Adi Release

February 22, 2025

Prasaga

Our first release, the Adi release, is an individual private test development sharded static blockchain to introduce the Sagachain concepts and allow some preliminary experimentation with these concepts in a limited, but live example system. Specifically, Sagachain nodes are instances of a docker image and communicate using a docker swarm overlay network. Nodes also communicate using the libp2p gossipsub package and libp2p rendezvous instances. A webserver instance and mongodb database provide a blockchain explorer to monitor block production and transactions of the running nodes. A configured system enables developers to run a multi-sharded blockchain, create new account objects, new class objects, new object instances, send transactions involving arbitrary sets of accounts, and monitor results with the block explorer.

Disclaimers: This release is a work in progress. It is a functioning blockchain, but it is not a crypto blockchain and is not secured by block signature authentication. Although the capability exists to implement transaction fees, generate coin, and general coin management, that is not the focus of this release and is not used as part of the transaction examples.

Prerequisites:

Sagachain is built for Linux x86\_64 architectures. Development and testing is generally carried out on Amazon Linux 2023 on EC2 instances. This release is in a Docker image minimizing system dependencies as much as possible. Configuration files require some manual editing, and a couple system specific utility replacements. The following are the minimal tools that must be installed:

Docker - Docker daemon (dockerd) and Docker CLI (docker)

Note: The Docker ports needed for the Docker swarm overlay bridge must be open between machines:

* TCP port 2377
* TCP and UDP port 7946
* UDP port 4789

(reference: [Networking with overlay networks | Docker Docs](https://docs.docker.com/engine/network/tutorials/overlay/))

Npm and node.js - [Downloading and installing Node.js and npm | npm Docs](https://docs.npmjs.com/downloading-and-installing-node-js-and-npm)

Pm2 (npm install pm2 -g)

Mongodb - [Install MongoDB Community Edition on Linux - MongoDB Manual v8.0](https://www.mongodb.com/docs/manual/administration/install-on-linux/?msockid=04790d77b32a6db30fda18f6b2766c13)

ssh (ssh, sshd, scp)

standard Linux utilities

Installation

The Sagachain Node is hosted at [Repositories | sagachain](https://hub.docker.com/repositories/sagachain), sagachain/node:v1.0.0. Configuration and runtime scripts are hosted at [SagaChain / configs · GitLab](https://code.prasaga.com/sagachain/configs). This short setup script will clone the configuration, databases, and block explorer related projects.

#!/bin/bash

echo "\*\*\*Adi release for developers\*\*\*"

git clone --branch v3.1.1 --depth 1 --single-branch https://code.prasaga.com/sagachain/configs.git

git clone --branch v3.1.1 --depth 1 --single-branch https://code.prasaga.com/sagachain/databases.git

git clone --branch v3.1.1 --depth 1 --single-branch https://code.prasaga.com/sagachain/sagascan-ui.git

git clone --branch v3.1.1 --depth 1 --single-branch https://code.prasaga.com/sagachain/sagascan-server.git

git clone --branch v3.1.1 --depth 1 --single-branch https://code.prasaga.com/sagachain/saga-bootstrap-node.git

To call:

curl <https://code.prasaga.com/sagachain/configs/-/raw/main/sagachain-bootstrap.sh> | bash -s

Docker Confguration

The Node containers use the docker overlay network for the consensus protocol, and host IP visible libp2p gossip rendezvous instances for the libp2p gossipsub module. Docker swarm and overlay network configuration documentation can be found on the docker website tutorial section. If all of the Node docker containers are run on the same machine, the default docker bridge without the overlay bridge configuration may be used.

Libp2p Gossip Bootstrap Configuration

The Node gossipsub module uses a bootstrap rendezvous instance for peer discovery. At least one rendezvous instance must be available via a host IP visible address. The tool saga-bootstrap-node/bootstrap-node implements a basic rendezvous instance. It outputs a bootstrap peer libp2p multiaddress template that is used added to each Node configuration file. Saga-bootstrap-node/bootstrap-node supports peer rendezvous instances which can be passed as a command line argument with the -peers switch. See the source code for details saga-bootstrap-node/main.go.

The bootstrap peers configuration is described in the setup section below.

Node Configuration

Each Node takes a commandline parameter “--config=<path>/<name>.yaml file. And a logfile configuration which defaults to noderlog.conf or an be set with the environment variable: NODE\_CONF\_FILE.

The Node configuration file and required configurations is described in the setup section below.

Sagachain Configuration

The set of shards, Nodes assigned to each shard, and operating modes of each Node are defined in a configuration file: saganodelist.yaml. Each Node configuration file described above includes a reference to this configuration file with the tag “nodelistpathname:”.

The Sagachain configuration file and required configurations are described in the setup section below.

Sagachain Launching

A series of rudimentary bash scripts are provided for setting up and launching a set of configured Node to run as a Sagachain multi-shard blockchain. Unlike adding a new Node instance to a running blockchain, the development scripts launch all of the Nodes as a complete set using the following scripts:

Make\_sagachain\_config.sh

Docker-sagachain-node.sh

Sagachain-manager.sh

Runnode\_sagachain.sh

The configuration and usage is described in the setup section below.

Sagascan Server Configuration

Details on configuration and launching the sagascan server can be found in the readme at:

<https://code.prasaga.com/sagachain/sagascan-server>

reproduced here:

**SagaScan-Server**

Web and websocket api server for the SagaScan-UI and as basis for external projects to access SagaChain. Streams recent blocks and contained transactions via websocket by connecting to a SagaChain Node via GRPC. Also stores the data in a mongoDb database for indexing and later retrieval via hash or block id.

In the future additional elements will be made available, e.g. nodes, accounts, objects, as well as the ability to filter by account, etc. Also the ability to submit transactions will be added.

API documentation to follow. Client library reference implementations will be provided.

The .proto files are copied from the repo saga-pbtypes.

**Environment variables (.env file)**

SAGA\_NODE defaults to localhost:5080

Uses mongodb with DB\_URL defaulting to mongodb://localhost:27017/sagachain. Set USE\_DB=0 to disable mongodb, useful for debugging. While debugging nodes to handle block resets set DETECT\_RESTARTS=1

To init the db see mongo.txt , to reset db use mongosh < resetDb.js

For example nginx setup see the frontend repo sagascan-ui.

npm install

npm run dev

Sagascan-UI

Details on configuration and launching the sagascan UI can be found in the readme at:

https://code.prasaga.com/sagachain/sagascan-ui

reproduced here:

**SagaScan-UI**

SagaChain viewer, offers a streaming display of latest blocks and transactions. Block detail view with included transactions, transaction details including executed script and results. Blocks and transactions can be searched by hash.

Display overall statics and charts with blocks/s and transactions/s

Connects to the SagaScan-server web and websocket api.

In the future it will be expanded to display and modify accounts and individual objects, submit transaction scripts and display node information as well as additional statistics.

**Installation**

npm install

npm run build

pm2 start .output/server/index.mjs

uses port 3000

Copy .env.default to .env and adjust VITE\_API for the SagaScan-server URL

example nginx config section

location / {

proxy\_pass http://localhost:3000;

proxy\_redirect off;

proxy\_set\_header Host $host;

proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;

proxy\_set\_header X-Real-IP $remote\_addr;

}

location /\_nuxt {

root /home/ec2-user/sagachain-explorer/.output/public/;

}

location /api {

rewrite ^/api(.\*)$ $1 break;

proxy\_pass http://localhost:3001;

proxy\_redirect off;

proxy\_set\_header Host $host;

proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;

proxy\_set\_header X-Real-IP $remote\_addr;

}

location /api/ws {

rewrite ^/api(.\*)$ $1 break;

proxy\_pass http://localhost:3001;

proxy\_redirect off;

proxy\_set\_header Host $host;

proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;

proxy\_set\_header X-Real-IP $remote\_addr;

proxy\_set\_header Upgrade $http\_upgrade;

proxy\_set\_header Connection "upgrade";

}

Example Test Transactions

A minimal set of two test scripts exercise the blockchain as follows:

A transaction that creates a new Sagapython class implementing a simple counting asset and owned by the built-in SystemAccount object, and creates a new instance of the class which is also owned by SystemAccount.

A transaction that sends a message to the new object instance increment the asset count. This transaction is submitted repeatedly by a shell script, increment the transaction sequence number for each submission.

A set of three transactions the can be used to create new accounts, create new object instances owned by the new accounts (currently uses the class previously created above), and sends messages to the object instances owned by the accounts. Some manual editing is required to capture the newly created the object IDs (32 byte hash values), for the new accounts and objects.

The transactions configuration and usage are described after setup section below.

Setup

Once the download script is run, the following directories will have been cloned:

Configs and databases.

The make\_sagachain\_config.sh creates a working copy of the configs and databases directory with a name for the new blockchain. It takes 3 parameters:

1: Path to the parent directory of the configs and databases directories from the install script.

2: Name of the blockchain to create – creates a directory by that name in the current working dir

3: Number of nodes for the blockchain. See blockchain configuration and Node configuration below. The default is 16, 4 nodes per shard and 4 shards.

>> make\_sagachain\_config.sh <install root>/ <blockchain> 16

On completion, the <blockchain> directory containing a configs directory and a databases directory should exist.

Configuring the Libp2p Gossip Bootstrap peers

The Nodes connect to one or more libp2p rendezvous bootstrap hosts to discover each other for gossip traffic. The saga-bootstrap-node directory contains bootstrap-node. The command will run a libp2p rendezvous process and output a unique key. The unique key is written to a file .priv in the current working directory.

>> saga-bootstrap-node/bootstrap-node

Output:

[\*] Your Bootstrap ID Is: /ip4/0.0.0.0/tcp/5000/p2p/QmPA5aeSDNaryS3b8dCXFdVY3MmoaTfqv16dsHRfYYazCh

Of which, the key is:

QmPA5aeSDNaryS3b8dCXFdVY3MmoaTfqv16dsHRfYYazCh

The command may be run detached in the background:

Nohup saga-bootstrap-node/bootstrap-node &

And the nohup.out will contain the output.

The saga-bootstrap-node/bootstrap-node takes a -peers flag which is a list of other bootstrap nodes such that several may refer to each other.

Configuring the bootstrap peers in the sagachainnodeXX.yaml files

In the <blockchain name/configs directory edit sagachainnode1.yaml.

Search for the bootstrap\_peers: entry list. There are several peers already listed. These can be deleted. Use the entries as a template, replacing the IP address with the host IP address and the key with the generated key from above.

Any number of bootstrap\_peers may be listed, at least one must be listed and available for the nodes to connect to.

SagachainnodeXX.yaml replication script

To avoid manual editing each sagachainnodeXX.yaml file a short script will duplicate sagachainnode1.yaml, editing the node name in the configuration file, node2, node3, …

It does a couple simple sed edits.

Run this in the <blockchain name>/configs directory:

>> gen-chainnode-configs.sh 17

This will recreate sagachainnode2.yaml, … sagachainnode16.yaml from the sagachainnode1.yaml edited version.

Sending the configuration to the rest of the machines:

Each machine needs a copy of the configuration files in a <blockchain name>/configs directory and a copy of the databases in <blockchain name>/databases. A minimal script is provided to do start, stop, copying commands, sagachain-manager.sh. This depends on ssh/sshd being configured correctly. It expects to find the keys in ~/.ssh as ${hostname}.pem files.

The make\_sagachain\_config.sh make a copy of the ./etc directory with 2 files: hosts and nodes. Example hosts file:

ec2-user@50.18.65.58 Node1

#ec2-user@54.215.87.20 Node2

ec2-user@54.176.27.64 Node3

ec2-user@3.101.67.251 Node4

localhost Node2

ec2-user@3.101.86.194 Node5

The first entry is the username and host for each machine that will run one or more Nodes. The second is the prefix name for the .pem files in .ssh, (e.g. ~/.ssh/Node1.pem)

The etc/nodes file lists the location of each Node instance by hostname and number. Example nodes file:

Node1

Node2 1 2 3 5 6

Node3

Node4 9 10 11 13 14 15

Node5 4 7 8 12 16

This means Node2 will run Nodes with configuration files sagachainnode1.yaml, sagachainnode2.yaml, etc. and Node4 and Node5 will run Nodes with the respective sagachainnodeXX.yaml config files.

Configuring the docker run script

The docker-sagachain-node.sh edits the sagachainnodeXX.yaml files to use the host machine IP address for each Node instance. Development has been hosted on AWS EC2 machines, such that the default uses the ec2-metadata command. The lines edit are near the top of the file, reproduced here:

# command to retrieve the hosts externally visible IP address

# if using AWS ec2, following command works. Replace for other platforms

export EXT=$(ec2-metadata -v --quiet)":$p4"

export LOC=$(ec2-metadata -R --quiet)

echo "- $EXT $LOC"

It is recommended to not hardcode these, use an appropriate host command to retrieve the host IP address on each run. This makes it easier to move Nodes around using the sagachain-manager.sh script described below.

Deploying the configs and databases

After completing configuring the hosts and nodes files, and editing the docker-sagachain-node.sh file, the sagachain-manager.sh may be used to copy the directories to the target machines using scp:

>> sagachain-manager.sh <blockchain name> scp <blockchain name> <blockchain name>

The first parameter to sagachain-manager.sh is always the blockchain name. The second is one of start, stop, start-debug, scp, and several others. Please refer to the case statement in the bash script for the commands.

The scp utility takes two directories, the source and destination. In this case it copies the configuration directories from the source machine to all of the target machines.

Docker Swarm and Overlay Configuration

For details on docker configuration please refer to the online docker network tutorial at:

[Networking with overlay networks | Docker Docs](https://docs.docker.com/engine/network/tutorials/overlay/)

The relevant commands are:

>> Docker swarm init

To create a network with this machine as the designated network manager. It returns a network token that is used for joining the other host machines.

>> docker swarm join - -token <returned token> <manager IP address>:2377

On each of the other host machines.

Note: there is a general “cmd” command for sagachain-manager.sh available:

>> sagachain-manager.sh <blockchain name> cmd docker swarm join –token <…> <ip address>:2377

Launching the Blockchain

The sagachain-manager.sh will start the blockchain by downloading the sagachain Node image from hub.docker.com and execute each of the instances on the configured host machines with the appropriate sagachainnodeXX.yaml configuration file. As part of the initial launch sagachain-manager creates a temporary directory: saganodetmp-<blockchainname> and makes a copy of the configuration files and the sagachainnodeXX.yaml files used on each host machine.

>> sagachain-manager.sh <blockchain name> start sagachain/node:v1.0.0

The start command takes 4 additional parameters with are TCP port numbers. The docker bridge configuration uses port numbers to distinguish between instances. The port numbers used for each instance are output as part of the start command.

The start command copies the configuration from the <blockchain name> directories to the saganodetmp-<blockchain-name> directories for each run. A separate command, start-debug will reuse the configuration files in the saganodetmp-<blockchain-name>. This is useful for enabling various log tracing parameters for a Node instance.

Note: the source code is heavily instrumented with log statements. The bottom section of the sagachainnodeXX.yaml files have examples of trace flags that may be enabled. The noderlog.conf file in the configs directory controls the overall logging using the Go rlog package. To enable trace logging:

RLOG\_TRACE\_LEVEL = 1000 or higher value.

To verify the Nodes are up:

Docker ps

Should show the list of nodes for the machine that you have configured the etc/nodes for.

Docker logs -f <blockchain name>-nodeX X

To see the log of the main node process for the container. Each container will have the following:

node process – implements consensus, and manages everything

saga-leveldb process – implements the object state database

executor processes – farm of sagapython processes to execute transactions

extractheader processes – farm of sagapython process for initial header reads on transaction submission.

Sagachain Blockchain Explorer and Submitting Transactions

The sagachain blockchain explorer is found in the directories sagascan-server and sagascan-ui. Each has a readme for launching. To run:

1: copy the .env.default to .env and edit if needed.

2: execute mongodb instance with the commands in mongo.txt found in sagascan-server

3: execute the server:

>> npm install ; npm run dev; npm run pm2

The server connects to a Node instance, which is usually co-resident on the same host machine. If not the .env should edited appropriately.

Sagascan-UI

Change to the sagascan-ui directory. Copy the .env.default to .env. Edit the VITE\_API if the sagascan-server instance is on a different machine.

Execute the UI:

>> npm install ; npm run dev; npm run pm2

Transaction Usage

The following walkthroughs demonstrate the following:

1: Creating a new class object on Sagachain

2: Creating an instance of the new class, owned by the built-in SystemAccount

3: Sending messages to the new instance to increment a stored field value

4: Creating a new account object

5: Creating a new object of the class previously created, owned by the new account object

6: Sending messages to the object owned by the new account to increment a stored field value.

The example transaction scripts may be found in configs/testscripts. The best way to run the transactions is from within a Node container. A separate container may be launched or one can connect to one of the existing Node containers:

>> docker exec -it <blockchain name>-nodeXX /bin/bash

Change to the testscripts directory:

>> cd /etc/sagachain/node/configs/testscript

>> . setup.sh