Portal Network Tools Documentation

Release 0.2.1

The Ethereum Foundation

Oct 05, 2022

CONTENTS

A collection of utilities related to Ethereum's Portal Network

CHAPTER

ONE

CONTENTS

1.1 Run a Bridge Node

1.1.1 Bridge Node Intro

A bridge node for the Portal Network is responsible for pushing new data into the network. It doesn't require any special authority. Anyone can run a bridge node.

It starts by monitoring the Ethereum network and storing newfound data into a Portal client that has joined the network, using a Portal-specific json-rpc API, like `portal_historyOffer`. See the full Portal RPC API.

1.1.2 When to Run a Bridge Node

You likely don't need to. You probably just want to run a Portal Client. If you have high uptime needs, maybe you should run an execution client like geth. A bridge node is just a weird artifact of how the Portal Network works, and most folks can ignore it.

1.1.3 How to Run a Bridge Node

First Installation

Use this bash code to launch the bridge for the very first time:

```
# If you put this code block into a bash script, then the following line will
#   cause the script to exit early if any variable is missing
set -o nounset

# Install a fresh virtualenv, to have a clean installation environment
python3 -m venv eth-portal-venv

# Enter the new virtualenv environment
. eth-portal-venv/bin/activate

# Make sure to use the latest pip & setuptools
pip install -U pip setuptools

# Install eth-portal
pip install eth-portal
```

(continues on next page)

(continued from previous page)

```
# Link to your already-built trin binary
ln -s $PATH_TO_TRIN_BINARY trin

# Verify that the Infura ID is provided
[ "$WEB3_INFURA_PROJECT_ID" ] || echo "Missing Infura project ID!"

# Verify that some trin private keys are provided
[ "$PORTAL_BRIDGE_KEYS" ] || echo "Missing Portal Bridge Keys!"

# Launch the bridge node
python -m eth_portal.bridge --latest
```

Note: Look for the ALL_CAPS variables that you must provide ahead of time with `export`.

Run after first installation

If you exit your terminal and want to restart the bridge node, you can relaunch with:

```
# Navigate to the directory that eth-portal was orignally installed in

# Launch virtalenv
. eth-portal-venv/bin/activate

# Launch the bridge node
python -m eth_portal.bridge --latest
```

You can find more detail about these steps in the sections below.

Detail on eth-portal Install

In a fresh virtualenv, run `pip install eth-portal` to install.

If you want to instead use the latest, greatest (and potentially buggiest), check out eth-portal via git and install dependencies with `pip install -e .[dev]`.

Detail on Linking to trin

The bridge launches a bunch of portal clients, in order to join into the network with peers that have a variety of different node IDs. In theory, the bridge can work with any Portal client, but for now, it is hard-coded to `trin`.

Get a copy of the `trin` binary. A straightforward approach is to check out trin from source and run `cargo build`. Then, place the binary of `trin` in the root of the eth-portal source directory.

For example, if trin is checked out in a sibling directory to eth-portal, you could run this from the parent directory of both:

```
ln -s "$PWD/trin/target/debug/trin" eth-portal/trin
```

Detail on Linking to Infura

The bridge currently uses Infura to track when new headers arrive. Eventually, this will switch to using an execution client like geth.

For now, create an Infura project and add the ID as an environment variable:

```
export WEB3_INFURA_PROJECT_ID=1234567890abcdef
```

Detail on Portal Bridge Keys

The bridge requires private keys to launch the Portal clients. There is some subtletly in choosing these keys ideally, but it's probably fine to just pick a bunch at random. How many is the ideal number to run? It is still an open question.

After selecting your private keys, concatenate them using commas and add it to your environment:

```
export PORTAL_BRIDGE_
↪KEYS=7261696e626f77737261696e626f77737261696e626f77737261696e626f7773,
↪756e69636f726e73756e69636f726e73756e69636f726e73756e69636f726e73
```

Detail on Launching the Bridge

If you have any trouble launching the bridge:

```
python -m eth_portal.bridge --latest
```

Then first make sure that you have activated your virtualenv, and are in the originally installed directory. There should be a `trin` binary linked there.

It's currently assumed that the `/tmp` is available, and the ports 9000, 9001, etc. are available. For each trin key you provide, the bridge will launch another instance of trin, which will use another port.

Running the bridge will use about 650k requests a day, at current mainnet levels. That requires a paid Infura account to run full-time.

How to See the trin Logs

One way to see the logs being emitted from trin is to run trin manually and set RUST_LOG to display the desired logging level. The bridge will notice that trin is already running, and use that instance.

In order to determine the correct trin command, you can inspect the shell output at the beginning of launching the bridge. Then shut down the bridge, use the printed command to launch trin, and re-launch the bridge.

1.1.4 Backfill historical blocks

The standard bridge node pushes all new network data in. Sometimes we want to push in a particular block range. To do so, read on.

If you have never run the bridge before, see *First Installation*.

In order to import blocks numbered 100 through 200 (ie~ including 100 and 200), run this command:

```
python -m eth_portal.bridge --block-range 100 200
```

To import a single block, just repeat the same block number twice.

This command will publish the specified blocks, and then shut down. The bridge will not try to insert any content besides what you specify here.

1.1.5 Inject Content Manually

The standard bridge node determines the latest data to push in by following the chain. Sometimes we want to locally generate the data and publish it. To do so, read on.

If you have never run the bridge before, see *First Installation* (although you can skip the Infura setup).

Next, generate Portal-valid content keys and values. Load them into files, formatted according to these rules:

- Each item of content is represented in its own file
- Files have the `.portalcontent` extension
- Files are named with the hex-encoded content key before the `.`
- File contents are the binary-encoded value to insert

Supply the paths to these content files using the bridge node CLI, like:

```
python -m eth_portal.bridge --content-files mycontentfiles/*.portalcontent
```

This is simply a regular path glob argument. So if you want to load every file in a folder, then this works too:

```
python -m eth_portal.bridge --content-files mycontentfiles/*
```

By passing in an argument to the bridge command, you indicate that you want to inject the specified content, and then shut down. The bridge will not try to insert any content besides what you specify here.

1.2 eth_portal package

1.2.1 Subpackages

eth_portal.bridge package

Submodules

eth_portal.bridge.handle module

eth_portal.bridge.history module

eth_portal.bridge.insert module

eth_portal.bridge.run module

Module contents

1.2.2 Submodules

1.2.3 eth_portal.portal_encode module

1.2.4 eth_portal.ssz_sedes module

1.2.5 eth_portal.trin module

1.2.6 eth_portal.web3_decode module

1.2.7 Module contents

1.3 Release Notes

1.3.1 Portal Network Tools v0.2.1 (2022-10-04)

Features

- Updated content key encoding ssz scheme & test vectors to new spec, removing the chain id. (#44)

Bugfixes

- Correct the maximum receipt length in SSZ encoding. No specific bug is known to be fixed by this change. (#40)

Improved Documentation

- Update `ln -s` example in bridge setup guide to use absolute source path. This prevents folks from getting a `Too many levels of symbolic links` error during setup, because soft links require an absolute source path. (#42)

1.3.2 Portal Network Tools v0.2.0 (2022-09-22)

Breaking changes

- In order to follow the tip of the chain, you must now specify `--latest` or `-l`. See "Run after first installation" under "Run a Bridge Node" (#38)

Features

- Publish data from a historical block range with `--block-range`. See "Backfill historical blocks" under "Run a Bridge Node" (#34)
- Add the ability to inject specific content items on-demand with `--content-files`. See "Inject content manually" under "Run a Bridge Node" (#35)

Internal Changes - for Portal Network Tools Contributors

- When receipts fail their check against the expected root hash, save them to a file in a temporary directory (#32)

Miscellaneous changes

- Merge in latest project template #41

1.3.3 Portal Network Tools v0.1.0 (2022-08-31)

Features

- Officially claim support for python versions up to 3.9 (won't support 3.10 until a stable web3 version does...) (#29)

Improved Documentation

- Add a compiled first-time install script to docs (#25)

Internal Changes - for Portal Network Tools Contributors

- – Upgrade ssz to v0.3.0 to get cleaner & faster ssz bytelist decoding
  – Apply `black` formatting to setup.py (#29)
- Update project template to get smaller releases, a towncrier duplicate-note fix, CircleCI image upgrade, breaking change newsfragments, and pydocstyle failure explanations. (#30)

Miscellaneous changes

- #28

Breaking changes

- Use the portal_historyOffer rpc endpoint of trin, instead of portal_historyStore. This requires a more recent version of trin. Follow the rpc update in trin for more. (#27)
- Drop Python3.6 to support the latest ssz v0.3.0, which doesn't support py3.6. (#29)

1.3.4 Portal Network Tools v0.1.0-beta.0 (2022-06-02)

Features

- Launch the bridge with `python -m eth_portal.bridge`. (#23)
- Add a new script to monitor new headers, with Infura. Also, add utility to encode header fields into serialized RLP object. (#1)
- Encode header hash into a Portal History Network content key (#2)
- Encode header content key & value, on each new header hash. (#3)
- Launch trin nodes with bridge, and push content to them, as new headers arrive. (#4)
- Encode the content key for block receipts (#7)
- Encode receipt values, tested against header's receipt root hash (#8)
- Encode a group of receipts to its Portal Network content value. Currently assumes that we switch to using an SSZ list from the RLP list. (#10)
- Push the block's receipts out to trin nodes, after detecting a new header. (#11)
- In bridge launcher, detect if trin "injector" node is already running, then use it. One benefit is being able to observe the logs on trin during bridge operation. (#14)
- Encode the content key for block bodies (#16)
- Decode a web3 transaction into a py-evm one, for encoding & hashing (#17)
- Encode uncles and transactions into a block body for Portal Network. (#18)
- Propagate block bodies on the Portal History Network. (#19)

Improved Documentation

- Add *bridge* node guide (#4)

Internal Changes - for Portal Network Tools Contributors

- Add web3 dependency in order to poll for new headers, and py-evm dependency to encode headers to RLP (#1)
- Use black as the code formatter, and add to CI for enforcement. Upgrade isort to v5 for compatibility. (#12)

1.3.5 v0.1.0-alpha.1

- Added a script to generate private keys and Node IDs
- Launched repository, claimed names for pip, RTD, github, etc

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex