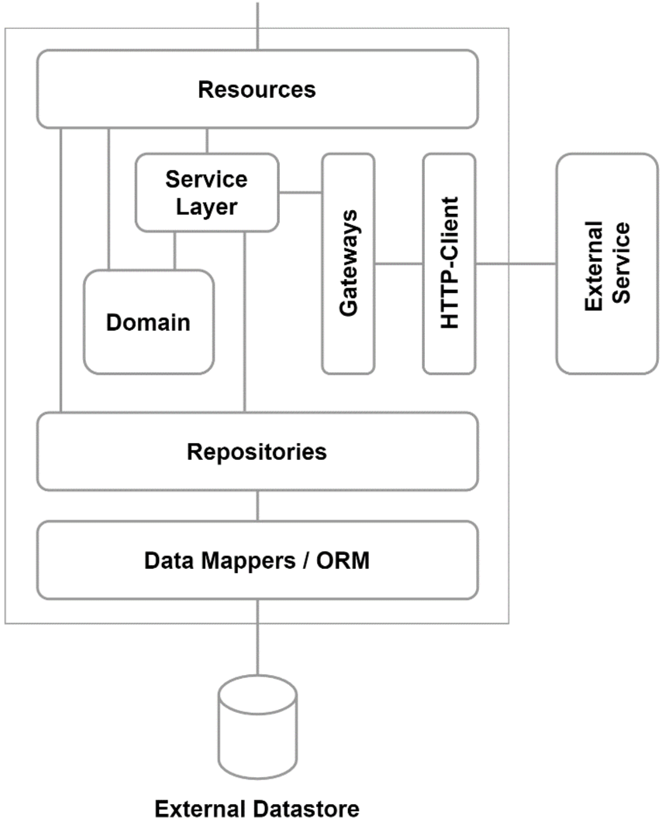
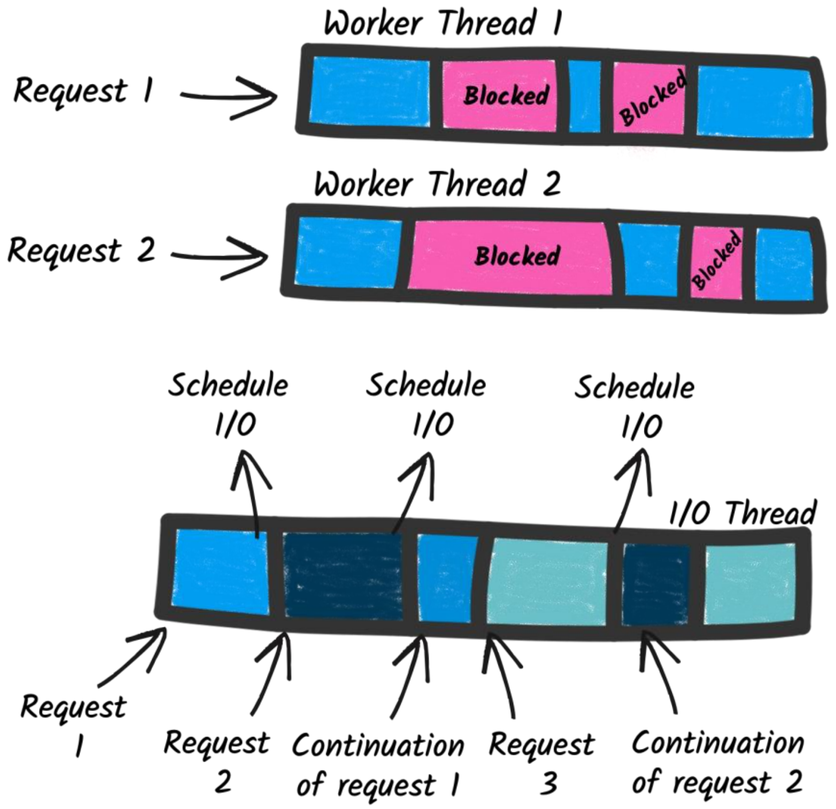
Investigating the Performance Of Reactive Libraries in a Quarkus Microservice

Denis Angeletta
RETIT GmbH

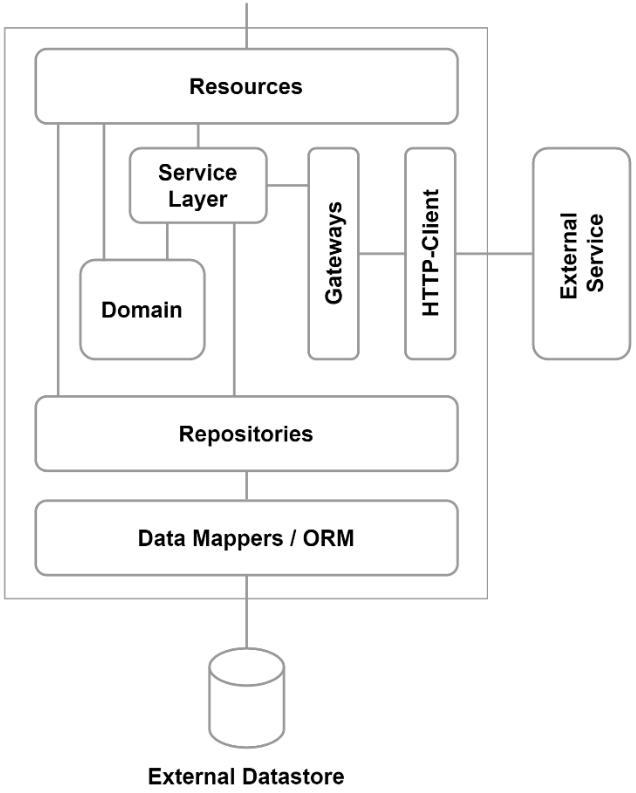
Reactive Stack



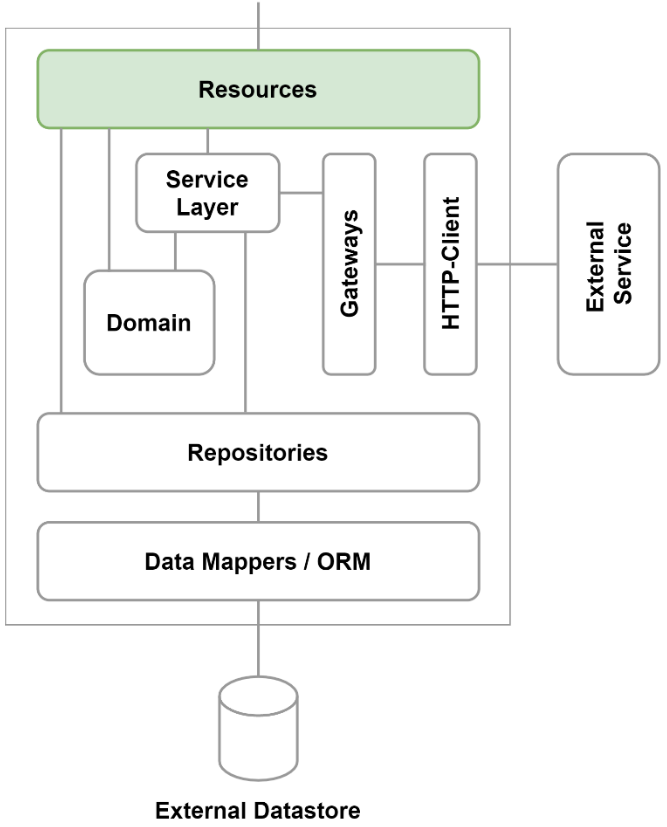
I/O-Threads


Worker Thread 1
Request 1
Blocked
Blocked
Worker Thread 2
Request 2
Blocked
Blocked
Schedule I/O
Schedule I/O
Schedule I/O
I/O Thread
Request 1
Request 2
Continuation of request 1
Request 3
Continuation of request 2

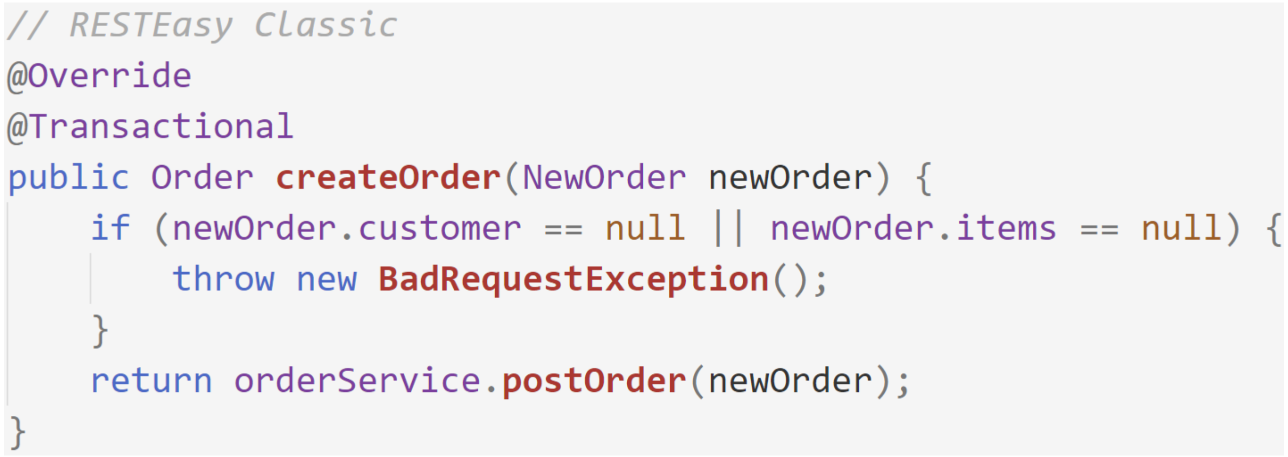
Reactive Stack - Quarkus



Reactive Stack - RESTEasy



RESTEasy - Example

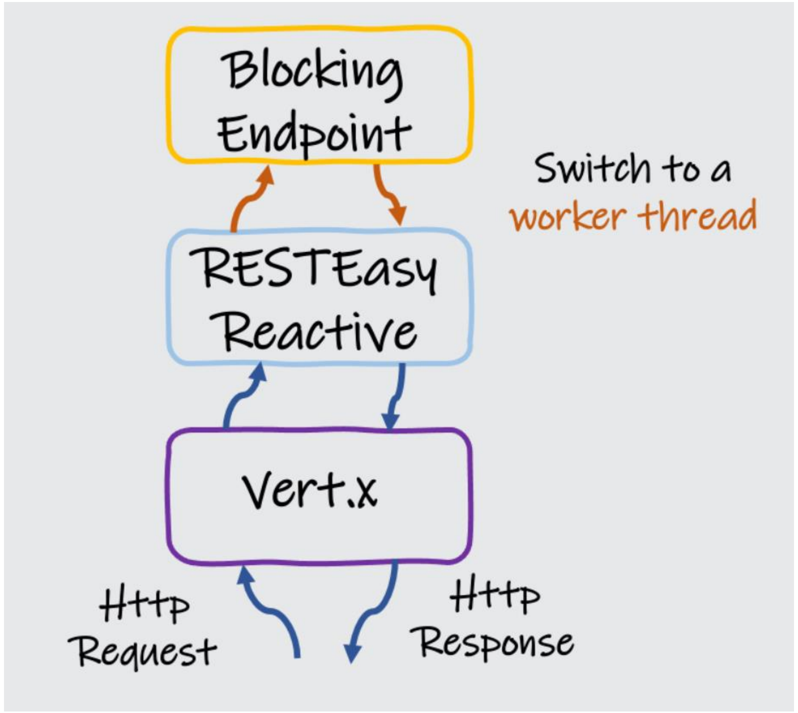


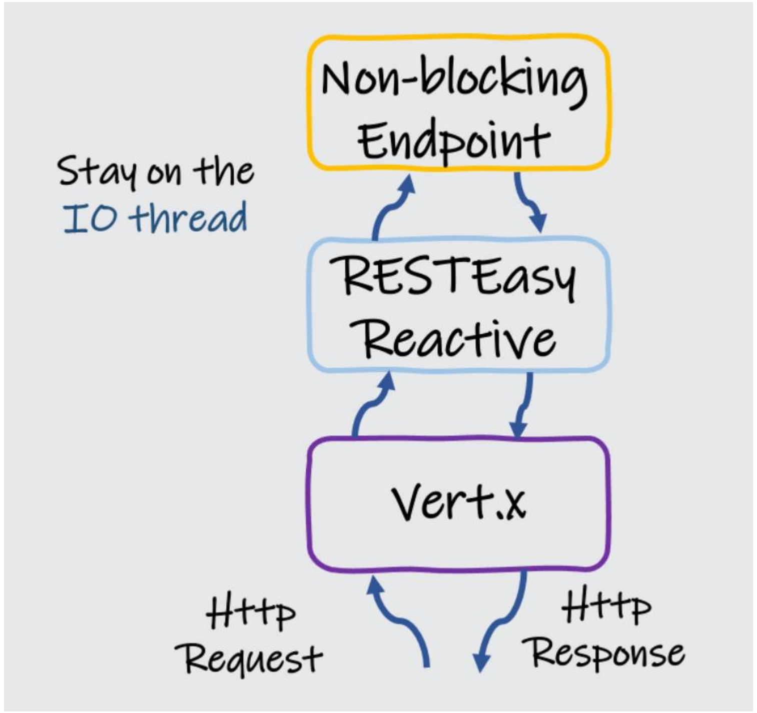
```java
// RESTEasy Classic
@Override
@Transactional
public Order createOrder(NewOrder newOrder) {
    if (newOrder.customer == null || newOrder.items == null) {
        throw new BadRequestException();
    }
    return orderService.postOrder(newOrder);
}
```

RESTEasy Reactive - Example

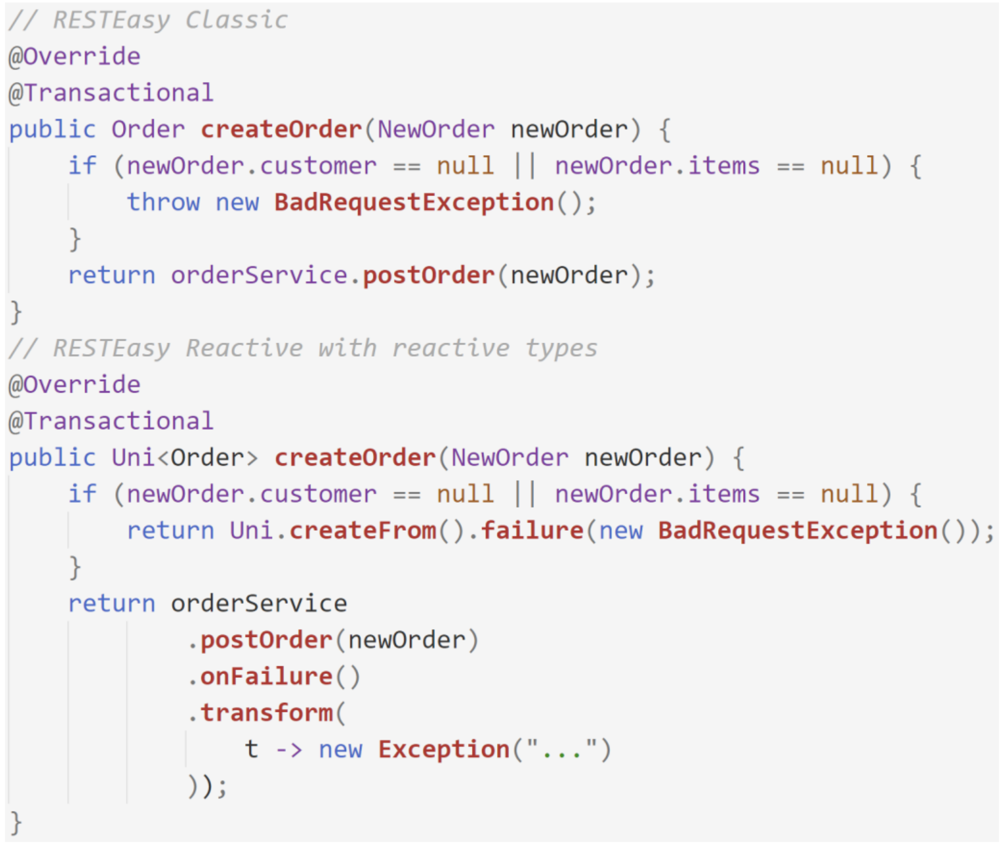
```java
// RESTEasy Classic
@Override
@Transactional
public Order createOrder(NewOrder newOrder) {
    if (newOrder.customer == null || newOrder.items == null) {
        throw new BadRequestException();
    }
    return orderService.postOrder(newOrder);
}

// RESTEasy Reactive
@Override
@Transactional
@Blocking
public Order createOrder(NewOrder newOrder) {
    if (newOrder.customer == null || newOrder.items == null) {
        throw new BadRequestException();
    }
    return orderService.postOrder(newOrder);
}
```

RESTEasy Classic vs RESTEasy Reactive





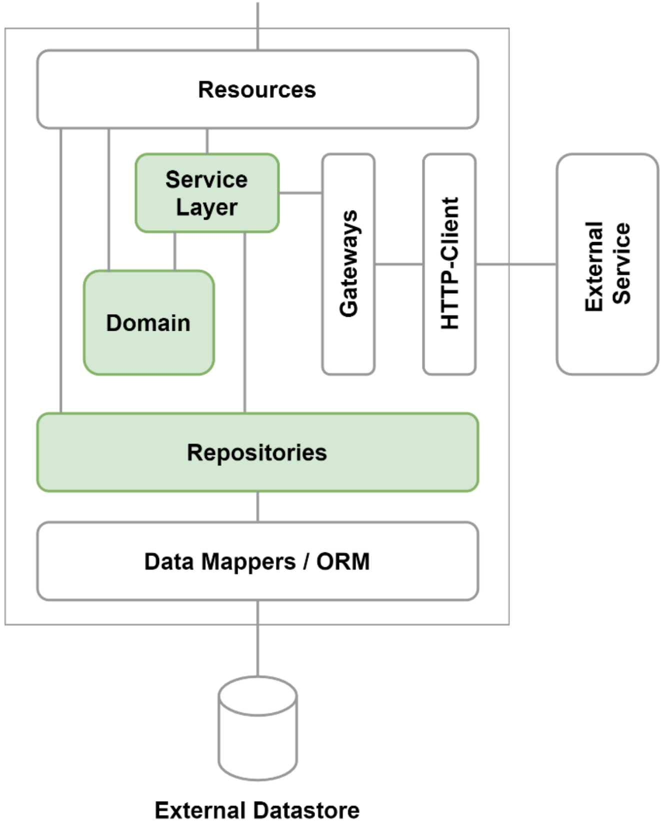
RESTEasy Reactive - Example



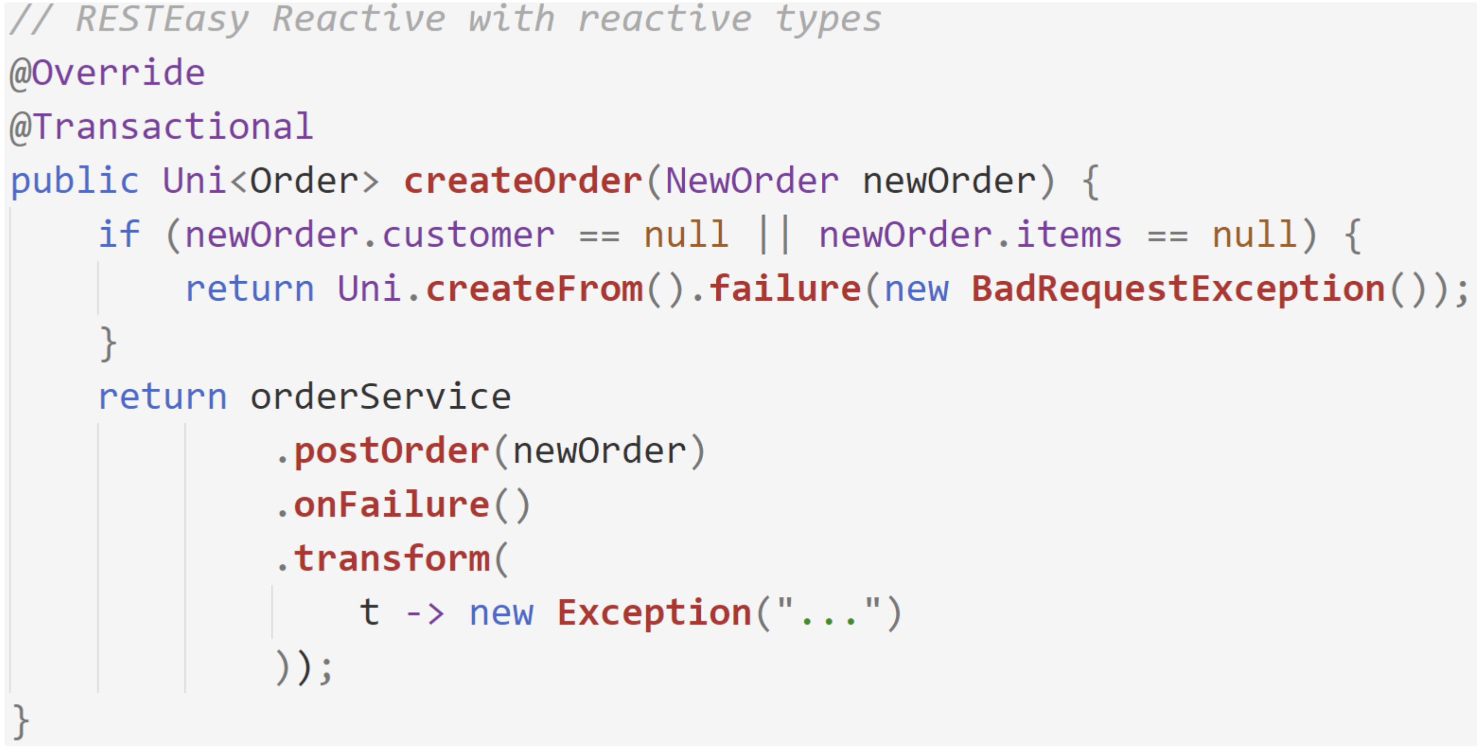
```java
// RESTEasy Classic
@Override
@Transactional
public Order createOrder(NewOrder newOrder) {
    if (newOrder.customer == null || newOrder.items == null) {
        throw new BadRequestException();
    }
    return orderService.postOrder(newOrder);
}
// RESTEasy Reactive with reactive types
@Override
@Transactional
public Uni<Order> createOrder(NewOrder newOrder) {
    if (newOrder.customer == null || newOrder.items == null) {
        return Uni.createFrom().failure(new BadRequestException());
    }
    return orderService
            .postOrder(newOrder)
            .onFailure()
            .transform(
                t -> new Exception("...")
            ));
}
```

Reactive Stack - Mutiny

Mutiny

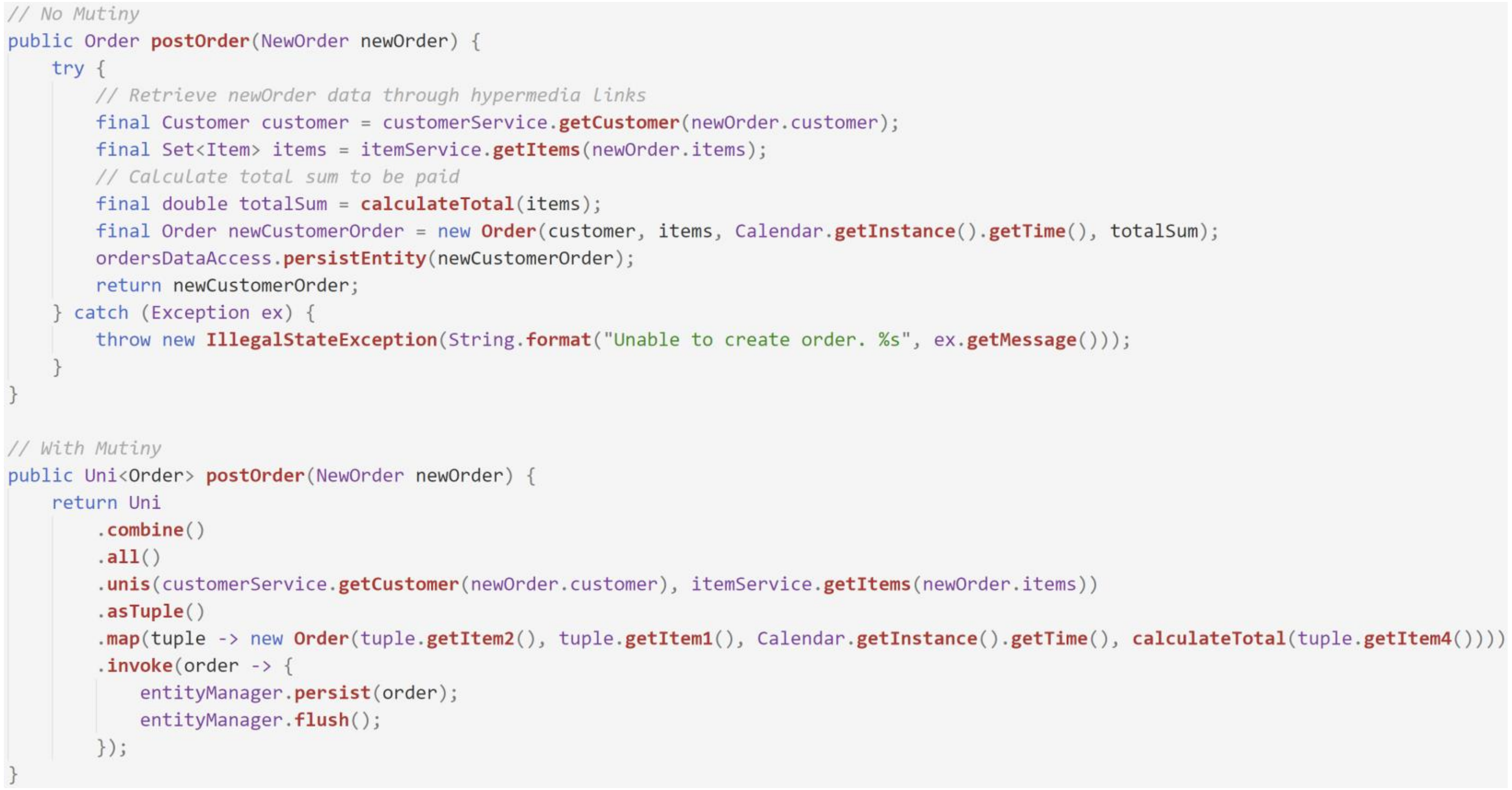


Mutiny - Example



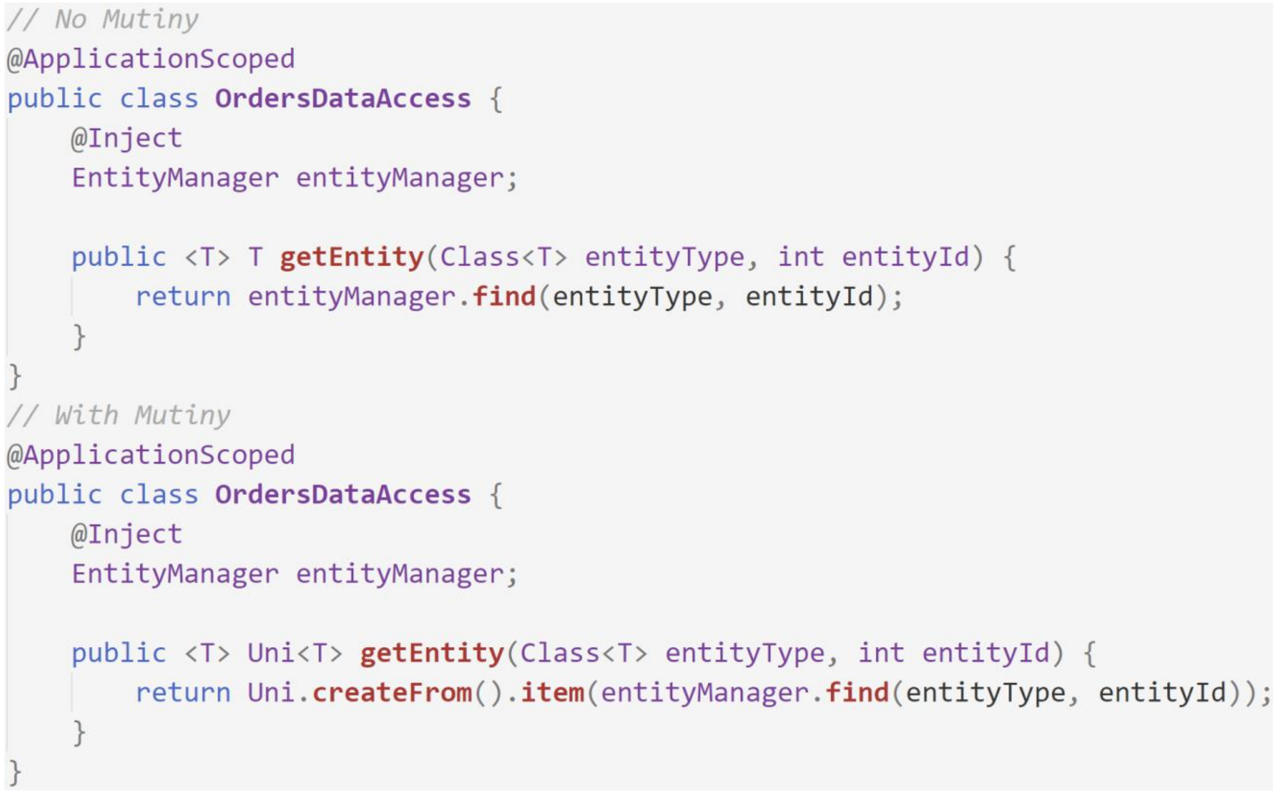
```java
// RESTEasy Reactive with reactive types
@Override
@Transactional
public Uni<Order> createOrder(NewOrder newOrder) {
    if (newOrder.customer == null || newOrder.items == null) {
        return Uni.createFrom().failure(new BadRequestException());
    }
    return orderService
            .postOrder(newOrder)
            .onFailure()
            .transform(
                t -> new Exception("...")
            ));
}
```

Mutiny - Example



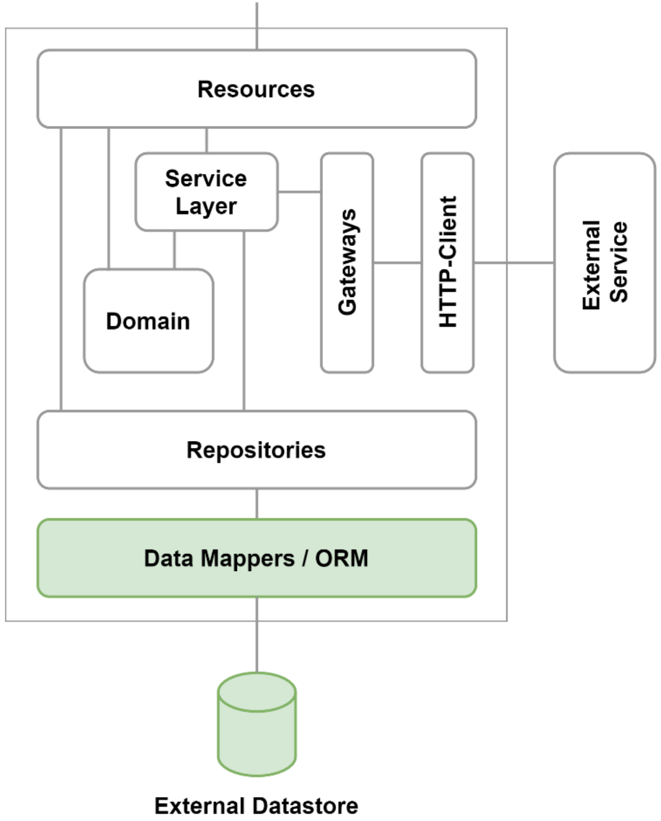
```java
// No Mutiny
public Order postOrder(NewOrder newOrder) {
    try {
        // Retrieve newOrder data through hypermedia links
        final Customer customer = customerService.getCustomer(newOrder.customer);
        final Set<Item> items = itemService.getItems(newOrder.items);
        // Calculate total sum to be paid
        final double totalSum = calculateTotal(items);
        final Order newCustomerOrder = new Order(customer, items, Calendar.getInstance().getTime(), totalSum);
        ordersDataAccess.persistEntity(newCustomerOrder);
        return newCustomerOrder;
    } catch (Exception ex) {
        throw new IllegalStateException(String.format("Unable to create order. %s", ex.getMessage()));
    }
}

// With Mutiny
public Uni<Order> postOrder(NewOrder newOrder) {
    return Uni
        .combine()
        .all()
        .unis(customerService.getCustomer(newOrder.customer), itemService.getItems(newOrder.items))
        .asTuple()
        .map(tuple -> new Order(tuple.getItem2(), tuple.getItem1(), Calendar.getInstance().getTime(), calculateTotal(tuple.getItem4())))
        .invoke(order -> {
            entityManager.persist(order);
            entityManager.flush();
        });
}
```

Mutiny - Example

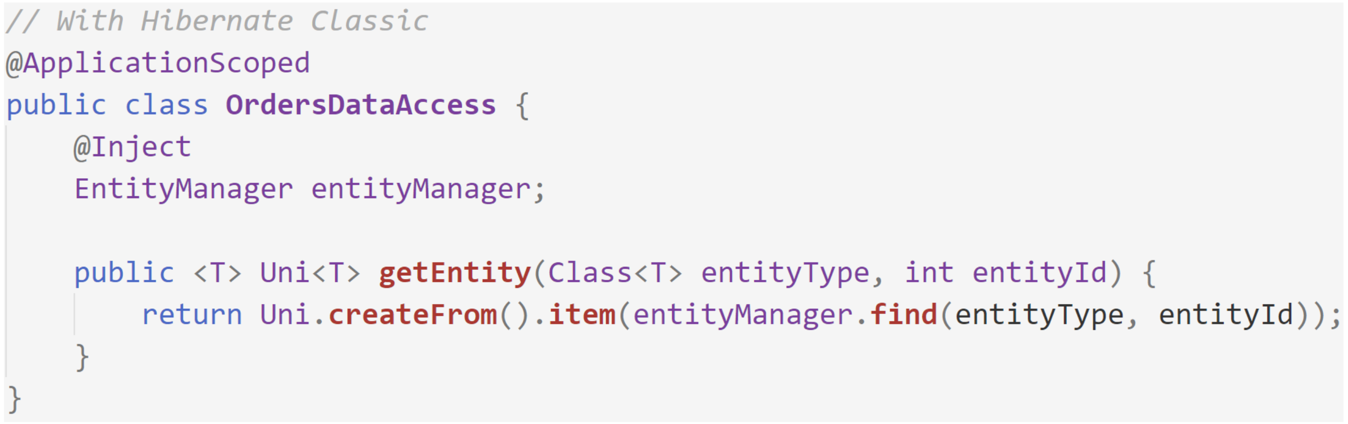


```java
// No Mutiny
@ApplicationScoped
public class OrdersDataAccess {
    @Inject
    EntityManager entityManager;

    public <T> T getEntity(Class<T> entityType, int entityId) {
        return entityManager.find(entityType, entityId);
    }
}
// With Mutiny
@ApplicationScoped
public class OrdersDataAccess {
    @Inject
    EntityManager entityManager;

    public <T> Uni<T> getEntity(Class<T> entityType, int entityId) {
        return Uni.createFrom().item(entityManager.find(entityType, entityId));
    }
}
```

Reactive Stack – Hibernate & Datasource Clients

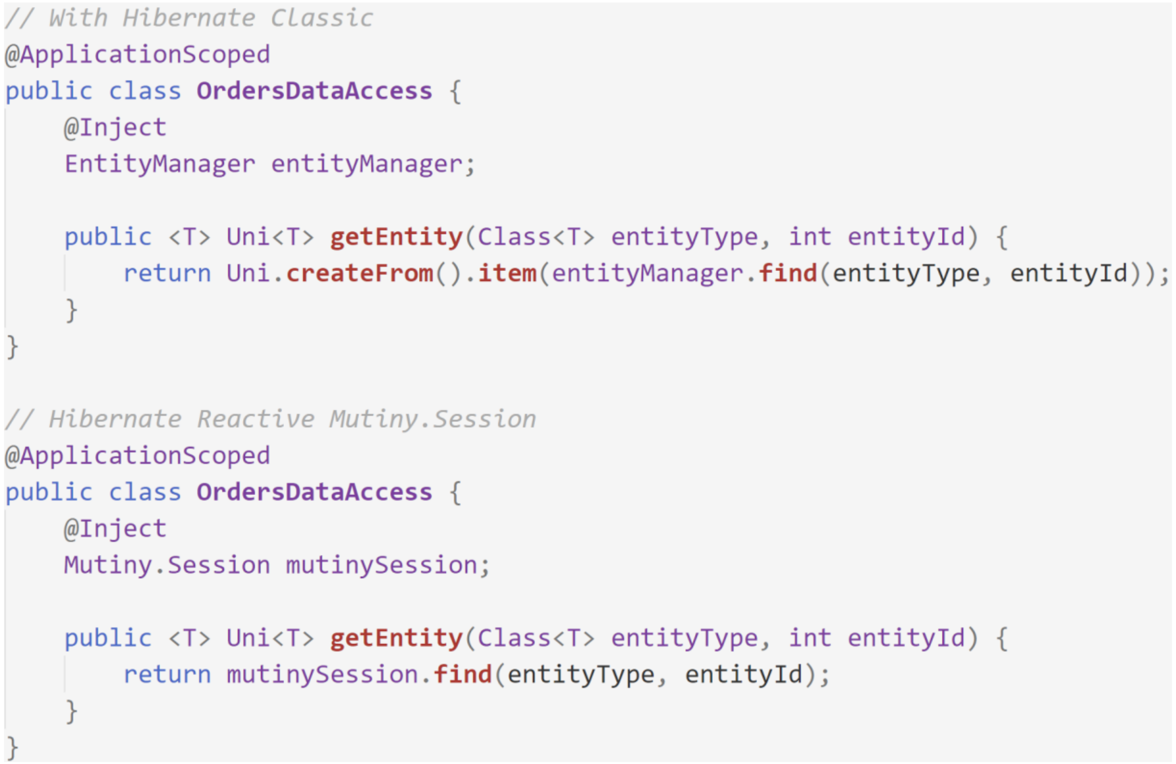


Hibernate - Example



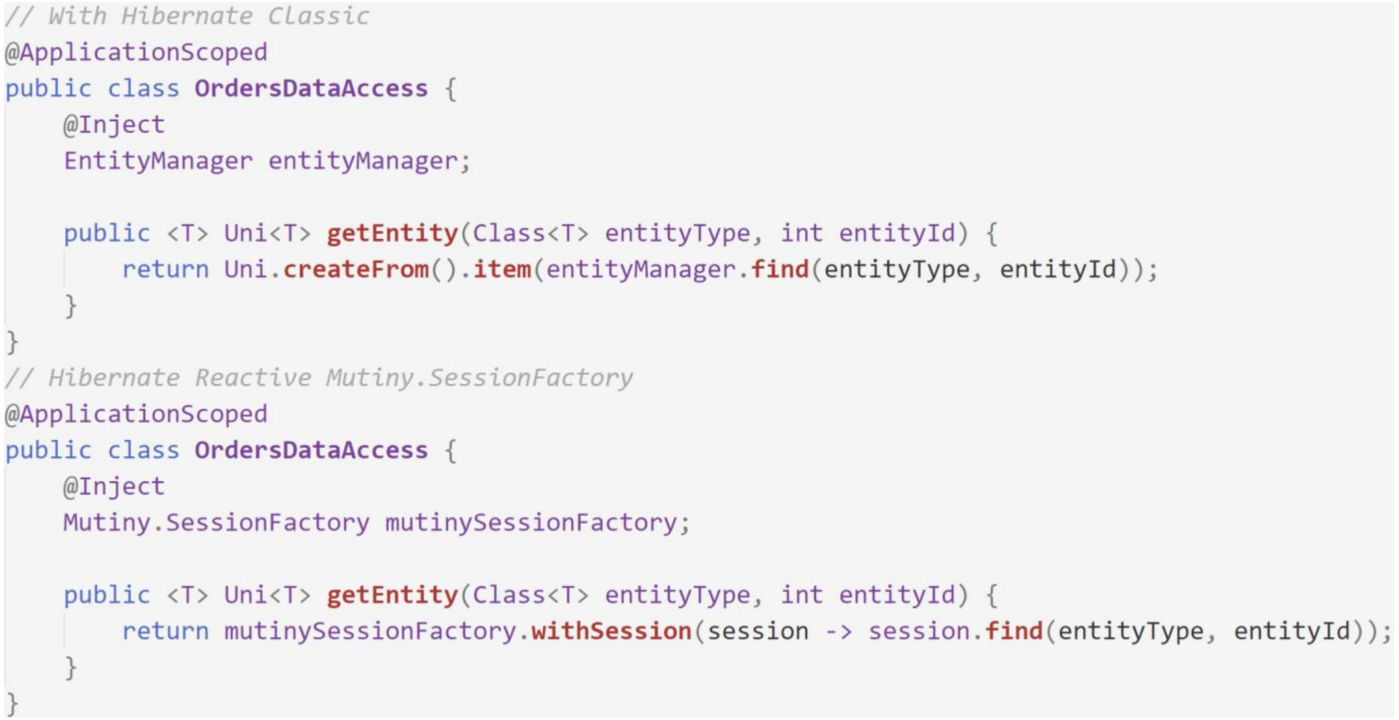
```java
// With Hibernate Classic
@ApplicationScoped
public class OrdersDataAccess {
    @Inject
    EntityManager entityManager;

    public <T> Uni<T> getEntity(Class<T> entityType, int entityId) {
        return Uni.createFrom().item(entityManager.find(entityType, entityId));
    }
}
```

Hibernate Reactive - Example



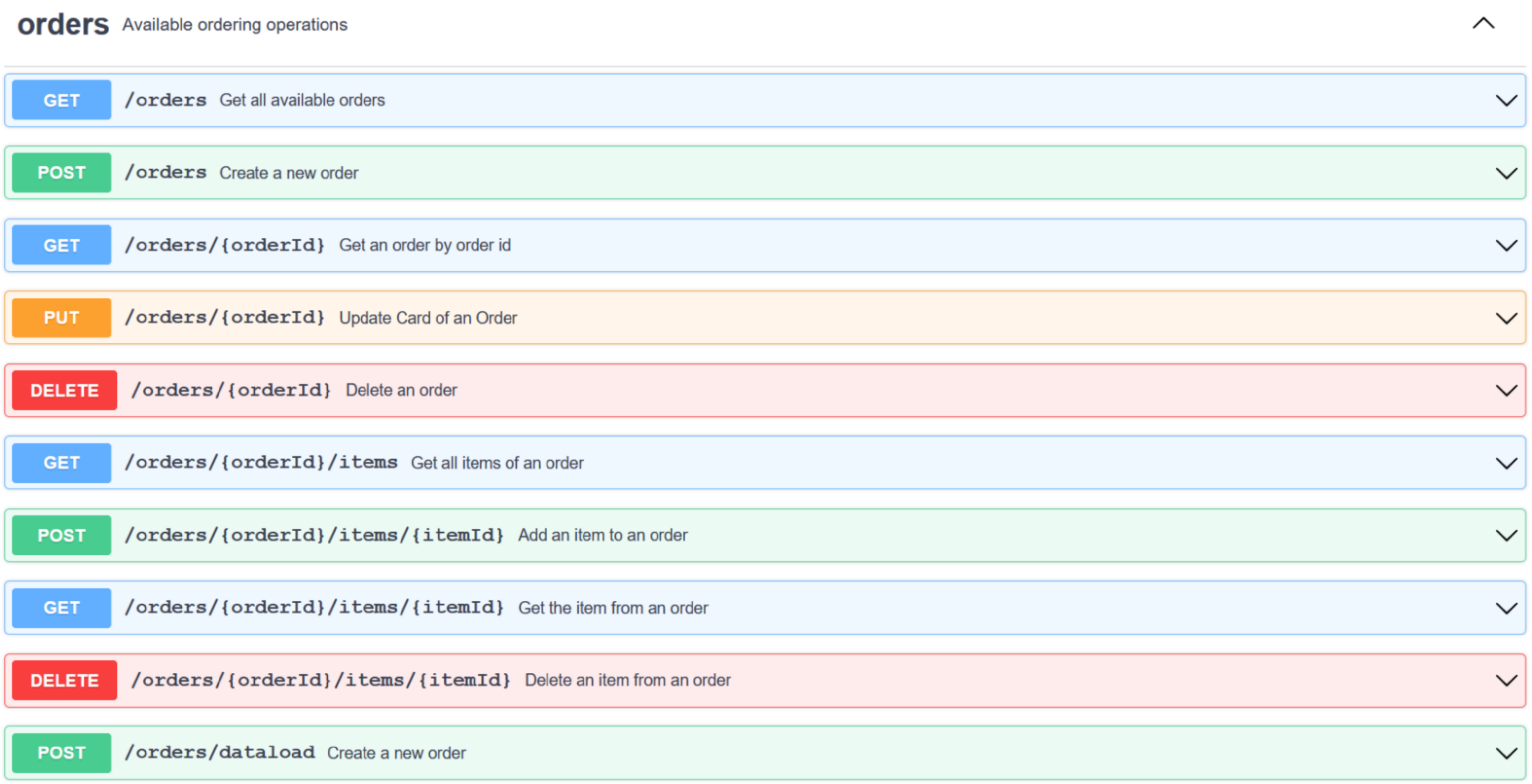
```java
// With Hibernate Classic
@ApplicationScoped
public class OrdersDataAccess {
    @Inject
    EntityManager entityManager;

    public <T> Uni<T> getEntity(Class<T> entityType, int entityId) {
        return Uni.createFrom().item(entityManager.find(entityType, entityId));
    }
}

// Hibernate Reactive Mutiny.Session
@ApplicationScoped
public class OrdersDataAccess {
    @Inject
    Mutiny.Session mutinySession;

    public <T> Uni<T> getEntity(Class<T> entityType, int entityId) {
        return mutinySession.find(entityType, entityId);
    }
}
```

Hibernate Reactive - Example



```java
// With Hibernate Classic
@ApplicationScoped
public class OrdersDataAccess {
    @Inject
    EntityManager entityManager;

    public <T> Uni<T> getEntity(Class<T> entityType, int entityId) {
        return Uni.createFrom().item(entityManager.find(entityType, entityId));
    }
}
// Hibernate Reactive Mutiny.SessionFactory
@ApplicationScoped
public class OrdersDataAccess {
    @Inject
    Mutiny.SessionFactory mutinySessionFactory;

    public <T> Uni<T> getEntity(Class<T> entityType, int entityId) {
        return mutinySessionFactory.withSession(session -> session.find(entityType, entityId));
    }
}
```

Software Experiments – Example REST-Service



orders Available ordering operations

GET `/orders` Get all available orders

POST `/orders` Create a new order

GET `/orders/{orderId}` Get an order by order id

PUT `/orders/{orderId}` Update Card of an Order

DELETE `/orders/{orderId}` Delete an order

GET `/orders/{orderId}/items` Get all items of an order

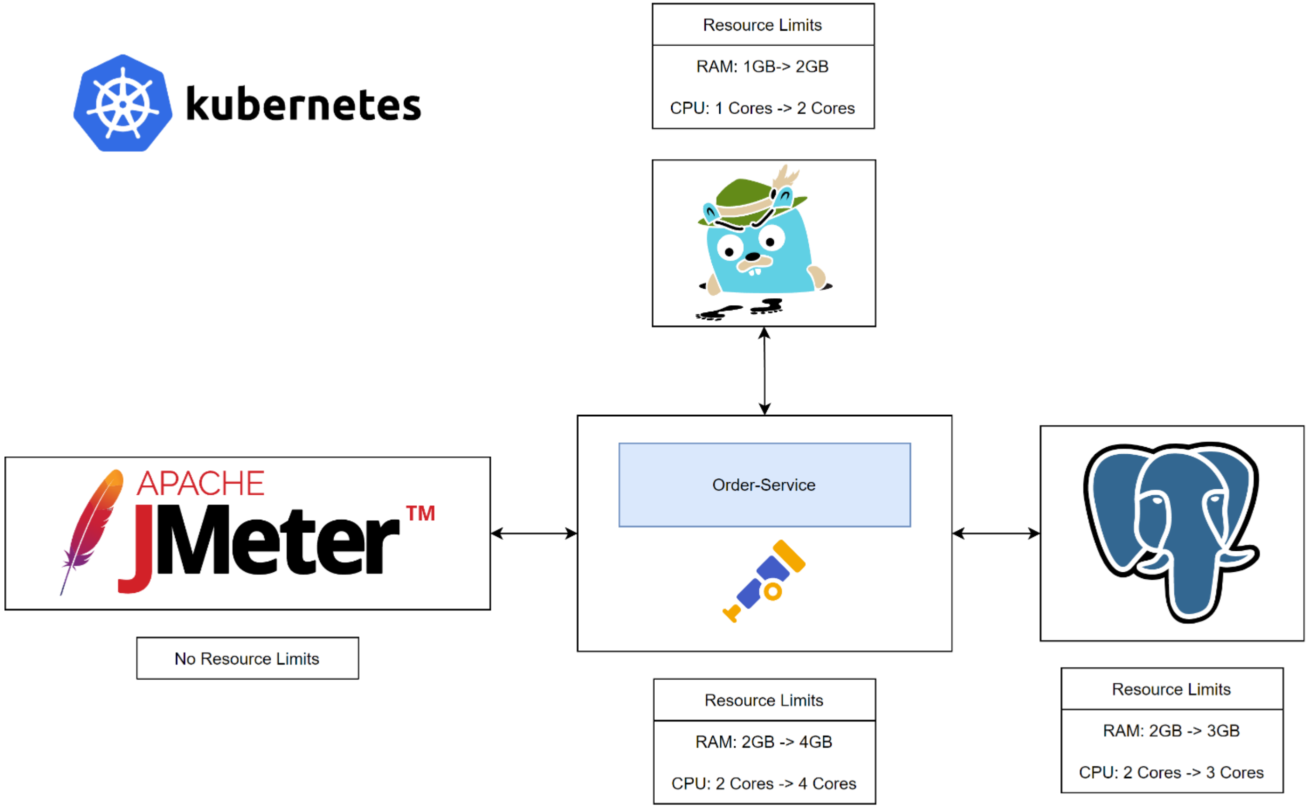
POST `/orders/{orderId}/items/{itemId}` Add an item to an order

GET `/orders/{orderId}/items/{itemId}` Get the item from an order

DELETE `/orders/{orderId}/items/{itemId}` Delete an item from an order

POST `/orders/dataload` Create a new order

Software Experiments - Setup



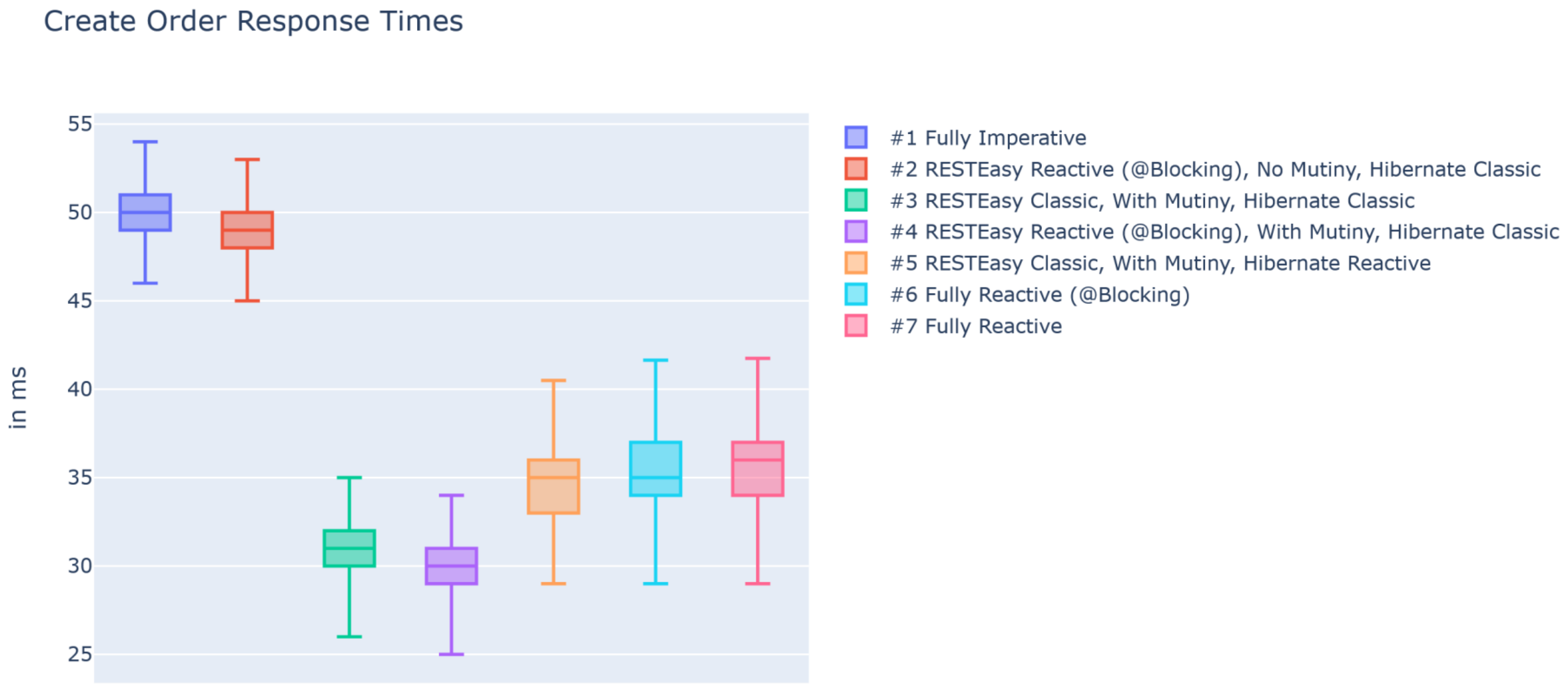
Software Experiments - Design

#	Use Case	Arrivals/minute	Request/arrival
1	Online Shop Order	300	2
2	Rescinding Orders	50	2
3	Changing Payment Information	50	2
4	Creating Orders with With Multiple Items	25	4
5	Order Creation With Unwanted Items	20	5

Software Experiments – Implementation Variants

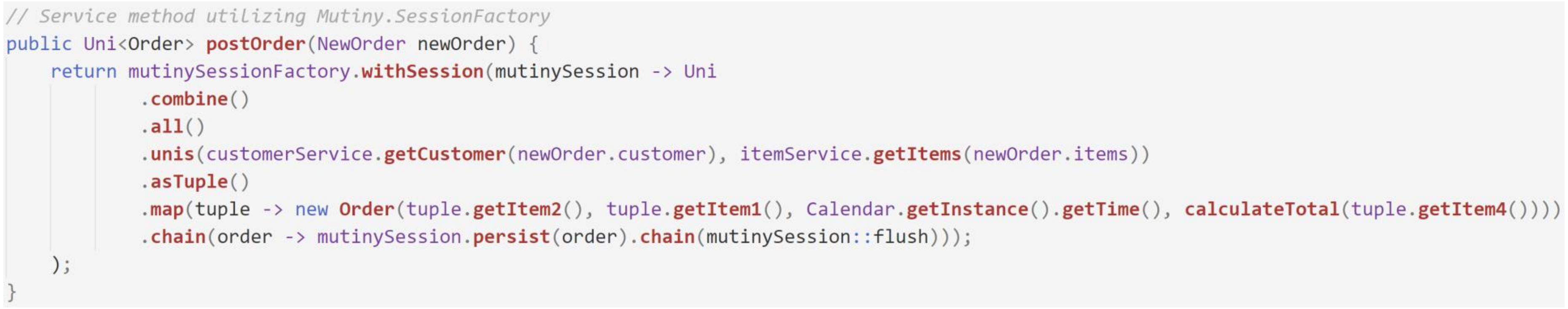
	RESTEasy (reactive)	Hibernate (reactive)	Mutiny	Notes
1				
2	X			@Blocking
3			X	
4	X		X	@Blocking
5		X	X	
6	X	X	X	@Blocking
7	X	X	X	

Software Experiments - Results



RETIT

Hibernate Reactive - Performance Implications



```java
// Service method utilizing Mutiny.SessionFactory
public Uni<Order> postOrder(NewOrder newOrder) {
    return mutinySessionFactory.withSession(mutinySession -> Uni
            .combine()
            .all()
            .unis(customerService.getCustomer(newOrder.customer), itemService.getItems(newOrder.items))
            .asTuple()
            .map(tuple -> new Order(tuple.getItem2(), tuple.getItem1(), Calendar.getInstance().getTime(), calculateTotal(tuple.getItem4())))
            .chain(order -> mutinySession.persist(order).chain(mutinySession::flush)));
    );
}
```

Conclusion

- RESTEasy Reactive @Blocking outperforms RESTEasy Classic
- Making the switch to RESTEasy Reactive can be simple if no reactive types are returned
- Performance gain with Mutiny can be substantial
- Transition to Hibernate Reactive can be challenging and not beneficial
- Wrapping DAOs with Mutiny more effective than Hibernate Reactive
- Easiest way to get started with Mutiny is to introduce it at entry-level

Thank you!

Denis Angeletta
angeletta@retit.de

RETIT

Resource Efficient Technologies & IT Systems