ORACLE CODE ONE

Break New Ground

San Francisco
September 16–19, 2019

Safe Harbor

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, timing, and pricing of any features or functionality described for Oracle's products may change and remains at the sole discretion of Oracle Corporation.

Statements in this presentation relating to Oracle's future plans, expectations, beliefs, intentions and prospects are "forward-looking statements" and are subject to material risks and uncertainties. A detailed discussion of these factors and other risks that affect our business is contained in Oracle's Securities and Exchange Commission (SEC) filings, including our most recent reports on Form 10-K and Form 10-Q under the heading "Risk Factors." These filings are available on the SEC's website or on Oracle's website at http://www.oracle.com/investor. All information in this presentation is current as of September 2019 and Oracle undertakes no duty to update any statement in light of new information or future events.

Oracle Code One 2019
Reactive Streams Processing with a New Java Library, Helidon, and Microservices [DEV4614]

Pablo Silberkasten, Software Development Manager, Oracle
Jean de Lavarene, Software Development Director, Oracle
Kuassi Mensah, Director, Product Management, Oracle

OJVM JDBC - Product Development
September 17, 2019

Program Agenda

- Challenges when Streaming Data into the Database.
- Introducing the Reactive Streams Ingestion library.
- API, Code Samples and Integration with Reactive Frameworks.
- Using the RSI library in Helidon.
- Creating a Microservice with Helidon-RSI.
- Deployment and Demo.

Challenges when Streaming Data into the DB

High Concurrency

Thousands of concurrent clients streaming data continuously and simultaneously to the same table/database.

Scalability

Target database being able to handle thousands of clients/connections competing for a limited amount of resources.

Responsiveness

Provide minimal response time, asynchronous processing, with non-blocking back-pressure.

Elasticity

Be able to allocate / release compute resources under a fluctuating workload.

One database might not be able to handle several thousands concurrent live connections without being strongly affected in performance.
Response time from a simple SQL Insert statement might not be acceptable for an IOT device.

Challenges when Streaming Data into the DB

Abstraction Layer

Transparently exploit (with no changes in the client) features of the Database.

Automatic Routing

Payload dependent: shard awareness / partitioning / RAC

High Availability

Disaster Recover, Planned Maintenance, Changes in DB topology.

Application Readiness

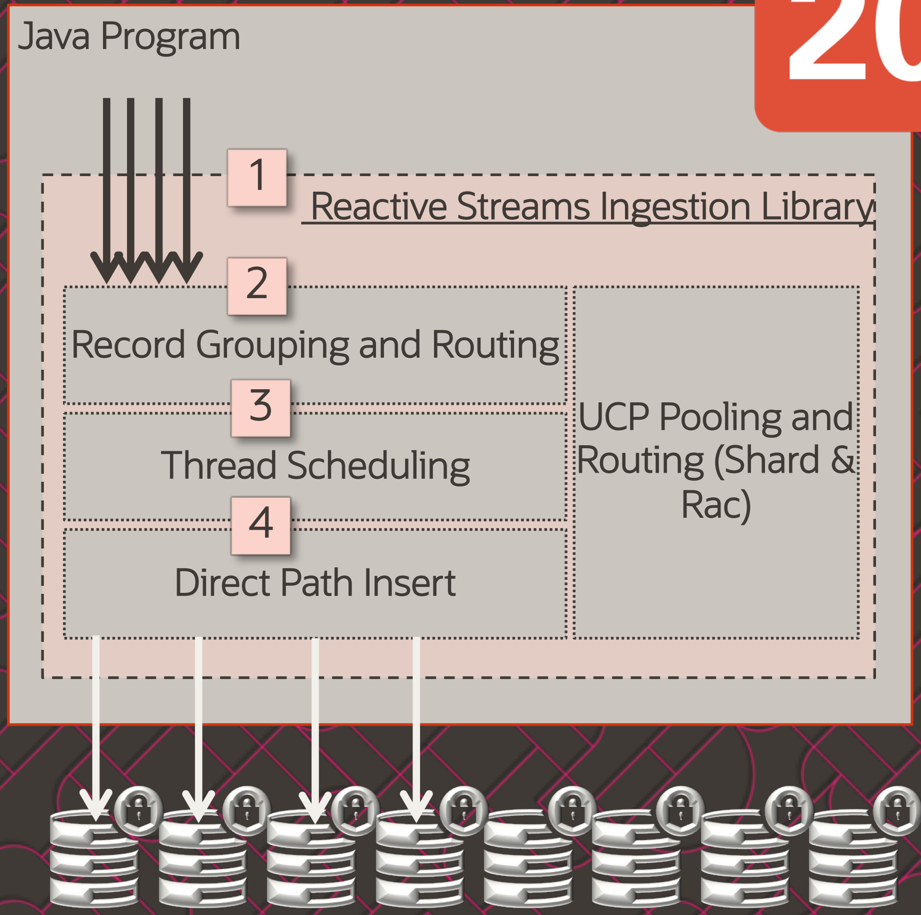
Exploit DB latest features with no impact on the Application layer.

Is very challenging to provide these features from a regular SQL insert without changes in the client (SQL, Code, Libraries, etc.).

Introducing the Reactive Streams Ingestion Library

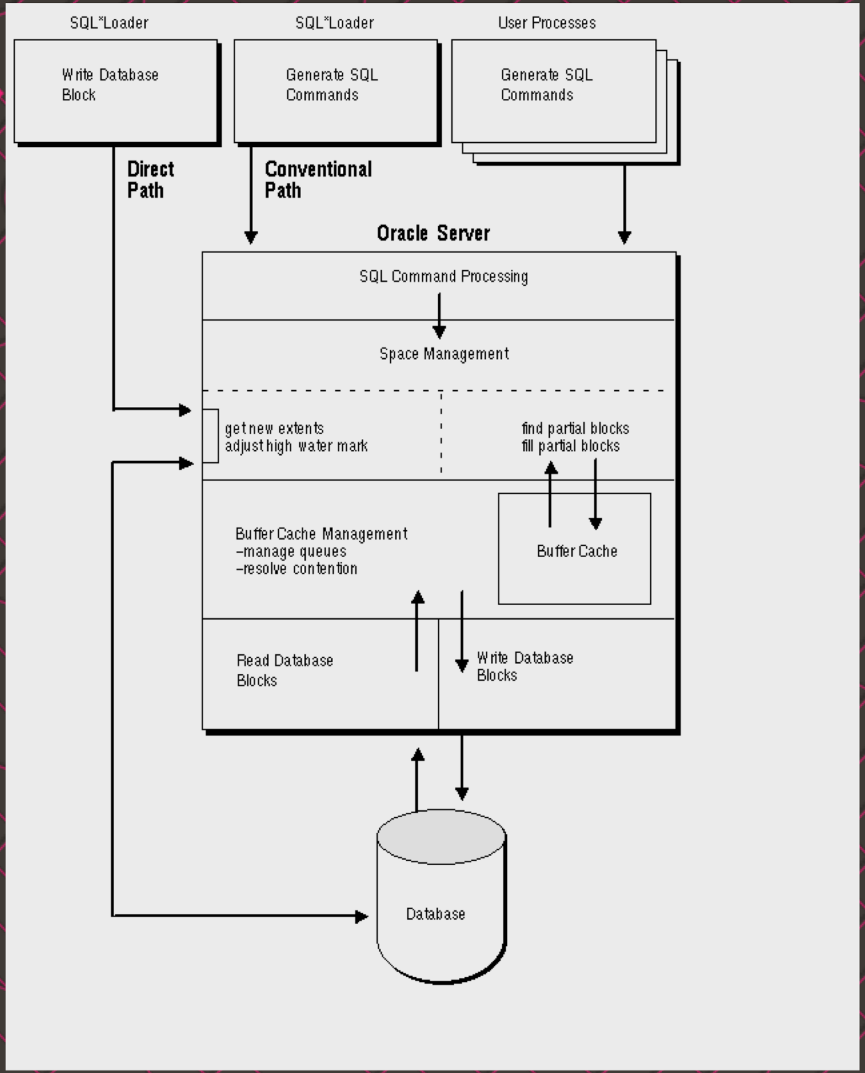
- 1) Streaming capability: unblockingly and reactively receive data from a large group of clients.
- 2) Group records through RAC and Shard affinity using native UCP.
- 3) Optimize CPU allocation while decoupling record processing from IO.
- 4) Fastest insert method for the Oracle Database through Direct Path Insert, bypassing SQL and writing directly in the Database files.

NEW IN 20c


Java Program
1
Reactive Streams Ingestion Library
2
Record Grouping and Routing
3
Thread Scheduling
4
Direct Path Insert
UCP Pooling and Routing (Shard & Rac)

Fastest insert method for the Oracle Database through Direct Path.

- In Direct Path the API creates data blocks in the Oracle Database block's format. These blocks are written directly in the Database files.
- There is no SQL command processing (no ”insert into…”), hence no contention with other processes using the Database.
- The Buffer Cache is bypassed, formatted data blocks are written directly in the Database files (also avoiding contention with other process).
- More options for performance: Parallelization, No-Log, Skip-Unusable-Indexes, Skip-Referential-Integrity.



Source: Oracle SQL Loader User's Guide

Introducing the Reactive Streams Ingestion library.
RAC and Shard awareness (routing capabilities).

Recognize sharding-keys specified by the users and allow them to connect to the specific shards and chunks (and connection re-usage).

Sharding-key ranges are cached, allowing to bypass the GSM (shard director).

Receive FAN Events from a RAC Cluster:

- Load Balancing Advisory.
- Node Down/Up.
- Service Down/Up.

Introducing the Reactive Streams Ingestion library.

Builder and Record API.

```
// Simple and intuitive constructor
ReactiveStreamsIngestion rsi =
    new ReactiveStreamsIngestion()
  .url(url)
  .schema("scott")
  .username("scott")
  .password("tiger")
  .executor(newScheduledThreadPool(2))
  .bufferRows(bufferRows)
  .bufferInterval(Duration.ofMillis(1000))
  .entity(Customer.class)
  .build();

// Record API (ORM Mapping) (.entity(class))
@Record
@Table(name = “customers”)
class Customer {
  @Column(name = “ID”) private long id;
  @Column(name = “NAME”) private String name;
  @Column(name = “REGION”) private String region;
}

// If no ORM Function<byte[], Record> or Object[]
.transformer(bytes -> new Customer(bytes))
.table("customers")
.columns(new String[] { "ID", "NAME", "REGION" })
```

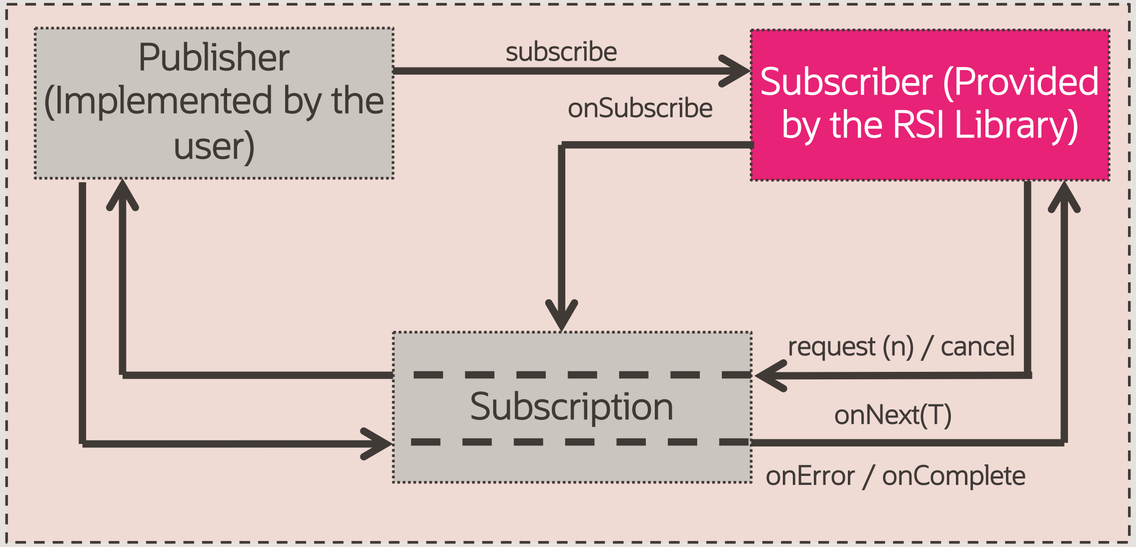
Introducing the Reactive Streams Ingestion library.

PushPublisher API.

```
// Push Publisher for Simple Usage
PushPublisher<Customer> pushPublisher =
  ReactiveStreamsIngestion.pushPublisher();
pushPublisher.subscribe(rsi.subscriber());

// Ad-hoc usage
pushPublisher.accept(
    new Customer(1, "John Doe", "North"));

// As a Consumer of a Stream
Customer[] customers = …
Stream
    .of(customers)
    .filter(c -> c.region.equals("NORTH"))
    .forEach(pushPublisher::accept);

// As a Consumer for 3rd party Reactive Stream
// Libraries eg: Reactor https://projectreactor.io/
Flux
  .just(
      new Customer(1, "John Doe", "North"),
      new Customer(2, "Jane Smith", "South"))
  .concatWithValues(new Customer(3, "Bob", "South"))
  .doOnComplete(() -> {System.out.println("Done!");})
  .doOnCancel(() -> {System.out.println("Canceled!");})
  .subscribe(pushPublisher::accept);
```

Introducing the Reactive Streams Ingestion library.

FlowPublisher API. Allowing Ad-Hoc Publishers.

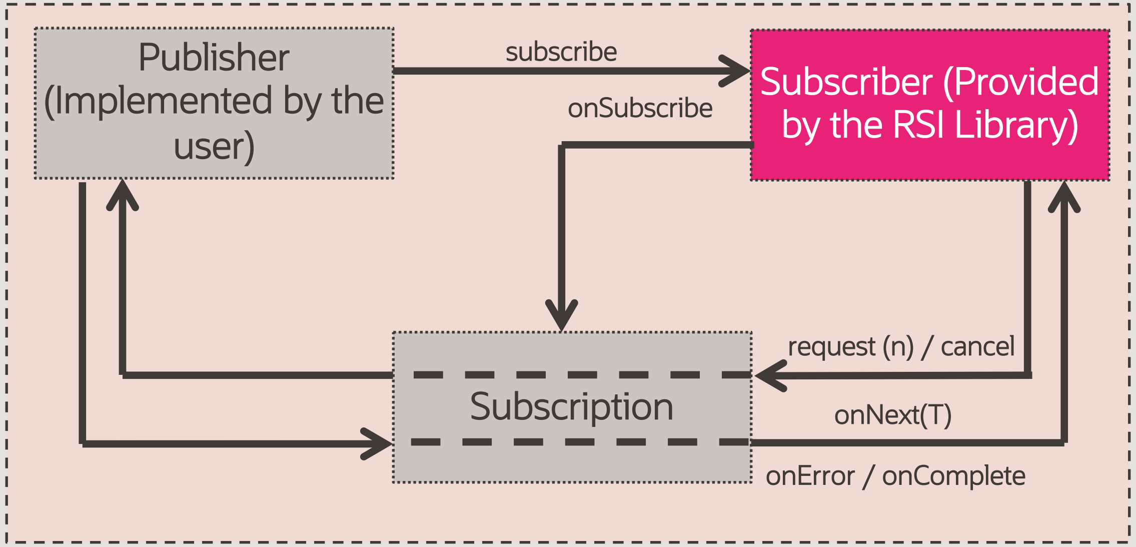


```
class SimpleCustomerPublisher<T>
    implements Publisher<T>, Consumer<T> {

  Subscriber<? super T> subscriber; // reference to lib
  Subscription subscription =
    new SimpleCustomerSubscription(); // get feedback

  @Override
  public void subscribe(Subscriber<? super T>
      subscriber) {
    this.subscriber = subscriber; // associate
    this.subscriber.onSubscribe(subscription); // call
  }

  @Override
  public void accept(T t) { // being a consumer
    subscriber.onNext(t); // send to the library
  }
}
```

Source: Grokonoz

Introducing the Reactive Streams Ingestion library.

FlowPublisher API.
Allowing Ad-Hoc Publishers.

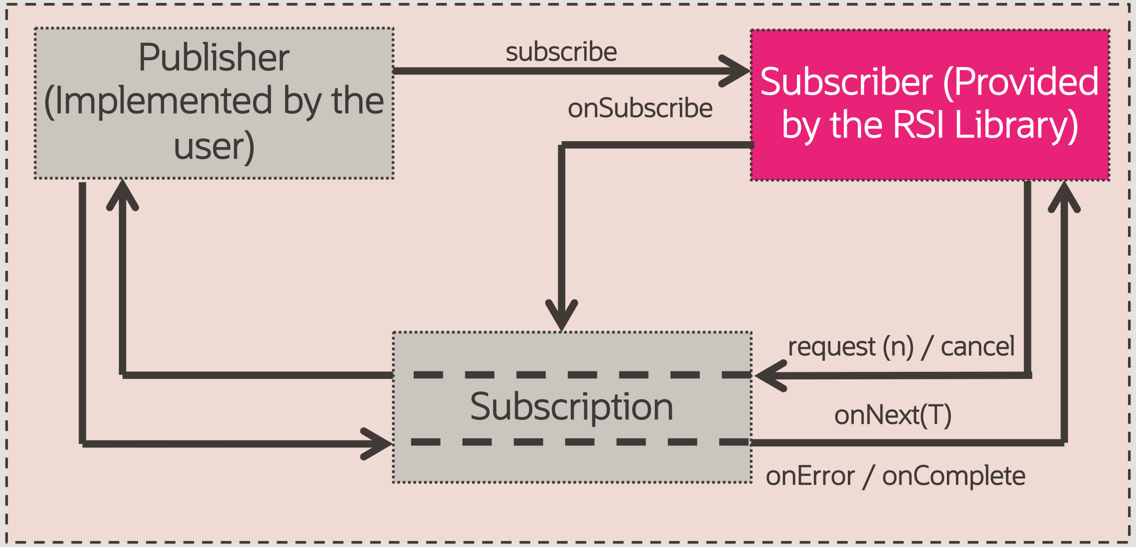


```
class SimpleCustomerSubscription
    implements Subscription {
  @Override
  // Signal unfulfilled demand
  public void request(long n) {
    log("Requesting: " + n + " records");
  }
  // Signal Stop
  @Override
  public void cancel() { . . .

SimpleCustomerPublisher<Customer> publisher =
    new SimpleCustomerPublisher<Customer>();

publisher.subscribe(rsi.subscriber());

publisher.accept(
    new Customer(1, "John Doe", "North")); //Ingestion
```

Source: Grokonoz

Introducing the Reactive Streams Ingestion library.

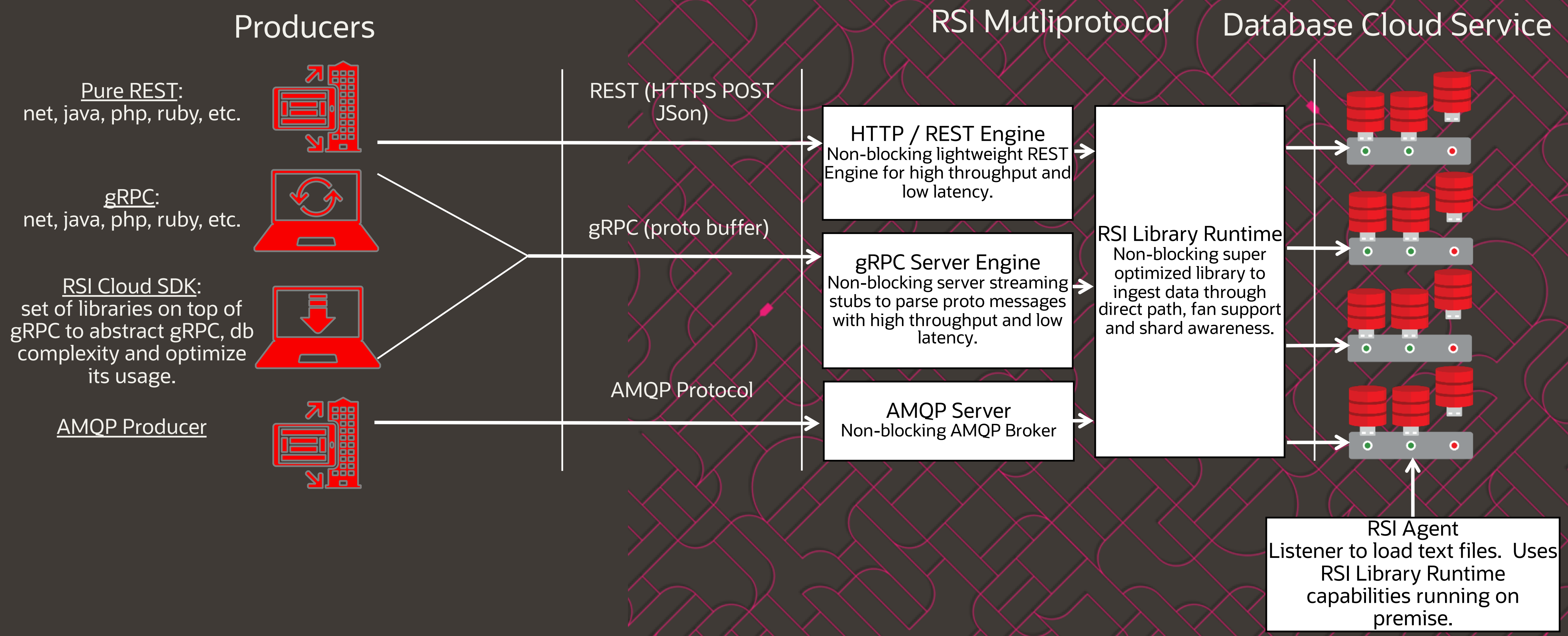
Using SubmissionPublisher from JDK Standard.



```
// Out of the box from the JDK
// java.util.concurrent.SubmissionPublisher
// Implementation than handles back-pressure

SubmissionPublisher<Customer> subpub =
     new SubmissionPublisher<>();

subpub.subscribe(rsi.subscriber());

subpub.submit(new Customer(1, "John Doe", "North"));
subpub.submit(new Customer(2, "Jane Doe", "North"));
subpub.submit(new Customer(3, "John Smith", "South"));

while (subpub.estimateMaximumLag() > 0);
```

Source: Grokonoz

Reactive Streams Ingestion Multiprotocol * Not a Product



Producers

Pure REST:
net, java, php, ruby, etc.

gRPC:
net, java, php, ruby, etc.

RSI Cloud SDK:
set of libraries on top of gRPC to abstract gRPC, db complexity and optimize its usage.

AMQP Producer

REST (HTTPS POST JSon)

gRPC (proto buffer)

AMQP Protocol

RSI Mutliprotocol

HTTP / REST Engine
Non-blocking lightweight REST Engine for high throughput and low latency.

gRPC Server Engine
Non-blocking server streaming stubs to parse proto messages with high throughput and low latency.

AMQP Server
Non-blocking AMQP Broker

RSI Library Runtime
Non-blocking super optimized library to ingest data through direct path, fan support and shard awareness.

Database Cloud Service

RSI Agent
Listener to load text files. Uses RSI Library Runtime capabilities running on premise.

Reactive Streams Ingestion Multiprotocol

Differentiators Features

Singled purposed, built to scale and perform.

Performance

10x better performance than similar tools or ad-hoc solutions.

Responsiveness

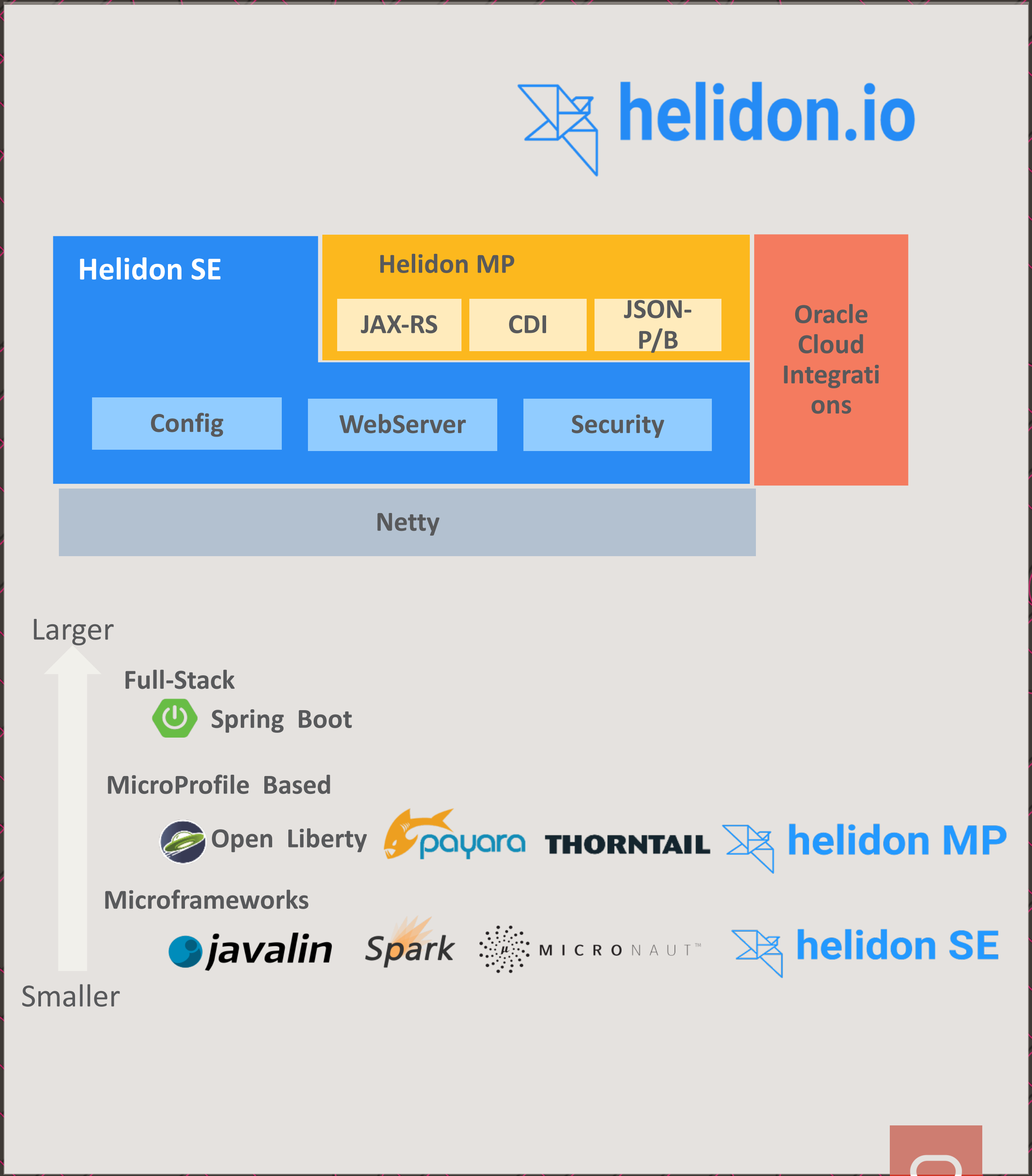
Standalone non-blocking reactive web server and internal libraries.

Multiprotocol

REST
gRPC
AMQP
Cloud SDK

Project Helidon

- A set of open source Java libraries for developing microservices.
- Your service is just a Java SE application.
- Implements Eclipse MicroProfile (Helidon MP).
- Two programming models:
  - Helidon MP: declarative style, familiar to Java EE developers (JAX-RS, CDI, etc).
  - Helidon SE: functional style, transparent, reactive.
- Built-in integrations to Oracle Cloud Services.



helidon.io

Helidon SE

Helidon MP

JAX-RS

CDI

JSON-P/B

Oracle Cloud Integrations

Config

WebServer

Security

Netty

Larger

Full-Stack

Spring Boot

MicroProfile Based

Open Liberty

payara

THORNTAIL

helidon MP

Microframeworks

javalin

Spark

MICRONAUT™

helidon SE

Smaller

RSI Multiprotocol in OCI

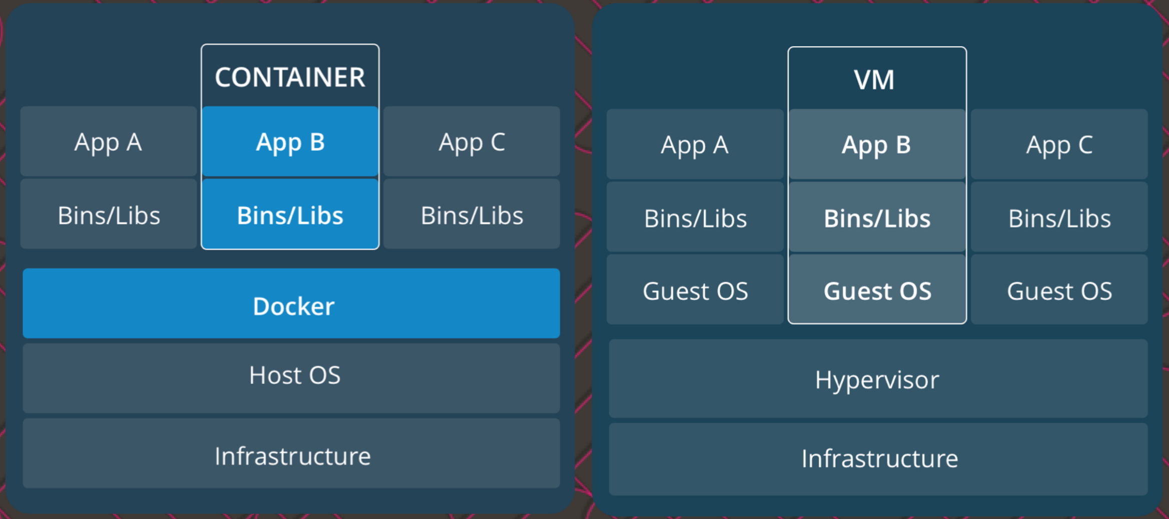
Containers

Next-generation virtualization

- Lighter weight
- More efficient
- Faster to spin up / down

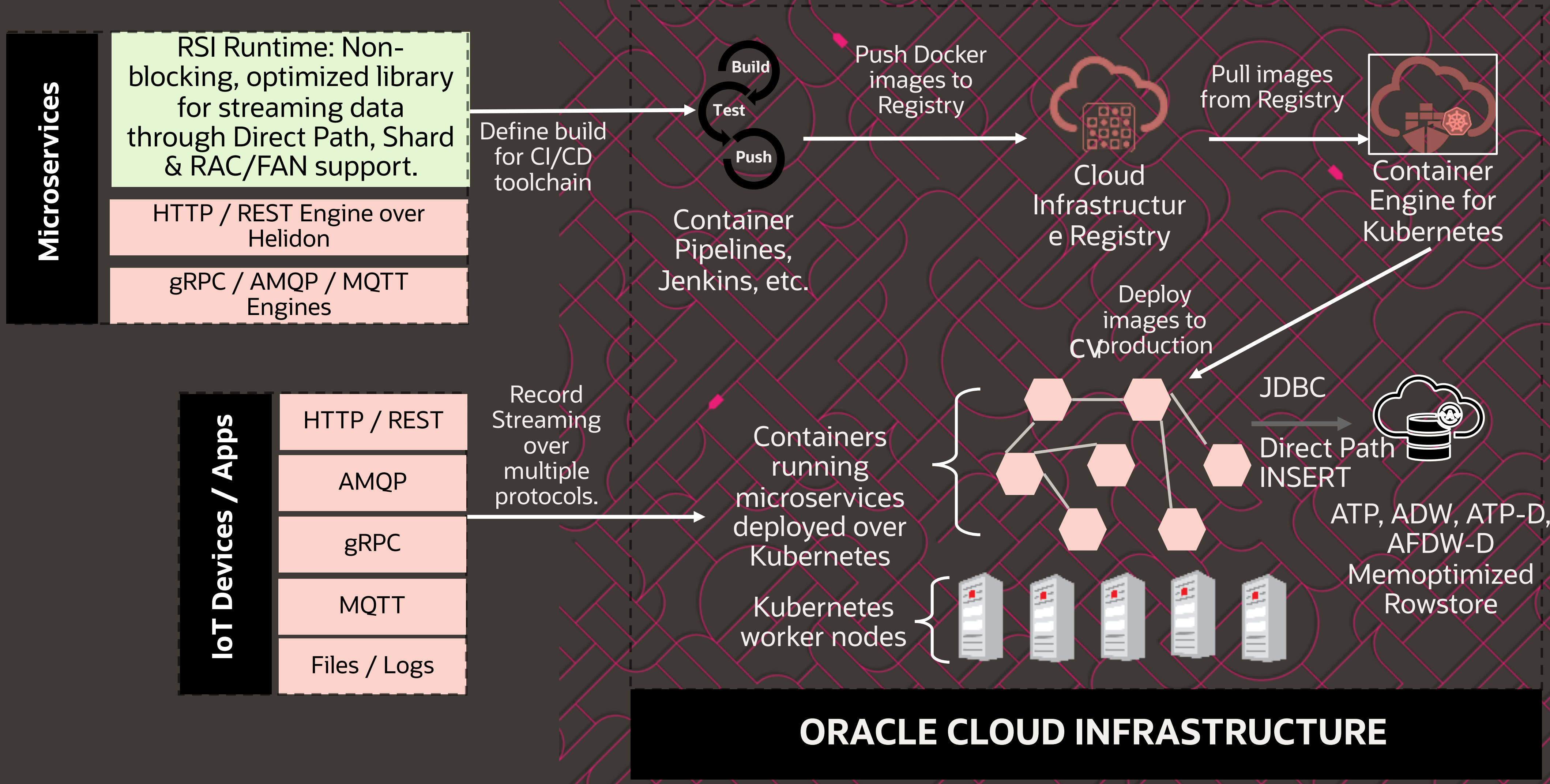
Docker is the de-facto technology to create, manage and run containers

Kubernetes is the leading solution for container orchestration



A **container** runs *natively* on Linux and shares the kernel of the host machine with other containers. It runs a discrete process, taking no more memory than any other executable, making it lightweight.
By contrast, a **virtual machine** (VM) runs a full-blown "guest" operating system with *virtual* access to host resources through a hypervisor. In general, VMs provide an environment with more resources than most applications need.

RSI Multiprotocol in OCI



Microservices

RSI Runtime: Non-blocking, optimized library for streaming data through Direct Path, Shard & RAC/FAN support.

HTTP / REST Engine over Helidon

gRPC / AMQP / MQTT Engines

Define build for CI/CD toolchain

Build

Test

Push

Container Pipelines, Jenkins, etc.

Push Docker images to Registry

Cloud Infrastructure Registry

Pull images from Registry

Container Engine for Kubernetes

Deploy images to CVproduction

JDBC

Direct Path INSERT

ATP, ADW, ATP-D, AFDW-D Memoptimized Rowstore

IoT Devices / Apps

HTTP / REST

AMQP

gRPC

MQTT

Files / Logs

Record Streaming over multiple protocols.

Containers running microservices deployed over Kubernetes

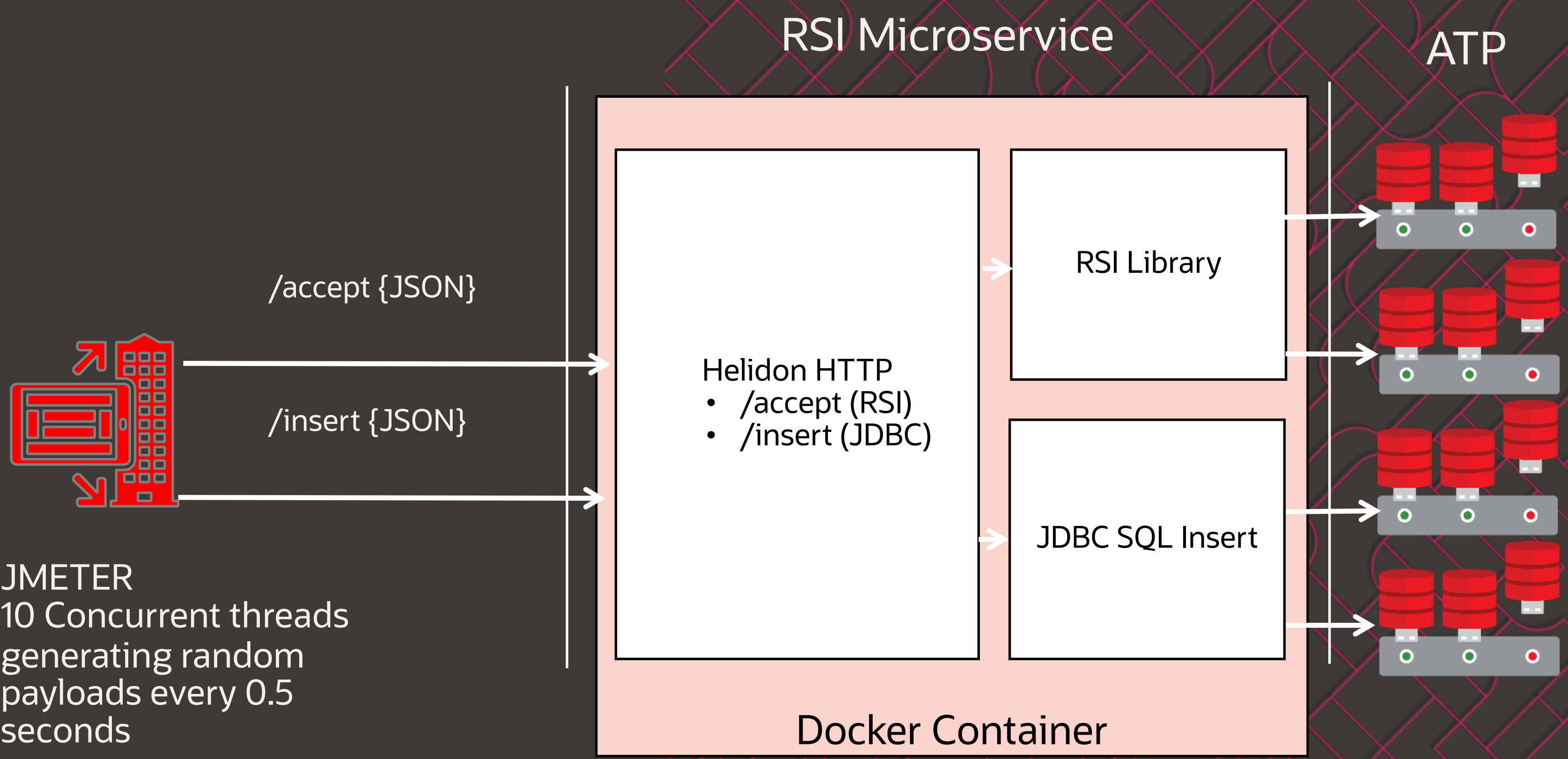
Kubernetes worker nodes

ORACLE CLOUD INFRASTRUCTURE

Helidon-RSI Code and Demo!

- Demo with Helidon SE.
- Demo with Helidon MP.
- Docker Image Build and Deploy.
- Test Performance.

Concurrency Demo with HTTP



RSI Microservice

ATP

/accept {JSON}

/insert {JSON}

JMETER
10 Concurrent threads generating random payloads every 0.5 seconds

Helidon HTTP
- /accept (RSI)
- /insert (JDBC)

RSI Library

JDBC SQL Insert

Docker Container

Thank You!
We Are Hiring!

Pablo Silberkasten
pablo.silberkasten@oracle.com
https://github.com/psilberk
+1 650 544 3995

Session Survey

Help us make the content even better. Please complete the session survey in the Mobile App.

Safe Harbor

The preceding is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, timing, and pricing of any features or functionality described for Oracle's products may change and remains at the sole discretion of Oracle Corporation.

Statements in this presentation relating to Oracle's future plans, expectations, beliefs, intentions and prospects are "forward-looking statements" and are subject to material risks and uncertainties. A detailed discussion of these factors and other risks that affect our business is contained in Oracle's Securities and Exchange Commission (SEC) filings, including our most recent reports on Form 10-K and Form 10-Q under the heading "Risk Factors." These filings are available on the SEC's website or on Oracle's website at http://www.oracle.com/investor. All information in this presentation is current as of September 2019 and Oracle undertakes no duty to update any statement in light of new information or future events.