metis

Lessons Learned From Maintaining SDK For Three Years

Adam Furmanek

https://www.metisdata.io/

What We Do?

metis

Goal

✓ Metis needs:

- REST
- SQL query
- Execution plan

✓ Applications:

- Web APIs
- Local or in the cloud
- Modern
- With CI/CD

✓ Tenets:

- Easy to use
- One-time integration
- No code changes
- No dependencies

How It Works

What:

- What interaction happened (*API X was called*)
- What query was executed (with parameters)
- What was the execution plan

How:

- Use OpenTelemetry to capture the interactions
- Extract details from REST and from SQL
- Capture the query parameters
- Ask for the execution plan (with *EXPLAIN* keyword)
- Send everything to Metis

Three Different Approaches

✓ SDK per tech stack
- One library for each tech stack
- No database changes

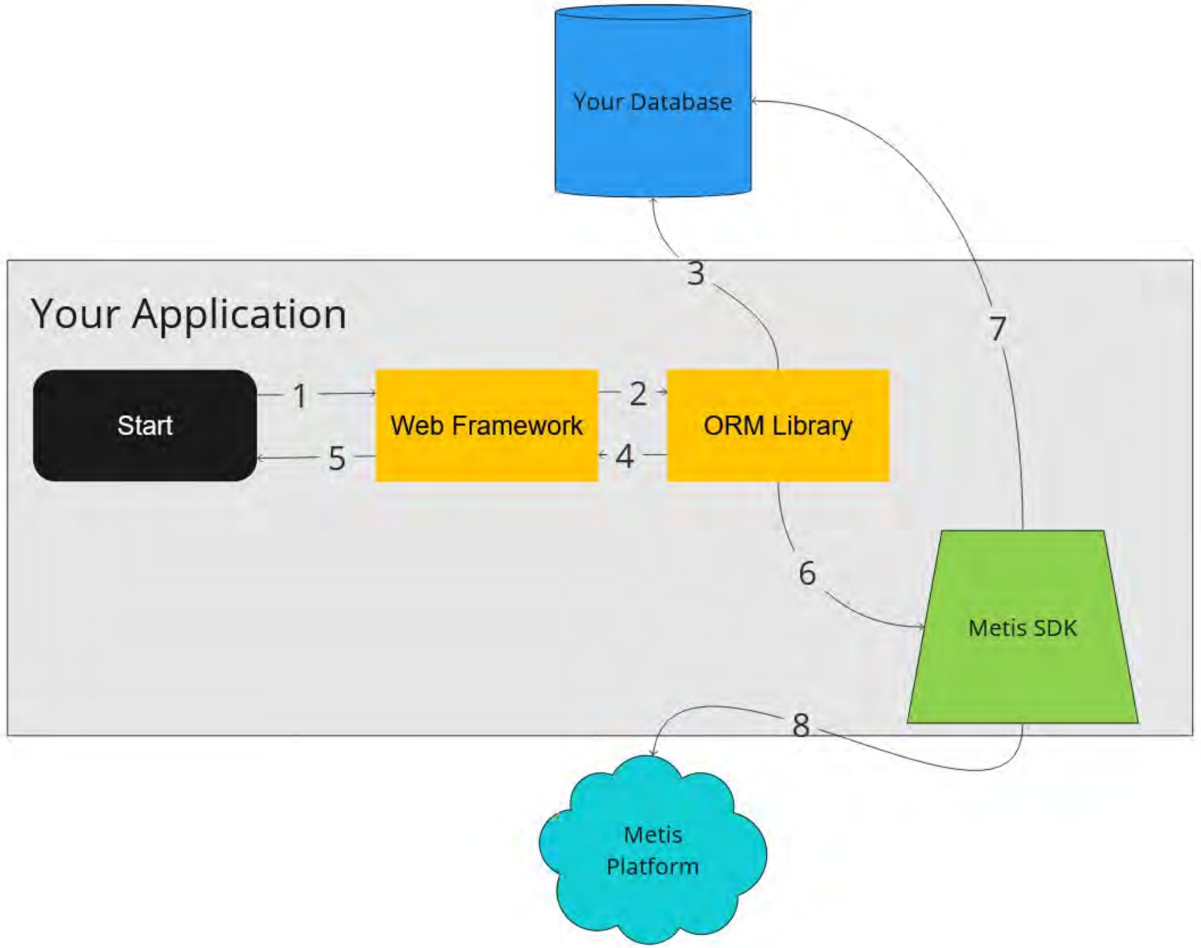
✓ Reading from the database
- Library for each tech stack
- Changes to the database
- Agent

✓ Moving the ownership
- No specific library
- No changes to the database
- Agent

metis

SDK Per Tech Stack

metis


Your Database
Your Application
Start
1
Web Framework
2
ORM Library
3
4
5
6
7
Metis SDK
8
Metis Platform

First Approach

Pros:

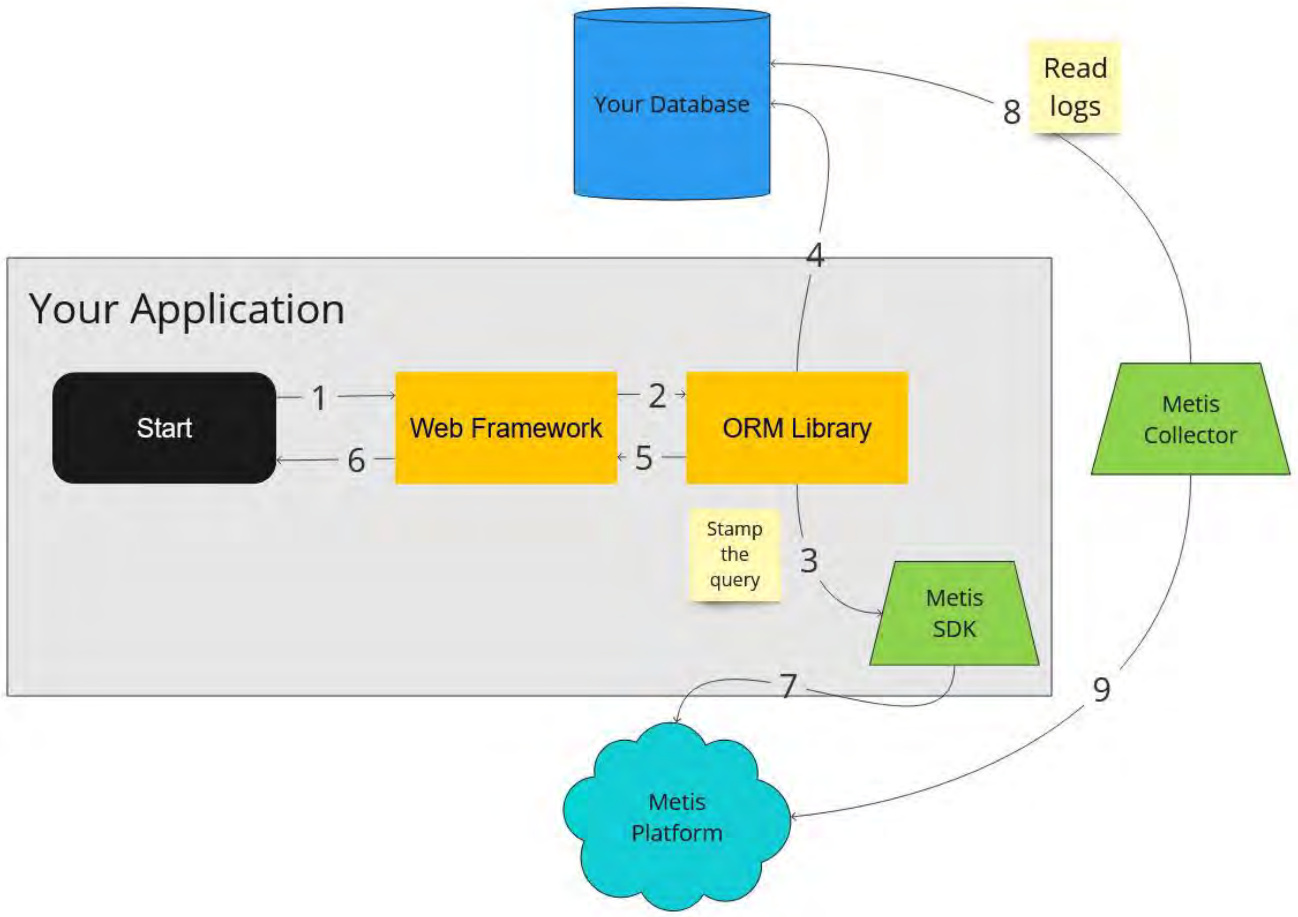
- Easy to install - just one command
- Integrates with the language
- No changes to the database
- Nearly no changes to the application code
- Works with automated tests (most of the time)
- Captures all the queries
- Can be easily disabled for production

Cons:

- No way to reuse the code between languages or libraries
- Differences between versions of dependencies
- Weird integrations with OpenTelemetry (lack of parameter values)
- Hard to correlate REST and SQL
- Problems with testing frameworks

Reading From The Database

metis


Your Database
Read logs
8
4
Your Application
Start
1
Web Framework
2
ORM Library
5
6
Stamp the query
3
Metis SDK
Metis Collector
7
9
Metis Platform

Second Approach

Pros:

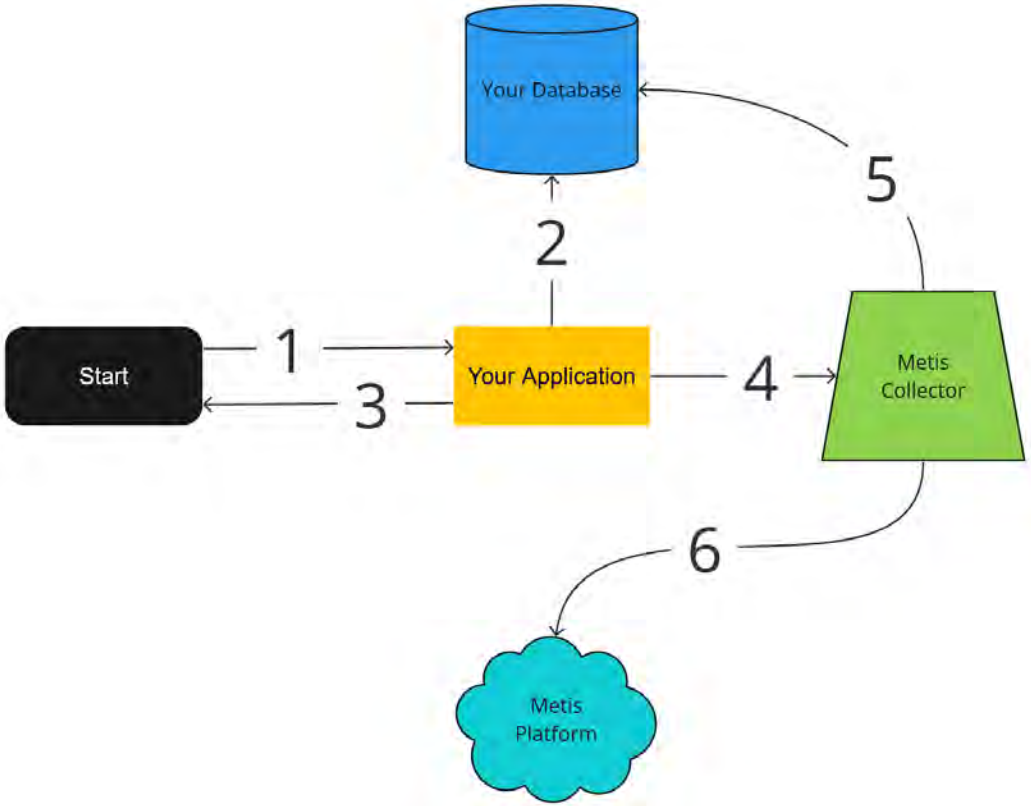
- Easy to install - just one command
- Integrates with the language
- Nearly no changes to the application code
- Works with automated tests (most of the time)
- Captures all the queries
- Can be easily disabled for production

Cons:

- Database must be reconfigured
- Hard to capture ephemeral databases (think of TestContainers)
- Difficult query stamping
- Very expensive
- No way to reuse the code between languages or libraries
- Differences between versions of dependencies
- Hard to correlate REST and SQL
- Problems with testing frameworks

Moving The Ownership

metis


Your Database
5
2
Start
1
Your Application
4
Metis Collector
3
6
Metis Platform

Third Approach

Pros:

- No changes to the application code*
- No changes to the database
- Integrates with the language
- Captures all the queries*
- Can be easily disabled for production
- We don’t own it!

Cons:

- Sometimes requires changes to the application code
- Not all libraries support auto-instrumentation
- Sometimes misses the queries or doesn’t capture parameter values
- Hard to correlate REST and SQL
- Problems with testing frameworks

metis

What We Learned

✓ Uniform Functionality

✓ Versions Management

✓ Diverse Languages

Uniform Functionality

metis

Uniform Functionality

- Languages are different
  - Static typing vs dynamic typing
  - Generics vs macros
  - Classes vs prototypes
- Features are nice but hard to port between technologies
- Idiomatic code vs reusing the implementation

- Can you represent structures uniformly between languages?
- Can you use the same protocols?
- Are there any implementation differences?
- How do you synchronize changes between languages?
- How can you introduce optional fields and evolve your schemas?
- How do you write documentation between languages?

JSON vs gRPC

JSON:

- JSON standard vs implementations
- Interpretation issues
- HTTP handling

gRPC:

- Single definition
- Strongly-typed and streamlined communication
- Consistency between languages

Proprietary vs well-known protocol

Proprietary:

- Control what and how you send
- Users need to learn it
- Most likely no libraries
- You own it forever and ever

Well-known:

- Open-source libraries available
- Users know how to use it*
- You don't need to own it
- You may need to squeeze your structures into existing definitions

Versions Management

metis

Version Management

- Semver shows what was changed
- Adding new features at the same pace
- Maintaining compatibility with older versions
- Adopting new language features

- How do you test things?
- How do you keep version numbers consistent between technologies?
- How to add features in all languages at once?
- What dependencies to use in different languages?
- What if dependencies differ?
- What about language-specific options?
- How to deal with logging between technologies?

Rigorous Testing

Testing:

- Isolate environments as much as possible - with Docker, TestContainers, Nix
- Run tests across all languages for each change
- Test all supported versions
- Try reproducing bugs in all languages
- Have uniform set of tests in all technologies

Tooling:

- Use tools for managing versions in one repository
- Be explicit about your dependencies
- Use as few tools for your CI/CD and installation process as possible
- Do not use things that may cause conflicts

Diversity

metis

Diversity

- Languages differ and nobody knows all of them
- Idiomatic code is nice to have but hard to write and maintain
- Using same code structure between languages makes it easier for maintenance but leads to worse results

- Have **Language Champion**
- Run regular sessions to share insights
- Have regular updates inside the team

Summary

- The less you maintain, the better
- Keep it consistent between platforms
- Think about backward and forward compatibility
- Test early, test often
- Rely on open standards
- Do not reinvent the wheel

metis

Q&A

metis

Thank you!

https://www.metisdata.io/