SMART-CONTRACT-AS-A-SERVICE

INTRODUCTION

SIMBA Chain Smart Contract as a Service (SCaaS) provides a tailored interface to your blockchain, defined by the smart contract you have designed at simbachain.com.

SIMBA
SCaaS

OVERVIEW

SIMBA Chain Smart Contract as a Service (SCaaS) provides a tailored interface to your Blockchain, defined by the smart contract you have designed at simbachain.com. SCaaS provides two main services related to a smart contract:

1. A **REST API** that models the methods and arguments in your smart contract to bring your business process endpoints to the enterprise. A simple POST to the API will result in a transaction on the Azure Blockchain.
2. The **Explorer Interface** to query for transactions on the block chain. Browse, search and view the details of all of your Blockchain transactions for each Smart Contract using the SCaaS Explorer.

It builds on top of that platform and allow you to easily create smart contracts.

ABOUT SIMBA CHAIN

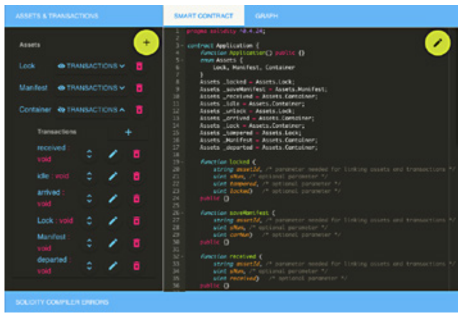
SIMBA stand for Simpler Blockchain Applications. The SIMBA Platform makes it easy to build, deploy and interface with Blockchain applications, democratizing the Blockchain.

BLOCKCHAIN

For the Blockchain layer, SIMBA provides a generic API to multiple Blockchain systems, thus the system does not have a dependency on a single Blockchain or DLT implementation. Currently, the platform supports Ethereum, Quorum and Stellar but several more are on the roadmap.

SMART CONTRACTS

Smart Contracts provide the interface, and business logic, to what is written on the Blockchain and what rules need to be satisfied for this write operation to take place. In SIMBA Chain, Smart Contracts are automatically generated from conceptual models that define the Assets and Transactions that transact on those Assets. Such models are specified using SIMBA's Web App's UI shown on the right. A user simply uses the GUI to add Asset or Transactions along with their methods and parameters, and SIMBA Chain automatically generates the smart contract for the platform they select (in this case, Solidity code for Ethereum). It also generates a graph of the relationships for the model as shown. The resulting smart contract, once deployed on the blockchain, is dynamically exposed as an application REST interface for simpler external application interaction with the Blockchain.



```
pragma solidity ^0.4.24;

contract Application {
    function Application() public {}
    enum Assets {
        Lock, Manifest, Container
    }
    Assets _locked = Assets.Lock;
    Assets _saveManifest = Assets.Manifest;
    Assets _received = Assets.Container;
    Assets _idle = Assets.Container;
    Assets _unlock = Assets.Lock;
    Assets _arrived = Assets.Container;
    Assets _Lock = Assets.Container;
    Assets _tampered = Assets.Lock;
    Assets _Manifest = Assets.Container;
    Assets _departed = Assets.Container;

    function locked (
        string assetId, /* parameter needed for linking assets and transactions */
        uint sNum, /* optional parameter */
        uint tampered, /* optional parameter */
        uint locked)   /* optional parameter */
    public {}

    function saveManifest (
        string assetId, /* parameter needed for linking assets and transactions */
        uint sNum, /* optional parameter */
        uint cartNum)   /* optional parameter */
    public {}

    function received (
        string assetId, /* parameter needed for linking assets and transactions */
        uint sNum, /* optional parameter */
        uint received)   /* optional parameter */
    public {}
```

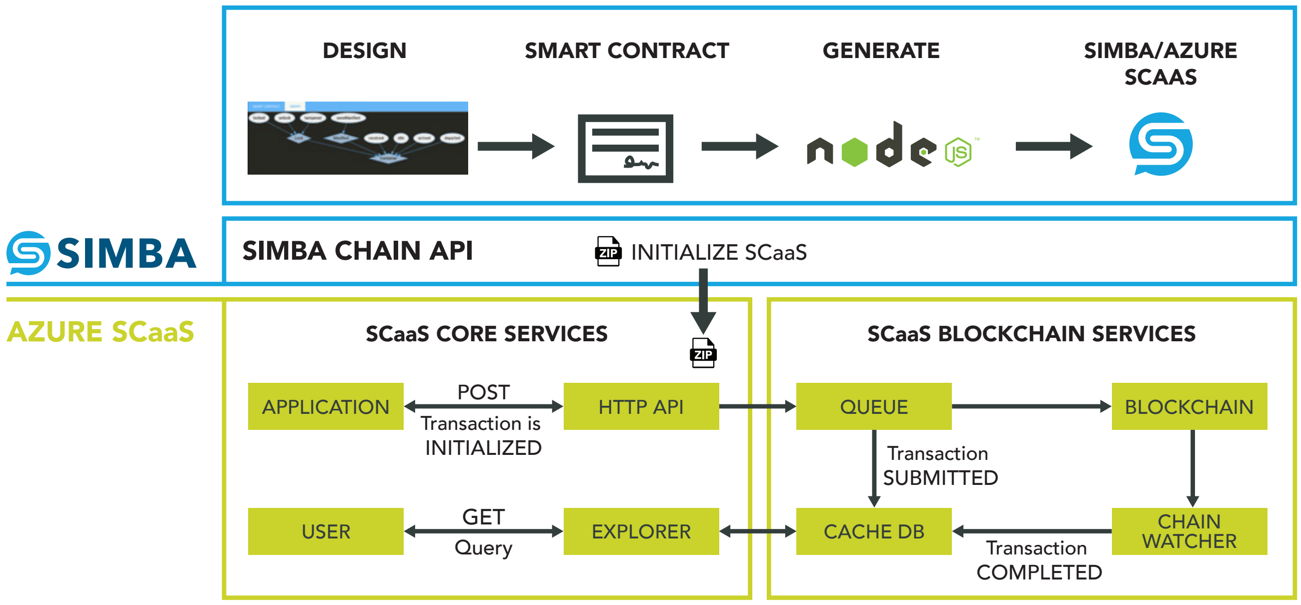
DATA STORES

Data Stores in SIMBA Chain uses the same adapter pattern as the Blockchain one, that is, a single generic REST interface can support the simple integration of different data stores. Currently SIMBA supports the Ceph, IPFS and flat file, system-based data stores. As shown, data flows into the system through the application's REST API (generated by SIMBA) by attaching one or multiple files to the transaction. This is achieved by using a simple multipart form post. Transactions are then checked for access before being passed to the data bundling mechanism, which stores all files into the Data Store and collects each hashcode into a JSON manifest file which is bundled along with the files. and stored into the Data Store with its hashcode stored onto the blockchain. Using this mechanism, the system can easily retrieve all files using the hashcode while the hash also serves as a digest to guarantee the integrity of the data.

General documentation on SIMBA Chain is available at
simbachain.com/documentation

SIMBA SMART CONTRACT-AS-A-SERVICE (SCaaS)

SCaaS allows you to export applications: we auto-generate source code that binds to the specific Blockchain system and Data Store configured for a Smart Contract. This contract and API can then be deployed to your enterprise environment. The diagram below shows how it works.



Azure SCaaS leverages the built-in capabilities of Azure including Azure Active Directory (AD) for authentication and authorization, Azure Key Vault for maintaining blockchain addresses or associated private keys, Azure's Service Bus Queue in order to create a robust mechanism for pushing data to the blockchain, Azure's blob store for off-chain storage and Azure managed instance of Postgres to store transaction data, allowing fast query on the contents of the blockchain. Finally, SCaaS leverages Azure's blockchain service. Currently this is the only blockchain supported, however, SIMBA supports multiple blockchains and SCaaS will in future allow a variety of possible blockchain backends.

HTTP API

The SCaaS HTTP API provides both POST and GET methods to push transactions and query for transactions. These methods are linked to the method in the smart contract. The POST is the primary API to use as Explorer (see below) exposes the GET API in addition to a general recent transactions view across all smart contract methods.

A transaction sent to the blockchain goes through a number of states. When you POST a transaction, and no errors occur, you receive a response that contains the unique requestId and a state of 'INITIALIZED'. If the request fails, you will see a state of 'FAILED'.

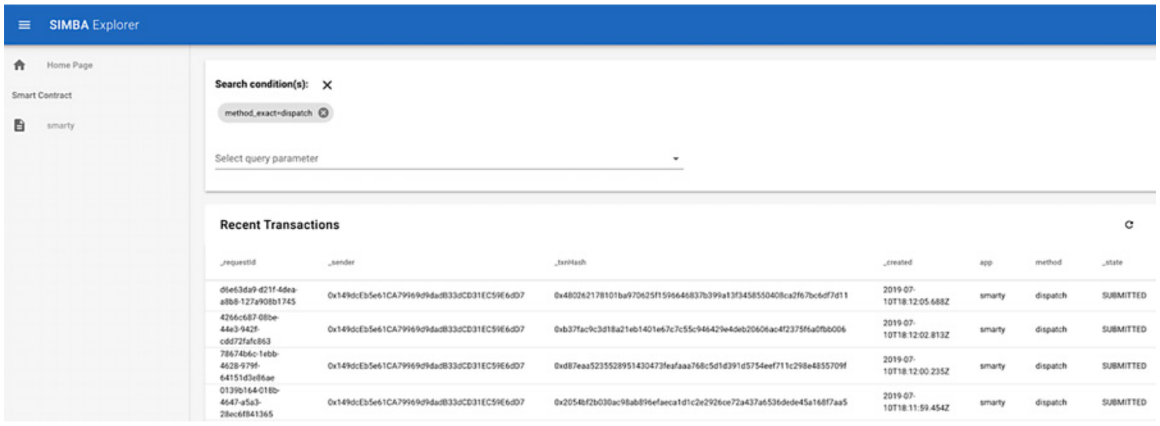
Then, the transaction is submitted to the blockchain, which results in a SUBMITTED state. Finally, once the transaction is embedded in the blockchain and will not be revoked, the transaction enters a COMPLETED state and is cached for query.

EXPLORER

Explorer provides an interface to querying the blockchain. There are two primary views:

1. **RECENT TRANSACTIONS** – these are all transactions you have created on the blockchain. Some may not have details available yet, especially the most recent.
2. **SMART CONTRACT METHOD VIEW** –This view shows the transactions relating to a particular smart contract method. These transactions have been verified on the blockchain.

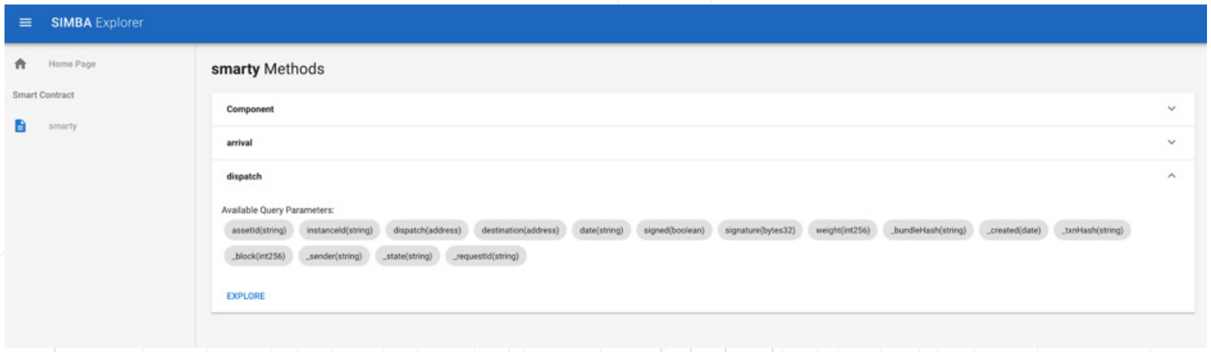
Recent Transactions shows submitted and completed transactions for all methods of your smart contract. You can query on the various common fields that all transactions have such as request ID, the sender and the transaction hash.



Recent Transactions

_requestId	_sender	_txnHash	_created	app	method	_state
d6e63da9-d21f-4dea-a8b8-127a908b1745	0x149dcEb5e61CA79969d9dadB33dCD31EC59E6dD7	0x480262178101ba970625f1596646837b399a13f3458550408ca2f67bc6df7d11	2019-07-10T18:12:05.688Z	smarty	dispatch	SUBMITTED
4266c687-08be-44e3-942f-cdd72fafc863	0x149dcEb5e61CA79969d9dadB33dCD31EC59E6dD7	0xb37fac9c3d18a21eb1401e67c7c55c946429e4deb20606ac4f2375f6a0fbb006	2019-07-10T18:12:02.813Z	smarty	dispatch	SUBMITTED
78674b6c-1ebb-4628-979f-64151d3e86ae	0x149dcEb5e61CA79969d9dadB33dCD31EC59E6dD7	0xd87eaa523552895143047 3feafaaa768c5d1d391d5754eef711c298e4855709f	2019-07-10T18:12:00.235Z	smarty	dispatch	SUBMITTED
0139b164-018b-4647-a5a3-28ec6f841365	0x149dcEb5e61CA79969d9dadB33dCD31EC59E6dD7	0x2054bf2b030ac98ab896efaeca1d1c2e2926ce72a437a6536dede45a168f7aa5	2019-07-10T18:11:59.454Z	smarty	dispatch	SUBMITTED

The Contract Method view allows you to view completed transactions. Here you can view the details of transactions by method.



Transactions that accept files also have two buttons on the left side of the row. These can be used to download the bundle from Azure's Blob Storage and to view the JSON manifest describing the contents of the files associated with the transaction.