

# Java aktuell

iJUG  
Verbund  
[www.ijug.eu](http://www.ijug.eu)

## Sicherheit

Schützen Sie Ihre Anwendung  
vor unbekannten Bedrohungen

## DevOps

Worauf es bei einer erfolgreichen  
DevOps-Kultur ankommt

## GraalVM

Der heilige Gral  
der Compiler?

# Immer auf der sicheren Seite

# ORAWORLD

Das e-Magazine für alle Oracle-Anwender!

EOUC  
EMEA ORACLE SERGROUP COMMUNITY

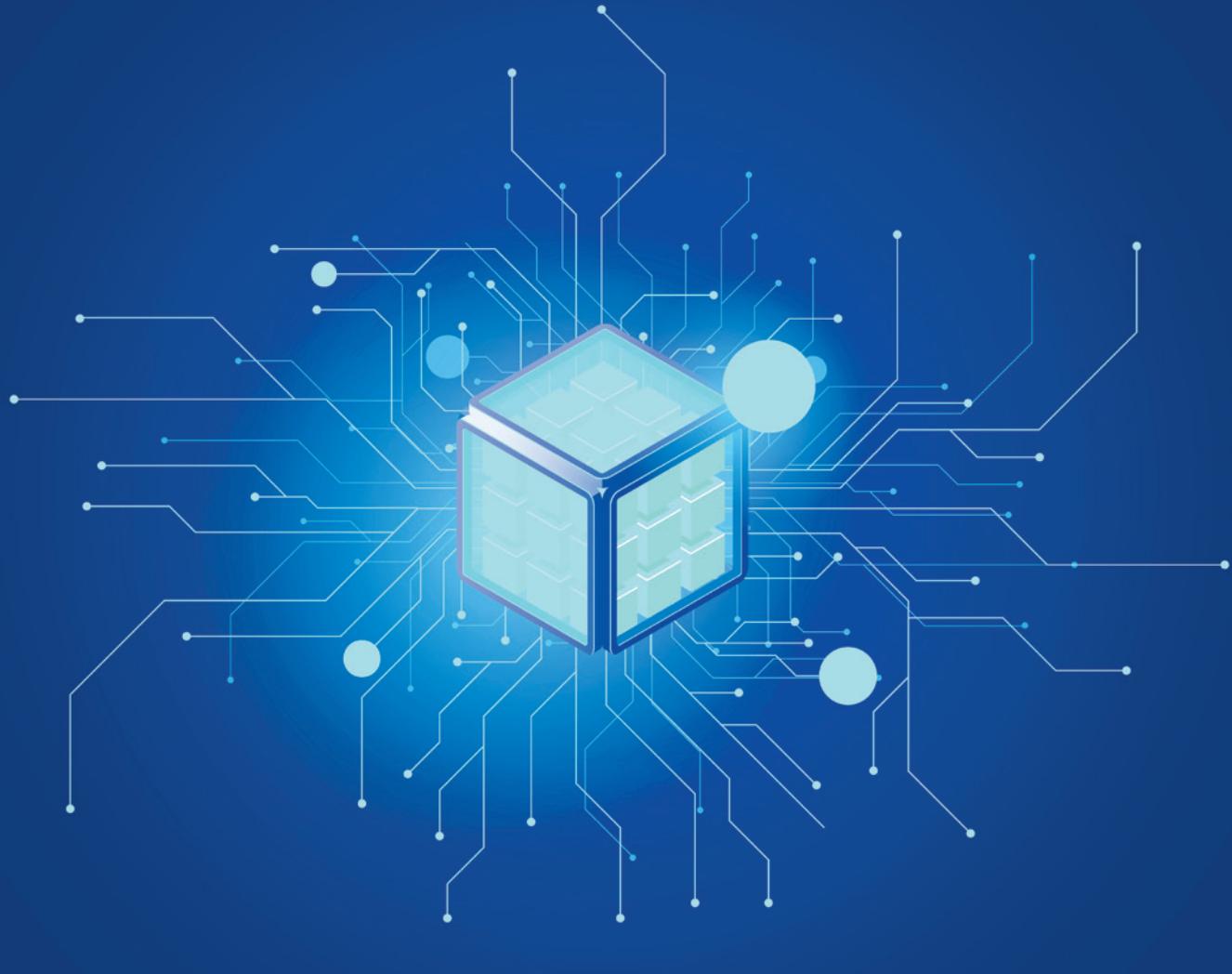


- Spannende Geschichten aus der Oracle-Welt
- Technologische Hintergrundartikel
- Leben und Arbeiten heute und morgen
- Einblicke in andere User Groups weltweit
- Neues (und Altes) aus der Welt der Nerds
- Comics, Fun Facts und Infografiken

Jetzt Artikel  
einreichen  
oder  
Thema  
vorschlagen!

Jetzt e-Magazine herunterladen  
[www.oraworld.org](http://www.oraworld.org)





# Entwickeln mit der Autonomous Database

Marcel Amende und Kersten Mebus, ORACLE Deutschland B.V. & Co. KG

*Hardware anschaffen, Betriebssysteme aufsetzen, Datenbanken installieren und dabei immer hoffen, dass der DBA einen guten Tag erwischt. In Zeiten schneller Zielerreichung, agiler Entwicklung und Rapid Prototyping kann der Entwickler solche langwierigen Prozesse nicht mehr hinnehmen. Mit der Oracle Autonomous Database [1] ist eine hochverfügbare, hochskalierbare, sichere, administrations- und wartungsfreie Datenbank in der Cloud nur wenige Klicks und Minuten entfernt. Der Artikel beleuchtet die Besonderheiten bei der Entwicklung von Java-, SpringBoot- und JavaScript-Anwendungen mit der Oracle Autonomous Database.*

**D**ie Autonomität der Datenbank bezieht sich auf drei Kernbereiche: Sie wird vollautomatisch provisioniert, unterbrechungs- und administrationsfrei gesichert, gewartet, aufgerüstet und optimiert. Ihre Daten sind immer verschlüsselt und die Sicherheitsfunktionalität wird zum Schutz vor internen und externen Bedrohungen laufend gepflegt. Fehlerzustände werden automatisch erkannt und behoben, um auf Grundlage der Clustertechnologie der Oracle-Datenbank („Real Application Cluster, RAC“) und einer regionsübergreifenden Replikation einen unterbrechungsfreien Zugang zu den Daten zu gewährleisten. Eine Verfügbarkeit von 99,995 Prozent ist so garantiert.

## Grundlagen der Autonomous Database

Die gute Nachricht vorweg: Die Oracle Autonomous Database lässt sich kostenfrei testen und sogar dauerhaft kostenfrei nutzen. Erste Gehversuche kann man im Rahmen eines kostenlosen Trials [2] (30 Tage, plus Credits im Wert von 300 US-Dollar) der Oracle Cloud unternehmen. Darüber hinaus sind mit jedem Oracle-Cloud-Zugang

## Autonomous Databases *in alwaysfree (root) Compartiment*

Create Autonomous Database								
Name	Database Name	State	Dedicated Infrastructure	CPU Core Count	Storage (TB)	Workload Type	Created	⋮
DB Java Aktuell <small>Always Free</small>	DBJavaAktuell	● Available	No	1	0.02	Transaction Processing	Mon, Sep 23, 2019, 11:31:13 AM UTC	⋮
DB ATP free <small>Always Free</small>	DBATPfree	● Available	No	1	0.02	Transaction Processing	Tue, Sep 17, 2019, 9:25:20 AM UTC	⋮
Displaying 2 Autonomous Databases								Page 1 < >

Abbildung 1: Autonome "Always Free" Datenbanken für verschiedene Workloads in der Oracle Cloud Konsole (Quelle: Marcel Amende)

zwei autonome Datenbanken mit jeweils einer OCPU (entspricht der CPU-Kapazität eines physischen Cores eines Intel-Xeon-Prozessors mit aktiviertem Hyperthreading) und 20 GB Datenspeicher als „Always Free“-Dienste [3] lebenslang kostenfrei nutzbar (*siehe Abbildung 1*). Gleiches gilt übrigens auch für Compute (zwei VMs mit 1/8 OCPUs und 1 GB RAM), einen Load Balancer mit 10 MBit/s Bandbreite, 100 GB Block Storage, 10 GB Object Storage, 10 GB Archive Storage, 10 GB Outbound Traffic, eine Million Notifications und 1.000 E-Mails im Monat.

Die autonome Datenbank ist in zwei Produktvarianten und auf zwei verschiedenen Infrastrukturplattformen verfügbar. Die Varianten sind:

- Autonomous Transaction Processing (ATP), für transaktionale Verarbeitungen
- Autonomous Data Warehouse (ADW), für Data-Warehouse-Anwendungen

Beide Varianten lassen sich wiederum „Serverless“ oder „Dedicated“ betreiben. Die Grundlage für die schnelle Provisionierung ist in beiden Fällen die Mandantenfähigkeit der Oracle-Datenbank („Multitenancy“). Eine „Serverless“-Datenbank ist aus technischer Sicht ein isolierter und elastischer Cluster von Datenbankcontainern („Pluggable Database, PDB“), die auf einer extrem leistungsfähigen Oracle-Exadata-Datenbankmaschine in der Oracle Cloud laufen. Bei den „Dedicated“-Datenbanken nutzen Kunden exklusiv zugeordnete Exadata-Datenbankmaschinen in ihrem privaten Subnetz der Oracle Cloud für den Betrieb eigener Containerdatenbanken.

### Provisionierung einer „Serverless“-Datenbank

Eine Autonomous Database kann über die Oracle Cloud Infrastructure (OCI) Konsole, die OCI-Kommandozeilenschnittstelle (OCI CLI) oder automatisiert per Terraform bereitgestellt werden. *Abbildung 2* zeigt die Provisionierung über die Konsole in der Rechenzentrumsregion Frankfurt. Es müssen lediglich ein Compartiment, der Datenbankname, die Art der Arbeitslast (Data Warehouse oder Transaction Processing), der Infrastrukturtyp (Serverless oder Dedicated), die Anzahl der CPUs (bei Serverless 1 bis 128), die Speichergöße (1 – 128 TB beziehungsweise 20 GB bei „Always Free“), die Art der Datenbanklizenz (bereits vorhanden oder inkludiert) gewählt und ein Passwort vergeben werden. Nicht einmal drei Minuten später steht die neue Datenbank inklusive Oracle Application Express (APEX), Oracle Rest Data Services (ORDS), Oracle SQL Developer Web und Oracle Machine Learning SQL Notebooks zur Verfügung (*siehe Abbildung 3*).

### JDBC

Der Zugriff auf die Datenbanken in der Cloud erfolgt immer über eine sichere SSL-Verbindung (TLSv1.2) und mit einem Zertifikat. Java-Applikationen benötigen daher entweder einen Java Key Store (JKS) oder das sogenannte Oracle Wallet, um sich zu verbinden. Nach Provisionierung einer Autonomous Database kann die Wallet-Datei (Wallet\_<Datenbankname>.zip) über die Administrationskonsole des Datenbankdienstes heruntergeladen werden. Dabei wird ein Passwort für einige Komponenten des Wallet vergeben. Das Wallet ist ein Zip-Archiv mit folgendem Inhalt:

- tnsnames.ora – Verbindungsdeskriptoren der Datenbank
- sqlnet.ora – Client-seitige SQL\*Net-Konfiguration
- ewallet.p12 – passwortgeschütztes Wallet mit dem Zugriffszerifikat
- cwallet.sso – Single-Sign-on-Datei für die automatisierte Arbeit mit dem Wallet
- keystore.jks – passwortgeschützter Java Keystore mit dem öffentlichen RSA-2048-Schlüssel
- truststore.jks – passwortgeschützter Java Truststore mit den Zertifikaten der vertrauenswürdigen Server
- ojdbc.properties – JDBC-Verbindungsparameter, verweist unter Nutzung der Umgebungsvariable TNS\_ADMIN auf den Speicherort des Wallet

Da das Wallet zusammen mit einem Benutzer und Passwort den Zugang zu einer Datenbank ermöglicht, sollte es immer an einem sicheren Ort und nur mit eingeschränkten Zugriffsrechten (zum Beispiel Dateiberechtigung 600 unter Linux) abgelegt werden. Natürlich sollte es auch niemals auf unsicherem Wege (beispielsweise per öffentlicher E-Mail) und immer getrennt von seinem Zugriffspasswort versendet werden. Erweiterungen in den JDBC-Treibern der Version 19c (19.3) und 18c (18.3) vereinfachen die Benutzung der Wallet-Dateien. Für die weitere Verwendung packt man das Wallet-Archiv in einem Verzeichnis nach Wahl aus und setzt die Umgebungsvariable TNS\_ADMIN auf das Wallet-Verzeichnis: \$ export TNS\_ADMIN=<WALLET\_VERZEICHNIS>.

Die neuesten JDBC-Treiber sind auf der Oracle-Website [4] zu finden. Passend zum verwendeten JDK lädt man hier die neuesten Treiber herunter. Wenn eine JDK-Version kleiner als JDK8u162 (JDK 8, Update 162) verwendet wird, muss man zusätzlich die „Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy Files 8“-Dateien herunterladen [5]. Mit JDK11, JDK10 und JDK9 ist das nicht nötig. Folgende Dateien werden benötigt:

The screenshot shows the Oracle Cloud interface for creating an Autonomous Database. It includes fields for compartment selection, display name, database name, and configuration for workload type (Data Warehouse or Transaction Processing) and deployment type (Serverless). It also includes options for Always Free configuration and storage settings.

Abbildung 2: Einfache Provisionierung einer Autonomous Database über eine einzige Parametrisierungsseite (Quelle: Marcel Amende)

- ojdbc10.jar (zertifiziert mit JDK10) bzw. ojdbc8.jar (JDK8) – die JDBC-Treiber
- ucp.jar – der Universal Connection Pool
- oraclepkj.jar, osdt\_core.jar und osdt\_cert.jar – für die Verwendung des Wallet

Für den Aufbau einer JDBC-Verbindung wird das Objekt OracleDataSource genutzt (siehe Listing 1).

Der TNS-Alias setzt sich dabei aus Datenbanknamen und dem Namen einer Regel für die Verwaltung der Datenbankressourcen zusammen. Eine solche Regel definiert ein Laufzeitlimit für Datenbankabfragen, ein IO-Limit und das Verhältnis der möglichen CPU- und IO-Nutzung. Die Regeln sind über die Service-Konsole anpassbar. Vordefinierte Namen sind LOW, MEDIUM, HIGH, TP und TPURGENT [6]. Die gültigen Aliasse werden in der Datei „tnsnames.ora“ im Wallet-Verzeichnis verwaltet. Für eine Datenbank namens „DBORCL“ lautet der Alias für das Profil HIGH beispielsweise dborcl\_high. Die JDBC-Verbindung lässt sich nun wie gewohnt verwenden (siehe Listing 2).

Der komplette Beispielcode ist auf GitHub zu finden [7]. Zum Ausführen des Beispiels braucht man zusätzlich lediglich ein VM-Argument, das auf das Wallet-Verzeichnis zeigt (siehe Listing 3).

The screenshot shows the Oracle Cloud Work Requests page for a successful work request. It details the creation of an Autonomous Database with specific parameters and timestamps. Below this, a log messages table lists the steps taken during the database creation process.

Abbildung 3: Erfolgreiche Provisionierung einer Autonomous Database in unter drei Minuten (Quelle: Marcel Amende)

```
ods = new OracleDataSource();
ods.setURL("jdbc:oracle:thin:@<TNS_ALIAS>");
ods.setUser("<DB_BENUTZER>");
ods.setPassword("<DB_PASSWORT>");
OracleConnection con = (OracleConnection) ods.getConnection();
```

Listing 1

```

Statement s = connection.createStatement();
ResultSet rs = statement.executeQuery("select name from emp");

```

*Listing 2*

```
$ java -Doracle.net.wallet_location=<WALLET_VERZEICHNIS> ... autonomous.jdbc.SimpleJDBCTest
```

*Listing 3*

```

PoolDataSource pds = PoolDataSourceFactory.getPoolDataSource();
pds.setConnectionFactoryClassName("oracle.jdbc.pool.OracleDataSource");
pds.setURL("jdbc:oracle:thin:@<TNS_ALIAS>");
pds.setUser("<DB_BENUTZER>");
pds.setPassword("<DB_PASSWORT>");
pds.setConnectionPoolName("JDBC_UCP_POOL");
pds.setInitialPoolSize(3);
pds.setMinPoolSize(3);
pds.setMaxPoolSize(5);
pds.setTimeoutCheckInterval(5); //in Sekunden
pds.setInactiveConnectionTimeout(20); //in Sekunden
Connection con = pds.getConnection();

```

*Listing 4*

```

<beans ...>
  <bean id="dataSource" class="oracle.ucp.jdbc.PoolDataSourceImpl">
    <property name="connectionFactoryClassName" value="oracle.jdbc.pool.OracleDataSource" />
    <property name="URL" value="jdbc:oracle:thin:@<TNS_ALIAS>" />
    <property name="user" value="" />
    <property name="password" value="" />
    <property name="maxPoolSize" value="5" />
    <property name="initialPoolSize" value="1" />
  </bean>
  ...
</beans>

```

*Listing 5*

```

...
@ImportResource({ "classpath*:AppConfig.xml" })
public class MyApplication {
  ...

```

*Listing 6*

## Universal Connection Pool

Für eine skalierbare Anwendung sollte man einen JDBC Connection Pool verwenden. Der Oracle-JDBC-Treiber bietet hierfür den „Universal Connection Pool“ (UCP), den man über die ucp.jar-Bibliothek einbinden kann. Ebenfalls auf GitHub [7] findet sich im Package `autonomous.jdbc.ucp` eine Beispielimplementierung als Singleton (siehe Listing 4).

## Microservices mit Spring Boot

Spring Boot ist das derzeit wohl beliebteste und gleichzeitig bewährteste Framework für die Entwicklung von Java-Applikationen in der Cloud. Es ist Open Source [8], wird aber von der Firma Pivotal [9] massiv kommerziell unterstützt. Es eignet sich für die schnelle Erstellung von Microservices, ist aber auch mächtig genug für die Entwicklung modularer Monolithen. Natürlich unterstützt es auch

JDBC-Verbindungen zur Oracle Autonomous Database und den Universal Connection Pool.

Zusätzlich zu den aufgeführten JDBC-Bibliotheken muss man in einem Spring-Boot-Projekt folgende Spring- (hier exemplarisch in der Version 4.3.4 [10]) und Spring-Boot-Bibliotheken (Version 1.4.2 [11]) einbinden:

- `spring-boot-1.4.2.jar`
- `commons-logging-1.2.jar`
- `spring-aop-4.3.4.RELEASE.jar`
- `spring-beans-4.3.4.RELEASE.jar`
- `spring-boot-1.4.2.RELEASE.jar`
- `spring-context-4.3.4.RELEASE.jar`
- `spring-core-4.3.4.RELEASE.jar`
- `spring-expression-4.3.4.RELEASE.jar`
- `spring-jdbc-4.3.4.RELEASE.jar`
- `spring-tx-4.3.4.RELEASE.jar`

Die Verbindungsparameter der JDBC-Datenquelle werden in der XML-Datei für die Spring-Applikationskonfiguration angegeben (siehe Listing 5). Die Konfigurationsdatei wird wiederum über eine

```

/* File main.js - start */
const oracledb = require('oracledb');

async function run() {
  let pool;

  try {
    pool = await oracledb.createPool({
      user: "<DB_USER>",
      password: "<DB_PASSWORD>",
      connectString: "<TNS_ALIAS>"
    });
    const con = await pool.getConnection();
    const res = await con.execute("select sysdate from dual");
    console.log(res.rows[0]);
  } catch (err) {console.error(err);}
}

run();
/* File main.js - end */

```

*Listing 7*

```
$ node main.js
[ 2019-09-19T13:08:08.000Z ]
```

*Listing 8*

@ImportResource-Annotation der Main-Klasse der Applikation referenziert (*siehe Listing 6*). Die komplette, lauffähige Spring-Applikation ist ebenfalls auf GitHub zu sehen [[7](#)].

## JavaScript mit Node.js

Für eine Datenbankapplikation in JavaScript und die Ausführung in Node.js muss man drei Dinge vorbereiten:

1. Setzen der Umgebungsvariable TNS\_ADMIN auf das Wallet-Verzeichnis: \$ set TNS\_ADMIN=<WALLET\_VERZEICHNIS>
2. Herunterladen des Basic- oder Basic-Light-Pakets des Oracle Instant Client [12]. Nach dem Auspacken beziehungsweise Instal-

lieren nimmt man das entstandene Verzeichnis in den Suchpfad auf: \$ set PATH=<INSTANT\_CLIENT\_VERZEICHNIS>

3. Installieren des Oracle-Datenbanktreibers mit dem Node-Paketmanager: \$ npm install oracledb

Ein minimales JavaScript-Programm „main.js“ sieht dann unter Verwendung eines Connection Pool wie in *Listing 7* abgebildet aus. Es handelt sich dabei natürlich um ein eigenständiges Programm, keinen Web-Service. Die Verwendung eines Connection Pool ist daher exemplarisch zu sehen. Bei Ausführung gibt es die aktuelle Systemzeit der Datenbank aus (*siehe Listing 8*).

## RESTful Services und SODA

Natürlich gibt es Alternativen zu Datenbankabfragen über Datenbanktreiber. Oft ist es einfacher, die Datenbank gleich als RESTful Service zu konsumieren. In der Oracle Autonomous Database stehen dafür die „RESTful Data Service“ (ORDS) und mit dem „Simple Oracle Document Access“ (SODA) eine API-Abfrage im NoSQL-Stil bereit.

```
ORDS.ENABLE_SCHEMA(p_enabled => TRUE,
  p_schema => '<DB_SCHEMA>',
  p_url_mapping_type => 'BASE_PATH',
  p_url_mapping_pattern => '<REL_URL_PFAD>',
  p_auto_rest_auth => FALSE);
```

*Listing 9*

```
ORDS.ENABLE_OBJECT(p_enabled => TRUE,
  p_schema => '<DB_SCHEMA>',
  p_object => '<DB_TABELLE>',
  p_object_type => 'TABLE',
  p_object_alias => '<SERVICE_NAME>',
  p_auto_rest_auth => FALSE);
```

*Listing 10*

```
https://<RANDOM>-<DB_NAME>.adb.<DATACENTER_NAME>.oraclecloudapps.com/ords/< REL_URL_PFAD >/<SERVICE_NAME>/
```

*Listing 11*

Ein Datenbankschema kann man mit einem einfachen PL/SQL-Prozedurauftruf unter einem relativen Pfad für REST Services vorbereiten (*siehe Listing 9*).

Nun kann eine Tabelle des Schemas als RESTful Service bereitgestellt werden, der CRUD-Operationen in SQL DML und die Daten in JSON-Formate übersetzt (*siehe Listing 10*). Dabei ist Vorsicht geboten, denn daraus resultiert ein öffentlich aufrufbarer REST Service unter der in *Listing 11* angegebenen URL.

REST-Ressourcen lassen sich durch die Vergabe von Privilegien sehr einfach mit einer OAuth2- oder BASIC-Authentifizierung versehen. Die Operationen zur Erstellung eines REST Service können mit dem SQL Developer auch per Knopfdruck ausgeführt werden.

## Zusammenfassung

Für den (Java-)Entwickler ändert sich mit der Oracle Autonomous Database gegenüber der Nutzung der klassischen Datenbank glücklicherweise wenig. Jede Kommunikation mit der Datenbank ist aber zwingend verschlüsselt und signiert. Der organisatorische Wert der autonomen Datenbank kann hingegen immens sein. Bei einer agilen Softwareentwicklung braucht man praktisch auf der Stelle die nötigen Ressourcen, ohne sich auf unabsehbare Kosten und Aufwände einzulassen. Dabei kann die Oracle Autonomous Database einen wichtigen Beitrag leisten. Als „Always Free“-Dienst steht sie in circa drei Minuten kostenfrei für die Umsetzung neuer Ideen bereit. Darüber hinaus zahlt man flexibel nur die tatsächliche Nutzung. Im Erfolgsfall kann man im laufenden Betrieb nahezu beliebig skalieren und hat vom Projektstart bis zum produktiven Einsatz immer ein sicheres, absolut wartungsfreies und hochverfügbares System. Bei höchstem Bedarf an Sicherheit und Datenschutz sogar in einem kompletten privaten Subnetz mit eigenen Schlüsseln. Für den Java-Entwickler sollte die autonome Datenbank daher einen Blick wert sein. Auch die System- und Datenbankadministratoren werden – befreit von Routinearbeiten – anspruchsvollere und interessantere Aufgaben im Entwicklunguprojekt übernehmen können.

## Quellen

- [1] <https://www.oracle.com/database/autonomous-database.html>
- [2] <https://cloud.oracle.com/tryit>
- [3] <https://www.oracle.com/cloud/free/>
- [4] <https://www.oracle.com/database/technologies/appdev/jdbc-ucp-19c-downloads.html>
- [5] <https://www.oracle.com/technetwork/java/javase/downloads/jce8-download-2133166.html>
- [6] <http://bit.ly/servicenames-doc>
- [7] <https://github.com/ora-autonomous/jdbc>
- [8] <https://github.com/spring-projects/spring-boot>
- [9] <https://spring.io/projects/spring-boot>
- [10] <https://repo.spring.io/release/org/springframework/spring/4.3.4.RELEASE/>
- [11] <https://jar-download.com/artifacts/org.springframework.boot/spring-boot/1.4.2.RELEASE/source-code>
- [12] <https://www.oracle.com/database/technologies/instant-client/downloads.html>
- [13] <https://www.oracle.com/tools/downloads/sqldev-v192-downloads.html>



**Marcel Amende**

Oracle

[Marcel.Amende@oracle.com](mailto:Marcel.Amende@oracle.com)

Geboren als Ingenieur, aufgewachsen bei Oracle, zuhause in der Cloud. Als Solution Engineer repräsentiert Marcel die „Cloud Native“-Entwicklergemeinde von Oracle, die aus den Diensten der Oracle Cloud kreative Kundenlösungen gestaltet.

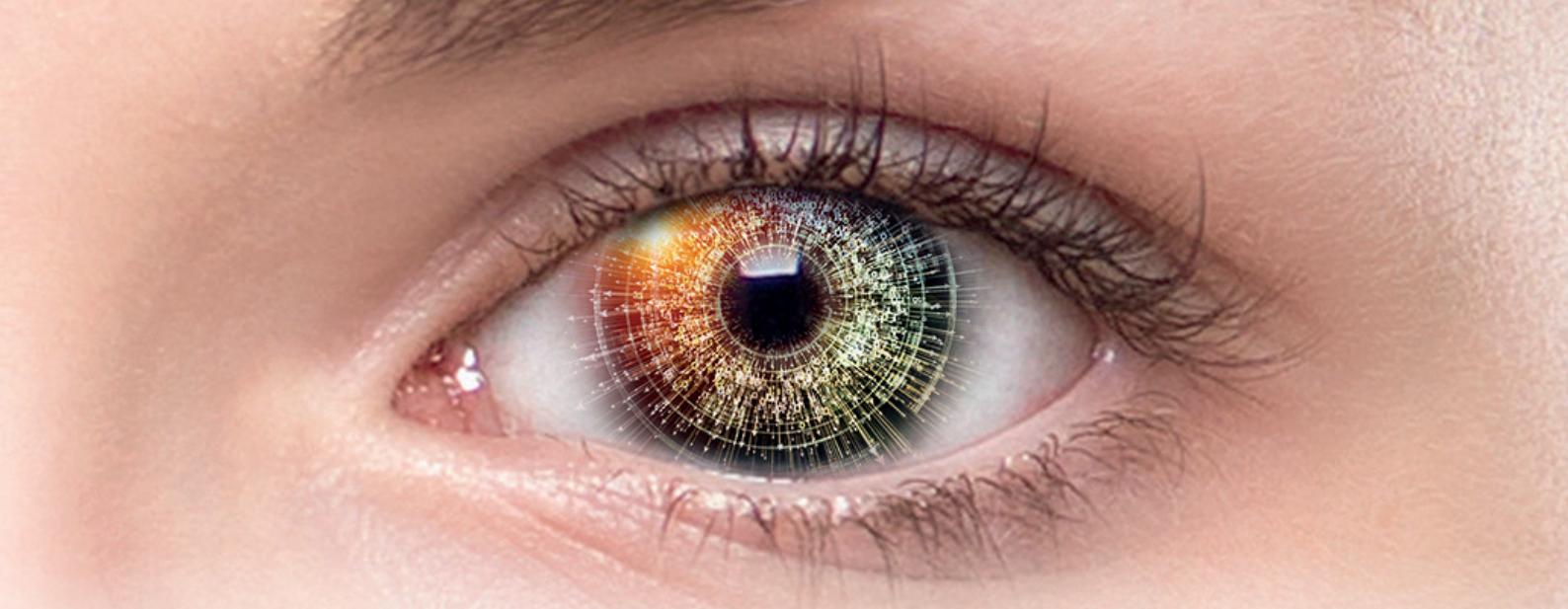


**Kersten Mebus**

Oracle

[Kersten.Mebus@oracle.com](mailto:Kersten.Mebus@oracle.com)

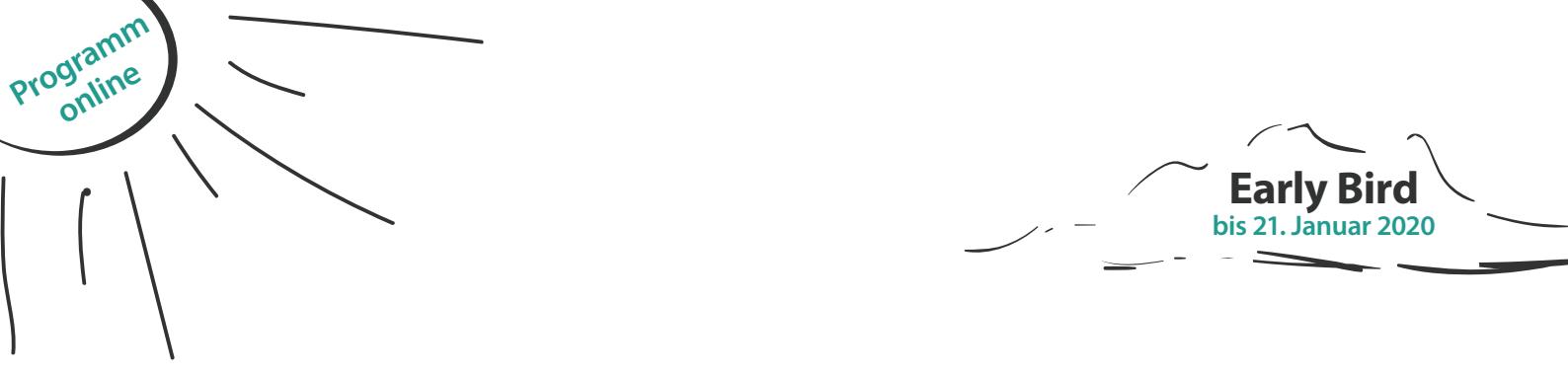
Kersten Mebus ist schon seit seiner Schulzeit von Technologie begeistert. Sein Motto ist: Alles, was digitalisiert werden kann, wird digitalisiert. Als Cloud-Experte und Solution Engineer für Integration/Prozesse, Anwendungsentwicklung und Datenbanken erarbeitet er zusammen mit dem Kunden kundenspezifische Lösungen für die Cloud.



# Data Analytics 2020

28. & 29. April | in Düsseldorf





**Early Bird**  
bis 21. Januar 2020

# JavaLand

*2020*

**17. - 19. März 2020 in Brühl bei Köln**

**Ab sofort Ticket & Hotel buchen!**

[www.javaland.eu](http://www.javaland.eu)

