**Deploy a mock Chainlink VRF**

**Chainlink VRF mock contract**

A mock contract is a type of smart contract used in testing and development environments to simulate the behavior of real contracts. It allows us to create controlled and predictable scenarios for testing purposes without relying on actual external contracts or data sources. Moreover, it facilitates testing using Anvil, which is extremely fast and practical in comparison to a testnet.

In the last lesson, we stopped on HelperConfig.s.sol:

function getOrCreateAnvilEthConfig()

public

returns (NetworkConfig memory anvilNetworkConfig)

{

// Check to see if we set an active network config

if (activeNetworkConfig.vrfCoordinator != address(0)) {

return activeNetworkConfig;

}

}

We need to treat the other side of the (activeNetworkConfig.vrfCoordinatorV2 != address(0)) condition. What happens if that is false?

If that is false we need to deploy a mock vrfCoordinatorV2\_5 and pass its address inside a NetworkConfig that will be used on Anvil.

Please use your Explorer on the left side to access the following path:

foundry-smart-contract-lottery-cu/lib/chainlink/contracts/src/v0.8/vrf/

Inside you'll find multiple folders, one of which is called mocks. Inside that folder, you can find the VRFCoordinatorV2\_5Mock mock contract created by Chainlink.

Add the following line in the imports section of HelperConfig.s.sol:

import {VRFCoordinatorV2\_5Mock} from "chainlink/src/v0.8/vrf/mocks/VRFCoordinatorV2\_5Mock.sol";

Amazing! Now let's keep on working on the getOrCreateAnvilEthConfig function. We need to deploy the vrfCoordinatorV2Mock, but if we open it we'll see that its constructor requires some parameters:

contract VRFCoordinatorV2\_5Mock is SubscriptionAPI, IVRFCoordinatorV2Plus {

uint96 public immutable i\_base\_fee;

uint96 public immutable i\_gas\_price;

int256 public immutable i\_wei\_per\_unit\_link;

}

The i\_base\_fee is the flat fee that VRF charges for the provided randomness. i\_gas\_price which is the gas consumed by the VRF node when calling your function. i\_wei\_per\_unit\_link is the LINK price in ETH in wei units. Given the way it's structured the callback gas is paid initially by the node which needs to be reimbursed.

We add the following lines to the getOrCreateAnvilEthConfig function:

/\* VRF Mock Values \*/

uint96 public constant MOCK\_BASE\_FEE = 0.25 ether;

uint96 public constant MOCK\_GAS\_PRICE\_LINK = 1e9;

int256 public constant MOCK\_WEI\_PER\_UNIT\_LINK = 4e15;

vm.startBroadcast();

VRFCoordinatorV2\_5Mock vrfCoordinatorMock = new VRFCoordinatorV2\_5Mock(

MOCK\_BASE\_FEE,

MOCK\_GAS\_PRICE\_LINK,

MOCK\_WEI\_PER\_UNIT\_LINK,

);

vm.stopBroadcast();

Amazing! Now that we have everything we need, let's perform the return, similar to what we did in getSepoliaEthConfig.

return NetworkConfig({

entranceFee: 0.01 ether,

interval: 30, // 30 seconds

vrfCoordinator: address(vrfCoordinatorMock),

// gasLane value doesn't matter.

gasLane: 0x787d74caea10b2b357790d5b5247c2f63d1d91572a9846f780606e4d953677ae,

subscriptionId: 0,

callbackGasLimit: 500\_000,

});

Great! Now this is fixed let's continue testing and deploying our Raffle contract.