Deploy & Call Contracts with Typescript

In this step, we We will now write a Typescript to interact with the sandbox and see our Solidity and Aztec.nr contracts in action.

In thepackages directory, go tosrc dir we created inthe token bridge tutorial .

cd src/test touch uniswap.test.ts Openuniswap.test.ts in your editor.

We will write two tests:

1. Test the private flow (i.e. mint tokens on L1, deposit them to L2, give your intention to swap L2 asset on L1, swap on L1, bridge swapped assets back to L2)
2. Do the same in the public flow

Compile our contracts

To compile the Solidity contracts, run this:

cd l1-contracts npx hardhat compile and the each of the Aztec.nr contracts by going into each folder and running:

aztec-nargo compile And then generate the typescript interface:

aztec-cli codegen ./target/ -o .. / .. / .. /src/test/fixtures uniswap --ts This will create a TS interface in oursrc/test folder that will help us write our test.

Test imports and setup

We will use the sameutils.ts andcross_chain_test_harness.ts we created in the tutorialhere .

Inutils.ts , add:

export

const

[UniswapPortalAbi , UniswapPortalBytecode]

= getL1ContractABIAndBytecode ("UniswapPortal") ;

Setup the fork

Since we want to use L1 Uniswap, we need the sandbox to execute against a fork of L1. This has be easily done: in your terminal add the following variables:

export FORK_BLOCK_NUMBER=17514288 export FORK_URL= Now rerun the sandbox:

/bin/sh -c " (curl -fsSL 'https://sandbox.aztec.network') "

Back to test setup

Okay now we are ready to write our tests:

openuniswap.test.ts and lets do the initial description of the test:

import

{ AccountWallet , AztecAddress , DebugLogger , EthAddress , Fr , PXE , TxStatus , computeAuthWitMessageHash , createDebugLogger , createPXEClient , waitForPXE , }

from

"@aztec/aztec.js" ; import

{ getInitialTestAccountsWallets }

from

'@aztec/accounts/testing' ; import

```
{ Chain , HttpTransport , PublicClient , createPublicClient , createWalletClient , getContract , http , parseEther , }
from
"viem" ; import
{ foundry }
from
"viem/chains" ; import
{ CrossChainTestHarness }
from
"./shared/cross_chain_test_harness.js" ; import
{ UniswapContract }
from
"./fixtures/Uniswap.js" ; import
{ beforeAll , expect , jest }
from
"@jest/globals" ; import
{ UniswapPortalAbi , UniswapPortalBytecode , delay , deployL1Contract , }
from
"./fixtures/utils.js" ; import
{ mnemonicToAccount }
from
"viem/accounts" ;
const
{ PXE_URL
=
"http://localhost:8080" , ETHEREUM_HOST
=
"http://localhost:8545" , }
= process . env ; const
MNEMONIC
=
"test test test test test test test test test test test junk" ; const hdAccount =
mnemonicToAccount ( MNEMONIC ) ; const expectedForkBlockNumber =
17514288 ;
const
TIMEOUT
=
360_000 ;
```

/* *Objects to be returned by the uniswap setup function*/ export

type

UniswapSetupContext

=

{ /* *The Private eXecution Environment (PXE).* / pxe :

PXE ; / **Logger instance named as the current test. */ logger : DebugLogger ;** /Viem Public client instance. */ publicClient : PublicClient < HttpTransport , Chain

; /* *Viem Wallet Client instance.*/ walletClient :

any ; / **The owner wallet. */ ownerWallet : AccountWallet ;** /The sponsor wallet. */ sponsorWallet : AccountWallet ; } ; const testSetup =

async

( ) :

Promise < UniswapSetupContext

=>

{ const

{ teardown : teardown_ , pxe , deployL1ContractsValues , wallets , logger , }

=

await

e2eSetup (2 ,

{ stateLoad : dumpedState }) ;

const walletClient = deployL1ContractsValues . walletClient ; const publicClient = deployL1ContractsValues . publicClient ;

const ownerWallet = wallets [0] ; const sponsorWallet = wallets [1] ;

teardown

teardown_ ;

return

{ pxe , logger , publicClient , walletClient , ownerWallet , sponsorWallet } ; } ; describe ('uniswap_trade_on_l1_from_l2' ,

( )

=>

{ jest . setTimeout (TIMEOUT) ;

const

WETH9_ADDRESS : EthAddress = EthAddress . fromString ('0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2') ; const

DAI_ADDRESS : EthAddress = EthAddress . fromString ('0x6B175474E89094C44Da98b954EedeAC495271d0F') ;

let pxe :

PXE ; let logger : DebugLogger ;

let walletClient :

any ;

let ownerWallet : AccountWallet ; let ownerAddress : AztecAddress ; let ownerEthAddress : EthAddress ; // does

transactions on behalf of owner on Aztec: let sponsorWallet : AccountWallet ; let sponsorAddress : AztecAddress ;

let daiCrossChainHarness : CrossChainTestHarness ; let wethCrossChainHarness : CrossChainTestHarness ;

let uniswapPortal :

any ; let uniswapPortalAddress : EthAddress ; let uniswapL2Contract : UniswapContract ;

const wethAmountToBridge =

parseEther ('1') ; const uniswapFeeTier =

3000n ; const minimumOutputAmount =

0n ; const deadlineForDepositingSwappedDai =

BigInt (2

**

32

-

1) ;

// max uint32

beforeAll (async

( )

=>

{ let publicClient : PublicClient < HttpTransport , Chain

; ({ pxe , logger , publicClient , walletClient , ownerWallet , sponsorWallet }

=

await

setup ( )) ;

// walletClient = deployL1ContractsValues.walletClient; // const publicClient = deployL1ContractsValues.publicClient;

if

(Number (await publicClient . getBlockNumber ( ))

< expectedForkBlockNumber)

{ throw

new

Error ('This test must be run on a fork of mainnet with the expected fork block') ; }

ownerAddress

ownerWallet . getAddress ( ) ; sponsorAddress = sponsorWallet . getAddress ( ) ; ownerEthAddress = EthAddress . fromString ((await walletClient . getAddresses ( )) [0]) ;

await

publicDeployAccounts (ownerWallet ,

[ownerAddress , sponsorAddress]) ;

logger ('Deploying DAI Portal, initializing and deploying l2 contract...') ; daiCrossChainHarness =

await CrossChainTestHarness . new (pxe , publicClient , walletClient , ownerWallet , logger , DAI_ADDRESS ,) ;

logger ('Deploying WETH Portal, initializing and deploying l2 contract...') ; wethCrossChainHarness =

await CrossChainTestHarness . new (pxe , publicClient , walletClient , ownerWallet , logger , WETH9_ADDRESS ,) ;

logger ('Deploy Uniswap portal on L1 and L2...') ; uniswapPortalAddress =

await

deployL1Contract (walletClient , publicClient , UniswapPortalAbi , UniswapPortalBytecode ,) ; uniswapPortal =

getContract ({ address : uniswapPortalAddress . toString ( ) , abi : UniswapPortalAbi , client : walletClient , }) ; // deploy l2 uniswap contract and attach to portal uniswapL2Contract =

await UniswapContract . deploy (ownerWallet) . send ({ portalContract : uniswapPortalAddress }) . deployed ( ) ;

const registryAddress =

(await pxe . getNodeInfo ( )) . l1ContractAddresses . registryAddress ; await uniswapPortal . write . initialize ([registryAddress . toString ( ) , uniswapL2Contract . address . toString ( )] , { }

as

any ,) ; }) ;

beforeEach (async

( )

=>

{ // Give me some WETH so I can deposit to L2 and do the swap... logger ('Getting some weth') ; await walletClient . sendTransaction ({ to :

WETH9_ADDRESS . toString ( ) , value :

parseEther ('1')

}) ; }) ;

Private flow test

uniswap_private it ('should uniswap trade on L1 from L2 funds privately (swaps WETH -> DAI)' ,

async

( )

=>

{ const wethL1BeforeBalance =

await wethCrossChainHarness . getL1BalanceOf (ownerEthAddress) ;

// 1. Approve and deposit weth to the portal and move to L2 const

[secretForMintingWeth , secretHashForMintingWeth]

= wethCrossChainHarness . generateClaimSecret ( ) ; const

[secretForRedeemingWeth , secretHashForRedeemingWeth]

= wethCrossChainHarness . generateClaimSecret ( ) ;

const entryKey =

await wethCrossChainHarness . sendTokensToPortalPrivate (secretHashForRedeemingWeth , wethAmountToBridge , secretHashForMintingWeth ,) ; // funds transferred from owner to token portal expect (await wethCrossChainHarness . getL1BalanceOf (ownerEthAddress)) . toBe (wethL1BeforeBalance - wethAmountToBridge ,) ; expect (await wethCrossChainHarness . getL1BalanceOf (wethCrossChainHarness . tokenPortalAddress)) . toBe (wethAmountToBridge ,) ; expect (await wethCrossChainHarness . inbox . read . contains ([entryKey . toString ( )])) . toBe (true) ;

// Wait for the archiver to process the message await

sleep (5000) ;

// Perform an unrelated transaction on L2 to progress the rollup. Here we mint public tokens. await wethCrossChainHarness . mintTokensPublicOnL2 (0n) ;

// 2. Claim WETH on L2 logger ('Minting weth on L2') ; await wethCrossChainHarness . consumeMessageOnAztecAndMintSecretly (secretHashForRedeemingWeth , wethAmountToBridge , secretForMintingWeth ,) ; await wethCrossChainHarness . redeemShieldPrivatelyOnL2 (wethAmountToBridge , secretForRedeemingWeth) ; await wethCrossChainHarness . expectPrivateBalanceOnL2 (ownerAddress , wethAmountToBridge) ;

// Store balances const wethL2BalanceBeforeSwap =

await wethCrossChainHarness . getL2PrivateBalanceOf (ownerAddress) ; const daiL2BalanceBeforeSwap =

await daiCrossChainHarness . getL2PrivateBalanceOf (ownerAddress) ;

// before swap - check nonce_for_burn_approval stored on uniswap // (which is used by uniswap to approve the bridge to burn funds on its behalf to exit to L1) const nonceForBurnApprovalBeforeSwap =

await uniswapL2Contract . methods . nonce_for_burn_approval ( ) . view ( ) ;

// 3. Owner gives uniswap approval to unshield funds to self on its behalf logger ('Approving uniswap to unshield funds to self on my behalf') ; const nonceForWETHUnshieldApproval =

new

Fr (1n) ; const unshieldToUniswapMessageHash =

computeAuthWitMessageHash (uniswapL2Contract . address , wethCrossChainHarness . l2Token . methods . unshield (ownerAddress , uniswapL2Contract . address , wethAmountToBridge , nonceForWETHUnshieldApproval) . request ( ) ,) ; await ownerWallet . createAuthWitness (unshieldToUniswapMessageHash) ;

// 4. Swap on L1 - sends L2 to L1 message to withdraw WETH to L1 and another message to swap assets. logger ('Withdrawing weth to L1 and sending message to swap to dai') ; const

[secretForDepositingSwappedDai , secretHashForDepositingSwappedDai]

= daiCrossChainHarness . generateClaimSecret ( ) ; const

[secretForRedeemingDai , secretHashForRedeemingDai]

= daiCrossChainHarness . generateClaimSecret ( ) ;

const withdrawReceipt =

await uniswapL2Contract . methods . swap_private (wethCrossChainHarness . l2Token . address , wethCrossChainHarness . l2Bridge . address , wethAmountToBridge , daiCrossChainHarness . l2Bridge . address , nonceForWETHUnshieldApproval , uniswapFeeTier , minimumOutputAmount , secretHashForRedeemingDai , secretHashForDepositingSwappedDai , deadlineForDepositingSwappedDai , ownerEthAddress , ownerEthAddress ,) . send ( ) . wait ( ) ; expect (withdrawReceipt . status) . toBe (TxStatus . MINED) ; // ensure that user's funds were burnt await wethCrossChainHarness . expectPrivateBalanceOnL2 (ownerAddress , wethL2BalanceBeforeSwap - wethAmountToBridge) ; // ensure that uniswap contract didn't eat the funds. await wethCrossChainHarness . expectPublicBalanceOnL2 (uniswapL2Contract . address ,

0n) ; // check burn approval nonce incremented: const nonceForBurnApprovalAfterSwap =

await uniswapL2Contract . methods . nonce_for_burn_approval ( ) . view ( ) ; expect (nonceForBurnApprovalAfterSwap) . toBe (nonceForBurnApprovalBeforeSwap +

1n) ;

// 5. Consume L2 to L1 message by calling uniswapPortal.swap_private() logger ('Execute withdraw and swap on the uniswapPortal!') ; const daiL1BalanceOfPortalBeforeSwap =

await daiCrossChainHarness . getL1BalanceOf (daiCrossChainHarness . tokenPortalAddress ,) ; const swapArgs =

[wethCrossChainHarness . tokenPortalAddress . toString ( ) , wethAmountToBridge , uniswapFeeTier , daiCrossChainHarness . tokenPortalAddress . toString ( ) , minimumOutputAmount , secretHashForRedeemingDai . toString ( ) , secretHashForDepositingSwappedDai . toString ( ) , deadlineForDepositingSwappedDai , ownerEthAddress . toString ( ) , true ,]

as

const ;

// this should also insert a message into the inbox. await uniswapPortal . write . swapPrivate (swapArgs ,

{ }

as

any) ;

// weth was swapped to dai and send to portal const daiL1BalanceOfPortalAfter =

await daiCrossChainHarness . getL1BalanceOf (daiCrossChainHarness . tokenPortalAddress ,) ; expect (daiL1BalanceOfPortalAfter) . toBeGreaterThan (daiL1BalanceOfPortalBeforeSwap) ; const daiAmountToBridge =

BigInt (daiL1BalanceOfPortalAfter - daiL1BalanceOfPortalBeforeSwap) ;

// Wait for the archiver to process the message await

sleep (5000) ; // send a transfer tx to force through rollup with the message included await wethCrossChainHarness . mintTokensPublicOnL2 (0n) ;

// 6. claim dai on L2 logger ('Consuming messages to mint dai on L2') ; await daiCrossChainHarness . consumeMessageOnAztecAndMintSecretly (secretHashForRedeemingDai , daiAmountToBridge , secretForDepositingSwappedDai ,) ; await daiCrossChainHarness . redeemShieldPrivatelyOnL2 (daiAmountToBridge , secretForRedeemingDai) ; await daiCrossChainHarness . expectPrivateBalanceOnL2 (ownerAddress , daiL2BalanceBeforeSwap + daiAmountToBridge) ;

const wethL2BalanceAfterSwap =

await wethCrossChainHarness . getL2PrivateBalanceOf (ownerAddress) ; const daiL2BalanceAfterSwap =

await daiCrossChainHarness . getL2PrivateBalanceOf (ownerAddress) ;

logger ('WETH balance before swap: '

+ wethL2BalanceBeforeSwap . toString ( )) ; logger ('DAI balance before swap : '

+ daiL2BalanceBeforeSwap . toString ( )) ; logger (* ♀ **SWAP L2 assets on L1 Uniswap** **) ; logger ('WETH balance after swap : ' , wethL2BalanceAfterSwap . toString ( )) ; logger ('DAI balance after swap : ' , daiL2BalanceAfterSwap . toString ( )) ; }) ; Source code: yarn-project/end-to-end/src/shared/uniswap_l1_l2.ts#L156-L297

Public flow test

uniswap_public it ('should uniswap trade on L1 from L2 funds publicly (swaps WETH -> DAI)' ,

async

( )

=>

{ const wethL1BeforeBalance =

await wethCrossChainHarness . getL1BalanceOf (ownerEthAddress) ;

// 1. Approve and deposit weth to the portal and move to L2 const

[secretForMintingWeth , secretHashForMintingWeth]

= wethCrossChainHarness . generateClaimSecret ( ) ;

const entryKey =

await wethCrossChainHarness . sendTokensToPortalPublic (wethAmountToBridge , secretHashForMintingWeth ,) ; // funds transferred from owner to token portal expect (await wethCrossChainHarness . getL1BalanceOf (ownerEthAddress)) . toBe (wethL1BeforeBalance - wethAmountToBridge ,) ; expect (await wethCrossChainHarness . getL1BalanceOf (wethCrossChainHarness . tokenPortalAddress)) . toBe (wethAmountToBridge ,) ; expect (await wethCrossChainHarness . inbox . read . contains ([entryKey . toString ( )])) . toBe (true) ;

// Wait for the archiver to process the message await

sleep (5000) ;

// Perform an unrelated transaction on L2 to progress the rollup. Here we transfer 0 tokens await wethCrossChainHarness . mintTokensPublicOnL2 (0n) ;

// 2. Claim WETH on L2 logger ('Minting weth on L2') ; await wethCrossChainHarness . consumeMessageOnAztecAndMintPublicly (wethAmountToBridge , secretForMintingWeth) ; await wethCrossChainHarness . expectPublicBalanceOnL2 (ownerAddress , wethAmountToBridge) ;

// Store balances const wethL2BalanceBeforeSwap =

await wethCrossChainHarness . getL2PublicBalanceOf (ownerAddress) ; const daiL2BalanceBeforeSwap =

await daiCrossChainHarness . getL2PublicBalanceOf (ownerAddress) ;

// 3. Owner gives uniswap approval to transfer funds on its behalf const nonceForWETHTransferApproval =

new

Fr (1n) ; const transferMessageHash =

computeAuthWitMessageHash (uniswapL2Contract . address , wethCrossChainHarness . l2Token . methods . transfer_public (ownerAddress , uniswapL2Contract . address , wethAmountToBridge , nonceForWETHTransferApproval) . request ( ) ,) ; await ownerWallet . setPublicAuth (transferMessageHash ,

true) . send ( ) . wait ( ) ;

// before swap - check nonce_for_burn_approval stored on uniswap // (which is used by uniswap to approve the bridge to burn funds on its behalf to exit to L1) const nonceForBurnApprovalBeforeSwap =

await uniswapL2Contract . methods . nonce_for_burn_approval ( ) . view ( ) ;

// 4. Swap on L1 - sends L2 to L1 message to withdraw WETH to L1 and another message to swap assets. const

[secretForDepositingSwappedDai , secretHashForDepositingSwappedDai]

= daiCrossChainHarness . generateClaimSecret ( ) ;

// 4.1 Owner approves user to swap on their behalf: const nonceForSwap =

new

Fr (3n) ; const action = uniswapL2Contract . withWallet (sponsorWallet) . methods . swap_public (ownerAddress , wethCrossChainHarness . l2Bridge . address , wethAmountToBridge , daiCrossChainHarness . l2Bridge . address , nonceForWETHTransferApproval , uniswapFeeTier , minimumOutputAmount , ownerAddress , secretHashForDepositingSwappedDai , deadlineForDepositingSwappedDai , ownerEthAddress , ownerEthAddress , nonceForSwap ,) ; const swapMessageHash =

computeAuthWitMessageHash (sponsorAddress , action . request ( )) ; await ownerWallet . setPublicAuth (swapMessageHash ,

true) . send ( ) . wait ( ) ;

// 4.2 Call swap_public from user2 on behalf of owner const withdrawReceipt =

await action . send ( ) . wait ( ) ; expect (withdrawReceipt . status) . toBe (TxStatus . MINED) ;

// check weth balance of owner on L2 (we first bridged wethAmountToBridge into L2 and now withdrew it!) await wethCrossChainHarness . expectPublicBalanceOnL2 (ownerAddress , wethL2BalanceBeforeSwap - wethAmountToBridge) ;

// check burn approval nonce incremented: const nonceForBurnApprovalAfterSwap =

await uniswapL2Contract . methods . nonce_for_burn_approval ( ) . view ( ) ; expect (nonceForBurnApprovalAfterSwap) . toBe (nonceForBurnApprovalBeforeSwap +

1n) ;

// 5. Perform the swap on L1 with the uniswapPortal.swap_private() (consuming L2 to L1 messages) logger ('Execute withdraw and swap on the uniswapPortal!') ; const daiL1BalanceOfPortalBeforeSwap =

await daiCrossChainHarness . getL1BalanceOf (daiCrossChainHarness . tokenPortalAddress ,) ; const swapArgs =

[wethCrossChainHarness . tokenPortalAddress . toString ( ) , wethAmountToBridge , uniswapFeeTier , daiCrossChainHarness . tokenPortalAddress . toString ( ) , minimumOutputAmount , ownerAddress . toString ( ) , secretHashForDepositingSwappedDai . toString ( ) , deadlineForDepositingSwappedDai , ownerEthAddress . toString ( ) , true ,]

as

const ;

// this should also insert a message into the inbox. await uniswapPortal . write . swapPublic (swapArgs ,

{ }

as

any) ; // weth was swapped to dai and send to portal const daiL1BalanceOfPortalAfter =

await daiCrossChainHarness . getL1BalanceOf (daiCrossChainHarness . tokenPortalAddress ,) ; expect (daiL1BalanceOfPortalAfter) . toBeGreaterThan (daiL1BalanceOfPortalBeforeSwap) ; const daiAmountToBridge =

BigInt (daiL1BalanceOfPortalAfter - daiL1BalanceOfPortalBeforeSwap) ;

// Wait for the archiver to process the message await

sleep (5000) ; // send a transfer tx to force through rollup with the message included await wethCrossChainHarness . mintTokensPublicOnL2 (0n) ;

// 6. claim dai on L2 logger ('Consuming messages to mint dai on L2') ; await daiCrossChainHarness . consumeMessageOnAztecAndMintPublicly (daiAmountToBridge , secretForDepositingSwappedDai) ; await daiCrossChainHarness . expectPublicBalanceOnL2 (ownerAddress , daiL2BalanceBeforeSwap + daiAmountToBridge) ;

const wethL2BalanceAfterSwap =

await wethCrossChainHarness . getL2PublicBalanceOf (ownerAddress) ; const daiL2BalanceAfterSwap =

await daiCrossChainHarness . getL2PublicBalanceOf (ownerAddress) ;

logger ('WETH balance before swap: ' , wethL2BalanceBeforeSwap . toString ( )) ; logger ('DAI balance before swap : ' , daiL2BalanceBeforeSwap . toString ( )) ; logger ('* ♀ **SWAP L2 assets on L1 Uniswap**) ; logger ('WETH balance after swap : ' , wethL2BalanceAfterSwap . toString ( )) ; logger ('DAI balance after swap : ' , daiL2BalanceAfterSwap . toString ( )) ; } ,

360_000) ; Source code: yarn-project/end-to-end/src/shared/uniswap_l1_l2.ts#L299-L431

Running the test

Make sure your sandbox is running.

cd ~/.aztec &&

docker-compose up Then run this in the root directory.

cd packages/src yarn

test uniswap Edit this page

Previous Redeeming Swapped Assets on L2 Next Testing with Typescript