Lesson 6

Hardhat Simple Storage

Hikmah Nisya - 1103184094
Radzis Araaf Jaya Jamaludin - 1103184234
Raudhatul Rafiqah Assyahiddini - 1103180225

Hardhat

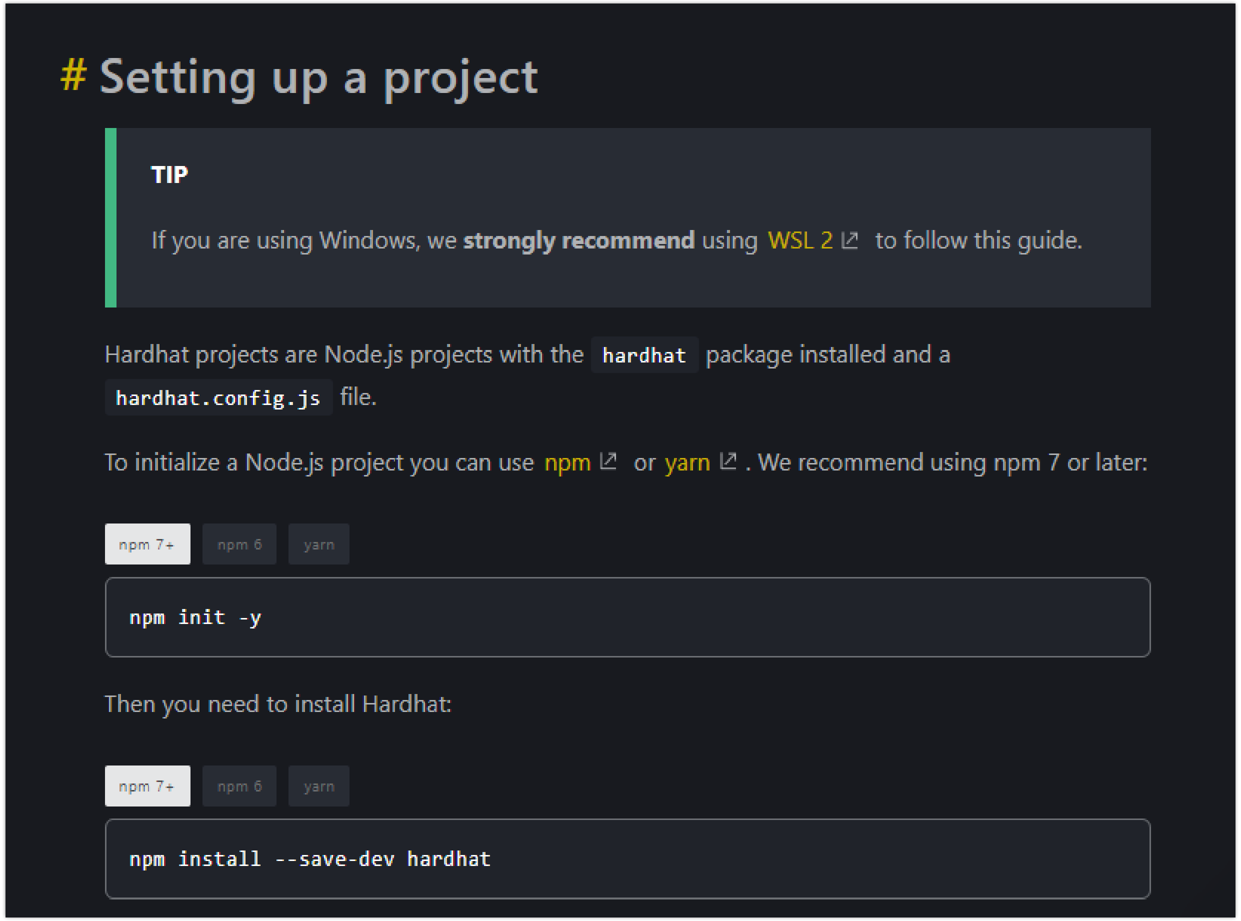
- Hardhat is a development environment for Ethereum software. It consists of different components for editing, compiling, debugging and deploying your smart contracts and dApps, all of which work together to create a complete development environment.

Installing

Hardhat is used through a local installation in your project. This way your environment will be reproducible, and you will avoid future version conflicts.

To install it, you need to create an npm project by going to an empty folder, running npm init, and following its instructions. You can use another package manager, like yarn, but we recommend you use npm 7 or later, as it makes installing Hardhat plugins simpler.

SETTING PROJECT HARDHAT



Setting up a project

TIP

If you are using Windows, we **strongly recommend** using WSL 2 to follow this guide.

Hardhat projects are Node.js projects with the `hardhat` package installed and a `hardhat.config.js` file.

To initialize a Node.js project you can use npm or yarn. We recommend using npm 7 or later:

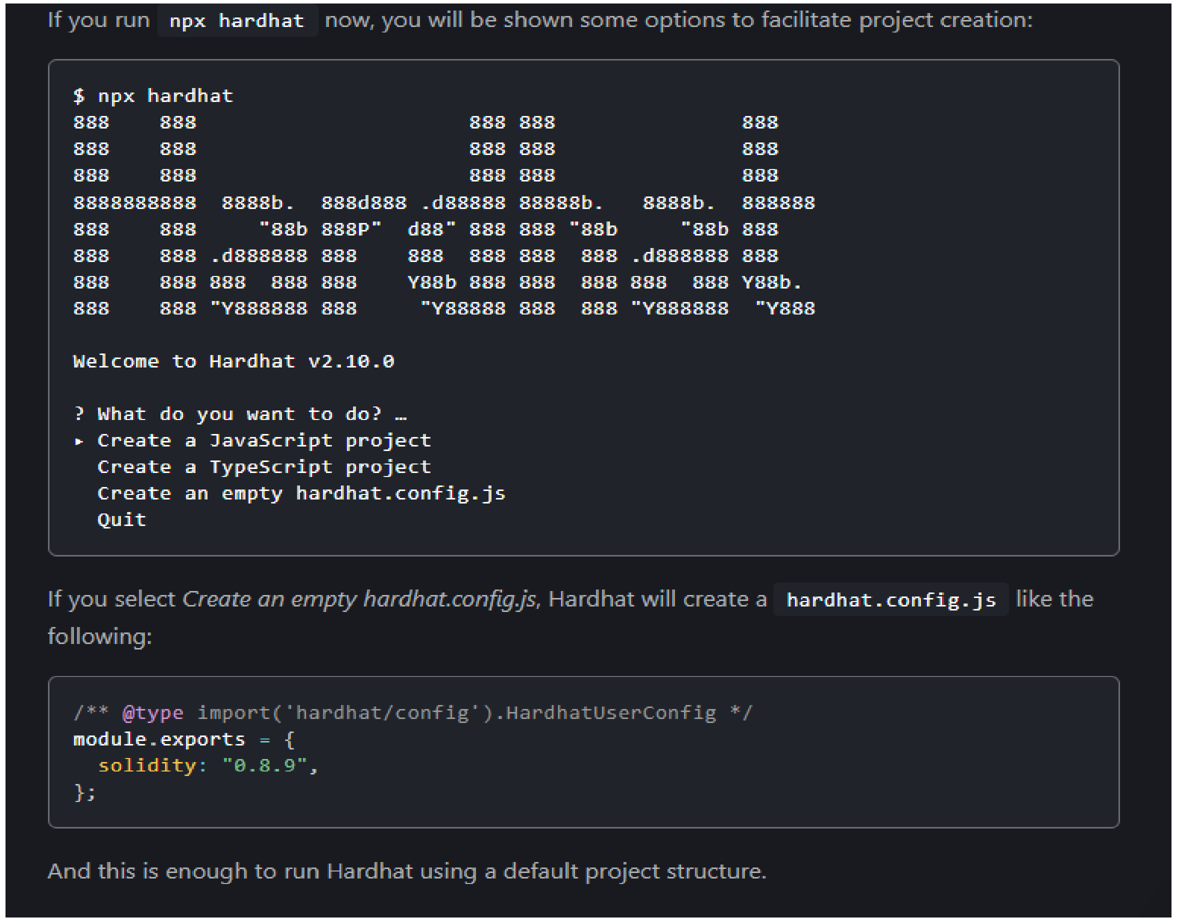
npm 7+ npm 6 yarn

```
npm init -y
```

Then you need to install Hardhat:

npm 7+ npm 6 yarn

```
npm install --save-dev hardhat
```



If you run `npx hardhat` now, you will be shown some options to facilitate project creation:

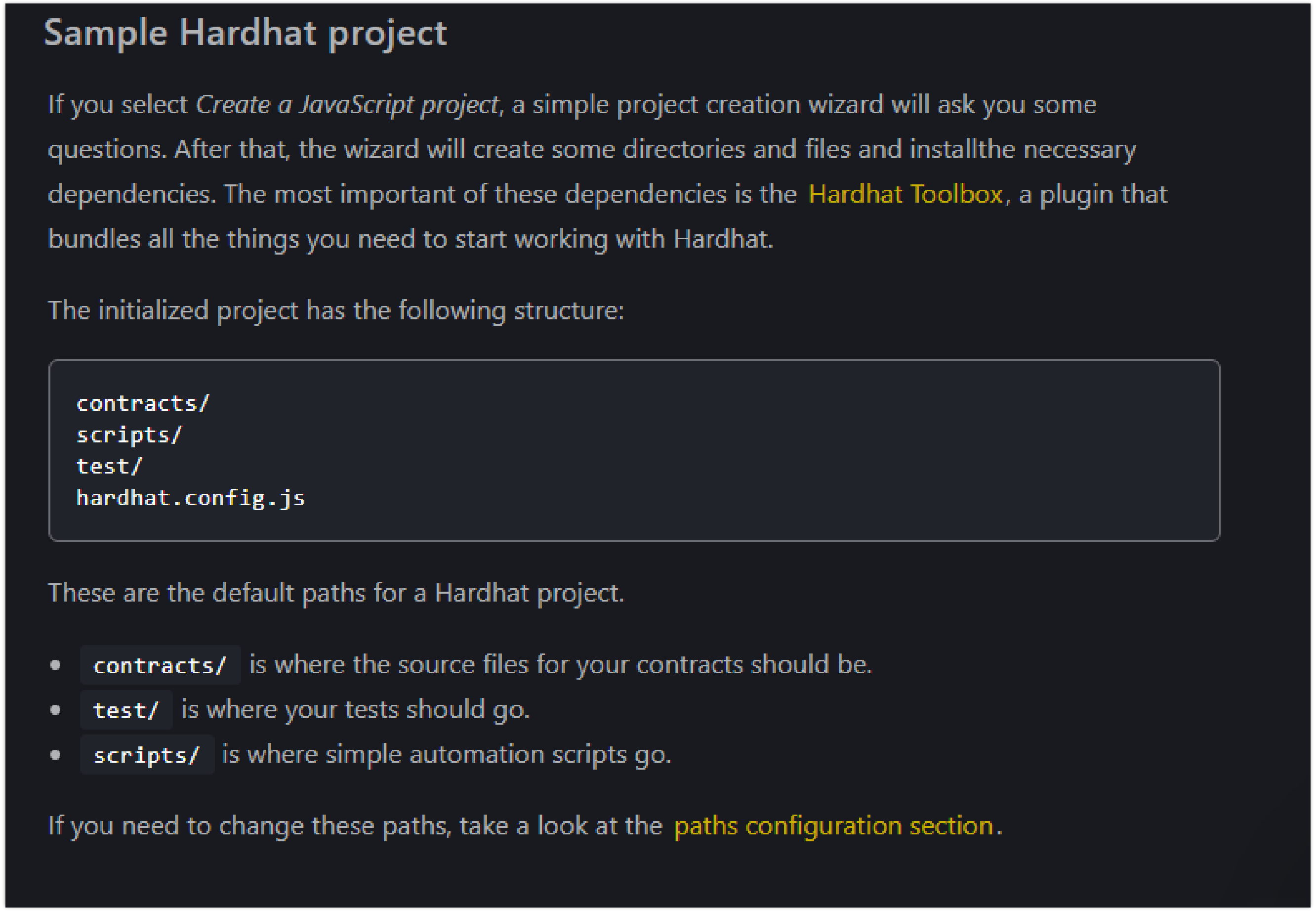
```
$ npx hardhat
888    888                      888 888               888
888    888                      888 888               888
888    888                      888 888               888
8888888888  8888b.  888d888 .d88888 88888b.   8888b.  888888
888    888     "88b 888P"  d88" 888 888 "88b     "88b 888
888    888 .d888888 888    888  888 888  888 .d888888 888
888    888 888  888 888    Y88b 888 888  888 888  888 Y88b.
888    888 "Y888888 888     "Y88888 888  888 "Y888888  "Y888

Welcome to Hardhat v2.10.0

? What do you want to do? …
▸ Create a JavaScript project
  Create a TypeScript project
  Create an empty hardhat.config.js
  Quit
```

If you select *Create an empty hardhat.config.js*, Hardhat will create a `hardhat.config.js` like the following:

```
/** @type import('hardhat/config').HardhatUserConfig */
module.exports = {
  solidity: "0.8.9",
};
```

And this is enough to run Hardhat using a default project structure.

SIMPLE HARDHAT PROJECT



Sample Hardhat project

If you select *Create a JavaScript project*, a simple project creation wizard will ask you some questions. After that, the wizard will create some directories and files and installthe necessary dependencies. The most important of these dependencies is the Hardhat Toolbox, a plugin that bundles all the things you need to start working with Hardhat.

The initialized project has the following structure:

```
contracts/
scripts/
test/
hardhat.config.js
```

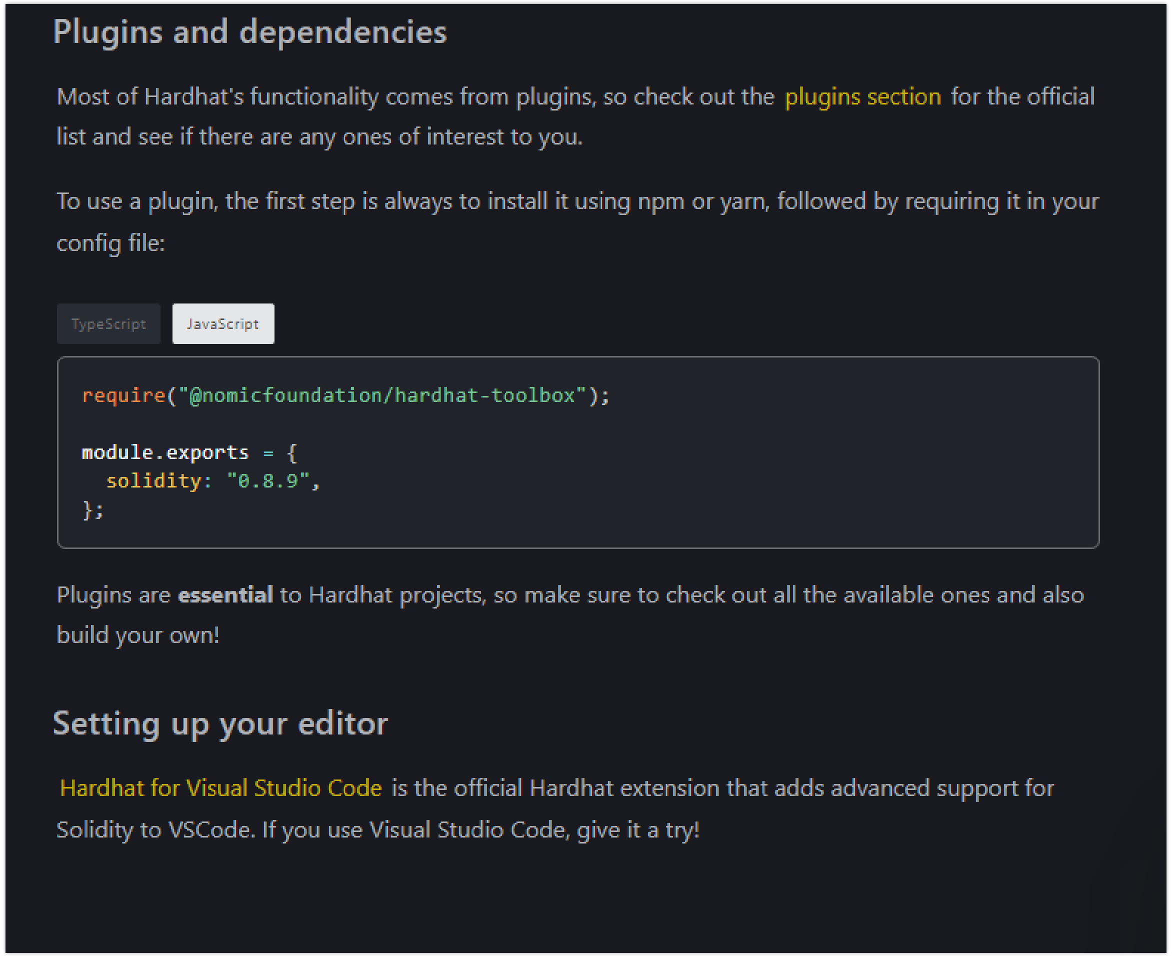
These are the default paths for a Hardhat project.

- `contracts/` is where the source files for your contracts should be.
- `test/` is where your tests should go.
- `scripts/` is where simple automation scripts go.

If you need to change these paths, take a look at the paths configuration section.

- #Testing
- When it comes to testing your contracts, the sample project comes with some useful functionality:
- The built-in Hardhat Network as the development network to test on, along with the Hardhat Network Helpers library to manipulate this network.
- Mocha as the test runner, Chai as the assertion library, and the Hardhat Chai Matchers to extend Chai with contracts-related functionality.
- The ethers.js library to interact with the network and with contracts.
- As well as other useful plugins. You can learn more about this in the Testing contracts guide.

- #External networks
- If you need to use an external network, like an Ethereum testnet, mainnet or some other specific node software, you can set it up using the networks configuration entries in the exported object in hardhat.config.js, which is how Hardhat projects manage settings.
- You can use of the --network CLI parameter to quickly change the network.
- Take a look at the networks configuration section to learn more about setting up different networks.


Plugins and dependencies

Most of Hardhat's functionality comes from plugins, so check out the plugins section for the official list and see if there are any ones of interest to you.

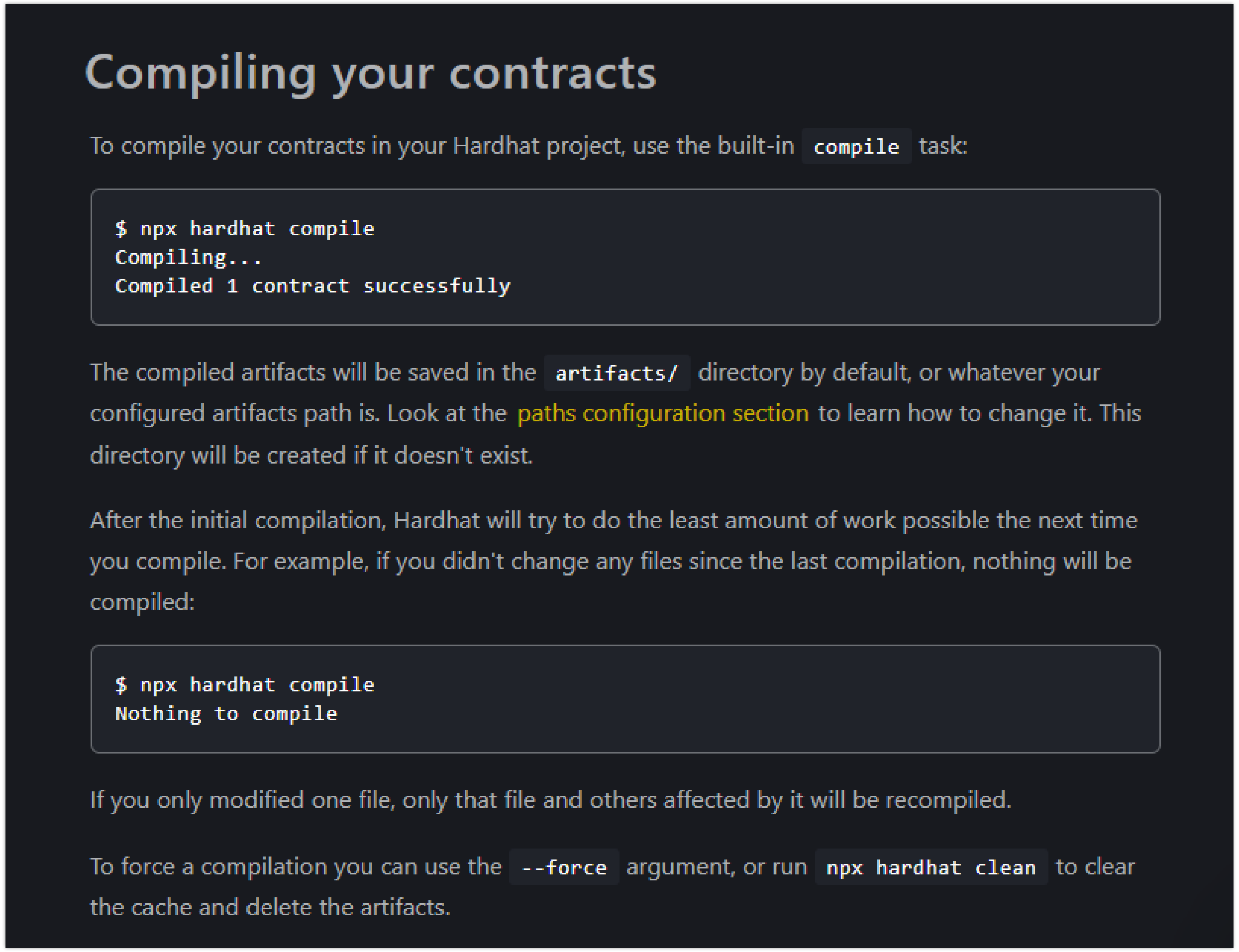
To use a plugin, the first step is always to install it using npm or yarn, followed by requiring it in your config file:

```
require("@nomicfoundation/hardhat-toolbox");

module.exports = {
  solidity: "0.8.9",
};
```

Plugins are **essential** to Hardhat projects, so make sure to check out all the available ones and also build your own!

Setting up your editor

Hardhat for Visual Studio Code is the official Hardhat extension that adds advanced support for Solidity to VSCode. If you use Visual Studio Code, give it a try!


Compiling your contracts

To compile your contracts in your Hardhat project, use the built-in `compile` task:

```
$ npx hardhat compile
Compiling...
Compiled 1 contract successfully
```

The compiled artifacts will be saved in the `artifacts/` directory by default, or whatever your configured artifacts path is. Look at the paths configuration section to learn how to change it. This directory will be created if it doesn't exist.

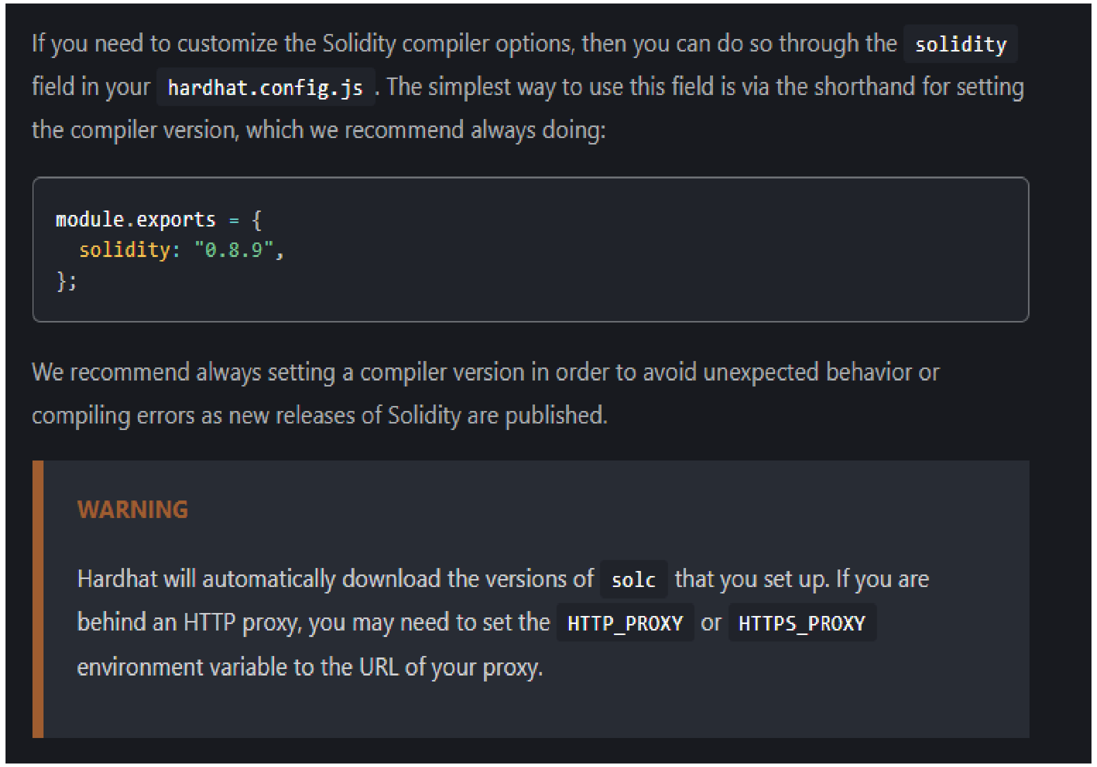
After the initial compilation, Hardhat will try to do the least amount of work possible the next time you compile. For example, if you didn't change any files since the last compilation, nothing will be compiled:

```
$ npx hardhat compile
Nothing to compile
```

If you only modified one file, only that file and others affected by it will be recompiled.

To force a compilation you can use the `--force` argument, or run `npx hardhat clean` to clear the cache and delete the artifacts.

CONFIGURING COMPILER



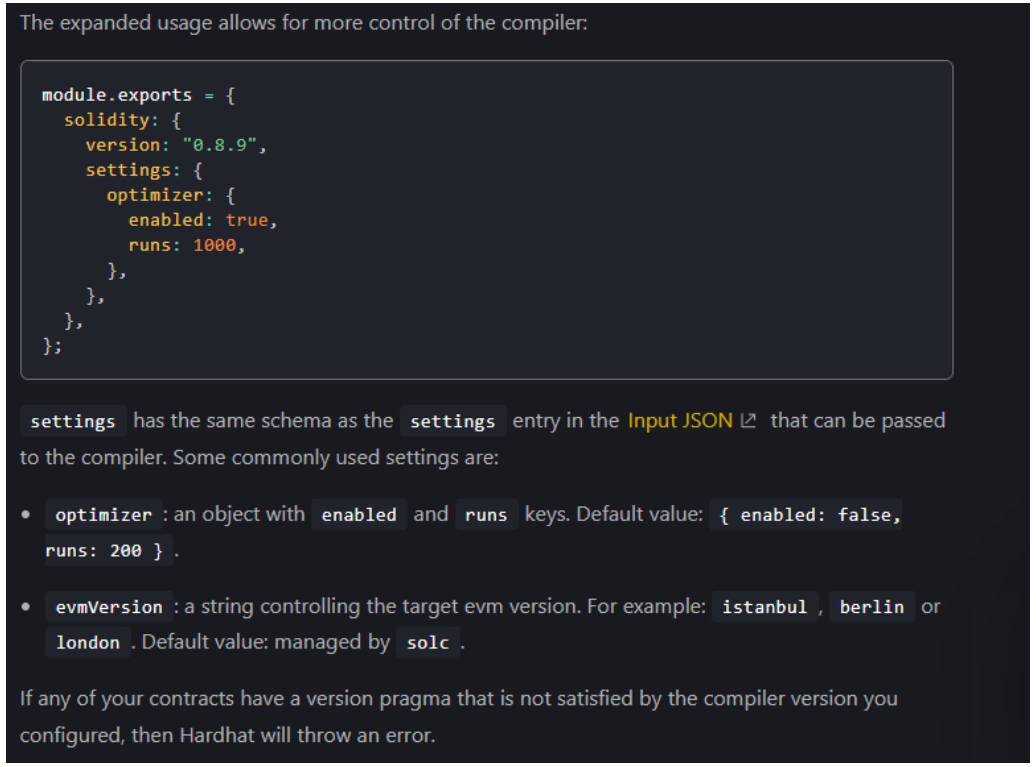
If you need to customize the Solidity compiler options, then you can do so through the `solidity` field in your `hardhat.config.js`. The simplest way to use this field is via the shorthand for setting the compiler version, which we recommend always doing:

```
module.exports = {
  solidity: "0.8.9",
};
```

We recommend always setting a compiler version in order to avoid unexpected behavior or compiling errors as new releases of Solidity are published.

WARNING

Hardhat will automatically download the versions of `solc` that you set up. If you are behind an HTTP proxy, you may need to set the `HTTP_PROXY` or `HTTPS_PROXY` environment variable to the URL of your proxy.



The expanded usage allows for more control of the compiler:

```
module.exports = {
  solidity: {
    version: "0.8.9",
    settings: {
      optimizer: {
        enabled: true,
        runs: 1000,
      },
    },
  },
};
```

`settings` has the same schema as the `settings` entry in the Input JSON that can be passed to the compiler. Some commonly used settings are:

- `optimizer`: an object with `enabled` and `runs` keys. Default value: `{ enabled: false, runs: 200 }`.
- `evmVersion`: a string controlling the target evm version. For example: `istanbul`, `berlin` or `london`. Default value: managed by `solc`.

If any of your contracts have a version pragma that is not satisfied by the compiler version you configured, then Hardhat will throw an error.