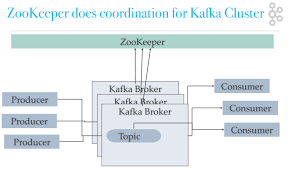
**Kafka Terminology:**

* Kafka cluster: whole system contains **brokers**, and is the media for **producer** and **consumer** to communicate with each other
* Kafka producer: view producer as the client side – it provides events to the cluster
* Kafka consumer: read streams of data from the cluster
* Kafka stream: stream of message that is processed in the cluster
* Source connector – connector for helping us to import data from external systems (**Producer**) into **kafka topic**
* Sink connector - connector for helping us to deliver data from topic to **Consumer**
* Apache zookeeper - broker registered with zookeeper; It serves as a backend for maintaining reliable distributed system

How?

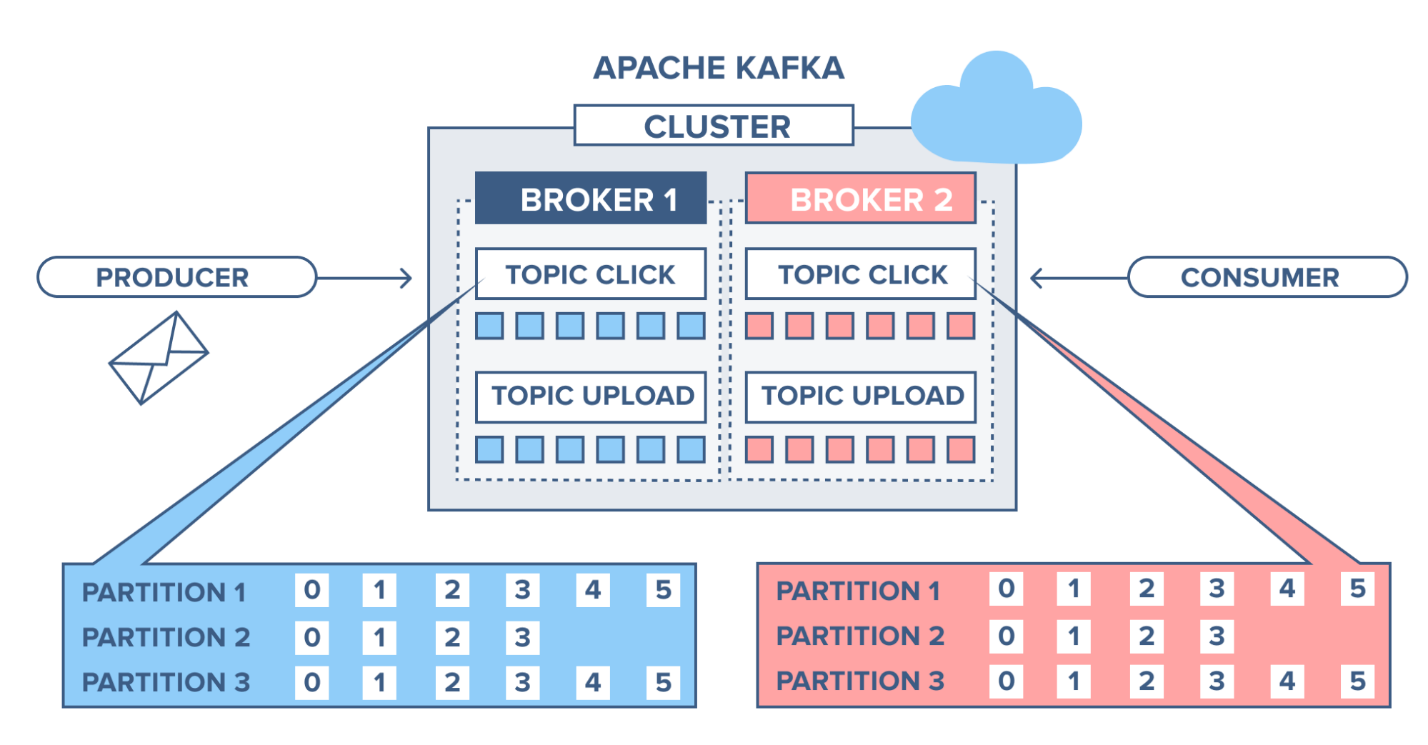
* + Serves as a microservice to maintain **metadata** for kafka cluster; It store the relationship of which brokers are at which part of the cluster; It store the **broker leader**(即当前数据通过哪一个broker) in a certain **partition**



<https://medium.com/@logeesan/zookeeper-in-kafka-ce31b3dd55b1>

Zoo Keeper(存Broker的Metadata，哪个data存在哪个broker)

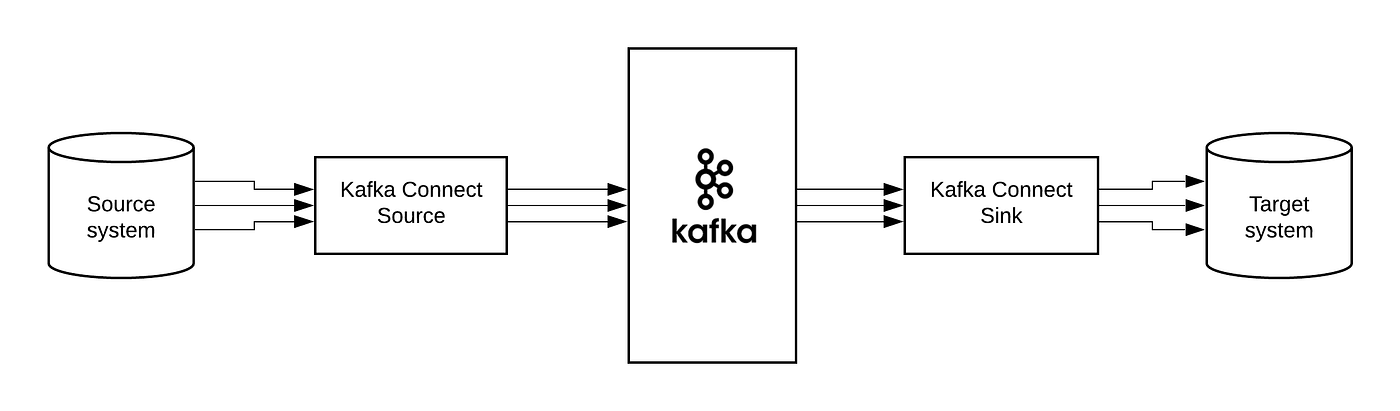
* **Topic** is an entity in Kafka, and message/event will be included in the topic and being processed from producer to broker/platform to consumer(pulling)
* **What’s the relationship between Topic, Broker, and Cluster:**



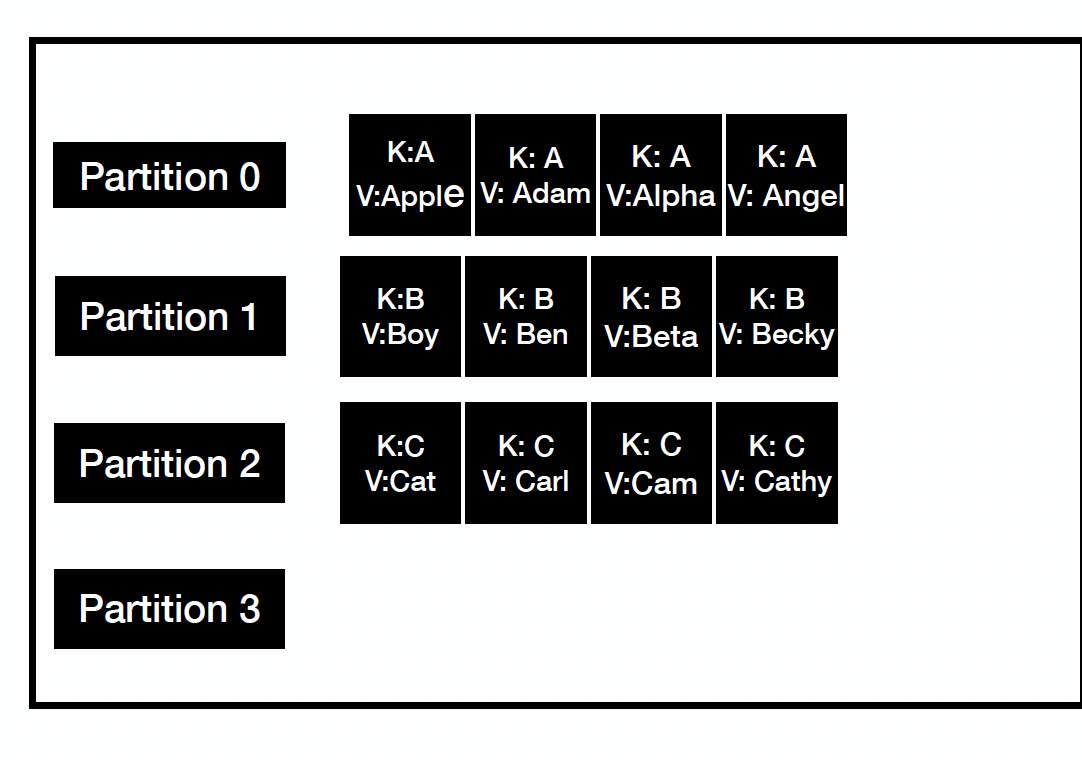
<https://www.cloudkarafka.com/img/blog/apache-kafka-partitions-topics.png>

每个小块是 Partition

* Each **message** lives in a partition in the topic, a topic contains one/more partitions; Partition is an immutable sequence/queue. Each partition is independent from others
* Each **item** in the partition is assigned with a sequential number called **offset**
* each message contains 2 elements:
  + key(optional)
  + Value



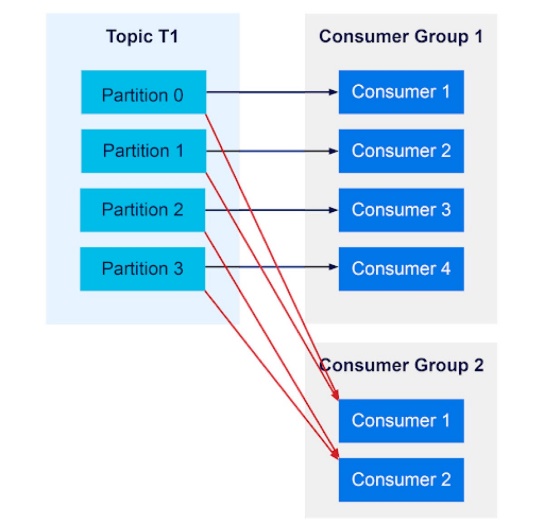
[https://miro.medium.com/v2/resize:fit:1400/1\*iXgWtgUfMSMnK82kVf9ccg.png](https://miro.medium.com/v2/resize:fit:1400/1*iXgWtgUfMSMnK82kVf9ccg.png)



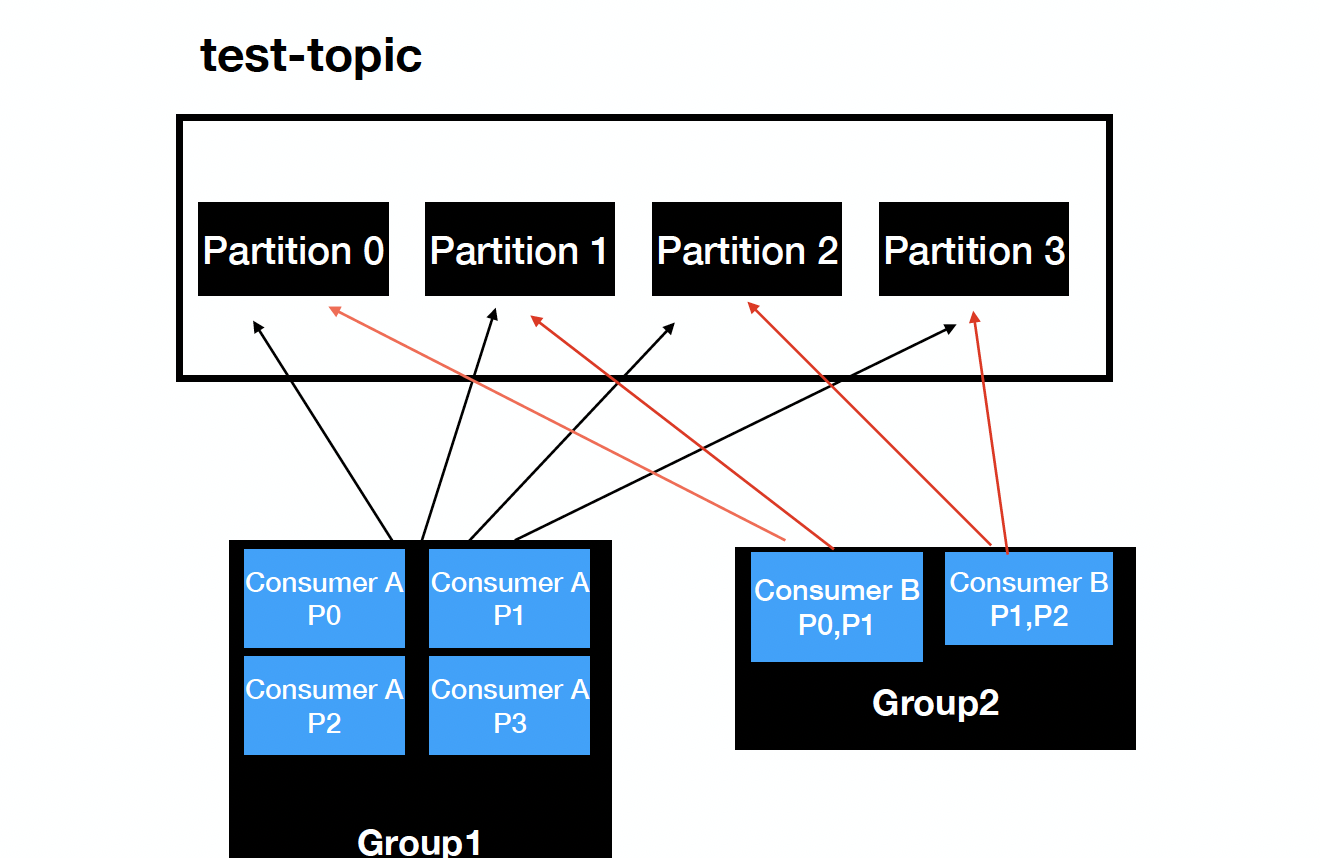
**Consumer:**

* **consumer offset** behaves like a bookmark insert into the queue for the consumer to read message from
* Consumer group - used for scalable message consumption; Consumer group is a set of consumers which cooperate to consume data from some topics
* With such group, the records will effectively be load-balanced over consumer instances, and in this way you can ensure parallel processing of records from a topic
* **Broker** manages the consumer group (在broker定义中决定), like a mapping in the
* **Commit log & retention policy**: retention policy has a default value of 7 days [how long events/message should be retained (before deleted)]

这些Groups 只接受来自 某一个P的数据.,



<https://images.idgesg.net/images/article/2023/04/apache-kafka-consumer-groups-05-100940401-large.jpg?auto=webp&quality=85,70>

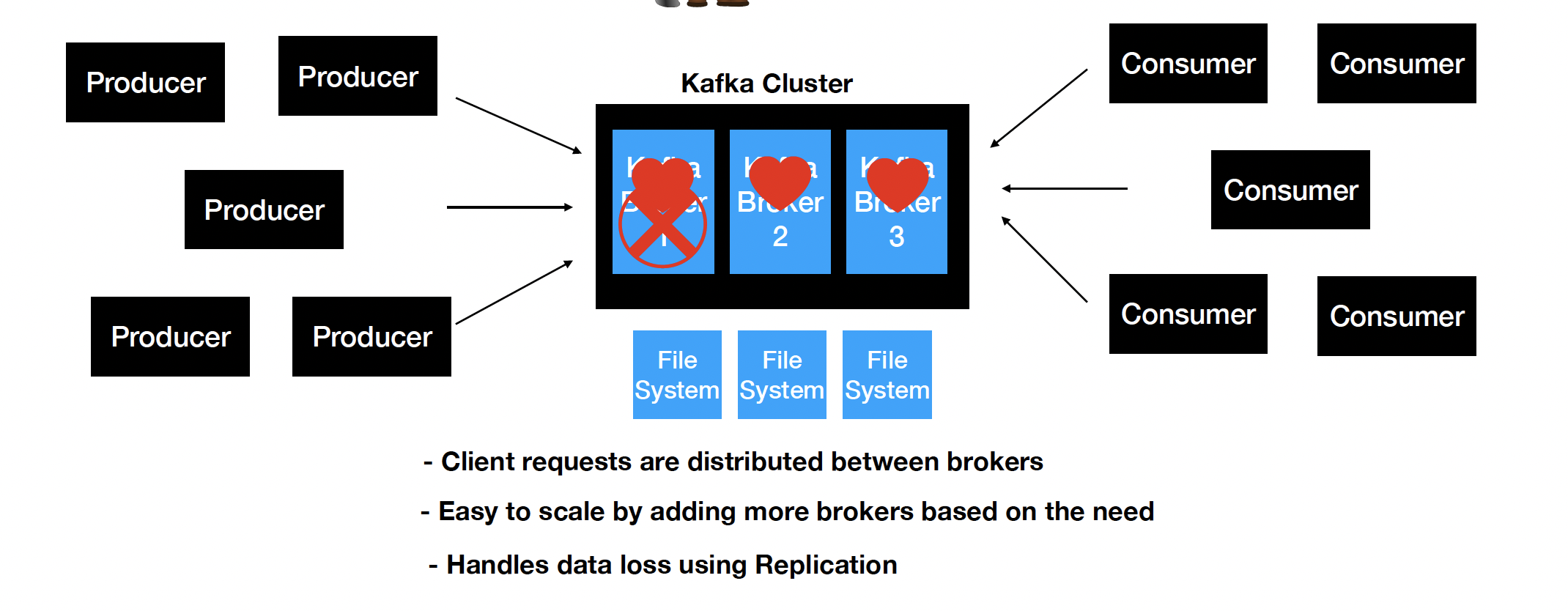


**How we combine all these components together and let them work?**

- Requests will be distributed throughout brokers

- We can add more brokers as needed

- Use replication to handle data lose



But how requests are distributed?

* According to the load, loads are evenly distributed between each brokers

How to handle data loss?

* use replication
* ‘—replication-factor 3’’
* In sync replica

Let’s say broker 1 is down for some reason, then if we set replication factor to 3, then we have 2 more replication factor, so that **zookeeper** can get notified about the failure and assign new leader so that the message can go through to the new broker[this switch is controlled by cluster] 不需要手动设置

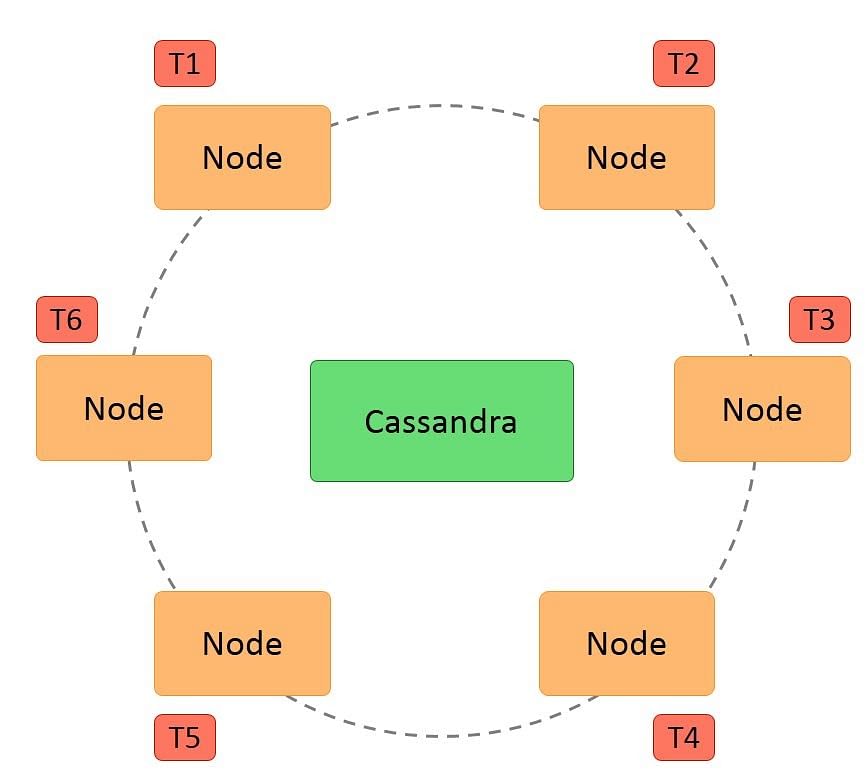
**Course 2:**

How to use **event streaming** in microservices?

* act like a communication layer in-between microservices
* Producer send events to platform(kafka cluster), platform send events to consumers
* Store events in queue
* Kafka differs from traditional streaming service as it’s a distributed streaming system, and any consumer can access the message from the platform
* Use case: Uber drive, food delivery notification, real-time order tracking,

**Cassandra brief introduction**

* Cassandra is a NoSQL database that often used in distributed system where there’re frequently streaming data
* Ring architecture, no master node, no single point of failure
* We can use kafka with Cassandra; Cassandra can serves as a long term storage database



<https://www.simplilearn.com/ice9/free_resources_article_thumb/Apache%20Cassandra%20Architecture/CassandraArchitecture_1.jpg>

**Idempotent design**

* Idempotence: no matter how many times we execute a workflow, the same result will always be returned
* When used in REST API, it means the API will not change the state of the system so that the result will always be the same; GET can be one example
* Idempotency in Kafka, aka **Kafka Excatly Once** Semantics: A mechanism to prevent process the same message multiple times
  + Assign Producer ID and sequence id to the message and send to the cluster;
  + Sequence number is monotonically increasing;
  + If the number is not exactly one larger than the last one, producer need to resend the message
* Other types of semantics?
  + **At least once**: high chance of duplication

Retro:

What is side car pattern?

Gives 2 examples of distributed systems?

What is distributed tracing?

Advantage for microservice?

Resources:

Learn terms of Kafka: <https://www.cloudkarafka.com/blog/part1-kafka-for-beginners-what-is-apache-kafka.html>

Cassandra: <https://medium.com/cloud-computing-management/understanding-the-basics-of-apache-cassandra-6e813f975720>

**Maven:**

**Gradle:**

Definition?

* Maven is a build tool that helps in building and documenting the Java project

What it does?

* Generate source code
* Generate documentation from source code
* Compile source code
* provide right project structure and dependency
* Packaging of the compiled codes into JAR files – used for building and deployment

Form?

* XML file(pom.xml) – project & configuration details
* Can integrate 3rd party dependency into Maven XML file

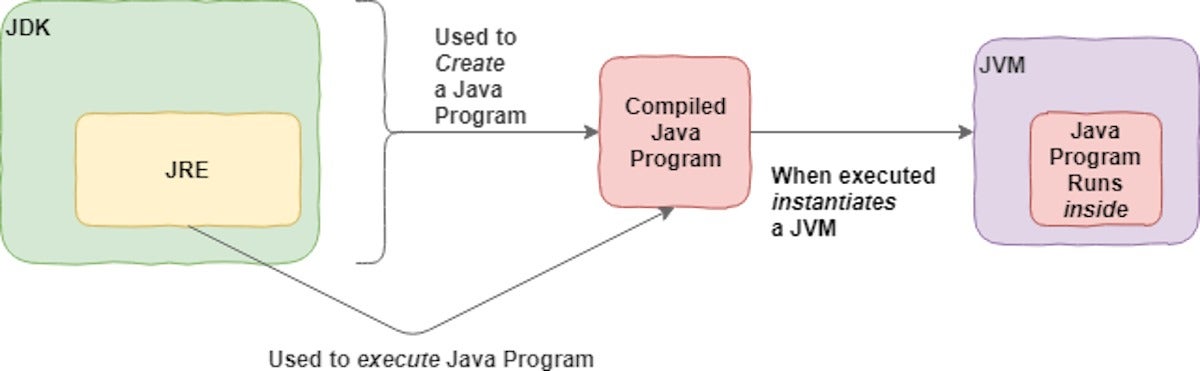
**JDK, JVM, JRE**

- 3 Cores of Java development – Java Development Kit, Java Virtual Machine, Java Runtime Environment

- JRE creates JVM and loads programs into it

- JDK provides necessary tools to write Java that can be run by JVM and JRE

- Java use Byte interpreter, to convert codes into Byte and let JVM execute the program

****

**API**

* What is Request URL & Body？What are their usage?
* Endpoints/REST API
* Third party API
* We have an application for second hand book buy and sell – API create a book into our database
* API request body:
  + Name, price, author, image
  + JSON: dictionary – { “Names”: {“book1”, “book2”}, “Price”: {100, 100}, “Author”: “AWS”}
* {"menu": {
* "id": "file",
* "value": "File",
* "popup": {
* "menuitem": [
* {"value": "New", "onclick": "CreateNewDoc()"},
* {"value": "Open", "onclick": "OpenDoc()"},
* {"value": "Close", "onclick": "CloseDoc()"}
* ]
* }
* }}

**Lombok**

* Lombok helps our backend application automatically generate getter and setters
* Normally, we need to create functions, to give models value
* Reduce boilerplate code – code that are repeated in lots of places with no variation
* Easy to use! Just insert **@Data** as the annotation on top of the entity class
* <https://projectlombok.org/>

@Data

Class A {

}

Annotation

@Data

Model: Book

{

Id

Name

Price

XXXX

}

Book1.getId()

**Database**

* Relational database – MySQL, PostgreSQL

- Table based storage. Developed since 1970s

- MySQL workbench

* NoSQL database – MongoDB

- Documents/JSON based storage. Developed since late 2000s

- More Flexibility and scalability, but less isolation and durability

* Cache type database – Redis [a type of NoSQL]

- key-value based storage

**Backend driven concept – what is the advantage?**

**If something really bad happen (i.e. severe bug found), what should we do?**

**Spring Boot**

* Annotation – learn more at <https://www.baeldung.com/spring-core-annotations>

@Entity, @Table(NAME = “Food”), @Data, @Column(name = “user\_name”), @Configuration, @Bean – dependency, cusomtized , @Autowired

Configuration: 加在Class-level 的，表明这个class是一个configuration文件  
Bean: 定义Spring的一些设置或function （function-level）  
Autowired: 使用Bean定义后的东西

Class WebSecurity {

@Bean

Public void A () {

}

}

Class B {

@Autowired – dependency injector

WebSecurity webSecutiry;

}

* Download a free spring boot project from their website - <https://start.spring.io/>
* IntelliJ IDE

@Entity – data model

@Table(NAME = “Food”)

Public class Food {

@Column(name = “user\_name”)

Private String username;

}

**Course3:**

**KafkaTemplate – from code perspective**

* Produce records into kafka topic
  + Similar to JDBCTemplate for DB
  + Kafkatemplate.send(events/message) -> serializer[key.serializer, value serializer] -> partitioner -> record accumulator [recordBatch]
  + Finally lead to events **cluster**
  + How to configure?
    - In **application.yml**
    - Servers
    - Key serializer
    - Value serializer

Example: 这是整个project的设置

spring:

config:

activate:

on-profile: local

kafka:

topic: TEST0-EVENTS

template:

default-topic: TEST0-EVENTS

producer:

bootstrap-servers: localhost:8080, localhost:8081

key-serializer: org.apache.kafka.common.serialization.IntegerSerializer

value-serializer: org.apache.kafka.common.serialization.StringSerializer

properties:

acks: all // 保证接收端全部收到消息

retries: 10

admin:

properties:

bootstrap.servers: localhost:8080,localhost:8081

**More on** **Spring application yml file:**

If we have different environments, we can have different application.yml files

We write the following code to create **kafkatemplate** programmatically

Spring:

Profiles: local

Kafka:

Producer:

Bootstrap-servers:

Key-serializer:

Value-serializer:

Spring has kafka auto configurer to scan KafkaTemplate.class file and if this file is missing or empty, spring will look for spring.kafka, and setup everything according to our application.yml file

**Endpoints/REST API**:

**Post new events**

**Put/Update existed events**

**How to achieve these 2 operations in code perspectives？**

**Client send requests**

**-> API layer on backend**

**-> Controller.Class has all our API interface**

**-> Service, producer, kafkatemplate.send(events from our API)**

**-> send events to cluster**

**-> consumer has a listener to listen to all the incoming events**

**-> service, consumer**

**-> process with our events and save to database**

**Producer service**:

KafkaTemplate<Integer, String> kafkaTemplate;

Template will be injected into **producer** class

**sendEvents**(Event) {

Integer key = Event.getId();

String value = objectMapper.writeValueAsString(Event);

var **completableFuture** = kafkaTemplate.send(key, value); -> cluster

// this can result in success or failure

// Thie method is an **asynchronous** call – meaning that we will get the result regardless our main process/time – your functionality will not failed if kafka is failed

// blocking call – get metadata about kafka cluster

// send message – return completablefuture

return completableFuture

.whenComplete((sendResult, throwable) -> {

if (throwable != null) {

handleFailure(key, value, throwable);

} else {

handleSuccess(key, value, sendResult);

}

});

// Option 2 – block and wait until the message is sent to the Kafka

= kafkaTemplate.send(key, value).get(); (同步的，因为会等待结果产生后再离开)

// This will wait 3 seconds before timeout exception is thrown

= kafkaTemplate.send(key, value).get(3, TimeUnit.SECONDS);

}

**KafkaAdmin to create topic**

* Create a Bean KafkaAdmin in SpringConfiguration
* Create a Bean NewTopic in SpringConfiguration
* For example:

@Configuration

public class AutoCreateConfig {

@Value("${spring.kafka.topic}") // ${} 的内容，就会去该内容下面找

public String topic; //

@Bean

public NewTopic eventsCretion(){

return TopicBuilder.name(topic)

.partitions(3)

.replicas(3)

.build();

}

}

// Re-usability: 这个样子，yml里面的值，可以被其它的文件所读取

We are gonna setup kafka using Docker - container:

List topics in the cluster:

docker exec --interactive --tty kafka \

kafka-topics --bootstrap-server kafka:XXXX --list

Describe topics:

docker exec --interactive --tty kafka \

kafka-topics --bootstrap-server kafka:XXXX --describe

Describe a topic:

docker exec --interactive --tty kafka \

kafka-topics --bootstrap-server kafka1:XXX --describe \

--topic test-topic

This can give all information about a topic such as TopicId, PartitionCount, ReplicationFactor

**What is Integration test and unit test?**

* What components needed to be tested for integration test?
  + XXX, XXX, XXX
* In test folder in java project
* Junit
* Specify test env, add test related annotation, get kafka configuration through @TestPropertySource from application.yml
  + @SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.RANDOM\_PORT)
  + @EmbeddedKafka(topics = {"test-events"}, partitions = 3)
  + @TestPropertySource(properties = {"spring.kafka.producer.bootstrap-servers=${spring.embedded.kafka.brokers}","spring.kafka.admin.properties.bootstrap.servers=${spring.embedded.kafka.admin.XXX }"})
* For test case, provide ‘given’ ‘when’ ‘then’
* ‘given’
  + Event, headers(不需要太了解), request (API,包装Event)
* ‘when’ (整体流程都写在这里)
  + Send API REQUEST using restTemplate.exchange -send HTTP request
  + ResponseEntity<Event> responseEntity = restTemplate.exchange("/v1/sendEvent", HttpMethod.POST, request, Event.class);
  + ‘then’
    - AssertEquals(HttpStatus.CREATED, responseEntity.getStatusCode());
    - Assert consumer get the record/event

**Unit test:**

* Spring MockMVC
* Mock api request, expect a value (模拟个值)

Given such situation, design set of unit test cases:

We want to stream events from A to B, we have the following functions ready to use:

SendEventsAPI – (only receive the request, without any processing)

SendEventsToKafka – (process the request from api, and send events to Kafka)

Event model – Database Model

What test cases are needed here？

SendEventsAPI:  
1. Empty  
- return Empty Response  
2. Normal  
- return Successful Response  
3. Abnormal  
- return Error Response

A screenshot of a computer

Description automatically generated

A screenshot of a chat

Description automatically generated

**Project Design:**

**Kafka Producer project:**

REST API Controller for send events to kafka cluster

* Create new events
* Update existed events

HTTP Error handler for useful and clear error message/readable response body

Event model

Event type model

Shopping item model

Kafka event producer

* kafkaTemplate
* Topic usage
* Send events method – sending to kafka cluster

Configuration:

Topic defination

Topic creationevent

Producer configuration

Test:

Integration test:

End to end workflow testing – define input, go through REST API, transfer events to producer, producer transfer events to cluster, confirm consumer can receive the exact events(assert value)

Unit test(mockMVC):

Controller test – when XXX thenReturn XXX, mockMvc.perform andExpect

Producer test

Util:

Docker – used to hold kafka

**Kafka Consumer project:**

Kafka Consumer Listener -listen to any incoming events, send to service to proceed the message/events

* @KafkaListener – handle concurrency

(model/entity matches with producer)

Event model

Event type model

Shopping item model

Retry mechanism

Events processor service

* Read events
* Validation
* Save different event to database
* Error handling

**Design a full stack ecommerce platform – front end, backend, database, cache**

**<https://docs.google.com/document/d/1swOwLAEo7LhE0bznNvfySipIoikW5bdTIHDL6pa3hp0/edit?usp=sharing>**

for kafka setup:

kafka1 | [2023-10-29 04:08:37,308] INFO Registered broker 1 at path /brokers/ids/1 with addresses: INTERNAL://kafka1:19092,EXTERNAL://127.0.0.1:9092,DOCKER://host.docker.internal:29092, czxid (broker epoch): 27 (kafka.zk.KafkaZkClient)

setup kafka and zookeeper:

docker-compose up

start the producer to send message to the topic:

docker exec --interactive --tty kafka1 \

kafka-console-producer --bootstrap-server kafka1:19092 \

--topic test-topic

docker exec --interactive --tty kafka1 kafka-console-producer --bootstrap-server kafka1:19092 --topic test-topic

Start the consumer to listen to any message sent to topic:

docker exec --interactive --tty kafka1 \

kafka-console-consumer --bootstrap-server kafka1:19092 \

--topic shopping-events \

--from-beginning

docker exec --interactive --tty kafka1 kafka-console-consumer --bootstrap-server kafka1:19092 --topic shopping-events --from-beginning

go to the container:

docker exec -it kafka1 bash

create a topic using command line tool:

kafka-topics --bootstrap-server kafka1:19092 \

--create \

--topic test-topic1 \

--replication-factor 1 --partitions 1

List topic:

kafka-topics --bootstrap-server kafka1:19092 --describe

**Producer class:**

1: set up IntelliJ starter project

2: create model, curl file, endpoints, and test out the endpoints

3: download docker desktop and setup kafka and zookeeper

4: test out kafka functionality

5: write the producer functionality and test out the workflow

6: if want to compose multiple brokers, what should we do for the docker compose file

**Test class:**

Control shift T to create test

* Design test case
* Instruct on test setup
* Implement one of the test and Let students to implement themselves the following test

1: Create unit test

2: controller advice

3: Integration test

4: write update endpoint – mock interview

**Consumer class:**

1 Create project:

<https://start.spring.io/>

kafka

data jpa

h2 database： --- <http://localhost:8081/h2-console>

* Select \* from shopping\_event;
* Select \* from item;

validation

spring web

Lombok

2 Update gradle file

Poll events from cluster, learn more about listener:

[https://docs.spring.io/spring-boot/docs/current/reference/html/messaging.html#messaging.kafka.receiving](https://docs.spring.io/spring-boot/docs/current/reference/html/messaging.html" \l "messaging.kafka.receiving)

3 write the consumer kafka class

4 test out the integration between producer and consumer

5 mock interview on writing consumer service for store data to the database(model, service, repository)