Local testing using a Mock contract

You are viewing the VRF v2 guide - Direct funding method.

To learn how to request random numbers with a subscription, see theSubscription Method guide.

Security Considerations

Be sure to review your contracts with thesecurity considerations in mind.

This guide explains how to test Chainlink VRF v2 on aRemix IDE sandbox blockchain environment.Note: You can reuse the same logic on another development environment, such as Hardhat or Truffle. For example, read the Hardhat Starter KitRandomNumberDirectFundingConsumer unit tests .

Test on public testnets thoroughly

Even though local testing has several benefits, testing with a VRF mock covers the bare minimum of use cases. Make sure to test your consumer contract throughly on public testnets.

Benefits of local testing

Testing locally using mock contracts saves you time and resources during development. Some of the key benefits include:

- Faster feedback loop: Immediate feedback on the functionality and correctness of your smart contracts. This helps you quickly identify and fix issues without waiting for transactions to be mined/validated on a testnet.
- Saving your native testnet gas: Deploying and interacting with contracts requires paying gas fees. Although native testnet gas does not have any associated value, supply is limited by public faucets. Using mock contracts locally allows you to test your contracts freely without incurring any expenses.
- Controlled environment: Local testing allows you to create a controlled environment where you can manipulate various parameters, such as block time and gas prices, to test your smart contracts' function as expected under different conditions.
- Isolated testing: You can focus on testing individual parts of your contract, ensuring they work as intended before integrating them with other components.
- Easier debugging: Because local tests run on your machine, you have better control over the debugging process. You can set breakpoints, inspect variables, and step through your code to identify and fix issues.
- Comprehensive test coverage: You can create test cases to cover all possible scenarios and edge cases.

Testing logic

Complete the following tasks to test your VRF v2 consumer locally:

1. Deploy theVRFCoordinatorV2Mock . This contract is a mock of theVRFCoordinatorV2 contract.
2. Deploy theMockV3Aggregator contract.
3. Deploy theLinkToken contract.
4. Deploy theVRFV2Wrapper contract.
5. Call the VRFV2WrappersetConfig function to set wrapper specific parameters.
6. Fund the VRFv2Wrapper subscription.
7. Call the the VRFCoordinatorV2MockaddConsumer function to add the wrapper contract to your subscription.
8. Deploy your VRF consumer contract.
9. Fund your consumer contract with LINK tokens.
10. Request random words from your consumer contract.
11. Call the VRFCoordinatorV2MockfulfillRandomWords function to fulfill your consumer contract request.

Prerequisites

This guide will require you to finetune the gas limit when fulfilling requests. When writing, manually setting up the gas limits on RemixIDE is not supported, so you will use RemixIDE in conjunction withMetamask .Ganache lets you quickly fire up a personal Ethereum blockchain. If you still need to install Ganache, follow theofficial guide .

1. Once Ganache is installed, click theQUICKSTARTbutton to start a local Ethereum node.

Note: Make sure to note the RPC server. In this example, the RPC server ishttp://127.0.0.1:7545/. 2. Follow the MetamaskofficialguideOpen in Remix What is Remix? to add a custom network manually. 3. Import a Ganache account into Metamask.

1. On Ganache, click on thekeysymbol of the first account:
2. Copy the private key:
3. Follow the MetamaskofficialguideOpen in Remix What is Remix? to import an account using a private key.
4. Your Metamask is connected toGanache, and you should have access to the newly imported account.

Testing

Open the contracts on RemixIDE

OpenVRFCoordinatorV2Mockand compile in Remix:

// SPDX-License-Identifier: MITpragmasolidity^0.8.7;import"@chainlink/contracts/src/v0.8/mocks/VRFCoordinatorV2Mock.sol"; Open in Remix What is Remix? OpenMockV3Aggregatorand compile in Remix:

// SPDX-License-Identifier: MITpragmasolidity^0.8.7;import"@chainlink/contracts/src/v0.8/tests/MockV3Aggregator.sol"; Open in Remix What is Remix? OpenLinkTokenand compile in Remix:

// SPDX-License-Identifier: MITpragmasolidity^0.4.24;import"@chainlink/contracts/src/v0.4/LinkToken.sol"; Open in Remix What is Remix? OpenVRFV2Wrapperand compile in Remix:

// SPDX-License-Identifier: MITpragmasolidity^0.8.7;import"@chainlink/contracts/src/v0.8/vrf/VRFV2Wrapper.sol"; Open in Remix What is Remix? OpenRandomNumberDirectFundingConsumerV2and compile in Remix:

// SPDX-License-Identifier: MIT// An example of a consumer contract that directly pays for each request.pragmasolidity^0.8.7;import{ConfirmedOwner}from"@chainlink/contracts/src/v0.8/shared/access/ConfirmedOwner.sol";import{VRFV2WrapperConsumerBase}from"@chainlink/contracts/src/v0.8/ *** THIS IS AN EXAMPLE CONTRACT THAT USES HARDCODED VALUES FOR CLARITY. * THIS IS AN EXAMPLE CONTRACT THAT USES UNAUDITED CODE. * DO NOT USE THIS CODE IN PRODUCTION.** ***/contractRandomNumberDirectFundingConsumerV2isVRFV2WrapperConsumerBase,ConfirmedOwner{eventRequestSent(uint256requestId,uint32numWords,uint256paid);eventRequestFul amount paid in linkboolfulfilled;// whether the request has been successfully fulfilleduint256[]randomWords;}mapping(uint256=>RequestStatus)publics_requests;/** requestId --> requestStatus */// past requests Id.uint256[]publicrequestIds;uint256publiclastRequestId;// configuration: https://docs.chain.link/vrf/v2/direct-funding/supported-networks#configurationsconstructor(address_linkAddress,address_wrapperAddress)ConfirmedOwner(msg.sender)VRFV2WrapperConsumerBase(_linkAddress,_wrapperAddress){}// Depends on the number of requested values that you want sent to the// fulfillRandomWords() function. Test and adjust// this limit based on the network that you select, the size of the request,// and the processing of the callback request in the fulfillRandomWords()// function.// The default is 3, but you can set this higher.// For this example, retrieve 2 random values in one request.// Cannot exceed VRFV2Wrapper.getConfig().maxNumWords.functionrequestRandomWords(uint32_callbackGasLimit,uint16_requestConfirmations,uint32_numWords)externalonlyOwnerreturns(uint256requ {requestId=requestRandomness(_callbackGasLimit,_requestConfirmations,_numWords);uint256paid=VRF_V2_WRAPPER.calculateRequestPrice(_callbackGasLimit);uint256balance=LINK.l ,fulfilled:false});requestIds.push(requestId);lastRequestId=requestId;emitRequestSent(requestId,_numWords,paid);returnrequestId;}functionfulfillRandomWords(uint256_requestId,uint256[{returnrequestIds.length;}functiongetRequestStatus(uint256_requestId)externalviewreturns(uint256paid,boolfulfilled,uint256[]memoryrandomWords) {RequestStatusmemoryrequest=s_requests[_requestId];if(request.paid==0)revertRequestNotFound(_requestId);return(request.paid,request.fulfilled,request.randomWords);}/** Allow withdraw of Link tokens from the contract */functionwithdrawLink(address_receiver)publiconlyOwner{boolsuccess=LINK.transfer(_receiver,LINK.balanceOf(address(this)));if(!success)revertLinkTransferError(msg.sender,_receiver,LINK.balanceO Open in Remix What is Remix? Your RemixIDE file explorer should display the opened contracts:

Select the correct RemixIDE environment

UnderDEPLOY & RUN TRANSACTIONS:

1. Set the Environment toInjected Provider - Metamask:
2. On Metamask, connect your Ganache account to the Remix IDE.
3. Click on Connect. The RemixIDE environment should be set to the correct environment, and the account should be the Ganache account.

Deploy VRFCoordinatorV2Mock

1. OpenVRFCoordinatorV2Mock.sol.
2. UnderDEPLOY & RUN TRANSACTIONS, selectVRFCoordinatorV2Mock.
3. UnderDEPLOY, fill in the_BASEFEEand_GASPRICELINK. These variables are used in theVRFCoordinatorV2Mockcontract to represent the base fee and the gas price (in LINK tokens) for the VRF requests. You can set:_BASEFEE=100000000000000000and_GASPRICELINK=1000000000.
4. Click ontransactto deploy theVRFCoordinatorV2Mockcontract.
5. A Metamask popup will open. Click onConfirm.
6. Once deployed, you should see theVRFCoordinatorV2Mockcontract underDeployed Contracts.
7. Note the address of the deployed contract.

Deploy MockV3Aggregator

TheMockV3Aggregatorcontract is designed for testing purposes, allowing you to simulate an oracle price feed without interacting with the existing Chainlink network.

1. OpenMockV3Aggregator.sol.
2. UnderDEPLOY & RUN TRANSACTIONS, selectMockV3Aggregator.
3. UnderDEPLOY, fill in_DECIMALSand_INITIALANSWER. These variables are used in theMockV3Aggregatorcontract to represent the number of decimals the aggregator's answer should have and the most recent price feed answer. You can set:_DECIMALS=18and_INITIALANSWER=3000000000000000(We are considering that1 LINK = 0.003 native gas tokens).
4. Click ontransactto deploy theMockV3Aggregatorcontract.
5. A Metamask popup will open. Click onConfirm.
6. Once deployed, you should see theMockV3Aggregatorcontract underDeployed Contracts.
7. Note the address of the deployed contract.

Deploy LinkToken

The Chainlink VRF v2 direct funding method requires your consumer contract to pay for VRF requests in LINK. Therefore, you have to deploy theLinkTokencontract to your local blockchain.

1. OpenLinkToken.sol.
2. UnderDEPLOY & RUN TRANSACTIONS, selectLinkToken.
3. UnderDEPLOY, click ontransactto deploy theLinkTokencontract.
4. A Metamask popup will open. Click onConfirm.
5. Once deployed, you should see theLinkTokencontract underDeployed Contracts.
6. Note the address of the deployed contract.

Deploy VRFV2Wrapper

As the VRF v2 direct fundingend-to-end diagram explains, theVRFV2Wrapperacts as a wrapper for the coordinator contract.

1. OpenVRFV2Wrapper.sol.
2. UnderDEPLOY & RUN TRANSACTIONS, selectVRFV2Wrapper.
3. UnderDEPLOY, fill in_LINKwith theLinkTokencontract address,_LINKETHFEEDwith theMockV3Aggregatorcontract address, and_COORDINATORwith theVRFCoordinatorV2Mockcontract address.
4. click ontransactto deploy theVRFV2Wrappercontract.
5. A Metamask popup will open. Click onConfirm.
6. Once deployed, you should see theVRFV2Wrappercontract underDeployed Contracts.
7. Note the address of the deployed contract.

Configure the VRFV2Wrapper

1. UnderDeployed Contracts, open the functions list of your deployedVRFV2Wrappercontract.

2. Click onsetConfigand fill in_wrapperGasOverheadwith60000,_coordinatorGasOverheadwith52000,_wrapperPremiumPercentagewith10,_keyHashwith0xd89b2bf150e3b9e13446986e571fb9cab24b13cea0a43ea20a6049a8 and_maxNumWordswith10.Noteon these variables:

3. _wrapperGasOverhead: This variable reflects the gas overhead of the wrapper's fulfillRandomWords function. The cost for this gas is passed to the user.

4. _coordinatorGasOverhead: This variable reflects the gas overhead of the coordinator'sfulfillRandomWordsfunction. The cost for this gas is billed to theVRFV2Wrappersubscription and must, therefore, be included in the VRF v2 direct funding requests pricing.
5. _wrapperPremiumPercentage: This variable is the premium ratio in percentage. For example, a value of 0 indicates no premium. A value of 15 indicates a 15 percent premium.
6. _keyHash: The gas lane key hash value is the maximum gas price you are willing to pay for a request in wei.
7. _maxNumWords: This variable is the maximum number of words requested in a VRF v2 direct funding request.
8. click ontransact.
9. A Metamask popup will open. Click onConfirm.

Fund the VRFV2Wrapper subscription

When deployed, theVRFV2Wrappercontract creates a new subscription and adds itself to the newly created subscription. If you started this guide from scratch, the subscription ID should be 1.

1. UnderDeployed Contracts, open the functions list of your deployedVRFCoordinatorV2Mockcontract.
2. ClickfundSubscriptionto fund theVRFV2Wrappersubscription. In this example, you can set the_subidto1(which is your newly created subscription ID) and the_amountto10000000000000000000(10 LINK).
3. A Metamask popup will open. Click onConfirm.

Deploy the VRF consumer contract

1. In the file explorer, openRandomNumberDirectFundingConsumerV2.sol.
2. UnderDEPLOY & RUN TRANSACTIONS, selectRandomNumberDirectFundingConsumerV2.
3. UnderDEPLOY, fill in_LINKADDRESS_with theLinkTokencontract address, and_WRAPPERADDRESS_with the deployedVRFV2Wrapperaddress.
4. Click ontransactto deploy theRandomNumberDirectFundingConsumerV2contract.
5. A Metamask popup will open. Click onConfirm.
6. Once deployed, you should see theRandomNumberDirectFundingConsumerV2contract underDeployed Contracts.
7. Note the address of the deployed contract.

Fund your VRF consumer contract

1. UnderDeployed Contracts, open the functions list of your deployedLinkTokencontract.
2. Click ontransferand fill in the_towith your consumer contract address and_valuewith LINK tokens amount. For this example, you can set the_valueto10000000000000000000(10 LINK).
3. Click ontransact.
4. A Metamask popup will open. Click onConfirm.

Request random words

Request three random words.

1. UnderDeployed Contracts, open the functions list of your deployedRandomNumberConsumerV2contract.
2. InrequestRandomWords, fill in_callbackGasLimitwith300000,_requestConfirmationswith3and_numWordswith3.
3. Click ontransact.
4. A Metamask popup will open.

Set your gas limit in MetaMask

Remix IDE doesn't set the right gas limit, so you mustedit the gas limit in MetaMaskwithin theAdvanced gas controlssettings.

For this example to work, set the gas limit to400,000in MetaMask.

First,enableAdvanced gas controlsin your MetaMask settings.

Before confirming your transaction in MetaMask, navigate to the screen where you can edit the gas limit: SelectSite suggested>Advanced>Advanced gas controlsand selectEditnext to theGas limitamount. Update theGas limitvalue to400000and selectSave. Finally, confirm the transaction. 5. Click onConfirm. 6. In the RemixIDE console, read your transaction logs to find the VRF request ID. In this example, the request ID is1. 7. Note your request ID.

Fulfill the VRF request

Because you are testing on a local blockchain environment, you must fulfill the VRF request yourself.

1. UnderDeployed Contracts, open the functions list of your deployedVRFCoordinatorV2Mockcontract.
2. ClickfulfillRandomWordsand fill in_requestIdwith your VRF request ID and_consumerwith theVRFV2Wrappercontract address.
3. Click ontransact.
4. A Metamask popup will open.

Set your gas limit in MetaMask

Remix IDE doesn't set the right gas limit, so you mustedit the gas limit in MetaMaskwithin theAdvanced gas controlssettings.

For this example to work, set the gas limit to1,000,000in MetaMask.

First,enableAdvanced gas controlsin your MetaMask settings.

Before confirming your transaction in MetaMask, navigate to the screen where you can edit the gas limit: SelectSite suggested>Advanced>Advanced gas controlsand selectEditnext to theGas limitamount. Update theGas limitvalue to1000000and selectSave. Finally, confirm the transaction. 5. Click onConfirm. 6. In the RemixIDE console, read your transaction logs to find the random words.

Check the results

1. UnderDeployed Contracts, open the functions list of your deployedRandomNumberDirectFundingConsumerV2contract.
2. Click onlastRequestIdto display the last request ID. In this example, the output is1.
3. Click ongetRequestStatuswith_requestIdequal to1:
4. You will get the amount paid, the status, and the random words.

Next steps

This guide demonstrated how to test a VRF v2 consumer contract on your local blockchain. We made the guide on RemixIDE for learning purposes, but you can reuse the samtesting logic on another development environment, such as Truffle or Hardhat. For example, read the Hardhat Starter KitRandomNumberDirectFundingConsumer unit tests .