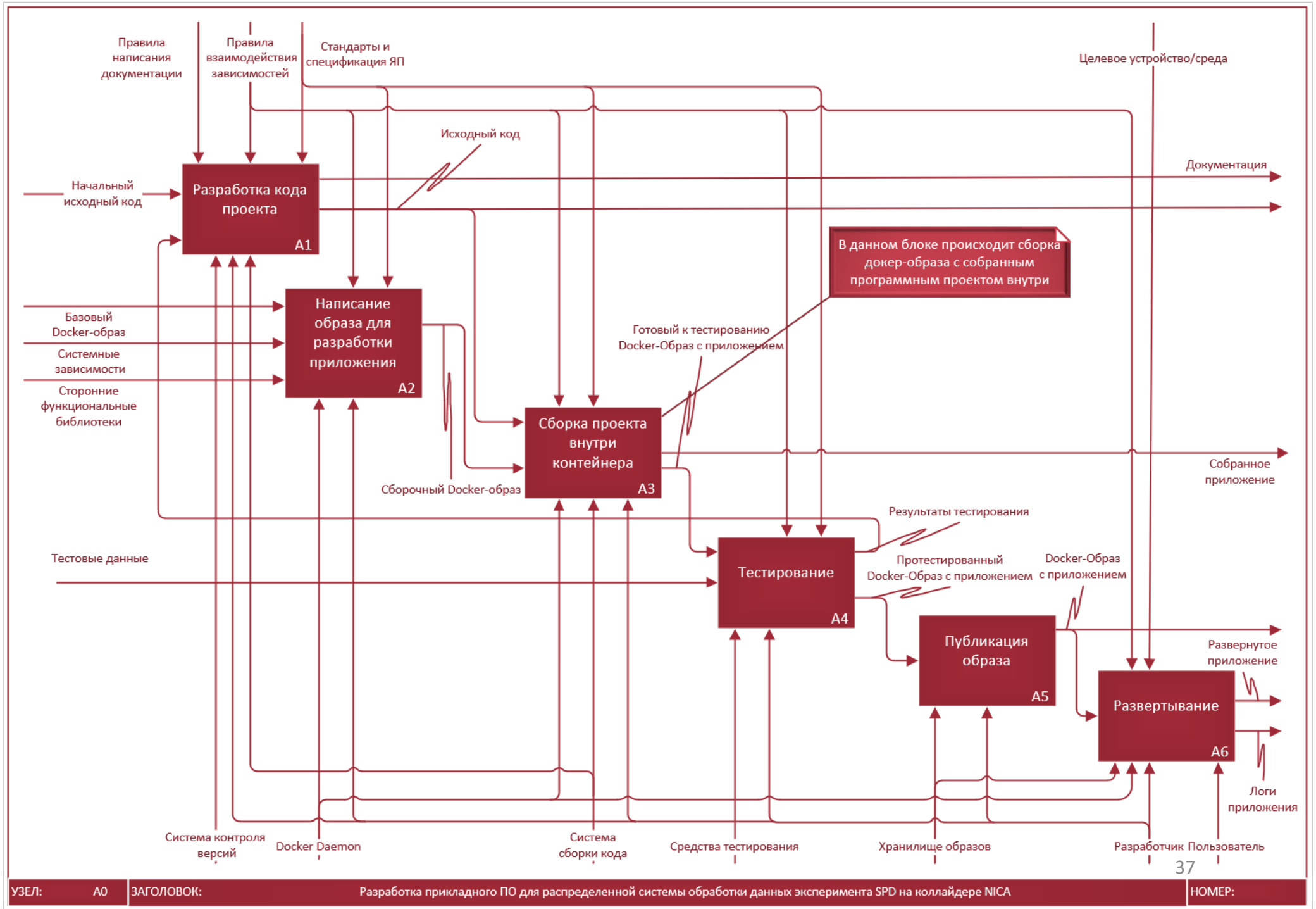


УЗЕЛ:	B0	ЗАГОЛОВОК:	Разработка прикладного ПО для распределенной системы обработки данных эксперимента SPD на коллайдере NICA	НОМЕР:

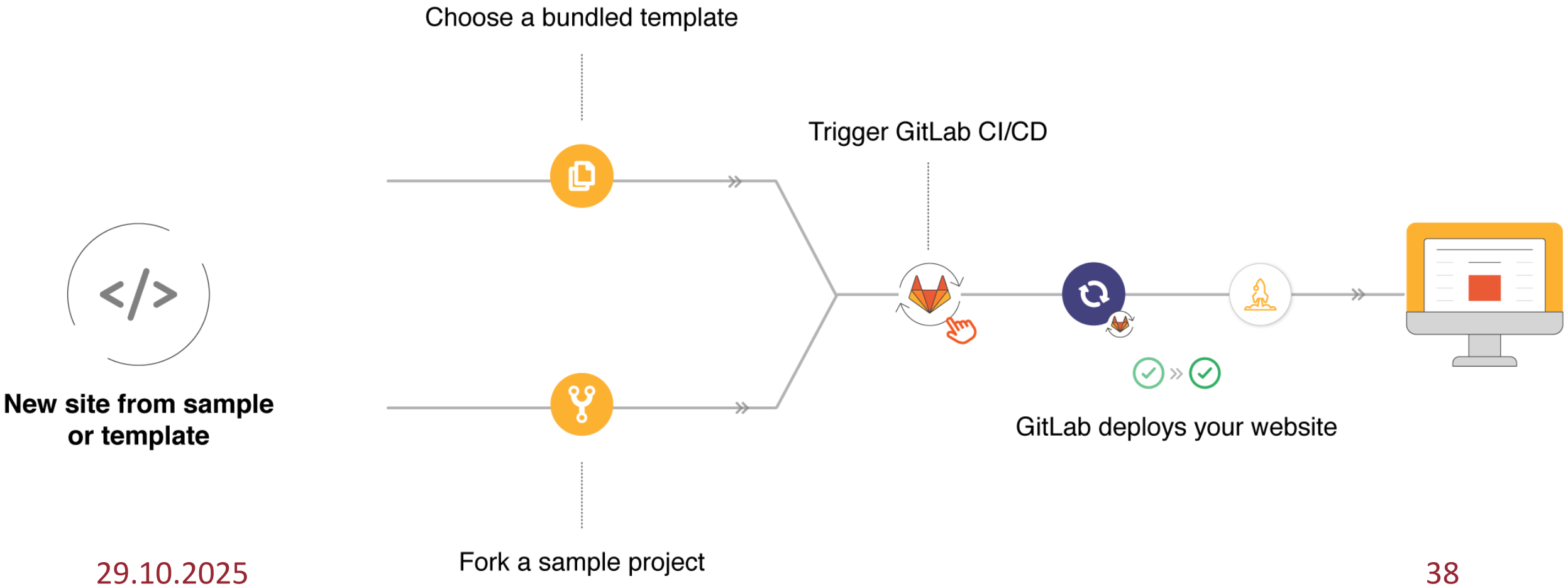
TO-BE


Правила написания документации
Правила взаимодействия зависимостей
Стандарты и спецификация ЯП
Набор системных зависимостей
Базовый Docker-образ
Сторонние Функциональные библиотеки
Целевое устройство/среда
Начальный исходный код
Разработка кода проекта
B1
Исходный код
Документация
Предполагается, что с разработчика снимается нагрузка в написании сборочного образа, а набор системных зависимостей, базовый образ и функциональные библиотеки будут в введении других программистов и жестко регламентированы
Процессы сборки проекта, тестирования, публикация образа с приложением и его развертывания будут выполняться автоматически, средствами CI-CD конвееров
Артефакты сборки
Подготовка образа с приложением
B2
Собранное приложение
Результаты тестирования
Тестирование
B3
Протестированный образ с приложением
Образ с приложением
Публикация образа
B4
Развернутое приложение
Развертывание
B5
Логи приложения
Система контроля версий
Платформа контейнеров
Система сборки кода
Внешние тестовые данные
Разработчик
Средства тестирования
Хранилище образов
Средства CI/CD
УЗЕЛ: B0
ЗАГОЛОВОК: Разработка прикладного ПО для распределенной системы обработки данных эксперимента SPD на коллайдере NICA
НОМЕР:

AS-IS


Правила написания документации
Правила взаимодействия зависимостей
Стандарты и спецификация ЯП
Целевое устройство/среда
Исходный код
Документация
Начальный исходный код
Разработка кода проекта
A1
В данном блоке происходит сборка докер-образа с собранным программным проектом внутри
Базовый Docker-образ
Системные зависимости
Сторонние функциональные библиотеки
Написание образа для разработки приложения
A2
Готовый к тестированию Docker-Образ с приложением
Сборка проекта внутри контейнера
A3
Собранное приложение
Сборочный Docker-образ
Результаты тестирования
Тестовые данные
Тестирование
A4
Протестированный Docker-Образ с приложением
Docker-Образ с приложением
Публикация образа
A5
Развернутое приложение
Развертывание
A6
Логи приложения
Система контроля версий
Docker Daemon
Система сборки кода
Средства тестирования
Хранилище образов
Разработчик
Пользователь
УЗЕЛ: A0
ЗАГОЛОВОК: Разработка прикладного ПО для распределенной системы обработки данных эксперимента SPD на коллайдере NICA
НОМЕР:

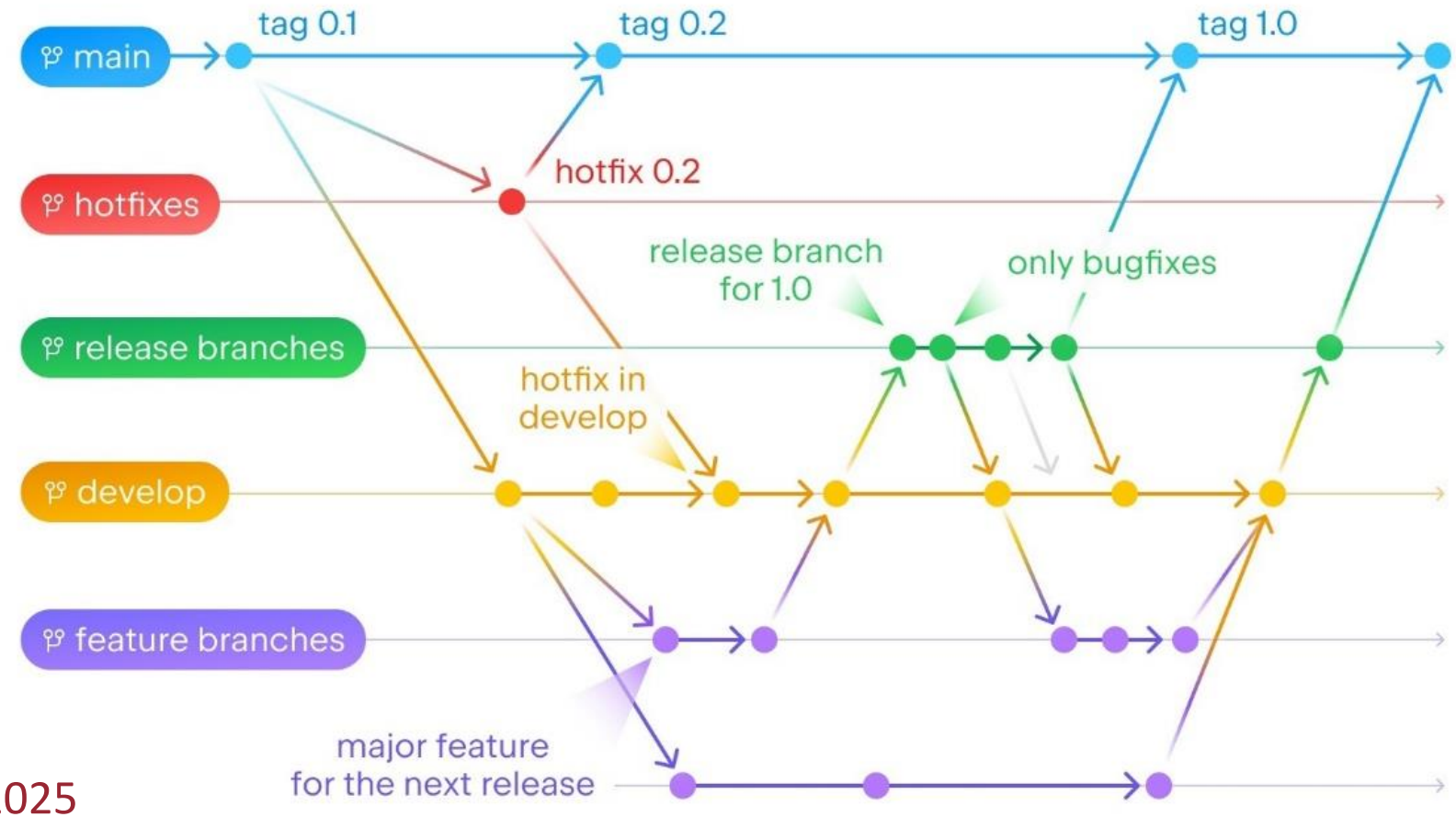
Gitlab Pages



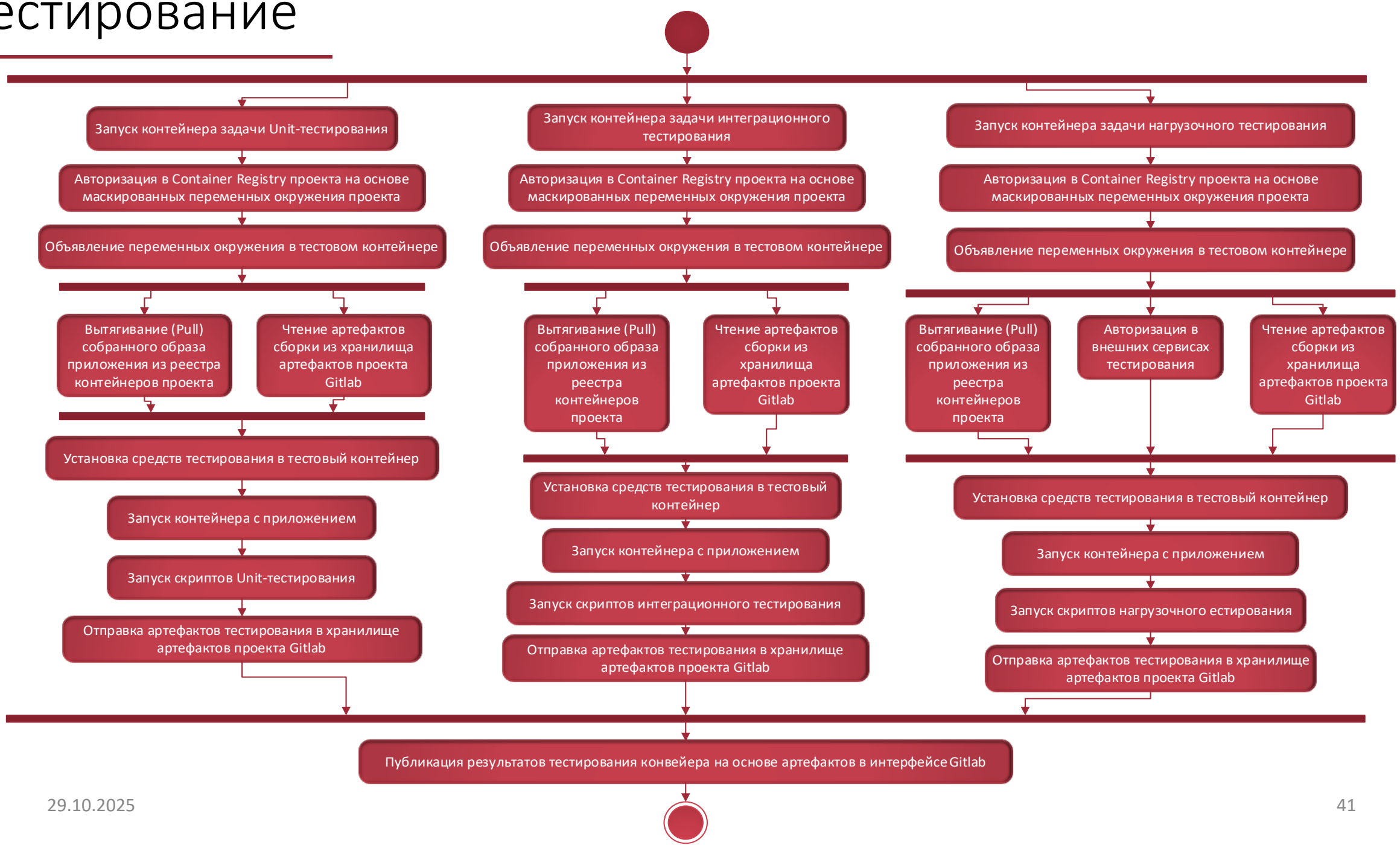
Сборка проекта внутри сборочного контейнера

Практики контроля и управления версиями

GitFlow



Тестирование



Публикация образа

Основные понятия

Docker-образ (Docker Image, Образ) – это неизменяемый файл, содержащий исходный код, библиотеки, зависимости, инструменты и другие файлы, необходимые для запуска приложения. Образ — это шаблон, на основе которого создается контейнер, существует отдельно и не может быть изменен. При запуске контейнерной среды внутри контейнера создается копия файловой системы (docker образа) для чтения и записи.

Контейнер Docker (Docker Container, Контейнер) – это виртуализированная среда выполнения, в которой пользователи могут изолировать приложения от хостовой системы. Эти контейнеры представляют собой компактные портативные хосты, в которых можно быстро и легко запустить приложение.

Приложение (программный продукт) – это разрабатываемое ПО для распределенной вычислительной системы эксперимента SPD.

Глоссарий

Модель AS IS («Как есть) – это модель существующего состояния. Она описывает существующие процессы, то, как они работают и взаимодействуют друг с другом здесь и сейчас. Эта модель позволяет оценить текущее состояние процессов, будь то организационные бизнес-процессы или технологические системные.

Модель TO BE («Как должно быть») – это модель для описания процессов. Она помогает описать идеальное или улучшенное состояние системы. Чаще всего эту модель используют в связке с моделью AS IS. С помощью AS IS фиксируют, как устроена работа в системе. А в TO BE отражают, как устранить недостатки, оптимизировать процессы и внедрить улучшения.

Docker-образ (Docker Image) – это неизменяемый файл, содержащий исходный код, библиотеки, зависимости, инструменты и другие файлы, необходимые для запуска приложения. Образ — это шаблон, на основе которого создается контейнер, существует отдельно и не может быть изменен. При запуске контейнерной среды внутри контейнера создается копия файловой системы (docker образа) для чтения и записи.

Контейнер Docker (Docker Container) – это виртуализированная среда выполнения, в которой пользователи могут изолировать приложения от хостовой системы. Эти контейнеры представляют собой компактные портативные хосты, в которых можно быстро и легко запустить приложение.

Базовый Docker-образ – это образ, на основе которого будет создан сборочный Docker-образ.

Сборочный Docker-образ – это образ, основанный на Базовом Docker-образе, в котором настроено окружение для разработки приложения (установлены необходимые зависимости, библиотеки, скопирован исходный код) и в котором будет вестись разработка.

Docker-образ приложения – это образ, основанный на сборочном Docker-образе в котором уже собрано приложение, на основе этого образа можно разворачивать контейнер с приложением.

Приложение (программный продукт) – это разрабатываемое ПО для распределенной вычислительной системы эксперимента SPD.

Платформа контейнеров (Docker Daemon) – это набор программ, позволяющих взаимодействовать с Docker-образами и контейнерами: передавать или забирать из дистанционного хранилища, запускать и управлять. Состоит из Docker Engine, Docker Client, Docker Host, Docker Daemon.

Начальный исходный код – это текст разрабатываемого Приложения, который требуется доработать или на основе которого будет вестись дальнейшая разработка.

Исходный код приложения – это разработанный код Приложения.

Сторонние функциональные библиотеки (библиотеки) – это набор готовых функций, классов и объектов для решения конкретных задач, написанный третьими лицами или самими разработчиками.

Системные зависимости – это зависимость между частями системы, когда систему рассматривают как единое целое, а также библиотеки, которые зависят от других системных библиотек во время компиляции. Например, CUDA, FFMPEG.

Тестовые данные – это наборы входных данных или информации, используемые для проверки корректности, производительности и надёжности программных систем.

Целевое устройство/среда – это виртуальная и физическая среда и окружение целевого устройства, где планируется развертывание приложения.

Система контроля версий (Gitlab) – это веб-платформа для управления проектами и репозиториями программного кода, работа которой основана на системе контроля версий (Git). Все дистанционные репозиторий для разрабатываемых приложений располагаются на экземпляре Gitlab ОИЯИ – git.jinr.ru.

Dockerfile – это текстовый документ, содержащий инструкции для генерации образа Docker. Он указывает Docker, как построить образ, который будет использоваться при создании контейнера.

Средства тестирования (ПО для тестирования) – вручную прописанные разработчиками приложения или сторонние готовые решения в виде функций, скриптов или ПО для тестирования приложения и Docker-контейнера с приложением.

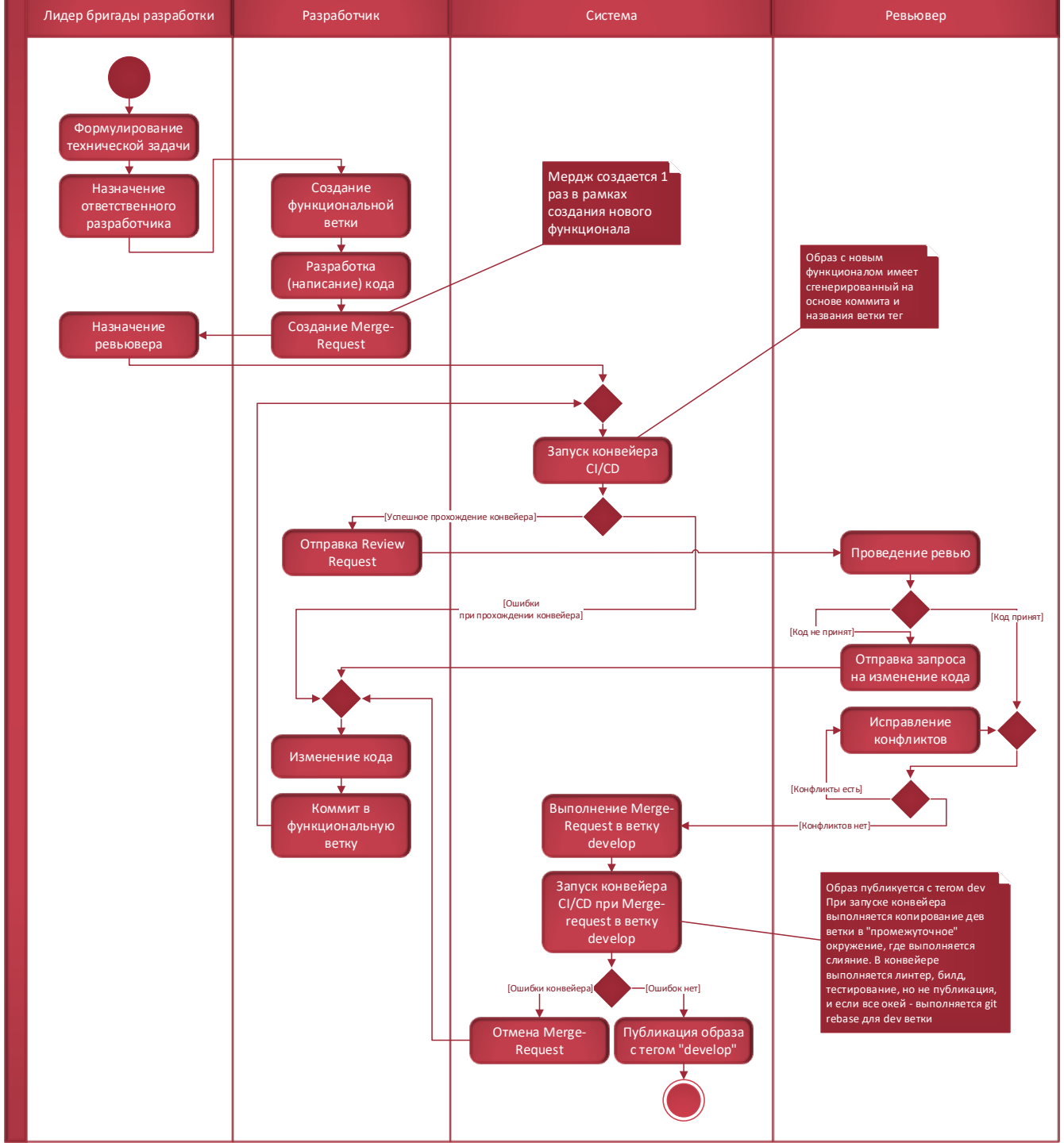
Системы сборки кода – это ПО для компиляции и сборки кода. Например, CCake + Make.

Хранилище образов – виртуальное или физическое хранилище данных формата Docker – образов.

Разработчик – лицо осуществляющее разработку приложения.

Пользователь – физик, другой разработчик.

Процесс внесения изменений в код



Интервьюирование

- Что из себя представляет проект, в котором задействован разработчик, в чем состоит цель проекта, и как давно проект существует?
- Какие шаги выполняются разработчиком, когда поступает запрос на внесение изменений в программный код?
- Какие технологии при этом задействованы?
- Сколько по времени занимают все процессы, кроме непосредственного внесения изменения в программный код?
- Какие, по мнению разработчика, существуют проблемы в процессе разработки проекта?

Интервью было проведено с разработчиками:

- фреймворка Gaudi (Sampo);
- фреймворка SPDRoot;
- сервиса SPD EventIndex;
- сервиса SPD Metadata DB;
- сервиса SPD Hardware DB.

DevOps

Методология включает в себя набор best practices – «лучших практик», направленных на ускорение внедрения изменений и выпуска продукта, автоматизацию процессов сборки, тестирования и доставки, улучшение координации между командами разработчиков:

- непрерывная интеграция и непрерывная доставка;
- управление версиями;
- непрерывное тестирование;
- непрерывное наблюдение;
- инфраструктура как код;
- микросервисная архитектура приложений.