AMBIENT-IT

+33 (0)1 48 10 09 46
formation@ambient-it.net
140 Avenue Jean Lolive 93500 Pantin - France

Updated on 27/06/2025

Sign up

Async Rust Training

3 days (21 hours)

Presentation

Our Async Rust training course will enable you to design highly concurrent, non-blocking, high-performance applications by taking advantage of Rust's unique asynchronous model. Through a comprehensive course, you'll learn how to write, structure and debug safe and efficient async code.

You'll learn in depth how Future works, how to use async/await, concurrency patterns (spawn, join, select) and how to manage asynchronous I/O (files, sockets, HTTP). Each notion is supported by practical workshops, enabling you to immediately apply the concepts studied.

The course will also guide you through the advanced use of Tokio: supervised tasks, cancellation, timeouts, asynchronous synchronization, and inter-task communication with mpsc or broadcast. You'll learn how to structure your asynchronous applications to avoid classic pitfalls such as deadlocks or infinite waits.

You will be introduced to testing and debugging async applications using the latest tools and dynamic instrumentation.

As with all our training courses, it is based on the latest developments in the Rust async ecosystem.

Objectives

- Understand how Future works in Rust and the asynchronous programming model without an integrated runtime
- Learn how to use async/await to structure non-blocking and concurrent functions.
- Master the use of Tokio and async-std runtimes to execute and orchestrate asynchronous tasks
- Implement efficient concurrency patterns with spawn, join!, select! and communication channels

- Integrate asynchronous I/O operations (file, network, HTTP, database) into a Rust application
- Manage errors, timeouts and cancellation in a multi-tasking async context
- Structure and test robust asynchronous Rust applications with tokio::test and tracing tools
- Design a secure, testable and high-performance full-async project, applying best practices of the language and its async ecosystem

Target audience

- Rust developers
- Software architects
- Back-end developers

Prerequisites

- Master the fundamentals of the Rust language

Async Rust training program

Introduction to asynchronism in Rust

- Differences between concurrency, parallelism and asynchronism
- Comparison with async models in Python, JavaScript and Go
- Futures: what is a future?
- The poll model and Pin, Waker, Context
- No GC, no integrated scheduler? design implications

Basic syntax: async / await

- Implicit return from Future<Output = T>
- No direct call: need .await
- Non-blocking: .await returns the hand to the runtime
- Possibility of waiting for several futures in sequence or in parallel
- Creation of temporary async blocks
- Nesting of blocks in a more complex logic

Use of runtimes

- Tokio architecture: scheduler, workers, event loop
- Use of #[tokio::main] and tokio::spawn
- Choice of compilation features
- API based on stdlib
- Less powerful than Tokio, but more readable for beginners
- Use cases for each runtime

Concurrent programming with async

- tokio::spawn, join!, select!
- Task hierarchy and supervision
- tokio::sync::mpsc, broadcast, oneshot
- Producer/consumer pattern
- Mutex and RwLock async
- Semaphores and barriers

Asynchronous I/O management

- tokio::fs, tokio::net
- Use of reqwest and hyper
- Async/await pattern with HTTP client
- sqlx, sea-orm, surrealdb
- Connections, transactions, asynchronous pooling

Advanced patterns and best practices

- API batch, timeouts, cancellation
- tokio::time::timeout, Abortable
- Managing long or blocked tasks
- Async deadlocks
- .await in locks
- Send and Sync in futures

Debugging and testing async applications

- Limitations of dbg! and println! in concurrent tasks
- tokio-console, tracing, instrument
- #[tokio::test] and #[async_std::test]
- Mocking of asynchronous services
- Coverage, profiling, benchmarks

Case studies and threaded projects

- Asynchronous routing
- Managing parallel requests
- Async processing with timeouts
- Event processing with backpressure
- Retry and error handling
- Simple backend
- SQLx + Axum/Tonic + Tokio

Going further

- Implementing your own Future
- When is it useful?
- Why async fn in traits is limited
- Using async-trait or dyn Trait with Box<dyn Future>
- Async with WASM
- WebAssembly network calls via futures

Companies concerned

This course is aimed at both individuals and companies, large or small, wishing to train their teams in a new, advanced computer technology, or to acquire specific business knowledge or modern methods.

Positioning on entry to training

Positioning at the start of training complies with Qualiopi quality criteria. As soon as registration is finalized, the learner receives a self-assessment questionnaire which enables us to assess his or her estimated level of proficiency in different types of technology, as well as his or her expectations and personal objectives for the training to come, within the limits imposed by the selected format. This questionnaire also enables us to anticipate any connection or security difficulties within the company (intra-company or virtual classroom) which could be problematic for the follow-up and smooth running of the training session.

Teaching methods

Practical training: 60% hands-on, 40% theory. Training material distributed in digital format to all participants.

Organization

The course alternates theoretical input from the trainer, supported by examples, with brainstorming sessions and group work.

Validation

At the end of the session, a multiple-choice questionnaire verifies the correct acquisition of skills.

Certification

A certificate will be awarded to each trainee who has completed the entire course.