Web3 Builders Alliance

Welcome Axelar

Basic Contract Structure (EVM) ERC-20

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.9;

import './IAxelarExecutable.sol';
import "@openzeppelin/contracts/token/ERC20/ERC20.sol";
import "@openzeppelin/contracts/access/Ownable.sol";

contract Devon is ERC20, Ownable {
    constructor() ERC20("DevonToken", "DEV") {}

    function mint(address to, uint256 amount) public onlyOwner {
        _mint(to, amount);
    }
}
```

Check out:

https://docs.openzeppelin.com/contracts/4.x/wizard

https://github.com/DevonMartens/WBA-Token

Imports and Set Up of a Hardhat Repo

How do I import openzepplin?

```https://www.npmjs.com/package/@openzeppelin/contracts```

```npm install @openzeppelin/contracts```

Where did all this `hardhat` stuff come from?

```https://hardhat.org/tutorial/creating-a-new-hardhat-project```

```npm init```

```npm install --save-dev hardhat```

```npx hardhat```

What about the other installs?

```npm i @nomiclabs/hardhat-etherscan```

```npm i dotenv```

```npm i @nomiclabs/hardhat-ethers ```
```

Look in the IAxelarGateway.sol

```
/*
@Dev: To call chain B from chain A and send some tokens along the way
@Params: `destinationChain`: The destination chain, which must be an EVM chain from Chain names.
Find your chain names here:  `https://docs.axelar.dev/dev/build/chain-names`
@Params: `contractAddress`: The destination contract address, which must implement the  IAxelarExecutable interface defined in IAxelarExecutable.sol.
@Params: `payload`: The payload bytes to pass to the destination contract.
@Params: `symbol`: The symbol of the token to transfer, which must be a supported asset.
`https://docs.axelar.dev/dev/build/contract-addresses/testnet`
@Params: `amount`: The amount of the token to transfer
*/

function callContractWithToken(
    string calldata destinationChain,
    string calldata contractAddress,
    bytes calldata payload,
    string calldata symbol,
    uint256 amount
) external;
```

Look in the IAxelarExecutable.sol

```
/*
@Dev: After `callContractWithToken` is called  _executeWithToken function that will be triggered by the Axelar network
@Dev: You can write any custom logic here.
Then: The destination contract will be authorized to transfer the ERC-20 identified by the tokenSymbol.
*/

function executeWithToken(
    bytes32 commandId,
    string calldata sourceChain,
    string calldata sourceAddress,
    bytes calldata payload,
    string calldata tokenSymbol,
    uint256 amount
) external {
    bytes32 payloadHash = keccak256(payload);
    if (!gateway.validateContractCallAndMint(commandId, sourceChain, sourceAddress, payloadHash, tokenSymbol, amount))
        revert NotApprovedByGateway();

    _executeWithToken(sourceChain, sourceAddress, payload, tokenSymbol, amount);
}
```

Sending Messages: IAxelarGateway.sol

```
  /*
   @Dev: To call a contract on chain B from chain A, the user needs to call callContract on the gateway of
chain A, specifying
   @Params: `destinationChain`: The destination chain, which must be an EVM chain from Chain names.
   @Params: `contractAddress`: The destination contract address, which must implement the IAxelarExecutable
interface defined in IAxelarExecutable.sol.
   @Params: `payload`: The payload bytes to pass to the destination contract.
   */

   function callContract(
       string calldata destinationChain,
       string calldata contractAddress,
       bytes calldata payload
   ) external;
```

Sending Messages: IAxelarExecutable.sol

```
/*
  @Dev: This will be triggered by the Axelar network after the callContract function
has been executed.
  You can write any custom logic here.
  */

   function _execute(
        string memory sourceChain,
        string memory sourceAddress,
        bytes calldata payload
   ) internal virtual {}
```

Encoding Data with JavaScript

- **The payload passed to callContract (and ultimately to _execute and _executeWithToken) has type bytes.**
- **Use the ABI encoder/decoder convert your data to bytes.**

Encoding and decoding with ethers.js:

```
const { ethers } = require("ethers");

  // encoding a string
  const payload = ethers.utils.defaultAbiCoder.encode(
  ["string"],
  ["Hello from contract A"]
  );
```

Encoding and decoding with Web3.js:

```
dataEncoded = web3.eth.abi.encodeFunctionCall(messageCall);
```

https://github.com/ChainSafe/web3.js/blob/0.20.7/test/coder.encodeParam.js

https://github.com/ChainSafe/web3.js/blob/0.20.7/test/coder.decodeParam.js

Decoding Data with Solidity

```
function _execute(
        string memory sourceChain,
        string memory sourceAddress,
        bytes calldata payload
    ) internal override {
        // decoding a string
        string memory _message = abi.decode(payload, (string));
    }
```

Additional Tooling for Solidity

Constant Address Deployer - Deploys to multiple chains with the same address:

- This can be achieved by deploying each contract from the same address with the same nonce at each network.
- By using `create2`.
- `ConstAddressDeployer` at `0x98b2920d53612483f91f12ed7754e51b4a77919e` on every EVM testnet and mainnet that is supported by Axelar.
- **We plan on deploying it on future supported testnets and mainnets, too.**

For more information:

https://docs.axelar.dev/dev/build/solidity-utilities

https://etherscan.io/address/0x98b2920d53612483f91f12ed7754e51b4a77919e#code

Questions?!

Axelar SDK

How do I import it?

```npm i @axelar-network/axelarjs-sdk@alpha```

3 Modules

- **AxelarAssetTransfer**
  - Used for cross-chain token transfer via deposit address generation.
  - Token Transfer via Deposit Address.
- **AxelarGMPRecoveryAPI**
  - API library to track and recover (if needed) GMP transactions (both callContract and callContractWithToken).
  - Transactions are indexed by the transaction hash initiated on the source chain when invoking either callContract or callContractWithToken.
  - GMP transaction status and recovery.
- **AxelarQueryAPI**
  - Collection of helpful predefined queries into the network, e.g., transaction fees for token transfers, cross-chain gas prices for GMP transactions, denom conversions, etc.
  - Axelar Query API.
```

To transfer ERC-20 Tokens

Import & instantiate the AxelarAssetTransfer

```
import { AxelarAssetTransfer, Environment } from "@axelar-network/axelarjs-sdk ";

const axelarAssetTransfer = new AxelarAssetTransfer({
       environment : Environment.TESTNET,
      });
```

Generate a deposit address using the SDK

```
const sdk = new AxelarAssetTransfer({
   environment: "testnet"
 });
 const depositAddress = await sdk.getDepositAddress(
   CHAINS.TESTNET.POLYGON, // source chain
   CHAINS.TESTNET.OSMOSIS, // destination chain
   "0x95cacEA3622f4E1171EC6E3DdAF447d19085eB23", // destination address
   "uausdc" // denom of asset. See note (2) below
 );
```

Before the challenge you better....

Get tokens from polygon's testnet mumbai:

https://mumbaifaucet.com/

Get tokens from polygon's testnet mumbai:

https://mumbai.polygonscan.com/

Get uaUSDC from the Axelar Discord Faucet

https://discord.com/invite/aRZ3Ra6f7D

example of using the faucet:
!faucet polygon 0x61A7F8572CBB62b259016Eb2cdB710CE9df086b1

Coding Challenge Instructions

Result: https://testnet.axelarscan.io/account/0x79f23812Becd273845253cda065d8911e205FDA1

- Clone https://github.com/ivmidable/axelar-tests
- Generate a new osmosis mnemonic and then save it inside of the 0_tests.ts file
- Make sure you have metamask installed and are connected to the polygon testnet
- Get uausdc testnet tokens from the Axelar discord “faucet” channel
- Import the uausdc ERC20 contract address into metamask
- Generate a deposit address from polygon to osmosis. use the address generated from the mnemonic to create this deposit address
- Send the uausdc to the deposit address.
- You can check the results of the transfer on axelarscan by searching the generated deposit address

Extra Credit

- The `AxelarGMPRecoveryAPI` module in the AxelarJS SDK can be used by your dApp to query the status of any General Message Passing (GMP) transaction.
- *Ttriggered by either `callContract` or `callContractWithToken`) on the gateway contract of a source chain and trigger a manual relay from source to destination if necessary.

-> https://docs.axelar.dev/dev/axelarjs-sdk/tx-status-query-recovery

Extra Credit

Instantiate the AxelarGMPRecoveryAPI module:

```
import {
 AxelarGMPRecoveryAPI,
 Environment,
} from "@axelar-network/axelarjs-sdk";

const sdk = new AxelarGMPRecoveryAPI({
 environment: Environment.TESTNET,
});
```

Query transaction status by txHash. Invoke queryTransactionStatus:

```
const txHash: string =
 "0xfb6fb85f11496ef58b088116cb611497e87e9c72ff0c9333aa21491e4cdd397a";
const txStatus: GMPStatusResponse = await sdk.queryTransactionStatus(txHash);
```