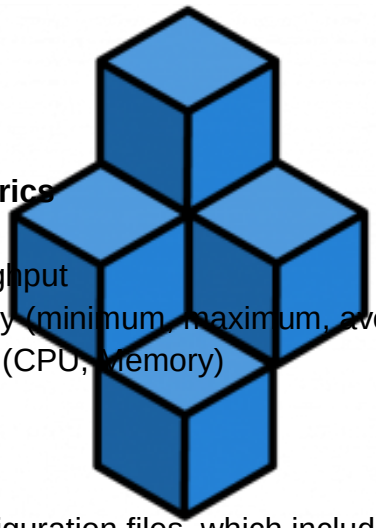
Caliper - A Blockchain performance Benchmark Framework

Caliper is a blockchain performance benchmark framework, which allows users to test different blockchain solutions with custom use cases, and get a set of performance test results.

Supported blockchain solutions

1. Hyperledger Besu
2. Ethereum
3. Hyperledger Fabric



Supported performance metrics

1. Transaction/read throughput
2. Transaction/read latency (minimum, maximum, average)
3. Resource consumption (CPU, memory)

How does it work ?

A user will have to define configuration files, which include:

- A benchmark file defining the arguments of a benchmark workload
- A blockchain file specifying the necessary information, which helps to interact with the system being tested
- Smart contracts defining what contracts are going to be deployed

These configuration files are then fed to the Caliper CLI. The Caliper CLI performs various benchmarking based on benchmark file and generates performance metrics.

During a test, all transactions are saved. The statistics of these transactions are recorded and collated. In addition, a resource monitor also records the consumption of resources. All of this data is pooled together into a single report.

Aim: Performance testing of a smart contract on a pre-existing Fabric network using

Hyperledger Caliper.

Prerequisites:

Note: The network for which you want to check performance must be running and required chaincode must be deployed,

Here a network with two organizations is considered, with the go **asset-transfer-basic** smart contract, built and ready for performance test.

```
curl -sSLO
https://raw.githubusercontent.com/hyperledger/fabric/ma
in/scripts/install-fabric.sh && chmod +x install-fabric.sh
```

```
./install-fabric.sh -f '2.5.3' -c '1.5.6'
```

(This will fetch all the Hyperledger fabric images, binaries and fabric -samples 2.5 version)

Start up the test-network

cd fabric-samples/test-network/

./network.sh up createChannel

./network.sh deployCC -ccn basic -ccp ../asset-transfer-basic/chaincode-go/ -ccl go

Step 1:
Create a Caliper Workspace

Create a folder named **caliper-workspace** at the same level as the fabric-samples directory, and then within the caliper-workspace folder, create three folders named **networks**, **benchmarks**, and **workload** respectively. ((Note: working directory should be at the same level of **fabric-samples**, so if the terminal is at fabric-samples/test-network/ folder, then use the command **cd ../../**)

mkdir caliper-workspace
mkdir -p caliper-workspace/networks caliper-workspace/benchmarks caliper-

workspace/workload

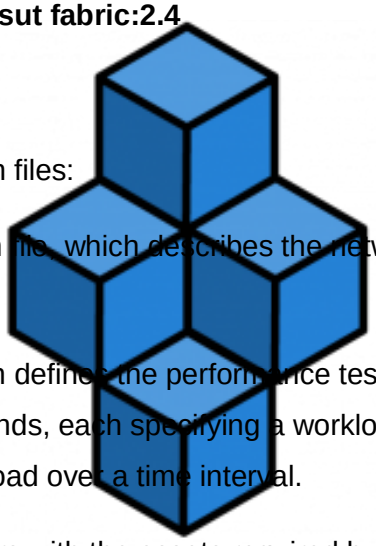
Caliper installation and use will be based on a local npm installation.

Within the caliper-workspace directory, install caliper CLI using the following terminal command:

npm install --only=prod @hyperledger/caliper-cli@0.5.0

Bind the SDK using the following terminal command:

npx caliper bind --caliper-bind-sut fabric:2.4



Caliper requires two configuration files:

- The network configuration file, which describes the network under test and provides the test identities to use.
- The benchmark file, which defines the performance test to be completed via a sequenced set of test rounds, each specifying a workload module and a series of options to drive the workload over a time interval.

We will now populate these folders with the assets required by Caliper.

Step 2
Build a Network Configuration File

Under the **networks** folder create a template file called **networkConfig.yaml**

cd networks/

touch networkConfig.yaml

(Above command will create a **networkConfig.yaml** inside **caliper-workspace/networks**)

Add the following content to **networkConfig.yaml .**

```
name: Caliper test
```

```
version: "2.0.0"

caliper:
 blockchain: fabric

channels:
 - channelName: mychannel
   contracts:
   - id: basic

organizations:
 - mspid: Org1MSP
   identities:
     certificates:
     - name: 'User1'
       clientPrivateKey:
         path:
'../fabric-samples/test-network/organizations/peerOrganizations/org1.
example.com/users/User1@org1.example.com/msp/keystore/priv_sk'
       clientSignedCert:
         path:
'../fabric-samples/test-network/organizations/peerOrganizations/org1.
example.com/users/User1@org1.example.com/msp/signcerts/
User1@org1.example.com-cert.pem'
   peers:
      - endpoint: localhost:7051
        grpcOptions:
          ssl-target-name-override: peer0.org1.example.com
          grpc.keepalive_time_ms: 600000
        tlsCACerts:
          path:
'../fabric-samples/test-network/organizations/peerOrganizations/org1.
example.com/peers/peer0.org1.example.com/msp/tlscacerts/
tlsca.org1.example.com-cert.pem'
```

name: The name for the configuration, in this instance “Caliper test”.

version: The version of the configuration file being used. “2.0.0” ensures the new fabric connectors are used

caliper: Indicates to Caliper the SUT(System under Test) that is being targeted.Here fabric

network is our SUT.

channels: Describes the Hyperledger Fabric channels and the smart contracts deployed on these channels to be benchmarked.

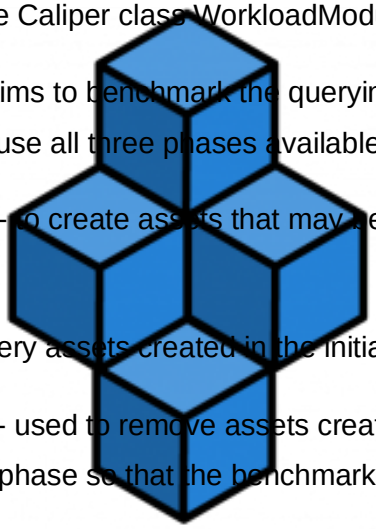
organizations: A list of the Hyperledger Fabric organizations with identities and connection profiles associated with each organization

Step 3
Build a Test Workload Module

The workload module interacts with the deployed smart contract during the benchmark round. The workload module extends the Caliper class WorkloadModuleBase from caliper-core.

The workload we will be driving aims to benchmark the querying of existing assets within the world state database. So we will use all three phases available in the workload module:

- initializeWorkloadModule - to create assets that may be queried in the submitTransaction phase
- submitTransaction - to query assets created in the initializeWorkloadModule phase
- cleanupWorkloadModule - used to remove assets created in the initializeWorkloadModule phase so that the benchmark may be repeated



initializeWorkloadModule

The initializeWorkloadModule function is called by the worker processes before each round, providing contextual arguments to the module.

1. workerIndex (Number) The 0-based index of the worker instantiating the workload module.
2. totalWorkers (Number) The total number of workers participating in the round.
3. roundIndex (Number) The 0-based index of the currently executing round.
4. roundArguments (Object) The user-provided arguments for the round from the benchmark configuration file.
5. sutAdapter (BlockchainConnector) The connector of the underlying SUT.

6. sutContext (Object) The custom context object provided by the SUT connector.

Within the **workload** folder create a file called **readAsset.js** with the following content:

```
'use strict';

const { WorkloadModuleBase } = require('@hyperledger/caliper-
core');

class MyWorkload extends WorkloadModuleBase {
   constructor() {
       super();
   }

   async initializeWorkloadModule(workerIndex, totalWorkers,
roundIndex, roundArguments, sutAdapter, sutContext) {
       await super.initializeWorkloadModule(workerIndex,
totalWorkers, roundIndex, roundArguments, sutAdapter,
sutContext);

       for (let i=0; i<this.roundArguments.assets; i++) {
           const assetID = `${this.workerIndex}_${i}`;
           console.log(`Worker ${this.workerIndex}: Creating
asset ${assetID}`);
           const request = {
               contractId: this.roundArguments.contractId,
               contractFunction: 'CreateAsset',
               invokerIdentity: 'User1',
               contractArguments:
[assetID,'blue','20','penguin','500'],
               readOnly: false
           };

           await this.sutAdapter.sendRequests(request);
       }
   }

   async submitTransaction() {
       const randomId =
Math.floor(Math.random()*this.roundArguments.assets);
```

```
        const myArgs = {
            contractId: this.roundArguments.contractId,
            contractFunction: 'ReadAsset',
            invokerIdentity: 'User1',
            contractArguments: [`${this.workerIndex}_$
{randomId}`],
            readOnly: true
        };

        await this.sutAdapter.sendRequests(myArgs);
    }

async cleanupWorkloadModule() {
        for (let i=0; i<this.roundArguments.assets; i++) {
            const assetID = `${this.workerIndex}_${i}`;
            console.log(`Worker ${this.workerIndex}: Deleting
asset ${assetID}`);
            const request = {
                contractId: this.roundArguments.contractId,
                contractFunction: 'DeleteAsset',
                invokerIdentity: 'User1',
                contractArguments: [assetID],
                readOnly: false
            };

            await this.sutAdapter.sendRequests(request);
        }
    }
}

function createWorkloadModule() {
    return new MyWorkload();
}

module.exports.createWorkloadModule = createWorkloadModule;
```

Step 4
Build a Benchmark Configuration File

The benchmark configuration file defines the benchmark rounds and references the defined workload module(s). It will specify the number of test workers to use when generating the load, the number of test rounds, the duration of each round, the rate control applied to the transaction load during each round, and options relating to monitors.

Under the **benchmarks** folder create a file called **myAssetBenchmark.yaml** with the following content:

```yaml
test:
    name: basic-contract-benchmark
    description: test benchmark
    workers:
      number: 3
    rounds:
      - label: readAsset
        description: Read asset benchmark
        txDuration: 30
        rateControl:
          type: fixed-load
          opts:
            transactionLoad: 2
        workload:
          module: workload/readAsset.js
          arguments:
            assets: 10
            contractId: basic
```

Step 5 - Run the Caliper Benchmark

Run the command

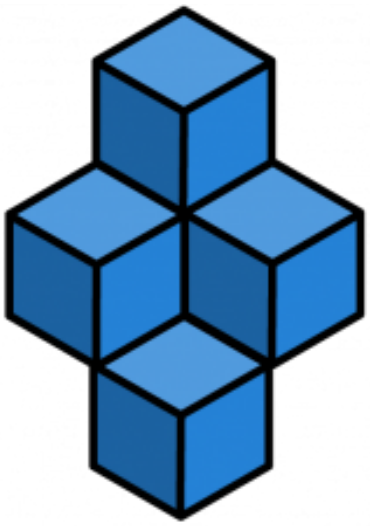
Ensure that you are in the **caliper-workspace** directory.

In the terminal run the following Caliper CLI command:

npx caliper launch manager --caliper-workspace ./ --caliper-networkconfig networks/networkConfig.yaml --caliper-benchconfig benchmarks/myAssetBenchmark.yaml --caliper-flow-only-test --caliper-fabric-gateway-enabled

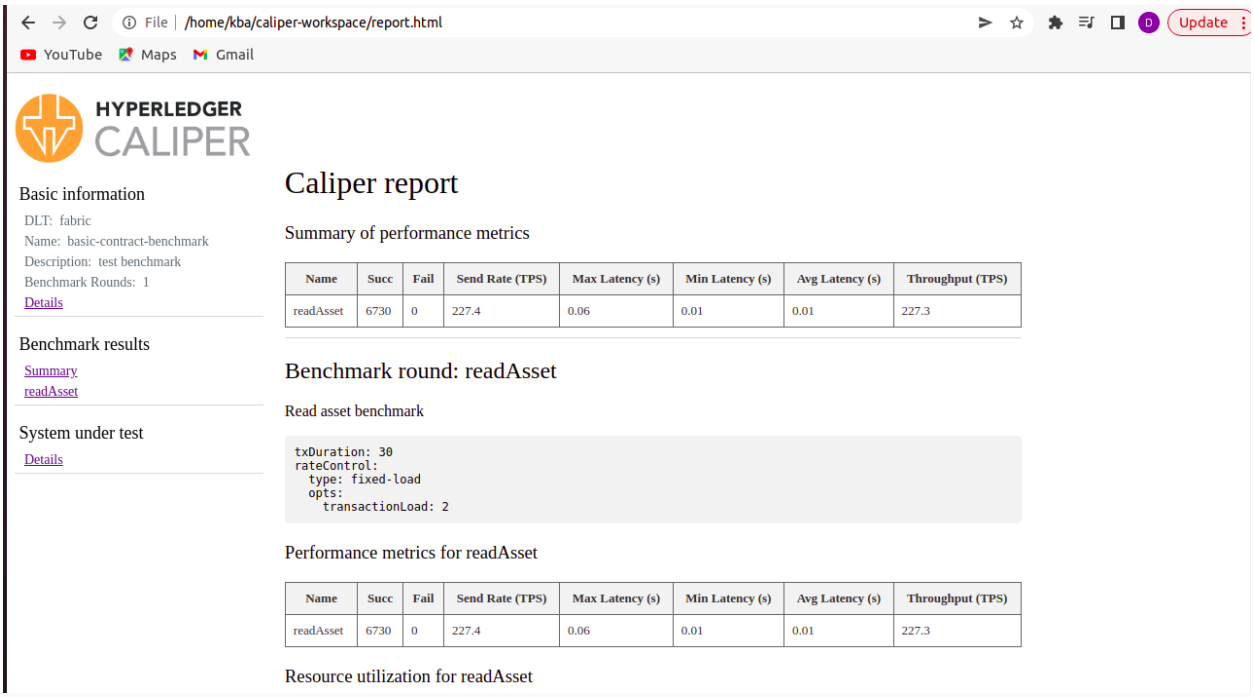
Once we are done with all the above steps our folder structure should look like this

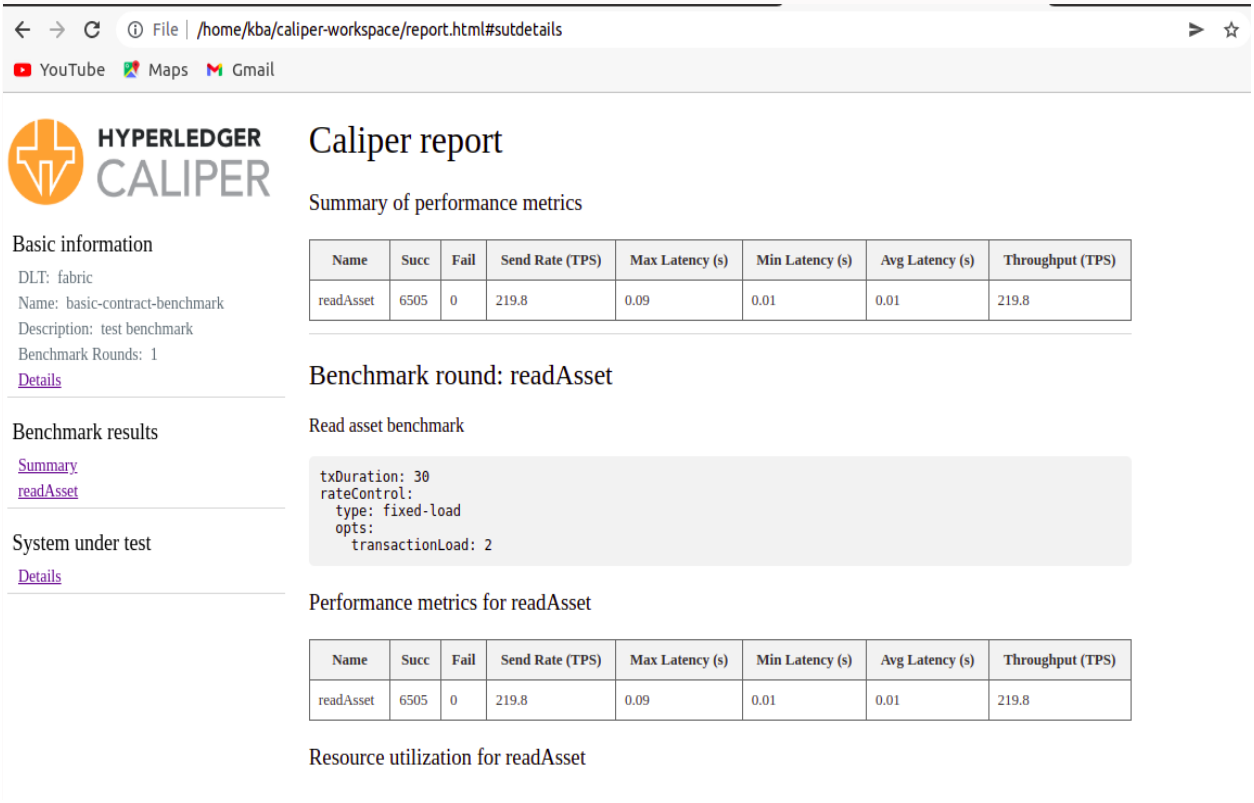
```
caliper-workspace/
├── benchmarks
│   └── myAssetBenchmark.yaml
├── caliper.log
├── networks
│   └── networkConfig.yaml
├── report.html
└── workload
    └── readAsset.js
```

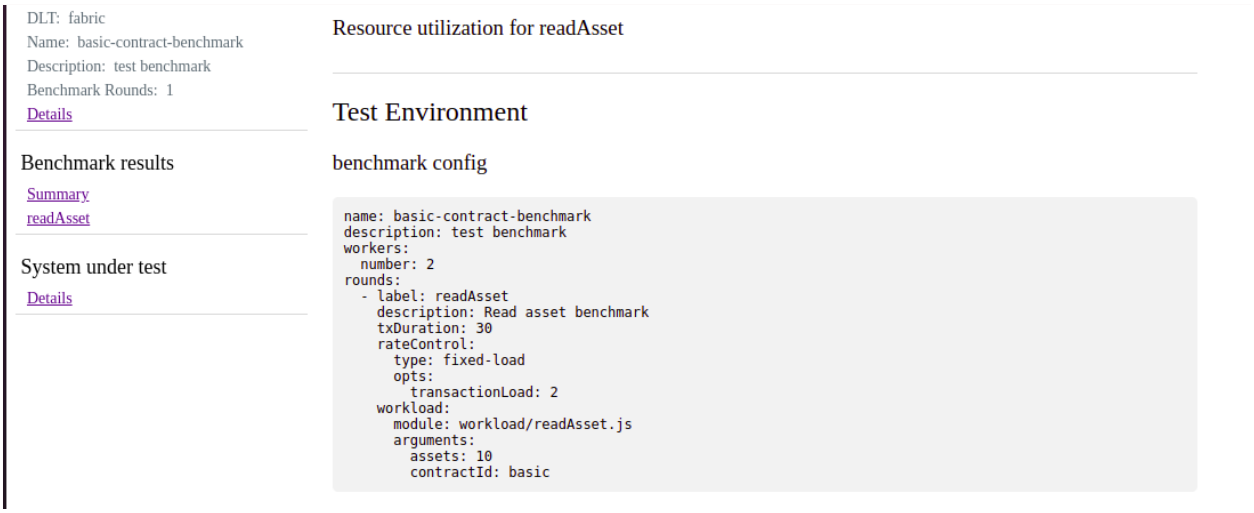


Result:

The result will be available in a file named **report.html** within the folder **caliper-workspace**(Open report.html to view the result)


File | /home/kba/caliper-workspace/report.html
YouTube
Maps
Gmail
Update
HYPERLEDGER
CALIPER
Basic information
DLT: fabric
Name: basic-contract-benchmark
Description: test benchmark
Benchmark Rounds: 1
Details
Benchmark results
Summary
readAsset
System under test
Details
Caliper report
Summary of performance metrics
Name | Succ | Fail | Send Rate (TPS) | Max Latency (s) | Min Latency (s) | Avg Latency (s) | Throughput (TPS)
readAsset | 6730 | 0 | 227.4 | 0.06 | 0.01 | 0.01 | 227.3
Benchmark round: readAsset
Read asset benchmark
txDuration: 30
rateControl:
type: fixed-load
opts:
transactionLoad: 2
Performance metrics for readAsset
Name | Succ | Fail | Send Rate (TPS) | Max Latency (s) | Min Latency (s) | Avg Latency (s) | Throughput (TPS)
readAsset | 6730 | 0 | 227.4 | 0.06 | 0.01 | 0.01 | 227.3
Resource utilization for readAsset


File | /home/kba/caliper-workspace/report.html#sutdetails
YouTube
Maps
Gmail
HYPERLEDGER
CALIPER
Basic information
DLT: fabric
Name: basic-contract-benchmark
Description: test benchmark
Benchmark Rounds: 1
Details
Benchmark results
Summary
readAsset
System under test
Details
Caliper report
Summary of performance metrics
Name | Succ | Fail | Send Rate (TPS) | Max Latency (s) | Min Latency (s) | Avg Latency (s) | Throughput (TPS)
readAsset | 6505 | 0 | 219.8 | 0.09 | 0.01 | 0.01 | 219.8
Benchmark round: readAsset
Read asset benchmark
txDuration: 30
rateControl:
type: fixed-load
opts:
transactionLoad: 2
Performance metrics for readAsset
Name | Succ | Fail | Send Rate (TPS) | Max Latency (s) | Min Latency (s) | Avg Latency (s) | Throughput (TPS)
readAsset | 6505 | 0 | 219.8 | 0.09 | 0.01 | 0.01 | 219.8
Resource utilization for readAsset



The parameters in the Summary of performance metrics are:-

1. **Name**:- name of the benchmark operation performed.
2. **Succ** :- Number of transactions submitted successfully.
3. **Fail** :- Number of transactions failed.
4. **Send Rate (TPS)** :- Number of transactions sent to the network per second.
5. **Max Latency(s)** :- Maximum time taken in the request-response cycle of a transaction.
6. **Min Latency(s)** :- Minimum time taken in the request-response cycle of a transaction.
7. **Average Latency(s)** :- Average time taken in the request-response cycle of all the transactions.
8. **Throughput(TPS)** :- Number of transactions submitted successfully per second.

Reference

1. Hyperledger Caliper Documentation

2. Setting up and Running a Performance Benchmark on an existing network

3.https://hyperledger.github.io/caliper/v0.4.2/fabric-tutorial/tutorials-fabric-existing/

4.https://hyperledger.github.io/caliper/v0.5.0/architecture/

5.https://www.hyperledger.org/learn/publications/blockchain-performance-metrics