Best Practices Discussion

- Common Problems in Deployments

Revised: June 22nd, 2020

CS300 Couchbase Server Administration

April 20, 2020

Couchbase

Learning Services

Labs – Day 4

Lab #7 (Advanced XDCR & Backup/Restore):

- **Study optimistic replication in detail**
- **Write 100,000 items under the optimistic threshold**
- **Write 100,000 items over the optimistic threshold**
- **Demonstrate that XDCR replication keeps occurring even after a node failure**
- **Use vBucketkeygen tool to write 1 key to each of the 1024 vBuckets**
- **Use cbbackup and cbrestore**

Labs – Day 4

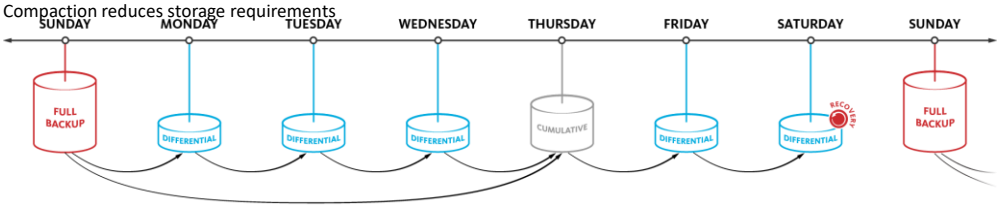
Lab #8 (Performance & Compaction):

- **Examine high water mark and low water mark**
- **Examine how ejection works**
- **Observe the Not-Recently-Used metadata setting**
- **Learn about the item pager**
- **Observe Couchbase memory management**
- **Compare Disk reads vs RAM reads**
- **Compare different traffic patterns in Couchbase**
- **Examine the 'resident %' in memory metric**
- **Observe out of memory (OOM) errors**
- **Display metrics via cbstats and study item expiration**
- **Display timing metrics with "cbstat timings"**
- **Perform a manual Compaction**

Backup and Restore

Key Features

- Zero downtime backup and restore
- Single utility to manage multiple clusters
- Restore from any point, to **any bucket or topology**
- Fully differential – merge backups to maintain desired restore points
- Built-in concurrency control
- Resume interrupted backups
- Compaction reduces storage requirements



COMBINING CUMULATIVE AND DIFFERENTIAL INCREMENTAL BACKUPS

Backup and Restore (cbbackupmgr Tool)

cbbackupmgr Tool

All backup is stored in and recovered from a Backup Respository.

In turn, a Backup Respository is stored in a Backup Archive on the filesystem.
Each backup job in the Backup Repository stores its backup in two ways:

- **All bucket data is stored in a secondary smaller database (ForestDB).**
- **All bucket creation scripts and configuration files are stored on the file system as files.**

Important:
The cbbackupmgr tool is not backward compatible for backups created using cbbackup.

8

How the Backup and Restore Tool Works

By default, the cbbackupmgr tool performs incremental backups to back up only the new data.
However, on a new cluster and for the first time, this tool generates a full backup.
Each of the subsequent incremental backups take a fraction of the time taken by the full backup.

Archive Repository

The backup archive is a directory that contains a set of backup repositories as well as logs for the backup client.
The backup directory should be modified only by the backup client, and any modifications that are not done by that client might result in a corruption of backup data.

Only one backup client can access the backup archive at one time.
If multiple instances of the backup client are running on the same archive at the same time, this might result in corruption.

To prevent such corruption instances, you may be required to create multiple backup archives depending on your use case.

Configuring a Backup

```
$ cbbackupmgr config --archive
/data/backup --repo cluster

Backup repository `cluster`
created successfully in archive
`/data/backup`
```

9

Before getting started with cbbackupmgr you must first decide the directory where to store all of your backups.
This directory is referred to as the backup archive. The backup archive contains one or more backup repositories.
These backup repositories are where your backups will be contained.

The easiest way to think of a backup repository is that it corresponds directly to a single cluster that you want to back up.
The backup repository also contains a configuration for how to back that cluster up.
A backup repository is created by using the **config** sub-command.

In this tutorial we will use a backup archive located at /data/backup.
The backup archive is automatically created if the directory specified is empty.

This is an example of how to create a backup repository called "cluster" which backs up all data and index definitions from all buckets in the target cluster.

Configuring a Backup (continued)

```
$ cbbackupmgr config --archive
/data/backup --repo single \
--include-buckets travel-sample
--disable-data

Backup repository `single` created
successfully in archive
`/data/backup`
```

10

One of the most important aspects of backup repository creation is that you can configure that backup repository in many different ways to change the way backups in each backup repository are taken.
Let's say you want a separate backup of only the index definitions in the travel-sample bucket.

To do this you can create a separate backup repository called "single" using the following command:

The config sub-command provides many options in order to customize how you backup your data.

For more information about the available options and how they are used, see cbbackupmgr config.

Backing up a Cluster

```
$ cbbackupmgr list --archive
/data/backup

Size        Items              Name
0B          -                  /
0B          -                  + cluster
0B          -                  + single
```

11

The list sub-command returns a directory print out of all of the backup repositories and backups in your backup archive.
Since there are no backups yet you can just see your archives list in the output of this command.
There is also information about how much disk space each folder and file contains and, if applicable, how many items are backed up in those folders/files.

For more information about the list sub-command, see cbbackupmgr list.

Run cbbackupmgr

```
$ cbbackupmgr backup --archive /data/backup --repo
cluster \
--host couchbase://127.0.0.1 --username Administrator -
-password password

Backing up to 2016-03-22T10_26_08.933579821-07_00
Copied all data in 6s (Avg. 6.67MB/Sec)        38894
items / 40.02MB
travel-sample
[====================================] 100.00%
beer-sample
[====================================] 100.00%

Backup successfully completed
```

12

When the backup command is executed, by default it prints out a progress bar which is helpful to understand how long your backup will take to complete and the rate of data movement.
While the backup is running, the progress bar gives an estimated time to completion, and when the backup completes, but this changes to the average backup rate. Information is also provided on the total data and items already backed up and the current rate of data movement.

If the backup completes successfully, the tool prints the message "Backup completed successfully" as the last line.

Run cbbackupmgr (continued)

```
$ cbbackupmgr backup --archive /data/backup --repo single
\
--host couchbase://127.0.0.1 --username Administrator --
password password

Backing up to 2016-03-22T10_33_20.812668465-07_00
Copied all data in 1s (Avg. 480B/Sec)                  0
items / 480B
travel-sample  [==================================] 100.00%
```

13

Since the "single" backup repository is only configured to back up index definitions for the travel-sample bucket you do not see a progress bar for the beer-sample bucket.

You can also see that the backup executed quicker since there was considerably less data to actually back up.

Restoring a Backup

```
$ cbbackupmgr restore --archive /data/backup --repo cluster \
--host http://127.0.0.1:8091 --username Administrator --password
password \
--start 2016-03-22T14_00_16.892277632-07_00 \
--end 2016-03-22T14_00_16.892277632-07_00 --force-updates

(1/1) Restoring backup 2016-03-22T14_00_16.892277632-07_00
Copied all data in 2s (Avg. 19.96MB/Sec)       38894 items /
39.91MB
travel-sample            [==================================]
100.00%
beer-sample              [==================================]
100.00%

Restore completed successfully
```

14

Now that you have some backup data let's restore that data backup to the cluster.

In order to restore data you just need to know the name of the backup that you want to restore.
To find the name you can use the list sub-command in order to see what is in our backup archive.
The backup name will always be a timestamp.

For example, let's say you want to restore the 2016-03-22T10_26_08.933579821-07_00 from the "cluster" backup repository. In order to do this, run the following command:

```
$ cbbackupmgr restore --archive /tmp/backup --repo cluster \
--host http://127.0.0.1:8091 --username Administrator --password password \
--start 2016-03-22T14_00_16.892277632-07_00 \
--end 2016-03-22T14_00_16.892277632-07_00 --force-updates
```

In the command above, notice the use of the --start and --end flags to specify the range of backups you want to restore.

Since you are only restoring one backup, specify the same value for both --start and --end.
The --force-updates flags skip Couchbase conflict resolution.
This tells cbbackupmgr to force overwrite key-value pairs being restored even if the key-value pair on the cluster is newer than the one being restored.
If you look at the two values that were updated on the cluster, you will now see that they have been reverted to what they were at the time we took the initial backup.
If you used the script in the backup-tutorial GitHub repository to update documents then you can use the 03-inspect.sh script to see the state of the updated documents after the restore.

Lab #7 (Advanced XDCR & Backup/Restore)

- Study optimistic replication in detail
- Write 100,000 items under the optimistic threshold
- Write 100,000 items over the optimistic threshold

Time: 1 hour

- Demonstrate that XDCR replication keeps occurring even after a node failure
- Use vBucketkeygen tool to write 1 key to each of the 1024 vBuckets
- Use cbbackupmgr

Labs

Doing labs

6.0	**XDCR**
7.0	**adv XDCR**
7.1	**NTP**
and	
7.2	**cbbackupmgr**

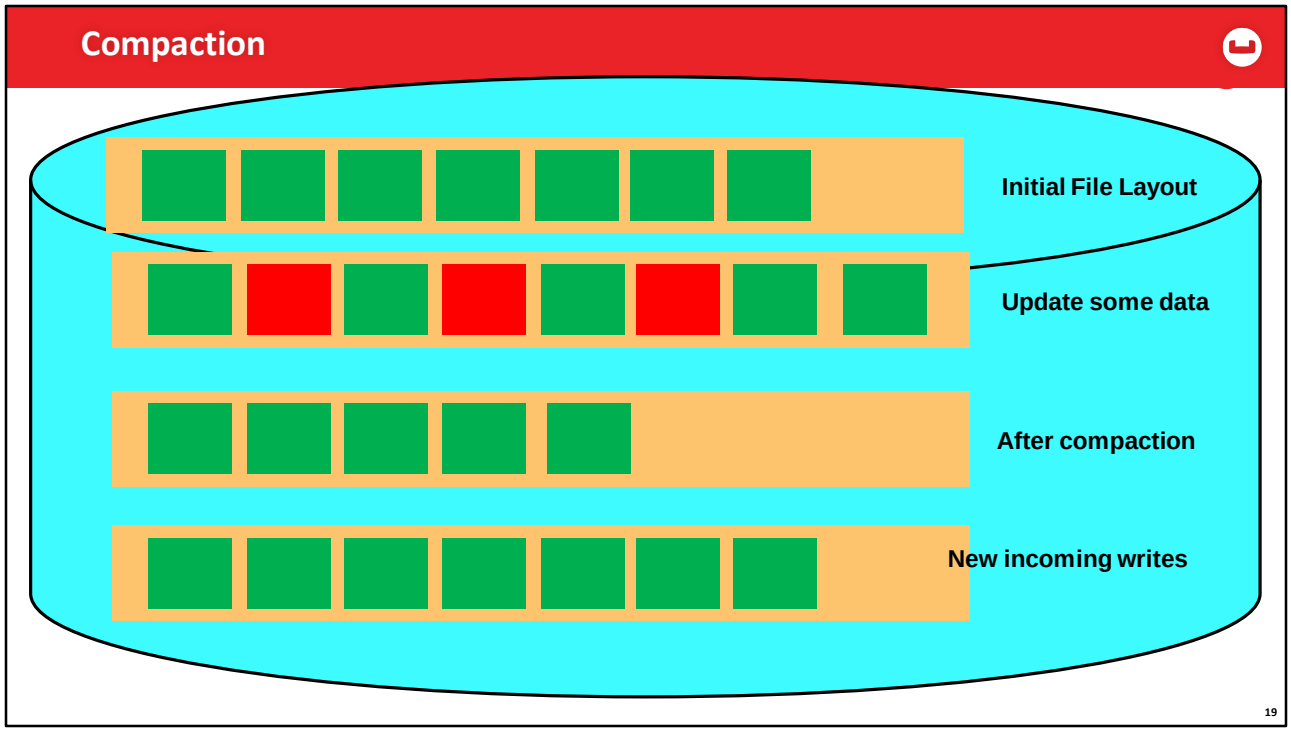
Compaction

sudo –i

cd /opt/couchbase/var/lib/couchbase/data/NYC-Bucket

ls –alh

what is the extension number on your files?



Append-only file format puts all new/updated/deleted items at the end of the on-disk file.
Better performance and reliability
No more fragmentation!

This can lead to invalidated data in the “back” of the file.

Need to compact data.

The compaction process operates incrementally on a per-vbucket basis, and is controlled by both a fragmentation threshold and a time of day setting. It works by creating a new file with just the latest data and then switching over from the main one to the new one when complete.

Compaction Operation

- Compaction creates a new file
  - **Available disk space is checked (need twice current data size)**
- Existing data is written to new file
  - **Existing file is used until compaction has completed**
  - **Data file extension is numeric; new version has uses next increment (i.e. current data in 1.couch.12 is compacted to 1.couch.13)**
- Once compaction is complete, old file disabled, new file enabled, old files deleted
- Compaction may not complete
  - **In write-intensive situations**
  - **Compaction may never 'catch-up' with**
- Compaction is both disk and CPU intensive

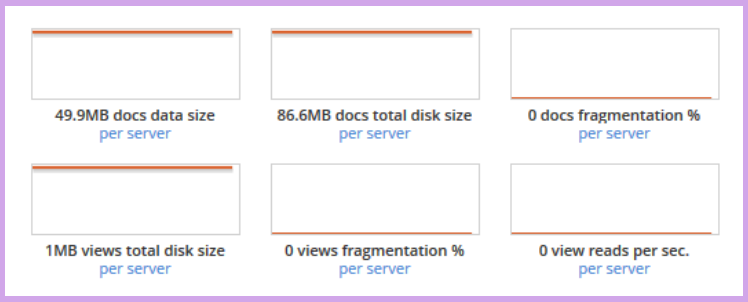
Compaction of write-heavy databases

It is not a good idea to attempt compaction on a database node that is near full capacity for its write load. The problem is the compaction process may never catch up with the writes if they never let up, and eventually it will run out of disk space.

Compaction should be attempted when the write load is less than full capacity. Read load won't affect its ability to complete, however. To have the least impact possible on clients, the database remains online and fully functional to readers and writers. It is a design limitation that database compaction can't complete when at capacity for write load. It may be reasonable to schedule compactions during off-peak hours.

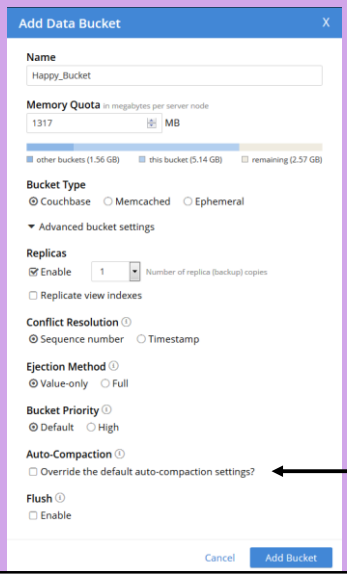
Identifying Compaction

- Check docs fragmentation percentage
- Check views fragmentation percentage
- Compare docs and views disk size and data size



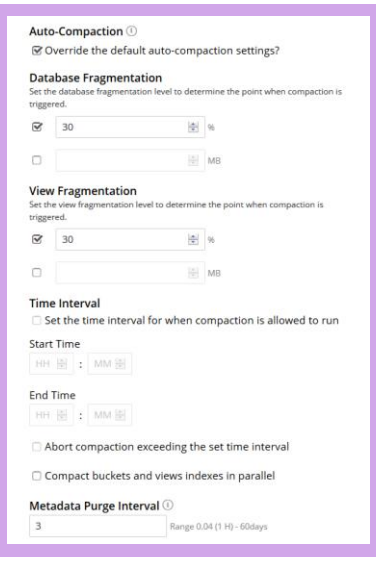
doc fragmentation %
Document fragmentation of persisted data as stored on disk.

Auto-Compaction

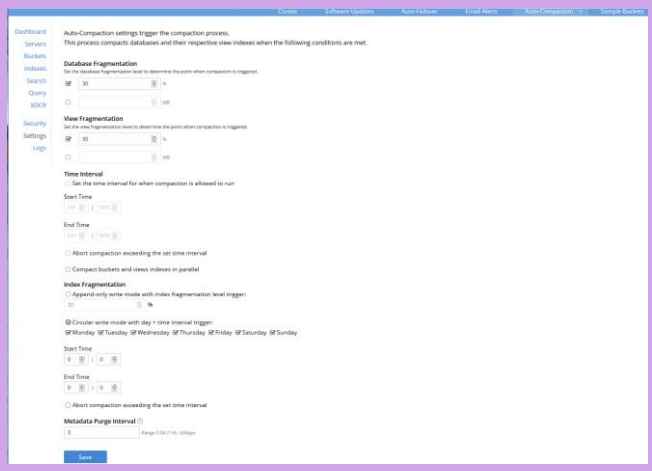


Auto-Compaction

Auto-compaction

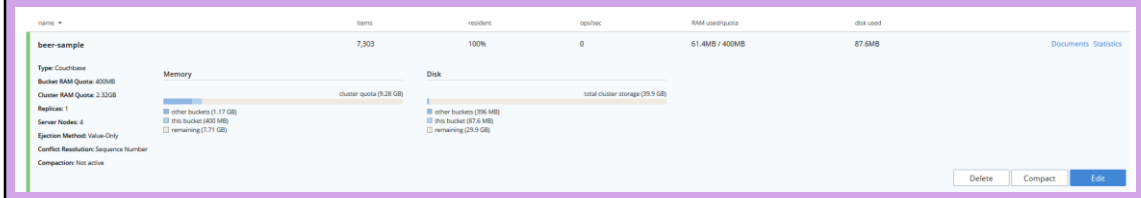


Auto-compaction



Manual Compaction

- **Start Manual Compaction from Data Buckets page**



Auto-Compaction

- Optimizes disk file usage/fragmentation
- Occurs in the background

- Triggered by:
  - Data file fragmentation percentage or MB
  - View file fragmentation percentage or MB

- Can be time-limited
- Can be stopped automatically if not complete within limits

- Default auto-compaction applies to all buckets
- Per-bucket auto-compaction allows for per bucket config

Compaction

Compaction takes place as a background process while Couchbase Server is running.

Make sure you perform compaction:
- **on every server**
- **during off-peak hours**
- **with adequate disk space**

Compaction activity is reported in the Couchbase Server log.

27

Compaction from web UI kicks it off on all nodes in parallel…

Compaction operates on only a single server within your Couchbase Server cluster. You will need to perform compaction on each node in your cluster, on each database in your cluster.

Because compaction occurs by creating new files and updating the information, you may need as much as twice the disk space of your current database and index files for compaction to take place.

Monitoring basics

Monitoring Objectives

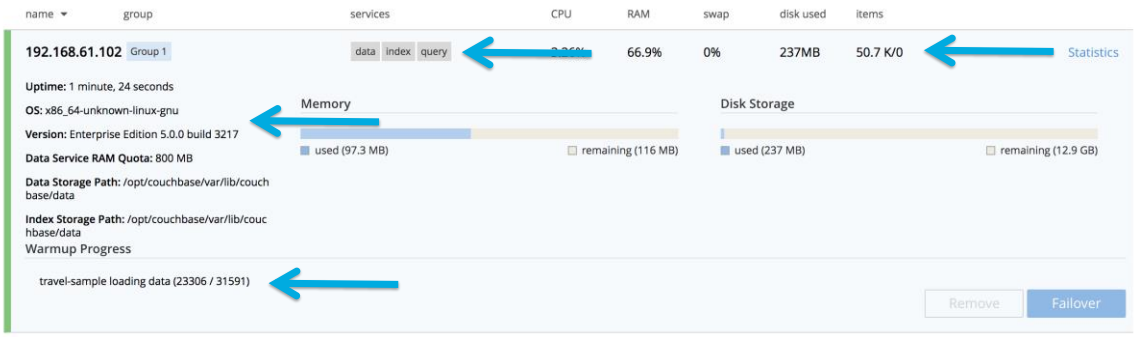
1. Ensure availability and performance.
2. Retroactively identify the cause of downtime or poor performance.
3. Predict scaling requirements.

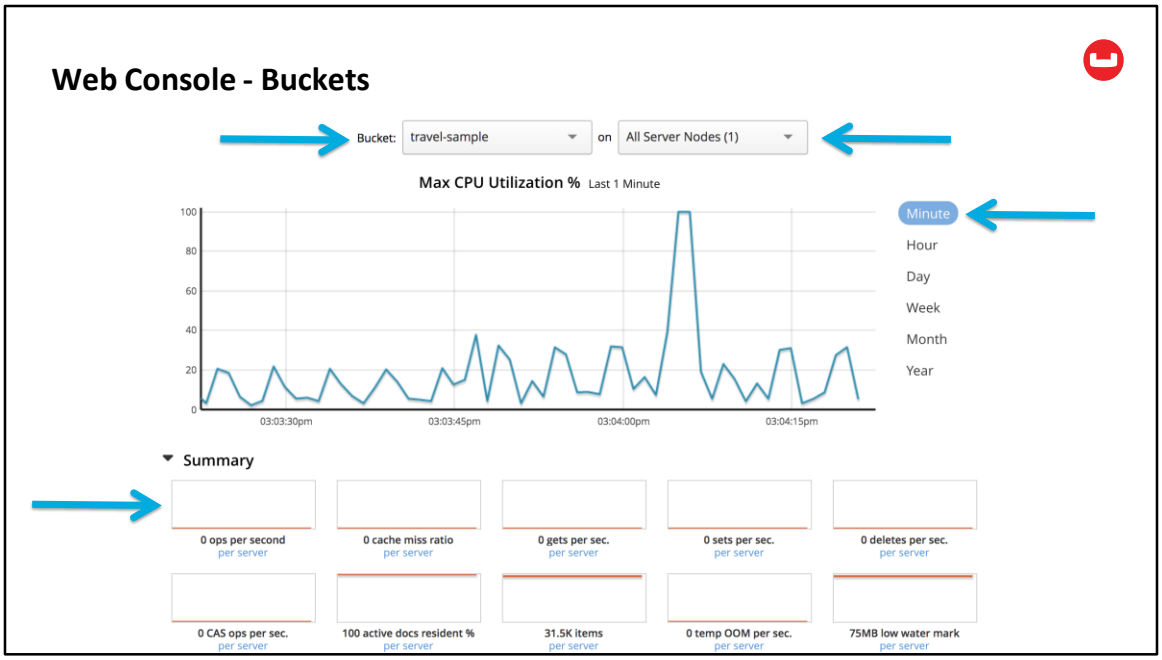
1. This means proactively identifying issues or resource consumption trends that could eventually lead to downtime or poor performance
2. Stuff happens and sometimes a bad batch of hard drives fail at the same time, or the new network guy sets all your ports to 10Mbps, or the application has a bug that DOSes the cluster. If we can't prevent it, we need to be able to figure out what it was so we can fix it.

What Couchbase Provides

1. High availability and memory-first architecture
2. Real time web console
3. Email alerting
4. REST APIs

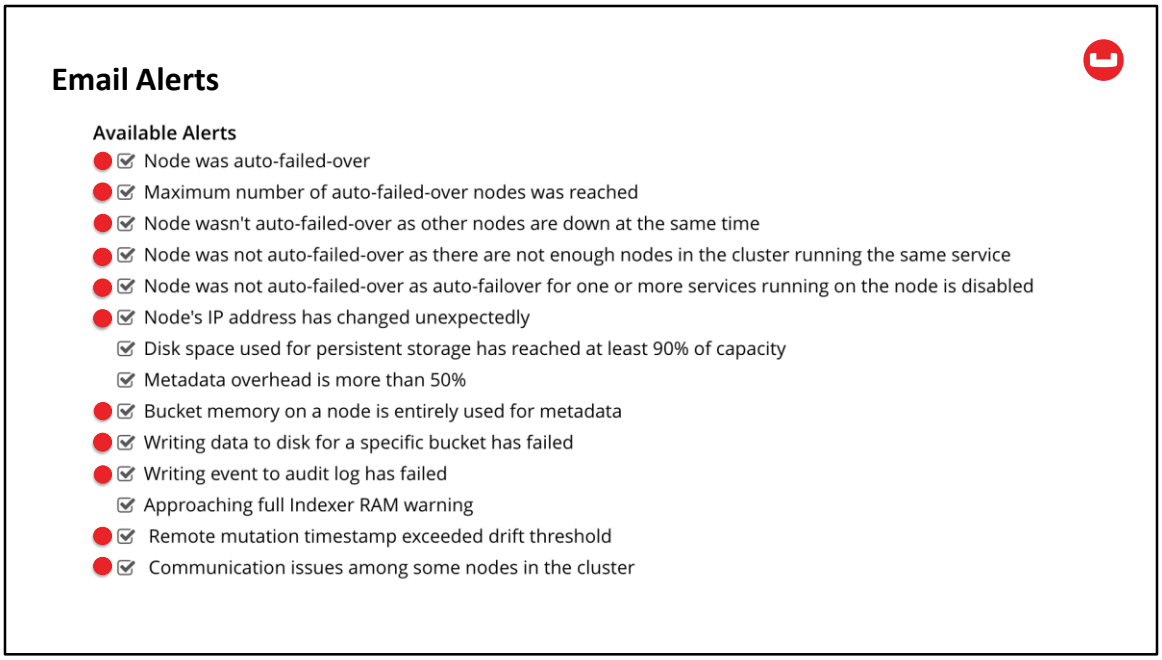
Web Console - Servers





1. 120+ base with additional stats added for each index and XDCR replication
2. Stats are viewable by bucket
3. Also viewable by server or aggregated across all servers
4. Stats can be trended from the default minutely to yearly, however note that the data points are heavily down sampled over time

This makes the interface great for realtime status and debugging, but historical data is only useful for high level trending. The interface likely won't help you identify the cause of a transient latency spike that occurred 3 days ago.



Most alerts are re-active – only sending notifications after a failure has already occurred.

What we want is proactive alerting so that we can avoid the failure altogether.

Components of an external monitoring system

- Couchbase REST APIs
- A metrics collector
- An alerting utility
- A trending utility

Integrations

Off-the-Shelf Integrations

Fetching Couchbase Stats

- /opt/couchbase/bin/cbstats
- /opt/couchbase/var/lib/couchbase/logs/stats.log
- REST API

The REST API

- APIs
  - Cluster
  - Data Service
  - Query Service
  - Index Service
  - Search Service
- All return JSON
- Require correct administrator permissions

Tips for interacting with the REST API

- **API calls can be expensive, try to be efficient.**
  - **Rather than calling the API for each metric, call the API once and evaluate the result for multiple metrics**
- **If your monitoring system supports dynamic config, the REST API can be used to "discover" new hosts, services, buckets, and replications.**
- **Use the Read-Only Administrator or a user with the "Data Monitoring" role.**

Reference Implementations

- **Plugins**
  - **Nagios:** https://github.com/deanproctor/nagios-plugin-couchbase
  - **Graphite:**
- **Monitoring installations**
  - **Nagios:** https://github.com/deanproctor/couchbase-reference-nagios
  - **Graphite:**

Reference Flow

1. Discover cluster configuration
   - Nodes
   - Services
   - Buckets
   - XDCR replications
2. Fetch stats
3. Evaluate values
4. Send results

Programmatic Discovery: Nodes and Services

```
$ curl http://localhost:8091/pools/default
"nodes": [
    {
        "clusterMembership": "active",
        "hostname": "192.168.61.102:8091",
        "services": [
              "index",
              "kv",
              "n1ql"
        ],
        "status": "healthy",
        "thisNode": true,
    }
]
```

Programmatic Discovery: Buckets

```
$ curl http://localhost:8091/pools/default/buckets
[
    {
        "name": "sessions",
        "replicaNumber": 1,
        "bucketType": "membase",
        "quota": {
              "ram": 104857600
        }
    }
]
```

Programmatic Discovery: XDCR Replications

```
$ curl http://localhost:8091/pools/default/buckets
[
    {
        "type": "xdcr"
        "id": "9ce71459d664173f14fea36d88cdd68c/test/travel-sample",
        "source": "test",
        "status": "running"
    },
    {
        "type": "rebalance"
        "status": "notRunning"
    }
]
```

Bucket Stats

```
$ curl http://localhost:8091/pools/default/buckets/<bucket>/stats
[
    {
        "op": {
            "samples": {
                "stat1": [0, 0, 0, 0, 0, 0, 0, 0, 0, 0 …],
                "stat2": [0, 0, 0, 0, 0, 0, 0, 0, 0, 0 …],
                "stat3": [0, 0, 0, 0, 0, 0, 0, 0, 0, 0 …],
            }
        }
    }
]
```

XDCR Stats

```
$ curl
http://localhost:8091/pools/default/buckets/<bucket>/stats/replic
ations/<taskId>/<stat>
{
    "nodeStats": {
        "localhost:8091": [0, 0, 0, 0, 0, 0, 0, 0, 0, 0 …],
    },
    "samplesCount": 60
}
```

Query Stats

```
$ curl http://localhost:8093/admin/stats
{
    "active_requests.count": 0,
    "deletes.count": 0,
    "errors.count": 0,
    "inserts.count": 0,
    "request_rate.1m.rate": 2.1440102386515474e-16,
    "request_rate.5m.rate": 0.000056098631329448915,
    "request_timer.1m.rate": 2.1440102386515474e-16,
    "request_timer.5m.rate": 0.000056098631329448915,
    "request_timer.75%": 126887937.5,
    "request_timer.95%": 723846717.5499995,
    "request_timer.99%": 2021658096.2799993
}
```

Search Stats

```
$ curl http://localhost:8094/api/nsstats
{
  "num_bytes_used_ram": 5155920,
  "pct_cpu_gc": 0.0002442683307446239,
  "myBucket:myIndex:avg_queries_latency": 0,
  "myBucket:myIndex:batch_merge_count": 0,
  "myBucket:myIndex:doc_count": 0,
  "myBucket:myIndex:iterator_next_count": 0,
  "myBucket:myIndex:iterator_seek_count": 0,
  "myBucket:myIndex:num_bytes_live_data": 0,
  "myBucket:myIndex:num_bytes_used_disk": 75162
}
```

Real-time Monitoring

Objectives

In order of precedence:

1. **Confirm services are online and accessible.**
2. **Ensure resource usage is within safe levels.**
3. **Evaluate state of failure predictors.**
4. **Ensure response times are within acceptable range.**

OS Metrics

- Processes
- Ports
- System load
- CPU utilization
- Disk free
- Memory free
- Swap utilization
- Used file descriptors
- TCP connections
- System time

Reference config: https://github.com/deanproctor/couchbase-reference-nagios/blob/master/etc/objects/services/

Trended Monitoring

Objectives

1. **Capture sufficient data to retroactively identify issues.**
2. **Provide dashboards that efficiently display the most necessary information.**
3. **Predict future scaling requirements.**

Query monitoring and troubleshooting

Agenda

- Objectives
- Monitoring
  - through REST
  - through N1QL
- Advanced use of the system keyspaces
- New for 5.0: profiling

55

In a nutshell

- There are times when things don't work as expected
- Monitoring gives you the option of
  - discovering the culprit
  - investigating the problem
- In quiet times, monitoring gives you the opportunity to explore the activity in your N1QL service and plan for growth.

56

REST API

- **Endpoints:**
  - http://localhost:8093/admin/vitals
  - http://localhost:8093/admin/active_requests
  - http://localhost:8093/admin/completed_requests
  - http://localhost:8093/admin/prepareds
- **(substitute localhost with desired host name)**

57

Vitals

- **Provide an overall picture of the individual node**
- **Only through REST**
- **GET http://localhost:8093/admin/vitals**

58

REST: active requests

- **Provide a list of requests running on an individual node**
  - **GET** http://localhost:8093/admin/active_requests
- **Provide information about an individual request**
  - **GET** http://localhost:8093/admin/active_requests/<request id>
- **Cancel a request**
  - **DELETE** http://localhost:8093/admin/active_requests/<request id>

59

REST: completed requests

- The completed requests cache provides a list of requests that have run longer than a predefined threshold
- Cache size and threshold configurable
- Provide a list of requests completed on an individual node
  - GET http://localhost:8093/admin/completed_requests
- Provide information about an individual request
  - GET http://localhost:8093/admin/completed_requests/<request id>
- Remove an entry from the completed requests cache
  - DELETE http://localhost:8093/admin/completed_requests/<request id>

60

REST: prepared statements

- **Provide a list of prepared statements on an individual node**
  - **GET** http://localhost:8093/admin/prepareds
- **Provide information about an individual prepared statement**
  - **GET** http://localhost:8093/admin/prepareds/<statement id>
- **Remove an entry from the prepared statements cache**
  - **DELETE** http://localhost:8093/admin/prepareds/<statement id>

61

Using Prepared Statements

When a N1QL statement is executed, the SDK sends the statement to the server as a string in the request body.
The server will inspect the string and make a query plan of which indexes it will need to query.
Once it has done this the server will generate a query plan which will require some additional computation which adds overhead to the query request.
To eliminate this overhead, statements can be prepared and the pre-generated plan reused

REST: cache configuration

- Change completed requests settings, per node
  - POST http://localhost:8093/admin/settings

    Settings passed as data, in json format
- Requires authorization
- Available completed requests settings
  - "completed-limit", cache size, integer, from zero upwards
  - "completed-threshold", minimum time duration in milliseconds, -1 (disabled) upwards
- Example
  - `Curl –X POST http://localhost:8093/admin/prepareds/<statement id> -u Administrator:password –d '{ "completed-limit": 10000 }'`

62

Couchbase (5.0): system keyspaces

- system:active_requests
- system:completed_requests
- system:prepareds
- system:nodes
- Can select from all
- Can delete from all bar system:nodes
- Cannot insert or update
- Operate against all known nodes
- Can perform node elimination
  - Document key includes node where item resides: fetch and scan automatically chose the correct node
  - Document include node name: can use in WHERE clause

63

Prepared statements with highest total execution time

- System:prepareds reports the average execution time and number of uses per each prepared statement
- Same prepared statement text might be prepared under different names
- Sum service time * uses and group by statement

```
select sum(str_to_duration(avgServiceTime)*uses/1000000000) ttime,
statement
  from system:prepareds
  where avgServiceTime is not missing
  group by statement
  order by ttime;
```

64

Prepared statements with highest execution or prepare counts

- As a variation of the previous query, we can find the most used statements

```
select sum(uses) usecount, statement
  from system:prepareds
  where uses is not missing
  group by statement
  order by usecount
```

- Or the statements prepared by the most clients

```
select count(*) prepcount, statement
  from system:prepareds
  group by statement
  order by prepcount;
```

65

Non prepared statements with highest execution times or counts

- Similar question as before, but on system:completed_requests
-
```
select duration_to_str(avg(str_to_duration(serviceTime)))
avgServiceTime,
     count(*) reqCount,
     statement
  from system:completed_requests
  group by statement
  order by avgServiceTime desc
```
- Same approach for system:active_requests

66

Cancelling unruly requests

- `DELETE FROM system:active_requests WHERE RequestId='…'`
- On 5.x, only works on the node the request is running on
  - Use the `–engine` option on cbq to connect to right node

67

Request Profiling – 5.0

Quick recap: EXPLAIN

- EXPLAIN shows the plan the optimizer selects in order to execute the query
- Takes the form of a json document containing a tree of operators

```
"plan": {
    "#operator": "Sequence",
    "~children": [
        {
            "#operator": "PrimaryScan",
            "index": "def_primary",
            "keyspace": "travel-sample",
            "namespace": "default",
            "using": "gsi"
        }, …
    ]
},
"text": "select * from `travel-sample`;”
```

69

Profiles

- We can collect execution timings and document processed on a per operator basis
- If the functionality is turned on, timings are reported with the metrics at the end of execution
  - in system:active_requests
  - in system:completed_requests
- Profiling is turned on at the request level via the “profile” REST API parameter
  - Eg from cbq:

```
\set -profile timings;
```

  - at the node level via the “profile” command line parameter or admin settings REST API parameter
  - takes 3 values, “off”, “phases”, “timings”
    - “phases” supplies total times for each operator class
    - “timings” supplies detailed information for each operator

70

Profile example

```
cbq> select * from `travel-sample` where source-airport is not missing;
…
"executionTimings": {
                "~children": [
                             {
                                 "#operator": "IndexScan2",
                                 "#stats": {
                                     "#itemsOut": 24024,
                                     "#phaseSwitches": 96099,
                                     "execTime": "55.370283ms",
                                     "kernTime": "5.397199311s"
                                 },
                                 "index": "def_sourceairport",
                                 "index_id": "29702e564c9d2ca4",
                                 "index_projection": {
                                     "primary_key": true
                                 },
                                 "keyspace": "travel-sample",
                                 "namespace": "default",
                                 "spans": [
                                     {
                                         "exact": true,
                                         "range": [
                                             {
                                                 "inclusion": 1,
                                                 "low": "null"
                                             }
```

Profiles usage

- For each operator, profiles report
  - Ingested documents
  - Produced documents
  - Execution time
  - Time spent waiting on services (indexing / datastore)
  - Time spent waiting to be scheduled
  - Execution phase switches.
- These provide very good insight on individual operator costs:
  - High incoming documents and low outgoing documents for filters indicate a poor index choice
  - High service times and low incoming documents indicate under stress services
  - High kernel times indicate a bottleneck down stream or a stressed n1ql service

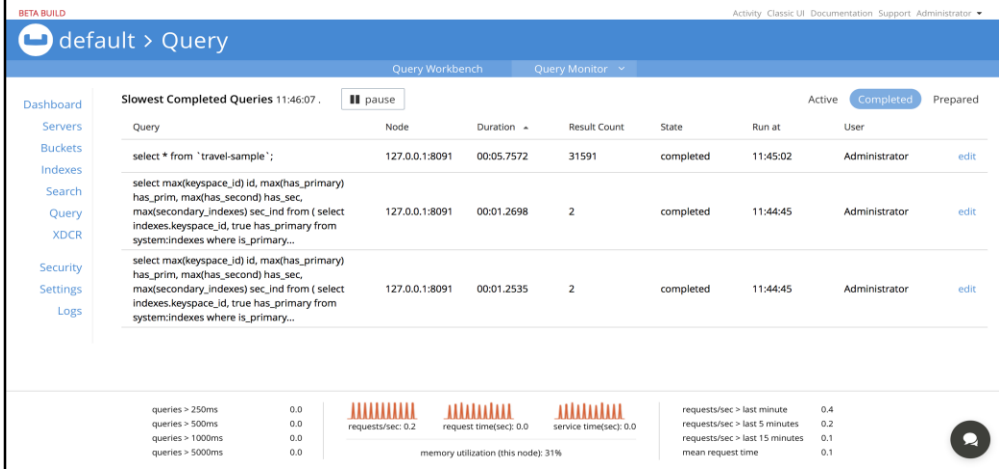
Controls

- Many times the request profile is dependent on placeholders
- For this reason, placeholder values used in specific requests can be attached to the request results, active_requests and completed_requests
- “controls” command line and REST API parameters used to control the functionality
- boolean
- Use is similar to “profile” command line and REST API parameters

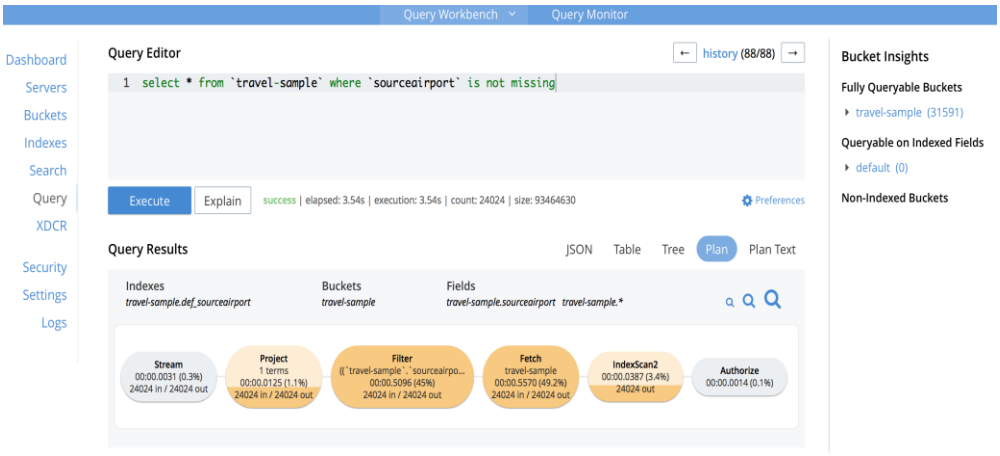
Query Work Bench

- You don’t need to use N1QL directly to access monitoring and profiling information
- QWB has a new monitoring tab that shows system keyspaces contents
- QWB will also show graphical representation of execution trees

Monitoring in UI



Profiling in UI

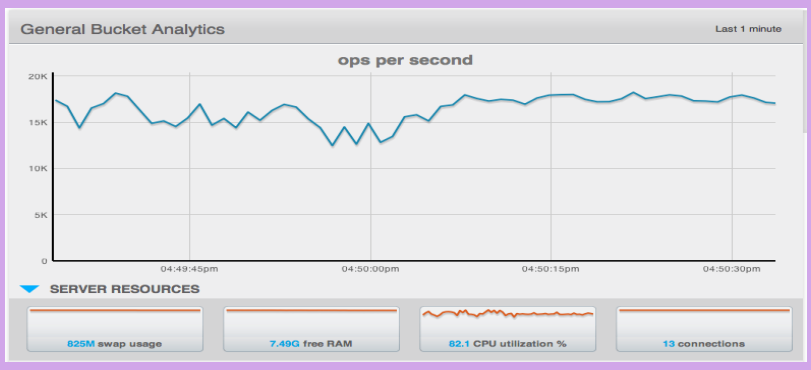


Performance

Aggregate versus per-server stats

Aggregate versus per-server stats

Server Resources



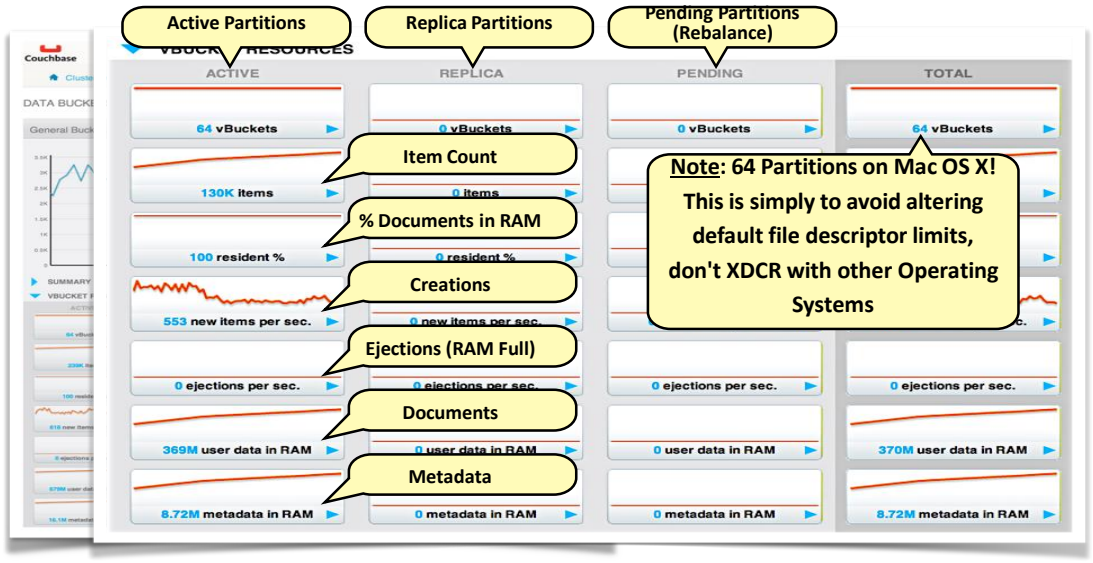
- **Free RAM**
- **CPU**
- **Swap usage**
- **Connections**

Store and Retrieve Operations

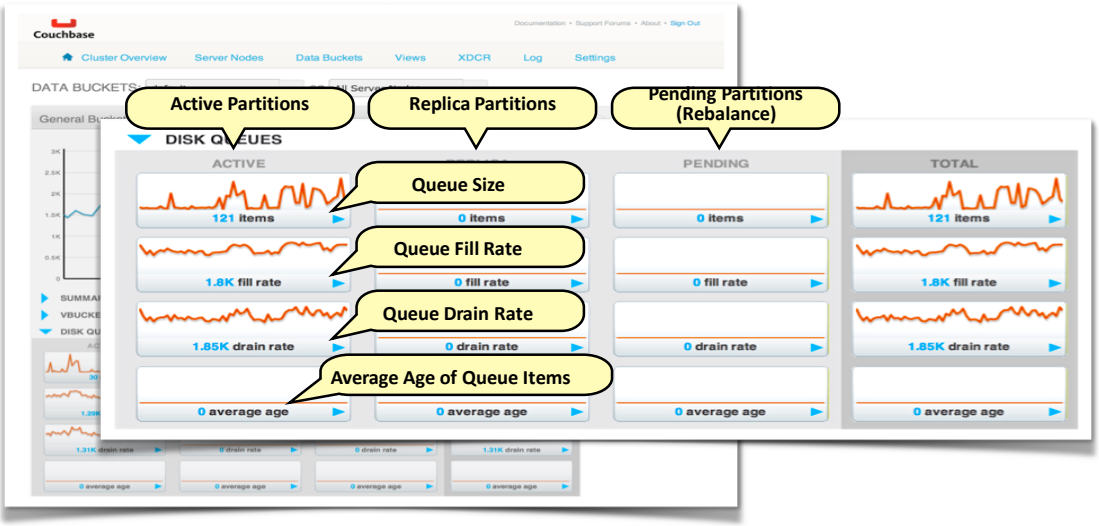
Store and Retrieve Operations

Store and Retrieve Operations

Partition (vbucket) Details



Disk IO Information



Bucket monitoring — summary statistics

86

Bucket monitoring — summary statistics

Perry blog: "*disk write queue*": This is **the** metric for understanding whether there is sufficient disk IO on a node. While there are many processes that contend for disk IO (data writing, compaction, views, XDCR, local backups, etc), we use the "*disk write queue*" as a general meter as insufficient IO will cause items to be written to disk slower. Anything approaching 1M items per-bucket per-node should be cause for concern, though many applications expect it to be much lower for their workload. This should be expected to be higher during a rebalance."

Perry blog: ""*temp OOM per sec.*" is a measure of how many write operations are failing due to an "out of memory" situation within the node/bucket. It will only occur if "memory used" reaches 90% of the total bucket quota. "

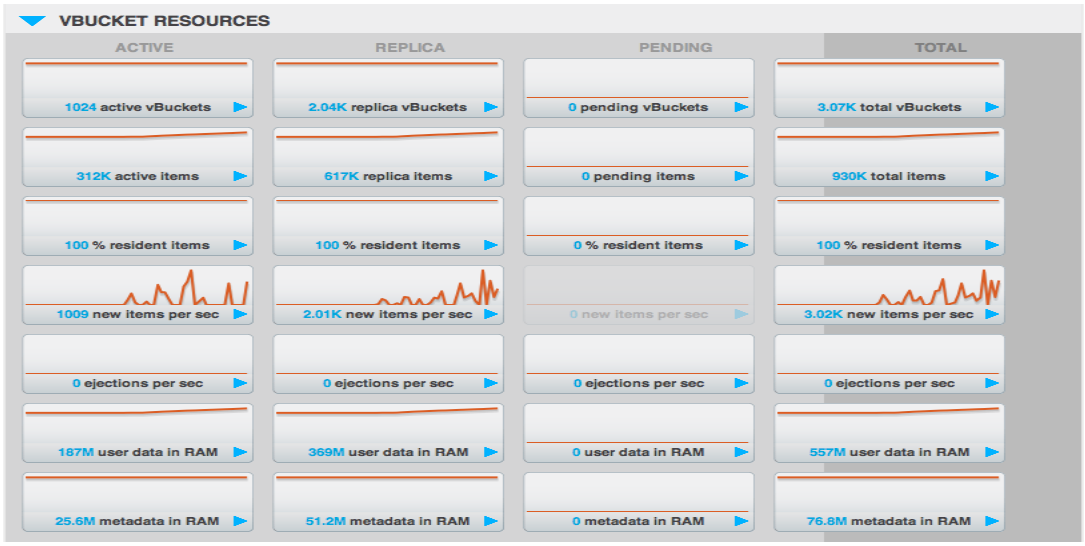
Perry blog: ""cache miss ratio": This is a percentage of the number of reads being served from disk as opposed to from RAM. A value of 0 means all reads are coming from RAM, while anything higher than that indicates some reads are coming from disk.
For applications that expect everything to be served from RAM, this should always be

0.
For applications that expect this to be non-0, it should ideally be as low as possible, most deployments are under 1% but some accept upwards of 10%. SSD's versus spinning disks have a big effect on what is a reasonable value."

The following graph types are available:

ops per second- The total number of operations per second on this bucket.
cache miss ratio- Ratio of reads per second to this bucket which required a read from disk rather than RAM.
creates per second- Number of new items created in this bucket per second.
updates per second- Number of existing items updated in this bucket per second.
XDCR ops per sec- Number of XDCR related operations per second for this bucket.
disk reads per sec- Number of reads per second from disk for this bucket.
temp OOM per sec- Number of temporary out of memory conditions per second.
gets per second- Number of get operations per second.
sets per second- Number of set operations per second.
deletes per second- Number of delete operations per second.
Items- Number of items (documents) stored in the bucket.
disk write queue- Size of the disk write queue.
docs data size- Size of the stored document data.
docs total disk size- Size of the persisted stored document data on disk.
doc fragmentation %- Document fragmentation of persisted data as stored on disk.
XDC replication queue- Size of the XDCR replication queue.
total disk size- Total size of the information for this bucket as stored on disk, including persisted and view index data.
views data size- Size of the view data information.
views total disk size- Size of the view index information as stored on disk.
views fragmentation %- Percentage of fragmentation for a given view index.
view reads per second- Number of view reads per second.
memory used- Amount of memory used for storing the information in this bucket.
high water mark- High water mark for this bucket (based on the configured bucket RAM quota).
low water mark- Low water mark for this bucket (based on the configured bucket RAM quota).
disk update time- Time required to update data on disk.

monitoring — summary statistics



87

Perry blog: ""active docs resident %": This is the percentage of items currently cached in RAM. 100% means all items are cached in RAM while anything less than that indicates some items have been ejected.
Some applications expect that this is always 100% and will alert if it goes below. Other applications expect this to be something lower than 100%, but the actual value is up to you. I would generally recommend not going below 30% unless your application is quite comfortable with it."

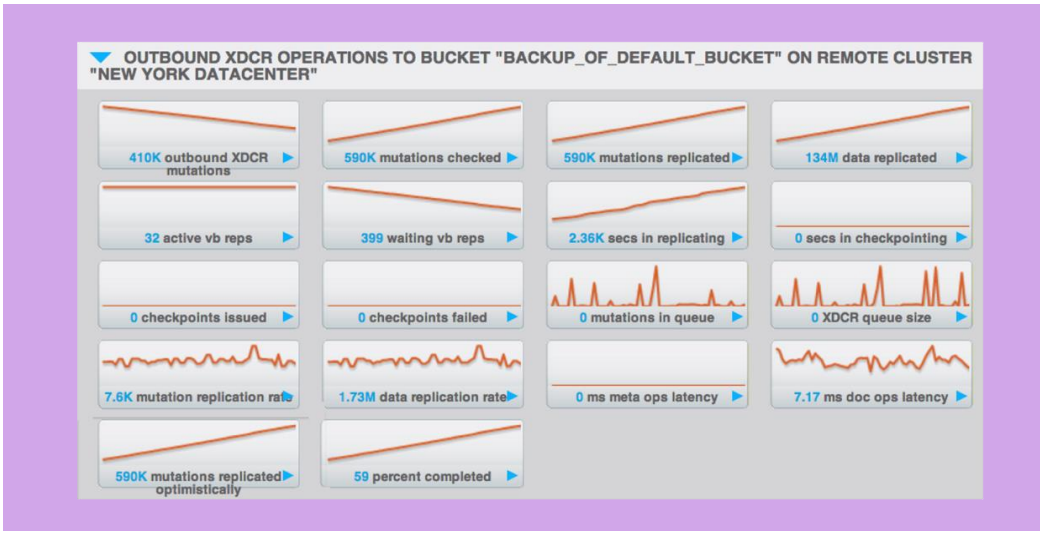
The vBucket statistics provide information for all vBucket types within the cluster across three different states. Within the statistic display the table of statistics is organized in four columns, showing the Active, Replica and Pending states for each individual statistic. The final column provides the total value for each statistic.

The Active column displays the information for vBuckets within the Active state. The Replica column displays the statistics for vBuckets within the Replica state (i.e. currently being replicated). The Pending columns shows statistics for vBuckets in the Pending state, i.e. while data is being exchanged during rebalancing.

The individual statistics, one for each state, shown are:

vBuckets- The number of vBuckets within the specified state.
Items- Number of items within the vBucket of the specified state.
resident %- Percentage of items within the vBuckets of the specified state that are resident (in RAM).
new items per sec.- Number of new items created in vBuckets within the specified state. Note that new items per second is not valid for the Pending state.
ejections per second- Number of items ejected per second within the vBuckets of the specified state.
user data in RAM- Size of user data within vBuckets of the specified state that are resident in RAM.
metadata in RAM- Size of item metadata within the vBuckets of the specified state that are resident in RAM.

monitoring — summary statistics



88

Perry blog: "If you're using XDCR with Couchbase, the most important metric to keep an eye on is the XDCR mutation queue - "*outbound XDCR mutations*". This is an indication of how many items are waiting to be replicated to buckets that are acting as destinations of this one. Like the disk write queue, this is expected to grow and shrink under load but is important to ensure eventually gets near zero over time and does not continuously grow higher and higher."

The statistics shown are:

outbound XDCR mutation
Number of changes in the queue waiting to be sent to the destination cluster.

mutations checked
Number of document mutations checked on source cluster.

mutations replicated
Number of document mutations replicated to the destination cluster.

data replicated

Size of data replicated in bytes.

active vb reps
Number of parallel, active vBucket replicators. Each vBucket has one replicator which can be active or waiting. By default you can only have 32 parallel active replicators at once per node. Once an active replicator finishes, it will pass a token to a waiting replicator.

waiting vb reps
Number of vBucket replicators that are waiting for a token to replicate.

secs in replicating
Total seconds elapsed for data replication for all vBuckets in a cluster.

secs in checkpointing
Time working in seconds including wait time for replication.

checkpoints issued
Total number of checkpoints issued in replication queue. By default active vBucket replicators issue a checkpoint every 30 minutes to keep track of replication progress.

checkpoints failed
Number of checkpoints failed during replication. This can happen due to timeouts, due to network issues or if a destination cluster cannot persist quickly enough.

mutations in queue
Number of document mutations waiting in replication queue.

XDCR queue size
Amount of memory used by mutations waiting in replication queue. In bytes.

mutation replication rate
Number of mutations replicated to destination cluster per second.

data replication rate
Bytes replicated to destination per second.

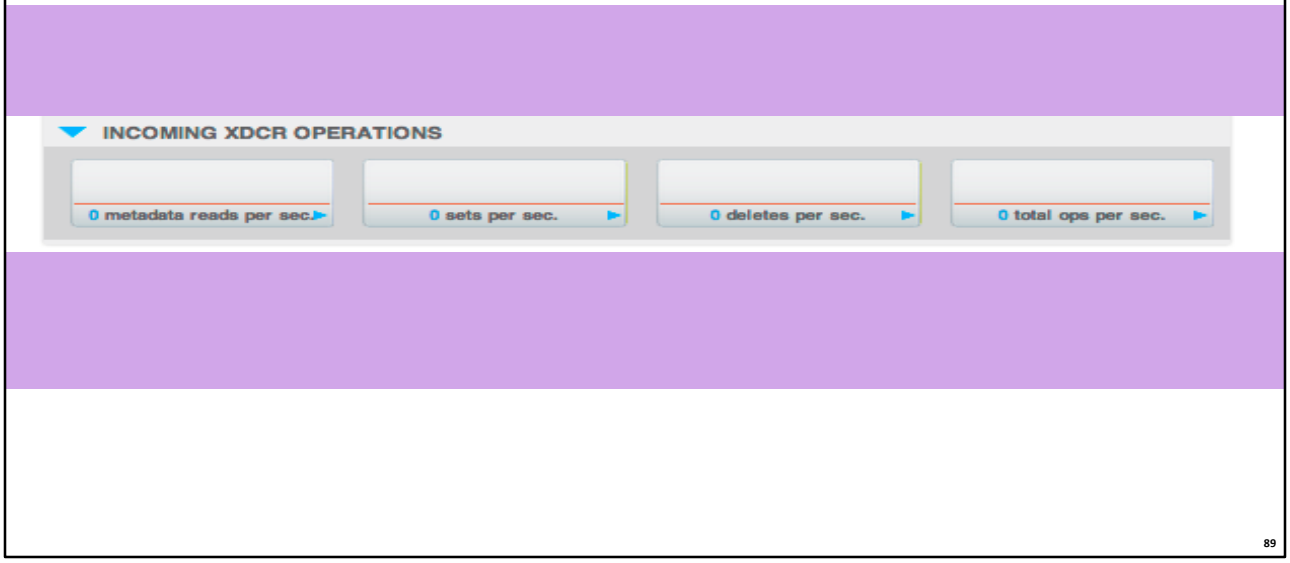
ms meta ops latency
Weighted average time for requesting document metadata. In milliseconds.

mutations replicated optimistically
Total number of mutations replicated with optimistic XDCR.

ms docs ops latency
Weighted average time for sending mutations to destination cluster. In milliseconds.

percent completed
Percent of total mutations checked for metadata.

monitoring — summary statistics



The statistics shown are:

metadata reads per sec.
Number of documents XDCR scans for metadata per second. XDCR uses this information for conflict resolution.

sets per sec.
Set operations per second for incoming XDCR data.

deletes per sec.
Delete operations per second as a result of the incoming XDCR data stream.

total ops per sec.
Total of all the operations per second.

Common tools used with couchbase

- Productivity, Basic Troubleshooting and Performance Testing

What are we covering here?

- Set of common tools that Couchbase uses internally and externally
  - Benchmark, Performance, Load Test
  - To identify and troubleshoot issues
  - Collect diagnostic information

Couchbase Client Command Line Utility - CBC

- Why?
  - Provides a utility for command line access to basic operations without writing a test client
    - Quick debugging
    - Great for scripting data operations against Couchbase
  - Working the needle in the haystack –
    - Useful for extracting individual large documents you cannot access via the the Couchbase Web Console 256 KB limit
    - Might be difficult to navigate through or filter those documents in the web console
- Reference:
  - http://docs.couchbase.com/sdk-api/couchbase-c-client-2.8.1/md_doc_cbc.html

```
$ cbc -h
cbc COMMAND [OPTIONS]
cbc help
cbc version
cbc cat KEYS ... [OPTIONS]
cbc create KEY -V VALUE [OPTIONS]
cbc create KEY [OPTIONS]
cbc cp FILES ... [OPTIONS]
cbc incr KEY [OPTIONS]
cbc decr KEY [OPTIONS]
cbc rm KEY [OPTIONS]
cbc hash KEY [OPTIONS]
cbc stats KEYS ... [OPTIONS]
cbc observe KEYS ... [OPTIONS]
cbc view VIEWPATH [OPTIONS]
cbc lock KEY [OPTIONS]
cbc unlock KEY CAS [OPTIONS]
cbc admin -P PASSWORD RESTAPI [OPTIONS]
cbc bucket-create -P PASSWORD NAME [OPTIONS]
cbc bucket-delete -P PASSWORD NAME [OPTIONS]
cbc bucket-flush NAME [OPTIONS]
cbc connstr SPEC
cbc n1ql QUERY ... [OPTIONS]
```

SubDoc API Inspection - cbc-subdoc

- **Why?**
  - **Interactively Inspect Document Using Subdocument API**
- **Reference:**
  - **http://docs.couchbase.com/sdk-api/couchbase-c-client-2.8.1/md_doc_cbc-subdoc.html**
- **Usage:**
  - **cbc-subdoc [*OPTIONS*]**

Load / Stress Testing – cbworkloadgen and cbc-pillowfight

- **Why?**
  - **Benchmark / Performance Test your new deployment to make sure your environment is healthy**
  - **Validate SLA/Response times**
- **Reference:**
  - **cbworkloadgen:** https://developer.couchbase.com/documentation/server/current/cli/cbworkloadgen-tool.html
  - **cbc-pillowfight:** http://docs.couchbase.com/sdk-api/couchbase-c-client-2.4.8/md_doc_cbc-pillowfight.html
- **Usage:**
  - **cbworkloadgen -n [host]:8091 -u [username] -p [password] [options]**
  - **cbc-pillowfight [*OPTIONS*]**

Cbc-pillowfight: needs installation of the c sdk

N1QL Performance Testing – cbc-n1qlback

- **Why?**
  - **Load Test N1QL**
- **Reference:**
  - **http://docs.couchbase.com/sdk-api/couchbase-c-client-2.8.1/md_doc_cbc-n1qlback.html**
- **Usage:**
  - **cbc-n1qlback -f QUERYFILE [*OPTIONS*]**

Data Generation – FakeIt (Community)

- **Why?**
  - Proof of Concepts
  - Sizing data sets
  - Performance and Load Testing
  - Comprehensive tool with declarative YAML definitions along with object dependencies
  - Includes Data Load Capability, enhanced to support Couchbase Mobile Sync Gateway
- **Reference:**
  - https://blog.couchbase.com/fakeit-series-generating-fake-data/
  - https://connect.couchbase.com/eu/connect-europe-2017 (Rapid Data Modeling and Testing)

Server Performance (Couchbase Statistics) - cbstats

- Why?
  - Retrieve lower level statistics with the cbstats utility, which provides deep insight into what occurs within a cluster.
  - Review a comprehensive list of couchbase statistics
  - Rule out server side performance issues / investigate root cause of issues
- Reference:
  - https://developer.couchbase.com/documentation/server/current/cli/cbstats-intro.html

```
$ /opt/couchbase/bin/cbstats

Usage: cbstats host[:dataport] command [options]

Commands:
    all
    allocator
    checkpoint [vbid]
    config
    dcp
    dcp-vbtakeover vb name
    dcpagg
    diskinfo [detail]
    dispatcher [logs]
    failovers [vbid]
    hash [detail]
    items (memcached bucket only)
    key keyname vbid
    kvstore
    kvtimings
    memory
    prev-vbucket
    raw argument
    reset
    responses [all]
    runtimes
    scheduler
    slabs (memcached bucket only)
    tap
    tap-vbtakeover vb name
```

Server Performance (KV / Memcached) - mctimings

- **Why?**
  - **The mctimings tool displays internal timing information for all operations since the last time Couchbase was restarted or stats were reset.**
  - **Identify or rule out server side performance issues**
- **Reference:**
  - https://developer.couchbase.com/documentation/server/current/cli/mctimings.html
- **Usage**
  - **mctimings [options] [opcode / stat_name]***

```
$ /opt/couchbase/bin/mctimings

Usage mctimings [-h host[:port]] [-p port] [-u user] [-P pass]
        [-S passFromStdin] [-b bucket] [-s]-v [opcode / stat_name]*

The following data is collected for "SET"
[   0 -    9]us (  0.01%)     1 |
[  10 -   19]us (  0.73%)    76 |
[  20 -   29]us ( 15.70%)  1574 | #######
[  30 -   39]us ( 59.49%)  4606 | ######################
[  40 -   49]us ( 85.08%)  2692 | #############
[  50 -   59]us ( 91.73%)   699 | ####
[  60 -   69]us ( 94.69%)   312 | ##
[  70 -   79]us ( 96.14%)   152 | #
[  80 -   89]us ( 97.39%)   131 | #
[  90 -   99]us ( 98.13%)    78 |
[ 100 -  109]us ( 98.62%)    52 |
[ 110 -  119]us ( 98.84%)    23 |
[ 120 -  129]us ( 98.97%)    14 |
[ 130 -  139]us ( 99.13%)    16 |
[ 140 -  149]us ( 99.15%)     3 |
[ 150 -  159]us ( 99.23%)     8 |
[ 160 -  169]us ( 99.30%)     7 |
[ 170 -  179]us ( 99.34%)     5 |
[ 180 -  189]us ( 99.35%)     1 |
[ 190 -  199]us ( 99.38%)     3 |
[ 200 -  209]us ( 99.42%)     4 |
[ 210 -  219]us ( 99.44%)     2 |
[ 220 -  229]us ( 99.47%)     3 |
[ 250 -  259]us ( 99.52%)     5 |
```

Query Performance – system:completed_requests

- **Why?**
  - **To understand top "N" problematic queries**
  - **Understand what's not optimal about the queries**
- **Reference:**
  - https://developer.couchbase.com/documentation/server/current/tools/query-monitoring.html
- **Usage:**
  - **SELECT * from system:completed_requests**

```
select *, meta().plan from system:completed_requests;
[
 {
  "completed_requests": {
  "elapsedTime": "2.18228764s",
   "errorCount": 0,
   "node": "192.168.1.7",
   "phaseCounts": {
    "fetch": 23568,
    "primaryScan": 24139
   },
   "phaseTimes": {
    "authorize": "1.26817ms",
    "fetch": "758.740363ms",
    "instantiate": "55.679µs",
    "parse": "713.903µs",
    "plan": "489.134µs",
    "primaryScan": "245.557631ms"
   },
   "resultCount": 194,
   "resultSize": 60651,
   "scanConsistency": "unbounded",
   "statement": "select * from `travel-sample where type="airline"`",
   "time": "2017-02-27 11:02:59.008480274 -0800 PST"
  },
```

Client Logging – SDK Debug / Trace Logs

- Why ?
  - Various issues could exist on the client side everything from DNS resolution, to intermittent network connectivity to poorly written client code
  - Client side SDK logs provide the “client’s view” of connections , operations and response times
- Reference:
  - Java: https://developer.couchbase.com/documentation/server/current/sdk/java/collecting-information-and-logging.html
  - .NET: https://developer.couchbase.com/documentation/server/current/sdk/dotnet/collecting-information-and-logging.html
  - C: https://developer.couchbase.com/documentation/server/current/sdk/c/collecting-information-and-logging.html
  - Other SDKs can be referenced directly from any one of these pages

Where is this key? vBucketTool / cbc-hash

- Why ?
  - A subset of KV document access is timing out (document counters, specific document)
  - Isolate whether keys are located on a specific vBucket or specific server which could be the root cause
- Reference:
  - /opt/couchbase/bin/tools/vbbuckettool –h
  - man cbc
- Usage:
  - curl http://localhost:8091/pools/default/buckets/<bucketname> -u Administrator:password|./vbuckettool - MyDocId
  - cbc hash MyDocId

```
$ cbc hash MyDocId

MyDocId: [vBucket=862,
Index=2]
Server: 192.168.61.12:11210,
Replica #0: Index=1,
Host=192.168.61.22:11210
```

Connectivity Issues – cbping / sdkdoctor (Community)

- Why ?
  - Connectivity, firewall, latency issues between client and server
  - Some operations/services work while others don't
  - Review timeouts on virtualized, containerized platforms
- Reference:
  - https://blog.couchbase.com/cbping-should-be-your-new-friend/
  - https://github.com/couchbasebrian/cbping
  - https://github.com/couchbaselabs/sdk-doctor

```
$ python cbping.py -H 172.23.99.170 -P 8091 \
      -u Administrator -p password -c -rc

I will connect to: http://172.23.99.170:8091/pools/default and
run some tests.
This node says that there are 2 nodes in the cluster.

Cluster Node           Node Status     Node CB version
------------           -----------     ---------------
172.23.99.170:8091     healthy         4.1.1-5914-enterprise
172.23.99.171:8091     healthy         4.1.1-5914-enterprise

Hostname             Port       Result         Elapsed Time*
--------             ----       ------         ------------
172.23.99.171        8091       SUCCESS        107.0
172.23.99.171        8092       SUCCESS        111.0
172.23.99.171        8093       SUCCESS        106.0
```

TCP Dumps / Wireshark

- **Why?**
  - **Debug network problems (timeouts, flaky connectivity, virtualization, starvation of resources, packet loss etc.)**
- **Reference:**
  - https://www.wireshark.org/docs/dfref/c/couchbase.html
  - https://www.wireshark.org/download.html
- **Usage:**
  - **First collect a tcpdump (*nix) or WinDump (https://www.winpcap.org/windump/) or TCPDump for Windows (https://www.microolap.com/products/network/tcpdump/)**
  - **Copy the files over to your machine**
  - **Review the dumps using wireshark or make them available to Couchbase Support if requested**

```
$ tcpdump -C 500 -W 10 -w
/path/to/save/file/(filename).pcap -s 0 port
11210

-C 500 = 500MB file size
-W 10 = make 10 files then rotate from
beginning  (example.pcap00, example.pcap01)
-w /path/to/save/file/example.pcap = where
you want the file stored & name of file
-s 0 = capture the whole packet
-i eth0 = the network interface that
Couchbase data is sent.
port 11210  = capture packets from/to port
11210 "Couchbase Data Port" (You can specify
additional ports as requested 8091,8092
etc.)
```

Sharing files with Support or your technical contact

- **Why?**
  - **Logs , backups, reference data are often large**
  - **Need a secure mechanism to share data**
- **Reference:**
  - **https://www.couchbase.com/support/working-with-technical-support**
- **Usage:**
  - **curl --upload-file <FILE-NAME> https://s3.amazonaws.com/customers.couchbase.com/<YOUR-COMPANY-NAME>/**

Couchbase and the Cloud – GitHub Repo Couchbase-Partners

- **Why?**
  - **Readily available templates to quickly get started with the cloud – Amazon, Azure, Google Cloud**
  - **Pre-configured with most tuneables**
  - **Documented best-practices**
  - **Couchbase Mobile and Couchbase Server**
- **References:**
  - https://github.com/couchbase-partners
  - https://github.com/couchbase-partners/amazon-cloud-formation-couchbase
  - https://github.com/couchbase-partners/azure-resource-manager-couchbase
  - https://github.com/couchbase-partners/google-deployment-manager-couchbase

Stats file location

/opt/couchbase/var/lib/couchbase/stats

ls –alh

file *

106

The default stats file location is /opt/couchbase/var/lib/couchbase/stats, however, if you want to change the default stats file location, create a symlink location to the new directory.

Note
When creating a symlink, stop and restart the Couchbase service.

Monitoring Scripts: cbstats

```
Usage: cbstats [options] Monitoring Scripts: cbstats
Options:
  -h, --help     show this help message and exit
  -a             iterate over all buckets (requires admin u/p)
  -b BUCKETNAME  the bucket to get stats from (Default: default)
  -p PASSWORD    the password for the bucket if one exists
Usage: cbstats host:port all
  or   cbstats host:port allocator
  or   cbstats host:port checkpoint [vbid]
  or   cbstats host:port config
  or   cbstats host:port dispatcher [logs]
  or   cbstats host:port hash [detail]
  or   cbstats host:port items (memcached bucket only)
  or   cbstats host:port key keyname vbid
  or   cbstats host:port klog
  or   cbstats host:port kvstore
  or   cbstats host:port kvtimings
  or   cbstats host:port memory
  or   cbstats host:port prev-vbucket
  or   cbstats host:port raw argument
  or   cbstats host:port reset
  or   cbstats host:port slabs (memcached bucket only)
  or   cbstats host:port tap
  or   cbstats host:port tapagg
  or   cbstats host:port timings
  or   cbstats host:port vbucket
  or   cbstats host:port vbucket-details
  or   cbstats host:port vkey keyname vbid
  or   cbstats host:port warmup
```

Commonly use "all", "dispatcher", "tap", "timings" and "reset"
Exhaustive list of stats available, full documentation:
http://www.couchbase.com/docs/couchbase-manual-2.0/couchbase-monitoring-nodestats.html

To-do update link to 2.5

http://www.couchbase.com/docs/couchbase-manual-2.0/couchbase-monitoring-nodestats.html

cbstats ‘all’

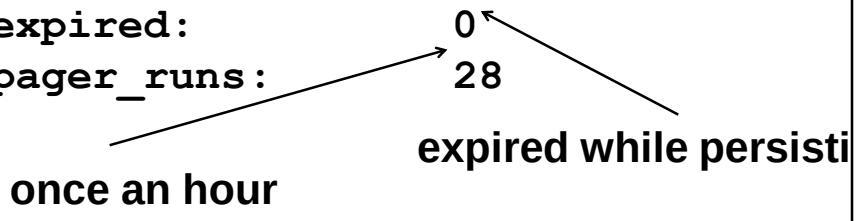
- ‘cbstats all’ is the most useful/common output:
  /opt/couchbase/bin/cbstats localhost:11210 all
- Equivalent to ‘stats’ in the standard memcached protocol (includes many memcached stats as well)
- Many of the UI stats come from this output
- Easiest to know what you’re looking for and ‘grep’ for a specific stat (or group):
-

```
/opt/couchbase/bin/cbstats localhost:11210 all | grep curr_items
curr_items:                     4100025
curr_items_tot:                 4100025
vb_active_curr_items:           4100025
vb_pending_curr_items:          0
vb_replica_curr_items:          0
```

Monitoring: expiration

```
/opt/couchbase/bin/cbstats localhost:11210 all
| grep exp

 ep_expired:                          0
 ep_item_flush_expired:               0
 ep_num_expiry_pager_runs:            28
```



cbstats 'timings'

```
/opt/couchbase/bin/cbstats localhost:11210 -b beer-sample timings
 data_age (1 total)
    0 - 1s         : (100.00%) 1 #############
 disk_commit (3 total)
    0 - 1s         : (100.00%) 3 ################
 disk_insert (1 total)
    64us - 128us   : (100.00%) 1 ###############
 disk_invalid_item_del (1 total)
    2us - 4us      : (100.00%) 1 #################
 get_cmd (1 total)
    64us - 128us   : (100.00%) 1 ####################
 set_vb_cmd (1024 total)
    32us - 64us    : (  0.59%)   6 #
    64us - 128us   : ( 91.70%) 933 #################
    128us - 256us  : ( 99.80%)  83 ##############
    256us - 512us  : ( 99.90%)   1
    1ms - 2ms      : (100.00%)   1
 storage_age (1 total)
    0 - 1s         : (100.00%) 1 ######################
 store_cmd (1 total)
    128us - 256us  : (100.00%) 1 ##########################
```

Displays histogram of time taken by various operations within server

3 underlying server processes

beam.smp (on linux) - responsible for monitoring and managing all other underlying server processes such as ongoing XDCR replications, cluster operations, and views.

There is a separate monitoring process running on each node. The process is small and simple and therefore unlikely to crash due to lack of memory. It is responsible for spawning and monitoring the second, larger process for cluster management, XDCR and views. It also spawns and monitors the processes for Moxi and memcached. If any of these three processes fail, the monitoring process will re-spawn them.

Memcached - This process is responsible for caching items in RAM and persisting them to disk.

Moxi - This process enables third-party memcached clients to connect to the server.

ps –elf pgrep -lf

ps –eLf pgrep -wlf

111

Add the windows process name

Disk write queue

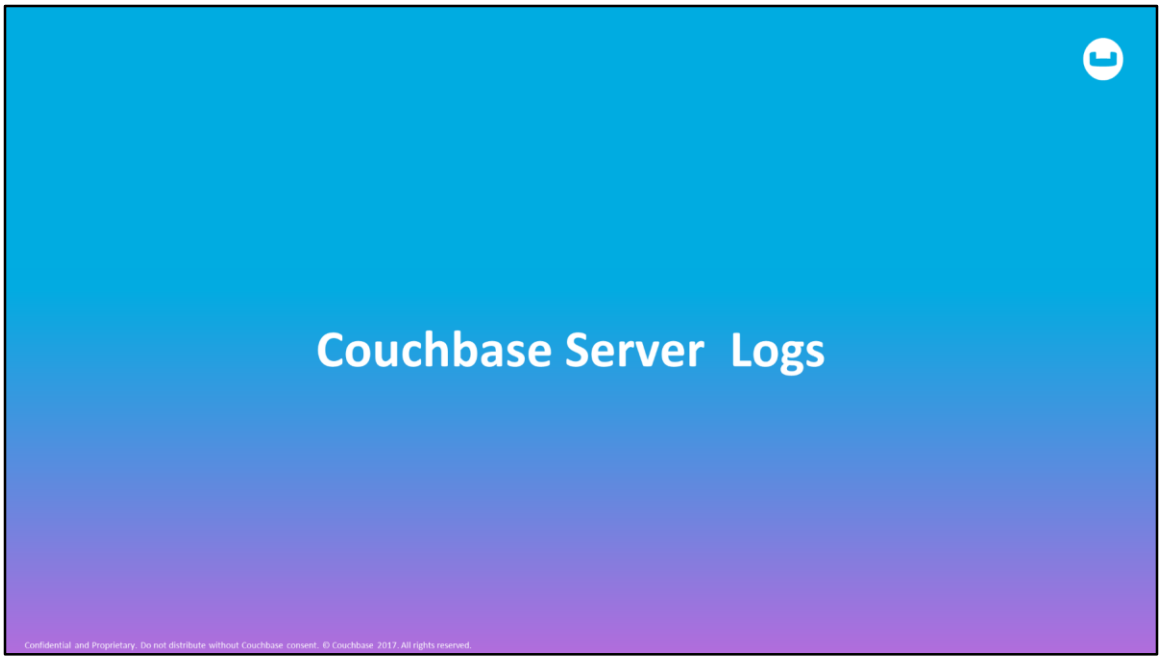
- **Disk writing is implemented as a 2-queue system and they are tracked by the stats**
- **The first queue is where mutations are immediately placed**
- **Whenever there are items in that queue, the"flusher" (disk writer) comes along and takes all the items off of that queue, places them into the other one and begins writing to disk**
- **Since disk performance is so dramatically different than RAM, this allows Couchbase to continue accepting new writes while it is (possibly slowly) writing new ones to the disk**
- **The flusher will process 250k items at a time, then perform a disk commit and continue this cycle until its queue is drained.**
- **ep_queue_size (where new mutations are placed)**
- **flusher_todo (the queue of items currently being written to disk)**

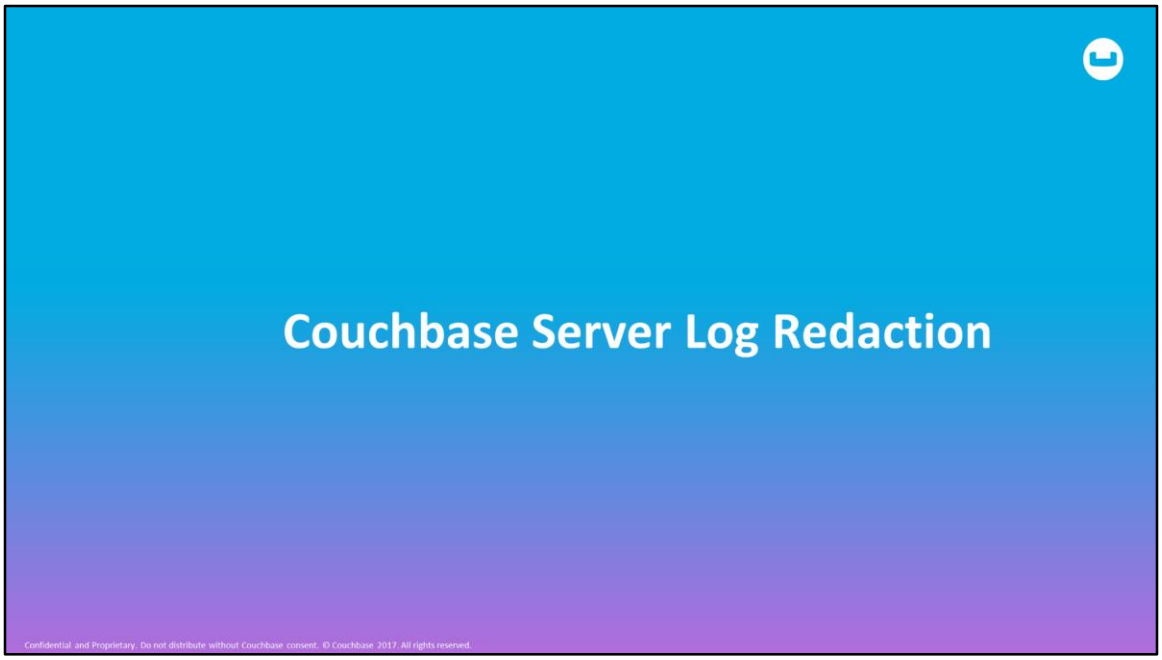
112

Point out Disk Write Queue in statistics

Troubleshooting Couchbase

- In general, Couchbase "just works":
  - No periodic maintenance for performance or stability
  - Very little configuration changing after setup
  - Monitor for appropriate sizing
- Most issues come from:
  - Initial setup errors
  - Under-provisioning
  - Bugs (yes, we have them occasionally)
- Hardest is troubleshooting "slowness":
  - Almost always related to RAM, network or disk & CPU(core)


Couchbase Server Logs
Confidential and Proprietary. Do not distribute without Couchbase consent. © Couchbase 2017. All rights reserved.



Feature Overview (1/2)

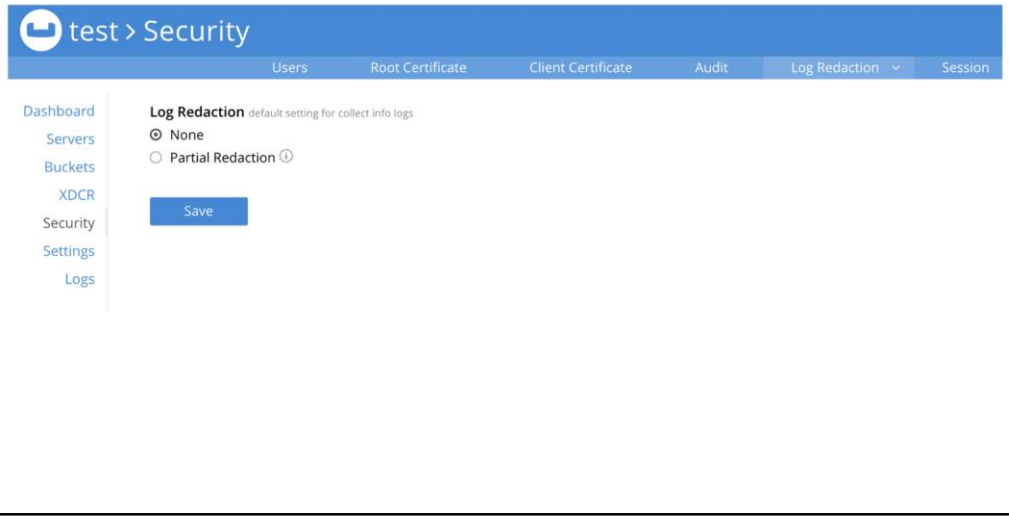
- Log redaction is the removal of sensitive or person identifying information from the log files that are transmitted to Couchbase
- Types of information
  - User data
    - Key and value pairs in JSON documents, or the key exclusively
    - Application/Admin usernames that identify the human person
    - Query statements included in the log file collected by support that leak the document fields (Select floor_price from stock).
    - Names and email addresses asked during product registration and alerting
    - Usernames
    - Document xattrs
  - System data (future work)
  - Metadata (future work)

Customers in regulated industries (FinTech, Healthcare) ask if there is a way to remove sensitive information (known as personally identifiable information PII) from the logs in compliance with HIPAA and other EU data protection policies
Forcing function: Article 17 *individuals have the Right to Erasure (Right to be Forgotten)*

Feature Overview (2/2)

- Redaction levels
  - None
    - No redaction
  - Partial
    - Sensitive information resides in the data itself, not the metadata or system data.
  - Full (future work)
    - Sensitive information could reside in user data, metadata and system data. All of this will be redacted from the logs.

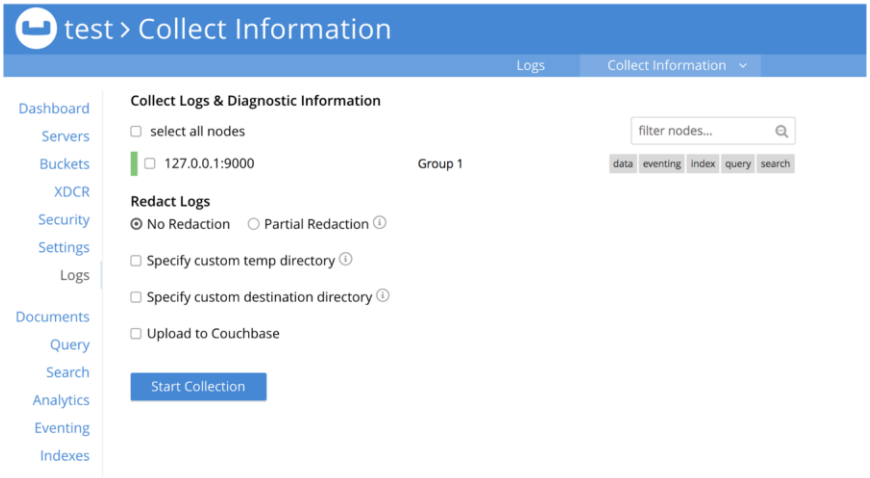
Cluster-wide configuration (UI)



Cluster-wide configuration (REST API)

```
$> curl -X POST
        -u Administrator:asdasd
        -d logRedactionLevel=partial
        http://localhost:8091/settings/logRedaction
```

Per log collection setting (UI)



Per log collection setting (REST API and CLI)

```
$> curl -X POST
        -u Administrator:asdasd
        -d logRedactionLevel=partial
        -d logRedactionSalt=Wl1XrXNuHWzr+ke7gnuUaKAtML6TFIynjhImeMYHEgI=
        -d nodes=*
        http://localhost:8091/controller/startLogsCollection
$> couchbase-cli/couchbase-cli collect-logs-start
        -c 127.0.0.1:9000
        -u Administrator -p asdasd
        --all-nodes
        --redaction-level partial
        --salt "foo"
```

How It Works

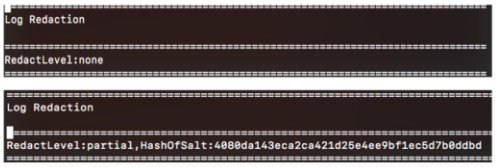
- Services identify and tag user data
  - Pre couchbase-server 5.5 log message
    - 2018-07-15T12:11:09.380-07:00 [INFO] Using plain authentication for user @cbq-engine
  - Couchbase-server 5.5 log message
    - 2018-07-15T12:11:09.380-07:00 [INFO] Using plain authentication for user <ud>@cbq-engine</ud>

- If log redaction is enabled during log collection
  - Two sets of logs are generated- one unredacted and one redacted
  - Regex is applied to each line of log file looking for the <ud>data</ud> tag
  - Content between the tags is replaced with hash(data + SALT*) in the redacted log files
  - If upload of collected logs is requested only the redacted files are uploaded

* **SALT: concept in hashing to protect against rainbow and dictionary attacks**

What Gets Logged

- Audit message example
- Tagged user data
  - Unredacted
    - 2018-07-14T23:52:57.458-07:00 [INFO] Using plain authentication for user <ud>@cbq-engine</ud>
  - Redacted
    - 2018-07-14T23:52:57.458-07:00 [INFO] Using plain authentication for user <ud>5fe70ca3f7d4896fc412366dd2a5119a335dff0d</ud>
- Identifying a redacted zip file
  - Filename: *.zip and *-redacted.zip
  - New couchbase.log message identifying redaction level and salt



Troubleshooting Couchbase

Logs	and	Stats
• Very verbose • Server startup • Process exit/crash diagnosis • Config changes • Timing events		• Performance monitoring • Traffic monitoring • ”External” monitoring • Real-time state

/opt/couchbase/var/lib/couchbase/logs

Logging-- Changing Log File Locations

As root or sudo and navigate to the directory where Couchbase Server is installed.

/opt/couchbase/etc/couchbase/static_config.

Edit the static_config file: change the error_logger_mf_dir variable, specifying a different directory.

For example: {error_logger_mf_dir, "/home/user/cb/opt/couchbase/var/lib/couchbase/logs"}

Stop and restart Couchbase Server.

Stored in text format under:
/opt/couchbase/var/lib/couchbase/logs/

Node-specific

Size-limited and automatically rotated
Some will be compressed .gz

UI contains a small subset of the log detail for easier viewing
UI logs are aggregated across cluster

Main logs are very verbose, but are still useful
Best for:
Process exit/crash diagnosis
Memcached
moxi
Vbucketmigrator
Server startup issues

Permissions
disk space
Search output for:
“ERROR”
“exited with”

Logs

File	Log Contents
audit	Security auditing for administrators.
babysitter	The babysitter process log captures when the larger processes are spawned for cluster management, cross data center replication, views, query, and N1QL.
couchdb	Errors relating to the CouchDB subsystem that supports views, indexes, and related REST API issues.
debug	Debug level error messages related to the core server management subsystem, excluding information included in the couchdb, xdcr and stats logs.
error	Error level messages for all subsystems excluding cross datacenter replication.
goxdcr	Log messages related to Cross Datacenter Replication (XDCR) written in Go.
http_access.log	The admin access log records server requests (including administrator logins) to the REST API or Couchbase Server web console. It is output in common log format and contains several important fields such as remote client IP, timestamp, GET/POST request and resource requested, HTTP status code, and so on.

126

Logs

File	Log Contents
info	Information level error messages related to the core server management subsystem, excluding information included in the CouchDB, cross datacenter replication, and stats logs.
mapreduce_errors	JavaScript and other view-processing errors are reported in this file.
memcached.log	Contains information relating to the core memcached component, including vBucket and replica and rebalance data streams requests.
ns-couchdb	Contains information related to starting up the CouchDB subsystem..
Projector	Contains information for the projector process that runs on each KV node and sends requested index data to the index nodes.
reports.log	Contains only progress report and crash reports for the Erlang processes.
ssl_proxy	

Logs

File	Log Contents
crash-log.bin	Used to pass service crash reports from the babysitter to the ns_server. For example, if the ns_server is available, any crash of the babysitter's child is passed directly to the special crash logger service within the ns_server. If the logger service is not attached to the babysiter, then the babysitter saves that crash report to the disk and the ns_server can later obtain and log it even if the babysitter is restarted.
stats	Contains periodic reports of the core statistics..
tmpfail	For XDCR, the destination cluster is not able to eject items fast enough to make room for new mutations. XDCR retries several times, without throwing errors, but after a fixed number of attempts the errors are shown to the user. Nevertheless, if a user waits long enough, XDCR eventually retries and can replicate the remaining data.
views	Errors relating to the integration of the view system and the core server subsystem
xdcr	Log messages related to starting up the cross datacenter replication subsystem..
xcdr_trace	Cross datacenter replication trace messages

Support Tool

```
/opt/couchbase/bin/cbcollect_info <output_file>
```

- Gathers:
  - **Logs**
  - **Couchbase stats snapshot**
  - **system level stats (netstat, df, top, etc)**

- <output_file> is zipped already
- Easy and small to send to Couchbase support

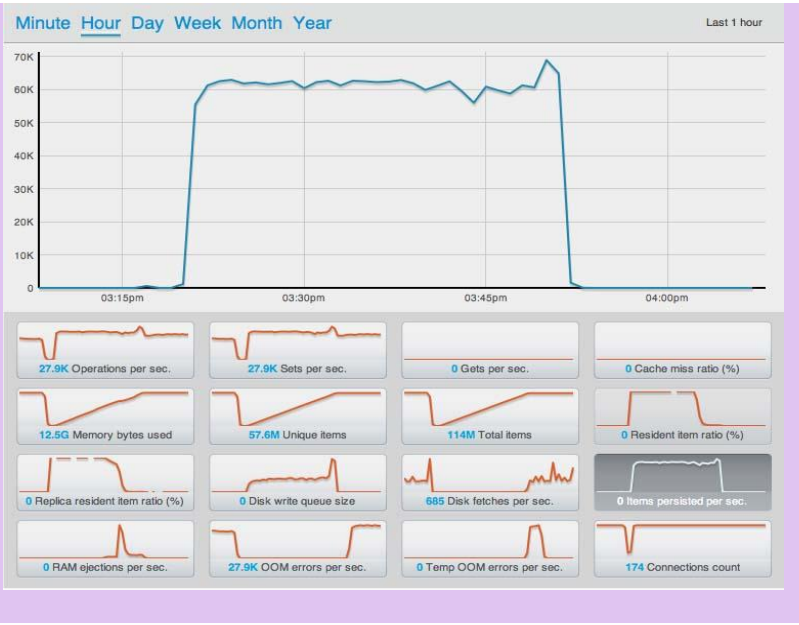
Log Collection – cbcollect-info

- Why ?
  - Couchbase logs provide Couchbase comprehensive information regarding what's going on with your cluster
  - Everything from configuration settings, OS limits, virtualization waits, performance statistics, process information
  - It's the first things Couchbase Support and Couchbase Center of Excellence will ask you to provide
- Reference:
  - https://developer.couchbase.com/documentation/server/current/cli/cbcollect-info-tool.html
  - https://www.couchbase.com/support/working-with-technical-support
- Usage:
  - sudo /opt/couchbase/bin/cbcollect_info <node_name><timestamp>.zip
  - e.g. sudo cbcollect_info `hostname`$(date +"%Y-%m-%d-%F").zip
  - Alternatively collect logs directly from the couchbase web console assuming your clusters have outbound web connectivity

Troubleshooting: Common Issues/Errors

- Installation/Setup:
  - **Firewall/AV**
  - **Permissions**
  - **ulimit**
  - **Conflicting software (hello puppet!)**
- Client connectivity:
  - **Firewall again**
  - **Use telnet to check connectivity**
  - **Permissions**
  - **Moxi**
- Loading/working with data:
  - **Check response codes**
  - **"temp oom" and "oom"**
  - **Timeouts**
  - **Disk IO (queues and fetches)**
  - **RAM ejections**
- Clustering:
  - **Firewall, again**
  - **Node versioning**
- XDCR:
  - **All nodes must talk to all other nodes**
- Indexing:
  - **Logs will show errors**
  - **Check for invalid document handling**
- Rebalancing:
  - **Network bandwidth**
  - **Disk IO**
  - **Performance during rebalancing**
  - **"wait_for_memcached" failed**
- Memory usage:
  - **OOM killer**
  - **Metadata**
- Disk Usage
  - **High update/disk writes don't allow compaction to complete**
  - **Auto-compaction window too short**
  - **Parallel compaction of data/views increases disk I/O**

Troubleshooting: Scenario



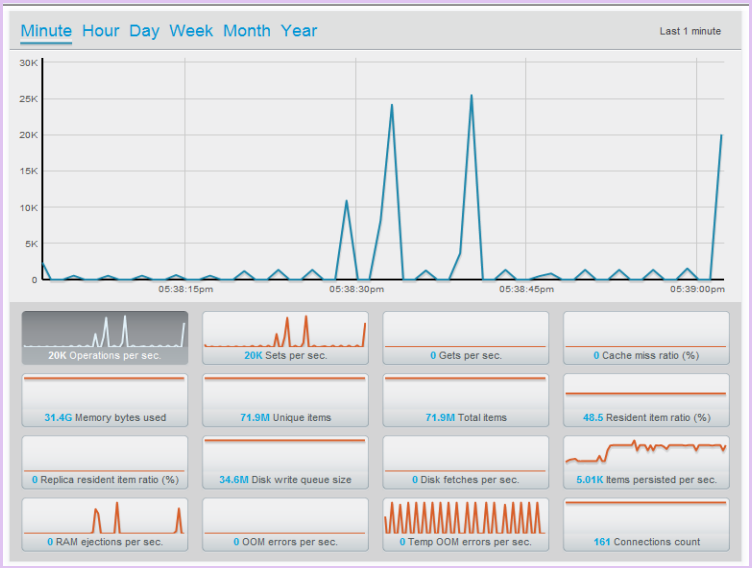
Problem: 'sets' are failin

Symptoms:

- **OOM Errors**
- **Resident Item Ratio**
- **0 Disk write queue**
- **0 RAM ejections**

Diagnosis:
RAM is full with metadata

Troubleshooting: Scenario



Problem: 'sets' are failing

Symptoms:

- **Temp OOM Errors**
- **Resident Item Ratio**
- **High Disk write queue**
- **Periodic RAM ejections**
- **Items being written**

Diagnosis:

- **"working" set is changing too rapidly**
- **data can't be written fast enough to be**

Best Practices Discussion

- **Missed Tunables**
  - THP set to Never
  - Vm.swapiness to Zero
  - Not disabling NUMA
- **Architecture / Deployment**
  - Shared Everything Architecture instead of Shared Nothing
  - Overcommitted VMs / Too Many Virtual Cores Assigned
  - vMotion Enabled
  - Inadequate IO for your needs
  - Co-located Services
  - Using Standard GSI / Forrest DB in production (Pre 5.0)
  - Disparate node configurations – CPU, Kernel, OS, CB versions
  - Deployment inefficiencies (Hops, Switches, Bandwidth)
  - Not Enough TMP space
- **Design**
  - Inefficient Document Model for use case (Referencing Vs Embedding / Document Size)
  - Deeply Nested Model
- **Development**
  - Outdated SDK
  - Missing Retry Logic
  - Not subscribing to Async Calls
  - Synchronous Batch Operations
  - Primary Indexes in Production
  - Improper / Inefficient indexes / queries without Pushdown
  - Not using Prepared Statements

Lab #8 (Performance & Compaction)

Time: 1 hou

- Understanding high water mark and low water mark
- Understand how ejection works
- Learn about the Not-Recently-Used metadata setting
- Learn about the item pager
- Become familiar with how Couchbase manages memory
- Learn about Disk reads vs RAM reads
- Examine that different traffic patterns require different Couchbase settings
- Understand the 'resident %' in memory metric
- Learn how to detect out of memory (OOM) errors
- Displaying metrics via cbstats and studying item expiration
- Display timing metrics with "cbstat timings"
- Learn about Compaction

Couchbase Learning Services