Architecture flow Oracle SDK

The Ethereum blockchain is designed to be entirely deterministic, meaning that if someone downloads the whole network history and replays it they should always end up with the same state. Determinism is necessary so that nodes can come to a consensus.

This service will query APIs, upon request by smart-contracts, and add the requested information to the blockchain. The described approach allows for having multiple competing parties deploying the same oracle and having the same say in the result. But can easily be converted into a simpler oracle to serve a single stakeholder.

Oracles are services that insert data on the blockchain to be used by smart contract. By adding a transaction with the required information to the blockchain the smart contract can run and always obtain the same information since it is retrieving it from the blockchain.

Solution

We gonna create an oracle service that can query JSON APIs and retrieve a single value from the API response. The oracle will save all the requests and answers and will have a predefined set of stakeholders.

These are the accounts running the node.js service that queries the APIs and returns a response to the oracle. The oracle also has a minimum number of equal responses that it must receive in order to confirm that the answer provided is valid.

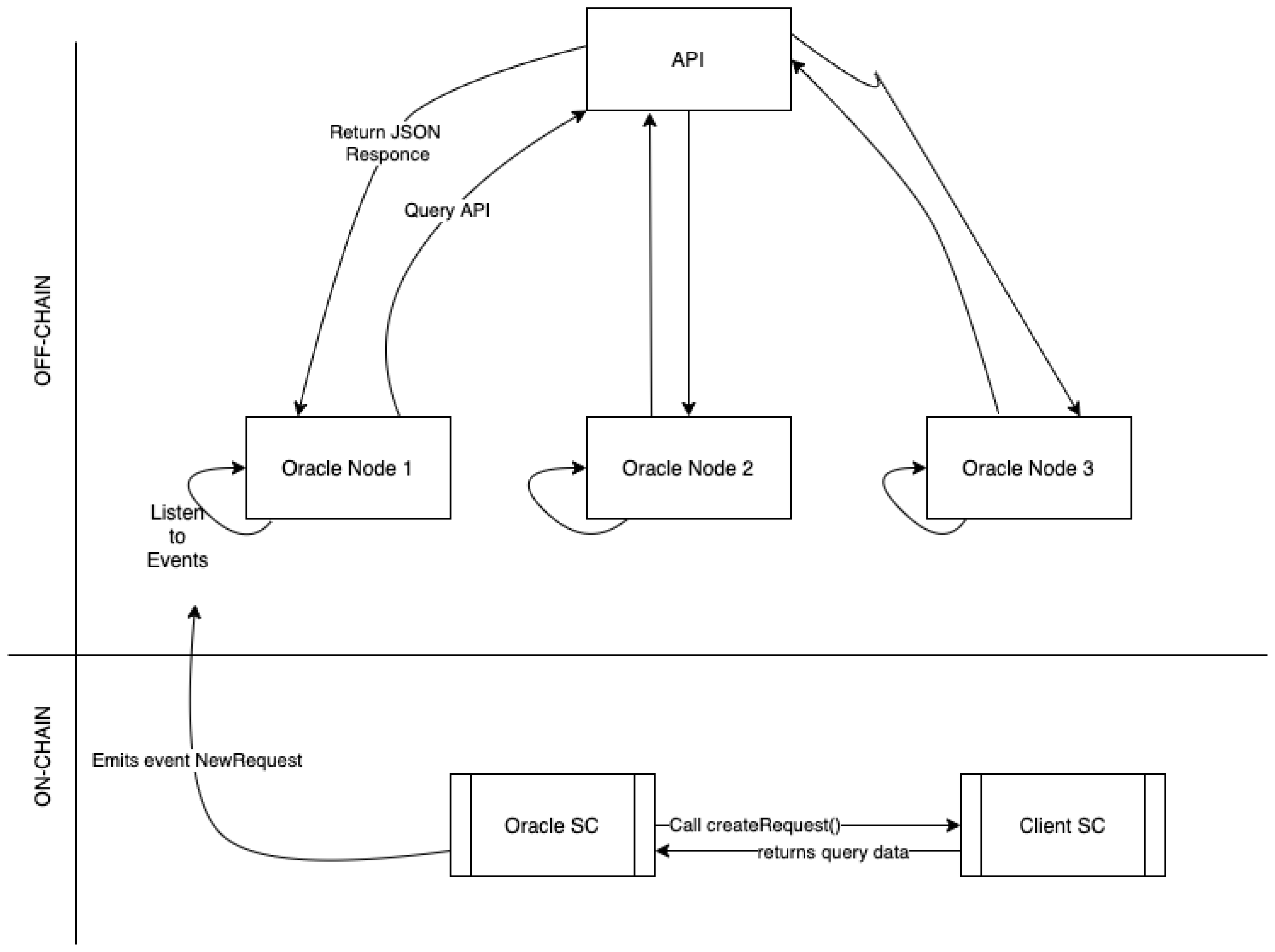
Architecture

This oracle will comprise two components. The on-chain oracle (a smart contract) and the off-chain oracle service (node.js server).

The on-chain oracle is a smart-contract that has a public function, **createRequest**, that receives the URL, to query, and the attribute to retrieve. And then, launches an event to fire the off-chain oracle of a new request.

The off-chain oracle is composed of several node.js services deployed by different parties that will query the API and return to the contract the response.

The on-chain oracle then verifies if the minimum number of equal responses has been reached and if so emits an event saying that it has achieved consensus on the value so that the client smart-contract that queried the oracle knows that it has its response.



On-chain Oracle Implementation

We define the oracle contract with the agreed terms: Minimum Quorum and Total Oracles. For this example, there are three stakeholders, and for achieving consensus 2 out of 3 must provide the same answer.

Then we add the Request Struct, which will hold the requests.

Now we can create the public function, **createRequest**, that client smart contracts (any contract that wants to use the oracle service) will call.

This function contains an important part of the agreement between the stakeholders. The addresses of the accounts that are trusted to take part in the final solution. And will emit the event, **NewRequest** that will be listened by the off-chain oracles.

Upon listening to this event the off-chain oracles will call the public function **updateRequest**.

This function will first check if the caller is one of the predefined addresses. Then it will check it the oracle hasn’t voted and if so it will save the oracle answer. Then it will check if that answer has been provided by at least the required minimum quorum. If so, then we have agreed on a result and will emit an event, **UpdatedRequest**, to alert the client contract of the result.

Off-chain Oracle Implementation

This is the simpler part, it’s any service that can listen to the emitted blockchain events and query APIs.

The off-chain oracle listens to the events emitted by the on-chain oracle, using web3, and queries the requested API, parses the retrieved JSON for the requested key and calls the public function **updateRequest**.Since it is not the focus of the post and it’s a simple service, I will not go further into the details of the implementation of the off-chain service.

Summing up

This approach allows to not depend on a single party to be the source of truth, by being the only one querying the API, but to have multiple parties agreeing on a result. It is also a very flexible service since it can query any public JSON API, allowing to be used in an any amount of use cases.