Agenda

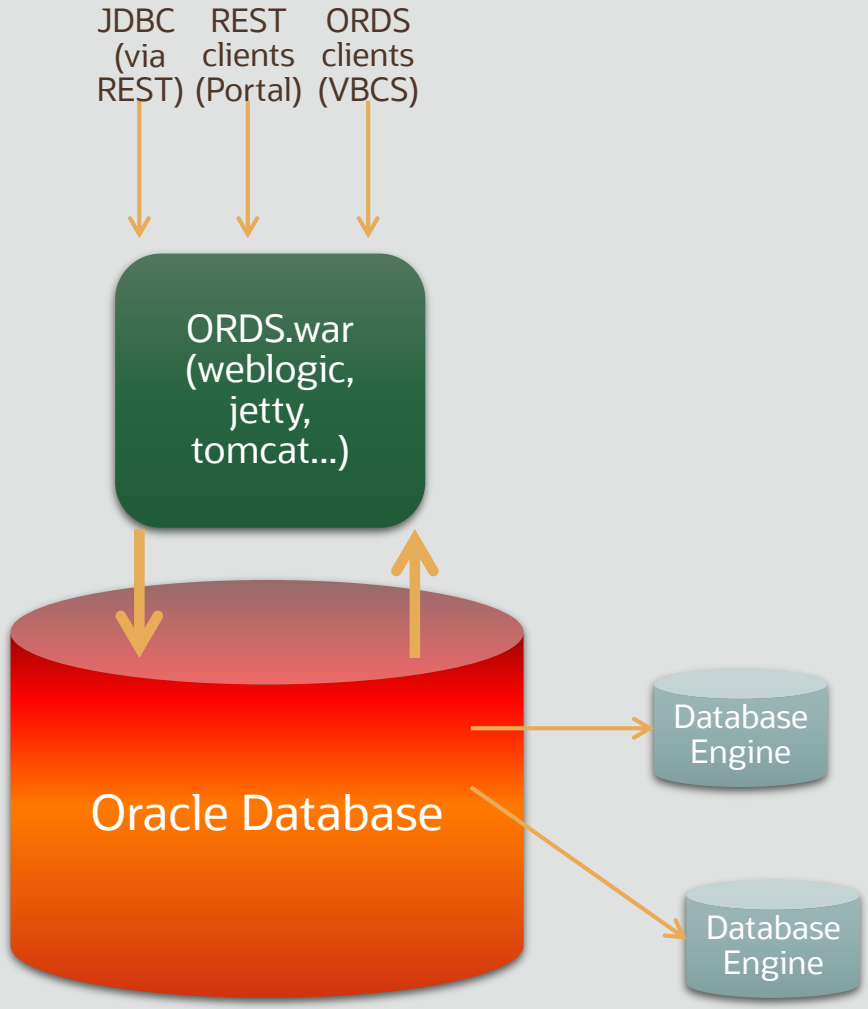
- Kurze Erklärung von graphQL versus ORDS
- graphQL Features live
- Aufbau der Entwicklungsumgebung
- Deployment von Demo-Anwendungen

ORDS – Oracle REST Data Services

REST & JSON -erzeugende Engine in der Datenbank
ORDS.war lediglich als „Durchreicher“ & Security

JSON Generierung + Speicherung in der Datenbank

Generierung durch Konfiguration
„auto-REST“ von Tabellen und Views
Oder freie Programmierung
mit SQL (JSON API), PL/SQL und Java



ORDS – Oracle REST Data Services

technische Besonderheiten

- Beliebige HTTP Methoden (GET, POST, PUT, DELETE,...)

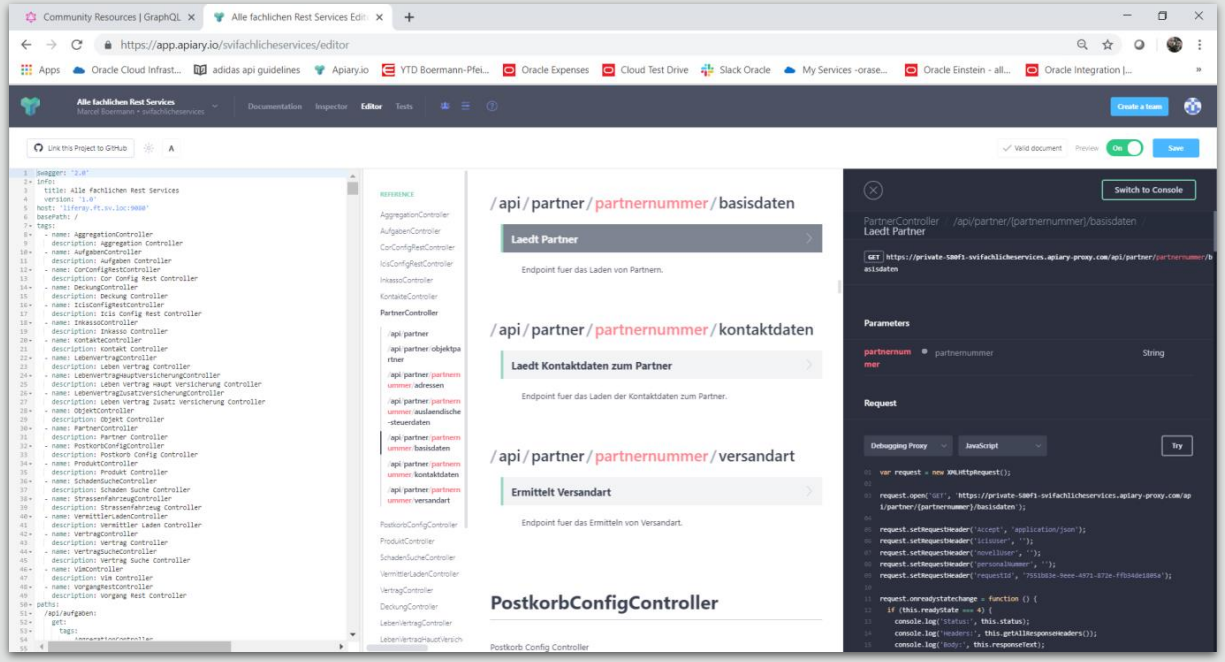
  Vordefinierte Parameter (filter, pagination, limits, query)
  bei AutoREST,
  zusätzliche freie Parameter
  bei Programmierung

  Query-Language (Filter) Konverter nach Oracle SQL

  Mapping von REST nach DB-Zugriff über „Handler“

- Metadaten-Syntax: Swagger/OpenAPI
  Introspection via JDBC Tools (ORDS JDBC Treiber),
  Swagger&APIary Tools

- denkbar ist auch GraphQL/GraphiQL „on top“



GraphQL

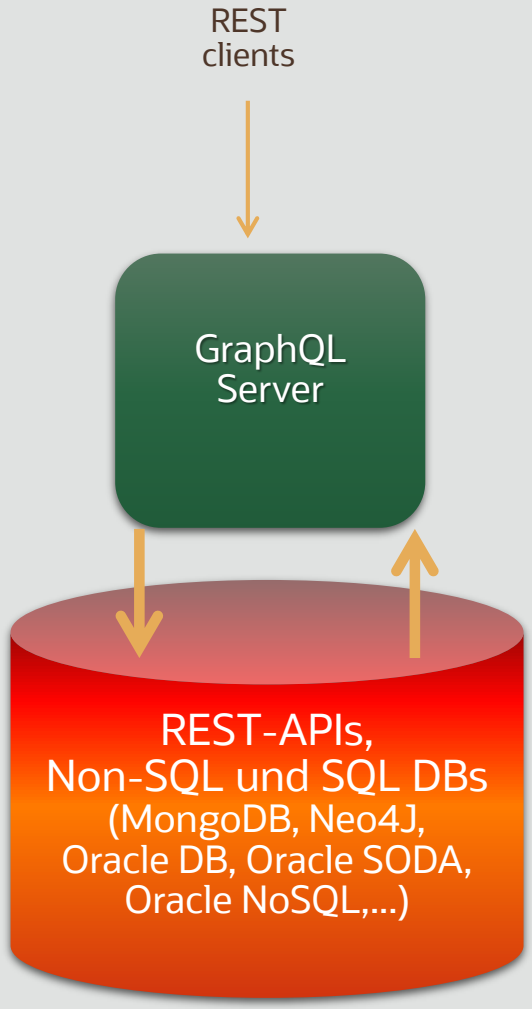
Graphql.org

GraphQL Foundation:
Unterstützung durch zahlreiche „neue“ consumer wie
PayPal, Facebook, Pinterest, yelp, Neo4J, ...
Ca. Mai 2019 hinzugekommen: AWS, IBM

Umfeld agiler (API-) Entwicklung:
„A Query Language for APIs“

Betonung auf
API-Introspection / Veranschaulichung,
API-Definition / Änderung

Java: graphql-java, graphql-mp,...
JavaScript: Apollo,...
Python: graphene,...
Aber auch Server in Perl, PHP, Ruby, Groovy, C#,.....



GraphQL

Lösung für typische REST-Probleme

- Overfetching
  Zu viele Attribute:
  Auswahl benötigter Attribute
  inkl. „Lazy Loading“ von Unterobjekten

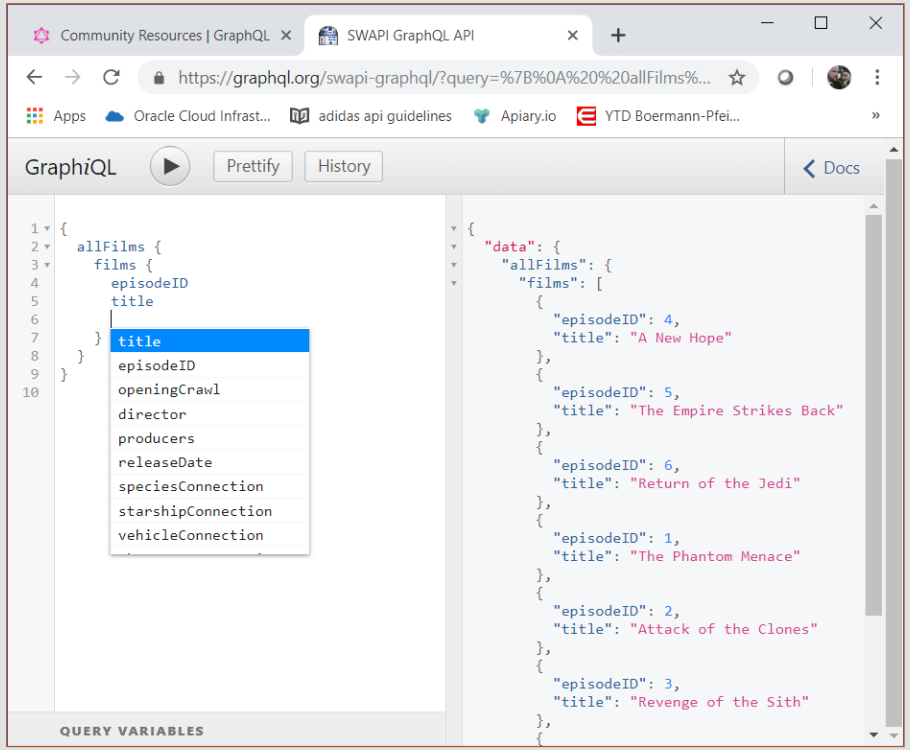
- Underfetching / n+1-Problem
  Zu wenige Informationen, nötige folge-Requests pro Objekt:
  Auswahl von Unterobjekten in einem einzigen Request

Folglich:
Verknüpft Services (& verteilte Daten)
Verknüpft Daten (aus unterschiedlichen Quellen)

- Übergeordnetes Datenmodell
- Resolver / Data Fetcher für idividuelle Zugriffslogik

GraphQL
technische Besonderheiten

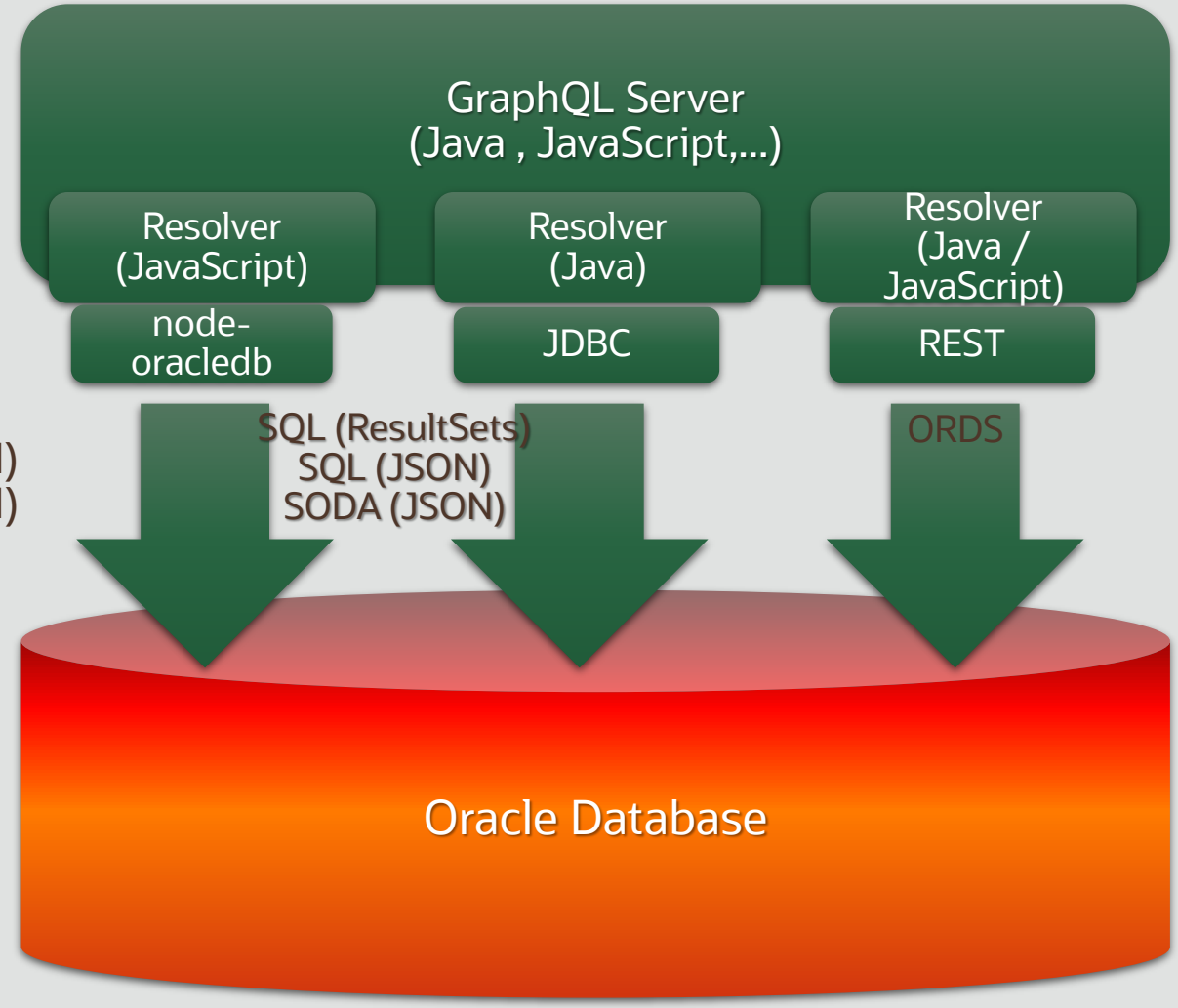
- POST Methode empfängt Operation (Query, Mutation/Änderung inkl. Parameter,....) und gewünschte Attribute & Unterobjekte
- Parameter grundsätzlich frei definierbar (Sortierung, Pagination,... ist manuell einzubauen) Zahlreiche APIs und Erweiterungen verfügbar!
- Sprache der Query/Mutation ist eine Objekt- & Attributstruktur, ähnlich JSON
- Mapping von REST nach DB-Zugriff, Auflösung der Objekthierarchie („Knoten“) über „Resolver“
- Datentypen-Definition über API oder GraphQL Syntax
- Metadaten-Syntax: SDL (Schema Definition Language) Introspection via REST bzw. „GraphiQL“ Tool



GraphQL
Anbindung von Oracle Datenbanken

Möglichkeiten:

- SQL Query
  erzeugt Zeilen und Spalten, sind im Resolver auf Objekte zu mappen
- SQL Query
  erzeugt JSON Dokumente in Datenbank (JSON API) oder ruft JSON Dokumente ab (Oracle native JSON)
- SODA (Simple Oracle Document Access)
  NVPair-Zugang zu Oracle Datenbanken ab 18c
  Ablage, Suche, Indizierung von JSON Dokumenten
- REST Services aus Oracle Datenbank
  via ORDS (Oracle REST Data Services)

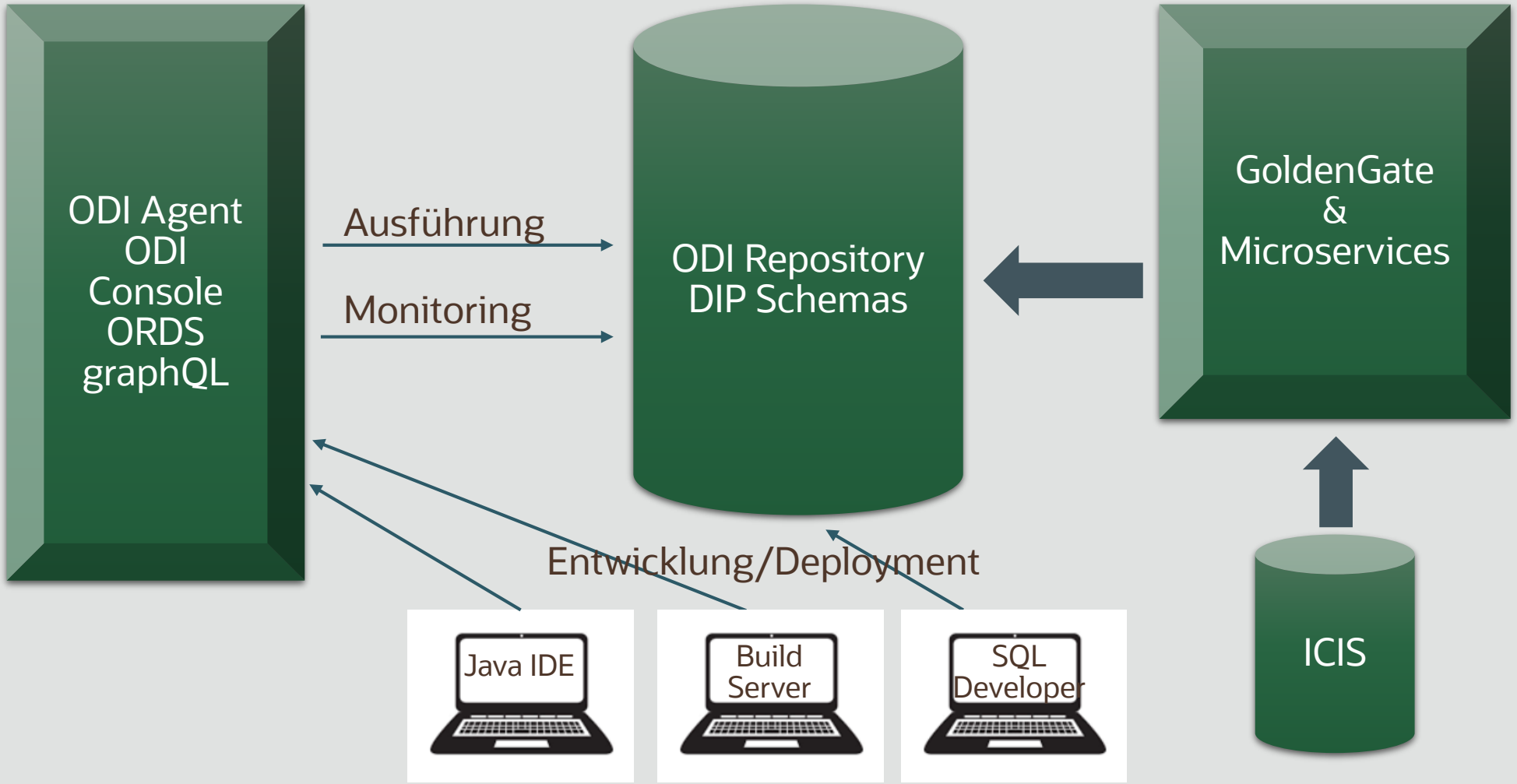


Demo & Features

- Queries & Tools
- CRUD Operationen
- Introspection
- Schema Language SDL
- Java API & Code Beispiele

https://graphql.org/learn
https://www.howtographql.com/

DIP schematisch bei SVi



Einrichtung

- Vorbedingungen bzw. Software

  git
  maven
  IDE, z.B. NetBeans
  Postman oder curl
  SQL*Developer

  optional, aber empfehlenswert: WebLogic “slim”
  WebLogic Maven Plugin
  locale Entwicklung und Test (start/stop, autodeploy…)
  https://download.oracle.com/otn/nt/middleware/12c/122140/fmw_12.2.1.4.0_wls_quick_slim_Disk1_1of1.zip

- Beispiel-Projekte
  für WebLogic / JavaEE Servlet
  und Oracle DB / JavaEE Data Sources:

  https://github.com/ilfur/graphql-starwars-weblogic
  https://github.com/ilfur/graphql-jpa-oracle

- Pakete / pom sinngemäß:

  java-ee 7 (provided)
  graphql-java
  graphql-servlet
  eclipselink-jpa
  graphiql (bzw. lediglich graphiql.html)

Ausblick: graphql-mp

- Java Microprofile Implementation von graphQL
- Elegant via Annotations
- Schlanker als Java EE

- Verfügbar mit Oracle Helidon 2.2
- Läuft nicht(!) in WebLogic (= Java EE)
- Typischerweise Docker + Kubernetes/OpenShift

- Oracle Beispiel (mit Oracle Coherence statt Oracle DB):

  Blog Artikel
  https://medium.com/oracle-coherence/access-coherence-using-graphql-9f24a5ff8f82

  GitHub Source:
  https://github.com/tmiddlet2666/coherence-graphql-example

Oracle Enterprise Cloud Native Java: Building Cloud Native Microservices

Helidon

- Standards-based Microservice Framework
- Supporting Eclipse MicroProfile specification

Coherence

- Scalable, high available InMemory Key-Value database
- stateful services as easy to scale as stateless services

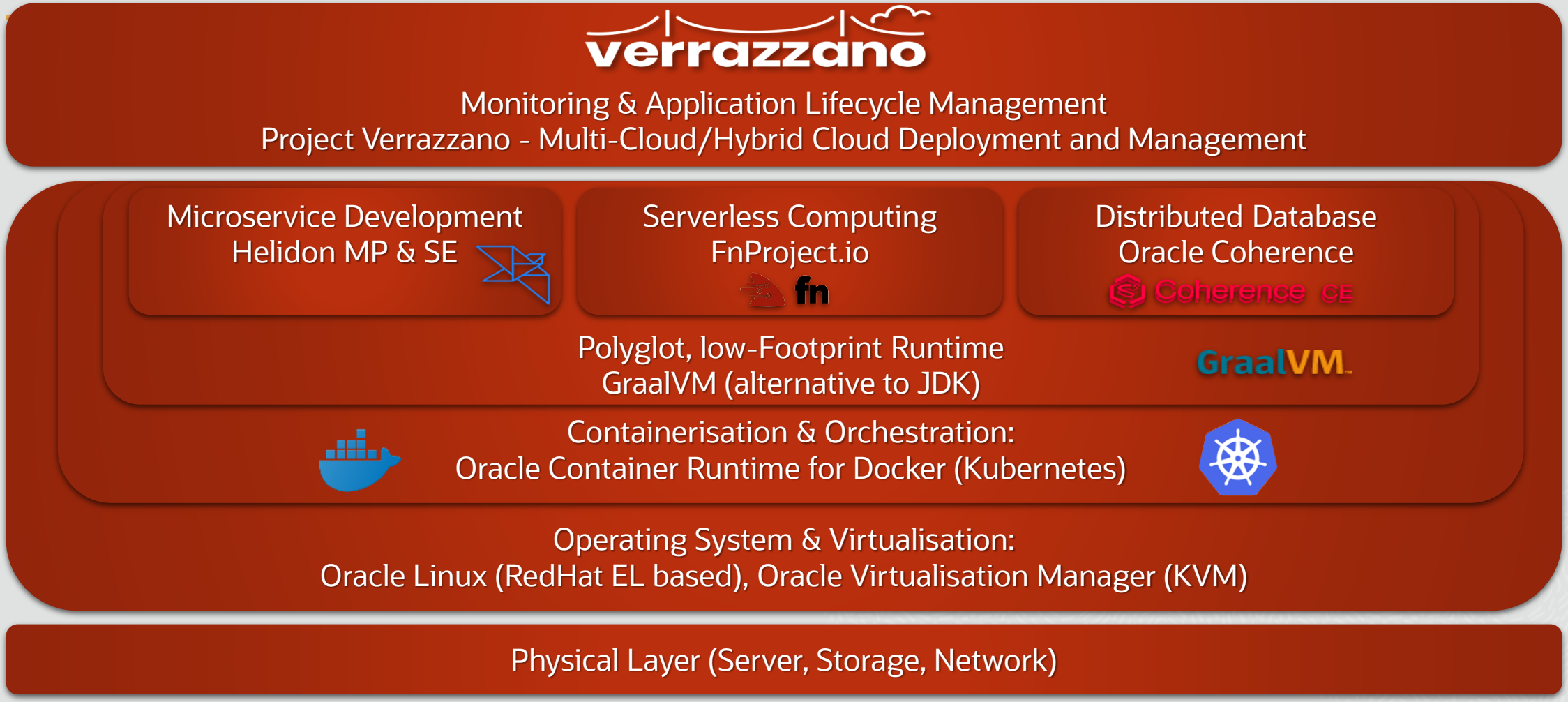
GraalVM

- Polyglot runtime (Java, Node.JS, Python, Perl,…)
- Faster than standard JavaVM , with “Native Image” Support

Verrazzano

- Application Lifecycle Management (ALM)
- Hybrid cloud Support

Oracle Cloud Native Platform – all OpenSource



Diskussion: Vorgehensweise im Projekt

Vorschlag:

- Anwendungsentwicklung definiert SDL Datenmodell (=übergeordnet, Objekt-Hierarchie)
- Pro Resolver/Data Fetcher eine DB-View, Zugriff per JPA oder REST (JSON-B)
- Jede View führt die ID/PK des Root-Objektes zwecks Verlinkung