ИНСТАНС KAFKA

Привет, мы подготовили для вас небольшую инструкцию, чтобы вы смогли развернуть инстанс kafka для работы локально с очередью. Дальше этот этап пригодится вам для выполнения практического задания

Инструкция состоит из 4-х последовательных шагов:

- Шаг 1. Docker compose;
- Шаг 2. Инсталляция образа Kafka
- Шаг 3. Работа в терминале.
- Шаг 4. Настройка Kafka скриптов

Начнем по порядку.

Шаг 1. Docker compose

Развернем первый инстанс Kafka. Для разворачивания будем использовать Docker compose, а также образ Kafka в KRaft моде. Работать будем в контейнере Kafka.

Вы уже работали docker compose, например, в модуле airflow. Если вы пропустили задания модуля airflow воспользуйтесь инструкцией и особенно обратите внимание на docker compose

Напомним, что Kafka использует собственный протокол поверх TCP для работы с огромным количеством API по принципу «запрос-ответ».

Дополнительные инструменты для реализации шага: docker compose.
В предыдущих уроках вы установили докер. Убедиться что он установлен успешно можно командой:

```
sudo docker run hello-world
```

Также в этом **файле** вы найдете полезные ссылки и команды для работы с Kafka.

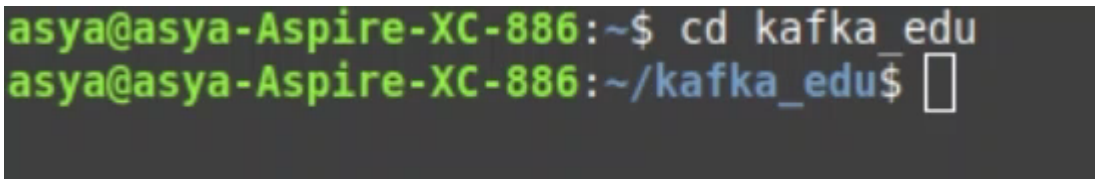
Шаг 2. Образ Kafka

Далее понадобится образ Kafka в Docker, на его основе будет составлен docker-compose файл.

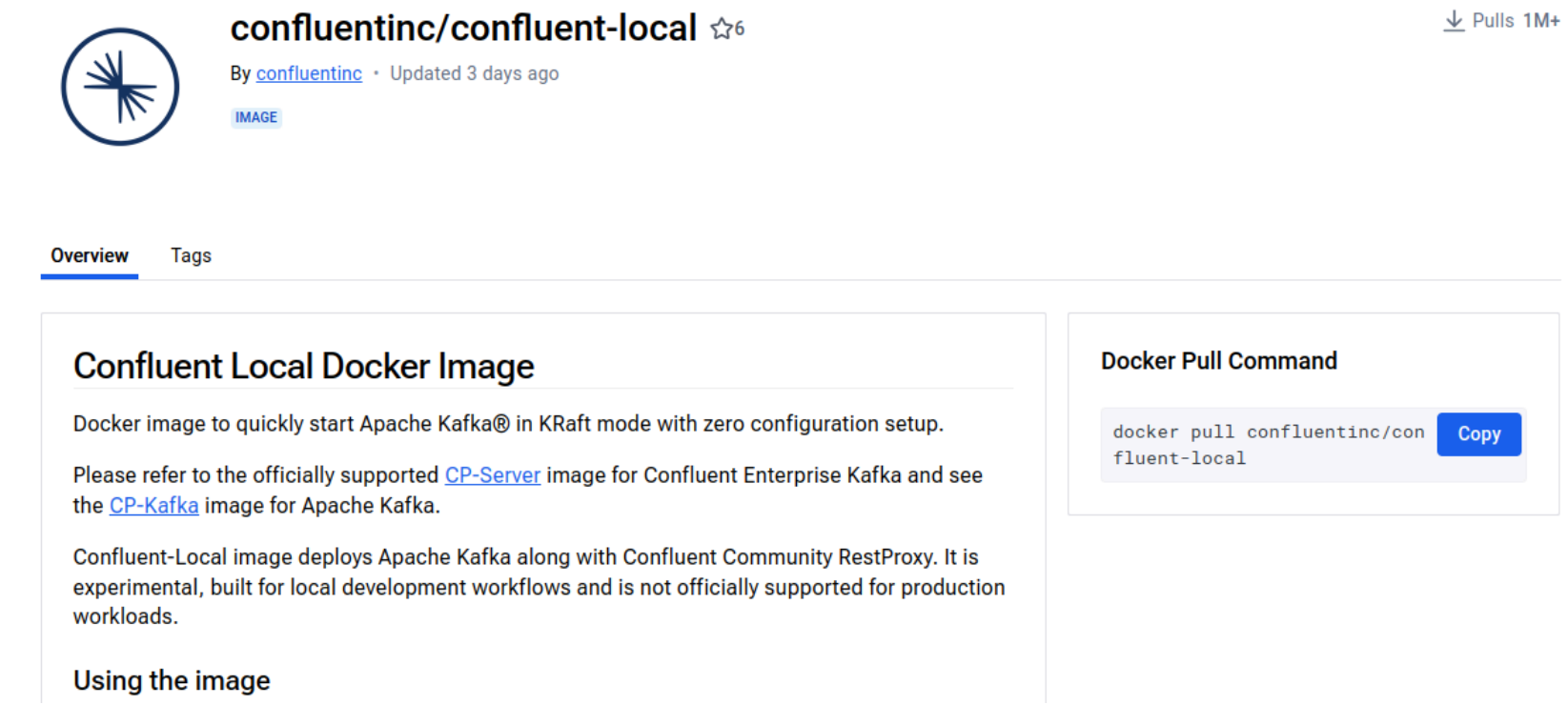
Дополнительные инструменты для реализации шага:

- docker и docker compose установлены
- образ Kafka Ссылка на образ Kafka необходим для быстрого запуска. Apache Kafka в режиме KRaft Работает без какой-либо настройки конфигурации. Установка через обычный pull не подойдет для наших задач - порт может быть занят, entrypoint нужно задавать.

Развернем первый инстанс Apache Kafka. Для этого создадим директорию для работы с Kafka и перейдем в нее, для этого в терминале:

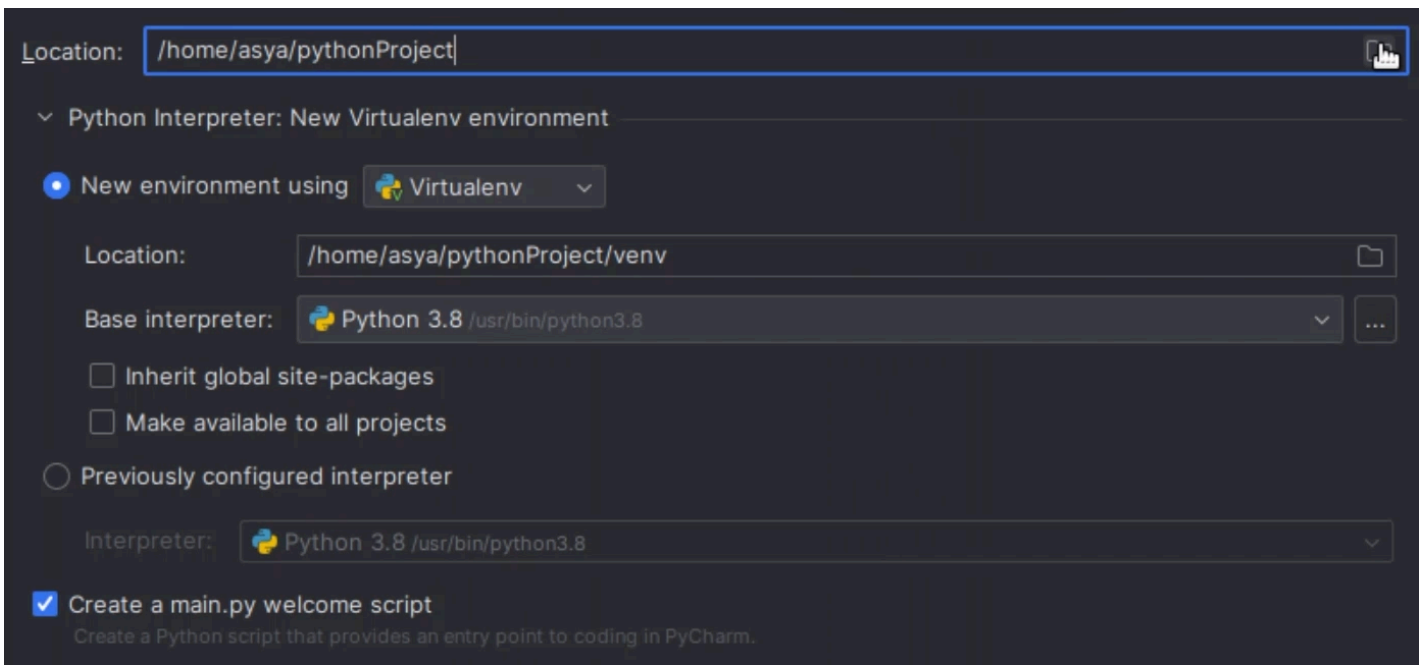


Теперь приступим к написанию Docker compose файла, в котором будем использовать образ для Kafka:



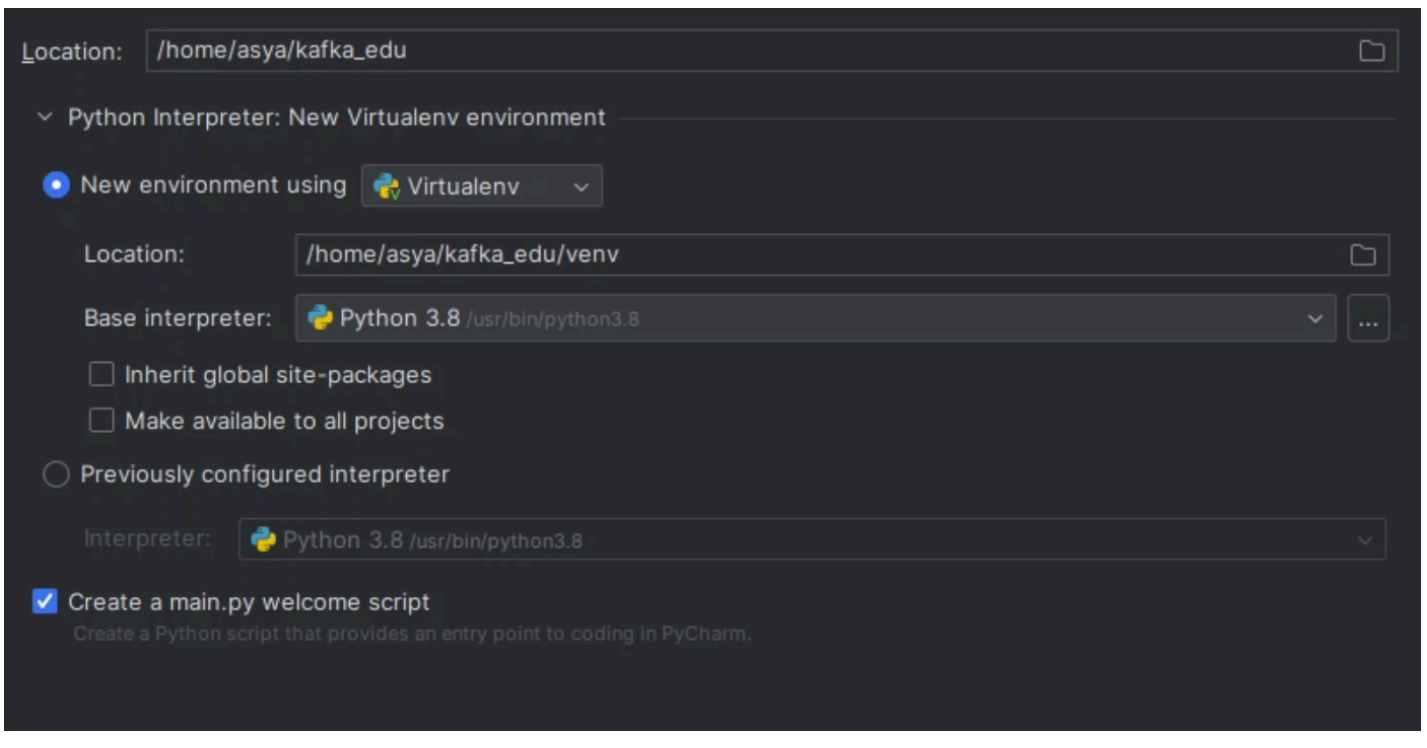
[Ссылка на образ Kafka]

Теперь открываем PyCharm и создаем в нем новый проект. Это нужно для того, чтобы создать виртуальную среду. Как обычно указываем путь к проекту и выбираем настройки как показано ниже на скриншоте:



Выбираем ранее созданную в первом шаге директорию, здесь это «kafka_edu»:

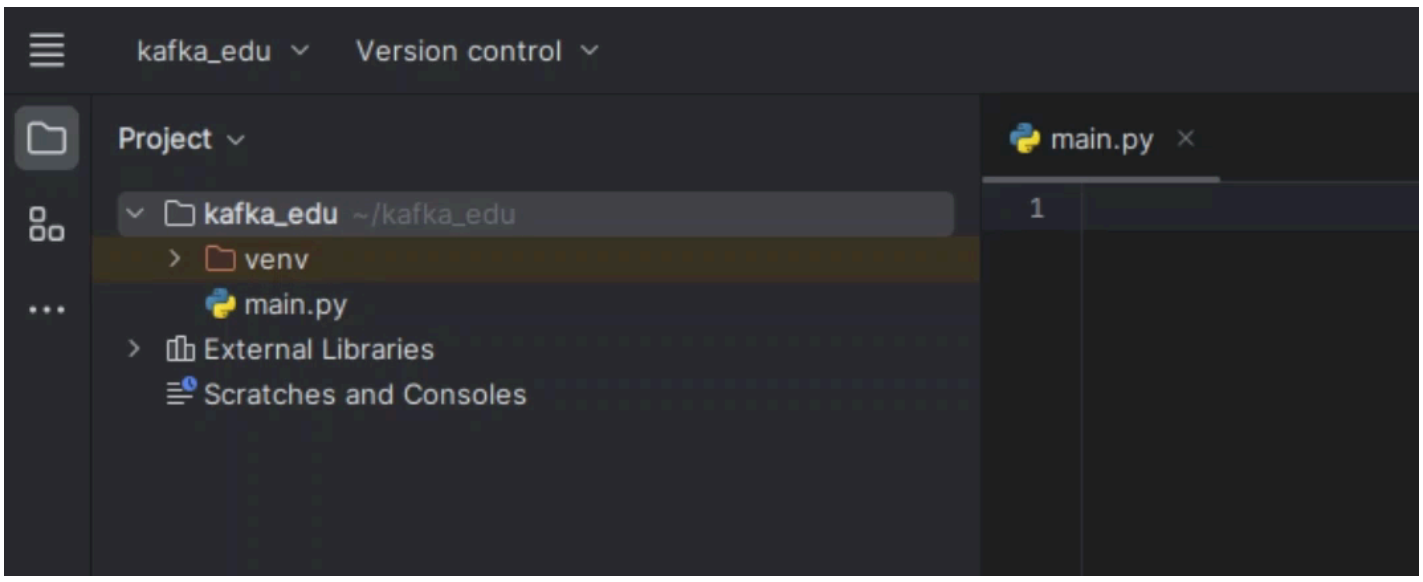
Здесь мы создаем виртуальную среду:



Если вы используете не PyCharm, то создайте ее самостоятельно. Как это сделать указано [по ссылке.]

Создаем проект. Далее возвращаемся к PyCharm.

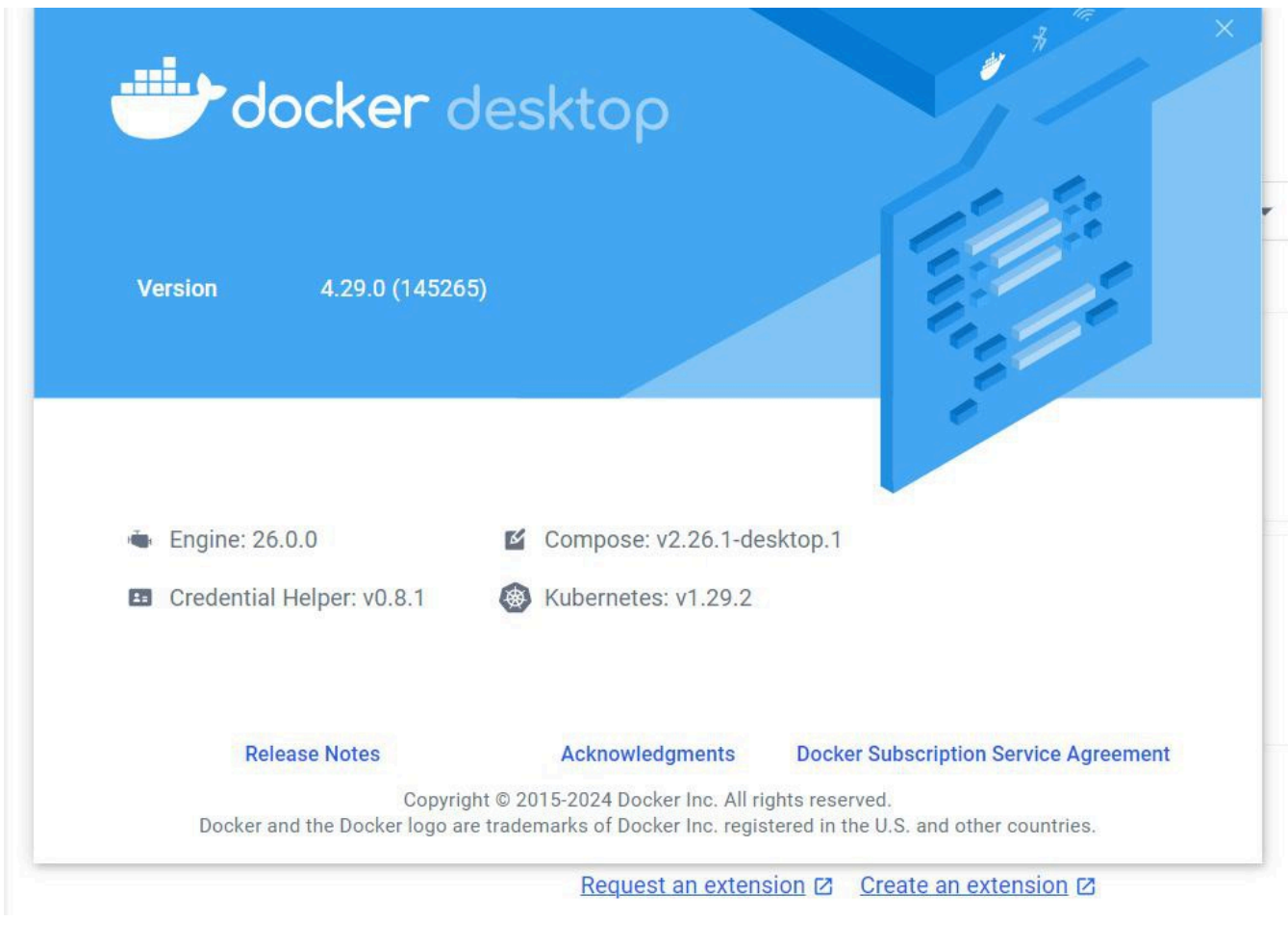
Из файла main.py можно все удалить, он не понадобится. Важно, что у нас есть виртуальная среда, как показано на скриншоте ниже:



Пора приступать к созданию в PyCharm файла «**docker-compose.yml**». Для этого опишем в нем Docker compose-структуру.

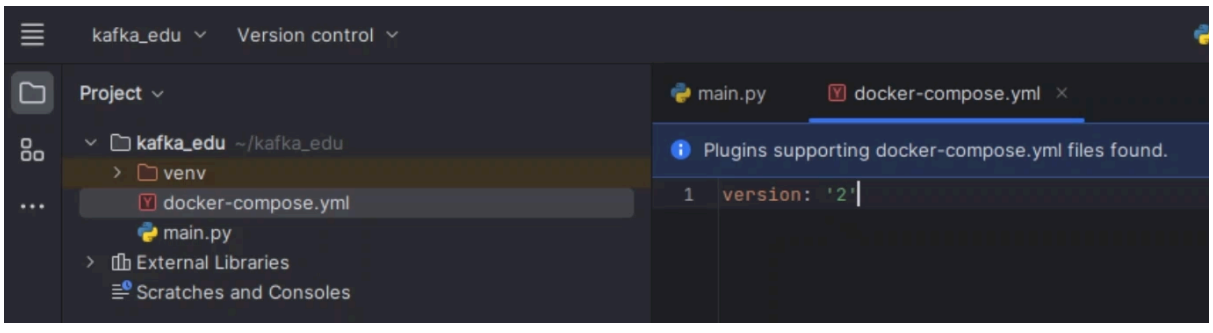
Сначала нам нужно определить версию Docker с которой мы работаем.

Чтобы узнать версию, нужно перейти в терминал и набрать docker compose –version или открыть информацию о программе - About Docker Desktop. В зависимости от версии делаем следующее:



Отлично, версия на июль 2024: **2.26.1.**

Возвращаемся в PyCharm, открываем файл *docker-compose.yml* и пишем `version: '2'` , как показано на скриншоте ниже.



Дальше мы будем описывать сервисы Kafka для работы с ними. У нас будет контейнер для Kafka. Пропишем, какие образы у нас используются, берем их на ранее указанном сайте docker-hub.

Копируем строчку docker pull, удаляем docker pull, оставляя лишь название image, который находится по ссылке Ссылка на образ Kafka и присваиваем параметру image в docker-compose.

Теперь напишем container_name, для этого прописываем порты для подключения **9092**. В данной сборке вы будите работать только с этим портом

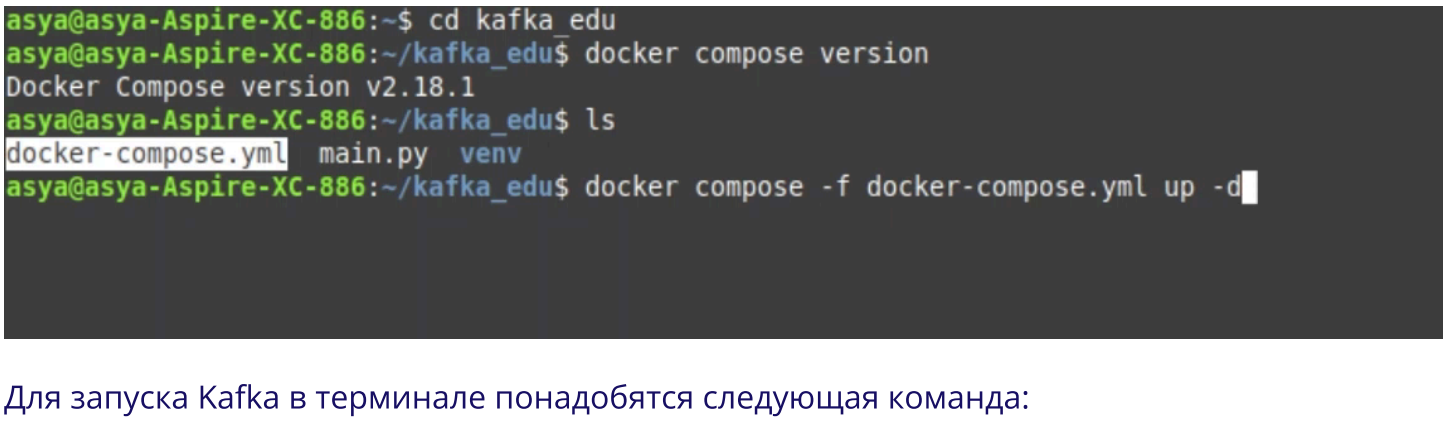
Обратите внимание: по умолчанию, также Kafka использует порт «9091».

Для Kafka необходимо использовать в docker-контейнере переменные среды. Чтобы они передались в docker-контейнер, пишем следующее:

```
version: '2'

services:

    kafka:

        image: “confluentinc/confluent-local:7.6.2”

        container_name: kafka

        ports:

            - "9092:9092"

        environment:

            KAFKA_ADVERTISED_HOST_NAME: localhost
```

Шаг 3. Работа в терминале

Docker compose-файл готов. Мы можем использовать его. Возвращаемся в терминал, где находится docker-файл. Чтобы его запустить пишем следующее:



Для запуска Kafka в терминале понадобятся следующая команда:

```
docker compose -f docker-compose.yml up -d
```

Если вы назвали файл как-то иначе, то с помощью параметра «-f» можно передать название файла.

Обязательно проверьте, что на этом этапе все работает корректно.

Чек-лист проверки работоспособности контейнера:

```
asya@asya-Aspire-XC-886:~$ cd kafka_edu
asya@asya-Aspire-XC-886:~/kafka_edu$ docker compose version
Docker Compose version v2.18.1
asya@asya-Aspire-XC-886:~/kafka_edu$ ls
docker-compose.yml  main.py  venv
asya@asya-Aspire-XC-886:~/kafka_edu$ docker compose -f docker-compose.yml up -d
[+] Building 0.0s (0/0)
[+] Running 3/3
 ✔ Network kafka_edu_default  Created
 ✔ Container kafka            Started
 ✔ Container zookeeper        Started
asya@asya-Aspire-XC-886:~/kafka_edu$ docker ps
CONTAINER ID   IMAGE                    COMMAND                  CREATED         STATUS         PORTS
              NAMES
a2038448c23d   wurstmeister/zookeeper   "/bin/sh -c '/usr/sb…"   8 seconds ago   Up 8 seconds   22/tcp, 2888/tcp, 3888/t
1->2181/tcp   zookeeper
f61a38164238   wurstmeister/kafka       "start-kafka.sh"         8 seconds ago   Up 8 seconds   0.0.0.0:9092->9092/tcp,
              kafka
asya@asya-Aspire-XC-886:~/kafka_edu$
```

1. docker ps

Оба в статусе «Up», все работает.

2. Для работы с Kafka нам необходимо зайти в контейнер с Kafka, где мы потренируемся создавать топики:

```
docker exec -it kafka /bin/bash
```

В контейнере

```
asya@asya-Aspire-XC-886:~/kafka_edu$ docker exec -it kafka /bin/bash
root@f61a38164238:/# ls
bin  boot  dev  etc  home  kafka  lib  lib64  media  mnt  opt  proc  root  run  sbin  srv  sys  tmp  usr  var
root@f61a38164238:/# cd opt
root@f61a38164238:/opt#
```

Шаг 4. Kafka скрипты

Сейчас мы находимся в корне, в нем есть папка bin. В ней лежат скрипты для работы с producer, consumer и для управления топиками.

```
cd /bin
```

```
root@f61a38164238:/opt/kafka_2.13-2.8.1/bin#
```

Создадим первый топик.

Для этого используем **kafka-topics.sh**. Используем команду «Create»:

```
kafka-topics  --create --bootstrap-server localhost:9092
```

```
--replication-factor 1  --partitions 1  --topic kafka_edu
```

Передаем фактор репликации (количество копий у каждой партиции в каждом топике на разных брокерах). Поскольку у нас один брокер, replication factor может быть не более единицы:

```
asya@asya-Aspire-XC-886:~/kafka_edu$ docker exec -it kafka /bin/bash
root@f61a38164238:/# ls
bin  boot  dev  etc  home  kafka  lib  lib64  media  mnt  opt  proc  root  run  sbin  srv  sys  tmp  usr  var
root@f61a38164238:/# cd opt
root@f61a38164238:/opt# ls
kafka  kafka_2.13-2.8.1  overrides
root@f61a38164238:/opt# cd kafka_2.13-2.8.1
root@f61a38164238:/opt/kafka_2.13-2.8.1# ls
LICENSE  NOTICE  bin  config  libs  licenses  logs  site-docs
root@f61a38164238:/opt/kafka_2.13-2.8.1# cd bin
root@f61a38164238:/opt/kafka_2.13-2.8.1/bin# ls
connect-distributed.sh        kafka-consumer-perf-test.sh          kafka-producer-perf-test.sh         kafka-verifiable-producer.sh
connect-mirror-maker.sh       kafka-delegation-tokens.sh           kafka-reassign-partitions.sh        trogdor.sh
connect-standalone.sh         kafka-delete-records.sh              kafka-replica-verification.sh       windows
kafka-acls.sh                 kafka-dump-log.sh                    kafka-run-class.sh                  zookeeper-security-migration.sh
kafka-broker-api-versions.sh  kafka-features.sh                    kafka-server-start.sh               zookeeper-server-start.sh
kafka-cluster.sh              kafka-leader-election.sh             kafka-server-stop.sh                zookeeper-server-stop.sh
kafka-configs.sh              kafka-log-dirs.sh                    kafka-storage.sh                    zookeeper-shell.sh
kafka-console-consumer.sh     kafka-metadata-shell.sh              kafka-streams-application-reset.sh
kafka-console-producer.sh     kafka-mirror-maker.sh                kafka-topics.sh
kafka-consumer-groups.sh      kafka-preferred-replica-election.sh  kafka-verifiable-consumer.sh
root@f61a38164238:/opt/kafka_2.13-2.8.1/bin# kafka-topics.sh --create --zookeeper zookeeper:2181 --replication-factor 1 --partitions 1 --topic kafka_edu
WARNING: Due to limitations in metric names, topics with a period ('.') or underscore ('_') could collide. To avoid issues it is best to use either, but n
ot both.
Created topic kafka_edu.
root@f61a38164238:/opt/kafka_2.13-2.8.1/bin#
```

Топик успешно создан. Чтобы посмотреть, какие есть топики, используется команда «List»:

```
kafka-topics  --list --bootstrap-server localhost:9092
```

Напомним, что существует команда «describe», которая показывает подробную информацию о топике:

```
Topic: kafka_edu        TopicId: q-XMAhsNRx6mDuhvBYhEAg PartitionCount: 1       ReplicationFactor: 1    Configs:
        Topic: kafka_edu        Partition: 0    Leader: 1001    Replicas: 1001  Isr: 1001
root@f61a38164238:/opt/kafka_2.13-2.8.1/bin#
```

Среди сервисов есть сервис «Kafka-console-producer». Этот продюсер умеет работать с Kafka-брокером, если передать следующие параметры:

```
root@f61a38164238:/opt/kafka_2.13-2.8.1/bin# kafka-console-producer.sh --broker-list kafka:9092 --topic kafka_edu
>
```

Мы можем отправлять сообщения. Например:

```
>Hello world
>
```

Вы видите, что ничего не происходит, потому что сообщения нужно считывать. Для этого мы откроем новое окошко и попробуем еще раз:

```
>^Croot@f61a38164238:/opt/kafka_2.13-2.8.1/bin# kafka-console-producer.sh --broker-list kafka:9092 --topic kafka_edu
>Hello world
>
```

Теперь попробуем открыть новое окошко терминала для написания consumer. Мы так же заходим в контейнер, переходим в директорию «bin», смотрим, где находится Kafka и ищем скрипты:

```
asya@asya-Aspire-XC-886:~/kafka_edu$ docker exec -it kafka /bin/bash
root@f61a38164238:/# cd opt/
root@f61a38164238:/opt# ls
kafka  kafka_2.13-2.8.1  overrides
root@f61a38164238:/opt# cd kafka_2.13-2.8.1/bin
root@f61a38164238:/opt/kafka_2.13-2.8.1/bin# ls
connect-distributed.sh        kafka-consumer-perf-test.sh         kafka-producer-perf-test.sh         kafka-verifiabl
connect-mirror-maker.sh       kafka-delegation-tokens.sh          kafka-reassign-partitions.sh        trogdor.sh
connect-standalone.sh         kafka-delete-records.sh             kafka-replica-verification.sh       windows
kafka-acls.sh                 kafka-dump-log.sh                   kafka-run-class.sh                  zookeeper-secur
kafka-broker-api-versions.sh  kafka-features.sh                   kafka-server-start.sh               zookeeper-serve
kafka-cluster.sh              kafka-leader-election.sh            kafka-server-stop.sh                zookeeper-serve
kafka-configs.sh              kafka-log-dirs.sh                   kafka-storage.sh                    zookeeper-shell
kafka-console-consumer.sh     kafka-metadata-shell.sh             kafka-streams-application-reset.sh
kafka-console-producer.sh     kafka-mirror-maker.sh               kafka-topics.sh
kafka-consumer-groups.sh      kafka-preferred-replica-election.sh kafka-verifiable-consumer.sh
root@f61a38164238:/opt/kafka_2.13-2.8.1/bin#
```

Нам нужно запустить «kafka-console-consumer». Для написания consumer передаем следующие параметры:

```
asya@asya-Aspire-XC-886:~/kafka_edu$ docker exec -it kafka /bin/bash
root@f61a38164238:/# cd opt/
root@f61a38164238:/opt# ls
kafka  kafka_2.13-2.8.1  overrides
root@f61a38164238:/opt# cd kafka_2.13-2.8.1/bin
root@f61a38164238:/opt/kafka_2.13-2.8.1/bin# ls
connect-distributed.sh        kafka-consumer-perf-test.sh         kafka-producer-perf-test.sh         kafka-verifiabl
connect-mirror-maker.sh       kafka-delegation-tokens.sh          kafka-reassign-partitions.sh        trogdor.sh
connect-standalone.sh         kafka-delete-records.sh             kafka-replica-verification.sh       windows
kafka-acls.sh                 kafka-dump-log.sh                   kafka-run-class.sh                  zookeeper-secur
kafka-broker-api-versions.sh  kafka-features.sh                   kafka-server-start.sh               zookeeper-serve
kafka-cluster.sh              kafka-leader-election.sh            kafka-server-stop.sh                zookeeper-serve
kafka-configs.sh              kafka-log-dirs.sh                   kafka-storage.sh                    zookeeper-shell
kafka-console-consumer.sh     kafka-metadata-shell.sh             kafka-streams-application-reset.sh
kafka-console-producer.sh     kafka-mirror-maker.sh               kafka-topics.sh
kafka-consumer-groups.sh      kafka-preferred-replica-election.sh kafka-verifiable-consumer.sh
root@f61a38164238:/opt/kafka_2.13-2.8.1/bin# kafka-console-consumer.sh --bootstrap-server kafka:9092 --topic kafka_edu
```

Прежние сообщения не считываются. Однако, если мы попробуем передать сообщение сейчас, то оно появится в консьюмере:

```
asya@asya-Aspire-XC-886:~/kafka_edu$ docker exec -it kafka /bin/bash
root@f61a38164238:/# cd opt/
root@f61a38164238:/opt# ls
kafka  kafka_2.13-2.8.1  overrides
root@f61a38164238:/opt# cd kafka_2.13-2.8.1/bin
root@f61a38164238:/opt/kafka_2.13-2.8.1/bin# ls
connect-distributed.sh        kafka-consumer-perf-test.sh          kafka-producer-perf-test.sh         kafka-verifiabl
connect-mirror-maker.sh       kafka-delegation-tokens.sh           kafka-reassign-partitions.sh        trogdor.sh
connect-standalone.sh         kafka-delete-records.sh              kafka-replica-verification.sh       windows
kafka-acls.sh                 kafka-dump-log.sh                    kafka-run-class.sh                  zookeeper-secur
kafka-broker-api-versions.sh  kafka-features.sh                    kafka-server-start.sh               zookeeper-serve
kafka-cluster.sh              kafka-leader-election.sh             kafka-server-stop.sh                zookeeper-serve
kafka-configs.sh              kafka-log-dirs.sh                    kafka-storage.sh                    zookeeper-shell
kafka-console-consumer.sh     kafka-metadata-shell.sh              kafka-streams-application-reset.sh
kafka-console-producer.sh     kafka-mirror-maker.sh                kafka-topics.sh
kafka-consumer-groups.sh      kafka-preferred-replica-election.sh  kafka-verifiable-consumer.sh
root@f61a38164238:/opt/kafka_2.13-2.8.1/bin# kafka-console-consumer.sh --bootstrap-server kafka:9092 --topic kafka_edu
Hi
```

Чтобы прежние сообщения считались, нужно указать параметр:

```
Hi
^CProcessed a total of 1 messages
root@f61a38164238:/opt/kafka_2.13-2.8.1/bin# kafka-console-consumer.sh --bootstrap-server kafka:9092 --topic kafka_edu --from-beginning
Hello world
Hello world
Hi
```

Теперь можно продолжать отправлять сообщения, которые также будут приходить в consumer:

```
Hi
^CProcessed a total of 1 messages
root@f61a38164238:/opt/kafka_2.13-2.8.1/bin# kafka-console-consumer.sh --bootstrap-server kafka:9092 --topic kafka_edu --from-beginning
Hello world
Hello world
Hi
New msg
^CProcessed a total of 4 messages
```

После того, как мы заканчиваем работу, присылается статистика того, сколько всего было сообщений.

Мы создали наш первый топик Kafka.

Работа с топиком настроена корректно, если при отправке сообщений в одном терминале, сообщения появляются в другом терминале с запущенным консьюмером.

Проверьте себя. Работа с топиком настроена корректно, если при отправке сообщений в одном терминале, сообщения появляются в другом терминале с запущенным потребителем (consumer)

Дополнительный материал к инструкции
В этом **файле** вы найдете полезные ссылки и команды для работы с Kafka.

Удачи в обучении 😉

Если у вас остались вопросы, смело пишите в чат курса:

- ✔ **вопросы по стендам** - тегните Ивана из тех. поддержки @uprimo
- ✔ **вопросы по заданию и инструкции** - тегните спикера курса Асю Гайламазян @Bagirasya
- ✔ **вопросы обо всем на свете** - тегните куратора курса @living8life

Как вам урок? 😠 😐 🔥

Изучил, далее >