Quick Start In order to get your VRF up and running you will need to first make your contract VRF Compatible.

Step 1: Set Up Your Development Environment

Ensure you have eitherFoundry orHardhat set up in your development environment.

Step 2: Install the Gelato VRF Contracts

Depending on your environment, use the following commands to import the Gelato VRF contracts:

- For Hardhat users:
-
    - Clone the repohere
-
    - Install dependenciesyarn install
-
    - Fill in.env
-
    - with variables in.env.example
- *
- For Foundry users:
- forge install gelatodigital/vrf-contracts --no-commit
-

Step 3: InheritGelatoVRFConsumerBase Contract

The recommended approach to integrate Gelato VRF is by inheriting from theGelatoVRFConsumerBase smart contract. Here's a simple example to help you set it up:

```

Copy // SPDX-License-Identifier: MIT pragmasolidity0.8.18;

import{GelatoVRFConsumerBase}from"./GelatoVRFConsumerBase.sol";

contractYourContractisGelatoVRFConsumerBase{ // Your contract's code goes here }

```

Understanding 1Balance

Before we dive into requesting randomness, it's crucial to understand the role of1Balance in using Gelato VRF. The Gelato VRF services necessitate that your Gelato balance is sufficiently funded. This balance caters to Gas fees and rewards Gelato Nodes for their computational tasks. For details about costs and funding your account, do visit our1balance documentation .

Note: It's important to remember that the current 1Balance system does not support withdrawals after depositing funds. Ensure to deposit only the amount you plan to utilize for Gelato VRF operations.

Step 4: Request Randomness

To request randomness, call the_requestRandomness() function. You should protect the call since it will take from your 1Balance. Thedata argument will be passed back to you by the W3F.

```

Copy functionrequestRandomness(bytesmemorydata)external{ require(msg.sender==...);
uint64requestId=_requestRandomness(data); }

```

Step 5: Implement the Callback function

Finally, implement the callback function.

```

Copy function_fulfillRandomness( bytes32randomness, uint64requestId, bytesmemorydata, )internaloverride{ } }
```

```

Step 6: Pass dedicated msg.sender

When you're ready to deploy your Gelato VRF-compatible contract, an important step is to include thededicated msg.sender as a constructor parameter. This ensures that your contract is set up to work with the correct operator for fulfilling the randomness requests.. It's crucial for ensuring that only authorized requests are processed.

Before deploying, visit theGelato VRF UI . There, you will find the specificdedicated msg.sender address assigned for your deployer address. This address is crucial for the security and proper functioning of your VRF requests. Learn more about it atSecurity Considerations#dedicated-msg.sender ```

Copy // SPDX-License-Identifier: MIT pragmasolidity0.8.18;

import{GelatoVRFConsumerBase}from"./GelatoVRFConsumerBase.sol";

contractYourContractisGelatoVRFConsumerBase{ constructor(addressoperator) GelatoVRFConsumerBase(operator) { // Additional initializations }

// The rest of your contract code }

```

and once you have your contract ready & deployed, grab the address andDeploy your VRF instance .

Previous Template Next Deploying your VRF Instance Last updated3 months ago On this page