# Oracle Multilingual Engine (MLE) –

## Java, JavaScript, Python or PL/SQL in the Database

**Philipp Salvisberg**


@phsalvisberg

# About Me

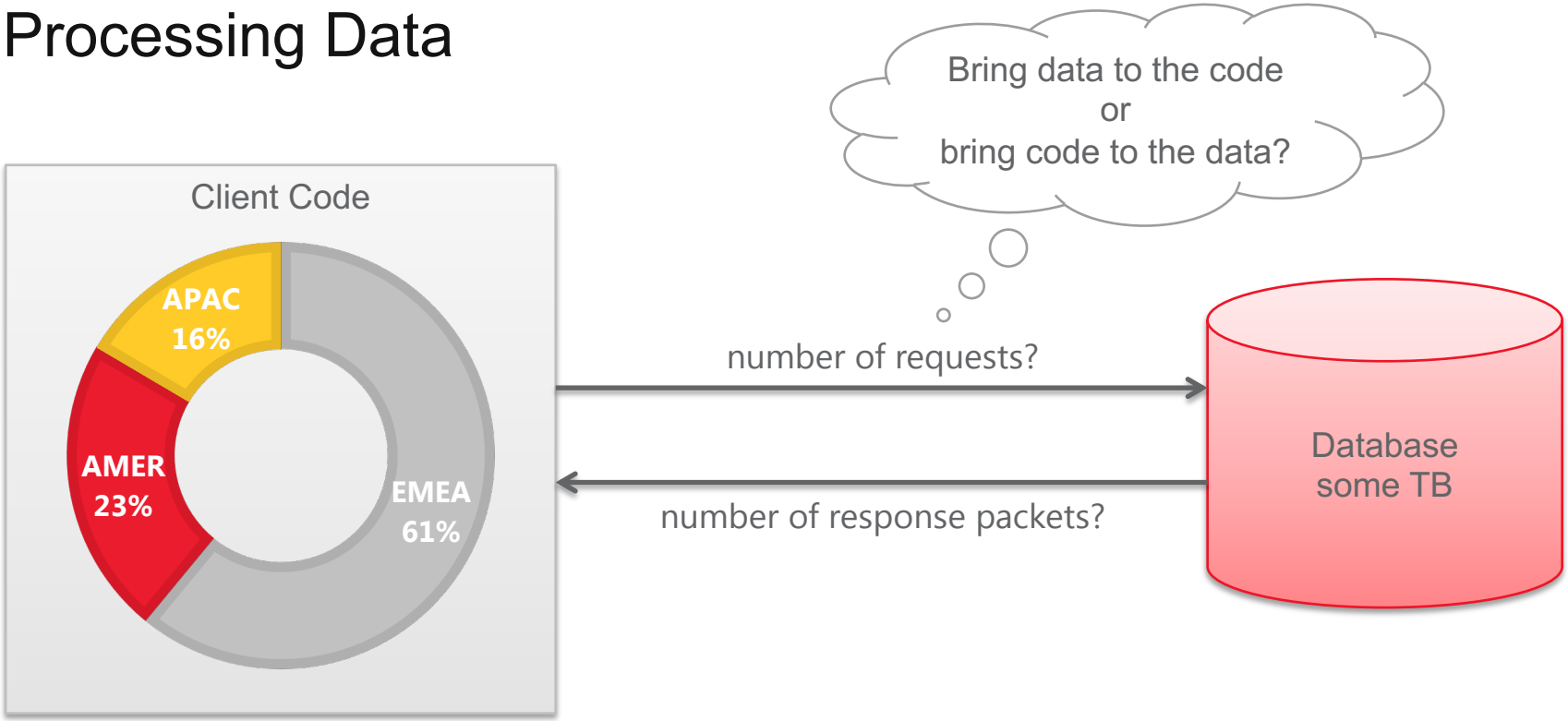

- Trivadian since April 2000
  - Senior Principal Consultant, Partner
  - Member of the Board of Directors
  - @phsalvisberg
  - https://www.salvis.com/blog
  - https://github.com/PhilippSalvisberg
- Database centric development with Oracle database
- Model Driven Software Development
- Author of free SQL Developer Extensions PL/SQL Unwrapper, PL/SQL Cop, utPLSQL, plscope-utils, oddgen and Bitemp Remodeler

# Agenda

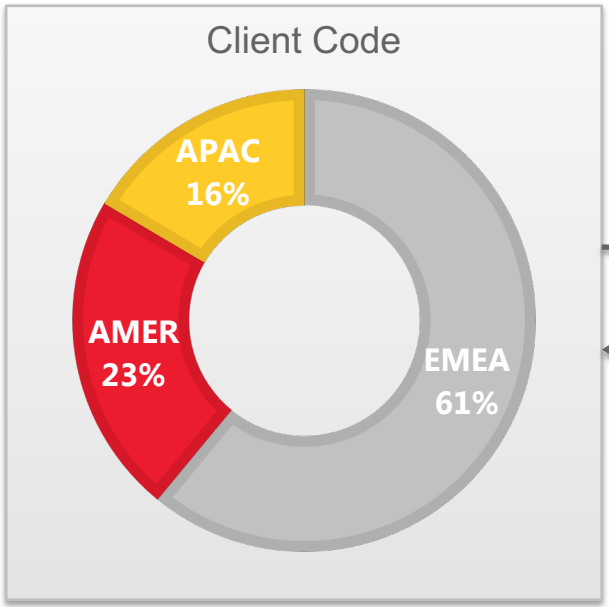
1. Code in the Database
2. GraalVM
3. Version 12c (Experimental)
4. Version 20c (Sensible Guess)
5. Version 30c (Highly Speculative)
6. Core Messages

# Code in the Database

# Processing Data



# Bring Code to the Data




Minimize communication for good performance

one request

one response packet

Database some TB

```
SELECT region, sum(sales)
FROM ... GROUP BY region
```

# Languages in Oracle Database 18c

**Connection is enough**

- SQL
- PL/SQL
- XSLT
- XQuery

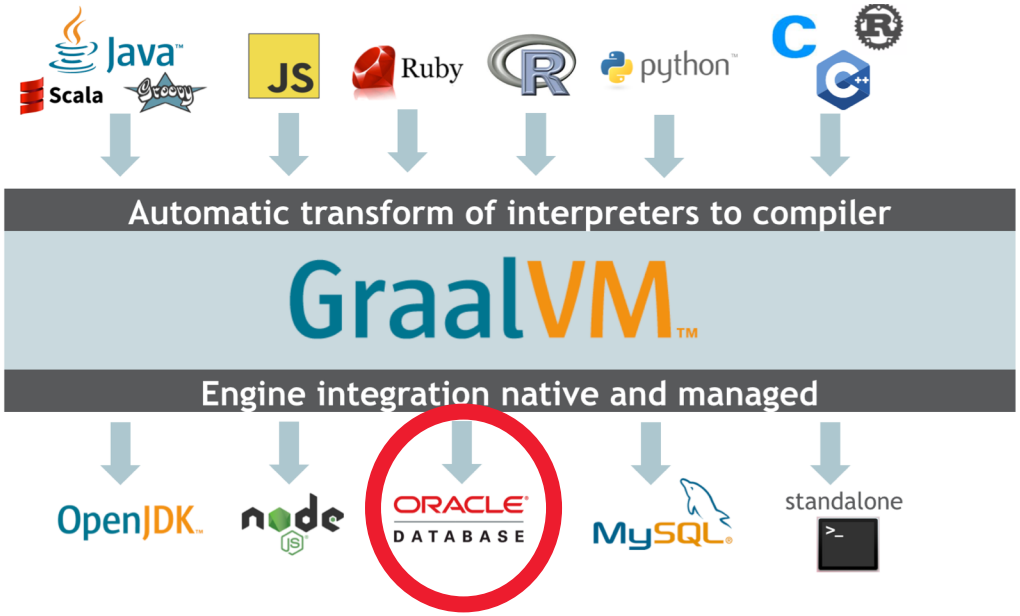
**Needs a PL/SQL wrapper**

- C
  - Requires command line access on the DB server
- Java
  - Requires access to loadjava
  - Simple Java sources without additional dependencies can be installed with SQL

# GraalVM

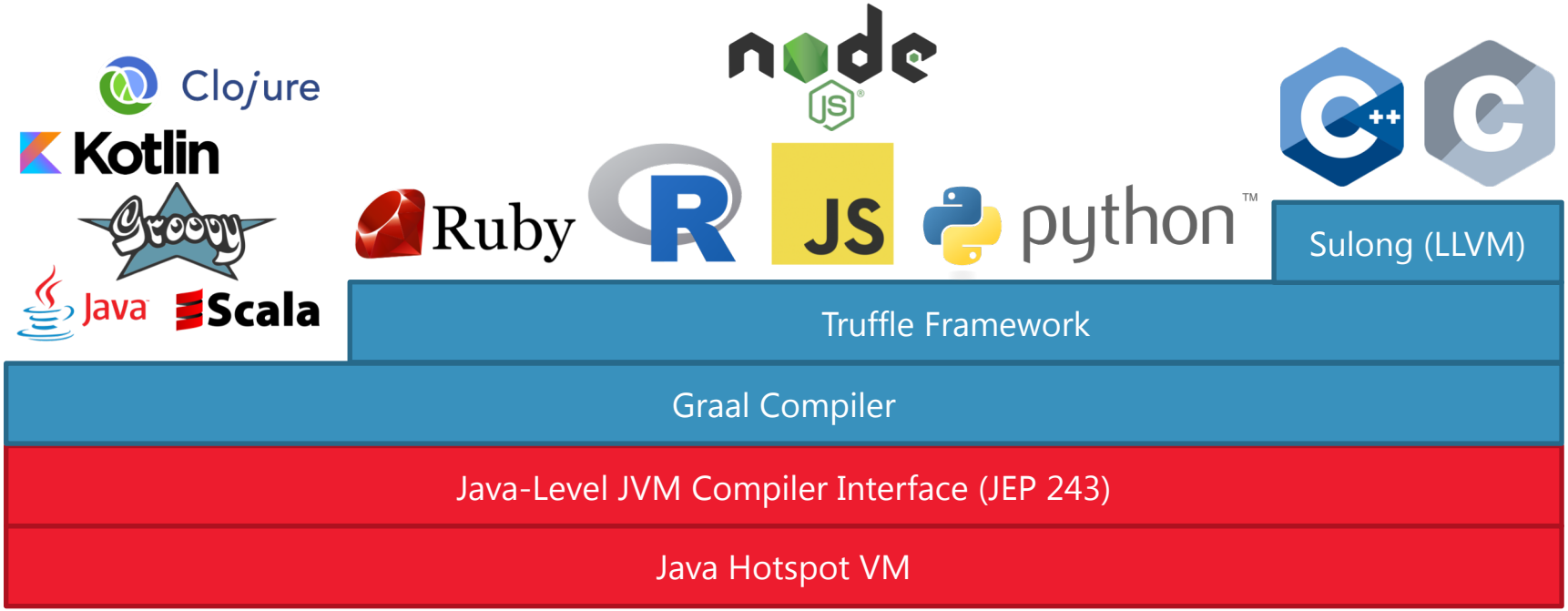
# Properties

- **Polyglot**
  - Languages
- **Embeddable**
  - Databases
  - HTTP-Server
- **Efficient**
  - Standalone
  - Interoperability (shared memory)

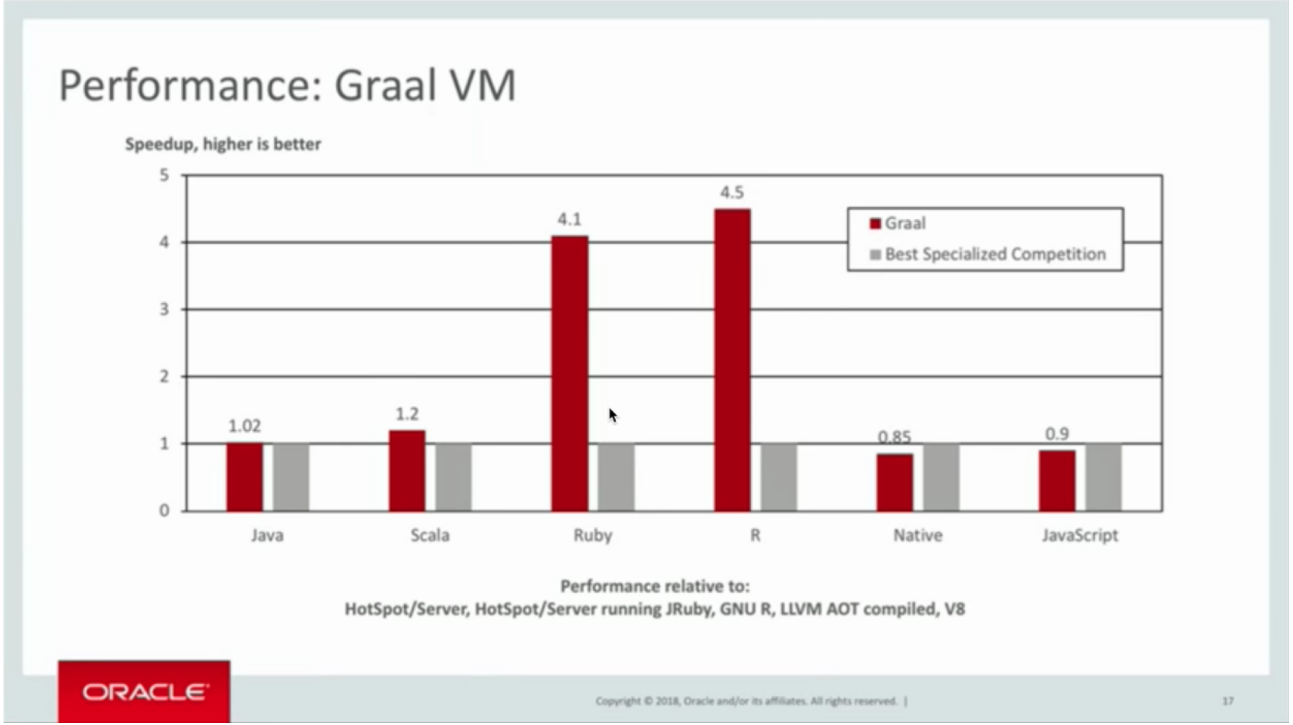



Source: https://www.graalvm.org/docs/

# Architecture



# Performance




Source: https://youtu.be/8AYESZIaacg?t=1002

Graal: High-Performance Polyglot Runtime by Thomas Wuerthinger and Aleksandar Prokopec, Voxxed Days, Zürich, March 2018

# Open Source

## Community Edition (CE)

GraalVM CE is available for free for development and production use. It is built from the GraalVM sources available on GitHub. We provide pre-built binaries for GraalVM CE for Linux and Mac OS X on x86 64-bit systems.

DOWNLOAD FROM GITHUB

## Enterprise Edition (EE)

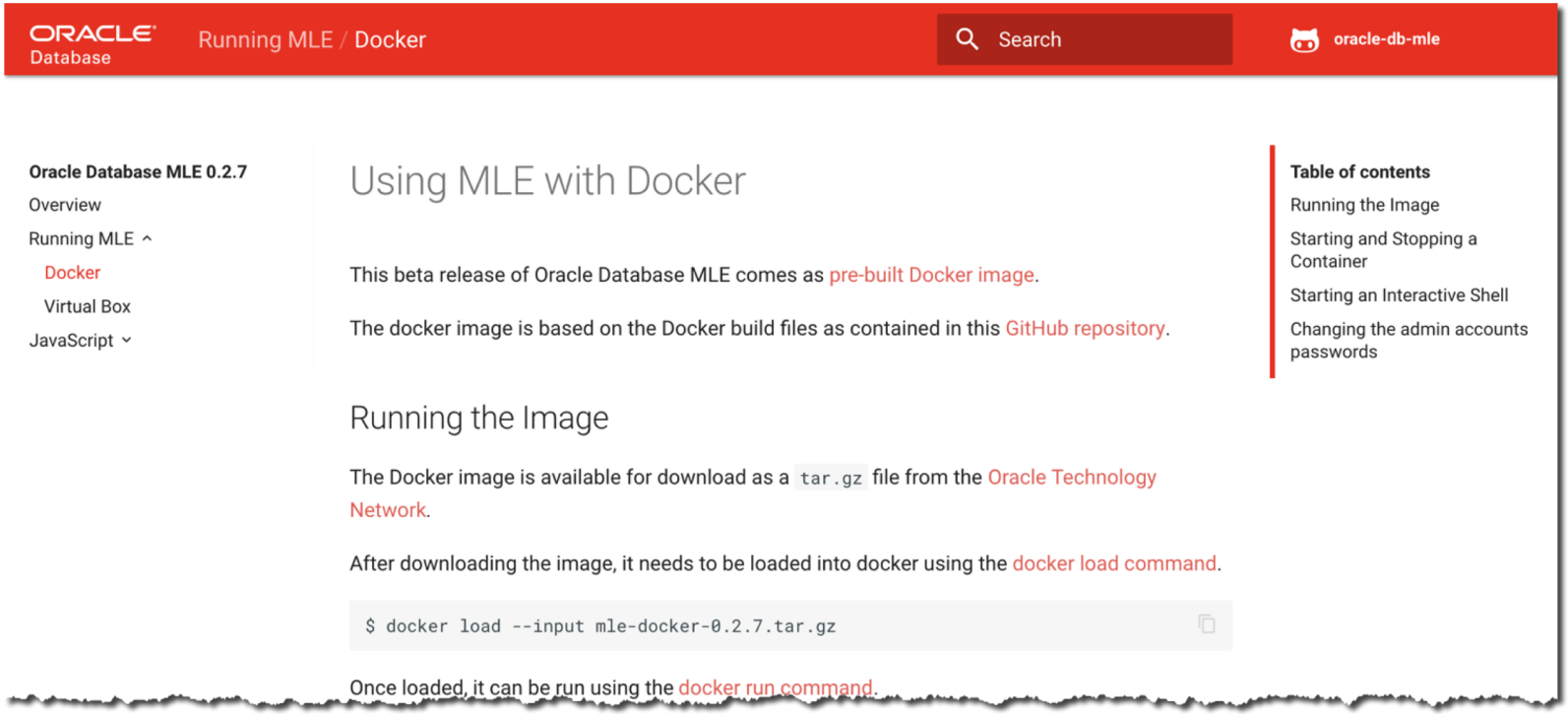
GraalVM EE provides additional performance, security, and scalability relevant for running critical applications in production. It is free for evaluation uses and available for download from the Oracle Technology Network. We provide binaries for GraalVM EE for Linux or Mac OS X on x86 64-bit systems.

DOWNLOAD FROM OTN

Source: https://www.graalvm.org/downloads/

# Version 12c (Experimental)

# Documentation




Source: https://oracle.github.io/oracle-db-mle/docker/

# Create Docker Container

Load image from downloaded archive file

```
docker import mle-docker-0.2.7.tar.gz
```

Create container

```
docker run --privileged --name mle \
  -p 1581:1521 -p 5081:5500 \
  -e ORACLE_SID=mle \
  -e ORACLE_PWD=oracle \
  -v /Users/phs/docker/mle/oradata:/opt/oracle/oradata \
  -v /Users/phs/docker/mle/myproject:/home/oracle/myproject \
  mle-docker-0.2.7
```

# Install Optional, Global Packages

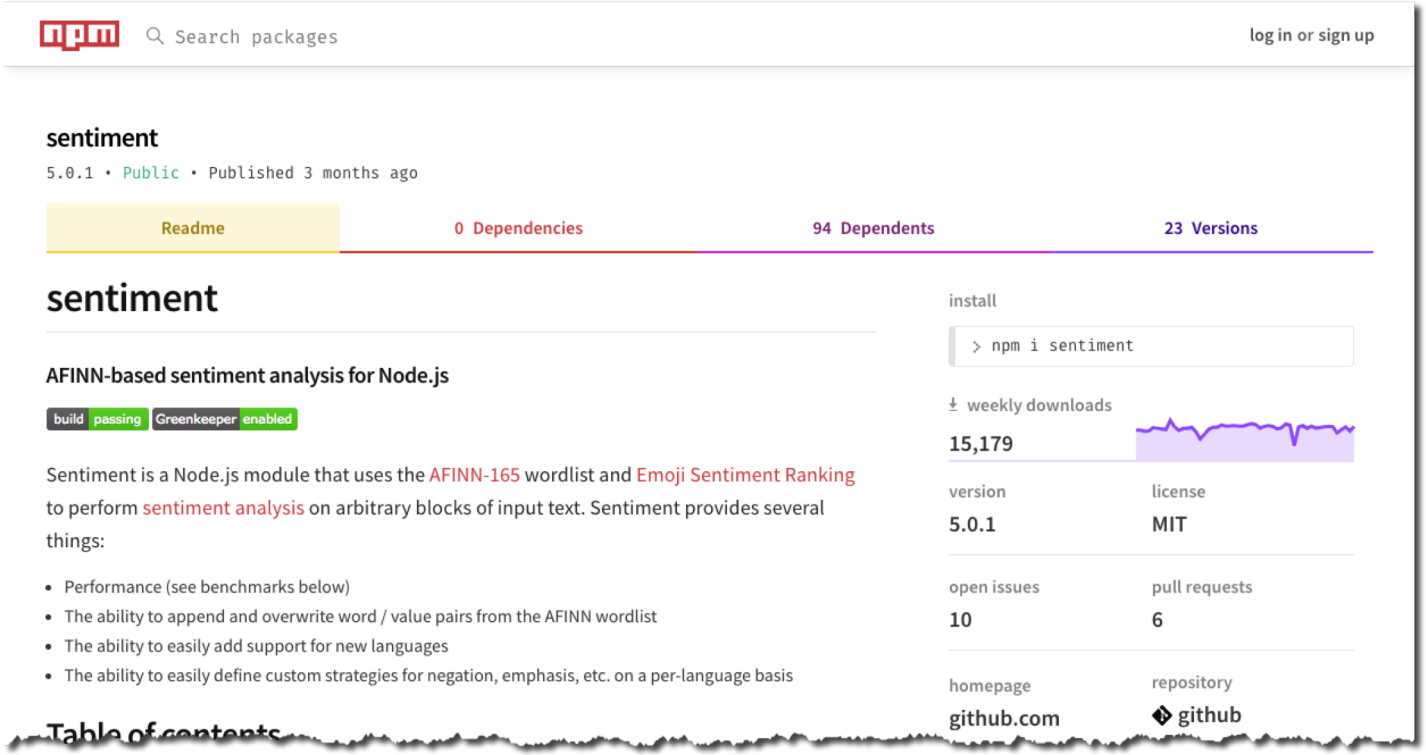
Open a shell in within the MLE docker container as ”root”

```
docker exec -u root --privileged -it mle bash
```

Install TypeScript compiler incl. Node.js support and dependencies-bundler browserify

```
npm install -g typescript
npm install -g ts-node
npm install -g browserify
```

# npm – Package Manager for JavaScript




Source: https://www.npmjs.com/package/sentiment

# Install “sentiment” Package

Open a shell in within the MLE docker container

```
docker exec -it mle bash
```

Install package in your project directory

```
cd myproject; mkdir -p demo; cd demo
npm install sentiment
```

# TypeScript Wrapper


sentiments.ts

```
import Sentiment = require('sentiment');

export function analyze(data: string, spaces: number = 0): string {
    var sentiment = new Sentiment();
    var result = sentiment.analyze(data);
    return JSON.stringify(result, null, spaces);
}
```

# Positive Sentiment – Test via Node.js

```
ts-node -p 'import {analyze} from "./sentiments";
   analyze("I like it. Works perfectly, awesome!", 3)'
```

```
{
   "score": 9,
   "comparative": 1.5,
   "tokens": [
      "i",
      "like",
      "it",
      "works",
      "perfectly",
      "awesome"
   ],
…
```

```
…
   "words": [
      "awesome",
      "perfectly",
      "like"
   ],
   "positive": [
      "awesome",
      "perfectly",
      "like"
   ],
   "negative": []
}
```

# Deploy Into Database

Use “dbjs” command line interface

```
dbjs deploy sentiments.ts -u demo -p demo -c localhost:1521/mle
```

- Creates JavaScript module “sentiments.js” (single file as result of browserify)
- Creates library "sentiments.js"
- Creates PL/SQL package specification ”SENTIMENTS”
  - FUNCTION ANALYZE("p0" IN VARCHAR2) RETURN VARCHAR2 AS LANGUAGE JS LIBRARY "sentiments.js" NAME "analyze" PARAMETERS("p0" STRING);
  - FUNCTION ANALYZE("p0" IN VARCHAR2, "p1" IN NUMBER) RETURN VARCHAR2 AS LANGUAGE JS LIBRARY "sentiments.js" NAME "analyze" PARAMETERS("p0" STRING, "p1" DOUBLE);

# Negative Sentiment – Test Query

```
SELECT sentiments.analyze(
           'WTF. Never bought a worse product, disappointing!', 3)
  FROM dual;
```

```
{
   "score": -9,
   "comparative": -1.2857142857142858,
   "tokens": [
      "wtf",
      "never",
      "bought",
      "a",
      "worse",
      "product",
      "disappointing"
   ],
…
```

```
…
   "words": [
      "disappointing",
      "worse",
      "wtf"
   ],
   "positive": [],
   "negative": [
      "disappointing",
      "worse",
      "wtf"
   ]
}
```

# Top 10 Negative Sentiments – Query

```
WITH
   base AS (
      SELECT sentiments.analyze(text) AS jdoc, text
        FROM router),
   scored AS (
      SELECT to_number(json_value(jdoc, '$.score')) AS score,
             round(to_number(
                json_value(jdoc, '$.comparative')), 2) AS comparative,
             text
        FROM base)
SELECT score, comparative, text
  FROM scored
 ORDER BY score
 FETCH FIRST 10 ROWS WITH TIES;
```

# Top 10 Negative Sentiments – Result (Excerpt)

```
SCORE COMPARATIVE TEXT
----- ---------- ----------------------------------------
  -12        -.44 So now , I will buy a - \/ + 12v 2.5 amp or
                  3amp power adapter or brick and splice this
                  stupid sized end on it

   -8        -1.6 Same problem - no DHCP
…
   -7        -.37 SETUP SOFTWARE - For my application , I found
                  the setup software on the enclosed CD to be
                  useless

11 rows selected.
```

# Version 20c
# (Sensible Guess)

# Embedded GraalVM Enterprise Edition

- JavaScript based on GraalVM
- Probably no other languages based on GraalVM
- Java based on OJVM (as in previous releases)
- No dependencies between OJVM and GraalVM

# Features

- Interoperability between JS, SQL, PL/SQL but no Java
- Security model similar to the one for OJVM
- ”dbjs” functionality as in 12c
- Extended type mapping (e.g. no need to convert JSON to string)
- Extended SQL grammar to cover additional languages for stored objects
- Deploying complex JS incl. dependencies via SQL (DIY browserify)

# Some Open Questions

# How to Deal with the Growing Language Base?

- Dedicated tool chain for each language (like “dbjs”)?
- Generic tool chain, configurable for each language?
- Include less common languages that are part of the GraalVM distribution?
- Include custom languages?
- Update, upgrade embedded GraalVM?

# How to Integrate Package Managers/Repositories?

- Examples
  - Maven Central for Java (incl. other public and private repositories)
  - npm for JavaScript
  - PyPi for Python
  - RubyGems for Ruby
  - NuGet for .NET
- Dealing with references to foreign packages during deployments?
  - Security vs. usability
- Solution for languages without package managers/repositories?

# How to Deal with “non-browserify-able” Languages?

- browserify produces a single source incl. dependencies for JavaScript
- browserify makes deployments simpler and faster
- Some languages like Java are “non-browserify-able”
- Deploying JAR files into the database
  - extracts files and stores them as dedicated database objects
  - is slow, confusing
- What are the concepts to store ”non-browserify-able” language files for GraalVM?

# What Happens to the Existing Languages in the DB?

- Namely
  - SQL
  - PL/SQL
  - XSLT
  - XQuery
  - Java
  - C
- Technically they could all run on GraalVM, right?
- SQL and PL/SQL are special though, as the compiler/interpreter needs a connection

# Version 30c
# (Highly Speculative)

# Embedded GraalVM Enterprise Edition

- All code in the Oracle Database is running on GraalVM
  - C (kernel, custom libraries), C++
  - Java, Scala, Kotlin, Groovy, Clojure, …
  - SQL (many dialects), PL/SQL, T-SQL, …
  - XSLT, XQuery, …
  - JavaScript, Phyton, R, Ruby, …
  - C#, Visual Basic .NET, …
- Run static SQL from all languages
- Enable/disable optional and custom languages
- OJVM is not available anymore (no need, replaced by Java on GraalVM)

# Features

- Interoperability between all languages
- Integration with “Oracle Universal Package Manager”
  - Free cloud service for public packages, supporting all languages
  - Cost option for private packages
  - Integration with other package managers/repositories such as npm, Maven Central
  - Manages local copies, security aspects, and more …
- Ad-hoc use of all languages in SQL
  - E.g. in the with_clause of a select statement (similar to PL/SQL today)
  - Including references to other packages known by the Universal Package Manager

# Core Messages

# The Future Is Bright

- GraalVM is polyglot, embedded, efficient, open-sourced.
- Code in the client or database server?
  - Option to choose
  - Easier to move
- Database centric development becomes easier when additional languages (besides PL/SQL) are treated as first-class citizens.

# Questions and Answers...

**Philipp Salvisberg**
**Senior Principal Consultant**

**Tel. +41 58 459 52 31**
**philipp.salvisberg@trivadis.com**




trivadis
makes IT easier.