

Oracle Multilingual Engine (MLE) –

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

Philipp Salvisberg



@phsalvisberg



<https://www.salvis.com/blog>

BASEL • BERN • BRUGG • BUKAREST • DÜSSELDORF • FRANKFURT A. M. • FREIBURG I. BR. • GENÈVE
HAMBURG • KOPENHAGEN • LAUSANNE • MANNHEIM • MÜNCHEN • STUTTGART • WIEN • ZÜRICH

trivadis

■ About Me

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

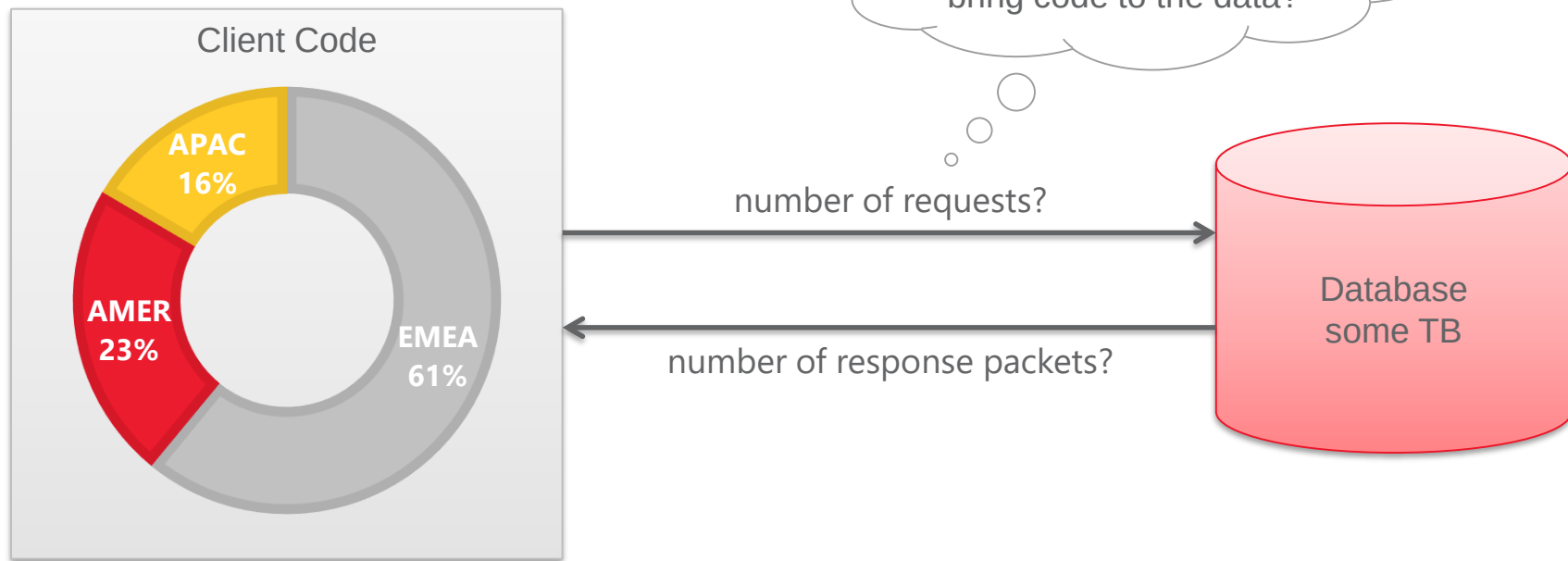


■ Agenda

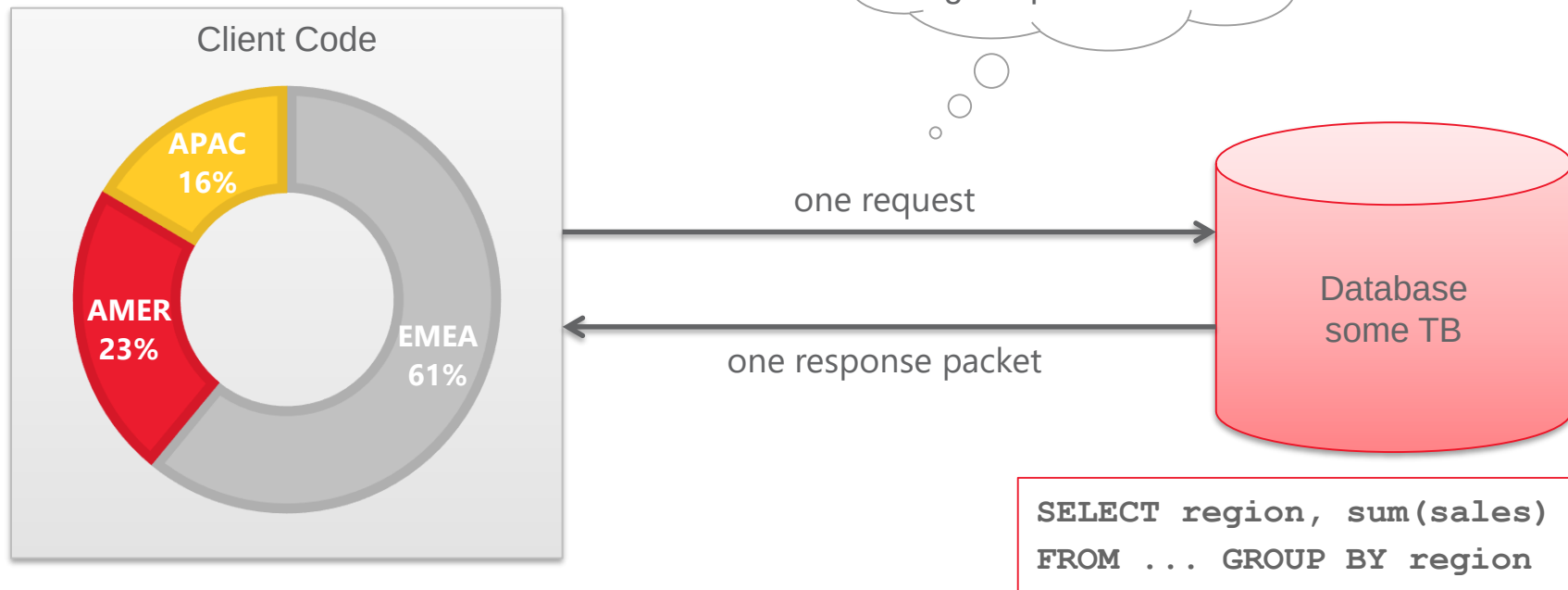
1. **Code in the Database**
2. **GraalVM**
3. **Version 12c – Beta (Experimental)**
4. **Version 20c – Production (Sensible Guess)**
5. **Version 30c – Production (Highly Speculative)**
6. **Core Messages**

Code in the Database

■ Processing Data



■ Bring Code to the Data



■ Languages in Oracle Database 19c

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

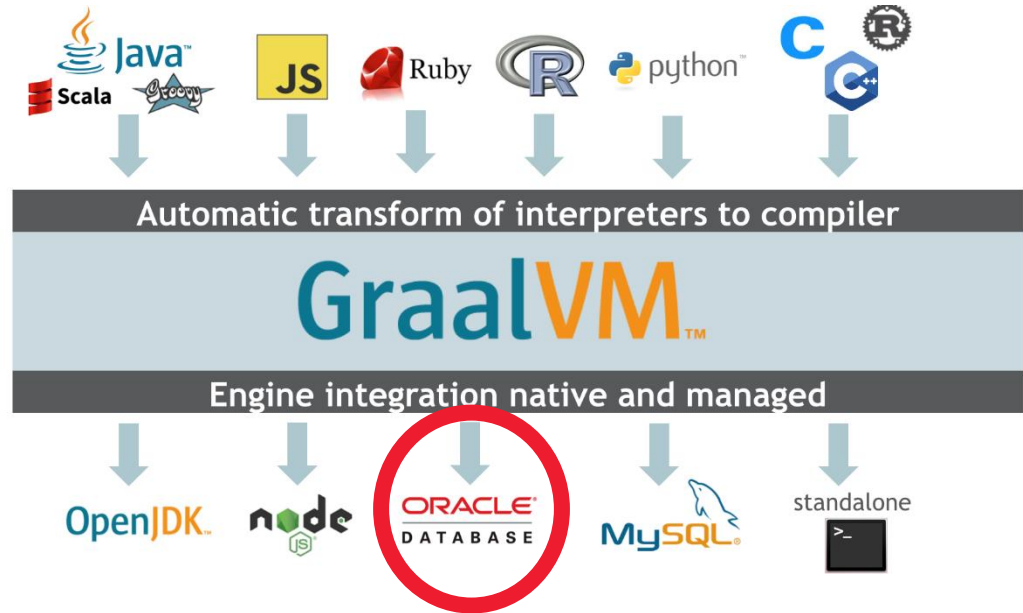
- Languages

■ Embeddable

- Databases
- HTTP-Server

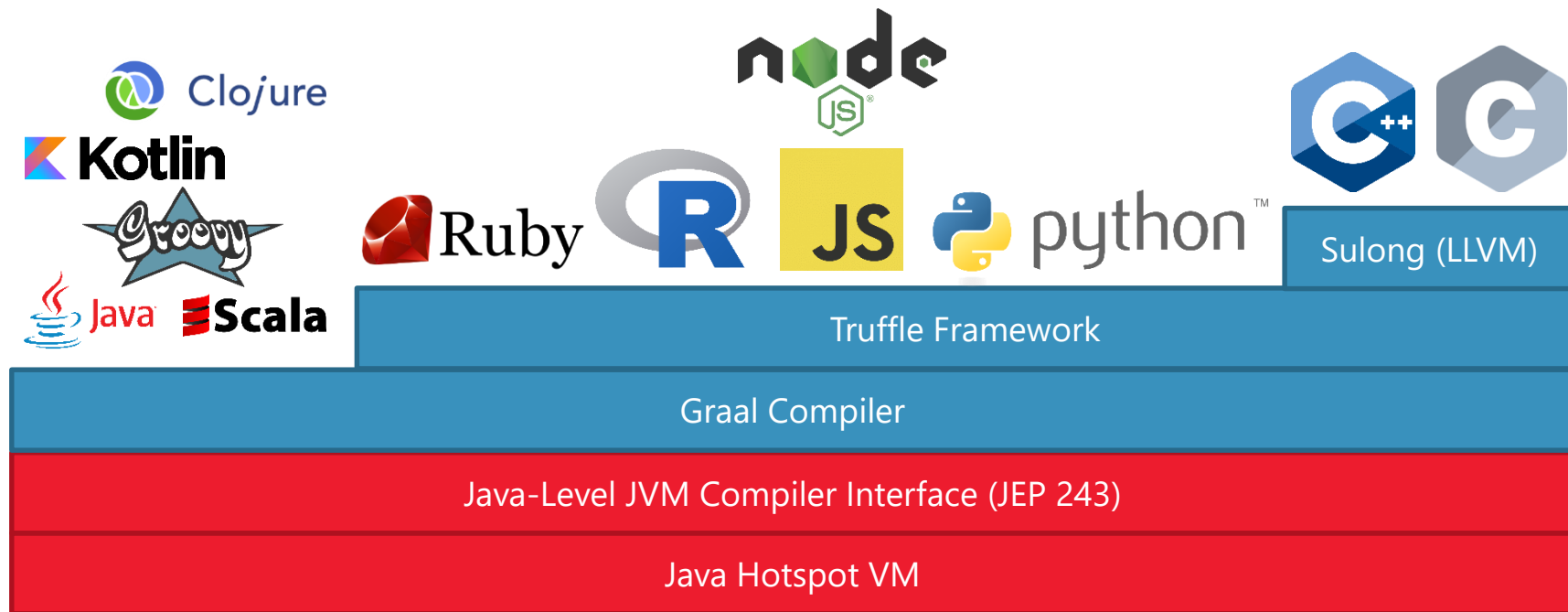
■ Efficient

- Standalone
- Interoperability
(shared memory)

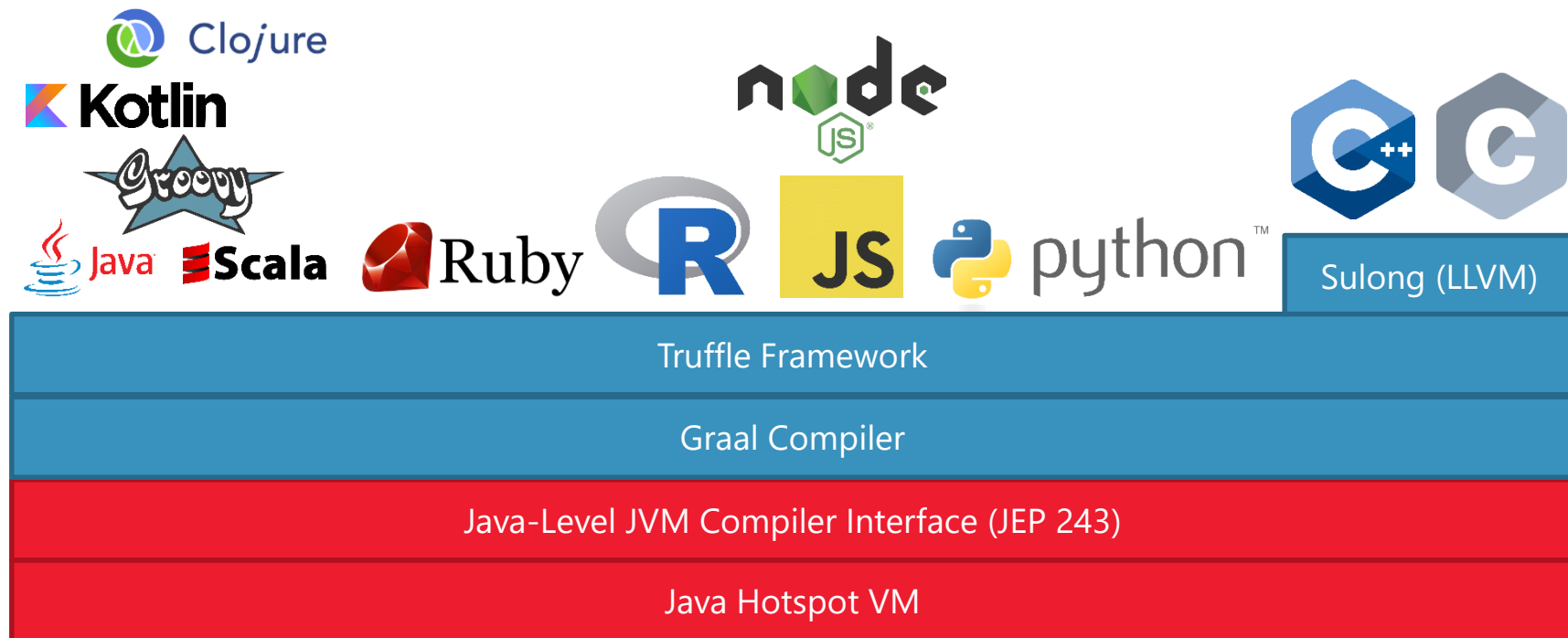


Source: <https://www.graalvm.org/docs/>

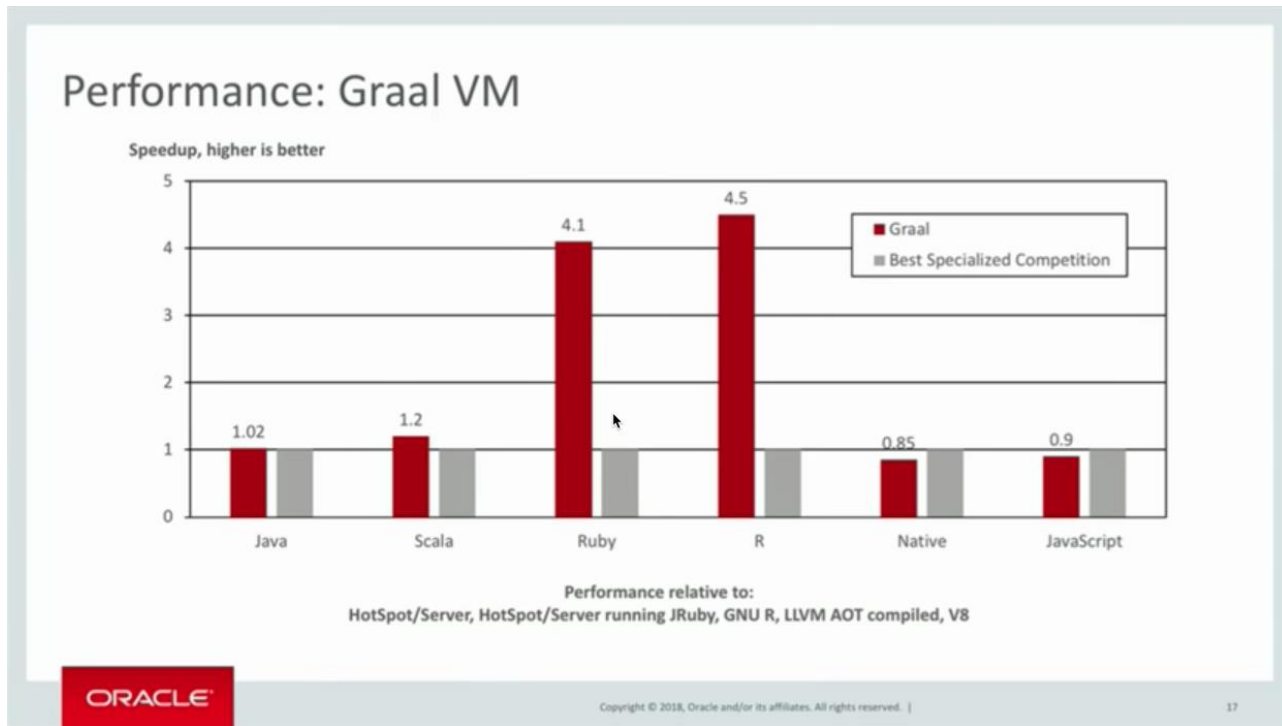
Architecture Today



Architecture Tomorrow



■ Performance



Source: <https://youtu.be/8AYESZlaacg?t=1002>

Graal: High-Performance Polyglot Runtime by Thomas Wuerthingner 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)

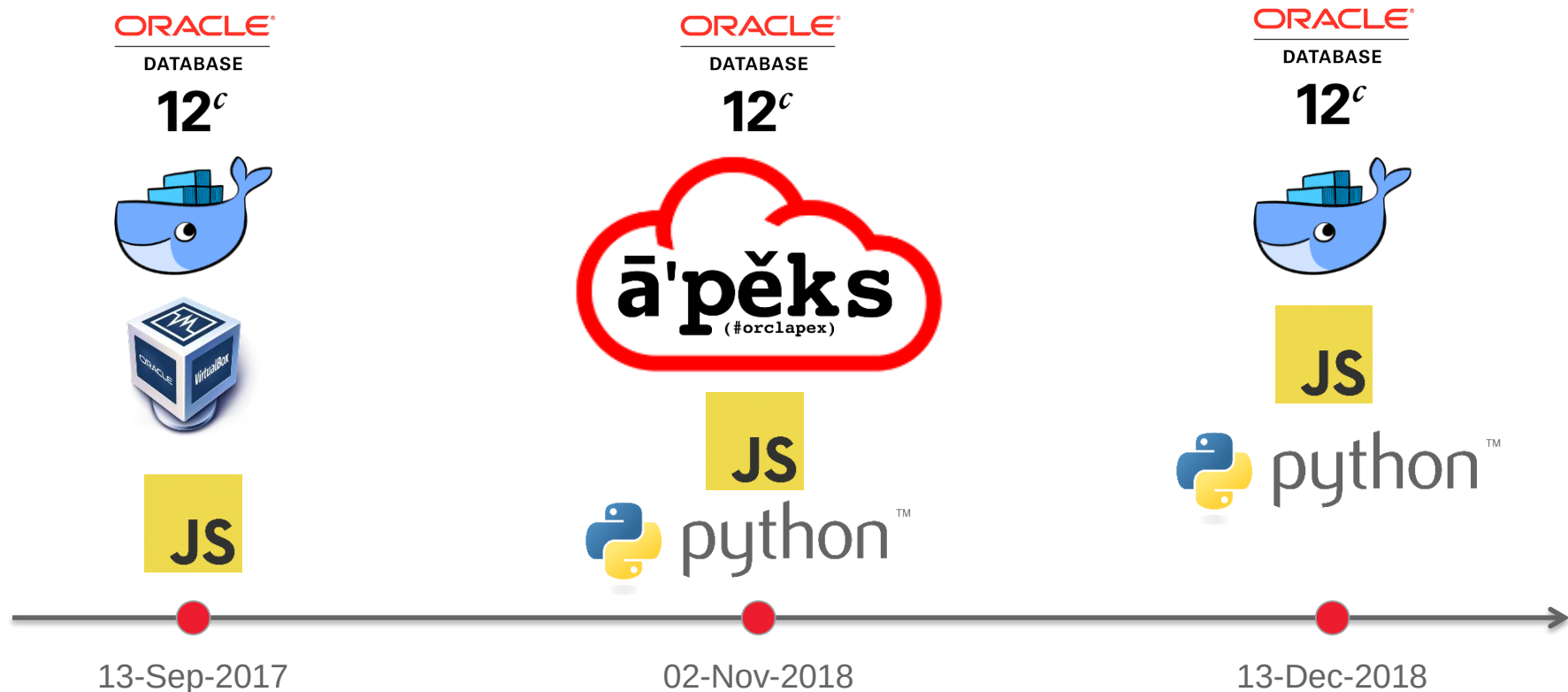
GraalVM EE provides additional performance, security, and scalability relevant for running critical applications in production. It is free for evaluation uses and available for download from the [Oracle Technology Network](#). We provide binaries for GraalVM EE for Linux or Mac OS X on x86 64-bit systems.

DOWNLOAD FROM OTN

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

Version 12c – Beta (Experimental)

MLE Beta History





Documentation

 Oracle Database MLE 0.3.0

 Search

 oracle-db-mle

Oracle Database MLE 0.3.0
Overview
Release notes
Running MLE via Docker
JavaScript ▾
Python ▾

Using MLE with Docker

This beta release of Oracle Database MLE comes as **pre-built Docker image**.

The docker image is based on the Docker build files as contained in this **GitHub repository**.

Running the Image

The Docker image is available for download as a `tar.gz` file from the **Oracle Technology Network**.

After downloading the image, it needs to be loaded into docker using the **docker load command**.

```
$ docker load --input mle-docker-0.3.0.tar.gz
```

Once loaded, it can be run using the **docker run command**.

Table of contents
Running the Image
Starting and Stopping a Container
Starting an Interactive Shell
Changing the admin accounts passwords

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

■ Create Docker Container

Load image from downloaded archive file

```
docker import mle-docker-0.3.0.tar.gz
```

Create container

```
docker run --privileged --name mle2 \  
  -p 1581:1521 -p 5081:5500 \  
  -e ORACLE_SID=mlecdb \  
  -e ORACLE_PDB=mlepdb1 \  
  -e ORACLE_PWD=oracle \  
  -v /Users/phs/docker/mle2/oradata:/opt/oracle/oradata \  
  -v /Users/phs/docker/mle2/myproject:/home/oracle/myproject \  
  mle-docker:0.3.0
```

■ Install Node.js

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

```
docker exec -u root --privileged -it mle2 bash
```

Install nodejs and dependencies-bundler browserify within the mle2 container

```
yum install -y gcc-c++ make  
curl -sL https://rpm.nodesource.com/setup_6.x | bash -  
yum install -y nodejs  
npm install -g browserify
```

■ Install SQLcl

Copy downloaded and extracted SQLcl folder into the mle2 container

```
docker cp sqlcl mle2:/opt/oracle/product/12.2.0.1/dbhome_1/sqlcl/
```

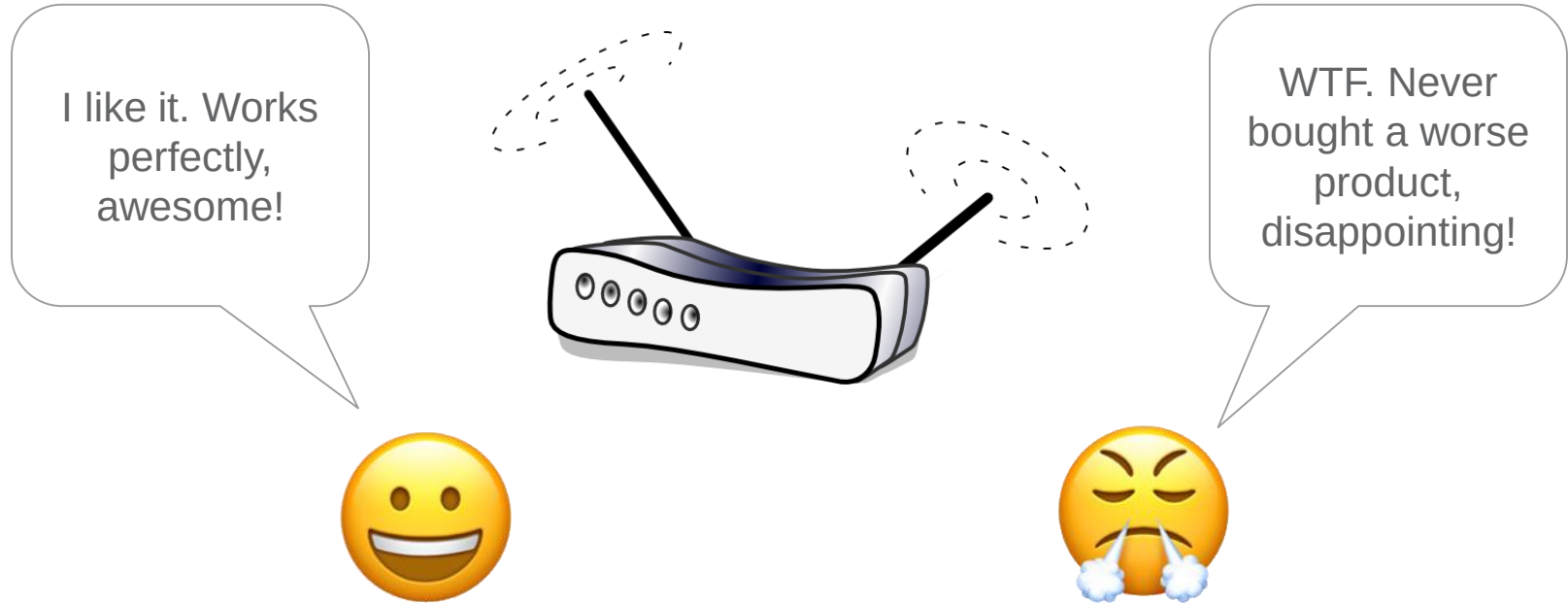
Open a shell in within the MLE docker container as "root"

```
docker exec -u root --privileged -it mle2 bash
```

Fix privileges and set environment within the mle2 container

```
chown -R oracle:oinstall /opt/oracle/product/12.2.0.1/dbhome_1/sqlcl/  
echo "export PATH=\$ORACLE_HOME/sqlcl/bin:\$PATH" >> /home/oracle/.bashrc  
echo "export LC_ALL=en_US.UTF-8" >> /home/oracle/.bashrc
```

■ Use Case – Sentiment Analysis of Customer Feedback



Open Data: <https://www.cs.uic.edu/~liub/FBS/sentiment-analysis.html#datasets>

■ npm – Package Manager for JavaScript

The screenshot shows the npm package page for 'sentiment'. At the top, there's a search bar and navigation links for 'Join' and 'Log In'. The package name 'sentiment' is prominently displayed, along with its version '5.0.1', status 'Public', and 'Published 9 months ago'. Below this, there are four tabs: 'Readme' (highlighted in yellow), '0 Dependencies', '102 Dependents', and '23 Versions'. The 'Readme' section describes 'sentiment' as an 'AFINN-based sentiment analysis for Node.js', showing 'build passing' and 'Greenkeeper enabled' status. It provides a brief description of the module's purpose and a list of features. To the right, there's an 'install' section with a code snippet for installing the package, a 'weekly downloads' chart showing 17,383 downloads, and a table of package metadata including version (5.0.1), license (MIT), open issues (13), and pull requests (8). At the bottom, there's a 'Usage example' section with a code snippet demonstrating how to use the 'sentiment' module.

sentiment 5.0.1 • Public • Published 9 months ago

Readme 0 Dependencies 102 Dependents 23 Versions

sentiment

AFINN-based sentiment analysis for Node.js

build passing Greenkeeper enabled

Sentiment is a Node.js module that uses the **AFINN-165** wordlist and **Emoji Sentiment Ranking** to perform **sentiment analysis** on arbitrary blocks of input text. Sentiment provides several things:

- Performance (see benchmarks below)
- The ability to append and overwrite word / value pairs from the AFINN wordlist
- The ability to easily add support for new languages
- The ability to easily define custom strategies for negation, emphasis, etc. on a per-language basis

install

```
> npm i sentiment
```

weekly downloads

17,383

version	license
5.0.1	MIT
open issues	pull requests
13	8

homepage repository

Usage example

```
var Sentiment = require('sentiment');
var sentiment = new Sentiment();
var result = sentiment.analyze('Cats are stupid.');
```

console.dir(result); // Score: -2, Comparative: -0.666

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

■ Install “sentiment” Package

Open a shell in within the MLE docker container

```
docker exec -it mle2 bash
```

Install package in your project directory

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

■ JavaScript Wrapper

analyze.js

```
function analyze(phrase) {  
    var Sentiment = require('sentiment');  
    var sentiment = new Sentiment();  
    var result = sentiment.analyze(phrase);  
    return JSON.stringify(result, null, 3);  
}  
  
module.exports.analyze = analyze;
```

■ Positive Sentiment – Test via Node.js



```
node -p 'require("./analyze.js")
        .analyze("I like it. Works perfectly, awesome!")'
```

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

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


■ Deploy Into Database – without dbjs (1)

1. Create single JavaScript bundle file including all dependencies

```
browserify analyze.js -s analyze -o analyzeBundle.js
```

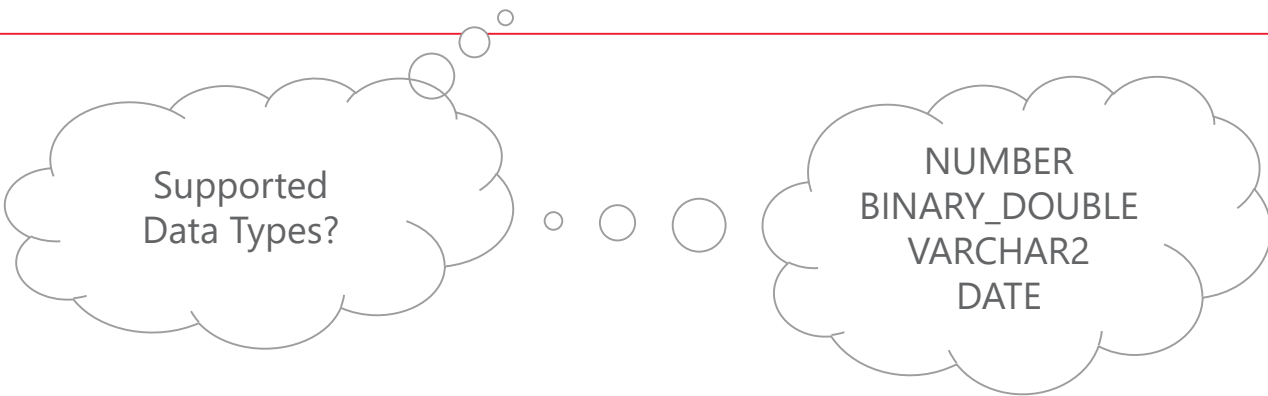
2. Create and JavaScript DDL and install it with SQLcl

```
SET SCAN OFF  
CREATE OR REPLACE JAVASCRIPT SOURCE NAMED "analyzeBundle.js" AS  
{content of analyze-bundle.js}  
/
```

■ Deploy Into Database – without dbjs (2)

3. Create PL/SQL wrapper and install it

```
CREATE OR REPLACE FUNCTION analyze(  
    in_phrase IN VARCHAR2  
) RETURN VARCHAR2 AS  
LANGUAGE JAVASCRIPT  
NAME 'analyzeBundle.js.analyze(phrase string) return string';
```



Supported
Data Types?

NUMBER
BINARY_DOUBLE
VARCHAR2
DATE

■ Negative Sentiment – Test Query

```
SELECT analyze('WTF. Never bought a worse product, disappointing!')
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 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, comparative
FETCH FIRST 10 ROWS ONLY;
```

■ Top 10 Negative Sentiments – Result (Excerpt)



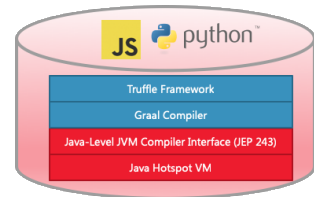
SCORE	COMPARATIVE	TEXT
-------	-------------	------

-8	-0.47	All in all , bad device , bad customer service ... run like hell from this product
-8	-0.42	Exasperated at 3 AM , I called it quits and will be sending this awful , deceptive product back
...		
-5	-0.13	I suspect Netgear takes defective returns , wraps them in new plastic , and labels them " Refurb , " and hopes they work or that whatever defect they had would not be noticed by the new buyer

10 rows selected.

Version 20c – Production (Sensible Guess)

■ MLE in Oracle Database 20c



- Stored Objects based on GraalVM for JavaScript and Python
- Dynamic execution of GraalVM languages for JavaScript and Python
- Interoperability between all languages via SQL and PL/SQL for a subset of data types
- Run dynamic SQL from JavaScript and Python via default connection
- No dbjs and dbpy as part of the Oracle Database
- No access to network nor file system via JavaScript and Python
- No references to other MLE modules
- Deliberately undocumented features, e.g. to implement aggregate or table functions in JavaScript
- Java based on OJVM without dependencies between OJVM and GraalVM

Some Open Questions

■ How to Deal with the Growing Language Base?

- Dedicated tool chain for each language (like "dbjs" and "dbpy")?
- 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 and webpack
 - produce a single source incl. dependencies for JavaScript
 - make deployments simpler and faster
- Some languages like Java are “non-browserify-able”
- What are the concepts to store “non-browserify-able” language files for GraalVM?
- Is it really necessary to store every file in a JAR file as a separate database object?

■ 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?
- Static SQL and PL/SQL are special though, as the compiler needs a connection

Version 30c – Production (Highly Speculative)

■ MLE in Oracle Database 30c (1)

- 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, Python, R, Ruby, ...
 - C#, Visual Basic .NET, ...
- Enable/disable optional and custom languages
- Efficient interoperability between all languages supporting all data types
- Run static and dynamic SQL from all languages via default connection



■ MLE in Oracle Database 30c (2)

- Controlled access to network and file system for 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
 - Including references to other packages known by the Universal Package Manager
- OJVM is not available anymore

■ MLE in Oracle Database 30c (3)

- Moving code at runtime between Java Virtual Machines
 - Based on statistics
 - To reduce network roundtrips
 - To improve overall runtime performance



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

@phsalvisberg