How to customize your Orbit chain's behavior

PUBLIC PREVIEW, MAINNET READY Orbit chains are now Mainnet ready ! Note that Orbit is still a public preview capability - the Orbit product and its supporting documentation may change significantly as we capture feedback from readers like you.

To provide feedback, click the Request an update button at the top of this document, join the Arbitrum Discord , or reach out to our team directly by completing this form .

Preface

Before customizing your Orbit chain, it's important to understand what the State Transition Function (aka the STF) is. The STF defines how new blocks are produced from input messages (i.e. transactions). This guide is only necessary for changes that modify the State Transition Function. To customize other node behavior, such as RPC behavior or the sequencer's ordering policy, you can simply build your own node without worrying about the rest of this guide. However, changes that modify the STF require updating the fraud proving system to recognize the new behavior as correct. Otherwise, the fraud prover would side with unmodified nodes, which would win fraud proofs against your modified node.

Here's some examples of modifications that affect the STF:

- Adding a new EVM opcode or precompile:
- This modifies the STF because a node with this change would disagree about the outcome of EVM execution compared to an unmodified Nitro node
- when the new opcode or precompile is invoked.
- Rewarding the deployer of a smart contract with a portion of gas spent in the smart contract's execution:
- This modifies the STF because a node which has this change applied would disagree about the balance of the deployer after transactions. Such changes would lead to disagreements about block hashes compared to unmodified Nitro nodes.

Here's some examples of modifications that don't affect the STF:

- Adding a new RPC method to query an address's balance across multiple blocks:
- This doesn't modify the STF because it doesn't change on-chain balances or block hashes.
- Changing the sequencer to order blocks by tip:
- The sequencer is trusted to order transactions in Arbitrum Nitro, and it can choose any ordering it wants.
- Nodes (and the fraud proofs) will simply accept the new transaction ordering as there is no single ordering they think is correct.

Modification compatibility with Arbitrum Nitro

Some potential modifications are incompatible with Arbitrum Nitro and would not result in a functioning blockchain. Here are some requirements for the Arbitrum Nitro State Transition Function:

- The STF must be deterministic. For instance, if you gave an address a random balance using the Go randomness library,
- every node would disagree on the correct amount of balance and the blockchain would not function correctly.
- However, it is acceptable to take a non-deterministic path to a deterministic output.
- For instance, if you randomly shuffled a list of addresses, and then gave them each 1 Ether, that would be fine,
- because no matter how the list of addresses is shuffled the result is the same and all addresses are given 1 Ether.
- The STF must not reach a new result for old blocks. For instance, if you have been running an Arbitrum Nitro chain for a while,
- and then you decide to modify the STF to not charge for gas, a new node that syncs the blockchain will reach a different result
- for historical blocks. It's also important to synchronize between nodes when an upgrade takes effect. A common mechanism
- for doing this is having an upgrade take effect at a certain timestamp, by which all nodes must be upgraded.
- The STF must be "pure" and not use external resources. For instance, it must not use the filesystem, make external network calls,
- or launch processes. That's because the fraud proving system does not (and for the most part, cannot) support these resources.
- For instance, it's impossible to fraud prove what the result of an external network call is, because the fraud prover smart contracts on L1 are unable to do networking.
- The STF must not carry state between blocks outside of the "global state". In practice this means persistent state must be stored within
- block headers or the Ethereum state trie. For instance, ArbOS stores all retryables in contract storage under a special ArbOS address.
- The STF must not modify Ethereum state outside of a transaction. This is important to ensure that replaying old blocks reaches the same result,
- both for tracing and for validation. The ArbOS internal transaction is useful to modify state at the start of blocks.

- The STF must reach a result in under a second. This is a rough rule, but for nodes to keep in sync it's highly recommended to keep blocks quick.
- It's also important for the fraud proofs that execution reliably finishes in a relatively short amount of time.
- A block gas limit of 32 million gas should safely fit within this limit.
- The STF must not fail or panic. It's important that the STF always produces a new block, even if user input is malformed.
- For instance, if the STF receives an invalid transaction as input, it'll still produce an empty block.

Building the modified node

To modify the State Transition Function, you'll need to build a modified Arbitrum Nitro node Docker image. This guide covers how to build the node and enable fraud proofs by building a new replay binary.

Step 1. Download the Nitro source code

Clone the Nitro repository before you begin:

git clone --branch v2.2.5 https://github.com/OffchainLabs/nitro.git cd nitro git submodule update --init --recursive --force

Step 2. Apply modifications

Next, make your changes to the State Transition Function. For example, you could add a custom precompile .

Step 3. Run the node without fraud proofs

To build the Arbitrum Nitro node image, you'll first need to install Docker. You can confirm if it's already setup by runningdocker version in a terminal. If not, try followingDocker's getting started guide , or if you're on Linux, install Docker from your distribution's package manager and start the Docker service.

Once you have Docker installed, you can simply rundocker build . --tag custom-nitro-node in thenitro folder to build your custom node.

Once you've built your new Nitro node image, you have to modify the configuration file of your node to add a new parameter,--node.staker.dangerous.without-block-validator , which disables fraud proof verification.

To do that, open yournodeConfig.json file (stored by default in theconfig folder) and add the new parameter inside thestaker element:

... "staker" :

{ ... "dangerous" :

{ "without-block-validator" :

true } ... } , ... With that, you have now two ways of running your node.

1. Using the docker-compose file

This is the recommended way if you're running your Orbit chain locally through the provideddocker-compose file . Indocker-compose.yml , modify the Docker image used for the Nitro container:

... nitro: image: custom-nitro-node ports: ... And rundocker compose up to run all of your containers.

1. Usedocker run to run your Nitro node only

This method will only run the customized Nitro node (i.e., it will not run Blockscout, or the DA server if you're using an AnyTrust chain). Use the following command:

docker run --rm -it -v /path/to/your/node/dir:/home/user/.arbitrum -p 0.0 .0.0:8449:8449 custom-nitro-node --conf.file /home/user/.arbitrum/nodeConfig.json info Note that the instructions provided inHow to run a full node will not work with your Orbit node. SeeCommand-line options (Orbit) for Orbit-specific CLI flags. Once your node is running, you can try out your modifications to the State Transition Function and confirm they work as expected.

Step 4. Enable fraud proofs

To enable fraud proofs, you'll need to build the "replay binary", which defines the State Transition Function for the fraud prover. The replay binary (sometimes called the machine) re-executes the State Transition Function against input messages to determine the correct output block. It has three forms:

- Thereplay.wasm

- binary is the Go replay binary compiled to WASM. It's used by the JIT validator to verify blocks against the fraud prover.
- Themachine.wavm.br
- binary is a compressed binary containing the Go replay binary and all its dependencies, compiled to WASM, then translated to the Arbitrum fraud proving variant WAVM.
- It's used by Arbitrator when actually entering a challenge and performing the fraud proofs,
- and has identical behavior toreplay.wasm
- .
- The WASM module root (stored inmodule-root.txt
-) is a 32 byte hash usually expressed in hexadecimal which is a merkelization ofmachine.wavm.br
- .
- The replay binary is much too large to post on-chain, so this hash is set in the L1 rollup contract to determine the correct replay binary during fraud proofs.

To run a validator node with fraud proofs enabled, the validator node's Docker image will need to contain all three of these versions of the replay binary.

4.1 Build a dev image

The simplest way to build a Docker image with the new replay binary is to build a dev image. These images contain a freshly built replay binary, but note that the replay binary and corresponding WASM module root will generally change when the code is updated, even if the State Transition Function has equivalent behavior. It's important that the validator's WASM module root matches the on-chain WASM module root, which is why this approach is harder to maintain. Over the longer term, you'll want to maintain a separate build of the replay binary that matches the one currently on-chain, usable by any node image.

To build the dev node image and get the WASM module root, run:

docker build . --target nitro-node-dev --tag custom-nitro-node-dev docker run --rm --entrypoint cat custom-nitro-node-dev target/machines/latest/module-root.txt Once you have the WASM module root, you can put it on-chain by callingsetWasmModuleRoot(newWasmModuleRoot) through the upgradeExecutor contract's methodexecuteCall() as the owner. To call this method, you need to settarget as your rollup contract address. Ensure thattargetCallData starts with0x89384960 (this is the signature ofsetWasmModuleRoot(byte32)), and that it's followed by your WASM module root. TheupgradeExecutor contract address and rollup contract address can be found in the chain deployment info JSON. You can confirm that the WASM module root was updated by callingwasmModuleRoot() on the rollup contract.

Once you have set the new WASM module root on-chain, you can re-enable fraud proofs and run your node.

To re-enable fraud proofs, open yournodeConfig.json file again, and remove the"dangerous" section (containing thewithout-block-validator property) that you previously added.

After that, you'll have, again, two ways of running your node.

1. Using the docker-compose file

As mentioned before, this is the recommended way if you're running your Orbit chain locally through the provided docker-compose file . Indocker-compose.yml , modify the Docker image used for the Nitro container. Notice that we'll now use thecustom-nitro-node-dev you just created:

... nitro: image: custom-nitro-node-dev ports: ... And rundocker compose up to run all of your containers.

1. Usedocker run to run your Nitro node only

This method will only run the customized Nitro node (i.e., it will not run Blockscout, or the DA server if you're using an AnyTrust chain). Use the following command:

docker run --rm -it -v /path/to/your/node/dir:/home/user/.arbitrum -p 0.0 .0.0:8449:8449 custom-nitro-node-dev --conf.file /home/user/.arbitrum/nodeConfig.json

4.2 Preserving the replay binary

The primary issue with simply using a nitro-node-dev build is that, whenever the code changes at all, the replay binary will also change.

If the node is missing the replay binary corresponding to the on-chain WASM module root, it will be unable to act as a validator. Therefore, when releasing new node Docker images it's important to include the currently on-chain WASM module root.

To do that, you'll need to first extract the replay binary from thenitro-node-dev Docker image built earlier:

docker run --rm --name replay-binary-extractor --entrypoint sleep custom-nitro-node-dev infinity docker

cp replay-binary-extractor:/home/user/target/machines/latest extracted-replay-binary docker stop replay-binary-extractor cat extracted-replay-binary/module.root mv extracted-replay-binary "target/machines/ (cat extracted-replay-binary/module.root) " These commands will output the new WASM module root, and create the directorytarget/machines/ . There you'll find the three versions of the replay binary mentioned earlier:replay.wasm ,machine.wavm.br , andmodule-root.txt , along with some other optional files. Now that you've extracted the replay binary, there are two ways to add it to future Docker images, including non-dev image builds. You can either keep it locally and copy it in, or host it on the web.

Option 1: Store the extracted replay binary locally

Now that we've extracted the replay binary, we can modify the Dockerfile to copy it into new Docker builds. Edit theDockerfile file in the root of the nitro folder, and after all theRUN ./download-machines.sh ... lines, add:

COPY target/machines/ RUN ln -sfT latest Replace each with the WASM module root you got earlier.

Option 2: Host the replay binary on the web

To support building the Docker image on other computers without this local machine directory, you'll need to either commit the machine to git, or preferably, host the replay binary on the web.

To host the replay binary on the web, you'll need to host thereplay.wasm andmachine.wavm.br files somewhere. One good option is GitHub releases, but any hosting service works.

Once you have those two files hosted, instead of theCOPY andRUN command mentioned in option 1, you'll need to add these new lines to theDockerfile file in the root of the nitro folder, after all theRUN ./download-machines.sh ... lines:

RUN wasm_module_root="" && \ mkdir "wasm_module_root" && \ wget-O "wasm_module_root/replay.wasm" && \ wget -O "wasm_module_root/machine.wavm.br" && \ echo "wasm_module_root" > "wasm_module_root/module-root.txt" && \ ln -sfT "wasm_module_root" latest Replace the with the WASM module root you got earlier, thewith the direct link to thereplay.wasm file (it must be a direct link to the file and not just a download site), and the with the direct link to themachine.wavm.br file.

Step 5. Verify the fraud proofs

In theory, fraud proofs should now be working with your newly built Docker images. Make some transactions on your new blockchain, test out your modifications to the State Transition Function, wait for a batch to be posted, and you should be seeing "validation succeeded" log lines!

If you see "Error during validation", then the replay binary is likely not up-to-date with your modifications to the State Transition Function. Ensure that the replay binary is freshly built and is not missing any modifications, and that the WASM module root set in the rollup contract matches your replay binary. Edit this page Last updatedonMar 7, 2024 Previous How to customize your Orbit chain's deployment configuration Next How to customize your Orbit chain's precompiles