why contract deployment approach in Starknet and ZKSync Era are not the same with rest of rollups

I recently delved into the design intricacies of rollup technologies and stumbled upon an intriguing design decision that piqued my interest, particularly in the way contracts are deployed. My curiosity was sparked by StarkNet's unique deployment method, which led me to compare it with other protocols like Scroll, Polygon ZKEVM, Arbitrum, and zkSync Era to understand the underlying reasons.

Here's a brief overview of what I discovered:

Rollups can be differentiated by several criteria, one of which is how they publish Layer-2 changes to the Data Availability (DA) layer. The two main approaches I identified are:

- Transaction Compression, used by Scroll-Polygon ZKEVM and Arbitrum.
- State Difference Publishing, employed by StarkNet and zkSync Era.

This distinction highlighted a significant difference, but it didn't fully explain the variances in contract deployment methods.

To shed light on this, let's discuss the typical contract deployment process in Ethereum:

a single transaction type is used to deploy contracts by including the init byteCode in the transaction data section and sending it to a zero address

. This method is widely adopted by most EVM-compatible Layer-2 solutions. (and also there is create2 approach for contract deploying)

However, StarkNet and zkSync Era introduce a different approach:

1. First, they publish the byteCode on Layer-1 using a messaging system.
2. Once the byteCode is finalized and persisted on Layer-1, another transaction is sent to create an instance of the contract, pointing to the published byteCode. This enables the creation of multiple contract instances from the same byteCode.

[

939×741 53.8 KB

](https://ethresear.ch/uploads/default/original/2X/a/ac533981ee240e64db1e2a1827ab1d1fa75ee1a5.png)

Upon further investigation, I discovered that this methodology is largely influenced by their utilization of native account abstraction

.

Both StarkNet and zkSync Era treat all addresses as smart contracts with their own smarcontract code, embracing account abstraction. This design choice avoids the need to publish identical code for every address, allowing them to publish code once for account abstraction and have Layer-2 accounts reference it and the same for other contracts.

You can find more related links in here:

HackMD

Contract Deployment Design Space in L2 - HackMD

Data Availability Problem in context of L2 smarcontract source code