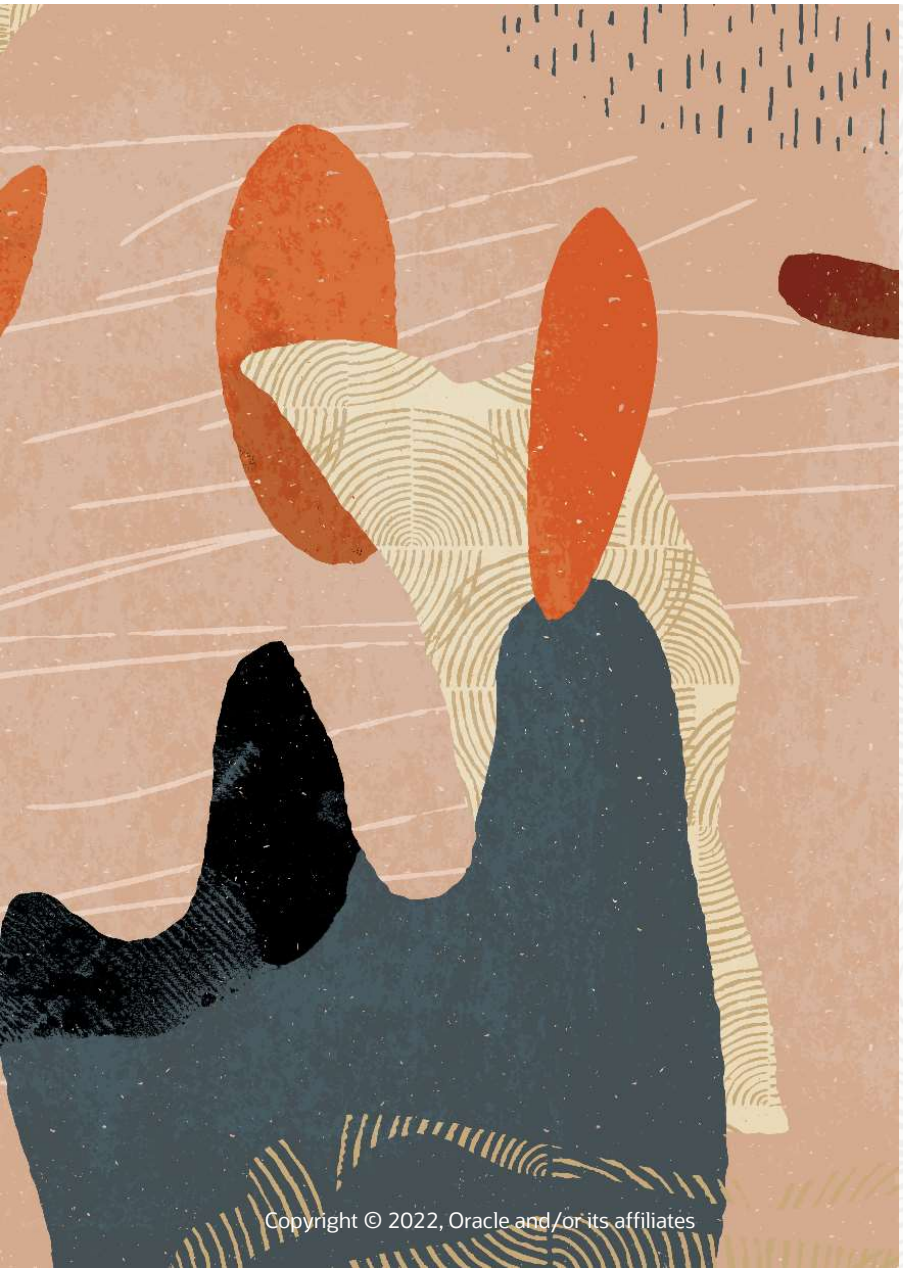
ORACLE

Oracle 高级队列介绍

张弘弢

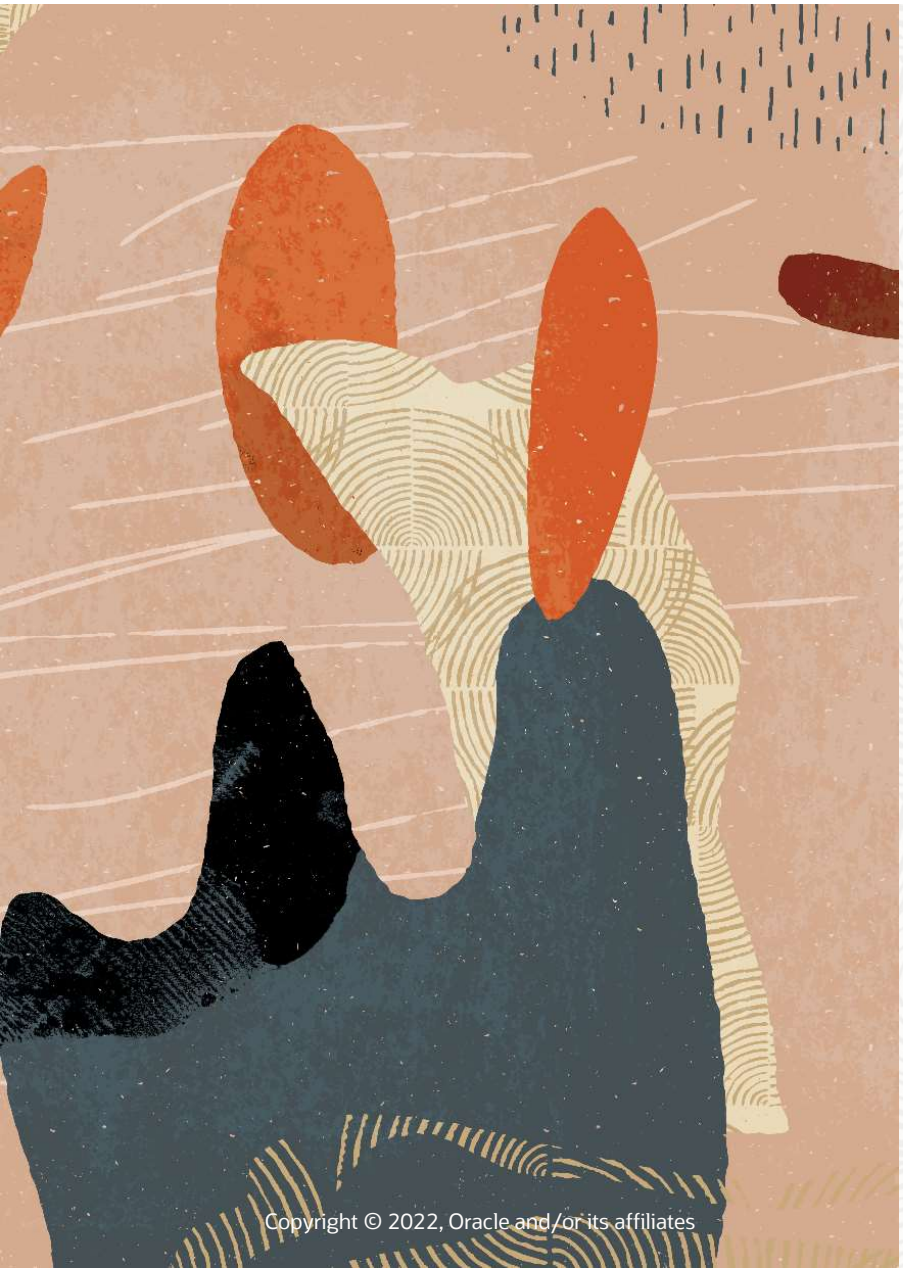
资深解决方案工程师

2022年4月29日



Agenda

1. 走近甲骨文高级队列
2. 事务事件队列TEQ
3. Oracle TEQ和EventMesh

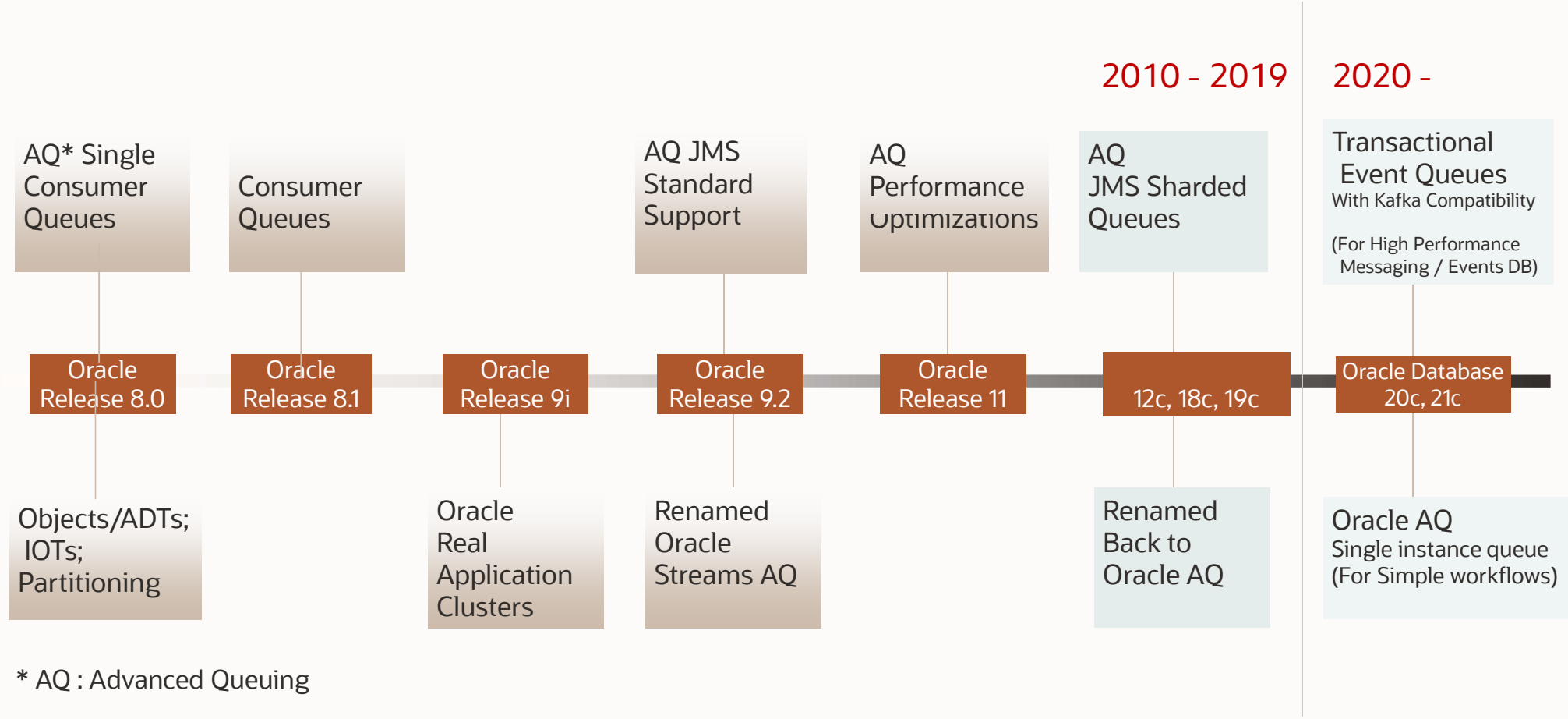


Agenda

1. **走近甲骨文高级队列**
2. **事务事件队列TEQ**
3. **Oracle TEQ和EventMesh**

Oracle 高级队列简史

从数据库的早期版本开始，不断演进的现代消息平台



* AQ : Advanced Queuing

用例参考

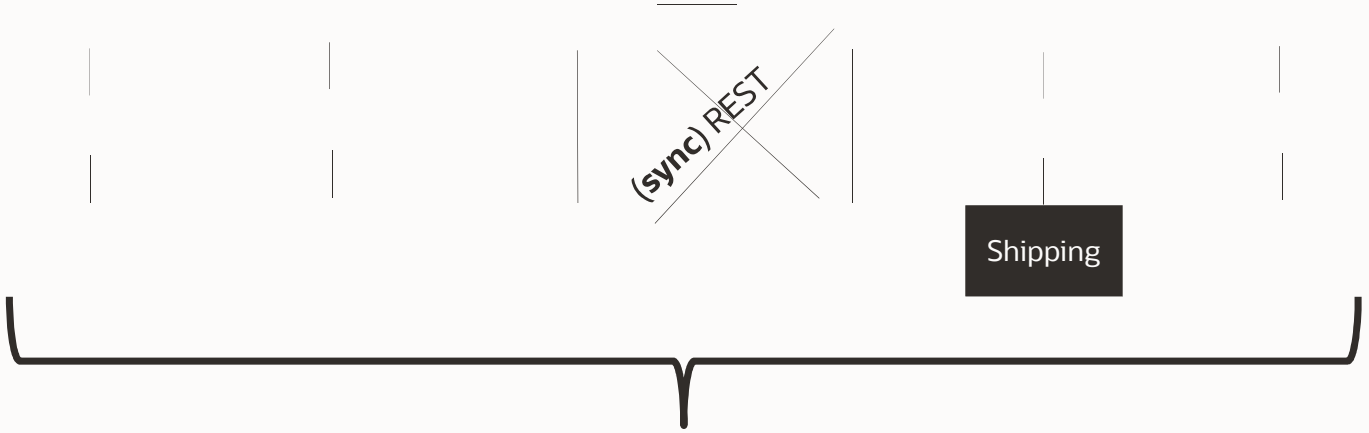
发布/订阅消息, 消息异步投递

- Oracle AQ 被广泛地用于万+客户
  - 在关键任务的应用中处理消息和事件

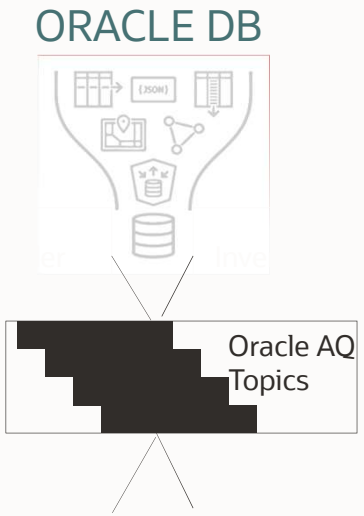
- 一些具有代表性的客户用例
  - 消息整合
  - 市场营销
  - 银行业务的安全认证OTP
  - 整合 IoT 设备/传感器事件
  - 简化应用工作流和在微服务结合 sagas 做处理
  - 云控制面板中的作业调度
  - 身份管理控制流程

用例 应用工作流的架构

Oracle 高级队列是富于弹性的异步消息平台，可简化工作流



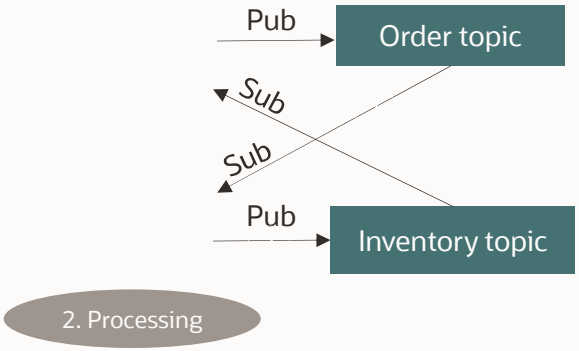
传统的工作流选择方案



AQ：简化&异步

使用 Oracle AQ 简化工作流

接收订单，检查库存，发送消息




```
inventory_json JSON_OBJECT_T;
BEGIN
  LOOP
    -- Wait for and dequeue the next order message
    dequeue_options.wait := dbms_aq.FOREVER;
    DBMS_AQ.DEQUEUE(
      queue_name => 'ORDERQUEUE',
      dequeue_options => dequeue_options,
      message_properties => message_properties,
      payload => message,
      msgid => message_handle);

    -- Parse the order message
    order_json := JSON_OBJECT_T.parse(message.text_vc);
    order_inv_id := order_json.get_string('itemid');

    -- Check the inventory
    update INVENTORYUSER.INVENTORY set inventorycount = inventorycount - 1
      where inventoryid = order_inv_id and inventorycount > 0 returning inventorylocation
      into order_inv_loc;
    if sql%rowcount = 0 then
      order_inv_loc := 'inventorydoesnotexist';
    end if;

    -- Construct the inventory message
    inventory_json := new JSON_OBJECT_T;
    inventory_json.put('orderid', order_json.get_string('orderid'));
    inventory_json.put('itemid', order_inv_id);
    inventory_json.put('inventorylocation', order_inv_loc);
    inventory_json.put('suggestiveSale', 'beer');

    -- Send the inventory message
    message := SYS.AQ$_JMS_TEXT_MESSAGE.construct;
    message.set_text(inventory_json.to_string());
    DBMS_AQ.ENQUEUE(queue_name => 'INVENTORYQUEUE',
      enqueue_options    => enqueue_options,
      message_properties => message_properties,
      payload            => message,
      msgid              => message_handle);

    -- commit
    commit;
  END LOOP;
END;
/
```

1 Wait for a message in the order queue

Extract the order information from the JSON message

2. Processing

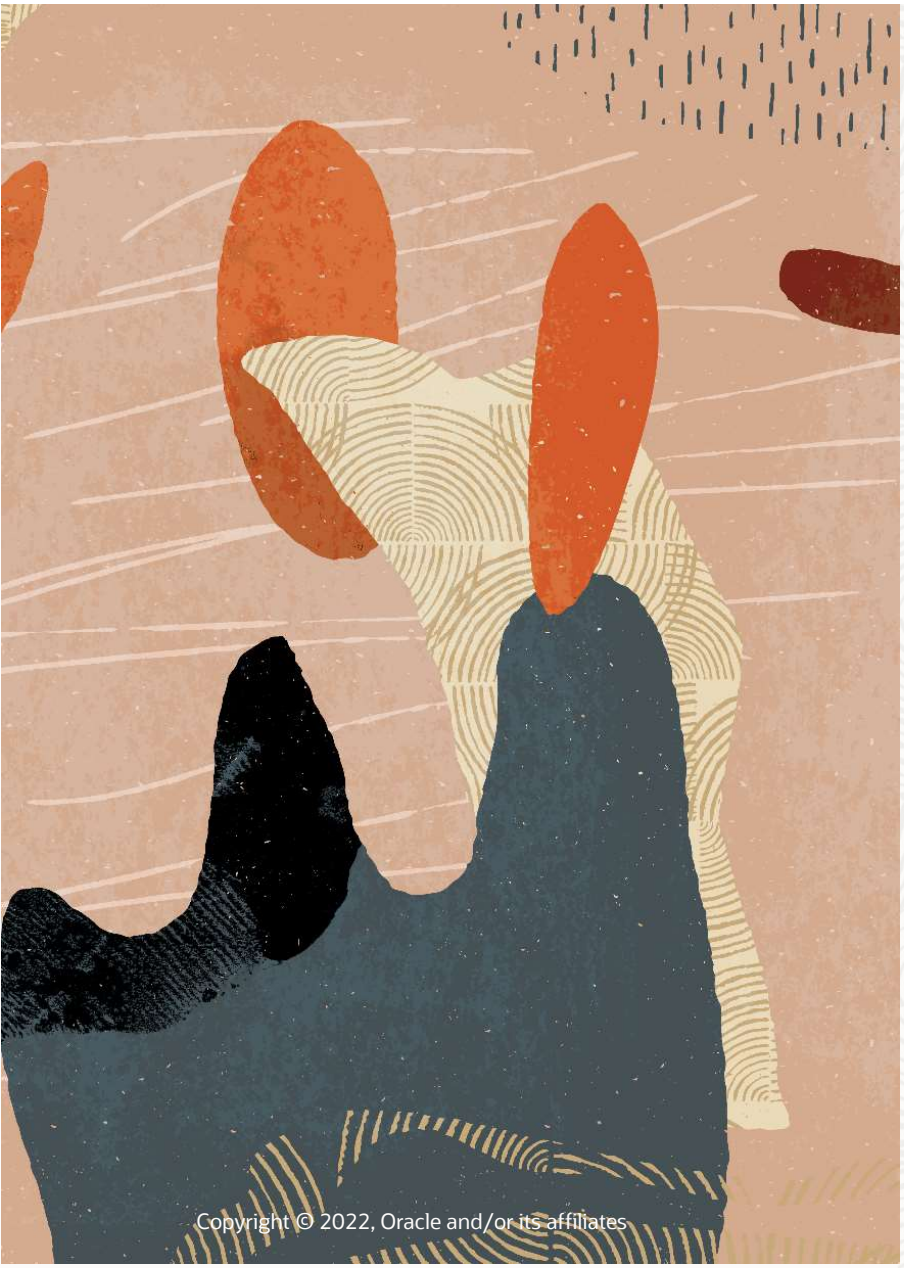
Check and update the inventory

Format an inventory message in JSON format

3 Send the inventory message

Commit and repeat

[详细参考]



Agenda

1 走近甲骨文高级队列

2 事务事件队列TEQ

3 Oracle TEQ和EventMesh

升级到 Oracle 数据库 (21c) 中的事务性事件队列 (TEQ)

数据库中的高吞吐量事务性事件网格

- Oracle AQ 经典队列的性能可以满足很多工作负载
  - 单节点数据库，每秒~2500 消息 (每天 **~200 万** 消息)
  - 使用跨多个队列的多个 PDB 能得到更高的吞吐量
- 对于具有更高的消息传递速率的用例，请考虑使用 TEQ
  - 几乎完全相同的API (少量例外)
  - In-memory 的持久队列 – 一致且高性能
  - 分区队列 – 基于键值的自动分区
  - 消息排序 – 每个生产者，每个事件流
  - 点对点的发布/订阅
  - 精确的消息传递
  - 多语言访问
  - 新订阅者可以回溯早期消息
  - Kafka Java 客户端API and Kafka JMS 连接器
  - 企业级安全

核心概念

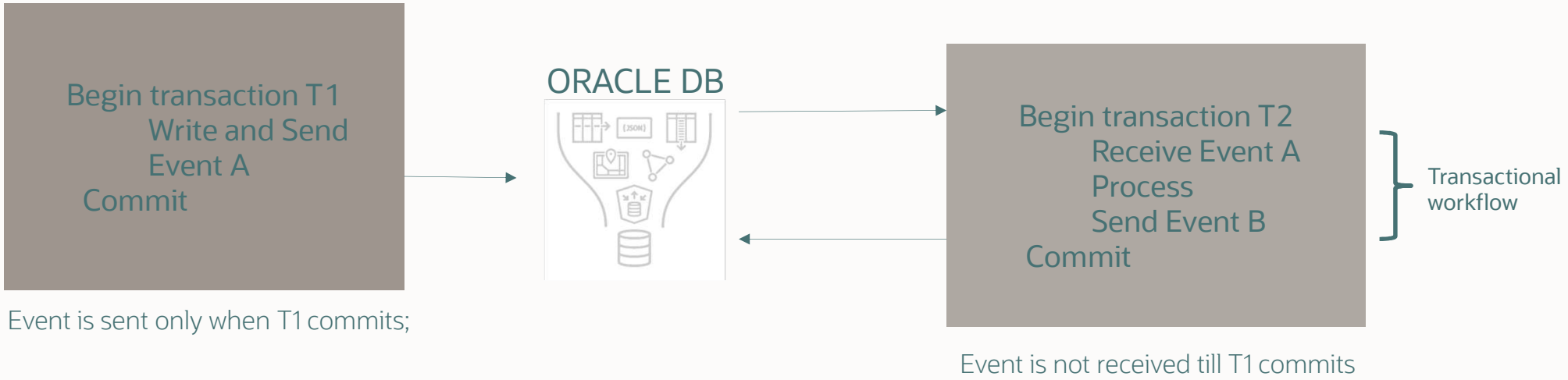
Oracle TEQ 队列表/事件流 和 Kafka 主题/分区的映射

产品	主体	组成部分	内部划分	寻址器
Transactional Event Queues	**队列表**	**事件流**	**分区**	**Offset**
Apache Kafka	主题	分区	段	Offset
OCI Streaming (OSS)	流	分区	-	Cursor
JMS / Oracle AQ / IBM MQ etc.	队列/主题		N/A	

Queue Table: 一个已分区的、仅附加的消息队列
Event Stream: 一个有序的消息集合。生产者和消费者可以同时处理多个事件流
Partition: 事件流的一部分，用于高效的内存和磁盘管理
Offset: 使用者用来出队消息的地址
Enqueue: 生产者向队列表入队一条消息
Dequeue: 消费者从队列表出队一条消息

准确的事务性消息传递

事务处理对数据和消息整体完整性的重要性

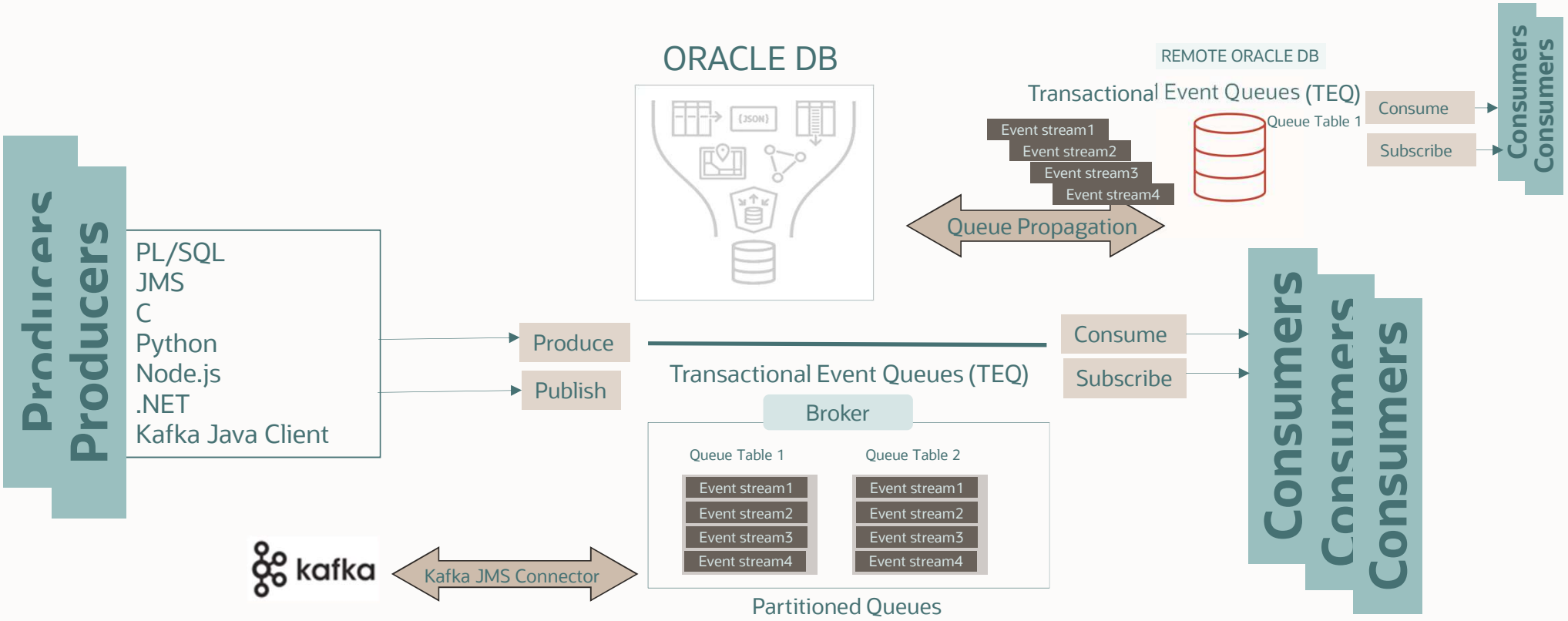


事务性事件队列的优势

1. 合并 DML 操作和 事件于同一个数据库事务
2. 保证一个事件交付且只交付一次
3. 可以轻松地创建链接式事件驱动的工作流 – 接收, 处理, 发送

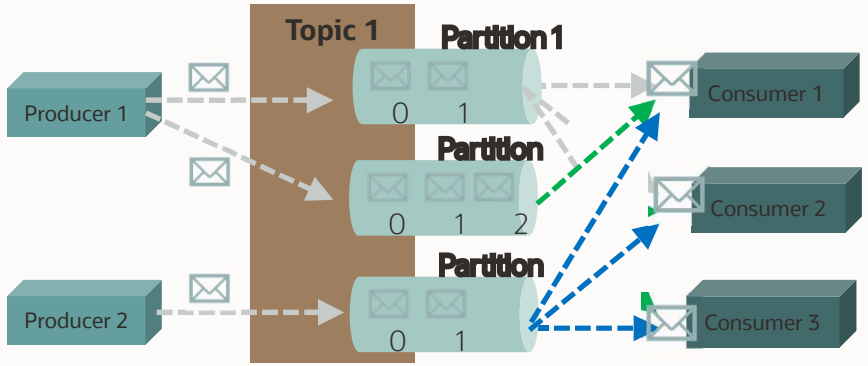
Oracle 融合数据库中的 TEQ

事件的发步/订阅 和 消息的生产/消费 合而为一

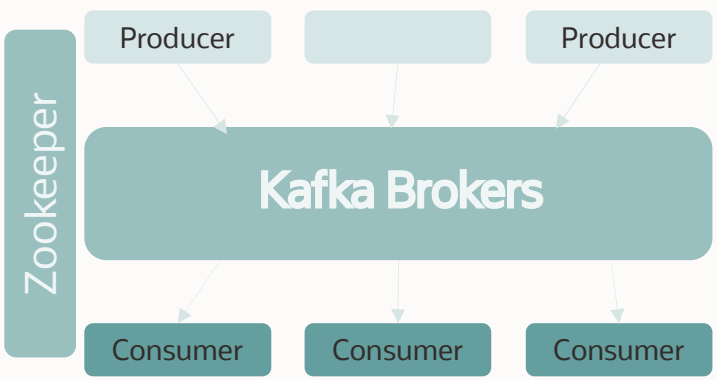


TEQ 和 Kafka Java 客户端结合

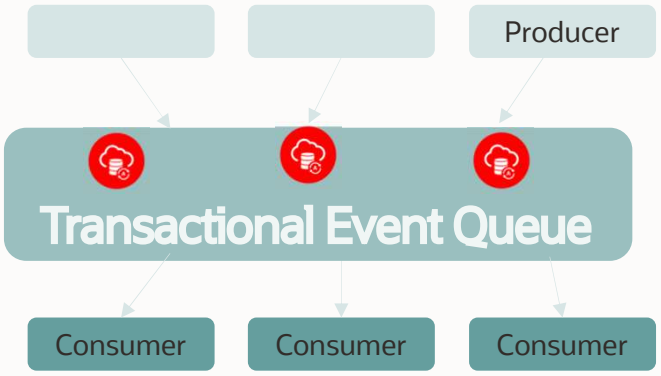
重用 Kafka Java 代码



Kafka Topics and Partitions map to TEQ Queue Tables and Event Streams



Interoperate
Or
Replace



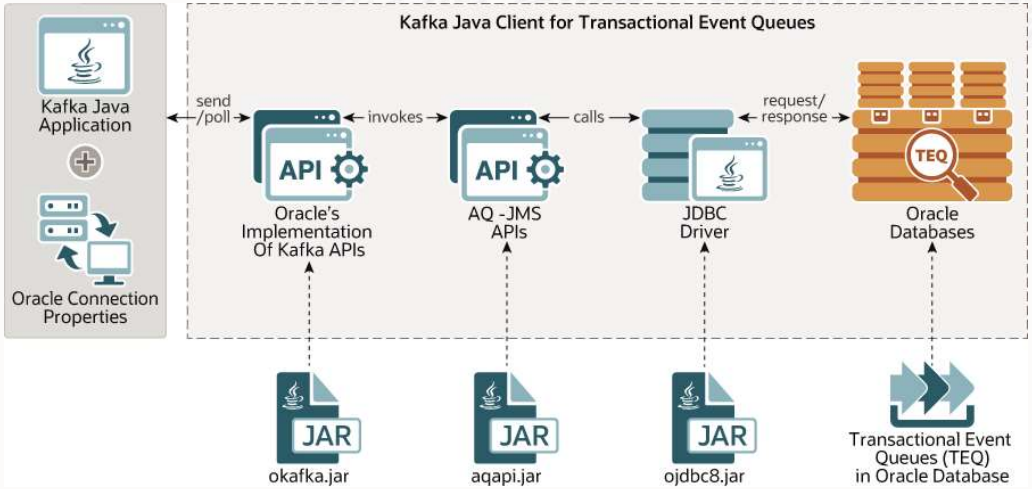
props.put("bootstrap.servers", "<kafka>:9092")

props.put("bootstrap.servers", "<db>:1521")

Kafka Java Client 兼容性

总结

- ✓ 代码修改量不大
- ✓ 新的RU可以工作



原理

- ✓ 数据库监听端口1521
- ✓ 通过JDBC连接
- ✓ okafka.jar的实现

平台	数据库版本	RU	兼容与否
OCI DBCS	21C	21.4	兼容
Compute VM(OP)	19C	19.12	兼容
ADB(ATP)	19C	19.14	兼容
Compute VM(OP)	19C	19.3	不兼容

参考

代码Sample - Producer

Kafka

```java
import java.util.Properties;
import org.apache.kafka.clients.producer.KafkaProducer;
import org.apache.kafka.clients.producer.ProducerRecord;

public class kkProducer {

public static void main(String[] args) {

    if(args.length != 1) {
      System.out.println("Please provide topic name to produce messages.");
      return ;
    }
    String topic = args[0].trim();
    KafkaProducer<String,String> prod = null;
    Properties props = new Properties();
    //
    props.put("bootstrap.servers", "172.17.0.1:9092");
    //
    props.put("enable.auto.commit", "true");
    props.put("auto.commit.interval.ms", "10000");
    props.put("linger.ms", 1000);
    //
    props.setProperty("key.serializer", "org.apache.kafka.common.serialization.StringSerializer");
    props.setProperty("value.serializer", "org.apache.kafka.common.serialization.StringSerializer");


    System.out.println("Creating producer now");
    prod=new KafkaProducer<>(props);
    System.out.println("Producer created.");
    System.out.println("==============");
     try {
       int i;
       for(i = 0; i < 20; i++)    {
         prod.send(new ProducerRecord<String, String>(topic , 0, i+"000","This is new message"+i));
         System.out.println("Sent "+ i + " messages");
       }
     } catch(Exception ex) {
       System.out.println("Failed to send messages:");
       ex.printStackTrace();
     }
     finally {
       prod.close();
     }
  }
}
```

TEQ

```java
import java.util.Properties;
import org.oracle.okafka.clients.producer.KafkaProducer;
import org.oracle.okafka.clients.producer.ProducerRecord;

public class kkProducer {

public static void main(String[] args) {

    if(args.length != 1) {
      System.out.println("Please provide topic name to produce messages.");
      return ;
    }
    String topic = args[0].trim();
    KafkaProducer<String,String> prod = null;
    Properties props = new Properties();
    //
    props.put("oracle.user.name", "admin");
    props.put("oracle.password", "WelcomE__12345");
    props.put("oracle.service.name", "db0305_pdb1.sub03180225480.vcnkz2021.oraclevcn.com");
    props.put("oracle.net.tns_admin", "/tmp");
    props.put("security.protocol", "PLAINTEXT");
    props.put("bootstrap.servers", "kkk21c.sub03180225480.vcnkz2021.oraclevcn.com:1521");
    //
    props.put("enable.auto.commit", "true");
    props.put("auto.commit.interval.ms", "10000");
    props.put("linger.ms", 1000);
    //
    props.setProperty("key.serializer", "org.oracle.okafka.common.serialization.StringSerializer");
    props.setProperty("value.serializer", "org.oracle.okafka.common.serialization.StringSerializer");


    System.out.println("Creating producer now");
    prod=new KafkaProducer<>(props);
    System.out.println("Producer created.");
    System.out.println("==============");
     try {
       int i;
       for(i = 0; i < 20; i++)    {
         prod.send(new ProducerRecord<String, String>(topic ,0, i+"000","This is new message"+i));
         System.out.println("Sent "+ i + " messages");
       }
     } catch(Exception ex) {
       System.out.println("Failed to send messages:");
       ex.printStackTrace();
     }
     finally {
       prod.close();
     }
  }
}
```

代码Sample - Consumer

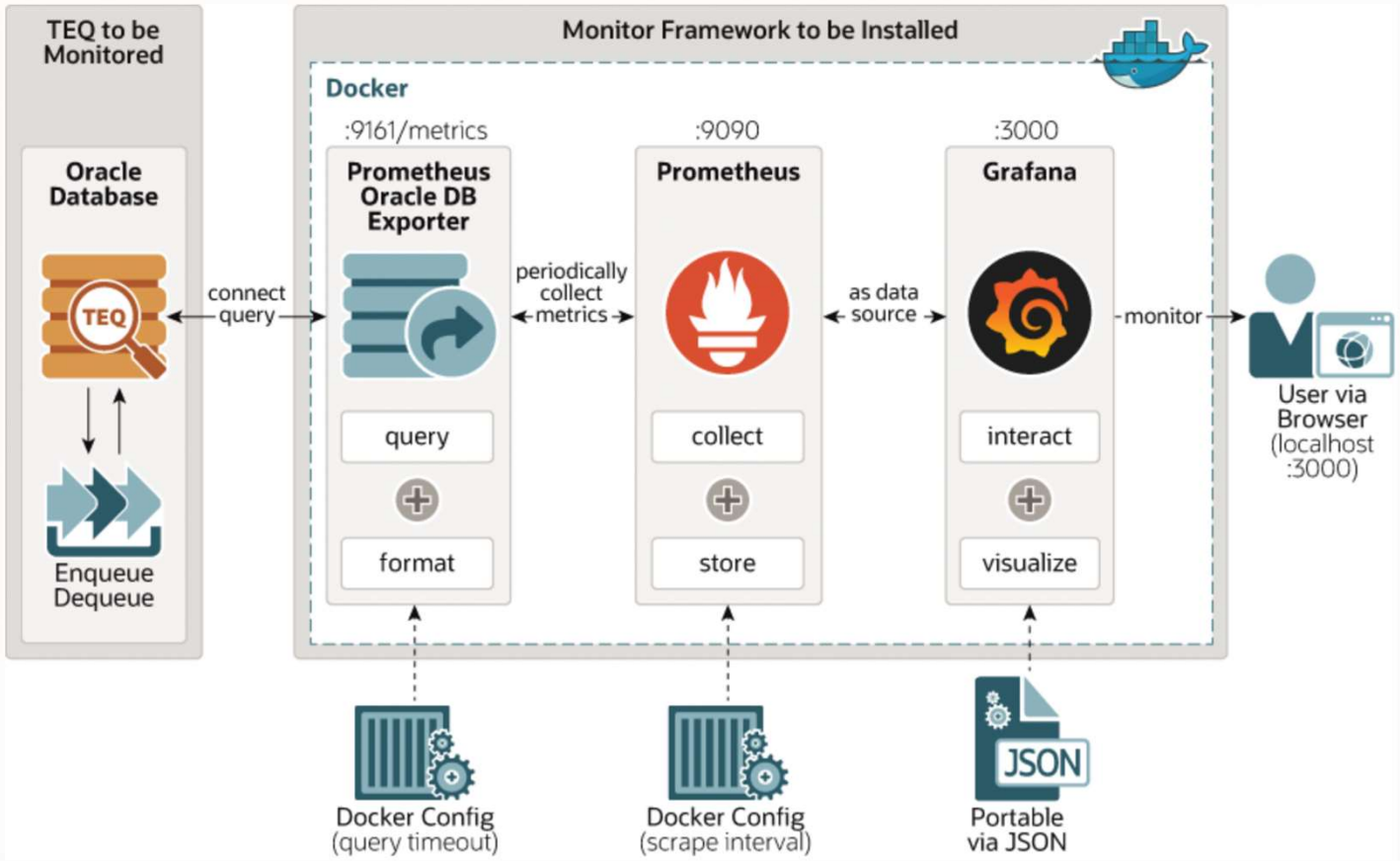
Kafka

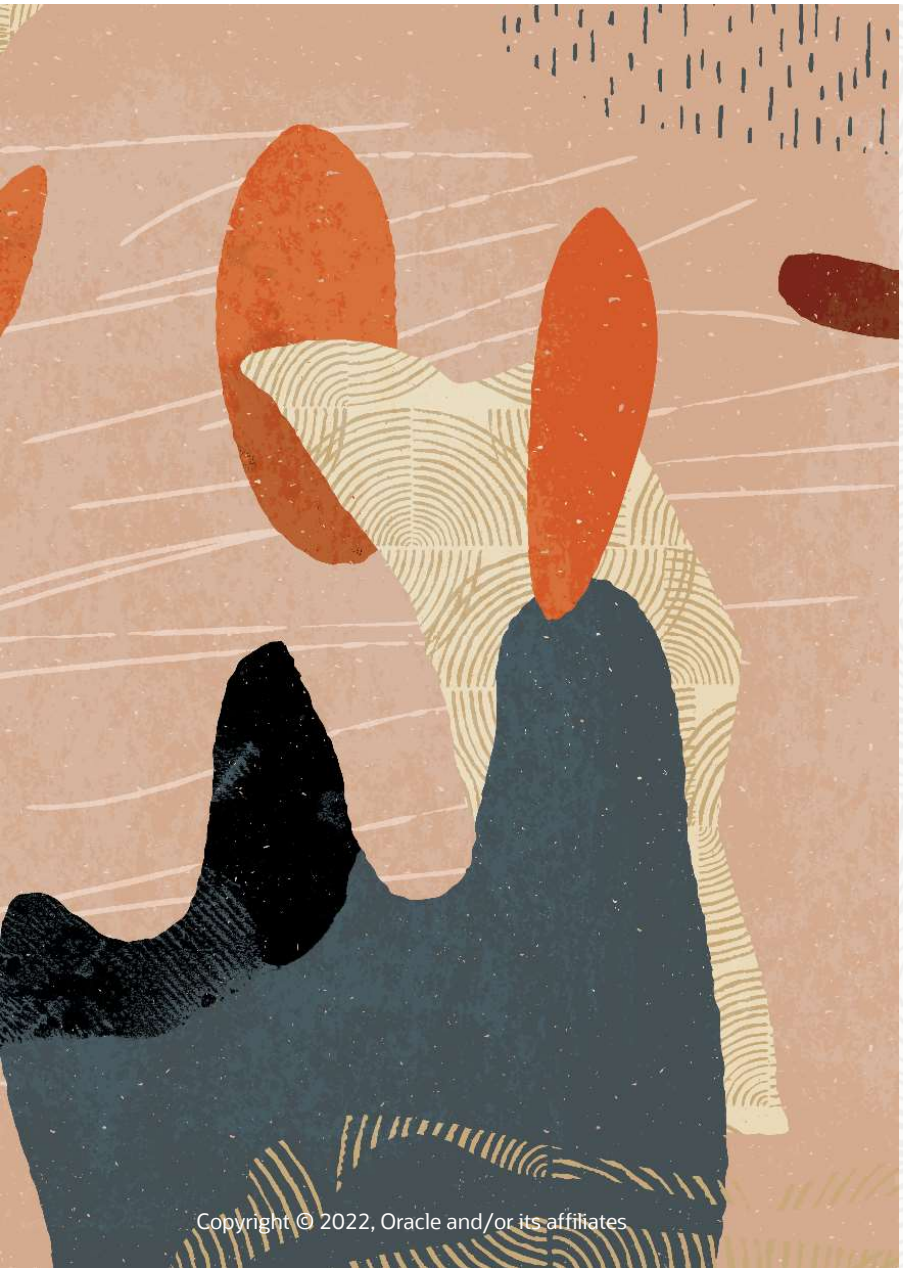
```java
import java.time.Duration;
import java.util.Arrays;
import java.util.Properties;
import org.apache.kafka.clients.consumer.ConsumerRecord;
import org.apache.kafka.clients.consumer.ConsumerRecords;
import org.apache.kafka.clients.consumer.KafkaConsumer;

public class kkConsumer {

public static void main(String[] args) {

    if(args.length != 1) {
      System.out.println("Please provide topic name to produce messages.");
      return ;
    }
    String topic = args[0].trim();
    Properties props = new Properties();
    props.put("bootstrap.servers", "172.17.0.1:9092");
    //
    props.put("enable.auto.commit", "true");
    props.put("auto.commit.interval.ms", "10000");
    props.put("linger.ms", 1000);
    props.put("group.id", "group-0");
    //
    props.put("key.deserializer", "org.apache.kafka.common.serialization.StringDeserializer");
    props.put("value.deserializer", "org.apache.kafka.common.serialization.StringDeserializer");
    //
    System.out.println("Creating consumer now");
    KafkaConsumer<String, String> consumer = null;
    consumer = new KafkaConsumer<String, String>(props);
    System.out.println("Consumer created.");
    //
    consumer.subscribe(Arrays.asList(topic));
    ConsumerRecords<String, String> records = null;
    System.out.println("===============");
    //
    try {
        //
        records = consumer.poll(Duration.ofMillis(15000));
        for (ConsumerRecord<String, String> record : records) {
            System.out.println("topic = , partition=  ,key= , value = \n"
                    + record.topic() + "  " + record.partition() + "  " + record.key() + "  " + record.value());
            System.out.println(".......");
        }

        consumer.commitSync();
    } catch (Exception ex) {
        ex.printStackTrace();

    } finally {
        consumer.close();
    }
  }
}
```

TEQ

```java
import java.time.Duration;
import java.util.Arrays;
import java.util.Properties;
import org.oracle.okafka.clients.consumer.ConsumerRecord;
import org.oracle.okafka.clients.consumer.ConsumerRecords;
import org.oracle.okafka.clients.consumer.KafkaConsumer;

public class kkConsumer {

public static void main(String[] args) {

    if(args.length != 1) {
      System.out.println("Please provide topic name to produce messages.");
      return ;
    }
    String topic = args[0].trim();
    //
    Properties props = new Properties();
    props.put("oracle.user.name", "admin");
    props.put("oracle.password", "WelcomE__12345");
    props.put("oracle.service.name", "db0305_pdb1.sub03180225480.vcnkz2021.oraclevcn.com");
    props.put("oracle.net.tns_admin", "/tmp"); //eg: "/user/home" if ojdbc.properies file is in home
    props.put("security.protocol", "PLAINTEXT");
    props.put("bootstrap.servers", "kkk21c.sub03180225480.vcnkz2021.oraclevcn.com:1521");
    //
    props.put("enable.auto.commit", "true");
    props.put("auto.commit.interval.ms", "10000");
    props.put("linger.ms", 1000);
    props.put("group.id", "consumers");
    //
    props.put("key.deserializer", "org.oracle.okafka.common.serialization.StringDeserializer");
    props.put("value.deserializer", "org.oracle.okafka.common.serialization.StringDeserializer");
    //
    System.out.println("Creating consumer now");
    KafkaConsumer<String, String> consumer = null;
    consumer = new KafkaConsumer<String, String>(props);
    System.out.println("Consumer created.");
    //
    consumer.subscribe(Arrays.asList(topic));
    ConsumerRecords<String, String> records = null;
    System.out.println("===============");
    //
    try {
        //
        records = consumer.poll(Duration.ofMillis(15000));
        for (ConsumerRecord<String, String> record : records) {
            System.out.println("topic = , partition=  ,key= , value = \n"
                    + record.topic() + "  " + record.partition() + "  " + record.key() + "  " + record.value());
            System.out.println(".......");
        }

        consumer.commitSync();
    } catch (Exception ex) {
        ex.printStackTrace();

    } finally {
        consumer.close();
    }
```

监控TEQ





Agenda

1 走近甲骨文高级队列

2 事务事件队列TEQ

3 Oracle TEQ和EventMesh

事件网格是什么

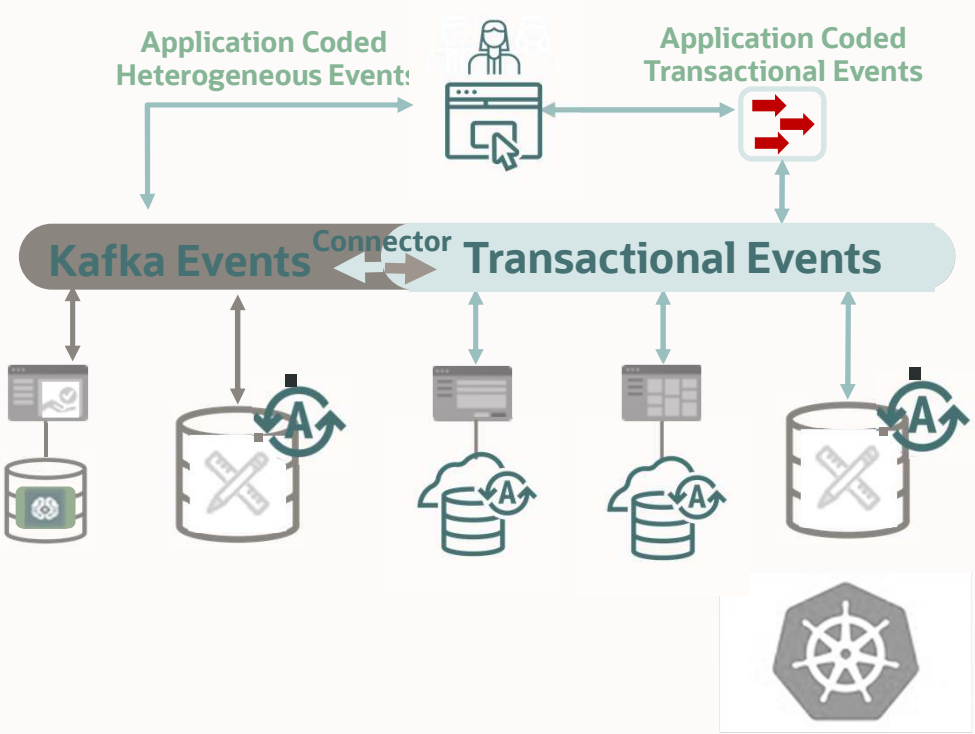
用于在微服务之间进行安全、高效和可靠地路由事件的，一个由事件代理组成的网络

事件网格的优势所在

- 处理所有的事件类型 – 逻辑，数据，应用程序等
- 保留事件的顺序
- 具有处理延迟、过期消息的能力
- 保证精确消息投递
- 为订阅者发送通知
- 针对智能和实时工作流的事件处理
- 事件和数据的安全性
- 性能 – 高吞吐和低延迟

Oracle TEQ 是为微服务而生的事件网格

应用程序编码的 Kafka 事件遍布于企业各处



应用编码事件

- 事件处理器往往被开发者硬编码
- 事件处理器实现操作、更改数据库状态，并生成其他事件

TEQ 事件网格的优势

- 事务性事件的完整性
- 微服务事件路由
- 应用程序编码事件
- Kafka 互用性
- 高吞吐
- 多语言和 REST
- Oracle 数据库开箱即用
- 集成自治数据库
- APEX 低代码可用

Oracle TEQ as EventMesh, 让您站在两个巨人肩膀之上!

接口开放，自动分区扩展，事件和消息

IBM MQ

RabbitMQ

JMS

Best of Messaging

kafka

Best of Streaming

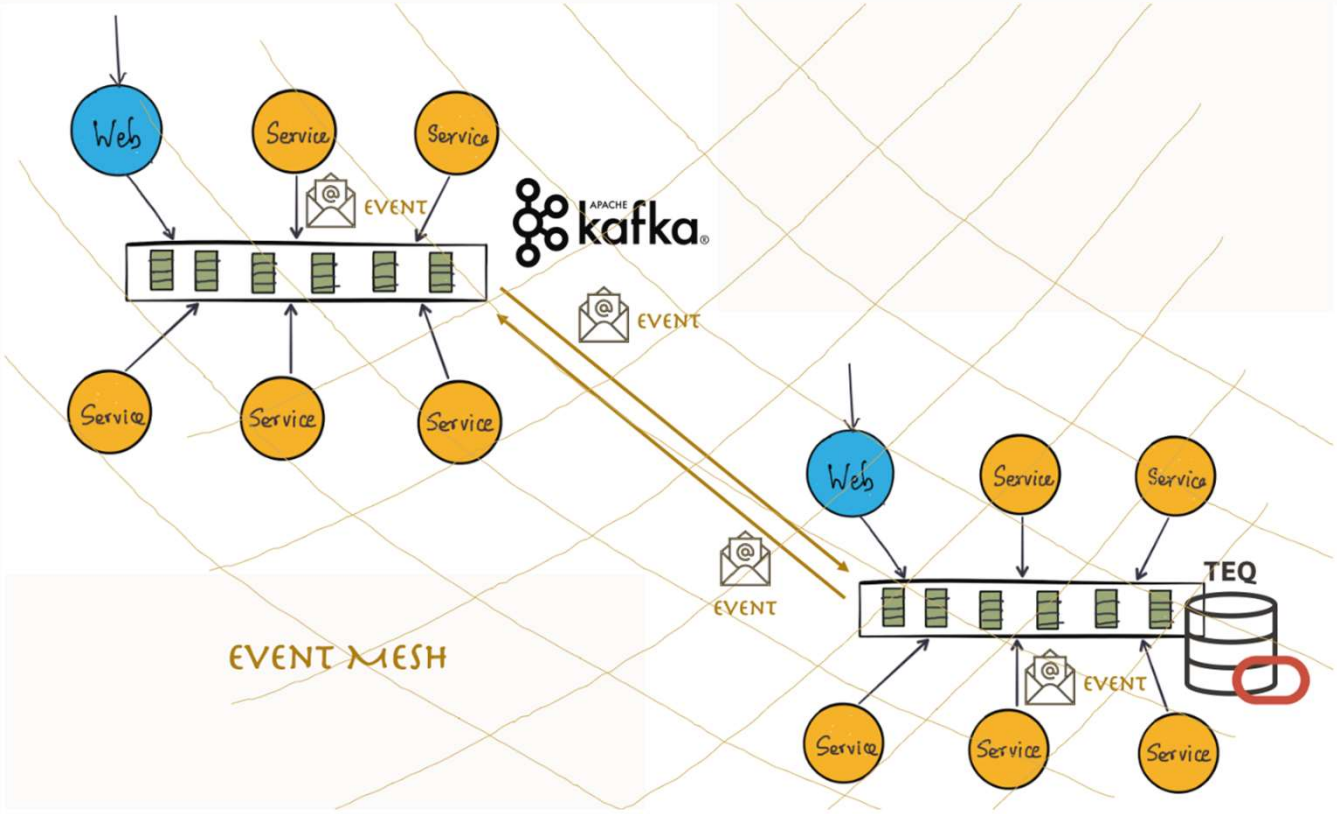
Transactional Event Queues (TEQ) in Oracle database

ORACLE DB

Transactional Event Queues
In Oracle Database

演示

Oracle TEQ & Event Mesh



Oracle 经典队列
简化工作流

Oracle TEQ
事件网格

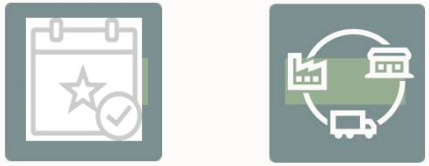
1

Oracle 经典队列/TEQ (21c)内置于数据库中，无需额外付费

In-Database Containers | Docker for Database | Kubernetes for Database | Database as Cloud Service | Autonomous Database

2

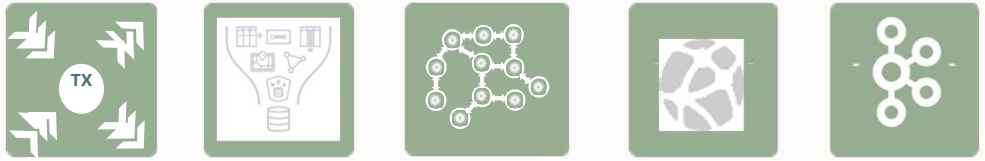
Oracle AQ 可以助力开发企业级应用的工作流



Events & Messages | Workflows

3

TEQ 和融合数据库可以简化微服务，TEQ是天生的EventMesh



Transactional Event | Converged Database | Microservices | Data Mesh | Apache Kafka

Q&A