ЦИФРОВАЯ
ИНДУСТРИАЛЬНАЯ
ПЛАТФОРМА

Руководство для системы развертывания (zifctl) (2.18.0)

Zyfra Industrial Internet of Things Platform (ZIIoT)

Изменения в документе

Версия	Дата	Автор	Описание
1.0	17.06.2024	Пеплин Ф.Н.	Создание документа

Содержание

1. Требования к рабочей станции, с которой будет производиться развертывание, и ее окружению

Таблица 1.1. Требования к оборудованию/рабочей станции

	Минимальные	Рекомендуемые
ЦПУ	2	4
ОЗУ	4 Гб	8Гб

Требования к окружению:

- ОС семейства **Linux** или **wsl2** (для ОС **Windows**).
- **Docker**.
- **Python** 3.10.

Для развертывания рекомендуется использовать скрипт-обертку (**wrapper script**).

Запуск системы автоматического развертывания платформы (**ZIIoT Deployment system** или **zifctl**) можно осуществить двумя способами:

- Выполнение развертывания с помощью **wrapper**, запуск **zifctl** интерактивно (рекомендуемый способ запуска).
- Запуск **zifctl** из контейнера **docker**.

2. Версионирование компонентов Платформы для релиза 2.18.0

2.1. Совместимость системы развертывания Платформы с Kubernetes

С помощью инструментов авторазвертывания **Платформа** может быть установлена только на системы оркестрации контейнеров, а именно на **Kubernetes** или производные от него дистрибутивы **ReD Hat Openshift Container Platform** (**OpenShift**/**OKD**).

Таблица 2.1 Совместимость системы развертывания ZIIoT 2.18.0 с Kubernetes

Дистрибутив	Назначение	Версия
Kubernetes	Системы оркестрации контейнеров, совместимые со стандартами CNCF.	1.21 и выше
OCP (OpenShift/OKD)		4.8 и выше
Deckhouse		1.52

Системы оркестрации контейнеров должны содержать некоторые компоненты, без которых **Платформа** не может быть установлена. Перечисленные компоненты устанавливаются, как правило, в рамках всего кластера и поэтому не могут быть включены в состав **Платформы**.

Таблица 2.2. Версии инфраструктурных компонентов в ZIIoT 2.18.0

Компонент	Обязательность	Версия
StorageClass	+	Сущность для хранения параметров подключения к системе хранения данных.
Container Network Interface (CNI)	+	Контроллер внутренней сети кластера. Входит в состав любого дистрибутива Kubernetes. Должен поддерживать работу ~150 IP сервисов, входящих в состав Платформы. Конкретная реализация (OpenShift SDN, Flannel, Canico и т.д.) не регламентируется.
external LoadBalancer	+	Общий сетевой LoadBalancer для организации сетевого взаимодействия конечных пользователей сервисов, развернутых с помощью оркестратора Kubernetes. Входит в состав любого дистрибутива Kubernetes и должен быть корректно сконфигурирован и доступен. Использование иных вариантов балансировки входящего трафика (например, установка внешнего Nginx на ноды кластера или использование NodePort) несовместимо с системой развертывания Платформы.
Ingress Controller	+	Контроллер входящего трафика для Kubernetes LoadBalancer. Для Kubernetes - только базовый ingress-nginx, для OCP это стандартный Route Controller, входящий в состав дистрибутива OCP. Для корректного заполнения IP адресов в сообщениях аудита, необходимо включить use-forwarded-headers.
cert-manager	-	Необязательный компонент Kubernetes, используется для генерации TLS-сертификатов: cert-manager.io. В типовом случае используется для генерации TLS-сертификатов для Ingress. Платформа может быть установлена и без него, в этом случае необходимо сгенерировать и указать в параметрах конфигурации (env-values.yaml) TLS-сертификаты для объектов Ingress вручную.

cert-manager.io/v1 Issuer или Cluster Issuer	-	Certificate Issuer для cert-manager - управляет TLS-сертификатами, в том числе Ingress Controller. Как и cert-manager, не является обязательным компонентом.

2.2. Совместимость Платформы с инфраструктурными компонентами

Данные компоненты могут либо устанавливаться вместе с Платформой в **Kubernetes/OCP**, либо использоваться в качестве внешних зависимостей, установленных в другом месте.

В связи с этим, в таблице совместимости перечислены только требования к версиям этих компонентов, без указания конкретной реализации.

Примечание. Совместимость с иными реализациями **S3** (**Ceph**, ...) не протестирована. Точно известно, что с отличными от **MinIO** хранилищами несовместима автоматика развертывания (инициализация **bucket**, **access/secret key** и **policy**).

Таблица 2.3. Версии инфраструктурных компонентов в ZIIoT 2.18.0

Компонент	Назначение	Версия
PostgreSQL	рСУБД (Реляционная система управления базами данных). **Примечание**. Для работы сервисов Платформы необходима установка следующих extensions для PostgreSQL, не входящих в базовый пакет: ▪ tablefunc; ▪ uuid-ossp; ▪ pg_trgm; ▪ btree_gist. Для большинства операционных систем для установки этих extensions достаточно установить пакет postgresql-contrib, в зависимостях у этого пакета - библиотеки от Perl. postgresql-contrib входит в состав базового образа PostgreSQL при его установке в Kubernetes с помощью Stolon. в случае использования внешнего PostgreSQL в закрытых контурах необходимо убедиться, что там всё это установлено	12 и выше
Apache Cassandra	БД временных рядов	3.11
Redis	key-value БД (кэш)	6
Apache Kafka	Брокер сообщений	3.5
RabbitMQ	Брокер сообщений (используется в сервисе UDL).	3.11
Apache NiFi	Сервис управления потоками данных	1.19.1
KeyCloak	Сервис авторизации	17, 21
S3	API объектного хранилища	MiniO 2024.2.17

2.3. Версии инфраструктурных и вспомогательных сервисов в составе инструмента по авторазвертыванию (zifctl)

Релиз платформы версии 2.18.0 поставляется с инструментом по автоматическому развертыванию Платформы.

Версия инструмента (**zifctl**) аналогична версии платформы — 2.18.0. С помощью инструмента автоматического развертывания развертываются сервисы платформы, вошедшие в релиз 2.18.0, а также инфраструктурные и вспомогательные сервисы из таблицы ниже:

Таблица 2.4. Версии вспомогательных сервисов (zifctl) в ZIIoT 2.18.0

Компонент	Реализация	Назначение	Версия	Образ	Лицензия
PostgreSQL	Zyfra	PostgreSQL + утилиты для создания отказоустойчивого кластера	12.18	zif-stolon-pg12:1.2.0-v5-240515	Apache License 2.0
	Zyfra	PostgreSQL + утилиты для создания отказоустойчивого кластера	14.11	zif-stolon-pg14:1.2.0-v5-240515	Apache License 2.0
pgAdmin4	pgAdmin	Графический клиент для PostgreSQL	8.6	zif-pgadmin4:8.6.0-v1-240515	The PostgreSQL Licence
Apache Cassandra	Bitnami + Zyfra	БД временных рядов	3.11.13	zif-cassandra:3.11.13-v21-240515	Apache License 2.0
Apache Cassandra Exporter	Bitnami + Zyfra	мониторинг Cassandra	2.3.8	zif-cassandra-exporter:2.3.8-v20-240515	Apache License 2.0
Redis	Bitnami + Zyfra	key-value БД (кэш)	6.2.14	zif-redis:6.2.14-v10-240515	Apache License 2.0
Redis Exporter	Bitnami + Zyfra	Мониторинг Redis	1.59.0	zif-redis-exporter:1.59.0-v2-240515	Apache License 2.0
Apache Kafka	Confluent + Zyfra	Брокер сообщений	7.5.4	zif-kafka:7.5.4-v1-240515	Apache License 2.0
Kafka Exporter	Bitnami + Zyfra	Мониторинг Apache Kafka	1.7.0	zif-kafka-exporter:1.7.0-v17-240515	Apache License 2.0
Apache Zookeeper	Confluent + Zyfra	Распределенное хранение конфигураций Apache Kafka и Apache NiFi	7.5.4	zif-zookeeper:7.5.4-v1-240515	Apache License 2.0
Kafka Rest	Confluent + Zyfra	REST Api для Apache Kafka	7.5.4	zfi-kafka-rest:7.5.4-v1-240515	Apache License 2.0
Kafka Schema Registry	Confluent + Zyfra	Схемы сообщений для топиков Apache Kafka	7.5.4	zif-schema-registry:7.5.4-v1-240515	Apache License 2.0
Kafka Connect	Confluent + Zyfra	Утилиты для интеграции различных источников данных с Apache Kafka. В Zyfra добавлена утилита jq и несколько коннекторов (например, Debezium)	7.5.4	zif-kafka-connect:7.5.4-v1-240515	Apache License 2.0

Debezium Connector	Red Hat	Сбор, сериализация и отправка в Apache Kafka логов транзакций PostgreSQL	1.7.1	см. Kafka Connect	Apache License 2.0
JDBC Connector	Confluent	Сбор, сериализация и отправка данных в Apache Kafka (как Source) и РСУБД (как Sink)	10.2.5	см. Kafka Connect	Apache License 2.0
JMX Exporter	Bitnami + Zyfra	Экспорт метрик JVM для сервисов экосистемы Kafka. В Zyfra на релиз по хэш-коммиту выставлен тег Stable	0.20.0	zif-jmx-exporter:0.20.0-v10-240515	Apache License 2.0
Kafka UI	Provectus Labs + Zyfra	UI для Apache Kafka, Kafka Connect, Kafka Schema Registry	0.7.2	zif-kafka-ui:0.7.2-v2-240515	Apache License 2.0
RabbitMQ	Bitnami + Zyfra	Брокер сообщений	3.12.14	zif-rabbitmq:3.12.14-v1-240515	Apache License 2.0
Apache NiFi	Apache Foundation + Zyfra	Сервис управления потоками данных. В Zyfra добавлен ряд процессоров для интеграции со сторонними источниками данных	1.19.1	zif-nifi:1.19.1-v30-240515	Apache License 2.0
KeyCloak	RedHat + Zyfra	Сервис авторизации. stolon В Zyfra добавлен корпоративный дизайн фронтенда и несколько плагинов	17.0.1	zif-keycloak:17.0.1-v28-240515	Apache License 2.0
			21.1.2	zif-keycloak:21.1.2-v4-240515	Apache License 2.0
KeyCloak RabbitMQ Event Listener	community	Экспорт KeyCloak Events в топики RabbitMQ	3.0.2	см. KeyCloak	Apache License 2.0
KeyCloak Metrics SPI	community	Экспорт метрик KeyCloak	2.5.3	см. KeyCloak	Apache License 2.0
OAuth2 Proxy	community	Прокси для авторизации в KeyCloak для Kafka UI, Apache NiFi	7.5.1	zif-oauth2-proxy:7.5.1-v7-240219	MIT
Alpine	Docker Inc + Zyfra	Набор утилит	3.19.1	zif-alpine-util:3.19.1-v7-240515	GPLv2
MinIO	Bitnami + Zyfra	S3-совместимое объектное хранилище	2024.5.10	zif-minio:2024.5.10-v1-240515	Apache License 2.0
Opensearch	community + Zyfra	Масштабируемая утилита полнотекстового поиска и аналитики	1.3.16	zif-opensearch:1.3.16-v1-240515	Apache License 2.0
Fluentd	Bitnami + Zyfra	Система сбора, парсинга, перенаправления и агрегации логов	1.14.6	zif-fluentd-opensearch:1.14.6-v16-240219	Apache License 2.0
Opensearch Dashboards	community + Zyfra	UI для Opensearch	1.3.16	opensearch-dashboards:1.3.16-v1-240515	Apache License 2.0

3. Специальные требования к процессу обновления

3.1. Обновление RabbitMQ

В релизе 2.18.0 **RabbitMQ** обновлен до версии 3.12.14.
При обновлении на эту версию следующие **feature** флаги должны быть включены в обязательном порядке (https://www.rabbitmq.com/docs/feature-flags#core-feature-flags), в обратном случае **RabbitMQ** не запустится:

- **classic_mirrored_queue_version;**
- **direct_exchange_routing_v2;**
- **feature_flags_v2;**
- **listener_records_in_ets;**
- **stream_single_active_consumer;**
- **tracking_records_in_ets;**
- **classic_queue_type_delivery_support;**
- **stream_queue.**

1. Перед обновлением на версию 2.18.0 необходимо проверить, включены ли вышеперечисленные флаги. Список флагов можно получить двумя способами:

- В консоли управления **RabbitMQ** -> вкладка **Admin** в верхнем горизонтальном меню -> пункт **Feature Flags** в правом меню.
  - Выполнив **команду rabbitmqctl -q --formatter pretty_table list_feature_flags** на любом узле **rabbitmq**.
- При наличии флагов из списка в выключенном состоянии, их необходимо включить:
  - В консоли управления **RabbitMQ** - нажав **Enabled** на выключенном **feature** флаге.
  - Выполнив команду **rabbitmqctl enable_feature_flag all** на любом узле **rabbitmq**.

Если флаги уже включены, ничего предпринимать не нужно.

3.2. Взаимодействие с `Apache Cassandra` при соединении по `TLS`

При нестандартном значении поля **cassandra.contactPoints в env-values.yaml** рекомендуется явно задать параметр **cassandra.tls.hostname**:

- при использовании внешней **Apache Cassandra** должен быть равен **CN** (**Common Name**) сертификатов, используемых **Apache Cassandra**.
- при использовании платформенной **Apache Cassandra** должен быть равен **zif-cassandra-headless.<namespace>.**

Примечание. При генерации сертификатов (вне зависимости от количества узлов) **CN** должен быть одинаковый у всех сертификатов, например: **cassandra-cluster**. В **SAN** сертификата должны быть указаны **hostname** и **IP** узла, для которого генерируется сертификат.

3.3. Развертывание сервиса zif-file-storage в виде sidecar MinIO

В релизе 2.18.0 добавлена возможность развертывания сервиса **zif-file-storage** в виде **sidecar'a MinIO** для **single**-тенант инсталляций с целью оптимизации сетевого трафика в кластере. Для развертывания в этом режиме в файле конфигурации развертывания **env-values.yaml** необходимо в секции **s3storage** добавить параметр **fileStorageServiceSidecar** со значением **True**:

```
## Конфигурация MinIO/S3

s3storage:
  default:
    ...
    fileStorageServiceSidecar:                                    True
    ...
```

Настройка применима только для `default` S3-профиля и при типе развертывания `instance`.

3.3.1. Исправления во взаимодействии с Apache Cassandra при соединении по TLS

В релизе 2.18.0 исправлен алгоритм присваивания значения переменной **CASSANDRA_SSL_HOSTNAME** для сервисов, взаимодействующих с **Apache Cassandra**, при использовании соединения по **TLS**.

При стандартном значении поля **cassandra.contactPoints** в **env-values.yaml** - **zif-cassandra-headless.<namespace>** (в случае развертывания по умолчанию), оно будет присвоено переменной окружения **CASSANDRA_SSL_HOSTNAME**.

Во всех остальных случаях **CASSANDRA_SSL_HOSTNAME** будет присвоено значение параметра **cassandra.tls.hostname**, которое при отсутствии в **env-values.yaml** приравнивается к пустой строке по умолчанию, вследствие чего сервисы взаимодействующие с **Apache Cassandra**, будут определять **hostname** самостоятельно.

При нестандартном значении поля **cassandra.contactPoints** в **env-values.yaml** рекомендуется явно задать параметр **cassandra.tls.hostname**:

- при использовании внешней **Apache Cassandra** должен быть равен `CN` (**Common Name**) сертификатов, используемых **Apache Cassandra**.
- при использовании платформенной **Apache Cassandra** должен быть равен **zif-cassandra-headless.<namespace>**.

Примечание. При генерации сертификатов (вне зависимости от количества узлов) **CN** должен быть одинаковый у всех сертификатов, например: **cassandra-cluster**.
В **SAN** сертификата должны быть указаны **hostname** и **IP** узла, для которого генерируется сертификат.

3.3.2. Postgresql 14: улучшение механизма управлением метрик Prometheus в авто-развертывании

Управление параметрами Prometheus

Для управления параметрами метрик в файл конфигурации **zifctl env-values.yaml** в дополнение к параметрам **postgres** необходимо добавить подсекцию **metrics**.

Если секция отсутствует, либо выставлены лишь некоторые параметры, для всех оставшихся параметров будут выставлены значения по умолчанию, приведенные ниже:

```
## Конфигурация Postgres
postgres:
  installed: True
  host: 'zif-postgres.namespace'
  port: 5432
  storageSizeMB: 5120
  version: stolon-14
  tls:
    enabled: True
    force: True
    mode: 'VerifyFull'
  metrics:
    enabled: True
    path: '/metrics'
```

Описание параметров:

Параметр	Значение по умолчанию	Возможные значения	Описание
enabled	`False`	`string`	Включение метрик
path	`/metrics`	`string`	Путь по которому должны отдаваться метрики.

Управление дополнительными переменными

Управление переменными возможно через изменение шаблона **values/infra/zif-postgres14.yaml.gotmpl**.

```
## Пример файла шаблона с измененной переменной
PG_EXPORTER_DISABLE_SETTINGS_METRICS
metrics:
  extraEnvVars:
    - name: PG_EXPORTER_DISABLE_SETTINGS_METRICS
      value: 'True'
```

Актуальный список переменных можно просмотреть на официальном сайте репозитория по ссылке https://github.com/prometheus-community/postgres_exporter#adding-new-metrics-via-a-config-file

Auto discovery prometheus

```
## Пример файла настройки auto discovery Prometheus
spec:
```

```
endpoints:
  - port: exporter
namespaceSelector:
  any: true
podTargetLabels:
  - release
  - app
selector:
  matchLabels:
    app: prometheus-postgres-exporter
```

app: prometheus-postgres-exporter является ключевым для определения пойнта с метриками.

4. Развертывание Платформы

Предварительно для развертывания Платформы необходимо:

1. Подать заявку в Цифровую Индустриальную Платформу.
2. Установить **Kubernettes**. Версионность продукта: от 1.18 до 1.27.
3. Получить ссылку на проект **gitlab, conflg-file** для **Kubernettes** с доступом в namespace. В этом namespace развернется вся платформа, это как бы отдельная область, которая будет содержать все микросервисы.

Развертывание возможно на мощностях Цифровой Индустриальной Платформы, с помощью сервиса Яндекс.облако. Информация распространяется в закрытом контуре.

4.1. Развертывание впервые автоматически пайплайном с помощью Gitlab и Kubernettes

Итак, после получения положительного ответа на заявку предоставляется доступ в репозиторий.

Для развертывания автоматически с помощью пайплайна командой инфраструктуры Платформы приготавливается проект, который содержит три служебных файла:

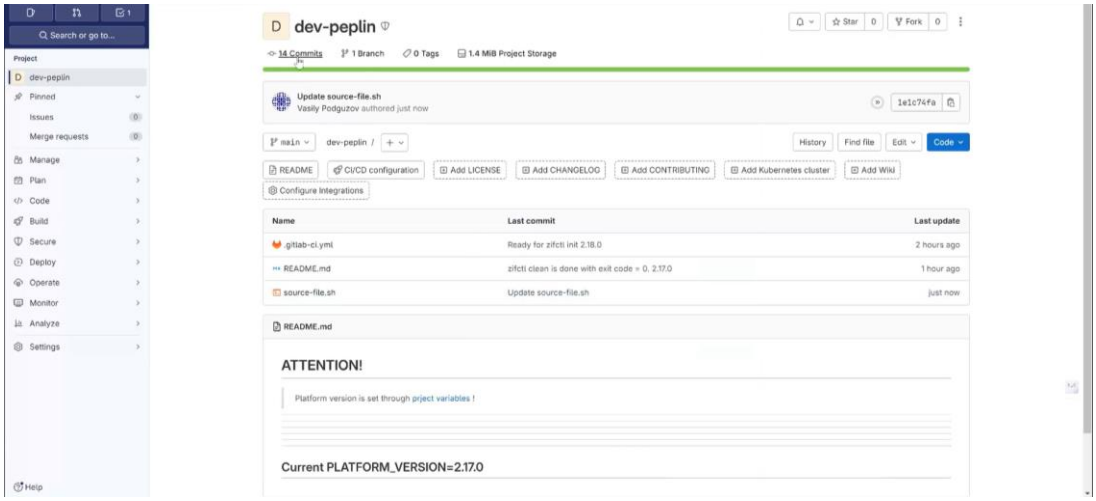

Рисунок 4.1 Изначальное состояние переданного проекта

В данном примере требуется перейти в раздел Settings, во всплывающем меню раздел **CI/CD**:

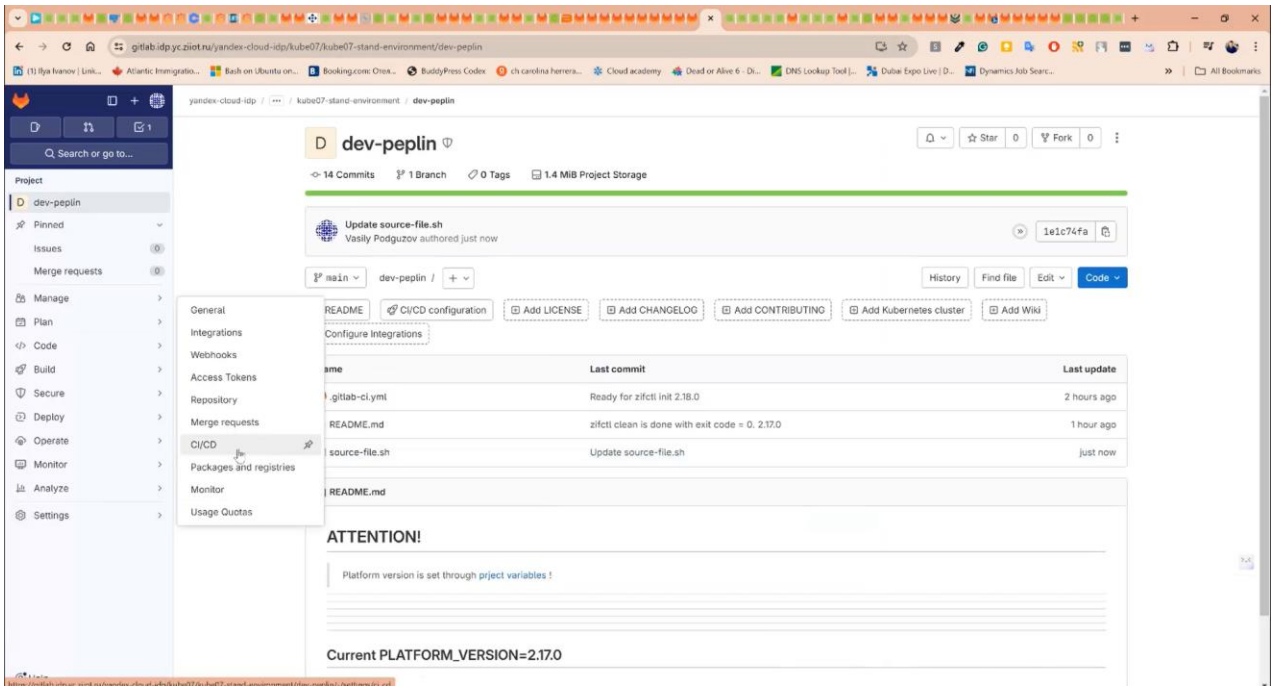

Рисунок 4.2 CI/CD

В данный момент там находится одна переменная **PLATFORM_VERSION**. Если открыть данную переменную, то в разделе **Value** можно увидеть значение – 2.17.0, которое можно изменить на 2.18.0 или любое другое доступное значение, необходимое для первого развертывания Платформы:

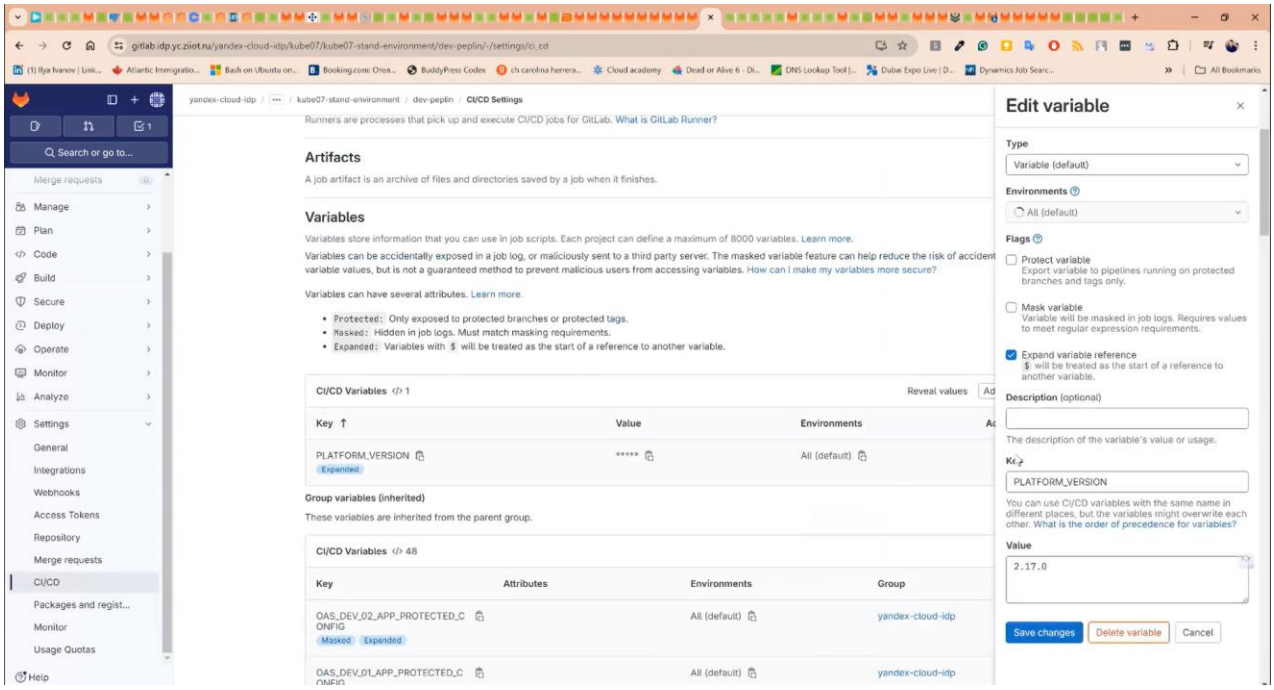

Рисунок 4.3 Переменные

Первым шагом будет первоначальная инициализация платформы. Она делается первый раз при создании стенда и проекта, в последующем делать её не требуется. На данном этапе создаются все секреты, служебные файлы и так далее.

Для первичной инициализации необходимо сделать следующие шаги:

4. Перейдите в раздел **Bulid**, подраздел Pipelines в **gitlab**.

5. Нажмите кнопку **Run Pipeline**.
6. В разделе запуска пайплайна запустите ветку **main**:

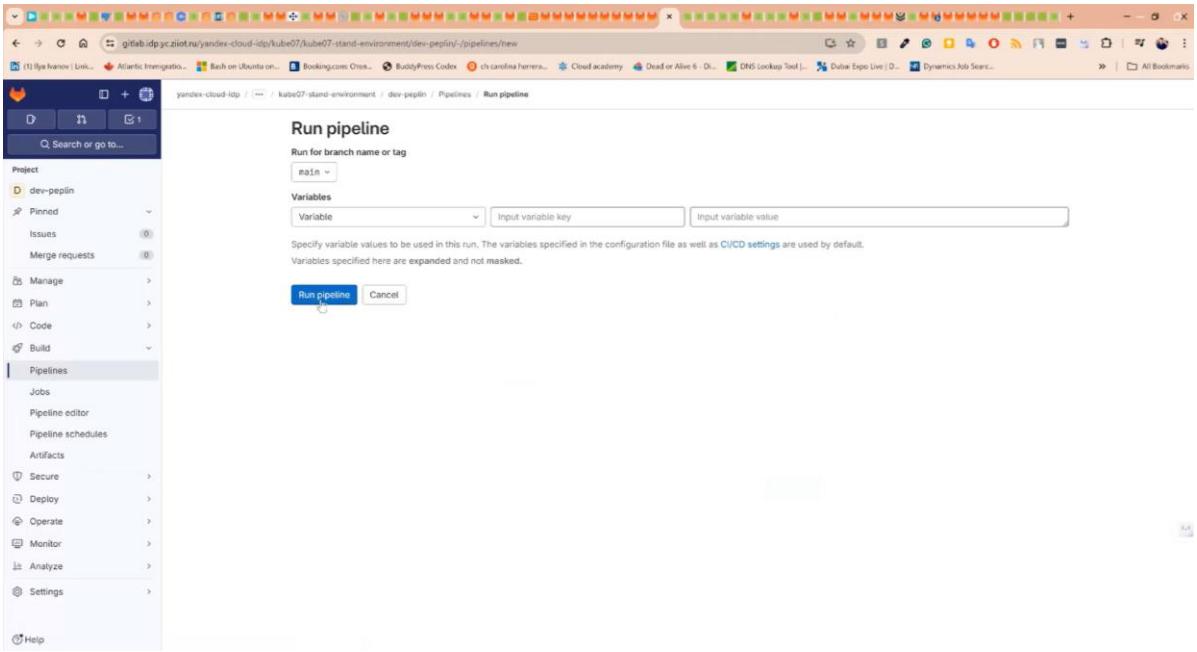

Рисунок 4.4 Запуск пайплайна

7. После запуска появится на выбор два процесса **init** и **do-action-job**. В блоке **init** нажмите **Play**. Запустится раннер, который проинициализирует проект - закладывается фундамент для будущих развертываний.

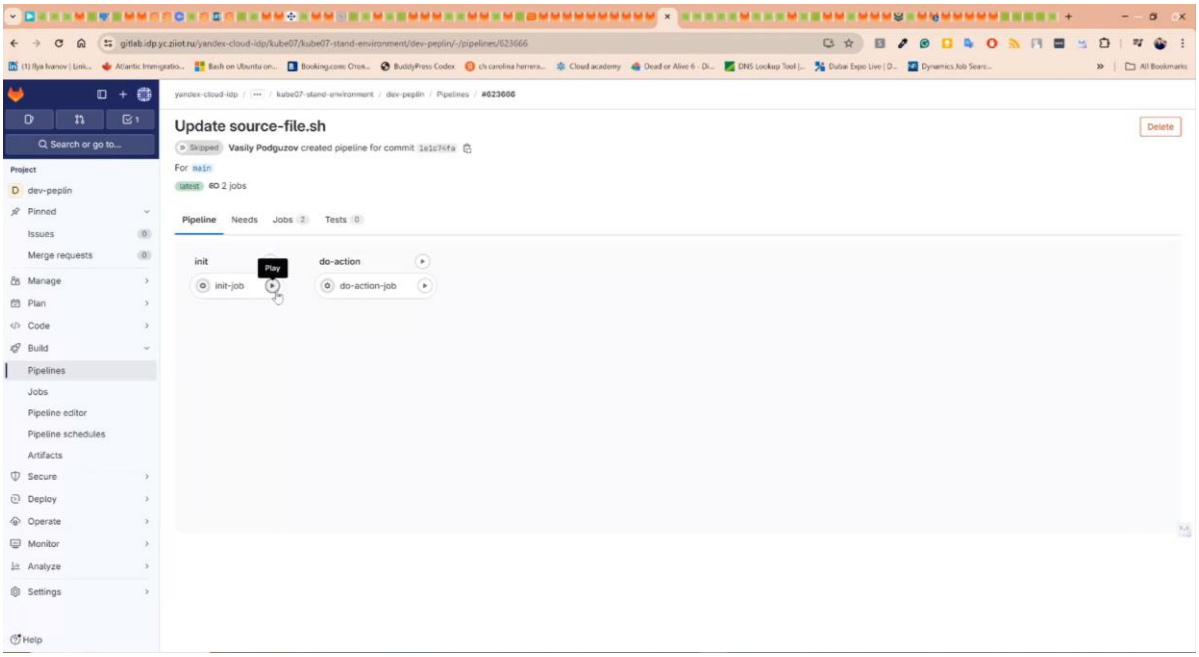

Рисунок 4.5 Запуск init

8. В первую очередь в процессе инициализации создается переменная **zif_age_key** в разделе **Variables**, который требуется для работы и синхронизации:

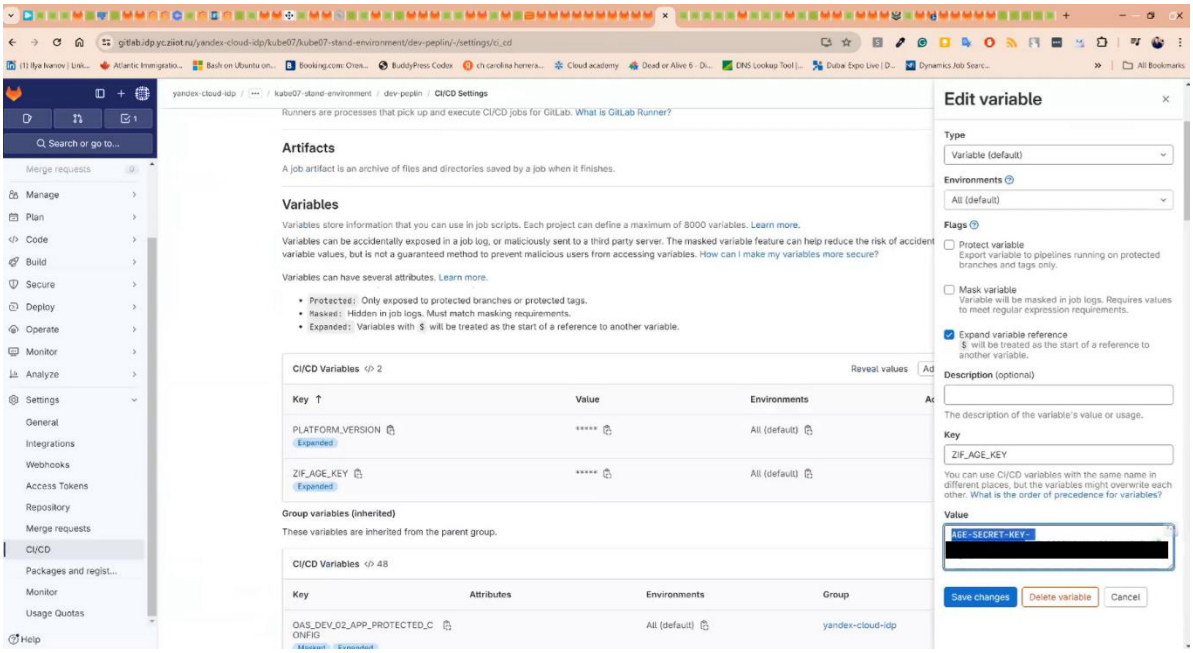

Рисунок 4.6 ZIF_AGE_KEY

9. После успешной инициализации проекта создается папка **env-values** в репозитории со следующим содержимым:

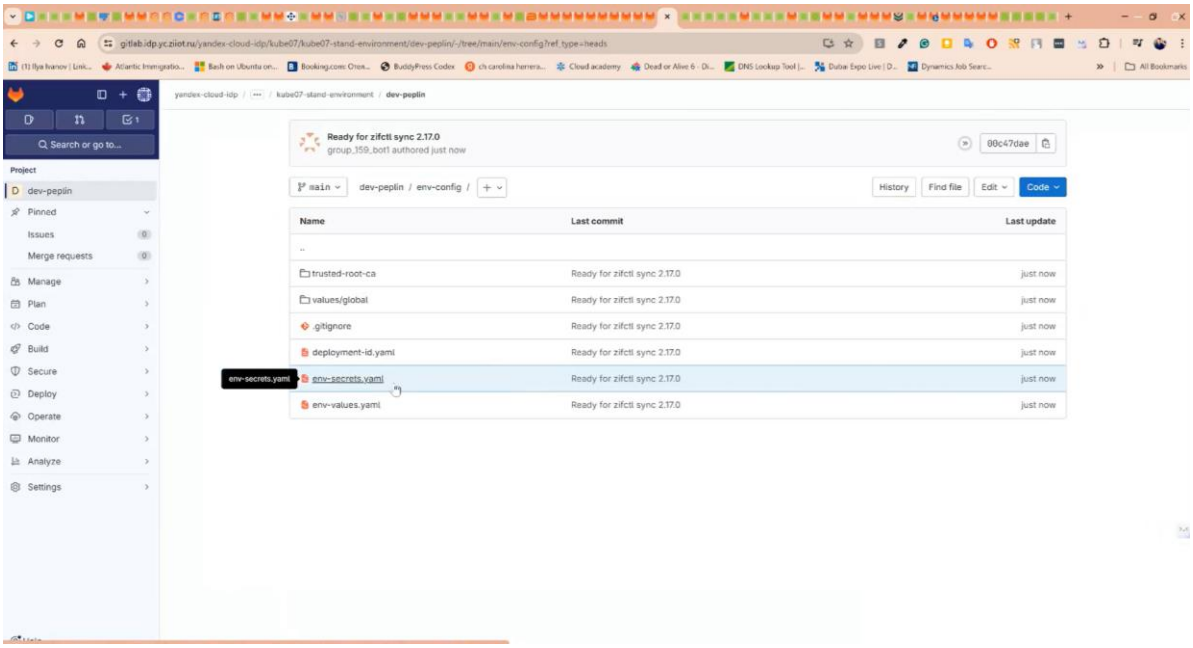

Рисунок 4.7 env-values.yaml

Основным будет файл **env.values.yaml**, в котором содержится полное описание с подробными комментариями – значения проекта, переменные. Описано какое будет имя Платформы для браузера, её размер, адрес docker-образов проекта микросервисов и прочая служебная информация: где будет **Postgres**, **Keycloak**, **Cassandra** и т.д.

10. Далее аналогичным образом запустите второй пайплайн **– do-action-job**. Этот процесс и будет являться непосредственным развертыванием Платформы:

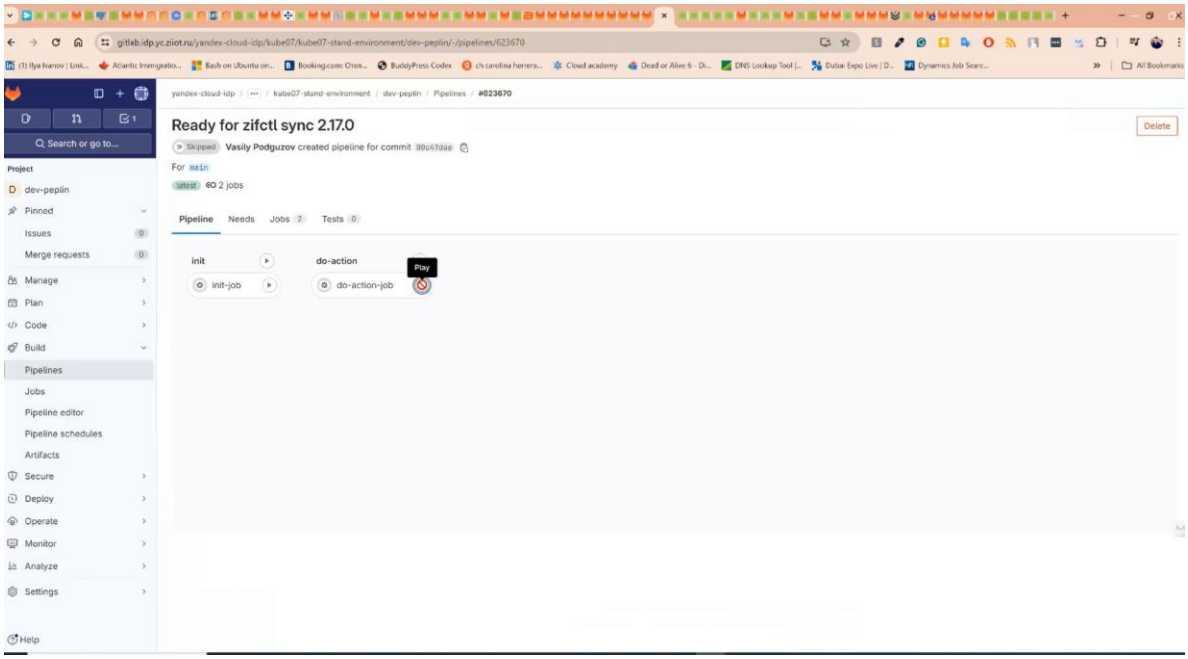

Рисунок 4.8 do-action-job

11. Помимо этого, параллельно в **Kubernettes** необходимо создать абсолютно пустой **namespace** желательно с таким же именем без **deployment**, без подов:

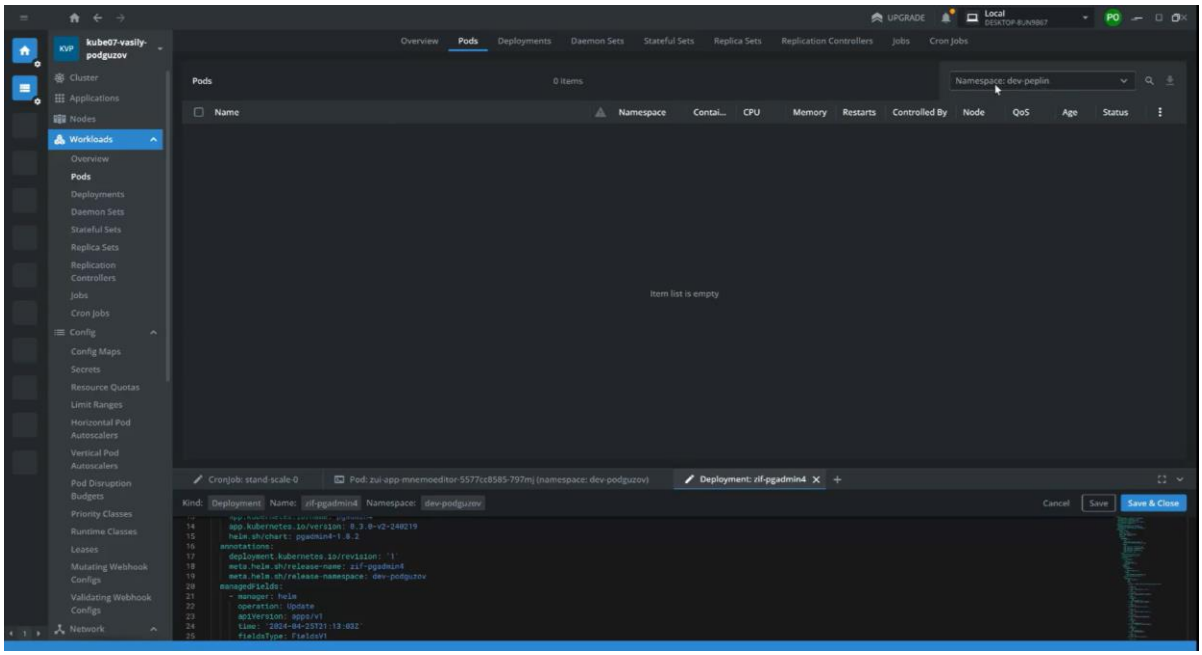

Рисунок 4.9 namespace

12. После того, как в секции **Jobs** закончится развертывание Платформы и в разделе **Conditions** будет статус **Completed**, а в разделе **Pods** развернутые поды со статусом **Running** – Платформу можно будет считать развернутой:

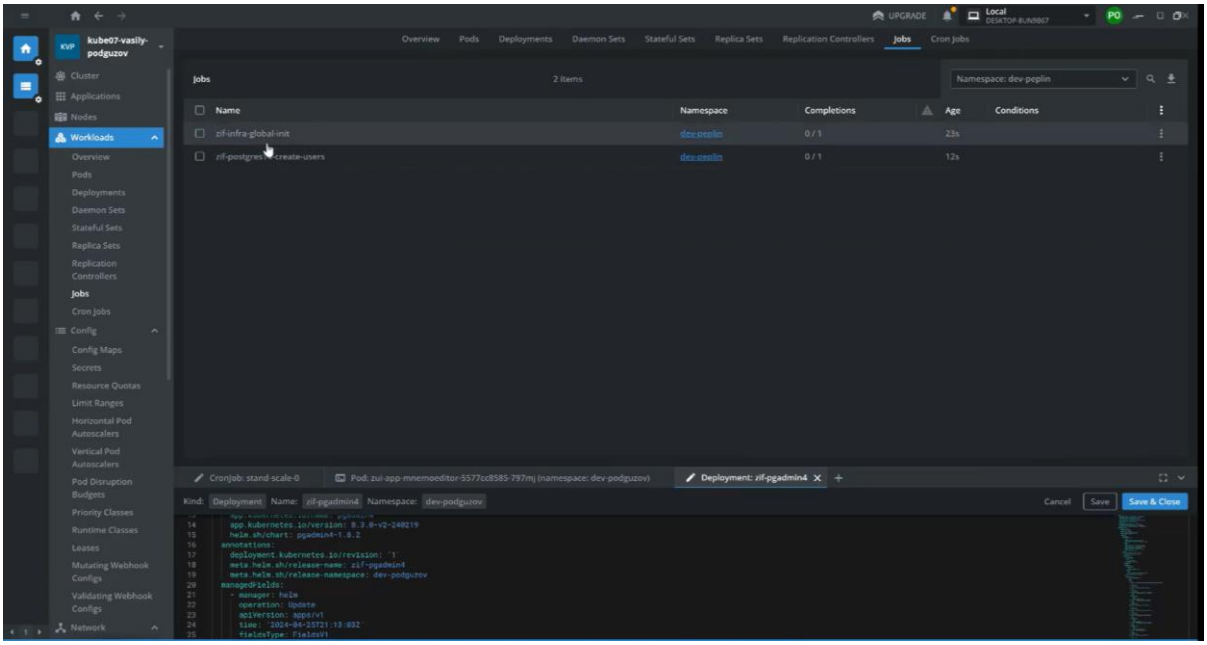

Рисунок 4.10 jobs

13. Проверка финального результата – переход по ссылке из файла **env-values.yaml**.

    Внимание! Для того, чтобы найти пароль – перейдите в раздел **Secrets** в **Kunernettes**.

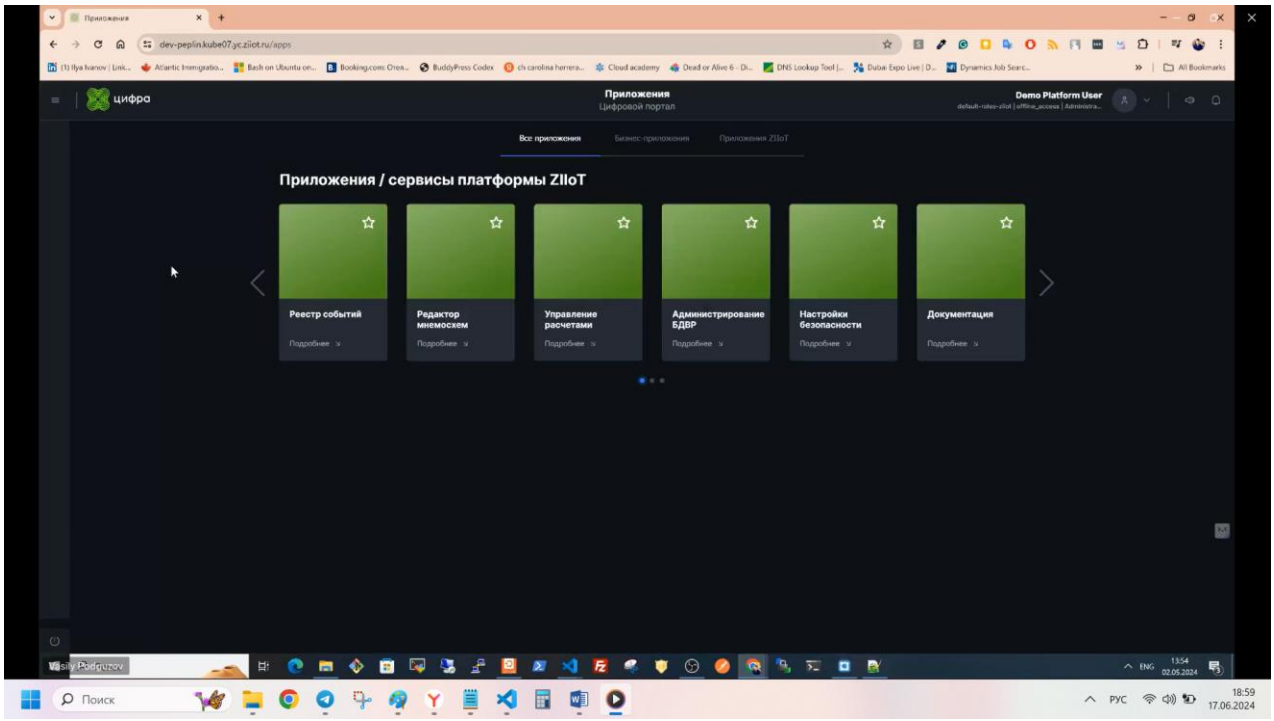

Рисунок 4.11 Развернутый вариант Платформы

4.1. Развертывание автоматически пайплайном с помощью Gitlab и Kubernettes при обновлении Платформы

При уже установленной Платформы для её обновления:

1. В разделе проекта перейдите в файл **source-file.sh**.

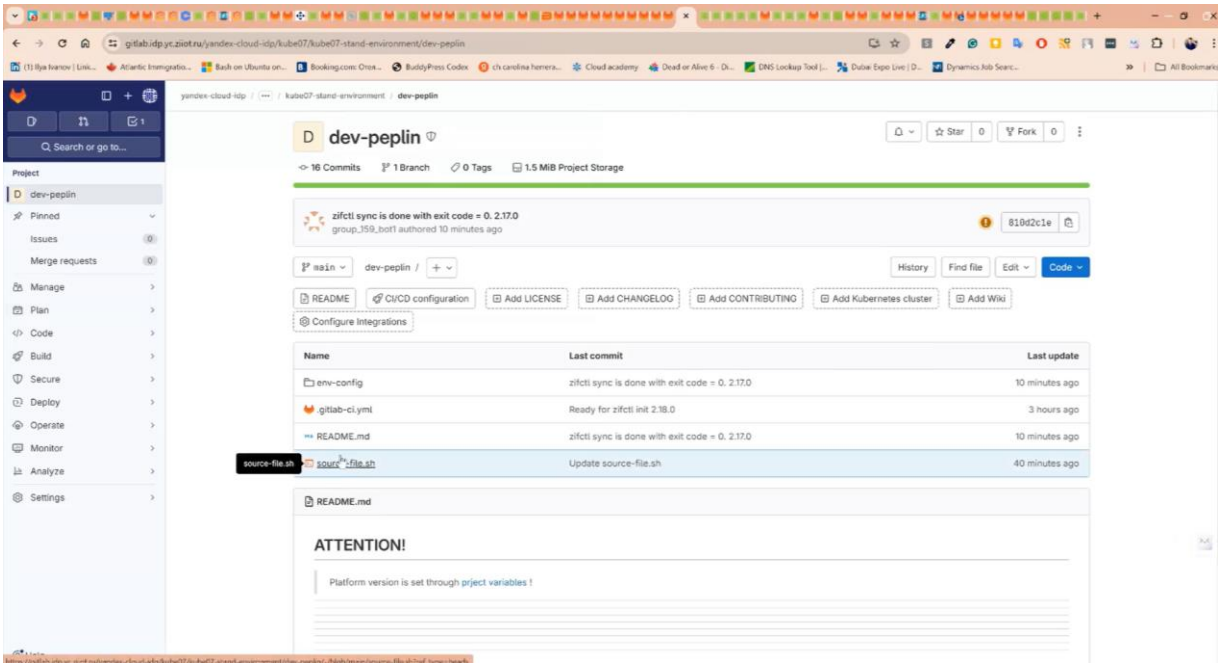

Рисунок 4.12 source-file.sh

По умолчанию в строке export **ACTION** будет значение **sync**, его нужно будет заменить:

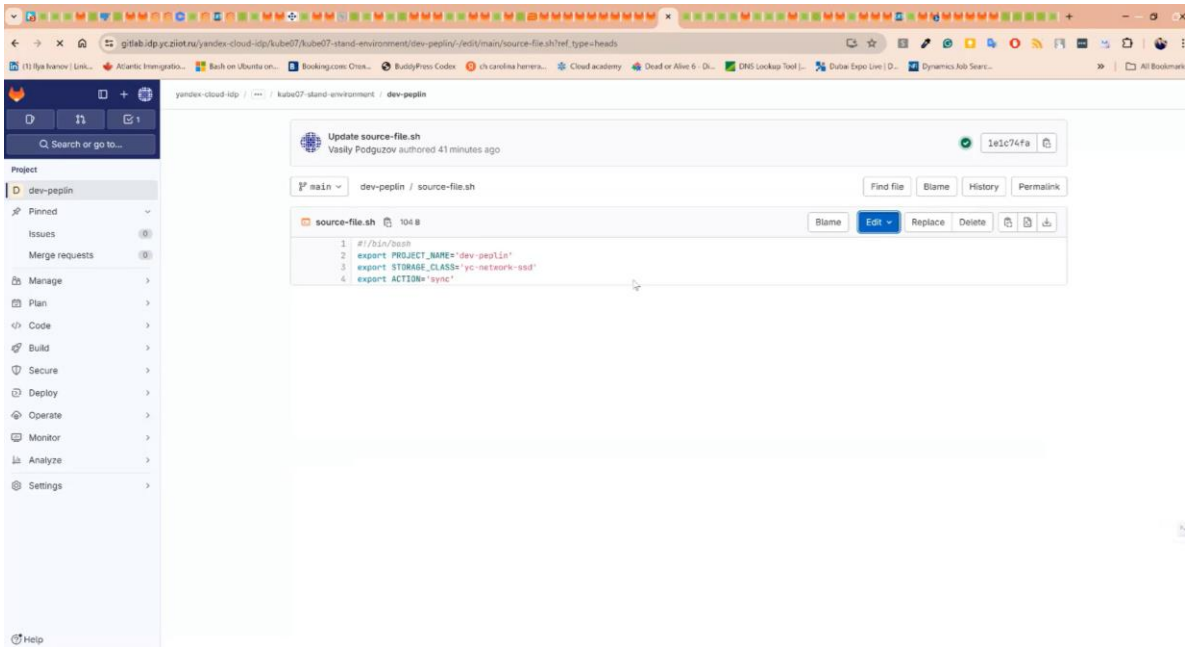

Рисунок 4.13 Содержимое файла source-file.sh

2. Нажмите кнопку **Edit**.
3. Нажмите опцию **Edit single file**.
4. Измените значение **sync** на **clean**:

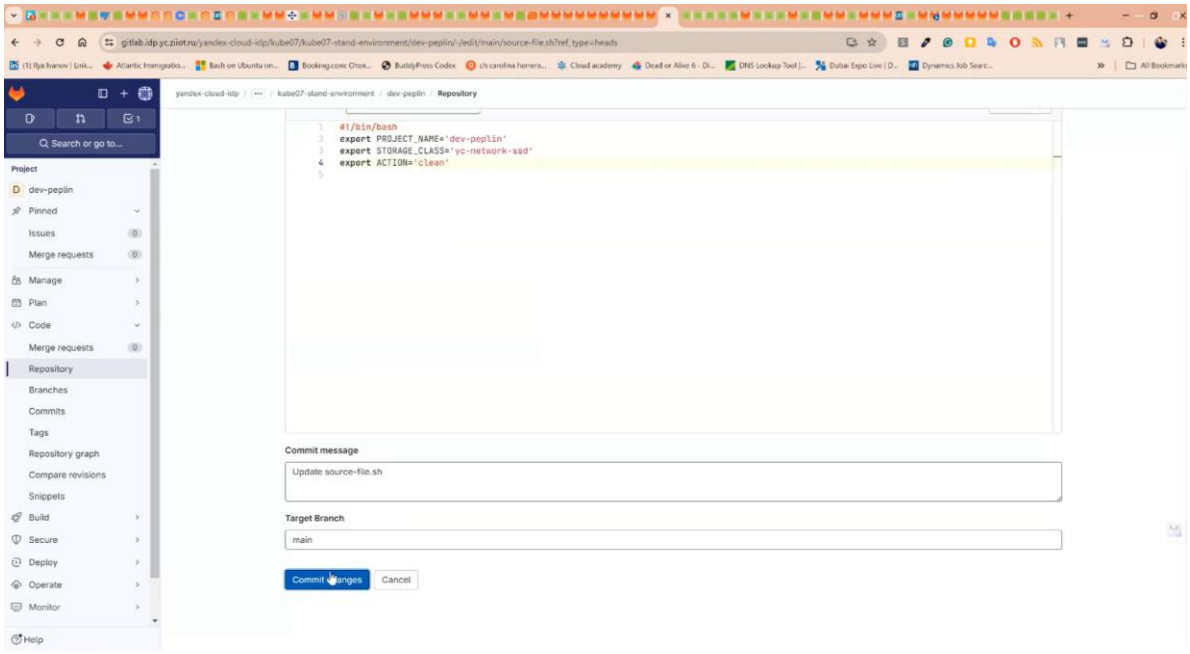

Рисунок 4.14 Редактирование sync на clean

5. Перейдите в раздел **Bulid**, подраздел Pipelines в **gitlab**.
6. Нажмите кнопку **Run Pipeline**.
7. В разделе запуска пайплайна запустите ветку main, процесс **do-action-job**:

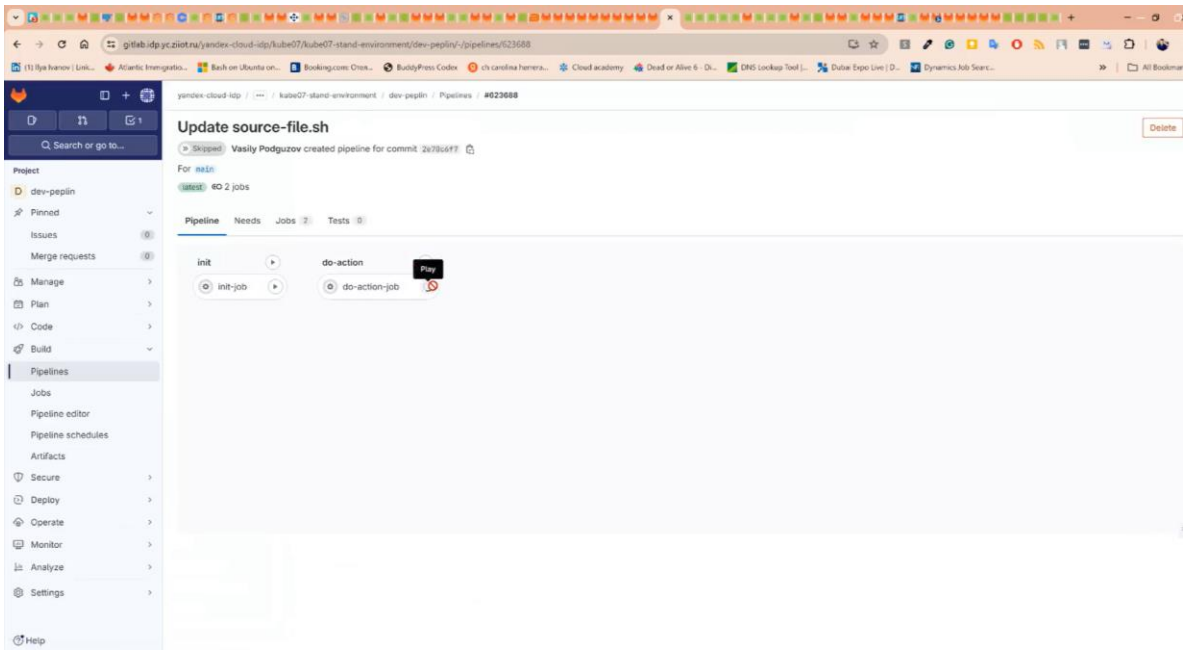

Рисунок 4.15 Запуск do-action-job

8. Перейдите в раздел **CI/CD** в раздел **Variables**:

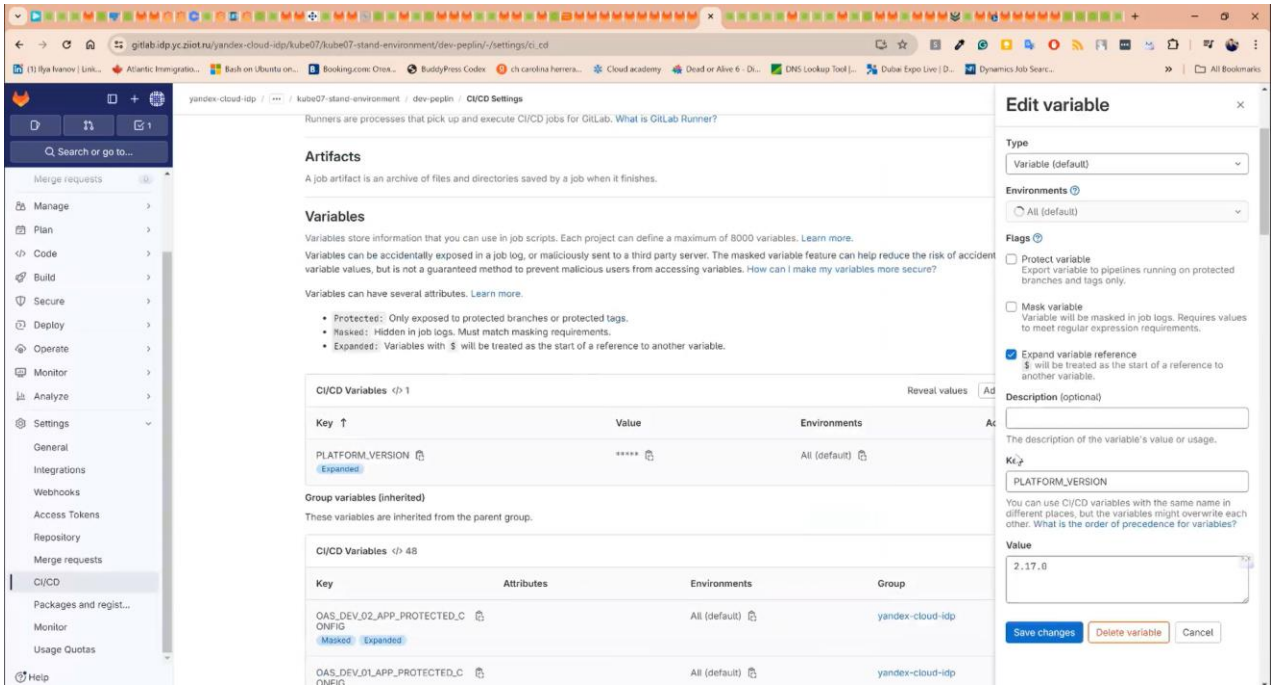

Рисунок 4.16 Variables

9. В поле **Value** измените значение с текущего на то, на которое хотите обновить Платформу.
10. Нажмите кнопку **Save changes**.
11. Разверните Платформу по аналогии с процессом из предыдущего раздела **Развертывание впервые автоматически пайплайном с помощью Gitlab и Kubernettes** – разверните новую версию.

4.2. Развертывание вручную: Развертывание Платформы в виде docker-образа

Для установки Платформы необходимо иметь доступ к **docker-registry**, где размещаются образы самой Платформы, ее инфраструктурных зависимостей и контейнер с системой развертывания.

Далее в командах будет использоваться репозиторий **Цифры**: https://registry.dp.zyfra.com/repository/. Если Платформа устанавливается в закрытом контуре, следует заменить адреса **registry**, и путь к контейнеру (для переноса контейнеров в registry в закрытом контуре, можно воспользоваться новой командой **zictl image scripts**).

4.2.1. Логин в docker registry

Для **Qauy** необходимо использовать **encrypted password** для логина через **CLI**, который можно получить в **UI** в своем профиле (**Account Settings**) и разделе **Docker CLI Password**:

```
echo $PASSWORD | docker login registry.dp.zyfra.com -u $USERNAME --password-stdin
```

4.2.2. Получение образа и запуск контейнера

Пробный запуск и получение помощи по доступным командам, получение информации о версии Платформы, которую развертывает контейнер, выглядит следующим образом:

```
# Получить образ
docker pull registry.dp.zyfra.com/infra/zifctl:2.9.0


# Запуск простых команд сразу с образа
docker run --rm  registry.dp.zyfra.com/infra/zifctl:2.9.0 --help
docker run --rm  registry.dp.zyfra.com/infra/zifctl:2.9.0 version
```

4.2.3. Получение скрипта-обертки (wrapper script)

Для интерактивной работы в терминале рекомендуется получить из контейнера скрипт-обертку для упрощенного запуска (скрипт позволяет указывать локальные пути и передает переменные окружения **ZIF_SERVER**, **ZIF_TOKEN**, **ZIF_AGE_KEY**, **ZIF_ENV_PATH** в запускаемый контейнер).

С точки зрения запуска скрипт-обертка примерно аналогична **zifctl** в прошлых релизах. Использование этого скрипта не обязательно (можно использовать **docker run**), но он добавляет возможность записи вывода контейнера в лог-файл (сам контейнер выводит все сообщения просто на консоль).

Скрипт-обертка обновляется вместе с системой развертывания. Необходимо обновлять его при переходе на новые версии системы и Платформы (каждый новый **docker**-образ может содержать новую версию **wrapper**).

Внутри скрипта зашита версия контейнера, с которого он получен и по умолчанию он будет запускать команды именно на нем. Посмотреть какой контейнер будет запускать скрипт можно выполнив команду **zifctl version.**

Для запуска этого скрипта на машине должен быть установлен **Python 3** (достаточно только штатных библиотек).

```
# Опционально скопировать скрипт в путь доступный через PATH
docker run --rm  registry.dp.zyfra.com/infra/zifctl:2.9.0 get-wrapper > zifctl
chmod +x zifctl
sudo cp zifctl /usr/local/bin
```

Проверка работы скрипта-обертки:

```
# Помощь по командам
./zifctl --help


# Информация о версиях используемых образов в скрипте, информация о версии Платформы
./zifctl version
```

Внимание! Скрипт-обертка обновляется вместе с системой развертывания. Необходимо обновлять его при переходе на новые версии системы и Платформы (каждый новый **docker**-образ может содержать новую версию **wrapper**).

Внимание! Внутри скрипта зашита версия контейнера, с которого он получен и по умолчанию он будет запускать команды именно на нем. Посмотреть какой контейнер будет запускать скрипт можно выполнив команду **zifctl version**

Внимание! Также внутри скрипта зашита ссылка на реестр/имя контейнера. Это можно поменять при помощи аргумента **--zifctl-registry** docker.idp.yc.ziiot.ru/infra (например, для переключения на реестр ЦИП).

4.2.4. Создание новой конфигурации окружения (если развертывается новый экземпляр Платформы)

Если **Платформа** устанавливается с нуля, то нужно создать конфигурацию окружения (конфигурации экземпляра) Платформы.

Конфигурация окружения представляет собой каталог с файлами, в которых сохранены все необходимые данные для разворачивания с нуля экземпляра **Платформы** или его обновления.

В общем случае предполагается, что конфигурация хранится в **git** и платформа ставится и обновляется по модели **GitOps**, в которой «источником правды» служит репозиторий, и по данным из него создаются все объекты в кластере (кроме **PVC** с данными, которые сохраняются при обновлениях).

4.2.5. Создание ключа шифрования и задание переменной ZIF_AGE_KEY

В конфигурации окружения так же хранятся и все пароли, **connection string** и прочие секреты, необходимые для работы Платформы. Т.к. секреты хранятся вместе с конфигурацией в **git**, они должны быть зашифрованы.

Система развертывания обеспечивает прозрачное шифрование секретов с помощью **Mozilla SOPS** (https://github.com/mozilla/sops) и утилиты (бэкенда) **age** (https://github.com/FiloSottile/age).

Шифрование требует секретного ключа (**private key**) для утилиты **age**, обычно уникального для данного экземпляра.

При развертывании нового экземпляра вам необходимо создать такой ключ и сохранить его (ключ не должен сохранятся в **git**-репозитории вместе с данными, которые он шифрует). Команда для создания нового ключа и сохранения его в файл:

```
# Создание нового ключа шифрования и запись его файл
./zifctl secrets new-age-key > key.txt
```

Полученный файл будет иметь вид ниже. Строка, начинающаяся с **AGE-SECRET-KEY-...** и есть нужный закрытый ключ. Полученный файл необходимо сохранить любым надежным способом:

```
AGE-SECRET-KEY-XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

Полученный ключ требуется указывать практически для всех команд утилиты развертывания, которые выполняют любую работу с экземпляром платформы. Для упрощения работы можно записать этот

ключ в переменную окружения **ZIF_AGE_KEY**, и не указывать параметр **--age-key** во всех командах ниже (при постоянной работе с одной площадкой можно записать создание этой переменной в профиль **shell**):

```
export ZIF_AGE_KEY="AGE-SECRET-KEY-XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX"
```

4.2.6. Создание новой конфигурации окружения

Новая конфигурация окружения создается командой **init**. Папка для создания задается параметром **--env-path** (папка должна существовать). В этой папке создаются файлы **env-values.yaml** и **env-secrets.yaml** и несколько дополнительных.

Все параметры команды **init** кроме **--env-path** *и* **--age-key** *(*или переменная **ZIF_AGE_KEY***)* опциональные и фактически задают значения для основных параметров в файлах **env-values.yaml** и **env-secrets.yaml**. Их можно задать или изменить и после создания каталога с конфигурацией (т.к. секреты в **env-secrets.yaml** зашифрованы то задать логин и пароль **registry** будет сложнее чем остальные).

```
# Помощь по параметрам команды init
./zifctl init --help

# Информация о версиях используемых образов в скрипте, информация о версии Платформы
./zifctl init --env-path /path/to/env/config --age-key $KEY \
      --namespace $NAMESPACE \
      --sa $SA \
      --hostname $ZIIOT_HOSTNAME \
      --storage $STOR_CLASS \
      --registry $REGISTRY \
      --registry-username $REG_USERNAME \
      --registry-password $REG_PASSWORD \
      --verbose
```

Основные параметры команды **init**, задающие значения в новой конфигурации:

- **--namespace** — задает **Namespace** куда будет устанавливаться **Платформы**.
- **--sa** — **Service Account** для запуска инфраструктурных контейнеров, которые требуют запуска от конкретного **UID** (т.е. повышенных привилегий). Актуально в первую очередь для **OpenShift/OKD** инсталляций. Если не задан, то все запускается от учетной записи по умолчанию.

- **--hostname** — базовое доменное имя для доступа к экземпляру Платформы (по этому имени будет доступен портал, а сервисы будут иметь пути вроде **https://<base-hostname>/<service-name>** *(указывается не ссылка* **URL***, а именно* **hostname***).*
- **--storage** — имя **Storage Class** для создания **PVC** всей инфраструктурой. Если не задано, то используется **SC** по умолчанию в кластере.
- **--registry** — имя репозитория откуда кластер **K8s/OpenShift** должен брать контейнеры Платформы и инфраструктуры (по умолчанию **registry.dp.zyfra.com**).
- **--registry-username** — имя пользователя для репозитория. Из него и **registry-password** формируется **pullSecret**, в большинстве случаев должно быть указано, если только это не полностью открытый внутренний репозиторий).
- **--registry-password** — пароль пользователя для репозитория.

При необходимости внесите изменения в созданную в команде **init** конфигурацию окружения. Для создания простого стенда Платформы с размещением всей инфраструктуры в кластере достаточно указанных выше параметров.

Все основные параметры развертывания указываются в **env-values.yaml***.*

Секреты для инфраструктурных сервисов указываются в файле **env-secrets.yaml***.* Данные в этом файле зашифрованы с помощью `age` и для доступа к ним можно воспользоваться группой команд **zifctl secrets** (краткая помощь **zifctl secrets --help**).

Секреты генерируются автоматически, и вам нужно их править только если используется внешние инфраструктурные сервисы, например СУБД, которые развернуты независимо от **zifct** (или если не указан пароль к **registry** при выполнении **Init**).

4.2.7. Доступ к кластеру Kubernetes или OpenShift для развертывания и переменные ZIF_SERVER и ZIF_TOKEN

Для развертывания Платформы в кластер **Kubernetes** или **OpenShift** вам понадобится токен сервисного аккаунта или пользователя, позволяющий выполнять изменения в указанном **Namespace** (у него должна быть роль **admin** или близкая к этому на данный **namespace**).

- Если Платформа устанавливается в **Kubernetes**, то необходимо создать **Service Account** для этой задачи и получить его токен (см. о **SA** и токенах по ссылке, токен указывается в раскодированном виде, не в **base64**).
- Если Платформа устанавливается в **OpenShift**/**OKD** то есть возможность получить токен через **UI** для своей рабочей записи, или создать **SA** для установки так же как и для **Kubernetes**.
- Этот **Service Account** не обязан быть тем же, что и указанный для запуска подов в **env-values.yaml***.* Он может отличаться, т.к. обычно аккаунту для запуска подов не рекомендуется давать права для внесения изменений в кластер.

Для всех команд утилиты развертывания, которые требуют доступа в кластер **Kubernetes**/**OpenShift**, необходимо задавать два параметра командной строки **--server** и **--token**, в которых указывается **URL API** кластера и токен для доступа соответственно.

При работе с одной и той же площадкой эти значения можно задать в виде переменных окружения и не указывать их в каждой команде (так же как ключом шифрования и переменной **ZIF_AGE_KEY**) или прописать их в профиль **shell**:

```
export ZIF_SERVER="https://api.cluster.local"
export ZIF_TOKEN="eyJhbGcNYWNjb3VuIsImtpZCI6IkZLOE5DSUlPZ0dtMXhPYUhvTDc2eGtf..."
```

4.2.8. Развертывание или обновление Платформы

Команда для развертывания и для обновления платформы одна и та же. Практически все операции, выполняемые системой развертывания, идемпотентны и при повторном выполнении установки, поверх уже выполненной ранее ничего не должно измениться.

Команда установки («синхронизации» конфигурации с конфигурацией в кластере). Параметры **--age-key**, **--server**, **--token** могут быть заданы через переменные окружения **ZIF_AGE_KEY**, **ZIF_SERVER** и **ZIF_TOKEN** соответственно).

```
# Запуск развертывания с выводом информации только на консоль
./zifctl sync --env-path /path/to/env/config --age-key $KEY --server $SERVER --token $TOKEN --verbose


# Запуск развертывания с выводом информации на консоль и в лог-файл с ротацией (параметр --log-keep не обязательный, по умолчанию сохраняется 10 последних логов)
./zifctl sync --env-path /path/to/env/config --age-key $KEY --server $SERVER --token $TOKEN --verbose --log-path /var/logs/ziiot-deploy --log-keep 20
```

Если выполняется установка поверх предыдущей версии, то в общем случае должно выполниться обновление сервисов до новой версии.

Но т.к. **Платформа** представляет собой сложный комплекс из большого количества сервисов и компонентов, то полностью автоматическое обновление условно гарантируется только в рамках одного релиза (например, из 2.9.0 в 2.9.1), а обновление между релизами (например, из 2.8.3 в 2.9.0) может потребовать ручных операций.

Перед обновлением необходимо прочитать **Release Notes Платформы** и системы развертывания и убедиться, что нет обязательных ручных операций.

4.2.9. Удаление развернутой Платформы из кластера

Команды для удаления развернутой платформы из кластера:

```
# Удаление всех сервисов и инфраструктуры без удаления PVC
./zifctl delete --env-path /path/to/env/config --age-key $KEY --server $SERVER --token $TOKEN --verbose


# Удаление всех сервисов и инфраструктуры с удалением PVC (полная очистка). Созданные в Namespace дополнительно вручную объекты удалены не будут.
./zifctl clean --env-path /path/to/env/config --age-key $KEY --server $SERVER --token $TOKEN --verbose
```

4.2.10. Отладочные команды

При выявлении каких-то проблем с развертыванием платформы на любом этапе работы системы, можно воспользоваться командами **write-values** и **template,** а также ключом выдачи отладочной информации **--verbose** и **–debug.**

Система развертывания **Платформы** в целом работает по следующей схеме (схема упрощенная, в ней не указаны различные вспомогательные скрипты на разных этапах и не указана система инициализации сервисов, запускаемая уже в кластере после загрузки манифестов):

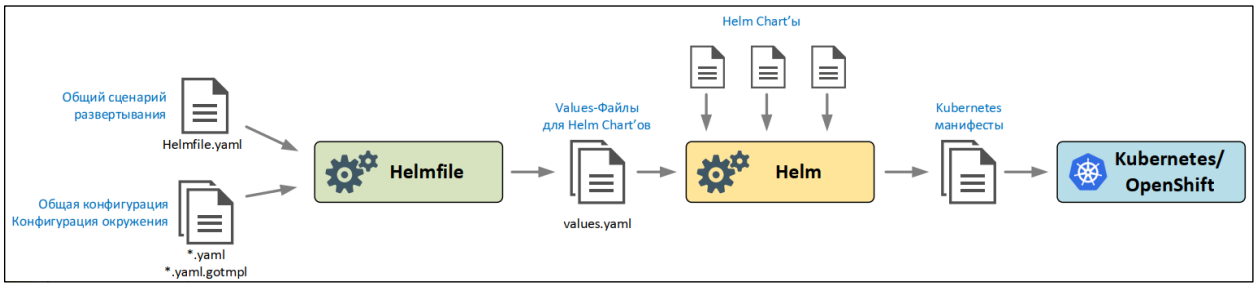

Рисунок 4.17 Отладочные команды

1) На основании файлов сценариев **Helmfile***, шаблонов конфигурации (**values***) и при участии доп. скриптов создаются параметры для **Helm Chart** (эту функцию выполняет утилита **Helmfile**).
2) Полученные параметры (**values**) передаются в **Helm Chart**, из которых генерируются манифесты для объектов **K8s** (эту функцию выполняет **Helm**).
3) Созданные манифесты загружаются в кластер **Kubernetes**, при необходимости с обновлением уже существующих (эту задачу тоже выполняет **Helm**).
4) Запускаются объекты заданий (**Job**) внутри кластера, которые выполняют начальную инициализацию развернутой инфраструктуры и сервисов (этот этап на схеме не указан).

В системе развертывания предусмотрены команды, позволяющие посмотреть генерируемые значения для **chart** и генерируемые манифесты на основании этих значений (аналогично командам **Helm**):

```
# Запись значений, полученных на первом этапе, которые затем должны быть переданы в Helm Chart. Значения записываются по пути, указанном в параметре --output-path

./zifctl write-values --env-path /path/to/env/config --age-key $KEY --output-path $OUTPUT_PATH


# Запись манифестов, сгенерированных Helm на основании переданных значений (результат второго этапа). Манифесты записываются по пути указанном в параметре --output-path

./zifctl clean --env-path /path/to/env/config --age-key $KEY --server $SERVER --token $TOKEN --verbose
```

Количество выдаваемой на консоль (и в лог при использовании параметра **--log-path**) информации регулируется двумя опциональными параметрами **zifctl**:

- **--verbose —** включает отладочные сообщения из скриптов самой системы развертывания **zifctl** и вызываемых им.

- **--debug —** включает помимо отладочных сообщений системы развертывания еще и отладочные сообщения в **Helmfile** и **Helm** (а также в скрипте обертке).

4.2.11. Получение списка требуемых docker-образов для развертывания и export/import образов

Для запуска Платформы в кластере **Kubernetes** требуются **docker**-образы сервисов Платформы, образы всей устанавливаемой инфраструктуры (**Postgres**, **Cassandra**, **Kafka** и т.д.) и образ самой системы развертывания (**zifctl**).

В отличие от предыдущих релизов, образы с **Helm Chart** и копия **git**-репозитория не требуются (это все включено в образ **zifctl**).

Все необходимые **docker**-образы размещаются в репозитории компании **Zyfra**, который система развертывания использует по умолчанию: https://registry.dp.zyfra.com.

Платформу часто приходится развертывать в закрытых контурах, куда **docker**-образы приходится переносить вспомогательными средствами, поэтому в систему развертывания включены команды, позволяющие получить список требуемых образов и сгенерировать заготовки скриптов для их выгрузки и загрузки в другой репозиторий.

```
# Получение списка требуемых docker-образов. Формат вывода указывается в параметре --output и может быть: table, list, json, yaml, csv  (для удобства интеграции с инструментами для экспорта-импорта образов)
./zifctl image list --output table

# Получение заготовок скриптов для экспорта (pull-save) и импорта в другой репозиторий (load-tag-save). Скрипты сохраняются по пути --output-path
./zifctl image scripts --output-path /path/to/save/scripts

# Получение заготовок скриптов для экспорта (pull-save) и импорта в другой репозиторий (load-tag-save). Скрипты сохраняются по пути --output-path
./zifctl image scripts --output-path /path/to/save/scripts
```

4.2.12. Руководство по обновлению

Специальные требования к процессу обновления:

1) Для некоторых топиков `debezium` заданы параметры в `services/kafka-topics-list.yaml`.

   При обновлении с предыдущего релиза и нестандартной конфигурации статичных топиков `debezium`, необходимо их отразить в `services/service-list-patch.yaml`. Количество партиций в топиках нельзя уменьшать автоматически. Управление доступно только для топиков начинающих с `<tenantid>__`.

2) Удален параметр `keylocak.baseUri`, при обновлении с предыдущего релиза его необходимо удалить из `env-values.yaml`.

4.2.13. Генерация документа (zifctl doc) env-values - конфигурирования переменных

Добавлена команда `zifctl doc`, вместе с подкомандой (пока единственной) `env-values`, которая и генерирует документацию по env-values.yaml.

Обязательный параметр: `--ouput=path` - папка в которую сохранять документацию. Необязательный: `--format` - принимает значения `md/html`, по умолчанию md

Пример команды:

1) Запуск из `docker`-образа:

```
docker run --rm -it \
-v "<путь для сгенерированных файлов документации>":/output-path \
docker-group.idp.yc.ziiot.ru/infra/zifctl:2.16.0 \
doc env-values --output-path=/output-path
```

По указанному пути будет сгенерирован файл `env-values.md` в формате `markdown`.

По указанному пути будет сгенерирован файл `env-values.html` в формате `html`:

```
docker run --rm -it \
-v "<путь для сгенерированных файлов документации>":/output-path \
docker-group.idp.yc.ziiot.ru/infra/zifctl:2.16.0 \
doc env-values --output-path=/output-path --format html
```

2) Запуск из враппера (wrapper) zifctl:

```
docker pull docker.idp.yc.ziiot.ru/infra/zifctl:2.16.0
docker run --rm  docker.idp.yc.ziiot.ru/infra/zifctl:2.16.0 get-wrapper > zifctl
./zifctl doc env-values --output-path .
```

По указанному пути будет сгенерирован файл `env-values.md` в формате `markdown`:

```
docker pull docker.idp.yc.ziiot.ru/infra/zifctl:2.16.0
docker run --rm  docker.idp.yc.ziiot.ru/infra/zifctl:2.16.0 get-wrapper > zifctl
./zifctl doc env-values --output-path . --format html
```

По указанному пути будет сгенерирован файл `env-values.html` в формате `html`.

4.3. Развертывание через Jenkins

4.3.1. Универсальный pipeline Платформы и дополнительных проектных решений

Универсальный **pipeline** предназначен для централизованного управления конфигурациями инфраструктуры, платформы и дополнительных проектных решений.

Данный **pipeline** при работе интерактивно запрашивает сведения, необходимые для установки (обновления), посредством стандартного ввода **jenkins**.

Установка платформы осуществляется при помощи установщика **zifctl**.

К дополнительным проектным решениям относятся:

- инфраструктурные сервисы, не входящие в поставку "Платформы" и развертываемые через **ansible** или **helm**;
- приложения, работающие поверх "Платформы" (развертывание через **helm**).

4.3.2. Подготовка запуска пайплайна

Подготовка Jenkins

Для подготовки **Jenkins** потребуется:

- ссылка на данный репозиторий, **https://** или **ssh://**;
- учетные данные от репозитория, могут быть двух видов:
  - для **https://** - как правило логин-токен для работы с gitlab. У токена должны быть права на pull из дефолтную ветки репозитория.
  - для **ssh://** - логин и приватный **ssh**-ключ (для **Azure DevOps** в Газпромнефти).

Далее:

1. Создаем в **Jenkins** новый **item** - **Folder**, в котором будут находиться пайплайны.
2. Если мы работаем с **gitlab**, тогда создаем **credential**, которые позволят нам добавить разделяемую библиотеку из git-репозитория, а также настроить нужные нам **multi-branch pipeline**. Здесь логин может быть произвольный (хорошая практика, когда он совпадает с токеном доступа к **репозитирию**). Ниже картинка, иллюстрирующая добавление **credential** для **gitlab** в **Jenkins**:

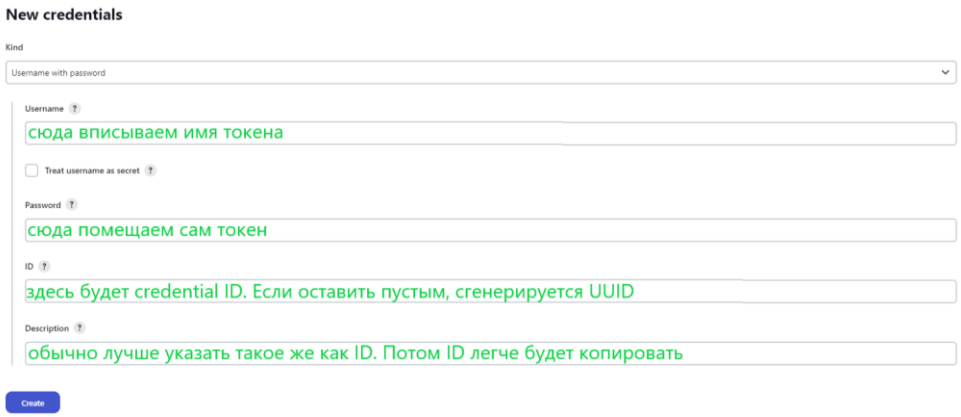

Рисунок 4.18 credential

Если мы работаем внутри закрытого контура, нам нужно проверить доступность **credentialID "git-tfs" (**служебная учетная запись для клонирования репозиториев). Создавать отдельно **credentials ID** с SSH-ключом внутри не нужно.

3. Переходим внутрь созданного на первом шаге **Folder`a Jenkins**, открываем **Configure**, и добавляем ссылку на разделяемую библиотеку, как на изображении ниже:

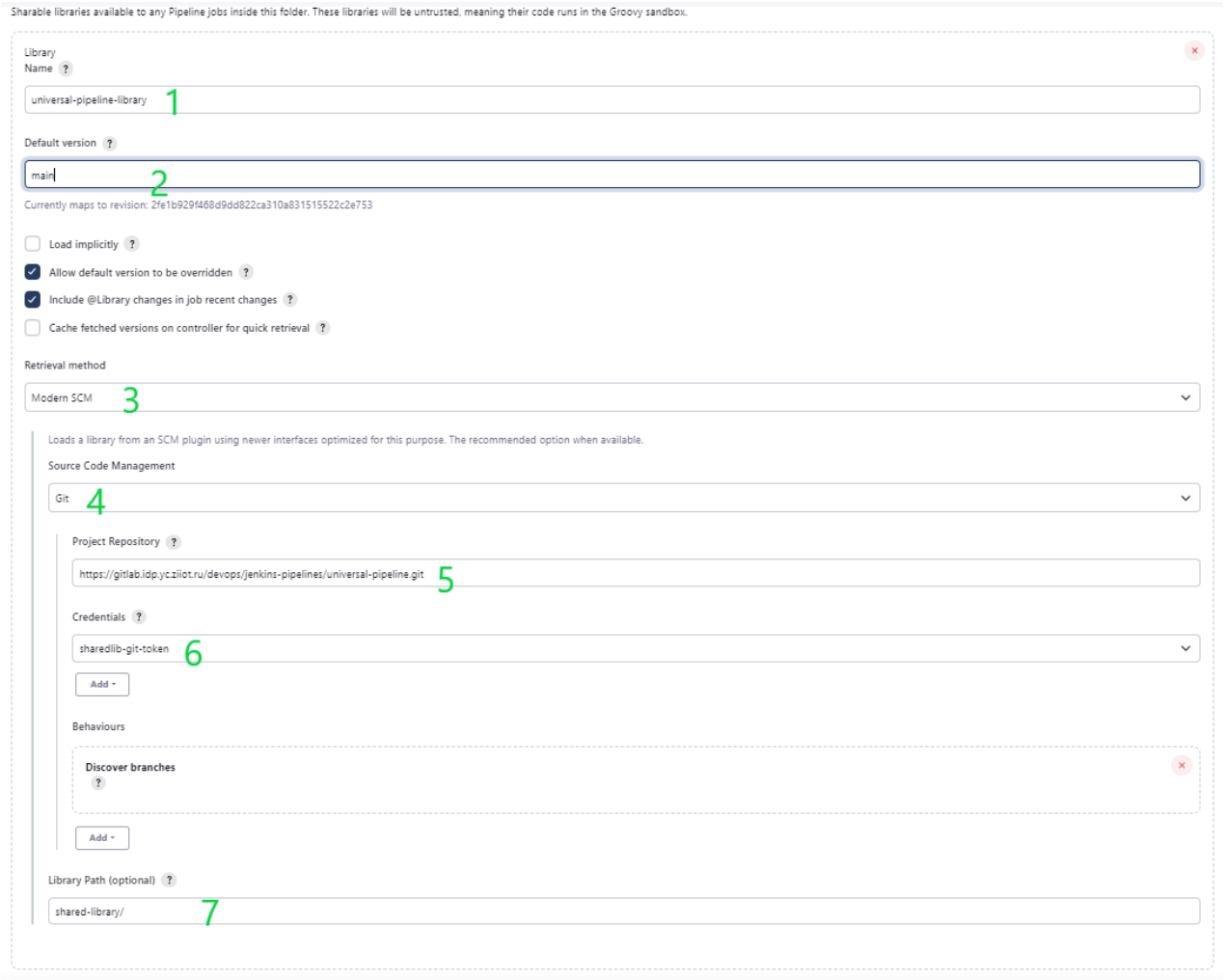

Рисунок 4.19 Configure

В поле 1 мы обязательно правильно указываем наименование библиотеки, под которым будем ее включать в пайплайн - **'universal-pipeline-library'**. В поле 2 указываем или имя тега, или ветку (на момент написания данного мануала не выпущен ни один тег, ветка **main** считается стабильной, и лучше указывать именно ее). **Retrieval method** (3) указываем как "Modern SCM", Source Code **Management** (4) указываем как "Git". В **Project Repository** (5) нам необходимо указать актуальный URL, по которому можно клонировать данный репозиторий. В поле **Credentials** (6) из выпадающего списка требуется выбрать **secretID** из предыдущего шага. Обязательно не забыть указать путь к библиотеке (7) - **'shared-library/'**.

4. Теперь можно создать **multibranch-pipeline**. Сейчас их доступно два - **Main** и **Helper**. **main** пайплайн отвечает за запуск **zifctl**(деплой платформы), запуск **ansible**-ролей для поднятия инфраструктуры(на данный момент доступно 2 роли для кластеров **Cassandra** и **Patroni**), запуск **helm-charts** для установки дополнительных проектных решений, и **helm-charts** для установки приложений поверх платформы. **Helper** позволяет выполнять действия, которые затруднительно выполнить в окружении ГПН - сгенерировать пару SSH-ключей, сгенерировать закрытые ключи для утилиты **age**, зашифровать строку утилитой **ansible-vault** или же зашифровать секретную часть **values helm-chart** при помощи утилиты **SOPs**. Поэтому, как правило, если мы внутри ГПН, то нам понадобится оба **multibranch-pipeline**, если вне ГПН -

то, как правило есть ВМ с нужными утилитами, и **helper** не так нужен. Само создание **multibranch-pipeline** внутри **Folder** из п.1 выглядит следующим образом. Мы нажимаем **New Item**, задаем любое удобное для нас имя, и выбираем тип **Item** как **Multibranch Pipeline**. Дальше в конфигурации жмем "**Add Branch Source**", выбираем **Git**, и задаем ссылку на репозиторий (1) и **credentialID** (2) как в п.3. Пример заполнения - на картинке ниже:

Рисунок 4.20 Пример заполнения конфигурации

После этого необходимо указать путь к **Jenkinsfile** в **Build Configuration**. Выбираем **Mode** (1) **Jenkinsfile**, и дальше указываем путь к **Jenkisfile** (2). Для **Main** пайплайна это **'pipelines/main.jenkinsfile'**, для **helper 'pipelines/helper.jenkinsfile'**.

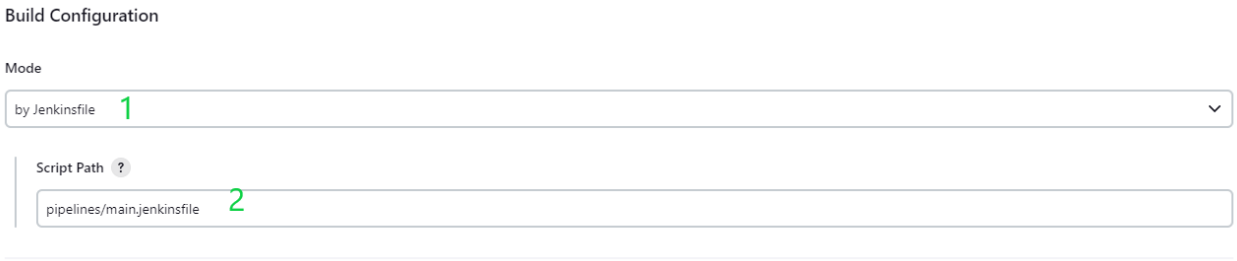

Рисунок 4.21 Указание пути Jenkinsfile

На этом подготовку **Jenkins** можно считать законченной.

Подготовка конфигурационного репозитория

Пайплайн разработан в соответствии с концепцией **IaC**, т.е. в конфигурационном репозитории будет хранится описание ландшафта в виде структурированного набора **YAML** и **JSON** файлов.

Нам потребуется:

- Пустой репозиторий, в который мы сможем пушить, так и пуллить оттуда. Важно - пока пайплайн поддерживает работу **только** с дефолтной веткой репозитория. Т.е., например роли **developer** в **gitlab** явно недостаточно, вы не сможете по умолчанию пушить в дефолтную ветку.

- Доступ в **docker-registry** с нужными образами или же **credential ID**, в котором содержаться нужные учетные данные. Достаточно только доступа на pull образов.

Дальше выполняем следующие действия:

1. Если мы запускаем пайплайн внутри ГПН - то запускаем **helper**, выбираем **generate ssh-key** (единственный функционал, который доступен без указания конфигурационного репозитория). Дальше открытую часть ключа помещаем в **Azure Devops** в настройки личной учетной записи, закрытую часть помещаем в **credential ID** типа '**SSH-key with username**'. Если мы работаем вне ГПН с **gitlab**, то достаточно сгенерировать или себе персональный токен для доступа к репозиторию, или же сгенерировать токен доступа к проекту.
2. (Опционально) - настройка **helper**. За его конфигурацию отвечает файл **helper.yaml** в корне конфигурационного репозитория. Образец файла с комментариями-пояснениями есть в директории данного репозитория **config-data-example**. Структура данного каталога иллюстрирует структуру конфигурационного репозитория. Если нет доступного **credential ID**, который даст нам доступ в docker-registry, его необходимо создать, желательно на уровне **Folder** в **Jenkins**.
3. Настройка **main pipeline**. За нее отвечает файл в корне репозитория **config.yaml**. Пример с пояснениями доступен в каталоге **config-data-example** данного репозитория. Он не маленький, поэтому лучше есть этого слона по частям. Копируем файл-пример в корень конфигурационного репозитория, и меняем под свое окружение секцию **common**. Если нет доступного **credential ID**, который даст нам доступ в **docker-registry**, его необходимо создать, желательно на уровне **Folder** в **Jenkins**.
4. (Опционально) - указываем URL от нашего репозитория в параметрах **multibranch-pipeline**. Это нужно, чтобы каждый раз не копировать-вставлять ссылку на конфигурационный репозиторий, а указывать только токен. Для этого в **jenkins** внутри **multibranch-pipeline** мы идем в **config**, пролистываем до **Folder Properties**, и добавляем свойство (1) **URL_CONFIG_REPO**, и устанавливаем значение (2) как на примере ниже:

Рисунок 4.22 Добавление свойства

Подготовка и деплой инфраструктуры при помощи Ansible

Нам потребуется:

- Предварительно созданные и подготовленные виртуальные машины, куда мы хотим поместить инфраструктуру. (Подготовка подразумевает настройку пользователя с правами администратора, а также настройку **SSH**-сервиса, которая позволит данному пользователю авторизоваться по **SSH**-ключам)
- Ознакомиться с доступными ролями **ansible**. На момент написания данного документа были протестированы роли развертывания для **Cassandra** и **Patroni**.

Порядок действий таков:

1. Настраиваем общую секцию **manualInfra** внутри файла **config.yaml**:
   - Указываем ссылку на актуальный для окружения репозиторий с ролями **ansible**, а также **credentials ID**, который позволят склонировать его. В случае отсутствия доступного **credential ID** его необходимо создать. Все роли в репозитории должны лежать внутри каталога **src**, расположенного в корне репозитория.
   - Указываем **credential ID**, содержащий имя пользователя и закрытый приватный **SSH**-ключ, под которыми можно подключится по **SSH** на целевые ВМ. В случае отсутствия доступного **credential ID** его необходимо создать.
   - Отдельно указываем **hostUser**, для которого настроен **SSH** из предыдущего пункта.
   - **roles-path** оставляем по умолчанию, параметр вскоре будет удален.
   - Указываем **credential ID**, который будет содержать **vault**-пароль для расшифровки строк с чувствительной к распространению информации. В случае отсутствия доступного **credential ID** его необходимо создать.
   - Указываем актуальную для окружения ссылку, по которой можно спуллить образ с **ansible** на борту.
2. Настраиваем секцию **manualInfra > services** и готовим **inventory** с плейбуками для запуска ролей. Количество элементов списка будет соответствовать количеству инфраструктурных сервисов. Для сервисов настраиваем **name** и **playbookPath** (имя каталога с **inventory** для и **playbook**). Внутри каждого **inventory** должны быть файлы **playbook.yaml** и **hosts**.
3. Настраиваем **inventory**. Указываем необходимые для роли переменные. Для шифрования при помощи **ansbile-vault** можно использовать **helper**-пайплайн.
4. Запускаем **main**-пайплайн в **Jenkins**, переходим в **console output** внутри сборки, ждем появления сообщения "**input requested**". Указываем в форме ссылку и токен на конфигурационный репозиторий, как на рисунке ниже:

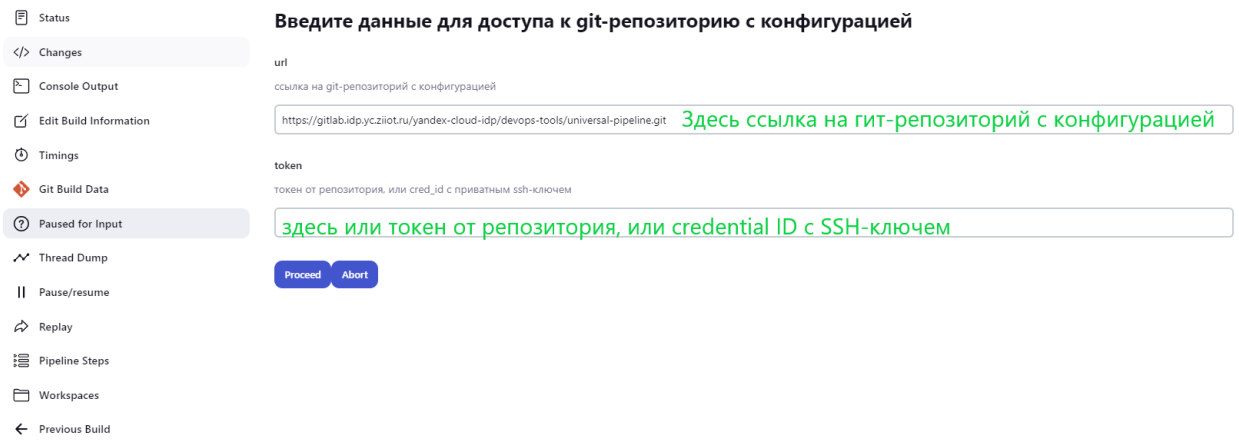

Рисунок 4.23 Ссылка на конфигурационный репозиторий

Дальше выбираем компонент "**infra**", как проиллюстрировано ниже:

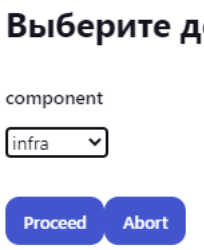

Рисунок 4.24 Выбор компонента

В форме по запуску **ansible** будет количество чекбоксов, равное числу элементов внутри списка секции **manualInfra > services config.yaml**. Отмечаем нужные нам, выбираем **verbosity** логов, и запускаем.

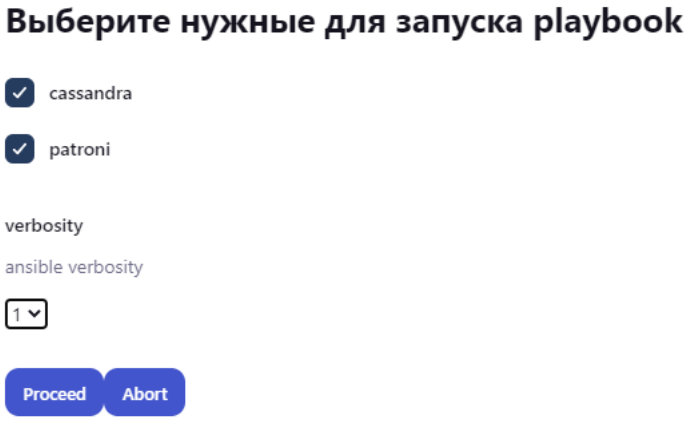

Рисунок 4.25 Выбор playbook

Отдельно заполняем форму подтверждения запуска:

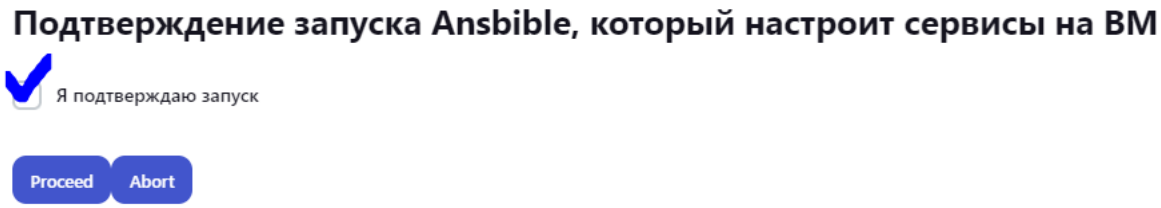

Рисунок 4.26 Подтверждение запуска

4.4. Подготовка и деплой платформы ZIIOT при помощи утилиты zifctl

Для разворота платформы нам понадобятся:

- Готовые **Namespace** внутри **Kubernetes/OKD/Openshift** и сервис-аккаунтами (SA) с соответствующими **RoleBinding**.

Схематично порядок действий описан ниже:

1. Настраиваем секцию **config.yaml ziiot > platforms**. Определяемся с количеством и типом платформ в заданном окружении. Элементы списка настраиваются со следующими параметрами:

- Параметр "**name**" - имя платформы. Для версий ZIIOT от 2.13.0 налагаются ограничения на имя, не длиннее 16 символов.
- Параметр "**type**" может принимать три значения, в зависимости от типа платформы внутри. Это "**sharedInfra**", "**tenant**" и "**instance**".
- Параметр "**namespace**" содержит значение, куда деплоится конкретная платформа.
- Параметр "**ageKeySecretID**" содержит **credential ID** с закрытым ключом, сгенерированным утилитой Age. Если нет доступного **credential ID** его необходимо создать. Закрытый ключ можно сгенерировать при помощи **helper**-пайплайна.
- Параметр "**saName**" - имя сервис-аккаунта, из-под которого осуществляется деплой платформы **ZIIOT**.
- Параметр "**saTokenSecretID**" - **credentials ID**, содержащий в себе токен от указанного пунктом выше сервис-аккаунта **saName**.
- Составной параметр "**externalCassandra**" отвечает за интеграцию платформы ZIIOT и внешнего сервиса **Cassandra**. Требуется указать внутри, используется ли (параметр "**used**"), список хостов кластера внутри (параметр "**contactPoints**"), и порт, по которому доступна на хостах **cassandra**. Параметр "**secretID**" содержит логин-пароль административного пользователя **Cassandra**.
- Составной параметр "**externalPostgres**" отвечает за интеграцию платформы ZIIOT и внешнего сервиса **Postgres**. Требуется указать внутри, используется ли (параметр "**used**"), хост("**host**") и порт("**port**") для подключения к кластеру. Параметр "**secretID**" содержит логин-пароль административного пользователя **Postgres**.

2. В случае если у нас пустой ландшафт, и нам нужно инициализировать новую платформу, то мы просто запускаем **main**-пайплайн, заполняем форму по конфигурационному репозиторию

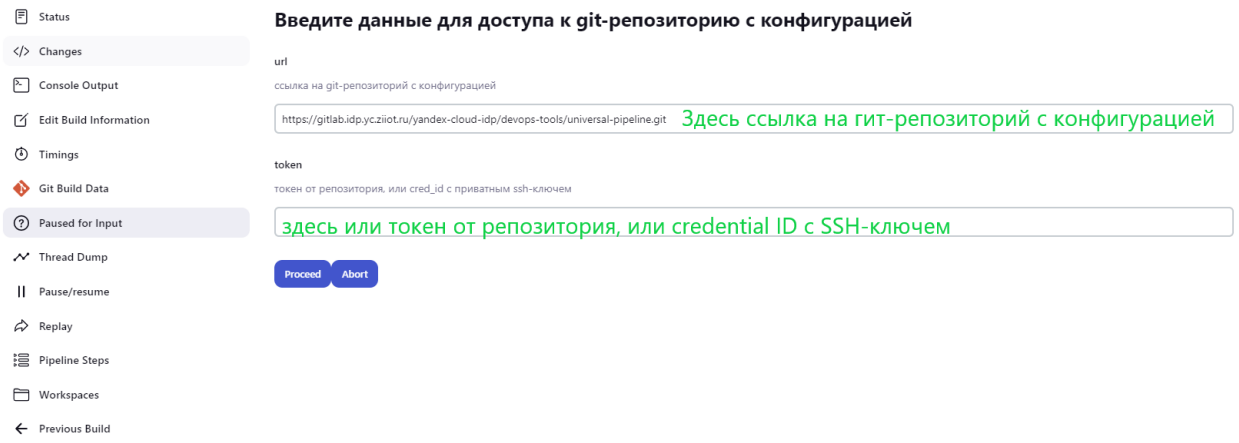

Рисунок 4.27 Данные git-репозитори

3. Выбираем компонент **platform**:

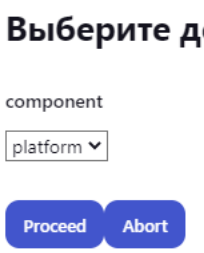

Рисунок 4.28 Выбор компонента

4. Выбираем тип платформы в зависимости от наших пожеланий

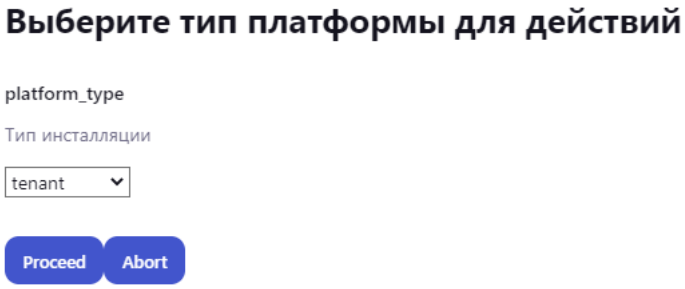

Рисунок 4.29 Выбор типа платформы

5. Выбираются детали операции (**name** платформы и версию), в случае инициализации это что то, содержащее подстроку **init** в нижнем выпадающем списке:

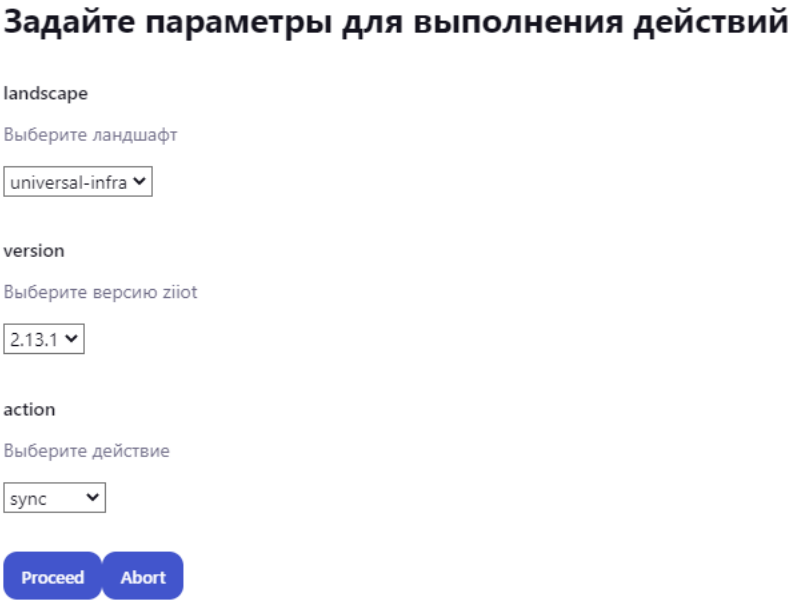

Рисунок 4.30 Выбор параметров действий

В ходе инициализации платформы будет сгенерирован конфиг окружения, который окажется в папке **ziiot** в корне конфигурационного репозитория, внутри каталога с таким же именем, как и параметр "**name**" конкретной платформы. Если же нам требуется интегрировать уже развернутую платформу с уже существующим конфигом окружения, необходимо данный каталог поместить внутрь каталога **ziiot** в папку с таким же именем, как и параметр "**name**" платформы.

6. В дальнейшем можно проделывать нужные Вам действия при помощи **main**-пайплайна, доступные при выборе параметра "**platform**". Деплой платформы осуществляет действие "**sync**". "**delete**" удалить все сущности платформы, кроме **PVC**. "**clean**" удалит все и **PVC** в том числе.

4.5. Дополнительные проектные решения

Развертывание дополнительных проектных решений сводится к развертыванию Helm. Здесь особых пререквизитов нет, предполагается лишь что у нас есть на момент развертывания NS и SA для развертывания.

1. Заполняем секцию **manualHelm**. Предполагается, что все дополнительные проектные решения будут лежать в едином репозитории. В секции **general** указываем **url** на репозиторий, секрет к нему, ссылку на **registry**, по которой можно сделать **pull** образа с **helm** на борту, а также **credentials ID age**-ключем, которым будем шифровать чувствительную к распространению информацию.

2. Дальше формируем список **namespaces**. Как легко догадаться, в каждом элементе списка параметр "**name**" соответствует имени **namespace**, "**saTokenSecretID**" - **credential ID**, содержащий токен от сервис-аккаунта kubernetes, под которым будет идти установка. Дальше в "**charts**" указываем **Helm** -чарты, которые мы хотим ставить. Опционально для некоторых чартов можно указать, какие файлы подкладывать в **workspace** в директории **certificates** и **keytabs**:

Внимание! названия чартов тождественно именам папок по каждому чарту, в репозитории с чартами по дополнительным проектным решениям.

3. В корне репозитория создаем папку **helms** (если ее еще нету там). Внутри создаем папку с названием, как **namespace**, куда планируем ставить чарт. Дальше создаем каталог как имя **чарта**, в нем файл **values.yaml** и (опционально) **secrets.yaml**. В **values.yaml** содержится незашифрованная нечувствительная к распространению информация, а в **secrets.yaml** зашифрованная утилитой **SOPs** закрытая информация. При установке **Helm** чарта пайплайном расшифровка будет осуществлена закрытым age-ключем из credential ID из пункта 1("**helmAgeSecretID**"). Зашифровать **secrets.yaml** можно при помощи **helper**-пайплайна.

4. Запускаем **main**-пайплайн, указываем репозиторий и токен от него (или **credID** с **SSH** ключем в случае **SSH**-ссылки):

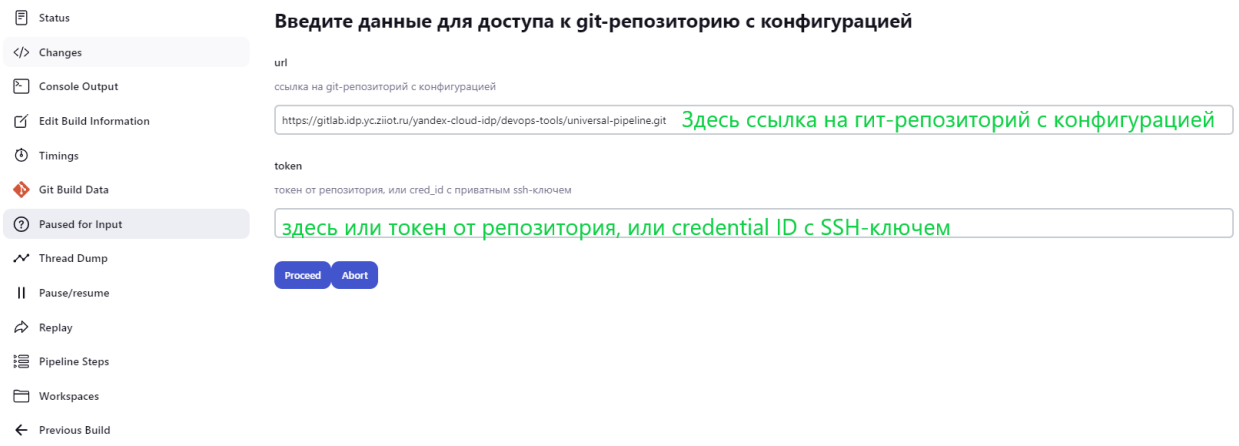

Рисунок 4.31 Ссылка на доступ git-репозиторий

5. Дальше выбираем параметр **helms**, как проиллюстрировано ниже:

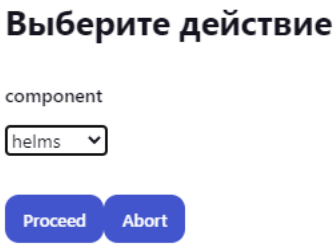

Рисунок 4.32 Выбор компонента helms

И теперь откроется диалоговое окно в Jenkins по выбору самих чатров, которые мы настроили в **config.yaml**. Каждый чарт будет указан как чекбокс с именем в формате **namespace::chart-name**. Выбираем нужные и выбираем действие с ними. "**template&dry-run**" - сделает **helm template**, **helm test** - протестирует уже выкаченный релиз, и **helm update --install --dry-run. "apply"** - установит чарт через **helm upgrade --install**. "**delete**" удалит чарт из неймспейса.

4.6. Деплой приложений

Развертывание приложений сводится к развертыванию **Helm**. Здесь особых пререквизитов нет, предполагается лишь что у нас есть на момент развертывания NS и SA для развертывания.

Проделываем следующее:

1. Секция **config.yaml** касательно приложений - **apps**. Здесь список приложений, заполняем его, для каждого приложения указываем:
   - Параметр "**name**" - наименование приложения. Выбираем какой больше нравится, но в списке приложений должен быть уникальным
   - Параметр "**helmRepoUrl**" - репозиторий с **Helm**, который задеплоит приложение.
   - Параметр "**helmRepoSecretID**" - **credential ID**, содержащий логин-пароль, который позволят склонировать репозиторий из **helmRepoUrl**. В случае отсутствия доступного **credential ID** его необходимо создать.
   - Параметр "**namespace**" - неймспейс k8s, куда будем ставить наше приложение.
   - Параметр "**saTokenSecretID**" - credential ID с токеном от SA на борту, из-под которого будет выполнятся установка. Если такого credential ID нет, то его необходимо создать.
   - Параметр "**helmTimeout**" - опционально указывается для переопределения таймаута установки **Helm** чарта приложения. Указывается если приложение не успевает установиться за 10 минут (таймаут по умолчанию).
   - Параметр "**valuesFiles**" - опционально указывается для использования нескольких **values** файлов для установки **helm**-чарта приложения. Представляет собой YAML массив. Values файлы применяются в порядке их указания. Эти файлы размещаются в том же каталоге, что и стандартный **values.yaml**.
2. После того как список в **config.yaml** готов, необходимо в корне репозитория создать папку **apps** (если ее еще нет). Далее внутри создаем каталог (или каталоги), которые называются как **namespace** для каждого элемента списка. Внутри этих каталогов создаем каталоги, которые соответсвуют **name** приложению. Пример иерархии можно увидит в каталоге **config-data-example** в этом репозитории. В каталоге для приложения создаем файлы values.yaml и (опционально) **secrets.yaml**. В **values.yaml** содержится незашифрованная нечувствительная к распространению информация, а в secrets.yaml зашифрованая утилитой **SOPs** закрытая информация. При установке **Helm** чарта пайплайном расшифровка будет осуществлена закрытым **age**-ключем из **credential ID** из **manualHelms > general > helmAgeSecretID**. Зашифровать **secrets.yaml** можно при помощи **helper**-пайплайна.

3. И теперь запускаем пайп, сначала указываем репозиторий и токен от него (или **credID** с **SSH** ключом в случае **SSH**-ссылки):

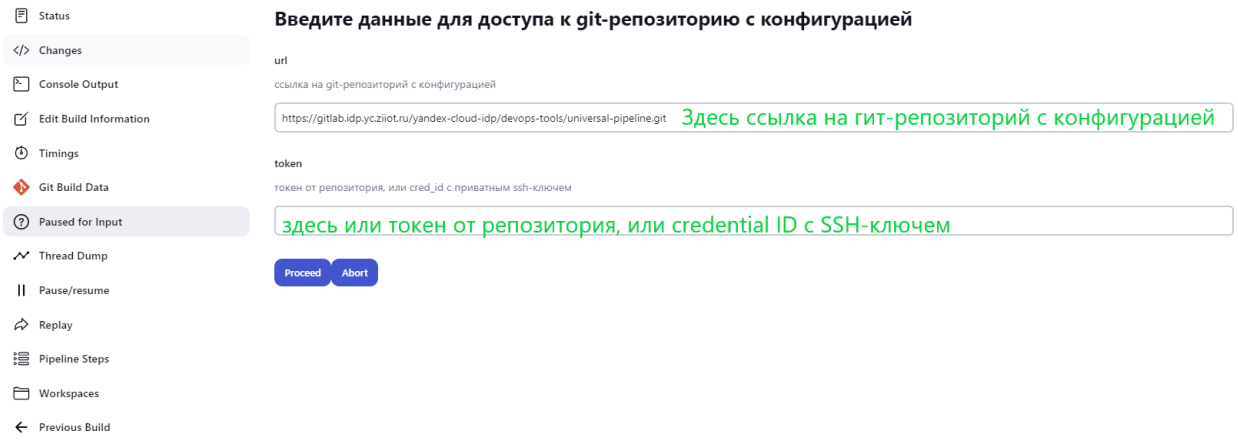

Рисунок 4.33 Выбор пути к репозиторию

Потом выбираем параметр **apps**:

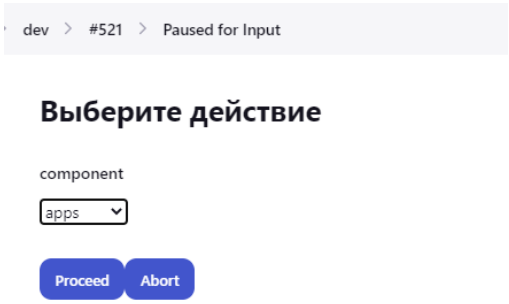

Рисунок 4.34 Выбор компонента apps

Теперь выбираем нужные нам приложения. Каждое приложение будет указано как чекбокс с именем в формате **namespace::name**. Выбираем нужные и выбираем действие с ними. "**test**" - сделает **helm template** и **helm update --install --dry-run. "apply**" - установит чарт через **helm upgrade --install**. "**delete**" удалит чарт из неймспейса.

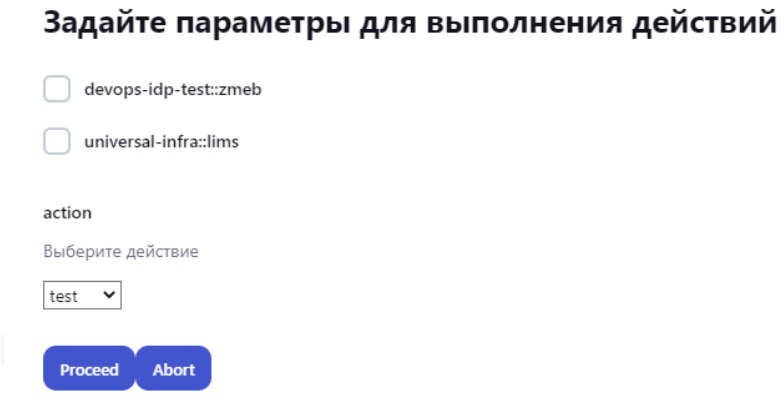

Рисунок 4.35 Выбор параметров для действий

4.6.1. Упрощенный пошаговый алгоритм развертывания через Jenkins

Для развертывания в Jenkins необходим репозиторий с хранимыми переменными **config.yaml**. Он аналогичен **env.values.yaml** из раздела **Развертывание впервые автоматически пайплайном с помощью Gitlab и Kubernettes**:

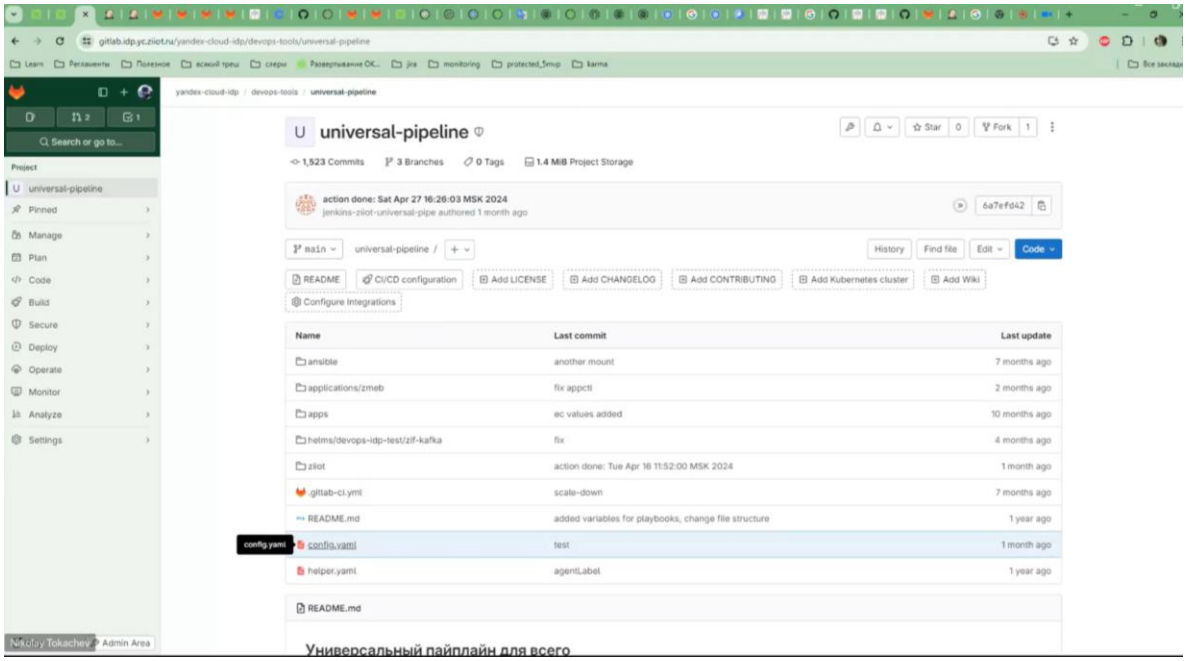

Рисунок 4.36 Выбор конфигурационного файла

1. Перейдите в **Jenkins** в переданный для сборки проект:

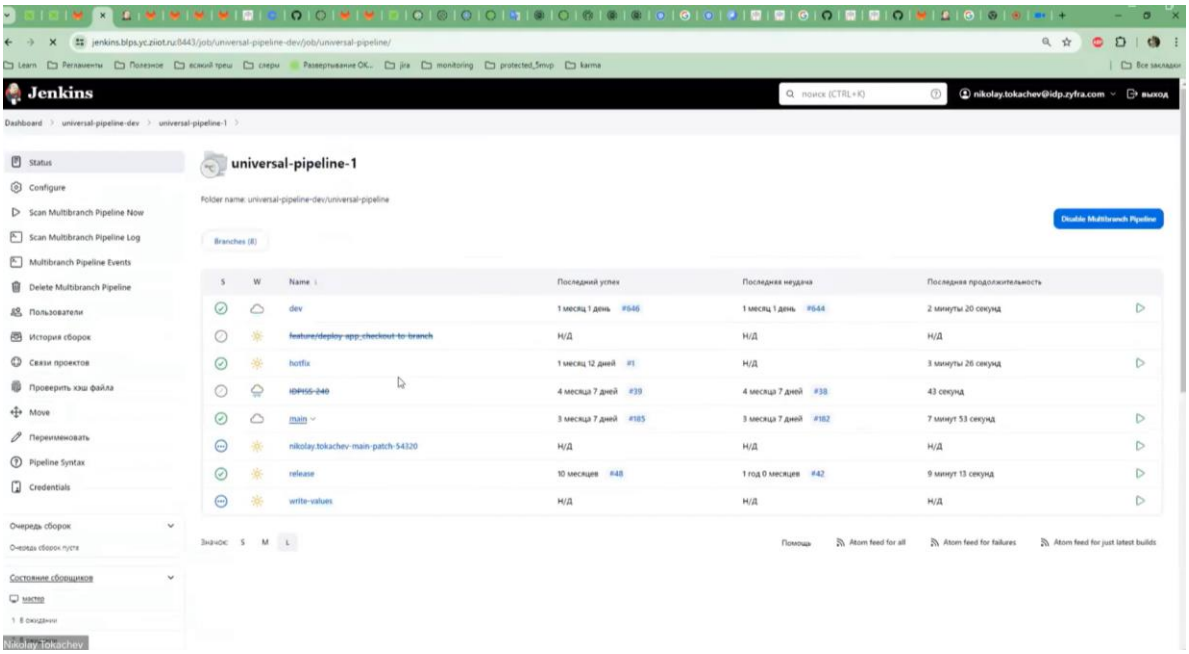

Рисунок 4.37 Переход в проект

2. Нажмите команду сборки **Собрать сейчас**:

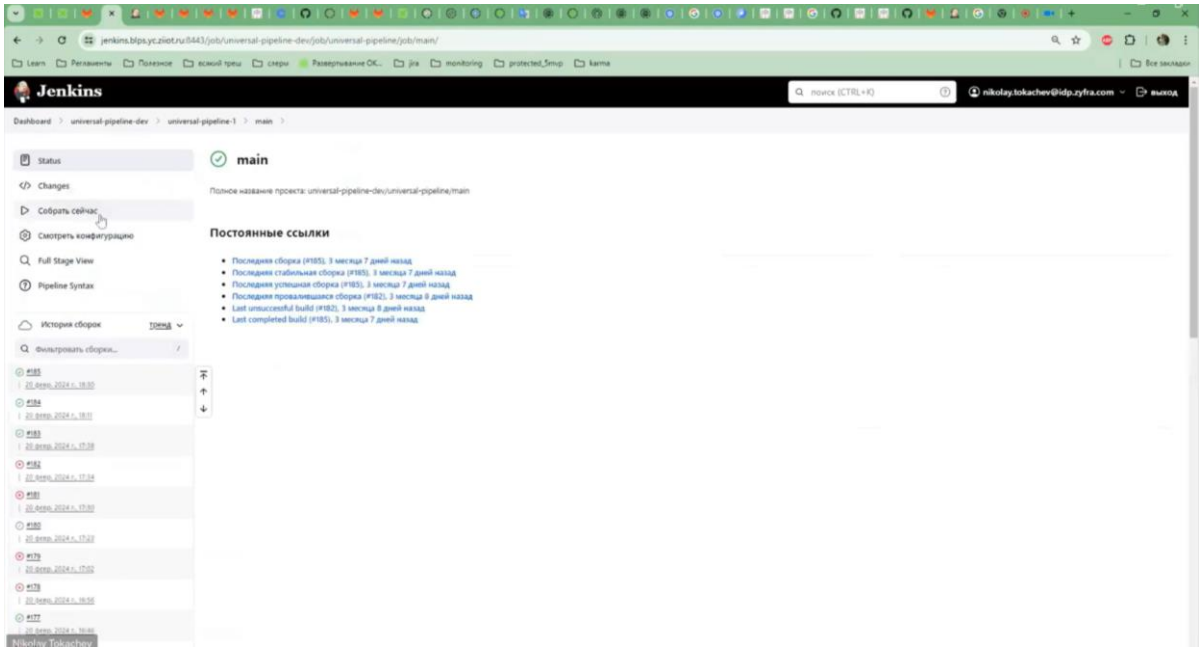

Рисунок 4.38 Запуск сборки

3. При сборке появится пауза с командой **Input Requested**:

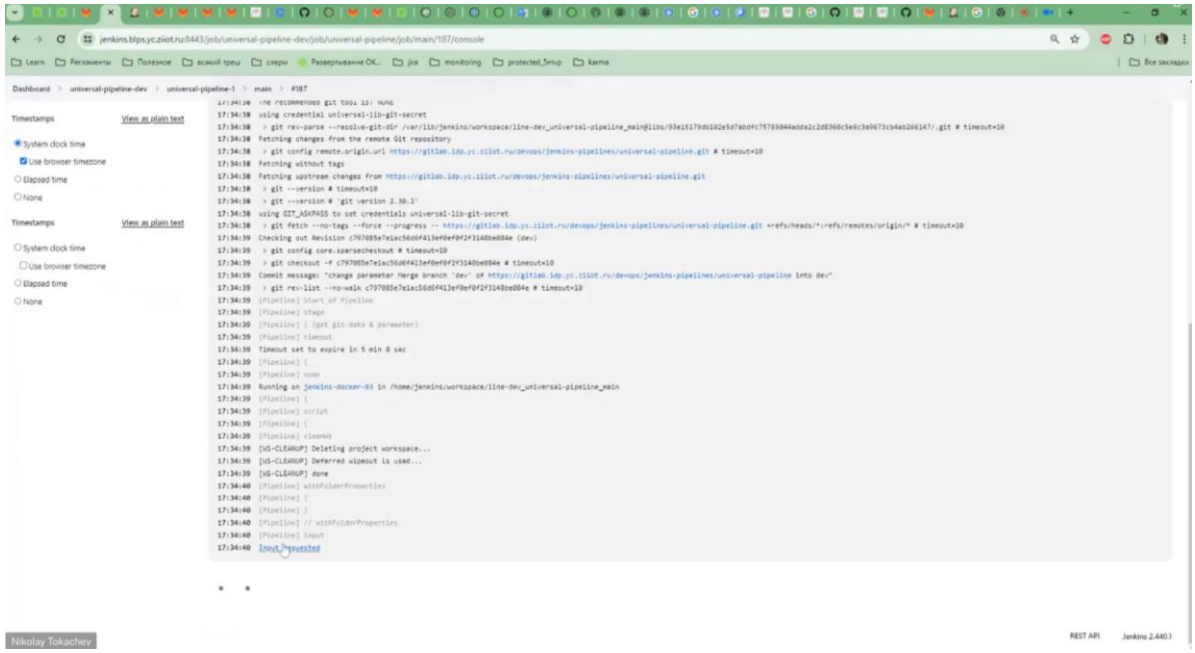

Рисунок 4.39 Выбор поля Input Requested

4. Введите адрес репозитория и токен на него:

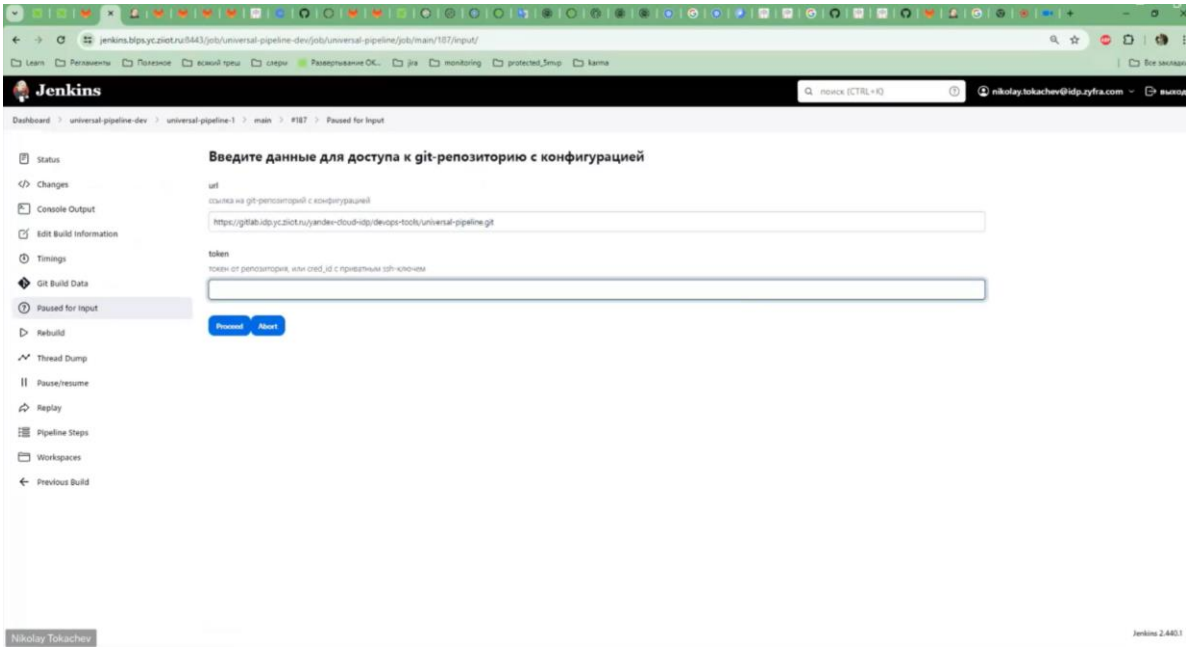

Рисунок 4.40 Ввод адреса репозитория

5. Следующая пауза **Input Requested** – нажмите на неё и выберите компонент для развертывания:

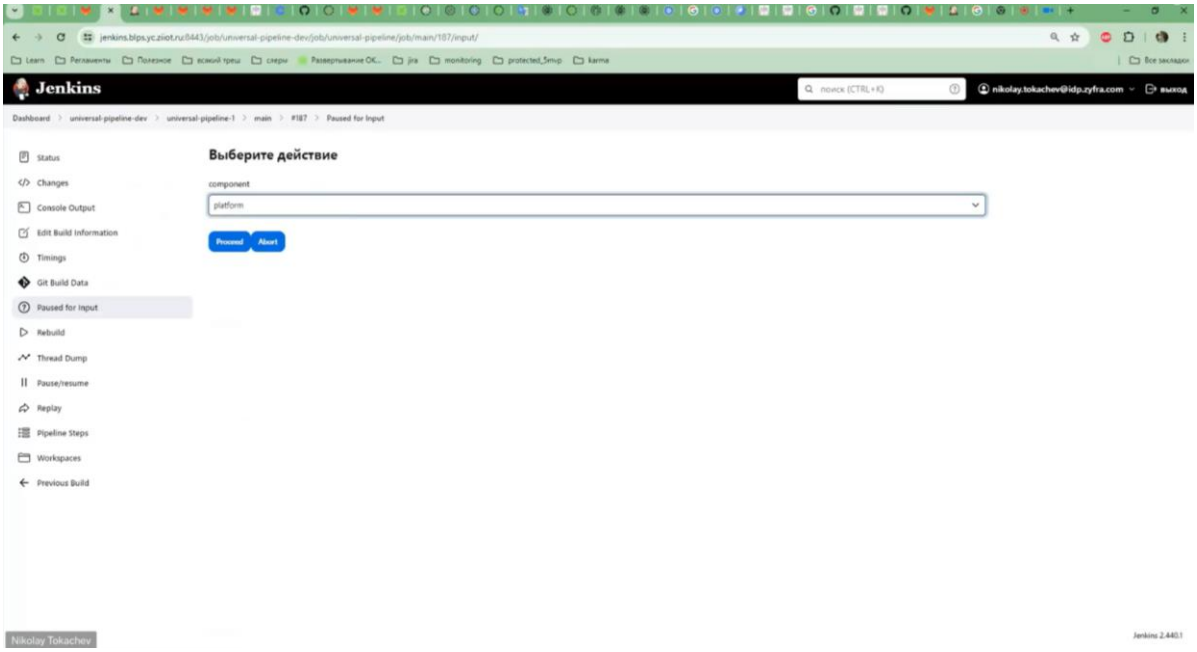

Рисунок 4.41 Выбор компонента развертывания

6. При следующей паузе **Input Requested** – выберите тип Платформы:

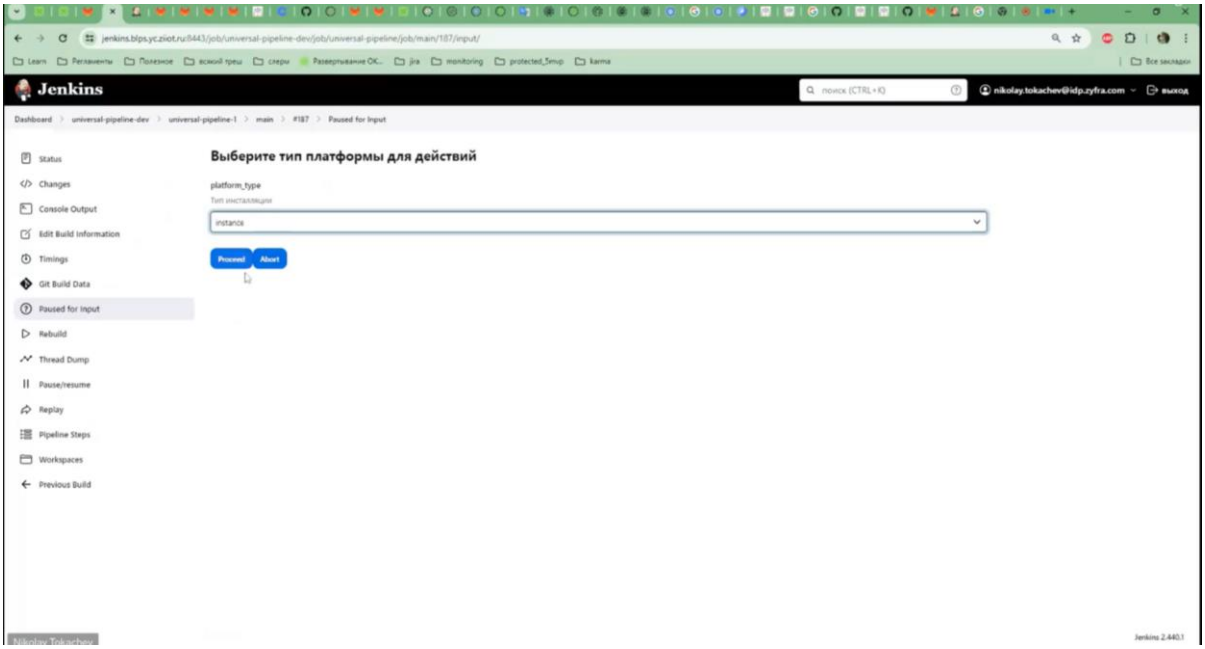

Рисунок 4.42 Выбор типа Платформы

7. Заполните завершающее окно параметров развертывания Платформы в соответствии с информацией из текущего раздела:

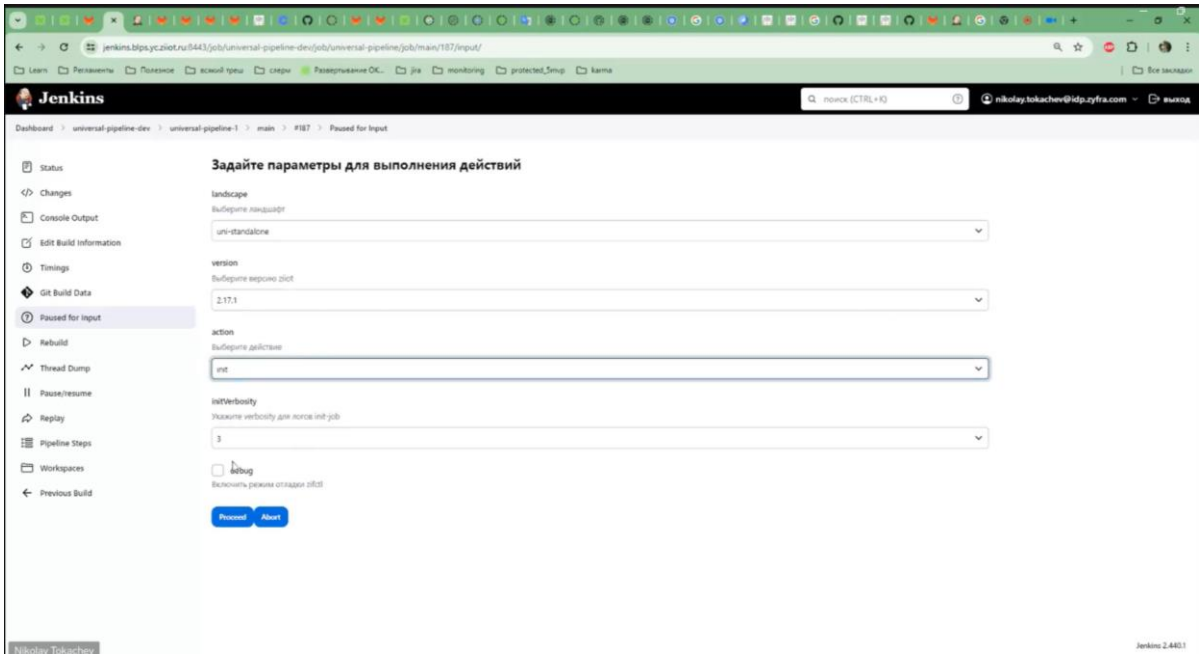

Рисунок 4.43 Заполнение параметров развертывания

8. Нажмите **Proceed**.