# Support Best Practices

Couchbase

NoEQUAL

# Notice and Disclaimer

# Support Best Practices

This section describes how to best interact with Couchbase Technical Support and how to work as your own internal support.

## Working with Support

To speed up the resolution of your issue, support will need some information to troubleshoot what is going on. The more information provided in the questions below the faster support will be able to identify the issue and propose a fix:

- Priority and impact of the issue (P1 and production impacting versus a P2 question)
- What versions of the software are you running,
- Operating system version and deployment (physical hardware, Amazon EC2, Kubernetes, etc.)
- What steps led to the failure or error?
- Information around whether this is something that has worked successfully in the past and if so what has changed in the environment since the last successful operation?
- Provide us with a current snapshot of logs taken from each node of the system and uploaded to our support system via the instructions below
- If your issue is a P1 and urgent, you must make a phone call as well as opening a support request. The phone call will ensure that your Initial Response Goal is met. Support phone number: +1-650-417-7500, option #1

We define our Support Request priorities in the following way:

**P1**: Software failures on a production system that cause complete loss or severe outage of service, resulting in a mission-critical business application being down or non-operational.

**P2**: Software failures on a production system that cause partial loss of service impacting business operations. Operations can continue in a restricted fashion and a workaround may be used to restore functionality. As to a non-production system, a software failure that causes a complete loss or severe outage of service where there is a time-sensitive impact to a planned production deployment.

**P3**: All non-time sensitive requests including product functionality issues in Development or Test, feature requests and documentation clarifications.

### SLAs

| | **Silver** | **Gold** | **Platinum** |
|---|---|---|---|
| Hours | 10 x 5 | 24 x 7 | 24 x 7 |
| Hours of Operation | 7am - 5pm* | 24 x 7 | 24 x 7 |
| Support Channel | Email, Web, Phone | Email, Web, Phone | Email, Web, Phone |
| Number of Cases | Unlimited | Unlimited | Unlimited |
| P1 SLA | 5 hours** | 2 hours | 30 minutes |

| | | | |
|---|---|---|---|
| P2 SLA | 1 Business Day | 5 hours | 3 hours |
| P3 SLA | 5 Business Days | 3 Business Days | 1 Business Day |

- *Within geo time zone, based on customer location: EST - Americas, East Coast; PST - Americas other than East Coast; GMT - EMEA; IST - APAC
- **Within business hours

## Opening a ticket

There are two ways to create a Couchbase Technical Support ticket: the web portal and a phone call.

## Web Portal

https://support.couchbase.com

## Phone Call

Support phone number: +1-650-417-7500, option #1

## Uploading logs

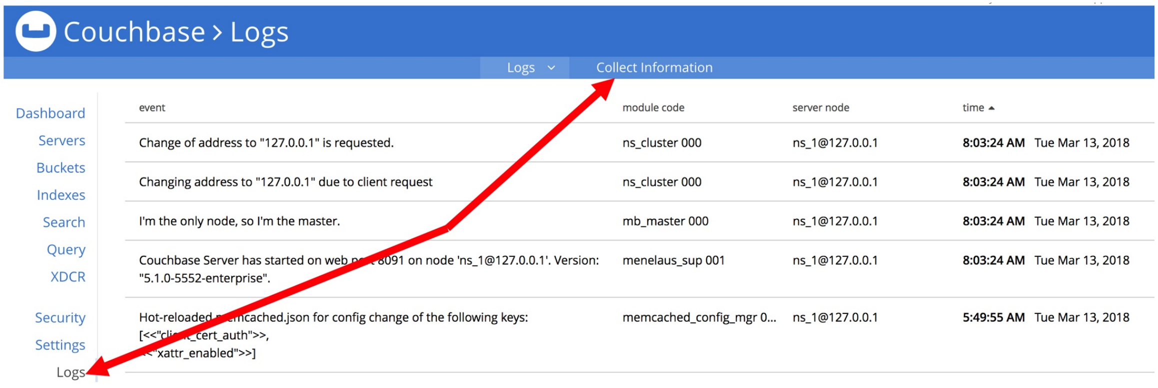
## Server logs

For nearly every ticket that gets generated, support will need to be able to review the logs for the Couchbase server cluster. Below is how to collect and upload those logs.

### Collecting logs via the web console UI

The logs can be generated in the web console UI:

1. Click **Log**
2. Click **Collect Information**



| event | module code | server node | time |
|---|---|---|---|
| Change of address to "127.0.0.1" is requested. | ns_cluster 000 | ns_1@127.0.0.1 | **8:03:24 AM** Tue Mar 13, 2018 |
| Changing address to "127.0.0.1" due to client request | ns_cluster 000 | ns_1@127.0.0.1 | **8:03:24 AM** Tue Mar 13, 2018 |
| I'm the only node, so I'm the master. | mb_master 000 | ns_1@127.0.0.1 | **8:03:24 AM** Tue Mar 13, 2018 |
| Couchbase Server has started on web port 8091 on node 'ns_1@127.0.0.1'. Version: "5.1.0-5552-enterprise". | menelaus_sup 001 | ns_1@127.0.0.1 | **8:03:24 AM** Tue Mar 13, 2018 |
| Hot-reloaded memcached.json for config change of the following keys: [<<"client_cert_auth">>, <<"xattr_enabled">>] | memcached_config_mgr 0... | ns_1@127.0.0.1 | **5:49:55 AM** Tue Mar 13, 2018 |

3. To upload the collected logs to us, please tick the **Upload to Couchbase** checkbox and use the following values:
   - **Upload to host:** `uploads.couchbase.com`
   - **Customer name:** ``

- **Ticket number:** ``

**Collecting server logs via the command-line**

Or you can use the `cbcollect_info` command (with `sudo` on Linux) to gather the logs and then upload them to us with the `curl` command:

Example usage:

- Linux (run as root or use sudo as below)

  ```
  sudo /opt/couchbase/bin/cbcollect_info <node_name>.zip
  ```

- Windows (run as an administrator)

  ```
  C:\Program Files\Couchbase\Server\bin\cbcollect_info <node_name>.zip
  ```

Run cbcollect_info on all nodes in the cluster, and upload all of the resulting files to us.

```
curl --upload-file <FILE NAME> https://uploads.couchbase.com/<customer>/<tic
ket number>/
```

Please note that the trailing slash ( `/` ) is required in the upload URL for a successful upload of the logs.

---

## Sync Gateway logs

Introduced in Sync Gateway 1.3, the sgcollect_info tool provides a method to collect and upload logs to Couchbase Support. `sgcollect_info` can be run from the command-line as per the example usage below. You can find full details on using the built-in Upload functionality in the documentation, or you can share the output manually using the `curl` command below.

- Linux (run as root or use sudo as below)

  ```
  sudo /opt/couchbase-sync-gateway/tools/sgcollect_info <node_name>.zip
  ```

- Windows (run as administrator)

  ```
  Program Files (x86)\Couchbase\tools\sgcollect_info.exe <node_name>.zip
  ```

Sync Gateway 1.2 and earlier:

- If running a version of Sync Gateway earlier than 1.3, you can still collect and share the log files with us manually.
- Sync Gateway allows a great degree of flexibility on how logging is configured , so the location of these logs will depend on your specific deployment.
- As well as Sync Gateway's logs, the Configuration File used in your deployment will help us gain a better

understanding of the situation and any potential issues.

To upload the `sgcollect_info` logs, please use the command below:

```
curl --upload-file <FILE NAME> https://uploads.couchbase.com/<customer>/<tic
ket>/
```

Please note that the trailing slash ( `/` ) is required in the upload URL for a successful upload of the logs.

Please see the *Couchbase Server Logs* and *Sharing Files with Us* sections of our Working with Couchbase Technical Support guide for more details.

## Application Logs

Depending on the type of ticket it may also be necessary to upload logs from the client application including the client SDK logs. The way you configure logging in each language is a little different. See the links below for your specific language. At a minimum you should have your applications configured to collect at the INFO level. They applications should be written in a way that the log level can be updated without having to restart the application.

- Java
- Python
- .Net
- PHP
- C
- Go
- Node.js

There may be times when engineering asks that the log level be updated to DEBUG or possibly TRACE. At that time the log level would need to be updated to the requested level, the logs collected and then if appropriate the log level returned to INFO. It is important that if it is possible to reproduce the error that prompted creation of the ticket that it is reproduced during log collection at the higher log level.

Once the logs are collected it is not necessary for them to be zipped together into a single file for upload but it may make uploading the files easier. To zip all the files together make sure the files are in the same directory.

```
tar -czf <hostname>_applicationLogs.tar.gz /path/to/directory-or-file
```

Then upload the files to the Ticket.

```
curl --upload-file <hostname_applicationLogs>.tar.gz \
    https://uploads.couchbase.com/<customer>/<ticket>/
```

## Networking Logs

- Wireshark/tshark

Use the following command on each Application node connecting to Couchbase. Let the collection run for a few minutes. The pcap filename needs to be named for the node it was run on and all of the Couchbase nodes need to be included in the list of hostnames. Then upload the pcap files to the ticket. It is imperative that the problem indicated in the ticket is reproduced during the collection of these pcaps.

```
tshark -w <hostname>.pcap -F pcap host <ip/hostname> or <ip/hostname> or ...
```

To read a pcap file use the following command:

```
tshark -r <hostname>.pcap
```

Then upload the files to the Ticket.

```
curl --upload-file <hostname>.pcap https://uploads.couchbase.com/<customer>/
<ticket>/
```

---

## Couchbase Lite Logs

Couchbase Lite CBL2.5 and greater

This version of Couchbase Lite supports both console based logging and file based logging. By default console based logging is enabled and accessible through adb for Android or NSLog for iOS. File based logging is disabled by default. Connecting to console based logging and getting logs off of devices is outside the scope of this guide.

Configuring and enabling logging is language dependent. See the links below for specific languages.

- java
- swift
- JavaScript
- objective-c
- C#

This is an example of how to configure file based logging in java.

```
final File path = context.getCacheDir();
Database.log.getFile().setConfig(new LogFileConfiguration(path.toString()));
Database.log.getFile().setLevel(LogLevel.INFO);
```

---

Couchbase Lite earlier than CBL2.5

The log messages are split into different domains (LogDomains) which can be tuned to different log levels. The following example enables verbose logging for the replicator and query domains.

**java**

```
Database.setLogLevel(LogDomain.REPLICATOR, LogLevel.VERBOSE);
Database.setLogLevel(LogDomain.QUERY, LogLevel.VERBOSE);
```

**swift**

```
Database.setLogLevel(.verbose, domain: .replicator)
Database.setLogLevel(.verbose, domain: .query)
```

---

## General Support Best Practices

**Recommendation:** Have an Incident Leader

- A leader should be chosen to manage the discussion on each incident. That leader should manage the conversation.
  Often a thread of investigation can be cut off before it bears fruit because someone on the call has a different opinion of the potential problem or simply doesn’t understand the current investigation and pushes their own agenda.

---

**Recommendation:** Keep an Incident Log

- This leader should be responsible (perhaps through delegation) for the keeping of an incident log. While ultimately this log should be transcribed to an official report, the best place to keep a running log is in the chat window of a Webex (or other) call. This allows someone who joins a call to catch up on what has happened, what has been discussed, and/or what has been tried so far. Webex chat is supported across many platforms and is available to anyone invited to the call, including vendors or people who may not be at their work desks.

---

**Recommendation:** Have Architectural Documents Ready

- You can reduce MTTR significantly by reducing the number of questions that need to be asked during the incident. Having information prepared can help expedite suggestions for remedial action and improve the accuracy and likelihood of success of any suggestions.
- For Couchbase, some information is standard, such as:
  - Product version, including server and SDKs used
  - Cluster size
  - Layout of Couchbase services
  - Peak document count, average document size, normal document growth rates
  - Overall data size and average data residency
  - Types of buckets used (Couchbase, Ephemeral, Memcached) and the ejection types (Value, Full)
- For any architecture, it is important to have
  - Information about the underlying hardware and/or virtualization platform

  - Upstream and downstream applications
- It is also critical to have an accurate history maintained of changes to the system, such as version upgrades, nodes added/removed/restarted etc. These events should be logged when they happen so that an accurate history doesn't need to be recreated during an incident.
- Ideally, a (potentially sanitized) version of this information should be made available to any Couchbase Support Engineers working the incident.

---

**Recommendation:** Progress through well defined Incident Stages

- There should be an ordered progression to the incident investigation.
- Suggested steps would be:
  - Preservation of logs and timeline of events preceding the incident
  - Resumption of service
  - Root Cause Analysis
  - Remediation

---

**Recommendation:** Upload all available logs to a Support Ticket as soon as possible

- Updating logs takes time. Having Couchbase support request logs takes additional time. Please upload all available logs immediately after opening a ticket.
- In most cases, these logs are used for Root Cause Analysis, so, log preservation and resumption of service should already have occurred.
- Ideally, the customer should be capable of resuming service without the help of Couchbase Support. If help is needed to achieve resumption of service, the priority of uploading all logs should be increased.

---

**Recommendation:** Have Additional Servers ready to deploy in each environment

- *Cloud native* technologies are designed to have failing nodes replaced with additional nodes.

---

**Recommendation:** Properly Size your Machines & Cluster

- Cloud database, including Couchbase, are designed to partition their load across multiple machines. They are also designed to heal if a machine is lost and/or removed from the cluster. For Couchbase, this is either a *Failover*, *Recovery* or *Rebalance*. There must be enough additional resources in the cluster to perform the recovery options.

---

**Recommendation:** Triage from the Top Down

- Triage gives clarity to the person trying to restore service because they know clearly what they 'don't have to worry about'.
- Triage has to start with the system as a whole, not a particular component. Develop a series of scripts particular to your environment and application that could validate which systems were responding and which were not.

---

**Recommendation:** Avoid Bias

- Experienced Support Engineers (like doctors) will often ignore your specific complaint until they have run through a series of tests or looked at key indicators in the logs. While this is part of a triage process, it also avoids observation bias. Bias can be introduced by any of the following factors:
    - *This is the area of the system where the problem is manifesting or is observed* - this isn't necessarily where the problem is caused
    - *This part of the architecture is the newest* or *has caused us problems before* - while a useful observation, this can cause your analysis to focus on the wrong component
    - *The problem can't be in the area of the stack that I'm responsible for* - personal pride could impede investigation into the area potentially responsible for the problem
    - *I don't understand that part of the architecture* - can lead to either avoidance or undue focus on one part of the architecture
- One of the prime roles of the Incident Leader it to avoid bias and ensure all options are considered until effectively ruled out.

**Recommendation:** Test Tickets

- New applications that are preparing to go into production should go through the process of creating a Test ticket to make sure there are no impediments to updating logs.

# Providing internal tier one Support

## Escalation path

1. Internal Support (Center of Excellence)
2. If the issue is related to debugging a product fault internal support should contact Couchbase technical support with the appropriate priority level
3. If the issue is related to reviewing Architecture, index definitions, query syntax, or client SDK code, internal support should contact Solutions Engineer or professional services representative.
4. Depending on the way to resolve your issue your ticket may be referred to Engineering.

## Building an internal support team

To start with, when building an internal tier one support team it is important that those providing the support review and understand all of the following guides and best practices.

- Backup and Restore Guide
- Cluster Setup
- Data Modeling
- Development Best Practices
- Monitoring
- Security
- Testing

If the use case involves query:

- N1QL Tuning

With a strong understanding of those guides, one should be able to provide support for most tier one support issues.

## Timeouts and RTO

Sometimes, client to Couchbase interactions can timeout. If you are having issues with timeouts, using the open-tracing API may help determine the cause or where the timeout is actually happening. How to use open-tracing is language dependent. The exposure of these APIs will allow you to use other tools like jaegertracing or other commercial products to dig into where the timeouts are happening.

- Java
- Python
- .Net
- PHP
- C
- Go
- Node.js

Response Time Observability (RTO) is part of open-tracing and as four parts:

- Data Service (KV Engine): Logs KV operations that exceed a threshold
- Threshold Logging (SDK): Logs requests that exceed a per-service threshold
- Orphan Logging (SDK): Log responses when the request has timed out
- Improved Timeout Messages (SDK)

### Data Service

Data Service Tracing must first be enabled to use RTO on the data service.

```
curl -u Administrator:password -X POST localhost:8091/pools/default/settings
/memcached/global \
    --data tracing_enabled=true/false
```

Which then gives the following messages in the memcached.log

```
2018-07-17T15:30:05.308518Z INFO 37: HELO [{"a":"libcouchbase/2.9.2 (Darwin-
17.6.0; x86_64; \
    Clang 9.1.0.9020039)","i":"00000000b35f58aa/39cc1fd4eeaf1d67"}] \
    TCP nodelay, XATTR, XERROR, Select bucket, Snappy, JSON, Tracing \
    [ 127.0.0.1:55200 - 127.0.0.1:11210 (not authenticated) ]
...
2018-07-17T15:30:07.084239Z WARNING 37: Slow operation. \
    {"cid":"00000000b35f58aa/39cc1fd4eeaf1d67/0","duration":"1771 ms", \
    "trace":"request=329316910555375:1771975 \
             get=329316911686879:35 \
             bg.wait=329316911701108:4715 \
             bg.load=329318677724103:1766022 \
             get=329318682489143:21",
    "command":"GET","peer":"127.0.0.1:55200"}
```

| Name | Definition | Comments |
|---|---|---|
| request | From KV-Engine receiving request (from OS) to sending response (to OS). | Overall KV-Engine view of the request. |
| bg.wait | From KV-Engine detecting background fetch needed, to starting to read from disk. | Long duration suggests contention on Reader Threads. |
| bg.load | From KV-Engine starting to read from disk to loading the document. | Long duration suggests slow disk subsystem. |
| get | From KV-Engine parsing a GET request to processing it. | For docs which are not resident, you'll see two instances of this span. |
| get.if | From KV-Engine parsing a GET _IF request to processing it. | Similar to get, but used for certain request (e.g. XATTR handling) |
| get.stats | From KV-Engine parsing a STATS request to processing it. | Different STATS keys have different costs (e.g. disk stats are slower). |
| store | From KV-Engine parsing a STORE request to processing it. | Typically fast (as only has to write into HashTable). |
| set.with.meta | From KV-Engine parsing a SET_WITH_META request to processing it. | Similar to store but for XDCR / restore updates. |

### SDK Logging

SDK Logging is on by default. Below is an example from an RTO Log.

```
{
  "service": "kv",
  "count": 15,
  "top": [
    {
      "operation_name": "get",
      "last_operation_id": "0x21",
      "last_local_address": "10.211.55.3:52450",
      "last_remote_address": "10.112.180.101:11210",
      "last_local_id": "66388CF5BFCF7522/18CC8791579B567C",
      "total_duration_us": 18908,
      "encode_us": 256,
      "dispatch_us": 825,
      "decode_us": 16,
      "server_duration_us": 14
    }
  ]
}
```

| Name | Definition | Comments |
|---|---|---|
| operation_name | Type of operation | |
| | | |

| | | |
|---|---|---|
| last_operation_id | A combination of type of operation and ID | Useful for troubleshooting in combination with the local_id |
| last_local_address | The local socket used for this operation | |
| last_remote_address | Socket used on server for this request | Useful when determining which node processed this request |
| last_local_id | This ID is negotiated with the server | Can be used to correlate logging information on both sides |
| total_duration_us | The total time it took to perform the full operation | |
| encode_us | Time the client took to encode the request | Is longer the larger and more complex the json |
| dispatch_us | Time from when client to sent the request to when it got a response into the clients ring buffer | Amount of time spent traversing the network can be found by subtracting dispatch_us - server_duration_us |
| decode_us | Time the client took to decode the response | Is longer the larger and more complex the json |
| server_duration_us | Time the server took to do its work. | - |

**Orphan Logging**

This is enabled by default in the SDK. This aggregates responses where the request has timed out. ie KV get exceeds time out, error returned to application, response received some time afterwards. The log interval is 10 seconds at the WARN level, and has a Per-service sample size of 10. Below is an example of an orphan log.

```
{
  "service": "kv",
  "count": 2,
  "top": [
    {
      "s": "kv:get",
      "i": "0x21",
      "c": "66388CF5BFCF7522/18CC8791579B567C",
      "b": "default",
      "l": "192.168.1.101:11210",
      "r": "10.112.181.101:12110",
      "d": 120
    }
  ]
}
```

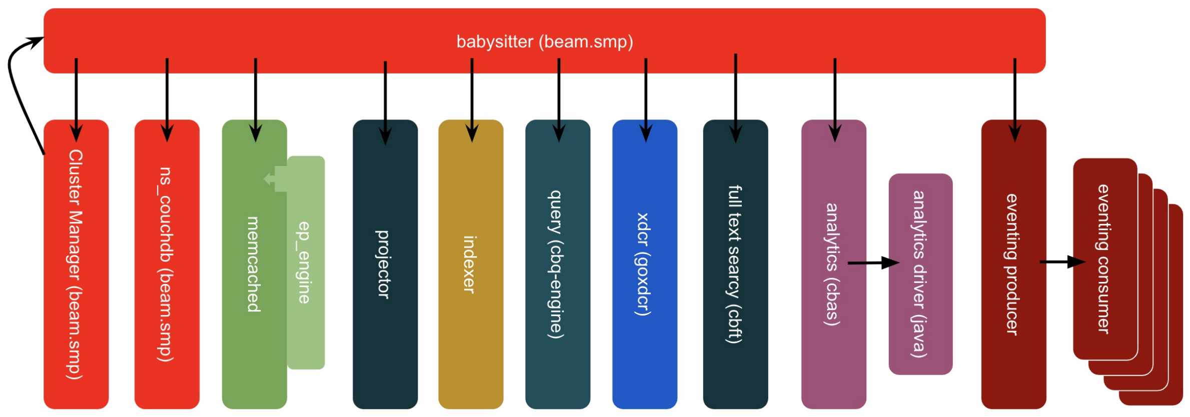
| Name | Definition |
|---|---|
| s | Service type |
| i | Operation ID |

| | |
|---|---|
| c | Connection ID |
| b | Bucket |
| l | LocalEndpoint and Port |
| r | Remote Endpoint and Port |
| d | Duration (us) |
| t | Timeout |

Combining these three logs, a troubleshooter should be able to trace timings all the way from the client to server and back. This should help identify any operations that timed out, or are performing slowly. Providing this information to support will greatly help in determining the cause of timeouts.

## Restarting services

On each Couchbase node there are multiple services running to maintain Couchbase and provide the services needed for operation. At times, and normally with the guidance of support, it may be necessary to restart some of those services in a recovery effort from degraded operations. A complete list of processes can be found at https://docs.couchbase.com/server/current/install/server-processes.html



## Cluster Manager

**What it does:** Provides admin UI & RESTful administrative APIs, coordinates cluster membership, manages rebalancing, etc.

**Why to restart:** Connections not being released, rebalance that won't 'stop'

**Impact of restarting:**

- Should not impact operations unless it doesn't restart faster than the auto-failover timeout. In that case restarting may cause a failover.

**How:**

```
curl --data “erlang:halt().” -u <Administrator>:<password> http://<host>:809
1/diag/eval
```

## Memcached

**What it does:** This is the data service. It manages all the vbuckets and the data within the vbuckets.

**Why to restart:**

- Do not restart without guidance from support.
- This would be done if a hard failover of a node was needed and the UI was unavailable.

**Impact of restarting:** Causes a hard failover of the data node.

**How:**

```
pkill memcached
```

## Projector

**What it does:** This process runs on each Data node. It consumes DCP messages, Filters and formats index entries, and sends those messages to the indexer.

**Why to restart:**

- New index build seems ‘stuck’
- items_queued + items_pending increasing and not going down

**Impact of restarting:**

- Should continue from last checkpoint, no direct effect on inflight ops
- Likely already affecting At_plus or Request_plus queries

**How:**

```
pkill projector
```

## Indexer

**What it does:** The indexer runs on each Index node. It manages index records in memory & disk. This process is invoked by query engine to select query candidates.

**Why to restart:**

- Memory use of the indexer process is growing without bound
- Existing memory leak of metadata (fixed in 6.0.2)

**Impact of restarting:**

- Inflight queries may fail if they are using indexes maintained by this indexer. *YOUR CLIENTS SHOULD CODE FOR THIS EVENT*
- Assuming you have redundant index definitions no impact should occur to new queries.

**How:**

```
pkill indexer
```

---

## Query Service

**What it does:** The query service runs on each Query node. Each N1QL query is assigned to one node. Calls from the query service go out to Index service, Data Service and FTS Service as required

**Why to restart:**

- Memory use growing without bound

**Impact of restarting:**

- Inflight queries will fail if they are assigned to this query node. *YOUR CLIENTS SHOULD CODE FOR THIS EVENT*
- New queries will not go to this node and should be fine

**How:**

```
pkill cbq-engine
```

---

## XDCR Service

**What it does:** the goxdcr process runs on each Data node. It has responsibilities for data replication work done on both source & target side.

**Why to restart:**

- High CPU usage
- Connection timeouts in log
- changes_left stat increasing without going back down
- bandwidth_usage stat at 0

**Impact of restarting:**

- Will restart from last checkpoint, may see a temporary increase in CPU & bandwidth usage

**How:**

```
pkill goxdcr
```