GitHubForumBlogWebsite

Preparing for mainnet

https://docs.openzeppelin.com/learn/preparing-for-mainnet

After running your project on a testnet for some time without issues, you will want to deploy it to the main Ethereum network (aka *mainnet*). However, the planning for going to mainnet should begin much earlier than your planned release date.

In this guide, we will go through Ethereum-specific considerations for taking your project to production, such as:

- Auditing and Security
- Verifying Source Code
- Managing Keys Securely
- Handling Project Governance

Remember that, while managing your contracts in a testnet and in mainnet is technically the same, there are important differences when on mainnet, since your project now manages real value for your users.

Auditing and security

While security affects all of software development, security in smart contracts is particularly important. Anyone can send a transaction directly to your contracts with any payload, and all your contract code and state is publicly accessible. To make matters worse, in the event you are hacked, there is no recourse to reclaim the stolen funds - they are gone for good in a decentralized network.

With this in mind, security should be a primary concern at all stages of development. This means that **security is not something that you sprinkle on your project a week before you release**, but a guiding principle starting day one of your project.

Review smart contract security best practices as you begin coding, join the security discussions in our forum, and make sure to go through our quality checklist to ensure your project is healthy.

Once you are done, it's a good time to request an audit with one or more auditing firms. You can request an audit from the OpenZeppelin Research Team - we are an experienced team with a long track record.

Remember that audits do not ensure the absence of bugs, but having several experienced security researchers go through your code certainly helps.

Verifying your source code

Right after you deploy your contracts to mainnet, you should **verify their source code**. This process involves submitting the Solidity code to a third-party, such as Etherscan or Etherchain, who will compile it and *verify* that it matches the deployed assembly. This allows any user to view your contract code in a block explorer, and know that it corresponds to the assembly actually running at that address.

You can verify your contracts manually on the Etherscan website.

You can also use Hardhat Etherscan plugin.

To do this, install the plugin:

```
npm install --save-dev @nomiclabs/hardhat-etherscan
```

Update your hardhat configuration:

```
// hardhat.config.js
const { etherscanApiKey, projectId, mnemonic } = require('./secrets.json');
require("@nomiclabs/hardhat-etherscan");        ...
module.exports = {
 networks: {    mainnet: { ... }    },
 etherscan: {
   apiKey: etherscanApiKey  }};
```

Finally run the `verify` task, passing the address of the contract, the network where it's deployed, and the constructor arguments that were used to deploy it (if any):

```
npx hardhat verify --network mainnet DEPLOYED_CONTRACT_ADDRESS "Constructor argument 1"
```

You will need an Etherscan API key to use their service.

When you deploy an upgradeable contract the contract that the user interacts with will be just a proxy, and the actual logic will be in the implementation contract. Etherscan does have support for correctly showing OpenZeppelin proxies and their implementations, but other explorers may not.

Key management

When working on mainnet you need to take special care to secure your private keys. The accounts you use to deploy and interact with your contracts will hold real Ether, which has real value and is a tempting target for hackers. Take every precaution to protect your keys, and consider using a hardware wallet if necessary.

Unlike on a testnet, you cannot get mainnet Ether from a faucet. You will need to head to an exchange to trade in other coins or fiat to get real Ether for deploying your contracts.

Additionally, you may define certain accounts to have special privileges in your system - and you should take extra care to secure them.

Admin accounts

An *admin* (short for *administrator*) account is one that has special privileges in your system. For example, an admin may have the power to pause a contract. If such an account were to fall in the hands of a malicious user, they could wreak havoc in your system.

A good option for securing admin accounts is to use a special contract, such as a multisig, instead of a regular externally owned account. A *multisig* is a contract that can execute any action, *as long as a predefined number of trusted members agree upon it*. Gnosis Safe is a good multisig to use.

Upgrades admin

A special administrator account in an OpenZeppelin Upgrades Plugins project is the account with the power to *upgrade* other contracts. This defaults to the externally owned account used to deploy the contracts: while this is good enough for a local or testnet deployment, in mainnet you need to better secure your contracts. An attacker who gets hold of your upgrade admin account can change any contract in your system!

With this in mind, it is a good idea to **change the ownership of the ProxyAdmin** after deployment - for example, to a multisig. To do this, you can use `admin.transferProxyAdminOwnership` to transfer ownership of our `ProxyAdmin` contract.

When you need to upgrade your contracts, we can use `prepareUpgrade` to validate and deploy a new implementation contract ready to be used when our proxy is updated.

Project governance

It can be argued that admin accounts reflect that a project is not actually *decentralized*. After all, if an account can single-handedly change any contract in the system, we are not exactly creating a trustless environment.

Here is where *governance* comes in. In many cases there will be some operations in your project that require special privileges, from fine-tuning system parameters, to running full contract upgrades. You will need to choose how those actions will be decided upon: whether it is by a small group of trusted developers, or by public voting of all project stakeholders.

There is no right answer here. Which governance scheme you pick for your project will largely depend on what you are building and who your community is.

Next steps

Congratulations! You have reached the end of the development journey, from writing your first contract to deploying to production. But work is far from over. Now you have to start collecting live user feedback, adding new features to your system (made possible via contract upgrades!), monitoring your application, and ultimately scaling your project.

On this site, you have at your disposal detailed guides and reference for all the projects in the OpenZeppelin platform, so you can pick whatever you need to build your Ethereum application. Happy coding!