Zeta Fluid USDC Universal Contract

Anthony Yang

April 2025

1. High-Level Flow

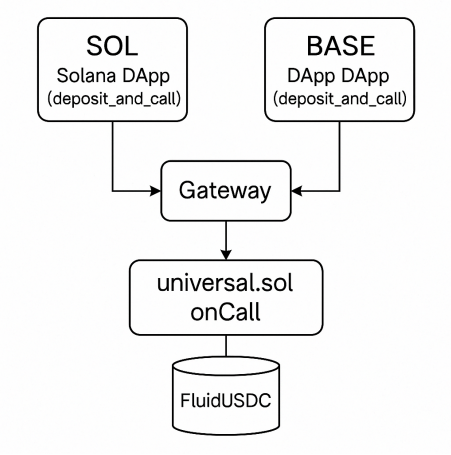

Figure 1: Universal Contract Flow

1. **Deposit**
   - **Solana/Base Side:**
     - User signs one transaction:
       - `deposit_and_call` (Solana)
       - `depositAndCall` (Base)
       - Gateways lock native USDC.
   - **Universal Contract:**
     - Mint ZRC-20 (`USDC.SOL` or `USDC.BASE`).
     - Call `onCall(...)` with `(zrc20Address, amount, message)`.
     - Call `CurveStableSwapNG.add_liquidity([...], minMint)`.
     - Mint LP token `USDC.4` and either forward to `context.sender` or hold it.
2. **Withdraw**
   - User calls `withdrawAndCall` on universal contract with destination chain + receiver.
   - Contract burns `USDC.4`, receives a ZRC-20 via `remove_liquidity_one_coin`, and approves GatewayZEVM.
   - Gateway burns ZRC-20 and sends real USDC to the receiver.

2. Starting from `Universal.sol`

Original Contract on docs

```
// SPDX-License-Identifier: MIT
pragma solidity 0.8.26;

import {RevertContext, RevertOptions} from "@zetachain/protocol-contracts/
    contracts/Revert.sol";
import "@zetachain/protocol-contracts/contracts/zevm/interfaces/UniversalContract.
    sol";
import "@zetachain/protocol-contracts/contracts/zevm/interfaces/IGatewayZEVM.sol";
import "@zetachain/protocol-contracts/contracts/zevm/GatewayZEVM.sol";

contract Universal is UniversalContract {
    GatewayZEVM public immutable gateway;

    event HelloEvent(string, string);
    event RevertEvent(string, RevertContext);

    error TransferFailed();
    error Unauthorized();

    modifier onlyGateway() {
        if (msg.sender != address(gateway)) revert Unauthorized();
        _;
    }

    constructor(address payable gatewayAddress) {
        gateway = GatewayZEVM(gatewayAddress);
    }

    function call(
        bytes memory receiver,
        address zrc20,
        bytes calldata message,
        CallOptions memory callOptions,
        RevertOptions memory revertOptions
    ) external {
        (, uint256 gasFee) = IZRC20(zrc20).withdrawGasFeeWithGasLimit(
            callOptions.gasLimit
        );
        if (!IZRC20(zrc20).transferFrom(msg.sender, address(this), gasFee)) {
            revert TransferFailed();
        }
        IZRC20(zrc20).approve(address(gateway), gasFee);
        gateway.call(receiver, zrc20, message, callOptions, revertOptions);
    }

    function callMulti(
        bytes[] memory receiverArray,
        address[] memory zrc20Array,
        bytes calldata messages,
        CallOptions memory callOptions,
        RevertOptions memory revertOptions
    ) external {
        for (uint256 i = 0; i < zrc20Array.length; i++) {
            (, uint256 gasFee) = IZRC20(zrc20Array[i])
```

```
            .withdrawGasFeeWithGasLimit(callOptions.gasLimit);
        if (
            !IZRC20(zrc20Array[i]).transferFrom(
                msg.sender,
                address(this),
                gasFee
            )
        ) {
            revert TransferFailed();
        }
        IZRC20(zrc20Array[i]).approve(address(gateway), gasFee);
        gateway.call(
            receiverArray[i],
            zrc20Array[i],
            messages,
            callOptions,
            revertOptions
        );
    }
}

function withdraw(
    bytes memory receiver,
    uint256 amount,
    address zrc20,
    RevertOptions memory revertOptions
) external {
    (address gasZRC20, uint256 gasFee) = IZRC20(zrc20).withdrawGasFee();
    uint256 target = zrc20 == gasZRC20 ? amount + gasFee : amount;
    if (!IZRC20(zrc20).transferFrom(msg.sender, address(this), target)) {
        revert TransferFailed();
    }
    IZRC20(zrc20).approve(address(gateway), target);
    if (zrc20 != gasZRC20) {
        if (
            !IZRC20(gasZRC20).transferFrom(
                msg.sender,
                address(this),
                gasFee
            )
        ) {
            revert TransferFailed();
        }
        IZRC20(gasZRC20).approve(address(gateway), gasFee);
    }
    gateway.withdraw(receiver, amount, zrc20, revertOptions);
}

function withdrawAndCall(
    bytes memory receiver,
    uint256 amount,
    address zrc20,
    bytes calldata message,
    CallOptions memory callOptions,
    RevertOptions memory revertOptions
) external {
    (address gasZRC20, uint256 gasFee) = IZRC20(zrc20)
        .withdrawGasFeeWithGasLimit(callOptions.gasLimit);
    uint256 target = zrc20 == gasZRC20 ? amount + gasFee : amount;
```

```
        if (!IZRC20(zrc20).transferFrom(msg.sender, address(this), target))
            revert TransferFailed();
        IZRC20(zrc20).approve(address(gateway), target);
        if (zrc20 != gasZRC20) {
            if (
                !IZRC20(gasZRC20).transferFrom(
                    msg.sender,
                    address(this),
                    gasFee
                )
            ) {
                revert TransferFailed();
            }
            IZRC20(gasZRC20).approve(address(gateway), gasFee);
        }
        gateway.withdrawAndCall(
            receiver,
            amount,
            zrc20,
            message,
            callOptions,
            revertOptions
        );
    }

    function onCall(
        MessageContext calldata context,
        address zrc20,
        uint256 amount,
        bytes calldata message
    ) external override onlyGateway {
        string memory name = abi.decode(message, (string));
        emit HelloEvent("Hello on ZetaChain", name);
    }

    function onRevert(
        RevertContext calldata revertContext
    ) external onlyGateway {
        emit RevertEvent("Revert on ZetaChain", revertContext);
    }
}
```

Key Extensions

```
import {ICurveStableSwapNG} from "./interfaces/ICurveStableSwapNG.sol";
import {SafeERC20, IERC20} from "openzeppelin/contracts/token/ERC20/utils/
    SafeERC20.sol";

contract FluidUSDCUniversal is Universal {
    using SafeERC20 for IERC20;

    address public immutable pool;
    int128 private constant IDX_SOL  = 1;
    int128 private constant IDX_BASE = 2;

    constructor(address payable gateway_, address pool_)
        Universal(gateway_) { pool = pool_; }
```

```
    function onCall(
        MessageContext calldata context,
        address zrc20In,
        uint256 amount,
        bytes calldata message
    ) external override onlyGateway {
        (uint8 cmd, uint256 minMint) = abi.decode(message,(uint8,uint256));
        if (cmd == 0) {
            _addLiquidity(zrc20In, amount, minMint, context.sender);
        }
        emit HelloEvent("Liquidity added", "USDC");
    }

    function _addLiquidity(
        address zrc20, uint256 amount, uint256 minMint, address lpReceiver
    ) internal {
        IERC20(zrc20).safeIncreaseAllowance(pool, amount);

        uint256[4] memory amounts;
        if (zrc20 == USDC_SOL) amounts[1] = amount;
        else if (zrc20 == USDC_BASE) amounts[2] = amount;
        else revert("unsupported ZRC20");

        uint256 lpOut = ICurveStableSwapNG(pool)
                        .add_liquidity(amounts, minMint, lpReceiver);
        require(lpOut > 0, "mint failed");
    }
}
```

4. Key Contract Addresses

- **GatewayZEVM (testnet)** – found in https://www.zetachain.com/docs/reference/network/contracts
  - SOL Gateway: ZETAjseVjuFsxdRxo6MmTCvqFwb3ZHUx56Co3vCmGis
  - BASE Gateway: 0x0c487a766110c85d301d96e33579c5b317fa4995
- **Pool (USDC.4)** – `0xCA4b0396064F40640F1d9014257a99aB3336C724`
- **ZRC-20 Tokens**:
  - USDC.SOL: `0x8344d6f84d26f998fa070BbEa6D2E15E359e2641`
  - USDC.BASE: `0x96152E6180E085FA57c7708e18AF8F05e37B479D`

Bottom Line

- `Universal.sol` is the new universal contract.
- Extend `onCall` to route USDC deposits into CurveStableSwapNG.