engage

Hello are you listening

There is stream for everything

Frank van der Linden - elstar IT

elstar IT

engage

Introduction

- Freelance Full stack developer
- Owner of elstar IT
- Curious of new technology
- Spring Framework enthusiast
- IBM Champion 2020
- Volleybal referee

IBM CHAMPION

elstar IT

engage

Disclaimer

This session about
'thinking-out-side-of-the-yellow-bubble'

engage

‘Everything in life can be broken down to producers, consumers and producers’

Mark Heckler

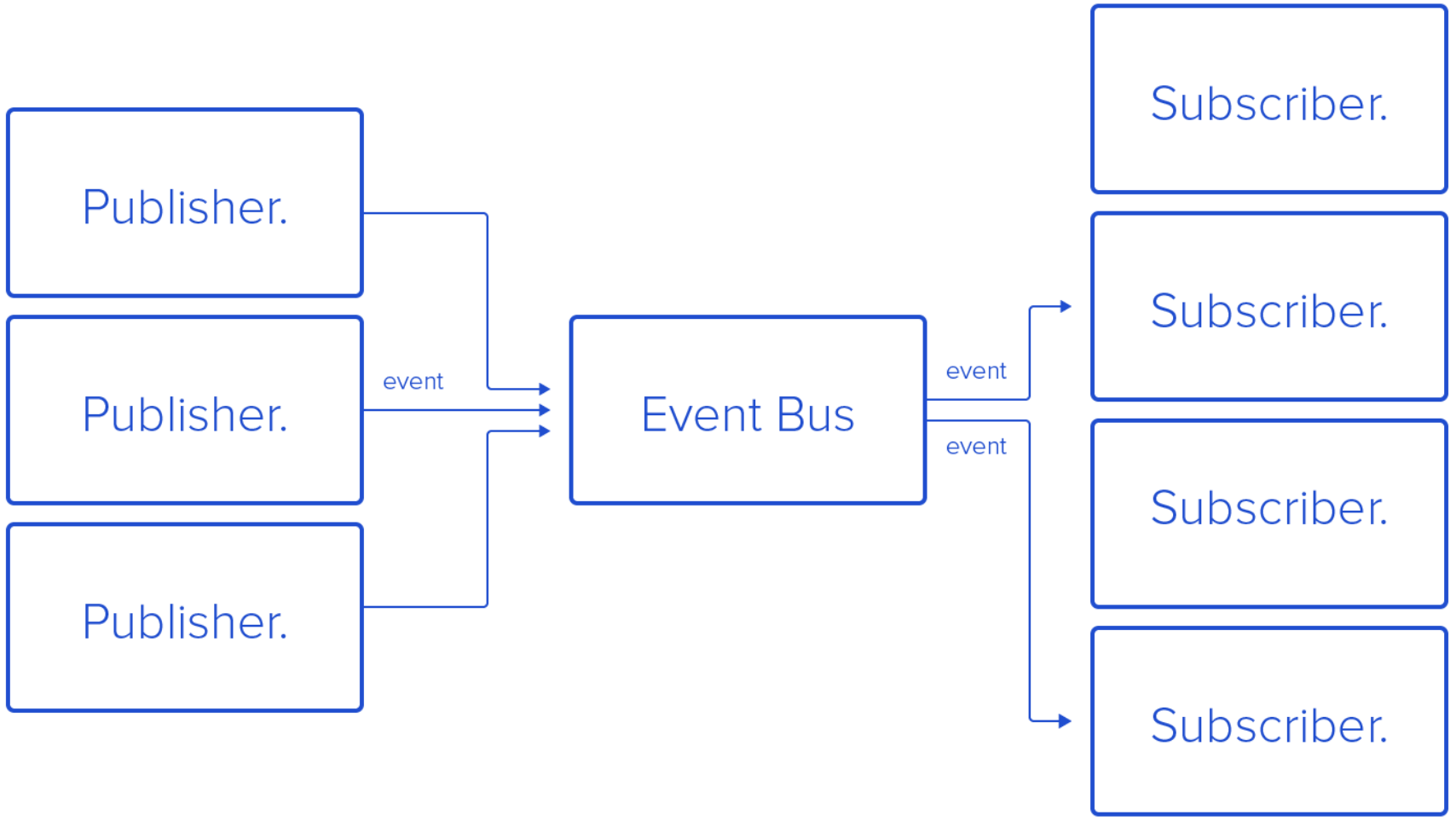
engage

How to proceed

- Pub Sub principle
- About streams
- Spring Cloud Streams
- Demo

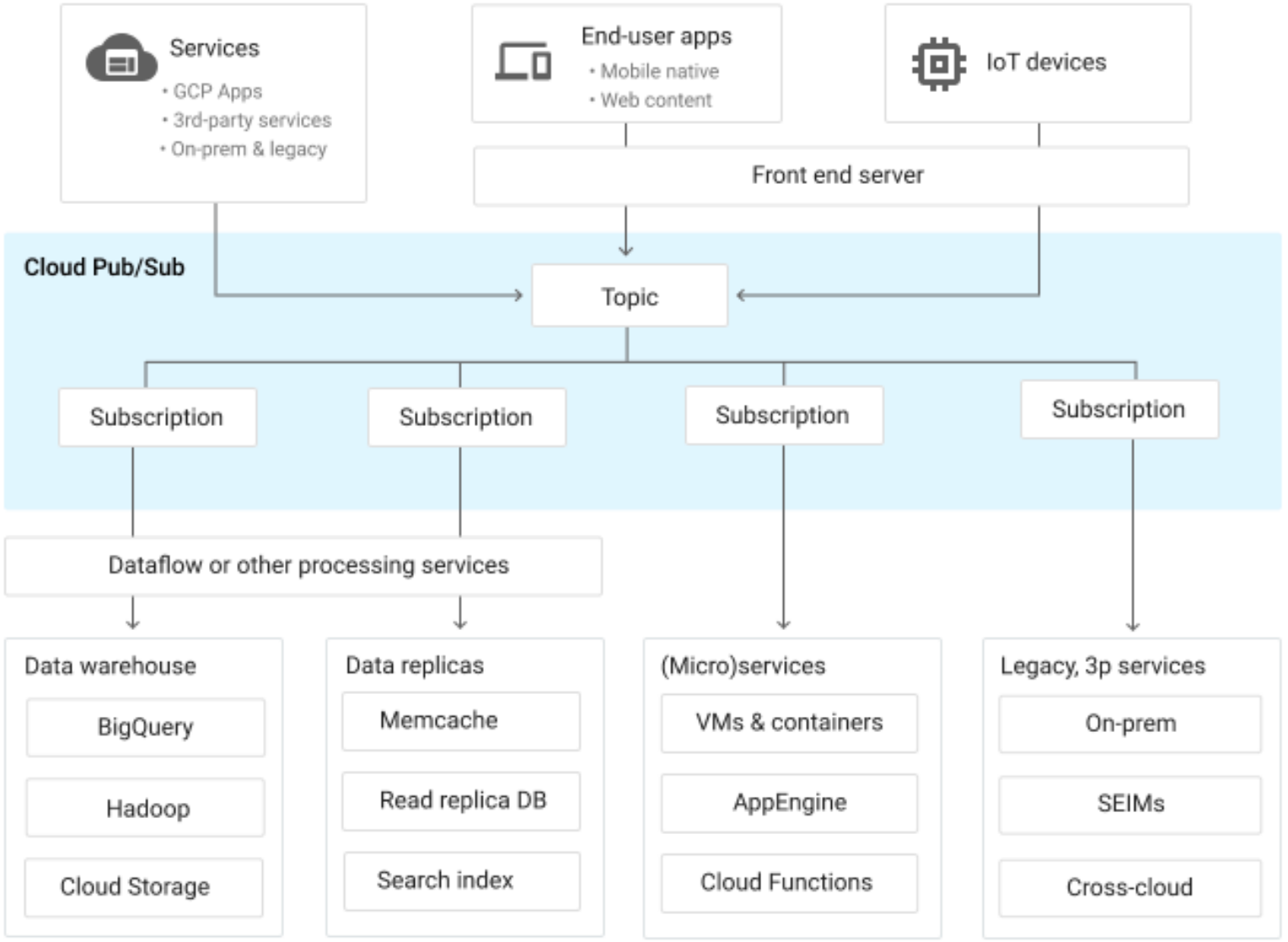
Publish/Subscribe principle

Publish/Subscribe principle



Publish/Subscribe principle

- Loose coupling
- Scalability
- Non-blocking
- Message driven
- 4 interfaces (Producer, Subscriber, Subscription and Processor)


Services
• GCP Apps
• 3rd-party services
• On-prem & legacy
End-user apps
• Mobile native
• Web content
IoT devices
Front end server
Cloud Pub/Sub
Topic
Subscription
Subscription
Subscription
Subscription
Dataflow or other processing services
Data warehouse
BigQuery
Hadoop
Cloud Storage
Data replicas
Memcache
Read replica DB
Search index
(Micro)services
VMs & containers
AppEngine
Cloud Functions
Legacy, 3p services
On-prem
SEIMs
Cross-cloud

About streams

Parts of the stream

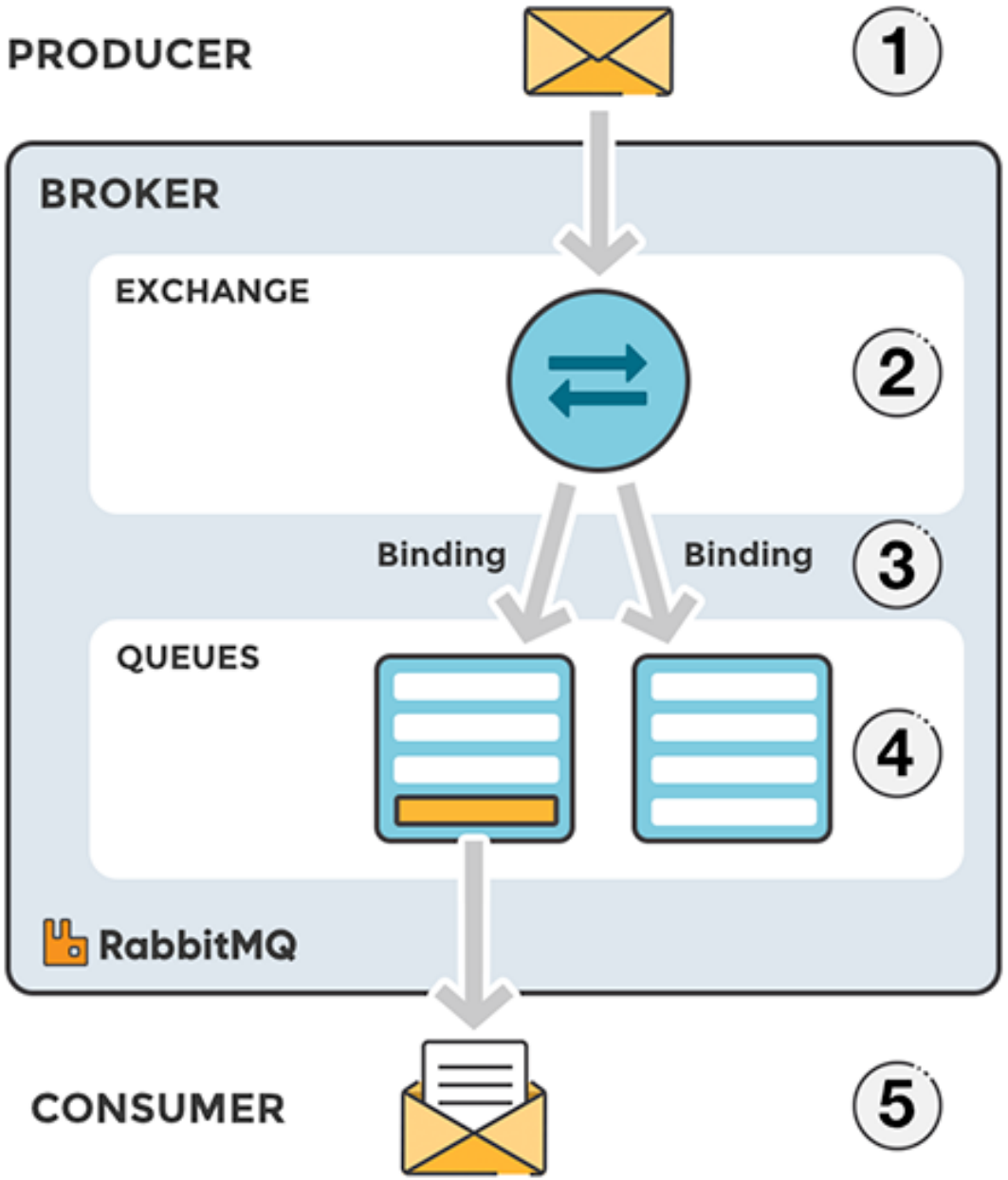
- **Producer** or **Source** or **Publisher**, publish the message on the topic or queue
- **Consumer** or **Sink** or **Receiver**, receiving the message from the topic or queue
- **Processor**, can consume message, manipulate it and produce it again.
- **Broker**, the middleware who is handling the queues or topics.

Message Broker

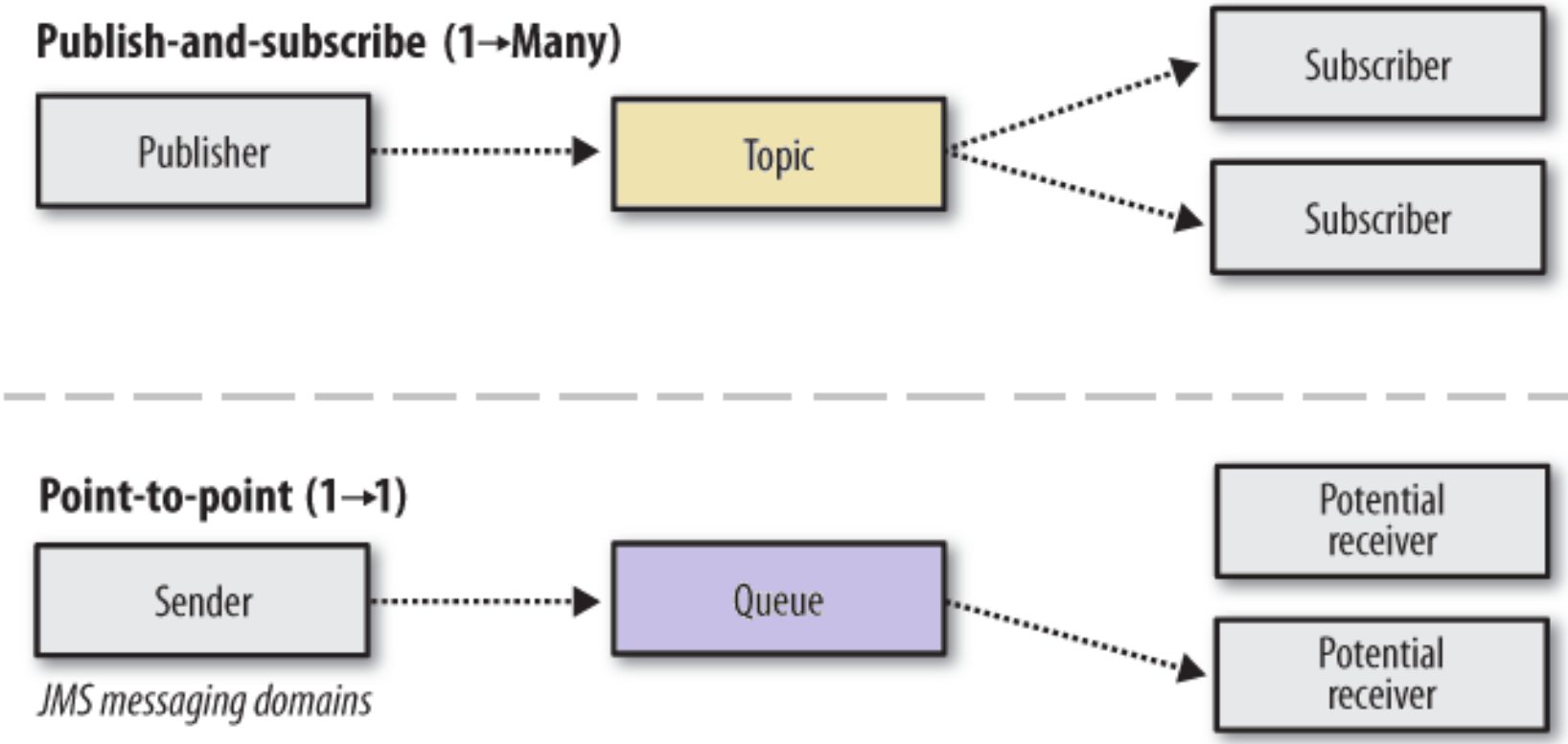
- Route messages to one or more destinations
- Transform messages to an alternative representation
- Provide content and topic-based message routing using the publish–subscribe pattern
- Perform message aggregation, decomposing messages into multiple messages and sending them to their destination, then recomposing the responses into one message to return to the use
- Respond to events or errors

Message Broker

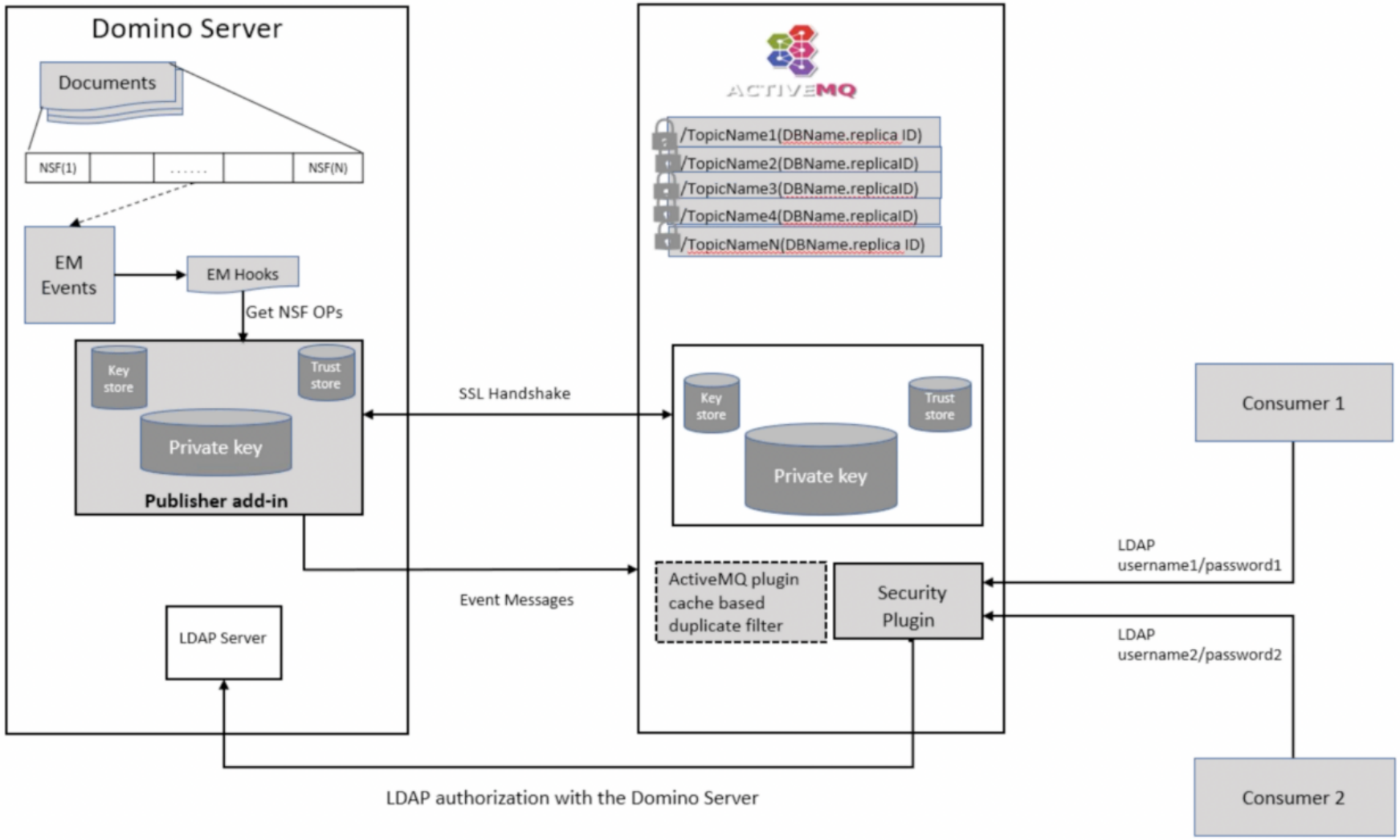
- Kafka
- RabbitMQ
- OpenMQ
- ActiveMQ (Amazone MQ)
- Google Cloud PubSub
- Azure Service Bus
- Amazone Kinesis
- IBM MQ
- ……..


PRODUCER
1
BROKER
EXCHANGE
2
Binding
Binding
3
QUEUES
4
RabbitMQ
CONSUMER
5

Queue versus Topic



Domino Event Publisher



LDAP authorization with the Domino Server

engage

Domino Event Publisher

- Add-on on Domino
- Security Plugin for MQ's needed to be installed
- Will share UniqueID of Document and event type
- Consumer needs to connect back to Domino to get all the information
- May be later also other information to produce.
- Private Beta will be available soon to HCL Masters.

Domino Event Publisher - my opinion

- A good first step to embrace the decoupled world
- Too much moving parts
- Too complicated
- Not following the pub/sub principle, because of the needed call back
- Hope it will be possible in the future to share just information with the broker.

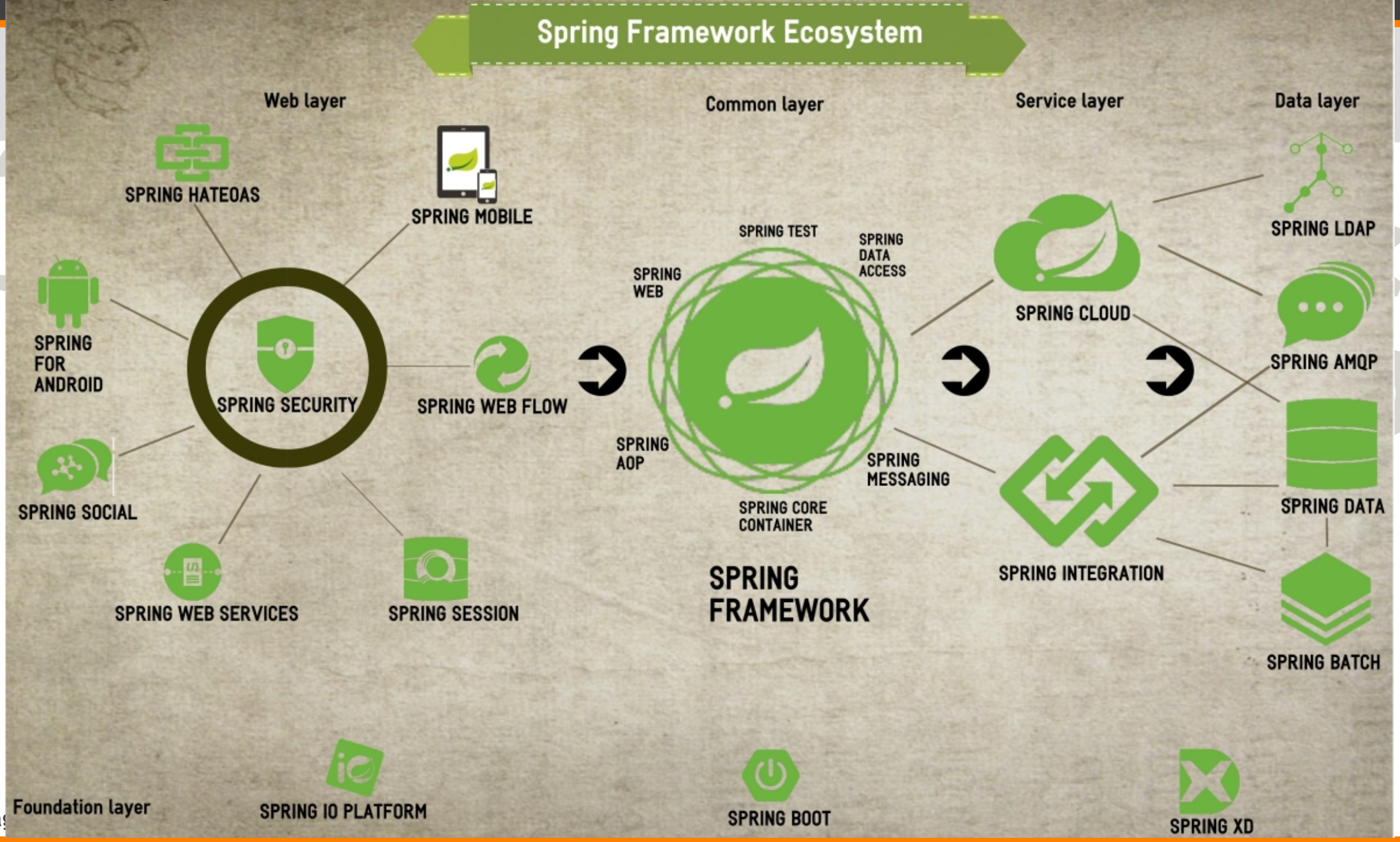
Spring Cloud Stream

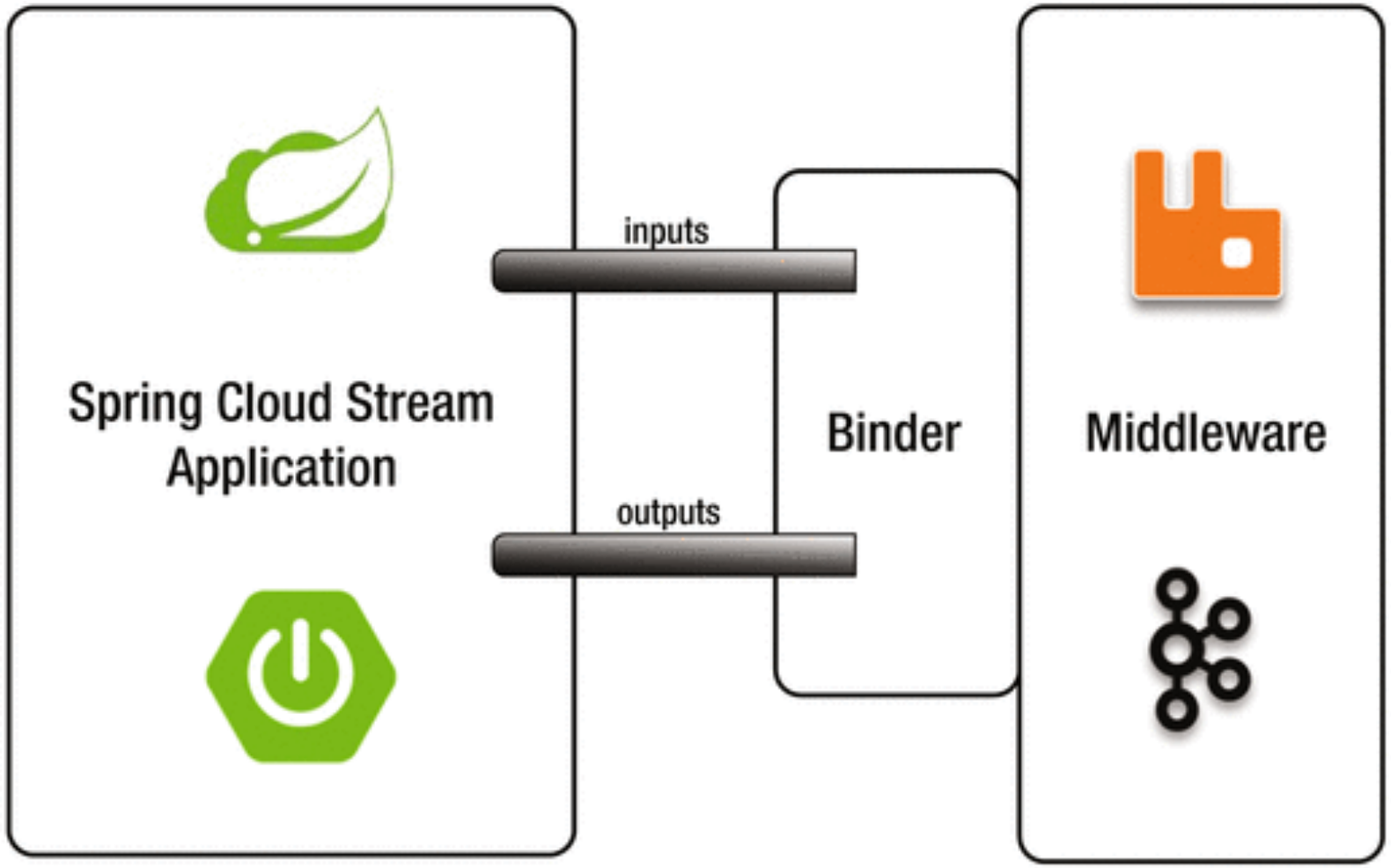
engage

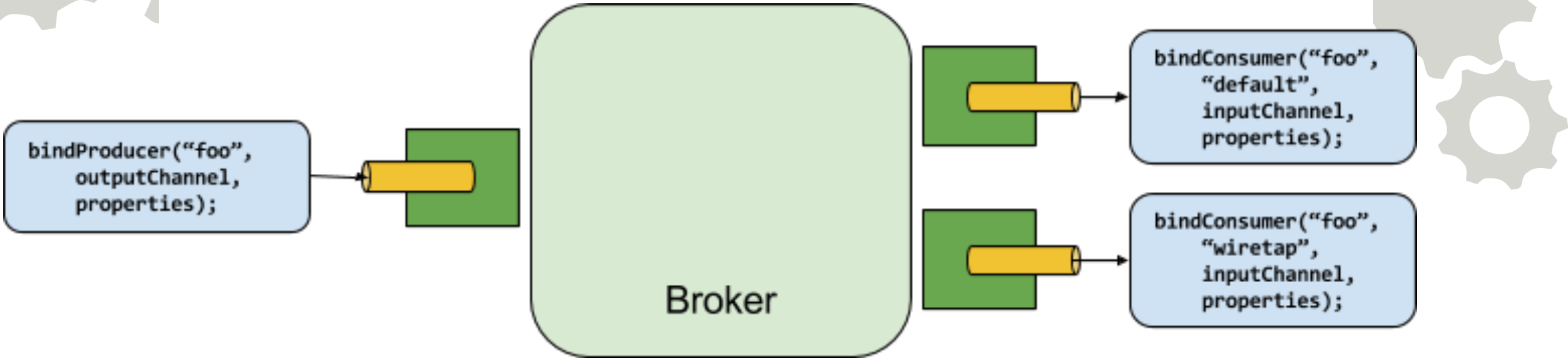
Spring Cloud Stream

- Part of the Spring Cloud Family
- Abstraction of the connection with a broker
- Works in existing Spring Boot apps
- Binders are the foundation
- Multiple brokers are supported

- RabbitMQ
- Apache Kafka
- Kafka Streams
- Amazon Kinesis
- Google PubSub *(partner maintained)*
- Solace PubSub+ *(partner maintained)*
- Azure Event Hubs *(partner maintained)*
- Apache RocketMQ *(partner maintained)*


Spring Framework Ecosystem
Web layer
Common layer
Service layer
Data layer
SPRING HATEOAS
SPRING MOBILE
SPRING LDAP
SPRING TEST
SPRING DATA ACCESS
SPRING WEB
SPRING CLOUD
SPRING FOR ANDROID
SPRING SECURITY
SPRING WEB FLOW
SPRING AMQP
SPRING AOP
SPRING MESSAGING
SPRING DATA
SPRING SOCIAL
SPRING CORE CONTAINER
SPRING INTEGRATION
SPRING WEB SERVICES
SPRING SESSION
SPRING FRAMEWORK
SPRING BATCH
Foundation layer
SPRING IO PLATFORM
SPRING BOOT
SPRING XD


Spring Cloud Stream
Application
inputs
outputs
Binder
Middleware


bindProducer("foo",
outputChannel,
properties);
Broker
bindConsumer("foo",
"default",
inputChannel,
properties);
bindConsumer("foo",
"wiretap",
inputChannel,
properties);

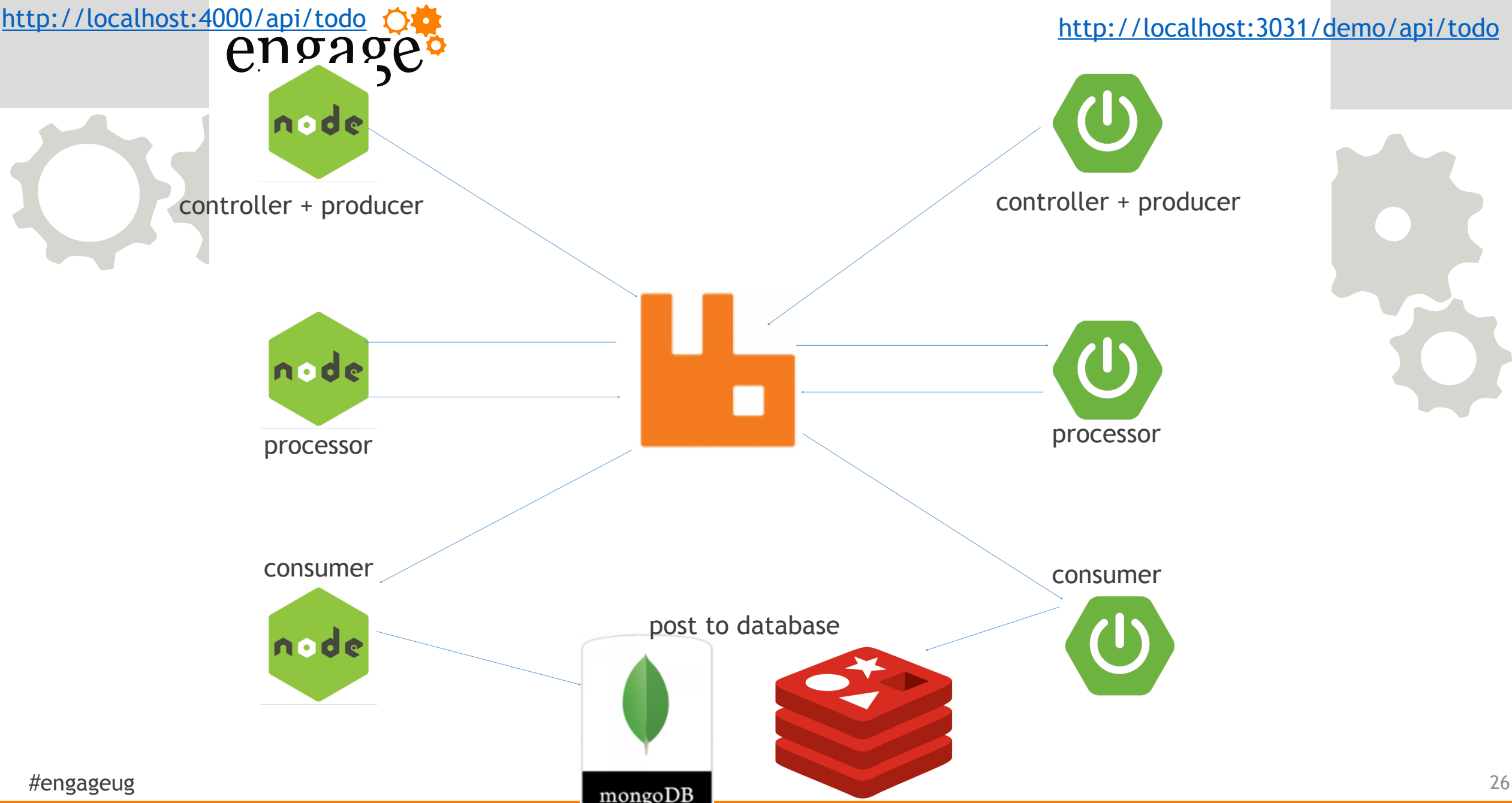
Demo application

Demo

- Spring Boot - foundation for Java API
- Spring data - for connection to datastore
- Spring Cloud Stream - abstraction to connect to broker
- RabbitMQ
- NodeJS + Typescript
- MongoDb
- Redis

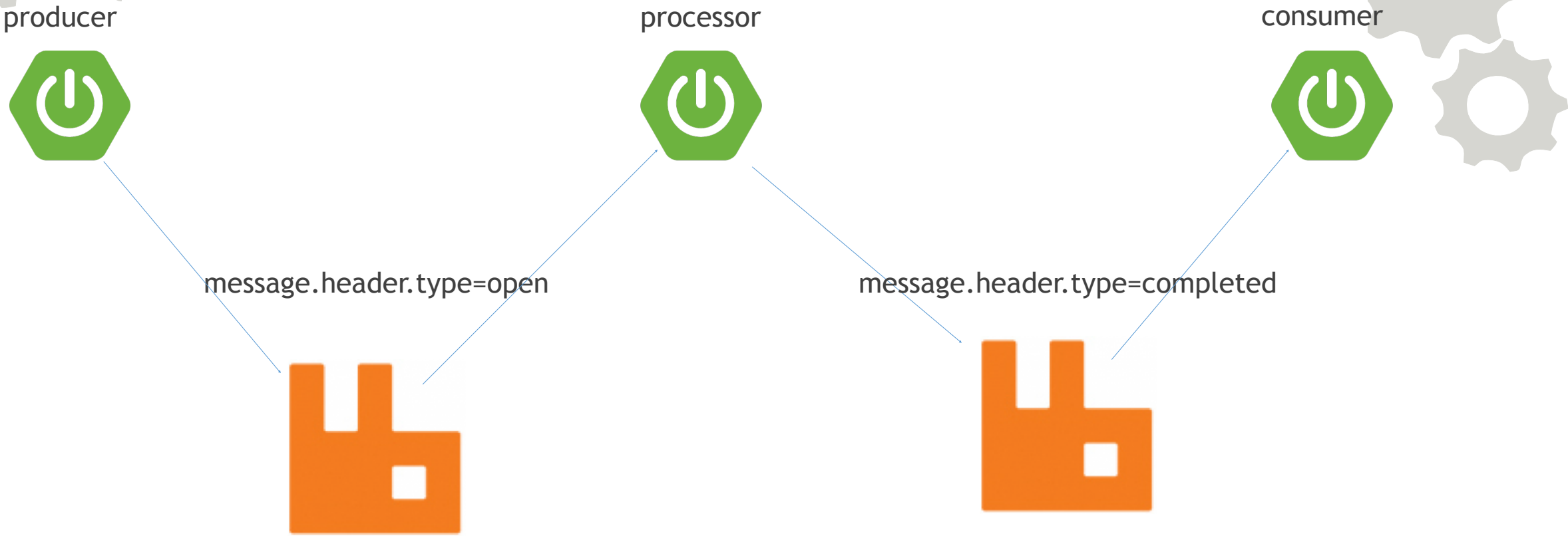
http://localhost:4000/api/todo

http://localhost:3031/demo/api/todo



engage

Demo Flow



engage

Spring Cloud Stream - things to remember

pom.xml

```
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-stream</artifactId>
</dependency>

<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-starter-stream-rabbit</artifactId>
</dependency>
```

application.yaml

```
spring:
  rabbitmq:
    addresses: amqp://localhost #amqp
  cloud:
    stream:
      rabbit:
        binder:
          headers: type
      bindings:
        consumerChannel:
          binder: rabbit
          destination: demo-queue
        producerChannel:
          binder: rabbit
          destination: demo-queue
```

engage

Spring Cloud Stream - things to remember

bindings

```java
public interface ProducerBinding {
  String OUTPUT = "producerChannel";
  @Output(OUTPUT)
  MessageChannel producerChannel();
}
```

```java
public interface ConsumerBinding {
  String INPUT = "consumerChannel";
  @Input(INPUT)
  SubscribableChannel consumerChannel();
}
```

producer

```java
@Slf4j
@EnableBinding({ProducerBinding.class})
public class DemoProducer {

  @Autowired
  private ProducerBinding producer;

  public void postTodoOnQueue(Todo todo){

    if(todo.getCreated()==null){
      todo.setCreated(LocalDateTime.now());
    }

    Message<Todo> msgTodo = MessageBuilder
      .withPayload(todo)
      .setHeader( headerName: "type",  headerValue: "open")
      .build();

    log.info("[DemoProducer]: Post Open Message on Queue - {}", msgTodo);
    producer.producerChannel().send(msgTodo);
  }
}
```

processor

```java
@Slf4j
@EnableBinding({ProducerBinding.class, ConsumerBinding.class})
public class DemoProcessor {

  @Autowired
  private ProducerBinding producer;

  @StreamListener(target = ConsumerBinding.INPUT, condition = "headers['type']=='open'")
  public void processOpen(Todo todo) {
    log.info("[DemoProcessor]: Received Open Todo from queue: {}", todo);
    todo.setCompleted(true);

    Message<Todo> msgTodo = MessageBuilder
      .withPayload(todo)
      .setHeader( headerName: "type",  headerValue: "completed")
      .build();

    log.info("[DemoProcessor]: Post Completed Message on Queue - {}", msgTodo);
    producer.producerChannel().send(msgTodo);
  }
}
```

consumer

```java
@Slf4j
@EnableBinding(ConsumerBinding.class)
public class DemoConsumer {

  @Autowired
  private ProducerBinding producer;

  @Autowired
  private MongoTodoRepository repository;

  @Autowired
  private RedisTodoRepository redisRepository;

  @StreamListener(target = ConsumerBinding.INPUT, condition = "headers['type']=='completed'")
  public void processCompleted(Todo todo) {
    log.info("[DemoConsumer]: Received Completed Todo from queue - {}", todo);
    repository.save(todo);
    redisRepository.save(todo);
  }
}
```

engage

NodeJS - things to remember

server.ts

```
import * as express from 'express';
import * as dotenv from 'dotenv';
import * as bodyParser from 'body-parser';
import {ApiRoutes} from "./routes/api-routes";
import {TodoConsumer} from "./consumer/todo.consumer";
import {TodoProcessor} from "./processor/todo.processor";

dotenv.config();

const app = express();

app.set('port', process.env.PORT || 4000);
app.use(bodyParser.json());
app.use(bodyParser.urlencoded( options: { extended: true }));

app.use('/api', ApiRoutes);

// @ts-ignore
let consumer = new TodoConsumer();
consumer.consume();

let processor = new TodoProcessor();
processor.process();

app.listen(app.get('port'), callback: () => {
    console.log(('App is running at http://localhost:%d in %s mode'),
        app.get('port'), app.get('env'));
    console.log('Press CTRL-C to stop\n');
});

module.exports = app;
```

router

```
router.post( path: '/todo', handlers: (req: Request, res: Response) => {
  todoController.create(req,res).then(function(result :Todo ) {
        res.send(result);
  });
});
```

controller

```
export class TodoController {
  private producer: TodoProducer;

  constructor() {
    this.init();
  }

  init() {
    this.producer = new TodoProducer();
  }

  async create(req: Request, res: Response): Promise<Todo> {
    let uuId = null;
    let todo = new Todo(uuId, req.body.subject, req.body.description);
    todo.setCreated(new Date());
    todo.setCompleted(false);
    this.producer.postTodoOnQueue(todo);
    return todo;
  }

}
```

engage

NodeJS - things to remember

producer

```
const opts = {
  headers : {
    "type" : "open",
  },
  contentType : CONTENT_TYPE_JSON
}

export class TodoProducer{

  constructor(){

  }

  postTodoOnQueue(todo : Todo){

    let connection = new Amqp.Connection(RABBITMQ_LOCAL_URL);
    let exchange = connection.declareExchange(EXCHANGE, type: "amq.topic", options: {durable: true, noCreate: true});
    connection.completeConfiguration().then( onFulfill: () => {
      let msgTodo = new Amqp.Message(todo, opts);
      console.log("Post to queue: " + JSON.stringify(msgTodo));
      exchange.send(msgTodo, ROUTING_KEY_TODO_OPEN);
    });
  }
}
```

processor

```
const opts = {
  headers : {
    "type" : "completed",
  },
  contentType : CONTENT_TYPE_JSON
}

export class TodoProcessor {

  constructor(){
    this.init();
  }

  init(){
    console.log("Processor open for business....");
  }

  process(){
      let connection = new Amqp.Connection(RABBITMQ_LOCAL_URL);
      let exchange = connection.declareExchange(EXCHANGE, type: "amq.topic", options: {durable: true, noCreate: true});
      connection.completeConfiguration().then( onFulfill: () => {
        let queueOpen = connection.declareQueue(QUEUE_OPEN);
        queueOpen.bind(exchange);
        queueOpen.activateConsumer( onMessage: (message : Message ) => {
          console.log("QUEUE OPEN: " + JSON.stringify(message.getContent()));
          if (message.properties.headers.type === 'open') {

            let todo: Todo = new Todo( id: null, message.getContent().subject, message.getContent().description);
            todo.setCreated(message.getContent().created);
            todo.setCompleted(true);

            let msgTodo = new Amqp.Message(todo, opts);
            exchange.send(msgTodo);
          }
        });
      });
  }
}
```

engage

NodeJS - things to remember

consumer

```
export class TodoConsumer {

  private mongoSservice : TodoMongoService;
  private redisService : TodoRedisService;

  constructor() {
    this.init();
  }

  init(){
    this.mongoSservice = new TodoMongoService();
    this.redisService = new TodoRedisService();

    console.log("Consumer open for business....");
  }

  consume() {
    let connection = new Amqp.Connection(RABBITMQ_LOCAL_URL);
    let exchange = connection.declareExchange(EXCHANGE, type: "amq.topic", options: {durable: true, noCreate: true});
    connection.completeConfiguration().then( onFulfill: () => {
      let queueCompleted = connection.declareQueue(QUEUE_COMPLETED);
      queueCompleted.bind(exchange);
      queueCompleted.activateConsumer( onMessage: (message :Message ) => {
        if(message.properties.headers.type == 'completed') {
          console.log("QUEUE COMPLETED: " + JSON.stringify(message.getContent()))
          let todo: Todo = new Todo( id: null, message.getContent().subject, message.getContent().description);
          todo.setCreated(message.getContent().created);
          todo.setCompleted(message.getContent().completed);
          this.mongoSservice.save(todo)
          this.redisService.save(todo);
          message.ack();
        }
      });
    });
  }
}
```

mongo service

```
const mongoUrl = 'mongodb://localhost:27017/';
const mongoUrlAtlas = 'mongodb://fa_events:ThHch9gn4RTXdZf@events-store-shard-0
const dbName = 'todo-store';
export class TodoMongoService{
  private db: any;
  private repository : TodoRepository;

  constructor() {
    this.init();
  }

  init(){
    MongoClient.connect(mongoUrl, { useNewUrlParser: true }, (err, client) => {
      if (err) return console.log(err)
      this.db = client.db(dbName) // whatever your database name is
      this.repository = new TodoRepository(this.db, collectionName: 'todo');
    })
  }

  save(todo : Todo){
    return this.repository.create(todo);
  }
}
```

redis service

```
export class TodoRedisService {

  private redisClient : Tedis;
  constructor() {
    this.init()
  }

  init(){
    this.redisClient = new Tedis( options: {
      port: 6379,
      host: "127.0.0.1",
      password: "redis",
    });
  }

  save(todo : Todo){
    let uuId = uuidv1();
    console.log(JSON.stringify(todo));
    this.redisClient.set(uuId, JSON.stringify(todo));
  }

}
```

engage

Questions

engage

Resources

- Spring Cloud Stream
  - https://spring.io/projects/spring-cloud-stream
- Demo application
  - https://github.com/flinden68/streams-demo
- RabbitMQ
  - https://www.rabbitmq.com/documentation.html
- MongoDb
  - https://www.mongodb.com/
- Redis
  - https://redis.io/topics/quickstart
- NodeJS
  - https://nodejs.org/en/
- MongoDb Compass Community
  - https://www.mongodb.com/products/compass
- Redily (GUI for Redis)
  - https://www.redily.app/

engage

Thank you

 @flinden68

 http://www.elstarit.nl

 flinden68@elstarit.nl

 https://nl.linkedin.com/in/flinden68

elstar IT