JANOS SZENDI-VARGA

CHAOS ENGINEERING WITH NEO4J

ABOUT ME

- I am an IT engineer
- I live in Budapest, Hungary
- I work for Graph Coding
- Graph Technology Landscape 2019
- I have been in the Neo4j community since 2013
- I am the main organiser of the Neo4j Budapest Meetup Group
- @szenyo
- janos@graphcoding.com

WHAT IS CHAOS ENGINEERING?

CHAOS ENGINEERING WITH NEO4J

HISTORY

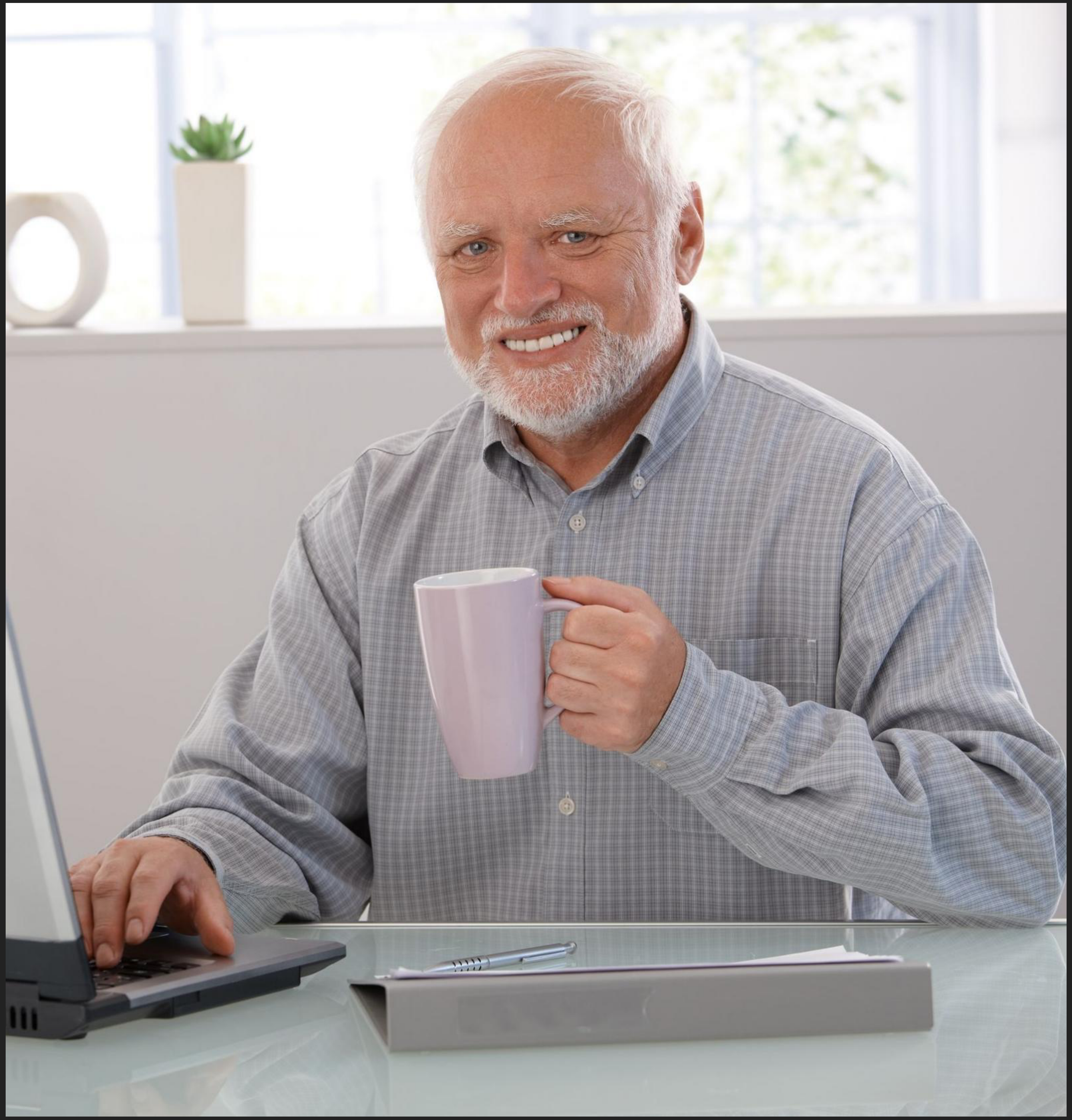
- Jessie Robbins https://en.wikipedia.org/wiki/Jesse_Robbins
- “Master of Disaster”
- GameDay: increase reliability by purposefully creating major failures on a regular basis
- Adopted by the large organisations
- “Netflix does it and thinks you should too”
- 2016: Principles of Chaos Engineering http://principlesofchaos.org
- 2017: Chaos Engineering book from O’Reilly byCasey Rosenthal, Lorin Hochstein, Aaron Blohowiak, Nora Jones, Ali Basiri

EXPERIMENTS STEPS (PRINCIPLESOFCHAOS.ORG):

1. Define a ‘steady state’ as some measurable output of a system that indicates normal behaviour.
2. Hypothesise that this steady state will continue in both the control group and the experimental group.
3. Introduce variables that reflect real world events like servers that crash, hard drives that malfunction, network connections that are severed, etc.
4. Try to disprove the hypothesis by looking for a difference in steady state between the control group and the experimental group.

EXECUTION ORDER

- Known Knowns - Things you are aware of and understand
- Known Unknowns - Things you are aware of but don't fully understand
- Unknown Knowns - Things you understand but are not aware of
- Unknown Unknowns - Things you are neither aware of nor fully understand



OMTM, ONE METRIC THAT MATTERS - MEASURING THE IMPACT

- Typical use case of Neo4j:
  - recommendation system
  - fraud detection
  - identity management
  - master data management

MONITORING

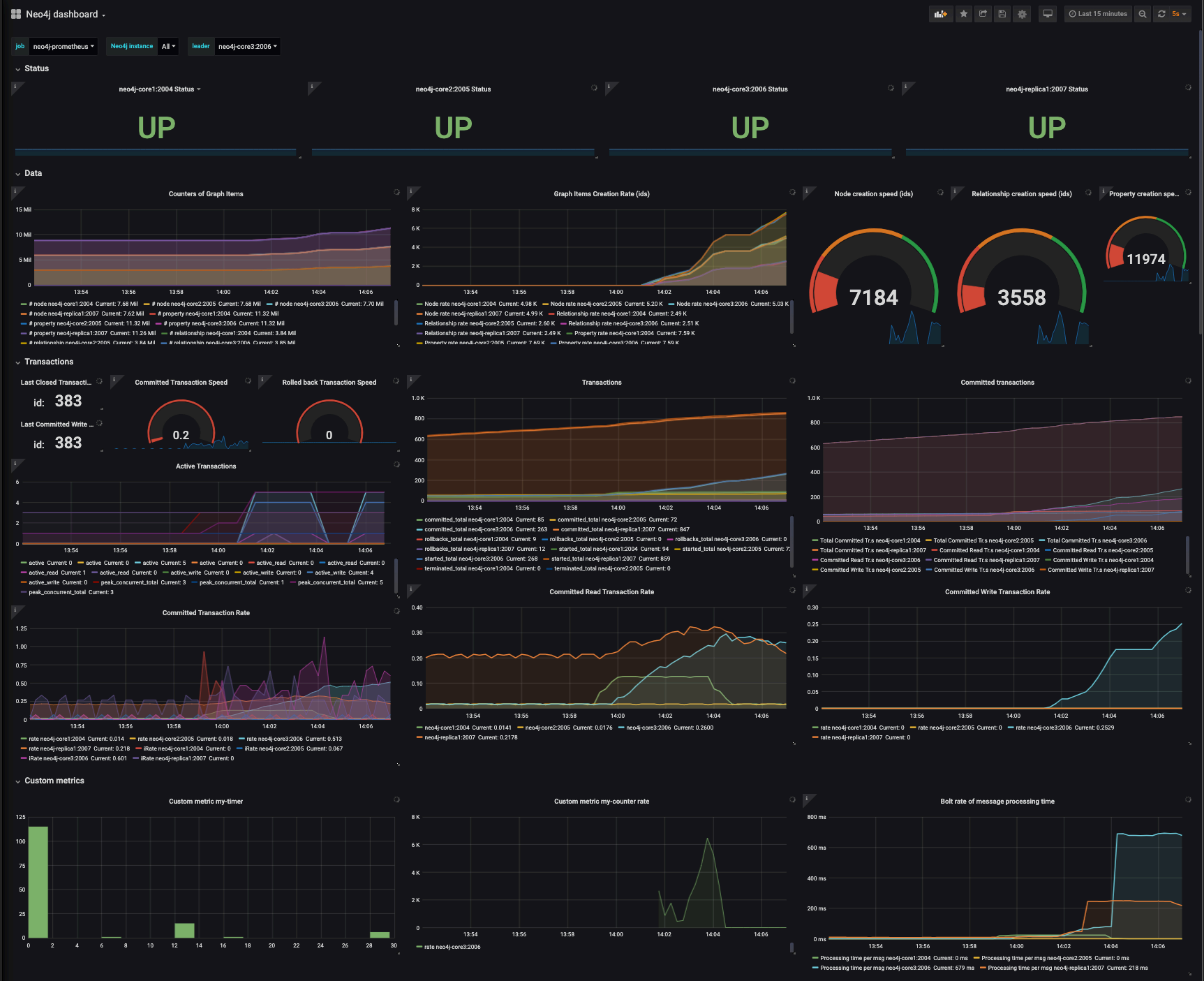
Monitoring Neo4j and Procedures with Prometheus and Grafana - Part 1

by Janos Szendi-Varga, Miro Marchi

11 June 2019

Neo4j Monitoring

- https://graphaware.com/neo4j/2019/06/11/monitoring-neo4j-prometheus.html



Status

neo4j-core1:2004 Status — UP
neo4j-core2:2005 Status — UP
neo4j-core3:2006 Status — UP
neo4j-replica1:2007 Status — UP

Data

Counters of Graph Items

```
# node neo4j-core1:2004  Current: 7.68 Mil   # node neo4j-core2:2005  Current: 7.68 Mil   # node neo4j-core3:2006  Current: 7.70 Mil
# node neo4j-replica1:2007  Current: 7.62 Mil   # property neo4j-core1:2004  Current: 11.32 Mil
# property neo4j-core2:2005  Current: 11.32 Mil   # property neo4j-core3:2006  Current: 11.32 Mil
# property neo4j-replica1:2007  Current: 11.26 Mil   # relationship neo4j-core1:2004  Current: 3.84 Mil
# relationship neo4j-core2:2005  Current: 3.84 Mil   # relationship neo4j-core3:2006  Current: 3.85 Mil
```

Graph Items Creation Rate (ids)

```
Node rate neo4j-core1:2004  Current: 4.98 K   Node rate neo4j-core2:2005  Current: 5.20 K   Node rate neo4j-core3:2006  Current: 5.03 K
Node rate neo4j-replica1:2007  Current: 4.99 K   Relationship rate neo4j-core1:2004  Current: 2.49 K
Relationship rate neo4j-core2:2005  Current: 2.60 K   Relationship rate neo4j-core3:2006  Current: 2.51 K
Relationship rate neo4j-replica1:2007  Current: 2.49 K   Property rate neo4j-core1:2004  Current: 7.59 K
Property rate neo4j-core2:2005  Current: 7.69 K   Property rate neo4j-core3:2006  Current: 7.59 K
```

Node creation speed (ids) — 7184
Relationship creation speed (ids) — 3558
Property creation spe... — 11974

Transactions

Last Closed Transacti... id: 383
Last Committed Write ... id: 383
Committed Transaction Speed — 0.2
Rolled back Transaction Speed — 0

Active Transactions

```
active  Current: 0   active  Current: 0   active  Current: 5   active  Current: 0   active_read  Current: 0   active_read  Current: 0
active_read  Current: 1   active_read  Current: 0   active_write  Current: 0   active_write  Current: 0   active_write  Current: 4
active_write  Current: 0   peak_concurrent_total  Current: 3   peak_concurrent_total  Current: 1   peak_concurrent_total  Current: 5
peak_concurrent_total  Current: 3
```

Transactions

```
committed_total neo4j-core1:2004  Current: 85   committed_total neo4j-core2:2005  Current: 72
committed_total neo4j-core3:2006  Current: 263   committed_total neo4j-replica1:2007  Current: 847
rollbacks_total neo4j-core1:2004  Current: 9   rollbacks_total neo4j-core2:2005  Current: 0   rollbacks_total neo4j-core3:2006  Current: 0
rollbacks_total neo4j-replica1:2007  Current: 12   started_total neo4j-core1:2004  Current: 94   started_total neo4j-core2:2005  Current: 7
started_total neo4j-core3:2006  Current: 268   started_total neo4j-replica1:2007  Current: 859
terminated_total neo4j-core1:2004  Current: 0   terminated_total neo4j-core2:2005  Current: 0
```

Committed transactions

```
Total Committed Tr.s neo4j-core1:2004   Total Committed Tr.s neo4j-core2:2005   Total Committed Tr.s neo4j-core3:2006
Total Committed Tr.s neo4j-replica1:2007   Committed Read Tr.s neo4j-core1:2004   Committed Read Tr.s neo4j-core2:2005
Committed Read Tr.s neo4j-core3:2006   Committed Read Tr.s neo4j-replica1:2007   Committed Write Tr.s neo4j-core1:2004
Committed Write Tr.s neo4j-core2:2005   Committed Write Tr.s neo4j-core3:2006   Committed Write Tr.s neo4j-replica1:2007
```

Committed Transaction Rate

```
rate neo4j-core1:2004  Current: 0.014   rate neo4j-core2:2005  Current: 0.018   rate neo4j-core3:2006  Current: 0.513
rate neo4j-replica1:2007  Current: 0.218   iRate neo4j-core1:2004  Current: 0   iRate neo4j-core2:2005  Current: 0.067
iRate neo4j-core3:2006  Current: 0.601   iRate neo4j-replica1:2007  Current: 0
```

Committed Read Transaction Rate

```
neo4j-core1:2004  Current: 0.0141   neo4j-core2:2005  Current: 0.0176   neo4j-core3:2006  Current: 0.2600
neo4j-replica1:2007  Current: 0.2178
```

Committed Write Transaction Rate

```
rate neo4j-core1:2004  Current: 0   rate neo4j-core2:2005  Current: 0   rate neo4j-core3:2006  Current: 0.2529
rate neo4j-replica1:2007  Current: 0
```

Custom metrics

Custom metric my-timer

Custom metric my-counter rate

```
rate neo4j-core3:2006
```

Bolt rate of message processing time

```
Processing time per msg neo4j-core1:2004  Current: 0 ms   Processing time per msg neo4j-core2:2005  Current: 0 ms
Processing time per msg neo4j-core3:2006  Current: 679 ms   Processing time per msg neo4j-replica1:2007  Current: 218 ms
```

2. BREAKING THINGS

- Simulating the failure of an entire region or datacenter.
- Partially deleting Kafka topics over a variety of instances to recreate an issue that occurred in production.
- DNS unavailability
- Injecting latency between services for a select percentage of traffic
- Function-based chaos (runtime injection): Randomly causing functions to throw exceptions.
- Code insertion: Adding instructions to the target program and allowing fault injection to occur prior to certain instructions.
- Time travel: Forcing system clocks out of sync with each other.
- Executing a routine in driver code emulating I/O errors.
- Maxing out CPU cores on a cluster.

BIG RED BUTTON

- we must have the control to stop the experiment any time
- each chaos engineering activity has the potential to cause a production outage
- every “fault tolerant” element of the infrastructure should be tested

TOOLS YOU CAN USE

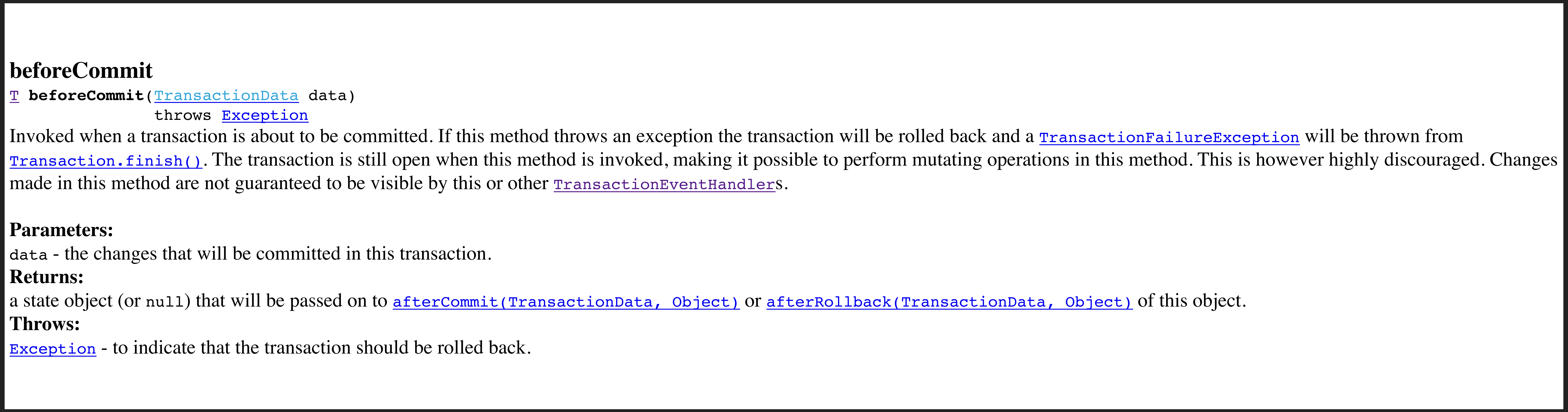
- Chaos Monkey (https://github.com/Netflix/chaosmonkey)
- Mangle (https://vmware.github.io/mangle/)
- https://github.com/asatarin/testing-distributed-systems
- https://github.com/dastergon/awesome-chaos-engineering
- Chaos Toolkit (https://github.com/chaostoolkit/chaostoolkit/)
- CM4SB https://github.com/codecentric/chaos-monkey-spring-boot
    - Latency Assault
    - Exception Assault
    - AppKiller Assault
    - Memory Assault
- Visualise your Chaos Engineering Experiments with Grafana https://www.youtube.com/watch?v=Gua-QcdoivU

LET'S DO CHAOS IN YOUR NEO4J

- Simulating failure in your Neo4j cluster
  - Kill one instance from the cluster (Known Knowns)
  - Ingestion problems
  - Time travel: Forcing system clocks out of sync with each other.
  - Injecting latency between services for a select percentage of traffic over a predetermined period of time.
  - Failure Injection Testing (FIT): Randomly causing faults in your Neo4j transactions and "see what happens."

HOW TO DO CHAOS IN YOUR NEO4J

- Randomly causing faults in your Neo4j transactions and "see what happens."
  - APOC trigger function - not the best
  - Improved Transaction Event API (GraphAware Framework)
  - Neo4j Transaction Event API - TransactionEventHandler - Triggers



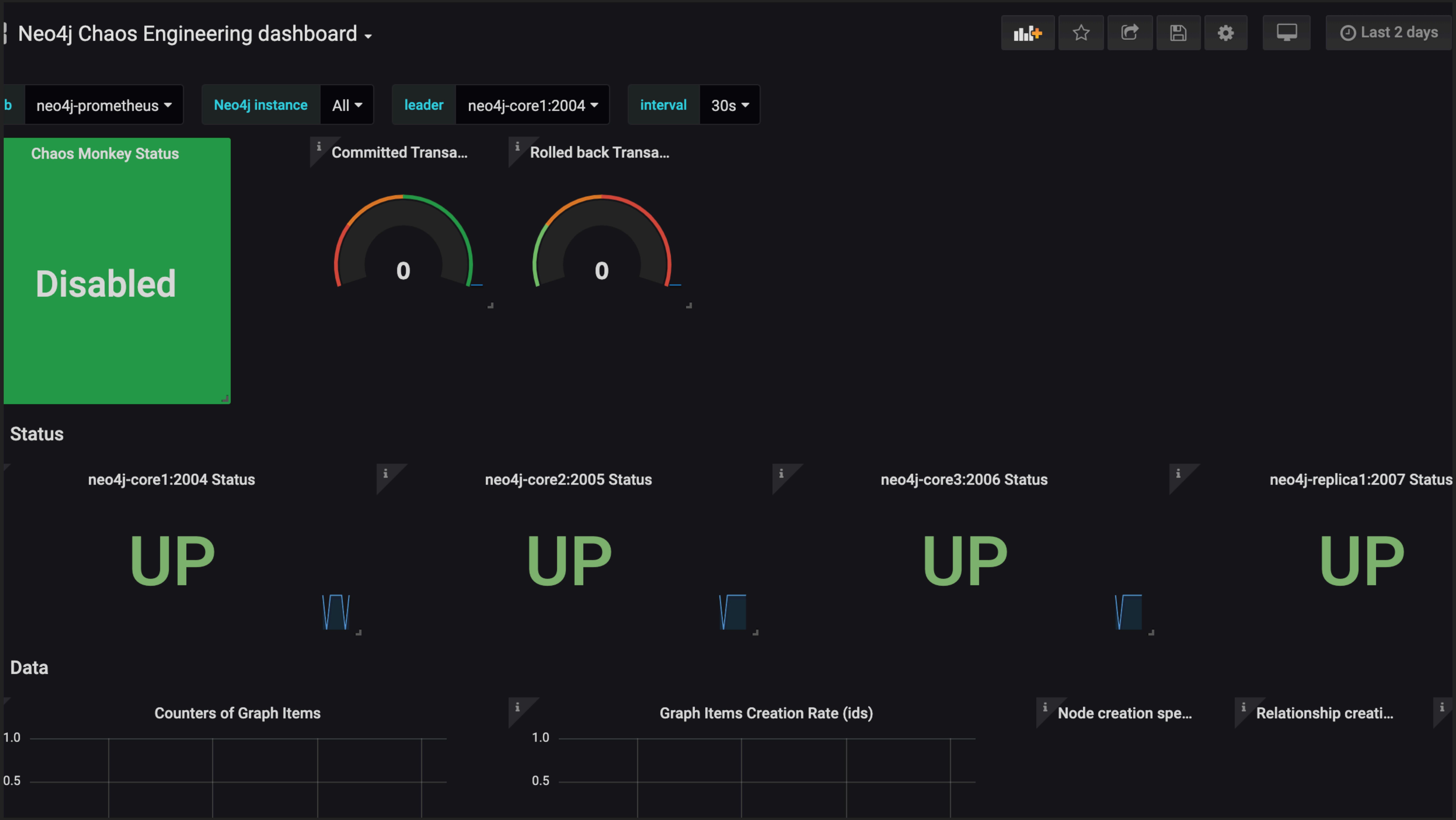
beforeCommit

```
T beforeCommit(TransactionData data)
               throws Exception
```

Invoked when a transaction is about to be committed. If this method throws an exception the transaction will be rolled back and a `TransactionFailureException` will be thrown from `Transaction.finish()`. The transaction is still open when this method is invoked, making it possible to perform mutating operations in this method. This is however highly discouraged. Changes made in this method are not guaranteed to be visible by this or other `TransactionEventHandler`s.

Parameters:
`data` - the changes that will be committed in this transaction.
Returns:
a state object (or `null`) that will be passed on to `afterCommit(TransactionData, Object)` or `afterRollback(TransactionData, Object)` of this object.
Throws:
`Exception` - to indicate that the transaction should be rolled back.

```java
public class MyTransactionEventHandler implements TransactionEventHandler {
    public static GraphDatabaseService db;
    private static ExecutorService ex;
    public static Log LOGGER;

    private final ChaosMonkeySettings settings;
    private static final MetricRegistry METRIC_REGISTRY = new MetricRegistry();
    private static final Counter exceptionCounter = METRIC_REGISTRY.counter(name(MyTransactionEventHandler.class,  ...names: "exceptionCounter"));

    public MyTransactionEventHandler(GraphDatabaseService graphDatabaseService, ExecutorService executor, LogService logService, ChaosMonkeySettings chaosMonkeySettings) {
        db = graphDatabaseService;
        ex = executor;
        this.LOGGER = logService.getUserLog(MyTransactionEventHandler.class);
        this.settings = chaosMonkeySettings;

        CollectorRegistry.defaultRegistry.register(new DropwizardExports(METRIC_REGISTRY));
    }

    @Override
    public Object beforeCommit(TransactionData transactionData) throws Exception {
        if (isActive()) attack();

        return null;
    }

    @Override
    public void afterCommit(TransactionData transactionData, Object o) {
    }

    @Override
    public void afterRollback(TransactionData transactionData, Object o) {
    }

    private void attack() {
        LOGGER.info( s: "Chaos Monkey - exception");
        AssaultException assaultException = this.settings.getAssaultProperties().getException();
        exceptionCounter.inc();
        assaultException.throwExceptionInstance();
    }

    private boolean isActive() {
        return settings.getAssaultProperties().isExceptionsActive();
    }
}
```


Neo4j Chaos Engineering dashboard

Last 2 days

b neo4j-prometheus | Neo4j instance All | leader neo4j-core1:2004 | interval 30s

Chaos Monkey Status

Disabled

Committed Transa...

0

Rolled back Transa...

0

Status

neo4j-core1:2004 Status	neo4j-core2:2005 Status	neo4j-core3:2006 Status	neo4j-replica1:2007 Status
UP	UP	UP	UP

Data

Counters of Graph Items

Graph Items Creation Rate (ids)

Node creation spe...

Relationship creati...

THANK YOU!

- @szenyo (Twitter)
- janos@graphcoding.com

HUNGER GAMES QUESTIONS FOR CHAOS ENGINEERING WITH NEO4J

1. Easy: What is Chaos Engineering?
    A. A typical day in any IT company
    B. Chaos engineering is the discipline of experimenting on a software system in production in order to build confidence in the system's capability to withstand turbulent and unexpected conditions
    C. To deploy a new release into production
2. Medium: Which company published Chaos Monkey?
    A. Amazon
    B. Google
    C. Netflix
3. Hard: What is the name of the method of the TransActionEventHandler we used for Failure Injection Testing (FIT)?

Answer here: http://r.neo4j.com/hunger-games