
Substrate Node Template

A fresh Substrate node, ready for hacking :rocket:

A standalone version of this template is available for each release of Polkadot in the Substrate Developer Hub Parachain Template repository. The parachain template is generated directly at each Polkadot release branch from the Node Template in Substrate upstream

It is usually best to use the stand-alone version to start a new project. All bugs, suggestions, and feature requests should be made upstream in the Substrate repository.

Getting Started

Depending on your operating system and Rust version, there might be additional packages required to compile this template. Check the Install instructions for your platform for the most common dependencies. Alternatively, you can use one of the alternative installation options.

Build

Use the following command to build the node without launching it:

```
1 cargo build --release
```

Embedded Docs

After you build the project, you can use the following command to explore its parameters and sub-commands:

```
1 ./target/release/node-template -h
```

You can generate and view the Rust Docs for this template with this command:

```
1 cargo +nightly doc --open
```

Single-Node Development Chain

The following command starts a single-node development chain that doesn't persist state:

```
1 ./target/release/node-template --dev
```

To purge the development chain's state, run the following command:

```
1 ./target/release/node-template purge-chain --dev
```

To start the development chain with detailed logging, run the following command:

```
1 RUST_BACKTRACE=1 ./target/release/node-template -ldebug --dev
```

Development chains:

- Maintain state in a `tmp` folder while the node is running.
- Use the **Alice** and **Bob** accounts as default validator authorities.
- Use the **Alice** account as the default `sudo` account.
- Are preconfigured with a genesis state (`/node/src/chain_spec.rs`) that includes several prefunded development accounts.

To persist chain state between runs, specify a base path by running a command similar to the following:

```
1 // Create a folder to use as the db base path
2 $ mkdir my-chain-state
3
4 // Use of that folder to store the chain state
5 $ ./target/release/node-template --dev --base-path ./my-chain-state/
6
7 // Check the folder structure created inside the base path after
  running the chain
8 $ ls ./my-chain-state
9 chains
10 $ ls ./my-chain-state/chains/
11 dev
12 $ ls ./my-chain-state/chains/dev
13 db keystore network
```

Connect with Polkadot-JS Apps Front-End

After you start the node template locally, you can interact with it using the hosted version of the Polkadot/Substrate Portal front-end by connecting to the local node endpoint. A hosted version is also available on IPFS (redirect) [here](#) or IPNS (direct) [here](#). You can also find the source code and instructions for hosting your own instance on the [polkadot-js/apps](#) repository.

Multi-Node Local Testnet

If you want to see the multi-node consensus algorithm in action, see [Simulate a network](#).

Template Structure

A Substrate project such as this consists of a number of components that are spread across a few directories.

Node

A blockchain node is an application that allows users to participate in a blockchain network. Substrate-based blockchain nodes expose a number of capabilities:

- **Networking:** Substrate nodes use the [libp2p](#) networking stack to allow the nodes in the network to communicate with one another.
- **Consensus:** Blockchains must have a way to come to consensus on the state of the network. Substrate makes it possible to supply custom consensus engines and also ships with several consensus mechanisms that have been built on top of Web3 Foundation research.
- **RPC Server:** A remote procedure call (RPC) server is used to interact with Substrate nodes.

There are several files in the [node](#) directory. Take special note of the following:

- [chain_spec.rs](#): A chain specification is a source code file that defines a Substrate chain's initial (genesis) state. Chain specifications are useful for development and testing, and critical when architecting the launch of a production chain. Take note of the [development_config](#) and [testnet_genesis](#) functions,. These functions are used to define the genesis state for the local development chain configuration. These functions identify some well-known accounts and use them to configure the blockchain's initial state.
- [service.rs](#): This file defines the node implementation. Take note of the libraries that this file imports and the names of the functions it invokes. In particular, there are references to consensus-related topics, such as the block finalization and forks and other consensus mechanisms such as Aura for block authoring and GRANDPA for finality.

Runtime

In Substrate, the terms “runtime” and “state transition function” are analogous. Both terms refer to the core logic of the blockchain that is responsible for validating blocks and executing the state changes they define. The Substrate project in this repository uses FRAME to construct a blockchain runtime. FRAME allows runtime developers to declare domain-specific logic in modules called “pallets”. At the heart of FRAME is a helpful macro language that makes it easy to create pallets and flexibly compose them to create blockchains that can address a variety of needs.

Review the FRAME runtime implementation included in this template and note the following:

-
- This file configures several pallets to include in the runtime. Each pallet configuration is defined by a code block that begins with `impl $PALLET_NAME::Config for Runtime`.
 - The pallets are composed into a single runtime by way of the `construct_runtime!` macro, which is part of the core FRAME pallet library.

Pallets

The runtime in this project is constructed using many FRAME pallets that ship with the Substrate repository and a template pallet that is defined in the `pallets` directory.

A FRAME pallet is comprised of a number of blockchain primitives, including:

- Storage: FRAME defines a rich set of powerful storage abstractions that makes it easy to use Substrate's efficient key-value database to manage the evolving state of a blockchain.
- Dispatchables: FRAME pallets define special types of functions that can be invoked (dispatched) from outside of the runtime in order to update its state.
- Events: Substrate uses events to notify users of significant state changes.
- Errors: When a dispatchable fails, it returns an error.

Each pallet has its own `Config` trait which serves as a configuration interface to generically define the types and parameters it depends on.

Alternatives Installations

Instead of installing dependencies and building this source directly, consider the following alternatives.

Nix

Install nix and nix-direnv for a fully plug-and-play experience for setting up the development environment. To get all the correct dependencies, activate direnv `direnv allow`.

Docker

Please follow the Substrate Docker instructions here to build the Docker container with the Substrate Node Template binary.