ORACLE

Ready, Set, REST!

Introducing MySQL's Built-In REST Service

Miguel Araújo

Senior Principal Software Engineer

MySQL, Oracle

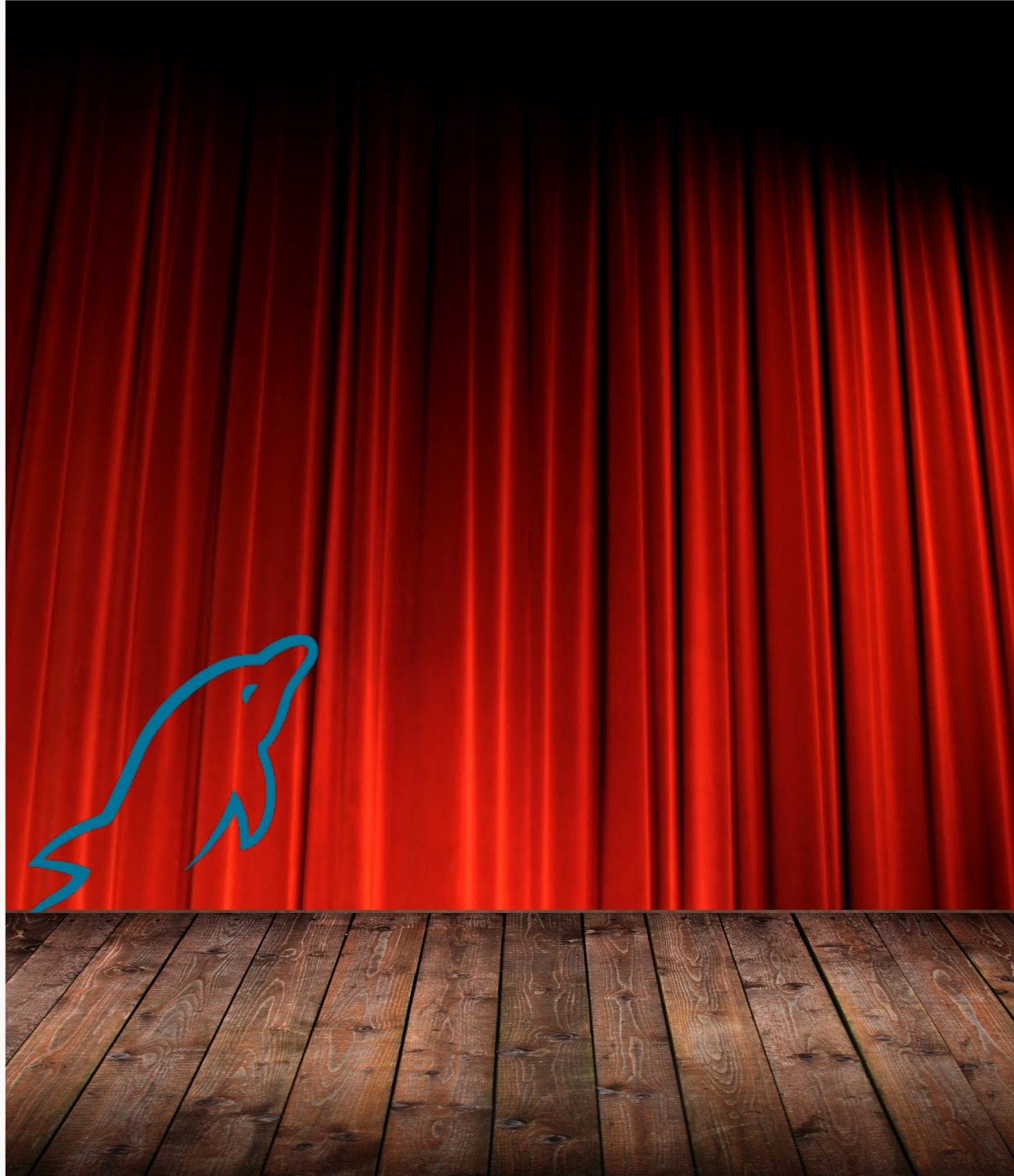
January 30, 2026

MySQL

```
$ whoami
miguel_araujo

$ curl -s ipinfo.io/country
PT

$ cat ~/.profile
Active Community User: MySQL Forums, Blogs, Slack, Conferences

$ history | grep MySQL
... <= 2009                     MySQL user & other jobs
[2009, 2011]                    Built a MySQL Proxy plugin enabling active-active
                                replication using a GCS
[2011, 2014]                    Joined Oracle/MySQL - MEM & MySQL Proxy development
[2014, 2017]                    MySQL Shell developer
[2017, 2024]                    AdminAPI Tech Lead & MySQL Database Architectures
[2025, now()]                   MySQL REST Service Component Tech Lead
```

Setting the stage

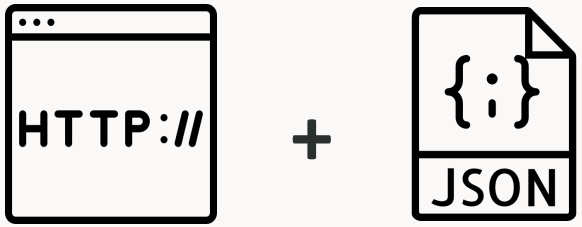
Why REST access matters today?



Why REST access matters today

Apps & Tools expect

- HTTP + JSON
- Zero-friction integration
- Stateless access



What teams end up building

- App-specific REST layers
- Duplicated auth & permission logic
- Custom serialization & validation
- Extra services to deploy & operate
- Drift from database security model
- ...

REST became the *lingua franca* at the application boundary

"So why should REST always live *'outside'* the database?"

What is the MySQL REST Service?

aka MRS

MySQL REST Service

"A next-generation JSON Document Store solution that enables fast and secure HTTPs REST access to MySQL data."

- Provides a RESTful web services interface for MySQL
- Designed for HTTP + JSON application access
- SQL-free access model for clients
- Exposes tables, views, stored procedures & functions

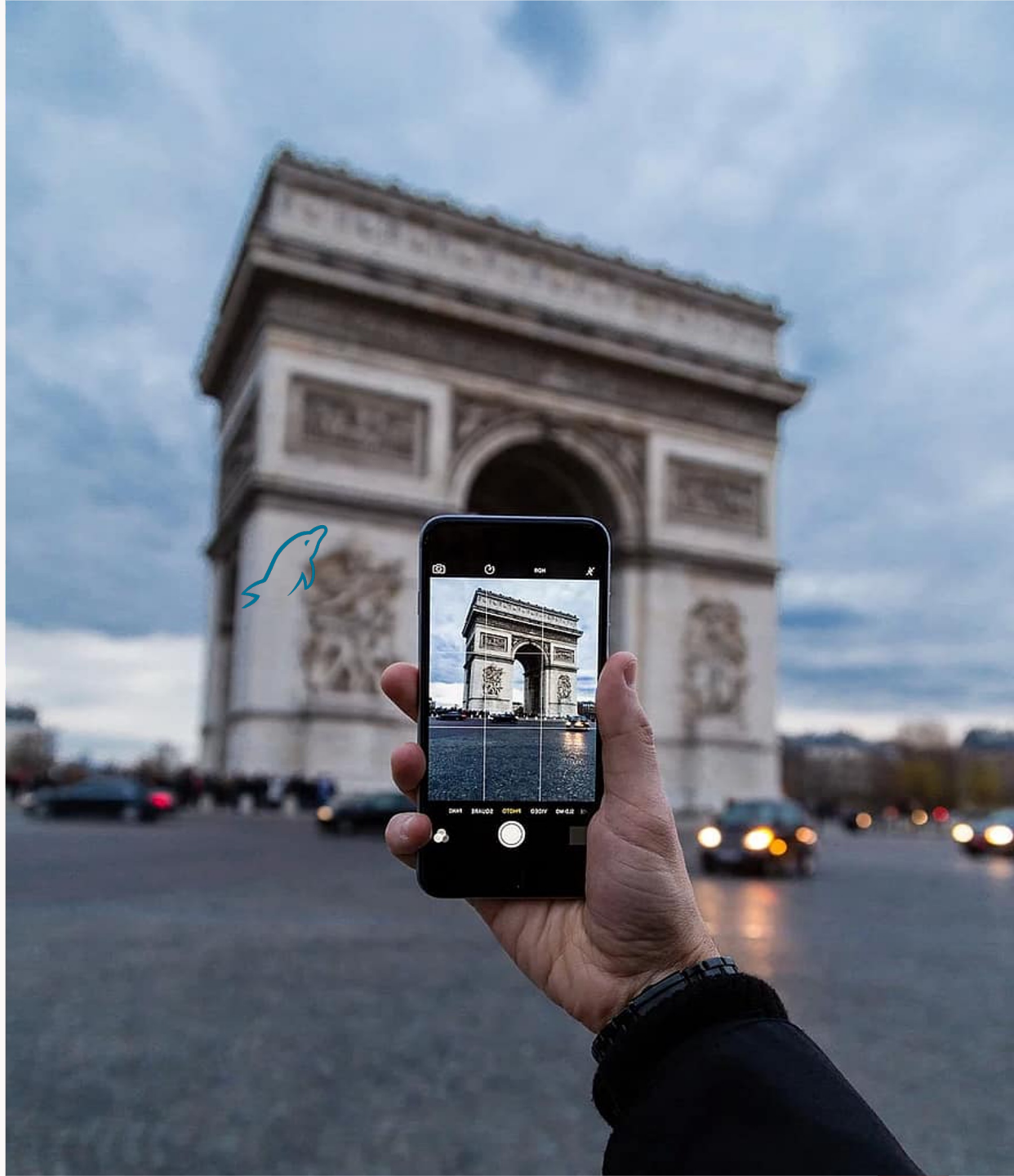
MySQL REST Service

MRS follows similar design principals to Oracle REST Data Services (ORDS)

- Initially developed and available as **middleware** via **MySQL Router**
- Available also as part of MySQL Heatwave

The goal

Bring the REST Service into the Server



From middleware to server-native

What MRS delivered well

- Clear **separation** of concerns
- **Proven** HTTP & REST stack via MySQL Router
- **Scales** horizontally with Router instances
- **Native support** on MySQL database architectures: InnoDB Cluster/ReplicaSet/ClusterSet
- Enables REST access without **modifying** MySQL Server
- Fast experimentation and iteration

What becomes easier inside the Server

- REST available **without** requiring MySQL Router
- **Reduced** operational and configuration setup
- **Native** authentication & authorization (MySQL users & privileges)
- **Unified** TLS & certificate management
- **Direct** access to server services:
  - Keyring
  - sysvars / statusvars
  - UDF-based control
- **Single** lifecycle, startup and shutdown model

REST for standalone MySQL

… not just InnoDB Cluster

What this enables

No Router or Cluster required

Same REST capabilities for single-node MySQL

Lower barrier to adoption

Extensibility in MySQL

The Components Infrastructure

MySQL Components Infrastructure

Service-oriented architecture

- Producers -> Registry -> Consumers
- Type agnostic registry

Modular and reusable by design

- Modular code
- APIs define the functionality
- Reusable across components and binaries
- Can be used by any binary
  (based on server-independent `libminchassis`)

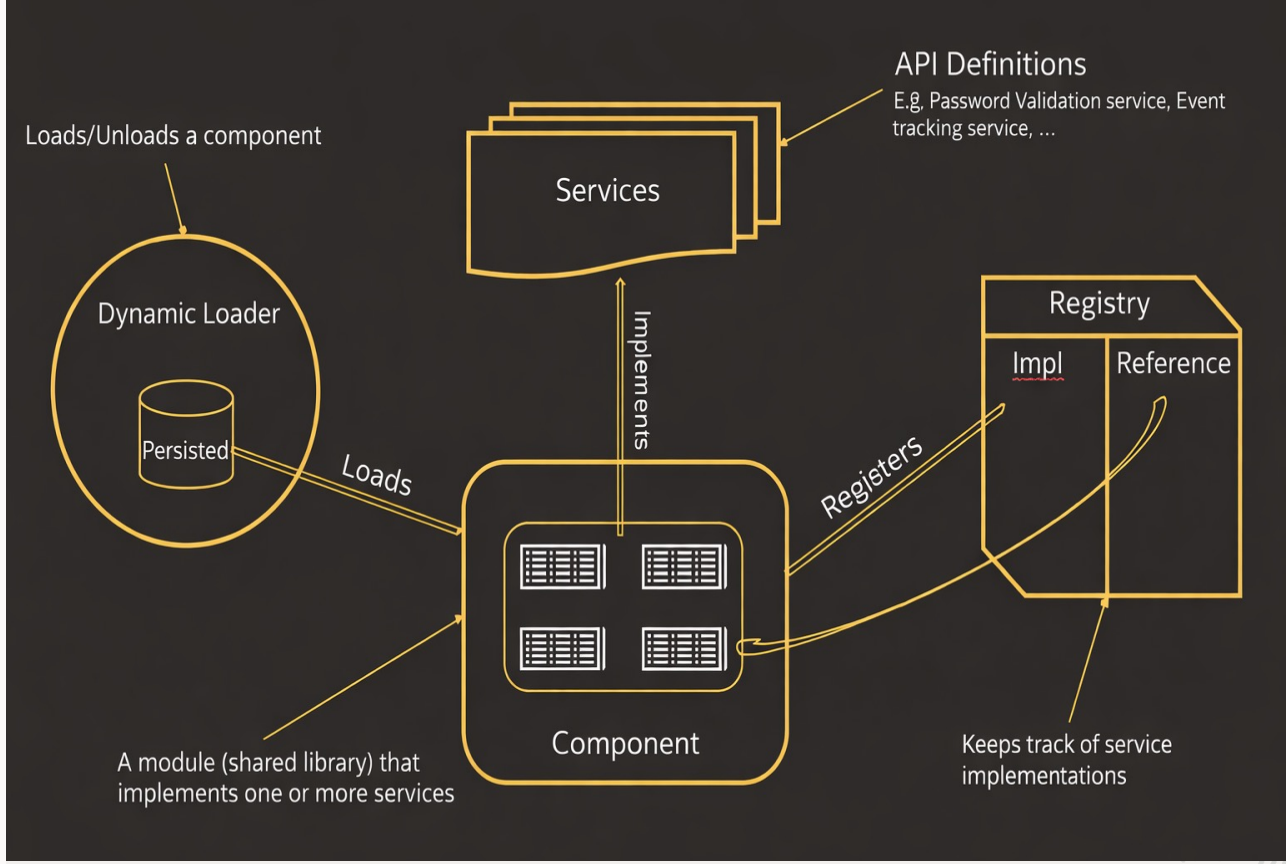
Stable and isolated

- Stable APIs
- No server interference in component-to-component communication

Safe and explicit dependencies

- Strong dependency tracking

  (explicit, trackable, enforced)



Explicit contracts. Predictable behavior. Safe extensibility.

The MRS Component architecture

MySQL as an HTTP server

MRS Component architecture

High level request lifecycle

1. **HTTPS REST request** from application
2. **Endpoints resolved** via MRS metadata
3. **Authorization** using MySQL **users & privileges**
4. **Data access** executed by MySQL Server
5. **JSON response** returned to the client

Inside `mysqld`
(single process)

MRS Component

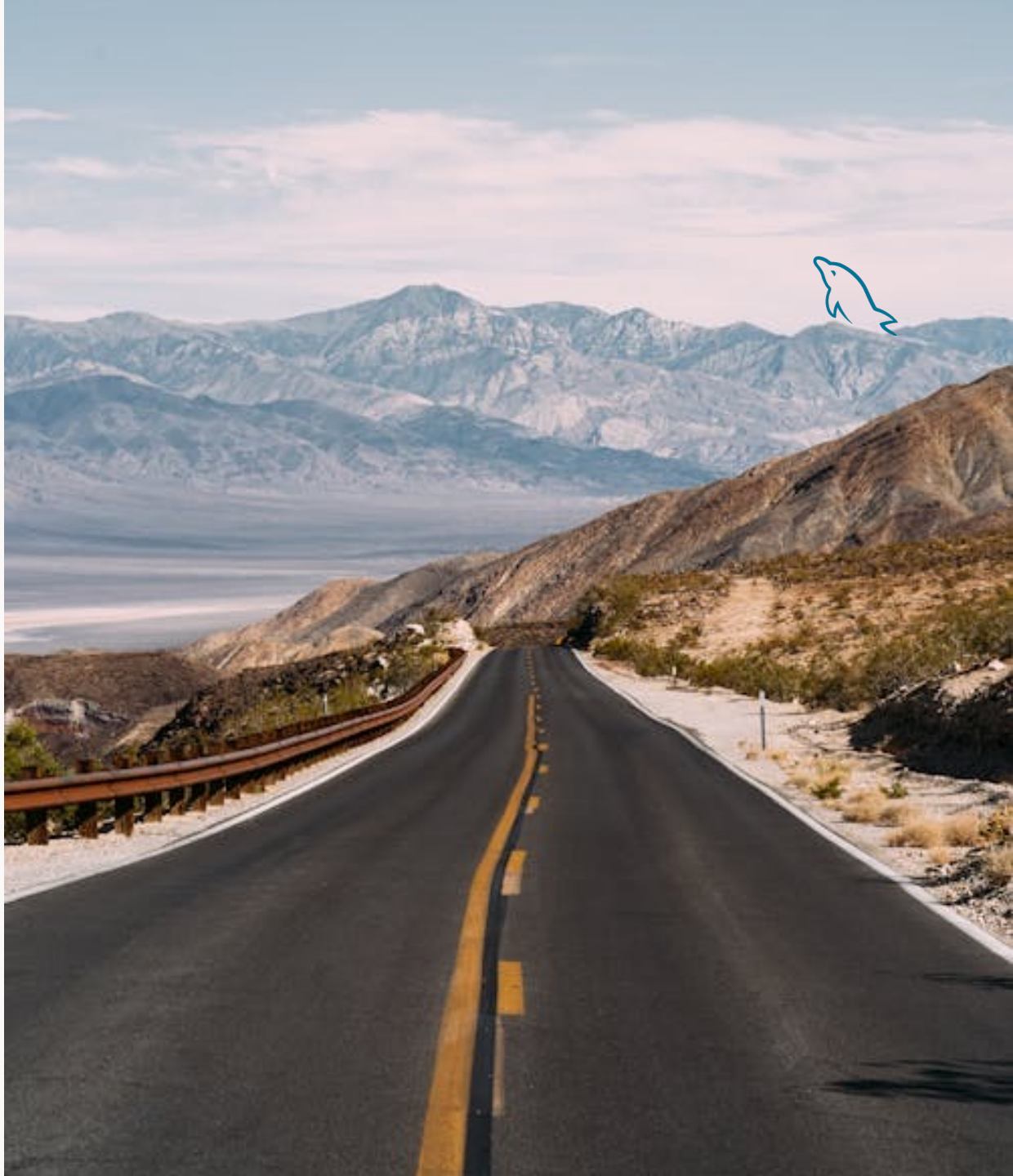
- HTTP handling
- REST endpoint logic
- Metadata-driven routing

MySQL Server Services

- Auth & Privileges
- Schema access
- Keyring
- TLS / SSL
- sysvars / status vars
- UDF-based control, etc.

How we built it

The engineering journey



Engineering journey

Prototype
Can this even run inside mysqld?

New component boilerplate
Component skeleton, lifecycle, services

Reuse
Extract Router plugins safely (http, io, mrs, ...)

Integration
Auth, Metadata, keyring...

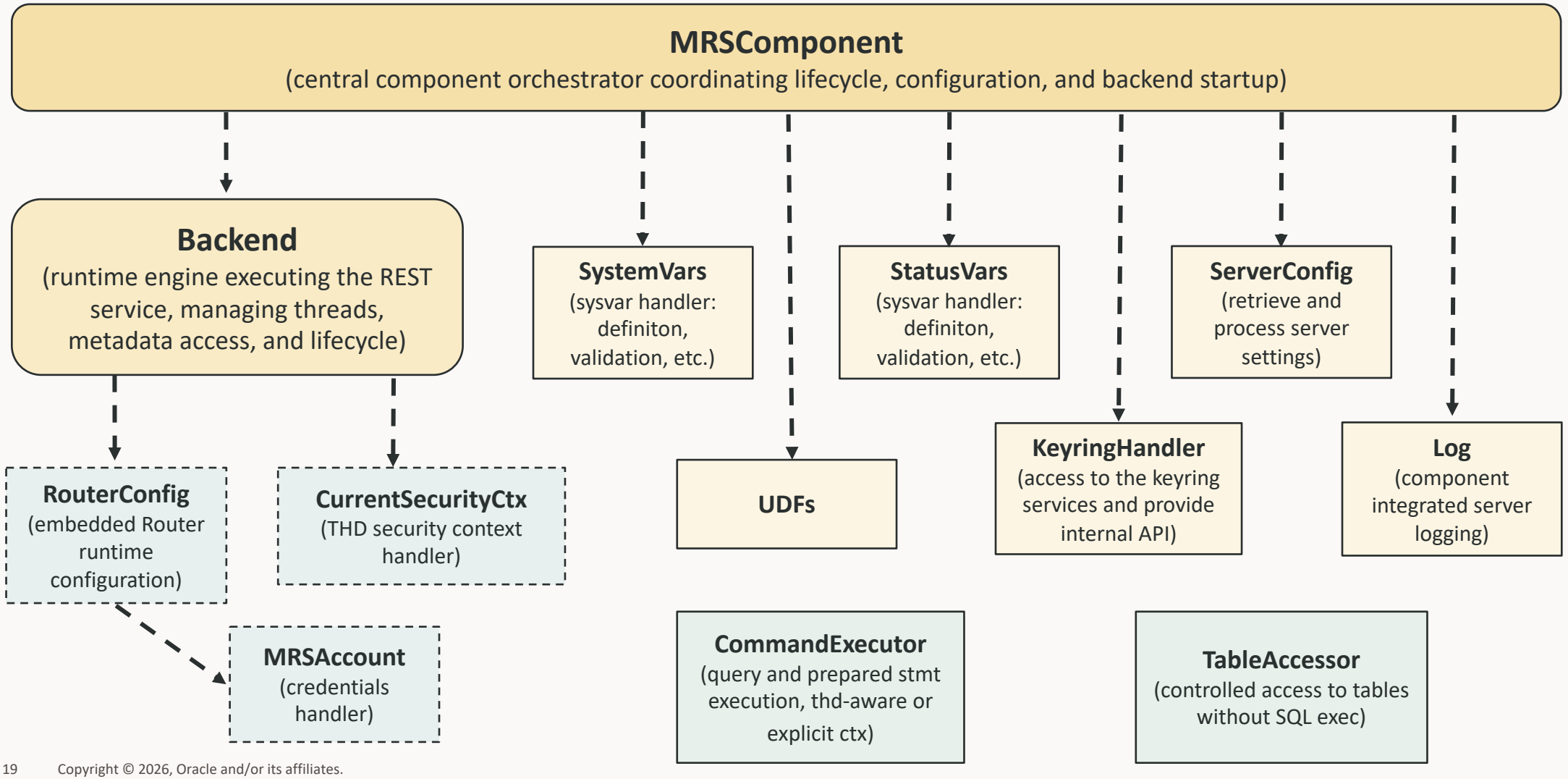
Hard parts
Logging, TLS, assumptions

Stabilization
Sysvars, accounts, bootstrap, statusvars...

Engineering milestones

Milestone	What this means in practice
Component Foundation	Create a new MySQL component with proper lifecycle
Reusing Router code	Adapt and embed Router HTTP, IO, MRS plugins inside `mysqld`
Logging integration	Replace Router logging with a component-compatible logging handler
Configuration model	Replace Router config files with MySQL sysvars and status variables
Security & accounts	Use MySQL's users, privileges, SSL/TLS, and Keyring. Automatically create and manage MRS accounts
Bootstrap	Initialize and maintain MRS metadata tied to server UUID
Runtime assumptions	Remove Router-specific assumptions about paths, users, and layout
Operational readiness	Clean startup/shutdown, packaging, and full MTR test coverage

MRS Component internal architecture



Building on MySQL Components Services

Internal area	MRS Component code	MySQL Components Services
Logging & observability	`Log, Backend`	`log_builtins, log_builtins_string`
Configuration	`SystemVars, ServerConfig`	`mysql_system_variable_reader, component_sys_variable_*`
Runtime status	`StatusVars`	`status_variable_registration, mysql_status_variable_string`
Cmd execution	`Command_executor`	`mysql_command_*, mysql_stmt_*`
Thread & sec context	`Backend, CurrentSecurityCtx`	`mysql_current_thread_reader, mysql_thd_security_context, mysql_security_context_options`
Table access	`Table_accessor`	`table_access_*, field_*`
UDFs	`UDFs`	`udf_registration, mysql_udf_metadata`
Credentials	`KeyringHandler, Backend`	`keyring_*`

The hard parts

Challenges, trade-offs, consequences

Challenges, trade-offs, consequences

Challenges we hit

- Component boundaries differ from plugins
- Threads inside components lose THD context
- `Init()` is intentionally limited (no SQL execution)
- Logging needed bridging
- Router assumptions don't hold inside mysqld
- Unifying TLS + secrets across server boundaries
- Metadata may not exist at startup

Design consequences

- Asynchronous backend initialization
- Custom query execution layer (THD-aware)
- Deferred init + readiness checks for metadata
- Server-native TLS & Keyring usage
- sysvars/UDFs for configuration & control (not files/CLI)
- Lifecycle-safe startup/shutdown + explicit error paths

Final state

What we achieved

What we achieved

Inside the Server

REST Service runs inside `mysqld`

MySQL acts as an HTTP Server

Component-Native

Built entirely on MySQL Components

Clean lifecycle & dependency model

Secure by default

MySQL users & privileges

Server-native TLS + Keyring

Operationally Ready

sysvars & status vars

Automatic bootstrap & metadata registration

MTR test coverage

Labs-ready

A Foundation

Not "just a feature"

Enables future server capabilities

Closing & Roadmap

What's next?

Closing & Roadmap

Next

- Instrumentation & observability
- Throttling / rate control
- TLS/SSL rotation
- Dev vs prod-mode refinements
- Metadata sys views
- Support across MySQL database architectures
- REST SQL available in the Server

Then

- Richer REST APIs
- Management & admin endpoints
- Server-side operational services
- REST as a first-class server interface

... foundation for future server capabilities

Community

- Invite feedback from users & contributors
- Validate real-world use cases and APIs
- Shape what comes next together

Try it early:
MySQL Labs - labs.mysql.com

Thank you!

Questions?