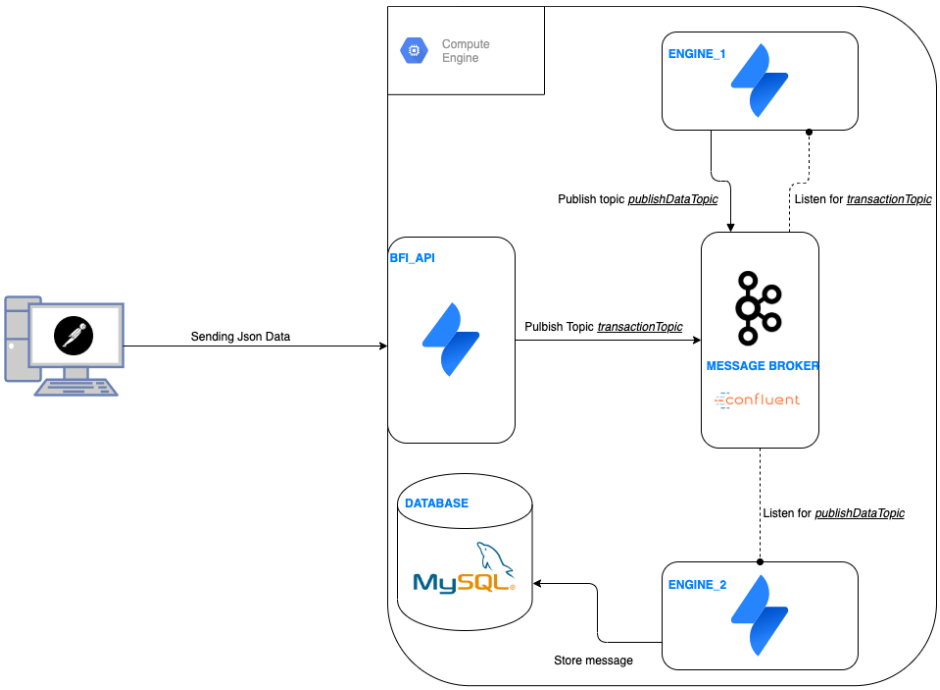
BFI Challenge

Data Message Flow



Penjelasan Infrastructure

Pada infrastructure diatas proses yang dilakukan adalah, user melakukan pengiriman data dengan menggunakan Postman dengan tipe data **Json** ke **BFI_API,** lalu pada **BFI_API** tersebut akan langsung mengirimkan Topic kepada broker dengan nama ***transactionTopic***, topic tersebut lalu akan didengar oleh **Engine_1** yang nantinya akan mengirimkan topik baru kepada broker dengan nama ***publishDataTopic***, lalu **Engine_2** akan mendengar topic baru tersebut dan langsung menyimpan data yang ada pada topic tersebut ke dalam **Database**.

Requirement Exam

Contoh data yg di kirim dari apps/web:
transaction_time
ID_product
Body_message

Engine 1 (Data message listener, publish message to message broker)
-Class
-Method
-Listener

Engine 2 (Subscribe to engine 1 messages -> save ke DB SQL/No-SQL)
-Class
-Method
-Listener

BFI_API

```
@RestController
public class APIController {

public static Logger logger = LoggerFactory.getLogger(MainApplication.class);

    @Autowired
    private KafkaTemplate<String, String> template;

    @PostMapping("/transaction")

    void publishTopic(@RequestBody String newTransaction){

         System.out.println(newTransaction);

         this.template.send("transactionTopic", newTransaction);

         logger.info("All Received");
    }
}
```

Untuk Class diatas digunakan untuk mengirim data berformat JSON melalui API method *publishTopic*:

```
void publishTopic(@RequestBody String newTransaction){
     …
    }
```

lalu dilanjutkan dengan membuat topic dan langsung di publish ke broker dengan *this.template.send()*:

```
this.template.send("transactionTopic", newTransaction);
```

Agar fungsi dari kafka tersebut dapat digunakan, diperlukan injeksi untuk class yang digunakan dengan:

```
@Autowired
    private KafkaTemplate<String, String> template;
```

Engine_1

```
@SpringBootApplication
public class MainApplication {

public static Logger logger = LoggerFactory.getLogger(MainApplication.class);

     public static void main(String... args){

        SpringApplication.run(MainApplication.class, args);

    }

     //Inject KafkaTempate Class
     @Autowired
     private KafkaTemplate<String, String> template;

     //Listen to topic from Sender
     @KafkaListener(topics = "transactionTopic")
     public void listen(ConsumerRecord<?, ?> cr) throws Exception{

        logger.info(cr.toString());

        //publish topic with only a Value
        this.template.send("publishDataTopic", cr.value().toString());
    }
}
```

Untuk membuat listener untuk topic apa yang ingin di dengar dengan menggunakan *@KafkaListener(topics = "_yourTopic_")* dan disini menggunakan *ConsumerRecord* untuk melihat info yang ada pada topic, lalu lanjut mempublish topic baru dengan nama *publishDataTopic* untuk didengarkan oleh *Engine_2*:

```
@KafkaListener(topics = "transactionTopic")
public void listen(ConsumerRecord<?, ?> cr) throws Exception{

        logger.info(cr.toString());

        //publish topic with only a Value
        this.template.send("publishDataTopic", cr.value().toString());
    }
```

Engine_2

TransactionModel

```
//Set this Class into Entity
@Entity
public class TransactionModel {

    //Auto Generate Id
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int Id;

    //Date Format for Json
    @JsonFormat(pattern="dd/MM/yyyy")
    public Date transactionTime;

    public int productId;

    public String bodyMessage;


    //Get Set Method//

    //Constructor
    TransactionModel(Date transactionTime, int productId, String bodyMessage
){

        this.transactionTime = transactionTime;

        this.productId = productId;

        this.bodyMessage = bodyMessage;
    }


    public TransactionModel(){

        super();

    }

}
```

Class *TransactionModel* dibuat untuk nantinya menjadi sebuah database atau entity dengan anotasi @Entity:

```
@Entity
```

```
public class TransactionModel {
     …
}
```

TransactionRepository

```
@Repository
public interface TransactionRepository extends
CrudRepository<TransactionModel, Integer> {

}
```

Class Repository inherit dari class *CrudRepository* yang digunakan untuk storing data ke database dengan type entity yang digunakan adalah *TransactionModel* dengan Id *IntegerI*

MainApplication

```
@SpringBootApplication
public class MainApplication {

 public static Logger logger = LoggerFactory.getLogger(MainApplication.class);

    //Inject TransactionRepository
    @Autowired
    private TransactionRepository repository;

    public static void main(String... args) {
        SpringApplication.run(MainApplication.class, args);
    }

    @KafkaListener(topics = "publishDataTopic")
    public void listen(String cr) throws Exception{

        //info Topic Value
        logger.info(cr.toString());

        //deserialize JSON to Object(TransactionModel)
        TransactionModel transaction = new Gson().fromJson(cr,
TransactionModel.class);

        //Store Object to Datasabase (MySql)
        repository.save(transaction);

    }

}
```

Pada class tersebut memiliki fungsi *Listener* untuk mendengarkan *publishDataTopic*:

```
@KafkaListener(topics = "publishDataTopic"){
      …
}
```

Karena yang data diterima dari API -> Kafka -> Engine_2 berupa data Json maka perlu di diserialize dengan menggunakan *Gson*:

```
TransactionModel transaction = new Gson().fromJson(cr,
TransactionModel.class);
```

Lalu agar data dapat di store di database, maka digunakan fungsi repository *save()* yang telah di inject pada class tersebut:

```
repository.save(transaction);
```