

Развёртывание компонент ИС метаданных SPD

Докладчик (Университет «Дубна»): Короткин Ринат Наильевич – студент 1 курса магистратуры, ПМИ ИСАУ

Научный руководитель (Университет «Дубна»): Прогулова Татьяна Борисовна – к.т.н., доцент, ИСАУ

Научный руководитель (ОИЯИ): Прокошин Фёдор Валерьевич – к.ф.-м.н., с.н.с., ЛЯП

Описание проблемной ситуации

Так как была выбрана концепция микросервисной архитектуры, то для эффективной разработки, поддержки системы и ее развертывания требуется автоматизация некоторых процессов (сборка, развертывание, тестирование).

Было принято решение использовать подход CI/CD и инструмент для его реализации – GitLab.

Задачи

1. Изучение проектируемых информационных систем эксперимента SPD, модели данных, а также реализации сходных систем в других экспериментах;
2. Выбор набора программных средств на основе имеющихся требований к развертыванию микросервисов;
3. Автоматизация процессов сборки, развертывания, генерации документации;
4. Автоматизация юнит-тестирования и нагрузочного тестирования;
5. Апробация настроенных инструментов автоматизации;
6. Внедрение практик автоматизации сборки, развертывания, тестирования в другие группы и подразделения работающие над ПО для SPD.

Используемые технологии

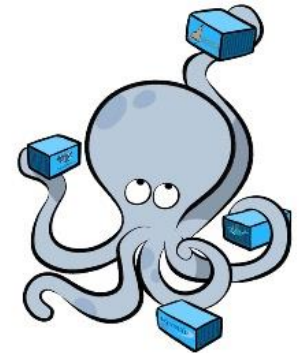
- Gitlab
- Docker
- Docker compose
- Sphinx python documentation
- HashiCorp Vault



GitLab



docker



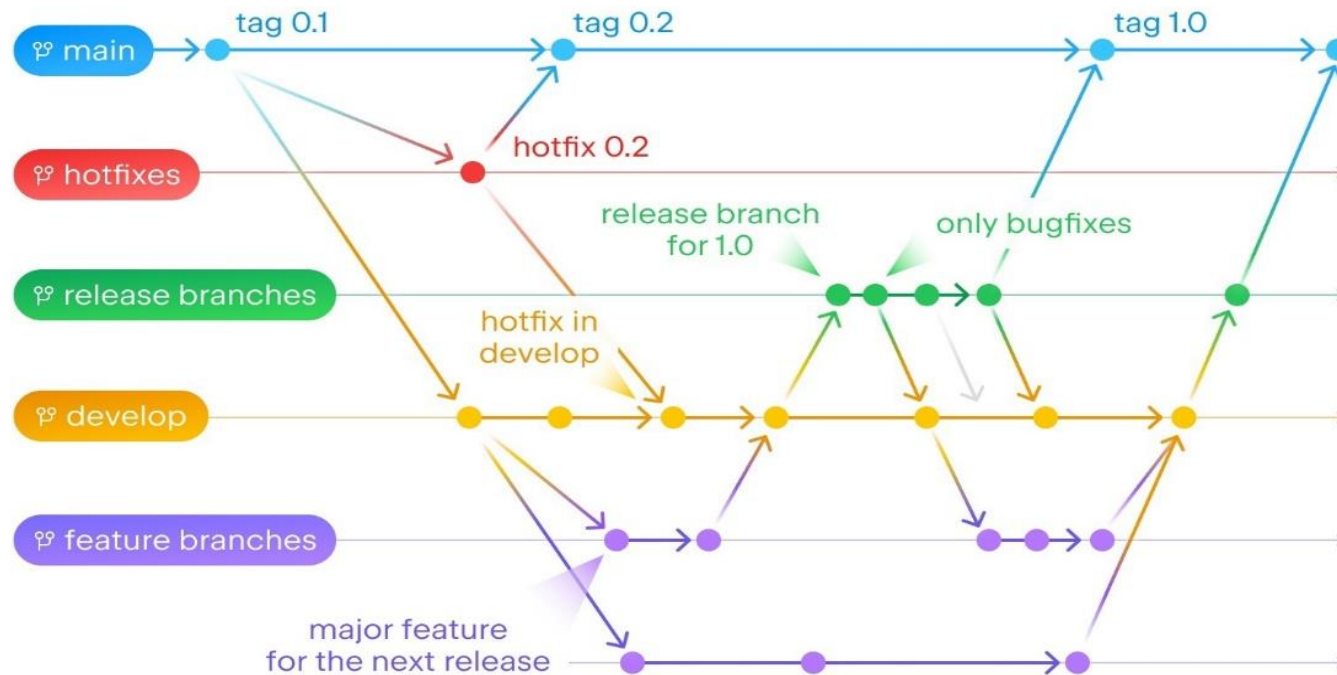
Vault



Ключевыми особенностями GitLab являются непрерывность процесса разработки и взаимная интеграция различных элементов.

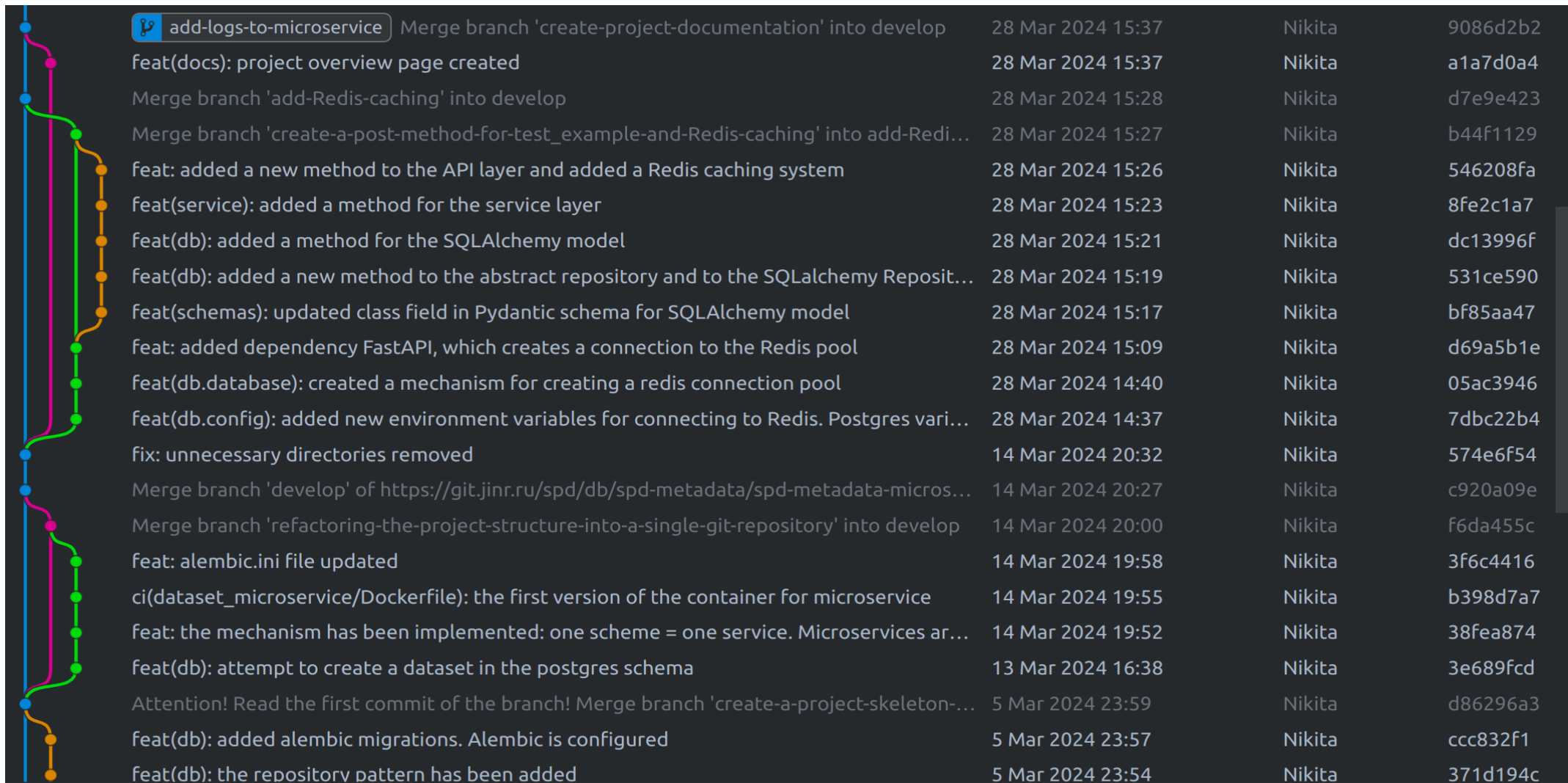
- Локальное развертывание (`jinr.gitlab`);
- Фичи по CI/CD (Gitlab CI/CD);
- Автоматическое обнаружение секретов и тестирование безопасности;
- Явные разрешения для ограничения слияния и отправки кода определенным пользователям;
- Бесплатные статические веб-сайты в репозиториях Git (Gitlab Pages);

GitFlow




- The `main` branch is for production code only.
- The `develop` branch is for development code.
- `feature` branches are created from the `develop` branch.
- `hotfix` branches are created from the `main` branch.
- `release` branches are created from the `develop` branch.

GitFlow



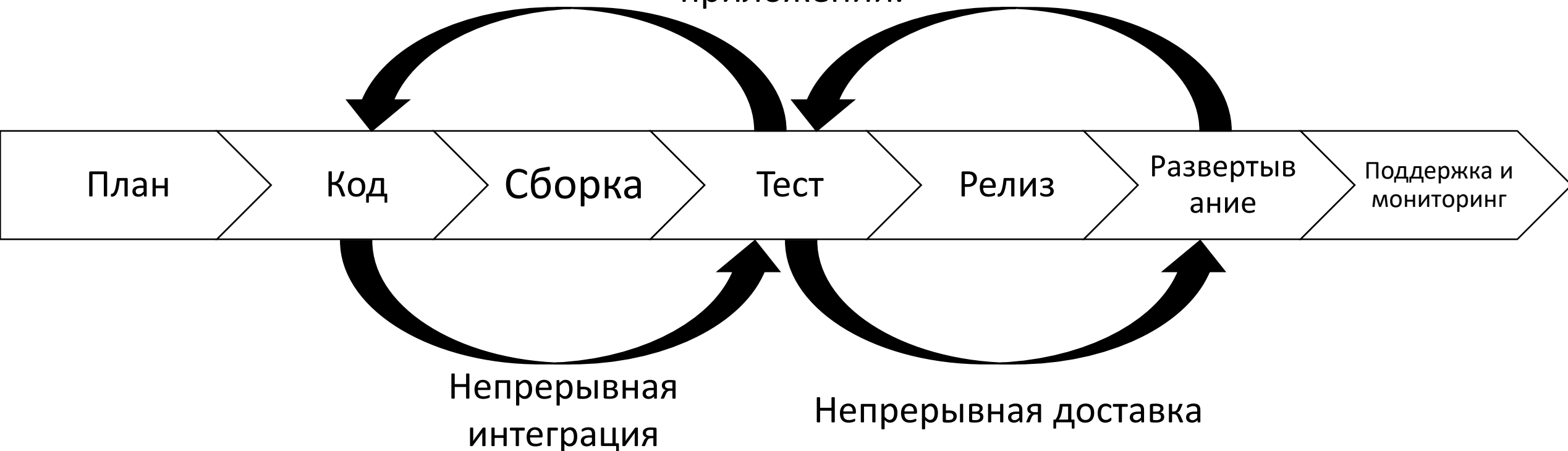
The image shows a Git commit history visualization. On the left, a vertical timeline represents the development process, with colored lines indicating branches and their merges. The main part of the image is a table of commit details.

 add-logs-to-microservice	Merge branch 'create-project-documentation' into develop	28 Mar 2024 15:37	Nikita	9086d2b2
feat(docs): project overview page created		28 Mar 2024 15:37	Nikita	a1a7d0a4
Merge branch 'add-Redis-caching' into develop		28 Mar 2024 15:28	Nikita	d7e9e423
Merge branch 'create-a-post-method-for-test_example-and-Redis-caching' into add-Redi...		28 Mar 2024 15:27	Nikita	b44f1129
feat: added a new method to the API layer and added a Redis caching system		28 Mar 2024 15:26	Nikita	546208fa
feat(service): added a method for the service layer		28 Mar 2024 15:23	Nikita	8fe2c1a7
feat(db): added a method for the SQLAlchemy model		28 Mar 2024 15:21	Nikita	dc13996f
feat(db): added a new method to the abstract repository and to the SQLAlchemy Reposit...		28 Mar 2024 15:19	Nikita	531ce590
feat(schemas): updated class field in Pydantic schema for SQLAlchemy model		28 Mar 2024 15:17	Nikita	bf85aa47
feat: added dependency FastAPI, which creates a connection to the Redis pool		28 Mar 2024 15:09	Nikita	d69a5b1e
feat(db.database): created a mechanism for creating a redis connection pool		28 Mar 2024 14:40	Nikita	05ac3946
feat(db.config): added new environment variables for connecting to Redis. Postgres vari...		28 Mar 2024 14:37	Nikita	7dbc22b4
fix: unnecessary directories removed		14 Mar 2024 20:32	Nikita	574e6f54
Merge branch 'develop' of https://git.jinr.ru/spd/db/spd-metadata/spd-metadata-micros...		14 Mar 2024 20:27	Nikita	c920a09e
Merge branch 'refactoring-the-project-structure-into-a-single-git-repository' into develop		14 Mar 2024 20:00	Nikita	f6da455c
feat: alembic.ini file updated		14 Mar 2024 19:58	Nikita	3f6c4416
ci(dataset_microservice/Dockerfile): the first version of the container for microservice		14 Mar 2024 19:55	Nikita	b398d7a7
feat: the mechanism has been implemented: one scheme = one service. Microservices ar...		14 Mar 2024 19:52	Nikita	38fea874
feat(db): attempt to create a dataset in the postgres schema		13 Mar 2024 16:38	Nikita	3e689fcd
Attention! Read the first commit of the branch! Merge branch 'create-a-project-skeleton-...		5 Mar 2024 23:59	Nikita	d86296a3
feat(db): added alembic migrations. Alembic is configured		5 Mar 2024 23:57	Nikita	ccc832f1
feat(db): the repository pattern has been added		5 Mar 2024 23:54	Nikita	371d194c

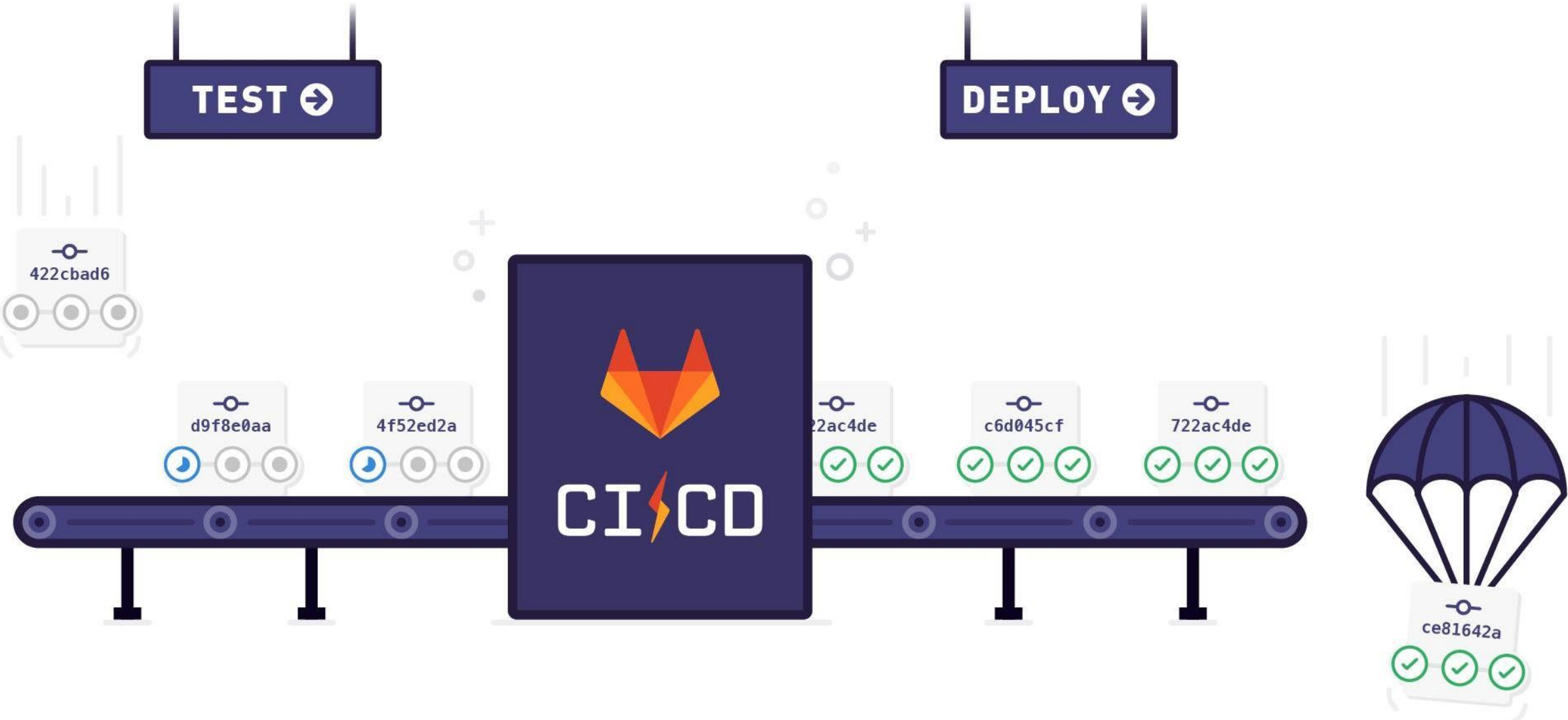
Continuous Integration Continuous Development

Цель CI/CD — ускорить обнаружение дефектов, повысить производительность и обеспечить более быстрые циклы выпуска.

CI/CD объединяет разработку, тестирование и развёртывание приложений.



Gitlab CI/CD



Gitlab CI/CD концептуально

Commit,
merge-
request

Выбор задачи в
соответствии с
конфигурацией

Организация
задач с
этапами

Выполнение
этапов по
очереди

Удачное
завершение
всех задач и
этапов =
пайплайн
успешен

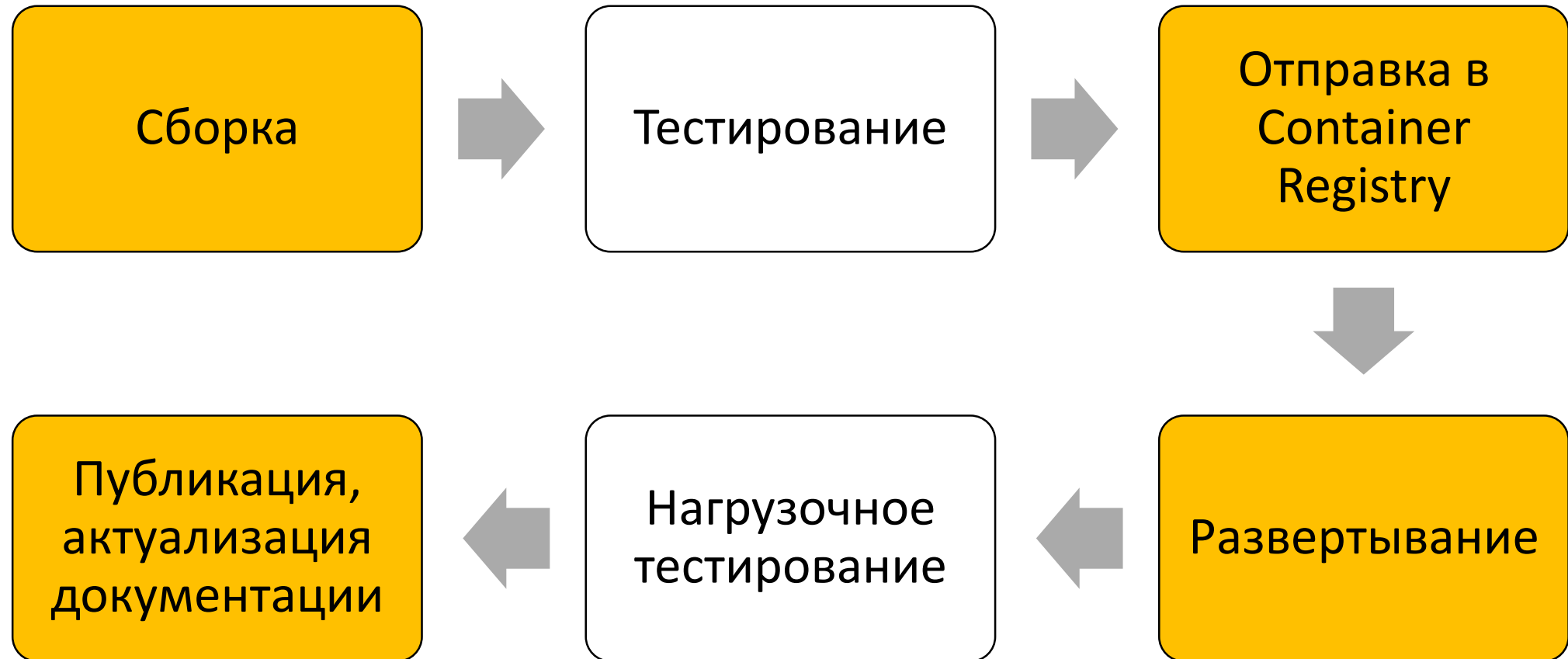
Остановка
пайплайна
если задача
или этап
завершаются
неудачей

Gitlab CI/CD концептуально

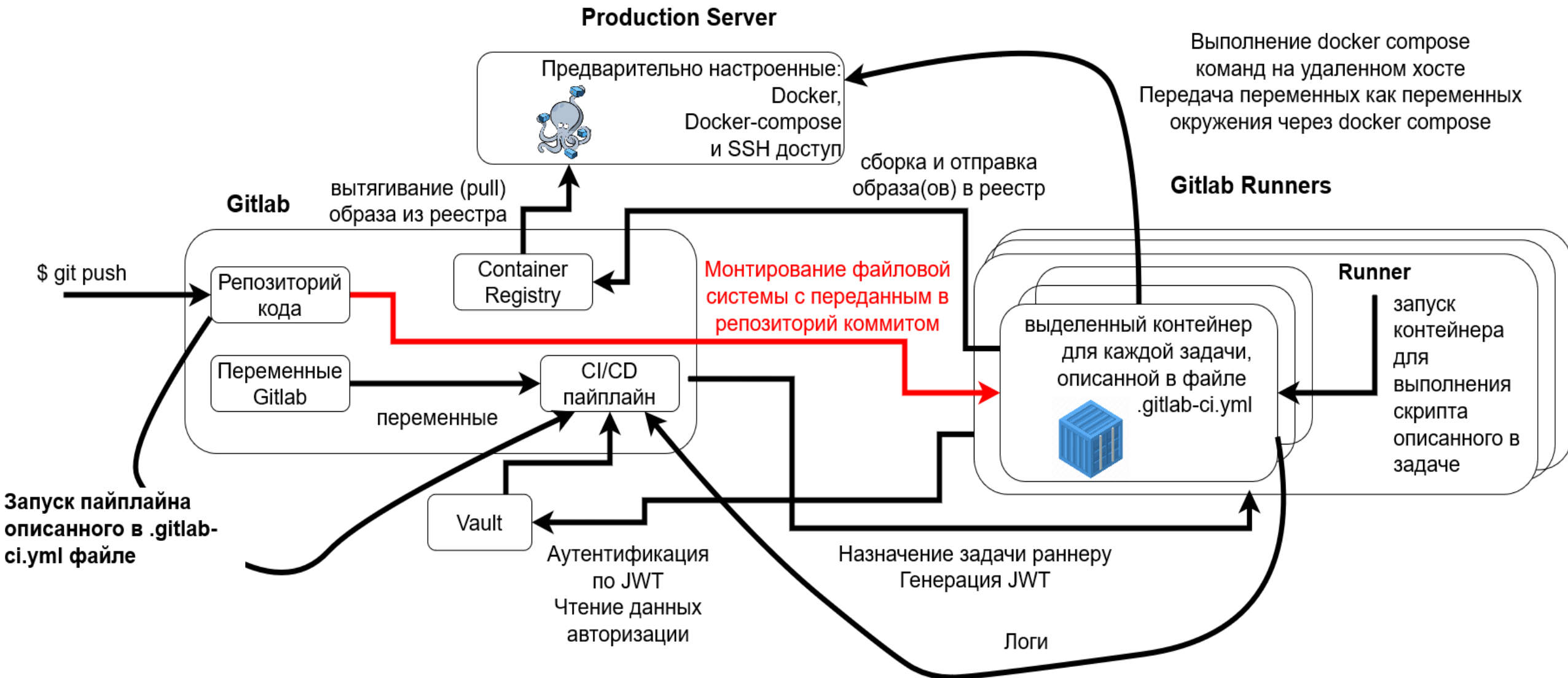
- пайплайн (**pipeline**) — набор задач, организованных в этапы, в котором можно собрать, протестировать, упаковать код, развернуть готовую сборку в облачный сервис и тд.;
- этап (**stage**) — единица организации пайплайна, содержит как минимум 1 задачу;
- задача (**job**) — единица работы в пайплайне. Состоит из скрипта (обязательно), условий запуска, настроек публикации/кеширования артефактов и много другого.

Соответственно, настройка CI/CD сводится к тому, чтобы создать набор задач, реализующих все необходимые действия для сборки, тестирования и публикации кода и артефактов.

Пример нашего пайплайна



Gitlab CI/CD



Gitlab Runner




GitLab Runner — это агент, который собственно и занимается выполнением инструкций из специального файла `.gitlab-ci.yml`.

- Выполняет задачи прописанные в пайплайне;
- Можно установить на любую платформу;
- Умеет запускать задачи совершенно различными способами: локально, в докер-контейнерах, в различных облаках или через ssh-коннект к какому либо серверу;
- Способен выполнять задачи параллельно с другими раннерами, а также обмениваться данными по средствам артефактов (artifacts).

Container Registry

GitLab Container Registry — это безопасный приватный реестр для образов (images) Docker. GitLab Container Registry *полностью* интегрирован в GitLab.


Container Registry

CLI Commands 

 5 Image repositories  Cleanup is not scheduled.

Filter results




Updated 



 spd-metadata-microservice-eventindex/project-mc_microservice 

1 tag



 spd-metadata-microservice-eventindex/project-run_tag_microservice 

1 tag



 spd-metadata-microservice-eventindex/project-software_tag_microservice 

1 tag

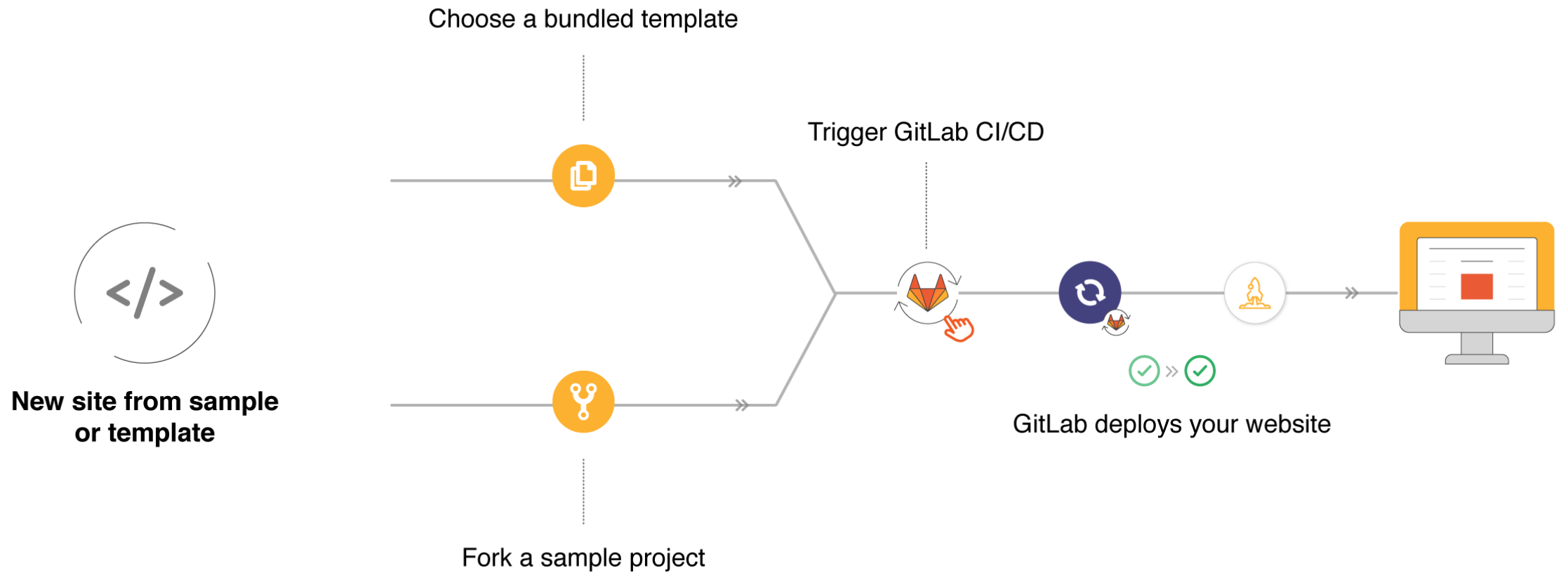


 spd-metadata-microservice-eventindex/project-dataset_microservice 

1 tag



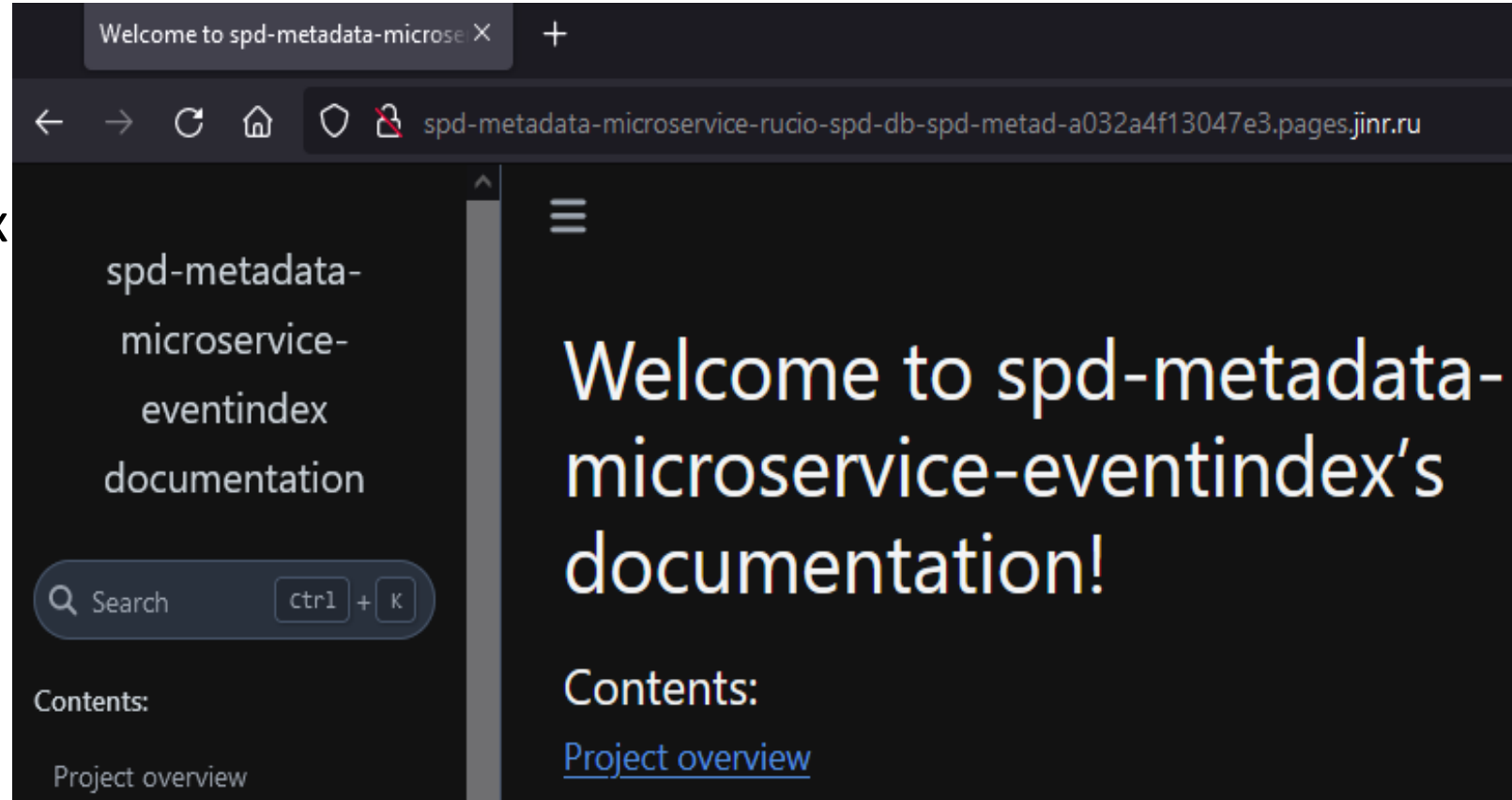
Gitlab Pages

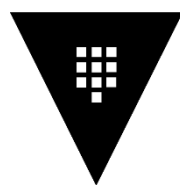


Gitlab Pages

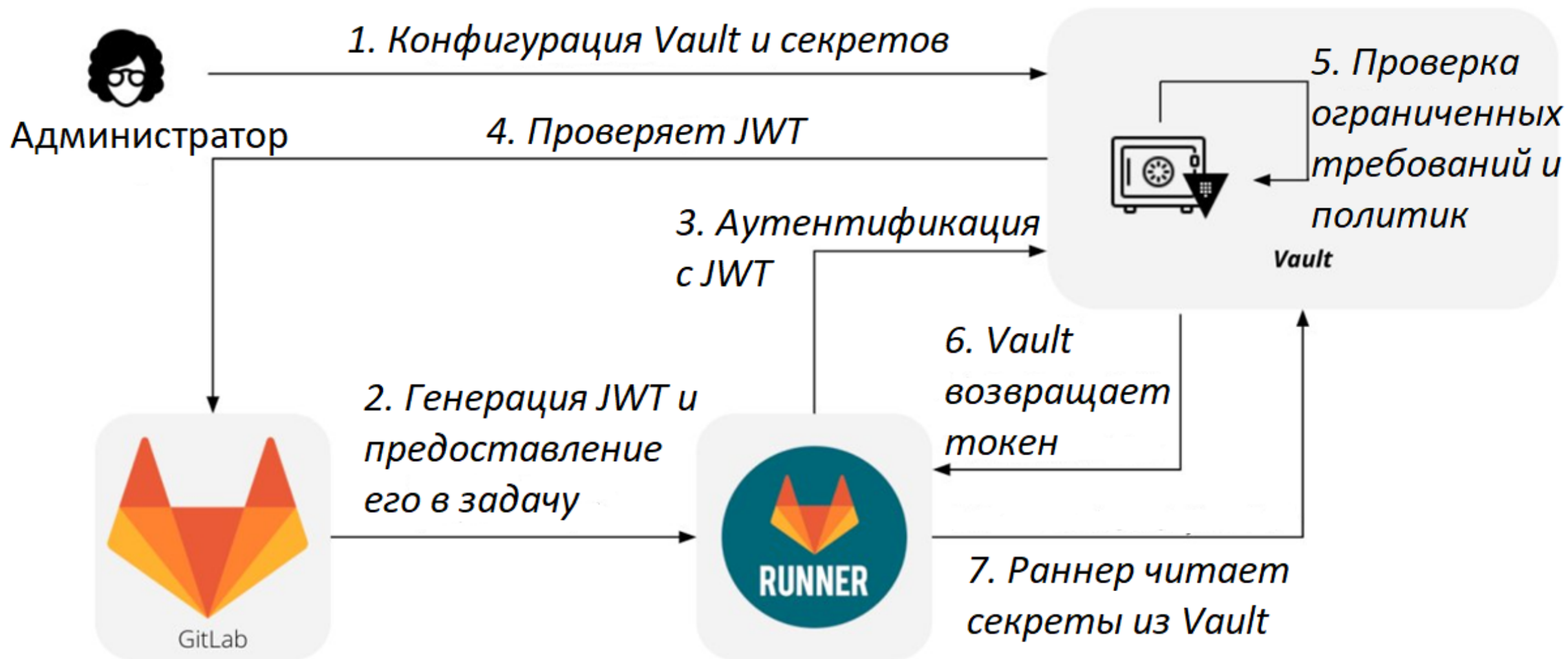
С помощью Gitlab Pages можно публиковать статические веб-сайты *непосредственно* из хранилища Gitlab

- Можно использовать любой генератор статических сайтов (SSG) или обычный HTML;
- Создание веб-сайта для своих проектов, групп или учетной записи пользователя;
- Размещение на собственном экземпляре Gitlab или Gitlab.com бесплатно;
- Подключение своих пользовательских доменов и сертификатов TLS.





Vault



Текущие результаты

- Изучены проектируемые информационные системы эксперимента SPD, модели данных, а также реализации сходных систем в других экспериментах;
- Внедрен GitFlow с использованием стандартизированных принципов коммита («Соглашение о коммитах»);
- Настроен реестр Container Registry, в котором уже хранятся Docker образы микросервисов БД ИС SPD;
- Подключены и настроены собственные Gitlab Runners;
- Реализован работающий пайплайн на базе Gitlab CI/CD, включающий автоматическую:
 - сборку Docker-образов микросервисов;
 - выгрузку образов Docker в реестр Container Registry проекта;
 - развертывание собранных образов из реестра на деплоймент-сервер;
 - публикацию документации на проектном домене экземпляра git.jinr;

Текущие результаты

- Поднят и актуализируется сайт с документацией на основе Sphinx на тестовом репозитории (из-за ограничений прав доступа, об этом в дальнейших планах);
- Сконфигурирован и поднят Vault-сервер для хранения зашифрованных переменных окружения и данных аутентификации.

Дальнейшие планы

- Реализовать автоматическое юнит и нагрузочное тестирование в рамках выполнения пайплайна на отдельном тестовом сервере (виртуальной машине);
- Связать имеющихся раннеров с хранилищем Vault для передачи чувствительной информации и переменных окружения в зашифрованном виде;
- Настройка прав доступа для сайта Gitlab Pages;
- Оптимизация пайплайна;
- Оркестрация;
- Переход с Docker на Apptainer;
- Внедрение подхода CI/CD в другие группы и подразделения работающие над ПО для SPD.

Спасибо за внимание!



Нагрузочное тестирование



LOCUST

