# Pantheon Onboarding

## Introduction

In this tutorial, we'll use Pantheon to run a private network of Pantheon nodes in a Docker container.

**Important** Before you begin, make sure you've followed the instructions in the Pantheon [Quickstart Setup Sheet](https://docs.google.com/document/d/1TLdNcIU3oCIPDuoby_yzFdpftHzH90WCmF5JBMqlFjs/edit). After you complete the setup, Pantheon should be ready to run.

Navigate to the pantheon-quickstart directory that you cloned or copied, and list the contents. You'll see the following scripts, which you'll use in this tutorial:

run.sh - Build and run the containers and network

list.sh - List the containers in the network

stop.sh - Stop the running nodes in the network

resume.sh - Resume running the containers in the network

remove.sh - Shut down the network and remove the containers

## Run the Private Network

### Start the Network

**On Windows** Use Windows PowerShell or Admin cmd shell.

To start the network, start a terminal window, and invoke the run script from the pantheon-quickstart directory:

./run.sh -p 32777

The optional -p option specifies a port number; if you don't specify a port, a random number is assigned.

**On Mac/Linux** The run script lists the nodes and endpoints.

**On Windows** The scripts spawn a child cmd window that exits on completion. However, the Pantheon nodes are running on Docker in the background. You can invoke docker or docker-compose commands on the running containers at any time. Run the following to list the nodes and endpoints:

# List the containers

docker-compose ps

# Get the RPC endpoint port

docker-compose port explorer 80

Note that the Quickstart run script creates a network with a number of nodes with various purposes:

* Peer nodes: the regular nodes sync to each other and have no other function
* Boot node: a gateway node through which the peer nodes can discover each other on the network
* Miner node: the node that performs Ethereum mining
* RPC node: allows HTTP/RPC clients such as Postman to communicate with the network using the JSON-RPC API

## Run the Block Explorer

In this step you'll run the Alethio light block explorer to view block information.

**On Mac OS / Linux**

When you run the list script, the Web block explorer URL is listed after the container list.

**On Windows**

To access the Web block explorer, enter it as a URL in the format http://localhost:<port> in your browser; for example: http://localhost:32777.

The block explorer displays a summary of the private network.

Note that the explorer indicates 6 peers: the 4 regular nodes, the mining node, and the bootnode.

* Click the block number to the right of **Best Block** to display the block details.
* Explore blocks by clicking on the blocks list under the **Bk** icon on the left side of the page.
* Search for a specific block, transaction hash, or address by clicking the magnifying glass in the top left-hand corner.

## Postman: Make JSON-RPC Requests on the Network

Now let's open Postman to run JSON-RPC commands.

Start Postman and click **Import**. Import the Postman collection JSON file in the pantheon-quickstart/docs directory. The collection contains the following methods:

* Get the peer count using net\_peerCount.
* Get the index of the most recently mined block, using eth\_blockNumber.
* Get the balance of the mining address defined in the miner node using eth\_getBalance.
* Get a transaction receipt using eth\_getTransactionReceipt.
* Get transaction information using eth\_getTransactionByHash.

##### Counting Peers

Poll the peer count using net\_peerCount. The result should be as follows:

{

"jsonrpc" : "2.0",

"id" : 1,

"result" : "0x6"

}

* The peer count of 6 includes peer nodes 1-4, the miner node, and the boot node. Because the RPC node does the polling, it only counts nodes other than itself.

##### **Requesting the Most Recently Mined Block Index**

Call eth\_blockNumber to retrieve the (0-based) index of the most recently mined block. The result should be similar to the following:

{

"jsonrpc": "2.0",

"id": 13,

"result": "0x11b3"

}

* Run this method and immediately switch to the block explorer; see how closely the block numbers match.
* Open the **Calculator**; switch to **Programmer**; enter the Hexadecimal number and convert it to Decimal.
* The block explorer polls the net and displays blocks about every 10 seconds, and the network mines about 1 block every 5 seconds.

## MetaMask: Balances and Transactions

Now we'll use MetaMask to check account balances and send transactions.

Click on the MetaMask plugin icon at the top of your browser near the address bar. Log in to MetaMask.

**Note** If you have previously done transactions on a MetaMask test account, you should reset the account (**Settings > Reset Account**) before sending transactions again when you re-run the network.

##### Connect to the Private Network

Connect to the private network RPC endpoint as follows:

1. In the MetaMask network list, select **Custom RPC**. This will take you to **Settings**.
2. In the **New RPC URL** field, enter the JSON-RPC HTTP service endpoint displayed when you started the private network (e.g. http://localhost:32777/jsonrpc).
3. Click **Save** to save the configuration.

You'll return to the MetaMask main screen. Your current network is now set to the private network RPC node.

**Note** If MetaMask doesn't indicate clearly that you switched networks, click the network gadget (or spinning icon) to refresh. The **Network** drop-down should indicate **Private Network**.

##### Import the Miner Coinbase Account

Import the mining node coinbase account into MetaMask:

* Click on the accounts icon in the upper right corner of MetaMask, then select **Import Account**.
* In the **New Account** dialog, select the **Import** tab and **Private Key** and copy and paste the mining node's private key in the edit box:

0x8f2a55949038a9610f50fb23b5883af3b4ecb3c3bb792cbcefbd1542c692be63

* MetaMask will assign the new account a name such as Account 1 (if you have existing accounts the number will increment) and tag it with an **Imported** label.

**Note** For this tutorial, we don't have to be concerned about securing the keys, but be sure to secure your accounts in a real world case.

##### Create a Test Account

Now create a test account. We'll send Ether to this account. (Remember, in this tutorial we're dealing with valueless ether because we're not on mainnet, but rather on a local private network.)

* Click on the accounts icon in the upper right corner of MetaMask, then select **Create Account**.
* In the **New Account** dialog, select the **Create** tab, and accept the default account name. Then rename the account something memorable like Test Account.

In MetaMask, copy the new account's address by clicking the account's name and selecting **Copy address to clipboard**. The address will be a long hexadecimal number such as:

0xCe30773FA39E9e409f59E07C5C54CEef3Bc900De

In the block explorer, search for the new account by clicking on the magnifying glass and pasting the account address into the search box. The account will be displayed with a balance of 0 ETH.

##### Check the Miner Account Balance

Call eth\_getBalance and pass the coinbase account address of the mining node to retrieve the balance. The account address is provided in the parameters as shown below:

{"jsonrpc":"2.0","method":"eth\_getBalance","params":["0xfe3b557e8fb62b89f4916b721be55ceb828dbd73","latest"],"id":1}

The result should be similar to the following:

{

"jsonrpc" : "2.0",

"id" : 1,

"result" : "0x21f82b357f9b2680000"

}

* The result is the mining node balance in Wei. Using the Ether converter at <https://etherconverter.online/> enter the resulting value in Wei and convert to Ether. For example, the value above is 10026 ETH.
* Compare the balance from eth\_getBalance to the balance shown in MetaMask.

##### Send Ether

In MetaMask, send Ether from the mining account (which contains Ether) to the new test account (which contains no Ether):

* Click on the accounts icon in the upper right corner of MetaMask, then select the mining account.
* On the mining account page, click **Send**.
* In **From**, select the mining account
* In **To**, select the test account.
* In **Amount**, enter 1 ETH.
* Change **Gas Fee** to 30000 (from the default value of 21000).
* Click **Next**. On the next page, review the information and click **Confirm**.

Now let's view the transaction in the Alethio block explorer:

* In MetaMask, view the mining account, which lists transactions. Click the most recent transaction to display details, then click the **View on Etherscan** link. (Although Etherscan can't show valid data for a private network transaction, it can display the TxHash.)
* On the Etherscan page, copy the TxHash to clipboard.
* Return to the block explorer in the browser. Enter the TxHash in the search box. The transaction view (Tx) will appear in the block explorer, showing the information for the transaction.
* Call eth\_getBalance again and compare the resulting balance to the balance shown in MetaMask.

##### View the Transaction Receipt

Let's go back to Postman and make a request via JSON-RPC. To get the receipt of a transaction, call eth\_getTransactionReceipt. Click the Body tab and make sure the params field is set to the transaction hash that we copied from MetaMask:

"params" :["0x31083c7d816efce314876b0707ca612ac5ca8241de1661db6ab982507aadd8b9"]

Now make sure the URL has the correct port number, and click **Send**. eth\_getTransactionReceipt returns information about the transaction like from (sending account), to (receiving account), blockHash, blockNumber, gasUsed, etc.

We can return to the Alethio Tx Explorer and verify that the transaction info displayed there corresponds to the values that eth\_getTransactionReceipt returned.

## Remix: Deploy Smart Contracts

In this step we'll deploy a smart contract using Remix and MetaMask. Remix is an IDE for writing, deploying, and running smart contracts that runs in your browser. It will work in conjunction with MetaMask to send the contract deployment.

##### Create and Deploy a Contract

Open <http://remix.ethereum.org> in your browser and enter the following code in the Remix editor:

pragma solidity ^0.5.1;

contract Storage {

uint256 public value;

function setValue(uint256 \_value) public {

value = \_value;

}

}

On the **Compile** tab, select **Auto-Compile**.

On the **Run** tab, make sure that **Environment** is **Injected Web3**, and that **Account** is set to the address of the mining node account.

Click **Deploy**.

A MetaMask transaction confirmation dialog will appear, indicating **Contract Deployment** and **0 ETH** to be transferred. Click **Confirm** to pay the fees and complete the contract deployment. Remix will indicate a successful contract deployment under the sections **Transactions Recorded** and **Contract Deployment** at the bottom right of the page.

##### Set a Parameter Value

The deployed contract has a parameter called value, set to 0 by default. Let's set it to another value.

In **Contract Deployment** at the bottom right of the page, click the deployed contract (which reads **Storage at <address>**).

Click the **value** button; it will display the default value 0: uint256: 0.

In the **setValue** box, enter 99 and click the **setValue** button. A MetaMask notification will appear asking for confirmation.Click **Confirm**.

Go to MetaMask and view the transaction listed as **Unknown Function**, with **0 ETH** transferred.

Return to Remix and click the **value** button; it will display the value you set: 0: uint256: 99.

Also, in the bottom left panel, Remix shows the transactions that have been sent. Click the down arrow gadget to expand each transaction and view detailed transaction data. For example, in the decoded input field, you can find the value setting (0: uint256: 99).

##### View the Function Transaction in Block Explorer

* In MetaMask, find the function transaction; click the link to Etherscan.
* Copy the TxHash from the Etherscan page.
* Paste the TxHash in block explorer (in the search box on the main page).
* Block explorer shows you the **Tx** View, with transaction data.
* Click the block number to view the **Bk** view, which has data about the block that contains the transaction.