Mina Foundation

QANet: Your Gateway to the Mina Protocol Berkeley Upgrade

Version 1.3
Last Revised: March 3rd, 2024

Table of Contents

Partners Berkeley Testing

The 'Partners Berkeley Testing' phase is a crucial milestone on our journey towards the major Mina Protocol Upgrade. This phase offers our partners a robust and stable environment to test their integrations and systems with the Berkeley release candidate. It's an opportunity to ensure seamless compatibility and performance with the upcoming features.

QANet, your stable Berkeley playground

QANet is a dedicated network running the latest Berkeley version. QANet the network for our partners to connect, deploy, and thoroughly test their integrations with the Mina Protocol Blockchain.

Understanding QANet's Scope

It's important to note that QANet is not intended to test the Mainnet upgrade mechanism. Instead, it's a stable network equipped with all the features that will be available post-Berkeley upgrade, tailored for comprehensive testing of your systems. If you are interested in participating in the testing of the Berkeley upgrade mechanism, please contact the designated supporting team for further information.

QANet start date and duration

QANet goes live on February 6th at 12 pm UTC and will be operational for a duration of one month. Please note that after this period, QANet will be decommissioned, resulting in the loss of all network state. This timeframe is your window to explore, test, and prepare for the next era of the Mina Protocol.

Objectives of the Partners Berkeley Testing

This phase is designed to ensure the readiness and robustness of integrations in the QANet environment.

- **Application Validation**: Participants should validate that their applications function correctly within the QANet. The validation includes verifying all standard operations, features, and integrations as they would exist in the post-upgrade environment.
- **System Compatibility**: Ensure all systems and tools are fully compatible with the new features and structural changes introduced in the Berkeley upgrade. Please refer to Appendix 1 for an extensive list of all the changes to the Archive Node and Rosetta API applications.
- **Issue Identification and Reporting**: Actively identify any issues, anomalies, or bugs within the QANet environment and report them in GitHub using the specially created label ***qanet-partners-testing***.
- **Feedback and Insights**: Please confirm your intention to validate your systems on QANet, including a specific timeline for your testing activities. Once testing is complete, share your results, observations, and any challenges or areas for improvement directly with your designated Mina Foundation contact through your chosen communication channel.

QANet test support tools

- Node GraphQL Endpoint
  - https://qanet.minaprotocol.network/graphql
- MinaScan QANet custom blockchain explorer
  - https://qanet.minaprotocol.network/
- Rosetta API online and offline
  - Online - https://rosetta-online-qanet.minaprotocol.network
  - Offline - https://rosetta-offline-qanet.minaprotocol.network

Getting Test Tokens and Setting Up Your QANet Wallet

How to Obtain Test Tokens

QANet will not have a self-service faucet. To receive test tokens, contact your Mina Foundation person of contact directly. Provide them with your QANet account address to facilitate the transfer of test tokens.

Generating Your QANet Account Address

In the Mina Protocol, account generation is network-agnostic. This means you can use the same account credentials across different networks, including QANet.

Create a QANet account using Auro Wallet

1. Install the Auro Wallet browser extension from the official site https://www.aurowallet.com/
2. Follow the wallet's instructions to install and create a new MINA account
3. Add QANet as a Custom Network
    a. Open the hamburger menu (top left) and select 'Network'.
    b. Click on 'Add Network'.
    c. Enter 'QANet' as the node name.
    d. In the Node URL field, enter *https://qanet.minaprotocol.network/graphql*.
    e. Click 'Confirm'.
    f. QANet will now appear in the network dropdown menu, allowing you to send and receive funds on QANet.

Support Channels

If you need personalized support or have specific questions, please contact the designated supporting team for further information through your preferred method of communication.

Raising Issues on GitHub

Please raise any issues encountered on GitHub. A specific label should be used when creating an issue related to QANet. Use the label ***qanet-partners-testing*** to categorize it appropriately for quick reference and response.

Connecting your nodes to QANet

QANet infrastructure provides all the tools and services required to test integration with Mina Protocol Blockchain. Use the following details if you want to spin up your own Nodes and Services connected to QANet.

Recommended Hardware Requirements

Node Type	Memory	CPU	Storage	Network
Mina Daemon Node	32 GB RAM	8 core processor	64 GB	1 Mbps Internet Connection
SNARK Coordinator	32 GB RAM	8 core processor	64 GB	
SNARK Worker	32 GB RAM	16/32 thread dedicated instance	64 GB	
Archive Node	32 GB RAM	8 core processor	64 GB	
Rosetta API standalone Docker image	32 GB RAM	8 core processor	64 GB	

Note: Multiple SNARK Worker processes can run on the same SNARK Work server. 4 core/8 threads should be provisioned per SNARK Worker process, giving a total of 4 workers per SNARK Work server. All your SNARK Worker processes should connect to your SNARK Coordinator.

QANet artifacts

The official release used in QANet is the Mina Daemon Berkeley Release Candidate 1 version 2.0.0berkeley-rc1

Installation Guides

- For complete instructions on setting up and running a Mina daemon node, please visit Connecting to the Network.
- To set up a Mina archive node, follow the guide at Archive Node.
- If you want to integrate Rosetta with the Mina Protocol, use the standalone Rosetta docker image.

Docker Images

- Standalone Mina Daemon Node
  - gcr.io/o1labs-192920/mina-daemon:2.0.0berkeley-rc1-1551e2f-focal-berkeley
- Standalone Archive Node
  - gcr.io/o1labs-192920/mina-archive:2.0.0berkeley-rc1-1551e2f-focal
- Standalone Rosetta API Docker, **detailed instructions can be found further down**
  - gcr.io/o1labs-192920/mina-rosetta:2.0.0berkeley-rc1-1551e2f-focal

Debian Packages

For installing Mina Daemon and Mina Archive Node as a Debian Package, please refer to the official Release Note: https://github.com/MinaProtocol/mina/discussions/15041

QANet connection details

- Chain id
  - e7f5e558ac741bf474c0ce21d372193b6de5bf0028598c90853ba4b4a81233b7

- Git SHA-1
  - 1551e2faaa246c01636908aabe5f7981715a10f4
- Seed Lists URL
  - https://673156464838-mina-seed-lists.s3.us-west-2.amazonaws.com/partners/qanet-seed-nodes.txt
- Genesis Ledger
  - http://673156464838-mina-genesis-ledgers.s3-website-us-west-2.amazonaws.com/qanet/genesis_ledger.json
- QANet pre-computed blocks
  - http://673156464838-mina-precomputed-blocks.s3-website-us-west-2.amazonaws.com/qanet
- Archive Node PostgreSQL Dumps
  - http://673156464838-mina-archive-node-backups.s3-website-us-west-2.amazonaws.com/qanet
- Archive Database Schemas
  - https://raw.githubusercontent.com/MinaProtocol/mina/1551e2faaa246c01636908aabe5f7981715a10f4/src/app/archive/create_schema.sql
  - https://raw.githubusercontent.com/MinaProtocol/mina/1551e2faaa246c01636908aabe5f7981715a10f4/src/app/archive/zkapp_tables.sql

Instructions to launch the standalone **Rosetta API Docker container**

1 - Create a file named *qanet.env* with the following content

```
None
MINA_NETWORK=qanet
PEER_LIST_URL=https://673156464838-mina-seed-lists.s3.us-west-2.amazonaws.com/partners/qanet-seed-nodes.txt
MINA_ARCHIVE_DUMP_URL=http://673156464838-mina-archive-node-backups.s3-website-us-west-2.amazonaws.com/qanet
MINA_GENESIS_LEDGER_URL=http://673156464838-mina-genesis-ledgers.s3-website-us-west-2.amazonaws.com/qanet/genesis_ledger.json
BLOCKS_BUCKET=http://673156464838-mina-precomputed-blocks.s3-website-us-west-2.amazonaws.com/qanet
POSTGRES_DBNAME=archive
```

2 - Run the following command

```
None
docker run --name rosetta --rm \
     --env-file qanet.env \
```

```
    -p 3085:3085 \
    -p 3087:3087 \
    -p 3088:3088 \
    --entrypoint '' \
    gcr.io/o1labs-192920/mina-rosetta:2.0.0berkeley-rc1-1551e2f-focal \
    bash -c "mkdir /genesis_ledgers && ./docker-start.sh"
```

Instructions to run **Mina Nodes with Docker**

1 - Retrieve the QANet Genesis Ledger

This step is critical. Because the QANet is a custom network, the Genesis Ledger needs to be downloaded first.

```
None
# Create config directory
mkdir -p .mina-config/keys

# Retrieve Genesis Ledger
export MINA_CONFIG_FILE=.mina-config/genesis_ledger.json
export
MINA_GENESIS_LEDGER_URL=http://673156464838-mina-genesis-ledgers.s3-website-
us-west-2.amazonaws.com/qanet/genesis_ledger.json
curl -o "$MINA_CONFIG_FILE" "$MINA_GENESIS_LEDGER_URL"
```

2- Generation of libp2p keypair

Each node within the network must possess its own distinct libp2p key pair. This requirement also extends to block producer nodes, despite utilizing a common block production key. Every node operator must generate unique libp2p keys locally on their respective machines using the following command:

```
None
# Generate libp2p keypair
docker run --rm --name generate-libp2p-keypair \
            -e MINA_LIBP2P_PASS='My_V3ry_S3cure_Password' \
            -v $(pwd)/.mina-config/keys:/keys \
            --entrypoint='' \

gcr.io/o1labs-192920/mina-daemon:2.0.0berkeley-rc1-1551e2f-focal-berkeley \
            mina libp2p generate-keypair --privkey-path /keys/libp2p
```

```
# Update directory permissions
chmod 700 $(pwd)/.mina-config/keys
```

See more on generating key pairs.

3- Run Mina Daemon

This command will connect your daemon to the QANet network with the basic flags, please refer to the Flags and Environment Variables section for a more granular configuration

```
None
docker run --rm --name daemon \
            -e MINA_LIBP2P_PASS='My_V3ry_S3cure_Password' \
            -v $(pwd)/.mina-config:/root/.mina-config \
            -p 3085:3085 \
            -p 10801:10801 \
            --entrypoint='' \

gcr.io/o1labs-192920/mina-daemon:2.0.0berkeley-rc1-1551e2f-focal-berkeley \
            mina daemon --libp2p-keypair /root/.mina-config/keys/libp2p \
            --rest-port 3085 \
            --insecure-rest-server \
            --peer-list-url
https://673156464838-mina-seed-lists.s3.us-west-2.amazonaws.com/partners/qan
et-seed-nodes.txt \
            --config-file /root/.mina-config/genesis_ledger.json
```

Flags and Environment Variables

Block Producer

```
None
mina daemon
--block-producer-key <path to the wallet private key file>
--config-directory <path to the mina configuration directory>
--config-file $MINA_CONFIG_FILE
--enable-peer-exchange true
```

```
--file-log-level Info
--file-log-rotations 500
--generate-genesis-proof true
--internal-tracing
--libp2p-keypair <path to the node libp2p private key file>
--log-json
--log-level Debug
--log-snark-work-gossip true
--node-error-url https://nodestats-itn.minaprotocol.tools/submit/stats
--node-status-url https://nodestats-itn.minaprotocol.tools/submit/stats
--peer-list-url
https://673156464838-mina-seed-lists.s3.us-west-2.amazonaws.com/partners/qanet-seed-
nodes.txt

ENVIRONMENT VARIABLES
RAYON_NUM_THREADS=6
MINA_LIBP2P_PASS
MINA_PRIVKEY_PASS
```

SNARK Coordinator

None

```
mina daemon
--config-directory <path to the mina configuration directory>
--config-file $MINA_CONFIG_FILE
--enable-peer-exchange true
--file-log-level Debug
--file-log-rotations 500
--internal-tracing
--libp2p-keypair  <path to the node libp2p private key file>
--log-json
--log-level Debug
--log-snark-work-gossip true
--node-error-url https://nodestats-itn.minaprotocol.tools/submit/stats
--node-status-url https://nodestats-itn.minaprotocol.tools/submit/stats
--peer-list-url
https://673156464838-mina-seed-lists.s3.us-west-2.amazonaws.com/partners/qanet-seed-
nodes.txt
--run-snark-coordinator <wallet public key>
--snark-worker-fee 0.001
--work-selection seq

ENVIRONMENT VARIABLES
RAYON_NUM_THREADS=6
MINA_LIBP2P_PASS
```

SNARK Worker

```
None
mina internal snark-worker
--proof-level full
--shutdown-on-disconnect false
--daemon-address <snark coordinator IP:port>

ENVIRONMENT VARIABLES
RAYON_NUM_THREADS:8
```

Appendix 1 - Key differences between Mainnet and Berkeley

Archive Node

If you are using the Archive Node database directly for your system integrations, then you should understand all the changes that might impact your applications. The most important change is that the **balances** table in the mainnet schema will no longer exist. In the new schema, it is replaced with the table **accounts_accessed** - from an application semantics point of view, the data in **accounts_accessed** is still the same.

In the Berkeley protocol, accounts can now have the same public key but a different token_id. This means accounts are identified by both their public key and token_id, not just the public key. Consequently, the foreign key for the account in all tables is account_identifier_id instead of public_key_id.

Schema differences

- **Removed Types**
  - The options create_token, create_account, and mint_tokens have been removed from the user_command_type enumeration.
- **Indexes Dropped**
  - We've removed several indexes from tables, this may affect how you search and organize data:
    - Idx_public_keys_id
    - Idx_public_keys_value
    - Idx_snarked_ledger_hashes_value
    - idx_blocks_id, idx_blocks_state_hash
- **Table Removed**
  - The balances table is no longer available.
- **New Tables Added**
  - We've introduced the following new tables:
    - tokens, token_symbols, account_identifiers, voting_for, protocol_versions, accounts_accessed, accounts_created, zkapp_commands, blocks_zkapp_commands, zkapp_field,

zkapp_field_array, zkapp_states_nullable, zkapp_states, zkapp_action_states, zkapp_events, zkapp_verification_key_hashes, zkapp_verification_keys, zkapp_permissions, zkapp_timing_info, zkapp_uris, zkapp_updates, zkapp_balance_bounds, zkapp_nonce_bounds, zkapp_account_precondition, zkapp_accounts, zkapp_token_id_bounds, zkapp_length_bounds, zkapp_amount_bounds, zkapp_global_slot_bounds, zkapp_epoch_ledger, zkapp_epoch_data, zkapp_network_precondition, zkapp_fee_payer_body, zkapp_account_update_body, zkapp_account_update, zkapp_account_update_failures

- **Updated Tables**
  - The following tables have been updated
    - timing_info, user_commands, internal_commands, epoch_data, blocks, blocks_user_commands, blocks_internal_commands

Differences per table

- **timing_info**
  - Removed columns: token and initial_balance.
- **user_commands**
  - Removed columns: fee_token, token
- **internal_commands**
  - Removed columns: token
  - Renamed column command_type to type.
- **epoch_data**
  - Added columns: total_currency, start_checkpoint, lock_checkpoint, epoch_length
- **blocks**
  - Added columns: last_vrf_output, min_window_density, sub_window_densities, total_currency, global_slot_since_hard_fork, global_slot_since_genesis, protocol_version_id, proposed_protocol_version_id
  - Removed column: global_slot
- **blocks_user_commands**
  - Removed columns: fee_payer_account_creation_fee_paid, receiver_account_creation_fee_paid, created_token, fee_payer_balance, source_balance, receiver_balance
  - Added index: idx_blocks_user_commands_sequence_no
- **blocks_internal_commands**
  - Removed columns: receiver_account_creation_fee_paid, receiver_balance
  - Added indexes: idx_blocks_internal_commands_sequence_no, idx_blocks_internal_commands_secondary_sequence_no

Rosetta API

Berkeley upgrade introduce two new operation types: zkapp_fee_payer_dec and zkapp_balance_change.

Appendix 2 - migration from @o1labs/client-sdk to mina-signer

Below you will find an example of how to use mina-signer library. Please keep in mind the following:

1. The network should be “testnet” for Devnet network.
2. Make sure to adjust the nonce to the correct nonce on the account you want to use
3. Use fee 1 MINA to get the TX submitted quickly

JavaScript

```
import { Client } from 'mina-signer';

// create testnet client and define hard-coded keypair

const client = new Client({ network: 'testnet' });

let privateKey = Your private key;
let publicKey = the public key - perhaps derived from the private key
using -> client.derivePublicKey(privateKey);

// define and sign payment

let payment = {
  from: publicKey,
  to: 'to public key',
  amount: 100,
  nonce: 1,
  fee: 1000000,
};

const signedPayment = client.signPayment(payment, privateKey);

// send payment to graphql endpoint

let url = 'https://qanet.minaprotocol.network/graphql';

let query = `mutation {
  sendPayment(
    input: ${objectToGraphqlQuery(signedPayment.data)},
    signature: ${objectToGraphqlQuery(signedPayment.signature)}
```

```
    ) {
      payment { id }
    }
  }`;

console.log('========================');
console.log(query);
console.log('========================');

let response = await fetch(url, {
  method: 'POST',
  headers: { 'Content-Type': 'application/json' },
   body: JSON.stringify({ operationName: null, query, variables: {}
}),
});
if (response.status == 200) {
  let json = await response.json();
  console.dir(json, { depth: null });
} else {
  let text = await response.text();
  console.log('Error:\n', text);
}

function objectToGraphqlQuery(obj: any) {
  let json = JSON.stringify(obj, null, 2);
  // removes the quotes on JSON keys
  return json.replace(/\"(\S+)\"\s*:/gm, '$1:');
}
```