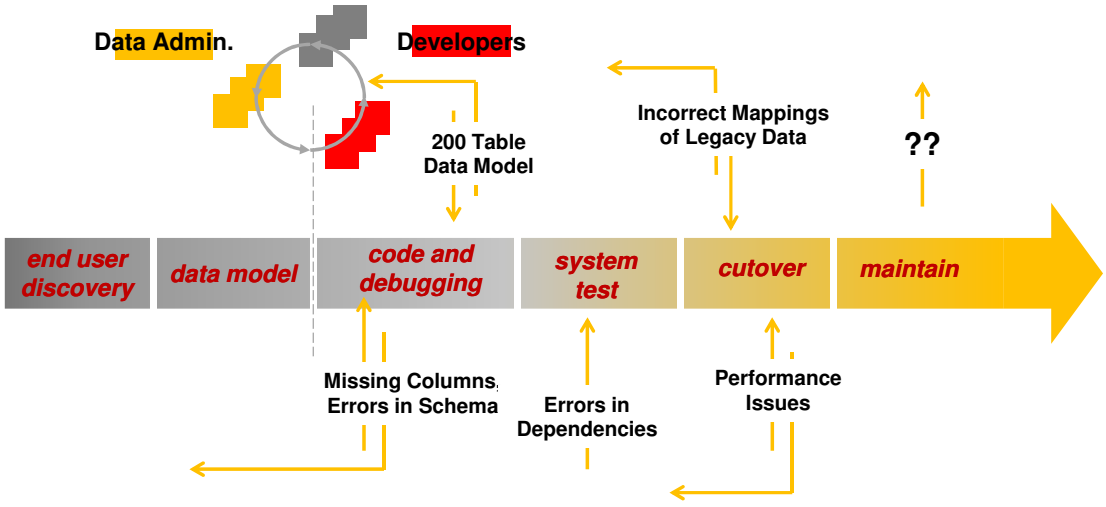
NoSQLBench: Why ?

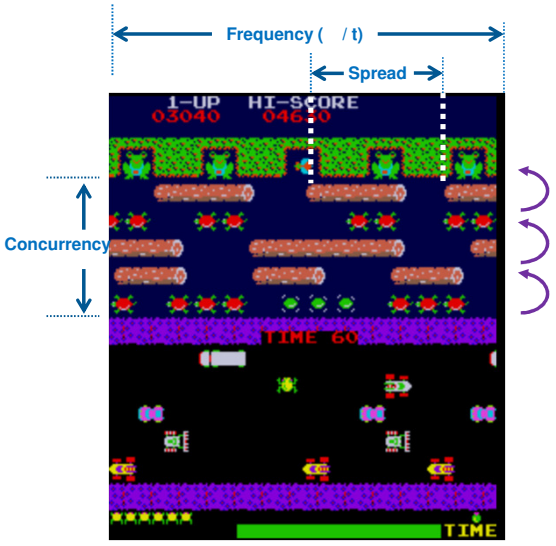
How/How-well do you answer these-

- **Can I add X% more users ?**
- **Can I add this new application module ?**
- **Can I handle the expected/ increased seasonal service load/spike ?**
- **Can I downsize this system for cost ?**
- **What will happen to customer satisfaction if I do (YYY) while on line, or on peak ?**
- **Routine-Z is now slow; Did it grow slow over months, or overnight ?**
  (2 different problems/solutions)
- **How can I improve application development velocity ?**

Legacy Application Development Lifecycle



Creating a "Query Harness"-



Every statement; (SEL, INS/UPD, DEL, other)

- Frequency
- Concurrency
- Spread (aka, Density)

Input

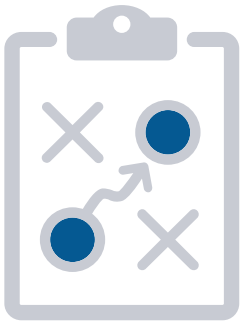
- Necessary SLA/SLO

Output/Measured

DATASTAX

NoSQLBench, Data-

Drastically reduce the time and effort to get sophisticated answers to simple questions-



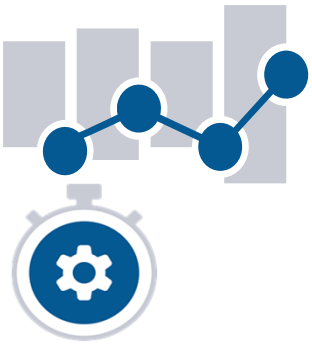
- Generated
  - Less operational overhead (on-disk storage systems normally the limiting factor when performance testing)
  - Nothing to manage
- Deterministic, statistically shaped, realistic data
  - Example: First|Last|Full name binding based on USA survey data, Daniel more common than Melvin
  - (Same generated data on every execution)
  - 80+ built in bindings, each fully customizable
  - Built-in/custom weighting, skew
- Out of the box workloads for,
  - IOT, key/value, wide-column, UUID, other
  - Or, define your own
- Drivers for,
  - 11 systems out of the box; CQL, Http, Kafka, ..
  - Or, add your own
- Built for distributed systems
  - Partition aware operators

DATASTAX

NoSQLBench, (Service Layer)-

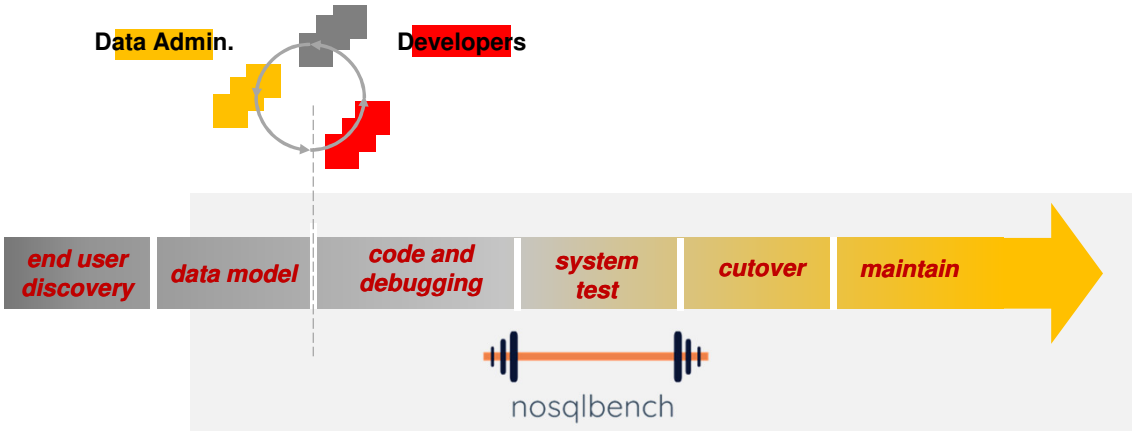
- **No programming-**
  - Single YAML file (a *Workload*), 3 or fewer common sections
  - (You can optionally design in JavaScript for very advanced needs)
- **Workload can be used variably, different; *Scenarios*, scale, duration, targeting different/ multiple nodes**
- **Sync/Async, blocking/non-blocking**
  - Extremely/simply configured
  - Extensible
- **Battle tested; it's what DataStax uses internally for years, and at large scale (1 to hundreds of nodes)**
- **Built in Prometheus/ Grafana container GUI for high fidelity metrics, or 3 other routes to analyze**
- ***Easy to emulate your application (minutes of work)***

Drastically reduce the time and effort to get sophisticated answers to simple questions-



DATASTAX

Where/When to use NoSQLBench-



DATASTAX

Hands on Lab: Using a Built-in Workload,
(Install verify, Hello World)

DATASTAX

Hands on Lab: Overview (25 minutes or less to complete)

Pre-requisites-
- **Linux system**
- **C*/DSE, or Pre-instantiated Astra instance**

Here/now:
- **Several simple LABS; *Learn-by-example***
- **Get the 100-level; object hierarchy, use**
- **Which is faster; Primary Key, or SAI ?**

Post-requisites- (not here/now, not formally)
- **Dive into the GUI (Prometheus/Grafana)**
- **More database nodes; distributed Scenarios**

DATASTAX

Step 01: Install, install-verify, (high level look)

docs.nosqlbench.io
https://github.com/nosqlbench/nosqlbench/blob/main/DOWNLOADS.md

- Get the binary (nb)
- chmod 755 nb

- From Linux Shell-

- nb --list-drivers # That's two hyphens
- nb --list-workloads
- nb --list-scenarios

- nb --copy cql-keyvalue # Overview / read this file

  - scenarios # Optional, used to variably (configure) a workload
  - bindings # the data generators
  - blocks # the statements/activities proper, (these are just names, not keywords)

    - schema # Generally DDL, run once, serially
    - rampup # Generally data creation, parallel
    - main # The Query Harness proper

DATASTAX

Step 02a: Copy, then edit, cql-iot.yaml, run (local, C*/DSE)

This step requires a C* or DSE operating locally and without authentication (without a password)

- cp cql-keyvalue.yaml my_file.yaml
- vi my_file.yaml

  - Under, scenarios → default # default is a keyword
  - Then both, rampup and main # not keywords, yes convention
  - Change the number (10 million) to just (10)

  - Exit and save

- From the command line, nb my_file

- Enter CQLSH, look for
  - A Keyspace titled, baselines
  - A Table titled, keyvalue
  - Count the rows-

DATASTAX

Step 02b: Copy, then edit, cql-iot.yaml, run against Astra

This step requires an Astra account, and a download of your "secure connection bundle" Zip file

This Zip file contains your Keyspace/Database name, fyi, and that Database name is auto-generated into the file name. Below, ours is "my_database".

- **cp cql-keyvalue.yaml my_file.yaml**
- **vi my_file.yaml**
  - **Under, scenarios → astra**
  - **Then both, rampup and main**
  - **Change the number (10 million) to just (10)**
  - **Exit and save**
- **From the command line,**
  nb my_file astra secureconnnectbundle=secure-connect-my-database.zip
- **From the Astra Home Page, enter CQLSH (or same basic steps, different means), look for**
  - **A keyspace titled, (your keyspace name)**
  - **A table titled, keyvalue**
  - **Count the rows-**

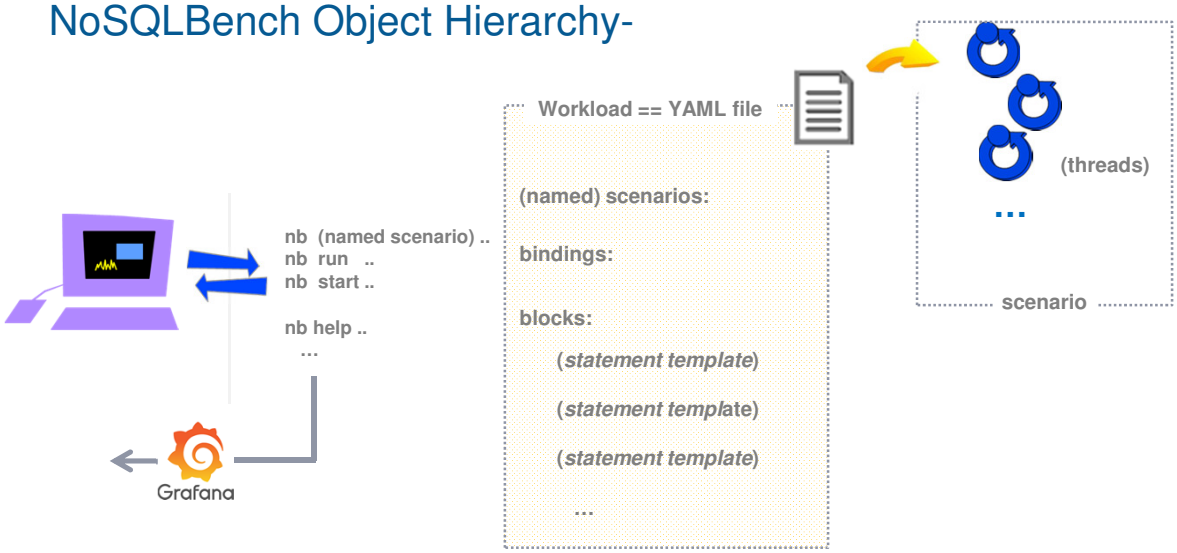
DATASTAX

Step 02: continued, ***Optional***

- **Using CQLSH, DROP the table**
  USE *keyspace_name* ; // Use correct value
  DROP TABLE keyvalue;
- **Add this clause to your "nb" command line,**
  tags:phase=schema
  tags:phase=schema-astra # if Astra
- **Look around (check tables and data)**
- **Add one row of data,**
  INSERT INTO keyvalue (key, value)
  VALUES ('xxx', 'yyy');
- **Look around**
- **Add this clause to your "nb" command line,**
  tags:phase=rampup
- **Look around**

DATASTAX

NoSQLBench Object Hierarchy-



DATASTAX

NoSQLBench Object Hierarchy: Scenarios

```
description: A workload with only text keys and text values

scenarios:
  default:
    schema: run driver=cql tags==phase:schema threads==1 cycles==UNDEF
    rampup: run driver=cql tags==phase:rampup cycles===TEMPLATE(rampup-cycles,10) threads=auto
    main: run driver=cql tags==phase:main cycles===TEMPLATE(main-cycles,10) threads=auto
  astra:
    schema: run driver=cql tags==phase:schema-astra threads==1 cycles==UNDEF
    rampup: run driver=cql tags==phase:rampup cycles===TEMPLATE(rampup-cycles,10) threads=auto
    main: run driver=cql tags==phase:main cycles===TEMPLATE(main-cycles,10) threads=auto
```

- (Named) scenarios
- 'default'
- run|start
- driver
- 1, 2, or 3 equal signs
- tags
- threads
  - 1
  - (Default, auto)
- cycles
  - UNDEF
  - 10 aka, 0..10
- (others, not shown)

DATASTAX

NoSQLBench Object Hierarchy: Bindings

- Many dozens to choose from
  - Different (types); deterministic, random, other
  - name : recipe
- Top level, also lower scope
- <<label:default>>
  - Not just for bindings, available elsewhere
  - Override passed on command line
- Casting too, E.g., .. -> int

```
bindings:
  seq_key: Mod(<<keycount:1000000000>>); ToString() -> String
  seq_value: Hash(); Mod(<<valuecount:1000000000>>); ToString() -> String
  rw_key: <<keydist:Uniform(0,1000000000)->int>>; ToString() -> String
  rw_value: Hash(); <<valdist:Uniform(0,1000000000)->int>>; ToString() -> String
```


DATASTAX

NoSQLBench Object Hierarchy: Statement (Templates)

```
- name: schema-astra
  tags:
    phase: schema-astra
  params:
    prepared: false
  statements:
  - create-table: |
      create table if not exists <<keyspace:baselines>>.<<table:keyvalue>> (
      key text,
       value text,
       PRIMARY KEY (key)
      );
    tags:
      name: create-table-astra
- name: rampup
  tags:
    phase: rampup
  params:
    cl: <<write_cl:LOCAL_QUORUM>>
  statements:
    - rampup-insert: |
        insert into <<keyspace:baselines>>.<<table:keyvalue>>
        (key, value)
        values ({seq_key},{seq_value});
      tags:
        name: rampup-insert
```

- Can have multiple statements per phase
- CQL DDL
  - Not prepared
  - **threads==1**
  - **cycles == UNDEF**
- Tags/name, execute subset
- **driver=stdout**
- <<write_cl: .. >>
- {seq_key}

DATASTAX

Step 03: Checkpoint

- ***Why*** **is it a better idea to develop against data dense nodes ?**
- **400 concurrent users execute a given routine with 3 SELECTS and 1 INSERT every 60 seconds, with a 20 second think-time/pause-**
  - **What are the Frequency, Concurrency, and Spread ?**
  - **What is the target SLA/SLO ?**
- **In NoSQLBench parlance, the YAML file is termed a _____ ?**
- **What are the 3 most common sections in that YAML ? Describe each ?**
- **What's the difference between; =, ==, === ?**
- **What does a << k : v >> represent ?**
- **What does a { bb } represent ?**

DATASTAX

Hands on Lab: scenarios, schema phase

DATASTAX

Step 04: Design/Run your own Workload (your own YAML file)

- **Assuming you know CQL, 98% of your errors will be indentation in the YAML- Refer-to/copy from our earlier YAML**
- **driver=stdout override later to CQL**
- **schema phase**
  - **Drop a keyspace**
  - **Create a keyspace**
  - **Create a table; 10 columns (really, we'll use 2)**
    - **Single column PK, type TEXT**
    - **Addition column, type TEXT, will contain same value as the PK**
  - **CREATE an SAI index**

Not conclusive because single node, low power nodes, but;

How close is a Primary Key lookup to a secondary index lookup ??

Step 04: Design/Run our own Workload, sample CQL


```
DROP KEYSPACE IF EXISTS ks_44;
CREATE KEYSPACE ks_44
  WITH replication = {'class': 'SimpleStrategy',
    'replication_factor': '1'};

CREATE TABLE ks_44.t1
    (
    col1    TEXT PRIMARY KEY,
    col4    TEXT
    );
CREATE CUSTOM INDEX col4_idx
    ON ks_44.t1 (col4) USING 'StorageAttachedIndex';

INSERT INTO ks_44.t1 (col1, col4) VALUES ('eee', 'fff');
SELECT * FROM ks_44.t1 WHERE col4 = 'fff';
```

C*/DSE only

Step 04: Get the schema phase working

As stated:

- **Get the schema phase working; keyspace, table, index**
- **driver=stdout then, driver=cql**
- **Errors with stdout, it's your YAML indents**
- **Errors with cql, copy and paste into CQLSH, debug that way**
- **Cheating; answer on next few pages**

DATASTAX

Step 05: Checkpoint, answer so far-

```
scenarios:
  default:
    schema: run driver=stdout tags==phase:schema threads==1 cycles==UNDEF
    #  more stuff goes here later

bindings:
  #  stuff will go here later

blocks:

  - tags:
      phase: schema
    params:
      prepared: false
    statements:
```

DATASTAX

Step 05: Checkpoint, answer so far-

```
- drop_keyspace: |
    DROP KEYSPACE IF EXISTS <<keyspace:ks_44>>;
  tags:
    name: drop_keyspace

- create_keyspace: |
    CREATE KEYSPACE <<keyspace:ks_44>>
      WITH replication = {'class': 'SimpleStrategy',
        'replication_factor': '1'};
  tags:
    name: create_keyspace
```

DATASTAX

Step 05: Checkpoint, answer so far-

```
- create_table: |
    CREATE TABLE <<keyspace:ks_44>>.<<table:t1>>
      (
      col1    TEXT PRIMARY KEY,
      col2    TEXT,
      col3    TEXT,
      col4    TEXT,
      col5    TEXT,
      col6    TEXT,
      col7    TEXT,
      col8    TEXT,
      col9    TEXT,
      col0    TEXT
      );
  tags:
    name: create_table
```

DATASTAX

Step 05: Checkpoint, answer so far-

```
- create_index: |
    CREATE CUSTOM INDEX col4_idx
      ON <<keyspace:ks_44>>.<<table:t1>> (col4) USING 'StorageAttachedIndex';
  tags:
    name: create_index
```

DATASTAX

Step 05: Checkpoint, answer so far-

To execute on the command line,

nb (file_name)
nb (file_name) driver=cql
nb (file_name) driver=cql secureconnectionbundle= …

From the answer provided so far; What else needs to change for Astra ?

DATASTAX

Hands on Lab: rampup phase

DATASTAX

Step 06: Add rampup phase

- **10 rows only until absolute end of exercise, until you are positive everything is perfect**
  - **Why ? Even those DROP KEYSPACE statements are not (entirely free)**
- **The primary key column and col4 should use the same binding (receive the same generated value)**
- **Experiment with other bindings for the other columns**
- ***In addition to more statements, you need to add to scenarios, and bindings***

DATASTAX

Step 06: Aids/snippets (these are not complete)

```
nb run workload=(file_name) driver=stdout tags=phase:rampup cycles=10


bindings:
 #  there are much easier solutions to the first line below; we prefer this one
 colx: Mod(<<uuidCount:100000000>>L); ToHashedUUID() -> java.util.UUID; ToString() -> String

 col8: FullNames() -> String



- tags:
    phase: rampup
  params:
    prepared: true
  statements:

    - insert: |
        INSERT INTO <<keyspace:ks_44>>.<<table:t1>> (col1, col4, col8)
          VALUES ( {colx}, {colx}, {col8} );
      tags:
        name: insert
```


Step 07: Checkpoint

- Using CQLSH, check for 10 rows in your table
- Could FullNames() have been used as the PK ?
- What did the new/sample "nb" command line invocation do differently ?

Hands on Lab: free Astra tier specific notes

DATASTAX

Notes specific to the free Astra tier

- This tier is free, *and* tiny
- INSERTing 100 rows/sec is super safe, higher than that and ..
- You'll likely need to add,

```
nb run workload=(file_name) driver=stdout tags=phase:rampup cycles=10 cyclerate=1
```

Above calls to run only one statement per second (for test)
You can do 100 on the Astra free tier, maybe as high as 600

DATASTAX

Hands on Lab: main phase

DATASTAX

Step 08: Design/Run our own Workload, main phase

- Still .. 10 rows only, until you're certain it's perfect
- Yes, any test under (n) million rows may have accuracy concerns

- SELECT using the PK
- SELECT using an SAI indexed column (col4)
- View results

DATASTAX

Step 08: Aids/snippets (these are not complete)

nb run workload=(file_name) driver=stdout tags=phase:main cycles=10

nb run workload=sol3 driver=cql tags=name:query1 cycles=10

(No new bindings needed, *although we could even further randomize the lookup key*)

```
- tags:
    phase: main
  params:
    prepared: true
  statements:

    - query1: |
        SELECT * FROM <<keyspace:ks_44>>.<<table:t1>> WHERE col1 = {colx} ;
      tags:
        name: query1

    - query1: |
        SELECT * FROM <<keyspace:ks_44>>.<<table:t1>> WHERE col4 = {colx} ;
      tags:
        name: query2
```


DATASTAX

Step 08: Aids/snippets (these are not complete)

- Several paths to view results. One is to view the 'logs', created by default, below the current working directory
  - rm –r logs
  - (Re-run just 'query1', then 'query2' to driver=cql
  - cd logs
  - vi scenario*.log # this file name is output when you run "nb"
  - Less than 100 lines, probably; see next page
  - And Yes, there's also Grafana, we'll *introduce* that shortly

DATASTAX

Step 08: Viewing the (scenario results) log file

- "Results" is its own whole set of topics-
  - Start here: http://docs.nosqlbench.io/#/docs/getting_started/03_reading_metrics
- Lots to look at, but .. let's look at,

2020-08-12 13:44:48,293 INFO [main] i.n.e.c.ScenarioResult [Slf4jReporter.java:374] type=TIMER, name=sol3.result-success, count=10, min=1046.496, max=5771.263, mean=2666.6912, stddev=1445.067675240492, median=2332.415, p75=3398.399, p95=5771.263, p98=5771.263, p99=5771.263, p999=5771.263, mean_rate=18.600062259244396, m1=0.0, m5=0.0, m15=0.0, rate_unit=events/second, duration_unit=microseconds

2020-08-12 13:44:47,957 INFO [main] i.n.e.c.ScenarioResult [Slf4jReporter.java:374] type=HISTOGRAM, name=sol3.resultset-size, count=10, min=0, max=1, mean=0.1, stddev=0.30000000000000004, median=0.0, p75=0.0, p95=1.0, p98=1.0, p99=1.0, p999=1.0

DATASTAX

Step 08: last page was 10 rows, the PK lookup, now SAI

name=sol3.result-success, count=10, min=1046.496, max=5771.263, mean=2666.6912
name=sol3.resultset-size, count=10, min=0, max=1

name=sol3.result-success, count=10, min=1222.784, max=5263.615, mean=2126.1504
name=sol3.resultset-size, count=10, min=0, max=1

What should we change ?

DATASTAX

Step 08: 10 rows, then 10M rows

10 Rows, 10 queries

name=sol3.result-success, count=10, min=1046.496, max=5771.263, mean=2666.6912 — PK lookup
name=sol3.resultset-size, count=10, min=0, max=1
name=sol3.result-success, count=10, min=1222.784, max=5263.615, mean=2126.1504 — SAI lookup
name=sol3.resultset-size, count=10, min=0, max=1

---

name=sol3.result-success, count=10000, min=123.524, max=10125.311, mean=458.1033824 — PK lookup
name=sol3.resultset-size, count=10000, min=1, max=1

name=sol3.result-success, count=10000, min=269.552, max=78245.887, mean=1546.370848 — SAI lookup
name=sol3.resultset-size, count=10000, min=1, max=1

10M Rows, 1000 queries

DATASTAX

Step 09: If time allows

- Prometheus/Grafana
  - Add to command line first time, --docker-metrics

    Different switch on subsequent use

DATASTAX

Further Study (next steps):

- More on the process model-
  - run|start Blocking, non-blocking
  - Sync/Async
  - More control over the (statements)
  - Multiple "nb" hosts (simple, but not covered..)
- More on parameters
- More on bindings; which are deterministic, random
- More on the object hierarchy
- Scripting
- More on results; client side (nb), server side (not nb)
- Basically .. .. a lot more

DATASTAX®

Thank You!

Final answer: Complete solution

```
#  Run via,
#
#      nb  (file_name)
#      nb  (file_name) driver=cql
#
#      nb run workload=(file_name) driver=stdout tags=phase:rampup cycles=10
#
#      nb run workload=(file_name) driver=stdout tags=name:query1 cycles=10

#  Because of the DROP/CREATE KEYSPACE, this file does not run against Astra


scenarios:
  default:
    schema: run driver=stdout tags==phase:schema threads==1 cycles==UNDEF
    rampup: run driver=stdout tags==phase:rampup threads==auto cycles=10000000
    main: run driver=stdout tags==phase:main threads==auto cycles=100000


bindings:
  col x: Mod(<<uuidCount:100000000>>L); ToHashedUUID() -> java.util.UUID; ToString() -> String
  col8: FullNames() -> String
```

DATASTAX

Final answer: Complete solution

```
blocks:

  - tags:
      phase: schema
    params:
      prepared: false
    statements:

      - drop_keyspace: |
          DROP KEYSPACE IF EXISTS <<keyspace:ks_44>>;
        tags:
          name: drop_keyspace

      - create_keyspace: |
          CREATE KEYSPACE <<keyspace:ks_44>>
            WITH replication = {'class': 'SimpleStrategy',
              'replication_factor': '1'};
        tags:
          name: create_keyspace
```

DATASTAX

Final answer: Complete solution

```
- create_table: |
        CREATE TABLE <<keyspace:ks_44>>.<<table:t1>>
          (
          col1    TEXT PRIMARY KEY,
          col2    TEXT,
          col3    TEXT,
          col4    TEXT,
          col5    TEXT,
          col6    TEXT,
          col7    TEXT,
          col8    TEXT,
          col9    TEXT,
          col0    TEXT
          );
      tags:
        name: create_table

    - create_index: |
        CREATE CUSTOM INDEX col4_idx
          ON <<keyspace:ks_44>>.<<table:t1>> (col4) USING 'StorageAttachedIndex';
      tags:
        name: create_index
```

DATASTAX

Final answer: Complete solution

```
- tags:
    phase: rampup
  params:
    prepared: true
  statements:

    - insert: |
        INSERT INTO <<keyspace:ks_44>>.<<table:t1>> (col1, col4, col8)
          VALUES ( {colx}, {colx}, {col8} );
      tags:
        name: insert
```

DATASTAX

Final answer: Complete solution

```
- tags:
    phase: main
  params:
    prepared: true
  statements:

    - query1: |
        SELECT * FROM <<keyspace:ks_44>>.<<table:t1>> WHERE col1 = {colx} ;
      tags:
        name: query1

    - query1: |
        SELECT * FROM <<keyspace:ks_44>>.<<table:t1>> WHERE col4 = {colx} ;
      tags:
        name: query2
```