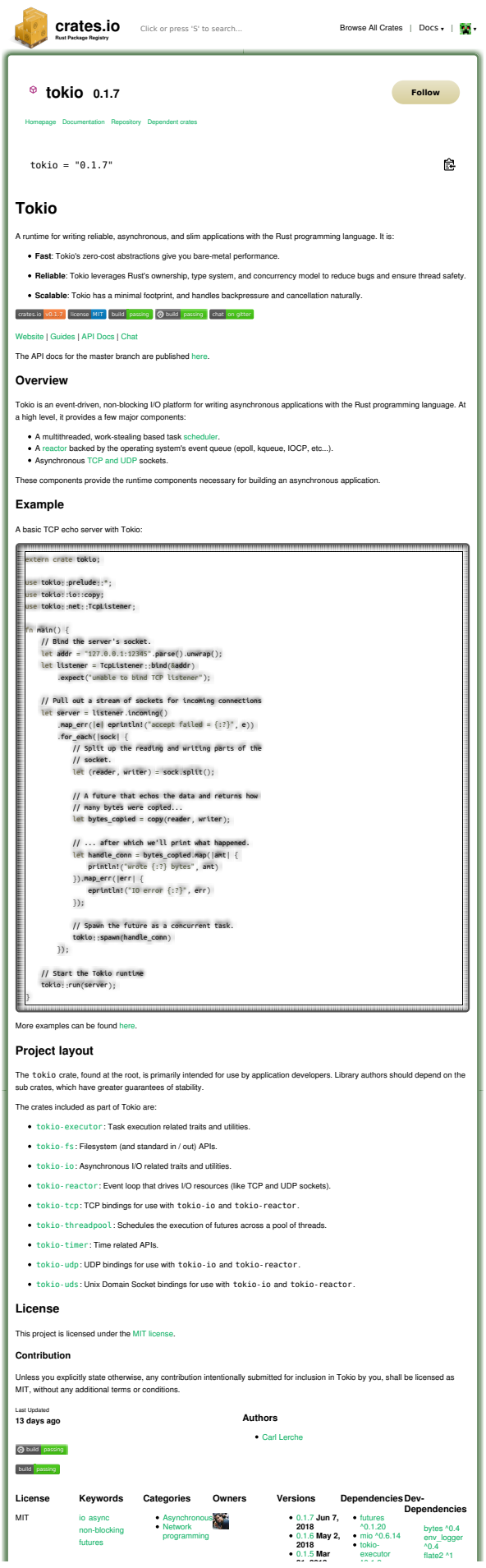

crates.io
Rust Package Registry
Click or press 'S' to search...
Browse All Crates | Docs | Fork me on GitHub

tokio 0.1.7
Follow

Homepage Documentation Repository Dependent crates

```
tokio = "0.1.7"
```

Tokio

A runtime for writing reliable, asynchronous, and slim applications with the Rust programming language. It is:

- **Fast**: Tokio's zero-cost abstractions give you bare-metal performance.
- **Reliable**: Tokio leverages Rust's ownership, type system, and concurrency model to reduce bugs and ensure thread safety.
- **Scalable**: Tokio has a minimal footprint, and handles backpressure and cancellation naturally.

crates.io v0.1.7 | license MIT | build passing | build passing | chat on gitter

Website | Guides | API Docs | Chat

The API docs for the master branch are published here.

Overview

Tokio is an event-driven, non-blocking I/O platform for writing asynchronous applications with the Rust programming language. At a high level, it provides a few major components:

- A multithreaded, work-stealing based task scheduler.
- A reactor backed by the operating system's event queue (epoll, kqueue, IOCP, etc...).
- Asynchronous TCP and UDP sockets.

These components provide the runtime components necessary for building an asynchronous application.

Example

A basic TCP echo server with Tokio:

```
extern crate tokio;

use tokio::prelude::*;
use tokio::io::copy;
use tokio::net::TcpListener;

fn main() {
    // Bind the server's socket.
    let addr = "127.0.0.1:12345".parse().unwrap();
    let listener = TcpListener::bind(&addr)
        .expect("unable to bind TCP listener");

    // Pull out a stream of sockets for incoming connections
    let server = listener.incoming()
        .map_err(|e| eprintln!("accept failed = {:?}", e))
        .for_each(|sock| {
            // Split up the reading and writing parts of the
            // socket.
            let (reader, writer) = sock.split();

            // A future that echos the data and returns how
            // many bytes were copied...
            let bytes_copied = copy(reader, writer);

            // ... after which we'll print what happened.
            let handle_conn = bytes_copied.map(|amt| {
                println!("wrote {:?} bytes", amt)
            }).map_err(|err| {
                eprintln!("IO error {:?}", err)
            });

            // Spawn the future as a concurrent task.
            tokio::spawn(handle_conn)
        });

    // Start the Tokio runtime
    tokio::run(server);
}
```

More examples can be found here.

Project layout

The `tokio` crate, found at the root, is primarily intended for use by application developers. Library authors should depend on the sub crates, which have greater guarantees of stability.

The crates included as part of Tokio are:

- `tokio-executor`: Task execution related traits and utilities.
- `tokio-fs`: Filesystem (and standard in / out) APIs.
- `tokio-io`: Asynchronous I/O related traits and utilities.
- `tokio-reactor`: Event loop that drives I/O resources (like TCP and UDP sockets).
- `tokio-tcp`: TCP bindings for use with `tokio-io` and `tokio-reactor`.
- `tokio-threadpool`: Schedules the execution of futures across a pool of threads.
- `tokio-timer`: Time related APIs.
- `tokio-udp`: UDP bindings for use with `tokio-io` and `tokio-reactor`.
- `tokio-uds`: Unix Domain Socket bindings for use with `tokio-io` and `tokio-reactor`.

License

This project is licensed under the MIT license.

Contribution

Unless you explicitly state otherwise, any contribution intentionally submitted for inclusion in Tokio by you, shall be licensed as MIT, without any additional terms or conditions.

Last Updated
13 days ago

Authors
- Carl Lerche

build passing
build passing

License	Keywords	Categories	Owners	Versions	Dependencies	Dev-Dependencies
MIT	io async non-blocking futures	• Asynchronous • Network programming		• 0.1.7 **Jun 7, 2018** • 0.1.6 **May 2, 2018** • 0.1.5 **Mar**	• futures ^0.1.20 • mio ^0.6.14 • tokio-executor	bytes ^0.4 env_logger ^0.4 flate2 ^1