

Lessons learned from running Rails apps on-premise

Andy Pfister, Simplificator AG



How Did We Get Here?

Simplificator builds custom software ...

How Did We Get Here?

Simplificator builds custom software ...

... docuteam provides services and software for digital archives ...

How Did We Get Here?

Simplificator builds custom software ...

... docuteam provides services and software for digital archives ...

... these services are also provided to customer on-premise ...

How Did We Get Here? 🛒

Simplificator builds custom software ...

... docuteam provides services and software for digital archives ...

... these services are also provided to customer on-premise ...

... "hey, couldn't we simplify our on-premise deployment?"

Windows  / Ubuntu 

MySQL  / Microsoft SQL Server  /
PostgreSQL 

Offline installation 

Support different database systems 

This is not a lot of trouble ...

Rails (ActiveRecord) -> database

```
group :mssql do
  gem "activerecord-sqlserver-adapter"
end
```

```
group :mysql do
  gem "mysql2"
end
```

```
group :pg do
  gem 'pg'
end
```

```
bundle config set without "mssql mysql"
bundle install
```


There are minor nuances

There are minor nuances

MySQL needs charset in schema.rb

```
create_table "active_storage_attachments", charset: "utf-8", force: :cascade do |t|
  t.string "name", null: false
  ...
end
```

There are minor nuances

MySQL needs charset in schema.rb

```
create_table "active_storage_attachments", charset: "utf-8", force: :cascade do |t|
  t.string "name", null: false
  ...
end
```

MySQL has LONGTEXT (4GB) vs PostgreSQL (1GB)

There are minor nuances

MySQL needs charset in schema.rb

```
create_table "active_storage_attachments", charset: "utf-8", force: :cascade do |t|
  t.string "name", null: false
  ...
end
```

MySQL has LONGTEXT (4GB) vs PostgreSQL (1GB)

MSSQL needs special annotation for JSON columns

```
class MyModel < ApplicationRecord
  if ActiveRecord::ConnectionAdapters.const_defined? :SQLServerAdapter
    attribute :my_json_column, ActiveRecord::Type::SQLServer::JSON.new
  end
end
```

You can capture almost anything with Continuous integration

We settled with Azure pipelines

web app: Run tests (Linux)		
>	✔ Run tests PostgreSQL 12	3m 21s
>	✔ Run tests PostgreSQL 13	3m 45s
>	✔ Run tests PostgreSQL 14	3m 37s
>	✔ Run tests PostgreSQL 15	3m 15s
>	✔ Run tests PostgreSQL 16	3m 33s
>	✔ Run tests MSSQL 2017	5m 8s
>	✔ Run tests MSSQL 2019	4m 52s
>	✔ Run tests MSSQL 2022	4m 41s

```
jobs:
- job: "Test job"
  strategy:
    matrix:
      "MySQL 5.7":
        container_image: "mysql:5.7"
        database_url: "mysql2://mysql:mymysqlpassword@db/pipelines"
      "PostgreSQL 16":
        container_image: "postgres:16"
        database_url: "postgresql://postgres:mypostgrespassword@db/pipelines"
      "MSSQL 2022":
        container_image: "mcr.microsoft.com/mssql/server:2022-latest"
        database_url: "sqlserver://sa:mymicrosoftpassword@db/pipelines"
```

To recap

- Organize database adapters in different groups
- Use CI to test against multiple databases

 **Support different operating systems** 

Regardless of Windows or Linux ...

... your Ruby code (should) look(s) the
same



Installing Ruby ...

We only really need one Ruby version on a server

You could use `uru` to switch between versions

Installing Ruby on Windows ...

Get it from rubyinstaller.org

”

"MSYS2 (Minimal SYStem 2) is a collection of tools and libraries providing you with an easy-to-use environment for building, installing and running native Windows software."

Gems with native extensions

```
$ gem install puma --no-document
Fetching puma-6.4.2.gem
Fetching nio4r-2.7.1.gem
Building native extensions. This could take a while...
Successfully installed nio4r-2.7.1
Building native extensions. This could take a while...
Successfully installed puma-6.4.2
2 gems installed
```

On Linux, you typically have gcc available for compilation.

On Windows, RubyInstaller2 sets up MSYS2 for you.

```
$ gem install nokogiri --no-document
Fetching nokogiri-1.16.5-x86_64-linux.gem
Successfully installed nokogiri-1.16.5-x86_64-linux
1 gem installed
```

```
$ tree nokogiri
```

```
nokogiri
```

```
├── 3.0
```

```
│   └── nokogiri.so
```

```
├── 3.1
```

```
│   └── nokogiri.so
```

```
├── 3.2
```

```
│   └── nokogiri.so
```

```
├── 3.3
```

```
│   └── nokogiri.so
```

```
├── class_resolver.rb
```

```
├── css
```

```
│   ├── node.rb
```

```
│   └── parser_extras.rb
```

Where am I going with this?

For Windows, there are more precompiled gems

It might take longer, until you can go to the next Ruby version.

The Windows pipeline

We only run one job per database system to save time.

web app: Run tests (Windows)		
>	✓	Run tests PostgreSQL 14 11m 4s
>	✓	Run tests MSSQL 2017 12m 54s
web app: Run tests (Linux)		
>	✓	Run tests PostgreSQL 12 3m 21s
>	✓	Run tests PostgreSQL 13 3m 45s
>	✓	Run tests PostgreSQL 14 3m 37s
>	✓	Run tests PostgreSQL 15 3m 15s
>	✓	Run tests PostgreSQL 16 3m 33s
>	✓	Run tests MSSQL 2017 5m 8s
>	✓	Run tests MSSQL 2019 4m 52s
>	✓	Run tests MSSQL 2022 4m 41s

The Windows pipeline

We only run one job per database system to save time.

Azure DevOps only has a few selected Ruby versions pre-installed

web app: Run tests (Windows)		
>	✓	Run tests PostgreSQL 14 11m 4s
>	✓	Run tests MSSQL 2017 12m 54s
web app: Run tests (Linux)		
>	✓	Run tests PostgreSQL 12 3m 21s
>	✓	Run tests PostgreSQL 13 3m 45s
>	✓	Run tests PostgreSQL 14 3m 37s
>	✓	Run tests PostgreSQL 15 3m 15s
>	✓	Run tests PostgreSQL 16 3m 33s
>	✓	Run tests MSSQL 2017 5m 8s
>	✓	Run tests MSSQL 2019 4m 52s
>	✓	Run tests MSSQL 2022 4m 41s

The Windows pipeline

We only run one job per database system to save time.

Azure DevOps only has a few selected Ruby versions pre-installed

Sometimes we have to install the entire database system (like MSSQL)

web app: Run tests (Windows)		
>	✓	Run tests PostgreSQL 14 11m 4s
>	✓	Run tests MSSQL 2017 12m 54s
web app: Run tests (Linux)		
>	✓	Run tests PostgreSQL 12 3m 21s
>	✓	Run tests PostgreSQL 13 3m 45s
>	✓	Run tests PostgreSQL 14 3m 37s
>	✓	Run tests PostgreSQL 15 3m 15s
>	✓	Run tests PostgreSQL 16 3m 33s
>	✓	Run tests MSSQL 2017 5m 8s
>	✓	Run tests MSSQL 2019 4m 52s
>	✓	Run tests MSSQL 2022 4m 41s

To recap

- The Ruby code looks mostly the same across different OS
- Precompiled gems can delay Ruby upgrades
- Use CI to test against different OS
- Windows build could be slow; keep a VM around

Offline installation 

What does that mean?

We need to ship everything that our Rails app needs to run in a package.

For other languages, this is easy ...

 =>  + JAR files

 => Executable

 =>  + Gems + your code

What about Docker?

TL;DR not a good solution on Windows (Server) ...

Linux Container on Windows require a "Linux virtual machine" (WSL)

Windows Container have large image sizes and limited compatibility between different OS versions





Assets ship precompiled - no ExecJS dependency on runtime.

```
$ bundle cache --all-platforms
...
Using toxiproxy 2.0.2
Bundle complete! 9 Gemfile dependencies, 10 gems now installed.
Use `bundle info [gemname]` to see where a bundled gem is installed.
Updating files in vendor/cache
Fetching gem metadata from https://rubygems.org/....
* rake-13.0.6.gem
* connection_pool-2.2.5.gem
* mini_portile2-2.5.3.gem
* minitest-5.14.4.gem
* minitest-ci-3.4.0.gem
* rake-compiler-1.2.1.gem
* rake-compiler-dock-1.4.0.gem
* toxiproxy-2.0.2.gem
```


How do we install it?

There is a custom PowerShell script which takes care of:

- Extracting the files
- Installs Ruby and Gems (from vendor/cache)
- Setting up the required configuration files (like database.yml)
- Running Rails migrations
- Configuring the required Windows services

This solution is not exclusive to our Rails applications ...

We combine several softwares which make sense to be deployed together and then run a job on CI for them.

After installation, we do some smoke tests (hello, are you there? 📞)

Why PowerShell?

- Ansible does not work on Windows.
- The On-Premise team at docuteam is proficient with PowerShell, allowing them to contribute.
- We wanted to avoid introducing additional technologies (like Chef, InstallShield, Chocolatey / NuGet), keep it simple.
- Promising: github.com/Largo/ocran (Turn Ruby projects into .exe files on Windows)

To recap

- A ZIP file with Ruby, gems and code
- A PowerShell script which helps with installation
- CI for confidence in our deployment scripts

Releases 

Every commit is a potential release.

Loosly based from Continuous Delivery: Reliable Software Releases Through Build, Test, and Deployment Automation by David Farley and Jez Humble

At Simplificator, every commit on the main branch is shipped to production.

For an on-premise scenario, this is rather difficult

We might not have direct access to the system from a pipeline.

Or certain policies are in place which take time per release.

So there has to be a compromise.

We do regular releases, which get shipped soon to the cloud offering.

On-Premise customer get (usually) one update a year, the so-called annual release.

Semantic versioning

The annual release is usually major release (and the only one per year)

We avoid to break features between two major releases, but rather "deprecate" first before rework or removal.

This requires some planning ahead.

- July 2021: Implementation of a replacement feature
- April 2022: Shipped to on-premise customers
- January 2023: Original feature removed from code base.
- May 2023: Shipped to on-premise customers


To recap

- Continuous deployment to on-premise infrastructure likely not possible
- Find a regular update cycle for your on-premise customers
- Decide if you want to break things in-between major releases

Thank you very much! ❤️

: andyundso

: @docker_enjoyer@ruby.social

: andy.pfister@simplificator.com /
andy.pfister@hotmail.ch