



Databbix

Manual de usuario

Recolector de datos en tiempo real para Zabbix

Versión 2.5 del software

Septiembre 2026

Databbix — Manual de usuario

Copyright © 2014–2026 Trustserver SL. Todos los derechos reservados.

Este documento se distribuye bajo los términos de la *Common Development and Distribution License* (CDDL), versión 1.0. No se puede utilizar este documento salvo de acuerdo con los términos de dicha licencia. Se puede obtener una copia de la licencia en <https://opensource.org/licenses/CDDL-1.0>.

Este documento se proporciona «tal cual», sin garantías de ningún tipo, ni expresas ni implícitas, incluidas, entre otras, las garantías de que el documento esté libre de defectos, sea comercializable, adecuado para un fin determinado o no infrinja derechos de terceros. Todo el riesgo en cuanto a la calidad y exactitud del documento corresponde a quien lo utiliza. Trustserver SL no será responsable, en ningún caso, de los daños directos o indirectos que se deriven de su uso.

Databbix es software libre, distribuido bajo la licencia GNU General Public License, versión 2 o posterior.

Zabbix es una marca registrada de Zabbix LLC. Oracle y JD Edwards son marcas registradas de Oracle Corporation y sus filiales. Microsoft y SQL Server son marcas registradas de Microsoft Corporation. El resto de marcas citadas pertenecen a sus respectivos propietarios.

Trustserver SL

<https://trustserver.com/es/>

Historial de versiones

Versión	Fecha	Descripción
1.0	29/09/2026	Primera versión del manual de usuario, para Databbix 2.5.

Índice

I	Fundamentos	1
1	Introducción	3
1.1	Qué es Databbix	3
1.2	Cómo está organizado este manual	3
2	Organización y funcionamiento	5
2.1	Un proceso, un hilo por monitor	5
2.2	Ficheros de configuración	6
2.2.1	Herencia de valores	7
2.3	Ciclo de vida de un monitor	7
3	Conceptos	9
3.1	Host, ítem y clave	9
3.2	Monitor	9
3.3	Motor de consultas	10
3.4	Colección de consultas	11
3.5	Consulta	11
3.6	Ítem de consulta	11
3.7	Parámetro	11
3.8	Módulo	12
3.9	Descubrimiento	12
3.10	Contexto	13
3.11	Variable de entorno	13
3.12	Datos de ejecución	13
3.13	Scripts y transformaciones	14
3.14	Caché y planificación	14
3.15	Autenticación de webservice	15
II	Configuración común	17
4	Instalación y ficheros	19
4.1	Ubicación de los ficheros	19
4.2	Ficheros de consultas incluidos	19
4.3	Variables de entorno	20
5	Configuración del demonio	21
5.1	Línea de comandos	21
5.2	El fichero <code>databbix.conf</code>	21
5.2.1	Sección <code>[General]</code>	21
5.2.2	Sección <code>[Log]</code>	22
5.2.3	Ejemplo	22

5.3	Servicio systemd	22
5.4	Señales	23
6	Configuración de monitores: parte común	25
6.1	Estructura del fichero	25
6.2	Sección [General]	26
6.3	Sección [Zabbix]	26
6.4	Sección [Data]	26
6.5	Sección [Log]	27
6.6	Caracteres especiales	27
7	Ficheros XML de consultas: parte común	29
7.1	Qué es común y qué depende del motor	29
7.2	Reglas generales	30
7.3	El elemento <Queries>	30
7.4	Variables de entorno: <environment>	31
7.5	Scripts: <script>	31
7.6	La consulta: <query>	32
7.6.1	Atributos comunes	32
7.6.2	Tipos de consulta por motor	32
7.7	El elemento <item>	32
7.8	El elemento <parameter>	33
7.9	Consultas de descubrimiento	33
7.10	Consultas constantes	33
8	Configuración en Zabbix	35
8.1	Claves de ítem	35
8.2	Ítems internos	35
III	Motores	37
9	Motor Database	39
9.1	Configuración: sección [Database]	39
9.2	Drivers	39
9.2.1	Driver Oracle	40
9.2.2	Driver PGsql	40
9.2.3	Driver ODBC	41
9.2.4	Datos de ejecución	41
9.3	Consultas SQL: elementos propios del motor	42
9.4	Tipos de consulta	42
9.4.1	Consulta básica	42
9.4.2	Consulta de filtro (select="FILTER")	43
9.4.3	Consulta con variables de enlace (select="BIND")	44
9.4.4	Consulta de descubrimiento	45
9.5	Contextos	46
9.5.1	Declaración	47
9.5.2	Uso en consultas	47
9.5.3	Ejemplo: filtro dentro de cada base de datos	47
9.6	SQL alternativo según la base de datos	48
9.7	Ejemplos de configuración	49
9.7.1	Ejemplo: Oracle	49

9.7.2	Ejemplo: Oracle multitenant	49
9.7.3	Ejemplo: ASM	49
9.7.4	Ejemplo: PostgreSQL	50
9.7.5	Ejemplo: SQL Server por ODBC	50
10	Motor Webservice	53
10.1	REST y SOAP	53
10.2	Configuración: sección [Webservice]	53
10.3	Consultas REST	54
10.3.1	Rutas JSON	55
10.3.2	Parámetros en la URL	56
10.3.3	Filtro sobre la respuesta	56
10.3.4	Descubrimientos REST	57
10.4	Consultas SOAP	58
10.4.1	Conversión de la respuesta SOAP a JSON	59
10.4.2	Transformación de respuestas SOAP	61
10.5	Datos de ejecución	62
10.6	Autenticación	62
10.6.1	Motores y métodos de autenticación	62
10.6.2	Cómo se elige la autenticación	62
10.6.3	Ejemplos	63
11	Motor MQTT	65
11.1	Configuración: sección [MQTT]	65
11.2	Ficheros de consultas	66
11.3	Lectura de valores	66
11.4	Descubrimiento de topics	67
11.5	Datos de ejecución	68
IV	Herramientas y uso	69
12	Programación de scripts	71
12.1	Lenguajes disponibles	71
12.1.1	JavaScript	71
12.1.2	Python	71
12.2	Dónde se escriben los scripts	71
12.3	Entorno de ejecución	72
12.3.1	Ámbito y estado	72
12.3.2	Funciones de log	73
12.3.3	Tipos de datos	73
12.3.4	Errores	73
12.4	Transformación de valores en consultas SQL	74
12.4.1	Ejemplo: fechas	74
12.4.2	Ejemplo: traducir estados a números	75
12.4.3	Ejemplo en Python	75
12.5	Transformación de descubrimientos	75
12.6	Transformación de respuestas de webservice	76
12.6.1	Tratar códigos de error	77
12.6.2	Reestructurar la salida	77
12.6.3	Filtrar la salida	78
12.6.4	Respuestas SOAP	78

12.6.5	Combinar con transformaciones de ítem	78
12.7	Programación de autenticaciones de webservice	78
12.7.1	Credenciales	78
12.7.2	Verificación de llamadas	79
12.7.3	Métodos basados en el motor RESTToken	79
12.7.4	Métodos basados en el motor HTTPAuth	80
13	La utilidad queryexplorer	83
13.1	Invocación	83
13.2	Modo interactivo	84
13.3	Modo no interactivo	84
13.4	Comandos	85
13.4.1	Ítems y consultas	85
13.4.2	Contextos	85
13.4.3	Variables y expresiones	85
13.4.4	Otros	86
13.5	Casos de uso	86
13.5.1	Validar un fichero de monitor y su conexión	86
13.5.2	Validar la autenticación de un webservice	86
13.5.3	Probar una consulta nueva	86
13.5.4	Comprobar el SQL alternativo	86
13.5.5	Comprobar los contextos	87
14	Guía práctica: crear una consulta nueva	89
14.1	Paso 1: escribir y probar el SQL	89
14.2	Paso 2: elegir el tipo de consulta	89
14.3	Paso 3: escribir el XML	89
14.4	Paso 4: añadir el fichero al monitor	90
14.5	Paso 5: probar con queryexplorer	90
14.6	Paso 6: crear los ítems en Zabbix	91
15	Diagnóstico y resolución de problemas	93
15.1	Niveles de log	93
15.2	Problemas frecuentes	93
V	Apéndices	95
A	Referencia rápida del XML	97
A.1	Nivel <Queries> (todos los motores)	97
A.2	Consulta constante (todos los motores)	97
A.3	Motor Database	97
A.3.1	Consulta básica	97
A.3.2	Consulta de filtro	98
A.3.3	Consulta con variables de enlace	98
A.3.4	Consulta de descubrimiento	98
A.4	Motor Webservice	99
A.4.1	Consulta REST	99
A.4.2	Consulta REST de filtro	99
A.4.3	Consulta REST de descubrimiento	99
A.4.4	Consulta SOAP	100
A.5	Motor MQTT	100

A.5.1	Lectura de valor	100
A.5.2	Descubrimiento de topics	100
B	Valores por defecto	101

Parte I

Fundamentos

Capítulo 1

Introducción

1.1 Qué es Databbix

Databbix es un demonio que recoge datos de fuentes externas y los envía a un servidor Zabbix. Las fuentes pueden ser bases de datos SQL (Oracle, PostgreSQL, SQL Server y cualquier otra accesible por ODBC), servicios web REST o SOAP y brokers MQTT. Databbix se comporta ante Zabbix como un *agente activo*: pide al servidor la lista de ítems que tiene que recoger para un host, los calcula ejecutando consultas y envía los resultados.

A diferencia de los *user parameters* del agente de Zabbix, Databbix:

- mantiene abierta la conexión con la base de datos entre ejecuciones;
- ejecuta cada consulta *una sola vez* para todos los ítems que la usan, y reparte las filas del resultado entre ellos;
- guarda los resultados en caché durante un tiempo configurable;
- define las consultas en ficheros XML, sin necesidad de programar, e incluye descubrimiento de bajo nivel (LLD) de Zabbix.

1.2 Cómo está organizado este manual

Parte I, Fundamentos. Cómo funciona Databbix y los conceptos que se usan en el resto del manual. Conviene leerla antes de configurar nada.

Parte II, Configuración común. Instalación, demonio, la parte común de los ficheros de monitor y de los ficheros XML de consultas, y la configuración en Zabbix. Todo lo que se describe aquí es igual para cualquier fuente de datos.

Parte III, Motores. Un capítulo por motor (Database, Webservice y MQTT) con su sección de configuración, sus elementos XML y ejemplos con datos y resultados.

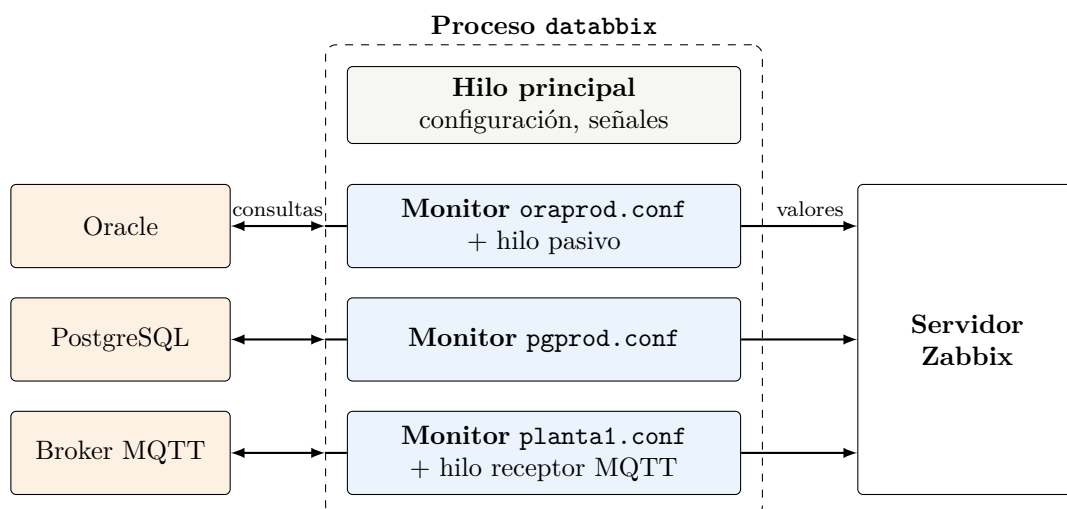
Parte IV, Herramientas y uso. Programación de scripts, la utilidad `queryexplorer`, una guía práctica y la resolución de problemas.

Capítulo 2

Organización y funcionamiento

2.1 Un proceso, un hilo por monitor

Databbix se ejecuta como un único proceso, `databbix`, que atiende todas las fuentes de datos de la máquina. Dentro del proceso, cada fuente de datos se atiende en su propio hilo, llamado *monitor*:



Hilo principal. Lee la configuración global, recorre el directorio de monitores y lanza un hilo por cada fichero `.conf`. Después se queda esperando: reabre los ficheros de log cuando recibe `SIGHUP` y ordena la parada de todos los monitores cuando recibe `SIGTERM` o `SIGINT`.

Hilo de monitor. Hay uno por fichero `.conf`. Cada monitor tiene su propia conexión con la fuente de datos, su propia conexión con Zabbix y su propio fichero de log.

Hilo pasivo. Si el monitor tiene `PassivePort` distinto de 0, lanza un hilo adicional que escucha en ese puerto y responde a las comprobaciones pasivas de Zabbix.

Hilos del motor. Algunos motores necesitan hilos propios. El motor MQTT, por ejemplo, tiene un hilo que recibe continuamente los mensajes del broker.

Los monitores son independientes entre sí:

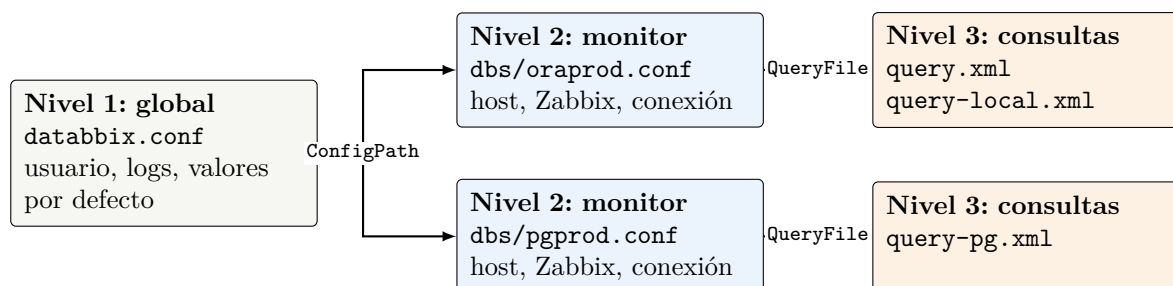
- Si una fuente de datos falla o está parada, solo se ve afectado su monitor. El resto sigue funcionando.
- Si un monitor no puede arrancar (por ejemplo, por un error en su `.conf` o en un XML), ese monitor termina y lo registra en su log, pero el proceso sigue atendiendo a los demás. El monitor no se vuelve a lanzar hasta que se reinicia el servicio.

- Un monitor también termina si supera `MaxErrors` ejecuciones fallidas seguidas.
- Cuando terminan todos los monitores, el proceso termina.

Atención. Como un monitor que falla no detiene el proceso, `systemctl status` puede mostrar el servicio activo aunque alguno de los monitores no esté funcionando. Para vigilarlo, use el ítem pasivo `agent.ping` o compruebe en Zabbix que llegan datos de cada host.

2.2 Ficheros de configuración

La configuración se reparte en tres niveles:



Fichero	Formato	Contenido
<code>databbix.conf</code> (uno)	INI	Configuración del proceso: usuario y grupo del demonio, directorio de monitores, módulos y valores por defecto de logs, ficheros de consultas y variables de entorno para todos los monitores.
<code>db/*conf</code> (uno por monitor)	INI	Todo lo propio de una fuente de datos: nombre del host en Zabbix, servidor Zabbix, motor, datos de conexión (driver, cadena de conexión, usuario, URL, autenticación...), ficheros de consultas y variables de entorno propias.
<code>query*.xml</code>	XML	Qué datos se obtienen y cómo: consultas SQL o llamadas a servicios, los ítems de Zabbix que produce cada una, contextos, scripts y autenticaciones. No contienen datos de conexión, así que un mismo XML sirve para todas las fuentes del mismo tipo.
<code>tnsnames.ora</code> , <code>odbc.ini</code>	proprios	Definición de las conexiones de Oracle y ODBC a las que se refieren los ficheros de monitor.
<code>/etc/sysconfig/databbix</code>	variables de shell	Entorno del servicio (<code>TNS_ADMIN</code> , <code>ODBCINI...</code>).

Fichero	Formato	Contenido
<code>AuthPersistFile</code> (opcional)	JSON	Estado de la autenticación de un webservice (por ejemplo, el token), para no tener que volver a autenticarse tras un reinicio. Lo escribe Databbix.

2.2.1 Herencia de valores

Algunos parámetros se pueden poner en los dos primeros niveles. En ese caso, el valor del fichero de monitor tiene prioridad sobre el de `databbix.conf`:

- Los parámetros de la sección `[Log]` se heredan uno a uno. Un monitor puede, por ejemplo, subir solo su `LogLevel`.
- `QueryFile`: si el monitor define alguno, su lista *sustituye* por completo a la global.
- `Environment`: el monitor ve las variables globales y las suyas. Si una variable está en ambos sitios, gana la del monitor. Las dos tienen prioridad sobre los valores por defecto del XML.

2.3 Ciclo de vida de un monitor

Al arrancar, cada monitor hace lo siguiente:

1. Lee su fichero `.conf` y comprueba que están los parámetros obligatorios.
2. Carga la colección de consultas de sus ficheros XML. Si hay un error de sintaxis, el monitor termina.
3. Se conecta a la fuente de datos. Si no puede, lo reintenta más tarde.
4. Pide al servidor Zabbix la lista de ítems activos del host (*active checks*).
5. Si está configurado, lanza el hilo pasivo.

Después entra en un bucle. En cada vuelta elige la primera tarea que corresponda y duerme hasta que llegue su momento:

1. Si aún no tiene la lista de ítems, la pide a Zabbix.
2. Si no hay conexión con la fuente de datos, vuelve a conectar.
3. Si quedaron valores sin enviar por un fallo de Zabbix, los reenvía.
4. Si toca actualizar la lista de ítems (cada `Zabbix.Refresh` segundos), la pide de nuevo.
5. Si no, ejecuta las consultas de los ítems que han cumplido su intervalo y envía los resultados.

Cuando falla la conexión con la fuente o con Zabbix, el monitor espera `General.RetryOnError` segundos antes de reintentar. Si se pierde la conexión con Zabbix, los valores obtenidos se guardan y se envían cuando vuelve la conexión.

Capítulo 3

Conceptos

Este capítulo define los términos que se usan en el resto del manual. Cada concepto va acompañado de un ejemplo de cómo aparece en los ficheros `.conf` y `.xml`. Los ejemplos son fragmentos: los capítulos siguientes describen todos los parámetros y atributos.

3.1 Host, ítem y clave

Son conceptos de Zabbix:

Host. Elemento monitorizado en Zabbix. Cada monitor de Databbix envía los datos de un único host, identificado por `General.Name`.

Ítem. Dato que Zabbix recoge periódicamente de un host (número de sesiones, espacio libre de un tablespace...). Los ítems que atiende Databbix son de tipo *Zabbix agent (active)*.

Clave. Identificador del ítem, con la forma `nombre[parámetro1,parámetro2,...]`. El `nombre` indica qué ítem de qué consulta hay que calcular y los parámetros indican sobre qué objeto (qué tablespace, qué base de datos...).

Ejemplo. El host `ORAPROD` de Zabbix tiene un ítem activo con la clave:

```
tez.ora.tspace.pctfree[USERS]
```

El monitor cuyo `.conf` tiene `Name = ORAPROD` busca en sus XML el `<item>` llamado `tez.ora.tspace.pctfree` y lo calcula para el tablespace `USERS`.

3.2 Monitor

Un monitor es la unidad de trabajo de Databbix: une *una fuente de datos* con *un host de Zabbix*. Se define en un fichero `.conf` y se ejecuta en su propio hilo. Por ejemplo, si una máquina vigila tres bases de datos, hay tres ficheros `.conf`, tres monitores y tres hosts en Zabbix.

Un monitor está formado por:

- un motor y una conexión con la fuente de datos;
- una colección de consultas;
- una conexión con el servidor Zabbix y la lista de ítems activos del host;
- un planificador que decide cuándo ejecutar cada consulta.

Ejemplo. Fichero `/etc/databbix/dbs/oraprod.conf` con lo mínimo para un monitor de Oracle:

```
[General]
Name = ORAPROD                # host en Zabbix

[Zabbix]
Server = zabbix.empresa.com    # a quién se envían los datos
Port = 10051

[Data]
Module = DB                    # módulos de consultas que usa

[Database]                     # de dónde se sacan los datos
DBDriver = Oracle
ConnectionString = oraprod
UserName = zabbix
Password = secreto
```

3.3 Motor de consultas

El motor (*engine*) es el componente que sabe hablar con un tipo de fuente de datos. Cada monitor usa un único motor, que se elige con `General.Engine`. El motor determina:

- qué sección de conexión lleva el fichero de monitor ([`Database`], [`Webservice`] o [`MQTT`]);
- qué elementos y atributos admiten los XML (por ejemplo, `<sql>` en `Database` y `<url>` en `Webservice`);
- qué tipos de consulta existen y cómo se obtienen los valores.

Motor	Fuente de datos	Qué es una consulta
Database	Bases de datos SQL mediante un <i>driver</i> : Oracle, Pgsql u ODBC.	Una sentencia SQL.
Webservice	Servicios web REST (JSON) o SOAP.	Una llamada HTTP a una URL; los valores se extraen de la respuesta.
MQTT	Un broker MQTT.	La lectura del último valor publicado en un <i>topic</i> .

Todos los ficheros XML de un monitor deben ser del motor del monitor: el motor se declara en el atributo `engine` de `<Queries>`.

Ejemplo. Un monitor de servicio REST declara el motor en su `.conf`:

```
[General]
Name = JDEPROD
Engine = Webservice
QueryFile = /etc/databbix/query-jde.xml

[Webservice]
BaseURL = https://jde.empresa.com:9888/restservice
```

y el XML que usa declara el mismo motor:

```
<Queries engine="Webservice">
  ...
</Queries>
```

3.4 Colección de consultas

Es el conjunto de consultas que resulta de cargar los ficheros XML de un monitor (`QueryFile`). Dentro de una colección, los nombres de consulta y de ítem son únicos: así, Databbix sabe qué consulta ejecutar para cada clave que le pide Zabbix.

Si varios monitores usan el mismo motor, los mismos ficheros y las mismas variables de entorno, comparten una única copia de la colección en memoria.

Ejemplo. Colección formada por el fichero de consultas de la distribución y uno propio:

```
[General]
QueryFile = /etc/databbix/query.xml
QueryFile = /etc/databbix/query-local.xml
```

3.5 Consulta

Una consulta (`<query>`) define una operación sobre la fuente de datos (una sentencia SQL, una llamada a un servicio...) y los ítems de Zabbix que se obtienen de su resultado. Una consulta puede producir muchos ítems, y Databbix la ejecuta una sola vez para todos ellos.

Según cómo se relaciona el resultado con los ítems, hay varios tipos: básica, filtro, bind, descubrimiento y constante. Se describen en el capítulo 7.

Ejemplo. Una consulta básica que produce dos ítems a partir de una sola sentencia:

```
<query name="PGWriter" module="PG">
  <sql><![CDATA[
    select checkpoints_timed, checkpoints_req
    from pg_stat_bgwriter
  ]]>
</sql>
<item name="tez.pg.checkpointtimed" column="checkpoints_timed"/>
<item name="tez.pg.checkpointreq" column="checkpoints_req"/>
</query>
```

3.6 Ítem de consulta

Dentro de una consulta, cada `<item>` indica de dónde sale el valor de un ítem de Zabbix: de qué columna del resultado SQL, de qué ruta de la respuesta JSON, de un dato interno (tiempo de ejecución)... Su atributo `name` es el nombre de la clave en Zabbix.

Además de los ítems definidos en los XML, Databbix responde a unos pocos ítems internos, como `agent.ping` o `agent.version`.

Ejemplo. Tres formas de obtener el valor:

```
<!-- de una columna SQL -->
<item name="tez.ora.sessions" column="n"/>
<!-- de una ruta dentro de la respuesta JSON de un servicio -->
<item name="ts.jde.instance.state" path="instanceState" type="INT"/>
<!-- de un dato interno: segundos que tardó la consulta -->
<item name="tez.ora.tspace.exectime" built_in="exec_time"/>
```

3.7 Parámetro

Un parámetro (`<parameter>`) relaciona un parámetro de la clave de Zabbix con la consulta. Según el tipo de consulta, el valor del parámetro sirve para:

- elegir una fila del resultado (*filtro*);
- pasarlo a la sentencia SQL o a la URL (*bind*);
- elegir el contexto en el que se ejecuta la consulta.

Ejemplo. Con la clave `tez.pg.sessions[ventas]`, el primer parámetro (`ventas`) elige la fila cuya columna `datname` vale `ventas`:

```
<parameter column="datname" order="1" type="STRING"/>
```

Con la clave `tez.pg.param[max_connections]`, el primer parámetro se pasa a la sentencia SQL en la variable `:name`:

```
<parameter bind=":name" order="1" type="STRING"/>
```

3.8 Módulo

Un módulo es un nombre que agrupa consultas relacionadas (DB, Instance, PG, MSSQL...). Cada consulta y cada contexto declaran su módulo con el atributo `module`, y cada monitor declara los módulos que usa con `Data.Module`.

Ejemplo.

```
<query name="Sessions" module="Instance"> ... </query>
```

```
[Data]
Module = DB
Module = Instance
```

3.9 Descubrimiento

El descubrimiento de bajo nivel (LLD) es el mecanismo de Zabbix para crear ítems automáticamente según los objetos que existen (un ítem por tablespace, por base de datos...). En Databbix, una consulta de descubrimiento devuelve la lista de objetos en el formato JSON que espera Zabbix. Normalmente va acompañada de una consulta de filtro que calcula los valores de cada objeto descubierto.

Ejemplo. Descubrimiento de las bases de datos de un PostgreSQL:

```
<query name="PGDiscoverDB" module="PG" discovery="yes">
  <sql><![CDATA[
    select datname from pg_database where datname not like 'template%'
  ]]>
</sql>
<item name="tez.pg.db.discovery"/>
<field name="DBNAME" column="datname"/>
</query>
```

El ítem `tez.pg.db.discovery` devuelve a Zabbix:

```
{"data": [{"#DBNAME": "postgres"}, {"#DBNAME": "ventas"}]}
```

y en Zabbix se define el prototipo `tez.pg.sessions[#{DBNAME}]`, que da lugar a `tez.pg.sessions[postgres]` y `tez.pg.sessions[ventas]`.

3.10 Contexto

Un contexto es un estado de la conexión que se puede cambiar entre consultas sin volver a conectar. Los ejemplos típicos son la base de datos activa en SQL Server o PostgreSQL y el contenedor (PDB) en Oracle multitenant. Gracias a los contextos, un único monitor con una sola conexión puede consultar todas las bases de datos de un servidor.

Los contextos solo existen en el motor Database. Se declaran en el XML con un valor por defecto. Cada consulta se ejecuta en el contexto por defecto, en uno fijo o en el que indique la clave del ítem. Databbix agrupa las consultas por contexto para cambiarlo el menor número de veces posible.

Ejemplo. Se declara el contexto Database, cuyo valor por defecto es `master`, y una consulta que se ejecuta en la base de datos que indique el primer parámetro de la clave:

```
<context name="Database" module="MSSQL" default="master"/>

<query name="MSFGSpace" module="MSSQL" select="FILTER">
  <sql> ... </sql>
  <parameter context="Database" order="1"/>
  <parameter column="filegroup_name" order="2" type="STRING"/>
  <item name="tez.mssql.fgspace.used" column="used"/>
</query>
```

La clave `tez.mssql.fgspace.used[ventas,PRIMARY]` ejecuta la consulta en la base de datos `ventas` y devuelve el valor del *filegroup* `PRIMARY`.

3.11 Variable de entorno

Una variable de entorno es un valor con nombre que se sustituye en los XML con la sintaxis `${Nombre}`. Sirve para escribir XML genéricos y adaptarlos a cada instalación desde la configuración: el nombre de un esquema, la base de datos de administración, etc.

No tienen relación con las variables de entorno del sistema operativo. Se definen con `<environment>` en el XML (valor por defecto) y con `Environment = Nombre=Valor` en `databbix.conf` o en el fichero del monitor, que tienen prioridad.

Ejemplo. El XML declara la variable y la usa en el SQL:

```
<environment name="SvmSchema" default="svm920"/>

<query name="JdeSpecJobs" module="JDEDB">
  <sql><![CDATA[
    select count(*) as n from ${SvmSchema}.f986110
  ]]>
  </sql>
  <item name="ts.jdedb.specjobs" column="n"/>
</query>
```

Un monitor que no dice nada consulta `svm920.f986110`. Otro monitor con esta línea consulta `svm910.f986110`:

```
[General]
Environment = SvmSchema=svm910
```

3.12 Datos de ejecución

Son valores que se conocen solo una vez establecida la conexión, como el producto de base de datos al que se ha conectado el driver. Se usan en el motor Database (sección 9.2.4). Se consultan

con la función `runtime('NOMBRE')` en las condiciones de `<when>`, para elegir el SQL adecuado a cada base de datos.

Ejemplo.

```
<sql>
  <multiple>
    <when condition="runtime('DB.DATABASE') = 'Oracle'"><![CDATA[
      select sysdate as ahora from dual ]]></when>
    <when><![CDATA[
      select current_timestamp as ahora ]]></when>
  </multiple>
</sql>
```

3.13 Scripts y transformaciones

Los XML pueden incluir funciones escritas en JavaScript o Python (`<script>`). Estas funciones se usan para:

- transformar un valor antes de enviarlo a Zabbix (por ejemplo, convertir una fecha en segundos transcurridos);
- procesar la respuesta de un webservice antes de extraer los valores;
- generar credenciales y validar respuestas en las autenticaciones de webservice.

Cada monitor tiene un intérprete para su colección de consultas. Los scripts declarados en el nivel de `<Queries>` son comunes a todas las consultas; los declarados dentro de un `<query>` solo son visibles en esa consulta. Todos los scripts de una colección deben estar en el mismo lenguaje. El capítulo 12 explica cómo programarlos.

Ejemplo. Una función global y un ítem que la usa:

```
<script language="Javascript"><![CDATA[
  function toUnix(fecha) {
    return Math.round(Date.parse(fecha) / 1000);
  }
]>
</script>

<query name="LastBackup" module="DB">
  <sql><![CDATA[
    select max(completion_time) as fin from v$backup_set ]]>
  </sql>
  <item name="local.ora.lastbackup" column="fin" transform="Javascript:toUnix"/>
</query>
```

3.14 Caché y planificación

Zabbix decide cada cuánto quiere cada ítem (el intervalo de actualización del ítem). Databbix agrupa los ítems por consulta y, cuando varios ítems de la misma consulta tienen que actualizarse, ejecuta la consulta una sola vez.

El resultado de una consulta se guarda en caché durante un tiempo máximo (`General.MaxCache` del monitor o el atributo `maxCache` de la consulta, el menor de los dos). Mientras el resultado sea válido, los ítems de esa consulta se sirven desde la caché, sin volver a ejecutarla.

Ejemplo. El monitor permite reutilizar resultados durante 60 segundos, pero una consulta concreta solo durante 30:

```
[General]
MaxCache = 60
```

```
<query name="TSSpace" module="DB" select="FILTER" maxCache="30"> ... </query>
```

3.15 Autenticación de webservice

La mayoría de los servicios web exigen autenticarse antes de responder. En Databbix intervienen tres piezas:

Motor de autenticación. Mecanismo de autenticación programado en C++ dentro de Databbix: HTTPAuth, RESTToken, JDE y Oasis (tabla 3.1). No hay que confundirlo con el motor de consultas del monitor (**Webservice**).

Método de autenticación. Personalización de un motor de autenticación para un servicio concreto, definida en un fichero de consultas con el elemento `<authentication>`. Tiene un nombre (atributo `name`), indica el motor que usa (atributo `type`) y aporta funciones en JavaScript o Python que deciden, por ejemplo, qué petición de *login* se envía, dónde está el token en la respuesta o cuándo hay que volver a autenticarse. Solo HTTPAuth y RESTToken admiten personalización.

Credenciales. Usuario, contraseña y otros datos (entorno, rol...) que se definen en el fichero de monitor con `UserName`, `Password` y `Credentials = clave=valor`. Así, los XML no contienen secretos.

El parámetro `Webservice.Authenticator` del monitor da un nombre, que Databbix resuelve en este orden:

1. Busca en los ficheros de consultas del monitor un método de autenticación (`<authentication>`) con ese `name`. Si lo encuentra, usa su motor con las funciones del método.
2. Si no lo encuentra, busca un motor de autenticación con ese nombre y lo usa directamente, sin personalizar.
3. Si tampoco existe, el monitor no puede conectar y lo registra en su log.

La sesión de autenticación (por ejemplo, el token vigente) se mantiene mientras el monitor está en marcha y se renueva cuando el servicio la rechaza. Si se configura `Webservice.AuthPersistFile`, además se guarda en disco para reutilizarla después de un reinicio.

Ejemplo 1. Motor HTTPAuth sin personalizar: como no hay ningún método llamado HTTPAuth en los XML, se usa directamente el motor, y las credenciales van en el `.conf`.

```
[Webservice]
BaseURL = https://api.empresa.com/
Authenticator = HTTPAuth
UserName = monitor
Password = secreto
Credentials = method=BASIC
```

Ejemplo 2. Método de autenticación AISAuth, que personaliza el motor RESTToken, definido en el XML y usado desde el `.conf`:

Motor	Mecanismo
HTTPAuth	Autenticación HTTP estándar. El método se indica con la credencial <code>method</code> : BASIC, DIGEST, NTLM (con <code>username</code> y <code>password</code>) o HEADER (una cabecera fija, con <code>header</code> y <code>value</code>). Opcionalmente, un script puede generar las credenciales.
RESTToken	Autenticación por token. Se hace una petición de <i>login</i> y el token obtenido se envía en cada llamada como cabecera o cookie. Se configura por completo en el XML, con scripts que construyen la petición de login, validan la respuesta y detectan cuándo hay que volver a autenticarse.
JDE	Token de los servicios REST de JD Edwards (<i>Server Manager</i>).
Oasis	Cabecera <i>WS-Security UsernameToken</i> para servicios SOAP.

Tabla 3.1: Motores de autenticación

```
<authentication type="RESTToken" name="AISAuth">
  <script language="Javascript"> ... </script>
  <login type="JsonPost">
    <url>v2/tokenrequest</url>
    <request-function language="Javascript" function="authGenerate"/>
    <accept-function language="Javascript" function="authAccept"/>
  </login>
  <token type="Header" name="jde-AIS-Auth"/>
  <verify-call language="Javascript" function="verifyCall"/>
</authentication>
```

```
[Webservice]
BaseURL = https://ais.empresa.com/jderest/
Authenticator = AISAuth
UserName = monitor
Password = secreto
Credentials = environment=JPD920
Credentials = role=*ALL
AuthPersistFile = /var/lib/databbix/ais-token.json
```

El capítulo 12 explica cómo escribir las funciones de una autenticación.

Parte II

Configuración común

Capítulo 4

Instalación y ficheros

4.1 Ubicación de los ficheros

Una instalación estándar usa estas rutas:

Ruta	Contenido
/opt/trustserver/sbin/databbix	Ejecutable del demonio.
/opt/trustserver/lib64	Bibliotecas de Trustserver y cliente de Oracle.
/opt/trustserver/modules/databbix	Módulos (motores de consultas, lenguajes, motores de autenticación).
/etc/databbix/databbix.conf	Configuración global del demonio.
/etc/databbix/dbs/	Un fichero .conf por monitor.
/etc/databbix/query*.xml	Ficheros de definición de consultas.
/etc/databbix/tnsnames.ora	Alias de conexión de Oracle.
/etc/databbix/odbc.ini	Fuentes de datos ODBC.
/etc/sysconfig/databbix	Variables de entorno para el servicio.
/var/log/databbix/	Ficheros de log.

4.2 Ficheros de consultas incluidos

La distribución incluye varios ficheros XML de ejemplo listos para usar:

Fichero	Uso
query.xml	Oracle (fichero por defecto).
query-mt.xml	Oracle multitenant (CDB/PDB).
query-ora-9.xml	Oracle 9.
query-pg.xml	PostgreSQL.
query-pg-9.xml, query-pg-10.xml	Consultas específicas de versiones de PostgreSQL.
query-pg-bareos.xml	Catálogo de Bareos en PostgreSQL. Se usa siempre junto con <code>query-pg.xml</code> *.
query-mssql.xml	Microsoft SQL Server (vía ODBC).
query-db-jde.xml	Base de datos de JD Edwards (Oracle, DB2/400 o SQL Server). Si se configura JDEDB, se usa junto con el fichero que declara el contexto de <code>ContextName</code> *.
query-jde.xml, query-ais.xml	Servicios REST de JD Edwards y AIS.
query-soap.xml	Ejemplo de servicio SOAP.
query-mqtt.xml	Broker MQTT.
query-emcunity.xml, query-sagex3.xml	Otros servicios web.
zabbix_templates.xml	Plantillas de Zabbix asociadas.

***Ficheros que dependen de otro.** `query-pg-bareos.xml` no declara el contexto Database que usa: lo declara `query-pg.xml`. Del mismo modo, `query-db-jde.xml` usa el contexto indicado en la variable `ContextName` (Container por defecto, declarado en `query-mt.xml`; Database, en `query-pg.xml` o `query-mssql.xml`). Como un contexto tiene que estar declarado antes de que se use, el fichero que lo declara debe ir *antes* en las líneas `QueryFile`:

```
[General]
QueryFile = /etc/databbix/query-pg.xml           # declara el contexto Database
QueryFile = /etc/databbix/query-pg-bareos.xml    # lo usa
```

4.3 Variables de entorno

El servicio lee `/etc/sysconfig/databbix`. Estas son las variables habituales:

```
TNS_ADMIN=/etc/databbix           # directorio de tnsnames.ora
ODBCINI=/etc/databbix/odbc.ini     # fichero de fuentes ODBC
```

La unidad de systemd añade además `LD_LIBRARY_PATH=/opt/trustserver/lib64`.

Capítulo 5

Configuración del demonio

5.1 Línea de comandos

```
databbix [opciones]
```

Opción	Descripción
-h, --help	Muestra la ayuda y termina.
-d, --daemon	Pasa a segundo plano (modo demonio). Sin esta opción, el log también se muestra por la consola.
-n, --no-send-data	No envía datos al servidor Zabbix. Útil para pruebas.
-c, --config-file <i>f</i>	Fichero de configuración global. Por defecto: <code>/etc/databbix/databbix.conf</code> .
-p, --config-path <i>d</i>	Directorio con los ficheros <code>.conf</code> de los monitores. Por defecto: <code>/etc/databbix/dbs</code> .
-m, --module-path <i>d</i>	Directorios adicionales donde buscar módulos.

Para probar una configuración nueva sin instalar el servicio, lo más práctico es arrancar el demonio en primer plano, sin enviar datos y con un nivel de log alto (véase el capítulo 15):

```
/opt/trustserver/sbin/databbix -n -c /etc/databbix/databbix.conf
```

5.2 El fichero `databbix.conf`

Es un fichero de tipo INI con dos secciones: `[General]` y `[Log]`. Los valores de la sección `[Log]` y los de `QueryFile` y `Environment` son los valores por defecto de todos los monitores, y cada monitor puede redefinirlos en su propio fichero.

5.2.1 Sección `[General]`

Parámetro	Descripción
<code>DaemonUser</code>	Usuario (nombre o UID) al que cambia el proceso al arrancar con <code>-d</code> . El proceso toma también los grupos suplementarios de ese usuario.
<code>DaemonGroup</code>	Grupo (nombre o GID) al que cambia el proceso al arrancar con <code>-d</code> . Si no se indica, se usa el grupo principal de <code>DaemonUser</code> .

Parámetro	Descripción
ConfigPath	Directorio de los ficheros de monitores. Si está presente, tiene prioridad sobre la opción <code>-p</code> .
QueryFile	Fichero XML de consultas. Se puede repetir la línea para cargar varios ficheros. Por defecto: <code>/etc/databbix/query.xml</code> .
ModulePath	Directorios adicionales de módulos. Se puede repetir.
Environment	Variable de entorno para las consultas, con la forma <code>Nombre=Valor</code> . Se puede repetir (véase la sección 7.4).

5.2.2 Sección [Log]

Parámetro	Defecto	Descripción
LogPath	<code>/var/log/databbix</code>	Directorio de los logs.
LogFile	<code>databbix.log</code>	Log del proceso principal.
LogLevel	3	Nivel de detalle (véase la tabla 15.1).
LogMaxSize	0	Tamaño máximo en bytes antes de rotar. 0 desactiva la rotación interna.
LogMaxBackupCopies	4	Número de copias que se conservan al rotar.
MaxQueryExecTimeLog	10	Si una consulta tarda más de estos segundos, se registra un aviso en el log.

5.2.3 Ejemplo

```
[General]
DaemonUser = databbix
DaemonGroup = databbix
QueryFile = /etc/databbix/query.xml

[Log]
LogPath = /var/log/databbix
LogLevel = 4
MaxQueryExecTimeLog = 10
```

Atención. Si `DemonUser` es un UID numérico que no existe en `/etc/passwd` y no se indica `DemonGroup`, no hay grupo principal que usar: el proceso conserva el grupo con el que arrancó, que es `root` si se lanza como `root`. En ese caso indique siempre `DemonGroup`.

5.3 Servicio systemd

La unidad incluida (`databbix.service`) arranca el demonio en modo `forking` con el usuario `databbix`:

```
[Unit]
Description=Databbix Agent
After=network-online.target
Wants=network-online.target

[Service]
```

```
EnvironmentFile=/etc/sysconfig/databbix
Environment="LD_LIBRARY_PATH=/opt/trustserver/lib64"
PassEnvironment=TNS_ADMIN ODBCINI LD_LIBRARY_PATH
Type=forking
Restart=on-failure
KillMode=process
ExecStart=/opt/trustserver/sbin/databbix -c /etc/databbix/databbix.conf -d
RestartSec=10s
User=databbix
Group=databbix

[Install]
WantedBy=multi-user.target
```

Operaciones habituales:

```
systemctl enable --now databbix    # activar y arrancar
systemctl restart databbix         # aplicar cambios de configuración o XML
systemctl status databbix
```

Atención. Databbix lee los ficheros `.conf` y XML solo al arrancar. Para aplicar cualquier cambio en ellos hay que reiniciar el servicio.

5.4 Señales

Señal	Efecto
SIGTERM, SIGINT	Parada ordenada. Espera hasta unos 4 segundos a que terminen los monitores.
SIGHUP	Reabre los ficheros de log. La usa <code>logrotate</code> .

El fichero `databbix.logrotate` incluido envía SIGHUP después de rotar:

```
/var/log/databbix/*.log
{
    sharedscripts
    postrotate
        /usr/bin/killall -HUP databbix 2> /dev/null || true
    endscript
}
```

Si se usa `logrotate`, conviene dejar `LogMaxSize = 0` para que no roten a la vez `logrotate` y Databbix.

Capítulo 6

Configuración de monitores: parte común

Cada fichero con extensión `.conf` del directorio de monitores (`/etc/databbix/dbs` por defecto) define un monitor. Los ficheros con otra extensión se ignoran, así que para desactivar un monitor basta con renombrar su fichero (por ejemplo, a `mydb.conf.off`).

6.1 Estructura del fichero

Un fichero de monitor tiene dos tipos de secciones:

- **Secciones comunes**, iguales para todos los motores: `[General]`, `[Zabbix]`, `[Data]` y `[Log]`. Se describen en este capítulo.
- **Una sección del motor**, con los datos de conexión con la fuente de datos. Su nombre coincide con el del motor elegido en `General.Engine`. Se describe en el capítulo de cada motor.

Sección	Ámbito	Contenido
<code>[General]</code>	común	Nombre del host, motor, ficheros de consultas, variables de entorno, reintentos y caché.
<code>[Zabbix]</code>	común	Servidor Zabbix y puerto pasivo.
<code>[Data]</code>	común	Módulos de consultas y tolerancia a errores.
<code>[Log]</code>	común	Log propio del monitor.
<code>[Database]</code>	Database	Driver, cadena de conexión y credenciales (capítulo 9).
<code>[Webservice]</code>	Webservice	URL base, autenticación y TLS (capítulo 10).
<code>[MQTT]</code>	MQTT	Broker, credenciales y suscripción (capítulo 11).

Esqueleto de un fichero de monitor:

```
# ---- Secciones comunes -----
[General]
Name = MIHOST
Engine = Database           # Database / Webservice / MQTT

[Zabbix]
Server = zabbix.empresa.com
Port = 10051

[Data]
Module = ...
```

```
[Log]
LogLevel = 4

# ---- Sección del motor (solo una, la de General.Engine) -----
[Database]
...
```

6.2 Sección [General]

Parámetro	Defecto	Descripción
Name	<i>obligatorio</i>	Nombre del host en Zabbix. Tiene que coincidir <i>exactamente</i> con el campo <i>Host name</i> del host en Zabbix.
Engine	Database	Motor: Database, Webservice o MQTT. Determina qué sección de conexión se lee y qué XML se admiten.
QueryFile	el global	Fichero(s) XML de consultas. Si aparece, sustituye por completo a la lista global. Se puede repetir.
Environment	—	Variables Nombre=Valor. Tienen prioridad sobre las de <code>dataabbix.conf</code> y sobre los valores por defecto del XML. Se puede repetir.
RetryOnError	60	Segundos entre reintentos tras un error de conexión con la fuente o con Zabbix.
MaxCache	10	Vida máxima en segundos de un resultado en caché.

6.3 Sección [Zabbix]

Parámetro	Defecto	Descripción
Server	<i>obligatorio</i>	Nombre DNS o IP del servidor (o proxy) Zabbix.
Port	<i>obligatorio</i>	Puerto del servidor, normalmente 10051.
Refresh	300	Segundos entre peticiones de la lista de ítems activos.
MaxItemsPerPacket	100	Número máximo de valores por envío.
PassivePort	0	Puerto TCP para comprobaciones pasivas. Con 0 no se abre ningún puerto. Si se usa, debe ser distinto en cada monitor.

El puerto pasivo responde a los ítems `agent.ping`, `agent.version`, `agent.hostname` y `tez.ora.comnstatus` (1 si la conexión con la fuente está activa, 0 si no). Para cualquier otro ítem devuelve `ZBX_NOTSUPPORTED`. El puerto pasivo sirve para comprobar desde Zabbix que el monitor está vivo.

6.4 Sección [Data]

Parámetro	Defecto	Descripción
Module	<i>obligatorio</i>	Módulo de consultas que usa este monitor (DB, Instance, PG, JDE...). Se puede repetir.
MaxErrors	5	Número máximo de ejecuciones fallidas consecutivas. Al superarlo, el monitor se detiene. 0 desactiva el límite.
ConnLostOnDefault CtxError	yes	Solo motor Database: si falla el cambio al contexto por defecto, da la conexión por perdida y reconecta (sección 9.5).

Nota. En configuraciones antiguas del motor Database, `Module` y `MaxErrors` aparecen en la sección `[Database]`. Esa forma sigue siendo válida, pero si existen en `[Data]`, prevalecen los de `[Data]`.

6.5 Sección [Log]

Admite los mismos parámetros que la sección `[Log]` de `databbix.conf` (sección 5.2.2). Si no se indican, hereda los valores globales. El fichero de log por defecto es `<Name>.log`, dentro de `LogPath`.

6.6 Caracteres especiales

El carácter `#` inicia un comentario. Si un valor necesita contener `#` (por ejemplo, un usuario común de Oracle multitenant o el comodín de una suscripción MQTT), se escribe con la secuencia `\043`:

```
# Usuario c##zabbix
UserName = c\043\043zabbix
```

Seguridad. Los ficheros de monitor contienen contraseñas en claro. Deben pertenecer a `root:databbix` con permisos `0640`, igual que los instala el RPM: el demonio puede leerlos, pero no modificarlos. El usuario `databbix` no debe poder escribir en su propia configuración (ficheros `.conf`, XML de consultas ni sus directorios), porque los XML pueden contener código que el demonio ejecuta.

Capítulo 7

Ficheros XML de consultas: parte común

Este capítulo describe la parte del formato de los ficheros de consultas que es igual para todos los motores. Lo que depende de cada motor (cómo se escribe la operación sobre la fuente y de dónde sale cada valor) se describe en los capítulos 9, 10 y 11.

7.1 Qué es común y qué depende del motor

Un fichero XML de consultas tiene siempre la misma estructura: un elemento raíz `<Queries>` con declaraciones globales y una lista de consultas. Cada consulta combina elementos comunes con elementos propios del motor:

```
<Queries engine="...">                                <!-- común -->

  <environment name="..." default="..." />           <!-- común -->
  <script language="...">...</script>                 <!-- común -->
  <context .../>                                          <!-- solo Database -->
  <authentication ...>...</authentication>              <!-- solo Webservice -->

  <query name="..." module="..." ...>               <!-- común + atributos del motor -->
    <sql>...</sql>                                       <!-- solo Database -->
    <url>...</url>                                       <!-- solo Webservice -->
    <parameter order="..." .../>                      <!-- común + atributos del motor -->
    <item name="..." .../>                             <!-- común + atributos del motor -->
    <field name="..." .../>                           <!-- común + atributos del motor -->
    <script>...</script>                                <!-- común -->
  </query>

</Queries>
```

La tabla 7.1 resume qué elementos y atributos existen en cada motor.

Tabla 7.1: Elementos y atributos por motor

Elemento / atributo	Database	Webserv.	MQTT	Sección
<i>Nivel <Queries></i>				
<Queries engine query databbix>	sí	sí	sí	7.3
<environment>	sí	sí	sí	7.4
<script>	sí	sí	sí	7.5

Elemento / atributo	Database	Webserv.	MQTT	Sección
<context>	sí	—	—	9.5
<authentication>	—	sí	—	10.6
<i>Consulta <query></i>				
name, module, maxCache, maxExecTime	sí	sí	sí	7.6
discovery="yes"	sí	sí	sí	7.9
type="Constant"	sí	sí	sí	7.10
select="FILTER BIND"	sí	—	—	9.4
type="REST SOAP", transport, encode	—	sí	—	10.3
<sql>, <default>	sí	—	—	9.3
<url>, <base>, <transform>	—	sí	—	10.3
<method>, <argument>	—	sí (SOAP)	—	10.4
<i><parameter></i>				
order, value, type	sí	sí	sí	7.8
context	sí	—	—	9.5
column, bind	sí	—	—	9.4
path, urlparameter	—	sí	—	10.3
<i><item></i>				
name, transform, default, built_in	sí	sí	sí*	7.7
value	solo en consultas constantes			7.10
column	sí	—	—	9.4
path, type	—	sí	—	10.3
<i><field> (descubrimientos)</i>				
name	sí	sí	sí	7.9
column, context	sí	—	—	9.4.4
path	—	sí	—	10.3.4

* En MQTT los nombres de ítem son fijos (capítulo 11).

7.2 Reglas generales

- Los nombres de elementos y atributos no distinguen mayúsculas de minúsculas (<Queries> y <queries> son equivalentes).
- Un atributo o elemento desconocido, o que no corresponde al motor, es un error: el monitor no arranca y el error aparece en su log.
- Conviene escribir el texto de las consultas y los scripts dentro de una sección <![CDATA[...]]> para no tener que escapar <, > ni &.
- Si un monitor carga varios ficheros (varias líneas QueryFile), todos deben ser del mismo motor. Los nombres de consulta y de ítem deben ser únicos en el conjunto de ficheros.

7.3 El elemento <Queries>

Es el elemento raíz y admite estos atributos:

Atributo	Descripción
engine	Motor: Database (defecto), Webservice o MQTT. Debe coincidir con <code>General.Engine</code> del monitor.
query	Versión mínima del formato de consultas que exige el fichero. La versión actual es 2.5.0.
databbix	Versión mínima de Databbix que exige el fichero.

Si la versión instalada es menor que la exigida, el monitor no arranca y lo indica en el log.

7.4 Variables de entorno: <environment>

Las variables de entorno permiten parametrizar un XML para reutilizarlo en varias instalaciones: nombres de esquema, bases de datos, contextos...

```
<environment name="SvmSchema" default="svm920"/>
```

Se usan con la sintaxis `${Nombre}`. Dónde se sustituyen depende del motor: en Database, en el texto SQL, en los atributos de contextos, `<default>` y `<parameter>`, y en las condiciones de `<when>` con `environment('Nombre')`; en Webservice, en el texto de los elementos `<header>` de las autenticaciones HTTPAuth.

```
<sql><![CDATA[
    select count(*) as n from ${SvmSchema}.f986110
]]>
</sql>
```

El valor de una variable se toma de la primera de estas fuentes que la defina:

1. `Environment` en el fichero `.conf` del monitor.
2. `Environment` en `databbix.conf`.
3. El atributo `default` del `<environment>` en el XML.

Por ejemplo, para que un monitor use el esquema `svm910`:

```
[General]
Environment = SvmSchema=svm910
```

Una variable sin definir se sustituye por una cadena vacía.

7.5 Scripts: <script>

Un `<script>` contiene funciones en JavaScript o Python que se usan para transformar valores o respuestas. Se puede poner en el nivel de `<Queries>`, y entonces está disponible en todas las consultas, o dentro de un `<query>`, y entonces solo está disponible en esa consulta.

```
<script language="Javascript"><![CDATA[
    function toUnix(fecha) {
        return Math.round(Date.parse(fecha) / 1000);
    }
]]>
</script>
```

Todos los scripts de una colección de consultas deben usar el mismo lenguaje. El capítulo 12 explica cómo programarlos.

7.6 La consulta: <query>

7.6.1 Atributos comunes

Atributo	Descripción
name	<i>Obligatorio.</i> Nombre único de la consulta. Aparece en el log y en <code>queryexplorer</code> .
module	<i>Obligatorio.</i> Módulo al que pertenece la consulta.
discovery	yes para una consulta de descubrimiento (LLD).
type	Constant para una consulta de valores constantes. En el motor Webservice, además, REST o SOAP.
maxCache	Vida máxima en caché de los resultados de esta consulta, en segundos. La vida efectiva es el menor valor entre este y <code>MaxCache</code> del monitor.
maxExecTime	Umbral en segundos a partir del cual se registra un aviso de ejecución lenta. Si no se indica, se usa <code>MaxQueryExecTimeLog</code> .

7.6.2 Tipos de consulta por motor

Motor	Tipo	Qué hace
Todos	Constante	Devuelve valores fijos del XML.
	Descubrimiento	Genera el JSON de una regla LLD de Zabbix.
Database	Básica	Una sentencia SQL que devuelve una fila.
	Filtro	Una sentencia que devuelve muchas filas; la clave elige la fila.
	Bind	La clave se pasa a la sentencia como variable.
Webservice	REST	Llamada HTTP con respuesta JSON; la clave se puede pasar en la URL o usar como filtro.
	SOAP	Llamada SOAP con argumentos fijos.
MQTT	Valor	Último valor publicado en un <i>topic</i> .

7.7 El elemento <item>

Cada <item> declara un ítem de Zabbix que produce la consulta. Estos son sus atributos comunes; cada motor añade los atributos que indican de dónde sale el valor (`column` en Database, `path` y `type` en Webservice).

Atributo	Descripción
name	<i>Obligatorio.</i> Clave del ítem en Zabbix, sin parámetros. Debe ser única entre todas las consultas cargadas.
transform	Función que transforma el valor antes de enviarlo, con la forma <code>Lenguaje:función</code> (por ejemplo, <code>Javascript:toUnix</code>). Véase el capítulo 12.
default	Valor que se envía si el dato buscado no existe (la fila de una consulta de filtro o la ruta de una respuesta JSON).
built_in	Ítem interno de la consulta: <code>exec_time</code> (tiempo de ejecución en segundos) o <code>exec_success</code> (1 si la última ejecución terminó bien, 0 si no).

Un mismo dato puede alimentar varios ítems con nombres distintos, por ejemplo para mantener una clave antigua y una nueva a la vez:

```
<item name="tez.ora.queryversion" column="version"/>
<item name="ts.db.queryversion" column="version"/>
```

7.8 El elemento <parameter>

Relaciona cada parámetro de la clave de Zabbix con la consulta. Estos son los atributos comunes; el atributo que dice *para qué* se usa el parámetro depende del motor (`column`, `bind` o `context` en Database; `path` o `urlparameter` en Webservice).

Atributo	Descripción
<code>order</code>	Posición del parámetro en la clave del ítem, empezando en 1.
<code>value</code>	Valor fijo, en lugar de tomarlo de la clave. Es incompatible con <code>order</code> .
<code>type</code>	STRING (defecto) o INT. Con INT, el valor se compara o se envía como número.

Los parámetros de la clave se separan solo por comas y se eliminan los espacios de los extremos. No se interpretan comillas, así que un parámetro no puede contener comas.

7.9 Consultas de descubrimiento

En todos los motores, una consulta con `discovery="yes"` produce el JSON de una regla de descubrimiento de bajo nivel de Zabbix:

```
{"data": [ {"#MACRO1": "valor", "#MACRO2": "valor", ... } ]}
```

- Cada entidad descubierta (una fila, un elemento de un array JSON, un *topic*) es un elemento de `data`.
- Cada `<field>` define una macro: el atributo `name` es el nombre de la macro sin `{#}` ni `}`, y el atributo propio del motor indica de dónde sale su valor.
- Los `<item>` de un descubrimiento solo llevan `name` y, opcionalmente, `transform`; en ese caso la función recibe el JSON completo (sección 12.5).

7.10 Consultas constantes

Con `type="Constant"`, la consulta no accede a la fuente de datos: cada `<item>` devuelve el valor de su atributo `value`. Funcionan con cualquier motor y se suelen usar para publicar la versión del fichero de consultas:

```
<query name="JDEVersion" module="JDE" type="Constant">
  <item name="ts.jde.queryversion" value="$Id: manual-databbix.tex 2674 2026-09-30
    20:30:56Z bruno $" />
</query>
```


Capítulo 8

Configuración en Zabbix

Para que Databbix recoja datos de un host, en Zabbix hay que hacer lo siguiente:

1. Crear el host con el *Host name* igual al valor de `General.Name` del monitor. Las mayúsculas cuentan.
2. Crear los ítems (o importar una plantilla, por ejemplo las de `zabbix_templates.xml`) de tipo **Zabbix agent (active)**. La clave de cada ítem debe coincidir con el atributo `name` de un `<item>` de los XML.
3. Si se ha configurado `PassivePort`, añadir una interfaz de agente con ese puerto para poder usar ítems pasivos como `agent.ping`.
4. Permitir que el servidor o proxy Zabbix acepte conexiones del equipo de Databbix en el puerto 10051.

8.1 Claves de ítem

La clave que se escribe en Zabbix tiene la forma:

```
nombre.del.item[parámetro1,parámetro2,...]
```

- `nombre.del.item` selecciona el `<item>` del XML y, con él, la consulta que se ejecuta.
- Los parámetros se asignan por posición a los `<parameter>` de la consulta según su atributo `order` (el primero es `order="1"`).
- Los parámetros se separan solo por comas y se eliminan los espacios en los extremos. **No se interpretan comillas**, así que un parámetro no puede contener comas.

Ejemplos:

<code>tez.ora.sessions</code>	consulta sin parámetros
<code>tez.ora.tspace.pctfree[USERS]</code>	consulta FILTER por tablespace
<code>tez.pg.param[max_connections]</code>	consulta BIND
<code>tez.mssql.fgspace.used[ventas,PRIMARY]</code>	contexto (base de datos) + filtro
<code>tez.ora.tspace.discovery</code>	regla de descubrimiento

8.2 Ítems internos

Las claves siguientes no necesitan ninguna consulta en el XML:

Clave	Valor
<code>agent.ping</code>	Siempre 1.
<code>agent.version</code>	Versión de Databbix, por ejemplo <code>databbix-2.5.0-2639</code> .
<code>agent.hostname</code>	Nombre del host (<code>General.Name</code>).
<code>tez.ora.ping</code>	Código de estado de la conexión con la fuente de datos (0 si no hay conexión).

Cualquier otra clave que no esté definida en los XML se devuelve a Zabbix como *no soportada*.

Parte III

Motores

Capítulo 9

Motor Database

El motor **Database** obtiene los datos ejecutando sentencias SQL sobre una base de datos. Es el motor por defecto: un monitor sin **General.Engine** usa este motor.

9.1 Configuración: sección [Database]

Parámetro	Descripción
DBDriver	<i>Obligatorio.</i> Driver: Oracle , PGsql u ODBC .
ConnectionString	<i>Obligatorio.</i> Cadena de conexión. Su formato depende del driver (sección 9.2).
UserName	<i>Obligatorio.</i> Usuario de la base de datos.
Password	<i>Obligatorio.</i> Contraseña.
ConnectOptions	Opciones especiales. La única admitida es SYSDBA (necesaria para ASM).

El usuario de base de datos solo necesita permisos de lectura sobre las vistas que consultan los XML.

9.2 Drivers

El parámetro **DBDriver** elige el driver con el que se conecta el monitor. Cada driver interpreta a su manera la cadena de conexión, admite unos contextos concretos y ofrece unos datos de ejecución propios.

DBDriver	Bases de datos	Contextos	Cadena de conexión
Oracle	Oracle (OCI)	Container	Alias de tnsnames.ora , <i>Easy Connect</i> o descriptor completo.
PGsql	PostgreSQL (libpq)	Database	Pares clave=valor de libpq.
ODBC	Cualquiera con driver ODBC (SQL Server, DB2/400...)	Database	Cadena de SQLDriverConnect : DSN=nombre o DRIVER={...};...

Nota. PostgreSQL y Oracle también se pueden usar a través del driver ODBC, con un driver ODBC del fabricante o de terceros. Aun así, se recomienda usar el driver nativo (PGsql u Oracle) siempre que sea posible: es el que conoce los contextos de cada base de datos (*Database* en PostgreSQL, *Container* en Oracle) y el que ofrece sus datos de ejecución específicos, como `ORACLE.CONTAINER`.

En todos los drivers, las variables de enlace de las consultas `BIND` se escriben siempre con la forma `:nombre`. El driver las convierte a la sintaxis de cada base de datos (`$1`, `$2...` en PostgreSQL, `?` en ODBC).

9.2.1 Driver Oracle

Cadena de conexión. Se pasa tal cual al cliente de Oracle, así que admite cualquier forma que este acepte:

```
ConnectionString = oraprod                                # alias de tnsnames.ora
ConnectionString = dbhost.empresa.com:1521/ORAPROD      # Easy Connect
ConnectionString = (DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(HOST=dbhost)(PORT=1521)
    )(CONNECT_DATA=(SERVICE_NAME=ORAPROD)))
```

Para usar alias, la variable `TNS_ADMIN` del servicio debe apuntar al directorio del `tnsnames.ora`.

ConnectOptions. `SYSDBA` conecta con privilegios de administrador (necesario para ASM). Sin valor, conexión normal.

Contextos. *Container*: el contenedor (PDB) activo, que se cambia con `ALTER SESSION SET CONTAINER`. Solo está disponible en Oracle 12c o posterior, en una base de datos multitenant y conectando al contenedor raíz (`CDB$ROOT`). En cualquier otro caso, el driver no admite contextos y `ORACLE.CONTAINER` vale 0.

Variables de enlace. Nativas de Oracle (`:nombre`).

9.2.2 Driver PGsql

Cadena de conexión. Pares `clave=valor` de libpq separados por espacios (`host`, `port`, `dbname`, `sslmode`, `connect_timeout...`). El usuario y la contraseña se añaden desde `UserName` y `Password`; no los ponga en la cadena.

```
ConnectionString = host=pg01.empresa.com port=5432 dbname=postgres sslmode=
require
```

ConnectOptions. No se usa.

Contextos. *Database*: la base de datos activa. Como PostgreSQL no permite cambiar de base de datos en una conexión, el driver abre una conexión por cada base de datos que se usa (sustituyendo el `dbname` de la cadena) y las mantiene abiertas.

Variables de enlace. Se convierten a `$1`, `$2...`

Atención. Incluya siempre `dbname` en la cadena de conexión: el driver lo usa como base de datos por defecto y lo sustituye al cambiar de contexto. La contraseña se añade a la cadena sin comillas, así que no puede contener espacios.

9.2.3 Driver ODBC

Cadena de conexión. Se pasa a `SQLDriverConnect` añadiendo `;UID=...;PWD=...` con el usuario y la contraseña. Puede referirse a una fuente de datos de `odbc.ini` o describir la conexión completa:

```
ConnectionString = DSN=MSSQL_PROD
ConnectionString = DRIVER={FreeTDS};SERVER=mssql.empresa.com;PORT=1433;DATABASE=master
```

Para usar DSN, la variable `ODBCINI` del servicio debe apuntar al fichero `odbc.ini`.

ConnectOptions. No se usa.

Contextos. `Database`: la base de datos (catálogo) activa, que se cambia en la misma conexión con el atributo `ODBC SQL_ATTR_CURRENT_CATALOG`. Funciona con los drivers que lo admiten, como los de SQL Server.

Variables de enlace. Se convierten a ?.

Nota. La contraseña se añade a la cadena sin comillas, así que no puede contener el carácter `;`.

9.2.4 Datos de ejecución

Los datos de ejecución se consultan en las condiciones de `<when>` con `runtime('NOMBRE')` (sección 9.6) y en `queryexplorer` con `getRuntimeValue`.

Nombre	Driver	Valor
<code>DATABBIX.PROGRAM_NAME</code>	todos	<code>databbix</code> , o <code>queryexplorer</code> en esa utilidad.
<code>CONN.TYPE</code>	todos	<code>Database</code> .
<code>DB.DRIVER</code>	Oracle	Oracle.
	PGsql	PostgreSQL.
	ODBC	ODBC.
<code>DB.DATABASE</code>	Oracle	Oracle.
	PGsql	PostgreSQL.
	ODBC	Nombre del producto que informa el driver ODBC (<code>SQL_DBMS_NAME</code>): por ejemplo, Microsoft SQL Server o DB2/400 SQL.
<code>DB.VERSION</code>	Oracle	Versión del servidor, con la forma <code>mayor.menor.actualización</code> .
	PGsql	Parámetro <code>server_version</code> del servidor, por ejemplo 15.4.
	ODBC	Versión que informa el driver ODBC (<code>SQL_DBMS_VER</code>).
<code>ORACLE.CONTAINER</code>	Oracle	1 si el contexto <code>Container</code> está disponible; 0 si no.

Así, un mismo XML puede elegir el SQL según el producto (`runtime('DB.DATABASE') = 'Microsoft SQL Server'`), según la versión (`major(runtime('DB.VERSION')) >= 12`) o según si la base de datos es multitenant (`runtime('ORACLE.CONTAINER') = '1'`).

9.3 Consultas SQL: elementos propios del motor

Además de los elementos comunes (capítulo 7), las consultas del motor Database usan:

Elemento / atributo	Descripción
<code><sql></code>	Texto de la sentencia. Obligatorio salvo en las consultas constantes. Admite variables <code>\${...}</code> y alternativas con <code><multiple></code> (sección 9.6).
<code><sql context="C"></code>	Solo en descubrimientos: sentencia que devuelve los valores del contexto <code>C</code> que hay que recorrer (sección 9.4.4).
<code>select</code>	Atributo de <code><query></code> : <code>FILTER</code> o <code>BIND</code> . Sin él, la consulta es básica.
<code><item column="..."></code>	Columna del resultado de la que sale el valor. No distingue mayúsculas de minúsculas.
<code><parameter column></code>	En consultas <code>FILTER</code> : columna que se compara con el parámetro.
<code><parameter bind></code>	En consultas <code>BIND</code> : variable de enlace del SQL.
<code><parameter context></code>	Contexto en el que se ejecuta la consulta.
<code><default></code>	Contexto fijo de la consulta (sección 9.5).
<code><field column context></code>	En descubrimientos: de dónde sale el valor de la macro.
<code><context></code>	En <code><Queries></code> : declaración de un contexto.

Nota. Conviene dar siempre un alias explícito (`as nombre`) a las columnas calculadas, y usar esos alias en `column`.

9.4 Tipos de consulta

Los ejemplos de esta sección muestran, para cada tipo, el resultado de la sentencia SQL en un momento dado y el valor que obtiene cada clave de Zabbix a partir de él.

9.4.1 Consulta básica

La sentencia devuelve **exactamente una fila**. Cada `<item>` toma el valor de una columna.

```
<query name="PGWriter" module="PG">
  <sql><![CDATA[
    select checkpoints_timed, checkpoints_req from pg_stat_bgwriter
  ]]>
</sql>
<item name="tez.pg.checkpointtimed" column="checkpoints_timed"/>
<item name="tez.pg.checkpointreq" column="checkpoints_req"/>
</query>
```

Resultado de la sentencia

checkpoints_timed	checkpoints_req
1842	37

Valores obtenidos

Clave	Valor
tez.pg.checkpointtimed	1842
tez.pg.checkpointreq	37

Si la sentencia devuelve cero filas o más de una, todos los ítems de la consulta pasan a *no soportados*.

9.4.2 Consulta de filtro (select="FILTER")

La sentencia devuelve **muchas filas**. Cada `<parameter>` con `column` relaciona un parámetro de la clave con una columna, y cada ítem se queda con la fila cuyas columnas coinciden con los parámetros de su clave. La sentencia se ejecuta *una sola vez* para todos los ítems, así que es la forma más eficiente de vigilar colecciones de objetos (tablespaces, bases de datos, trabajos...).

```
<query name="PGSessions" module="PG" select="FILTER">
  <sql><![CDATA[
    select datname,
           count(st.pid) as n,
           sum(case st.state when 'active' then 1 else 0 end) as active
    from pg_database db
         left outer join pg_stat_activity st using (datname)
    where datname not like 'template%'
    group by datname
  ]]>
</sql>
<parameter column="datname" order="1" type="STRING"/>
<item name="tez.pg.sessions" column="n"/>
<item name="tez.pg.sessionsactive" column="active" default="0"/>
</query>
```

Resultado de la sentencia

datname	n	active
postgres	3	1
ventas	12	4
rrhh	0	0

Valores obtenidos

Clave	Fila elegida	Valor
tez.pg.sessions[ventas]	ventas	12
tez.pg.sessionsactive[ventas]	ventas	4
tez.pg.sessions[rrhh]	rrhh	0
tez.pg.sessions[compras]	ninguna	<i>no soportado</i>
tez.pg.sessionsactive[compras]	ninguna	0 (atributo default)
tez.pg.sessions[Ventas]	ninguna	<i>no soportado</i>

Las cuatro primeras claves se resuelven con una única ejecución de la sentencia. La comparación es exacta: distingue mayúsculas y minúsculas (*Ventas* no coincide con *ventas*) y solo elimina los espacios de los extremos *del parámetro*. Si una columna tiene espacios de relleno (tipo `char`), recórtelos en el SQL con `trim()`.

Varias columnas de filtro

Con varios parámetros de filtro, la clave lleva un parámetro por cada uno, en el orden que indica `order`, y la fila tiene que coincidir en todos:

```
<query name="MSStateRepl" module="MSSQL" select="FILTER">
  <sql><![CDATA[ exec sp_replmonitorhelppublication ]]></sql>
  <default context="Database" value="distribution"/>
  <parameter column="publisher_db" order="1" type="STRING"/>
  <parameter column="publication" order="2" type="STRING"/>
  <item name="tez.mssql.repl.status" column="status"/>
</query>
```

Resultado de la sentencia

publisher_db	publication	status
ventas	pub_pedidos	3
ventas	pub_clientes	2
compras	pub_pedidos	6

Valores obtenidos

Clave	Valor
tez.mssql.repl.status[ventas, pub_pedidos]	3
tez.mssql.repl.status[compras, pub_pedidos]	6
tez.mssql.repl.status[ventas]	<i>no soportado</i> (falta un parámetro)

Parámetros numéricos

Con `type="INT"`, el parámetro y la columna se comparan como números. Así, la clave no tiene que reproducir el formato exacto de la columna:

```
<parameter column="job_id" order="1" type="INT"/>
<item name="local.job.status" column="status"/>
```

Resultado de la sentencia y valores obtenidos

job_id	status	Clave	Valor
7	OK	local.job.status[7]	OK
42	ERROR	local.job.status[042]	ERROR

Con `type="STRING"`, la clave `local.job.status[042]` no encontraría la fila, porque el texto 042 es distinto de 42.

9.4.3 Consulta con variables de enlace (`select="BIND"`)

El parámetro de la clave se pasa a la sentencia como variable de enlace. La sentencia se ejecuta *una vez por cada clave distinta* y debe devolver exactamente una fila.

```
<query name="PGParameter" module="PG" select="BIND">
  <sql><![CDATA[
    select setting from pg_settings where name = :name
  ]]>
</sql>
<parameter bind=":name" order="1" type="STRING"/>
<item name="tez.pg.param" column="setting"/>
</query>
```

Ejecuciones y valores obtenidos

Clave	Valor de :name	Valor
tez.pg.param[max_connections]	'max_connections'	200
tez.pg.param[work_mem]	'work_mem'	4096
tez.pg.param[no_existe]	'no_existe'	<i>no soportado</i> (0 filas)

La variable se escribe siempre como :nombre, sea cual sea el driver (sección 9.2). Con type="INT", el valor se enlaza como número.

Atención. Como se ejecuta una vez por clave, una consulta BIND es mucho más costosa que una FILTER cuando hay muchas claves. Úsela solo cuando el conjunto de valores posibles sea muy grande y se consulten pocos.

9.4.4 Consulta de descubrimiento

Cada fila del resultado es una entidad descubierta, y cada <field> toma el valor de una columna (column) o de un contexto (context).

```
<query name="TSDiscover" module="DB" discovery="yes">
  <sql><![CDATA[
    select tablespace_name, contents
    from dba_tablespace where status = 'ONLINE'
  ]]>
</sql>
<item name="tez.ora.tspace.discovery"/>
<field name="TSNAME" column="tablespace_name"/>
<field name="CONTENTS" column="contents"/>
</query>
```

Resultado de la sentencia

tablespace_name	contents
SYSTEM	PERMANENT
USERS	PERMANENT
TEMP	TEMPORARY

Valor del ítem tez.ora.tspace.discovery:

```
{"data": [
  {"#TSNAME": "SYSTEM", "#CONTENTS": "PERMANENT"},
  {"#TSNAME": "USERS", "#CONTENTS": "PERMANENT"},
  {"#TSNAME": "TEMP", "#CONTENTS": "TEMPORARY"}
]}
```

En Zabbix, el prototipo `tez.ora.tspace.pctfree[#{TSNAME}]` genera los ítems `tez.ora.tspace.pctfree[SYSTEM]`, `[USERS]` y `[TEMP]`, que resuelve una consulta `FILTER` con `parameter column="tablespace_name"`.

Los descubrimientos solo admiten parámetros de contexto.

Descubrimiento a través de contextos

Un descubrimiento puede recorrer todos los valores de un contexto: por ejemplo, listar los *filegroups* de *todas* las bases de datos de un SQL Server. Para ello se añade un `<sql>` con atributo `context` que devuelve, en su primera columna, los valores que hay que recorrer. La sentencia principal se ejecuta una vez en cada uno:

```
<query name="MSDiscoverFG" module="MSSQL" discovery="yes">
  <sql context="Database"><![CDATA[
    select name from sys.databases where name not in ('model')
  ]]>
</sql>
<sql><![CDATA[
  select name from sys.filegroups
]>
</sql>
<item name="tez.mssql.fg.discovery"/>
<field name="DBNAME" context="Database"/>
<field name="FGNAME" column="name"/>
</query>
```

Resultado de las sentencias

Sentencia de contexto	Sentencia principal en cada base de datos	
name	base de datos	name
master	master	PRIMARY
ventas	ventas	PRIMARY
	ventas	HISTORICO

Valor del ítem `tez.mssql.fg.discovery`:

```
{"data": [
  {"#{DBNAME}": "master", "#{FGNAME}": "PRIMARY"},
  {"#{DBNAME}": "ventas", "#{FGNAME}": "PRIMARY"},
  {"#{DBNAME}": "ventas", "#{FGNAME}": "HISTORICO"}
]}
```

Los valores de contexto en los que no se puede entrar (por ejemplo, una base de datos fuera de línea) se saltan y quedan registrados en el log.

9.5 Contextos

Un contexto es un estado de la conexión que se puede cambiar entre consultas, como la base de datos activa en SQL Server o PostgreSQL o el contenedor (PDB) en Oracle multitenant. Así, un único monitor con una sola conexión puede consultar varias bases de datos de un mismo servidor. Los contextos solo existen en el motor Database.

9.5.1 Declaración

Los contextos se declaran en el nivel de <Queries>:

```
<environment name="PGAdminDB" default="postgres"/>
<context name="Database" module="PG" default="{PGAdminDB}"/>
```

Atributo	Descripción
name	<i>Obligatorio.</i> Nombre del contexto. El driver debe saber cambiarlo; los nombres habituales son Database (SQL Server, PostgreSQL) y Container (Oracle multitenant).
module	<i>Obligatorio.</i> Módulo al que pertenece.
default	<i>Obligatorio.</i> Valor por defecto. Admite variables \${...}.

9.5.2 Uso en consultas

Hay tres formas de elegir el contexto de una consulta:

1. **Por defecto.** Si la consulta no dice nada, se ejecuta con el valor `default` del contexto.
2. **Fijo para la consulta,** con <default>:

```
<default context="Database" value="msdb"/>
```

Si `value` vale `__IGNORE__` (por ejemplo, a través de una variable de entorno sin configurar), el elemento se ignora y se mantiene el valor por defecto.

3. **Tomado de la clave del ítem,** con un parámetro de contexto:

```
<parameter context="Database" order="1"/>
```

9.5.3 Ejemplo: filtro dentro de cada base de datos

Con un parámetro de contexto, el primer parámetro de la clave elige la base de datos y los siguientes, la fila:

```
<context name="Database" module="MSSQL" default="master"/>

<query name="MSFGSpace" module="MSSQL" select="FILTER">
  <sql><![CDATA[
    select g.name as filegroup_name,
           sum(cast(fileproperty(f.name, 'SPACEUSED') as bigint) * 8192) as used
    from sys.database_files f
           inner join sys.filegroups g on g.data_space_id = f.data_space_id
    group by g.name
  ]]>
</sql>
<parameter context="Database" order="1"/>
<parameter column="filegroup_name" order="2" type="STRING"/>
<item name="tez.mssql.fgspace.used" column="used"/>
</query>
```

Resultado de la sentencia en cada base de datos

base de datos	filegroup_name	used
master	PRIMARY	5242880
ventas	PRIMARY	73400320
ventas	HISTORICO	524288000

Valores obtenidos

Clave	Contexto	Valor
tez.mssql.fgspace.used[ventas,HISTORICO]	ventas	524288000
tez.mssql.fgspace.used[ventas,PRIMARY]	ventas	73400320
tez.mssql.fgspace.used[master,PRIMARY]	master	5242880

Databbix agrupa las ejecuciones por contexto: entra en **ventas** una sola vez, ejecuta la sentencia y resuelve las dos primeras claves con el mismo resultado. Si no puede entrar en un contexto, marca como *no soportados* los ítems de ese contexto. Si el contexto que falla es el *por defecto* y `ConnLostOnDefaultCtxError = yes`, da la conexión por perdida y reconecta.

9.6 SQL alternativo según la base de datos

Un mismo XML puede servir para distintos productos o versiones de base de datos si el texto SQL se elige al ejecutar, con `<multiple>`:

```
<sql>
  <multiple>
    <when condition="runtime('DB.DATABASE') = 'Oracle'"><![CDATA[
      select count(*) as n from ${SvmSchema}.f986110
      where trim(jcfndfuf2) = :jobname
    ]]>
  </when>
  <when condition="runtime('DB.DATABASE') = 'Microsoft SQL Server'"><![CDATA[
    select count(*) as n from ${SvmSchema}.f986110
    where rtrim(jcfndfuf2) = :jobname
  ]]>
  </when>
  <when><![CDATA[
    -- sin condición: se usa si ninguna anterior se cumple
    select count(*) as n from ${SvmSchema}.f986110
    where jcfndfuf2 = :jobname
  ]]>
  </when>
</multiple>
</sql>
```

Se usa el primer `<when>` cuya condición se cumple. Un `<when>` sin condición se cumple siempre, así que debe ir el último. Si no se cumple ninguna condición, la consulta falla.

Las condiciones son expresiones que pueden usar estas funciones:

Función	Descripción
<code>runtime('X')</code>	Dato de ejecución de la conexión activa, por ejemplo <code>DB.DATABASE</code> o <code>DB.VERSION</code> (sección 9.2.4).
<code>environment('X')</code>	Valor de una variable de entorno.
<code>major('X.Y.Z')</code>	Primer número de una versión, como número.
<code>toscalar('X')</code>	Convierte un texto en número.

Admiten operadores de comparación (=, !=, <, >...) y lógicos (and, or, not). Los datos de ejecución disponibles con cada driver se pueden consultar con el comando `getRuntimeNames` de `queryexplorer`.

9.7 Ejemplos de configuración

9.7.1 Ejemplo: Oracle

```
[General]
Name = ORAPROD

[Zabbix]
Server = zabbix.empresa.com
Port = 10051
PassivePort = 10153

[Data]
Module = DB
Module = Instance

[Database]
DBDriver = Oracle
ConnectionString = oraprod           # alias de tnsnames.ora
UserName = zabbix
Password = secreto
```

9.7.2 Ejemplo: Oracle multitenant

```
[General]
Name = ORACDB
QueryFile = /etc/databbix/query-mt.xml

[Zabbix]
Server = zabbix.empresa.com
Port = 10051

[Data]
Module = MT

[Database]
DBDriver = Oracle
ConnectionString = oracdb
UserName = c\043\043zabbix         # c##zabbix
Password = secreto
```

9.7.3 Ejemplo: ASM

```
[General]
Name = asm

[Zabbix]
Server = zabbix.empresa.com
Port = 10051
```

```
[Data]
Module = ASM

[Database]
DBDriver = Oracle
ConnectionString = asm
ConnectOptions = SYSDBA
UserName = sys
Password = secreto
```

9.7.4 Ejemplo: PostgreSQL

```
[General]
Name = PGPROD
QueryFile = /etc/databbix/query-pg.xml
QueryFile = /etc/databbix/query-pg-10.xml
Environment = PGAdminDB=postgres

[Zabbix]
Server = zabbix.empresa.com
Port = 10051

[Data]
Module = PG

[Database]
DBDriver = PGsql
ConnectionString = host=pg01.empresa.com port=5432 dbname=postgres
UserName = zabbix
Password = secreto
```

9.7.5 Ejemplo: SQL Server por ODBC

Primero se define la fuente de datos en `/etc/databbix/odbc.ini`:

```
[MSSQL_PROD]
Driver=FreeTDS
Description=SQL Server de producción
Server=mssql.empresa.com
Port=1433
Database=master
```

Y después el monitor la referencia por su nombre:

```
[General]
Name = MSSQLPROD
QueryFile = /etc/databbix/query-mssql.xml

[Zabbix]
Server = zabbix.empresa.com
Port = 10051

[Data]
Module = MSSQL

[Database]
DBDriver = ODBC
```

```
ConnectionString = DSN=MSSQL_PROD  
UserName = zabbix  
Password = secreto
```


Capítulo 10

Motor Webservice

El motor **Webservice** obtiene los datos llamando a servicios web por HTTP. Admite dos tipos de servicio, REST y SOAP, que pueden convivir en un mismo fichero de consultas y un mismo monitor.

10.1 REST y SOAP

Los dos tipos comparten la conexión (URL base, TLS, autenticación) y el modo de extraer los valores: en ambos casos la respuesta se convierte en un documento JSON y cada ítem indica una *ruta* dentro de él. Se diferencian en cómo se construye la petición y en el formato original de la respuesta:

	REST (type="REST")	SOAP (type="SOAP")
Petición	GET o POST a una URL.	POST de un sobre XML SOAP con un método y sus argumentos.
Parámetros de la clave	Se pueden enviar en la URL (<code>urlparameter</code>) o usar para elegir un elemento de la respuesta (path con <code><base></code>).	No se usan: los argumentos del método son fijos en el XML.
Respuesta	JSON.	XML SOAP, que Databbix convierte en JSON (sección 10.4.1).
<code>status_code</code> en <code><transform></code>	Código HTTP de la respuesta.	Siempre 0; los errores llegan como un <i>Fault</i> en el propio JSON.
Descubrimientos	Sí.	No.
Autenticación	HTTPAuth, RESTToken, JDE.	Oasis (WS-Security) o HTTPAuth.

10.2 Configuración: sección [Webservice]

Parámetro	Descripción
BaseURL	<i>Obligatorio.</i> URL base. Las URL de las consultas son relativas a ella.
UserName, Password Credentials	Credenciales. Credencial adicional con la forma clave=valor (entorno, rol, método...), según el motor de autenticación. Se puede repetir.
Authenticator	Nombre de un método de autenticación definido en los XML o, si no existe, de un motor de autenticación: HTTPAuth , JDE u Oasis (sección 10.6).
Timeout	Tiempo máximo de espera en milisegundos (defecto: 20000).
AuthPersistFile	Fichero donde se guarda el estado de la autenticación (por ejemplo, el token) entre reinicios.
VerifyCertificate	Verificar el certificado del servidor (defecto: yes).
CertCAFile	Fichero de CA que se usa en lugar del almacén del sistema.
ClientCertificate	Certificado de cliente (PEM) para autenticación TLS.
ClientPrivKey	Clave privada del certificado de cliente. Obligatoria si se usa ClientCertificate .
ClientPrivKeyPass	Contraseña de la clave privada, si está cifrada.
DisableContentEncoding	Desactiva <i>Content-Encoding</i> , para servidores antiguos (defecto: no).

```
[General]
Name = JDEPROD
Engine = Webservice
QueryFile = /etc/databbix/query-jde.xml

[Zabbix]
Server = zabbix.empresa.com
Port = 10051

[Data]
Module = JDE

[Webservice]
BaseURL = http://10.1.1.45:9888/restservice
UserName = monitor
Password = secreto
Authenticator = JDE
```

10.3 Consultas REST

```
<Queries engine="Webservice">

<query name="JDEEntStatus" module="JDE" type="REST" transport="GET" encode="Json">
  <url>entserverinstanceinfo</url>
  <parameter urlparameter="instanceName" order="1" type="STRING"/>
  <item name="ts.jde.enterprise.kerneljobs"
    path="entServerInstanceInfoDetail:[0]:kernelJobs" type="INT"/>
</query>

</Queries>
```

Elemento / atributo	Descripción
<code>type="REST"</code>	Atributo de <code><query></code> . Obligatorio.
<code>transport</code>	<i>Obligatorio</i> . GET o POST.
<code>encode</code>	Formato de la respuesta. Solo se admite <code>Json</code> .
<code><url></code>	<i>Obligatorio</i> . Ruta relativa a <code>BaseURL</code> .
<code><parameter</code> <code>urlparameter></code>	Envía el parámetro de la clave en la URL, como <code>?nombre=valor</code> (sección 10.3.2).
<code><base path></code>	Array de la respuesta que se recorre para buscar el elemento que indica la clave (sección 10.3.3).
<code><parameter path></code>	Con <code><base></code> : campo de cada elemento que se compara con el parámetro de la clave.
<code><item path type></code>	Ruta del valor dentro del JSON (sección 10.3.1) y tipo: <code>INT</code> o <code>STRING</code> . Ambos obligatorios, salvo en ítems <code>built_in</code> .
<code><transform></code>	Función que procesa la respuesta antes de extraer los valores (sección 12.6).

Sin `<transform>`, una respuesta con código HTTP distinto de 200 es un error y todos los ítems de la consulta pasan a no soportados.

10.3.1 Rutas JSON

Una ruta indica cómo llegar a un valor desde la raíz del documento JSON. Se compone de pasos separados por dos puntos (:):

Paso	Significado
<code>nombre</code>	Miembro <code>nombre</code> de un objeto.
<code>[n]</code>	Elemento <code>n</code> de un array, empezando en 0.
<code>[]</code>	Todos los elementos de un array. Solo en <code><base></code> y en el <code><item></code> de un descubrimiento.
<code>*</code>	Todos los miembros de un objeto. Solo en <code><base></code> y en el <code><item></code> de un descubrimiento.

La ruta de un ítem tiene que terminar en un valor simple (texto, número, booleano o `null`). Dado este documento:

```
{
  "server": { "name": "ent01", "state": "RUNNING", "up": true },
  "jobs": [
    { "id": 1, "name": "backup", "status": "OK", "duration": 120 },
    { "id": 2, "name": "purge", "status": "FAILED", "duration": 30 }
  ],
  "counters": { "cpu": 12.5, "memory": 2048 },
  "note": null
}
```

Rutas y valores obtenidos

path	type	Valor
server:name	STRING	ent01
server:up	INT	1
server:up	STRING	true
jobs:[1]:status	STRING	FAILED
jobs:[0]:duration	INT	120
counters:cpu	STRING	12.5
note	STRING	(cadena vacía)
server:host	STRING	<i>no soportado</i> : el miembro no existe
jobs:[5]:status	STRING	<i>no soportado</i> : el array tiene 2 elementos
server	STRING	<i>error</i> : no es un valor simple

En los casos *no soportado*, si el ítem tiene atributo `default`, se envía ese valor. Con `type="INT"`, los booleanos se convierten en 1 y 0.

Nota. Los nombres de miembro no pueden contener dos puntos, porque es el separador de pasos. Si el servicio los usa, reestructure la respuesta con una transformación (sección 12.6).

10.3.2 Parámetros en la URL

Cada `<parameter>` con `urlparameter` añade a la URL un parámetro con ese nombre y el valor de la clave. La consulta se ejecuta una vez por cada clave distinta:

```
<query name="JDEInstanceState" module="JDE" type="REST" transport="GET" encode="Json"
">
  <url>instancestate</url>
  <parameter urlparameter="instanceName" order="1" type="STRING"/>
  <item name="ts.jde.instance.state" path="instanceState" type="STRING"/>
</query>
```

Llamadas y valores obtenidos

Clave	URL llamada	Valor
ts.jde.instance.state[ENT1]	.../instancestate?instanceName=ENT1	RUNNING
ts.jde.instance.state[WEB1]	.../instancestate?instanceName=WEB1	STOPPED

En cada llamada el servicio responde, por ejemplo, `{"instanceState":"RUNNING"}`, y la ruta `instanceState` extrae el valor.

10.3.3 Filtro sobre la respuesta

Cuando el servicio devuelve de una vez la lista de todos los objetos, es más eficiente hacer una sola llamada y elegir el elemento en la respuesta, igual que una consulta `FILTER` de base de datos. Para ello:

- `<base path="...">` indica el array que se recorre (su ruta termina en `[]`);
- cada `<parameter path="...">` indica un campo de cada elemento que se compara con un parámetro de la clave;
- las rutas de los ítems son relativas al elemento elegido.

```
<query name="LocalJobs" module="API" type="REST" transport="GET" encode="Json">
  <url>status</url>
  <base path="jobs:[]" />
  <parameter path="name" order="1" type="STRING"/>
  <item name="local.api.job.status" path="status" type="STRING" default="
UNKNOWN"/>
  <item name="local.api.job.duration" path="duration" type="INT"/>
</query>
```

Con el documento de la sección anterior como respuesta:

Valores obtenidos

Clave	Elemento elegido	Valor
local.api.job.status[backup]	jobs[0]	OK
local.api.job.duration[backup]	jobs[0]	120
local.api.job.status[purge]	jobs[1]	FAILED
local.api.job.status[reindex]	ninguno	UNKNOWN (default)
local.api.job.duration[reindex]	ninguno	<i>no soportado</i>

Las cinco claves se resuelven con una sola llamada al servicio. Con `type="INT"` en el `<parameter>`, la comparación es numérica (por ejemplo, `parameter path="id"` y la clave `...[2]`).

10.3.4 Descubrimientos REST

La ruta del `<item>` de un descubrimiento recorre un array (`[]`) o los miembros de un objeto (`*`). Cada elemento recorrido es una entidad descubierta, y las rutas de los `<field>` son relativas a él:

```
<query name="LocalJobDiscovery" module="API" type="REST"
  transport="GET" encode="Json" discovery="yes">
  <url>status</url>
  <item name="local.api.job.discovery" path="jobs:[]" />
  <field name="JOB" path="name"/>
  <field name="JOBID" path="id"/>
</query>
```

Con el documento de la sección 10.3.1, el valor del ítem es:

```
{"data": [ {{"#JOB"}:"backup", {"#JOBID"}:"1"},
           {"#JOB"}:"purge", {"#JOBID"}:"2"} ]}
```

Recorridos anidados

Una ruta puede tener varios pasos de recorrido. Con esta respuesta:

```
{ "clusters": [
  { "name": "A", "nodes": [ {"host": "a1"}, {"host": "a2"} ] },
  { "name": "B", "nodes": [ {"host": "b1"} ] }
] }
```

Ruta del ítem y entidades descubiertas

path del ítem	Entidades (y field path="..." disponibles)
clusters:[]	2: los clusters A y B (name, nodes:[0]:host...)
clusters:[]:nodes:[]	3: los nodos a1, a2 y b1 (host)

Los `<field>` solo ven el elemento recorrido: en el segundo caso no pueden obtener el nombre del cluster, y al recorrer un objeto con `*` no pueden obtener el nombre de cada miembro. Cuando se necesita, se reestructura la respuesta con una transformación. Por ejemplo, para descubrir los nodos junto con su cluster:

```
<query name="LocalNodeDiscovery" module="API" type="REST"
  transport="GET" encode="Json" discovery="yes">
  <url>clusters</url>
  <script language="Javascript"><![CDATA[
    function aplanar(status_code, result) {
      if (status_code != 200) throw "ZBX_NOTSUPPORTED";
      var res = { nodes: [] };
      for (var i = 0; i < result.clusters.length; i++) {
        var c = result.clusters[i];
        for (var j = 0; j < c.nodes.length; j++)
          res.nodes.push({ cluster: c.name, host: c.nodes[j].host });
      }
      return res;
    }
  ]]>
</script>
<transform language="Javascript" function="aplanar"/>
<item name="local.api.node.discovery" path="nodes: []"/>
<field name="CLUSTER" path="cluster"/>
<field name="HOST" path="host"/>
</query>
```

```
{"data": [ {"#{CLUSTER}": "A", "#{HOST}": "a1"}, {"#{CLUSTER}": "A", "#{HOST}": "a2"},
  {"#{CLUSTER}": "B", "#{HOST}": "b1"} ]}
```

10.4 Consultas SOAP

Una consulta SOAP llama a un método de un servicio SOAP con argumentos fijos y extrae los valores de la respuesta:

```
<query name="GoliveWS" module="golive" type="SOAP" transport="POST">
  <url>FoundationEnvironmentManager</url>
  <method uri="http://oracle.e1.bssv.JP000000/" name="getCurrentUserProfile"
    action="">
    <argument name="environment" type="string" value="JPD920"/>
  </method>
  <item name="golive.ws.addressnumber" path="addressNumber" type="INT" default="0"
  />
  <item name="golive.ws.language" path="userProfile:language" type="STRING"/>
  <item name="golive.ws.exectime" built_in="exec_time"/>
</query>
```

Elemento / atributo	Descripción
<code>type="SOAP"</code>	Atributo de <code><query></code> . Obligatorio.
<code>transport="POST"</code>	Obligatorio.
<code><url></code>	<i>Obligatorio.</i> Ruta del servicio, relativa a BaseURL.
<code><method></code>	Método que se invoca: <code>uri</code> (espacio de nombres del método), <code>name</code> (nombre del método) y <code>action</code> (valor de la cabecera HTTP SOAPAction, puede estar vacío).

Elemento / atributo	Descripción
<argument>	Argumento del método: name , type (string o int) y value . Puede haber varios. Los valores son fijos.
<item path type>	Igual que en REST, sobre el JSON obtenido de la respuesta.
<transform>	Función que procesa el JSON de la respuesta (sección 10.4.2).

Con esta definición, Databbix envía a `BaseURL/FoundationEnvironmentManager` un POST con la cabecera `SOAPAction: ""` y un cuerpo similar a este:

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/" ...>
  <SOAP-ENV:Body>
    <ns1:getCurrentUserProfile xmlns:ns1="http://oracle.e1.bssv.JP000000/">
      <environment xsi:type="xsd:string">JPD920</environment>
    </ns1:getCurrentUserProfile>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Si el monitor usa el motor de autenticación `Oasis`, se añade además una cabecera `SOAP WS-Security` con el usuario y la contraseña del `.conf`.

10.4.1 Conversión de la respuesta SOAP a JSON

Databbix convierte en JSON el *elemento de respuesta del método*, es decir, el primer hijo de `<Body>`. Ese elemento es la raíz del JSON: su nombre no forma parte de las rutas. La conversión sigue estas reglas:

1. Un elemento con elementos hijos se convierte en un objeto. Cada hijo es un miembro con su nombre local, *sin el prefijo del espacio de nombres*.
2. Si un elemento tiene varios hijos con el mismo nombre, ese miembro es un array con todos sus valores, en orden.
3. Un elemento sin hijos se convierte en un *texto*, aunque su contenido sea un número o un booleano, y aunque lleve un atributo `xsi:type`.
4. Los atributos XML se ignoran.
5. El orden de los miembros del objeto puede no coincidir con el del XML. Esto no afecta a las rutas.

Como todos los valores simples son texto, en las funciones de transformación hay que convertirlos para operar con ellos (`Number(r.sessions)` en JavaScript, `int(r['sessions'])` en Python). En los ítems no hace falta: `type="INT"` envía el texto tal cual y convierte `true/false` en `1/0`.

Ejemplo. Respuesta del servicio:

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <soap:Body>
    <ns2:getCurrentUserProfileResponse xmlns:ns2="http://oracle.e1.bssv.JP000000/">
      <addressNumber>4242</addressNumber>
      <active xsi:type="xsd:boolean">true</active>
      <sessions xsi:type="xsd:int">3</sessions>
      <userProfile>
        <language>S</language>
```

```
<dateFormat>DMY</dateFormat>
</userProfile>
<role>ADMIN</role>
<role>USER</role>
</ns2:getCurrentUserProfileResponse>
</soap:Body>
</soap:Envelope>
```

JSON que se obtiene (y que recibe la función de `<transform>`, si la hay):

```
{
  "addressNumber": "4242",
  "active": "true",
  "sessions": "3",
  "userProfile": { "language": "S", "dateFormat": "DMY" },
  "role": [ "ADMIN", "USER" ]
}
```

Rutas y valores obtenidos

path	type	Valor
addressNumber	INT	4242
active	INT	1
sessions	INT	3
userProfile:language	STRING	S
role:[0]	STRING	ADMIN
role:[1]	STRING	USER

Elementos repetidos. Un elemento solo se convierte en array si aparece *más de una vez*. Si el usuario del ejemplo tuviera un único `<role>`, el JSON sería `"role": "ADMIN"` y la ruta `role:[0]` fallaría. Cuando un elemento puede aparecer una o varias veces, normalice la respuesta con una transformación (sección 10.4.2).

Errores SOAP

Si el servicio responde con un *Fault*, el elemento convertido es el propio `<Fault>`:

```
<soap:Body>
  <soap:Fault>
    <faultcode>soap:Server</faultcode>
    <faultstring>User not authorized</faultstring>
  </soap:Fault>
</soap:Body>
```

```
{ "faultcode": "soap:Server", "faultstring": "User not authorized" }
```

Sin transformación, las rutas de los ítems no existen en este JSON y los ítems pasan a no soportados (o toman su valor `default`). Con una transformación se puede detectar el error y decidir qué hacer.

10.4.2 Transformación de respuestas SOAP

La función de `<transform>` recibe (`status_code`, `result`), igual que en REST, pero `status_code` vale siempre 0 y `result` es el JSON obtenido con las reglas anteriores. Este ejemplo detecta los *Fault*, normaliza `role` para que sea siempre un array y calcula valores derivados:

```
<query name="LocalUserProfile" module="golive" type="SOAP" transport="POST">
  <url>FoundationEnvironmentManager</url>
  <method uri="http://oracle.e1.bssv.JP000000/" name="getCurrentUserProfile"
    action="">
  <script language="Javascript"><![CDATA[
    function perfil(status_code, r) {
      if (r.faultcode !== undefined) {
        log_info("SOAP Fault:", r.faultstring);
        throw "ZBX_NOTSUPPORTED";
      }
      var roles = r.role;
      if (roles === undefined)
        roles = [];
      else if (!(roles instanceof Array))
        roles = [ roles ];           // un solo <role>
      return {
        address: Number(r.addressNumber),
        nroles:  roles.length,
        admin:   roles.indexOf("ADMIN") >= 0,
        language: r.userProfile ? r.userProfile.language : ""
      };
    }
  ]]>
</script>
<transform language="Javascript" function="perfil"/>
<item name="local.golive.address" path="address" type="INT"/>
<item name="local.golive.nroles" path="nroles" type="INT"/>
<item name="local.golive.admin" path="admin" type="INT"/>
<item name="local.golive.language" path="language" type="STRING"/>
</query>
```

Valores obtenidos con la respuesta del ejemplo anterior

Clave	Valor
local.golive.address	4242
local.golive.nroles	2
local.golive.admin	1
local.golive.language	S

La misma función en Python:

```
def perfil(status_code, r):
    if 'faultcode' in r:
        log_info("SOAP Fault:", r.get('faultstring'))
        raise Exception("ZBX_NOTSUPPORTED")
    roles = r.get('role', [])
    if not isinstance(roles, list):
        roles = [roles]
    return {
        'address': r.get('addressNumber'),
```

```

    'nroles':    len(roles),
    'admin':     'ADMIN' in roles,
    'language':  r.get('userProfile', {}).get('language', '')
}

```

10.5 Datos de ejecución

El motor Webservice ofrece estos datos de ejecución:

Nombre	Valor
DATABBIX.PROGRAM_NAME	databbix, o queryexplorer en esa utilidad.
CONN.TYPE	Webservice.

Como las consultas de este motor no tienen condiciones `<when>`, estos datos solo se consultan con `getRuntimeValue` en `queryexplorer`.

10.6 Autenticación

10.6.1 Motores y métodos de autenticación

Hay que distinguir dos cosas:

Motor de autenticación. Está programado en C++ dentro de Databbix e implementa un mecanismo completo: cómo se envían las credenciales, cómo se obtiene y renueva un token, etc. Hay cuatro: `HTTPAuth`, `RESTToken`, `JDE` y `Oasis`.

Método de autenticación. Se define en un fichero de consultas con `<authentication>`. Toma un motor (atributo `type`) y lo personaliza para un servicio concreto con funciones en JavaScript o Python y otros elementos (URL de login, cabeceras, dónde va el token...). Se identifica por su atributo `name`.

Motor	Servicios	Uso directo	Personalización con un método
HTTPAuth	REST y SOAP	Sí: <code>Credentials = method=BASIC...</code>	Opcional: <code><auth-generate></code> , <code><header></code> , <code><verify-call></code> .
RESTToken	REST	No (error <i>RESTToken cannot be directly used</i>)	Obligatoria: <code><login></code> , <code><token></code> , <code><verify-call></code> .
JDE	REST	Sí	No admite: un método de este tipo solo le da otro nombre.
Oasis	SOAP	Sí	No admite: un método de este tipo solo le da otro nombre.

10.6.2 Cómo se elige la autenticación

El parámetro `Webservice.Authenticator` del `.conf` contiene un nombre. Databbix lo busca en este orden:

1. **Un método de autenticación** en los ficheros de consultas del monitor: un `<authentication>` cuyo `name` coincida. Si lo encuentra, usa el motor que indica su `type`, personalizado con las funciones del método.

2. **Un motor de autenticación** con ese nombre. Si lo encuentra, lo usa directamente, sin personalizar, con las credenciales del `.conf`. (Se admite también el nombre con el sufijo `Authenticator`, como `HTTPAuthAuthenticator`.)
3. Si no encuentra ninguno de los dos, el monitor no puede conectar y lo registra en su log.

Nota. Como los métodos se buscan primero, un método cuyo `name` coincida con el de un motor (por ejemplo, `name="HTTPAuth"`) oculta ese motor. Use nombres propios para los métodos (`AISAuth`, `ApiKeyAuth`...).

10.6.3 Ejemplos

Motor usado directamente: no hay ningún método llamado `JDE` en los XML, así que se usa el motor `JDE` tal cual.

```
[Webservice]
BaseUrl = http://jdesm.empresa.com:8999/manage/mgmtrestservice/
Authenticator = JDE
UserName = jde_admin
Password = secreto
```

Método de autenticación: el XML define `AISAuth`, que personaliza el motor `RESTToken`, y el `.conf` lo usa por su nombre.

```
<Queries engine="Webservice">
  <authentication type="RESTToken" name="AISAuth">
    <script language="Javascript"> ... </script>
    <login type="JsonPost"> ... </login>
    <token type="Header" name="jde-AIS-Auth"/>
    <verify-call language="Javascript" function="verifyCall"/>
  </authentication>
  <query ...> ... </query>
</Queries>
```

```
[Webservice]
Authenticator = AISAuth
```

La programación de las funciones de autenticación se describe en la sección 12.7.

Capítulo 11

Motor MQTT

El motor MQTT se suscribe a un broker MQTT y guarda el último mensaje recibido en cada *topic*. Los ítems de Zabbix leen esos valores o descubren los *topics* disponibles. A diferencia de los otros motores, no ejecuta ninguna operación cuando Zabbix pide un valor: los datos llegan continuamente desde el broker, a través de un hilo receptor.

11.1 Configuración: sección [MQTT]

Parámetro	Defecto	Descripción
Server	<i>obligatorio</i>	Nombre o IP del broker.
Port	1883	Puerto del broker.
Id	<i>obligatorio</i>	Identificador de cliente MQTT. Debe ser único en el broker.
UserName	<i>obligatorio</i>	Usuario.
Password	<i>obligatorio</i>	Contraseña.
Timeout	20000	Tiempo máximo de conexión, en ms.
Subscription	#	Jerarquía de <i>topics</i> a la que se suscribe. Escríbase \043 para #.
DiscoveryDelay	10000	Tras el arranque, espera en ms antes de responder a los descubrimientos, para dar tiempo a recibir todos los <i>topics</i> .
AbsentDelay	10000	Espera en ms antes de volver a intentar un ítem cuyo <i>topic</i> no tiene mensajes nuevos.

```
[General]
Name = PLANTA1
Engine = MQTT
QueryFile = /etc/databbix/query-mqtt.xml

[Zabbix]
Server = zabbix.empresa.com
Port = 10051

[Data]
Module = MQTT

[MQTT]
Server = mqtt.empresa.com
Id = databbix-plantal
```

```
UserName = zabbix
Password = secreto
Subscription = planta1/\043      # planta1/#
```

11.2 Ficheros de consultas

Las consultas MQTT no llevan elementos propios: la operación es siempre la misma (leer el último valor de un *topic*). El fichero `query-mqtt.xml` de la distribución contiene las dos consultas necesarias:

```
<Queries engine="MQTT" databbix="2.5.0" query="2.3.0">

<query name="ItemValue" module="MQTT">
  <parameter order="1" type="STRING"/>
  <item name="ts.mqtt.getvalue"/>
</query>

<query name="ItemDiscover" module="MQTT" discovery="yes">
  <parameter order="1" type="STRING"/>
  <item name="ts.mqtt.discovery"/>
  <field name="TOPIC"/>
</query>

</Queries>
```

Atención. En este motor, los nombres de ítem son fijos: el motor solo responde a `ts.mqtt.getvalue` y `ts.mqtt.discovery`. Un `<item>` con otro nombre se carga sin error, pero sus valores son siempre *no soportados*.

Reglas de las consultas MQTT:

- Los `<parameter>` solo llevan `order` (o `value`) y `type`. El parámetro es el *topic* o, en los descubrimientos, el patrón de *topics*.
- Los `<item>` solo llevan `name`; no admiten atributos que indiquen de dónde sale el valor.
- Un descubrimiento debe tener exactamente un `<parameter>` y un `<field>`. El `<field>` solo lleva `name`: la macro toma como valor el nombre completo del *topic*.

11.3 Lectura de valores

La clave `ts.mqtt.getvalue[topic]` devuelve el contenido del último mensaje publicado en ese *topic*. Dados estos mensajes recibidos del broker:

Últimos mensajes recibidos

Topic	Contenido
planta1/linea1/temperatura	21.5
planta1/linea1/presion	1.02
planta1/linea2/temperatura	22.1
planta1/estado	{"modo":"auto","alarmas":0}

Valores obtenidos

Clave	Valor
<code>ts.mqtt.getvalue[planta1/linea1/temperatura]</code>	21.5
<code>ts.mqtt.getvalue[planta1/estado]</code>	<code>{"modo":"auto","alarmas":0}</code>
<code>ts.mqtt.getvalue[planta1/linea3/temperatura]</code>	<i>no soportado: no se ha recibido</i>

El contenido se envía tal cual. Si es un JSON, como en `planta1/estado`, se pueden extraer sus campos en Zabbix con un preprocesado *JSONPath* en el ítem.

Un valor solo se envía a Zabbix cuando ha llegado un mensaje nuevo desde la última vez. Si no ha llegado ninguno, el ítem se vuelve a intentar al cabo de `AbsentDelay` milisegundos, en lugar de repetir el valor anterior.

11.4 Descubrimiento de topics

La clave `ts.mqtt.discovery[patrón]` devuelve todos los *topics* recibidos que encajan con el patrón. El patrón usa los comodines de MQTT:

Comodín	Significado
+	Exactamente un nivel cualquiera.
#	Cualquier número de niveles.

Con los mensajes de la sección anterior:

Patrones y topics descubiertos

Clave	Valores de {#TOPIC}
<code>ts.mqtt.discovery[planta1/+temperatura]</code>	planta1/linea1/temperatura planta1/linea2/temperatura
<code>ts.mqtt.discovery[planta1/linea1/#]</code>	planta1/linea1/temperatura planta1/linea1/presion
<code>ts.mqtt.discovery[planta1/+]</code>	planta1/estado planta1/linea1/temperatura planta1/linea1/presion planta1/linea2/temperatura

Nota. A diferencia de MQTT estándar, en Databbix un patrón también encaja con los *topics* que tienen más niveles que él: funciona como un prefijo. Así, `planta1/+` descubre de una vez todos los *topics* que cuelgan de cualquier subnivel de `planta1`. Para limitar el resultado a un nivel concreto, escriba el patrón con todos los niveles (`planta1/+temperatura`).

El valor del primer ítem es:

```
{"data": [ {"{#TOPIC}": "planta1/linea1/temperatura"},
            {"{#TOPIC}": "planta1/linea2/temperatura"} ]}
```

En Zabbix se crea una regla de descubrimiento con esa clave y el prototipo de ítem `ts.mqtt.getvalue[{#TOPIC}]`.

Solo se descubren los *topics* de los que ya se ha recibido algún mensaje. Por eso, durante los primeros `DiscoveryDelay` milisegundos tras el arranque, los descubrimientos se retrasan.

11.5 Datos de ejecución

El motor MQTT ofrece estos datos de ejecución:

Nombre	Valor
DATABBIX.PROGRAM_NAME	databbix, o queryexplorer en esa utilidad.
CONN.TYPE	MQTT.

Como las consultas de este motor no tienen condiciones **<when>**, estos datos solo se consultan con `getRuntimeValue` en `queryexplorer`.

Parte IV

Herramientas y uso

Capítulo 12

Programación de scripts

Los scripts permiten añadir lógica propia allí donde el XML por sí solo no llega: convertir formatos, calcular valores derivados, reestructurar o filtrar la respuesta de un servicio y programar el diálogo de autenticación con un webservice. Este capítulo explica el entorno en el que se ejecutan los scripts y cómo se escriben para cada uno de esos usos.

12.1 Lenguajes disponibles

Lenguaje	Nombre en el XML	Intérprete
JavaScript	Javascript	Duktape 2.5, integrado en Databbix. Implementa ECMAScript 5.1.
Python	Python	CPython del sistema, integrado en Databbix.

Todos los scripts de una colección de consultas (incluidos los de las autenticaciones) deben estar escritos en el mismo lenguaje.

12.1.1 JavaScript

- Escriba el código en ECMAScript 5.1: `var` y `function`. Construcciones más modernas como `let`, `const`, las funciones flecha o las plantillas de texto pueden no estar disponibles.
- Están disponibles los objetos estándar del lenguaje (`Math`, `Date`, `JSON`, `RegExp...`), pero no hay acceso a ficheros, red, temporizadores ni módulos (`require`).
- `Date.parse()` solo reconoce con seguridad fechas en formato ISO 8601, por ejemplo: 2026-09-29T10:15:00Z. Si la base de datos devuelve las fechas en otro formato, conviértalas en el SQL (`to_char`, `extract(epoch...)`) o analícelas a mano en el script.

12.1.2 Python

- Se usa la versión de Python instalada en el sistema, con su biblioteca estándar: se puede importar `re`, `datetime`, `json`, etc.
- El código se ejecuta dentro del proceso de Databbix. Evite operaciones lentas o bloqueantes (red, disco): retrasan todo el monitor.

12.2 Dónde se escriben los scripts

Un script es un elemento `<script>` con el atributo `language`. El código se escribe dentro de una sección `CDATA` para no tener que escapar los caracteres `<`, `>` y `&`:

```
<script language="Javascript"><![CDATA[
  function kb2mb(valor) {
    return Math.round(Number(valor) / 1024);
  }
  ]]>
</script>
```

Puede aparecer en tres lugares:

En <Queries>. Script global. Sus funciones están disponibles en todas las consultas de la colección.

En <query>. Script de consulta. Sus funciones solo están disponibles en esa consulta. Puede usar también las funciones globales.

En <authentication>. Script de autenticación. Sus funciones se usan en los elementos de esa autenticación.

Un script define funciones, que después se referencian por su nombre:

Referencia	Uso
Atributo transform de <item>	Transformación de un valor (secciones 12.4 y 12.5).
Elemento <transform> de <query>	Transformación de la respuesta de un webservice (sección 12.6).
Funciones de <authentication>	Autenticación de webservice (sección 12.7).

12.3 Entorno de ejecución

12.3.1 Ámbito y estado

Cada monitor tiene su propio intérprete. Dentro de él:

- Los scripts globales se ejecutan una vez, al cargar la colección.
- Cada consulta tiene su propio espacio de variables, que parte de una copia de las variables globales y al que se añaden los scripts de la consulta.
- Ese espacio se conserva mientras el monitor está en marcha. Una variable global que una función modifique mantiene su valor en la siguiente ejecución *de la misma consulta*, pero no la ven las demás consultas.

Esto permite, por ejemplo, calcular una tasa a partir de dos lecturas consecutivas:

```
<query name="LocalCommitRate" module="PG">
  <script language="Javascript"><![CDATA[
    var anterior = null;           // se conserva entre ejecuciones
    function tasa(valor) {
      var ahora = { t: Date.now() / 1000, v: Number(valor) };
      var r = 0;
      if (anterior !== null && ahora.t > anterior.t)
        r = (ahora.v - anterior.v) / (ahora.t - anterior.t);
      anterior = ahora;
      return r.toFixed(2);
    }
  ]]>
</script>
```

```

    ]]>
  </script>
  <sql><![CDATA[
    select sum(xact_commit) as commits from pg_stat_database ]]>
  </sql>
  <item name="local.pg.commitrate" column="commits" transform="Javascript:tasa"/>
</query>

```

Nota. El estado se pierde al reiniciar el servicio. Además, en Python la copia de las variables globales es superficial: una lista o diccionario creado en un script global es el mismo objeto en todas las consultas. Si necesita estado independiente, créelo dentro del script de la consulta.

12.3.2 Funciones de log

Los scripts pueden escribir en el log del monitor con dos funciones, que aceptan cualquier número de argumentos:

Función	Nivel de log
<code>log_debug(...)</code>	Depuración (se ve con LogLevel 7 o superior).
<code>log_info(...)</code>	Información (se ve con LogLevel 6 o superior).

Los mensajes aparecen con el prefijo `From Javascript:` o `From Python:`.

```

<script language="Javascript"><![CDATA[
  function estado(valor) {
    log_debug("estado recibido:", valor);
    return valor == "OPEN" ? 1 : 0;
  }
]]>
</script>

```

12.3.3 Tipos de datos

Databbix pasa los datos a las funciones de dos formas:

- **Como texto:** el valor de una columna o de una ruta JSON. Aunque la columna sea numérica, la función recibe una cadena; use `Number(valor)` en JavaScript o `float(valor)` en Python para operar con él.
- **Como objeto:** las respuestas de los webservices, el JSON de un descubrimiento y las credenciales. Databbix decodifica el JSON antes de llamar a la función y codifica a JSON el objeto que devuelve. Si el texto recibido no es JSON válido, la función recibe la cadena tal cual.

12.3.4 Errores

Si una función lanza una excepción, la operación falla y el mensaje de la excepción aparece en el log del monitor. El efecto depende del uso: en una transformación de ítem, el ítem pasa a no soportado; en la transformación de la respuesta de un webservice, todos los ítems de la consulta.

Para indicar a propósito que un dato no está disponible (y no registrarlo como error), la transformación de un webservice puede lanzar exactamente la cadena `ZBX_NOTSUPPORTED`:

```
throw "ZBX_NOTSUPPORTED";           // JavaScript
```

```
raise Exception("ZBX_NOTSUPPORTED")           # Python
```

Atención. En JavaScript hay que lanzar la cadena, no un objeto `Error`: `throw new Error("ZBX_NOTSUPPORTED")` produce el texto `Error: ZBX_NOTSUPPORTED`, que Databbix no reconoce.

12.4 Transformación de valores en consultas SQL

El atributo `transform` de un `<item>` aplica una función al valor de la columna antes de enviarlo a Zabbix. Tiene la forma `Lenguaje:función`:

```
<item name="local.ora.lastbackup.age" column="fin" transform="Javascript:edad"/>
```

Firma	función(valor)
valor	Valor de la columna, como texto.
Devuelve	El valor que se envía a Zabbix. Se convierte a texto: un número se envía tal cual, <code>true/false</code> como <code>true/false</code> (JavaScript) o <code>True/False</code> (Python).

La función se llama una vez por ítem y ejecución. Si la consulta es `FILTER`, se llama para la fila que corresponde a la clave del ítem.

12.4.1 Ejemplo: fechas

Segundos transcurridos desde una fecha y fecha en formato UNIX. Para que `Date.parse` funcione con cualquier base de datos, el SQL devuelve la fecha en formato ISO 8601:

```
<script language="Javascript"><![CDATA[
    function edad(fecha) {
        return Math.round((Date.now() - Date.parse(fecha)) / 1000);
    }
    function toUnix(fecha) {
        return Math.round(Date.parse(fecha) / 1000);
    }
]]>
</script>

<query name="LocalLastBackup" module="DB">
    <sql><![CDATA[
        select to_char(max(completion_time) at time zone 'UTC',
            'YYYY-MM-DD"THH24:MI:SS"Z') as fin
        from v$backup_set
    ]]>
    </sql>
    <item name="local.ora.lastbackup" column="fin" transform="Javascript:toUnix" />
    <item name="local.ora.lastbackup.age" column="fin" transform="Javascript:edad"/>
</query>
```

12.4.2 Ejemplo: traducir estados a números

Zabbix trabaja mejor con valores numéricos para los disparadores. Una tabla de traducción evita escribir la lógica en el SQL de cada base de datos:

```
<script language="Javascript"><![CDATA[
  var ESTADOS = { "OPEN": 0, "MOUNTED": 1, "STARTED": 2 };
  function estadoInstancia(valor) {
    var v = ESTADOS[String(valor).trim()];
    return (v === undefined) ? 99 : v;
  }
  ]]>
</script>

<query name="LocalInstanceStatus" module="Instance">
  <sql><![CDATA[ select status from v$instance ]]></sql>
  <item name="local.ora.instance.status" column="status"
    transform="Javascript:estadoInstancia"/>
</query>
```

12.4.3 Ejemplo en Python

La misma idea en Python, extrayendo un número de versión de un texto:

```
<script language="Python"><![CDATA[
import re

def version_mayor(valor):
    m = re.search(r'(\d+)\.', valor)
    return int(m.group(1)) if m else 0
  ]]>
</script>

<query name="LocalPGVersion" module="PG">
  <sql><![CDATA[ select version() as v ]]></sql>
  <item name="local.pg.version.major" column="v" transform="Python:version_mayor"/>
</query>
```

Nota. En Python, la indentación es significativa: el código dentro de CDATA debe empezar en la primera columna, como en el ejemplo.

12.5 Transformación de descubrimientos

En una consulta de descubrimiento, el `transform` del `<item>` recibe el JSON completo del descubrimiento como objeto y debe devolver un objeto con la misma estructura. Sirve para filtrar entidades o añadir macros calculadas.

Firma	función(lld)
lld	Objeto { data: [{ "{#MACRO}": valor, ... }, ...] }.
Devuelve	Objeto con la misma forma, que se envía a Zabbix.

```
<query name="LocalTSDiscovery" module="DB" discovery="yes">
  <script language="Javascript"><![CDATA[
    function soloDatos(lld) {
```

```

    var res = { data: [] };
    for (var i = 0; i < lld.data.length; i++) {
        var e = lld.data[i];
        // descarta los tablespaces del sistema
        if (/^(SYSTEM|SYSAUX)$/.test(e["{#TSNAME}"]))
            continue;
        // macro nueva: nombre en minúsculas
        e["{#TSLOWER}"] = e["{#TSNAME}"].toLowerCase();
        res.data.push(e);
    }
    return res;
}
]]>
</script>
<sql><![CDATA[
    select tablespace_name from dba_tablespaces where status = 'ONLINE' ]]>
</sql>
<item name="local.ora.tspace.discovery" transform="Javascript:soloDatos"/>
<field name="TSNAME" column="tablespace_name"/>
</query>

```

Nota. Filtrar en el SQL (*where*) es más eficiente. La transformación es útil cuando el criterio es difícil de expresar en SQL o cuando el mismo XML se usa con varias bases de datos.

12.6 Transformación de respuestas de webservice

En las consultas REST, el elemento `<transform>` de la consulta procesa la respuesta del servicio *antes* de extraer los valores con las rutas (*path*) de los ítems:

```

<query name="..." module="..." type="REST" transport="GET" encode="Json">
    <url>...</url>
    <transform language="Javascript" function="miFuncion"/>
    ...
</query>

```

Firma	función(status_code, result)
status_code	Código HTTP de la respuesta (200, 404, 500...). En las consultas SOAP vale 0.
result	Cuerpo de la respuesta decodificado como JSON, o la cadena de texto si no es JSON. En SOAP, la respuesta convertida en JSON (sección 10.4.1).
Devuelve	Objeto sobre el que se aplican los <i>path</i> de los ítems y de los <code><field></code> .

Atención. Sin `<transform>`, una respuesta con código distinto de 200 es un error. Con `<transform>`, la función recibe *todas* las respuestas y es la responsable de tratar los códigos de error.

La transformación se ejecuta una vez por llamada al servicio. Los usos típicos son tres: tratar errores, reestructurar la salida y filtrarla.

12.6.1 Tratar códigos de error

El servicio de JD Edwards devuelve un error 500 con un código propio cuando el agente de la instancia está caído. La función lo convierte en un estado que Zabbix puede vigilar, y marca como no soportado lo que no tiene sentido medir:

```
<query name="JDEInstanceState" module="JDE" type="REST" transport="GET" encode="Json"
">
  <url>instancestate</url>
  <script language="Javascript"><![CDATA[
    function instancestate(status_code, result) {
      if (status_code == 200)
        return result;
      if (status_code == 500) {
        switch (result.errorCode) {
          case 13: return { instanceState: "AGENT_DOWN" };
          case 14: throw "ZBX_NOTSUPPORTED";
        }
      }
      throw "Error HTTP " + status_code;
    }
  ]]>
</script>
<transform language="Javascript" function="instancestate"/>
<parameter urlparameter="instanceName" order="1" type="STRING"/>
<item name="ts.jde.instance.state" path="instanceState" type="STRING"/>
</query>
```

12.6.2 Reestructurar la salida

A menudo, el dato que interesa en Zabbix no está directamente en la respuesta: hay que contarlo, sumarlo o calcularlo. La función construye un objeto nuevo con los valores ya calculados, y los ítems apuntan a él:

```
<query name="LocalJobSummary" module="API" type="REST" transport="GET" encode="Json"
">
  <url>jobs</url>
  <script language="Javascript"><![CDATA[
    function resumen(status_code, result) {
      if (status_code != 200)
        throw "ZBX_NOTSUPPORTED";
      var r = { total: 0, running: 0, failed: 0, maxDuration: 0 };
      for (var i = 0; i < result.jobs.length; i++) {
        var j = result.jobs[i];
        r.total++;
        if (j.state == "RUNNING") r.running++;
        if (j.state == "FAILED") r.failed++;
        if (j.duration > r.maxDuration) r.maxDuration = j.duration;
      }
      return r;
    }
  ]]>
</script>
<transform language="Javascript" function="resumen"/>
<item name="local.api.jobs.total" path="total" type="INT"/>
<item name="local.api.jobs.running" path="running" type="INT"/>
<item name="local.api.jobs.failed" path="failed" type="INT"/>
<item name="local.api.jobs.maxdur" path="maxDuration" type="INT"/>
```

```
</query>
```

La misma técnica sirve para respuestas que no son JSON: si el servicio devuelve texto, `result` es una cadena que la función puede analizar y convertir en un objeto.

12.6.3 Filtrar la salida

En un descubrimiento REST, la función puede quedarse solo con los elementos que interesan. Aquí, de todas las instancias que devuelve el servicio, solo se descubren los servidores *enterprise*:

```
<query name="JDEDiscoverEnterprise" module="JDE" type="REST"
  transport="GET" encode="Json" discovery="yes">
  <url>instancesinfo</url>
  <script language="Javascript"><![CDATA[
    function filtraenterprise(status_code, result) {
      if (status_code != 200)
        throw "ZBX_NOTSUPPORTED";
      var res = { result: [] };
      for (var i = 0; i < result.result.length; i++)
        if (result.result[i].targetType == "entserver")
          res.result.push(result.result[i]);
      return res;
    }
  ]]>
  </script>
  <transform language="Javascript" function="filtraenterprise"/>
  <item name="ts.jde.enterprise.discovery" path="result: []"/>
  <field name="INSTANCE" path="instanceName"/>
  <field name="TYPE" path="targetType"/>
</query>
```

12.6.4 Respuestas SOAP

En las consultas SOAP la función recibe el JSON obtenido de la respuesta XML y `status_code` vale siempre 0. Hay que detectar los errores buscando un *Fault* en el propio JSON y tener en cuenta que un elemento repetido puede llegar como array o como valor único. La sección 10.4.2 muestra un ejemplo completo en JavaScript y Python.

12.6.5 Combinar con transformaciones de ítem

Los ítems de una consulta REST también admiten el atributo `transform`, que se aplica después, al valor de cada ruta, igual que en las consultas SQL (sección 12.4). En los ítems de tipo INT, un resultado `true/false` se convierte en 1/0.

12.7 Programación de autenticaciones de webservice

Un método de autenticación (sección 10.6) personaliza un motor de autenticación con funciones propias. Solo los motores HTTPAuth y RESTToken admiten personalización. Las funciones del método deciden cómo se obtienen las credenciales o el token y cómo se detecta que hay que volver a autenticarse.

12.7.1 Credenciales

Todas las funciones que construyen credenciales reciben el mismo objeto, generado a partir del fichero de monitor:

```
[Webservice]
UserName = monitor
Password = secreto
Credentials = environment=JPD920
Credentials = role=*ALL
```

```
{ "username": "monitor", "password": "secreto",
  "environment": "JPD920", "role": "*ALL" }
```

UserName y Password se convierten en las claves `username` y `password`; cada línea `Credentials` añade una clave. De este modo, el XML describe el mecanismo y el `.conf`, los datos de cada instalación.

12.7.2 Verificación de llamadas

Tras cada llamada al servicio, Databbix pregunta a la función de `<verify-call>` si la respuesta es válida:

Firma	función(status_code, result)
Devuelve	Objeto con: <code>call_ok</code> (true si la llamada es válida), <code>error</code> (texto del error) y, solo en HTTPAuth, <code>reauthenticate</code> (true si hay que volver a autenticarse).

Si `call_ok` es false, Databbix vuelve a autenticarse y repite la llamada *una vez*. En HTTPAuth, si además `reauthenticate` es false, la llamada falla directamente con el error indicado (credenciales incorrectas).

12.7.3 Métodos basados en el motor RESTToken

Obtiene un token con una petición de *login* y lo envía en todas las llamadas siguientes.

```
<authentication type="RESTToken" name="nombre">
  <script language="Javascript"> ...funciones... </script>
  <login type="JsonPost">
    <url>ruta/de/login</url>
    <request-function language="Javascript" function="...">
    <accept-function language="Javascript" function="...">
  </login>
  <token type="Header|Cookie" name="...">
  <verify-call language="Javascript" function="...">
</authentication>
```

Elemento	Descripción
<code><login></code>	Petición de login. Se envía como POST a BaseURL + <code><url></code> , con cuerpo JSON. Use <code>type="JsonPost"</code> .
<code><request-function></code>	<code>función(credenciales)</code> . Devuelve el objeto que se envía como cuerpo de la petición de login.
<code><accept-function></code>	<code>función(status_code, result)</code> . Analiza la respuesta del login y devuelve <code>{authentication_ok, token, error}</code> .
<code><token></code>	Dónde se envía el token en las llamadas: en una cabecera (<code>type="Header"</code>) o en una cookie (<code>type="Cookie"</code>), con el nombre <code>name</code> .

Elemento	Descripción
<verify-call>	Verificación de cada llamada. Obligatorio.

Ejemplo completo para el servidor AIS de JD Edwards:

```
<authentication type="RESTToken" name="AISAuth">
  <script language="Javascript"><![CDATA[
    function authGenerate(credentials) {
      // cuerpo de la petición de login
      return {
        username:    credentials.username,
        password:    credentials.password,
        role:        credentials.role,
        environment: credentials.environment
      };
    }

    function authAccept(status_code, result) {
      if (status_code == 200)
        return { authentication_ok: true,
                  token: result.userInfo.token };
      var errores = {
        400: "Petición incorrecta",
        403: "Credenciales no válidas",
        415: "Content-Type incorrecto"
      };
      return { authentication_ok: false,
                error: errores[status_code] || ("Error " + status_code) };
    }

    function verifyCall(status_code, result) {
      // 444: token caducado -> volver a autenticarse
      if (status_code == 403 || status_code == 444)
        return { call_ok: false, error: "Token no válido" };
      return { call_ok: true };
    }
  ]]>
</script>
<login type="JsonPost">
  <url>v2/tokenrequest</url>
  <request-function language="Javascript" function="authGenerate"/>
  <accept-function language="Javascript" function="authAccept"/>
</login>
<token type="Header" name="jde-AIS-Auth"/>
<verify-call language="Javascript" function="verifyCall"/>
</authentication>
```

Nota. Si `WebService.AuthPersistFile` está configurado, el token se guarda en ese fichero y se reutiliza tras un reinicio, sin pedir uno nuevo. El fichero contiene un secreto: protéjalo como el propio `.conf`.

12.7.4 Métodos basados en el motor HTTPAuth

Usa los mecanismos de autenticación de HTTP. Sin `<authentication>`, toma las credenciales directamente del `.conf` (sección 3.15). Con `<authentication>`, admite:

Elemento	Descripción
<auth-generate>	función(credenciales). Devuelve las credenciales efectivas: {method, username, password} con method BASIC, DIGEST o NTLM; o {method: "HEADER", header, value} para enviar una cabecera fija.
<header>	Cabecera adicional que se envía en todas las llamadas, con atributos name y value. El valor también puede ir como texto del elemento, y admite variables \${...}.
<verify-call>	Verificación de cada llamada. Opcional: si no existe, solo es válida una respuesta con código 200, y cualquier otro código se trata como un error de credenciales.

Ejemplo: un servicio que se autentica con una clave de API en la cabecera X-API-Key. La clave se configura en el .conf, y el XML añade además una cabecera fija con la versión de la API:

```
<authentication type="HTTPOAuth" name="ApiKeyAuth">
  <script language="Javascript"><![CDATA[
    function apiKey(credenciales) {
      return { method: "HEADER",
               header: "X-API-Key",
               value:  credenciales.apikey };
    }
    function verify(status_code, result) {
      if (status_code == 401)
        return { call_ok: false, reauthenticate: false,
                 error: "Clave de API rechazada" };
      return { call_ok: true };
    }
  ]]>
</script>
<auth-generate language="Javascript" function="apiKey"/>
<header name="Accept-Version" value="2"/>
<verify-call language="Javascript" function="verify"/>
</authentication>
```

```
[Webservice]
BaseURL = https://api.empresa.com/v2/
Authenticator = ApiKeyAuth
Credentials = apikey=3f9a...c21
```


Capítulo 13

La utilidad queryexplorer

queryexplorer es una herramienta interactiva para probar la configuración de un monitor sin arrancar el demonio y sin enviar nada a Zabbix. Carga el mismo fichero `.conf` y los mismos XML que usaría el monitor, se conecta a la fuente de datos y permite ejecutar ítems uno a uno y ver el resultado. Sirve para:

- comprobar que un fichero de monitor y sus XML son correctos;
- comprobar que la conexión con la base de datos o el servicio funciona (y, en webservices, que la autenticación es correcta);
- probar una consulta nueva y sus transformaciones antes de crear los ítems en Zabbix;
- ver los contextos, variables de entorno y datos de ejecución, y evaluar las condiciones de `<when>`.

Se instala en `/opt/trustserver/bin/queryexplorer`.

13.1 Invocación

```
queryexplorer [opciones] -f <monitor.conf> [comando [argumentos] ...]
```

Opción	Descripción
<code>-f, --conf-config-file f</code>	<i>Obligatoria.</i> Fichero de monitor que se prueba. Si la ruta no empieza por <code>/</code> , se busca en el directorio de monitores (<code>/etc/databbix/dbs</code>).
<code>-l, --log-console</code>	Muestra el log también en la consola. Muy recomendable: sin ella, los errores de carga solo quedan en el fichero de log.
<code>-v, --logLevel n</code>	Nivel de log (sustituye al de <code>databbix.conf</code>). Con 9 se ven las sentencias SQL, las llamadas HTTP y sus respuestas.
<code>-c, --config-file f</code>	Fichero de configuración global. Por defecto, <code>/etc/databbix/databbix.conf</code> .
<code>-p, --config-path d</code>	Directorio de monitores, para resolver rutas relativas en <code>-f</code> .
<code>-m, --module-path d</code>	Directorios adicionales de módulos.
<code>-h, --help</code>	Muestra la ayuda.

Al arrancar, **queryexplorer** hace lo mismo que un monitor:

1. lee `databbix.conf` y el fichero indicado con `-f`;
2. carga los XML de consultas;
3. se conecta a la fuente de datos.

Si cualquiera de estos pasos falla, muestra el error y termina. Por tanto, *arrancar sin errores ya valida la configuración y la conexión*.

Nota. `queryexplorer` escribe su log en el fichero de log global (`LogPath/LogFile` de `databbix.conf`). Ejecútelo con el usuario `databbix` (o con otro que tenga permiso de escritura en ese fichero) y con las mismas variables de entorno que el servicio.

```
sudo -u databbix env TNS_ADMIN=/etc/databbix ODBCINI=/etc/databbix/odbc.ini \
LD_LIBRARY_PATH=/opt/trustserver/lib64 \
/opt/trustserver/bin/queryexplorer -l -f oraprod.conf
```

`queryexplorer` abre su propia conexión, así que se puede usar con el servicio en marcha.

13.2 Modo interactivo

Sin comandos en la línea de órdenes, `queryexplorer` entra en modo interactivo:

```
$ queryexplorer -l -f pgprod.conf

Databbix QueryExplorer: Interactive Query Tool Version 2.5.0-2639.
Databbix Database Engine Version 2.5.0-2639.
(C) Trustserver SL 2014-2024. All Rights Reserved.

QueryExplorer> get tez.pg.sessions[ventas]
Item: tez.pg.sessions[ventas], value: 12
QueryExplorer> quit

Quitting Databbix. See you soon!
```

- La tecla *Tab* completa los nombres de comando y, según el comando, los nombres de ítem, de consulta, de contexto o de variable.
- Las flechas recorren el historial, que se guarda en `~/.queryexplorer_history`.
- Se sale con `quit` o con *Ctrl-D*.
- Los comandos no distinguen mayúsculas de minúsculas. Los argumentos se separan por espacios, así que una clave de ítem no puede contener espacios.

13.3 Modo no interactivo

Si se ponen comandos después de las opciones, se ejecutan en orden y el programa termina. Cada comando toma sus argumentos de los que le siguen:

```
queryexplorer -l -f pgprod.conf \
  get 'tez.pg.sessions[ventas]' \
  get 'tez.pg.sessions[postgres]' \
  get tez.pg.db.discovery
```

Conviene poner las claves con corchetes entre comillas para que la shell no los interprete. Este modo es útil en scripts de comprobación tras una instalación o actualización.

13.4 Comandos

13.4.1 Ítems y consultas

Comando	Descripción
<code>get clave</code>	Ejecuta la consulta del ítem y muestra su valor, igual que lo calcularía el monitor: con los parámetros y el contexto de la clave y aplicando las transformaciones. Ignora la caché: siempre ejecuta la consulta.
<code>listItems</code>	Lista los nombres de todos los ítems de la colección.
<code>listItems full</code>	Lista todas las consultas con su texto (SQL o URL) y, para cada ítem, sus parámetros y la columna o ruta de la que sale.
<code>listQueries</code>	Lista los nombres de las consultas.
<code>descQuery nombre</code>	Muestra una consulta (por su nombre o por el de uno de sus ítems): texto y relación de ítems.
<code>descItem ítem</code>	Muestra la consulta a la que pertenece un ítem y de dónde sale su valor.

13.4.2 Contextos

Comando	Descripción
<code>getDefinedContexts</code>	Contextos declarados en los XML, con su valor por defecto.
<code>getSupportedContexts</code>	Contextos que admite la conexión, con su valor actual.
<code>getContext nombre</code>	Valor actual de un contexto en la conexión.
<code>setContext nombre valor</code>	Cambia un contexto de la conexión. Sirve para comprobar que se puede entrar en él (por ejemplo, en una base de datos concreta).
<code>setContext nombre</code>	Vuelve a poner el contexto en su valor por defecto.
<code>setContext</code>	Vuelve a poner todos los contextos en su valor por defecto.

El comando `get` cambia al contexto que indica la clave y, al terminar, deja la conexión en el contexto en que estaba.

13.4.3 Variables y expresiones

Comando	Descripción
<code>getEnvironment nombre</code>	Valor de una variable de entorno, tal como la ve el monitor (combinando XML, <code>databbix.conf</code> y <code>.conf</code>).
<code>getRuntimeNames</code>	Nombres de los datos de ejecución que ofrece la conexión.
<code>getRuntimeValue nombre</code>	Valor de un dato de ejecución.
<code>evaluateExpression expresión</code>	Evalúa una expresión como las condiciones de <code><when></code> . Muestra 1 si se cumple y 0 si no.

Nota. En `queryexplorer`, `runtime('DATABBIX.PROGRAM_NAME')` devuelve `queryexplorer` en lugar de `databbix`. Una condición puede usarlo para comportarse de forma distinta durante las pruebas.

Atención. En modo interactivo, `evaluateExpression` toma como expresión todo el resto de la línea. En modo no interactivo toma un único argumento, así que la expresión debe ir entre comillas para que la shell la pase como un solo parámetro:

```
queryexplorer -l -f oraprod.conf \
    evaluateExpression "runtime('DB.DATABASE') = 'Oracle'"
```

13.4.4 Otros

Comando	Descripción
help	Lista de comandos.
quit	Sale del modo interactivo.

13.5 Casos de uso

13.5.1 Validar un fichero de monitor y su conexión

```
queryexplorer -l -f oraprod.conf listQueries
```

Si el comando lista las consultas, el `.conf` es correcto, los XML se han cargado sin errores y la conexión funciona. Si no, el log en consola indica qué ha fallado: un parámetro obligatorio que falta, un error de sintaxis en un XML (con el nombre de la consulta) o el error de conexión del driver.

13.5.2 Validar la autenticación de un webservice

En un monitor `WebService`, la autenticación se produce en la primera llamada. Basta con pedir un ítem con el log detallado para ver la petición de login, la respuesta y el resultado de las funciones de autenticación:

```
queryexplorer -l -v 9 -f aisprod.conf get ts.ais.scheduler.state
```

13.5.3 Probar una consulta nueva

```
QueryExplorer> descQuery LocalInvalidObjects
QueryExplorer> get local.ora.invalid[HR]
Item: local.ora.invalid[HR], value: 3
QueryExplorer> get local.ora.invalid[NOEXISTE]
Item: local.ora.invalid[NOEXISTE], value: 0
QueryExplorer> get local.ora.schema.discovery
Item: local.ora.schema.discovery, value: {"data":[{"#SCHEMA":"HR"}, ...]}
```

El segundo `get` comprueba que el atributo `default` funciona cuando la fila no existe. Si un valor no es el esperado, repita con `-v 9` para ver la sentencia ejecutada, las filas obtenidas y las llamadas a las funciones de transformación.

13.5.4 Comprobar el SQL alternativo

Para saber qué rama de un `<multiple>` se usará con una base de datos:

```
QueryExplorer> getRuntimeValue DB.DATABASE
DB.DATABASE = 'Oracle'
QueryExplorer> evaluateExpression runtime('DB.DATABASE') = 'Oracle'
Result: 1
```

13.5.5 Comprobar los contextos

Para verificar que el usuario puede entrar en todas las bases de datos de un SQL Server antes de activar un descubrimiento por contextos:

```
QueryExplorer> getDefinedContexts
Contexts defined in query file:
- DATABASE, default: master
QueryExplorer> setContext Database ventas
QueryExplorer> getContext Database
Actual value for context 'Database': 'ventas'
QueryExplorer> setContext
```


Capítulo 14

Guía práctica: crear una consulta nueva

Este capítulo recorre el proceso completo con un ejemplo: monitorizar en Oracle el número de objetos inválidos de cada esquema.

14.1 Paso 1: escribir y probar el SQL

Primero se escribe la consulta y se prueba con el mismo usuario que usa Databbix, por ejemplo con SQL*Plus o psql. Hay que comprobar que devuelve una fila por objeto monitorizado y que las columnas tienen alias claros:

```
select owner as esquema, count(*) as invalidos
from dba_objects
where status = 'INVALID'
group by owner;
```

14.2 Paso 2: elegir el tipo de consulta

- ¿Devuelve siempre una fila? → consulta básica.
- ¿Devuelve una fila por objeto y se quieren ítems por objeto? → FILTER.
- ¿El conjunto de valores posibles es enorme y solo interesan unos pocos? → BIND.
- ¿Se quiere que Zabbix cree los ítems automáticamente? → una consulta **discovery** más una FILTER.

En el ejemplo, un esquema sin objetos inválidos no aparece en el resultado, así que se usa FILTER con `default="0"`, y un descubrimiento para los esquemas.

14.3 Paso 3: escribir el XML

Lo más recomendable es crear un fichero propio en lugar de modificar los ficheros de la distribución, que se sobrescriben al actualizar. Se guarda como `/etc/databbix/query-local.xml`:

```
<?xml version="1.0" encoding="UTF-8"?>
<Queries engine="Database" query="2.5.0">

<query name="LocalSchemaDiscovery" module="DB" discovery="yes">
  <sql><![CDATA[
```

```

        select username as esquema
        from dba_users
        where oracle_maintained = 'N'
    ]]>
</sql>
<item name="local.ora.schema.discovery"/>
<field name="SCHEMA" column="esquema"/>
</query>

<query name="LocalInvalidObjects" module="DB" select="FILTER" maxCache="60">
    <sql><![CDATA[
        select owner as esquema, count(*) as invalidos
        from dba_objects
        where status = 'INVALID'
        group by owner
    ]]>
    </sql>
    <parameter column="esquema" order="1" type="STRING"/>
    <item name="local.ora.invalid" column="invalidos" default="0"/>
    <item name="local.ora.invalid.exectime" built_in="exec_time"/>
</query>

</Queries>

```

Nombres de ítem. Los nombres de ítem deben ser únicos en todos los ficheros que carga el monitor. Usar un prefijo propio (*local.*, el nombre de la empresa...) evita conflictos con los ítems de la distribución (*tez.**, *ts.**).

14.4 Paso 4: añadir el fichero al monitor

QueryFile en el monitor *sustituye* a la lista global, así que hay que incluir también el fichero principal:

```

[General]
Name = ORAPROD
QueryFile = /etc/databbix/query.xml
QueryFile = /etc/databbix/query-local.xml

```

14.5 Paso 5: probar con queryexplorer

Antes de crear nada en Zabbix, se prueba la consulta con **queryexplorer** (capítulo 13), sin tocar el servicio:

```
sudo -u databbix /opt/trustserver/bin/queryexplorer -l -f oraprod.conf
```

```

QueryExplorer> get local.ora.schema.discovery
QueryExplorer> get local.ora.invalid[HR]
QueryExplorer> get local.ora.invalid.exectime[HR]

```

Si el XML tiene un error, **queryexplorer** no llega a mostrar el indicador y el mensaje explica qué consulta falla. Si un valor no es el esperado, se repite con **-v 9** para ver el SQL y las filas obtenidas. Cuando todo es correcto, se reinicia el servicio para que el monitor cargue el fichero nuevo:

```
systemctl restart databbix
```

14.6 Paso 6: crear los ítems en Zabbix

En el host `ORAPROD` (o en una plantilla):

1. Regla de descubrimiento de tipo *Zabbix agent (active)* con clave:
`local.ora.schema.discovery`
2. Prototipo de ítem numérico de tipo *Zabbix agent (active)* con clave:
`local.ora.invalid[{-#SCHEMA}]`
3. Opcionalmente, un prototipo de disparador, por ejemplo:
`last(/ORAPROD/local.ora.invalid[{-#SCHEMA}])>0`

Los datos llegan cuando Databbix vuelve a pedir la lista de ítems activos, es decir, en un máximo de `Zabbix.Refresh` segundos (300 por defecto), o inmediatamente si se reinicia el servicio.

Capítulo 15

Diagnóstico y resolución de problemas

15.1 Niveles de log

Nivel	Qué se registra
3	Errores (valor por defecto).
4	Avisos.
5	Eventos relevantes: arranque de monitores, errores de conexión y reintentos.
6	Información de funcionamiento: ejecuciones y tiempo hasta el siguiente ciclo.
7	Depuración, incluida la configuración completa del monitor al arrancar.
9	Además, el texto de cada SQL ejecutado y cada fila obtenida.
10	Además, la estructura completa de los XML analizados.

Tabla 15.1: Niveles de log

El nivel se puede fijar para todo el demonio en `databbix.conf` o solo para un monitor en su fichero. Los niveles 9 y 10 generan mucho volumen: conviene usarlos solo durante las pruebas.

Seguridad. Los niveles altos de log están pensados para depurar y registran datos sensibles. Desde el nivel 7, el log incluye los tokens de sesión que obtienen los autenticadores de servicios web. Desde el nivel 9, incluye también las credenciales completas (usuario, contraseña y demás datos) que se envían al servicio, porque pueden ser la causa del problema que se está depurando. Los logs se crean con permisos `0640`; aun así, al terminar de depurar baje el nivel y borre los logs generados con esos niveles.

Cada monitor escribe en su propio fichero, por defecto `/var/log/databbix/<Name>.log`. El proceso principal escribe en `databbix.log`.

15.2 Problemas frecuentes

Síntoma	Causa probable y solución
El monitor no arranca: “ <i>General.Name is not present</i> ” u otro parámetro obligatorio	Falta ese parámetro en el <code>.conf</code> . Revise las secciones <code>[General]</code> , <code>[Zabbix]</code> , <code>[Data]</code> y la del motor.

Síntoma	Causa probable y solución
<i>“Monitor initialization error”</i> con un error de sintaxis	Error en el XML: atributo desconocido, consulta sin <code>module</code> , ítem duplicado... El mensaje indica la consulta afectada.
<i>“Item name ... already defined”</i>	Dos consultas definen el mismo nombre de ítem en los ficheros cargados.
<i>“Zabbix server error ..., retry in 60 seconds”</i>	No se llega al servidor Zabbix, o el servidor no conoce el host. Compruebe <code>Server</code> , <code>Port</code> , el cortafuegos y que el <code>Host name</code> de Zabbix coincide con <code>General.Name</code> .
<i>“Error connecting to data source”</i>	Cadena de conexión, usuario o contraseña incorrectos, o variables <code>TNS_ADMIN</code> / <code>ODBCINI</code> sin definir para el servicio.
El ítem aparece como no soportado en Zabbix	El nombre del ítem no existe en los XML cargados, el número de parámetros de la clave no coincide, la fila buscada no existe (use <code>default</code>), la consulta básica devuelve 0 o varias filas, o el SQL falla (véase el log con nivel 9).
<i>“Excessive execution time”</i>	La consulta tarda más que <code>maxExecTime/MaxQueryExecTimeLog</code> . Optimice el SQL o aumente <code>maxCache</code> para ejecutarla menos.
<i>“Too many errors ..., exiting”</i>	El monitor ha superado <code>MaxErrors</code> ejecuciones fallidas seguidas y se ha detenido. Corrija la causa y reinicie el servicio.
Los ítems nuevos tardan en recibir datos	Databbix pide la lista de ítems cada <code>Zabbix.Refresh</code> segundos. Espere o reinicie el servicio.
Los cambios en el XML o el <code>.conf</code> no se aplican	Solo se leen al arrancar. Reinicie el servicio.

Parte V

Apéndices

Apéndice A

Referencia rápida del XML

Este apéndice resume, para cada tipo de consulta, qué elementos y atributos son obligatorios, cuáles son opcionales y cuáles no se admiten. En los esquemas, los corchetes indican partes opcionales y la barra vertical, alternativas.

A.1 Nivel <Queries> (todos los motores)

```
<Queries [engine="Database|Webservice|MQTT"] [query="2.5.0"] [databbix="2.5.0"]>
  <environment name="VAR" default="valor"/>                                <!-- todos -->
  <script language="Javascript|Python"> ... </script>                    <!-- todos -->
  <context name="Database" module="MOD" default="valor"/>                <!-- Database -->
  <authentication type="RESTToken|HTTPAuth" name="N"> ...                <!-- Webservice -->
  <query ...> ... </query>
</Queries>
```

En todas las consultas, name y module son obligatorios, y maxCache, maxExecTime y <script> son opcionales. Los <field> solo se admiten en las consultas de descubrimiento.

A.2 Consulta constante (todos los motores)

```
<query name="..." module="..." type="Constant">
  <item name="clave" value="valor"/>
</query>
```

Obligatorio: type="Constant"; uno o más <item> con name y value.
Opcional: transform en los <item>.
No admite: <sql>, <url>, <parameter>, <field>.

A.3 Motor Database

A.3.1 Consulta básica

```
<query name="..." module="...">
  <sql> texto </sql>
  [<default context="C" value="v"/>]
  [<parameter context="C" order="N"/>]
  <item name="clave" column="COL" [transform="L:f"]/>
  [<item name="clave" built_in="exec_time|exec_success"/>]
</query>
```

- Obligatorio:** un `<sql>`; uno o más `<item>` con `column` (o `built_in`).
Opcional: `<default>`; `<parameter>` de contexto; `transform` en los ítems.
No admite: `select`; `<parameter>` con `column` o `bind`; `<field>`; `<sql context>`.

La sentencia debe devolver exactamente una fila.

A.3.2 Consulta de filtro

```
<query name="..." module="..." select="FILTER">
  <sql> texto </sql>
  [<default context="C" value="v"/>]
  [<parameter context="C" order="1"/>]
  <parameter column="COL" order="N" [type="STRING|INT"]/>
  <item name="clave" column="COL" [default="v"] [transform="L:f"]/>
</query>
```

- Obligatorio:** `select="FILTER"`; un `<sql>`; al menos un `<parameter>` con `column`; uno o más `<item>` con `column` (o `built_in`).
Opcional: `<default>`; `<parameter>` de contexto; `default` y `transform` en los ítems.
No admite: `<parameter>` con `bind`; `<field>`; `<sql context>`.

A.3.3 Consulta con variables de enlace

```
<query name="..." module="..." select="BIND">
  <sql> texto con :var </sql>
  [<default context="C" value="v"/>]
  [<parameter context="C" order="1"/>]
  <parameter bind=":var" order="N" [type="STRING|INT"]/>
  <item name="clave" column="COL" [transform="L:f"]/>
</query>
```

- Obligatorio:** `select="BIND"`; un `<sql>`; al menos un `<parameter>` con `bind`; uno o más `<item>` con `column` (o `built_in`).
Opcional: `<default>`; `<parameter>` de contexto; `transform` en los ítems.
No admite: `<parameter>` con `column`; `<field>`; `<sql context>`.

La sentencia debe devolver exactamente una fila para cada clave.

A.3.4 Consulta de descubrimiento

```
<query name="..." module="..." discovery="yes">
  [<sql context="C"> texto </sql>]
  <sql> texto </sql>
  [<default context="C" value="v"/>]
  [<parameter context="C" order="N"/>]
  <item name="clave" [transform="L:f"]/>
  <field name="MACRO" column="COL" | context="C"/>
</query>
```

- Obligatorio:** `discovery="yes"`; un `<sql>` principal; al menos un `<field>` con `column` o `context`; uno o más `<item>` solo con `name`.
Opcional: `<sql context>` (uno por contexto que se recorre); `<default>`; `<parameter>` de contexto; `transform` en los ítems.
No admite: `select`; `<parameter>` con `column` o `bind`; `column`, `built_in` y `default` en los ítems.

A.4 Motor Webservice

A.4.1 Consulta REST

```
<query name="..." module="..." type="REST" transport="GET|POST" encode="Json">
  <url>ruta/relativa</url>
  [<transform language="L" function="f"/>]
  [<parameter urlparameter="nombre" order="N" [type="STRING|INT"]/>]
  <item name="clave" path="a:b:[0]:c" type="INT|STRING" [default="v"]/>
</query>
```

- Obligatorio:** type="REST"; transport; encode="Json"; un <url>; uno o más <item> con path y type (o built_in).
- Opcional:** <transform>; <parameter> con urlparameter; default y transform en los ítems.
- No admite:** <sql>; <field>; <parameter> con column, bind o context.

A.4.2 Consulta REST de filtro

```
<query name="..." module="..." type="REST" transport="GET|POST" encode="Json">
  <url>ruta/relativa</url>
  <base path="lista:[]" />
  [<transform language="L" function="f"/>]
  <parameter path="campo" order="N" [type="STRING|INT"]/>
  [<parameter urlparameter="nombre" order="M"/>]
  <item name="clave" path="campo" type="INT|STRING" [default="v"]/>
</query>
```

- Obligatorio:** lo mismo que una consulta REST, más <base> y al menos un <parameter> con path.
- Opcional:** <transform>; <parameter> con urlparameter; default y transform en los ítems.
- No admite:** <field>. Las rutas de los ítems son relativas al elemento elegido.

A.4.3 Consulta REST de descubrimiento

```
<query name="..." module="..." type="REST" transport="GET|POST" encode="Json"
  discovery="yes">
  <url>ruta/relativa</url>
  [<transform language="L" function="f"/>]
  [<parameter urlparameter="nombre" order="N"/>]
  <item name="clave" path="lista:[]" [transform="L:f"]/>
  <field name="MACRO" path="campo"/>
</query>
```

- Obligatorio:** discovery="yes"; type="REST"; transport; encode="Json"; un <url>; uno o más <item> con path (con [] o *); <field> con path.
- Opcional:** <transform>; <parameter> con urlparameter; transform en los ítems.
- No admite:** type, built_in y default en los ítems.

A.4.4 Consulta SOAP

```
<query name="..." module="..." type="SOAP" transport="POST">
  <url>ruta/del/servicio</url>
  <method uri="espacio:de:nombres" name="metodo" [action="accion"]>
    [<argument name="arg" type="string|int" value="valor"/>]
  </method>
  [<transform language="L" function="f"/>]
  <item name="clave" path="a:b" type="INT|STRING" [default="v"]/>
</query>
```

Obligatorio: type="SOAP"; transport="POST"; un <url>; un <method> con name; uno o más <item> con path y type (o built_in).

Opcional: <argument>; <transform>; default y transform en los ítems.

No admite: discovery; <field>. Los <parameter> se aceptan pero no tienen efecto en esta versión.

A.5 Motor MQTT

A.5.1 Lectura de valor

```
<query name="..." module="...">
  <parameter order="1" [type="STRING"]/>
  <item name="ts.mqtt.getvalue"/>
</query>
```

Obligatorio: un <parameter> (el *topic*); el <item> ts.mqtt.getvalue.

Opcional: —

No admite: atributos en el <item> distintos de name; <field>.

A.5.2 Descubrimiento de topics

```
<query name="..." module="..." discovery="yes">
  <parameter order="1" [type="STRING"]/>
  <item name="ts.mqtt.discovery"/>
  <field name="MACRO"/>
</query>
```

Obligatorio: discovery="yes"; exactamente un <parameter> (el patrón); el <item> ts.mqtt.discovery; exactamente un <field>, solo con name.

Opcional: —

No admite: atributos en el <item> o el <field> distintos de name.

Apéndice B

Valores por defecto

Parámetro	Valor
-c (fichero global)	/etc/databbix/databbix.conf
-p (dir. de monitores)	/etc/databbix/dbs
General.QueryFile	/etc/databbix/query.xml
General.Engine	Database
General.RetryOnError	60
General.MaxCache	10
Zabbix.Refresh	300
Zabbix.MaxItemsPerPacket	100
Zabbix.PassivePort	0
Data.MaxErrors	5
Data.ConnLostOnDefaultCtxError	yes
Log.LogPath	/var/log/databbix
Log.LogLevel	3
Log.LogMaxSize	0
Log.LogMaxBackupCopies	4
Log.MaxQueryExecTimeLog	10
Webservice.Timeout / MQTT.Timeout	20000 ms
MQTT.Port	1883