# # System Specifications

- Operating System: Ubuntu 20.04

- Virtualization Software: Docker 19.03.13

- CPU: AMD Ryzen 5 5600x

- Memory: 32GB 3600MHz

- Storage: Samsung 970 Evo Plus 2TB (M.2 SSD)

# # Default Benchmark Specifications

Unless stated otherwise, the following base configurations are used.

## ## Base Configuration (Common)

- Transaction Scope: public

- Network Load: `OUTPUT OF EXPERIMENT 1`

- Block Time: 2 seconds (default for both frameworks)

- Total number of requests: 10000 txs warm-up, 50000 txs test

- Number of assets to iterate over for the transfer

## ## Base Configuration Besu

- Framework Version: 21.1.5

- Block Limit: 12.5 MB bytes (current default of Ethereum protocol)

- Consensus Protocol: IBFT 2.0

- Block timeout: 2 seconds

-

## ## Base Configuration Fabric

- Framework Version: 2.2

- Block limit: 2MB preferred, 10MB max, max 500 transactions (default of Fabric)

- Block timeout: 2 seconds

- Consensus Protocol: Raft

- Ordering Service Nodes: 2

- Endorsement policy: Default, i.e. majority of orgs must endorse

- Peers per organization: 1

# # Experiments

## ## Experiment #1: Out-of-the-box performance

- Context: We start by not touching any of the configuration settings of the frameworks themselves by keeping them at their default values.

- Goal: Investigate Throughput and Latency over time between public and private transactions.

- Variables: Network Load and transaction scope (pub/priv).

- Network Load: Start by 10, and keep doubling (10\*2^i)

- Transaction Scope: Public + Private

- Total benchmarks to conduct: 2(i+1) per framework, 4(i+1) total

- Conclusions to make:

- Saturation point at which throughput of both frameworks start to throttle.

- Shape of the throughput and avg. latency as load grows (linear / exponential / log)

- Overhead of private transactions compared to public transactions

- If priv. tx. significantly reduce performance, assume/explain this is due to the added overhead of distributing the transactions before ordering/creating blocks. For the remainder of experiments, assume the same effect applies on its curve, and thus, omit private txs.

## ## Experiment #2: Impact of Block Time on performance

- Context: Block time determines how often a block is created, and thus, how often (and quickly) transactions are ordered.

- Goal: How does changing the block time affect the network performance?

- Changing Variables: Network load and Block Time

- Network Load: Start by 10, and keep doubling (10\*2^i)

- Block Time: 1, 2 (default), 4, 8

- Total benchmarks to conduct: 4(i+1) per framework, 8(i+1) total

- Conclusions to make:

- Low/high block time? sweet spot or as low as possible?

## ## Experiment #3: Impact of Block Limit on performance

- Context: Block Limit determines how many transactions are created per block.

- Goal: How does changing the block limit affect the network performance?

- Changing Variables: Network load and Block Limit

- Network Load: Start by 10, and keep doubling (10\*2^i)

- Block Limit: D(efault), 0.5D, 2D

- Total benchmarks to conduct: 3(i+1) per framework, 6(i+1) total

- Conclusions to make:

- Does increasing the block limit result in higher throughput because there is less block-overhead?

- Does increasing the block limit result in lower throughput because blocks fill up and are created more often?

## ## Experiment #4: Impact of Network Size on performance

- Context: We now know what network load the default settings support.

- Goal: Investigate the effects of changing the network size on throughput

- Changing Variables: Network load (same values Experiment #1 to make #2 immediately comparable) and Network Size

- Network Load: Start by 10, and keep doubling (10\*2^i)

- Network Sizes: 3, 4, 5

- Total benchmarks to conduct: 3(i+1) per framework, 6(i+1) total

- Conclusions to make:

- Insights in the most performant network size / trend

- Maybe correlation of load / traffic (Does load accumulate or shared)

## Extra Ideas:

Write vs. Read+Write