DATASTAX®

Realistic Pulsar Load testing made easy with NoSQLBench

Yabin Meng | ApacheCon NA 2022 | Oct 5, 2022

Agenda

1. Introduction
2. Apache Pulsar and Load Testing
3. NoSQLBench(NB) Introduction and NB Pulsar driver(s)
4. Demo
5. Q&A

Introduction

Who am I?

- DataStax Streaming Solutions Architecture Practice Lead
- 6 years at DataStax - C* and Pulsar
- 20+ years IT experience with the focus on database, messaging/streaming and related technologies
- Dedicated to NoSQL and Streaming technology consulting in recent years

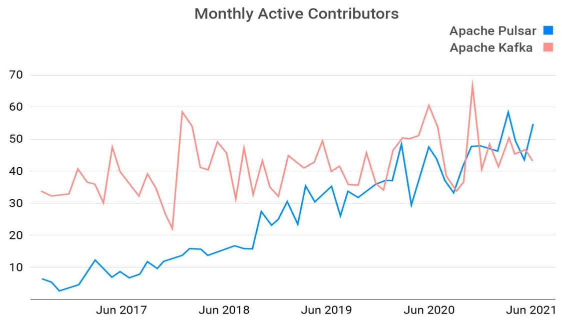
Who is DataStax?

- DataStax was founded by Jonathan Ellis and Matt Pfeil in 2010 - The C* company
- Opens Source @ DataStax
  - DataStax has always been a major contributor to the Apache Cassandra project since early days of C*
  - DataStax has become one of the top contributors to Apache Pulsar project and Bookkeeper project
  - Other key OSS contributions: Stargate, K8ssandra, Starlight APIs for Pulsar etc.
- Multi-cloud SaaS offering
  - Astra DB
  - Astra Streaming
- Enterprise support or Opens source
  - Luna
  - Luna Streaming

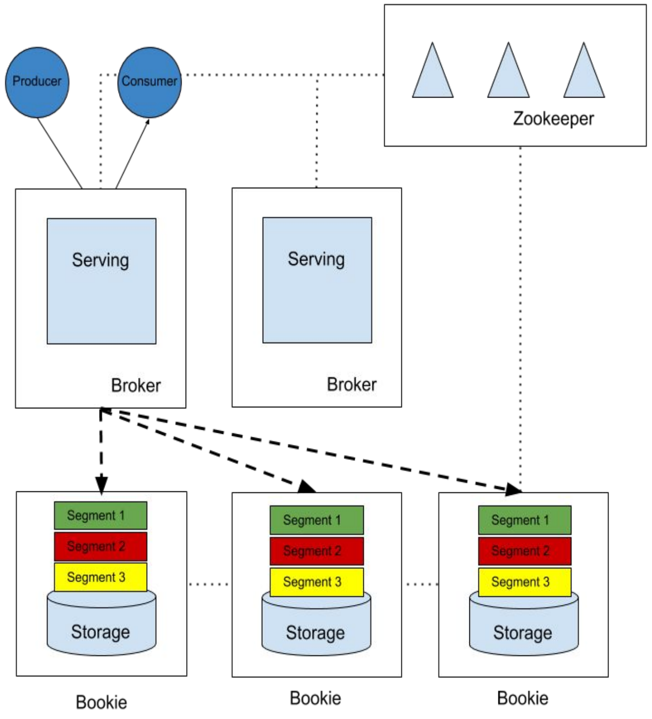
Apache Pulsar and Load Testing

Apache Pulsar

- Distributed messaging and streaming platform
  - Donated to ASF in 2016 by Yahoo!
  - Top-level Project in 2018 and very fast growing
  - Yahoo, Splunk, Overstock, Tencent, etc.
- Key Differentiators
  - Separation between Compute and Storage
  - Native Geo-replication
  - Robust multi-tenancy
  - Flexible message processing model



Source: APISeven.com



Pulsar Load Testing Tools

- pulsar-perf
  - Native piece of Pulsar; a standalone Pulsar client application
  - Built-in workload type: produce, consume, read, websocket-producer, managed-ledgers, transaction, …
  - Metrics: throughput and latency percentile
    - command line output and histogram file (HdrHistogram Plotter)
- Open Messaging Benchmark (OMB) framework
  - A Linux Foundation Collaborative project
  - A generic load test engine specifically designed for different messaging systems, including JMS, RabbitMQ, Kafka, Pulsar, and more
  - Takes a driver-worker model
    - Driver is for creating topic, producer, and consumer
    - Worker is for actual workload execution
  - Metrics: pub rate, cons rate, backlog, pub latency, delay latency
    - command line output and json file(s) (create_chart.py)
  - Out of the box only supports running on AWS and AliCloud (with terraform automation)

Pulsar Load Testing Tools, continued

Workload Modeling

pulsar-perf	● Simplified workload generation ● No workload modeling
OMB	

Workload Execution

pulsar-perf	● No scheduling capability
OMB	● Limited capability

A **common challenge** shared by many general-purpose load testing tools for different system!

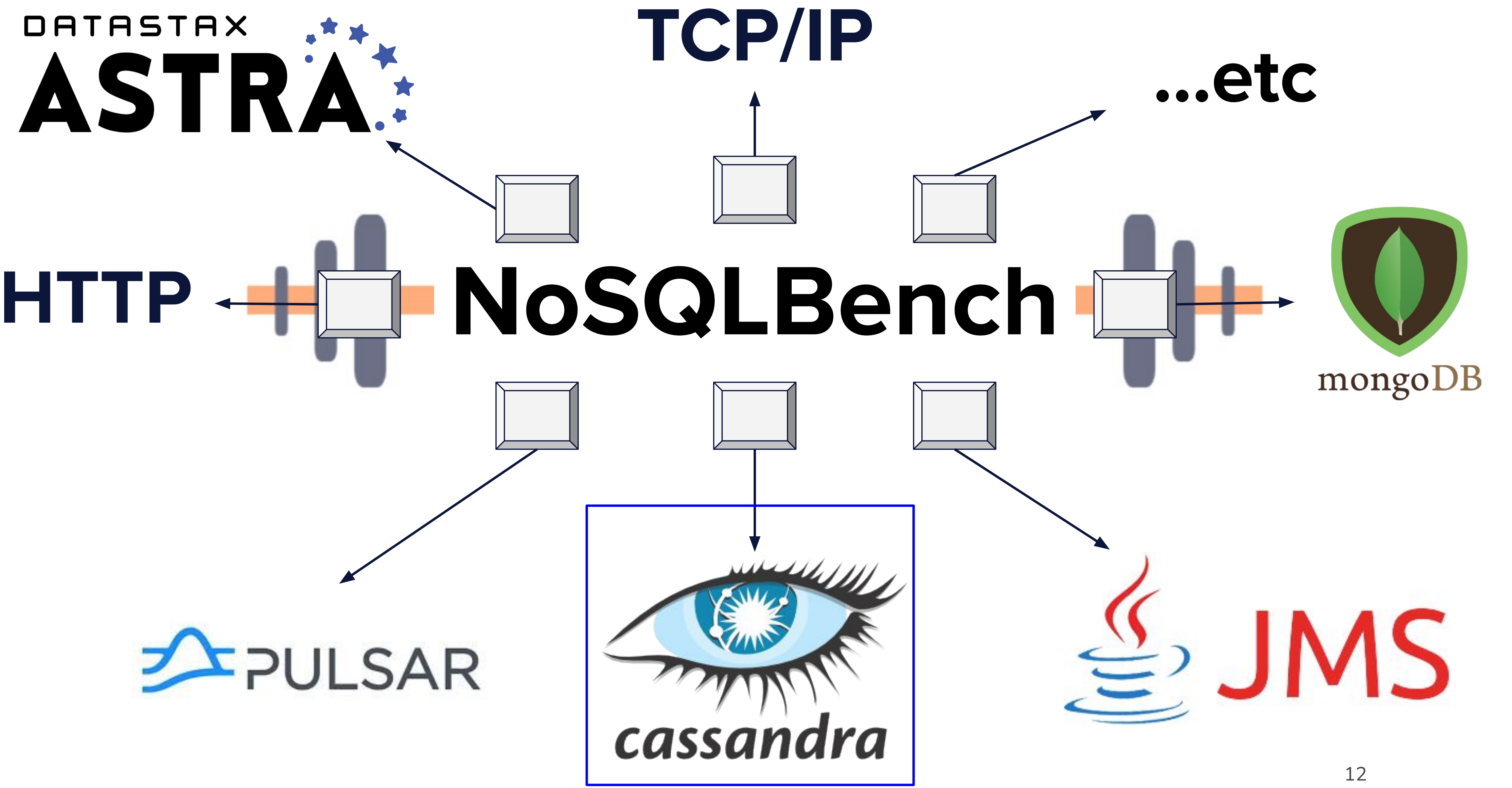
NoSQLBench

Overview

What is Nosqlbench?

NoSQLBench

"...is a workload simulation and performance testing tool for the ~~NoSQL~~ ecosystem"



NoSQLBench

DATASTAX ASTRA

TCP/IP

...etc

HTTP

mongoDB

PULSAR

cassandra

JMS

Why NoSQLBench?

Focus on business domain (data modeling)
Deterministic and repeatable testing
(Realistic Load Testing)

CAN'T YOU DO ANYTHING RIGHT?
GLASBERGEN
Copyright 2003 Randy Glasbergen. www.glasbergen.com

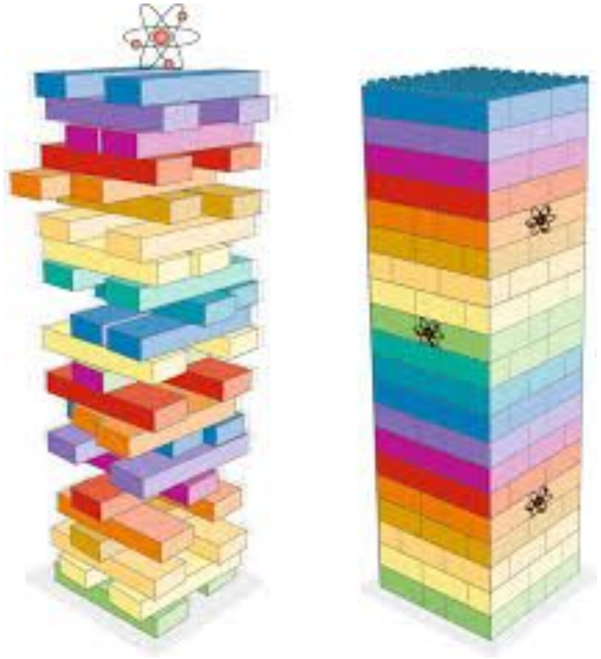
- Recipe-oriented procedural data generation
- Deterministic workload behavior
- Modular protocol (driver) support
- Configuration language for statements and data
- Scripting built-in (automation)
- Cycle-specific operations and diagnostics
- Docker-metrics dashboard
- Consistent view of high-fidelity metrics, multiple output formats and reporting options
- Support for coordinated omission

Note: credits to Jonathan Shook, the author of NosqlBench

NoSQLBench Core Concepts

- Scenario Definition (Yam File)
  - Bindings (data generation)
  - Statements (core execution)
    - Parameters
  - Blocks (of statements)
  - Tags

- Scenario Execution
  - Driver/Protocol (workload type: C*, Pulsar, etc.)
  - Cycles
  - Threads
  - Strides
  - … …



NoSQLBench C* Example - IoT Workload

```
nb run driver=cql workload=cql-iot-basic.yaml tags=phase:rampup threads=10 cycles=10M hosts=”<C*_hostname/ip>”
```

Simulate C* insert workload (10M records)

```
description: |
  This workload emulates a time-series data model and ramps up data after schema creation.
bindings:
  machine_id: Mod(10000); ToHashedUUID() -> java.util.UUID
  sensor_name: HashedLineToString('data/variable_words.txt')
  time: Mul(100); Div(10000L); ToDate()
  cell_timestamp: Mul(100L); Div(10000L); Mul(1000L)
  sensor_value: Normal(0.0,5.0); Add(100.0) -> double
  station_id: Div(10000);Mod(100); ToHashedUUID() -> java.util.UUID
  data: HashedFileExtractToString('data/lorem_ipsum_full.txt',800,1200)
blocks:
  - tags:
      phase: rampup
      statements:
        - insert-rampup: |
            insert into <<keyspace:baselines>>.<<table:iot>>
            (machine_id, sensor_name, time, sensor_value, station_id, data)
            values ({machine_id}, {sensor_name}, {time}, {sensor_value}, {station_id}, {data})
            using timestamp {cell_timestamp}
          idempotent: true
          prepared: true
          cl: LOCAL_QUORUM
```

data binding

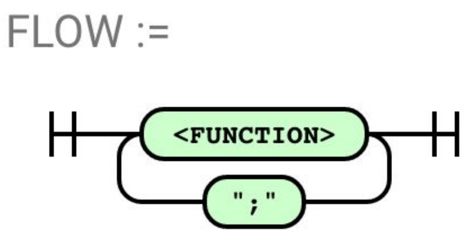
execution tag:phase

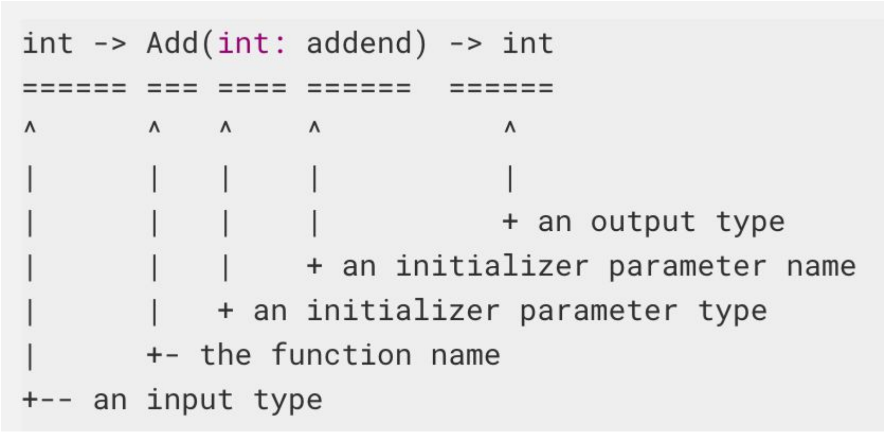
statement template

execution parameter

NoSQLBench Data Binding

- Procedural, deterministic data generation through bindings
  - http://docs.virtdata.io/
- Binding: <binding_name>:<binding_receipe>
  - e.g. **`machine_id`**`: Mod(10000); ToHashedUUID() -> java.util.UUID`
- Binding recipe: a **Function Flow**
- **Function**
  - Add(5)
  - int -> Add(5)
  - Add(5) -> int
  - int -> Add(5) -> 5
- A large set of available functions

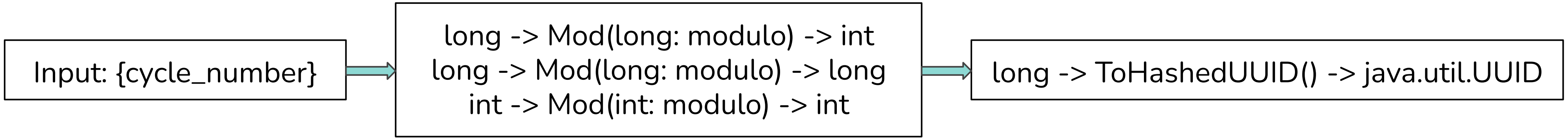




```
int -> Add(int: addend) -> int
====== === ==== ======  ======
^      ^   ^    ^          ^
|      |   |    |          |
|      |   |    |          + an output type
|      |   |    + an initializer parameter name
|      |   + an initializer parameter type
|      +- the function name
+-- an input type
```

NoSQLBench Data Binding Example

- **`machine_id`**`: Mod(5); ToHashedUUID() -> java.util.UUID`



Input: {cycle_number}

long -> Mod(long: modulo) -> int
long -> Mod(long: modulo) -> long
int -> Mod(int: modulo) -> int

long -> ToHashedUUID() -> java.util.UUID

- nb run driver=stdout workload=**mod-only.yaml** cycles=10
  - `machine_id: Mod(<<sources:5>>)`
- nb run driver=stdout workload=**mod-hash.yaml** cycles=10
  - `machine_id: Mod(<<sources:5>>); ToHashedUUID()`

```
machine_id=0
machine_id=1
machine_id=2
machine_id=3
machine_id=4
machine_id=0
machine_id=1
machine_id=2
machine_id=3
machine_id=4
```

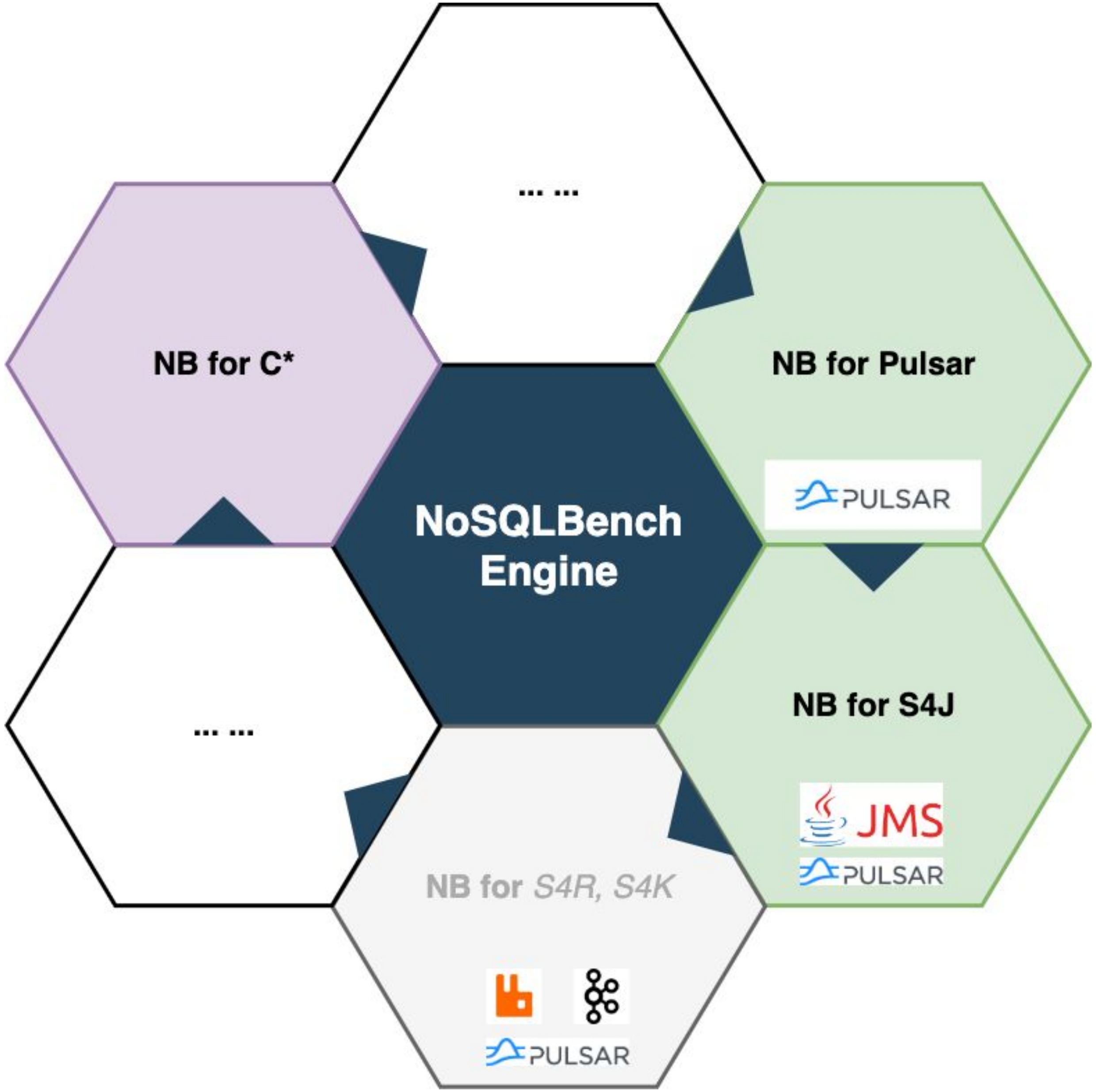
```
machine_id=28df63b7-cc57-43cb-9752-fae69d1653da
machine_id=5752fae6-9d16-43da-b20f-557a1dd5c571
machine_id=720f557a-1dd5-4571-afb2-0dd47d657943
machine_id=6fb20dd4-7d65-4943-9967-459343efafdd
machine_id=19674593-43ef-4fdd-bdf4-98b19568b584
machine_id=28df63b7-cc57-43cb-9752-fae69d1653da
machine_id=5752fae6-9d16-43da-b20f-557a1dd5c571
machine_id=720f557a-1dd5-4571-afb2-0dd47d657943
machine_id=6fb20dd4-7d65-4943-9967-459343efafdd
machine_id=19674593-43ef-4fdd-bdf4-98b19568b584
```

NoSQLBench Standard Metrics

- Metrics report options
  - --report-csv-to <dirname>
  - --report-graphite-to <addr>[:<port>]
    - --metrics-prefix <metrics-prefix>
  - --report-interval <interval-seconds> (default 10 seconds)
- HDR histogram
  - --log-histograms histodata.log
  - --log-histostats stats.csv
- Docker metrics
  - --docker-metrics
    - Dockerized Grafana and Prometheus

Field/Column Name	Meaning
t	The time that the metric sample was written
count	The running op count at that time
min, max	minimum and maximum latency (for that histogram, different for time-decaying vs discrete buckets)
mean	The mean value (since start)
stddev	one standard deviation for the current histogram
p50,p75,p95,p98,p99,p999	percentiles from the current histogram (latencies, unless otherwise specified)
mean_rate	mean rate since the start of the timer
m1_rate,m5_rate,m15_rate	one, five, fifteen minute moving average rates
rate_unit	generally "ops/second"
duration_unit	"nanoseconds"

NoSQLBench Pulsar Driver



- NB for native Pulsar client driver
- NB for Starlight for JMS API (S4J)
- NB for Starlight for RabbitMQ and Kafka (S4R and S4K)

Why NB Pulsar Driver?

Realistic Message Modeling

Complete Workload Execution Behavior Tuning

Authentic Multi-tenancy Testing

Deterministic and Repeatable Workload Execution

NB Pulsar Scn. Yaml and Workload Type

```
bindings:
 ... ...

params:
 topic_uri: "<pulsar_topic_name> (dynamic or static)"
 async_api: "[true|false]"
  … …

blocks:
 - name: <statement_block>
   tags:
     phase: <phase_identifier>
   statements:
     - name: <statement_name_1>
       optype: <statement_identifier>
       ... <statement_specific_parameters> ...
     - name: <statement_name_2>
       ... ...

 - name: <command_block_2>
   ...
```

- create/delete tenants
- create/delete namespaces
- create/delete topics
- producer
- consumer (single topic)
- reader
- consumer (multi-topic)

NB Pulsar - Realistic Messaging Modeling

```
blocks:
 - name: block1
   tags:
     phase: <phase_identifier>
   statements:
     - name: s1
       optype: msg-send
       topic_uri: "{topic_name}"
       msg_key: "{mykey}"
       msg_property: |
         {
           "site_id":  "{site_id_uuid}",
           "site_desc": "{site_desc_text}"
         }
       msg_value: |
         {
           "SensorID": "{sensor_id_uuid}",
           "SensorType": "{sensor_type}",
           "ReadingTime": "{reading_time}",
           "ReadingValue": {reading_value}
         }
       ratio: 1
```

- ☑ Complete message structure
  - Message Key, Properties, and Payload
- ☑ Message format reflecting actual business need
  - Avro and KeyValue schema support
- ☑ Multi-topics with varied message formats
  - Different tenants and/or namespaces
- ☑ Flexible mixture of message producing and consuming
- ☑ Precise workload ratio control

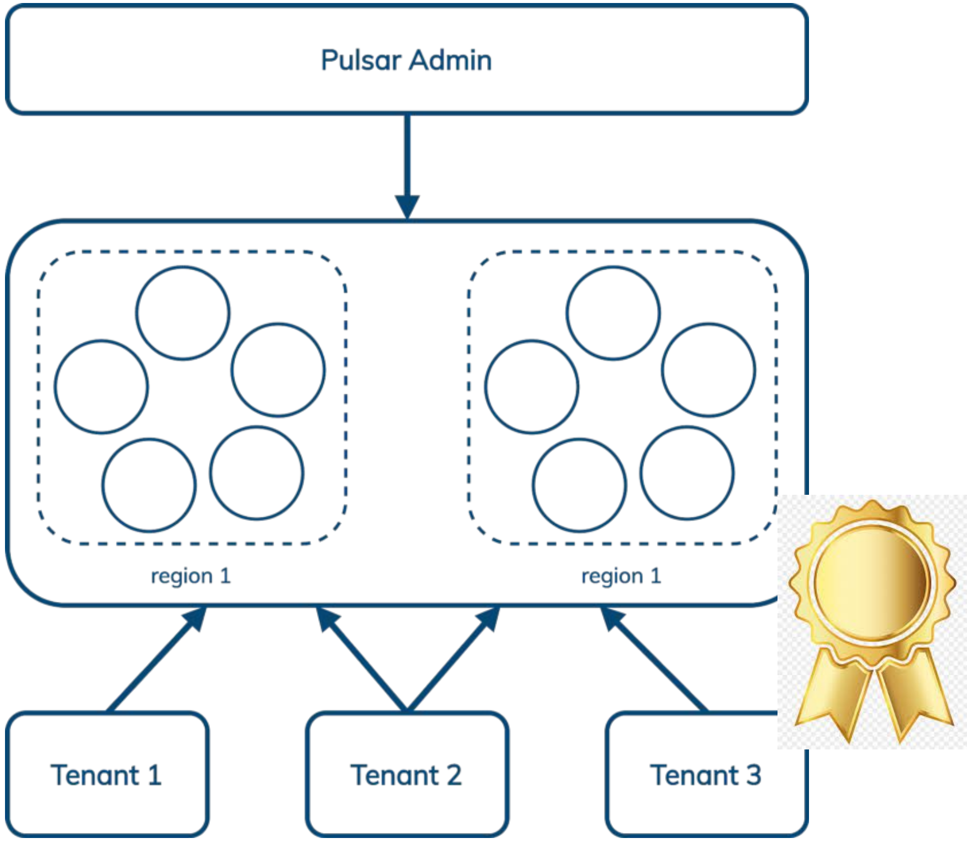
NB Pulsar - Workload Exec. Tuning (Tiered Config)

```
nb run driver=pulsar threads=10 cycles=10M web_url=http://localhost:8080 service_url=pulsar://localhost:6650
workload=</path/to/Scn.yaml> config=</path/to/config.properties>
```

```
### Schema related configurations
schema.type=
schema.definition=
...

### Pulsar client related configurations
client.connectionTimeoutMs=5000
client.authPluginClassName=
client.authParams=
...

### Producer related configurations - producer.xxx
producer.producerName=
producer.blockIfQueueFull=true
...

### Consumer related configurations - consumer.xxx
consumer.consumerName=
consumer.receiverQueueSize=
...
```

```
params:
  async_api: [true|false]
  use_transaction: [true|false]
  admin_delop: [true|false]
  seq_tracking: [true|false]
  … …

blocks:
 - name: consumer-block
   tags:
       phase: consumer
   statements:
       - name: s1
         optype: msg-consume
         subscription_name:
         subscription_type:
         consumer_name:
```

NB Pulsar - Authentic Multi-tenancy Testing

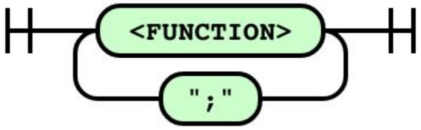


persistent://<tenant_name>/<namespace_name>/<topic_name>

- ❑ Multi-topic, multi-tenancy based testing is the **reality** !
  - ○ It is critical to understand of how multiple tenants impact each other in a single cluster and how effective resource segregation works.
- ❑ Existing Pulsar performance testing tools are pretty much single-topic oriented.

NB Pulsar - Deterministic and Repeatable Workload Execution

FLOW :=



NoSQLBench

NB Pulsar Examples - tenants, namespaces

```
# 10 tenants
bindings:
 tenant: Mod(10); ToString(); Prefix("tnt")

params:
 admin_delop: "false"

blocks:
 - name: create-tenant-block
   tags:
     phase: admin-tenant
     admin_task: true
   statements:
     - name: s1
       optype: admin-tenant
       admin_roles:
       allowed_clusters:
       tenant: "{tenant}"
```

```
bindings:
 # 20 namespaces: 10 tenants, 2 namespaces/tenant
 tenant: Mod(20); Div(2L); ToString(); Prefix("tnt")
 namespace: Mod(2); ToString(); Prefix("ns")

params:
 admin_delop: "false"

blocks:
 - name: create-namespace-block
   tags:
     phase: admin-namespace
     admin_task: true
   statements:
     - name: s1
       optype: admin-namespace
       namespace: "{tenant}/{namespace}"
```

NB Pulsar Examples - topics

```
bindings:
 # 100 topics: 10 tenants, 2 namespaces/tenant, 5 topics/namespace
 tenant: Mod(100); Div(10L); ToString(); Prefix("tnt")
 namespace: Mod(10); Div(5L); ToString(); Prefix("ns")
 core_topic_name: Mod(5); ToString(); Prefix("t")

params:
 admin_delop: "false"

blocks:
 - name: create-topic-block
   tags:
     phase: admin-topic
     admin_task: true
   statements:
     - name: s1
       optype: admin-topic
       topic_uri: "persistent://{tenant}/{namespace}/{core_topic_name}"
       enable_partition: "true"
       partition_num: "10"
```

NB Pulsar Examples - producers and consumers

Basic producer

```
statements:
  - name: producer-basic
    optype: msg-send
    topic_uri: "{topic_name_1}"
    msg_key:
    msg_property:
    msg_value: "{msg_value}"
    producer_name:
```

Avro producer

```
statements:
  - name: producer-avro
    optype: msg-send
    topic_uri: "{topic_name_2}"
    msg_key: "{msg_key}"
    msg_property: |
     {
        ……
     }
    msg_value: |
     {
        ……
     }
    producer_name:
```

KeyValue with Avro is also supported

```
schema.type=avro
schema.definition=/path/to/avr
o/definition/file
```

Single topic consumer

```
statements:
  - name: consumer-single
    optype: msg-consume
    topic_uri: "{topic_name_1}"
    subscription_name: "{sub_name}"
    subscription_type:
    consumer_name:
```

Multi topic consumer

```
statements:
 - name: s1
   optype: msg-mt-consume
   topic_names: "{topic_name_list}"
   topics_pattern: "{topic_name_pattern}"
   subscription_name: "{sub_name}"
   subscription_type:
   consumer_name:
```

NB Pulsar Examples - topics

```
nb run driver=pulsar threads=1 cycles=10M \
   web_url=http://localhost:8080 \
   service_url=pulsar://localhost:6650 \
   workload=</path/to/Scn.yaml> \
   config=</path/to/config.properties> \
   cyclerate_per_thread=true cyclerate=1K \
   --progress console:10s -v \
   --report-graphite-to <graphite_server_ip>:9109 \
   --report-csv-to metrics
```

NB Pulsar Examples - Producer Execution Output

```
… …
2022-10-02 03:45:28,141 [pulsar-client-io-5-1] INFO : [[id: 0x401df668, L:/172.31.4.34:62285 - R:/10.166.90.105:6650]] Connected
to server
2022-10-02 03:45:28,686 [pulsar-client-io-5-1] INFO : Starting Pulsar producer perf with config: {
 "topicName" : " persistent://public/default/tp_large",
 "producerName" : null,
……
2022-10-02 03:45:29,730 [pulsar-client-io-5-1] INFO : [persistent://public/default/tp_large] [MyCluster1-1-26]    Created producer
on cnx [id: 0x9af6440d, L:/172.31.4.34:62286 - R:ip-10-166-90-105.us-west-2.compute.internal/10.166.90.105:6650]
… …
2022-10-02 03:45:30,028 [pulsar-client-io-5-1] INFO : Starting Pulsar producer perf with config: {
 "topicName" : " persistent://public/default/tp_small",
… ….
2022-10-02 03:45:30,449 [pulsar-client-io-5-1] INFO : [persistent://public/default/tp_small] [MyCluster1-1-28]    Created producer
on cnx [id: 0x9af6440d, L:/172.31.4.34:62286 - R:ip-10-166-90-105.us-west-2.compute.internal/10.166.90.105:6650]
… …
2022-10-02 03:47:24,860 [ProgressIndicator/logonly:10s] INFO :
/Users/yabinmeng/MyFolder/Yabin.Work/PSA.Vanguard/Conference/ApacheCon/NA2022/yaml/producer.yaml:    18.43%/Running (details: min=0
cycle=921680 max=5000000)
2022-10-02 03:47:28,717 [pulsar-timer-8-1] INFO : [persistent://public/default/tp_large] [MyCluster1-1-22] Pending messages: 67
--- Publish throughput: 2442.23 msg/s --- 26.68 Mbit/s --- Latency: med: 104.000 ms - 95pct: 184.000 ms - 99pct: 272.000 ms -
99.9pct: 325.000 ms - max: 392.000 ms --- Ack received rate: 2444.81 ack/s --- Failed messages: 0
2022-10-02 03:47:34,860 [ProgressIndicator/logonly:10s] INFO :
/Users/yabinmeng/MyFolder/Yabin.Work/PSA.Vanguard/Conference/ApacheCon/NA2022/yaml/producer.yaml:    20.16%/Running (details: min=0
cycle=1007900 max=5000000)
… …
2022-10-02 03:55:50,468 [pulsar-client-io-5-1] INFO : [persistent://public/default/tp_large] [MyCluster1-1-26]    Closed Producer
2022-10-02 03:55:50,468 [pulsar-client-io-5-1] INFO : [persistent://public/default/tp_small] [MyCluster1-1-28]    Closed Producer
… …
2022-10-02 03:55:52,560 [scenarios:001] INFO : scenario state: Finished
10/1/22, 10:55:52 PM ============================================================

-- Gauges ----------------------------------------------------------------------
… …

-- Histograms ------------------------------------------------------------------
… …

-- Timers ----------------------------------------------------------------------
… ….

2022-10-02 03:55:52,768 [main] INFO : executions: 1 scenarios, 1 normal, 0 errored
```

Initializing connection and producers

Real time progress report and message processing feedback

Closing connection and producers

Execution metrics high level summary

2 producers, 1 large payload and 1 small payload with ratio of 3:7

NB Pulsar Examples - Driver Metrics

- **Producer**/Consumer/Reader Gauges, e.g. for producer
  - Sent message count rate
  - Sent message bytes rate
  - Send failed
- Error detection Counters
  - Message duplication
  - Message loss
  - Message out-of-sequence
- Histograms
  - End-to-end message latency

NoSQLBench Demo

NB Pulsar Demo - Overview

- A small Pulsar cluster (K8s deployment) running on GKE
  - 3 zookeepers
  - 3 brokers
  - 3 bookkeepers
  - Other supporting components
- Two NB Pulsar testing client processes running locally
  - 1 producer
  - 1 consumer
- Message Model
  - 256 bytes text payload
  - 1 message property
  - No message key

Resources

- Project and Document Home page
  - Source code: https://github.com/nosqlbench/nosqlbench
  - Document: http://docs.nosqlbench.io/#/docs/
- NB Pulsar driver (Main/NB5 branch)
  - https://github.com/nosqlbench/nosqlbench/tree/main/driver-pulsar
- NB S4J driver (NB4 branch)
  - https://github.com/nosqlbench/nosqlbench/tree/nb4-maintenance/driver-s4j [**Doc**]
- Download
  - https://github.com/nosqlbench/nosqlbench/releases
- Online workshop (DataStax Academy)
  - DataStax Academy Github
    https://github.com/DataStax-Academy/nosqlbench-workshop-online

DATASTAX®

Thank you