An introduction to Aztec.js

In this guide, we will retrieving the Sandbox and deploy a pre-written contract to it using Aztec.js.

This guide assumes you have followed thequickstart .

Prerequisites

- A runningAztec sandbox

Set up the project

We will deploy a pre-compiled token contract, and send tokens privately, using the Sandbox.

info Find the full codehere We will create ayarn project calledtoken (althoughnpm works fine too).

1. Initialize a yarn project

mkdir token cd token yarn init -yp 1. Create asrc 2. folder inside your newtoken 3. directory:

mkdir src 1. Add necessary yarn packages (and optionally add typescript too)

yarn add @aztec/aztec.js @aztec/accounts @aztec/noir-contracts.js typescript @types/node 1. [Optional]If creating a typescript file, add atsconfig.json 2. file into the project root, here is an example:

{ "compilerOptions" :

{ "outDir" :

"dest" , "rootDir" :

"src" , "target" :

"es2020" , "lib" :

["dom" ,

"esnext" ,

"es2017.object"] , "module" :

"NodeNext" , "moduleResolution" :

"NodeNext" , "strict" :

true , "declaration" :

true , "allowSyntheticDefaultImports" :

true , "esModuleInterop" :

true , "downlevelIteration" :

true , "inlineSourceMap" :

true , "declarationMap" :

true , "importHelpers" :

true , "resolveJsonModule" :

true , "composite" :

true , "skipLibCheck" :

true } , "references" :

[ ] , "include" :

["src" ,

"src/*.json"] } 1. Updatepackage.json 2. - Add ascripts 3. section topackage.json 4. and set"type": "module" 5. :

{ "name" :

"token" , "version" :

"1.0.0" , "description" :

"My first token contract" , "main" :

"index.js" , "author" :

"1000x Dev" , "license" :

"MIT" , "type" :

"module" , "scripts" :

{ "build" :

"yarn clean && tsc -b" , "build:dev" :

"tsc -b --watch" , "clean" :

"rm -rf ./dest tsconfig.tsbuildinfo" , "start" :

"yarn build && DEBUG='token' node ./dest/index.js" } , "dependencies" :

{ "@aztec/accounts" :

"latest" , "@aztec/aztec.js" :

"latest" , "@aztec/noir-contracts.js" :

"latest" , "@types/node" :

"^20.6.3" , "typescript" :

"^5.2.2" } } 1. Create anindex.ts 2. file in thesrc 3. directory with the following sandbox connection setup:

import

{ getSchnorrAccount }

from

'@aztec/accounts/schnorr' ; import

{ getDeployedTestAccountsWallets }

from

'@aztec/accounts/testing' ; import

{ ExtendedNote , Fr , GrumpkinScalar , Note , PXE , computeMessageSecretHash , createDebugLogger , createPXEClient , waitForPXE , }

from

'@aztec/aztec.js' ; import

{ TokenContract }

from

'@aztec/noir-contracts.js/Token' ;

import

{ format }

from

'util' ;

const

{

PXE_URL

=

'http://localhost:8080'

}

= process . env ;

async

function

main ( )

{ ////////////// CREATE THE CLIENT INTERFACE AND CONTACT THE SANDBOX ////////////// const logger =

createDebugLogger ('token') ;

// We create PXE client connected to the sandbox URL const pxe =

createPXEClient (PXE_URL) ; // Wait for sandbox to be ready await

waitForPXE (pxe , logger) ;

const nodeInfo =

await pxe . getNodeInfo ( ) ;

logger (format ('Aztec Sandbox Info ' , nodeInfo)) ; }

main ( ) ; 1. Finally, run the package:

In the project root, run

yarn start A successful run should show something like this:

token Aztec Sandbox Info { token sandboxVersion: '0.28.1', token chainId: 31337, token protocolVersion: 1, token l1ContractAddresses: { token rollupAddress: EthAddress { token buffer: token }, token registryAddress: EthAddress { token buffer: token }, token inboxAddress: EthAddress { token buffer: token }, token outboxAddress: EthAddress { token buffer: token }, token } token } +0ms Great! The Sandbox is running and we are able to interact with it.

Load accounts

The sandbox is preloaded with multiple accounts so you don't have to sit and create them. Let's load these accounts. Add this code to themain() function inindex.ts below the code that's there:

load_accounts ////////////// LOAD SOME ACCOUNTS FROM THE SANDBOX ////////////// // The sandbox comes with a set of created accounts. Load them const accounts =

await

getDeployedTestAccountsWallets (pxe) ; const aliceWallet = accounts [0] ; const bobWallet = accounts [1] ; const alice = aliceWallet . getAddress ( ) ; const bob = bobWallet . getAddress ( ) ; logger (Loaded alice's account at { alice . toShortString ( ) }) ; logger (Loaded bob's account at { bob . toShortString ( ) }) ;Source code: yarn-project/end-to-end/src/e2e_sandbox_example.test.ts#L45-L55 An explanation on accounts on Aztec can be foundhere .

If you want more accounts, you can find instructions in theAccount creation section .

Deploy a contract

Now that we have our accounts loaded, let's move on to deploy our pre-compiled token smart contract. You can find the full code for the contracthere . Add this toindex.ts below the code you added earlier:

Deployment ////////////// DEPLOY OUR TOKEN CONTRACT //////////////

const initialSupply =

1_000_000n ; logger (Deploying token contract...) ;

// Deploy the contract and set Alice as the admin while doing so const contract =

await TokenContract . deploy (aliceWallet , alice ,

'TokenName' ,

'TokenSymbol' ,

18) . send ( ) . deployed ( ) ; logger (Contract successfully deployed at address { contract . address . toShortString ( ) }) ;

// Create the contract abstraction and link it to Alice's wallet for future signing const tokenContractAlice =

await TokenContract . at (contract . address , aliceWallet) ;

// Create a secret and a corresponding hash that will be used to mint funds privately const aliceSecret = Fr . random ( ) ; const aliceSecretHash =

computeMessageSecretHash (aliceSecret) ;

logger (Minting tokens to Alice...) ; // Mint the initial supply privately "to secret hash" const receipt =

await tokenContractAlice . methods . mint_private (initialSupply , aliceSecretHash) . send ( ) . wait ( ) ;

// Add the newly created "pending shield" note to PXE const pendingShieldsStorageSlot =

new

Fr (5) ;

// The storage slot of pending_shields is 5. const noteTypeId =

new

Fr (84114971101151129711410111011678111116101n) ;

// TransparentNote

const note =

new

Note ([new

Fr (initialSupply) , aliceSecretHash]) ; await pxe . addNote (new

ExtendedNote (note , alice , contract . address , pendingShieldsStorageSlot , noteTypeId , receipt . txHash) ,) ;

// Make the tokens spendable by redeeming them using the secret (converts the "pending shield note" created above // to a "token note") await tokenContractAlice . methods . redeem_shield (alice , initialSupply , aliceSecret) . send ( ) . wait ( ) ; logger ({ initialSupply } tokens were successfully minted and redeemed by Alice) ; Source code: yarn-project/end-to-end/src/e2e_sandbox_example.test.ts#L57-L91 yarn start will now give something like this:

token Aztec Sandbox Info { token sandboxVersion: '0.28.1', token chainId: 31337, token protocolVersion: 1, token l1ContractAddresses: { token rollupAddress: EthAddress { token buffer: token }, token registryAddress: EthAddress { token buffer: token }, token inboxAddress: EthAddress { token buffer: token }, token outboxAddress: EthAddress { token buffer: token }, token } token } +0ms token Loaded alice's account at 0x25048e8c...70d0 +4s token Loaded bob's account at 0x115f123b...6483 +0ms token Deploying token contract... +0ms token Contract successfully deployed at address 0x11a03dce...afc7 +5s token Minting tokens to Alice... +18ms token 1000000 tokens were successfully minted and redeemed by Alice +10s We can break this down as follows:

1. We create and send a contract deployment transaction to the network.
2. We wait for it to be successfully mined.
3. We retrieve the transaction receipt containing the transaction status and contract address.
4. We connect to the contract with Alice
5. Alice initialize the contract with herself as the admin and a minter.
6. Alice mints 1,000,000 tokens to be claimed by herself in private.
7. Alice redeems the tokens privately.

View the balance of an account

A token contract wouldn't be very useful if you aren't able to query the balance of an account. As part of the deployment, tokens were minted to Alice. We can now call the contract'sbalance_of_private() function to retrieve the balances of the accounts.

Call thebalance_of_private function using the following code (paste this):

Balance ////////////// QUERYING THE TOKEN BALANCE FOR EACH ACCOUNT //////////////

// Bob wants to mint some funds, the contract is already deployed, create an abstraction and link it his wallet // Since we already have a token link, we can simply create a new instance of the contract linked to Bob's wallet const tokenContractBob = tokenContractAlice . withWallet (bobWallet) ;

let aliceBalance =

await tokenContractAlice . methods . balance_of_private (alice) . view ( ) ; logger (Alice's balance { aliceBalance }) ;

let bobBalance =

await tokenContractBob . methods . balance_of_private (bob) . view ( ) ; logger (Bob's balance { bobBalance }) ; Source code: yarn-project/end-to-end/src/e2e_sandbox_example.test.ts#L96-L110 Running now should yield output:

token Aztec Sandbox Info { token sandboxVersion: '0.28.1', token chainId: 31337, token protocolVersion: 1, token l1ContractAddresses: { token rollupAddress: EthAddress { token buffer: token }, token registryAddress: EthAddress { token buffer: token }, token inboxAddress: EthAddress { token buffer: token }, token outboxAddress: EthAddress { token buffer: token }, token } token } +0ms token Loaded alice's account at 0x25048e8c...70d0 +4s token Loaded bob's account at 0x115f123b...6483 +0ms token Deploying token contract... +0ms token Contract successfully deployed at address 0x1b388d99...4b55 +4s token Minting tokens to Alice... +10ms token 1000000 tokens were successfully minted and redeemed by Alice +10s token Alice's balance 1000000 +80ms token Bob's balance 0 +31ms Above, we created a second instance of theTokenContract contract class. This time pertaining to Bob. This class offers a TypeScript bindings of ourToken contract.. We then callbalance_of_private() as aview method. View methods can be thought as read-only. No transaction is submitted as a result but a user's state can be queried.

We can see that each account has the expected balance of tokens.

Calling an unconstrained (view) function

Create and submit a transaction

Now lets transfer some funds from Alice to Bob by calling thetransfer function on the contract. This function takes 4 arguments:

1. The sender.
2. The recipient.
3. The quantity of tokens to be transferred.
4. The nonce for theauthentication witness
5. , or 0 if msg.sender equal sender.

Here is the Typescript code to call thetransfer function, add this to yourindex.ts at the bottom of themain function:

Transfer ////////////// TRANSFER FUNDS FROM ALICE TO BOB //////////////

// We will now transfer tokens from ALice to Bob const transferQuantity =

543n ; logger (Transferring { transferQuantity } tokens from Alice to Bob...) ; await tokenContractAlice . methods . transfer (alice , bob , transferQuantity ,

0) . send ( ) . wait ( ) ;

// Check the new balances aliceBalance =

await tokenContractAlice . methods . balance_of_private (alice) . view ( ) ; logger (Alice's balance { aliceBalance }) ;

bobBalance

await tokenContractBob . methods . balance_of_private (bob) . view ( ) ; logger (Bob's balance { bobBalance }) ; Source code: yarn-project/end-to-end/src/e2e_sandbox_example.test.ts#L115-L129 Our output should now look like this:

token Aztec Sandbox Info { token sandboxVersion: '0.28.1', token chainId: 31337, token protocolVersion: 1, token l1ContractAddresses: { token rollupAddress: EthAddress { token buffer: token }, token registryAddress: EthAddress { token

buffer: token }, token inboxAddress: EthAddress { token buffer: token }, token outboxAddress: EthAddress { token buffer: token }, token } token } +0ms token Loaded alice's account at 0x25048e8c...70d0 +4s token Loaded bob's account at 0x115f123b...6483 +0ms token Deploying token contract... +0ms token Contract successfully deployed at address 0x01d8af7d...9a4d +5s token Minting tokens to Alice... +18ms token 1000000 tokens were successfully minted and redeemed by Alice +11s token Alice's balance 1000000 +59ms token Bob's balance 0 +33ms token Transferring 543 tokens from Alice to Bob... +0ms token Alice's balance 999457 +6s token Bob's balance 543 +39ms Here, we used the same contract abstraction as was previously used for reading Alice's balance. But this time we calledsend() generating and sending a transaction to the network. After waiting for the transaction to settle we were able to check the new balance values.

Finally, the contract has 2mint functions that can be used to generate new tokens for an account. We will focus only onmint_private . This function is public but it mints tokens privately. This function takes:

1. A quantity of tokens to be minted.
2. A secret hash.

This function is public and it inserts a new note into the note hash tree and increases the total token supply by the amount minted.

To make the note spendable the note has to be redeemed. A user can do that by calling theredeem_shield function.

Let's now use these functions to mint some tokens to Bob's account using Typescript, add this toindex.ts :

Mint /////////////// MINT SOME MORE TOKENS TO BOB'S ACCOUNT ///////////////

// Now mint some further funds for Bob

// Alice is nice and she adds Bob as a minter await tokenContractAlice . methods . set_minter (bob ,

true) . send ( ) . wait ( ) ;

const bobSecret = Fr . random ( ) ; const bobSecretHash =

computeMessageSecretHash (bobSecret) ; // Bob now has a secret

const mintQuantity =

10_000n ; logger (Minting { mintQuantity } tokens to Bob...) ; const mintPrivateReceipt =

await tokenContractBob . methods . mint_private (mintQuantity , bobSecretHash) . send ( ) . wait ( ) ;

const bobPendingShield =

new

Note ([new

Fr (mintQuantity) , bobSecretHash]) ; await pxe . addNote (new

ExtendedNote (bobPendingShield , bob , contract . address , pendingShieldsStorageSlot , noteTypeId , mintPrivateReceipt . txHash ,) ,) ;

await tokenContractBob . methods . redeem_shield (bob , mintQuantity , bobSecret) . send ( ) . wait ( ) ;

// Check the new balances aliceBalance =

await tokenContractAlice . methods . balance_of_private (alice) . view ( ) ; logger (Alice's balance { aliceBalance }) ;

bobBalance

await tokenContractBob . methods . balance_of_private (bob) . view ( ) ; logger (Bob's balance { bobBalance }) ;Source code: yarn-project/end-to-end/src/e2e_sandbox_example.test.ts#L134-L170 Our complete output should now be something like:

token Aztec Sandbox Info { token sandboxVersion: '0.28.1', token chainId: 31337, token protocolVersion: 1, token l1ContractAddresses: { token rollupAddress: EthAddress { token buffer: token }, token registryAddress: EthAddress { token buffer: token }, token inboxAddress: EthAddress { token buffer: token }, token outboxAddress: EthAddress { token buffer: token }, token } token } +0ms token Loaded alice's account at 0x25048e8c...70d0 +4s token Loaded bob's account at 0x115f123b...6483 +0ms token Deploying token contract... +0ms token Contract successfully deployed at address 0x03a0bb2c...02c2 +7s token Minting tokens to Alice... +19ms token 1000000 tokens were successfully minted and redeemed by Alice +9s token Alice's balance 1000000 +43ms token Bob's balance 0 +31ms token Transferring 543 tokens from Alice to Bob... +0ms token Alice's balance 999457 +6s token Bob's balance 543 +36ms token Minting 10000 tokens to

Bob... +5s token Alice's balance 999457 +9s token Bob's balance 10543 +43ms That's it! We have successfully deployed a token contract to an instance of the Aztec network and mined private state-transitioning transactions. We have also queried the resulting state all via the interfaces provided by the contract. To see exactly what has happened here, you can learn about the transaction flowhere .

Next Steps

Learn more about writing Aztec.nr contracts on thenext page . Edit this page

Previous Writing your first smart contract Next Tutorials