**Rust multi module microservices Part 1 — Introduction**

[[](https://medium.com/@omprakashsridharan?source=post_page-----fb0121f1d541--------------------------------)](https://medium.com/@omprakashsridharan?source=post_page-----fb0121f1d541--------------------------------)

[Omprakash Sridharan](https://medium.com/@omprakashsridharan?source=post_page-----fb0121f1d541--------------------------------)

·

Follow

2 min read

·

Jun 27

10

Hey folks, welcome to a mini-series of articles in which I continue my Rust learning journey, aiming to share knowledge and enjoyment throughout the process. In this series, I will focus on visualizing a multi-crate workspace project that allows for extensive code reuse across applications while maintaining the highest level of type safety. I plan to implement a variety of real-world use cases, such as shareable library crates, database migration, asynchronous communication using Kafka, distributed tracing with Zipkin, Avro, and much more 🤩

**Workspace Setup**

Create a directory for initializing the cargo root project. I’ll skip the part about how to install and update Rust and its tooling, as there are numerous articles and videos available on the subject. I have faith in your Googling skills! 😆

mkdir rust-superapp && cd rust-superapp

Let’s create a Cargo.toml file with the below contents

[workspace]  
 members = []  
  
 [workspace. Dependencies]  
 # We will add all our common dependent crates for our sub crates to share  
 # the same version. How cool is that :D

For this series, I have chosen a simple example of a Books HTTP REST API application crate, where you can create a book using a POST API. Additionally, there is a Books Analytics Kafka consumer application crate that prints the newly created book, which comes to this service through the power of Kafka 🚀. Let’s create our sub-packages.

Execute the following commands to generate the sub-crates, all within the root directory. (Run them one at a time, of course, 😌)

# Library crates  
 cargo new --vcs=none --lib common  
 cargo new --vcs=none --lib database  
 cargo new --vcs=none --lib kafka  
  
 # Application   
 cargo new --vcs=none books\_api  
 cargo new --vcs=none books\_analytics

This creates folders, but they remain fairly disconnected and independent. Recall the Cargo.toml file mentioned earlier, which included a workspace section with a members array? 🤓 Let’s add our crates there! After doing so, our Cargo.toml in the root directory will appear as follows.

**Expectation**

Now, we have a foundation to begin building our services. Your setup should resemble the following! (Please disregard the Easter eggs )

