**Мой Python/Java/Spring/Go/любой клиент не подключается к моему кластеру Apache Kafka в Docker/AWS/ноутбуке моего брата. Пожалуйста помоги!**

**Описание:**

**Иллюстрированный пример подключения клиента Kafka к брокеру - 1**

**Сценарий 0: Клиент и Kafka работают на одном локальном компьютере – 4**

**Сценарий 1: Клиент и Kafka работают на разных машинах – 4**

**Сценарий 2: Kafka и клиент работают в Docker - 5**

**Сценарий 3: Kafka в Docker Compose – 8**

**Сценарий 4: Kafka в контейнере Docker с локально работающим клиентом – 9**

**Добавление нового слушателя к брокеру – 11**

**Сценарий 5: Kafka работает локально с клиентом в контейнере Docker – 12**

**Часто задаваемые вопросы - 15**

Когда клиент хочет отправить или получить сообщение от Apache Kafka®, существует два типа соединения, которые должны быть успешными:

Первоначальное подключение к брокеру (бутстрап). При этом клиенту возвращаются метаданные, включая список всех брокеров в кластере и их конечных точек подключения.

Затем клиент при необходимости подключается к одному (или нескольким) брокерам, возвращенным на первом этапе. Если брокер настроен неправильно, соединения не будут установлены.

Иногда случается так, что люди сосредотачиваются только на шаге 1, указанном выше, и отвлекаются на шаге 2. Сведения о брокере, возвращаемые на шаге 1, определяются настройкой Advertised.listeners брокера(ов) и должны быть разрешаемыми и доступными из клиентской машины.

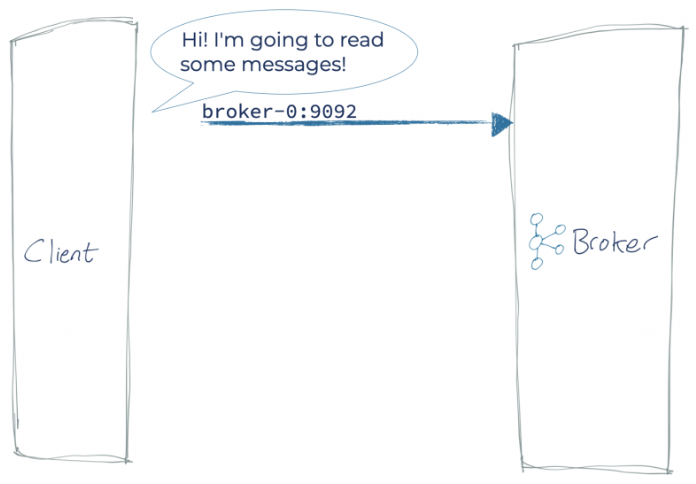
Чтобы узнать больше о протоколе, см. документацию, а также предыдущую статью, которую я написал. Если основы протокола — последнее, что вас интересует, и вы просто хотите писать приложения с помощью Kafka, вам следует попробовать Confluent Cloud. Это полностью управляемый сервис Apache Kafka в облаке, без конфигурации Advertised.listeners, о которой вам стоит беспокоиться!

Ниже я использую клиент, подключающийся к Kafka, в различных вариантах топологии развертывания. Он написан на Python с librdkafka (confluent\_kafka), но этот принцип применим к клиентам на всех языках. Вы можете найти код на GitHub. Это очень просто и служит лишь для иллюстрации процесса подключения. Это упрощено для ясности за счет хорошего кодирования и функциональности.

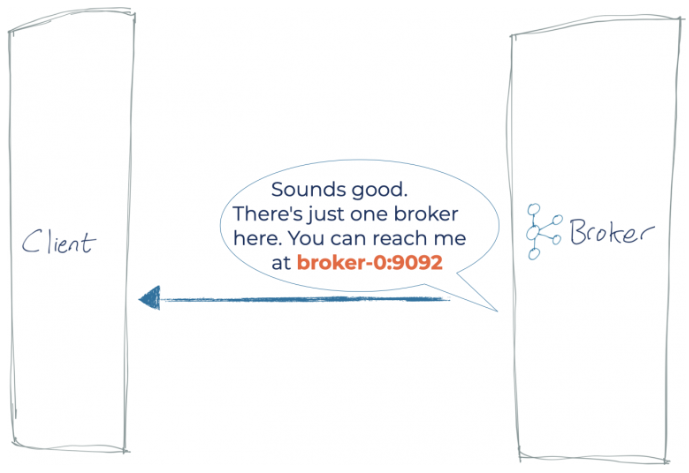
**Иллюстрированный пример подключения клиента Kafka к брокеру**

Предположим, у нас есть два сервера. На одном находится наш клиент, а на другом — единственный брокер нашего кластера Kafka (забудьте на мгновение, что в кластерах Kafka обычно есть минимум три брокера).

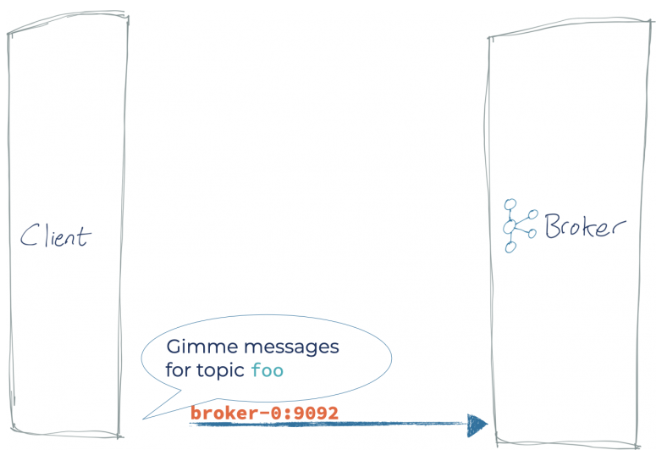
1. Клиент инициирует соединение с сервером(ами) начальной загрузки, который является одним (или несколькими) брокерами в кластере.



1. Брокер возвращает метаданные, в том числе хост и порт, по которым можно связаться со всеми брокерами в кластере.

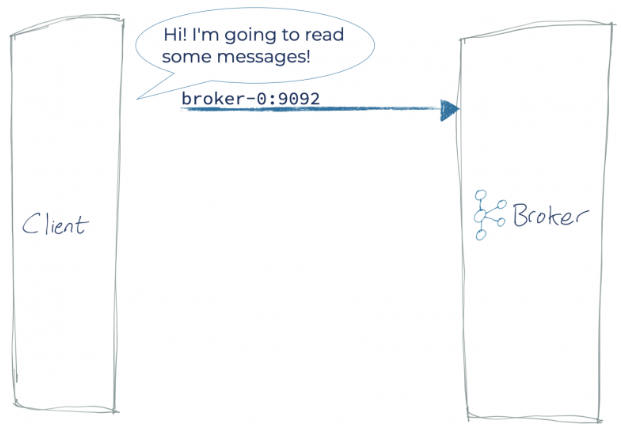


1. Этот список затем используется клиентом для всех последующих подключений для создания или потребления данных. Адрес, используемый при начальном соединении, предназначен просто для того, чтобы клиент мог найти сервер начальной загрузки в кластере из n брокеров, от которого клиенту затем предоставляется текущий список всех брокеров. Таким образом, клиенту не нужно постоянно знать список всех брокеров. Причина, по которой клиенту нужны сведения обо всех брокерах, заключается в том, что он будет напрямую подключаться к одному или нескольким брокерам, в зависимости от того, у кого есть данные для тематического раздела, с которым он хочет взаимодействовать.

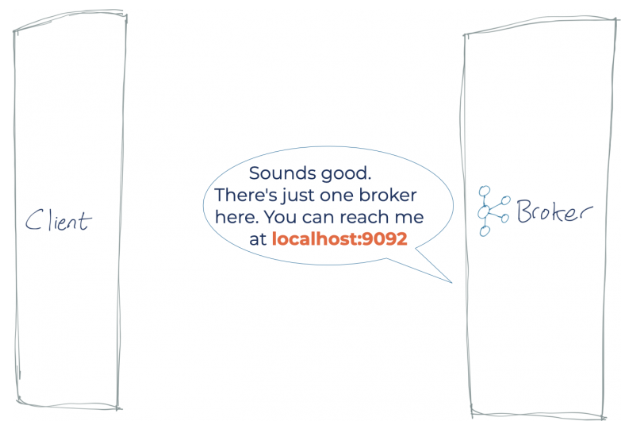


Что часто идет не так, так это то, что брокер неправильно настроен и возвращает адрес (Advertised.listeners), по которому клиент не может правильно подключиться к брокеру. В этом случае временная шкала выглядит следующим образом:

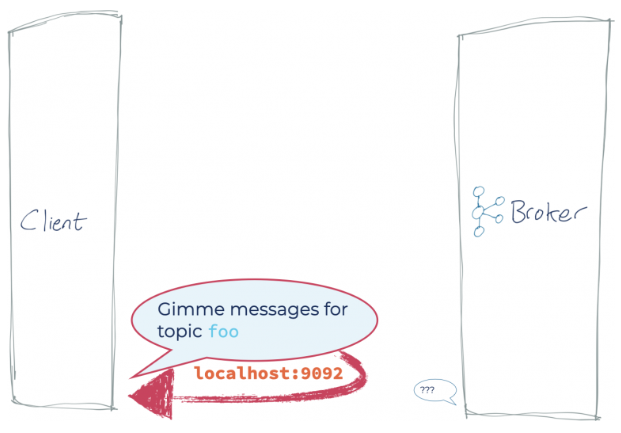
1. Клиент инициирует соединение с сервером(ами) начальной загрузки, который является одним (или несколькими) брокерами в кластере.



1. Брокер возвращает клиенту неправильное имя хоста



1. Затем клиент пытается подключиться к этому неправильному адресу, но затем терпит неудачу (поскольку брокера Kafka нет на клиентском компьютере, на что указывает localhost)



В этой статье мы рассмотрим некоторые распространенные сценарии и объясним, как исправить каждый из них.

**Всего один брокер?**

Во всех этих примерах используется только один брокер, что хорошо для «песочницы», но совершенно бесполезно для чего-либо, приближенного к реальной среде. На практике в вашем кластере должно быть как минимум три брокера. Ваш клиент будет загружаться с одним (или несколькими) из них, и этот брокер будет возвращать клиенту метаданные каждого из брокеров в кластере.

**Сценарий 0: Клиент и Kafka работают на одном локальном компьютере.**

В этом примере я использую Confluent Platform на своем локальном компьютере, но вы также можете запустить ее в любом другом UI Kafka, который вам нужен.

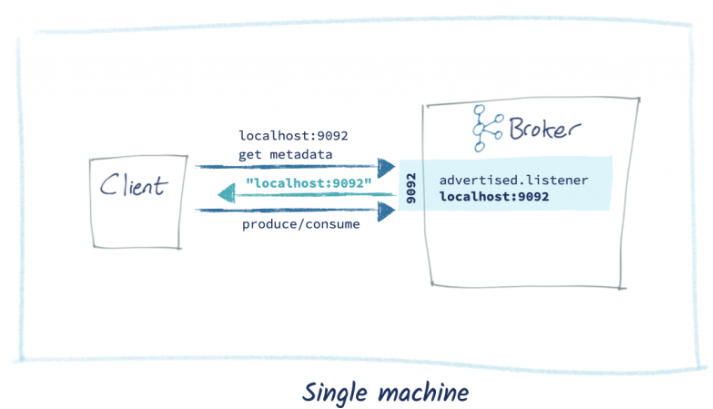
$ confluent local status kafka

…

kafka is [UP]

zookeeper is [UP]

Мой клиент Python подключается с настройкой сервера начальной загрузки localhost:9092.

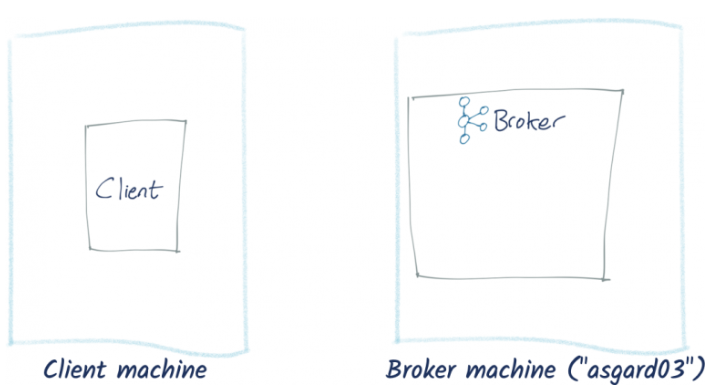


Это работает отлично:

Примечание. Возвращаемые метаданные брокера — 192.168.10.83, но поскольку это IP-адрес моего локального компьютера, он работает нормально.

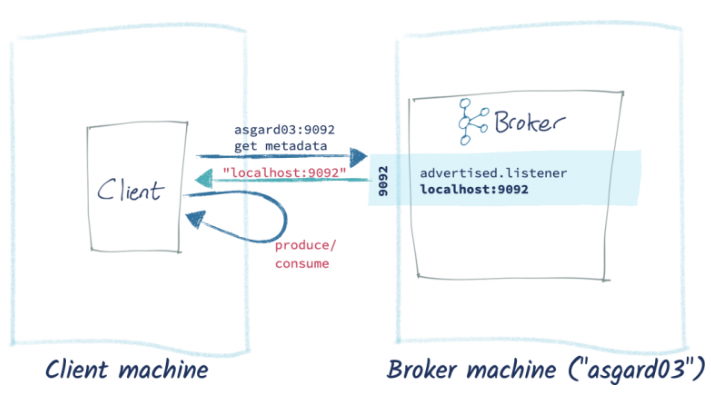
**Сценарий 1: Клиент и Kafka работают на разных машинах**

Теперь проверим подключение к брокеру Kafka, работающему на другой машине. Это может быть машина в вашей локальной сети или, возможно, работающая в облачной инфраструктуре, такой как Amazon Web Services (AWS), Microsoft Azure или Google Cloud Platform (GCP).



В этом примере мой клиент работает на моем ноутбуке и подключается к Kafka *(работающему на другом компьютере в моей локальной сети)* под названием **asgard03**:

Первоначальное соединение выполнено успешно. Но учтите, что полученные нами BrokerMetadata показывают, что существует один брокер с именем хоста localhost. Это означает, что наш клиент будет использовать localhost, чтобы попытаться подключиться к брокеру при создании и потреблении сообщений. Это плохая новость, потому что на нашей клиентской машине на локальном хосте нет брокера Kafka (а если бы он был, вероятно, произошли бы какие-то действительно странные вещи).



И так происходит:

Так как же нам это исправить? Мы идем и говорим с нашим администратором Kafka (которым вполне можем быть мы) и исправляем server.properties на брокере(ах), чтобы Advertised.listeners правильно предоставлял имя хоста и порт, по которым клиенты могут связаться с брокером. Выше мы видели, что он возвращает localhost. Пойдем и исправим это. В файле server.properties моего брокера я беру это:

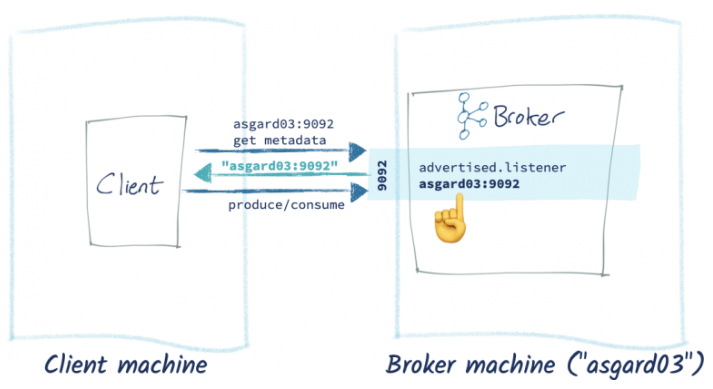
advertised.listeners=PLAINTEXT://localhost:9092

listeners=PLAINTEXT://0.0.0.0:9092

И измените конфигурацию advertised.listeners следующим образом:

advertised.listeners=PLAINTEXT://asgard03.moffatt.me:9092

listeners=PLAINTEXT://0.0.0.0:9092



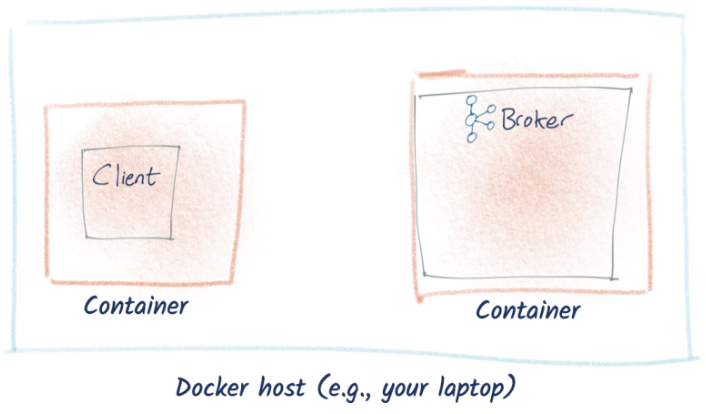
Сам прослушиватель остается неизменным (он привязывается ко всем доступным сетевым картам через порт 9092). Единственное отличие состоит в том, что этот прослушиватель скажет клиенту связаться с ним по asgard03.moffatt.me вместо localhost.

Итак, после применения этих изменений к Advertised.listener на каждом брокере и перезапуска каждого из них производитель и потребитель работают правильно:

Метаданные брокера теперь отображаются с именем хоста, которое правильно разрешается клиентом.

**Сценарий 2: Kafka и клиент работают в Docker**

Когда вы запускаете что-то в Docker, оно выполняется в контейнере в своем собственном маленьком мире. У него есть то, что кажется ему собственным именем хоста, собственным сетевым адресом и собственной файловой системой. Так, например, когда вы просите код в контейнере Docker подключиться к локальному хосту, он будет подключаться к самому себе, а не к хост-компьютеру, на котором вы его запускаете. Это привлекает людей, потому что они привыкли к тому, что их ноутбук является локальным хостом, поэтому кажется загадочным, почему код, работающий на ноутбуке, не может подключиться к локальному хосту. Но помните, что код не запускается на вашем ноутбуке. Он работает в контейнере на вашем ноутбуке.



Здесь мы начнем с простейшей перестановки и запустим Kafka и наш клиент в Docker в одной сети Docker. Сначала создайте Dockerfile, чтобы включить наш клиент Python в контейнер Docker:

**FROM** python:3

*# Мы добавим netcat, потому что это действительно полезный инструмент для устранения неполадок в сети.*

**RUN** apt-get update

**RUN** apt-get install -y netcat

*# Устанавливаем библиотеку Python Confluent Kafka*

**RUN** pip install confluent\_kafka

*# Add our script*

**ADD** python\_kafka\_test\_client.py /

**ENTRYPOINT** [ "python", "/python\_kafka\_test\_client.py"]

Создайте образ Docker:

docker build -t python\_kafka\_test\_client .

Затем создайте сеть / зукипер / кафку(брокера Kafka):

*#Создаем внутреннюю сеть со своим названием)*

docker network create rmoff\_kafka

*# Запускаем в созданной нами сети “*rmoff\_kafka*” с параметрами “*—rm” и “—d” образ зукипера

*#* “confluentinc/cp-zookeeper” c тэгом “5.5.0” и плюс создаем переменную окружения

*#* “ZOOKEEPER\_CLIENT\_PORT” со значением “2181”

docker run --network=rmoff\_kafka --rm --d --name zookeeper -e ZOOKEEPER\_CLIENT\_PORT=2181 confluentinc/cp-zookeeper:5.5.0

*# Запускаем в созданной нами сети “*rmoff\_kafka*” с параметрами “*—rm” и “—d” образ кафки

*# “*confluentinc/cp-kafka” c тэгом “5.5.0” плюс создаем переменные среды со значениями и выводим

*# ip-соединение наружу контейнера с портом “*9092”

docker run --network=rmoff\_kafka --rm --d --name broker \ *(“\” – знак переноса строки)*

-p 9092:9092 \

-e KAFKA\_BROKER\_ID=1 \

-e KAFKA\_ZOOKEEPER\_CONNECT=zookeeper:2181 \

-e KAFKA\_ADVERTISED\_LISTENERS=PLAINTEXT://localhost:9092 \

-e KAFKA\_OFFSETS\_TOPIC\_REPLICATION\_FACTOR=1 \

confluentinc/cp-kafka:5.5.0

Убедитесь, что у вас работают два контейнера: один Apache ZooKeeper™ и один брокер Kafka:

$ docker ps

IMAGE STATUS PORTS NAMES

confluentinc/cp-kafka:5.5.0 Up 32 seconds 0.0.0.0:9092->9092/tcp broker

confluentinc/cp-zookeeper:5.5.0 Up 33 seconds 2181/tcp, 2888/tcp, 3888/tcp zookeeper

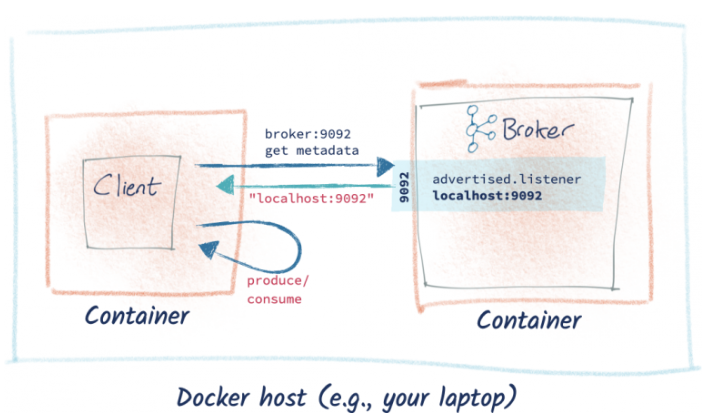
Обратите внимание, что мы создаем нашу собственную сеть Docker, в которой можно запускать эти контейнеры, чтобы мы могли взаимодействовать между ними. Несмотря на то, что они работают на Docker на моем ноутбуке, что касается каждого контейнера, они находятся на отдельных машинах и взаимодействуют через сеть.

Давайте запустим клиент и посмотрим, что произойдет:

$ docker run --network=rmoff\_kafka --rm --name python\_kafka\_test\_client \

--tty python\_kafka\_test\_client broker:9092

В возвращенных метаданных вы можете видеть, что, хотя мы изначально успешно подключаемся к брокеру, он возвращает нам localhost в качестве хоста брокера. Это означает, что производитель и потребитель потерпят неудачу, потому что они будут пытаться подключиться к нему, а localhost из клиентского контейнера сам по себе является, а не брокером.

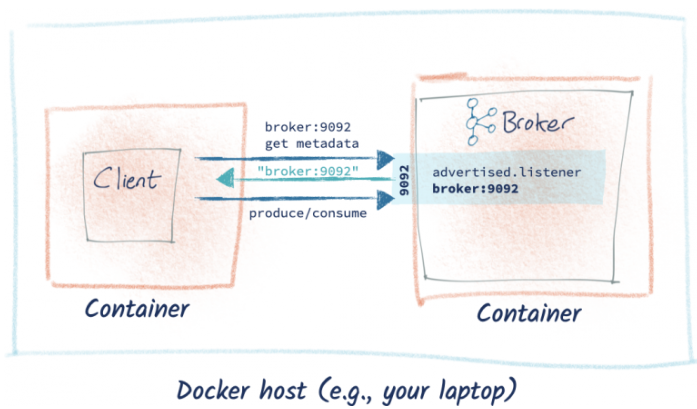


Как исправить это? Скажите брокеру, чтобы он правильно извещал своего слушателя. Поскольку имя брокера Kafka в сети —broker (унаследовано от имени его контейнера), нам нужно установить его в качестве объявленного прослушивателя и изменить:

-e KAFKA\_ADVERTISED\_LISTENERS=PLAINTEXT://localhost:9092 \

…к:

-e KAFKA\_ADVERTISED\_LISTENERS=PLAINTEXT://broker:9092 \



Итак, теперь наш брокер выглядит так:

docker stop broker

docker run --network=rmoff\_kafka --rm --detach --name broker \

-p 9092:9092 \

-e KAFKA\_BROKER\_ID=1 \

-e KAFKA\_ZOOKEEPER\_CONNECT=zookeeper:2181 \

-e KAFKA\_ADVERTISED\_LISTENERS=PLAINTEXT://broker:9092 \

-e KAFKA\_OFFSETS\_TOPIC\_REPLICATION\_FACTOR=1 \

confluentinc/cp-kafka:5.5.0

И клиент работает просто идеально:

**Сценарий 3: Kafka в Docker Compose**

Возиться с флагами командной строки для настройки контейнеров Docker через короткий промежуток времени становится довольно неприятно. Гораздо лучше использовать Docker Compose.

Сначала закройте контейнеры Docker сверху (docker rm -fbroker; docker rm -f Zookeeper), а затем создайте docker-compose.yml локально, используя этот пример (this [example](https://github.com/rmoff/kafka-listeners/blob/master/python/docker-compose_initial.yml" \t "_blank)):

version: '3.5'

networks:

rmoff\_kafka:

name: rmoff\_kafka

services:

zookeeper:

image: confluentinc/cp-zookeeper:5.5.0

container\_name: zookeeper

networks:

- rmoff\_kafka

environment:

ZOOKEEPER\_CLIENT\_PORT: 2181

broker:

image: confluentinc/cp-kafka:5.5.0

container\_name: broker

networks:

- rmoff\_kafka

depends\_on:

- zookeeper

environment:

KAFKA\_BROKER\_ID: 1

KAFKA\_ZOOKEEPER\_CONNECT: zookeeper:2181

KAFKA\_ADVERTISED\_LISTENERS: PLAINTEXT://broker:9092

KAFKA\_OFFSETS\_TOPIC\_REPLICATION\_FACTOR: 1

client:

image: python\_kafka\_test\_client

container\_name: python\_kafka\_test\_client

networks:

- rmoff\_kafka

depends\_on:

- broker

entrypoint:

- bash

- -c

- |

echo 'Giving Kafka a bit of time to start up…'

sleep 30

# Run the client code

python /python\_kafka\_test\_client.py broker:9092

Перед запуском следующей команды убедитесь, что вы находитесь в той же папке, что и приведенный выше файл docker-compose.yml:

docker-compose up

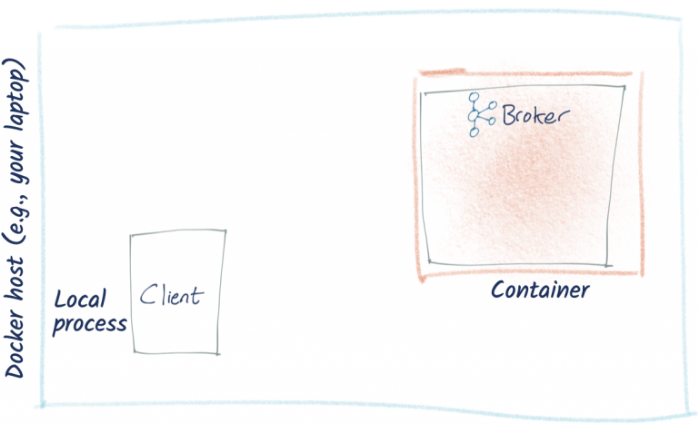
Вы увидите запуск ZooKeeper и брокера Kafka, а затем тестовый клиент Python:

Довольно приятно, да 👍

В этом репозитории вы можете найти полноценные файлы Docker Compose для Apache Kafka и Confluent Platform, включая несколько брокеров (this [repository](https://github.com/confluentinc/cp-all-in-one).).

**Сценарий 4: Kafka в контейнере Docker с клиентом, работающим локально**

Что делать, если вы хотите запустить клиент локально? Возможно, именно там находится ваша IDE, или вы просто не хотите Docker-ифицировать свой клиент?



Давайте возьмем пример, который мы закончили выше, в котором Kafka запускается в Docker через Docker Compose. Если мы попытаемся подключить к нему нашего клиента локально, это не удастся:

$ python python\_kafka\_test\_client.py localhost:9092

Выше мы использовали частную сеть Docker для контейнеров и не открыли ни одного порта для вневшего доступа с хост-машины. Давайте изменим это и предоставим хосту доступ к 9092. Я собираюсь сделать это в YAML-файле Docker Compose — если вы хотите запустить его напрямую из Docker Run, вы можете это сделать, но вам нужно будет напрямую перевести Docker Compose в CLI (что является ошибкой и некрасивым, и почему вам следует просто использовать Docker Compose 😉):

…

broker:

image: confluentinc/cp-kafka:5.5.0

container\_name: broker

networks:

- rmoff\_kafka

ports:

- "9092:9092"

…

Вы можете запустить docker-compose up -d, и он перезапустит все контейнеры, конфигурация которых изменилась (например, брокер). Обратите внимание: если вы просто запустите брокер перезапуска docker-compose, он перезапустит контейнер, используя существующую конфигурацию (а не подберет добавление портов).

Перезапустив контейнер, мы можем проверить, что порт 9092 перенаправляется:

$ docker ps -a

CONTAINER ID IMAGE … PORTS NAMES

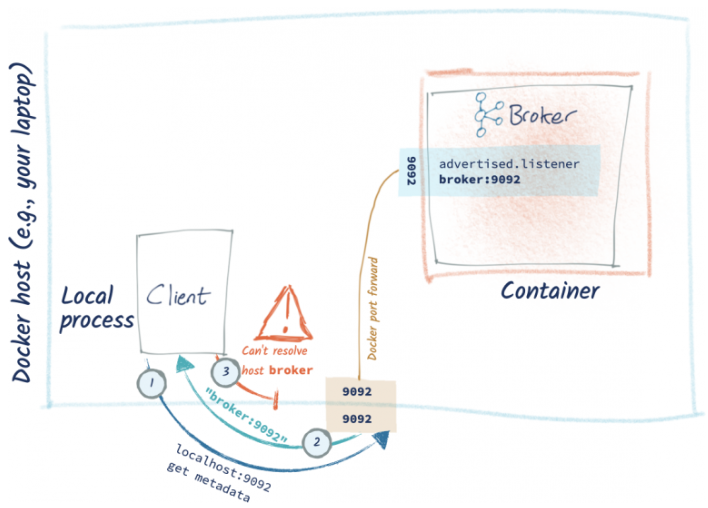
37f5df65e9fc confluentinc/cp-kafka:5.5.0 … 0.0.0.0:9092->9092/tcp broker

…

Давайте еще раз попробуем наш локальный клиент. Все начинается хорошо — мы можем подключиться!

Но дальше дела идут плохо:

Хотя мы можем подключиться к серверу начальной загрузки, он возвращает брокер: 9092 в метаданных.



Это именно то, что мы сказали ему сделать в предыдущем разделе, когда исправляли его для работы с клиентами, работающими в сети Docker. Если сейчас мы изменим Advertised.listener обратно на localhost, брокер Kafka не будет работать, за исключением подключений с хоста.

**Добавление нового слушателя к брокеру**

Так как же нам манипулировать соединениями внутри и снаружи Docker? Создав нового слушателя. Именно для этой цели брокеры могут иметь несколько слушателей. Сетевые топологии становятся странными, а когда дела идут плохо, Кафка привлекает еще несколько слушателей.

Изменения выглядят следующим образом:

…

ports:

- "19092:19092"

environment:

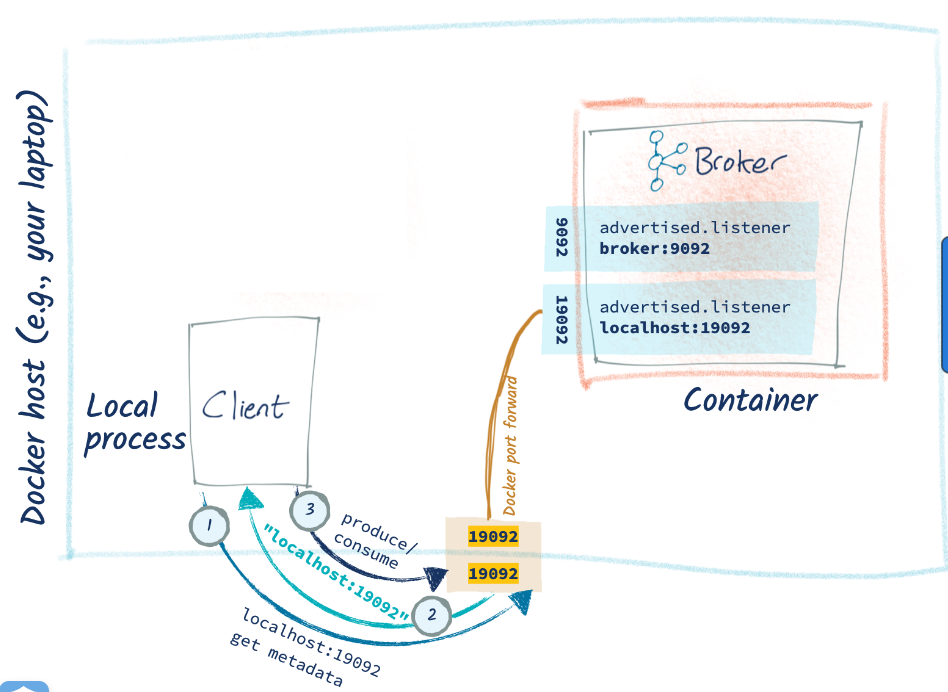
KAFKA\_ADVERTISED\_LISTENERS: PLAINTEXT://broker:9092,CONNECTIONS\_FROM\_HOST://localhost:19092

KAFKA\_LISTENER\_SECURITY\_PROTOCOL\_MAP: PLAINTEXT:PLAINTEXT,CONNECTIONS\_FROM\_HOST:PLAINTEXT

…

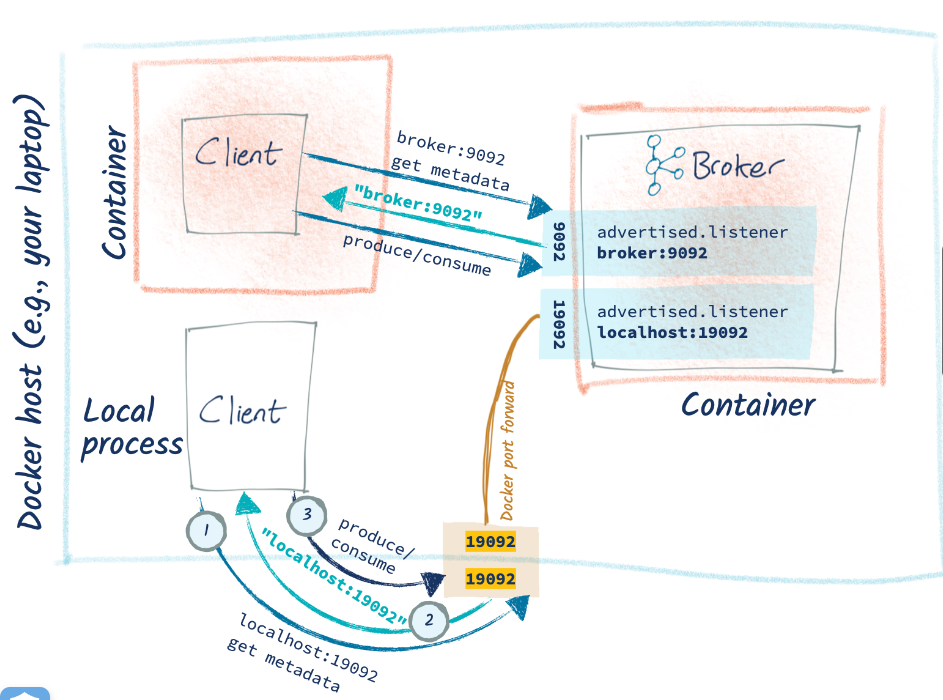
Мы создаем новый прослушиватель под названием CONNECTIONS\_FROM\_HOST, используя порт 19092, и новый Advertised.listener находится на локальном хосте, что очень важно. Поскольку он находится на другом порту, мы меняем сопоставление портов (открывая 19092 вместо 9092).

Существующий прослушиватель (PLAINTEXT) остается неизменным. Нам также необходимо указать KAFKA\_LISTENER\_SECURITY\_PROTOCOL\_MAP. Раньше для одного прослушивателя использовалось значение по умолчанию, но теперь, когда мы добавили еще один, нам нужно настроить его явно.



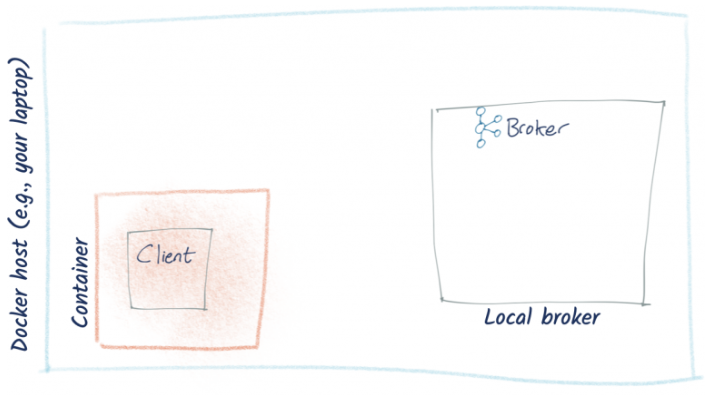
После того, как брокер забрал новую конфигурацию, наш локальный клиент работает отлично — до тех пор, пока мы не забываем указать ему новый порт прослушивателя (19092):

В Docker Compose мы видим, что наш клиент на базе Docker все еще работает:



**Сценарий 5: Kafka работает локально с клиентом в контейнере Docker.**

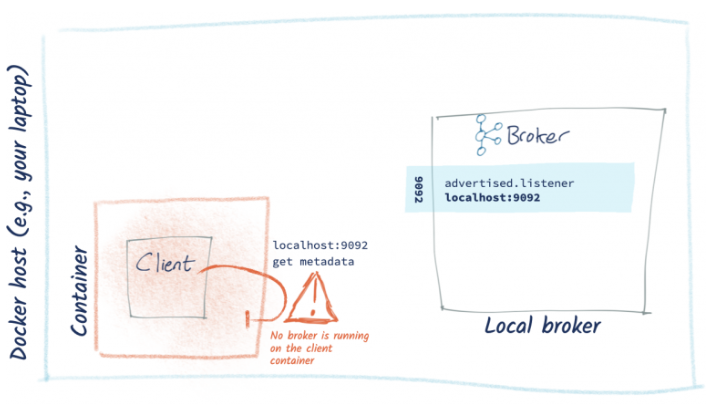
А что, если мы инвертируем это и запустим Kafka локально на нашем ноутбуке, как и изначально, и вместо этого запустим клиент в Docker? Это неочевидный способ управления делами, но ¯\\_(ツ)\_/¯



Сначала я закрываю контейнеры Docker сверху (docker-compose down), а затем запускаю Kafka локально (слитный локальный запуск kafka). Если мы запустим наш клиент в его контейнере Docker (образ, для которого мы создали выше), мы увидим, что он недоволен:

docker run --tty python\_kafka\_test\_client localhost:9092

Если вы помните парадокс Docker/localhost, описанный выше, вы поймете, что здесь происходит. В клиентском Docker-контейнере localhost сам по себе является — это не тот «локальный хост», каким мы думаем о нашем ноутбуке, как хост Docker. И, конечно же, в Docker-контейнере нашего клиента нет брокера Kafka, работающего под номером 9092, отсюда и ошибка.



Если вы мне не совсем верите, попробуйте запустить это, которое проверяет изнутри Docker-контейнера, открыт ли порт 9092 на локальном хосте:

$ docker run -it --rm --entrypoint "/bin/nc" \

python\_kafka\_test\_client -vz \

localhost 9092

localhost [127.0.0.1] 9092 (?) : Connection refused

На хост-компьютере Docker Kafka запущен и порт открыт:

$ nc -vz localhost 9092

Connection to localhost port 9092 [tcp/XmlIpcRegSvc] succeeded!

Итак, как нам подключить нашего клиента к нашему хосту? Прежде чем ответить на этот вопрос, давайте подумаем, почему мы можем захотеть это сделать. Есть две причины, по которым вы окажетесь в таком состоянии:

Вы находитесь на этом этапе, потому что вы просто разрабатываете что-то и пытаетесь заставить его работать любым возможным способом, и позже будете беспокоиться о том, чтобы сделать это «правильно».

Вы создаете клиентское приложение, которое будет работать на Docker и подключаться к Kafka, работающему где-то еще.

В последнем сценарии вам необходимо сослаться на «клиент и Kafka на разных машинах» выше и убедиться, что (а) брокеры объявляют свои правильные данные о прослушивателе и (б) контейнер может правильно разрешить эти адреса хостов.

В первом случае (попытка получить доступ к Kafka, работающему локально, из клиента, работающего в Docker), у вас есть несколько вариантов, ни один из которых не особенно приятен. Если вы используете Docker на Mac, есть хитрый обходной путь: использовать host.docker.internal в качестве адреса, по которому можно получить доступ к хост-компьютеру изнутри контейнера:

$ docker run -it --rm --entrypoint "/bin/nc" \

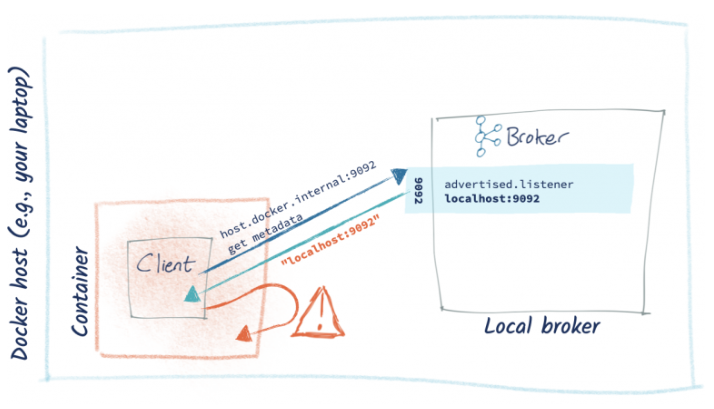
python\_kafka\_test\_client -vz \

host.docker.internal 9092

host.docker.internal [192.168.65.2] 9092 (?) open

Таким образом, контейнер может видеть порт 9092 хоста. Что, если мы попытаемся подключиться к нему из нашего настоящего клиента Kafka?

Итак, первоначальное соединение действительно работает, но проверьте метаданные, которые мы получаем: localhost:9092. Почему? Потому что рекламируются слушателями. Итак, теперь производитель и потребитель не будут работать, потому что они пытаются подключиться к localhost:9092 внутри контейнера, что не сработает.



Время взлома? ХОРОШО. Давайте возьмем нашего бедного местного брокера Kafka и задействуем его, чтобы открыть прослушиватель на host.docker.internal. Поскольку мы не хотим ломать брокер Kafka для других клиентов, которые действительно хотят подключиться к локальному хосту, мы создадим себе нового прослушивателя. Измените server.properties у брокера с:

listeners=PLAINTEXT://:9092

advertised.listeners=PLAINTEXT://localhost:9092

listener.security.protocol.map=PLAINTEXT:PLAINTEXT

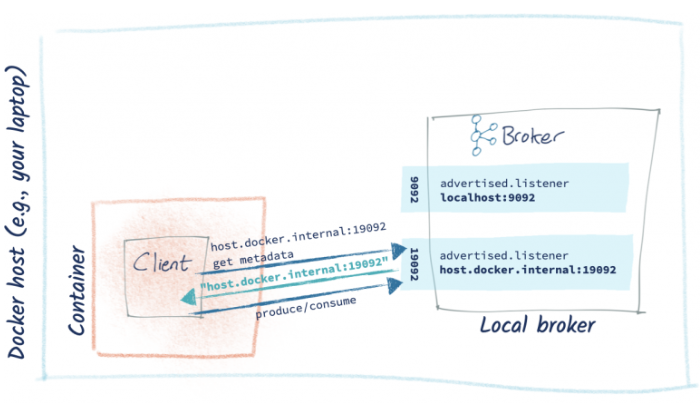
… к:

listeners=PLAINTEXT://:9092,RMOFF\_DOCKER\_HACK://:19092

advertised.listeners=PLAINTEXT://localhost:9092,RMOFF\_DOCKER\_HACK://host.docker.internal:19092

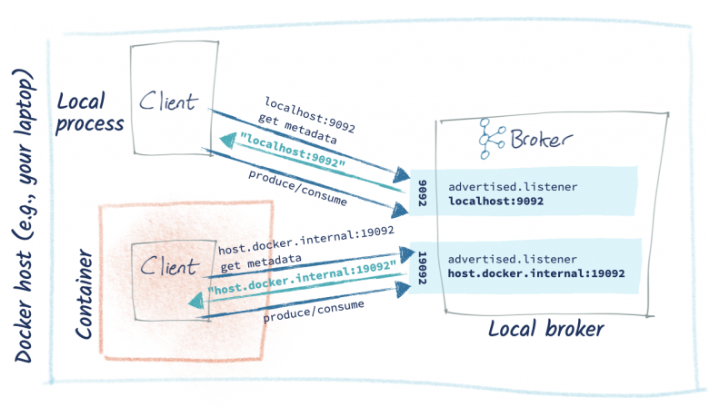
listener.security.protocol.map=PLAINTEXT:PLAINTEXT,RMOFF\_DOCKER\_HACK:PLAINTEXT

Исходный слушатель остается неизменным. Однако самое волшебное, что мы здесь сделали, — это добавление нового прослушивателя (RMOFF\_DOCKER\_HACK), который находится на новом порту. Если вы подключитесь к брокеру по номеру 9092, вы получите рекламируемый прослушиватель, определенный для прослушивателя на этом порту (локальный хост). А если вы подключитесь к брокеру по номеру 19092, вы получите альтернативный хост и порт: host.docker.internal:19092.



Давайте попробуем (сначала убедитесь, что вы перезапустили брокер, чтобы изменения вступили в силу):

Оно работает! Это ГРЯЗНЫЙ ХАК, но он работает 😅. Не менее важно и то, что мы не сломали Kafka для локальных (не Docker) клиентов, поскольку исходный прослушиватель 9092 все еще работает:



**Часто задаваемые вопросы**

**Разве я не могу просто взломать файл /etc/hosts?**

Не могли бы вы? Да

Вы должны? **НЕТ!** 🙂

Нет, если только вы не хотите, чтобы ваш клиент случайно переставал работать каждый раз, когда вы развертываете его на машине, для которой вы забыли взломать файл хостов. В этом вся суть имен хостов и разрешения DNS — именно благодаря им машины узнают, как общаться друг с другом, вместо того, чтобы вы жестко запрограммировали это на каждой машине индивидуально.

**У меня не установлено значение «advertized.listeners» в моем файле server.properties.**

По умолчанию он принимает то же значение, что и сам прослушиватель. Вы можете проверить используемые настройки, проверив файл журнала брокера:

[2020-04-27 21:21:00,939] INFO KafkaConfig values:

advertised.host.name = null

advertised.listeners = PLAINTEXT://localhost:9092,RMOFF\_DOCKER\_HACK://host.docker.internal:19092

advertised.port = null

…

listener.security.protocol.map = PLAINTEXT:PLAINTEXT,RMOFF\_DOCKER\_HACK:PLAINTEXT

listeners = PLAINTEXT://:9092,RMOFF\_DOCKER\_HACK://:19092

…

*А как насчет «\_advertized.host.name» и «advertized.port»?*

Они устарели. Не используйте их.

**Но… я могу нормально связаться с брокером по телнету, так что, конечно же, это должно сработать?**

Да, вам необходимо иметь возможность связаться с брокером на хосте и порту, который вы указали при первоначальном загрузочном соединении. Однако, как объяснялось выше, последующие подключения к хосту и порту, возвращаемые в метаданных, также должны быть доступны с вашего клиентского компьютера.

**Нужно ли мне использовать Python для этого?**

Нет, любая клиентская библиотека (см. этот список и GitHub) также должна иметь возможность предоставлять метаданные. Вот пример использования kafkacat:

$ kafkacat -b asgard05.moffatt.me:9092 -L

Metadata for all topics (from broker 1: asgard05.moffatt.me:9092/1):

3 brokers:

broker 2 at asgard05.moffatt.me:19092

broker 3 at asgard05.moffatt.me:29092

broker 1 at asgard05.moffatt.me:9092 (controller)

Вы также можете использовать kafkacat из Docker, но тогда вы столкнетесь с некоторыми необычными сетевыми последствиями, если попытаетесь устранить какие-либо неполадки в локальной сети.

**В заключение…**

Существует два типа соединения вашего клиента с брокерами Kafka, которые должны быть успешными:

1. **Первоначальное** подключение к брокеру (бутстрап). При этом клиенту возвращаются метаданные, включая список всех брокеров в кластере и их конечных точек подключения.
2. Затем клиент при необходимости подключается к одному (или нескольким) брокерам, **возвращенным на первом этапе**. Если брокер настроен неправильно, соединения не будут установлены.