Benchmark a Cluster

The Solana git repository contains all the scripts you might need to spin up your own local testnet. Depending on what you're looking to achieve, you may want to run a different variation, as the full-fledged, performance-enhanced multinode testnet is considerably more complex to set up than a Rust-only, singlenode testnode. If you are looking to develop high-level features, such as experimenting with smart contracts, save yourself some setup headaches and stick to the Rust-only singlenode demo. If you're doing performance optimization of the transaction pipeline, consider the enhanced singlenode demo. If you're doing consensus work, you'll need at least a Rust-only multinode demo. If you want to reproduce our TPS metrics, run the enhanced multinode demo.

For all four variations, you'd need the latest Rust toolchain and the Solana source code:

First, setup Rust, Cargo and system packages as described in the SolanaREADME

Now checkout the code from github:

git clone https://github.com/solana-labs/solana.git cd solana The demo code is sometimes broken between releases as we add new low-level features, so if this is your first time running the demo, you'll improve your odds of success if you check out thelatest release before proceeding:

TAG

(git describe --tags (git rev-list --tags --max-count = 1)) git checkout TAG

Configuration Setup

Ensure important programs such as the vote program are built before any nodes are started. Note that we are using the release build here for good performance. If you want the debug build, use justcargo build and omit theNDEBUG=1 part of the command.

cargo build --release The network is initialized with a genesis ledger generated by running the following script.

NDEBUG

1 ./multinode-demo/setup.sh

Faucet

In order for the validators and clients to work, we'll need to spin up a faucet to give out some test tokens. The faucet delivers Milton Friedman-style "air drops"(free tokens to requesting clients)to be used in test transactions.

Start the faucet with:

NDEBUG

1 ./multinode-demo/faucet.sh

Singlenode Testnet

Before you start a validator, make sure you know the IP address of the machine you want to be the bootstrap validator for the demo, and make sure that udp ports 8000-10000 are open on all the machines you want to test with.

Now start the bootstrap validator in a separate shell:

NDEBUG

1 ./multinode-demo/bootstrap-validator.sh Wait a few seconds for the server to initialize. It will print "leader ready..." when it's ready to receive transactions. The leader will request some tokens from the faucet if it doesn't have any. The faucet does not need to be running for subsequent leader starts.

Multinode Testnet

To run a multinode testnet, after starting a leader node, spin up some additional validators in separate shells:

NDEBUG

1 ./multinode-demo/validator-x.sh To run a performance-enhanced validator on LinuxCUDA 10.0 must be installed on your system:

./fetch-perf-libs.sh NDEBUG = 1

SOLANA_CUDA

1 ./multinode-demo/bootstrap-validator.sh NDEBUG = 1

SOLANA_CUDA

1 ./multinode-demo/validator.sh

Testnet Client Demo

Now that your singlenode or multinode testnet is up and running let's send it some transactions!

In a separate shell start the client:

NDEBUG

1 ./multinode-demo/bench-tps.sh

runs against localhost by default

What just happened? The client demo spins up several threads to send 500,000 transactions to the testnet as quickly as it can. The client then pings the testnet periodically to see how many transactions it processed in that time. Take note that the demo intentionally floods the network with UDP packets, such that the network will almost certainly drop a bunch of them. This ensures the testnet has an opportunity to reach 710k TPS. The client demo completes after it has convinced itself the testnet won't process any additional transactions. You should see several TPS measurements printed to the screen. In the multinode variation, you'll see TPS measurements for each validator node as well.

Testnet Debugging

There are some useful debug messages in the code, you can enable them on a per-module and per-level basis. Before running a leader or validator set the normal RUST_LOG environment variable.

For example

- To enableinfo
- everywhere anddebug
- only in the solana::banking_stage module:
- export
- RUST_LOG
- =
- solana
- =
- info,solana::banking_stage
- =
- debug
- To enable SBF program logging:
- export
- RUST_LOG
- =
- solana_bpf_loader
- =
- trace

Generally we are usingdebug for infrequent debug messages,trace for potentially frequent messages andinfo for performance-related logging.

You can also attach to a running process with GDB. The leader's process is namedsolana-validator :

sudo gdb attach < PID

> set logging on thread apply all bt This will dump all the threads stack traces into gdb.txt

Developer Testnet

In this example the client connects to our public testnet. To run validators on the testnet you would need to open udp ports8000-10000 .

NDEBUG

1 ./multinode-demo/bench-tps.sh --entrypoint entrypoint.devnet.solana.com:8001 --faucet api.devnet.solana.com:9900 --duration 60 --tx_count 50 You can observe the effects of your client's transactions on ourmetrics dashboard Previous Available Solana Clusters Next Cluster Performance Metrics