Article
Nikolay Solovyev · Feb 19, 2019 6m read

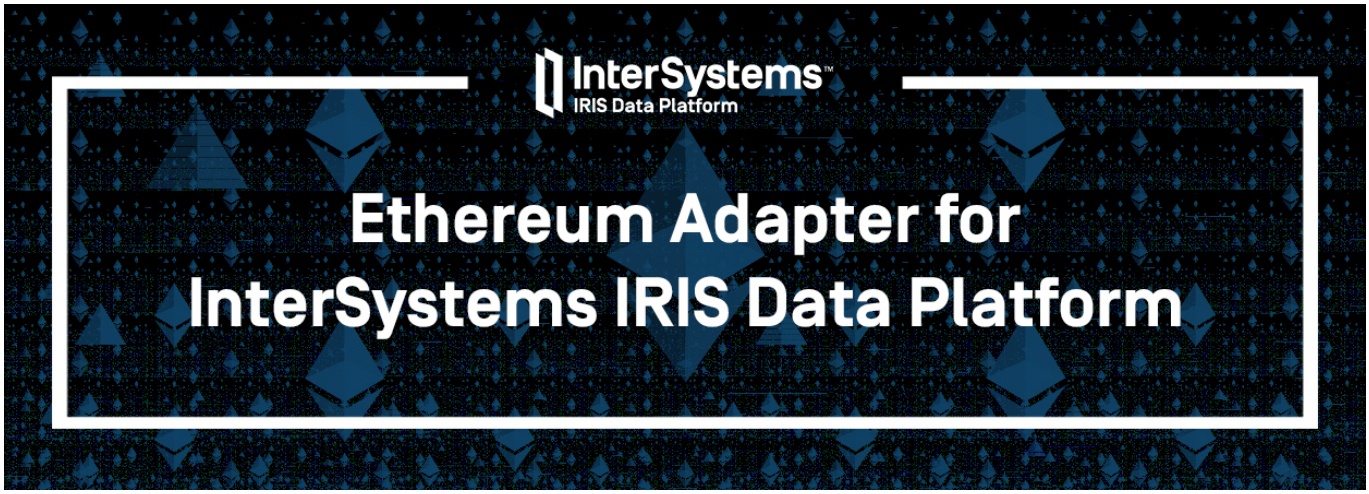
Open Exchange

Ethereum Adapter for InterSystems IRIS Data Platform

1. Blockchain

As I am writing this article, Bitcoin costs less than one-fifth of what it used to be at the pinnacle of its success. So when I start telling someone about my blockchain experience, the first thing I hear is undisguised skepticism: "who needs this blockchain stuff now anyway?"

That's right, the blockchain hype has waned. However, the technologies it is based on are here to stay and will continue being used in particular areas.The Internet in general offer tons of materials describing the general usage of these technologies



(for example on medium and forbes).

As we know, a blockchain is a distributed registry, i.e. a database distributed between several nodes with each node storing a full copy of the registry. The key feature of a blockchain is that records (transactions) form blocks and blocks form a chain of blocks. A blockchain supports append operations only. It means that it's almost impossible to make changes to transactions already saved in the blockchain.

There are countless blockchain tutorials online (if you've never heard about blockchain, you can start with this simple video).

When blockchain was on a roll, there were multiple calls to use the technology literally everywhere. However, there are certain distinct characteristics of projects/tasks where blockchain may be required.

First of all, there have to be many players/users writing a lot of data that must be consistent and trusted.

Then, there is not supposed to be a third party that everybody trusts.

There has to be a mechanism for public data validation.
If all of these criteria are met, it may be a good idea to consider using a blockchain.

Such tasks can be found in any industry. The www.101blockchains.com project aggregates information about potential and existing blockchain projects, as well as the nuances of using blockchain technology in various industries.

For example, blockchain can be used for the following tasks in the healthcare domain:

- for safe remote management of patient records;
- for combating counterfeit drugs through unchangeable transactions across the supply chain;
- for improving the monitoring and effectiveness of clinical trials by ruling out the possibility of fraud and tampering with data.

The corporate segment typically uses a special type of blockchain called Private Permissioned Blockchain. Such networks have a special set of nodes for verifying transactions.

However, while developing the first InterSystems IRIS blockchain adapter, we chose Ethereum, a type of blockchain that belongs to the Permissionless Blockchain category - an open platform without a single control center. The decision was based on the popularity of this blockchain engine and a sufficiently mature infrastructure with lots of tools and libraries. Note that you can also create a private blockchain using Ethereum tools.

2. Adapter

Let's actually go back to the adapter.

An adapter in InterSystems IRIS (just like in Ensemble) is a class or package of InterSystems IRIS classes allowing you to interact with an external system. InterSystems IRIS adapters are divided into inbound (for receiving data from an external system when the external system is the initiator of the interaction) and outbound (for working with an external system when InterSystems IRIS is the initiator of the interaction).

The IRIS Ethereum adapter is an outbound adapter and is slightly different from most other InterSystems IRIS adapters. This adapter also includes a small NodeJS module. The architecture is shown in Figure 1.

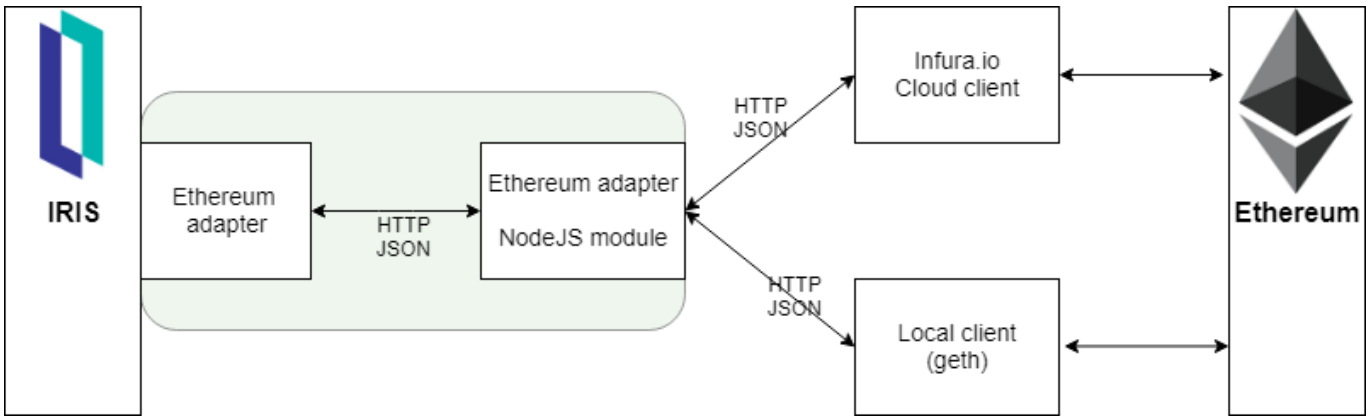


Figure 1.

The NodeJS module of the adapter uses existing NodeJS libraries for working with Ethereum.

The adapter allows you to do the following:

- Deploy a smart contract to Ethereum (we have plans for another article covering smart contracts, development tools, and an example).

- Call smart contract methods: those that change the state of the blockchain and those that don't
- Save transactions (transfer funds from one wallet to another)
- Call additional methods to get the status of the blockchain
- Log all requests (done by the NodeJS module, comes in handy for debugging)

The adapter comes with the source codes on OpenExchange.

3. A simple example

The adapter is supplied with a "Hello world" example.

To start working with Ethereum (and running this example), you will need the following:

- Choose the network that you are going to be working with. Test networks like Ropsten are typically used for development purposes
- Create a wallet in this network and make a deposit to it
- Install a local Ethereum client (e. g. Geth) or get a key for working with a cloud provider (e. g. Infura)

You need to set the following while configuring a business operation (figure 2):

1. The server and the port that the NodeJS module is working on (port 3000 is used by default)
2. Provider settings (access to Infura in this case)
3. Access credentials (specify your wallet number as user name and your private key as password. InterSystems IRIS stores access credentials in a separate database that you have to enable encryption for)

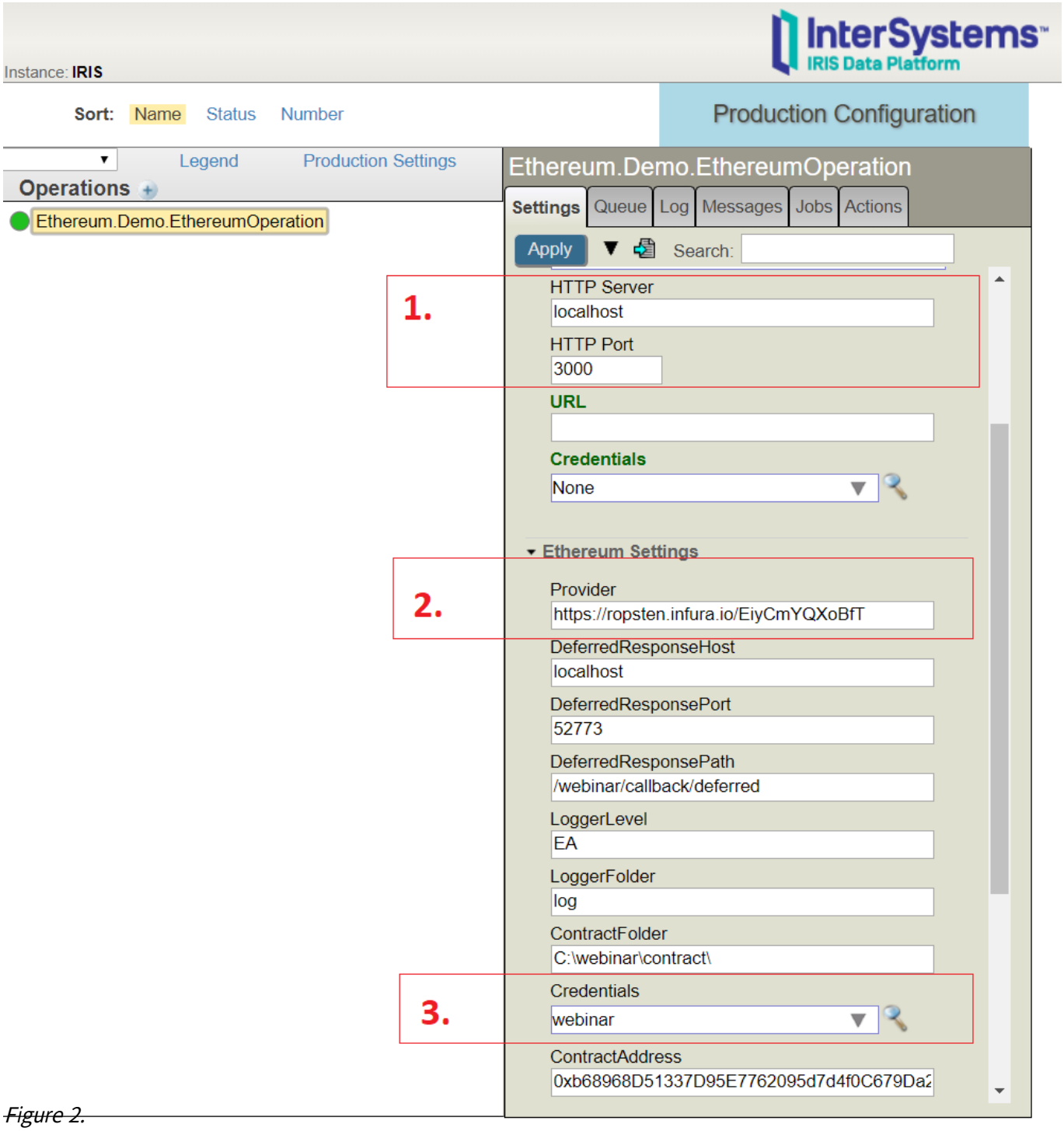


Figure 2.

In order to work with smart contracts, you will need to create (for each smart contract you will work with) a folder in your file system and place two files there:
*abi.txt
*bytecode.txt

These files should contain the ABI of the smart contract and its Bytecode. A smart contract's ABI is a formal description of its interface in the JSON format. ABI and Bytecode are created when the smart contract is compiled.

Bytecode is only required for deploying the contract.

You can use InterSystems IRIS Interoperability testing service to test business operations.

Figure 3 illustrates how a smart contract is deployed using the testing service. The result of calling this business operation is a message containing the hash of the transaction.

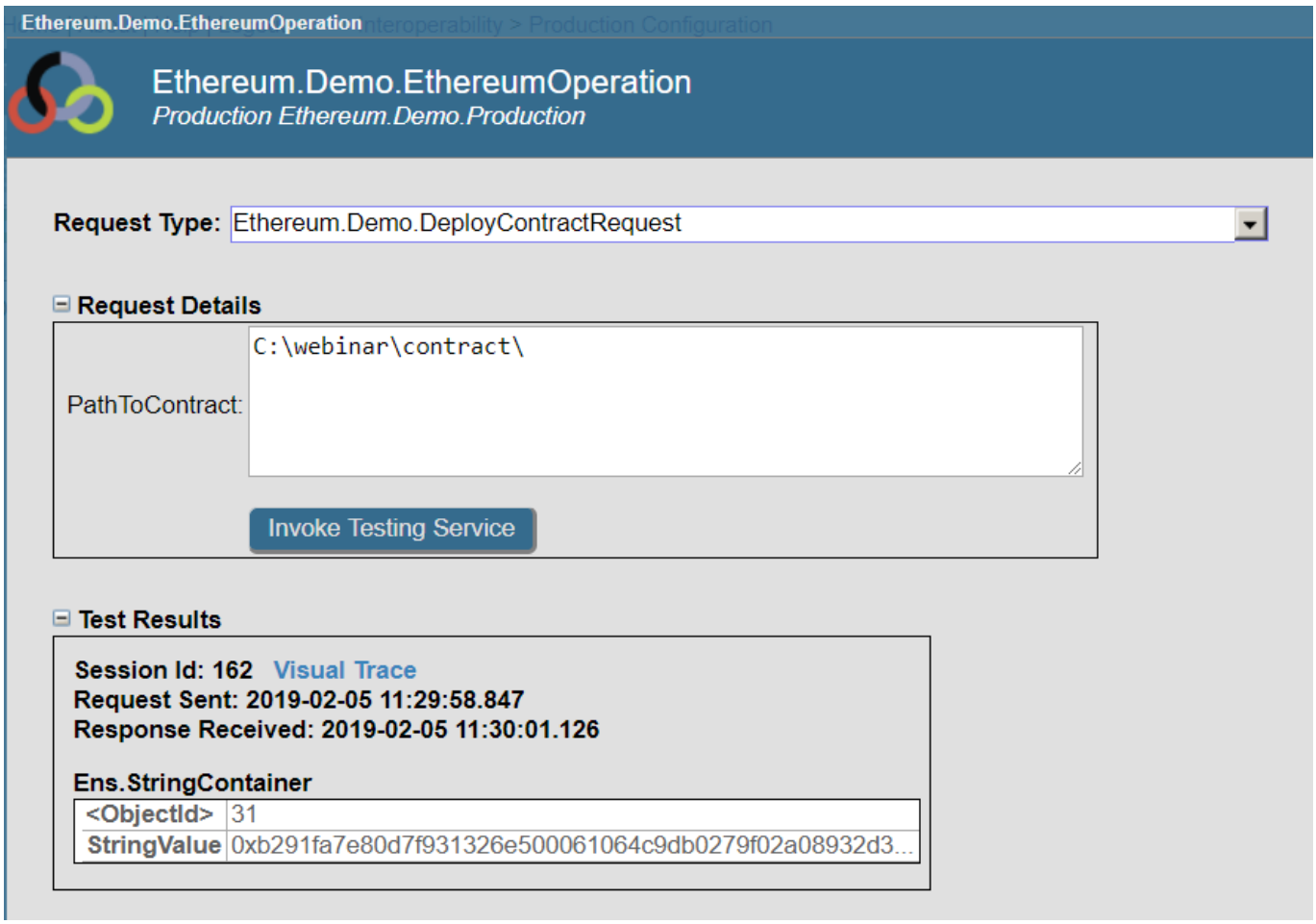


Figure 3.

You can find this transaction using the ropsten.etherscan.io (https://etherscan.io/) browser and get the address of the deployed smart contract.

To call the methods of the smart contract using the adapter, you need to fill out the following fields in the production configuration: ContractFolder and ContractAddress.

The execution code for the smart contract is pretty simple:

```
set ..ContractABI = [].%FromJSON(..ContractFolder_"abi.txt")
set contract = ..Adapter.GetContract(##class(Ethereum.Address).%New(..ContractAddress
),..ContractABI)
set result = contract.hello()
set pResponse = ##class(Ens.StringContainer).%New(result.args)
```

Pass the address of the smart contract and the ABI to the GetContract method of the adapter to create a smart contract object that you will then use for calling methods. In this case, a hello() method returning a string must be defined in the smart contract.

In this example, the hello() method does not change the blockchain status, so it can be called synchronously. However, the execution time of methods that change the state of the blockchain can be quite long (due to having to wait for transactions to be verified).

To call such methods, use the deferred response mechanism offered by InterSystems IRIS. The adapter will have to submit a deferred response token and when a transaction is approved, the NodeJS module will pass the result of its execution to InterSystems IRIS. To do that, you will need to configure a web application and add an extra business service to the production that will process the received response.

Here is the code for calling a method that alters the blockchain state:

```
// getting EthereumAccount(wallet) and privateKey
do ##class(Ens.Config.Credentials).GetCredentialsObj(.cred, "Ethereum.Demo.EthereumOp
eration", "Ens.Config.Credentials", ..Credentials)
set account = ##class(Ethereum.Address).%New(cred.Username)
set privateKey = cred.Password

//reading contract ABI
set ..ContractABI = [].%FromJSON(..ContractFolder_"abi.txt")

// get contract object
set contract = ..Adapter.GetContract(##class(Ethereum.Address).%New(..ContractAddress
),..ContractABI)
$$$ThrowOnError(..DeferResponse(.deferred))

// estimate gas
do contract.SetOptions(account)
set gasJSON = contract.EstimateGas("setName",pRequest.Name)
$$$TRACE(gasJSON.gas)

do contract.SetOptions(account, privateKey,  ##class(Ethereum.Wei).%New(1000000000),
##class(Ethereum.Wei).%New(100*gasJSON.gas),,deferred)

set result = contract.setName(pRequest.Name)
```

In this case, prior to calling the setName() method of the smart contract, you will need to specify a number of parameters, including the deferred response token.

In our next article, we will elaborate on smart contracts and provide an example of solving a practical problem using the InterSystems IRIS Ethereum adapter.

#Interoperability #Ensemble #InterSystems IRIS
Check the related application on InterSystems Open Exchange