

1. Deploy:

yarn hardhat deploy

The screenshot shows the VS Code interface with the Explorer panel on the left displaying the file structure of a project named 'HARDHAT-SMARTCONTRACT-LOTTERY-FCC'. The main editor area shows the 'Start' page with options to 'New File...', 'Open File...', or 'Clone Git Repository...'. The 'Terminal' panel at the bottom displays the output of the 'yarn hardhat deploy' command. The output shows that the contract was successfully deployed to a local network. The terminal text is as follows:

```
Compiled 8 Solidity files successfully
Local network detected! Deploying mocks...
deploying "VRFCoordinatorV2Mock" (tx: 0xb1811099ad4653047586f3d30dea9d8cae51f74a48277c9ab76ac8ab3ed95b61)...: deployed at 0x5Fb0B2315678afe
cb367f032d93f642f64180aa3 with 1803306 gas
Mocks Deployed!
-----
You are deploying to a local network, you'll need a local network running to interact
Please run `yarn hardhat console --network localhost` to interact with the deployed smart contracts!
-----
deploying "Raffle" (tx: 0xf63ebeb7d2200809a92b251f909079f5ccc657ddc85e7ca13b6ebd843d219fe0)...: deployed at 0xcF7Ed3AccA5a67e9e704C703E808
7F634fB0FC9 with 1231792 gas
Run Price Feed contract with command:
yarn hardhat run scripts/enterRaffle.js --network localhost
-----
Done in 17.62s.
gitpod /workspace/ethers-simple-storage-fcc/hardhat-smartcontract-lottery-fcc (typescript) $
```

2. Testing

yarn hardhat test

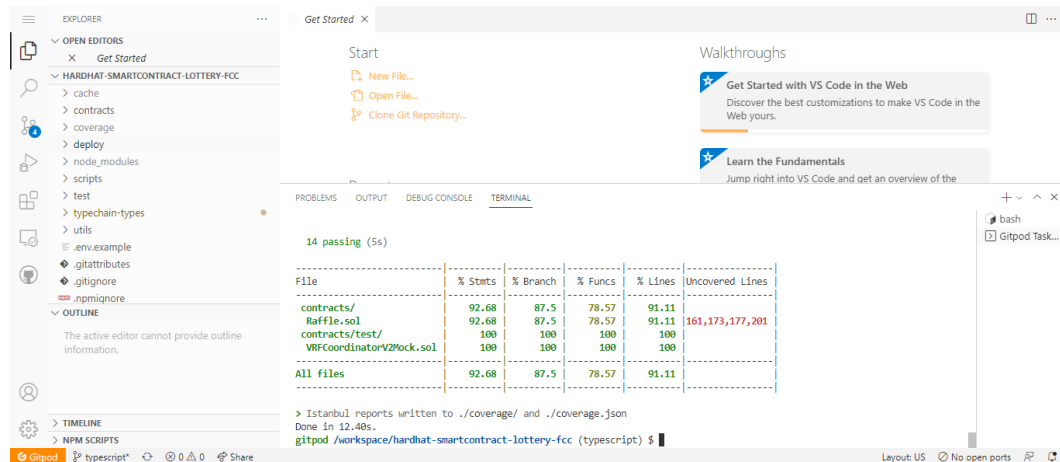
The screenshot shows the VS Code interface with the Explorer panel on the left displaying the file structure of a project named 'HARDHAT-SMARTCONTRACT-LOTTERY-FCC'. The main editor area shows the 'Start' page with options to 'New File...', 'Open File...', or 'Clone Git Repository...'. The 'Terminal' panel at the bottom displays the output of the 'yarn hardhat test' command. The output shows that the tests passed successfully. The terminal text is as follows:

```
✓ returns false if raffle isn't open (74ms)
✓ returns false if enough time hasn't passed
✓ returns true if enough time has passed, has players, eth, and is open
performUpkeep
✓ can only run if checkpoint is true (47ms)
✓ reverts if checkpoint is false
✓ updates the raffle state and emits a requestId (64ms)
fulfillRandomWords
✓ can only be called after performUpkeep
WinnerPicked event fired!
✓ picks a winner, resets, and sends money (4077ms)

14 passing (6s)
Done in 9.34s.
gitpod /workspace/hardhat-smartcontract-lottery-fcc (typescript) $
```

3. Test Coverage

yarn hardhat coverage



Deployment to a testnet or mainnet

1. Setup environment variables

You'll want to set your `RINKEBY_RPC_URL` and `PRIVATE_KEY` as environment variables. You can add them to a `.env` file, similar to what you see in `.env.example`.

- `PRIVATE_KEY`: The private key of your account (like from [metamask](#)). **NOTE: FOR DEVELOPMENT, PLEASE USE A KEY THAT DOESN'T HAVE ANY REAL FUNDS ASSOCIATED WITH IT.**
 - You can [learn how to export it here](#).
- `RINKEBY_RPC_URL`: This is url of the rinkeby testnet node you're working with. You can get setup with one for free from [Alchemy](#)

2. Get testnet ETH

Head over to [faucets.chain.link](#) and get some tesnet ETH & LINK. You should see the ETH and LINK show up in your metamask. You can read more on setting up your wallet with LINK.

3. Setup a Chainlink VRF Subscription ID

Head over to [vrf.chain.link](#) and setup a new subscription, and get a subscriptionId. You can reuse an old subscription if you already have one.

You can follow the [instructions](#) if you get lost. You should leave this step with:

1. A subscription ID
2. Your subscription should be funded with LINK
3. Deploy

In your helper-hardhat-config.js add your subscriptionId under the section of the chainId you're using (aka, if you're deploying to rinkeby, add your subscriptionId in the subscriptionId field under the 4 section.)

Then run:

```
yarn hardhat deploy --network rinkeby
```

And copy / remember the contract address.

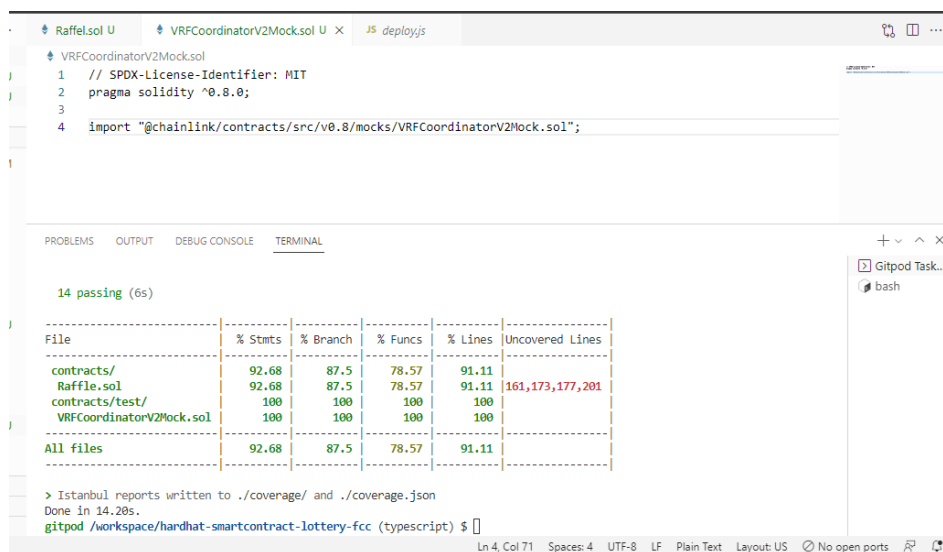
4. Add your contract address as a Chainlink VRF Consumer

Go back to [vrf.chain.link](#) and under your subscription add Add consumer and add your contract address. You should also fund the contract with a minimum of 1 LINK.

5. Register a Chainlink Keepers Upkeep

You can follow the [documentation](#) if you get lost.

Go to [keepers.chain.link](#) and register a new upkeep. Your UI will look something like this [once](#)



The screenshot shows a VS Code editor with a file named `VRFCoordinatorV2Mock.sol` open. The file contains the following code:

```
1 // SPDX-License-Identifier: MIT
2 pragma solidity ^0.8.0;
3
4 import "@chainlink/contracts/src/v0.8/mocks/VRFCoordinatorV2Mock.sol";
```

Below the editor, the `TERMINAL` tab is active, displaying the output of a test run:

```
14 passing (6s)
```

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
contracts/	92.68	87.5	78.57	91.11	
Raffle.sol	92.68	87.5	78.57	91.11	161,173,177,201
contracts/test/	100	100	100	100	
VRFCoordinatorV2Mock.sol	100	100	100	100	
All files	92.68	87.5	78.57	91.11	

Below the table, the following text is displayed:

```
> Istanbul reports written to ./coverage/ and ./coverage.json
Done in 14.28s.
gitpod /workspace/hardhat-smartcontract-lottery-fcc (typescript) $
```

```
1 // SPDX-License-Identifier: MIT
2
3 pragma solidity ^0.8.7;
4
5 import "@chainlink/contracts/src/v0.8/interfaces/VRFCoordinatorV2Interface.sol";
6 import "@chainlink/contracts/src/v0.8/VRFConsumerBaseV2.sol";
7 import "@chainlink/contracts/src/v0.8/interfaces/KeeperCompatibleInterface.sol";
8 import "hardhat/console.sol";
9
10 error Raffle_UpgradeNotNeeded(uint256 currentBalance, uint256 numPlayers, uint256 raffleState);
11 error Raffle_TransferFailed();
12 error Raffle_SendMoreToEnterRaffle();
13 error Raffle_RaffleNotOpen();
14
15 /**@title A sample Raffle Contract
16  * @author Patrick Collins
17  * @notice This contract is for creating a sample raffle contract
18  * @dev This implements the Chainlink VRF Version 2
19  */
20 contract Raffle is VRFConsumerBaseV2, KeeperCompatibleInterface {
21     /* Type declarations */
22     enum RaffleState {
23         OPEN,
24         CALCULATING
25     }
26     /* State variables */
27     // Chainlink VRF Variables
```

```
28 VRFCoordinatorV2Interface private immutable i_vrfCoordinator;
29 uint64 private immutable i_subscriptionId;
30 bytes32 private immutable i_gasLane;
31 uint32 private immutable i_callbackGasLimit;
32 uint16 private constant REQUEST_CONFIRMATIONS = 3;
33 uint32 private constant NUM_WORDS = 1;
34
35 // Lottery Variables
36 uint256 private immutable i_interval;
37 uint256 private s_lastTimeStamp;
38 address private s_recentWinner;
39 uint256 private i_entranceFee;
40 address payable[] private s_players;
41 RaffleState private s_raffleState;
42
43 /* Events */
44 event RequestedRaffleWinner(uint256 indexed requestId);
45 event RaffleEnter(address indexed player);
46 event WinnerPicked(address indexed player);
47
48 /* Functions */
49 constructor(
50     address vrfCoordinatorV2,
51     uint64 subscriptionId,
52     bytes32 gasLane, // keyHash
53     uint256 interval,
54     uint256 entranceFee,
```

```
53     uint256 interval,
54     uint256 entranceFee,
55     uint32 callbackGasLimit
56 ) VRFConsumerBaseV2(vrfCoordinatorV2) {
57     i_vrfCoordinator = VRFCoordinatorV2Interface(vrfCoordinatorV2);
58     i_gasLane = gasLane;
59     i_interval = interval;
60     i_subscriptionId = subscriptionId;
61     i_entranceFee = entranceFee;
62     s_raffleState = RaffleState.OPEN;
63     s_lastTimeStamp = block.timestamp;
64     i_callbackGasLimit = callbackGasLimit;
65 }
66
67 function enterRaffle() public payable {
68     // require(msg.value >= i_entranceFee, "Not enough value sent");
69     // require(s_raffleState == RaffleState.OPEN, "Raffle is not open");
70     if (msg.value < i_entranceFee) {
71         revert Raffle_SendMoreToEnterRaffle();
72     }
73     if (s_raffleState != RaffleState.OPEN) {
74         revert Raffle_RaffleNotOpen();
75     }
76     s_players.push(payable(msg.sender));
77     // Emit an event when we update a dynamic array or mapping
78     // Named events with the function name reversed
79     emit RaffleEnter(msg.sender);
80 }
81
82 /**
```

```
Raffel.sol U x VRFCoordinatorV2Mock.sol U
Raffel.sol
81
82 /**
83  * @dev This is the function that the Chainlink Keeper nodes call
84  * they look for 'upkeepNeeded' to return True.
85  * the following should be true for this to return true:
86  * 1. The time interval has passed between raffle runs.
87  * 2. The lottery is open.
88  * 3. The contract has ETH.
89  * 4. Implicity, your subscription is funded with LINK.
90  */
91 function checkUpkeep(
92     bytes memory /* checkData */
93 )
94     public
95     view
96     override
97     returns (
98         bool upkeepNeeded,
99         bytes memory /* performData */
100     )
101 {
102     bool isOpen = RaffleState.OPEN == s_affleState;
103     bool timePassed = ((block.timestamp - s_lastTimeStamp) > i_interval);
104     bool hasPlayers = s_players.length > 0;
105     bool hasBalance = address(this).balance > 0;
106     upkeepNeeded = (timePassed && isOpen && hasBalance && hasPlayers);
107     return (upkeepNeeded, "0x0"); // can we comment this out?
108 }
109
110 /**
```