SIMBA
SCaaS

SMART-CONTRACT-AS-A-SERVICE

USER GUIDE

SIMBA Smart-Contract-as-a-Service (SCaaS) provides a tailored interface to your blockchain, defined by the smart contracts you have designed.

OVERVIEW

SIMBA Smart-Contract-as-a-Service (SCaaS) provides a tailored interface to your blockchain, defined by the smart contracts you have designed. SCaaS provides two main services related to a smart contract:

1. A **REST API** that models the methods and arguments in your smart contracts 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 blockchain. Browse, search and view the details of all of your Blockchain transactions for each smart contract using the SCaaS Explorer.

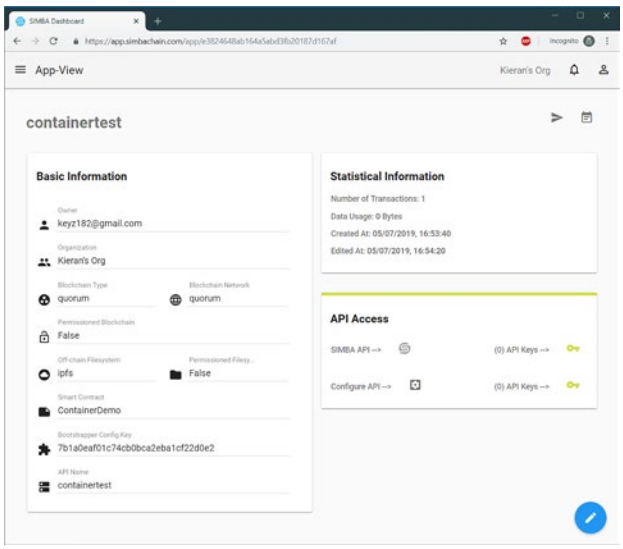
This document describes how to create the contracts and how to deploy them to your Azure environment.

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

CONTRACT DESIGN

The first step to getting going is to design an Ethereum Smart Contract at SIMBA Chain. Sign up to simbachain.com and create a contract and SIMBA App using the simple UI. Detailed documentation on how to do it is provided at https://simbachain.com/documentation/.

Once you have this created your contract and app, you can retrieve your Bootstrapper Config Key. Keep this handy; it'll be needed later.

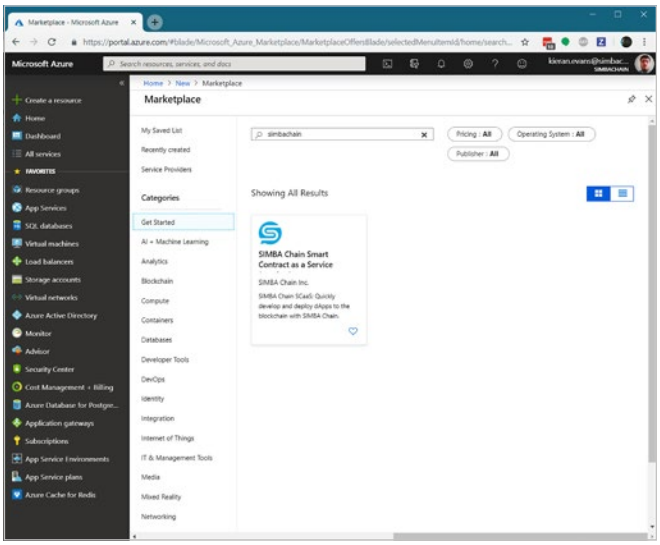


Once you have your Bootstrapper Config key, you are ready to set up your Azure environment.

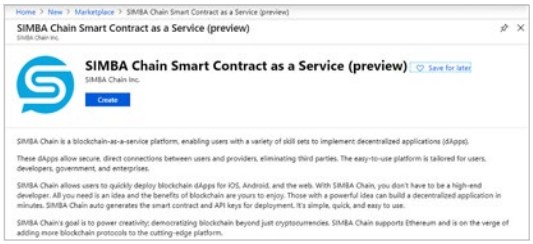
NOTE: Do not start your method parameter names with underscores, e.g. '_my_param'. Underscores are reserved for SCaaS usage, to ensure there are no clashes between system properties and user defined properties. A parameter name of 'my_param' is fine.

DEPLOYING FROM AZURE MARKETPLACE

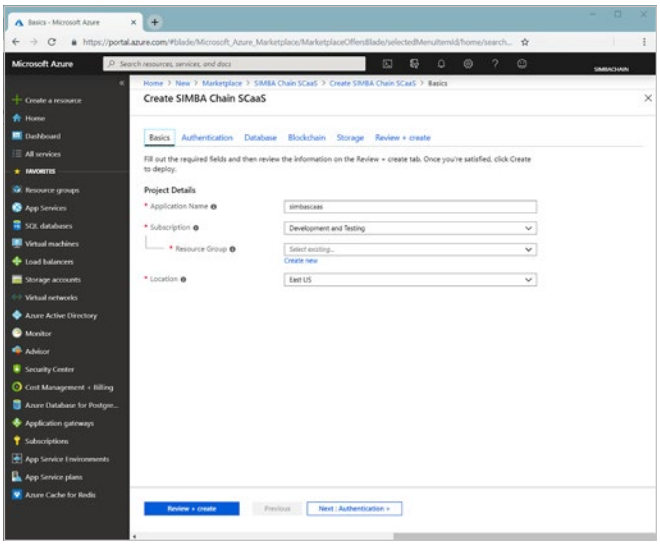
Search for SIMBAChain in the marketplace, then click on the SIMBA Chain Smart-Contract-as-a-Service entry.



On the next screen, click create.



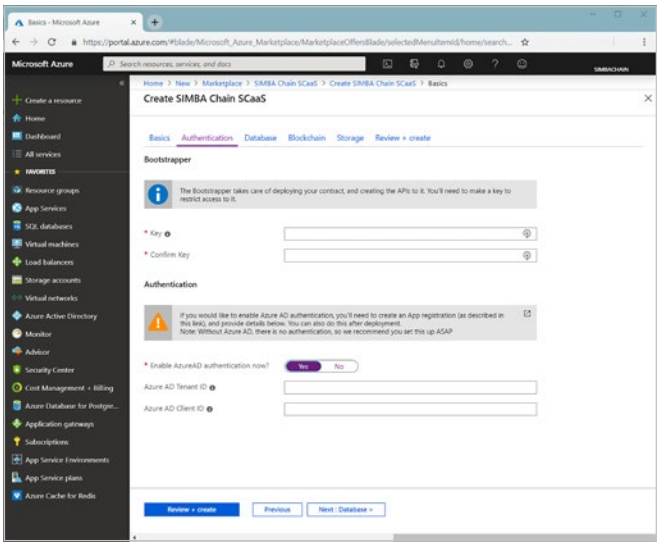
Choose the subscription you wish to deploy to, select the location to deploy to, and create a new resource group to deploy to. Choose an application name (note that this will be the name assigned to your API endpoint, e.g. https://yourappname.azurewebsites.net).



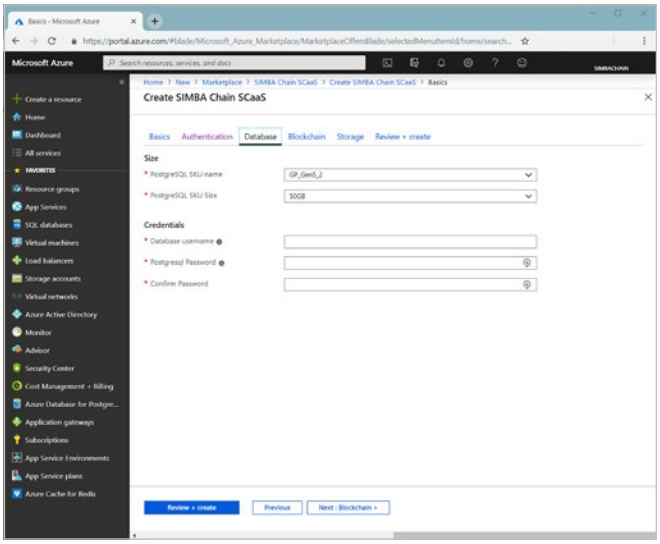
Create a Bootstrapper Key - this key is used to protect access to the bootstrapper.

Configure Active Directory auth. We recommend you select yes, and activate AD auth now, as otherwise, your API will have no authentication. Follow this guide to set up an Azure AD App Registration in order to obtain the Tenant ID and Client ID - *https://docs.microsoft.com/en-us/azure/active-directory/develop/quickstart-register-app*

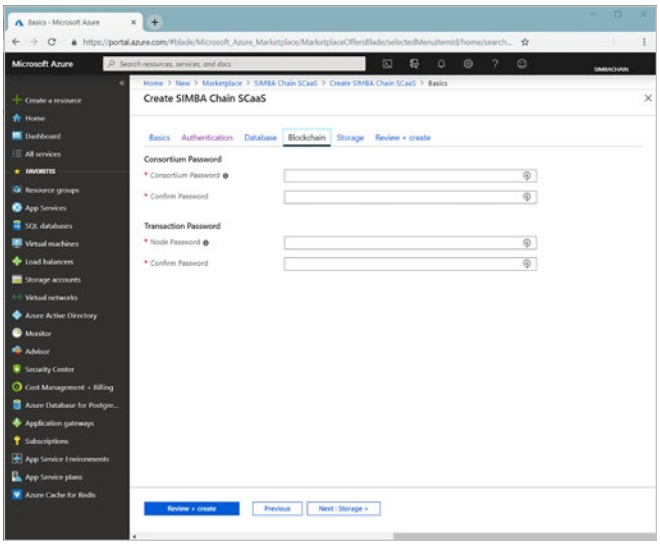
For the Redirect URI, use *https://YOURAPPNAME.azurewebsites.net/.auth/login/done*



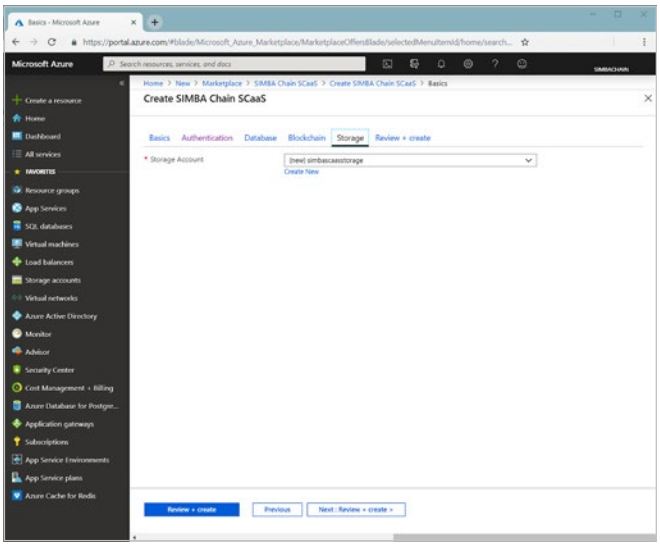
Choose your database SKU and Size, and set credentials for access.



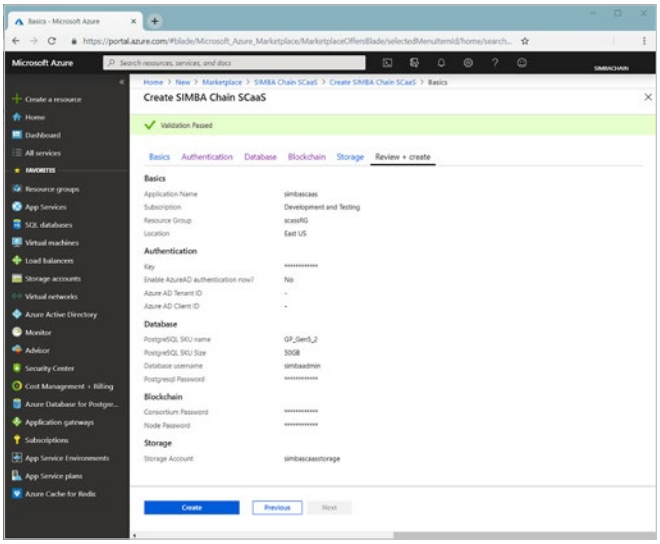
Generate Consortium and Transaction passwords for your Azure Blockchain Service based Quorum deployment. These are not currently used in SCaaS, but are required nonetheless for an Azure Blockchain Service deployment.



Create or select your storage account.

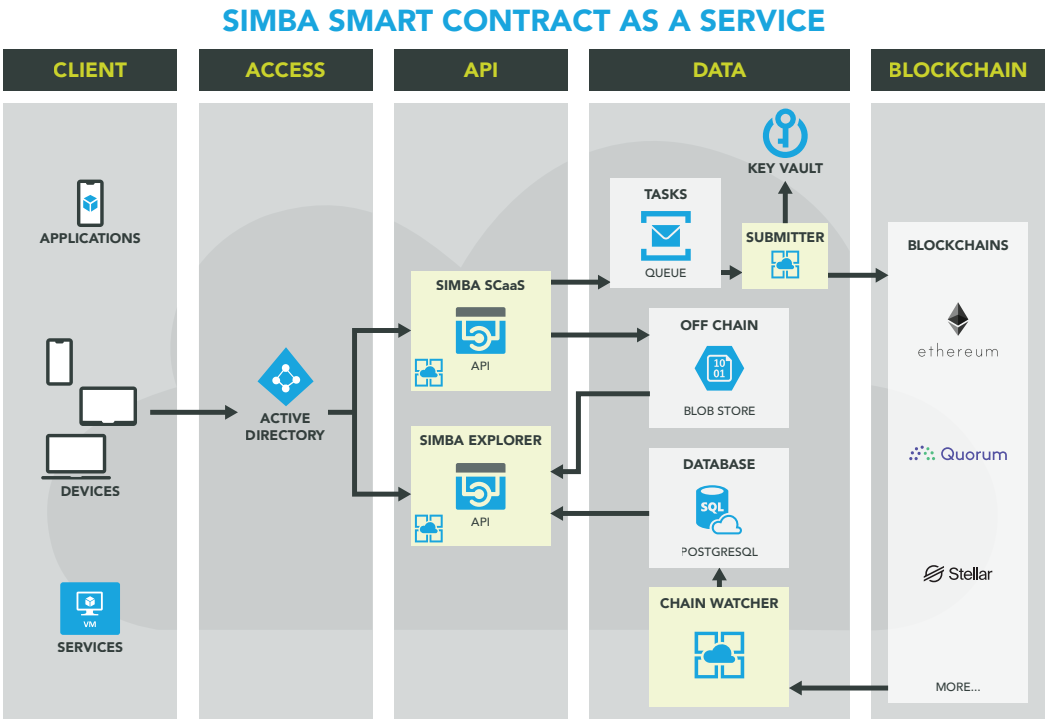


Finally, confirm the details
are correct, and press create.



SERVICES

The figure below shows the architecture of your deployed SCaaS instance. As you can see, SCaaS leverages many of the built-in features of Azure helping to make your experience as seamless as possible.



SCaaS leverages Azure Active Directory (AD) for authentication and authorization. This provides a single point of access control that you can integrate easily with other existing deployed services, as well as providing you with full control over access to SCaaS services. Additionally SCaaS integrates with Azure Key Vault meaning you don't have to worry about creating or maintaining blockchain addresses or associated private keys. Instead, SCaaS creates a new blockchain address and private signing key for each user that accesses the HTTP API and reuses those details for each request they make. The secret blockchain address and private key are mapped to the user's AD Object ID in Key Vault. This also means that you can easily track who has created which transactions on the blockchain by matching up the user's object ID in AD to the key in Key Vault.

SCaaS uses Azure's Service Bus Queue in order to create a robust mechanism for pushing data to the blockchain. Messages are pulled from the queue before pushing to the blockchain.

For off-chain data, SCaaS leverages Azure's blob store. This allows you to store large files associated with transactions outside of the blockchain itself and referenced by a hash value on the blockchain, providing a pointer to the data and ensuring data integrity via the hash value. When files are received via the HTTP API, they are zipped up and stored in the blob store. Along with the files, the zip file contains a manifest giving the hash of each individual file, the hash algorithm used (currently SHA256), and the file's mime type and size. The filename of the zip itself is the hash of the zip file. The manifest also describes the algorithm for this hashing (currently also SHA256), allowing you to validate the zip and its contents for integrity.

SCaaS uses an Azure managed instance of Postgres to store transaction data, allowing fast query on the contents of the blockchain. The database is populated by the chain watcher component which checks for blocks being created on the blockchain and captures transactions on your smart contract that have been verified by the blockchain.

Finally, SCaaS leverages Azure's blockchain service. Currently this is the only blockchain supported, however, SIMBA supports multiple blockchains and SCaaS will allow a variety of possible blockchain backends in the future.

The pale yellow boxes in the diagram are the SIMBA specific components that have been deployed to Azure Web Apps. Specifically, you interact with the Bootstrapper Service initially to pass in your export key and to trigger the deployment of your smart contract and the services used by it. You use the SIMBA SCaaS HTTP API to send data to the blockchain and you use SIMBA Explorer to query for transactions sent to the blockchain. The following sections describe these services in more detail.

AUTHENTICATION

Before we describe the individual services, we should talk about authentication and authorization.

Your SCaaS deployment should be integrated with your Azure Active Directory (AD). If it is not, you should add AD protection. As described above, SCaaS leverages AD in order to simplify access control for you. When accessing the user interfaces of SCaaS, the user will be redirected to the Azure login page. From here, they will use an OAuth 2 token to access the SCaaS services.

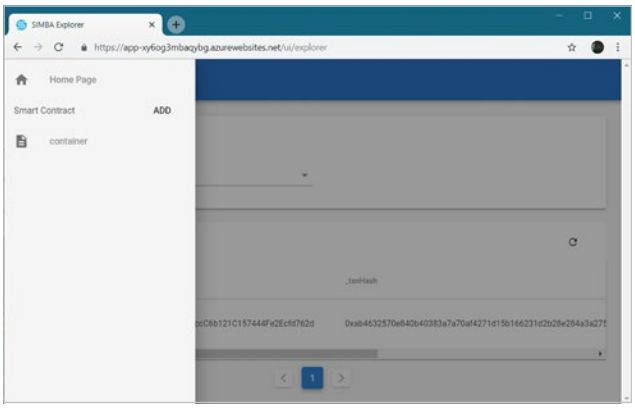
To access the HTTP API from your enterprise software, you should implement OAuth 2 implicit flow authentication. More details on this can be found here *https://docs.microsoft.com/en-us/azure/active-directory/develop/v2-overview*, for a variety of programming languages.

Note that the Swagger HTTP UI expects the OAuth 2 token in the Authorization header. This token should have the form 'Bearer: <base64 encoded token received from the authentication server>'.

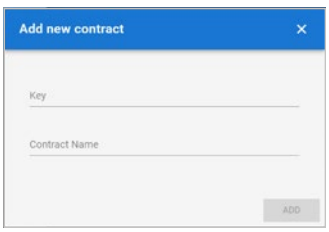
DEPLOYING YOUR SMART CONTRACT

Navigate to your newly deployed app. If you are unsure where this is, look at the resource group in Azure that you deployed SCaaS to, and click on the App Service Web App. It will provide details, including the URL.

Once you've navigated to the running app, scroll down and click the link to "SIMBA Explorer". You'll be presented with the screen below (if the menu isn't visible, click the menu icon on the top left).



Click on the "ADD" button to see the Add new contract dialog. Enter your config key from *https://app.simbachain.com* and a name for the contract. This will trigger the deployment of your smart contract API, customised to the contract's method signatures. Once complete, your new contract will appear in the list on the left.



You can add multiple contracts here, which will be processed and access provided via API.

HTTP API

The 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.

This api can be found at *https://<my-scaas-url>/v1-docs*

The POST API receives multipart/form-data encoded payloads. The payload should be the parameters used to execute the smart contract method. Even if you do not intend to use the Swagger HTTP UI for POSTing data, it is useful to examine the expected inputs to an HTTP call. The Solidity types you defined in your contract may not map directly to the types understood by the HTTP API. Specifically, types like Solidity bytes should be encoded as strings. As a rule, a string representation of bytes should be hex encoded which means the string should be '0x' followed by twice the number of characters defined by the bytes type. For example, if a type is defined as 'bytes32' then the string representation would be 64 characters long prepended with the hex prefix '0x'.

Solidity address types should likewise be encoded as hex strings.

If you defined file uploads as part of your smart contract method, then the '_bundleHash' parameter will be present. This does not need to be filled in if you are POSTing files. It is used to identify the method as allowing file uploads and used for querying a previously uploaded file.

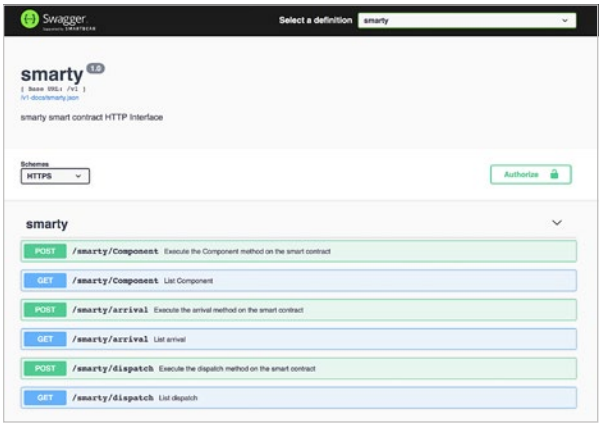
A transaction sent to the blockchain goes through a number of states. When you POST a transaction, and no errors occur, you receive a response that contains the unique requestId and a state of 'INITIALIZED'. If the request fails, you will see a state of 'FAILED'.

At this point the transaction has not been sent to the blockchain yet. Instead, it is put into the submission queue and picked up asynchronously by the SCaaS submitter that submits the method call to the blockchain. Once this submission has taken place, the state of the request moves to 'SUBMITTED'. This can be seen in Explorer's recent transactions view. Transactions that have been submitted are now visible to query from recent transactions. At this point the full transaction is not available.

Once the chain watcher has checked that the transaction has been confirmed and is embedded in the blockchain, it updates the state to 'COMPLETED' and the full details of the transaction become available, including the inputs to the method which are the parameters you are most interested in. If an error occurs while reading the transaction from the blockchain, then the state will be updated to 'FAILED'.

From the home page of your SCaaS deployment, you have a link to a Swagger UI of the HTTP API. This allows you to manually test out your API, although typically the API is used to integrate directly with your enterprise software stack.

This screenshot shows an example page for a contract called Smarty. It has three methods and the API supports both POST – i.e., send data to the blockchain, and GET, i.e. query for data from the blockchain. The Explorer UI uses this API for showing query results.



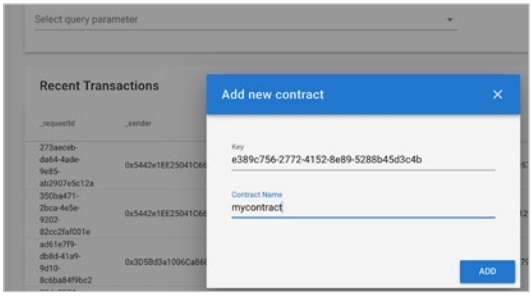
The UI shows you the input parameters for POSTs and query options for GETs.

At the top right of the page is a drop down menu for you to select the contract you want to see the API for.

When using the Swagger UI, you have already logged into Azure AD so you do not need to add Authorization headers.

EXPLORER

Explorer allows you to add new contracts to your SCaaS deployment. Once you have a deployment key from simbachain.com, you can register the contract with SCaaS. Click on the 'ADD' link above the left hand navigation bar. A dialog will display as shown below. Paste your key in and give the contract a unique name. Names should only contain letters, no spaces and be lowercase.



The newly deployed contract will appear in the navigation bar on the left.

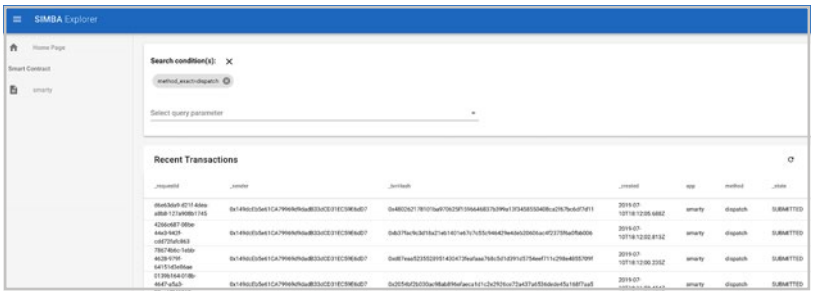
Explorer provides an interface to querying the blockchain. There are two primary views:

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.

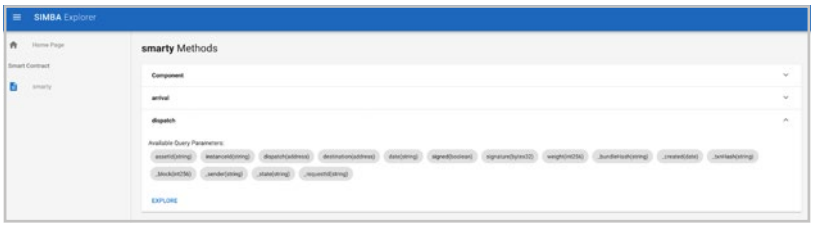
Irrespective of the method parameters defined in your smart contract, transactions have a number of common properties. These all begin with an underscore. They are described in more detail below.

NAME	TYPE	NOTES
_txnHash	string	The unique hash of the transaction once it has been sent to the blockchain.
_sender	address	This is the address of the identity that published the transaction.
_created	Date string	This is the time and date the transaction was submitted and the transaction hash was received.
_requestId	string	This is the unique id for the initial request which is returned from an HTTP call.
_state	string	The current state of the transaction. One of INITIALIZED, FAILED, SUBMITTED or COMPLETED
_block	integer	The block number the transaction was placed into on the blockchain

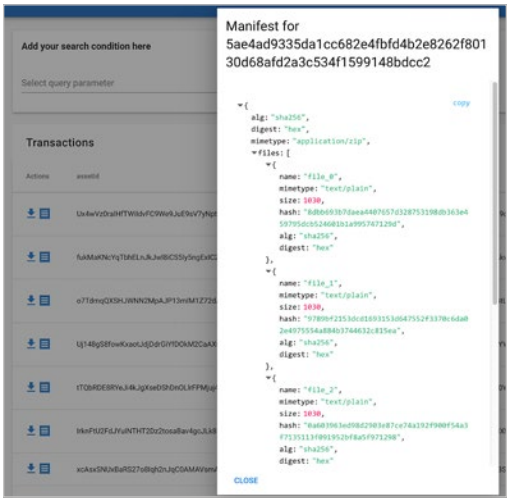
The Recent Transactions view shows submitted and completed transactions for all methods of your smart contract. You can query on the various common fields that all transactions have such as request ID, the sender and the transaction hash.



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.



UTILITIES

In the event of network failures and other unforeseen events, messages coming from the queue may be inserted into the dead letter queue. This sub-queue contains messages that failed to be processed after a number of retries. To view the raw dead letter queue, you can go to `https://<my-scaas-url>/admin/viewDlq` to see the messages in the queue. To attempt to move them back to the main queue, you can go to `https://<my-scaas-url>/admin/clearDlq`. This will attempt to move the messages back to the main queue for reprocessing.

Additionally, unrecoverable errors are logged to the log table in your SCaaS database. You can view errors here and determine the possible causes.

Swagger documentation for the administration APIs can be found at *https://<my-scaas-url>/docs*

simbachain.com
info@simbachain.com
574-914-4446
6100 Michigan Rd., Plymouth, IN 46563