ЦИФРОВАЯ
ИНДУСТРИАЛЬНАЯ
ПЛАТФОРМА

Руководство для системы развертывания (zifctl) (2.20.0)

Zyfra Industrial Internet of Things Platform (ZIIoT)

Изменения в документе

Версия	Дата	Автор	Описание
1.0	14.08.2024	Пеплин Ф.Н.	Создание документа
1.1	24.10.2024	Маркова А.В.	Добавление раздела 4.11 Команда zifctl debug
1.2	12.11.2024	Кот О.В.	Редактирование раздела 4.11
1.3	20.11.2024	Кот О.В.	Добавлена инструкция по переходу на Cassandra 4

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 2.20.0

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 2.20.0 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 2.20.0

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.
monitoring.coreos.com/v1	-	Необязательный компонент Kubernetes, используется для создания объектов ServiceMonitor, который автоматически генерирует конфигурацию Prometheus scrape на основе текущего состояния объектов на сервере API. Требуется версия prometheus-operator >0.61.0

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 2.20.0

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 ▪ pgcrypto 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, 14
Apache Cassandra	БД временных рядов	3.11, 4.1
Redis	key-value БД (кэш)	6
Apache Kafka	Брокер сообщений	3.7
RabbitMQ	Брокер сообщений (используется в сервисе UDL).	3.12
Apache NiFi	Сервис управления потоками данных	1.19.1
KeyCloak	Сервис авторизации	17, 21
S3	API объектного хранилища	MiniO 2024.7.31

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

Релиз платформы версии 2.20.0 поставляется с инструментом по автоматическому развертыванию Платформы.

Версия инструмента (**zifctl**) аналогична версии платформы — 2.20.0. С помощью инструмента автоматического развертывания развертываются сервисы платформы, вошедшие в релиз 2.20.0, а также инфраструктурные и вспомогательные сервисы из таблицы ниже:

Таблица 2.4. Версии вспомогательных сервисов (zifctl) в ZIIoT 2.20.0

Компонент	Реализация	Назначение	Версия	Образ	Лицензия
PostgreSQL	Zyfra	PostgreSQL + утилиты для создания отказоустойчивого кластера	12.19	zif-stolon-pg12:1.2.0-v6-240801	The PostgreSQL Licence, Apache License 2.0
			14.12	zif-stolon-pg14:1.2.0-v6-240801	
Postgresql exporter	Bitnami + Zyfra	Мониторинг PostgreSQL	0.15.0	0.15.0-debian-12-r38	Apache License 2.0
pgAdmin4	pgAdmin	Графический клиент для PostgreSQL	8.10	zif-pgadmin4:8.10.0-v1-240801	The PostgreSQL Licence
Apache Cassandra	Bitnami + Zyfra	БД временных рядов	3.11.13	zif-cassandra:3.11.13-v22-240801	Apache License 2.0
			4.1.5	zif-cassandra4:4.1.5-v1-240801	
Apache Cassandra Exporter	Bitnami + Zyfra	мониторинг Cassandra	2.3.8	zif-cassandra-exporter:2.3.8-v21-240801	Apache License 2.0
Redis	Bitnami + Zyfra	key-value БД (кэш)	6.2.14	zif-redis:6.2.14-v11-240801	Apache License 2.0
Redis Exporter	Bitnami + Zyfra	Мониторинг Redis	1.62.0	zif-redis-exporter:1.62.0-v1-240801	Apache License 2.0
Apache Kafka	Confluent + Zyfra	Брокер сообщений	7.7.0	zif-kafka:7.7.0-v1-240801	Apache License 2.0
Kafka Exporter	Bitnami + Zyfra	Мониторинг Apache Kafka	1.7.0	zif-kafka-exporter:1.7.0-v18-240801	Apache License 2.0

Apache Zookeeper	Confluent + Zyfra	Распределенное хранение конфигураций Apache Kafka и Apache NiFi	7.7.0	zif-zookeeper:7.7.0-v1-240801	Apache License 2.0
Kafka Rest	Confluent + Zyfra	REST Api для Apache Kafka	7.7.0	zif-kafka-rest:7.7.0-v1-240801	Apache License 2.0
Kafka Schema Registry	Confluent + Zyfra	Схемы сообщений для топиков Apache Kafka	7.7.0	zif-schema-registry:7.7.0-v1-240801	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)	7.7.0	zif-kafka-connect:7.7.0-v1-240801	Apache License 2.0
Debezium Connector	Red Hat	Сбор, сериализация и отправка в Apache Kafka логов транзакций PostgreSQL	2.5.3	см. Kafka Connect	Apache License 2.0
JDBC Connector	Confluent	Сбор, сериализация и отправка данных в Apache Kafka (как Source) и РСУБД (как Sink)	10.7.6	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-v11-240802	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-v3-240801	Apache License 2.0
RabbitMQ	Bitnami + Zyfra	Брокер сообщений	3.12.14	zif-rabbitmq:3.12.14-v2-240801	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-v35-240801	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-v32-240802	Apache License 2.0
			21.1.2	zif-keycloak:21.1.2-v8-240801	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	7.6.0	zif-oauth2-proxy:7.6.0-v1-240801	MIT

Alpine	Docker Inc + Zyfra	Набор утилит	3.20.2	zif-alpine-util:3.20.2-v1-240801	GPLv2
MinIO	Bitnami + Zyfra	S3-совместимое объектное хранилище	2024.7.31	zif-minio:2024.7.31-v1-240801	Apache License 2.0
Opensearch	community + Zyfra	Масштабируемая утилита полнотекстового поиска и аналитики	1.3.18	zif-opensearch:1.3.18-v1-240801	Apache License 2.0
Fluentd	Bitnami + Zyfra	Система сбора, парсинга, перенаправления и агрегации логов	1.14.6	zif-fluentd-opensearch:1.14.6-v22-240801	Apache License 2.0
Opensearch Dashboards	community + Zyfra	UI для Opensearch	1.3.18	zif-opensearch-dashboards:1.3.18-v1-240801	Apache License 2.0

3. Специальные требования к процессу обновления

3.1. Переход на Cassandra 4

Начиная с версии платформы 2.20, по умолчанию будет устанавливаться 4 версия cassandra. При обновлении предусмотрена автоматическая миграция данных. Новая версия Cassandra значительно улучшает производительность и стабильность базы данных, а также предоставляет доступ к расширенным функциональным возможностям.

ВАЖНО! После перехода на версию 4 откат на версию 3 невозможен из-за изменений в структуре данных и механизмах работы самой базы данных. Это связано с тем, что Cassandra 4 использует новые внутренние форматы и алгоритмы, которые не совместимы с предыдущей версией.

3.1.1. Рекомендации

- Перед обновлением настоятельно рекомендуем **создать резервную копию** всех данных для предотвращения потерь в случае необходимости отката.
- Проверьте и протестируйте все критически важные функции приложения на версии 4 Cassandra перед развертыванием в продуктивной среде.

3.1.2. Настройки перехода

Переход осуществляется автоматически, однако, при необходимости, доступны варианты, описанные ниже.

Переход с увеличением количества реплик

С переходом на Cassandra 4 вы можете увеличить количество реплик в кластере, что повышает отказоустойчивость и доступность данных.

В файл **env-values.yaml** внести следующие изменения:

```
cassandra:
  installed: True
  nodes: 3
```

Переход с версии Cassandra без включенного TLS на версию с активированным TLS

Для перехода с Cassandra без включенного TLS на версию с активированным TLS (Transport Layer Security) необходимо внести следующие изменения в файл **env-values.yaml**:

```
cassandra:
  installed: True
  nodes: 3
  tls:
    enabled: True
```

3.1.3. Отказ от перехода

Возврат к версии 3

Возврат к предыдущей версии возможен только через восстановление из резервной копии (см. Рекомендации)

Отказ от обновления

Для отказа от перехода на 4 версию Cassandra, в конфигурационном файле **env-values.yaml** необходимо явно указать версию

```
cassandra:
  version: 3
  installed: True
```

3.2. Apache Nifi

Внимание! При переключении режима авторизации из oauth-proxy в native, в соответствии с требованиями ИБ, происходит шифрование репозиториев NiFi. При этом создаётся специальный keystore и прописываются параметры nifi.repository.encryption.* в nifi.properties

Шифруются файлы в поде zif-nifi-0 (также 1,2 если включен high availability), контейнер server, директории:

- /opt/nifi/nifi-current/provenance_repository
- /opt/nifi/nifi-current/content_repository

Данный процесс необратим! Переключение в автоматическом режиме с native в oauth-proxy не поддерживается в инсталляторе zifctl.

Возможно переключение NiFi с native в oauth-proxy вручную:

1. В nifi.properties вставить параметры, связанные с шифрованием репозиториев:

```
nifi.repository.encryption.protocol.version=1
  nifi.repository.encryption.key.id=nifi-repo-key
  nifi.repository.encryption.key.provider=KEYSTORE
```

```
   nifi.repository.encryption.key.provider.keystore.location=/opt/nifi/nifi-current/config-data/certs/repo_keystore.p12
   nifi.repository.encryption.key.provider.keystore.password=<пароль для keystore>
```

Пароль для keystore можно взять из секретов развёртывания nifi: keystorePassword: (можно взять из лога развёртывания или вывода команды zifctl secrets show)

2. Взять, созданный при native режиме keystore, путь к файлу /opt/nifi/nifi-current/config-data/certs/repo_keystore.p12:

Скопировать его в файл /opt/nifi/nifi-current/config-data/certs/repo_keystore.p12 или примонтировать его в виде secret к поду zif-nifi-0 (1,2 при HA) в контейнер server по тому же пути.

3. Перезапустить NiFi

3.3. Доступ сервисов к нескольким БД в рамках одного модуля

В рамках zifctl реализована возможность получения доступа сервисом, даже не имеющего БД, к БД соседних сервисов в рамках одного модуля установки

Для получения доступа необходимо до установки модуля разместить файл **service-list-patch.yaml** в директории services.

Рассмотрим пример.

Условия:

1. Сервис которому необходимо предоставить доступ - test-srv1
2. Сервис test-srv1 не имеет собственной БД
3. Пользователь для сервиса test-srv1 должен создать инсталлятор zifctl
4. Сервис к БД которого необходимо получить доступ test-srv2
5. Сервис к БД с нестандартным именем (`not_standart__db_name`) необходимо получить доступ

Пример файла service-list-path.yaml

```
modules:
- name: test
  services:
   - name: test-srv1
     linkDb:
      - srvName: test-srv2
      - dbName: not_standart__db_name
        postgres: true
        postgresCreateDb: false
```

Примеры из практики

Вариант 1

Контекст:

У сервиса А есть своя БД и соответственно пользователь А.

Сервису А необходимо иметь доступ в базу сервиса Б под пользователем А.

Решение:

В файле `service-list-patch.yaml` добавляем необходимые строчки:

```
postgres: true
linkDb:
- srvName: zmeb-data-cache
```

В итоге блок будет выглядеть так:

```
- name: zmeb-user-activity #(Сервис А)
  image: zmeb-user-activity
  imageNamespace: zmeb
  tag: 4.11.1
  auth: confidential
  ingress: true
  ingressPath: "/zmeb-user-activity"
  ingressUrlRewrite: false
  type: netcore
  postgres: true
  linkDb:
   - srvName: zmeb-data-cache  #(Сервис Б)
     abac:
     enabled: false
     startupCheck: false
```

В **gotmpl** шаблон файла сервиса А `zmeb-user-activity.yaml.gotmpl` вставляем необходимые переменные:

```
environmentVars:
  POSTGRES_USER_DATA_CACHE:
    valueFrom:
      secretKeyRef:
        name:  "{{ .Values.ZifModule.chart }}-secrets"
        key: "zmeb-user-activity-psql-user"
  POSTGRES_PASSWORD_DATA_CACHE:
```

```
    valueFrom:
      secretKeyRef:
        name:  "{{ .Values.ZifModule.chart }}-secrets"
        key: "zmeb-user-activity-psql-pass"
  POSTGRES_DATABASE_DATA_CACHE:
    valueFrom:
      secretKeyRef:
        name:  "{{ .Values.ZifModule.chart }}-secrets"
        key: "zmeb-data-cache-psql-db"
  POSTGRES_HOST_DATA_CACHE: {{ .Values.postgres.host }}
  POSTGRES_PORT_DATA_CACHE: {{ .Values.postgres.port }}
```

Результат:

Сервис А может ходить в базу сервиса Б, потому что у пользователя БД сервиса А есть права на базу сервиса Б.

Вариант 2

Контекст:

У сервиса А нет своей БД и соответственно пользователя А.

Сервису А необходимо иметь доступ в базу сервиса Б и создать пользователя А.

Решение: В файле service-list-patch.yaml добавляем необходимые строчки:

```
...
postgres: true
postgresCreateDb: false
linkDb:
   - srvName: zmeb-data-cache
...
```

В итоге блок будет выглядеть так:

```
- name: zmeb-datacollector #(Сервис А)
  image: zmeb-datacollector
  imageNamespace: zmeb
  tag: 4.11.1
  auth: confidential
  ingress: true
  ingressPath: "/zmeb-datacollector"
  ingressUrlRewrite: false
  type: netcore
  postgres: true
  postgresCreateDb: false
  linkDb:
   - srvName: zmeb-data-cache #(Сервис Б)
```

```
    abac:
    enabled: false
    startupCheck: false
```

В **gotmpl** шаблон файла сервиса А `zmeb-user-activity.yaml.gotmpl` вставляем необходимые переменные:

```
environmentVars:
  POSTGRES_USER_DATA_CACHE:
    valueFrom:
      secretKeyRef:
        name:  "{{ .Values.ZifModule.chart }}-secrets"
        key: "zmeb-datacollector-psql-user"
  POSTGRES_PASSWORD_DATA_CACHE:
    valueFrom:
      secretKeyRef:
        name:  "{{ .Values.ZifModule.chart }}-secrets"
        key: "zmeb-datacollector-pass"
  POSTGRES_DATABASE_DATA_CACHE:
    valueFrom:
      secretKeyRef:
        name:  "{{ .Values.ZifModule.chart }}-secrets"
        key: "zmeb-data-cache-psql-db"
  POSTGRES_HOST_DATA_CACHE: {{ .Values.postgres.host }}
  POSTGRES_PORT_DATA_CACHE: {{ .Values.postgres.port }}
```

Результат:

Создан пользователь А, без базы для сервиса А. Пользователь А может ходить в базу сервиса Б, потому что у пользователя БД сервиса А есть права на базу сервиса Б.

3.4. Плагин логирования в формате CEF в Keycloak v21

В **Keycloak v21**, поставляемый в рамках авторазвертывания, добавлен плагин для логирования в формате **CEF**. Логирование в этом формате активируется при установке параметра **logging.cef**: **True** в файле конфигурации авторазвертывания **env-values.yaml**:

```
logging:
  cef: True
```

При установке данного параметра в **Keycloak** будет отдавать логи уровня **INFO** с событиями **CEF**:

```
{"timestamp":"2024-06-
19T08:14:14.35Z","sequence":8438,"loggerClassName":"org.slf4j.impl.Slf4jLogger","logge
```

```
rName":"com.zyfra.idp.eventlistenerprovider.ZIIOTEventListenerProvider","level":"INFO"
,"message":"CEF:0|ZYFRA|ZIIoT|2.20.0|Security Admin Audit Event|CREATE
CLIENT_SCOPE|5|src=127.0.0.1 dst=<hostname> shost= suid=db4ebd54-a14d-47b0-b12d-
1e8b855def7b suser=<user>
msg=realmId:<realmId>,realmName:<realmName>,clientId:69a11568-0195-4e1d-bdf7-
810eb1a54c3a,resourcePath:client-scopes/ba782c60-4a9e-488f-b64c-c0e7c3eee80b
end=1718784854347","threadName":"executor-thread-
0","threadId":16,"mdc":{"CefMessage":"CEF:0|ZYFRA|ZIIoT|2.20.0|Security Admin Audit
Event|CREATE CLIENT_SCOPE|5|src=127.0.0.1 dst=<hostname>> shost= suid=db4ebd54-a14d-
47b0-b12d-1e8b855def7b suser=<user>
msg=<realmId>:ziiot,realmName:<realmName>,clientId:69a11568-0195-4e1d-bdf7-
810eb1a54c3a,resourcePath:client-scopes/ba782c60-4a9e-488f-b64c-c0e7c3eee80b
end=1718784854347"},"ndc":"","hostName":"zif-keycloak21-
0","processName":"QuarkusEntryPoint","processId":1}
```


Внимание! Сбор логов в формате **CEF** необходимо настроить на поле **mdc.CefMessage**.

3.5. Настройки OpenTelemetry

3.5.1. Описание

OpenTelemetry – новый стандарт сбора телеметрии приложений. Он объединил стандарты OpenTracing (используется в ZIIoT) и OpenCensus.

В рамках OpenTelemetry телеметрия приложений складывается из сигналов, их состав в будущем может быть расширен, но сейчас это:

- **Метрики** - числовые измерения ключевых характеристик, сформированные во время выполнения приложения.
- **Трейс** - распределенный ациклический граф выполняемой работы, который состоит из отдельных диапазонов - Span. Span (в .NET класс System.Diagnostics.Activity) и описывает локальную единицу работы приложения и может расширяться атрибутами, событиями (лог, детализирующий, что произошло в рамках выполняемой единицы работы) и вложенными единицами работы.
- **Лог** - это текстовая запись с меткой времени, лог являются независимым источником данных телеметрии, но в рамках концепции OpenTelemetry рекомендуется сопоставлять их с контекстом выполняемой работы, т.е. лог рекомендуется прикреплять к Span в виде событий.
- **Багаж** (baggage) - описывает контекстную информацию связанную с активностью (span), которая будет передаваться во все вложенные активности

Кроме концептуального подхода, OpenTelemetry предоставляет:

- Инструментарии для сбора телеметрии – представляет собой библиотеки для сервисов под каждый технологический стек (их развитие определяет сообщество и может отличаться по возможностям, в этом документе мы рассматриваем только .NET)
- Экспортёры – библиотеки, которые позволяют экспортировать собранную телеметрия в указанный источник в формате источника (нас интересует экспорт в консоль, Jaeger, Propmetheus (pull режим через /metrics конечную точку)
- SDK позволяет:

    - расшить перечень процессоров, обрезывающих собранную телеметрия до ее представления в экспортере.
    - расширять перечень экспортеров собственными реализациями)
    - расширить возможности интеграции для распространения контекстов трассировки (context propagation) - что позволяет передавать\получать контекст из различных источников (http headers, rabbitmq message, kafka message и т.д.) и различных форматах (b3(zipkin)\uber (jaeger) header и т.д)
- Спецификации - набор рекомендаций по формированию телеметрии
- Протокол передачи телеметрии OTLP
- OpenTelemetry Collector - сервис, который позволяет в унифицированном формате собирать телеметрию с сервисов, обогащать ее и адаптировать собранную телеметрию под потребности конечных приемников телеметрии

3.5.2. Настройки

В рамках поддержки будущего перехода на сбор телеметрии приложений с использованием **OpenTelemetry**, в модель конфигурации авторазвертывания добавлены новые параметры:

```
tracing:
  samplerRatio: 1.0
  otlpPort: 4317
```

В сервисы передаются следующие переменные окружения (при условии **jaeger.tracing_enabled** или **jaeger.enabled**):

- **OTEL_TRACING_ENABLED** - регулируется параметром **jaeger.tracing_enabled** или **jaeger.enabled**, значение по умолчанию **False**.
- **OTEL_TRACING_SAMPLER_RATIO** - регулировка сэмплирования, значение по умолчанию 1.0 (т.е. все трейсы отправляются в коллектор).
- **OTEL_OTLP_COLLECTOR_URL** - **jaeger.host:tracing.otlpPort**.

3.6. Миграция PostgreSQL (Stolon) на новую версию

3.6.1. Система миграции

Система миграции вынесена в отдельную команду **zifctl postgres** и не является частью команды **zifctl sync**, соответственно миграция не выполняется автоматически с командой **zifctl sync.**

При выполнении миграции будет создан новый **PVC** для данных нового кластера **Stolon**, куда будут скопированы данные для новой версии СУБД. В процессе работы мигратора, будут остановлены все зависимые от БД сервисы, а также **stolon-keeper**.

Для управление всеми этапами миграции используется только команда **zifctl postgres**, которая содержит список субкоманд:

```
$ zifctl postgres –help

usage: zifctl postgres [SUB-COMMAND] [OPTIONS] [--help]
```

```
Migrating PostgreSQL servers to new versions running inside Kubernetes cluster

required arguments:

  Specify this options or declare corresponding environment variables

common options:

  -v, --verbose       Enable verbose command output

  -vv, --debug        Enable verbose command output and helmfile/helm debug

                      output

  -h, --help          Show help message and exit

available subcommands (and aliases):

  Sub-command         Sub-command description

    migration-cancel   Stop running migration

    migration-clean    Clean all migration objects, include PVCs

    migration-get-logs   Save logs to disk

    migration-start   Upgrade PostgreSQL databases to a new version
```

Подготовка к миграции

Миграция состоит из нескольких этапов:

1. Проверить параметр **postgres.version** в конфигурации окружения. В нем должна быть указана текущая версия **PostgreSQL**. Также удостовериться в достаточном размере тома для данных **PostgreSQL**, т.к. новый том будет создан с тем же размером, который указан в конфигурации окружения.

```
postgres:
  storageSizeMB:       <указать       необходимое       значение       в       МБ>
  version: stolon-12
```

2. Удостовериться, что все сервисы в текущем тенанте успешно запущены и работают в штатном режиме:

```
# можно проверить командой

 $ kubecl get pods --field-selector "status.phase!=Succeeded" | awk 'match($0,"([0-
9]+)/([0-9]+)|STATUS",r) {if (r[1]!=r[2] || $3 ~ /Terminating|STATUS/) print}'
```

3. Запуск миграции командой **zifctl postgres migration-start**. Также рекомендуется использовать отдельный **PVC** для сохранения логов миграции (в случае длительной миграции размер логов, выводимых в **stdout**, может превысить лимит, что приведет их ротации и потере части логов).

Чтобы сохранять логи в персистентном хранилище (по умолчанию не используется) можно указать аргумент **--logs-pvc-size <size>,** где **size** может быть указан в мегабайтах или гигабайтах, например: **500Mi** или **1Gi**. Данные на диске хранятся в не архивированном виде.

Миграция PostgreSQL на версию 14

```
# single-tenant
$ zifctl postgres migration-start --env-path /path-to-config/env-config --postgres-version stolon-14
# multi-tenant
# первым запускаем tenant
$ zifctl postgres migration-start --env-path /path-to-tenant-config/env-config --postgres-version stolon-14
# вторым запускаем shared-infra
$ zifctl postgres migration-start --env-path /path-to-infra-config/env-config --postgres-version stolon-14
```

1. Будут остановлены все сервисы платформы, использующие БД, включая **Keycloak**, Pgadmin и сам **Stolon-Keeper** (**PostgreSQL**).
2. Создание PVC для данных нового экземпляра **PostgreSQL**. Создается только один PVC, даже если в конфигурации окружения указано **infraHA: true**. Создание оставшихся PVC будет выполнено в рамках деплоя нового чарта **Stolon2**.
3. Запуск **Kubernetes Job** в том же namespace, где запущен **Stolon-keeper**. К поду подключаются PVC со старыми и новыми данными, а также **PVC** для логов, если задан аргумент **--logs-pvc-size**. После запуска **job**, можно прервать выполнение команды, сама **job** продолжит выполняться в фоновом режиме. Подключиться к просмотру логов текущей **job** возможно, запустив повторно команду **zifctl postgres migration-start ...**. Остановка миграции возможна только субкомандой **migration-cancel**.

В процессе миграции будут возникать ошибки. Данные ошибки можно игнорировать:

```
```log
2023-05-22 14:39:01,697 INFO kube: ERROR: type "opaque" does not exist
2023-05-22 14:39:02,361 INFO kube: ERROR: function pg_catalog.pg_create_logical_replication_slot(name, name, boolean) does not exist
2023-05-22 14:39:03,225 INFO kube: ERROR: function pg_catalog.pg_terminate_backend(integer) does not exist
2023-05-22 14:39:03,720 INFO kube: ERROR: function pg_catalog.set_bit(bytea, integer, integer) does not exist
...
2023-05-22 14:39:04,456 INFO kube: ERROR: function pg_catalog.utf8_to_johab(integer, integer, cstring, internal, integer) does not exist
2023-05-22 14:39:04,457 INFO kube: ERROR: function pg_catalog.utf8_to_koi8r(integer, integer, cstring, internal, integer) does not exist
```
```

```
2023-05-22 14:39:04,458 INFO   kube: ERROR:  function pg_catalog.utf8_to_koi8u(integer,
integer,       cstring,       internal,       integer)       does       not       exist
2023-05-22    14:39:04,459    INFO                  kube:    ERROR:         function
pg_catalog.utf8_to_shift_jis_2004(integer, integer, cstring, internal, integer) does
not                                                                          exist
2023-05-22 14:39:04,460 INFO   kube: ERROR:  function pg_catalog.utf8_to_sjis(integer,
integer,       cstring,       internal,       integer)       does       not       exist
2023-05-22 14:39:04,462 INFO    kube: ERROR:  function pg_catalog.utf8_to_uhc(integer,
integer,       cstring,       internal,       integer)       does       not       exist
2023-05-22 14:39:04,464 INFO    kube: ERROR:  function pg_catalog.utf8_to_win(integer,
integer,       cstring,       internal,       integer)       does       not       exist
```

4. После успешного завершения миграции, в последней строке логов будет соответствующее сообщение **postgres: Job return status: Succeeded**, необходимо в конфигурации окружения заменить параметр **postgres.version** на **stolon-14**.
5. Выполнить команду **zifctl sync**.

Внимание! В случае мультитенантности синхронизация должна быть выполнена в определенном порядке: первой выполняется синхронизация **shared-infra**, вторым **tenant**.

Во время синхронизации будут созданы недостающие PVC, если в конфигурации окружения указано **infraHA**: **true**, и будет запущена репликация данных на новых инстансах **stolon-keeper**.

Получение логов миграции

В случае, если процесс миграции завершился со статусом **Failed** и для решения проблемы потребуются логи, то можно получить логи следующими способами:

```
$ kubectl logs -l job-name=pg-upgrade  pg-upgrade.log
```

Если при старте миграции был указан аргумент **--logs-pvc-size**, тогда возможно получить файл с логами командой:

```
$ zifctl postgres migration-get-logs --env-path /path-to-config/env-config --output-
path . /var/deploy/output/pg-upgrade.log
```

Остановка миграции

Команда останавливает текущую миграцию. Повторный запуск миграции, после ее остановки, приведет к запуску миграции с самого начала. Данные нового **PostgreSQl** будут очищены.

```
$ zifctl postgres migration-cancel --env-path /path-to-config/env-config
```

Очистка данных после не успешной миграции

Удалить данные (**job**, **PVC** с логами), созданные при запуске миграции, можно командой:

```
$ zifctl postgres migration-clean --env-path /path-to-config/env-config
```

Если указать аргумент **--force**, также будет удален **PVC** с данными нового **PostgreSQL**, созданный в результате миграции.

4. Развертывание Платформы в виде docker-образа

Внимание! Данный раздел применим только для Релиза 2.9.0 и более поздних.

Система развертывания **Платформы** (обычно называемая **zifctl** по названию скрипта запуска) до релиза 2.9.0 поставлялась в виде исходного кода, как **git**-репозиторий. Для установки требовалось скачать репозиторий и запустить из него скрипт **zifctl** с необходимыми параметрами.

Начиная с релиза 2.9.0, инструмент развертывания **Платформы** поставляется в виде готового **docker**-образа, тег которого совпадает с версией устанавливаемой версии **Платформы** (плюс, возможно, суффикс-идентификатор сборки).

Это дает следующие преимущества:

1) Нет необходимости иметь доступ к **git**-репозиторию на целевой площадке или приносить его туда в архиве. Достаточно только наличия **docker**-образа в локальном **registry** (образы для Платформы все равно туда переносить или зеркалировать).
2) Нет необходимости в скачивании и переносе **helm-chart** для развертывания Платформы. Необходимые версии чартов положены в образ на этапе сборки.

Из недостатков можно отметить более сложную схему запуска инструмента т.к. приходится пользоваться **docker**-контейнером, в который необходимо передавать пути с локальной машины. Для облегчения этой задачи разработан специальный скрипт-обертка.

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**).

4.1. 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. 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
```

4.3. 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).

4.4. 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).

4.5. 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-XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

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-XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX"
```

4.6. 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**).

4.7. 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..."
```

4.8. 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.

4.9. 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
```

5. Возможности инсталлятора zifctl

При работе с командами используется набор подкоманд и обязательны аргументов. Если аргумент не задается явно, то система пытается взять значение из стандартных переменных окружения в соответствии с таблицей:

Подкоманда	Переменная окружения	Описание
--env-path	$ZIF_ENV_PATH	Абсолютный или относительный путь к директории с конфигурацией окружения
--age-key	$ZIF_AGE_KEY	Сгенерированный утилитой Age приватный ключ, который используется для шифрования/дешифрования секретов
--output-path	$ZIF_OUTPUT_PATH	Директория, куда будут сохраняться сгенерированные командами zifctl doc env-values, zifctl template и т.п. файлы
--server	$ZIF_SERVER	адрес API кластера Kubernetes/OpenShift
--token	$ZIF_TOKEN	токен сервис-аккаунта для доступа к API кластера Kubernetes/OpenShift
--log-path	$ZIF_LOG_PATH	Путь к директории с логами
--version	$ZIFCTL_VERSION	Версия Ziiot, которая будет устанавливаться (явно задавать не надо, по дефолту соответствует версии образа zifctl, например, 2.20.0)
--zifctl-image	$ZIFCTL_IMAGE	Используемый образ zifctl (явно задавать не надо, дефолтное значение: --zifctl-registry)
--zifctl-registry	$ZIFCTL_REGISTRY	Репозиторий, где хранится образ zifctl (нет неоходимости использовать, если образ уже в наличии. Дефолтное значение: docker.idp.yc.ziiot.ru)

В случае, если система не находит значения ни в аргументе, ни в переменной окружения, выдается ошибка.

5.1. 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):

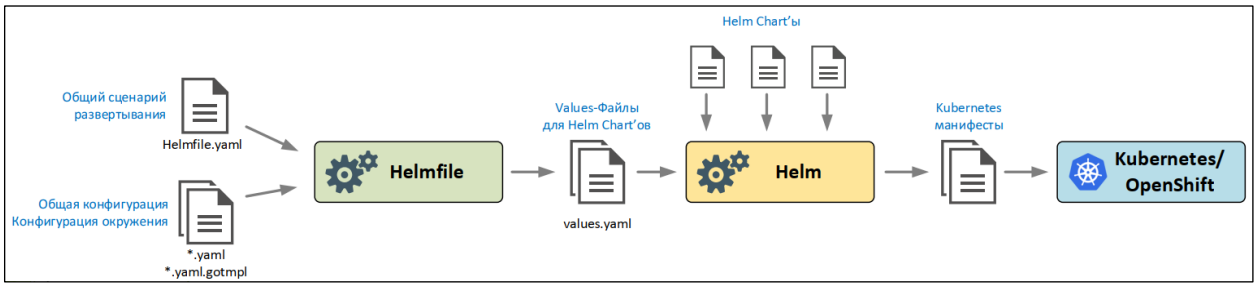

Рисунок 5.1 Отладочные команды

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. Можно сгенерировать манифесты для всей установки или для модуля
./zifctl template --env-path /path/to/env/config --age-key $KEY --output-path $OUTPUT_PATH [--module rtdb ]
```

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).

5.2. Команда zifctl debug

При выполнении команды zifctl debug доступен следующий набор подкоманд:

- restart-init-task
- show-defaults
- show-config
- show-tenant-id
- scale-down-platform

Подробные описания подкоманд даны ниже.

5.2.1. restart-init-task

Перезапуск задачи **init-job** в случае возникновения проблем или зависания.

Важно:

Перезапуск возможен только для модулей из файла **service-list.yaml** и в статусе **enabled = True.** Перезапустить **init-job** инфраструктуры или отключенные модули данной командой нельзя!

Команда:

./**zifctl debug restart-init-task**

Обязательные ключи:

--env_path

--module

--server

--token

Рассмотрим для примера перезапуск задачи **init-job** для модуля **rtdb.**

Результат выполнения команды можно увидеть в логе zifctl. В частности, лог содержит информацию о том, что предыдущая init-job удалена.

```
~$ ./zifctl debug restart-init-task \
    --env-path /home/user/env_config/ \
    --module rtdb \
    --server https://12.34.56.78 \
    --token $MY_TOKEN

Status: Image is up to date for docker.idp.yc.ziiot.ru/infra/zifctl:feature-
platform01-44787
 2024-09-04 21:14:43,590 INFO    debug: Environment path: /var/deploy/env
 2024-09-04 21:4:43,590 INFO    debug: Working namespace: dev-develop
 2024-09-04 21:14:43,590 INFO    debug: Module name: rtdb
 2024-09-04 21:14:43,590 INFO    debug: Job name: zif-rtdb-module-init
 2024-09-04 21:14:43,590 INFO    debug: Ansible log verbosity: 2
 2024-09-04 21:14:43,673 INFO    debug: Found existed job: zif-rtdb-module-
init in namespace dev-develop
 2024-09-04 21:14:43,734 INFO    debug: Wait for job deletion...
```

```
 2024-09-04 21:14:45,761 INFO    debug: Job was deleted: zif-rtdb-module-init
 2024-09-04 21:14:45,772 INFO    debug: Render template init-job.yaml
 2024-09-04 21:14:45,816 INFO    debug: Create new job: zif-rtdb-module-init
```

5.2.1. show-defaults

Вывод значений по умолчанию для параметров развертывания. Сортировка списка в алфавитном порядке.

Для переопределения значений параметров, необходимо внести их в файл **env-values.yaml**.

Команда:

./zifctl debug show-defaults

У команды нет обязательных ключей.
Пример результата работы подкоманды приведен ниже. Он сокращен, так как является очень объемным.

```
~$ ./zifctl debug show-defaults

abacAuthorization:
  enabled: false
  importPolicies: true
  pip:
    keycloak:
      enabled: false
      rabbitmq:
        enabled: true
        loginEvents: false
    om:
      enabled: false
      kafka_enabled: false
appName: test_app
cassandra:
  auth: true
  commitLogSegmentSizeMB: 32
  commitLogSizeMB: 256
  contactPoints: ''
  defaultServiceName: zif-cassandra-headless
  installed: false
  metricsEnabled: false
  nodes: 1
  port: 9042
  replicationFactor: 1
  storageSizeMB: 1024
  tls:
    enabled: false
    force: false
    hostname: ''
    internode: all
    keystoreFile: /bitnami/cassandra/certs/stores/keystore.jks
    truststoreFile: /bitnami/cassandra/certs/stores/truststore.jks
  version: 3

------------Часть вывода удалена-----------------

fileStorage:
```

```
  anonymousUsername: filestorage_anonymous_user
  basePath: /files
healthcheck:
  livenessFailureThreshold: 3
  livenessInitialDelaySeconds: 60
  livenessPath: /health/liveness
  livenessPeriodSeconds: 10
  livenessSuccessThreshold: 1
  livenessTimeoutSeconds: 10
  readinessFailureThreshold: 3
  readinessInitialDelaySeconds: 60
  readinessPath: /health/readiness
  readinessPeriodSeconds: 10
  readinessSuccessThreshold: 1
  readinessTimeoutSeconds: 10

------------Часть вывода удалена------------------

init:
  ansibleCallbackPlugin: default
  ansibleHideKeycloakSecrets: true
  ansibleLogVerbosity: 2
  disabledTasks: []
  fullInitTaskList:
  - postgres
  - postgresGrant
  - postgresOwnership
  - keycloak
  - cassandra
  - redis
  - rabbitmq
  - debezium
  - portalSettings
  - rest
  - objectModel
  - custom
  - logging
  - s3storage
  - authorization
  - kafka
  - nifiMetrics
  longRetries: 60
  longRetriesDelay: 10
  shortRetries: 3
  shortRetriesDelay: 5
  tasks:
  - all

------------Часть вывода удалена------------------
```

5.2.2. show-config

Вывод текущей конфигурации Платформы (содержимое файла env-values.yaml). Сортировка по алфавиту.

Команда*:*

./zifctl debug show-config

Обязательные ключи:

--env-path

Пример:

```
~$ ./zifctl debug show-config --env-path /home/path_to/env_config/
```

Вывод команды похож на предыдущий и также является очень объемным.

5.2.3. show-tenant-id

Вывод ID тенанта из текущей конфигурации окружения и из развернутого экземпляра платформы.

Команда:

./zifctl debug show-tenant-id

Ключи:

--env-path - обязательный

--server

--token

В случае использования единственного ключа env_path, результатом выполнения команды будет вывод значения параметра tenantId из файла env-values.yaml

Пример выполнения команды с ключом env_path:

```
~$ ./zifctl debug show-tenant-id  --env-path ~/dev-develop-2.19-default/

Show ID of the tenant for configuration:
  env-values.yaml:       ziiot
```

При использовании ключей server и token (или установке значений переменных окружения ZIF_SERVER и ZIF_TOKEN), выводится значение параметра TENANT_ID из конфигурации zif-global-config. При этом идет обращение к значению параметра namespace и используются аргументы ключей server и token.

Пример выполнения команды с ключами server и token

```
~$ ./zifctl debug show-tenant-id  --env-path /home/path_to/env_config/   --
server https://12.34.56.78 --token $MY_TOKEN

Show ID of the tenant for configuration:
  env-values.yaml:       ziiot
  zif-global-config:     ziiot             from installation in k8s
```

5.2.4. scale-down-platform

Отключение стенда без удаления каких-либо сущностей: обнуляет количество подов во всех развертываниях (deployment) и наборах состояний (statefulset).

Примечание. Стенд включается командой **zifctl sync**.

Команда*:*

./zifctl debug scale-down-platform

Обязательные ключи:

--env-path

--server

--token

Пример результата работы команды. Сокращен.

```
~$ ./zifctl debug scale-down-platform  --env-path /home/path_to/env_config/ --
server https://12.34.56.78 --token $MY_TOKEN

 2024-09-04 21:41:20,928 INFO    kube: deployment.apps/zif-cm-context-functions
scaled
 2024-09-04 21:41:21,039 INFO    kube: deployment.apps/zif-cm-engine-mvel
scaled
 2024-09-04 21:41:21,173 INFO    kube: deployment.apps/zif-cm-metadata scaled
 2024-09-04 21:41:21,301 INFO    kube: deployment.apps/zif-dashboard scaled
 2024-09-04 21:41:21,415 INFO    kube: deployment.apps/zif-data-emulator scaled
 2024-09-04 21:41:21,545 INFO    kube: deployment.apps/zif-datainput scaled
 2024-09-04 21:41:21,672 INFO    kube: deployment.apps/zif-datalink scaled
 2024-09-04 21:41:21,804 INFO    kube: deployment.apps/zif-datalink-scripts
scaled
 2024-09-04 21:41:21,909 INFO    kube: deployment.apps/zif-datalink-xl scaled
 2024-09-04 21:41:22,053 INFO    kube: deployment.apps/zif-docs scaled
 2024-09-04 21:41:22,167 INFO    kube: deployment.apps/zif-document-archive
scaled
 2024-09-04 21:41:22,330 INFO    kube: deployment.apps/zif-events scaled
 2024-09-04 21:41:22,438 INFO    kube: deployment.apps/zif-events-integration
scaled
 2024-09-04 21:41:22,564 INFO    kube: deployment.apps/zif-export-nifi-
collectors scaled
 2024-09-04 21:41:22,721 INFO    kube: deployment.apps/zif-file-storage-default
scaled
 2024-09-04 21:41:22,835 INFO    kube: deployment.apps/zif-hierarchies scaled
 2024-09-04 21:41:22,932 INFO    kube: deployment.apps/zif-interface-connect
scaled
 2024-09-04 21:41:23,059 INFO    kube: deployment.apps/zif-interface-manager
scaled

------------Часть вывода удалена-------------------
```

5.3. 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
```

6. 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`.

7. 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`.

8. Изменения Apache Keycloak, начиная с релиза Платформы 2.17.0

Обновление текущей версии 17, входящей в состав Платформы на версию 21.

Обновление не должно повлиять или исказить/повредить данные как самой Платформы, так и информацию о текущих пользователях.

Изменения:

1) `baseUri` больше не будет использоваться, поэтому был удалён из модели конфига `keycloak`.
2) `KEYCLOAK_URL` не используется и была удалена.
3) `Helm chart keycloak` в будущем будет удален так как будет использоваться `helm`-чарт от `Bitnami`.
4) Добавлен новый класс `version` для указания версии `keycloak`, которую необходимо развернуть.
5) Изменена логика установки значения `default_servicename` — для версии 17 имя сервиса было с добавлением `-http`.
6) Добавлен новый файл с переменными `zif-keycloak21.yaml.gotmpl` для развертывания нового чарта `keycloak`.
7) Внесены изменения в чарте `zif-infra-ingress` чтобы при изменении версии ссылка `/auth` ссылалась на верный `keycloak` сервис (`zif-keycloak-http` или `zif-keycloak21`).
8) Внесены изменения в главный `helmfile` чтобы при установке выбиралась версия, указанная в конфигурационном файле `env-values.yaml`.
9) В базовый образ добавлен `keycloak-config-cli-21.0.1` для работы с серверной версией 21 `keycloak`.
10) Образ `zif-keycloak21` содержит темы `zif`. Темы необходимо было переработать для работы с новой версией.
11) Образ `zif-keycloak21` подготовлен для работы в `ОКД` без использования `security context`.

Необходимые действия для обновления:

Внимание! Необходимо удалить `baseUri` из блока `keycloak` в конфигурационном файле `env-values.yaml`:

1) Откройте файл `env-values.yaml` на редактирование и удалите из блока `keycloak` строку:

```
baseUri: '[http://zif-keycloak-http.dev-guseynov/auth'](#описание-задачи)**
```

2) Укажите версию 21, если хотите обновить текущий `keycloak`:

```
version: 21
```

Если версия не указана, то по умолчанию будет взята версия 21.

Файл до обновления:

```
env-config.yaml
## Конфигурация встроенного keycloak
keycloak:
  installed: True
  baseUri: 'http://zif-keycloak-http.dev-guseynov/auth'
  baseExtUri: 'https://dev-guseynov.kube07.yc.ziiot.ru/auth'
```

Файл после обновления:

```
env-config.yaml
## Конфигурация встроенного keycloak
keycloak:
  installed: True
  baseExtUri: 'https://dev-guseynov.kube07.yc.ziiot.ru/auth'
  version: 21
```

9. Поддержка встроенной (native) OpenID Connect аутентификации для Apache NiFi

9.1. Два типа аутентификации для NiFI

Касательно Apache Nifi, поставляемом в составе Платформы, начиная с релиза 2.13.0 произошли следующие существенные изменения:

- Повышение поставляемой версии Apache Nifi до 1.19.1.
- Добавлена поддержка native-аутентификации пользователей через KeyCloak, с использованием протокола OpenId Connect.
- Добавлена возможность генерации TLS-сертификатов для узлов NiFi при развертывании, что необходимо для включения native-аутентификации.

В результате этих доработок, Платформы теперь поддерживает два варианта аутентификации и авторизации пользователей для NiFi:

1. Аутентификация с использованием OAuth Proxy (oauth-proxy), этот режим существовал и в предыдущих версиях.

- Плюсы: проще в настройке: не требует сертификатов на узлах NiFi, не требует TLS на узлах NiFi, не требует создания учетных записей в NiFi.
- Минусы: нет полноценной авторизации (разделения прав доступа) - любой пользователь, который сможет аутентифицироваться через OAuth Proxy в NiFi будет являться полным администратором и фактически на уровне NiFi нет разделения учетных записей пользователей (а сама возможность аутентификации на proxy ограничивается одной общей группой в KeyCLoak).

2. Аутентификация NiFi напрямую в OpenID Connect Provider (в KeyCloak), она же native-аутентификация

- Плюсы: с точки зрения NiFi пользователи обладают отдельными полноценными учетными записями, которым можно предоставлять и ограничивать права средствами самого NiFi, все действия в NiFi реализуются от имени конкретной (а не общей) учетной записи.
- Минусы: требуется включение TLS на узлах NiFi, что в свою очередь требует выдачи и обновления TLS-сертификатов узлов. Так же требуется создание и управление учетными записями в NiFi (фактически надо поддерживать два согласованных набора учетных записей для пользователей - в KeyCloak и отдельно в NiFi).

9.2. Генерация сертификатов для узлов NiFi

Для поддержки аутентификации пользователей средствами самого NiFi (а не внешними вроде oauth-proxy), узлы Apache NiFi должны быть сконфигурированы в т.н. защищенный кластер и на них должен быть включен TLS.

Для работы TLS необходимо обеспечить каждый узел кластера NiFi собственным уникальным закрытым ключом и TLS-сертификатом.

Для решения этой задачи в систему установки Платформы начиная с версии 2.13.0 добавлен инструментарий PKI, позволяющий генерировать сертификаты на этапе развертывания для узлов некоторых сервисов инфраструктуры, включая NiFi. Инструментарий поддерживает два режима работы:

- Генерация сертификатов силами решения cert-manager, который должен быть предварительно установлен в кластере. В этом случае zifctl создает Namespace-level Issuer (если нужно) и создает объекты certificate с необходимыми параметрами. Это рекомендованный метод создания и управления сертификатами.
- Генерация сертификатов силами самого zifctl: в этом случае на этапе создания конфигурации создается RootCA-сертификат, уникальный для данной инсталляции, и на этапе развертывания zifctl выпускает необходимые TLS-сертификаты и самостоятельно загружает их в объекты secret кластера Kubernetes. Обновление сертификатов выполняется при очередных обновлениях платформы (zifctl sync).

9.3. Управление пользователями в NiFI

Если используется oauth-proxy аутентификация, то управление пользователями в NiFI невозможно: все пользователи с точки зрения Apache NiFi являются анонимными с полными административными правами (фактически в самом NiFi аутентификация и авторизация отключены). Администратор платформы может только ограничить список тех, кто может зайти в NiFi с помощью специальной роли в KeyCloak: apache-nifi-client:nifi-admin (client role).

Если используется native аутентификация, то пользователи будут представлены в NiFi полноценными индивидуальными учетными записями, которым можно назначить права и ограничения средствами самого NiFI. Но при это Apache NiFi не предоставляет штатных средств синхронизации этих учетных записей с провайдером аутентификации. Для логина пользователя с использованием KeyCloak необходимо:

- Создать учетную запись пользователя в KeyCloak (или синхронизировать ее из ActiveDirectory).
- Создать точно такую же учетную запись в NiFi (она должна совпадать с полем email УЗ в KeyCloak).
- Назначить права доступа учетной записи в NiFi (и/или включить ее в соответствующие группы в NiFi).
- При удалении учетной записи из KeyCloak, необходимо удалить и соответствующую учетную запись из NiFi.

После начального развертывания в NiFi будет создана только одна учетная запись (initial admin identity), соответствующая создаваемой zifctl учетной записи в KeyCloak nifi-admin.

9.4. Возможность развертывания стороннего приложения на кластеры kubernetes при помощи zifctl

9.4.1. Основные принципы развертывания приложения при помощи zifctl

- приложение инициализируется (создает конфигурационные файлы) используя образ zifctl в режиме установки приложения (--type app)
- приложение развертывается в отдельный неймспейс k8s, отличный от неймспейса платформы

- приложение "читает" конфигурацию платформы, из неймспейса платформы для получения необходимых учетных записей и информации об окружении на этапе инициализации;
- приложение при помощи образа `zifctl` производит установку (`sync`) конфигурации на кластер `k8s`.

9.4.2. Функционал установки приложения при помощи zifctl

- создание необходимых баз в `postgres`;
- создание клиентов, пользователей, ролей в `keycloak`;
- публикация необходимых сведений о платформе в неймспейсе приложения;
- настройка политики ABAC на платформе для приложения.

9.4.3. Этапы развертывания приложения инсталлятором

Установка стороннего приложения средствами `zifctl` во многом повторяет логику развертывания платформы (`init` → `sync`).

Подготовка

Этапы подготовки окружения к развертыванию приложения практически идентичны этапам, которые необходимо пройти в случае установки платформы:

- создание `AGE`-ключа шифрования;
- создание `namespace` на целевом кластере;
- создание сервисной учетной записи (`Service Account`) с правами на запись в namespace приложения и с правами на чтение объектов `configmaps`/`secrets` из `namespace` платформы.

Инициализация конфигурации (`init`)

Обязательным условием для инициализации конфигурации приложения является указание параметра `--type app`.

Пример команды:

```
# Определяем переменные
export ZIF_REGISTRY=docker-group.idp.yc.ziiot.ru
export ZIF_AGE_KEY=************
export ZIF_SERVER=https://51.250.9.223
export ZIF_TOKEN=************
export SA=sa-name-for-deployment
export ZIFCTL_REG_USERNAME=************@idp.zyfra.com
export ZIFCTL_REG_PASSWORD=************
# Помощь по параметрам команды init
```

```
zifctl init –help

zifctl init --env-path /path/to/env/config \

            --type app \

            --app-name sampleapp \

            --sa $SA \

            --infra-ns $INFRA_NS \

            --age-key $ZIF_AGE_KEY \

            --namespace $PROJECT_NAME \

            --hostname $KUBE-API \

            --registry-username $ZIFCTL_REG_USERNAME \

            --registry-password $ZIFCTL_REG_PASSWORD \

            --registry $ZIF_REGISTRY

            --token $ZIF_TOKEN
```

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

- `--namespace` - задает `Namespace` куда будет устанавливаться приложение;
- `--type app` - режим установки приложения;
- `--app-name` - опциональный параметр, задает имя развертываемого приложения
- `--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
- `--infra-ns` - `Namespace` в котором установлена инфраструктура платформы (учетная запись, используемая в ключе "--sa" должна иметь `rolebindings` с правами на чтение конфигмэпов и секретов в неймсейсе, указанном в `--infra-ns`);
- `--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>` (ВАЖНО: указывается не ссылка URL, а именно `hostname`);
- `--age-key` - (или переменная `ZIF_AGE_KEY`) опциональные и фактически задают значения для основных параметров в файлах `env-values.yaml` и `env-secrets.yaml`;
- `--registry` - имя репозитория откуда кластер `K8s/OpenShift` должен брать контейнеры Платформы и инфраструктуры (по умолчанию `docker.idp.yc.ziiot.ru`);
- `--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 `--token - kubeconfig` сервисной учетной записи кластера `k8s`.

При успешной инициации должно появиться следующее сообщение:

```
2024-10-12 11:32:10,242 DEBUG   cmd: Finish processing command: init, execution time:
0.6128787994384766 s
```

После инициализации в каталоге, указанном в ключе `--env-path` появятся следующие файлы:

```
├── deployment-id.yaml
├── env-secrets.yaml
├── env-values.yaml
├── trusted-root-ca
│   └── README.md
└── values
    └── global
        └── global-config.yaml.gotmpl
```

- `deployment-id.yaml` - `id` развертывания;
- `env-secrets.yaml` - секреты в зашифрованном виде;
- `env-values.yaml` - общая конфигурация развертываемого приложения;
- `trusted-root-ca` - директория с корневыми сертификатами, используемые сервисами приложения;
- `values` - директория с конфигурациями конкретного сервиса.

Файл "`global-config.yaml.gotmpl`" содержит конфигурацию, которая будет применена ко всему приложению. В следующих версиях `zifctl` данный файл не будет создаваться при инициации конфигурации.

После добавления файлов необходимых для дальнейшего развертывания, директория будет выглядеть следующим образом:

```
├── services
│   └── service-list.yaml
├── deployment-id.yaml
├── env-secrets.yaml
├── env-values.yaml
├── trusted-root-ca
│   └── README.md
└── values
    ├── global
```

```
    │   └── global-config.yaml.gotmpl
    └── app-folder
        ├── service1.yaml.gotmpl
        ├── service2.yaml.gotmpl
        └── serviceN.yaml.gotmpl
```

- `service-list.yaml` - `service-list.yaml` содержит список сервисов, являющихся частью приложения
- `env-values.yaml` - основной конфигурационный файл развертываемого приложения
- `*.gotmpl` файлы конфигурации под каждый конкретный сервис

В файле `env-values.yaml` необходимо указать имя приложения, которое планируется к развертыванию:

```
appName: sample-app
```

Примечание. Указанное имя приложения должно быть идентичным следующему значению, указанному в `service-list.yaml`:

```
modules:
  - name: sample-app
```

Модификация service-list.yaml

`service-list.yaml` - один из ключевых конфигурационных файлов приложения. Данный файл описывает сервисы, которые необходимо развернуть, настраивая, следующие параметры:

- имя сервиса;
- путь к образу;
- тэг;
- пробы;
- сетевой доступ;
- инициализации;
- политики доступа;
- лимиты/реквесты;
- и т.д.

Пример конфигурации `service-list.yaml`:

```
platform:
  version: 1.1.1
  configSchemaVersion: 100
modules:
  - name: sample-app
    chart: zif-sample-module
    description: Sample app to be deployed via zifctl
    importRoles: true
    services:
      - name: app-number-1
        enabled: true
        gitLabProjectID: 2190
        image: app-gatewayrouting
        imageNamespace: idp-pmc
        tag: 1.1.1
        auth: confidential
        ingressPath: ""
        type: netcore
        postgres: true
        abac:
          enabled: true
      - name: app-number-2
        enabled: true
        gitLabProjectID: 2191
        image: app-number-2
        imageNamespace: idp-pmc
        tag: 1.1.1
        auth: confidential
        ingress: true
        ingressPath: /
        type: netcore
        postgres: no
```

Модификация "*.gotmpl" файлов

`Helmfile` может наполняться значениями как с помощью `yaml` файлов, так и посредством `Go` шаблонов. Например, шаблон конфигурации сервиса `service1` будет "читаться" из файла `values/app-folder/service1.yaml.gotmpl`. В файле `values/app-folder/defaults.yaml.gotmpl` задаются параметры, переменные, которые будут переданы всем сервисам приложения.

Пример параметров, которые возможно определить в `Go` шаблонах:

```
# Переменные окружения для сервиса
environmentVars:
  TZ: Europe/Moscow
  JAEGER_ENABLED: "false"
# Порт, который будет использоваться для маппинга сервиса
containerPorts:
  - name: http
    container: 7991
    service: 80
# Переопределение реквестов и лимитов
resources:
  requests:
    cpu: 100m
    memory: 128Mi
  limits:
    cpu: 500m
    memory: 512Mi
# Определение конфигурационных файлов для сервиса
extraConfigMaps:
  json-sample-config:
    sample.json: |-
      {
        "wsUrl": "wss://{{ .Values.externalBaseHostname }}/ws",
        "restUrl": https://{{ .Values.externalBaseHostname }}",
        "projectName": "SampleProject"
      }

  yaml-sample-config:
```

```
    sample.yaml: |-
      sampleconf:
        - enabled: true
        - type: yaml
```

Включение ABAC

Для включения\импорта `ABAC` политик приложения, данные политики необходимо определить в `Go` темплейтах, после чего `zifctl` монтирует политики и импортирует их в `zif-security`.

Примечание. За формирование и предоставление корректных шаблонов `ABAC` политик отвечает команда приложения.

Процедура:

1. Включить политики `ABAC` в `env-values.yaml` платформы:

```
## Конфигурация ABAC
abacAuthorization:
  enabled: True
  importPolicies: True
```

2. Аналогичным образом включить `ABAC` политики в Приложении:

```
## Конфигурация ABAC
abacAuthorization:
  enabled: True
  importPolicies: True
```

Так как политики `ABAC` "включены" для всех сервисов по умолчанию, то в случае отсутствия необходимости использования `ABAC`, отключение политики необходимо указать в `service-list.yaml` явным образом:

```
 services:
      - name: app-number-1
        abac:
          enabled: false
```

3. Синхронизировать версию `keycloak` приложения и платформы

Версия `keycloak` в `env-values.yaml` платформы должна быть идентичной версии `keycloak` указанной в `env-values.yaml` приложения. В данный момент наличествует совместимость приложения с `keycloak` версий 17 и 21.

```
## Конфигурация KeyCloak
keycloak:
  installed: False
  baseExtUri: 'https://sample-platform-path.kube07.yc.ziiot.ru/auth'
  version: 21
```

4. Внести изменения в `Go` темплейты:

```
extraConfigMaps:
{{/* configMap for abac policies */}}
{{- if (index .Values.ZifService "abac") }}
  {{- if (.Values.ZifService.abac | get "enabled" true) }}

  {{ .Values.ZifService.name }}-abac-policy:
    {{ .Values.ZifService.name }}.json: |-
      {
          "id": "sample-policy",
          "description": "Политика sample",
          "actions": [
              {
                  "id": "update",
                  "name": "Обновить"
              }
          ],
          "policy": {
              "defaults": {
                  "advice": "Обратитесь к руководителю АСУТП"
              },
              "allowRules": [
                  {
                      "name": "Доступ на чтение и изменение",
```

```
                    "description": "Чтение и запись для роли: zif-datalink.folders-
editor",
                    "subj": {
                        "contains": [
                            {
                                "attribute": "Roles",
                                "comparison": "contains",
                                "value": "zif-datalink.folders-editor"
                            }
                        ]
                    },
                    "obj": {
                        "match": [
                            {

                                "attribute": "Id",
                                "comparison": "match",
                                "value": "*"
                            }
                        ]
                    },
                    "act": ["write", "update", "create", "read"]
                }
            ]
        }
    }
  {{ end }}
{{ end }}
```

5. Настроить `init job` модуля приложения в файле `module-config.yaml.gotmpl`.

```
initjobs:
  {{ .Release.Name }}-init:
    mountConfigMaps:
      {{ .Values.ZifService.name }}-abac-policy:
```

```
        configMap: {{ .Values.ZifService.name }}-abac-policy
        mountPath: /platform/init/module/data/abac/policies/{{ .Values.ZifService.name }}/{{ .Values.ZifService.name }}-abac-policy.json
        subPath: {{ .Values.ZifService.name }}.json
    environmentVars:
      REST_ZIF_SECURITY_URL:  http://zif-security.sample-platform-ns
      LARGE_CLIENT_HEADER_BUFFERS: "4 32k"
```

Модификация env-values.yaml

Файл `env-values.yaml` содержит верхнеуровневые настройки приложения, которые редко изменяются. В `env-values.yaml` указываются имя приложения, неймспейс платформы, неймспейс приложения, `storage class`, и.т.д. Так же в этом конфигурационном файле можно настроить установку сторонних сервисов, таких как `redis`, `kafka`, `cassandra`, `rabbitmq`, etc.

Эталонный `env-values.yaml` приложения:

```
configType: app
namespace:  'app-sample-ns'
serviceAccount: 'app-sample-ns-sa'
tenantName: 'ziiot'
tenantId: 'ziiot'
externalBaseHostname: 'app-sample-ns.kube02.yc.ziiot.ru'
infraHA: False
appName: sample-app
sharedInfraNamespace: 'ziiot-sample-ns'
storage:
  storageClass: sample-sc-ssd
## Конфигурация Redis
redis:
  installed: True
  host: 'zif-redis-master.app-sample-ns'
  port: 6379
  storageSizeMB: 1024
```

Примечание. На текущий момент работоспособность приложения протестирована с платформой, развернутой в режиме монолитного инстанса. Значения полей `'tenantName'` и `'tenantId'` должны

совпадать с соответствующими полями развернутой платформы. Значение полей `'tenantName'` и `'tenantId'` должно быть следующим:

```
tenantName: 'ziiot'
tenantId: 'ziiot'
```

Примечание. При формировании имен баз данных и учетных записей БД, будут учитываться имя приложения, неймспейс, имя сервиса, но результирующая строка будет ограничена 63-мя символами.

Применение (sync) конфигурации приложения

Команда для развертывания и обновления приложения одна и та же (`'zifctl sync'`). Практически все операции, выполняемые системой развертывания, идемпотентны и при повторном выполнении «установки», поверх уже выполненной ранее, конфигурация приложения не изменится. 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. Подробнее о процедуре генерации пары ключей средствами `age-keygen` описано в документации `zifctl`.

Синхронизация стороннего приложение на целевой ландшафт:

```
zifctl sync --env-path /path/to/env/config \
                    --age-key $ZIF_AGE_KEY \
                    --server $ZIF_SERVER \
                    --token $ZIF_TOKEN
```

10. Профили postgres (postgres profiles) или настройки postgres для мультикластерных PG-инсталяций

Малые инсталяции платформы или инсталяции с небольшим количеством данных позволяют обойтись скейлингом Платформы в `kubernetes`.

В редких случаях требуется не только выносить `PG` на ВМ (для удовлетворения требований `ИБ`), но и раскидывать `БД` инсталяции по разным кластерам `PG` для разгрузки ВМ и удовлетворения нужного количествава "9-ок" по доступности.

Начиная с версии `zifctl 2.16.X` появилась возможность в конфигурации учитывать такое разделение. Как его настроить описано ниже:

Внутренняя структура работы с `PG` у Платформы:

К PG ходят 2 сущности платформы:

- `init job`, которые получают `user/pass` как `admin user`, `dbname` генерируют на лету на основе `tenantID` и соответствующего сервиса, `host/port/connectdb` из переменных `ENV (POSTGRES_HOST / POSTGRES_PORT / POSTGRES_CONNECT_DB)`.
- сервисы Платформы, который получают `user/pass/dbname/host/port` из переменных окружения.

Начиная с версии `zifctl 2.16.X` можно полноценно переместить сервис на другие `postgres.host / postgres.port`, а `init job` заставить создать БД/поправить права на неё и т.д. на нужном вам кластере с помощью `postgres profiles` и параметра `connectDB`.

Внимание! Для сервиса настройка `connectDB` не нужна. Эта настройка нужно только для `init job` для выполнения задач по созданию БД сервиса, пользователя и создания/исправления прав на БД. Т.е. чтобы выполнить запрос `CREATE DATABASE` и подобные `init job` должна куда-то подключится (не указывая БД при запуске командочке в `psql` вы на самом деле подключаетесь к `БД postgres`).

10.1. Переключение сервиса Keycloak на другой PG

В рамках `Keycloak` теперь возможно полноценно с текущей версии начинается работать на двухкластерной инсталяции Платформы, когда всё кроме `keycloak` работало на `default.cluster`, а сам `keycloak` на `new.cluster`:

- создайте новый профиль `PG` и переопределить в нём `host/port/connectDB` сущности (одну из них как минимум):

```
$ grep -A 11 -F 'postgres:' ./env-config/env-values.yaml
postgres:
  installed: True
  host: 'default.cluster'
  port: 5432
  connectDB: postgres
  storageSizeMB: 5120
```

```
  profiles:
    fakeprofilename:
      host: 'new.cluster'
      port: 5432
      connectDB: postgres
```

- переключить сервис `keycloak` на нужный профиль в `env-values.yaml`:

```
$ grep -A 4 -F 'keycloak:' ./env-config/env-values.yaml
keycloak:
  installed: True
  postgresProfile: fakeprofilename
  baseUri: 'http://zif-keycloak-http.dev-dyukov/auth'
  baseExtUri: 'https://dev-dyukov.kube07.yc.ziiot.ru/auth'
```

10.2. Переключение сервиса X (не keycloak) на другой PG

В рамках `Keycloak` теперь возможно полноценно с текущей версии начинается работать на двухкластерной инсталяции Платформы, когда всё кроме `keycloak` работало на `default.cluster`, а сам `keycloak` на `new.cluster`:

В текущей реализации Платформы сервис `X` можно полноценно подключить к работе на двухкластерной инсталяции Платформы, когда всё кроме `X` работало на `default.cluster`, а сам `X` на `new.cluster`:

- создать новый профиль PG и переопределить в нём `host/port/connectDB` сущности (одну из них как минимум):

```
$ grep -B 9 -A 1 myfakeprofilename  ./env-config/env-values.yaml
postgres:
  installed: True
  host: 'default.cluster'
  port: 5432
  connectDB: postgres
  storageSizeMB: 5120
  profiles:
    fakeprofilename:
      host: 'new.cluster'
```

```
      port: 5432
      connectDB: postgres
```

- переключить сервис X на нужный профиль через `service-list-patch.yaml`:

```
$ cat ./services/service-list-patch.yaml
modules:
  - name: rtdb
    services:
      - name: zif-rtdb-metadata
        postgresProfile: fakeprofilename
```

- **Внимание**! Если Вы забыли (или намеренно) не указали параметры в профиле они будут взяты из `postgres.host / postgres.port / postgres.connectDB`, т.е.:

```
postgres:
  host: 'default.cluster'
  port: 5555
  connectDB: postgres1
  profiles:
    fakeprofilename:
      connectDB: postgres
```

Обращение выше можно заменить аналогичным более подробным:

```
postgres:
  host: 'default.cluster'
  port: 5555
  connectDB: postgres1
  profiles:
    fakeprofilename:
      host: 'default.cluster'
      port: 5555
      connectDB: postgres
```

- пустой профиль является некорректной конфигурацией, потому как нет смысла создавать пустой профиль в виду пункта выше:

```
# некорректная конфигурация
postgres:
  host: 'default.cluster'
  port: 5555
  connectDB: postgres1
  profiles:
    myprofile1:
    myprofile2:
```

- после создания двух дополнительных кластеров PG для двух баз и не используете никакие балансировщики, то типичная конфигурация будет выглядеть так:

```
postgres:
  host: 'default.cluster'
  port: 5432
  connectDB: postgres1
  profiles:
    cluster1:
      host: 'custom1.cluster'
    cluster2:
      host: 'custom2.cluster'
# port не нужно указывать, если он совпадает с postgres.port
# В данном случае у меня 3 кластера на разных машинах, мастера которых доступны по hostname:
# 1) custom1.cluster
# 2) custom2.cluster
# 3) default.cluster
```

- если установлен балансировщик и несколько кластеров PG, то порядок настройки:
  1) На каждом из кластеров создать маркировочные БД "пустышки" для создания маршрута через балансировщик для `init job`, например `cluster1db` , `cluster2db`, `cluster3db`.
  2) Настройте профили:

```
postgres:
  host: 'pgbouncer.manualinfra'
  port: 5432
  connectDB: cluster1db
  profiles:
    secondcluster:
      connectDB: cluster2db
    thirdcluster:
      connectDB: cluster3db
# host и port в каждом из профилей не требуется дублировать, т.к. он совпадает с postgres.host и postgres.port
# В данном случае у меня 3 кластера на разных машинах, ко всем мы ходим через балансировщик на pgbouncer.manualinfra:5432
```

3) Переключите сервисы на нужный профиль:

```
$ cat ./services/service-list-patch.yaml
modules:
  - name: rtdb
    services:
      - name: zif-rtdb-metadata
        postgresProfile: thirdcluster
  - name: om
    services:
      - name: zif-om-object
        postgresProfile: secondcluster
```

4) Произвести команду `zifctl sync`.

11. Автоматическое создание и конфигурация топиков Kafka в процессе развертывания

Система развертывания должна автоматически создавать топики в `Kafka` с указанными параметрами (`replicas`, `partitions`) и обновлять параметры топиков, если они изменились.

В рамках данной задачи решается вопрос только управления "статическими" топиками, состав и названия которых известны заранее на этапе конфигурации системы. Управление "динамическими" топиками, которые создаются сервисами в процессе работы "по запросу" вне этой задачи.

В рамки этой задачи так же входит управление топиками `Debezium` (`zifctl` должен взять на себя создание и изменение параметров топиков). Конфигурация `Debezium` не меняется относительно существующей, просто добавляется их предварительное создание вместе со всеми остальными.

Изначально заложенные платформой параметры топиков определены в файле `/services/kafka-topics-list.yaml`, по формату структуре аналогичен `/services/service-list-patch.yaml`. Так же файл является статичным в образе `zifctl` и изменению не подлежит.

Начиная с 2.16.0 появилась возможность управления `kafka` топиками.

Для того, чтобы внести изменения в `kafka` топики для конкретного экземпляра Платформы, вам необходимо создать патч-файл `/services/service-list-patch.yaml` в каталоге конфигурации окружения, в котором будут перечислены те модули и сервисы, в параметры которых вы хотите внести корректировки:

- для включения роли по управлению топиков необходимо убедиться что в `env.values.yaml` включена `tasks -kafka` или `-all`. Пример файла `service-list-patch.yaml``:

```
init:
  tasks:
    - all
```

- для управления топика необходимо в `/services/service-list-patch.yaml` на уровне сервиса добавить блок `kafka`: вложенным словарем топиков и их параметрами. Все указанные параметры указаны в примере ниже:

```
modules:
  - name: <<module>>
    services:
      - name: <<service>>
        enabled: true
        kafka:                                          #required.
           enabled: true                                #required.
           topics:                                      #required.
              topic-1:                                        #required. Должно
совпадать с названием топика без префикса "tenantid__"
```

```
                name: "topic-1"                                #required. Наименование топика без префикса "tenantid__". Пр. <<tenantid>>__zif-om-object__dbz.public.properties_s, должен быть указан как zif-om-object__dbz.public.properties_s
                partitions: "1"                                #required. default = "1". ВАЖНО!!! Данный параметр не может быть ниже чем текущее количество партиций в указанном топике
                cleanup_policy: "delete,compact"               #optional. default = "delete" ["compact","delete","delete,compact"]
                segment_ms: "604800000"                        #optional. default = "604800000" (7 days) [1,...]
                flush_messages: "9223372036854775807"          #optional. default = "9223372036854775807" [1,...]
                max_message_bytes: "1048588"                   #optional. default = "1048588" [0,...]
                min_insync_replicas: "1"                       #optional. default = count pods cp-kafka
                retention_bytes: "-1"                         #optional. default = "-1" [-1, 0,...]
                retention_ms: "86400000"                       #optional. default = "86400000" (1 day) [-1, 0,...]
                flush_ms: "9223372036854775807"                #optional. default = "9223372036854775807" [0,...]
                segment_bytes: "1073741824"                    #optional. default = "1073741824" (1 gibibyte) [14,...]
                segment_jitter_ms: "0"                        #optional. default = "0" [0,...]
                unclean_leader_election_enable : "false"     #optional. default = "false" ["false","true"]
              topic-2:
                name: "topic-2"
                partitions: "2"
                retention_ms: "100000"
                flush_ms: "200000"
                cleanup_policy: "compact"
                segment_ms: "2000000000"
              topic-N:
                name: "topic-N"
                partitions: "2"
```

```
  - name: <<module>>
    services:
      - name: <<service>>
        enabled: true
        kafka:
          enabled: true
          topics:
            topic-N42:
              name: "topic-N42"
              partitions: "16"
              retention_ms: "14400200"
```

- для 2.16.0 `/services/kafka-topics-list.yaml` имеет следующий вид:

```
modules:
  - name: om
    services:
      - name: zif-om-object
        kafka:
          enabled: true
          topics:
            zif-om-object__dbz.public.propertyconfigurations_s:
              name: "zif-om-object__dbz.public.propertyconfigurations_s"
              partitions: "1"
            zif-om-object__dbz.public.properties_s:
              name: "zif-om-object__dbz.public.properties_s"
              partitions: "1"
            zif-om-object__dbz.public.objects_s:
              name: "zif-om-object__dbz.public.objects_s"
              partitions: "1"
```

Технические детали решения:

1) В репозитории системы развертывания состав релиза (список топиков и их параметры) хранится в файле `/services/kafka-topics-list.yaml`.

2) При выполнении установки утилита `zifctl` с помощью вспомогательного скрипта `/scripts/hooks/prepare/15-merge-services-list.py` выполняет полный merge `/services/kafka-topics-list.yaml` с `/services/service-list-patch.yaml` (если этот файл существует).

При конфликте значение берется из `/services/service-list-patch.yaml`. Далее полученный файл объединяется с `/services/*service-list.yaml` согласно Правилам объединения базового и патч-файлов.

3) Добавлены кастомные роли (`roles`) (`initrolessrv_kafkatasksmain.yaml`), `module_utils` (`scriptslibansiblemodule_utils`), `modules` (`scriptslibansiblemoduleskafka_topic.py`) для управления `kafka topics`.
4) Вызов роли происходит с помощью существующих `Job` формирующих `initmodule00-init.yaml`.
5) Добавлена новая `init tasks ("kafka")` отвечающая за вызов роли.
6) Вынесена конфигурирование топиков по умолчанию в отдельный файл `services/kafka-topics-list.yaml`. Все кастомные изменения вносятся в `services/service-list-patch.yaml`.
7) В базовый образ `zifctl-base` добавлены компоненты `images/zifctl-base/requirements.txt`:

```
kafka-python>=2.0.0,<2.1.0
kazoo==2.6.1
pure-sasl==0.5.1
jsonmerge==1.9.2
```

8) Управление топиков основано на https://github.com/StephenSorriaux/ansible-kafka-admin.

12. Включение TLS, аутентификации и авторизации в Redis

Для обеспечения безопасности данных и шифрование соединения необходимо реализовать защищенное `TLS` соединение с `Redis` а так же добавить возможность включение аутентификации и авторизации.

В данной документации будет рассмотрено, как включить различные режимы безопасности и аутентификации для `Redis`, а также какие параметры буду добавлены в деплойменты в каждом режиме.

12.1. Включение режима TLS (Transport Layer Security)

`TLS` обеспечивает шифрование данных между клиентами и сервером `Redis` для обеспечения безопасности передачи данных.

Для включения режима `TLS`, установите следующий параметр в файле конфигурации `env-values.yaml`:

```
pki:
  enabled: true
redis:
  tls:
    enabled: true
```

Создаваемые параметры:

При включении режима `TLS` будут сгенерированы следующие переменные окружения:

- `REDIS_HOST=\<адрес подключения к redis>`;
- `REDIS_PORT=\<порт подключения, по умолчанию 6380>`;
- `REDIS_SSL=true`: Устаревшая переменная для указания поддержки `TLS`. Оставлена для совместимости;
- `REDIS_TLS_ENABLED=true`: Эта переменная указывает на поддержку `TLS`;
- `REDIS_TLS_CA_CRT_PATH=\<путь к корневому сертификату>`: Путь к корневому сертификату для `TLS`.

Подключены и смонтированы следующие `volumes`:

- `zif-generic-tls`;
- `zif-redis-tls`.

12.2. Включение режима AUTH (аутентификация)

Режим `AUTH` предоставляет аутентификацию клиентов при подключении к серверу `Redis`.

Для включения режима `AUTH`, установите следующий параметр в файле конфигурации `env-values.yaml`:

```
redis:
  auth: true
```

Создаваемые переменные окружения:

При включении режима `AUTH` будет сгенерирована следующая переменная окружения:

- `REDIS_HOST=\<адрес подключения к redis>;`
- `REDIS_PORT=\<порт подключения, по умолчанию 6379>;`
- `REDIS_PASSWORD=\<ваш пароль>`: Эта переменная указывает на установленный пароль для аутентификации.

12.3. Включение режима AUTH и ACL

Режим `AUTH` и `ACL` объединяет в себе аутентификацию клиентов и управление списками контроля доступа для максимальной безопасности и управления доступом.

Для включения режима `AUTH` и `ACL`, установите следующие параметры в файле конфигурации `env-values.yaml`:

```
redis:
  auth: true
  acl: true
```

Создаваемые переменные окружения при включении режима `AUTH` и `ACL` будут сгенерированы следующие переменные окружения:

- `REDIS_HOST=\<адрес подключения к redis>;`
- `REDIS_PORT=\<порт подключения, по умолчанию 6379>;`
- `REDIS_PASSWORD=\<ваш пароль>`: Эта переменная указывает на установленный пароль для аутентификации;
- `REDIS_USER=\<имя пользователя, по умолчанию tenant_modulename>`.

12.4. Включение режима TLS, AUTH и ACL

Режим `TLS` и `ACL` объединяет в себе шифрование данных, обязательной аутентификации и управление списками контроля доступа для максимальной безопасности.

Для включения режима `TLS` и `ACL`, установите следующие параметры в файле конфигурации `env-values.yaml`:

```
pki:
  enabled: true
redis:
```

```
  tls:
    enabled: true
  auth: true
  acl: true
```

При включении режима `TLS` и `ACL` должны создаваться все переменные окружения и монтироваться `volumes` как и в режимах `TLS` и `ACL`.

12.5. Режим публикации внешнего Ingress/Route соединения для сервера Redis

Этот режим позволяет обеспечить внешний доступ к серверу `Redis` через `Ingress` или `Route` с использованием `SSL passthrough`.

Параметры, необходимые для включения:

```
pki:
  enabled: true
redis:
  tls:
    enabled: true
  externalAccess: True
```

Для корректного создания внешнего доступа к `redis` в кластере с `ingress nginx` должен быть включен параметр `--enable-ssl-passthrough` при запуске `ingress controller`.

Создаваемые сущности:

При включении внешнего доступа будет создан `ingress/route` с адресом формата `\<namespace>-redis.\<domain.example>`.

В ingress присутствует аннотация `[nginx.ingress.kubernetes.io/ssl-passthrough](http://nginx.ingress.kubernetes.io/ssl-passthrough): "true"\`, указывающая на использовании `nginx` режима `ssl-passthrough` для этого адреса.

Подключение:

При подключении через любую утилиту или программу нужно указать:

- адрес указанный в `ingress/route`;
- порт подключения 443;
- указать путь до `CA` сертификата или сертификат в текстовом виде (зависит от способа);
- указать `SNI` который равен адресу;
- при использовании авторизации указать логин и пароль.

13. Включение отдельного Redis для инфраструктуры и настройка Apache Nifi

В ходе тестирования работы сервисов UDL обнаружилась избыточная сетевая нагрузка на сервис zif-redis-master, вызванная логикой работы платформенного процессора Apache Nifi_ProcessTSDSData.

Для решения этой проблемы в авто-развертывание добавлена возможность установки отдельного экземпляра Redis, который будет использоваться только инфраструктурными компонентами платформы.

В документе описаны действия, необходимые для установки инфраструктурного Redis и конфигурации процессоров Nifi, использующих Redis.

13.1. Установка Redis для инфраструктуры

При переустановке на уже развернутую Платформу:

1) Добавьте в файл конфигурации *\<путь до папки конфигурации zifctl>/env-config/env-values.yaml* следующие параметры (включение Redis для инфраструктурных сервисов):

```
env-values.yaml | Включение Redis для инфраструктурных сервисов
## Конфигурация Redis используемого только инфраструктурными сервисами
redisInfra:
  installed: True
  host: 'zif-redis-infra-master.<namespace>'
  auth: True
  port: 6379 # по умолчанию, можно не указывать
  storageSizeMB: 1024 # по умолчанию, можно не указывать
```

2) Расшифровуйте файл в секретами *\<путь до папки конфигурации zifctl>/env-config/env-secrets.yaml*, используя команду zifctl secrets dec бинарного установщика zifctl (Дешифрование):

```
env-secrets.yaml | Дешифрование
./zifctl secrets dec --env-path=$ENV_PATH --age-key=$AGE_KEY
 2023-09-05 16:15:25,083 INFO    secrets: Write decrypted file: /var/deploy/env/env-secrets.dec.yaml, don't forget to delete it!
```

В результате будет создан файл *\<путь до папки конфигурации zifctl>/env-config/env-secrets.dec.yaml*, в который необходимо добавить параметры для redisInfra (Добавление параметров Redis для инфраструктуры):

```
env-secrets.yaml | Добавление параметров Redis для инфраструктуры
...
redisInfra:
    adminPassword: <password>
    adminUser: admin
...
```

3) Зашифруйте файл при помощи команды `zifctl secrets enc`:

```
env-secrets.yaml | Шифрование
./zifctl secrets enc --env-path=$ENV_PATH --age-key=$AGE_KEY
 2023-09-05 16:24:55,527 INFO   secrets: Encrypt secrets file: /var/deploy/env/env-secrets.dec.yaml => /var/deploy/env/env-secrets.yaml
 2023-09-05 16:24:55,539 INFO   secret_store_base: Save secrets to file: /var/deploy/env/env-secrets.yaml
```

В результате в \<путь до папки конфигурации zifctl>/env-config/env-secrets.yaml`` будут добавлены зашифрованные параметры redisInfra`.

После удаления файла *\<путь до папки конфигурации zifctl>/env-config/env-secrets.dec.yaml* можно начать установку платформы командой `zifctl sync`.

При установке с нуля:

В `env-values.yaml` и `env-secrets.yaml` уже будут записаны параметры для `redisInfra`, перед выполнением команды `zifctl sync` необходимо только включить установку `redisInfra` (с аутентификацией):

```
env-values.yaml | Включение Redis для инфраструктурных сервисов
...
## Конфигурация Redis используемого только инфраструктурными сервисами
redisInfra:
  installed: True
  auth: True
...
```

13.2. Конфигурация Apache Nifi

Для тестирования корректной работы `Nifi` с `Redis` необходимо воспользоваться руководством из данного раздела.

Добавление сервисов контроллера `RedisConnectionPoolService` и `RedisDistributedMapCacheClientService`:

1) Войти в UI `Nifi` под административной УЗ.
2) Нажмите ПКМ и выберите `Configure`:

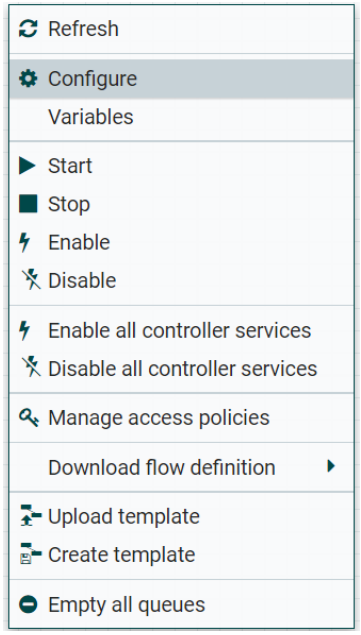

Рисунок 13.1 Configure

3) Перейдите на вкладку `Controller Services → кнопка "+"` в правом верхнем углу.
4) В открывшемся диалоговом окне `Add Controller Service` в текстовом поле `Filter` введите `Redis`:

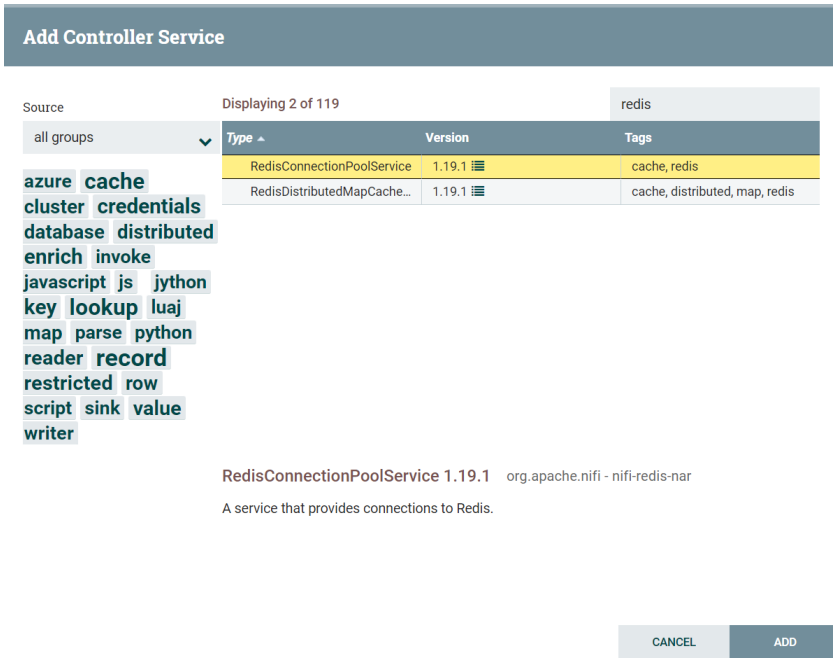

Рисунок 13.2 Redis

Последовательно добавляем сервисы контроллера `RedisConnectionPoolService`и `RedisDistributedMapCacheClientService`.

13.3. Конфигурация сервисов контроллера RedisConnectionPoolService и RedisDistributedMapCacheClientService

Конфигурация сервиса контроллера `RedisConnectionPoolService`:

1) Выберите кнопку конфигурации `RedisConnectionPoolService`:

Рисунок 13.3 RedisConnectionPoolService

2) В открывшемся диалоговом окне `Configure Controller Service RedisConnectionPoolService 1.19.1`, вкладка `Properties`, введите значения полей:

   - `Connection String`: `zif-redis-infra-headless.<namespace>:6379`;
   - `Password`: пароль администратора `redisInfra`; (`redisInfra.adminPassword` из `env-secrets.yaml` либо значение переменной `REDIS_INFRA_ADMIN_PASSWORD` из секрета `zif-global-secrets` в кластере);
   - `Apply`.

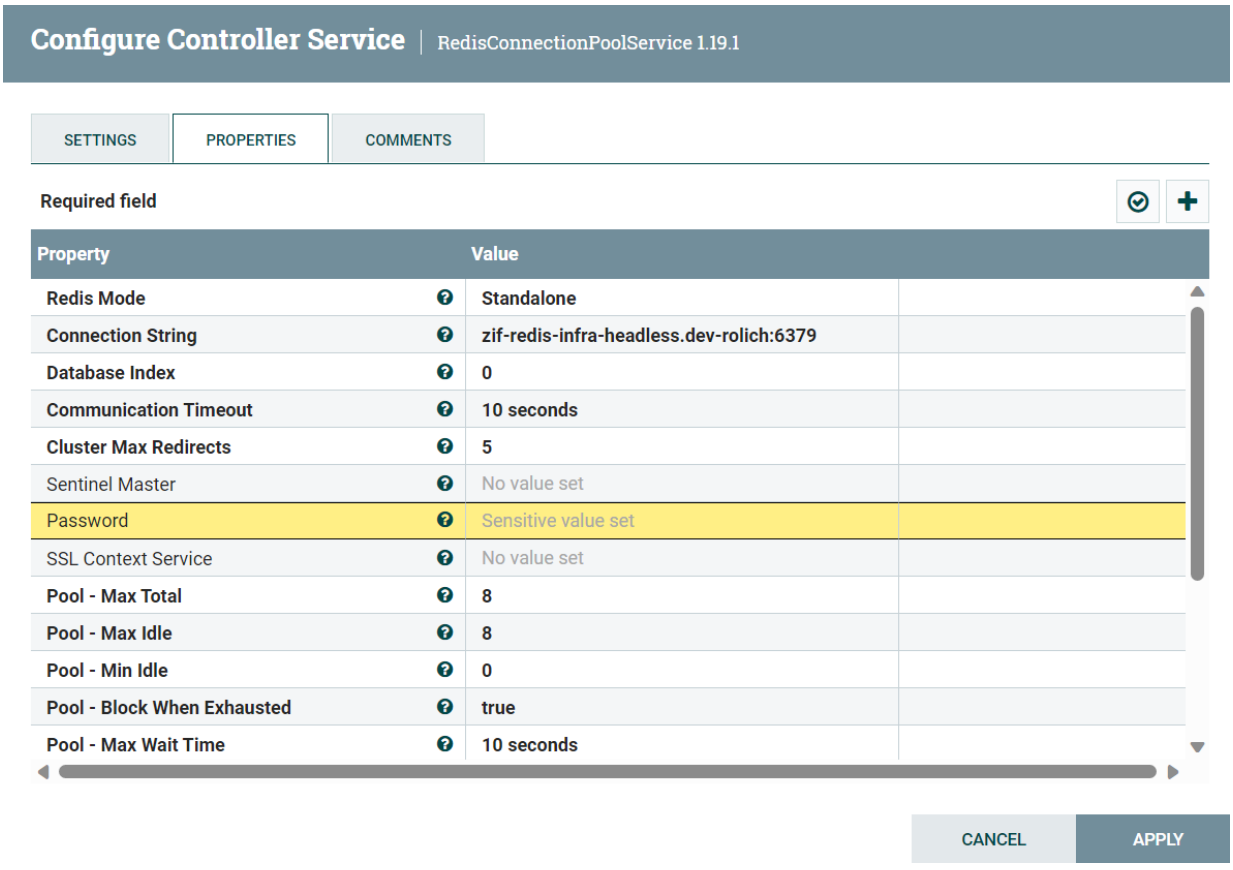

Рисунок 13.4 Вкладка Properties

Конфигурация сервиса контроллера `RedisDistributedMapCacheClientService`:

1) Нажмите кнопку конфигурации сервиса `RedisDistributedMapCacheClientService`.
2) В открывшемся диалоговом окне `Configure Controller \| ServiceRedisDistributedMapCacheClientService 1.19.1` выберите вкладку `Properties`.
3) Поле `Redis Connection Pool``, нажмите Value, в выпадающем списке выберите RedisConnectionPoolService → OK → Apply`.

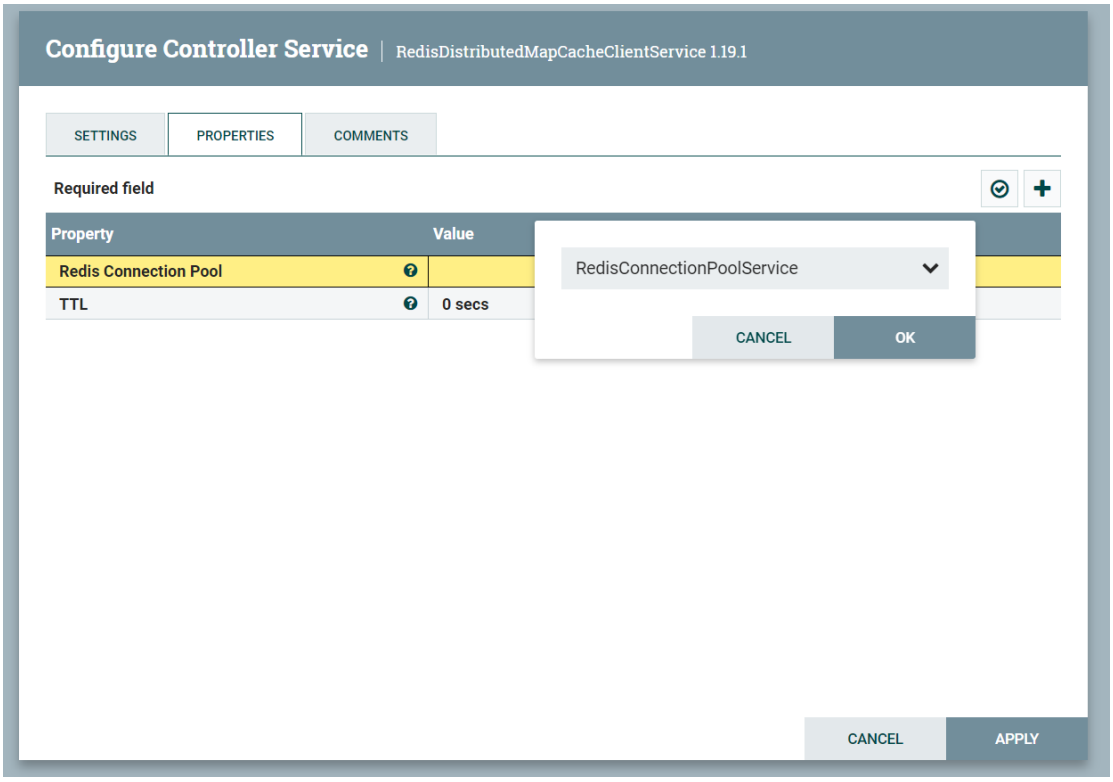

Рисунок 13.5 Configure Controller

13.4. Включение сервисов контроллера RedisConnectionPoolService и RedisDistributedMapCacheClientService

1) Включите сервис `RedisConnectionPoolService`:

Рисунок 13.6 RedisConnectionPoolService

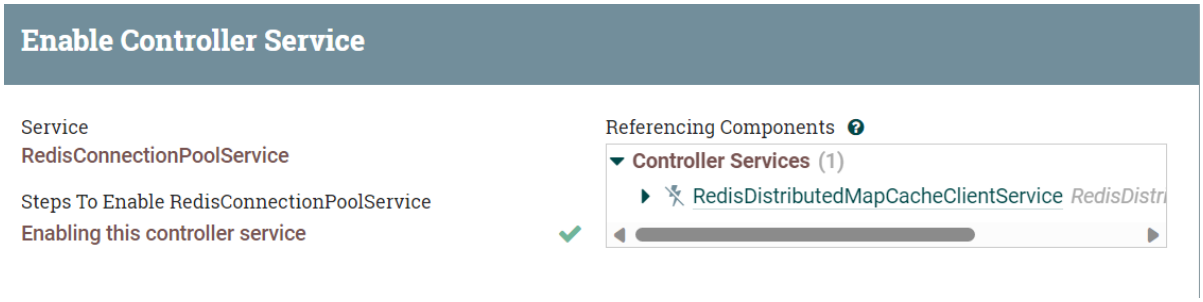

Рисунок 13.7 RedisConnectionPoolService

2) Включите сервис `RedisDistributedMapCacheClientService`:

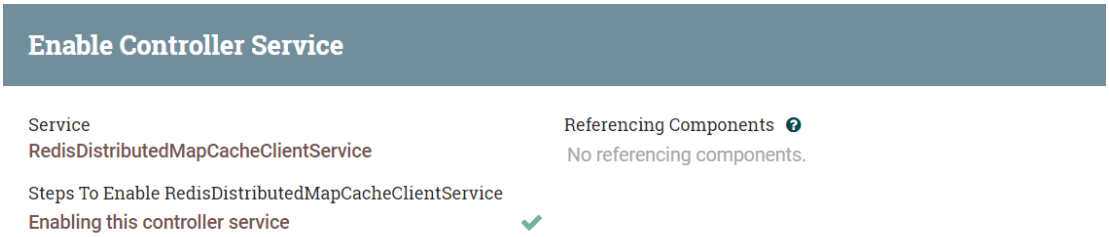

Рисунок 13.8 RedisDistributedMapCacheClientService

Добавление процессоров `GenerateFlowFile` и `PutDistributedMapCache`:

1) Для добавления процессоров наведите курсор на `Processors` и перетащите на поле:

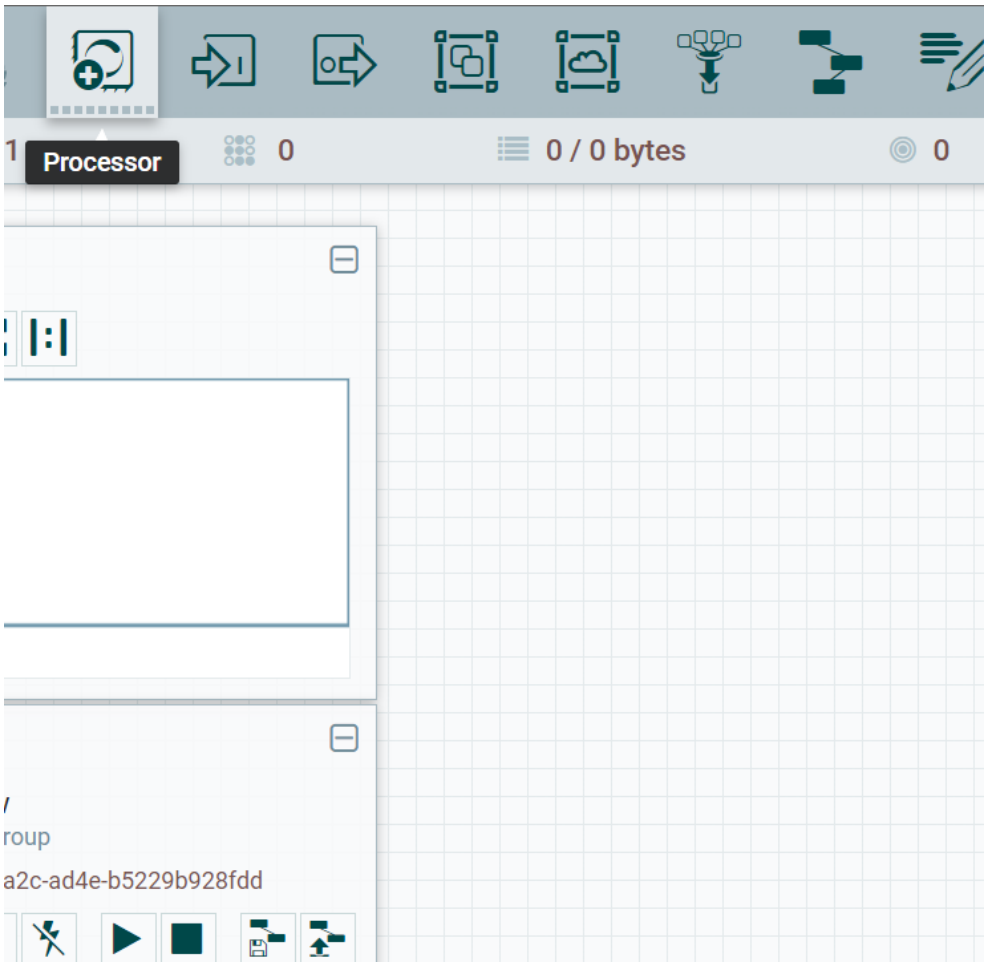

Рисунок 13.9 Processors

2) В открывшемся диалоговом окне `Add Processor` в поле `Filter` введите `GenerateFlowFile` → `Add`:

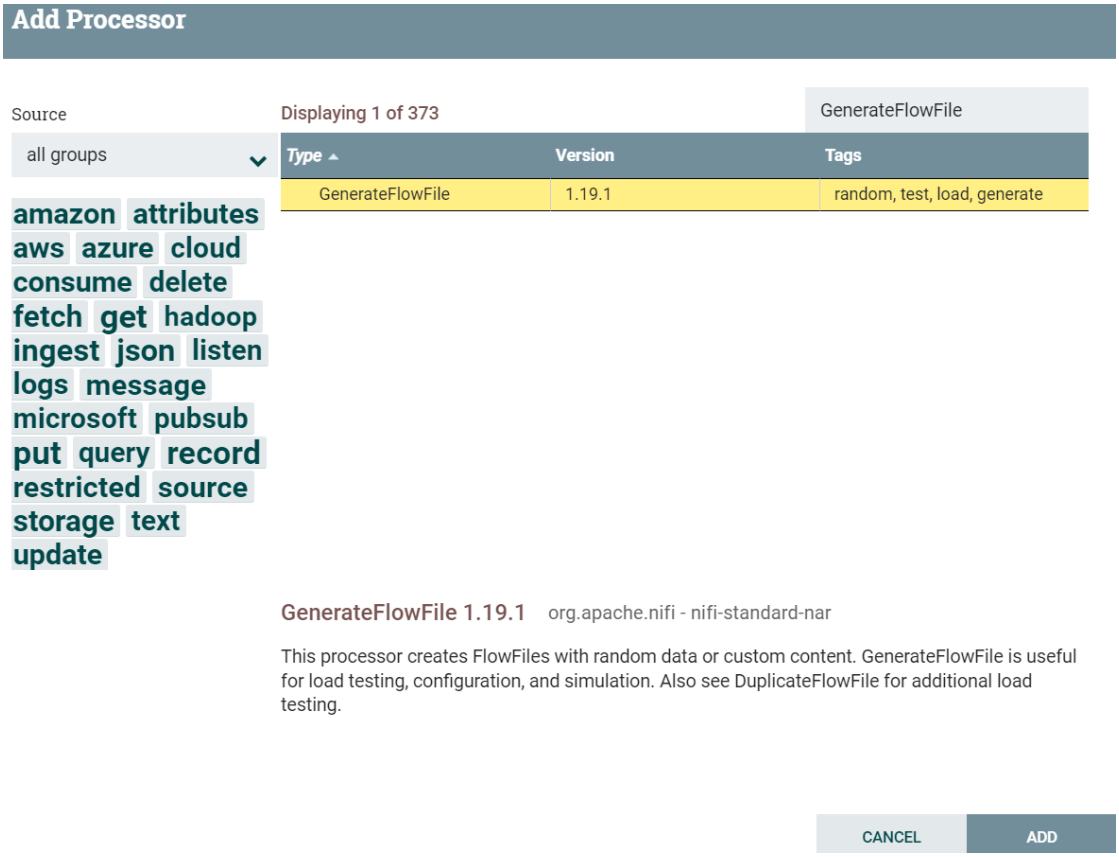

Рисунок 13.10 GenerateFlowFile → Add

3) Аналогичным образом добавьте процессор `PutDistributedMapCache`:

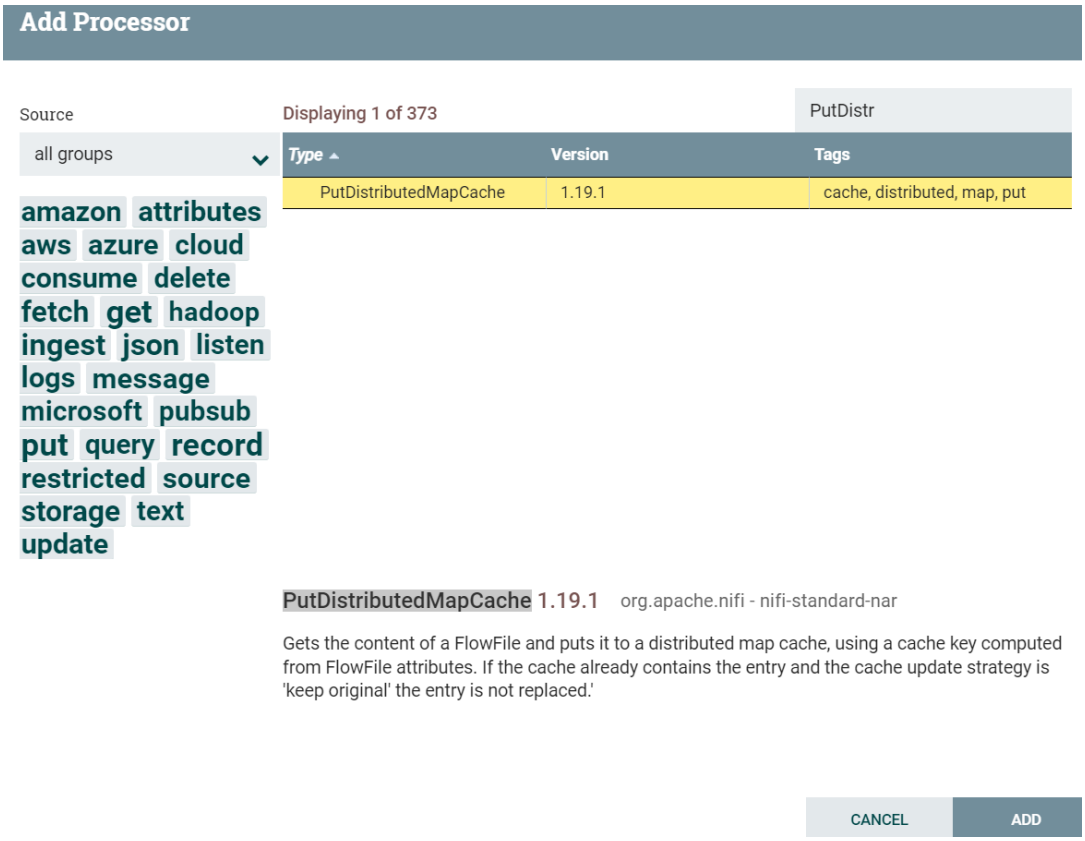

Рисунок 13.11 PutDistributedMapCache

4) Нажмите на стрелочку процессора `GenerateFlowFile` и тянем до `PutDistributedMapCache`:

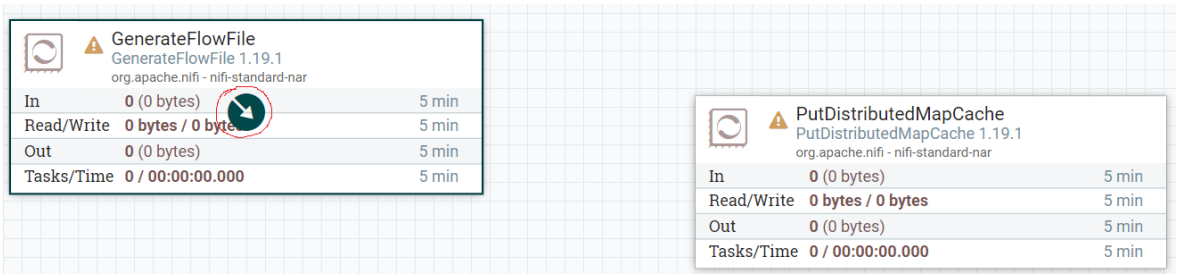

Рисунок 13.12 PutDistributedMapCache

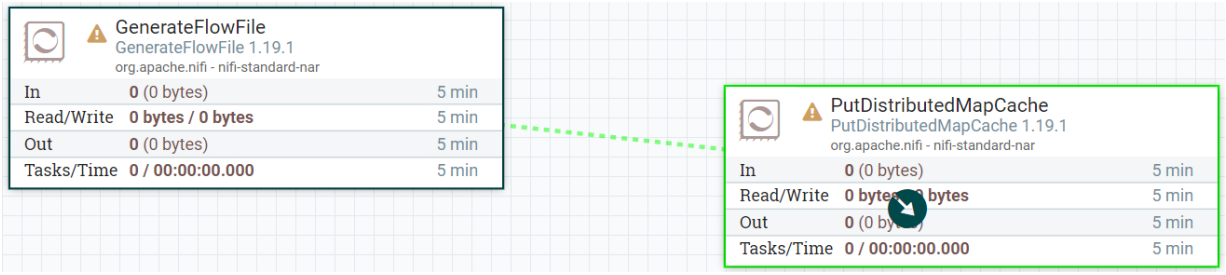

Рисунок 13.13 PutDistributedMapCache

5) В открывшемся диалоговом окне выберите `Create Connection → Add`.

13.5. Конфигурация процессоров GenerateFlowFile и PutDistributedMapCache

Конфигурация процессора `GenerateFlowFile`:

1) Двойной клик (либо ПКМ), затем `Configure` на процессоре `GenerateFlowFile`.
2) Вкладка `Properties`, в поле `Custom Text` введите `hello from nifi`, нажмите `Apply`:

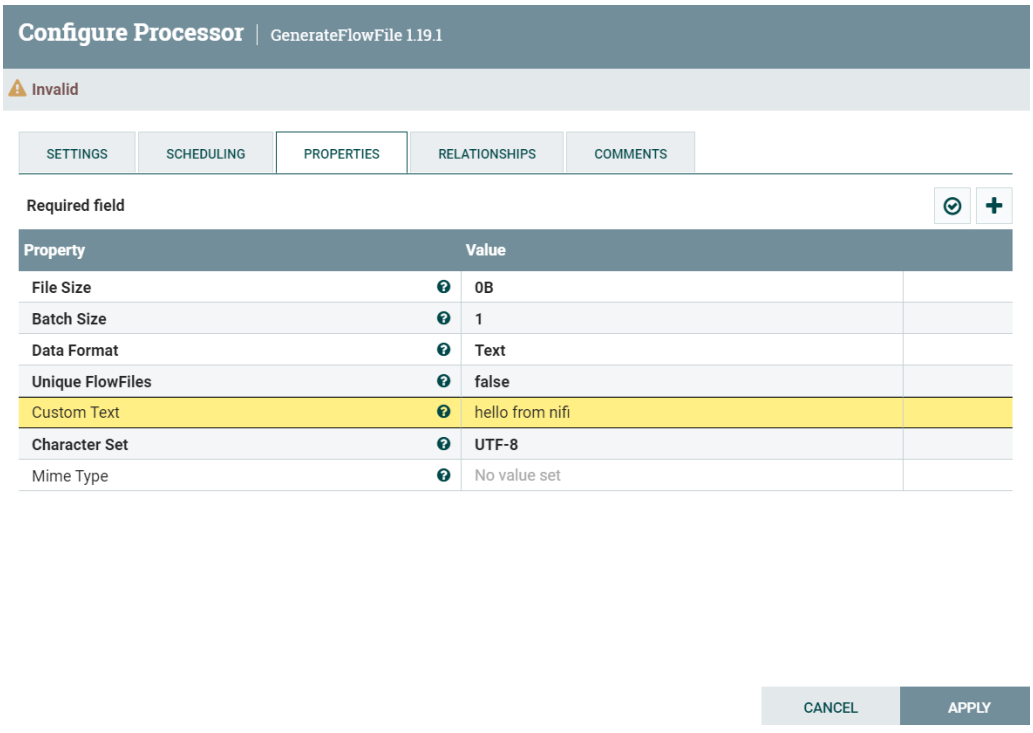

Рисунок 13.14 Properties

Конфигурация процессора `PutDistributedMapCache`:

1) Двойной клик (либо правый клик → `Configure`) на процессоре `PutDistributedMapCache`.
2) Вкладка `Properties`, в значения в следующие поля:
   - **Cache Entry Identifier**: nifi-key;
   - **Distributed Cache Service**: из выпадающего списка выбираем **RedisDistributedMapCacheClientService.

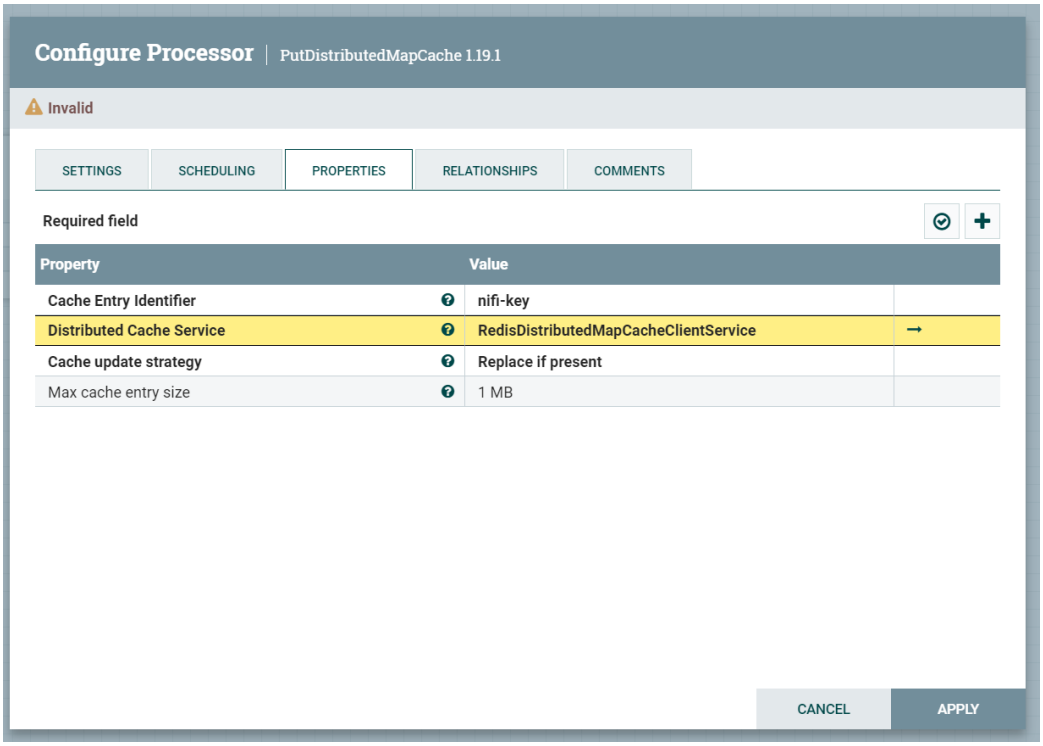

Рисунок 13.15 Properties

3) Вкладка `Relationships`, выберите галочки `terminate` в секциях `failure` и `success`:

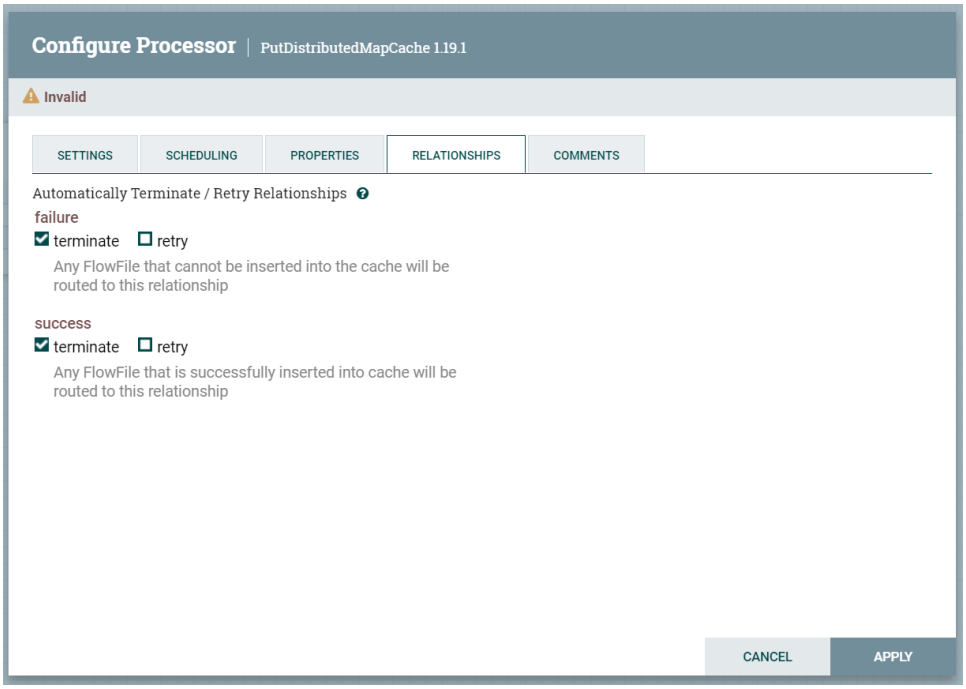

Рисунок 13.16 Relationships

13.6. Запуск процессоров GenerateFlowFile и PutDistributedMapCache

1) Нажмите ПКМ на процессоре `PutDistributedMapCache`, нажмите `Start`.
2) Нажмите ПКМ на процессоре `GenerateFlowFile`, затем `Run Once`.
3) Нажмите ПКМ на самом поле (вне процессоров), затем `Refresh`.
4) Убедитесь про отсутствие ошибок в правом верхнем угле процессора `PutDistributedMapCache`:

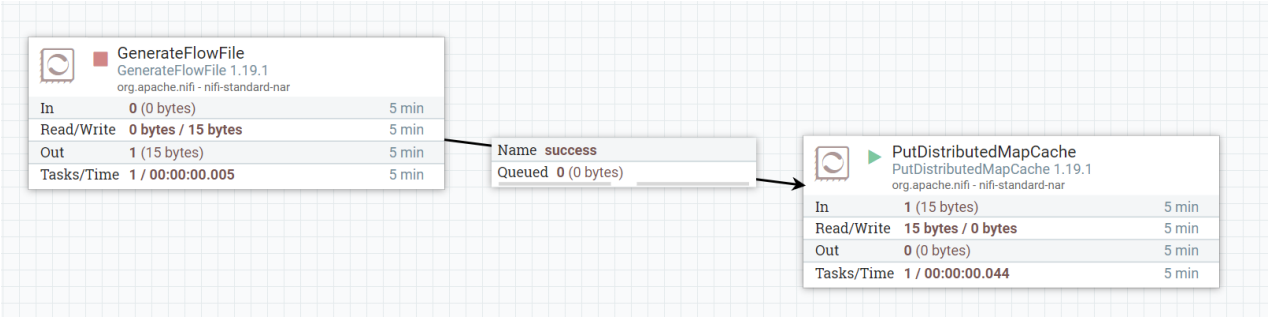

Рисунок 13.17 Relationships

13.7. Тестирование Redis

1) Используйте `Lens` заходим в оболочку (`shell`) пода `zif-redis-infra-master-0.
2) Введите команды:

```
**redis-cli
AUTH "\<пароль администратора Redis>"
KEYS \* →** в результате должен быть выведен ключ **nifi-key
GET nifi-key →** значение ключа должно быть **"hello from nifi"
```

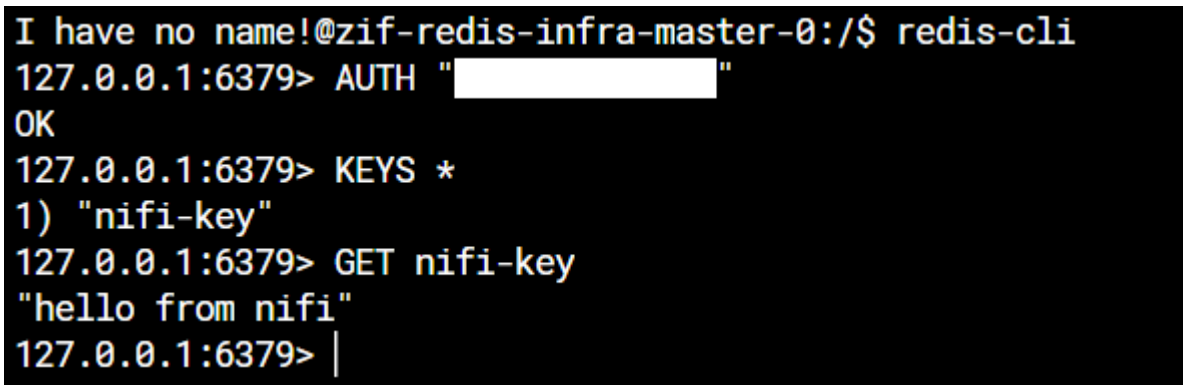

Рисунок 13.18 Тестирование Redis