Substrate Kitties - Basic Setup

Learn how to use the Substrate Node template

Polkadot Intermediate Rust 1 hour

Written by Kresna Sucandra

Show Table of Contents

Part I: Basic set-up

> This tutorial assumes that you have already installed the prerequisites for building with Substrate on your machine. If you haven't already, head over to our installation guide.

Overview

Before we can start making Kitties, we first need to do a little groundwork. This part covers the basic patterns involved with using the Substrate Node Template to set up a custom pallet and include a simple storage item.

Learning outcomes

- Renaming a Substrate Node Template using the Kickstart tool.
- Basic patterns for building and running a Substrate node.
- Create a storage item to keep track of a single `u64` value.

Steps

Set-up your template node

The Substrate Node Template provides us with an "out-of-the-box" blockchain node. Our biggest advantage in using it are that both networking and consensus layers are already built and all we need to focus on is building out our runtime and pallet logic. Before we get there, we need to set-up our project in terms of naming and dependencies.

We'll use a CLI tool called kickstart to easily rename our node template.

In the root directory of your local workspace, run the following command:

```
kickstart https://github.com/sacha-l/kickstart-substrate
```

This command will clone a copy of the most recent Node Template and ask you would like to call your node and pallet. Type in:

- `kitties` - as the name of our node
- `mykitties` - as the name of your pallet

This will create a directory called `kitties` with a copy of the Substrate Node Template containing the name changes that correspond our template node, runtime and pallet.

Open the `kitties` directory in your favorite code editor and rename it to `kitties-tutorial`. Renaming this directory will be helpful once you start creating other projects with the node template — it'll help keep things organized!

Notice the directories that the `kickstart` command modified:

- `/node/` - This contains all the logic that allows your node to interact with your runtime and RPC clients.
- `/pallets/` - Here's where all your custom pallets live.
- `/runtime/` - This is where all pallets (both custom "internal" and "external" ones) are aggregated and implemented for the chain's runtime.

By default, you'll notice that the instance of our modified template pallet name remains `TemplateModule`. Change it to `SubstrateKitties` (in `runtime/src/lib.rs`):

```
construct_runtime!(
    pub enum Runtime where
    Block = Block,
    NodeBlock = opaque::Block,
    UncheckedExtrinsic = UncheckedExtrinsic
    {
        // --snip
```

```
        SubstrateKitties: pallet_mykitties::{Pallet, Call, Storage, Event<T>},
    }
);
```

We can already build the node as is by navigating to directory `kitties` in terminal and running this command:

```
cargo +nightly build --release
```

It's normal if this command takes a little while depending on your machine — it's building a whole bunch of crates from the Substrate crates and libraries. The nice thing is that once we run this command the first time, it won't need to rebuild all the crates when we build subsequent times.

Assuming that your node builds successfully, launch it in development mode to make sure it works:

```
./target/release/node-kitties --tmp --dev
```

You should see blocks being created in your terminal. The `--tmp` and `--dev` flags mean we're running a temporary node in development mode.

Write out pallet_kitties scaffold

We'll be spending most of this tutorial in the `pallets` directory of our template node. Let's take a glance at the folder structure in our workspace:

```
kitties-tutorial           <--  The name of our project directory
|
+-- node
|
+-- pallets
|   |
|   +-- mykitties
|       |
|       +-- Cargo.toml
|       |
|       +-- src
|           |
|           +-- lib.rs     <-- Remove contents
|           |
|           +-- mock.rs    <-- Remove file
```

```
|           |
|           +-- tests.rs   <-- Remove file
|
+-- Cargo.toml
```

> You can go ahead and remove `mock.rs` and `tests.rs`. **We won't be learning about using these in this tutorial.** Have a look at this how-to guide if you're curious to learn how testing works.

Pallets in Substrate are used to define runtime logic. In our case, we'll be creating a single pallet that manages all of the logic of our Substrate Kitties application.

Let's lay out the basic structure of our pallet by outlining the parts inside the `pallets/mykitties/src/lib.rs`.

> Our pallet's directory `pallets/mykitties/` is not the same as our pallet's name. The name of our pallet as Cargo understands it is `pallet-mykitties`.

Every FRAME pallet has:

- A set of `frame_support` and `frame_system` dependencies.
- Required attribute macros (i.e. configuration traits, storage items and function calls).

> We'll be updating additional dependencies as we progress through the next parts of this tutorial.

Here's the most bare-bones version of the Kitties pallet we will be building in this tutorial. It contains the starting point for adding code for the next sections of this tutorial, with comments marked with "TODO" to indicate code we will be writing later, and "ACTION" to indicate code that will be written in the current part of the tutorial.

Paste the following code in `/pallets/mykitties/src/lib.rs`:

```
#![cfg_attr(not(feature = "std"), no_std)]

pub use pallet::*;

#[frame_support::pallet]
```

```rust
pub mod pallet {
    use frame_support::{sp_runtime::traits::{Hash, Zero},
                        dispatch::{DispatchResultWithPostInfo, DispatchResult}
                        traits::{Currency, ExistenceRequirement, Randomness},
                        pallet_prelude::*};
    use frame_system::pallet_prelude::*;
    use sp_core::H256;

    // TODO Part II: Struct for holding Kitty information.

    // TODO Part II: Enum and implementation to handle Gender type in Kitty st

    #[pallet::pallet]
    #[pallet::generate_store(trait Store)]
    pub struct Pallet<T>(_);

    /// Configure the pallet by specifying the parameters and types it depends
    #[pallet::config]
    pub trait Config: pallet_balances::Config + frame_system::Config {
        /// Because this pallet emits events, it depends on the runtime's defi
        type Event: From<Event<Self>> + IsType<<Self as frame_system::Config>:

        // TODO Part II: The type of Random we want to specify for runtime.
    }

    // Errors.
    #[pallet::error]
    pub enum Error<T> {
        // TODO Part III
    }

    #[pallet::event]
    #[pallet::metadata(T::AccountId = "AccountId")]
    #[pallet::generate_deposit(pub(super) fn deposit_event)]
    pub enum Event<T: Config> {
        // TODO Part III
    }

    // ACTION: Storage item to keep track of all Kitties.

    // TODO Part II: Remaining storage items.

    // TODO Part III: Our pallet's genesis configuration.

    #[pallet::call]
    impl<T: Config> Pallet<T> {

        // TODO Part III: create_kitty
```

```
        // TODO Part III: set_price

        // TODO Part III: transfer

        // TODO Part III: buy_kitty

        // TODO Part III: breed_kitty
    }

    // TODO Parts II: helper function for Kitty struct

    impl<T: Config> Pallet<T> {
        // TODO Part III: helper functions for dispatchable functions

        // TODO: increment_nonce, random_hash, mint, transfer_from

    }
}
```

Your turn! Copy over the bare-bones of the Kitties pallet into `mykitties/src/lib.rs`.

Now try running the following command to rebuild your chain:

```
cargo +nightly build --release
```

Get an error about dependencies? That's normal! Our pallet structure is using FRAME's `pallet-balances` and `sp-io` but these aren't part of the node template we used so we must specify them ourselves. In `pallets/mykitties/Cargo.toml`, add the following:

```
[dependencies.sp-core]
default-features = false
git = 'https://github.com/paritytech/substrate.git'
tag = 'monthly-2021-08'
version = '4.0.0-dev'

[dependencies.pallet-balances]
default-features = false
git = 'https://github.com/paritytech/substrate.git'
tag = 'monthly-2021-08'
version = '4.0.0-dev'
```

WARNING! Check that you're using the correct `monthly-*` tag and `version` otherwise you will get a dependency error. Here, we're using the most up-to-date tag as of the writing of this tutorial.

Now run `cargo +nightly build --release` again to make sure it builds without errors.

You'll notice the Rust compiler giving you warnings about unused imports. That's fine! Just ignore them — we're going to be using those imports in the later parts of the tutorial.

In the next step we will include the first storage item our Kitty application will require.

Include a storage item to track all Kitties

Let's start by adding the most simple logic we can to our runtime: a function which stores a variable in runtime.

To do this we'll use `StorageValue` from Substrate's storage API which is a trait that depends on the storage macro.

All that means for our purposes is that for any storage item we want to declare, we must include the `#[pallet::storage]` macro beforehand. Learn more about declaring storage items click here.

In `mykitties/src/lib.rs`, replace the ACTION line with:

```
#[pallet::storage]
#[pallet::getter(fn all_kitties_count)]
pub(super) type AllKittiesCount<T: Config> = StorageValue<
    _,
    u64,
    ValueQuery
    >;
```

This creates a storage item for our pallet to keep track of a counter that will correspond to the total amount of Kitties in existence.

Build pallet

From the previous step, your pallet should contain a storage item called `AllKittiesCount` which keeps track of a single `u64` value. As part of the basic setup, we're doing great!

As mentioned in the overview of this tutorial series, you'll be implementing a total of 9 storage items which you'll discover as you write out your pallet's logic in the next parts.

Before we move on, let's make sure everything compiles. We don't need to rebuild our entire node each time we update our pallet. Instead, we can use a command that only builds our pallet. From inside your pallet directory, run the following:

```
cargo build -p pallet-mykitties
```

Does your pallet compile without error? Well done if it does! If not, go back and check that all the macros are in place and that you've included the FRAME dependencies.

Congratulations! You've completed the first part of this series. At this stage, you've learnt the various patterns for:

- Customizing the Substrate node template and including a custom pallet.
- Building a Substrate chain and checking that a target pallet compiles.
- Declaring a single value `u64` storage item.

Next steps

- Writing a struct in a `StorageMap` to store details about our Kitties
- Using the Randomness trait to create unique Kitties
- Creating our pallet's remaining storage items

Click here to go to the next part of this tutorial series.