**Rust multi module microservices Part 2 — Workspace dependencies**

[[](https://medium.com/@omprakashsridharan?source=post_page-----bba64efb4bae--------------------------------)](https://medium.com/@omprakashsridharan?source=post_page-----bba64efb4bae--------------------------------)

[Omprakash Sridharan](https://medium.com/@omprakashsridharan?source=post_page-----bba64efb4bae--------------------------------)

·

Follow

3 min read

·

Jun 27

3

Before we delve into the sub-crate modules, let’s add a few crates that are common to the workspace and can be shared among all the crates. Update your Cargo.toml with the dependencies and it should look like the below.

[workspace]  
members = ["common", "books\_api", "books\_analytics", "database", "kafka"]  
  
[workspace.dependencies]  
tokio = { version = "1.28.2", features = ["full"] }  
tracing = "0.1.37"  
tracing-subscriber = { version = "0.3", features = [  
 "json",  
 "time",  
 "env-filter",  
]}  
serde = { version = "1.0.164", features = ["derive"]}  
serde\_json = "1.0"  
sea-orm = { version = "0.11.3", features = ["macros", "runtime-tokio-rustls", "sqlx-postgres"] }  
derive\_builder = "0.12.0"  
thiserror = "1.0.40"  
testcontainers = "0.14.0"  
strum = { version = "0.24.1", features = ["derive"] }  
axum = "0.6.18"  
opentelemetry = { version = "0.19.0", features = ["rt-tokio", "metrics"] }  
tracing-opentelemetry = "0.19.0"  
opentelemetry-zipkin = { version = "0.17.0", features = [  
 "reqwest-client",  
], default-features = false }  
axum-tracing-opentelemetry = "0.11.0"  
apache-avro= { version = "0.14", features=["derive"] }  
schema\_registry\_converter = { version = "3.1.0", features = ["avro","easy","kafka\_test"] }

Let’s go through the dependencies and what purpose they serve.

1. tokio: A popular asynchronous runtime for Rust that enables writing asynchronous code using the async/await syntax. It provides features for building scalable and efficient applications.
2. tracing: A framework for instrumenting Rust applications with structured, event-based diagnostics. It helps developers understand the behaviour of their code by providing flexible logging and debugging capabilities.
3. tracing-subscriber: A crate that provides a set of subscribers for integrating with the tracing framework. It allows customization of how traces are formatted, filtered, and dispatched to various outputs.
4. serde: A powerful serialization and deserialization library for Rust. It enables the conversion of Rust data structures to and from various formats, such as JSON, YAML, and more.
5. serde\_json: A JSON support crate for Serde. It provides functions to serialize and deserialize Rust data structures to and from JSON format.
6. sea-orm: An asynchronous, simple-to-use ORM (Object-Relational Mapping) for Rust that supports multiple database backends. It offers a high-level API for working with databases and simplifies common database operations.
7. derive\_builder: A procedural macro crate that provides a convenient way to generate builder patterns for Rust structs. It automates the creation of builder methods for setting struct fields.
8. thiserror: A crate for defining custom error types in Rust. It simplifies the process of defining and deriving error types with custom payloads.
9. testcontainers: A library that provides lightweight, disposable containers for running integration tests. It helps in setting up and managing isolated test environments for applications that rely on external dependencies, such as databases or message brokers.
10. strum: A set of macros and traits for working with enums in Rust. It provides convenient ways to iterate over enum variants, convert between enum variants and strings, and more.
11. axum: A minimalist web framework for Rust that is built on top of tokio and provides an expressive and ergonomic API for building HTTP applications.
12. opentelemetry: A vendor-agnostic observability framework for cloud-native applications. It provides APIs and components for distributed tracing, metrics, and other telemetry data.
13. tracing-opentelemetry: A bridge between the tracing and opentelemetry crates. It allows seamless integration of tracing instrumentation with the opentelemetry ecosystem.
14. opentelemetry-zipkin: A Zipkin exporter for the opentelemetry crate. It provides functionality to export telemetry data to a Zipkin backend for further analysis and visualization.
15. axum-tracing-opentelemetry: A crate that combines the functionalities of axum, tracing, and opentelemetry for building web applications with distributed tracing and observability support.
16. apache-avro: A crate that enables working with the Apache Avro data serialization format in Rust. It provides APIs for encoding, decoding, and manipulating Avro data.
17. schema\_registry\_converter: A crate that integrates with the Apache Kafka Schema Registry and provides Avro serialization and deserialization capabilities. It simplifies the process of working with Avro messages in Kafka-based applications.

With this base setup, let us start our journey of building the reusable modules in the next article 🚀