Build a Token Bridge

In this tutorial, we will learn how to build the entire flow of a cross-chain token using portals. If this is your first time hearing the word portal, you'll want to readthis .

A refresher on Portals

A portal is the point of contact between L1 and a specific contract on Aztec. It allows for arbitrary message passing between L1 and Aztec, siloed just for the portal contract and its sister contract on Aztec. For applications such as token bridges, this is the point where the tokens are held on L1 while used in L2.

But why?

Before portals, you had legos either just on L1 or just on L2. But for cross-chain, there was no arbitrary message passing bridge that didn't introduce their own trust assumptions.

Portals change this. With portals you can now have arbitrary execution across L1 and L2, paving the ground for seamless trustless composability with L1 and L2 legos, without introducing any additional trust assumptions across the L1 or Aztec network.

That means your L1 application can have a sister application residing on L2 and both of them can work together across the two networks.

Cheap and private access to Ethereum

Using portals, you could implement Aztec Connect-like functionality where you could deposit funds into a variety of DeFi protocols that reside on Ethereum even though your funds are on Aztec. This enables cheaper and private usage of existing dapps on Ethereum and prevents liquidity fragmentation.

You could swap your L2 WETH into DAI on Uniswap on Ethereum and get the DAI on Aztec. Similarly, you could stake your L2 ETH into Lido on Ethereum and get stETH on Aztec!

L1<>L2 communication on Aztec

Aztec has the following core smart contracts on L1 that we need to know about:

- Rollup.sol
-
  - stores the current state of the rollup and includes logic to progress the rollup (i.e. the state transition function)
- Inbox.sol
-
  - a mailbox to the rollup for L1 to L2 messages (e.g. depositing tokens). Portals put messages into the box, and the sequencers then decide which of these messages they want to include in their blocks, based on the inclusion fees they receive.
- Outbox.sol
-
  - a mailbox to the rollup for L2 to L1 messages (e.g. withdrawing tokens). Aztec contracts emit these messages and the sequencer adds these to the outbox. Portals then consume these messages.
- Registry.sol
-
  - just like L1, we assume there will be various versions of Aztec (due to upgrades, forks etc). In such a case messages must not be replayable in other Aztec "domains". A portal must decide which version/ID of Aztec the message is for. The registry stores the rollup, inbox and outbox address for each version of Aztec deployments, so the portal can find out the address of the mailbox it wants to talk to

For more information, readcross-chain calls .

Building a Token Bridge with Portals

The goal for this tutorial is to create functionality such that a token can be bridged to and from Aztec. We'll be using L1 to refer to Ethereum and L2 to refer to Aztec.

This is just a reference implementation for educational purposes only. It has not been through an in-depth security audit.

Let's assume a token exists on Ethereum and Aztec (see aguide on writing a token contract on Aztec here).

We will build:

- aToken Portal
- solidity contract on L1 that will be responsible for sending messages to the Inbox and consuming from the Outbox.
- aToken Bridge
- aztec-nr contract on L2 that can consume L1 to L2 messages to mint tokens on L2 and create L2 to L1 messages to withdraw tokens back to L1.
- Some TypeScript code that can call the methods on the contracts and communicate with the sandbox.

Our contracts will be able to work withboth private and public state i.e. how to deposit tokens into Aztec privately and publicly and withdraw tokens privately and publicly.

This just shows the private flow. The green is the deposit to L2 flow, while the red is the withdrawal from L2 flow. The blue user represents an operator - a 3rd person who can act on behalf of the user!

The token portal resides on L1 and must be able to deposit tokens to Aztec (both privately and publicly). It must also be able to withdraw funds from Aztec and cancel any deposit messages (L1->L2 messages) should the user change their mind or if the message wasn't picked up on time.

The token bridge resides on L2 and is the "sister" contract that can claim the deposit message to mint tokens on L2 (publicly or privately). Similarly, it should be able to burn tokens on L2 and withdraw them on L1.

More about the flow will be clear as we code along! In the next section, we'll set up our Ethereum and Aztec environments.
Edit this page

Previous Testing Next Setup and Installation