

Migrating Arbitrum Orbit Chains to Espresso

Espresso Nitro Integration

You may wish to familiarize yourself with the [security audits](#) for this integration.

The target audience for this document are developers who would like to migrate an existing Arbitrum Orbit chain to use Espresso's Global Confirmation Layer (GCL). The document should also be interesting for developers who are thinking of deploying a new Arbitrum Orbit chain that uses Espresso's GCL.

The goal of this document is to describe how to migrate an Arbitrum Orbit chain to using Espresso's global confirmation layer for fast confirmations. By leveraging Espresso's fast confirmations, a rollup will only accept a batch after it has been finalized by Espresso. This integration ensures that each batch processed by the rollup is consistent with HotShot-finalized blocks within its namespace.

Quickstart

The quickest way to get familiar with what the migration entails is to run through our [migration test](#).

The script creates up an ephemeral Nitro chain on the local computer and migrates it to using Espresso's Global Confirmation Layer (also running locally on the computer). This test mocks out the TEE part by using a mocked TEE verifier contract and configured the batch poster to bypass the TEE interactions. Thereby it's possible to run through the migration locally, without requiring a special CPU and extra software to support the TEE.

For a safe production deployment the TEE is required and the real TEE verifier contract must be deployed.

The test requires [Docker](#), [Foundry](#), [Yarn](#), openssl and jq to be installed.

For convenience the nix package manager can take care of installing all dependencies except for Docker. Nix can be installed by running

Copy

```
curl --proto '=https' --tlsv1.2 -sSf -L https://install.determinate.systems/nix | sh -s -- install
```

To run the test first some setup:

Copy

```
git clone --recursive https://github.com/EspressoSystems/nitro-testnode
```

```
cd nitro-testnode
```

```
git checkout integration
```

```
nix develop # omit if you have foundry, yarn, jq, openssl installed already
```

Now you can run the test itself.

Copy

```
espresso-tests/migration-test.bash
```


You may need to run the following command to ensure the submodules are initialized:

Copy

```
git submodule update --init --recursive
```

You should see substantial output. The first time you run this, it can take up to 15-20 minutes due to initial setup steps like installing dependencies. Subsequent runs should take less than 10 minutes.

After some time, if successful, you should see **Migration successfully completed!**.

 **Warning:** If you see "Waiting for confirmed nodes" logs for more than 60 seconds, it's recommended to restart the process.

It's encouraged to read through the [migration test](#) to get an idea of all the steps involved and what information is required for each step. For the real migration using .env files instead of environment variables to provide the necessary inputs may be more convenient.

Once familiar, you might run some commands to further your understanding. Transfer some eth, for example:

Copy

```
cast send $RECIPIENT_ADDRESS --value 1ether --rpc-url $CHILD_CHAIN_RPC_URL --private-key $PRIVATE_KEY
```

Assumptions

Before attempting a migration please verify that the assumptions listed here make sense for your nitro deployment. If not, please get in touch with us via our [contact form](#).

1. The Orbit chain to be migrated is using vanilla Nitro stack, or Celestia DA via the [Celestia fork of nitro](#). If your Orbit chain is using Celestia for DA you need to use the celestia-integration branches of Espresso's nitro forks.
2. You are able to run the batch poster in an SGX TEE environment. This is required to provide TEE attestations that the L2 transactions have been finalized by Espresso. These attestations are verified in the TEE verifier smart contract on the parent chain as part of the batch submission.

Production Setup Requirements

For production migrations, you will need the following system requirements (note: these are not needed for running the test migration):

- SGX
- Gramine
- [16GB RAM](#)

You will also need these Espresso projects. Depending on whether you use celestia DA or not, you will need either the integration branch or celestia-integration branch. Please take the time to review each project's respective README.

- <https://github.com/EspressoSystems/nitro-espresso-integration>

- <https://github.com/EspressoSystems/nitro-contracts>
- <https://github.com/EspressoSystems/nitro-testnode>
- <https://github.com/EspressoSystems/orbit-actions>
- <https://github.com/EspressoSystems/gsc>

As well as [Espresso's nitro-node](#) docker image:

Copy

```
ghcr.io/espressosystems/nitro-espresso-integration/nitro-node:v3.3.2-fcd633f
```

Migration Flow

This flow is more precisely defined in our [migration test](#). Since that script is intended for testing migrations, it does not not require SGX. Therefore it should not be use for production deployments.

1. Run the new batch poster inside SGX and get the MREnclave and MRSigner values (which is the hash of the code running inside the SGX). These values are needed to deploy the EspressoTEEVERifier contract.
2. Deploy the EspressoTEEVERifier contract.
3. Stop Nitro node with the batch poster and copy the batch poster's databases files to the TEE.
4. Perform the sequencer inbox migration.
5. Run new Batch poster. You will notice it starts catching up messages and building up state.

Please also be aware of [steps to revert a migration](#) should you need them.

Verify SGX Setup

Setup of SGX TEE is beyond the scope of this document. We have had success with [Azure SGX VMs](#)

First verify you have linux version with in-kernel SGX. To ensure you have in-kernel SGX run the following command to verify you are running a Linux kernel greater than version 5.11.

Copy

```
uname -r
```

Ensure that your machine has an [Intel SGX2](#) (Software Guard Extensions) enabled CPU to run our batch poster (See [here](#) and [here](#)). You can verify if your CPU supports SGX2 on Linux by inspecting CPU information:

Copy

```
grep -i sgx /proc/cpuinfo
```

If your CPU supports SGX, the output should resemble the following. If it does not, your CPU either doesn't support SGX2, or it isn't enabled in the BIOS.

Copy

SGX2 supported = true

Install Gramine

To install gramine follow the following instructions:

1. Install [software packages](#)
2. Setup [host configuration](#)
3. Install the [SGX software stack](#)
4. If your machine is running on Microsoft Azure, you can refer to [this document](#) for configuring aesm.

You don't need to follow the [build gramine steps](#) as we will be using [Gramine Shielded Containers \(GSC\)](#)

At this point systemctl status aesmd should report a healthy service (**Active: active (running)**).

If you see any errors here, ensure your aesmd is configured properly. If your machine is running on Microsoft Azure, you can refer to [this document](#) for configuring aesm.

Running the Batch Poster Inside SGX

Building the Poster Image

These steps need to happen inside SGX

Use our default [Dockerfile.sgx-poster](#) or the Celestia [Dockerfile.sgx-poster](#) to build the sgx-poster Docker image.

First obtain the source:

Copy

```
git clone https://github.com/EspressoSystems/nitro-espresso-integration
```

```
cd nitro-espresso-integration
```

- Alternative 1: using default Nitro stack

Copy

```
git checkout integration
```

- Alternative 2: using Celestia DA

Copy

```
git checkout celestia-integration
```

Then build the image.

Copy

```
docker build -f Dockerfile.sgx-poster -t sgx-poster .
```

Building the Gramine Image

Once the docker image is built, you need to build a Gramine Shielded Container (GSC) image.

The gsc python script requires a few python packages. For ubuntu 24.04 run

Copy

```
sudo apt-get install -y python3 python3-docker python3-jinja2 python3-tomli python3-tomli-w python3-yaml
```

For more information see the [GSC docs](#).

Clone Espresso's gsc repo and build the Gramine image:

Copy

```
git clone https://github.com/EspressoSystems/gsc.git
```

```
cd gsc
```

```
git checkout master
```

```
cp config.yaml.template config.yaml
```

```
./gsc build sgx-poster ./nitro-espresso.manifest
```

Now before building the gramine image, you need to edit your poster_config.json file to include the following fields (given parent chain is arbitrum sepolia):

Copy

```
"batch-poster": {  
  "hotshot-url": "https://query.decaf.testnet.espresso.network/v0",  
  "light-client-address": "0x08d16cb8243b3e172dddcdf1a1a5dacca1cd7098",  
  "resubmit-espresso-tx-deadline": "2m"  
},  
"transaction-streamer": {  
  "user-data-attestation-file": "/dev/attestation/user_report_data",  
  "quote-file": "/dev/attestation/quote"  
}
```

You need the sha256 hash of your poster_config.json file. You can get the hash using the following command:

Copy

```
sha256sum poster_config.json
```

Replace the nitro-espresso.manifest in the gsc with these contents and replace the <YOUR_SHA256_HERE> with the sha256 of your poster_config.json file.

Copy

```
sys.enable_extra_runtime_domain_names_conf = true

sgx.edmm_enable = true

sgx.remote_attestation = "dcap"

sgx.use_exinfo = true

sys.experimental__enable_flock = true

fs.mounts = [

  { path = "/home/user/.arbitrum/", uri = "file:/home/user/.arbitrum/"},

  { path = "/config/", uri = "file:/config"}

]

sgx.allowed_files = ["file:/home/user/.arbitrum"]

[[sgx.trusted_files]]

uri = "file:/home/user/kzg10-aztec20-srs-1048584.bin"

sha256 = "cded83e82e4b49fee4cb2e0f374f996954fe12548ad39100432ee493069ef09d"

[[sgx.trusted_files]]

uri = "file:/config/poster_config.json"

sha256 = "<YOUR_SHA256_HERE>"
```

Next we will need to sign the image in order to run this container inside the SGX enclave.

Generate the signing key (if you don't already have one).

Copy

```
openssl genrsa -3 -out enclave-key.pem 3072
```

PLEASE KEEP THIS KEY SAFE in some local private storage, but delete it from the server after you have signed.

Sign the container

Copy

```
./gsc sign-image sgx-poster enclave-key.pem
```

The final step is to run the container inside the SGX enclave. This requires a config folder which contains the poster_config.json file (available from the legacy batch poster).

Finally we also need a .arbitrum folder which contains the state of the batch poster. At this point it can be an empty folder but once the legacy batch poster is shut down, we should fill this up with the contents of the legacy batch poster and re-start the poster.

Run the batch poster using the following command, replacing \$CONFIG_PATH with the actual path to these folders on your host machine.

These folders are mounted in the docker container, so any changes to them on the host change them in the container.

Copy

```
docker run \
```

```
--device=/dev/sgx_enclave \
```

```
-v /var/run/aesmd/aesm.socket:/var/run/aesmd/aesm.socket \
```

```
-v $CONFIG_PATH/.arbitrum:/home/user/.arbitrum \
```

```
-v $CONFIG_PATH/config:/config \
```

gsc-sgx-poster

At this stage, you will see an attestation report similar to the following. The hex value, is the report data which contains the MR_ENCLAVE (the hash of the code running inside SGX) and the MR_SIGNER. After you see the attestation report, you can shut down the batch poster.

Copy

[illegible]

Successfully read attestation report.

You can decode all this information using the following steps:

1. Create a report.txt file with the hex value and create a bin file as follows:

Copy

```
xxd -r -p report.txt report.bin
```

2. Use [this](#) bash script to decode the binary, it will print out the MR_ENCLAVE and MR_SIGNER.

Deploying the EspressoTEEverifier contract

Contract Deployment

You will need to use our version of [nitro contracts](#). The instructions below assume you are using celestia for DA. If not, simply checkout on integration branch instead.

Please clone this repo at the given branch to follow the next steps.

Copy

```
git clone --recurse-submodules git@github.com:EspressoSystems/nitro-contracts.git
```

```
git checkout celestia-integration
```

To run this script to deploy a rollup on arbSepolia follow the given steps:

Compile the contracts using yarn

Copy

```
yarn install && forge install
```

```
yarn build:all
```

Create a .env file with variables from the previous steps

Copy

```
DEVNET_PRIVKEY="" # Private key with funds on Arbitrum Sepolia.
```

```
MR_ENCLAVE="MR_ENCLAVE_VALUE" # MR_ENCLAVE value from the last step
```

```
MR_SIGNER="MR_SIGNER_VALUE" # MR_SIGNER value from the last step
```

```
ARBISCAN_API_KEY="" # [Arbscan API Key](https://docs.arbiscan.io/getting-started/viewing-api-usage-statistics)
```

Deploy the contract

Copy

```
npx hardhat run scripts/deployEspressoTEEverifier.ts --network arbSepolia
```

Sequencer Inbox Migration

Obtain the Contracts

In this section we will be working out of our orbit-actionsrepo. Again the choice of integration or celestia-integration depends on DA.

Copy

```
git clone git@github.com:EspressoSystems/nitro-contracts.git
```


Be aware that the contract to deploy the sequencer inbox needs to locally deploy a mock ArbitrumChecker to prevent foundry from declaring calls to precompiles as invalid opcodes. This shouldn't affect on chain deployment from these scripts, nor should it affect the onchain execution of the migration action.

Configuration

Create a .env file in the orbit-actions directory that contains the following values:

Copy

#The private key for use during the migration. This should be the rollup owner's private key for steps involving performing the migration.

PRIVATE_KEY=""

Environment variables for chain name and rpc_url

These are essential for the upgrade

PARENT_CHAIN_CHAIN_ID=""

PARENT_CHAIN_RPC_URL=""

Addresses to the upgrade executors on both chains.

These are essential for the upgrade

PARENT_CHAIN_UPGRADE_EXECUTOR=""

Environment variables for the sequencer inbox migration action contract

ROLLUP_ADDRESS=""

PROXY_ADMIN_ADDRESS=""

The reader address should be set to the zero address for orbit chains, for other chains, set this to the same address as your current reader for the sequencer inbox

READER_ADDRESS="0x0"

IS_USING_FEE_TOKEN=""

MAX_DATA_SIZE="104857" #for orbit chains, use this value, for chains posting to ethereum, use 117964

The old batch poster address will be removed from the sequencer inbox proxy to ensure only the batch poster running in the TEE will be allowed to post batches.

OLD_BATCH_POSTER_ADDRESS=""

NEW_BATCH_POSTER_ADDRESS=""

The new batch poster address and batch poster manager address will be provided to you to run the migration.

```
BATCH_POSTER_MANAGER_ADDRESS=""
```

This should be the address of the contract you deployed in the last step.

```
ESPRESSO_TEE_VERIFIER_ADDRESS=""
```

This will allow you to deploy a reverting sequencer inbox migration action if desired (there are additional steps in the migration readme)

```
IS_REVERT=""
```

Source it.

Copy

```
source ./env
```

Install all dependencies.

Copy

```
yarn
```

```
yarn prepare
```

```
yarn build
```

Run the migration deployment scripts

Run the migration deployment scripts for the parent chain. Including both

DeployAndInitEspressoSequencerInbox.s.sol, and

DeployEspressoSequencerInboxMigrationAction.s.sol. From the base directory of the orbit actions repo, you can use the following commands to run these scripts:

DeployAndInitEspressoSequencerInbox.s.sol

Copy

```
forge script --chain $PARENT_CHAIN_CHAIN_ID contracts/parent-chain/espresso-  
migration/DeployAndInitEspressoSequencerInbox.s.sol:DeployAndInitEspressoSequencerInbo  
x --rpc-url $PARENT_CHAIN_RPC_URL --broadcast -vvvv --skip-simulation
```

Before you proceed: make sure to store the address of the new SequencerInbox in the environment variable NEW_SEQUENCER_INBOX_IMPL_ADDRESS

DeployEspressoSequencerInboxMigrationAction.s.sol

Copy

```
forge script --chain $PARENT_CHAIN_CHAIN_ID contracts/parent-chain/espresso-  
migration/DeployEspressoSequencerMigrationAction.s.sol:DeployEspressoSequencerMigratio  
nAction --rpc-url $PARENT_CHAIN_RPC_URL --broadcast -vvvv --skip-simulation
```

Before you proceed: make sure to store the address of the new SequencerInbox in the environment variable SEQUENCER_MIGRATION_ACTION

Execute the Upgrade

The final step for executing the migration involves using cast to call the perform() function of the sequencer inbox migration action via the upgrade executor. You can use the following command to accomplish this:

Copy

```
cast send $PARENT_CHAIN_UPGRADE_EXECUTOR "execute(address, bytes)"
$SEQUENCER_MIGRATION_ACTION $(cast calldata "perform()") --rpc-url
$PARENT_CHAIN_RPC_URL --private-key $PRIVATE_KEY
```

After running this command, your rollup contracts should be set up to accept batches from your new batch poster.

Run the Batch Poster with Legacy State

[As stated in upstream documentation](#), you need to copy the contents of the .arbitrum folder of the legacy batch poster to the .arbitrum folder of the new

Then re-start the batch poster.

Verify the Migration

You should be able to see batch sent logs once the batcher starts posting batches. This would indicate that the batcher has started successfully.

Nitro Testnet

TL;DR

The Espresso Network is a confirmation layer that provides chains with information about the state of their own chain and the states of other chains, which is important for cross-chain composability. Espresso confirmations can be used in addition to the soft confirmations from a centralized sequencer, are backed by the security of the Espresso Network, and are faster than waiting for Ethereum finality (12-15 minutes).

Overview

Purpose

This document describes how the Espresso Network provides fast confirmations to Arbitrum Orbit chains. Espresso has developed a TEE based integration, which is ready for chain operators and rollup-as-a-service providers to implement. There is some assumed familiarity with the [Arbitrum Nitro stack](#).

How It Works

In a regular chain, the transaction lifecycle will look something like this:

1. A user transacts on an Arbitrum chain.
2. The transaction is processed by the chain's sequencer, which provides a soft-confirmation to the user, and the transactions are packaged into a block.
3. The sequencer, responsible for collecting these blocks, compressing, and submitting, submits the transactions to the base layer.

1. If the base layer is Arbitrum One or Ethereum, then the transaction will take at least 12-15 minutes to finalize, or longer depending on how frequently the sequencer posts to the base layer.
2. In this transaction lifecycle, the user must **trust** that the chain's sequencer provided an **honest soft-confirmation** and will not act **maliciously**. There are limited ways to verify that the sequencer and batcher acted honestly or did not censor transactions.

This is a strong assumption, and the key thing that the Espresso Network helps with. When the chain is integrated with the Espresso Network: The sequencer provides a soft-confirmation to the user, while the transactions are also sent to the Espresso Network to provide a stronger confirmation secured by BFT consensus. A software component of the sequencer called the **batch poster** (henceforth referred to as "batcher") is run inside a TEE and must honor the Espresso Network confirmation. It cannot change the ordering or equivocate. This gives a strong guarantee that the transaction will ultimately be included and finalized by the base layer.

The user must trust that the chain's sequencer provided an honest soft-confirmation; however the **Espresso Network provides a stronger confirmation that keeps the sequencer accountable** and prevents the sequencer from equivocating or acting maliciously. The initial implementation of the batch poster is permissioned and the user must trust that it will not reorder blocks produced by the sequencer.

Integration

Integrating with the Espresso Network requires minimal changes to Arbitrum Nitro's existing rollup design. The Espresso Team has already [done that](#), and in the following sections we will provide a comprehensive guide for running your own instance and building on Espresso!

Components

We model the rollup as a collection of three components:

- The **sequencer**
- The **batcher**
- The **TEE contract**

Transaction Flow

Chain Config

Local Deployment (`docker compose`)

Overview

For those seeking to evaluate their infrastructure and to get a clearer picture of what a "working" implementation looks like, we have made available a docker config that will allow for evaluation of the various components of the protocol needed to gain initial familiarity with the system.

Copy

services:

nitro:

image: ghcr.io/espressosystems/nitro-espresso-integration/nitro-node:integration

container_name: nitro-node

ports:

- "8547:8547"

- "8548:8548"

- "8549:8549"

command: --conf.file /config/full_node.json

volumes:

- ./config:/config

- ./wasm:/home/user/wasm/

- ./database:/home/user/.arbitrum

depends_on:

- validation_node

validation_node:

image: ghcr.io/espressosystems/nitro-espresso-integration/nitro-node:integration

container_name: validation_node

ports:

- "8949:8549"

volumes:

- ./config:/config

entrypoint: /usr/local/bin/nitro-val

command: --conf.file /config/validation_node_config.json