10

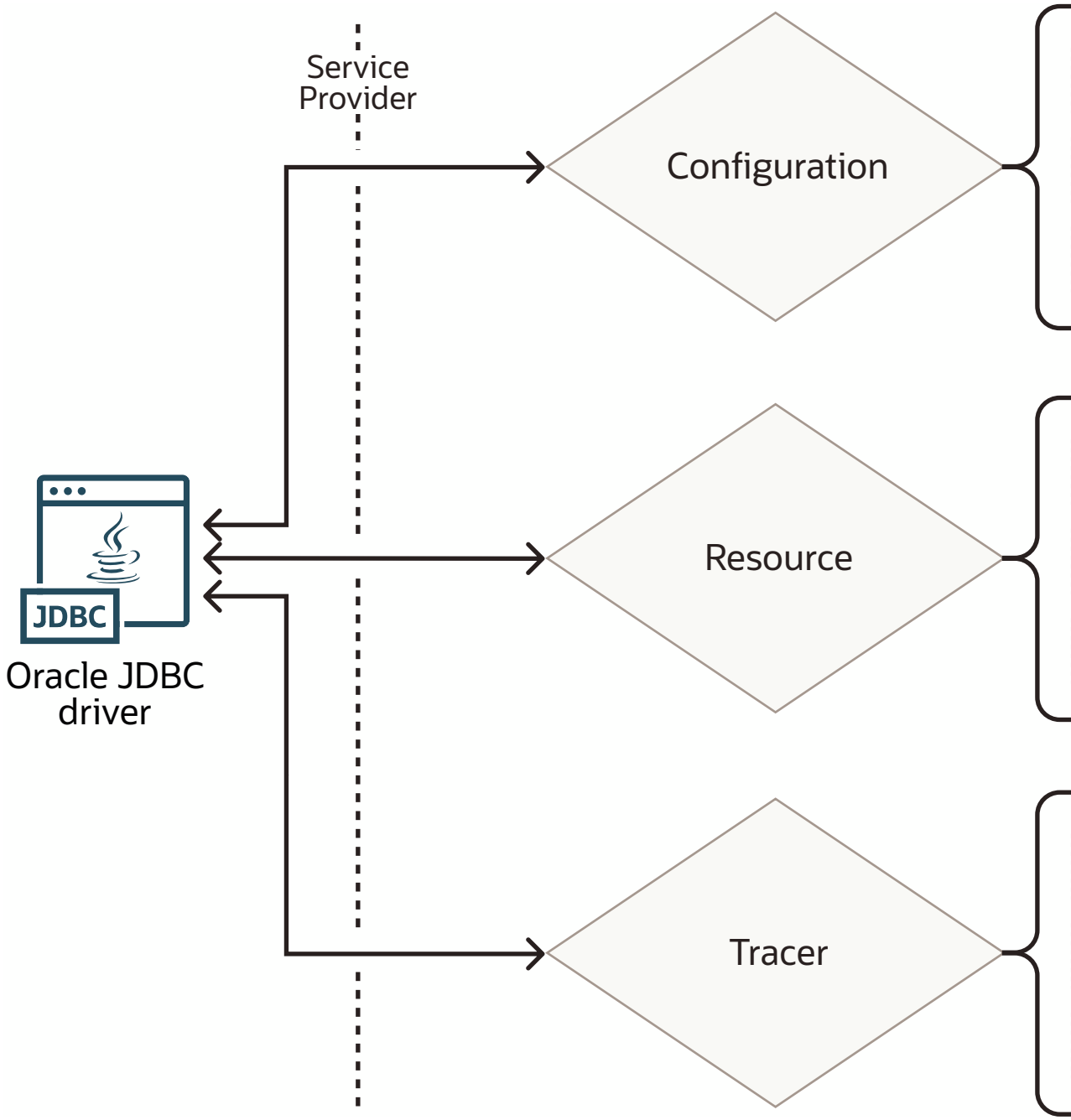
JDBC Service Provider Extensions

Starting with Oracle Database Release 23ai, you can extend the capabilities of Oracle JDBC drivers through service providers. You can use the standard Service Provider Interface to define custom providers and load them at run time.

You can load the following three types of providers:

- Centralized Configuration Providers
- Resource Providers
- Trace Event Listener Providers
- JDBC Extensions for Cloud Vendors

The following diagram illustrates the use cases for each provider:



Oracle provides Open Source Providers for the following:

- Azure App Configuration
- OCI Object Storage
- Azure Vault
- Azure Active Directory Token
- OCI IAM Token
- OCI Vault
- Open Telemetry

- OCI APM
- HashiCorp Vault (through partnership)

10.1 Centralized Configuration Providers

Use these providers to support distributed configurations of database clients.

> **Note:**
>
> You can find the specification for the payloads and formats across .NET and C++ clients.

A Centralized Configuration Provider furnishes the Oracle JDBC drivers with the necessary information for creating a database connection including a connection string (JDBC URL), and optionally, JDBC connection properties such as user ID, database password, or wallet location. Sensitive information such as the Database password or wallet location is typically stored separately in a vault.

The Service Provider contract is defined in such a way that if the JDBC URL is of the following format:

```
jdbc:oracle:<driver type>:@config-<provider>
```

then, the driver attempts to load the *<provider>* from the list of registered service providers. For example:

```
jdbc:oracle:thin:@config-azure:myappconfig?key=sales_app1&label=dev
```

To make the external provider discoverable to the applications, you must take care of the following:

- Implement the `oracle.jdbc.spi.OracleConfigurationProvider` interface available in the JDBC Driver
- Provide the URL string of the `<provider>` by overriding the `getType` method
- Provide an implementation of the `getConnectionProperties` method to return the properties to be used by the JDBC Driver

Oracle distributes individual JAR files as Configuration Providers for the following services in the Maven Central repository:

- Azure App Configuration (with optional reference to Azure Key Vault for secrets)
- OCI Object Storage (with optional reference to OCI Vault for secrets)
- OCI Database Tools Connections (with optional reference to OCI Vault for secrets)

If you want to build your own providers, then you must extend the `oracle.jdbc.spi.OracleConfigurationProvider` interface and follow the Java SPI specification.

10.1.1 Azure App Configuration

The Oracle data source uses a new prefix `jdbc:oracle:thin:@config-azure:` to be able to identify the configuration parameters that should be loaded to use the Azure App Configuration.

For enabling the existing applications to use this feature transparently, you only need to specify the name of the Azure App Configuration, and optionally, a prefix for the key-names and the Label, with the following syntax:

```
jdbc:oracle:thin:@config-azure:{appconfig-name}[?
key=prefix&label=value&option1
=value1&option2=value2]
```

The only requirements are the following:

- Include the provider JAR file in the classpath or the provider reference in the POM file
- Replace the existing URL values with the required values

This feature attaches the values of a data source to a key, which is the prefix of multiple keys in the Azure App Configuration, and a label. Both the key and the label are optional. If those are not specified, then it attaches all the values with no labels and no prefixes in the configuration.

Following are the four fixed values that are looked at with this key and label pair:

- `connect_descriptor` (required)
- `user` (optional)
- `password` (optional)
- `wallet_location` (optional)

The rest of the values are dependent on the JDBC Driver, which have the `jdbc/` prefix. Multiple key-values pairs are retrieved for a specific label and key, which are applied to a data source. The key values are the properties (constant keys) defined in the `OracleConnection` interface.

For example, a data source URL with the value `jdbc:oracle:thin:@config-azure:myappconfig?key=/sales_app1/&label=dev` in an App Configuration, say `myappconfig`, generates an `OracleDatasource` with values similar to the following values:

> **Note:**
>
> - Keep in mind that the prefix is `/sales_app1/` and the label is `dev`.
> - The JDBC Driver internally concatenates the connection descriptor with `jdbc:oracle:thin:@` to set the URL. This enables sharing the property with the other driver implementations.

Key	Value	Label	JDBC Driver Connection Property
`/sales_app1/user`	`scott`	dev	`user=scott`

Key	Value	Label	JDBC Driver Connection Property
/sales_app1/password	{"uri":"https://mykeyvault.vault.azure.net/secrets/*<password>*"}	dev	password=*<password>* (Value of the secret in the URI)
/sales_app1/wallet_location	{"uri":"https://mykeyvault.vault.azure.net/secrets/*<wallet_location>*"}	dev	oracle.net.wallet_location (value: "data:;base64,<value of the secret in the URI>")
/sales_app1/connect_descriptor	(description=(retry_count=20)(retry_delay=3s)(address=(protocol=tcps)(port=1521)(host=myserver.oraclecloud.com))(connect_data=(service_name=myservice.oraclecloud.com))(security=(ssl_server_dn_match=yes)(ssl_server_cert_dn="CN=DN1.oraclecloud.com, OU=Oracle US, O=Oracle Corporation, L=Redwood City, ST=California, C=US")))	dev	URL=jdbc:oracle:thin:@(description=(retry_count=20)(retry_delay=3s)(address=(protocol=tcps)(port=1521)(host=myserver.oraclecloud.com))(connect_data=(service_name=myservice.oraclecloud.com))(security=(ssl_server_dn_match=yes)(ssl_server_cert_dn="CN=DN1.oraclecloud.com, OU=Oracle US, O=Oracle Corporation, L=Redwood City, ST=California, C=US")))
/sales_app1/jdbc/autoCommit	false	dev	autoCommit=false
/sales_app1/jdbc/oracle.jdbc.loginTimeout	20	dev	oracle.jdbc.loginTimeout=20
/sales_app1/jdbc/oracle.jdbc.fanEnabled	false	dev	oracle.jdbc.fanEnabled=false

The following code snippet shows the relevant details in a sample Java client:

```
OracleDataSource ds = new OracleDataSource();
ds.setURL("jdbc:oracle:thin:@config-azure:myappconfig?
key=sales_app1&label=dev");
Connection cn = ds.getConnection();
Statement st = cn.createStatement();
ResultSet rs = st.executeQuery("select sysdate from dual");
```

10.1.2 OCI Object Storage

For using this Configuration Provider, you must make optional references to OCI Vault for secrets.

In this case, the configuration is stored in JSON format, which is common to all the clients. The provider is identified by `ociobject` provider in the URL, as shown in the following example:

```
jdbc:oracle:thin:@config-ociobject:
https://objectstorage.oraclecloud.com/myjava/bucket1/
payload_ojdbc_objectstorage.json
```

The only required parameter is the URL Path (URI) of the object. You can retrieve this value from the OCI Web Console, using the following navigation: `Object Storage / Buckets / Object → Object Details`, which retrieves a value similar to the following:

```
https://objectstorage.oraclecloud.com/myjava/bucket1/
payload_ojdbc_objectstorage.json
```

There are four fixed values that are looked with this `prefix` and `label` pair:

- `connect_descriptor` (required)
- `user` (optional)
- `password` (optional)
- `wallet_location` (optional)

The rest of the values are dependent on the JDBC Driver, which have the `jdbc/` prefix. Multiple key-values pairs are retrieved for a specific label and key, which are applied to a data source. The key values are the properties (constant keys) defined in the `OracleConnection` interface.

The `password` and `wallet_location` are references to a Vault provider, as shown in the following code snippet:

> **Note:**
>
> The Vault provider can also be an Azure Key Vault, which uses `vault-azure` as its type.

```
{"connect_descriptor": "(description=(retry_count=20)(retry_delay=3s)
(address=(protocol=tcps)(port=1521)(host=myhost.oraclecloud.com))
(connect_data=(service_name=myservice.oraclecloud.com))
(security=(ssl_server_dn_match=yes)))",
  "user": "scott",
  "password": {
    "type": "vault-oci",
    "value": "myvalue",
    "authentication": {
      "method": "OCI_INSTANCE_PRINCIPAL"
    }
```

```
  },

  "wallet_location": {
    "type": "vault-oci",
    "value": "myvalue",
    "authentication": {
      "method": "OCI_INSTANCE_PRINCIPAL"
    }
  },
  "jdbc": {
    "oracle.jdbc.ReadTimeout": 1000,
    "defaultRowPrefetch": 20,
    "autoCommit": "false"
}}
```

You can set multiple keys in one payload, adding the optional name in the URL. For example:

```
jdbc:oracle:thin:@config-ociobject:{object-url}
[?key=name&option1=value1&option2=value2]
```

The following example uses two keys, namely, `sales_app1` and `hr_internal_app1`:

```
{
  "sales_app1":
  {"connect_descriptor": "(description=(address=(protocol=tcps)
(port=1521)(host=myhost.oraclecloud.com))
    (connect_data=(service_name=myservice.oraclecloud.com))
(security=(ssl_server_dn_match=yes)))",
    "user": "scott",
    "password": {
      "type": "vault-oci",
      "value": "myvalue",
      "authentication": {
        "method": "OCI_INSTANCE_PRINCIPAL"
      }
    },
    "jdbc": {
      "oracle.jdbc.ReadTimeout": 1000,
      "defaultRowPrefetch": 20,
      "autoCommit": "false"
    }
  },
  "hr_internal_app1":
  {"connect_descriptor": "(description=(address=(protocol=tcps)
(port=1521)(host=myhost.oraclecloud.com))
    (connect_data=(service_name=myservice.oraclecloud.com))
(security=(ssl_server_dn_match=yes)))",
    "user": "scott",
    "password": {
      "type": "vault-oci",
      "value": "myvalue",
      "authentication": {
        "method": "OCI_INSTANCE_PRINCIPAL"
      }
```

```
    },
    "jdbc": {
     "oracle.jdbc.ReadTimeout": 0,
     "defaultRowPrefetch": 100,
     "autoCommit": "true"
    }
  }
}
```

10.1.3 OCI Database Tools

For using this Configuration Provider, you must make optional reference to OCI Vault for secrets.

The OCI Database Tools service is a managed service that can be used to configure connections to a database, either Oracle Autonomous Database or MySQL. You can then use the `connection` objects in the SQL worksheet in the Web console. You can also use it as a directory of database connection configurations, and also as a provider that allows access to these configurations. Each configuration has an Oracle Cloud Identifier (OCID) that is used to identify which connection is used. It contains a `connectionString`, a `userName`, a `userPassword`, `keyStores`, and `advancedProperties`. The `advancedProperties` are currently limited to JDBC properties.

The example shows a JDBC URL that uses the OCI Database Tools provider:

```
jdbc:oracle:thin:@config-ocidbtools:ocid1.databasetoolsconnection
```

10.1.4 Built-in Configuration Providers

The JDBC Driver includes two built-in Configuration Providers, one HTTPS Provider and one File provider.

You do not need any extra JAR files for using these providers as they are based on the same JSON Schema as used for OCI Object Storage, that is, the schema is the same for all the JSON based providers, namely, HTTPS, File, and OCI Object Storage.

> **See Also:**
>
> Oracle Database JDBC Java API Reference

HTTPS Configuration Provider

You can provide the JSON configuration document through an HTTPS endpoint like Oracle REST Data Services (ORDS). HTTP is not supported because the JSON configuration document may contain multiple aliases, so the URL must end with the alias name that needs to be loaded, for example, `https://<URL>/aliasname`.

You can protect the access to this configuration document using IP ACL, TCPS, and Basic HTTP Authentication.

The JDBC URL follows the following format:

```
jdbc:oracle:thin:@config-https://<URL>[?key=name&option1=value1]
```

As the `https` prefix is followed by a host, the double slash (`//`) is required after the provider name, whether it is `http` or `https`.

You can configure client authentication with basic HTTP authentication over HTTPS, using wallet through properties, similar to the following example:

```
jdbc:oracle:thin:@config-https://confighost.mydomain.com/oracleconfig?
key=name&authentication=BASIC_AUTH&wallet_location=/path/to/wallet
```

You can use the authentication option to make the HTTP Configuration Provider use the basic HTTP authentication to retrieve the JSON configuration document.

> **Note:**
>
> The HTTPS Configuration Provider that Oracle distributes, supports Basic HTTP Authentication.

File Configuration Provider

In this case, the JSON configuration document is provided through the file system and access to this configuration is protected by file system protections. File Configuration Provider, The JDBC URL follows the following format, in case of this Configuration Provider:

```
jdbc:oracle:thin:@config-file:{path-to-file}[?option list]
```

Here, the `{path-to-file}` parameter accepts the same rules as the `java.io.File` class. The option list includes an attribute to indicate the connection key name, for example, `sales_app1` in the following example:

```
jdbc:oracle:thin:@config-file:path/to/file.json?key=sales_app1
```

10.2 Resource Providers

A Resource Provider provides Oracle JDBC with a single resource, such as a password or a connection string.

Resource providers are configured by the connection properties. For example, you can use the following connection properties to configure a password provider:

```
oracle.jdbc.provider.password=example-provider
oracle.jdbc.provider.password.vaultId=9999-8888-7777
```

In the preceding example, the `oracle.jdbc.provider.password` property configures the name of a password provider. The `oracle.jdbc.provider.password.vaultId` property configures a parameter that is recognized by the password provider.

A resource provider may provide any of the following resources:

- Database connection string
- Database user name
- Database password

- Database access token
- TLS/SSL configuration
- Trace event listener

Corresponding to the list above, the following connection properties identify the name of Resource Provider:

- `oracle.jdbc.provider.connectionString`
- `oracle.jdbc.provider.username`
- `oracle.jdbc.provider.password`
- `oracle.jdbc.provider.accessToken`
- `oracle.jdbc.provider.tlsConfiguration`
- `oracle.jdbc.provider.traceEventListener`

You can also configure additional parameters as connection properties.

> **See Also:**

10.3 Trace Event Listener Providers

Starting with Oracle Database Release 23ai, the JDBC driver can generate events that can be used to monitor a JDBC application.

The JDBC driver defines an `oracle.jdbc.spi.TraceEventListenerProvider` interface that can be used to register a custom `TraceEventListener` through the SPI mechanism.

For example, you can use this feature to register a listener that publishes the following events to OpenTelemetry:

- Database round trips during query execution
- Virtual IP address retries while establishing a connection
- Starting of a recovery from a database outage when Application Continuity is used
- Successful recoveries from a database outage when Application Continuity is used

> **See Also:**
>
> The JDBC Javadoc for more information

10.4 JDBC Extensions for Cloud Vendors

The Oracle JDBC Driver Extensions include providers for centralized configuration or token providers for authentication with the Database.

These extensions help you in implementing the Service Provider Interfaces (SPIs) for integration with widely used services, such as Cloud computing platforms. The extensions are open-source on GitHub at the following link, and the artifacts are available on Maven Central:

https://github.com/oracle-samples/ojdbc-extensions/tree/main

For Centralized Configuration Providers, the extensions include providers for the following:

- OCI DBTools connection
- Azure App Configuration

For Resource Providers, these extensions include providers for the following:

- Authentication tokens issued by either OCI IAM (Identity and Access Management) or Azure AD (Active Directory), while using Oracle Autonomous Database Serverless
- The database password or user name stored in an OCI Vault secret or Azure Vault secret
- JDBC URL and client wallet from an Oracle Autonomous Database Serverless for mTLS connections
- JDBC events that are stored in OpenTelemetry

> **See Also:**
>
> Oracle JDBC Driver Extensions in GitHub for information