How to Deploy a Rollup chain using the Orbit SDK

This document explains how to use the Orbit SDK to deploy a Rollup Orbit chain .

UNDER CONSTRUCTION This document is under construction and may change significantly as we incorporate style guidance and feedback from readers. Feel free to request specific clarifications by clicking the Request an update button at the top of this document. info See the "create-rollup-eth" example in the Orbit SDK repository for additional guidance. The main benefit of the Orbit SDK lies in facilitating the deployment and fine-tuning of Orbit chains core Smart-Contracts.

These contracts are deployed on parent chains , they are:

- Rollup contracts
- Bridge contracts
- Contracts handling fraud proofs

Core contracts are the backbone of Arbitrum's Nitro stack , ensuring its robust and efficient operation. You can explore their code in the nitro-contracts GitHub repository .

Rollup Deployment Parameters Configuration

The createRollup function in the RollupCreator contract takes a complex input named deployParams , structured to encapsulate various configurable parameters customizing the Orbit chain.

Breaking down the structure of these parameters:

1. RollupDeploymentParams Struct

struct

RollupDeploymentParams

{ Config config ; address batchPoster ; address [ ] validators ; uint256 maxDataSize ; address nativeToken ; bool deployFactoriesToL2 ; uint256 maxFeePerGasForRetryables ; } This Solidity struct includes key settings like the chain configuration (Config), validator addresses, maximum data size, the native token of the chain, and more.

2. Config Struct

struct

Config

{ uint64 confirmPeriodBlocks ; uint64 extraChallengeTimeBlocks ; address stakeToken ; uint256 baseStake ; bytes32 wasmModuleRoot ; address owner ; address loserStakeEscrow ; uint256 chainId ; string chainConfig ; uint64 genesisBlockNum ; ISequencerInbox . MaxTimeVariation sequencerInboxMaxTimeVariation ; } The Config struct defines the chain's core settings, including block confirmation periods, stake parameters, and the chain ID.

3. MaxTimeVariation Struct

struct

MaxTimeVariation

{ uint256 delayBlocks ; uint256 futureBlocks ; uint256 delaySeconds ; uint256 futureSeconds ; } This nested struct within Config specifies time variations related to block sequencing, providing control over block delay and future block settings.

4. chainConfig

The chainConfig parameter within the Config struct allow you to customize your Orbit chain. It's a stringified JSON object containing various configuration options that dictate how the Orbit chain behaves and interacts with the parent chain network.

Here's a brief overview of chainConfig :

{ chainId : number; homesteadBlock : number; daoForkBlock :

null ; daoForkSupport : boolean; eip150Block : number; eip150Hash : string; eip155Block : number; eip158Block : number; byzantiumBlock : number; constantinopleBlock : number; petersburgBlock : number; istanbulBlock : number; muirGlacierBlock : number; berlinBlock : number; londonBlock : number; clique :

{ period : number; epoch : number; } ; arbitrum :

{ EnableArbOS : boolean; AllowDebugPrecompiles : boolean; DataAvailabilityCommittee : boolean; InitialArbOSVersion : number; InitialChainOwner : Address; GenesisBlockNum : number; MaxCodeSize : number; MaxInitCodeSize : number; } ; } Out of these parameters, a few are particularly important and are likely to be configured by the chain owner:chainId ,DataAvailabilityCommittee ,InitialChainOwner ,MaxCodeSize , andMaxInitCodeSize . While part of thechainConfig , the other parameters typically use default values and are less frequently modified. We will detail these parameters in theRollup Configuration Parameters section. Additionally, we'll guide you through using the Orbit SDK to effectively set and customize these configurations, ensuring that your Orbit chain is tailored to your specific requirements and operational needs.

All the parameters explained in this section are customizable, allowing the chain deployer to stick with default settings or specify new values. In the upcoming sections, we will dive deeper into what each parameter represents and how you can utilize the Orbit SDK to configure them effectively for your Orbit chain deployment.

For an easier config preparation, the Orbit SDK provides theprepareChainConfig API, which takes config parameters as arguments and returns achainConfig JSON string. Any parameters not provided will default to standard values, which are detailed in theOrbit SDK documentation .

Here are the parameters you can use withprepareChainConfig :

Parameter Description chainId Your Orbit chain's unique identifier. It differentiates your chain from others in the ecosystem. DataAvailabilityCommittee Set tofalse , this boolean makes your chain as a Rollup, set totrue configures it as an AnyTrust chain. InitialChainOwner Identifies who owns and controls the chain. MaxCodeSize Sets the maximum size for contract bytecodes on the Orbit chain. e.g. Ethereum mainnet has a limit of 24,576 Bytes. MaxInitCodeSize Similar toMaxCodeSize , defines the maximum size for your Orbit chain'sinitialization code. e.g. Ethereum mainnet limit is 49,152 Bytes. Below is an example of how to useprepareChainConfig to set up a Rollup chain with a specificchainId , anInitialChainOwner (named asdeployer_address):

import

{ prepareChainConfig }

from

'@arbitrum/orbit-sdk' ;

const chainConfig =

prepareChainConfig ({ chainId :

Some_Chain_ID , arbitrum :

{

InitialChainOwner : deployer_address ,

DataAvailabilityCommittee :

false

} , }) ;

Rollup Configuration Parameters

In this section, we'll provide detailed explanations of the various chain configuration parameters used in the deployment of Orbit chains. Understanding these parameters is critical to customizing your Orbit chain to suit your needs.

Parameter Description batchPoster Sets the batch poster address of your Orbit chain. The batch poster account batches and compresses transactions on the Orbit chain and transmits them back to the parent chain. validators Array ofvalidator addresses. Validators are responsible for validating the chain state and posting Rollup Blocks (RBlocks) back to the parent chain. They also monitor the chain and initiate challenges against potentially faulty RBlocks submitted by other validators. nativeToken Determines the token used for paying gas fees on the Orbit chain. It can be set toETH for regular chains or to anyERC-20 token forgas fee token network Orbit chains. confirmPeriodBlocks Sets the challenge period in terms of blocks, which is the time allowed for validators to dispute or challenge state assertions. On Arbitrum One and Arbitrum Nova, this is currently set to approximately seven days in block count.confirmPeriodBlocks is measured in L1 blocks, we recommend a value of45818 baseStake Orbit chain validator nodes must stake a certain amount to incentivize honest participation. Thebasestake parameter specifies this amount. stakeToken Token in which thebasestake is required. It represents the token's address on the parent chain. Can beETH or aERC-20 token. Note that the use of anERC-20 token as thestakeToken is currently not supported by Nitro, but will be soon. owner Account address responsible for deploying, owning, and managing your Orbit chain's base contracts on its parent chain. chainId Sets the unique chain ID of your Orbit chain. note chainId and owner parameters must be equal to the chain ID andInitialOwner defined in thechainConfig section. While other configurable parameters exist, they are set to defaults, and it's generally not anticipated that a chain deployer would need to modify them. However, if you believe there's a need to alter any other parameters not listed here, please feel free tocontact us on our Discord server for further details and support.

Configuration of Rollup Params and Deployment on Orbit SDK

The Orbit SDK provides two APIs,createRollupPrepareConfig andcreateRollupPrepareTransactionRequest to facilitate the configuration and deployment of Rollup parameters for an Orbit chain. These APIs simplify the process of setting up and deploying the core contracts necessary for an Orbit chain.

createRollupPrepareConfig API

:

This API is designed to take parameters defined in the Config struct and fill in the rest with default values. It outputs a complete Config struct that is ready for use.

For example, to create a Config struct with a specific chain ID (chainId), an owner address (deployer_address), and achainConfig as described in theprevious section , you would use the Orbit SDK as follows:

import

{ createRollupPrepareConfig }

from

'@arbitrum/orbit-sdk' ;

const config =

createRollupPrepareConfig ({ chainId :

BigInt (chainId) , owner : deployer . address , chainConfig , }) ;

createRollupPrepareTransactionRequest API :

This API accepts parameters defined in theRollupDeploymentParams struct, applying defaults where necessary, and generates theRollupDeploymentParams . This struct is then used to create a raw transaction which calls thecreateRollup function of theRollupCreator contract. As discussed in previous sections, this function deploys and initializes all core Orbit contracts.

For instance, to deploy using the Orbit SDK with a Config equal toconfig , abatchPoster , and a set of validators such as[validator] , the process would look like this:

import

{ createRollupPrepareTransactionRequest }

from

'@arbitrum/orbit-sdk' ;

const request =

await

createRollupPrepareTransactionRequest ({ params :

{ config , batchPoster , validators :

[validator] , } , account : deployer_address , publicClient , }) ; After creating the raw transaction, you need to sign and broadcast it to the network.

Getting the Orbit Chain Information After Deployment

Once you've successfully deployed your Orbit chain, the next step is to retrieve detailed information about the deployment. The Orbit SDK provides a convenient way to do this through thecreateRollupPrepareTransactionReceipt API.

After sending the signed transaction and receiving the transaction receipt, you can use thecreateRollupPrepareTransactionReceipt API to parse this receipt and extract the relevant data. This process will provide comprehensive details about the deployed chain, such as contract addresses, configuration settings, and other information.

Here's an example of how to use the Orbit SDK to get data from a deployed Orbit chain:

import

{ createRollupPrepareTransactionReceipt }

from

'@arbitrum/orbit-sdk' ;

const data =

createRollupPrepareTransactionReceipt (txReceipt) ; In this example,txReceipt refers to the transaction receipt you received after deploying the chain. By passing this receipt to thecreateRollupPrepareTransactionReceipt function, you can easily access a wealth of information about your Orbit chain. This feature of the Orbit SDK simplifies the post-deployment process, allowing you to quickly and efficiently gather all necessary details about your chain for further use or reference. Edit this page Last updatedonApr 2, 2024 Previous Get started Next Deploy an AnyTrust chain