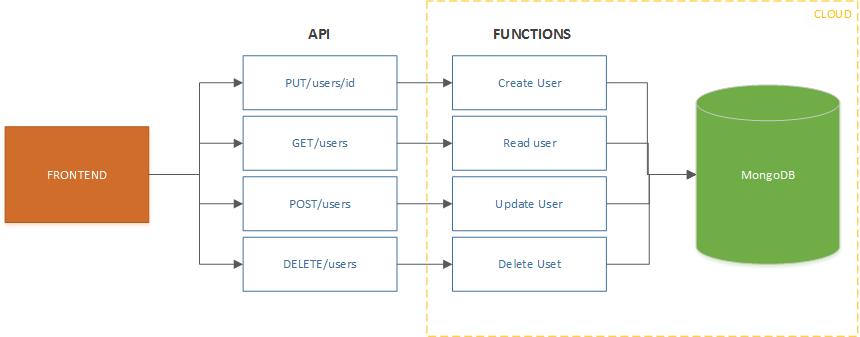
**Apache Kafka With Spring Boot Application**

* **Prerequisites:**

1. Minimum Java-8 should be installed in your Linux machine.
2. Spring tool Suite/IntelliJ
3. Gradle
4. Kafka
5. Kafka Tool Offset-Explorer

* **Work Flow for the Project:**

Transaction Database

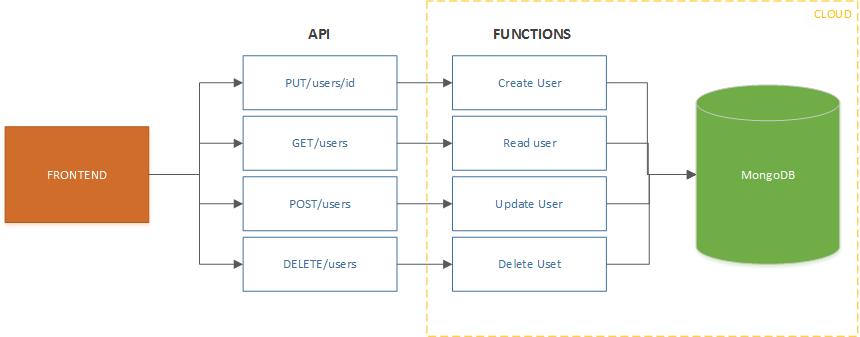


||||

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  | | --- | | Kafka Cluster |   Zookeeper-2181   |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | | Partitions | | | | | | | | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |   Topic🡪Multiple-Producers |

Kafka-Ecosystem

Consumer Database



* **Create a Spring Boot Project**
  + Create a Project from **Spring Initializer**.
    - Go to the [Spring Initializer.](https://start.spring.io/)
    - Enter a Group name
    - Mention the Artifact Id,
    - Add the following dependencies,
  1. Spring Web.
  2. Spring-boot-starter-cache
* Click on the Generate button, the project will be downloaded on your local system.
* Unzip and extract the project.
* Import the project in your IDE such as Spring Tool Suite.
  + Select File -> Import -> Existing Gradle Projects -> Browse -> Select the folder  -> Finish.
* Configure the Producer and Consumer.

We can either configure through **application. properties** or **Java Class with @Configuration.Here we will do producer with @Configuration and Consumer with application. properties file.**

* **Producer Configuration:**

**Method 1: Producer Confi**guration **using Java Class with @C**onfiguration.

|  |
| --- |
| @EnableKafka  @Configuration  public class KafkaConfiguration  {  @Value("${spring.kafka.bootstrap-servers}")  private String kafkaServer;  @Value("${spring.kafka.producer.key-serializer}")  private String keySerializer;  @Value("${spring.kafka.producer.value-serializer}")  private String valueSerializer;  @Bean  public ProducerFactory<String,User > userProducerConfig()  {  Map<String,Object> configuration=new HashMap<>(); configuration.put(ProducerConfig.BOOTSTRAP\_SERVERS\_CONFIG, kafkaServer); configuration.put(ProducerConfig.KEY\_SERIALIZER\_CLASS\_CONFIG, keySerializer); configuration.put(ProducerConfig.VALUE\_SERIALIZER\_CLASS\_CONFIG,valueSerializer);  return new DefaultKafkaProducerFactory<>(configuration);  }  @Bean  public KafkaTemplate<String ,User> userKafkaTemplate(){  return new KafkaTemplate<String,User>(userProducerConfig());  }  } |

**Method 2:** Producer Configuration using Application properties file.

|  |
| --- |
| **application.properties**  **spring.data.mongodb.uri=mongodb://localhost:27017/Banking**  **spring.kafka.topic=Multiple**  **spring.kafka.producer.bootstrap-servers=localhost:9092,localhost:9093,localhost:9094**  **spring.kafka.producer.key-serializer= org.apache.kafka.common.serialization.StringSerializer**  **spring.kafka.producer.value-serializer= org.apache.kafka.common.serialization.JsonSerializer** |

* **Consumer Configuration:**

**Method 1: Consumer Confi**guration **using Java Class with @C**onfiguration.

|  |
| --- |
| @EnableKafka  @Configuration  public class KafkaConsumerConfig {  @Value("${spring.kafka.consumer.group-id}")  String group;  @Value("${spring.kafka.consumer.bootstrap-servers}")  private String server;  @Value("${spring.kafka.consumer.auto-offset-reset}")  private String offsetRest;  @Value("${spring.kafka.consumer.key-deserializer}")  private String keyDeserializer;  @Value("${spring.kafka.consumer.value-deserializer}")  private String valueDeserializer;  @Value("${spring.kafka.consumer.properties.spring.json.trusted.packages}")  private String trustPackage;      @Bean      public ConsumerFactory<String, User> userConsumerFactory()      {         Map<String, Object> config = new HashMap<>();         config.put(ConsumerConfig.BOOTSTRAP\_SERVERS\_CONFIG,server);         config.put(ConsumerConfig.GROUP\_ID\_CONFIG, group);         config.put(ConsumerConfig.KEY\_DESERIALIZER\_CLASS\_CONFIG, StringDeserializer.class);         config.put(JsonDeserializer.TRUSTED\_PACKAGES, "\*");         config.put(ConsumerConfig.VALUE\_DESERIALIZER\_CLASS\_CONFIG,JsonDeserializer.class);         config.put(JsonSerializer.ADD\_TYPE\_INFO\_HEADERS, false);         return new DefaultKafkaConsumerFactory<>(config);      }      @Bean      public ConcurrentKafkaListenerContainerFactory<String, User> userKafkaListenerContainerFactory() {          ConcurrentKafkaListenerContainerFactory<String, User> factory = new ConcurrentKafkaListenerContainerFactory();          factory.setConsumerFactory(userConsumerFactory());          return factory;}} |

**Method 2:** Consumer Configuration using Application properties file.

**application.properties**

|  |
| --- |
| spring.kafka.topic: Multiple  spring.kafka.consumer.group-id: Multiple\_Consumer\_Group  spring.kafka.consumer.bootstrap-servers:localhost:9092  spring.data.mongodb.uri=mongodb://localhost:27017/ConsumerData  spring.kafka.consumer.auto-offset-reset: earliest  spring.kafka.consumer.key-deserializer: org.apache.kafka.common.serialization.StringDeserializer  spring.kafka.consumer.value-deserializer: org.springframework.kafka.support.serializer.JsonDeserializer  spring.kafka.consumer.properties.spring.json.trusted.packages=\* |

* This is the common class between all the instances.
* **User.java**

|  |
| --- |
| @Data  @AllArgsConstructor  @RequiredArgsConstructor  public class User {  public String name;  public int age;  public int salary;  } |

**Main Application:**

* Here in this application, we are having an API which accepts json data from the user.
* The controller class receives data from the user and process it to the service class.
* In the Service.java we are checking whether the json object is stored in the Database or not.
* If not we are returning message to the customer that some data is missing.
* In the Service.java we are having a logic to send the URL through the Rest Template binded with the Json object.
* **UserController.java**

|  |
| --- |
| @RestController  @RequestMapping("/user")  public class UserController  {  Logger logger = LoggerFactory.getLogger(UserController.class);  @Value( "${server.port}" )  private String serverPort;  @Autowired  private UserService userService;  @Autowired(required = true)  RestTemplate restTemplate;  @PostMapping("/insertUser")  public String sendMessage(@RequestBody User user) throws UnknownHostException, IOException  {  if(user.name==null){  return "UserName should not be empty";  }  else if(user.age<18){  return "age should be greater than <18";  }  else if(user.salary<=10000)  {  return "salary should be grater then 100";  }  else  {  return userService.insertData(user,serverPort);  }  }  } |

* **UserService.java**

|  |
| --- |
| @Service  public class UserService2  {  private String a="8081";  private String b="8082";  private String c="8083";  private String d="8004";  private String []arr={a,b,c,d};    final Logger logger=LoggerFactory.getLogger(UserService2.class);  @Autowired(required = true)  private UserRepository userRepository;  @Autowired  MongoTemplate mongo;  public String insertData(User user, String serverPort)  {  logger.info(user.name +" "+user.age+" "+user.salary);    if(userRepository.save(user)!=null)  {  return sendToProducer(user,serverPort);  }  else  {  return "The data is not stored into transaction database";  }  }    public String getPortNumber()  {  PortConfig portVerification=new PortConfig();  Random r=new Random();  int randomNumber=r.nextInt(arr.length);  String portNumber=arr[randomNumber];  String portNumber=portNumber();  if(portVerification.isSocketAlive(portNumber)==false)  {  return getPortNumber();  }  else  {  return portNumber;  }  }  public String sendToProducer(User user,String serverPort)  {  RestTemplate template=new RestTemplate();  String portNumber=getPortNumber();  String result = template.postForObject("http://localhost:"+serverPort+"/user/insertUser",user,String.class);  ] return result;    if(portNumber.equals(a))  {  String result = template.postForObject("http://localhost:"+serverPort+"/user/insertUser",user,String.class);  return result;  }  else if(portNumber.equals(b))  {  String result = template.postForObject("http://localhost:"+serverPort+"/user/insertUser",user,String.class);  return result;  }  else if(portNumber.equals(c))  {  String result = template.postForObject("http://localhost:8083/user/insertUser",user,String.class);  return result;  }  else  {  return "some thing is wrong ";  }  }  } |

* **UserRepository.java**

|  |
| --- |
| @Repository  @Component  public interface UserRepository extends MongoRepository<User, String> {  } |

* **PortConfig.java**

|  |
| --- |
| @Configuration  public class PortConfig  {  private final String HOSTNAME="localhost";  @Value("${timeout}")  private int timeout;  public boolean isSocketAlive(String portDetails)  {  int port=Integer.parseInt(portDetails);  SocketAddress socketAddress = new InetSocketAddress(HOSTNAME, port);  Socket socket = new Socket();  log("HOSTNAME: " + HOSTNAME + ", port: " + port);  try {  socket.connect(socketAddress, timeout);  socket.close();  } catch (SocketTimeoutException exception) {  System.out.println("SocketTimeoutException " + HOSTNAME + ":" + port + ". " + exception.getMessage());  } catch (IOException exception) {  System.out.println(  "IOException - Unable to connect to " + HOSTNAME + ":" + port + ". " + exception.getMessage());  }  return socket.isConnected();  }  private static void log(String string) {  System.out.println(string);  }  } |

**Producer Application:**

* Here the operational sequence is as follows:
  + The producer controller receives data from the transaction application.
  + And from the controller class we are sending data to the kafka cluster based on the **broker-id**, **topic-id** and by using the **@KafkaTemplate.**
  + From here the data is posted to the topic and its partitions in the Kafka cluster.
* The Producer controller class looks like:

|  |
| --- |
| @RestController  @RequestMapping("/user")  public class KafkaProducerController  {  Logger logger=LoggerFactory.getLogger(KafkaProducerController.class);    @Autowired  private KafkaTemplate<String,User> userKafkaTemplate;  @Value("${spring.kafka.topic}")  private String TOPIC;    @PostMapping("/insertUser")  public String sendMessage(@RequestBody User user)  {  System.out.println(" i am in Producer");  System.out.println(user.name +" "+user.age+" "+user.salary);  User student=new User();  student.name=user.name;  student.age=user.age;  student.salary=user.salary;  ListenableFuture<SendResult<String, User>> future = userKafkaTemplate.send(TOPIC, student);  future.addCallback(new ListenableFutureCallback<SendResult<String, User>>()  {  @Override  public void onSuccess(SendResult<String, User> result)  {  logger.info("Sent message: " + student  + " with offset: " + result.getRecordMetadata().offset());  }  @Override  public void onFailure(Throwable exception)  {  logger.error("Unable to send message : " + student,exception);  }  });  return "Json Object Sucessfully published";  }  } |

* From here the json object is published to the topic and it is send to the partitions of the topic.
* The application.properties file of producer application looks like

**application.properties:**

|  |
| --- |
| **spring.data.mongodb.uri=mongodb://localhost:27017/Banking**  **spring.kafka.topic=Multiple**  **spring.kafka.producer.bootstrap-servers=localhost:9092,localhost:9093,localhost:9094**  **spring.kafka.producer.key-serializer= org.apache.kafka.common.serialization.StringSerializer**  **spring.kafka.producer.value-serializer= org.apache.kafka.common.serialization.JsonSerializer** |

**Consumer Application:**

* In Consumer instance we are having the process as follows.
  + The **consumer class** consumes the message based on **topic-id** and **group-id** and process the message for the further operations.
  + Form consumer class the data is further processed to the **service.java** from which we are dumping the consumer messages into the **consumer database**.
  + We will be doing further operation on Consumer database based on some conditions i.e., **Fetching the last 24 hours data** and sending it to the main database.
* **Consumer.java**

|  |
| --- |
| @Service  @Configuration  public class Consumer  {  private static final Logger logger=(Logger) LoggerFactory.getLogger(Consumer.class);    @Autowired  private ConsumerService consumerService;    @KafkaListener(topics="${spring.kafka.topic}",groupId ="${spring.kafka.consumer.group-id}")  public void consumeMessageJson(User user)  {  logger.info(String.format("User created as -> %s", user));  consumerService.consumeMessage(user);  }  } |

* **ConsumerService.java**

|  |
| --- |
| @Service  public class ConsumerService  {  private static final Logger logger=(Logger) LoggerFactory.getLogger(ConsumerService.class);  @Autowired  private ConsumerRepository consumerRepository;  public void consumeMessage(User user)  {  consumerRepository.save(user);  logger.info("consumer consumed the message and it is --> "+user.name);  }  } |

* **ConsumerRepository.java**

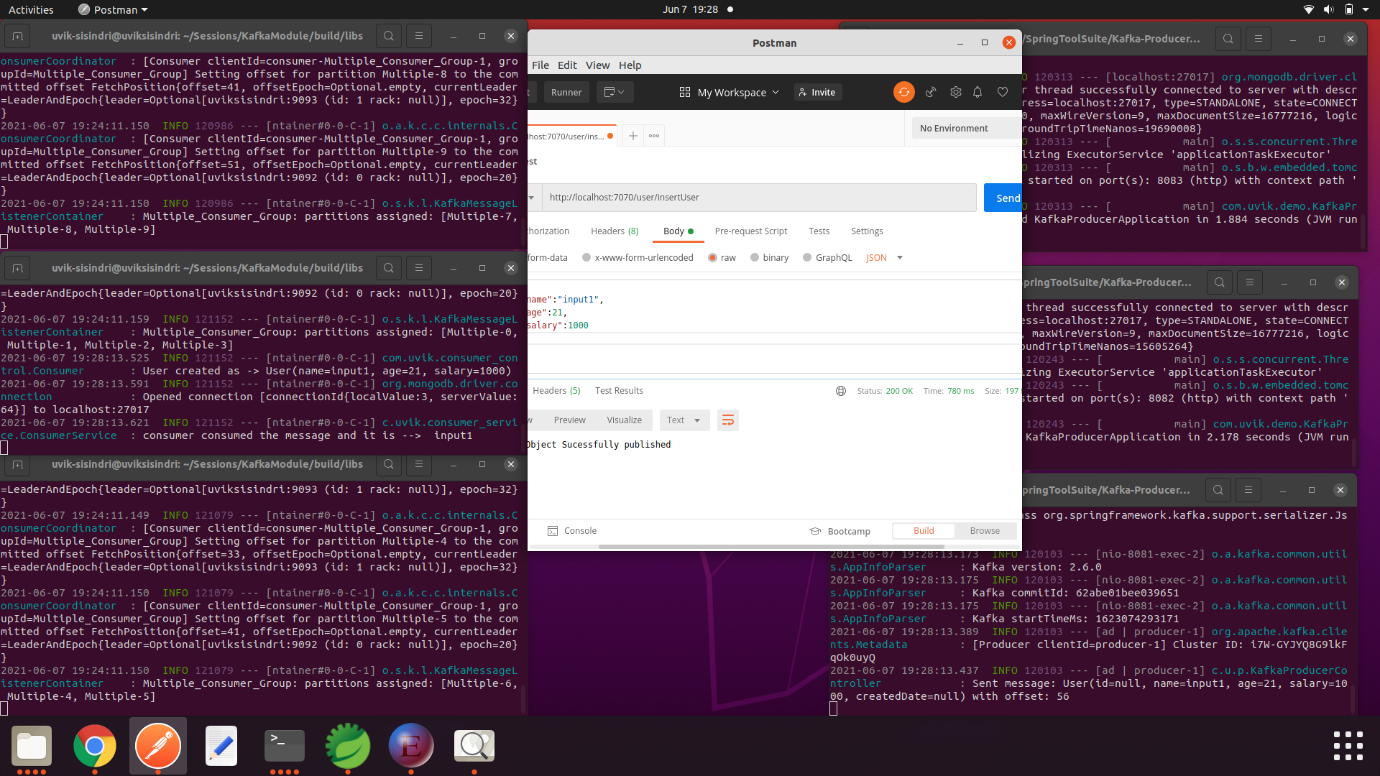
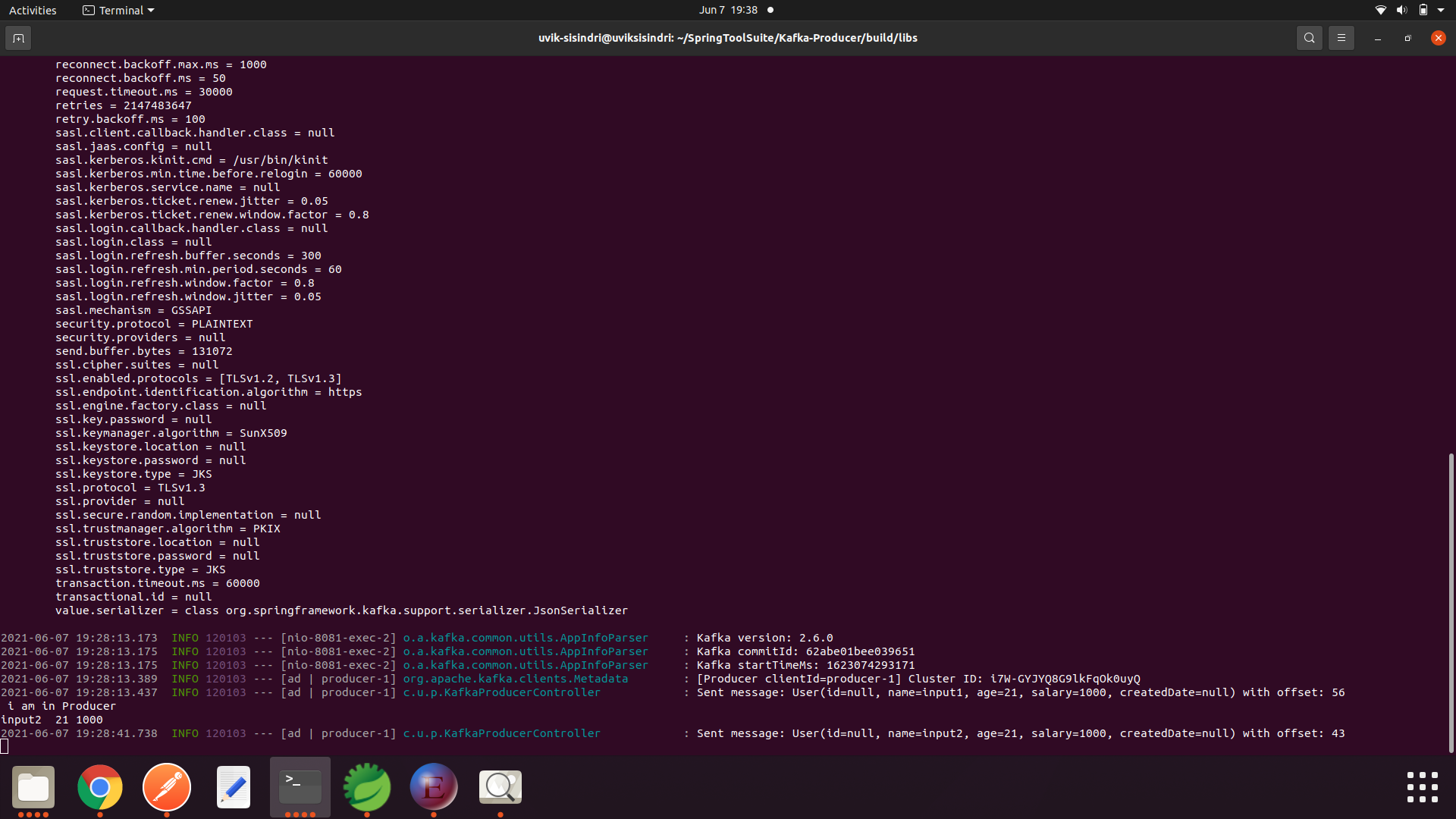
|  |
| --- |
| @Repository  @Component  public interface ConsumerRepository extends MongoRepository<User, String>  {  } |

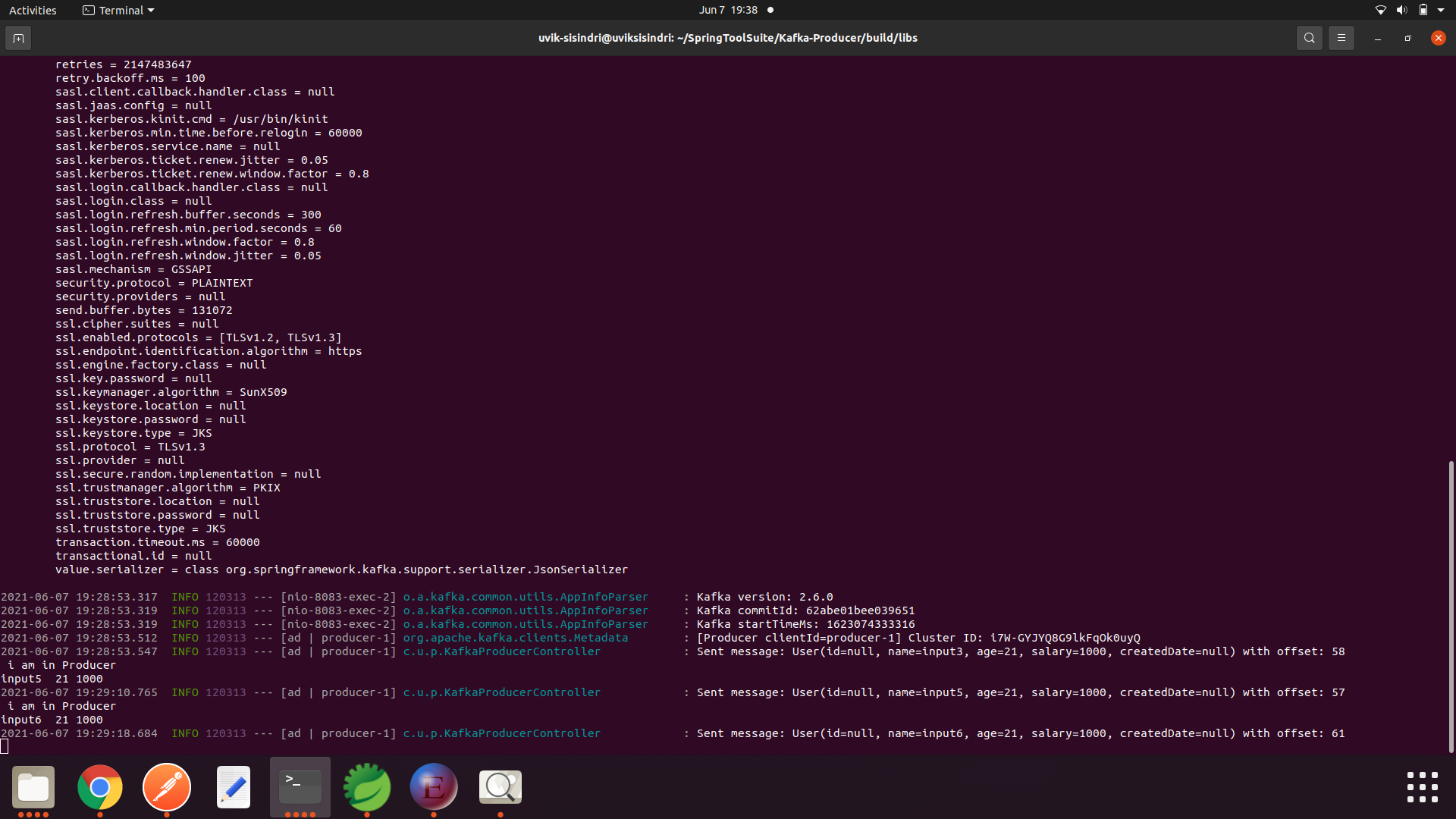
* **Application.properties file is as follows**

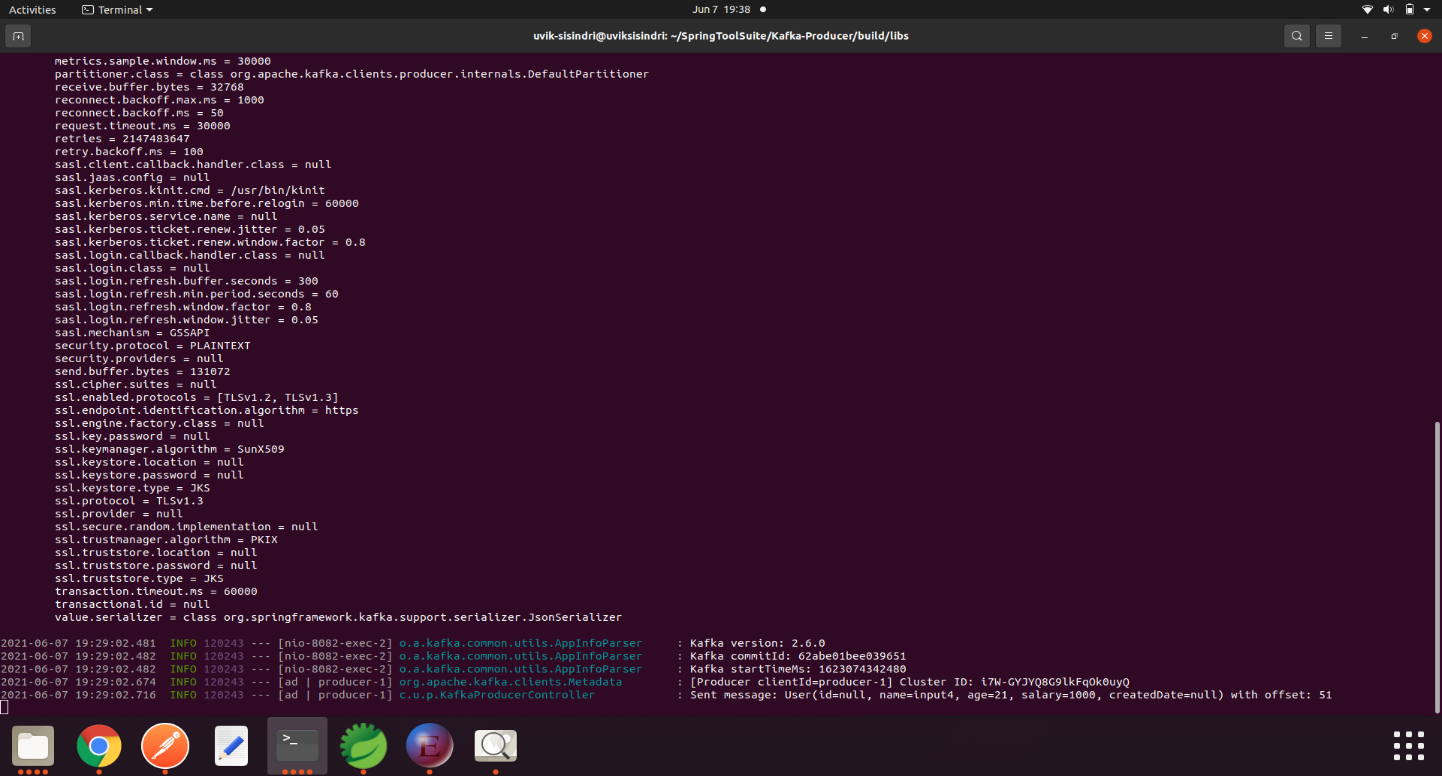
**application.properties**

|  |
| --- |
| spring.kafka.topic: Multiple  spring.kafka.consumer.group-id: Multiple\_Consumer\_Group  spring.kafka.consumer.bootstrap-servers:localhost:9092,localhost:9093,localhost:9094  spring.data.mongodb.uri=mongodb://localhost:27017/ConsumerData  spring.kafka.consumer.auto-offset-reset: earliest  spring.kafka.consumer.key-deserializer: org.apache.kafka.common.serialization.StringDeserializer  spring.kafka.consumer.value-deserializer: org.springframework.kafka.support.serializer.JsonDeserializer  spring.kafka.consumer.properties.spring.json.trusted.packages=\* |

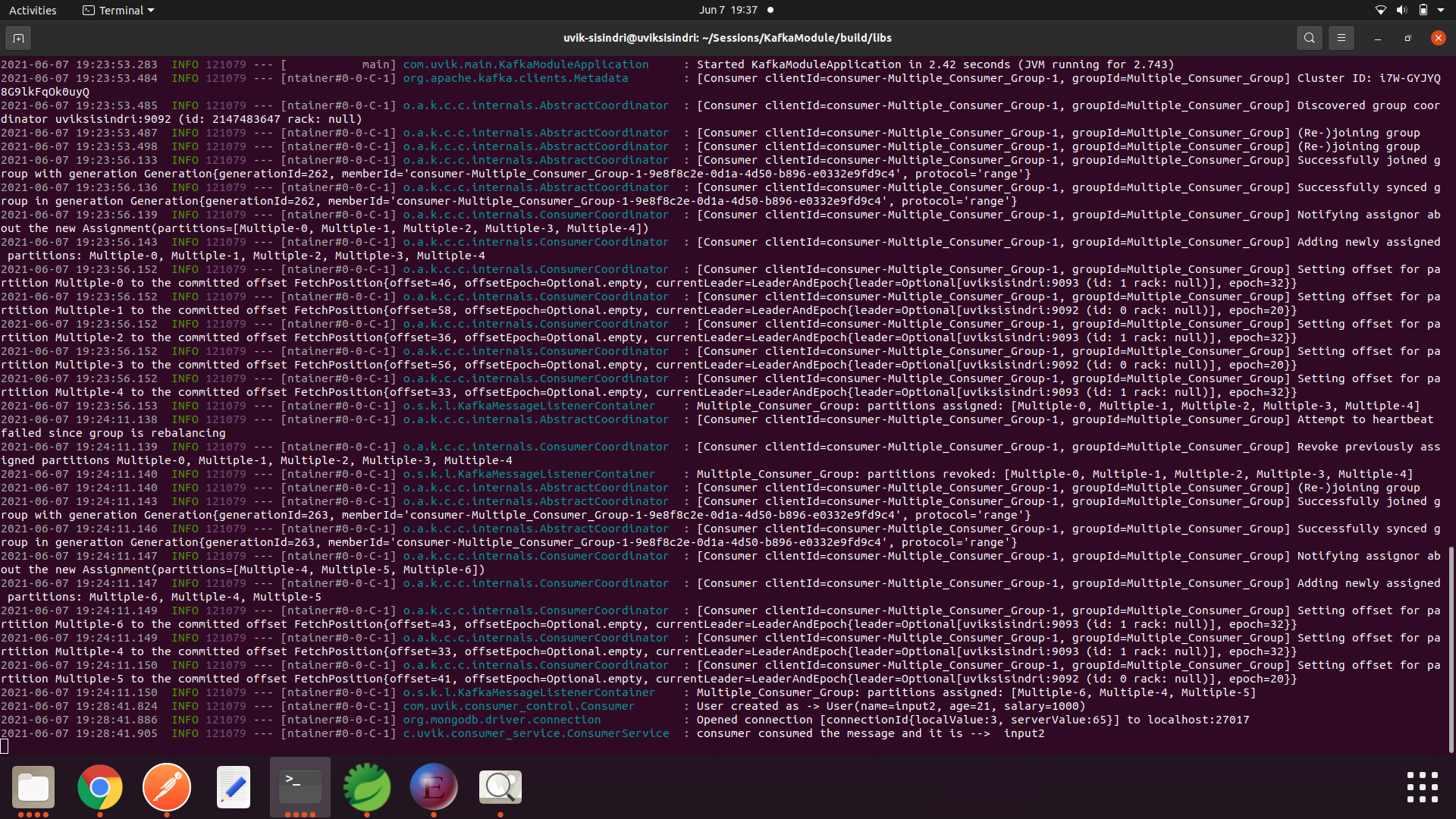
**How to achieve Multiple Producers and Multiple Consumers**

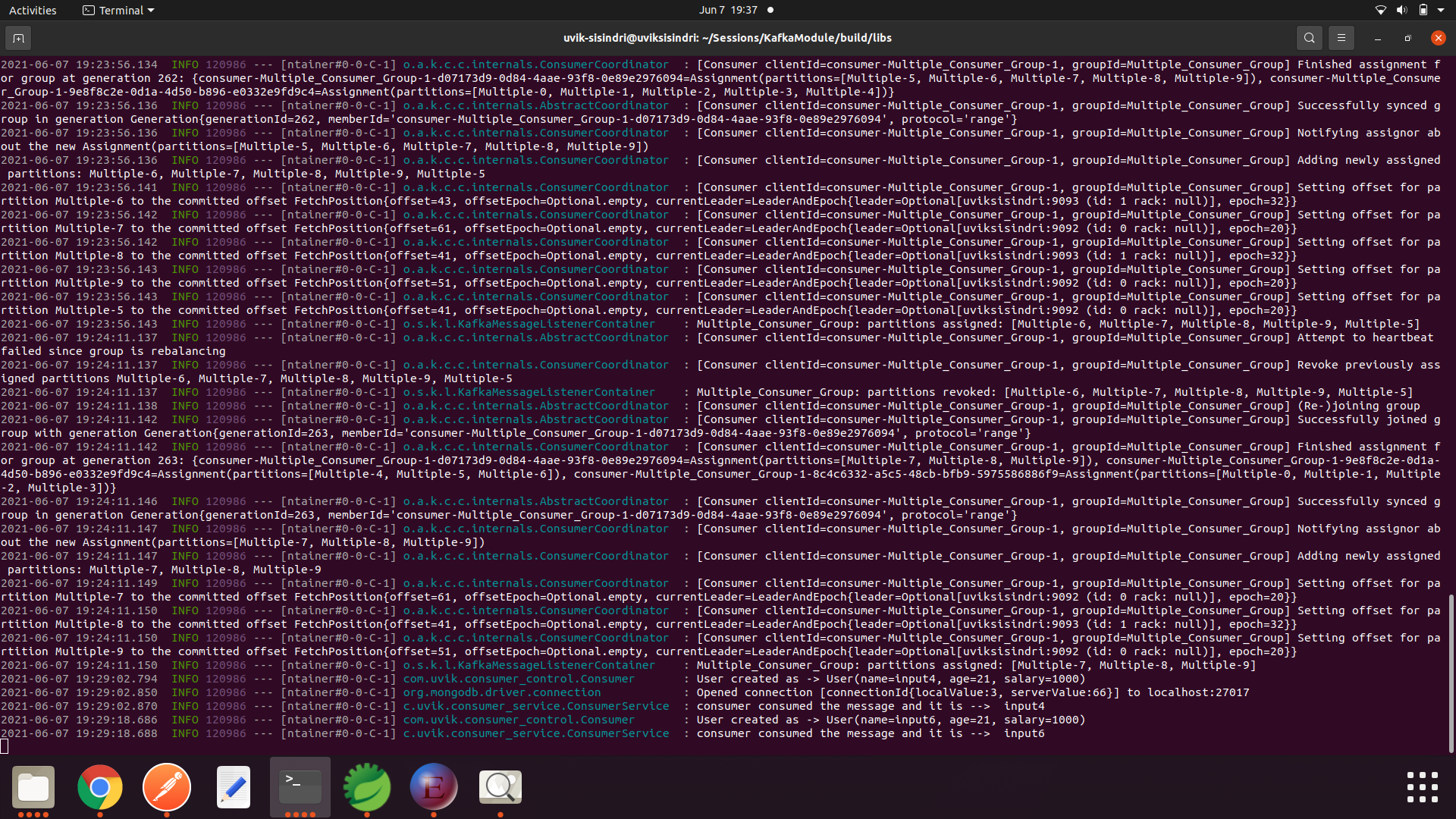
* I have used two ways to achieve it one is by using custom code and second is using ha proxy as load balancer.
* After main application we are having a load balancer where for every 5 seconds. The load balancer will be verifying whether the serves (**Producers**) are ready to accept the load from the main application.
* To achieve multiple producers and multiple consumer scenario. I have used to launch the producer application at multiple instances with the different port number and at the same I have launched the consumer application at different ports.
* So here I have used the haproxy config as a load balancer between the producers and mail application.
* Here the ha proxy is located in main application and the main application is running at localhost:7000. But the haproxy is running at 4002. When we need to achieve the above scenario, we are going to hit the haproxy server port every time and the proxy select the random ports where the producer instance is running. Since we are binding the haproxy port with the producer’s ports. From here we achieve the above scenario of multiple producers and multiple consumers.
* The Multiple producers and consumers scenario look like as follows.
  + From postman we are passing input as a json through the main application.
  + in the application it will select the random producer who is currently active and send the data to the topic through the producer.



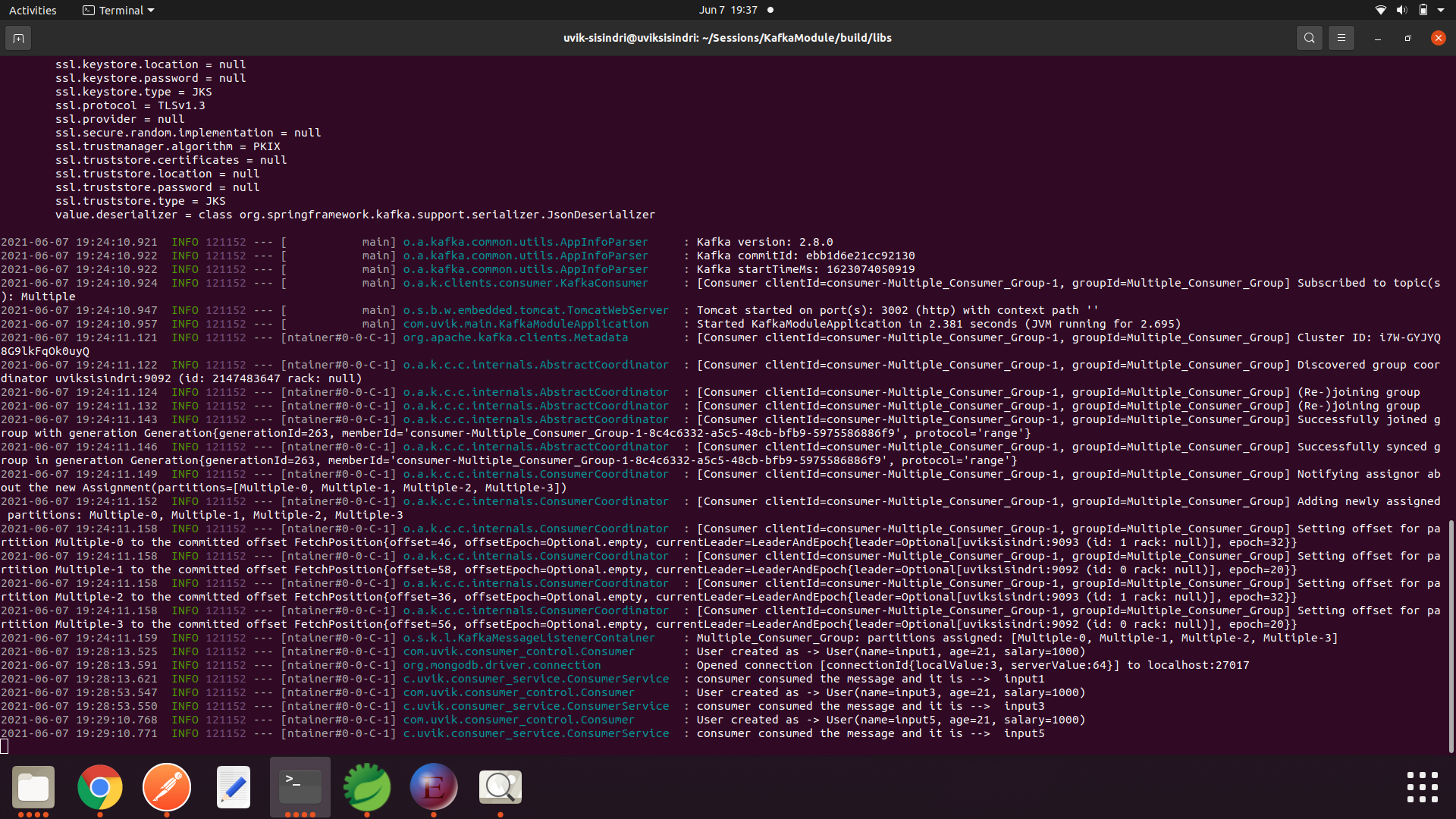


* The above three pictures show the three different producers running at different instances at **localhost:8081, localhost:8082, localhost:8083**
* The port selection is done by the main application based on the logic.
* The below pictures are consumers running at different instance but listening to same topic.



 Running at localhost:3001

Running at localhost:3002



Running at localhost:3000

* Here the consumer selection is based on the Round-Robbin algorithm with the zookeeper.
* And after consuming message from the partitions, it will inform the cluster about the offset value.
* And from there the cluster metadata is stored in the zookeeper for the further use.
* In other case we are using a load balancer instead of manual code.
* For this we are using the haproxy as a load balancer.
* Here we are hitting the port where the haproxy is running currently and it looks like as follows.
* **haproxy-http.cfg**

|  |
| --- |
| Global    maxconn 4096    log /dev/log local0  defaults    log   global    mode   http    # logs which servers requests go to, plus current connections and a whole lot of other stuff    option   httplog    option   dontlognull    retries   3    option redispatch    maxconn   2000    timeout connect 5000    timeout client 5000    timeout server 5000    # check webservers for health, taking them out of the queue as necessary    option httpchk HEAD /health    http-check expect status 200  listen www  timeout client 180s     timeout server 180s    balance roundrobin  **bind :4002**    server www1 localhost:**8001** check    server www2 localhost:**8002** check    server www3 localhost:**8003** check |

* Here we are binded the 4002.which signifies when we hit the 4002 along the api it will call one among the localhost:8001,8002,8003.
* In previous scenario we had three different applications.
  + Main Application
  + Producer Application (Multiple Instance)
  + Consumer Application (Multiple Instance)
* If we are using the haproxy load balancer we will be having only two applications one is Main Application where we are including and other is consumer application.
* The program looks like as follows
* **UserController.java**

|  |
| --- |
| @RestController  @Component  public class UserController  {  Logger logger=LoggerFactory.getLogger(UserController.class);  @Autowired  private ProxyService proxyService;    @PostMapping("/insertUser")  public String sendMessage(@RequestBody User user)  {  if(user.name==null)  {  return "userName should not be empty";  }  else if(user.age<18)  {  return "age should be greater than <18";  }  else if(user.salary<=10000)  {  return "salary should be grater then 1000";  }  else  {  logger.info("Hello api called at server with port: ");  return proxyService.insertData(user);  }  }  } |

* This looks like the same controller class in the above application.
* **UserRepository.java**

|  |
| --- |
| @Repository  public interface ProxyRepository extends MongoRepository<User, String>{} |

* **UserService.java**

|  |  |  |
| --- | --- | --- |
| @Service  public class ProxyService {  Logger logger = LoggerFactory.getLogger(ProxyService.class);  @Value( "${server.port}" )  private String serverPort;  @Value("${spring.kafka.topic}")  private String TOPIC;  @Autowired  private KafkaTemplate<String,User> userKafkaTemplate;  @Autowired  private ProxyRepository proxyRepository;  public String hello() throws InterruptedException {  logger.info("Hello api called at server with port: " + serverPort);  Thread.sleep(50);  return "Hello World from " + serverPort + "!";  }  public String insertData(User user)  {  logger.info(user.name +" "+user.age+" "+user.salary);    if(proxyRepository.save(user)!=null)  {  return sendToProducer(user);  }  else  {  return "The data is not stored into transaction database";  }  }  public String sendToProducer(User user)  {  User student=new User();  student.name=user.name;  student.age=user.age;  student.salary=user.salary;  ListenableFuture<SendResult<String, User>> future = userKafkaTemplate.send(TOPIC, student);  future.addCallback(new ListenableFutureCallback<SendResult<String, User>>()  { | | |
| @Override  public void onSuccess(SendResult<String, User> result)  {  logger.info("Sent message: " + student  + " with offset: " + result.getRecordMetadata().offset());  }  @Override  public void onFailure(Throwable exception)  {  logger.error("Unable to send message : " + student,exception);  }  });  return "Json Object Sucessfully published";  }  } |

* Here every thing depends on the haproxy config file and the binded port when we are hitting the binded port we will be sending data to the various instances running at different ports with the help of Round Robbin algorithm.
* Here we need to run the same application at different instances at.
  + Localhost:8001
  + Localhost:8002
  + Localhost:8003
* And run the cfg file using the following script.

#!/bin/bash

nohup haproxy -f haproxy-http.cfg > haproxy.log &