CERN

A never-ending database migration

Charles Delort IT-DB

November 20, 2017

Table of Contents

Years ago, decisions were made

DISCLAIMER

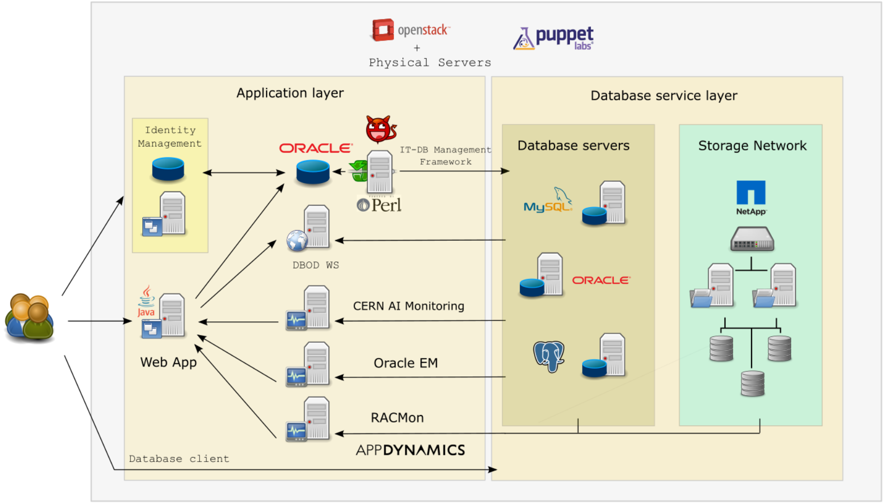
The whole application is very small compared to what you probably have

- The first version of DB On Demand was built using an Oracle database back-end (a schema in ITCORE like any other you can obtain in the Oracle services)
- Integration with FIM is done via triggers and schema object sharing between the **FIM** schema and the **db_ondemand** schema.
- The database schema was designed to be simple:
  - Very few tables
  - but also: no ids and complex multi-column keys (including Foreign Keys)

Years ago, decisions were made

- Instance metadata was not stored in the DB but in an LDAP server, which is used as source of truth for many internal IT-DB tools.
  - The LDAP schema was designed for Oracle-centric information, and hard to extend or modify
  - This led to duplicated information as some items were needed in both places
  - Duplicated data sources means you have to enforce consistency, which can get tricky
- Some PL/SQL was taking care of application logic a bit "behind the scenes"
- Different clients accessing directly the database. Database changes require deploying new client versions at the same time if there are breaking changes.
- Scheduled tasks on the Oracle Scheduler

The DB On Demand Infra (then)



A few years later

We had the following picture:

- A successful and expanding service :)

but also:

- A database design nobody on the team had worked on (with additional "evolution").
- Some application logic in the database (PL/SQL)
- Dependency with other Oracle schemas/DBs (e.g: DB links)
- Dependency on vendor specific features (e.g: Scheduler)

A few years later

With the experience of running the server for a few years and having migrated from Quattor to Puppet and extended the platform to support.
We wanted:

- A Single point of access for ALL the service info
- Based on either PostgreSQL or MySQL
- A better schema for service extensions (i.e: new systems supported, clusters, etc.)
- Avoid hard migration, if possible.

PostgreSQL Foreign Data Wrappers

PostgreSQL extensions enabling the PostgreSQL server to interact with different remote data stores, from other SQL databases to NoSQL systems to flat files:

```
-- Run as administrator
CREATE EXTENSION oracle_fdw;

CREATE SERVER <server_name> FOREIGN DATA WRAPPER oracle_fdw
    OPTIONS (dbserver '<server_name>');
GRANT USAGE ON FOREIGN SERVER <server_name> TO <dbuser>;

CREATE USER MAPPING FOR <dbuser> SERVER <server_name>
    OPTIONS (user '<oracle_username>', password '<oracle_password>');
```

PostgreSQL Foreign Data Wrappers

```
CREATE SCHEMA IF NOT EXISTS fdw;

------------------------------
-- CREATION OF FOREIGN TABLES
------------------------------

-- DOD_COMMAND_DEFINITION
CREATE FOREIGN TABLE fdw.dod_command_definition (
    command_name varchar(64) NOT NULL,
    type varchar(64) NOT NULL,
    exec varchar(2048),
    category varchar(20)
)
SERVER oradb
OPTIONS (
    schema 'DBONDEMAND_TEST',
    table 'DOD_COMMAND_DEFINITION'
);
ALTER FOREIGN TABLE fdw.dod_command_definition ALTER COLUMN command_name OPTIONS (key 'true');
```

PostgreSQL Foreign Data Wrappers

```
------------------------------------
-- VIEWS FOR BACKWARD COMPATIBILITY
------------------------------------

CREATE OR REPLACE VIEW public.dod_instances AS
SELECT * FROM fdw.dod_instances;

CREATE OR REPLACE VIEW public.dod_command_definition AS
SELECT * FROM fdw.dod_command_definition;

CREATE OR REPLACE VIEW public.dod_command_params AS
SELECT * FROM fdw.dod_command_params;
```

PostgreSQL Foreign Data Wrappers

```
postgres@dbod-dbod01:dbod> describe dod_command_definition;
+--------------+-------------------------+-------------+---------------+
| Column       | Type                    | Modifiers   | FDW Options   |
|--------------+-------------------------+-------------+---------------|
| command_name | character varying(64)   |  not null   | (key 'true')  |
| type         | character varying(64)   |  not null   | (key 'true')  |
| exec         | character varying(2048) |             |               |
| category     | character varying(20)   |             | (key 'true')  |
+--------------+-------------------------+-------------+---------------+
Server: itcore_dbod
FDW Options: (schema 'DBONDEMAND', "table" 'DOD_COMMAND_DEFINITION')

+--------------+-------------------------+-------------+
| Column       | Type                    | Modifiers   |
|--------------+-------------------------+-------------|
| command_name | character varying(64)   |             |
| type         | character varying(64)   |             |
| exec         | character varying(2048) |             |
| category     | character varying(20)   |             |
+--------------+-------------------------+-------------+
Time: 0.009s
```

First step of Migration

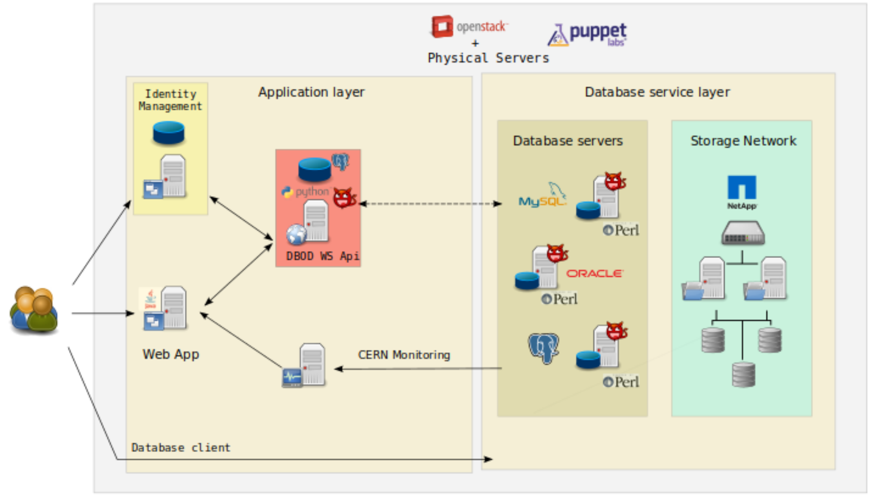
- With the FDW in place we started to model the new schema around the FDW version of the old one

Decoupling direct access to what we'll call the database INNER schema using an additional schema with views

- INNER schema
- API schema: accessed by clients, interacts with the INNER schema via VIEWS and stored procedures

- We could maintain "legacy" components talking with the old database
- All new developments now using the new one

The DB On Demand Infra (now)

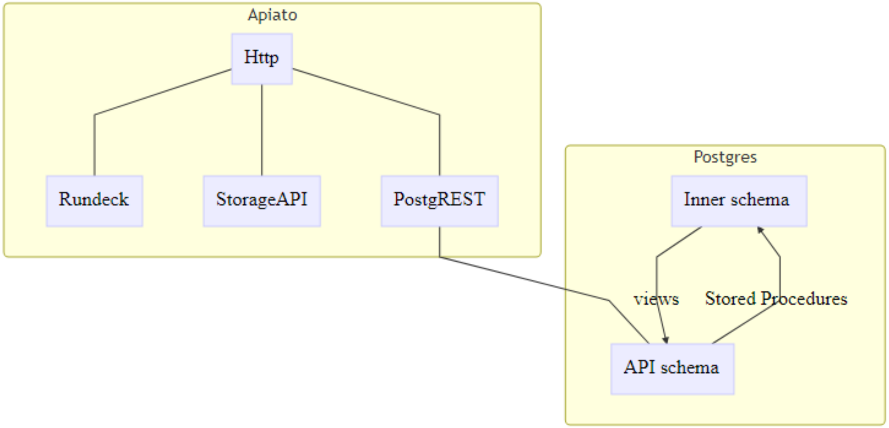


Two use cases, One API: Apiato

- REST API, written in Python using Tornado
- Used by DB On Demand and the CERN Nile Streaming service (Apache Kafka)

- Single point of entry for any component to service data (database)
- Wraps interactions with 3rd party APIs (Rundeck, StorageAPI, FIM, ...)
- **Clients**: Web Interface, Puppet (using custom Facts), Instance Actuators

Apiato: Overview



Apiato: Design choices

- Needs to support different use cases with the same core components
- The application code and the **inner** database schema are common for all instances
- DBOD/Nile specific implementations are defined in the API database schema, which is then exposed with PostgREST.

PostgREST
Serve a RESTful API from any Postgres database

- PostgREST is a standalone web server that turns your PostgreSQL database directly into a RESTful API

Apiato: Design choices

- When the Nile team started using Apiato they didn't have any legacy codebase to maintain compatibility with, so they started using directly the new database model from the first moment.
- The Nile team continued developing new features (improving cluster support in the database model) while the DBOD continued working in improving deployment and cutting dependencies with the old *stack*.

What is PostgREST

- Connects to a DB and exposes a CRUD REST API to a database schema

```
# postgrest.conf

# The standard connection URI format, documented at
# https://www.postgresql.org/docs/current/static/libpq-connect.html#AEN45347
db-uri       = "postgres://user:pass@host:5432/dbname"

# The name of which database schema to expose to REST clients
db-schema    = "api"

# The database role to use when no client authentication is provided.
# Can (and probably should) differ from user in db-uri
db-anon-role = "anon"
```

- JWT support
- Latest versions generate Swagger docs for the API

What is PostgREST

```
 create table api.todos (
   id serial primary key,
   done boolean not null default false,
   task text not null,
   due timestamptz
  );
  insert into api.todos (task) values
    ('finish tutorial 0'), ('pat self on back');

curl http://localhost:3000/todos

  [{
      "id": 1,
      "done": false,
      "task": "finish tutorial 0",
      "due": null
    },
    {
      "id": 2,
      "done": false,
      "task": "pat self on back",
      "due": null
   }]
```

What is PostgREST

Warning

- Using a direct REST API to interact with the DB involves some decisions
- Transactions need to be encapsulated in stored procedures or managed on the application side

Apiato: DB Schemas

```
postgres> \c dbod
You are now connected to database "dbod" as user "postgres"
Time: 0.011s
postgres@dbod-dbod01:dbod> \dn
+-----------------+---------+
| Name            | Owner   |
|-----------------+---------|
| api             | dbod    |
| fdw             | dbod    |
| password_checker | dbod    |
| public          | dbod    |
+-----------------+---------+
SELECT 4
Time: 0.003s
postgres@dbod-dbod01:dbod>
```

Apiato: Example of instance metadata

```
{
"active": true,
"attributes": {
  "buffer_pool_size": "1G",
  "eos_archive": "true",
  "eos_backup": "true",
  "notifications": "true",
  "port": "5500"
},
"basedir": "/usr/local/mysql/mysql-5.7.15",
"class": "REF",
"db_name": "pinocho",
"db_type": "MYSQL",
"hosts": [
  "db-gc505"
],
"id": 20,
"logdir": "/ORA/dbs02/PINOCHO/mysql",
"socket": "/var/lib/mysql/mysql.sock.pinocho.5500",
"state": "RUNNING",
"username": "icoteril",
```

Apiato Example of instance metadata

```
"version": "5.7.15",
 "volumes": [
   {
     "file_mode": "0755",
     "group": "mysql",
     "instance_id": 20,
     "mount_options": "rw,bg,hard,nointr,tcp,vers=3,noatime,timeo=600,rsize=65536,wsize=65536",
     "mounting_path": "/ORA/dbs02/PINOCHO",
     "owner": "mysql",
     "server": "dbnash5141"
   },
   {
     "file_mode": "0755",
     "group": "mysql",
     "instance_id": 20,
     "mount_options": "rw,bg,hard,nointr,tcp,vers=3,noatime,timeo=600,rsize=65536,wsize=65536",
     "mounting_path": "/ORA/dbs03/PINOCHO",
     "owner": "mysql",
     "server": "dbnash5111"
   }
  ]
 }
```

An example: Configuration with Puppet

We use a **Puppet** custom fact (**$::dbod_instances**) which queries the Apiato cluster in each Puppet run on any given server and fetches the set of metadata for all the instances hosted in that particular server.
Customization according to the instance metadata is then applied:

```
# Operate on dbod_instances fact
if (is_hash($::dbod_instances)) {
  $defaults = {
    require => [
      Class['::dbod::certificates'],
      Class['::dbod::users'],
      ],
  }
  create_resources(dbod::instance, $::dbod_instances, $defaults)
}
```

Job Scheduling

We have some kind of feature lock in with the Oracle DBMS scheduler:

- Scheduled instance tasks (backups, cleanups)
- Internal operations like FIM syncing, instance expiration notifications, etc.

We are looking at two alternatives:

- **Rundeck**
- **pg_cron** (Another PostgreSQL extension)

```
-- run as superuser:
  CREATE EXTENSION pg_cron;

  -- optionally, grant usage to regular users:
  GRANT USAGE ON SCHEMA cron TO <user>;
```

Job Scheduling on PostgreSQL

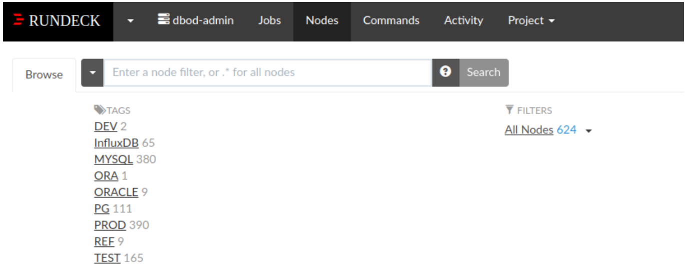
```
-- Delete old data on Saturday at 3:30am (GMT)
SELECT cron.schedule('30 3 * * 6',
        $$DELETE FROM events WHERE event_time < now() - interval '1 week'$$);
 schedule
----------
          42

-- Vacuum every day at 10:00am (GMT)
SELECT cron.schedule('0 10 * * *', 'VACUUM');
 schedule
----------
          43

-- Stop scheduling a job
SELECT cron.unschedule(43);
 unschedule
------------
                  t
```

Rundeck

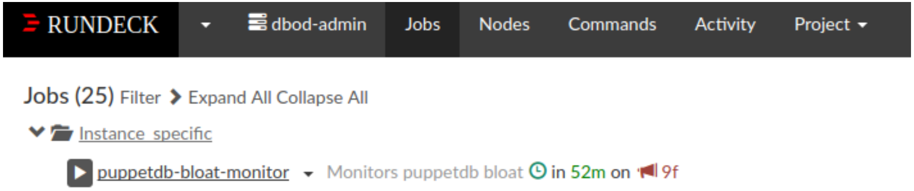
- A workflow manager for operations supporting many different executors (e.g: local executables, Kerberos SSH, HTTP, etc.), job scheduling, ACL, REST API interactions, ...
- We use Apiato to generate a dynamic list of Rundeck targets, as well as to interact with the Rundeck API.



DEV	2
InfluxDB	65
MYSQL	380
ORA	1
ORACLE	9
PG	111
PROD	390
REF	9
TEST	165

All Nodes 624

Job Scheduling with Rundeck



Jobs (25) Filter › Expand All Collapse All

Instance_specific

puppetdb-bloat-monitor Monitors puppetdb bloat in 52m on 9f

Summary

Issues we overcame during the migration:

- Avoid hard migration: postgres FDW
- Replace DBMS Scheduler: Rundeck and/or pgcron
- Avoid duplication and remove hidden application logic from the database: Redesign our application

Where are we now. Next steps

- We are in the middle of merging the Nile clustering development into the main Apiato branch.
- After that change is done and rolled out, only the integration with the FIM service and the Job Scheduling will rely on the legacy oracle model.
- In contact with the FIM team in order to have an API to interact with FIM. Ideally this will also allow us to stream line the instance request process.

A note on PostgreSQL extensions on DBOD

- Extensions require DB **superuser** role to be loaded.
- If you need to use an extension please let us know with a SNOW Request.
- We distribute extensions compiled and packaged with the server binaries.
- If they are not loaded they don't affect the database server at all.

www.cern.ch