

Performance, Scalability, and High-Availability with JDBC and UCP in Oracle Database 12c Release 2 (12.2.0.1)

Oracle JDBC and Universal Connection Pool (UCP)

ORACLE WHITE PAPER MARCH 2017



Disclaimer

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, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle



Contents

Disclaimer	1
Introduction	3
Java Connectivity to Database on Cloud	3
Oracle Database Cloud Service (DBCS)	4
Oracle Database Exadata Express Cloud Service	4
JDBC and UCP - Support for Latest Java Standards	4
Java Performance with Oracle JDBC and UCP	6
De-prioritization of database nodes	6
UCP Connection Health Frequency Check	7
Redesign of the connection pool	7
JDBC and UCP Scalability	7
Sharding Support in JDBC and UCP	8
Shared Pool for Multitenant Datasource	9
DRCP Enhancements	11
Java High-Availability with Oracle JDBC and UCP	11
Driver support for Fast Application Notification (FAN)	11
Application Continuity (AC) for XA DataSource	12
Transaction Guard (TG) for XA DataSource	12
Java Applications Security with Oracle JDBC and UCP	13
Manageability for Java Applications with Oracle JDBC and UCP	13
Conclusion	14

Introduction

Oracle Database 12c Release 2 (12.2.0.1), the latest generation of the world's most popular database is now available in the Oracle Cloud, as well as on premise and offers solutions to the problems faced by today's web/mobile applications with respect to performance, scalability, high availability, and security.

Some of the known challenges in achieving a quality of services are, **scaling** customer facing applications to thousands of users without impacting the **availability** and **manageability**; **optimizing** the database resources in a multi-tenant architecture; hiding the planned maintenance without making any application code change etc.,

Oracle Database 12.2.0.1 release addresses these challenges and also offers many useful features for Java developers. A glimpse of the solutions is, 12.2.0.1 supports **database sharding** architecture that helps applications scale very easily and provides an infrastructure that eliminates single points of failure; **Multi-tenant architecture** leverages multi-faceted Universal Connection Pool (UCP) to optimize the usage of expensive database resources; JDBC driver is enhanced to support Fast Application Notification (FAN) events and hides the planned maintenance.

This whitepaper provides more details on how these solutions can be implemented by Java applications using Oracle Database 12c Release 2 (12.2.0.1), Oracle JDBC driver (**ojdbc8.jar**)¹, and Universal Connection Pool (**ucp.jar**). These new solutions will enable Java developers, Java architect, or an Oracle DBA to improve performance, scalability, availability, security, and manageability of the enterprise Java applications.

For more details refer to JDBC Developer's Guide² and UCP Developer's Guide³.

Java Connectivity to Database on Cloud

The Oracle Database on Cloud gives access to a single Oracle Database with full access to its features and operations. Each one of the cloud offerings possesses its own security requirement.

¹ 12.2.0.1 JDBC and UCP Download page @ <http://www.oracle.com/technetwork/database/features/jdbc/jdbc-ucp-122-3110062.html>

² JDBC Developer's Guide @ <https://docs.oracle.com/database/122/JJDBC/toc.htm>

³ Universal Connection Pool Developer's Guide @ <https://docs.oracle.com/database/122/JJUCP/toc.htm>

The following sections aim at covering some of the cloud offerings addressing JDBC connectivity and security requirements of these database cloud services.

Oracle Database Cloud Service (DBCS)

Oracle Database Cloud Service (DBCS)⁴ provides the full power of the Oracle Database in the cloud suitable for enterprise applications. Oracle Net Services can be used to connect to the database cloud service. As a security requirement, the port 1521 is blocked when the database is created. Make sure to unblock the port 1521 before attempting to connect to the database. Also, DBCS allows SSH access to the compute node which provides full control to the database.

Follow the instructions on “**Using Java Applications & IDEs with Oracle Database Cloud Service (DBCS)**”⁵ for more information.

Oracle Database Exadata Express Cloud Service

Oracle Exadata Express Cloud Service (EECS)⁶ is a fully managed database suitable for small to medium sized data. It leverages pluggable database (PDB) containerization technology, provisioned within minutes, and runs on Oracle Exadata engineered systems.

As a security requirement, it mandates a secure connection using TLSv1.2 protocol. TLSv1.2 is enabled by default in the 12.2.0.1 JDBC driver. However, when using with 12.1.0.2, a separate property `oracle.net.ssl_version=1.2` should be set as either a system property or as a connection property. The JDBC drivers utilize Java Key Store (JKS) files for secure connections. The JKS files (`keystore.jks` and `truststore.jks`) along with the required configuration files (`tnsnames.ora`) must be downloaded from the service console.

The detailed instructions to connect to Oracle Database Exadata Express Cloud Service using JDBC driver and UCP are present on the OTN page “**Java/UCP Connectivity for Java SE, Java EE Containers, and Java Cloud Service**”⁷.

Java Developer tools such as JDeveloper, Eclipse, IntelliJ, and NetBeans can also connect seamlessly to Oracle Database Exadata Express Cloud Service and browse schemas. The instructions are on the following OTN page “**JDBC/UCP Connectivity for Java IDEs (JDeveloper, NetBeans, Eclipse, IntelliJ)**”⁸

JDBC and UCP - Support for Latest Java Standards

Java SE 8 Support and JDBC 4.2

⁴ Oracle Database Cloud Service Guide @ http://docs.oracle.com/cloud/latest/dbcs_dbaas/index.html

⁵ JDBC instructions to connect to Oracle Database Cloud Service @ <http://www.oracle.com/technetwork/database/application-development/jdbc/documentation/default-3396167.html>

⁶ Oracle Database Exadata Express Cloud Service Guide @ <http://docs.oracle.com/cloud/latest/exadataexpress-cloud/CSDBP/toc.htm>

⁷ JDBC/UCP Connectivity for Java SE, Java EE Containers, and Java Cloud Service @ <http://www.oracle.com/technetwork/database/application-development/jdbc-eecloud-3089380.html>

⁸ JDBC/UCP Connectivity for Java IDEs (JDeveloper, NetBeans, Eclipse, IntelliJ) @ <http://www.oracle.com/technetwork/database/application-development/jdbc/index-3093936.html>

Oracle Database 12c Release 2 (12.2.0.1) JDBC driver and UCP support the new features introduced as part of JDBC 4.2⁹ which is a part of Java SE 8.

JDBC driver and UCP have leveraged lambdas and builder pattern of Java SE 8. Some of the new APIs use the builder pattern. Refer to the Sharding and Multitenant section for a sample of the builder pattern .

Some of the new methods related to JDBC 4.2 are highlighted here.

These new methods can be used when you expect the number of rows returned to exceed `INTEGER.MAX_VALUE`.

- `executeLargeUpdate(String sql)`
- `executeLargeUpdate(String sql, int[] columnIndexes)`
- `executeLargeUpdate(String sql, String[] columnNames)`
- `getLargeMaxRows()`
- `getLargeUpdateCount()`
- `setLargeMaxRows(long max)`

Also, `setObject()` method is supported in addition to `getObject()` already available in DB 12.1 JDBC.

Example: the following code fragment shows how `setObject(...)` and `getObject(...)` are used.

```
// Statement and ResultSet are AutoCloseable and closed automatically.
try (PreparedStatement preparedStatement = connection.prepareStatement("SELECT
FIRST_NAME, LAST_NAME FROM EMPLOYEES WHERE EMPNO = ?"))
{
    preparedStatement.setObject(1, Integer.valueOf(id));
    try (ResultSet resultSet = preparedStatement.executeQuery()) {
        System.out.println("FIRST_NAME" + " " + "LAST_NAME");
        System.out.println("-----");
        while (resultSet.next())
            System.out.println(resultSet.getObject(1, String.class) + " "
                + resultSet.getObject(1, String.class) + " ");
    }
}
```

JDBC and UCP New Data Types Support

In Oracle database release 12.2.0.1, JDBC expose a new PLSQL Boolean datatype for Java programmers.

PLSQL Boolean Datatype Support

The PLSQL Boolean data type takes true or false values and is different from the SQL boolean, which is a number (0/1). The 12.2.0.1 JDBC driver supports PLSQL Boolean and can be used as a bind value into a PLSQL block. `OracleTypes.PLSQL_BOOLEAN` can be used for IN, OUT,

⁹ JDBC 4.2 specification @ https://docs.oracle.com/javase/8/docs/technotes/guides/jdbc/jdbc_42.html

or IN OUT parameters.

Example:

```
// Shows how a PLSQL procedure is used to find if the
// first number is bigger than the second number passed as input.
String sql = "{? = call function_bigger (?, ?)}";
CallableStatement cstmt = conn.prepareCall(sql);
// Test boolean as output in a function
cstmt.registerOutParameter(1, OracleTypes.PLSQL_BOOLEAN);
// Test values for true
cstmt.setInt(2, 9);
cstmt.setInt(3, 7);
// Execute the callable statement
cstmt.execute();
// Returns a boolean (true/false)
boolean biggerThan = cstmt.getBoolean(1);
cstmt.close();
```

Java Performance with Oracle JDBC and UCP

In Oracle Database 12c Release 2 (12.2.0.1), new performance enhancements are built into the JDBC driver and UCP. The enhancements are: de-prioritization of database nodes, redesign of UCP, and enhancements at the network and connection pool levels.

Network Compression Over WAN

In a constrained network bandwidth environment, i.e., Wide Area Network (WAN) sending large amounts of data, with a good performance can be achieved by enabling data compression. Compression is achieved by compressing data in the session data unit (SDU) buffers thus reducing the time required to transmit a SQL query and the result across the network. Compressed data uses less bandwidth; the compression process is transparent to the application layer.

Example:


```
OracleDataSource ds = new OracleDataSource();
Properties prop = new Properties();
prop.setProperty("user", "user1");
prop.setProperty("password", <password>);

// Enabling Network Compression
prop.setProperty("oracle.net.networkCompression", "on");
//Optional configuration for setting the client compression threshold.
prop.setProperty("oracle.net.networkCompressionThreshold", "1024");

ds.setConnectionProperties(prop);
ds.setURL(url);
Connection conn = ds.getConnection();
```

De-prioritization of database nodes

In Oracle Database 12c Release 2 (12.2.0.1), the JDBC drivers are smarter and capable of de-prioritizing instances which are down during connection establishment. For example, if there are



three instances A, B, C; if instance A is down, then connections are allocated first from instances B and C, and at last resort, attempt is made to connect to instance A. The de-prioritization is active until the expiration period, which is set to 10 minutes by default. Upon the expiration period, instance A is no longer deprioritized and connections are allocated from all three instances equally. The default expiry time can be overridden using a system property, `oracle.net.DOWN_HOSTS_TIMEOUT`.

UCP Connection Health Frequency Check

The Universal Connection Pool furnished the `setValidateConnectionOnBorrow(Boolean) property` which instructs it to validate connections during checkout. Connection Validation is disabled by default. Though, this is an important feature, it could increase the time taken to checkout a connection. So, in order to improve the performance and minimize the time taken for checking out a connection, a new UCP property `setSecondsToTrustIdleConnection(int)` can be used. When this property is set with an appropriate value, the *recently-used* and *recently-tested* database connections are trusted and connection validation is skipped. This way, it reduces the frequency of connection validation and improves the performance.

Redesign of the connection pool

Wait Free UCP:

Performance is crucial in a highly concurrent environment i.e., an environment where a lot of threads are involved. UCP has been re-designed to achieve faster connection checkouts in such an environment. Some of the key connection pooling operations have been fine-tuned so that connection borrowing is less expensive. These performance improvements are available as part of 12.1.0.2. Our internal benchmark shows that the new improved wait free UCP has a significant performance gain compared to the older versions of UCP. Check out the JavaOne presentation¹⁰ for more details.

Multi-dimensional tree search:

Connection requests may sometimes include additional properties such as instance name, service name, database name etc., Associating these properties/labels to the connections avoids time and cost for the connection re-initialization. Faster retrieval of a specific connection with labels is a challenging problem for high performance scalable systems. Regular Java collections such as maps are not suitable once UCP searches with several orthogonal keys. To solve this, the searching mechanism has been redesigned for better performance and is based on KD-Trees which is known in a number of graphical, AI, machine learning, and scientific applications. The optimized connection pool has consistently displayed faster borrowing of connections even when the pool size is increased.

JDBC and UCP Scalability

Scalability is a key requirement for Java applications as we see an exponential growth in number of customers and the data. Customers should choose an appropriate architecture to handle huge volumes of data that is easily manageable and operationally cost effective.

¹⁰ Wait Free UCP from JavaOne @ <http://www.oracle.com/technetwork/database/application-development/con2158-javaoneucpsession-2769404.pdf>

In 12.2.0.1, the JDBC driver and UCP both support Oracle Sharding which is the solution to achieve linear database scalability and better manageability.

In 12.2.0.1, UCP, a client side connection pool, furnishes a “Shared pool” in a multitenant environment thus optimizes the database resources. Similar to UCP, Database Resident Connection Pool (DRCP) which is the server side connection pool also brings new enhancements to share proxy connections and supports multiple labels/tags for faster connection retrievals.

Sharding Support in JDBC and UCP

For modern web applications where the data, transactions, and users are increasing every day, *Oracle Sharding* offers scaling without affecting the performance and availability.

Java applications using Sharded databases must use the new Sharding APIs. Shards are created based on a single or multiple sharding keys with or without a super sharding key. JDBC and UCP applications must build the sharding key and optionally the super sharding key then use these to request a connection to a specific shard.

Universal Connection Pool (UCP) offers an efficient support for Sharding. UCP caches the shard topology and eliminates the need to go through shard director for every connection request. This way, UCP acts as a shard director and provides a fast path to the sharded database. UCP also handles shard or chunk life cycle management transparently by subscribing to events related to chunk movements.

A glimpse of the sharding APIs is present in this code snippet. Please refer to the JDBC Developer’s Guide¹¹ and UCP Developer’s Guide¹² for more details.

Example:

```
// Get the PoolDataSource for UCP
PoolDataSource pds = PoolDataSourceFactory.getPoolDataSource();
Connection connection;
// Set the connection factory first before all other properties
pds.setConnectionFactoryClassName("oracle.jdbc.pool.OracleDataSource");
pds.setURL(DB_URL);
pds.setUser(DB_USER);
pds.setPassword(DB_PASSWORD);
pds.setConnectionPoolName("UCP_SHARDING_POOL");
// Set the initial number of connections created when UCP is started.
pds.setInitialPoolSize(5);
// Set the minimum number of connections maintained by UCP at runtime.
pds.setMinPoolSize(5);
// Set the maximum number of connections allowed on the connection pool.
pds.setMaxPoolSize(20);
// set max connection per shard to ensure fair use of the pool
pds.setMaxConnectionsPerShard(10);
// Build the Sharding key by passing the email id
```

¹¹ JDBC Developer’s Guide @ <https://docs.oracle.com/database/122/JJDBC/toc.htm>

¹² Universal Connection Pool Developer’s Guide @ <https://docs.oracle.com/database/122/JJUCP/toc.htm>

```

OracleShardingKey shardKey = pds.createShardingKeyBuilder()
                                .subkey(email, OracleType.VARCHAR2)
                                .build();
// Get the connection by passing the sharding key
connection = pds.createConnectionBuilder()
              .shardingKey(shardKey)
              .build();
// Perform a database operation
doSQLWork(connection, email);

```

Shared Pool for Multitenant Datasource

In Oracle Database 12c Release 2 (12.2.0.1), UCP allows multitenant Java applications accessing the Oracle database multitenant architecture to share a common pool of connections. UCP allows connection to be reused or repurposed across tenants (PDBs) when there is no free connection, thus improving the performance and scalability.

There are two use cases or techniques supported by UCP:

- (i) Single datasource shared by all tenants
- (ii) One datasource per tenant.

Usecase 1: Single DataSource shared by all tenants

With this configuration, multiple tenants use the common data source. A shared pool is used to connect to each of the tenants using different service names for each tenant. Use the new APIs to get connections from a shared pool as shown below. Note that a common user (example: `c##commonuser`) should be used to take advantage of the shared pool.

Example:

```

PoolDataSource multiTenantDS = PoolDataSourceFactory.getPoolDataSource();
...
// password enabled role for tenant-1
Properties tenant1Roles = new Properties();
tenant1Roles.put("tenant1-role", "tenant1-password");

//Create Connection to Tenant-1 and apply the tenant specific PDB roles.
Connection tenant1Connection =
    multiTenantDS.createConnectionBuilder()
                  .serviceName("tenant1Svc.oracle.com")
                  .pdbRoles(tenant1Roles)
                  .build();

// password enabled role for tenant-2
Properties tenant2Roles = new Properties();
tenant1Roles.put("tenant2-role", "tenant2-password");

//Create Connection to Tenant-2 and apply the tenant specific PDB roles.
Connection tenant2Connection =
    multiTenantDS.createConnectionBuilder()
                  .serviceName("tenant2Svc.oracle.com")

```

```
.pdbRoles(tenant2Roles)
.build();
```

Usecase 2: One Datasource per tenant

With this configuration, multitenant applications have separate data sources with a specific service name for each tenant and a common shared pool of connections. In this scenario, the configuration needs to be specified in an XML file.

Some of the pre-requisites required to use this feature are:

- (a) Common user (example: `c##commonuser`) for repurposing the connections.
- (b) An application service must be used and it should be homogeneous. Homogeneous services are the services that have the same settings for Application Continuity (AC), Transaction Guard (TG), DRCP etc., The switch service capability is used transparently for re-purposing the connections for homogeneous services.
- (c) UCP XML configuration file to specify the shared pool configurations such as `connection-pool-name`, `shared=true`, `data-source`, `max-connections-per-service` etc.,

Refer to the “JDBC and UCP Manageability“ for more details on using a UCP XML Configuration file.

Make sure to use the property “`max-connections-per-service`”, which caps the number of connections that can be checked out by a tenant, thereby ensuring a fair distribution of connections among tenants.

There are new methods introduced to establish a connection, reconfigure the shared pool, and reconfigure the datasource as shown below.

```
// Get the first datasource instance "pds1" from the UCP config XML.
PoolDataSource pds1 = PoolDataSourceFactory.getPoolDataSource("pds1");
Connection pds1Conn = pds1.getConnection();

// Get the second datasource instance "pds2" from the UCP config XML.
PoolDataSource pds2 = PoolDataSourceFactory.getPoolDataSource("pds2");
Connection pds2Conn = pds2.getConnection();

PoolDataSource multiTenantDS = PoolDataSourceFactory.getPoolDataSource();
// Reconfigures the datasource properties
multiTenantDS.reconfigureDataSource(Properties prop)
// Reconfigure connection pool("pool1") using the new properties

Properties newPoolProps = new Properties();
newPoolProps.put("initialPoolSize", <newInitialPoolSizeValue>);
newPoolProps.put("maxPoolSize", <newMaxPoolSizeValue>);
UniversalConnectionPoolManager ucpMgr =
UniversalConnectionPoolManagerImpl.getUniversalConnectionPoolManager();
ucpMgr.reconfigureConnectionPool("pool1", newPoolProps);

// Configure a new datasource(pds3) to running pool
Properties dataSourceProps = new Properties();
dataSourceProps.put("serviceName", <serviceName>);
```

```
dataSourceProps.put("connectionPoolName", <poolName>);
dataSourceProps.put("dataSourceName", <dataSourceName>);
PoolDataSource pds3 =
    PoolDataSourceFactory.getPoolDataSource(dataSourceProps);
```

DRCP Enhancements

DRCP Proxy Sessions Sharing

Starting with Oracle Database 12c Release 2 (12.2.0.1), DRCP sessions belonging to the same database user but a different proxy user can now be shared amongst themselves. In the prior release, an idle session belonging to a proxy user could not be used by a different proxy user even though both proxy users map to the same database user. In this release, this restriction has been removed.

DRCP Multiple Tagging

Connection tagging is a mechanism where a tag is applied to a connection and that tag is used for retrieving the same connection. Connection tagging enhances the session pooling as you can retrieve specific sessions faster.

With Oracle Database 12c Release 2 (12.2.0.1), DRCP provides support for multiple tagging. Multiple properties can be associated to a connection and used while retrieving a connection with the same properties. Set the `oracle.jdbc.UseDRCPMultipleTag` connection property to `TRUE` for enabling this feature.

Java High-Availability with Oracle JDBC and UCP

Oracle Database aims at furnishing continuous service. Oracle Real Application Clusters (RAC), Data Guard (DG), Active Data Guard (ADG), the drivers, and connection pools are the building blocks for achieving such a goal. In 12.2.0.1, new capabilities have been added to enhance high availability.

Driver support for Fast Application Notification (FAN)

In Oracle Database 12c Release 2 (12.2.0.1), the JDBC driver supports Oracle RAC FAN events and thus, applications using third party connection pools can leverage the high availability feature through 12.2 JDBC driver.

There are three easy steps required for enabling FAN with the 12.2 JDBC driver.

- (1) Make sure to have **simplefan.jar** and **ons.jar** in the classpath along with **ojdbc8.jar**. All three jars must be from 12.2.0.1 version.
- (2) Set safe draining points, allowing the driver to safely close the connections.
 - (a) Third party connection pools should use the following safe draining APIs.

```
java.sql.Connection.isValid(int timeout)
oracle.jdbc.OracleConnection.pingDatabase()
oracle.jdbc.OracleConnection.pingDatabase(int timeout)
oracle.jdbc.OracleConnection.endRequest()
```

(b) For the standard JDBC and Oracle JDBC extension EXECUTE*** calls, the validation SQL must contain the SQL hint (`/*+ CLIENT_CONNECTION_VALIDATION */`) as the first non-comment token.

Application Continuity (AC) for XA DataSource

Application Continuity (AC) was introduced in Oracle Database 12c to enable transparent recovery of in-flight database work during an unplanned database instance outage. The outage can be related to system, communication, or hardware following a repair, or a configuration change; With AC, every database request is protected.

In Oracle Database 12c Release 2 (12.2.0.1), the JDBC driver and UCP extend the Application Continuity (AC) support for XA data sources (`javax.sql.XADataSource`) with local transactions.

Some points related to supporting XA datasource:

- (a) AC support for XA-capable data sources is limited to local transactions and local transactions that are promotable to global/XA transactions.
- (b) The replay feature is disabled whenever a connection participates in a global/XA transaction or is part of the XA operation.
- (c) Applications can still perform XA operations at runtime but, without AC protection
- (d) Switching from XA/global transactions to local transaction does not automatically enable AC.

Applications must use the XA replay driver `oracle.jdbc.replay.OracleXADataSource` to leverage this functionality. Check out the code sample in JDBC Developer's Guide for more details.

Transaction Guard (TG) for XA DataSource

In scenarios when there is a communication failure or unexpected failures during the COMMIT execution, a reliable determination of the transaction outcome is a challenge. Transaction Guard (TG) solves this problem by determining the outcome of the transaction in a guaranteed and scalable manner.

Every transaction is tagged with a Logical Transaction Identifier (LTXID), which can be used by the application upon a failure to check whether the transaction had committed or not. This way, Transaction Guard ensures that every transaction has at-most-once execution preventing applications from submitting duplicate transactions.

Starting from Oracle Database 12c Release 2 (12.2.0.1), Transaction Guard provides support for XA transactions for one-phase commit optimization, read-only optimization, and promotable XA. Transaction Guard with XA provides safe replay following recoverable outages for XA transactions. With the addition of XA support, Transaction Managers can now provide replay with Transaction Guard.

Fast Application Notification(FAN) APIs for High Availability

Do you want to take advantage of High Availability features of the Oracle Database and build more responsive applications in scenarios when you are not using any of the Oracle's gold standards i.e., Oracle Web Logic Service (WLS) with Active Grid Link (AGL) or Universal Connection Pool (UCP) but rather third-party connection pools, and non-Oracle Java containers? The solution is to

leverage the Oracle RAC Fast Application Notification (FAN) APIs¹³. Make sure to have **simplefan.jar** and **ons.jar** files in the class path.

The FAN events are the notifications sent by a cluster to inform the subscribers (connection pools, drivers) about FAN events at the service level or at the node level. The supported FAN events are Service UP, Service DOWN, Node UP, Node DOWN, Planned DOWN, Load Balancing Advisory.

A sample of the DOWN event payload is as shown below.

```
VERSION=1.0 event_type=INSTANCE service=myorclservice instance=INSTANCE1
database=myorcldb db_domain=us.oracle.com host=myorclhost status=DOWN
reason=FAILURE timestamp=2016-11-01 19:32:43 timezone=-08:00
```

Java Applications Security with Oracle JDBC and UCP

Oracle Database on cloud mandates high security for data protection. To enforce a tight security, the database connections to Oracle Database Exadata Express Cloud Service uses Java Key Store (JKS) files for SSL encryption and TLSv1.2 for enhanced security. In 12.2.0.1, TLSv1.2 is supported by default.

TLSv1.1 and TLSv1.2 Support

Java SE 7 and Java SE 8 support the following protocols, SSLv3, TLSv1, TLSv1.1, and TLSv1.2. Oracle JDBC Thin driver version 12.1.0.2 has been originally enhanced to support TLSv1.1 or TLSv1.2. Note that when you are using 12.1.0.2 JDBC driver, you will need to explicitly set a property `oracle.net.ssl_version=1.2` to use TLSv1.2. However, Oracle Database 12c Release 2 (12.2.0.1) JDBC driver does not require this property as TLSv1.2 is supported by default.

Manageability for Java Applications with Oracle JDBC and UCP

Applications must be easily manageable. In 12.2.0.1, UCP can be configured through XML configuration file and all the connection pool properties can be defined as XML attributes. Some of the new features such as “Shared Pool for multitenant architecture” mandates using the XML configuration file.

UCP XML Configuration File

With this new feature, Java containers have an alternate method of defining connection pool properties at one place. This eliminates the need to update or initialize the pool through the console. All pool properties can be defined in the XML file.

Note that, the actual path of XML configuration file for UCP should be provided through a system property `oracle.ucp.jdbc.xmlConfigFile="file:/user_directory/ucp.xml"`. The location of the initial XML configuration file should be specified as a URI. If you want to know the

¹³ Oracle FAN APIs @ <https://docs.oracle.com/database/122/JAFAN/toc.htm>

specific pool property required to create a connection pool, then refer to the `configuration.xsd` schema file included in the `ucp.jar` file.

Example:

```
<ucp-properties>
<connection-pool
connection-pool-name="pool1"
connection-factory-class-name="oracle.jdbc.pool.OracleDataSource"
url="jdbc:oracle:thin:@(DESCRIPTION=(ADDRESS=(PROTOCOL=tcp)(HOST=myhost)
(PORT=1521))(CONNECT_DATA=(SERVICE_NAME=myorclpdbservice)))"
user= "C##1"
password= "password"
initial-pool-size="10"
min-pool-size="2" max-pool-size="30"
max-connections-per-service="15" shared="true" >
<!-- First Datasource properties -->
<data-source data-source-name="pds1" service="pdb1" description="pdb1
data source" />
<!-- Second Datasource properties -->
<data-source data-source-name="pds2" service="pdb2" description="pdb2
data source" />
</connection-pool>
</ucp-properties>
```

Wider System Change Number (SCN)

The System Change Number (SCN) is an Oracle database internal clock used to set transaction timestamps. The SCN had a maximum limit of 6 bytes and could max out when there is an extremely high number of transactions. In order to handle the growth in transaction throughput and also to avoid exhaustion, the SCN has been widened from 6 bytes to 8 bytes. This is an internal change transparent to Java applications.





Conclusion

This whitepaper described the use cases and the new capabilities of JDBC driver and UCP that can be leveraged to address the challenges faced by web applications. These new capabilities include: Java connectivity to Oracle Database in the Cloud; JDBC support for latest Java standards and new data types; how and when to enable network compression over WAN; the new UCP performance enhancements; Using JDBC/UCP APIs in a sharding architecture; optimizing database connectivity in a multitenant environment; achieving planned maintenance without application code change ; Application Continuity support for XA datasources; enhanced coverage of FAN APIs; support for TLSv1.2 and TLSv1.1; using XML configuration file to create the pool for better manageability. Java applications can be powered with these features to achieve performance, scalability, availability, security, and manageability.



Oracle Corporation, World Headquarters **Worldwide Inquiries**
500 Oracle Parkway Phone: +1.650.506.7000
Redwood Shores, CA 94065, USA Fax: +1.650.506.7200


CONNECT WITH US

-  blogs.oracle.com/oracle
-  facebook.com/oracle
-  twitter.com/oracle
-  oracle.com

Integrated Cloud Applications & Platform Services

Copyright © 2016, Oracle and/or its affiliates. All rights reserved. This document is provided *for* information purposes only, and the contents hereof are subject to change without notice. This document is not warranted to be error-free, nor subject to any other warranties or conditions, whether expressed orally or implied in law, including implied warranties and conditions of merchantability or fitness for a particular purpose. We specifically disclaim any liability with respect to this document, and no contractual obligations are formed either directly or indirectly by this document. This document may not be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without our prior written permission.

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

 Oracle is committed to developing practices and products that help protect the environment