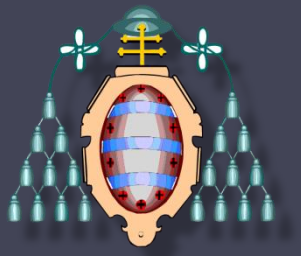
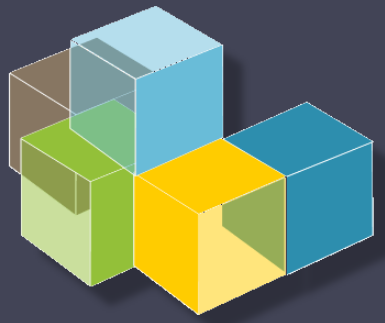
ES
Español



Universidad de Oviedo

Escuela de Ingeniería Informática



ARQUITECTURA DEL SOFTWARE

Arquitectura del Software

Lab.
Apache Kafka & Spring Kafka

2016-17

Herminio García González

Escuela de Ingeniería Informática de Oviedo

¿Qué es?

- Apache Kafka es una plataforma de *streaming* distribuida
  - Publicar y suscribirse a *streams* de eventos (similar a una cola de mensajes)
  - Almacenar los *streams* de una manera tolerante a fallos
  - Procesar los *streams* según vayan ocurriendo

Usos

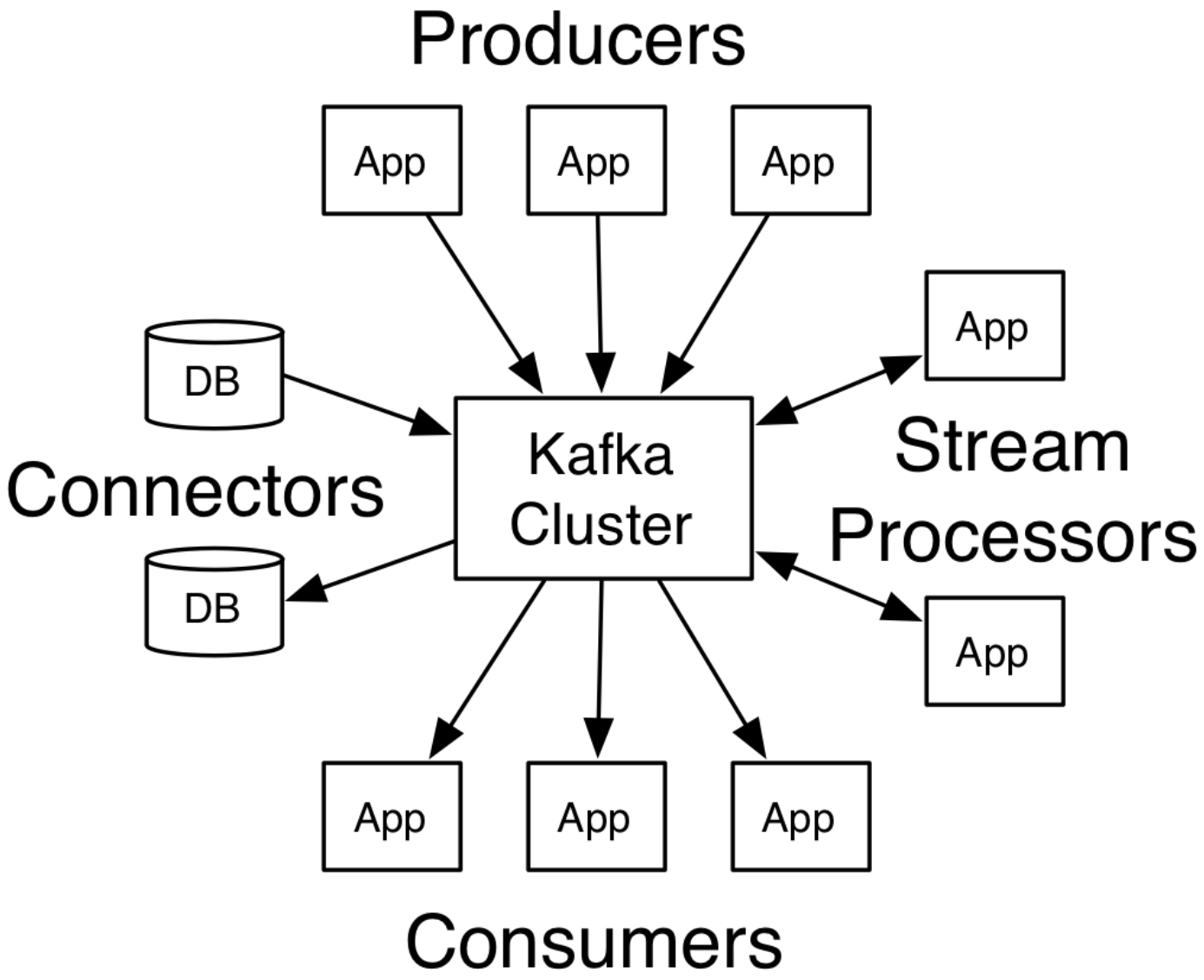
- *Streaming* de datos en tiempo real entre sistemas y aplicaciones
- Aplicaciones que transformen o reaccionen a *streaming* de datos en tiempo real

Conceptos básicos

- Arquitectura
  - *Publish/Subscribe*
- Sistema distribuido
  - Uno o más servidores
- Almacena los eventos en registros y categorías llamadas *topics*
- Cada registro consta de
  - Clave (*key*)
  - Valor (*value*)
  - Marca temporal (*timestamp*)

APIs

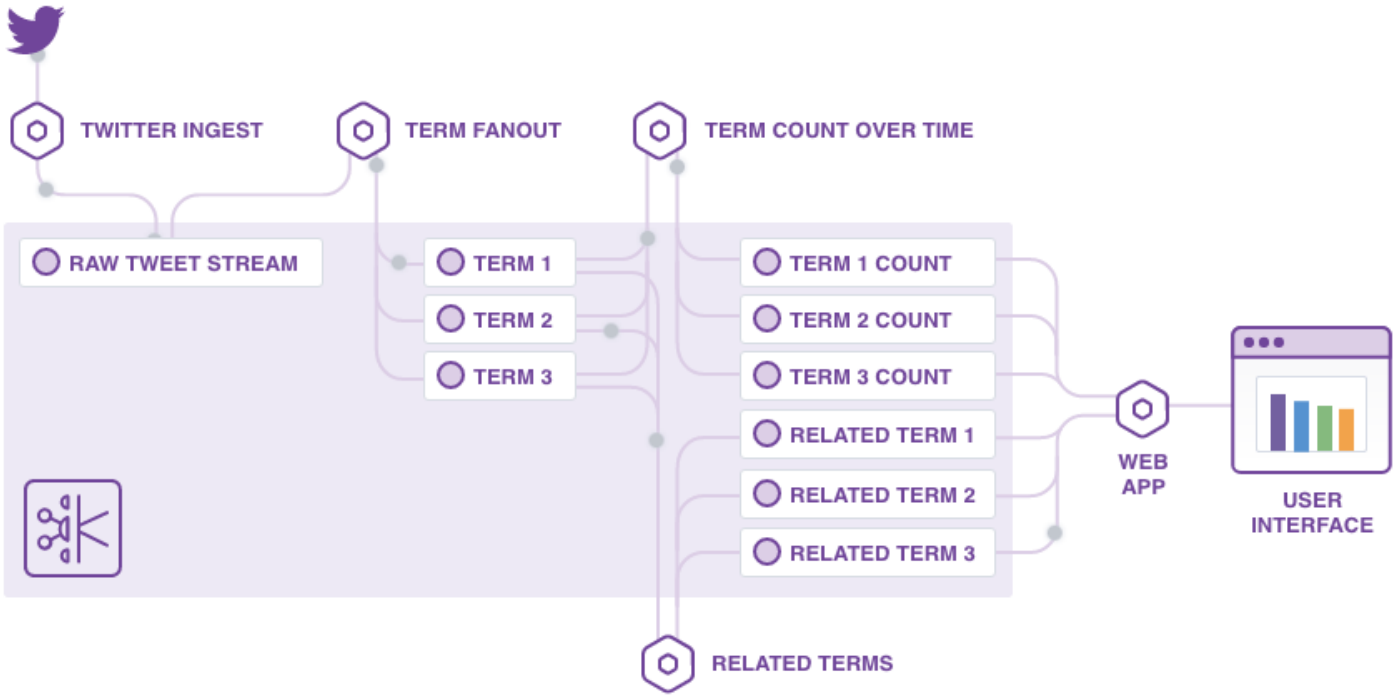
- Producer API
  - Permite publicar un *stream* en un *topic*
- Consumer API
  - Permite suscribirse a un *topic* para recibir *streams*
- Streams API
  - Permite transformar los *streams* de un *topic*. Recoge datos de un *topic* y los publica en otro ya transformados.
- Connector API
  - Permite hacer y ejecutar *producers* y *consumers* reutilizables para conectar Kafka con sistemas existentes.


Producers
App
App
App
DB
Connectors
DB
Kafka
Cluster
App
Stream
Processors
App
App
App
App
Consumers

Topics

- Son categorías donde se publican los registros
- Para cada *topic* Kafka mantiene un *log* particionado
- Cada *topic* es una categoría de eventos o transformaciones distinta

Ejemplo usando Twitter en Heroku



¿Y cómo integro yo esto en mi práctica?

Keep calm

&

Spring Kafka

Producer con Spring

```
@Configuration
@EnableKafka
public class KafkaProducerFactory {

    @Bean
    public ProducerFactory<String, String> producerFactory() {
        return new DefaultKafkaProducerFactory<>(producerConfigs());
    }

    @Bean
    public Map<String, Object> producerConfigs() {
        Map<String, Object> props = new HashMap<>();
        props.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG, "localhost:9092");
        props.put(ProducerConfig.RETRIES_CONFIG, 0);
        props.put(ProducerConfig.BATCH_SIZE_CONFIG, 16384);
        props.put(ProducerConfig.LINGER_MS_CONFIG, 1);
        props.put(ProducerConfig.BUFFER_MEMORY_CONFIG, 33554432);
        props.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG, StringSerializer.class);
        props.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG, StringSerializer.class);
        return props;
    }

    @Bean
    public KafkaTemplate<String, String> kafkaTemplate() {
        return new KafkaTemplate<String, String>(producerFactory());
    }


}
```

Producer con Spring (II)

```
@ManagedBean
public class KafkaProducer {

    private static final Logger logger = Logger.getLogger(KafkaProducer.class);

    @Autowired
    private KafkaTemplate<String, String> kafkaTemplate;

    public void send(String topic, String data) {
        ListenableFuture<SendResult<String, String>> future = kafkaTemplate.send(topic, data);
        future.addCallback(new ListenableFutureCallback<SendResult<String, String>>() {
            @Override
            public void onSuccess(SendResult<String, String> result) {
                logger.info("Success on sending message \"" + data + "\" to topic " + topic);
            }

            @Override
            public void onFailure(Throwable ex) {
                logger.error("Error on sending message \"" + data + "\", stacktrace " + ex.getMessage());
            }
        });
    }

}
```

Consumer con Spring

```
@Configuration
@EnableKafka
public class KafkaListenerFactory {

   @Bean
   KafkaListenerContainerFactory<ConcurrentMessageListenerContainer<Integer, String>> kafkaListenerContainerFactory() {
      ConcurrentKafkaListenerContainerFactory<Integer, String> factory =
            new ConcurrentKafkaListenerContainerFactory<>();
      factory.setConsumerFactory(consumerFactory());
      factory.setConcurrency(3);
      factory.getContainerProperties().setPollTimeout(3000);
      return factory;
   }

   @Bean
   public ConsumerFactory<Integer, String> consumerFactory() {
      return new DefaultKafkaConsumerFactory<>(consumerConfigs());
   }

   @Bean
   public Map<String, Object> consumerConfigs() {
      Map<String, Object> props = new HashMap<>();
      props.put(ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG, "localhost:9092");
      props.put(ConsumerConfig.ENABLE_AUTO_COMMIT_CONFIG, false);
      props.put(ConsumerConfig.AUTO_COMMIT_INTERVAL_MS_CONFIG, "100");
      props.put(ConsumerConfig.SESSION_TIMEOUT_MS_CONFIG, "15000");
      props.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG, StringDeserializer.class);
      props.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG, StringDeserializer.class);
      props.put(ConsumerConfig.GROUP_ID_CONFIG, "es.uniovi");
      return props;
   }

}
```

Consumer con Spring (II)

```
@ManagedBean
public class MessageListener {

    private static final Logger logger =
Logger.getLogger(MessageListener.class);

    @KafkaListener(topics = "exampleTopic")
    public void listen(String data) {
        logger.info("New message received: \"" + data + "\"");
    }



}
```

One more thing…

- ¿Y cómo se reflejan los cambios en tiempo real en la interfaz del usuario?
- ¿Alguna idea?
- Debéis buscar una solución arquitectónica para este problema

Bibliografía

- [1] https://kafka.apache.org/intro
- [2] https://heroku.github.io/kafka-demo/#architecture