1. Proton Chain Installation Guide

Determine the installation directory e.g. */home/sh/proton_mainnet/*

```
# Clone repository
$ cd /home/sh/proton_mainnet/
$ git clone https://github.com/EOSIO/eos.git --recursive
$ cd eos

# Checkout latest version in https://github.com/EOSIO/eos/releases
$ git checkout v2.0.7
$ git submodule update --recursive

# Build nodeos software
$ scripts/eosio_build.sh -s XPR -P

# Built nodeos, cleos and keosd software locations:
$ /home/sh/proton_mainnet/eos/build/programs/nodeos/
$ /home/sh/proton_mainnet/eos/build/programs/cleos/
$ /home/sh/proton_mainnet/eos/build/programs/keosd/
```

Determine the directory to run nodeos e.g. */home/sh/proton_mainnet/run/*

```
# Linking built nodeos software and others such as cleos and keosd in run directory
# cleos software is a command line tool to interact with the blockchain through API
exposed by nodeos software
# keosd software is a key manager service daemon for storing private keys and signing
transactions in proton chain
$ cd /home/sh/proton_mainnet/run/
$ ln -s /home/sh/proton_mainnet/wax-blockchain/build/programs/nodeos/nodeos .
$ ln -s /home/sh/proton_mainnet/wax-blockchain/build/programs/cleos/cleos .
$ ln -s /home/sh/proton_mainnet/wax-blockchain/build/programs/keosd/keosd .
```

Before running nodeos software, you can download *genesis.json* and sample *config.ini* here:

1. https://storage.googleapis.com/eosio-public-backup/proton-mainnet/genesis.json
2. https://storage.googleapis.com/eosio-public-backup/proton-mainnet/sample_nodeos-config.ini

```
# Run nodeos software
# Acquire genesis.json if syncing to the network from scratch
# Determine the directory to store proton chain block log and state memory. For
better performance, place the data directory in a separate partition. In this example
it is assumed that the partition is mounted in /home/sh/proton_mainnet/run/data/
# Execute nodeos software with the following arguments if syncing from scratch
$ ./nodeos --genesis-json /home/sh/proton_mainnet/run/genesis.json --config-dir
/home/sh/proton_mainnet/run/  --data-dir /home/sh/proton_mainnet/run/data/
```

Proton chain operators can provide block log and state memory archives to start syncing to the network quicker. Here are the downloadable blocks log and state memory archives:

3. https://storage.googleapis.com/eosio-public-backup/proton-mainnet/blocks.tar.gz
4. https://storage.googleapis.com/eosio-public-backup/proton-mainnet/state.tar.gz

```
# Extract archives (blocks.tar.gz and state.tar.gz) in data directory (e.g.
/home/sh/proton_mainnet/run/data/)
# Execute nodeos software with the following arguments
$ ./nodeos --config-dir /home/sh/proton_mainnet/run/  --data-dir
/home/sh/proton_mainnet/run/data/
```

2. Hyperion History API Installation Guide

Hyperion History API is an EOSIO blockchain's full history indexing tool used for storing and retrieving Proton Chain`s historical data. EOS Rio develops the Hyperion History API tool and provides the product documentation and as well as an installation guide (https://hyperion.docs.eosrio.io/). Hyperion History API requires the following components:

1. EOSIO State History
2. Elasticsearch
3. RabbitMQ
4. Redis
5. NodeJs v14 (*Recommended for Ubuntu 18.04*)
6. PM2
7. Hyperion Indexer
8. Hyperion API

2.1. Running EOSIO State History

Before Installing Hyperion History API, you need to run nodeos software with state-history plugin. The installation guide for nodeos with state-history plugin is similar to the previous installation guide with additional changes in the *config.ini* and nodeos run arguments

```
# update config.ini with the following changes:
        ### state-history-dir = "state-history"
        ### trace-history = true
        ### chain-state-history = true
        ### state-history-endpoint = 0.0.0.0:8080
        ### trace-dir = "traces"
        ### plugin = eosio::state_history_plugin
# Execute nodeos software with the following arguments to populate state-history
folder in data directory (e.g. /home/sh/proton_mainnet/run/data/) with historical
action traces
$ ./nodeos --config-dir /home/sh/proton_mainnet/run/  --data-dir
/home/sh/proton_mainnet/run/data/ –disable-replay-opts
```

2.2. Elasticsearch Installation

Below is the summary of Elasticsearch installation from the APT Repository for Debian package and detailed installation instructions can be found in the official documentation
(https://www.elastic.co/guide/en/elasticsearch/reference/current/deb.html)

```
# Download and install Elasticsearch's public signing key
$ wget -qO - https://artifacts.elastic.co/GPG-KEY-elasticsearch | sudo apt-key add –

# Elasticsearch installation from the APT Repository
$ sudo apt-get install apt-transport-https
$ echo "deb https://artifacts.elastic.co/packages/7.x/apt stable main" | sudo tee
/etc/apt/sources.list.d/elastic-7.x.list
$ sudo apt-get update
$ sudo apt-get install elasticsearch
```

```
# Running Elasticsearch with system
# Configure Elasticsearch to start automatically when the system boots up
$ sudo /bin/systemctl daemon-reload
$ sudo /bin/systemctl enable elasticsearch.service
$ sudo systemctl stop elasticsearch.service
$ sudo systemctl start elasticsearch.service
```

Edit the following Elasticsearch files:

1. /etc/elasticsearch/elasticsearch.yml

   ```
   Cluster.name: myCluster
   Bootstrap.memory_lock: true
   ```

2. /etc/elasticsearch/jvm.options

   ```
   Xmsg16g
   Xmsx16g
   ```

3. run sudo systemctl edit elasticsearch and add the following lines:

   ```
   [Service]
   LimitMEMLOCK=infinity
   ```

```
# Restart elastic search after applying changes
$ sudo systemctl stop elasticsearch.service
$ sudo systemctl start elasticsearch.service

# Test Elasticsearch REST API
$ curl http://localhost:9200

# The Elasticsearch REST API should display response similar to this:
{
  "name" : "proton-hyperion-node",
  "cluster_name" : "myCluster",
  "cluster_uuid" : "jIvdZt82QEGZPZxxxxxxx",
  "version" : {
    "number" : "7.9.1",
    "build_flavor" : "default",
    "build_type" : "deb",
    "build_hash" : "083627f112ba94dffc1232e8b42b73492789ef91",
    "build_date" : "2020-09-01T21:22:21.964974Z",
    "build_snapshot" : false,
    "lucene_version" : "8.6.2",
    "minimum_wire_compatibility_version" : "6.8.0",
    "minimum_index_compatibility_version" : "6.0.0-beta1"
  },
  "tagline" : "You Know, for Search"
}
```

2.3. RabbitMQ Installation

Below is the summary of RabbitMQ installation from the APT Repository on Bintray for Ubuntu and detailed installation instructions can be found in the official documentation (https://www.rabbitmq.com/install-debian.html#installation-methods).

```
# RabbitMQ requires Erlang/OTP packages. In this documentation, we will install
Debian packages of Erlang provided by RabbitMQ team
# Download and install RabbitMQ's public signing key
$ curl -fsSL https://github.com/rabbitmq/signing-keys/releases/download/2.0/rabbitmq-
release-signing-key.asc | sudo apt-key add –

# The following is Erlang installation steps:
$ sudo apt-get install apt-transport-https
## Adding a source list file for 3rd party APT repository
$ sudo vi /etc/apt/sources.list.d/bintray.erlang.list
## Enter the following entry into the above file:
        ### deb https://dl.bintray.com/rabbitmq-erlang/debian bionic erlang-22.x
$ sudo apt-get update
$ sudo apt-get install -y erlang-base \
                  erlang-asn1 erlang-crypto erlang-eldap erlang-ftp erlang-inets \
                  erlang-mnesia erlang-os-mon erlang-parsetools erlang-public-key \
                  erlang-runtime-tools erlang-snmp erlang-ssl \
                  erlang-syntax-tools erlang-tftp erlang-tools erlang-xmerl

# The following is RabbitMQ installation steps:
$ sudo apt-get install curl gnupg -y
$ curl -fsSL https://github.com/rabbitmq/signing-keys/releases/download/2.0/rabbitmq-
release-signing-key.asc | sudo apt-key add –
$ sudo apt-get install apt-transport-https
$ sudo tee /etc/apt/sources.list.d/bintray.rabbitmq.list <<EOF
deb https://dl.bintray.com/rabbitmq-erlang/debian bionic erlang
deb https://dl.bintray.com/rabbitmq/debian bionic main
EOF
$ sudo apt-get update -y
$ sudo apt-get install rabbitmq-server -y --fix-missing

# Enable RabbitMQ webUI
$ sudo rabbitmq-plugins enable rabbitmq_management

# Add RabbitMQ vHost
$ sudo rabbitmq-plugins enable rabbitmq_management

# Create user and password for RabbitMQ
$ sudo rabbitmqctl add_user {my_user} {my_password}

# Set RabbitMQ user as administrator
$ sudo rabbitmqctl set_user_tags {my_user} administrator

# Set RabbitMQ user's permission to the vHost
$ sudo rabbitmqctl set_permissions -p /hyperion {my_user} ".*" ".*" ".*"

# Check access to the RabbitMQ webUI
$ curl http://localhost:15672
```

```
# Start rabbitmq-server service
$ sudo /bin/systemctl daemon-reload
$ sudo /bin/systemctl enable rabbitmq-server.service
$ sudo systemctl stop rabbitmq-server.service
$ sudo systemctl start rabbitmq-server.service

# Check access to the RabbitMQ webUI
$ curl http://localhost:15672
```

2.4. Redis Installation

Redis is used to cache data in memory. Below is the summary of Reddis installation.

```
# Install Redis
# sudo apt install redis-server

# Edit /etc/redis/redis.conf to change supervised argument to supervised systemd
$ sudo vi /etc/redis/redis.conf
### supervised system

# Start Redis service
$ sudo /bin/systemctl daemon-reload
$ sudo /bin/systemctl enable redis.service
$ sudo systemctl stop redis.service
$ sudo systemctl start redis.service
```

2.5. NodeJS Installation

```
# Add NodeJS source
$ curl -sL https://deb.nodesource.com/setup_13.x | sudo -E bash –

# Install nodeJS
$ sudo apt-get install -y nodejs
```

2.6. PM2 Installation

```
# Install PM2
$ sudo npm install pm2@latest -g

# Configure PM2 for system startup
$ sudo pm2 startup
```

2.7. Hyperion History API Setup

Below is the summary of Hyperion History API installation and its configuration file setup.

```
# Clone EOS Rio's Hyperion History API repository and install its packages
$ cd /home/sh/proton_mainnet/
$ git clone https://github.com/eosrio/hyperion-history-api.git
$ cd hyperion-history-api
$ sudo npm install
```

Edit *ecosystem.config.js* file located in the *hyperion-history-api* directory

```
$ cp example-ecosystem.config.js ecosystem.config.js
$ vi ecosystem.config.js
# update ecosystem.config.js with the following changes:
        ### module.exports = {
        ###    apps: [
        ###        addIndexer('proton'),
        ###        addApiServer('proton', 1)
        ###    ]
        ###};
```

Edit *connections.json* file located in the *hyperion-history-api* directory

```
$ cp example-connections.json connections.json
$ sudo vi connections.json
# update connections.json with the following changes:
        ###{
        ###  "amqp": {
        ###    "host": "127.0.0.1:5672",
        ###    "api": "127.0.0.1:15672",
        ###    "user": "RabbitMQ user",
        ###    "pass": "RabbitMQ password",
        ###    "vhost": "hyperion"
        ###  },
        ###  "elasticsearch": {
        ###    "protocol": "http",
        ###    "host": "127.0.0.1:9200",
        ###    "ingest_nodes": ["127.0.0.1:9200"],
        ###    "user": "",
        ###    "pass": ""
        ###  },
        ###  "redis": {
        ###    "host": "127.0.0.1",
        ###    "port": "6379"
        ###  },
        ###  "chains": {
        ###    "proton": {
        ###      "name": "proton",
        ###      "chain_id":
"384da888112027f0321850a169f737c33e53b388aad48b5adace4bab97f437e0",
        ###      "http": "http://192.168.0.1:8130",
        ###      "ship": "ws://192.168.0.1:8080",
        ###      "WS_ROUTER_PORT": 7001,
        ###      "WS_ROUTER_HOST": "127.0.0.1" } } }
```

Edit *proton.config.json* file located in the *hyperion-history-api/chains/* directory

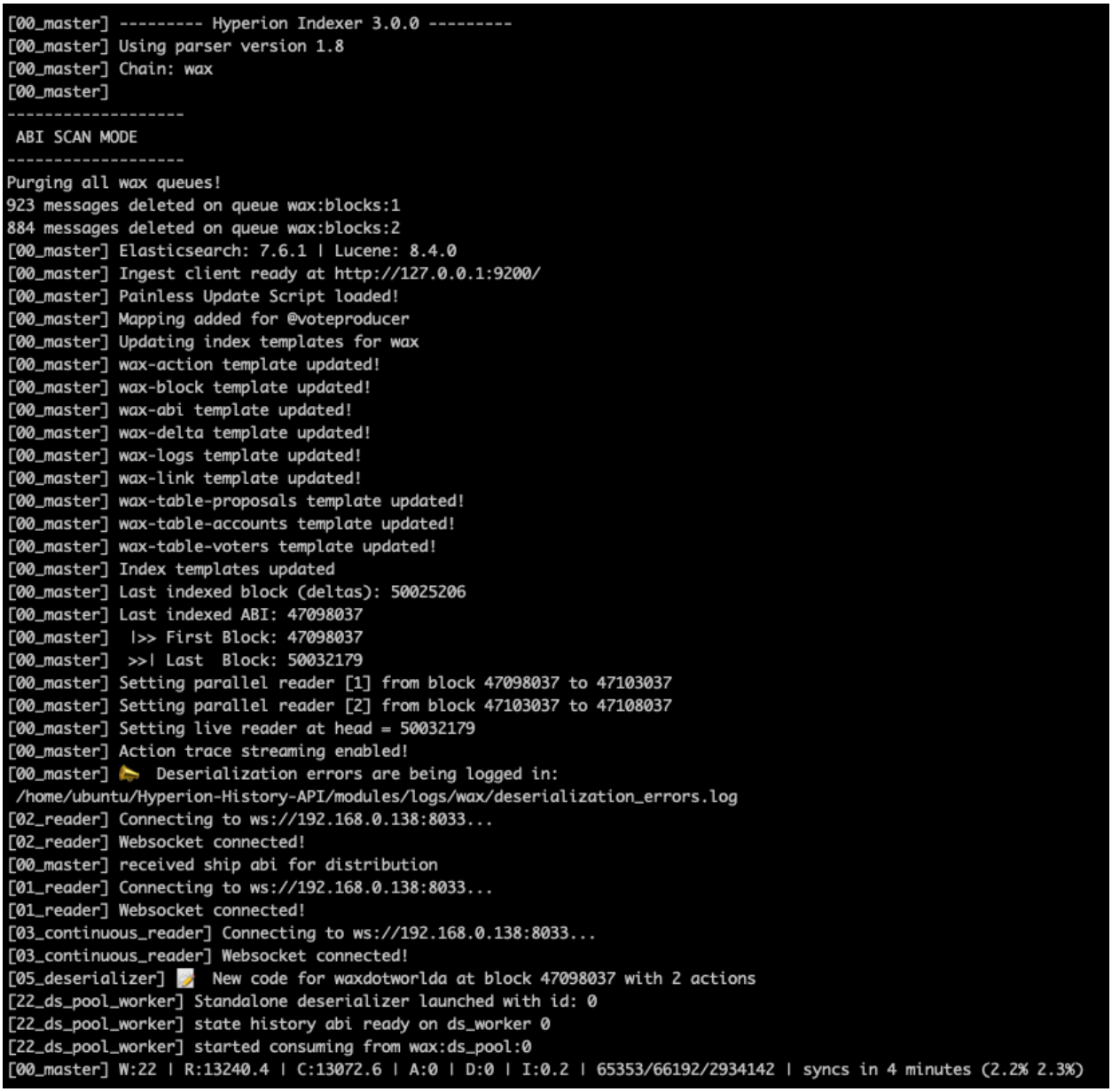
```
$ cp chains/example.config.json chains/proton.config.json
$ vi chains/proton.config.js
# update chains/proton.config.js with the following changes:
        ###{
        ###  "api": {
        ###    "chain_name": "proton",
        ###    "server_addr": "internal server address",
        ###    "server_port": 7000,
        ###    "server_name": "external accessible server address e.g. proton-
hyperion.eoscafeblock.com",
        ###    "provider_name": "your entity name",
        ###    "provider_url": "your website e.g. https://eoscafeblock.com",
        ###    "chain_api": "",
        ###    "push_api": "",
        ###    "chain_logo_url": "",
        ###    "enable_caching": true,
        ###    "cache_life": 1,
        ###    "limits": {
        ###      "get_actions": 1000,
        ###      "get_voters": 100,
        ###      "get_links": 1000,
        ###      "get_deltas": 1000
        ###    },
        ###    "access_log": false,
        ###    "enable_explorer": false,
        ###    "chain_api_error_log": false,
        ###    "custom_core_token": ""
        ###  },
        ###  "settings": {
        ###    "preview": false,
        ###    "chain": "proton",
        ###    "eosio_alias": "eosio",
        ###    "parser": "1.8",
        ###    "auto_stop": 300,
        ###    "index_version": "v1",
        ###    "debug": false,
        ###    "bp_logs": false,
        ###    "bp_monitoring": false,
        ###    "ipc_debug_rate": 60000,
        ###    "allow_custom_abi": false,
        ###    "rate_monitoring": true,
        ###    "max_ws_payload_kb": 256,
        ###    "ds_profiling": false,
        ###    "auto_mode_switch": false
        ###  },
        ###  "blacklists": {
        ###    "actions": [],
        ###    "deltas": []
        ###  },
        ###  "whitelists": {
        ###    "actions": [],
        ###    "deltas": [],
        ###    "max_depth": 10,
```

```
        ###    "root_only": false
        ###  },
        ###  "scaling": {
        ###    "readers":            2,
        ###    "ds_queues":          2,
        ###    "ds_threads":         1,
        ###    "ds_pool_size":       1,
        ###    "indexing_queues":    1,
        ###    "ad_idx_queues":      1,
        ###    "max_autoscale":      4,
        ###    "batch_size":         10000,
        ###    "resume_trigger":     5000,
        ###    "auto_scale_trigger": 20000,
        ###    "block_queue_limit":  10000,
        ###    "max_queue_limit":    50000,
        ###    "routing_mode":       "heatmap",
        ###    "polling_interval":   10000
        ###  },
        ###  "indexer": {
        ###    "start_on": 1,
        ###    "stop_on": 0,
        ###    "rewrite": false,
        ###    "purge_queues": true,
        ###    "live_reader": false, #NOTE: this is set to false before running
indexer, once it's done, set this to true then restart Hyperion History API
        ###    "live_only_mode": false,
        ###    "abi_scan_mode": true, #NOTE: this is set to true before running
indexer, once it's done, set this to false then restart Hyperion History API
        ###    "fetch_block": true,
        ###    "fetch_traces": true,
        ###    "disable_reading": false,
        ###    "disable_indexing": false,
        ###    "process_deltas": true
        ###  },
        ###  "features": {
        ###    "streaming": {
        ###      "enable": false, #NOTE: this is set to false before running
indexer, once it's done, set this to true then restart Hyperion History API
        ###      "traces": false, #NOTE: this is set to false before running
indexer, once it's done, set this to true then restart Hyperion History API
        ###      "deltas": false
        ###    },
        ###    "tables": {
        ###      "proposals": true,
        ###      "accounts": true,
        ###      "voters": true
        ###    },
        ###    "index_deltas": true,
        ###    "index_transfer_memo": true,
        ###    "index_all_deltas": true,
        ###    "deferred_trx": false,
        ###    "failed_trx": false,
        ###    "resource_limits": false,
        ###    "resource_usage": false
        ###  },
```

```
###  "prefetch": {
###    "read": 50,
###    "block": 100,
###    "index": 500
###  }
###}
```

After completing Hyperion History API setup, it is time to run the indexer using the *run.sh* executable file located in the *hyperion-history-api* directory

```
# Start indexing proton mainnet
$ ./run.sh proton-indexer
# On the first run of Hyperion Indexer abi_scan_mode is set to true (See:
proton.config.json), and you should see something like this on the terminal:
```



After Hyperion Indexer has completed, edit *proton.config.json* once again to start running in live mode and to enable streaming.

Start Proton History API once Hyperion Indexer has completed and restarted for the second run using the updated *proton.config.json*.

```
# Start proton history API
$ ./run.sh proton-api
```

You should be able to check the health of your Hyperion History API as follow:

```
$ curl http://192.168.0.44:7000/v2/health | jq
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   941  100   941    0     0  15949      0 --:--:-- --:--:-- --:--:-- 15949
{
  "version": "3.1.2",
  "version_hash": "d122d79022136e4ea9be15a9fe691daaa9a6a7db",
  "host": "proton-hyperion.eoscafeblock.com",
  "features": {
    "streaming": {
      "enable": true,
      "traces": true,
      "deltas": false
    },
    "tables": {
      "proposals": true,
      "accounts": true,
      "voters": true
    },
    "index_deltas": true,
    "index_transfer_memo": true,
    "index_all_deltas": true,
    "deferred_trx": false,
    "failed_trx": false,
    "resource_limits": false,
    "resource_usage": false
  },
  "health": [
    {
      "service": "RabbitMq",
      "status": "OK",
      "time": 1603667552385
    },
    {
      "service": "NodeosRPC",
      "status": "OK",
      "service_data": {
        "head_block_num": 32172214,
        "head_block_time": "2020-10-25T23:12:32.000",
        "time_offset": 437,
        "last_irreversible_block": 32171886,
        "chain_id":
"384da888112027f0321850a169f737c33e53b388aad48b5adace4bab97f437e0"
      },
```

```
      "time": 1603667552437
    },
    {
      "service": "Elasticsearch",
      "status": "OK",
      "service_data": {
        "last_indexed_block": 32172213,
        "total_indexed_blocks": 32172213,
        "active_shards": "100.0%"
      },
      "time": 1603667552440
    }
  ],
  "query_time_ms": 58.446
}
```


Or, if you setup a proxy with ssl certificate in front of Hyperion History API, you should be able to access it from the internet. Here is some example to check our setup:

1. https://proton-hyperion.eoscafeblock.com/v2/health
2. https://proton-hyperion.eoscafeblock.com/v2/history/get_actions
3. https://proton-hyperion.eoscafeblock.com/v2/docs/index.html

3. Listening to Actions

Below is an example for listening to actions in the Proton Chain:

```
const HyperionSocketClient = require('@eosrio/hyperion-stream-client').default;
const client = new HyperionSocketClient('http://localhost:7000', {async: true});
client.onConnect = () => {
  // Incoming XPR transfers to  bithumb
  client.streamActions({
      contract: 'eosio.token',
      action: 'transfer',
      account: 'bithumb',
      start_from: '2020-03-15T00:00:00.000Z',
      read_until: 0, // Live
      filters: [
        {field: '@transfer.to', value: 'bithumb'},
        {field: '@transfer.symbol', value: 'XPR'}
      ],
  });
}
// see 3 for handling data
client.onData = async (data, ack) => {
    console.log(data); // process incoming data, replace with your code
    ack(); // ACK when done
}
client.connect(() => {
  console.log('connected!');
});
```