API3

API3 is a collaborative project to deliver traditional API services to smart contract platforms in a decentralized and trust-minimized way. API3 provides the technology forAirnodes to push off-chain data to on-chain contracts. This data can then be queried directly through the Airnode (initiating a “pull-type” request) or throughdAPIs (data-feeds sourced directly from multiple first-party oracles owned and operated by API providers).

Querying the price of ARB through API3

Here’s an example of how to use an API3 data feed to query the current price of ARB on-chain. TheAPI3 market provides a list of all the dAPIs available across multiple chains including testnets. You can go forward and activate the dAPI you want to use.

API3 provides an npm package with the contracts needed to access their feeds. We first install that package in our project:

yarn

add @api3/contracts To use a data feed, we retrieve the information through the specific proxy address for that feed. We’ll use the IProxy interface to do so.

import

"@api3/contracts/api3-server-v1/proxies/interfaces/IProxy.sol" ; In this case, we want to obtain the current price of ARB in USD in Arbitrum One, so we need to know the proxy address that will provide that information. We will search the feed on the API3 Market and connect our wallet. We would then want to see if the feed is active, and if it is, we can check its configuration parameters, deploy the proxy contract and click onIntegrate. You can find the proxy address of ARB/USDhere .

info If a dAPI is already active, you can use the proxy address directly. If it is not active, you can activate it by clicking onActivate and following the instructions to deploy a proxy contract. We can now build the function to get the latest price of ARB. We’ll use this example contract:

contract

ARBPriceConsumer

{ /* *Network: Arbitrum One * Aggregator: ARB/USD * Proxy: 0x0cB281EC7DFB8497d07196Dc0f86D2eFD21066A5* / address

constant PROXY =

0x0cB281EC7DFB8497d07196Dc0f86D2eFD21066A5 ;

/* *Returns the latest price.* / function

getLatestPrice ( ) external view returns

(int224 value ,

uint256 timestamp) { (value , timestamp)

=

IProxy (PROXY) . read ( ) ; // If you have any assumptions about `value` and `timestamp`, make sure // to validate them right after reading from the proxy. } } You can adapt this contract to your needs. Just remember to use the address of the asset you want to request the price for in the appropriate network and todeploy your contract to the same network . Remember we have aQuickstart available that goes through the process of compiling and deploying a contract.

Querying a random number through API3

API3 QRNG is a public utility provided with the courtesy ofAustralian National University (ANU) . It is served as a public good, it is free of charge (apart from the gas costs), and it provides quantum randomness when requiring RNG on-chain.

To request randomness on-chain, the requester submits a request for a random number toAirnodeRrpV0 . The ANU Airnode gathers the request from theAirnodeRrpV0 protocol contract, retrieves the random number off-chain, and sends it back toAirnodeRrpV0 . Once received, it performs a callback to the requester with the random number.

Here’s an example of a basicQrngRequester that requests a random number.

API3 provides an npm package with the contracts needed to access the ANU QRNG airnode. We first install that package in our project:

yarn

add @api3/airnode-protocol We’ll need several pieces of data to request a random number:

- address airnodeRrp
- : Address of the protocol contract. See theChains
- page for a list of addresses on different chains. For Arbitrum, we’ll use0xb015ACeEdD478fc497A798Ab45fcED8BdEd08924
- .
- address airnode
- : The address that belongs to the Airnode that will be called to get the QRNG data via its endpoints. See theProviders
- page for a list of addresses on different chains. For Arbitrum we’ll use0x9d3C147cA16DB954873A498e0af5852AB39139f2
- .
- bytes32 endpointId
- : Endpoint ID known by the Airnode that will map to an API provider call (allowed to bebytes32(0)
-). You can also find that information in theProviders
- page. For Arbitrum we’ll use0xfb6d017bb87991b7495f563db3c8cf59ff87b09781947bb1e417006ad7f55a78
- .

- address sponsorWallet
- : The address of the wallet that will pay for the gas costs for the callback request to get the random number on-chain. You need to fund this wallet with enough ETH to cover the gas costs.

To derive your sponsorWallet address, you can use the following command:

yarn @api3/airnode-admin derive-sponsor-wallet-address \ --airnode-address 0x9d3C147cA16DB954873A498e0af5852AB39139f2 \ --airnode-xpub xpub6DXSDTZBd4aPVXnv6Q3SmnGUweFv6j24SK77W4qrSFuhGgi666awUiXakjXruUSCDQhhctVG7AQt67gMdaRAsDnDXv23bBRKsMWvRzo6kbf \ --sponsor-address < use-the-address-of-your-requester-contract

The command outputs.

Sponsor wallet address: 0x6394 .. .5906757

Use this address as the value for _sponsorWallet.

We can now build the function to get a random number. We'll use this example contract:

import

"@api3/airnode-protocol/contracts/rrp/requesters/RrpRequesterV0.sol" ;

contract

QrngRequester

is RrpRequesterV0 { event

RequestedUint256 (bytes32

indexed requestId) ; event

ReceivedUint256 (bytes32

indexed requestId ,

uint256 response) ;

/* * Network: Arbitrum One * AirnodeRrpV0 Address: 0xb015ACeEdD478fc497A798Ab45fcED8BdEd08924 * Airnode: 0x9d3C147cA16DB954873A498e0af5852AB39139f2 * Endpoint ID: 0xfb6d017bb87991b7495f563db3c8cf59ff87b09781947bb1e417006ad7f55a78 / address

constant _airnodeRrp =

0xb015ACeEdD478fc497A798Ab45fcED8BdEd08924 ; address

constant airnode =

0x9d3C147cA16DB954873A498e0af5852AB39139f2 ; bytes32

constant endpointIdUint256 =

0xfb6d017bb87991b7495f563db3c8cf59ff87b09781947bb1e417006ad7f55a78 ; mapping (bytes32

=>

bool)

public waitingFulfillment ; address sponsorWallet ;

constructor ( )

RrpRequesterV0 (_airnodeRrp)

{ }

// Set the sponsor wallet address that you just derived. function

setSponsorWallet (address _sponsorWallet)

external

{ sponsorWallet = _sponsorWallet ; }

function

makeRequestUint256 ( )

external

{ bytes32 requestId = airnodeRrp . makeFullRequest (airnode , endpointIdUint256 , address (this) , sponsorWallet , address (this) , this . fulfillUint256 . selector , "") ; waitingFulfillment [requestId]

=

true ; emit

RequestedUint256 (requestId) ; }

function

fulfillUint256 (bytes32 requestId ,

bytes

calldata data) external onlyAirnodeRrp { require (waitingFulfillment [requestId] , "Request ID not known") ; waitingFulfillment [requestId]

=

false ; uint256 qrngUint256 = abi . decode (data ,

(uint256)) ;

// Use qrngUint256 here...

emit

ReceivedUint256 (requestId , qrngUint256) ; } } You can adapt this contract to your needs. Just remember to set thesponsorWallet address before making the request to use the appropriate network's addresses, and todeploy your contract to the same network . Remember, we have aQuickstart available that goes through the process of compiling and deploying a contract.

More examples

Refer toAPI3's documentation for more examples of querying other data feeds and Airnodes.

You can also check out some other detailed guides:

- Subscribing to dAPIs
- Reading a dAPI Proxy
- Using QRNG with Remix Edit this page Last updatedonJan 27, 2025 Previous Oracles overview Next Chainlink