**Deploy your ERC20 crypto currency**

**ERC20 Deploy Script**

With our simple token contract written, we'll of course want to test and deploy it. Let's get started with writing a deploy script.

In your workspace's script folder, create a file named DeployOurToken.s.sol.

We expect OurToken to behave the same, regardless of the chain it's deployed on, so we don't really need a HelperConfig for this example. We'll skip that step and move write into writing the deploy script.

To begin, we can import Script and OurToken as well as add the skeleton of our run function:

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.18;

import {Script} from "forge-std/Script.sol";

import {OurToken} from "../src/OurToken.sol";

contract DeployOurToken is Script {

function run() external {}

}

We're going to keep this really basic, we just want to deploy OurToken. We know that OurToken requires an initial supply as a constructor parameter, so let's declare that and then deploy our contract.

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.18;

import {Script} from "forge-std/Script.sol";

import {OurToken} from "../src/OurToken.sol";

contract DeployOurToken is Script {

uint256 public constant INITIAL\_SUPPLY = 1000 ether;

function run() external returns (OurToken) {

vm.startBroadcast();

OurToken ot = new OurToken(INITIAL\_SUPPLY);

vm.stopBroadcast();

return ot;

}

}

Our deploy script looks great! To make things a little easier on ourselves when using the CLI to run this script, copy the Makefile from the course GitHub repo and add this to our workspace (I've included it below to copy if needed).

**Makefile**

-include .env

.PHONY: all test clean deploy fund help install snapshot format anvil

DEFAULT\_ANVIL\_KEY := 0xac0974bec39a17e36ba4a6b4d238ff944bacb478cbed5efcae784d7bf4f2ff80

help:

@echo "Usage:"

@echo " make deploy [ARGS=...]\n example: make deploy ARGS=\"--network sepolia\""

@echo ""

@echo " make fund [ARGS=...]\n example: make deploy ARGS=\"--network sepolia\""

all: clean remove install update build

# Clean the repo

clean :; forge clean

# Remove modules

remove :; rm -rf .gitmodules && rm -rf .git/modules/\* && rm -rf lib && touch .gitmodules && git add . && git commit -m "modules"

install :; forge install Cyfrin/foundry-devops@0.0.11 --no-commit --no-commit && forge install foundry-rs/forge-std@v1.5.3 --no-commit && forge install openzeppelin/openzeppelin-contracts@v4.8.3 --no-commit

# Update Dependencies

update:; forge update

build:; forge build

test :; forge test

snapshot :; forge snapshot

format :; forge fmt

anvil :; anvil -m 'test test test test test test test test test test test junk' --steps-tracing --block-time 1

NETWORK\_ARGS := --rpc-url http://localhost:8545 --private-key $(DEFAULT\_ANVIL\_KEY) --broadcast

ifeq ($(findstring --network sepolia,$(ARGS)),--network sepolia)

NETWORK\_ARGS := --rpc-url $(SEPOLIA\_RPC\_URL) --private-key $(PRIVATE\_KEY) --broadcast --verify --etherscan-api-key $(ETHERSCAN\_API\_KEY) -vvvv

endif

deploy:

@forge script script/DeployOurToken.s.sol:DeployOurToken $(NETWORK\_ARGS)

# cast abi-encode "constructor(uint256)" 1000000000000000000000000 -> 0x00000000000000000000000000000000000000000000d3c21bcecceda1000000

# Update with your contract address, constructor arguments and anything else

verify:

@forge verify-contract --chain-id 11155111 --num-of-optimizations 200 --watch --constructor-args 0x00000000000000000000000000000000000000000000d3c21bcecceda1000000 --etherscan-api-key $(ETHERSCAN\_API\_KEY) --compiler-version v0.8.19+commit.7dd6d404 0x089dc24123e0a27d44282a1ccc2fd815989e3300 src/OurToken.sol:OurToken

Now, by running make anvil (open a new terminal once your chain has started!) followed by make deploy...

**Wrap Up**

Woo! Deployment to our anvil chain successful, let's go!

In the next lesson, we'll test our contracts with the help of some AI tools and recap everything we've gone over so far. See you there!

**Updates**

Obtaining testnet funds -> Tenderly virtual testnets

*Last updated on May 27, 2025*

Obtaining testnet funds can be difficult as faucets often require you to hold mainnet funds or sign in with some other service like GitHub.

So, instead of deploying and interacting on a testnet like Sepolia, you can use a Tenderly virtual testnet. More information on creating your own virtual testnet can be found in [this video](https://updraft.cyfrin.io/courses/blockchain-basics/basics/sending-your-first-transaction-tenderly)!

Note that, as of today, Tenderly does not support ZKsync.