Быстрый старт в Kafka

Когда должно быть еще вчера

Подготовил:
Галонза Пётр Валерьевич

Чего не будет

- Разбора внутренней механики
- Разбора работы кластера и кворума
- Разбор репликации
- Разговоров о лучших практиках
- Тонкой настройки

Для чего вот это все?

- Не сталкивались с Kafka
- Что необходимо проработать и к чему подготовиться
- Что нам и разработчикам нужно друг от друга
- Отсутствие постепенного изучения и наличие планки вхождения

И что же такое Kafka

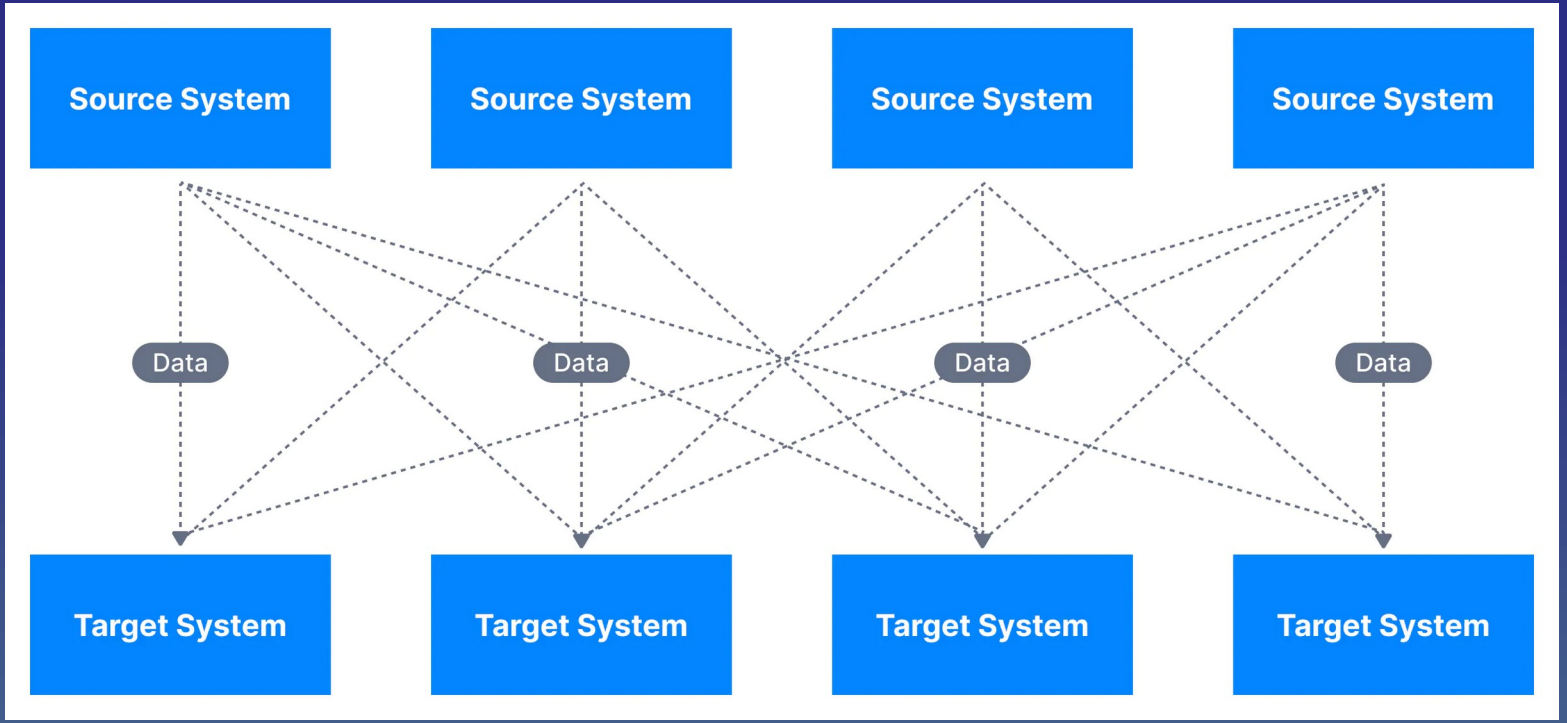
Это платформа потоковой обработки данных реализующая паттерн проектирования публикация/подписка, изначально разработанная компанией LinkedIn и названа в честь немецкого писателя.

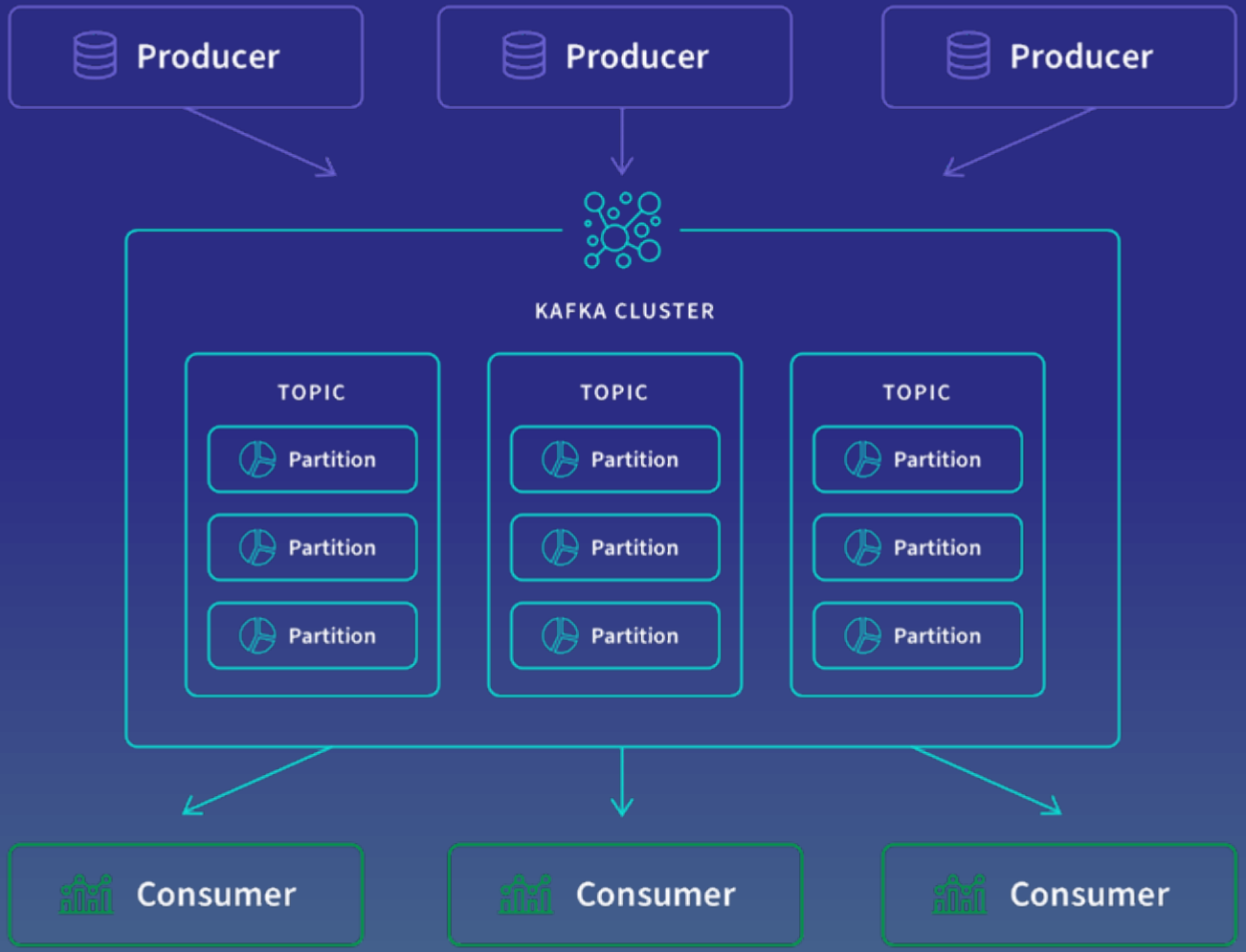
По простому брокер сообщений.

Что нам нужно

- Kafka 3.1.0
- Zookeeper 3.6.3
- Java 8+

Зачем нам Kafka?




Producer
Producer
Producer
KAFKA CLUSTER
TOPIC
Partition
Partition
Partition
TOPIC
Partition
Partition
Partition
TOPIC
Partition
Partition
Partition
Consumer
Consumer
Consumer

Почему Kafka?

- Настройка пакетирования сообщений
- Сохранение сообщений на диск
- Pull модель получателей
- Логика на стороне клиента
- Гарантия порядка сообщений

Все то, что нужно в работе с бизнес-данными

С чем столкнемся?

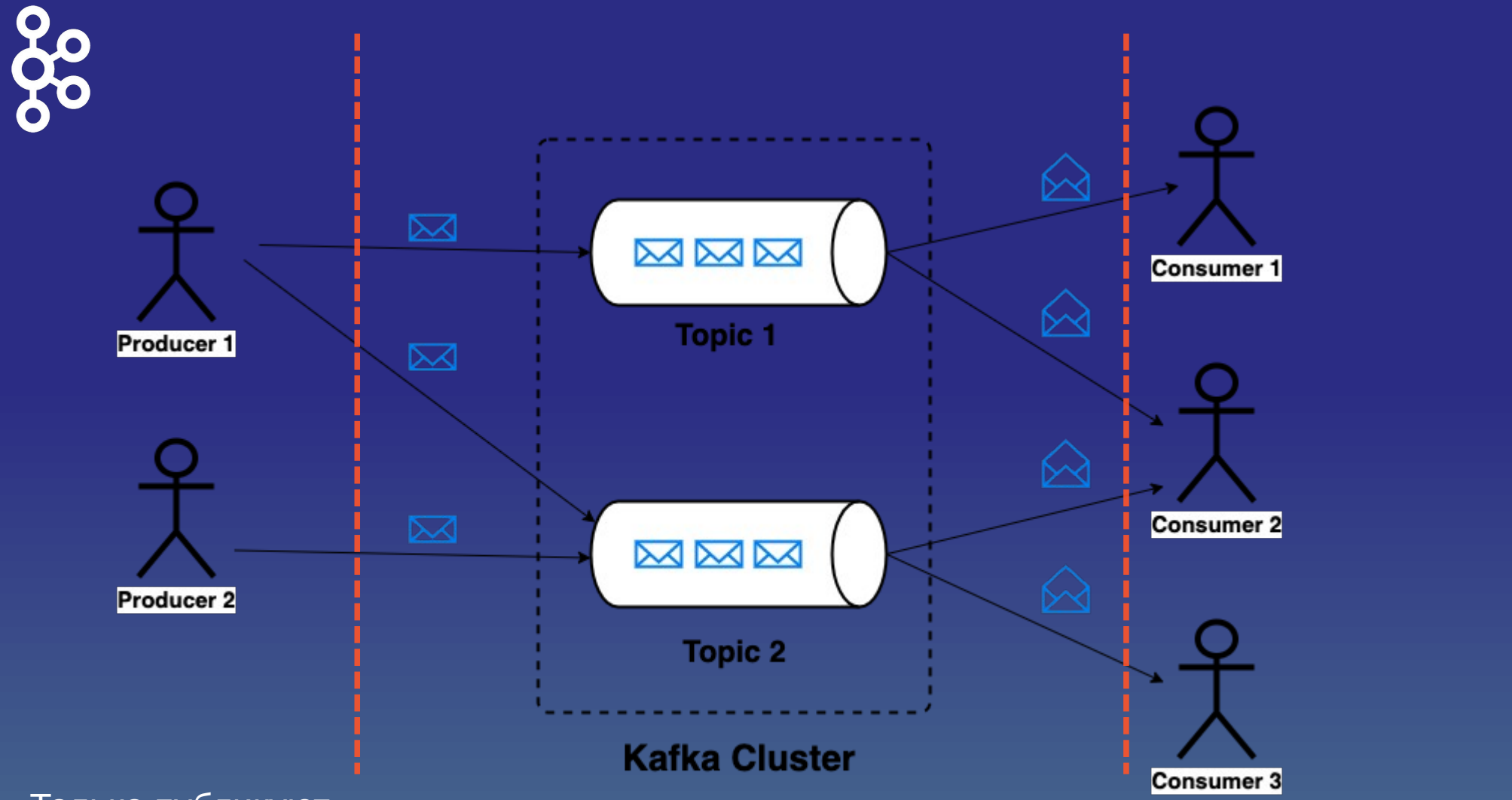
- Messages
- Topics
- Partitioons
- Message keys
- Consumer
- Consumer Groups
- Producer
- Auntetification
- Authorization

Message

- Единица данных
- В Kafka сообщения пишутся и читаются пакетами
- Сообщения пишутся в partition
- У сообщений может быть задан ключ (отвечает за порядок)
- Сообщения имеют структуру(сериализацию), чаще всего JSON

Topics

- Объединяет сообщения по определенной тематике (тип данных, событий и т.д. к которому относится сообщения)
- В топик публикуем(producer)
- На топик подписываемся(consumer)
- Топик содержит разделы

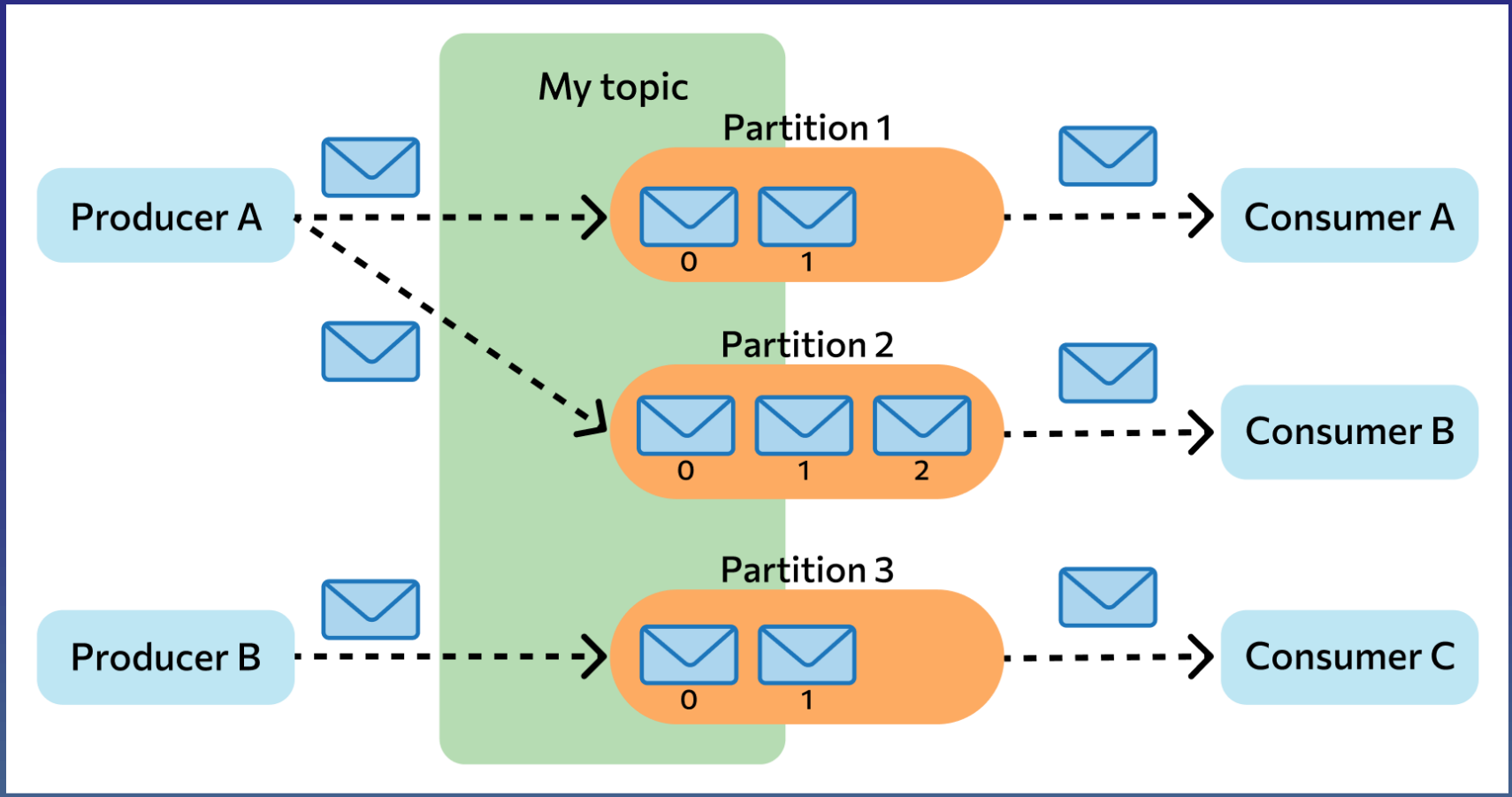


Только публикуют

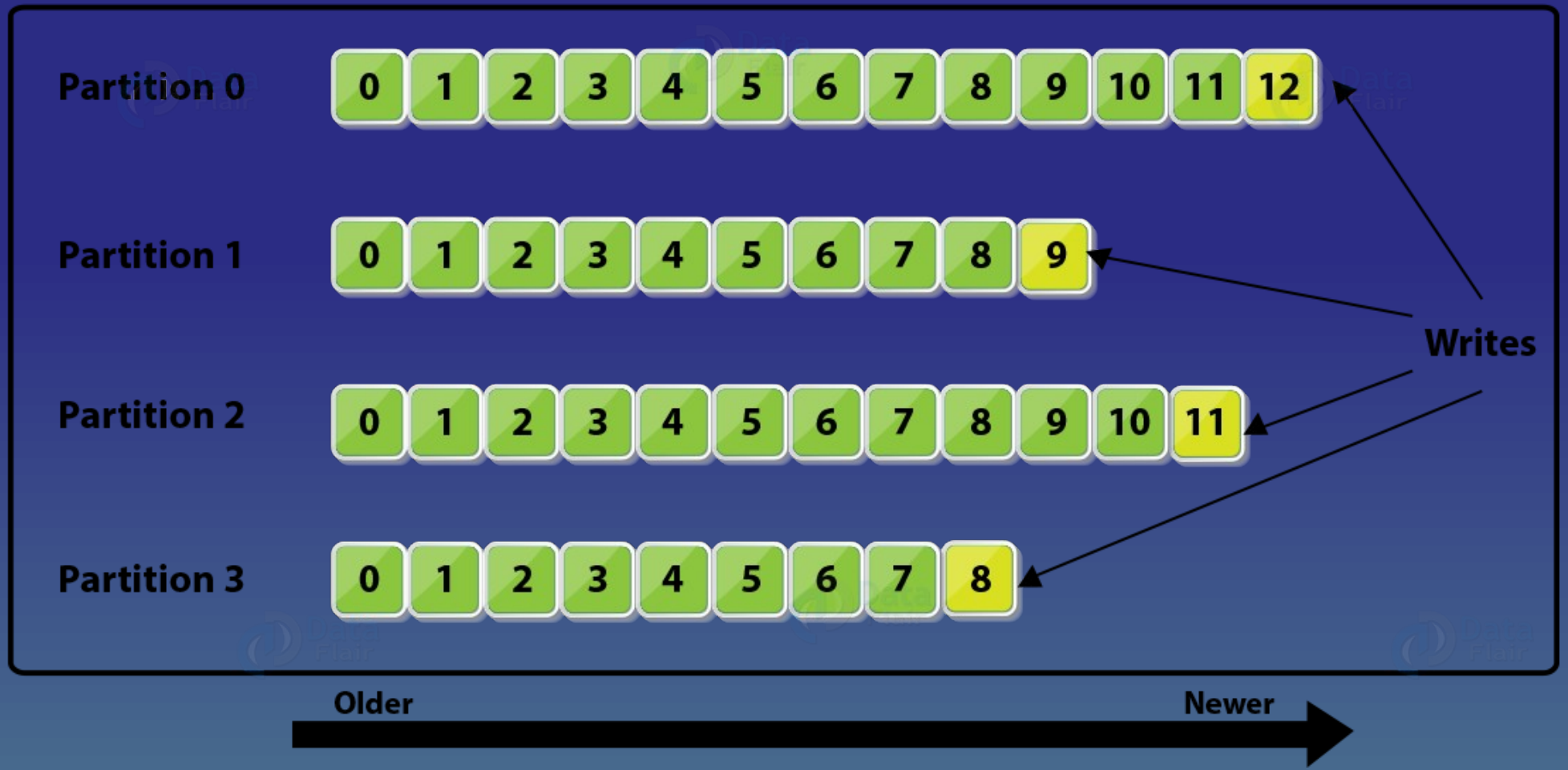
Только читают и обрабатывают

Partitions

- Труба для сообщений
- Сообщения записываются последовательно в конец, а читаются с начала
- Прочитанные данные отмечаются смещением
- При наличии нескольких разделов в теме отсутствует гарантия упорядоченности сообщений в топике
- Количество разделов влияет на пропускную способность, масштабируемость, параллельность и отказоустойчивость
- Добавлять разделы можно, уменьшать нельзя


My topic
Partition 1
Partition 2
Partition 3
Producer A
Producer B
Consumer A
Consumer B
Consumer C
0
1
0
1
2
0
1

Kafka Topic Partitions Layout



Количество разделов

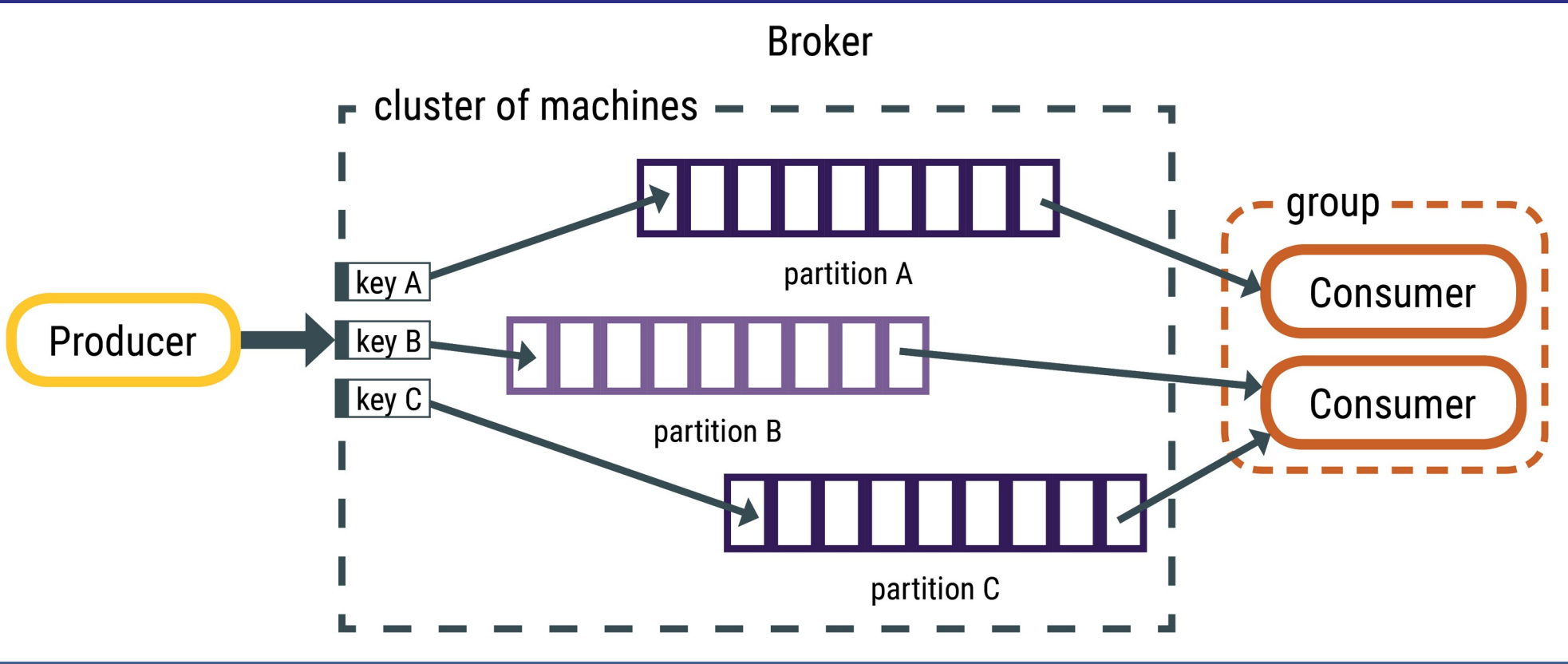
- Размер сообщения
- Желаемая пропускная способность
- Время обработки сообщения
- Интенсивность записи
- Требуемая надежность
- Расход оперативной памяти на раздел
- Расход дискового пространства

Это минимум, что стоит учитывать!

- На проекте выбрали 16 разделов
- Завышать плохо

Гарантия упорядоченности сообщений

Чтобы гарантировать порядок чтения сообщений, к ним нужно добавить ключ, сообщения с общим ключем, будут записаны в один раздел.

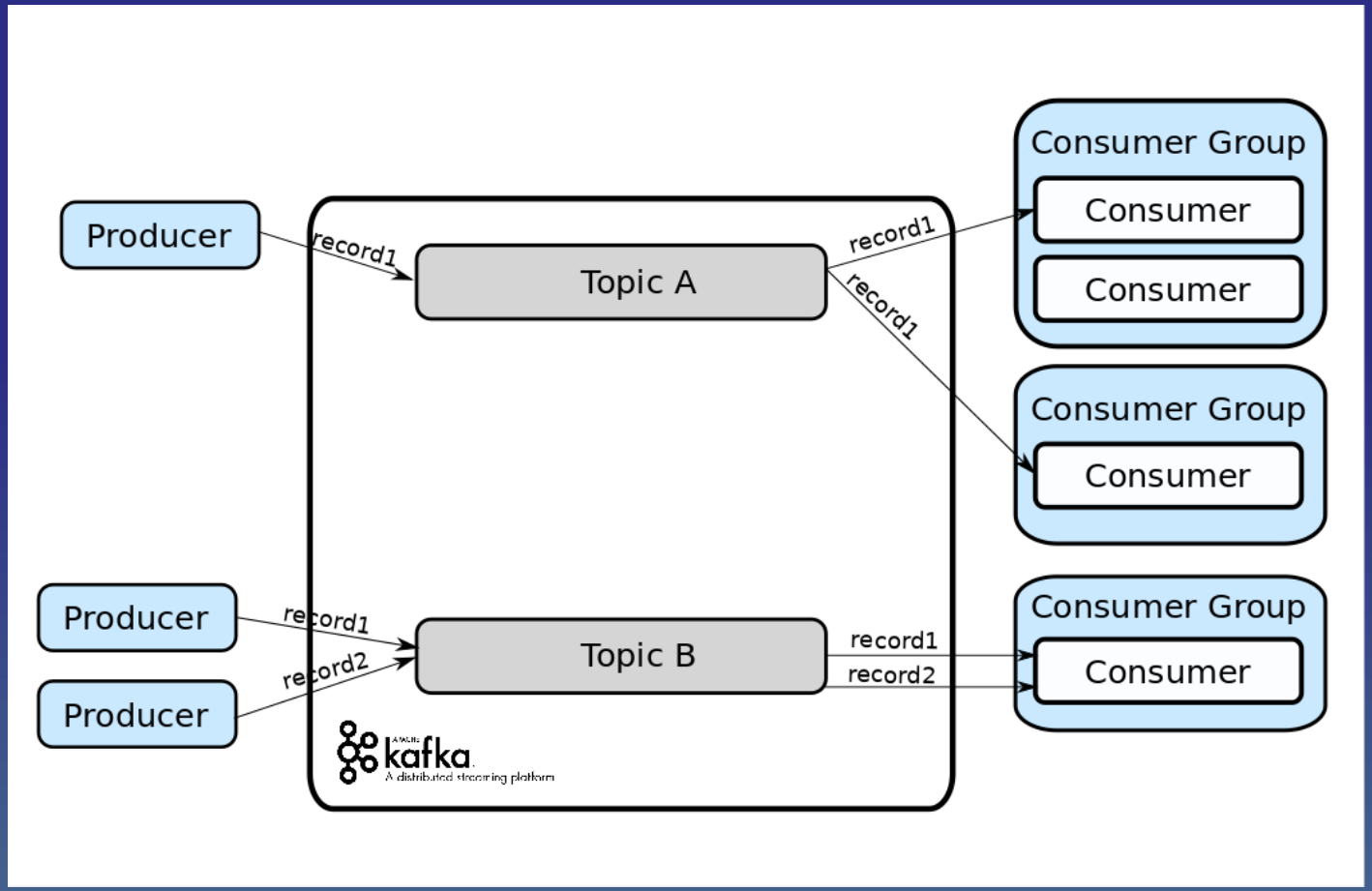

Broker
cluster of machines
Producer
key A
key B
key C
partition A
partition B
partition C
group
Consumer
Consumer

Producer

- Генерирует сообщения в топики
- Распределяет сообщения по разделам
- При наличии ключа пишет в один раздел
- Логика распределения, интенсивности записи, размера пакета и подтверждения записи на стороне клиента
- При добавлении разделов, теряется гарантия порядка сообщений

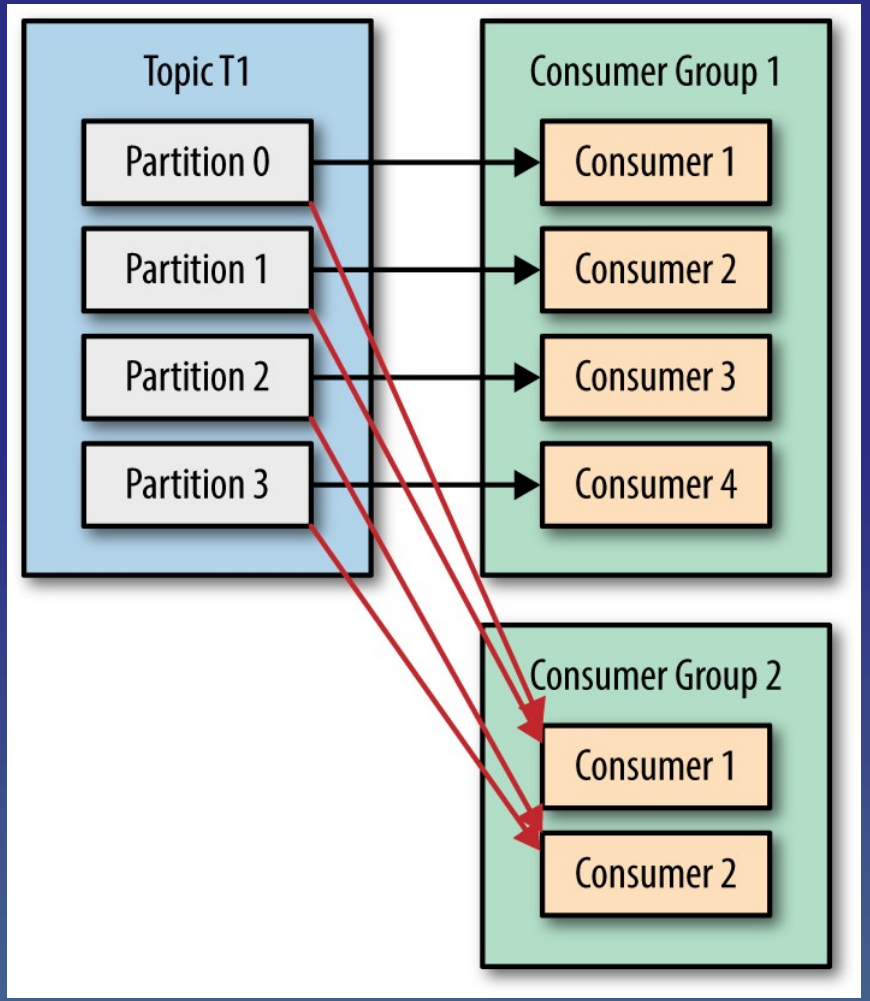
Consumer

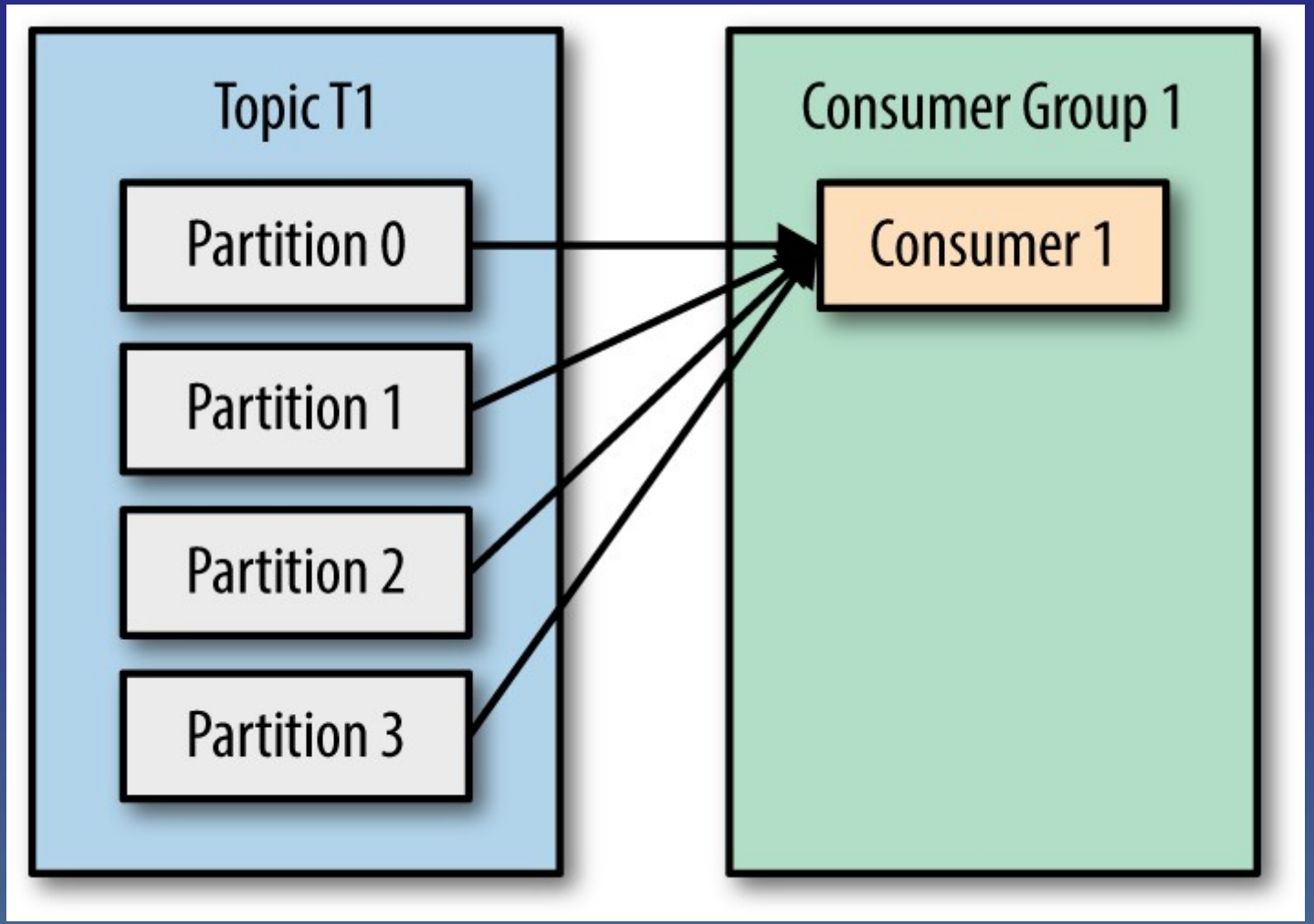
- Читает сообщения из разделов и обрабатывает их
- Отслеживает какие сообщения прочитаны и фиксирует смещение(не фиксирует какие именно сообщения прочитаны)
- Работает в составе групп(Consumer group)
- Логика фиксации смещения, размер пакета и интенсивность на стороне клиента

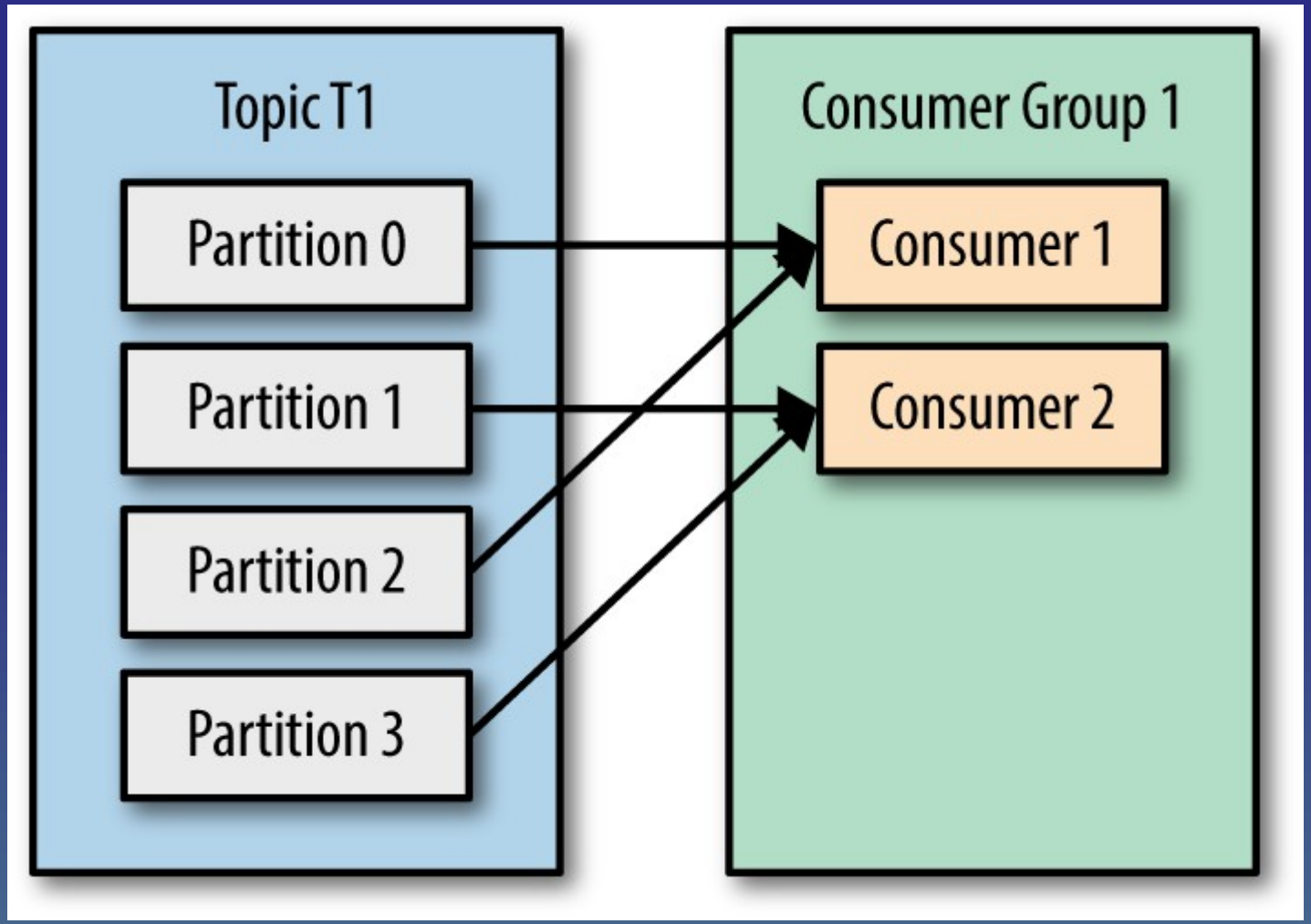

Producer
record1
Topic A
record1
record1
Consumer Group
Consumer
Consumer
Consumer Group
Consumer
Producer
record1
Producer
record2
Topic B
record1
record2
Consumer Group
Consumer
kafka
A distributed streaming platform

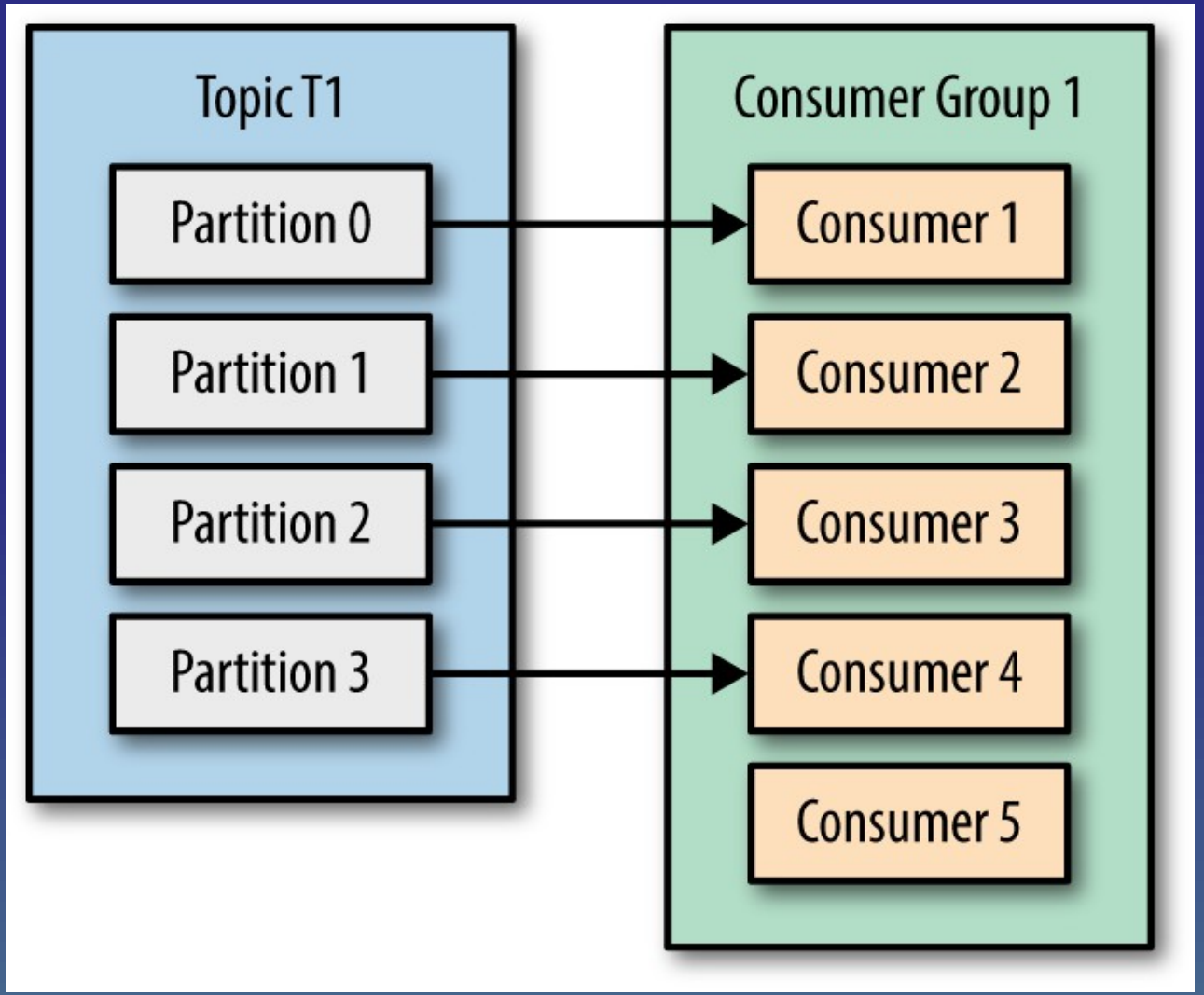
Consumer group

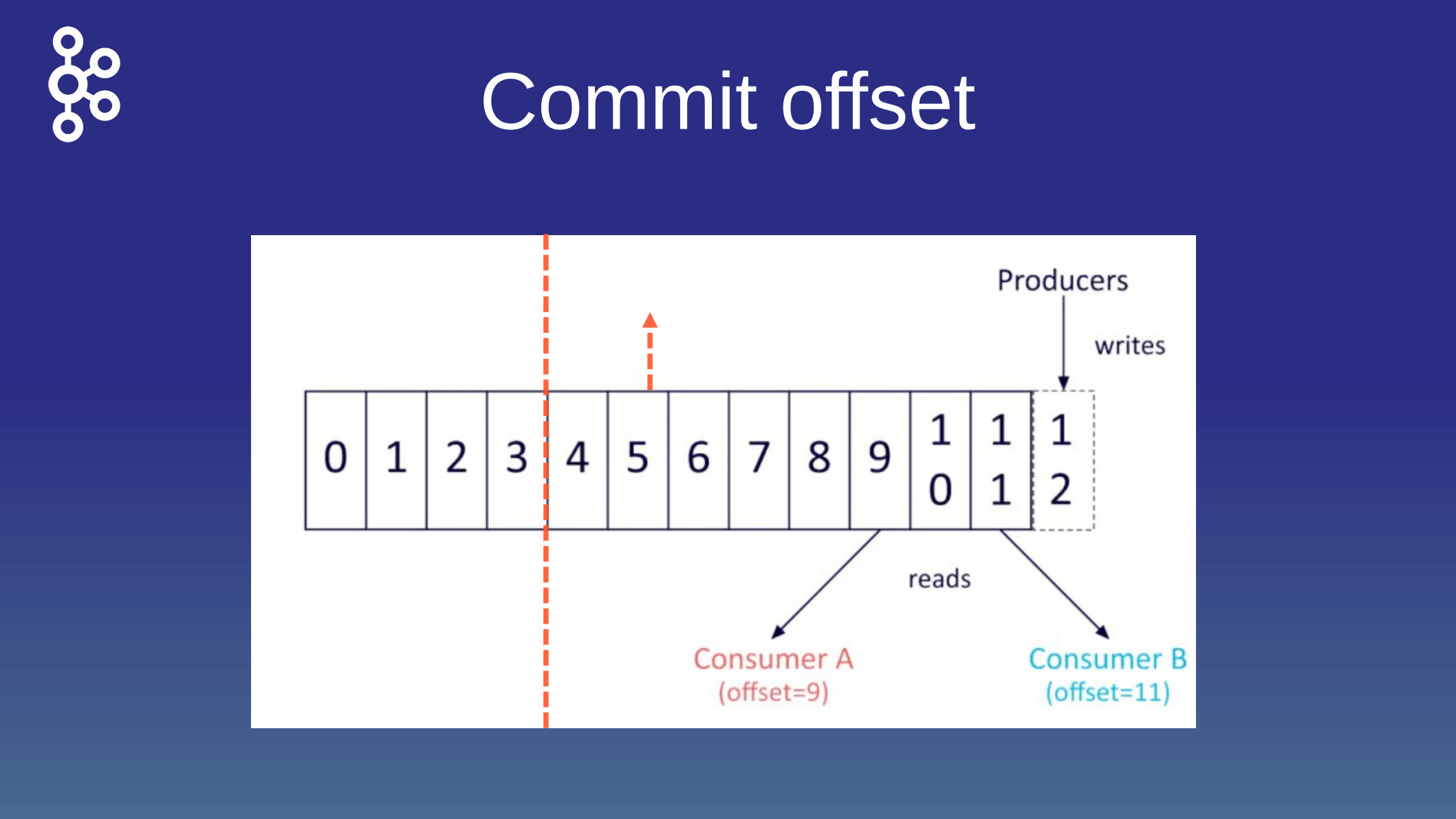
- Задается на стороне потребителя(не на стороне брокера)
- Реализует параллельную обработку сообщений из топика несколькими потребителями
- В рамках группы раздел читает только один потребитель
- На топик могут подписаться несколько групп
- Если не указать отношение потребителя к группе, потребитель генерирует свою группу
- Количество потребителей не должно превышать количество разделов в топике.
- В пределах группы сообщение потребителями будет прочитано единожды(не гарантировано)


Topic T1
Partition 0
Partition 1
Partition 2
Partition 3
Consumer Group 1
Consumer 1
Consumer 2
Consumer 3
Consumer 4
Consumer Group 2
Consumer 1
Consumer 2


Topic T1
Partition 0
Partition 1
Partition 2
Partition 3
Consumer Group 1
Consumer 1


Topic T1
Partition 0
Partition 1
Partition 2
Partition 3
Consumer Group 1
Consumer 1
Consumer 2


Topic T1
Partition 0
Partition 1
Partition 2
Partition 3
Consumer Group 1
Consumer 1
Consumer 2
Consumer 3
Consumer 4
Consumer 5



Commit offset

Каков итог?

- ✓ Что такое Kafka
- ✓ Что такое Topic
- ✓ Что такое Partition
- ✓ Что такое Producer
- ✓ Что такое Consumer
- ✓ Что такое Consumer Grouop
- ✓ Как добиться упорядоченности сообщений

Профит

- Можем вести диалог с командой разработки:
  - Кто Производитель
  - Кто Потребитель
  - Имена Топиков
  - Количество разделов
  - Кто и в какую группу потребители будут входить
- Диалог с смежными отделами (Vector):
  - В какой топик писать
  - Нужен ключ или нет
  - Как запустить больше экземпляров

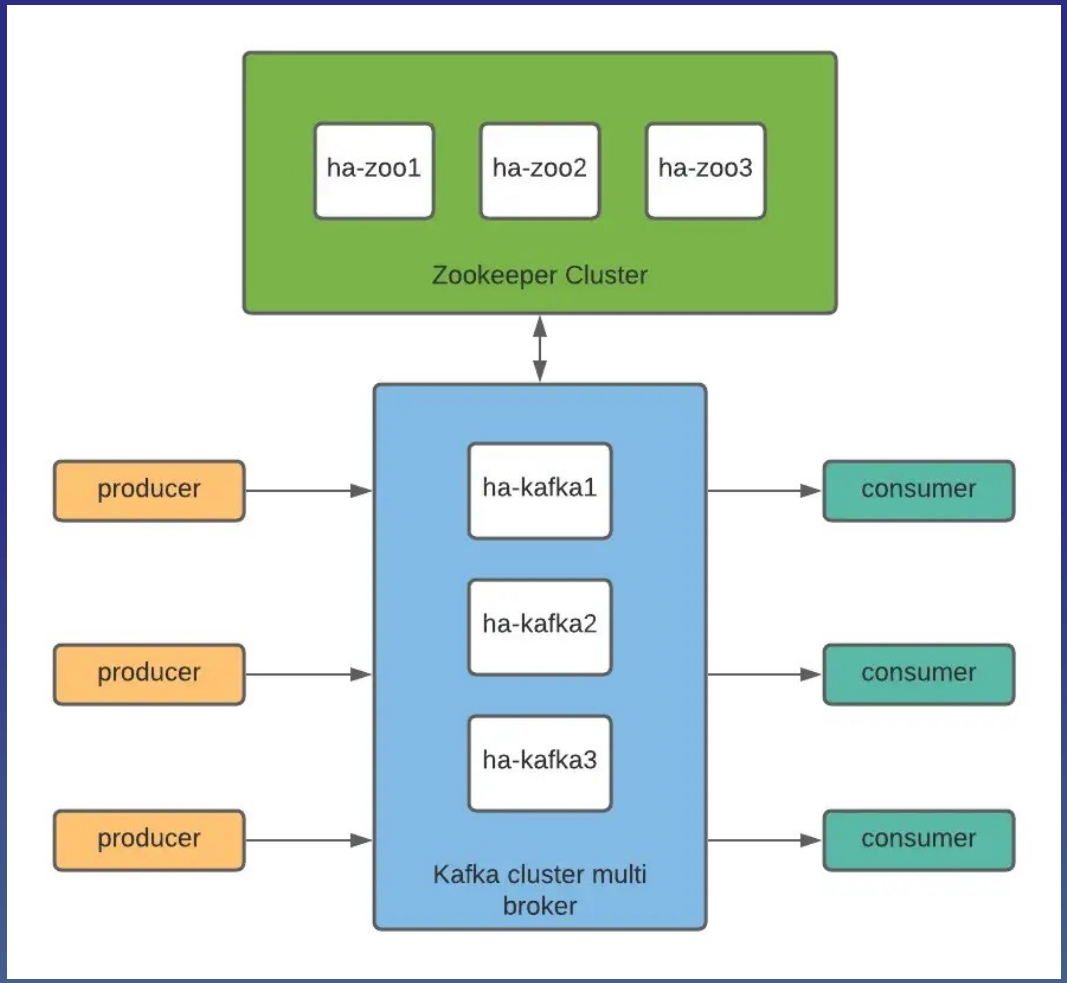
А как же Zookeeper?

Это key-value хранилище с древовидной ирархией, используется для синхронизации и координации кластеров распределенных систем

APACHE ZooKeeper™

Но зачем?

- Хранить метаданные о разделах
- Хранить метаданные о брокерах в кластере
- Хранить информацию о потребителях их группах
- Хранить данные для SASL/SCRAM
- Выбор ведущего брокера


ha-zoo1
ha-zoo2
ha-zoo3
Zookeeper Cluster
producer
producer
producer
ha-kafka1
ha-kafka2
ha-kafka3
Kafka cluster multi broker
consumer
consumer
consumer

К сведению

- С версии Kafka 2.8.0 вносятся изменение направленные на отказ от Zookeeper
- Producer и consumer с версии 2.8.0 не в взаимодействуют с Zookeeper напрямую, появились высокоуровневые API
- С дистрибутивом Kafka поставляется Zookeeper(light), но рекомендуется разворачивать полноценный дистрибутив
- Не использовать Zookeeper для Кафки с другими приложениями

Аунтетификация

- JAAS (Java Authentication and Authorization Service)
  - SASL (Simple Authentication and Security Layer)
    - ✗ PLAIN
    - ✓ SCRAM
    - ✗ LDAP
    - ◑ OAUTHBEARER
    - ✗ GSSAPI
- mTLS

SASL/SCRAM

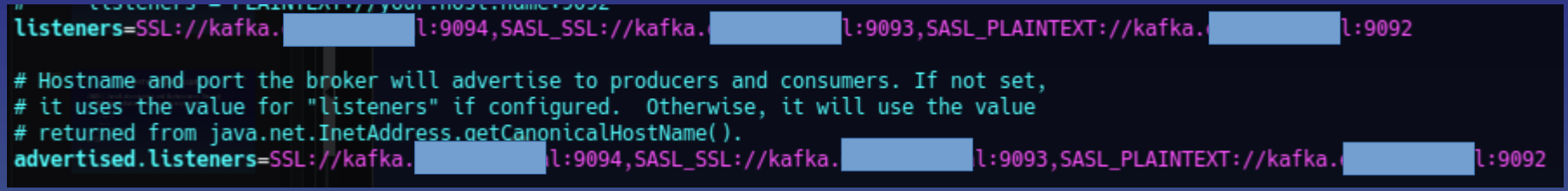
- Логин, пароль
- Механизмы хранения данных
  - SCRAM-SHA-512
  - SCRAM-SHA-256

Работаем с SASL/SCRAM

Если есть доступ к серверу, убедимся, что данный метод аунтетификации доступен

/srv/kafka/confing/server.properties

- advertised.listeners: SASL_SSL
- sasl.enabled.mechanisms: SCRAM-SHA-512,SCRAM-SHA-256



В противном случае telnet на 9093

Получаем superuser

/srv/kafka/confing/kafka_server_jaas.conf

```
KafkaServer {
   org.apache.kafka.common.security.scram.ScramLoginModule required
   username="admin"
   password="admin";
};
```

Создаем файл для клиентского подключения

```
security.protocol=SASL_SSL
sasl.mechanism=SCRAM-SHA-512
ssl.truststore.location=/opt/kafka/ssl/truststore.jks
ssl.truststore.password=changeme
sasl.jaas.config=org.apache.kafka.common.security.scram.ScramLoginModule required \
  username="admin" \
  password="admin";
~
```

Создаем пользователя

```
kafka-configs.sh \
--bootstrap-server kafka.example.local:9093 \
--alter \
--add-config 'SCRAM-SHA-256=[password=secret-client],SCRAM-SHA-512=[password=secret-client]' \
--entity-type users \
--entity-name client-service \
--command-config sasl-client.properties
```

```
[vagrant@kafka bin]$ ./kafka-configs.sh --bootstrap-server kafka          l:9093 --entity-type users --describe  --command-config ../config/sasl-client.properties
SCRAM credential configs for user-principal 'admin' are SCRAM-SHA-256=iterations=4096, SCRAM-SHA-512=iterations=4096
SCRAM credential configs for user-principal 'client' are SCRAM-SHA-256=iterations=4096, SCRAM-SHA-512=iterations=4096
SCRAM credential configs for user-principal 'client-service' are SCRAM-SHA-256=iterations=4096, SCRAM-SHA-512=iterations=4096
[vagrant@kafka bin]$
```

SASL/TLS

- Запущен на порту 9092/9094

Получаем superuser

```
security.protocol=SSL
ssl.truststore.location=/opt/kafka/ssl/truststore.jks
ssl.truststore.password=changeme
ssl.keystore.location=/opt/kafka/ssl/client.jks
ssl.keystore.password=changeme
```

Создаем запрос на сертификат

```
[req]
default_bits = 2048
default_md = sha256
distinguished_name = req_distinguished_name
prompt = no
utf8 = yes

[req_distinguished_name]
countryName = RU
stateOrProvinceName = Moscow
localityName = Moscow
organizationName = example
organizationalUnitName = localhost
commonName = client
emailAddress = test@example.local ← deprecated
```

```
openssl req -new -keyout private.key -config clinet-example.cfg -out request.csr
```

Авторизация

- ACL (Access Control Lists)
- LDAP

ACLs

- Кластер
- Топик
- Группу потребителей
- Группы, топики, кластер нельзя указать регуляркой

ACLs producer

```
kafka-acls.sh \
--bootstrap-server kafka.example.local:9093 \
--add
--topic test
--allow-principal User:client
--producer
--command-config sasl-client.properties
```

- Write
- Create
- Describe

ACLs consumer

```
kafka-acls.sh \
--bootstrap-server kafka.example.local:9093 \
--add
--topic test
--allow-principal User:client
--consumer
– group client-group
--command-config sasl-client.properties
```

- Read
- Describe
- Group (Read)

ACLs для mTLS

- CN=client,OU=localhost,O=Example,L=Moscow,ST=Moscow,C=RU
- 1.2.840.113549.1.9.1=#<email in hex>,CN=client,OU=localhost,O=Example,L=Moscow,ST=Moscow,C=RU

```
ssl.principal.mapping.rules= \
RULE:^CN=([a-zA-Z0-9.]*).*$/$1/L ,\
RULE:^CN=(.*?),OU=ServiceUsers.*$/$1/,\
RULE:^CN=(.*?),OU=(.*?),O=(.*?),L=(.*?),ST=(.*?),C=(.*?)$/$1@$2/L,\
RULE:^.*[Cc][Nn]=([a-zA-Z0-9.]*).*$/$1/L,\
DEFAULT
```

Не regexp, но хоть что то

```
kafka-acls.sh \
--bootstrap-server kafka.example.local:9093 \
--add \
--topic sed- \
--allow-principal User:client \
--consumer \
– group sed- \
--resource-pattern-type prefixed \
--command-config sasl-client.properties
```

DLC OAUTHBEARER

- KIP-768: Extend SASL/OAUTHBEARER with Support for OIDC
- Не смог добиться настройки ролей
- Настроена, но не видел применения
- Авторизация с использованием клиента

Server

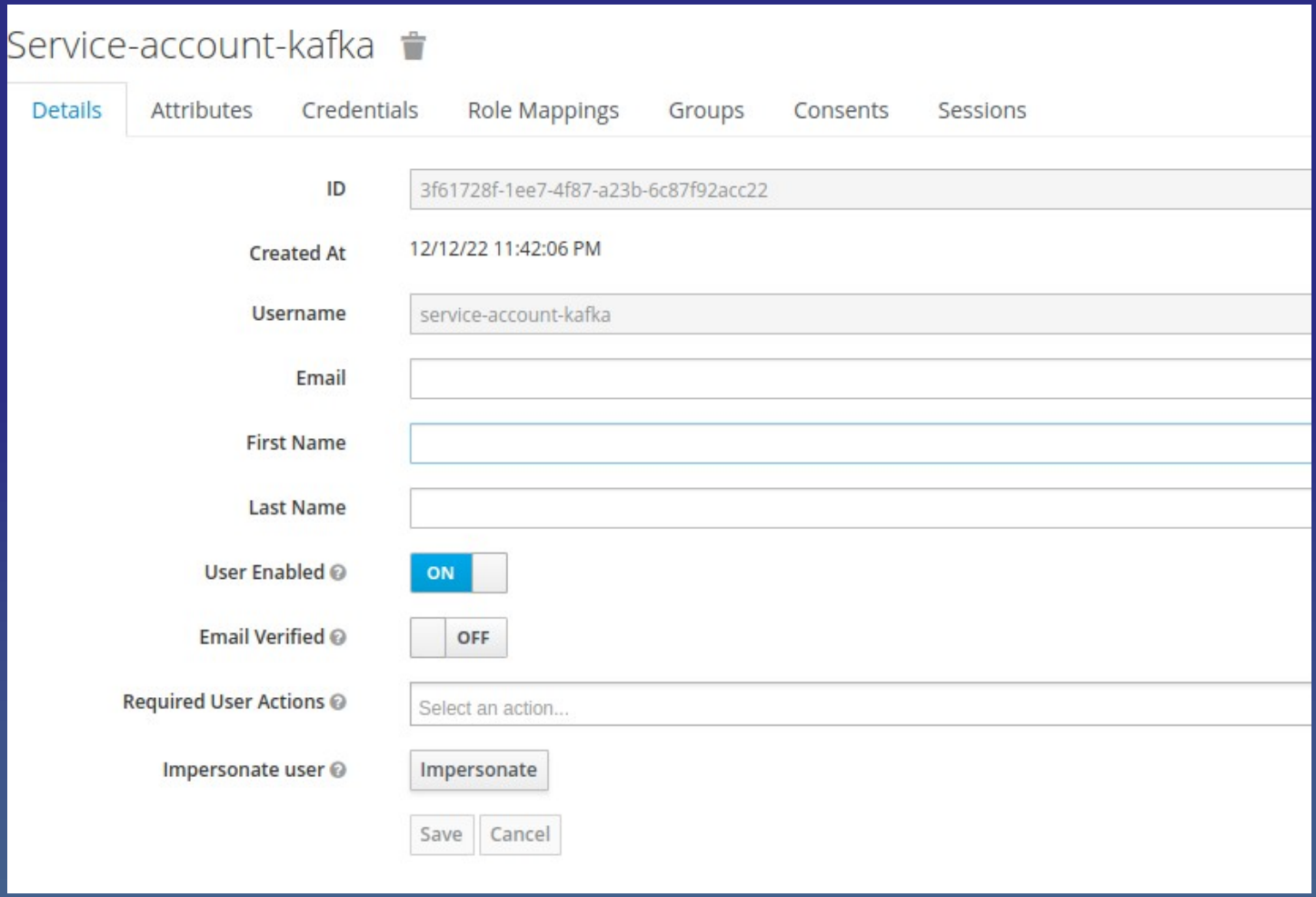
```
sasl.enabled.mechanisms=SCRAM-SHA-512,SCRAM-SHA-256,OAUTHBEARER

listener.name.sasl_ssl.sasl.enabled.mechanisms=OAUTHBEARER,SCRAM-SHA-512
listener.name.sasl_ssl.oauthbearer.sasl.server.callback.handler.class=org.apache.kafka.common.security.oauthbearer.secured.OAuthBearerValidatorCallbackHandler
listener.name.sasl_ssl.oauthbearer.sasl.jaas.config=org.apache.kafka.common.security.oauthbearer.OAuthBearerLoginModule required;
sasl.oauthbearer.expected.audience=account
sasl.login.connect.timeout.ms:15000
sasl.oauthbearer.token.endpoint.url=https://kafka.example.local:8443/realms/kafka/protocol/openid-connect/token
sasl.oauthbearer.jwks.endpoint.url=https://kafka.example.local:8443/realms/kafka/protocol/openid-connect/certs
sasl.oauthbearer.expected.issuer=https://kafka.example.local:8443/realms/kafka
```

Keycloak

Kafka

Settings | Credentials | Keys | Roles | Client Scopes | Mappers | Scope | Revocation | Sessions | Offline Access | Clust

Client ID	kafka
Name	
Description	
Enabled	ON
Always Display in Console	OFF
Consent Required	OFF
Login Theme	
Client Protocol	openid-connect
Access Type	confidential
Standard Flow Enabled	OFF
Implicit Flow Enabled	OFF
Direct Access Grants Enabled	ON
Service Accounts Enabled	ON
OAuth 2.0 Device Authorization Grant Enabled	OFF
OIDC CIBA Grant Enabled	OFF
Authorization Enabled	OFF
Front Channel Logout	OFF


Service-account-kafka
Details
Attributes
Credentials
Role Mappings
Groups
Consents
Sessions
ID
3f61728f-1ee7-4f87-a23b-6c87f92acc22
Created At
12/12/22 11:42:06 PM
Username
service-account-kafka
Email
First Name
Last Name
User Enabled
ON
Email Verified
OFF
Required User Actions
Select an action...
Impersonate user
Impersonate
Save
Cancel

Client

```
security.protocol=SASL_SSL
sasl.mechanism=OAUTHBEARER
sasl.login.callback.handler.class=org.apache.kafka.common.security.oauthbearer.secured.OAuthBearerLoginCallbackHandler
sasl.login.connect.timeout.ms=15000
sasl.oauthbearer.token.endpoint.url=https://kafka.example.local:8443/realms/kafka/protocol/openid-connect/token
```

- sasl.jaas.config=org.apache.kafka.common.security.oauthbearer.OAuthBearerLoginModule required \
- clientId="kafka" \
- ClientSecret="<>";
-