Hanging hundreds of Postgres shutdowns with a simple CDC app

Yoann La Cancellera
Lead senior support at Percona

PERCONA

Story of an original prod failure

Not a tutorial.
The conclusion is really simple but getting to it was exotic

PERCONA®

Story of an original prod failure

Constraints:

- I can never touch any environment
- I can only work with data received by email(tickets)
- We can plan a screenshare session, but **I can’t get control over it!**
- Before blaming any other component, we must prove exactly why

PERCONA®

Problem

- For every switchover, rollout, restart, **Postgres would never stop**
  - K8s prod based on Patroni operator, mostly PG 16.x
- Only the primary is affected. Replicas can stop
- Reproduces on **every** clusters (50-100), **everytime!**
- Even waiting 3 hours with 0 activity beforehand, it would not stop

PERCONA

Symptoms

- Indeed it’s not stopping

```
2025-11-05 12:40:39.296 UTC [76] LOG:  received fast shutdown request
2025-11-05 12:40:39.299 UTC [76] LOG:  aborting any active transactions
2025-11-05 12:40:39.299 UTC [266158] FATAL:  terminating connection due to administrator command
2025-11-05 12:40:39.299 UTC [266157] FATAL:  terminating connection due to administrator command
...
2025-11-05 12:40:39.518 UTC [76] LOG:  background worker "logical replication launcher" (PID 608) exited with exit code 1
2025-11-05 12:40:39.522 UTC [79] LOG:  shutting down
2025-11-05 12:40:39.528 UTC [266227] FATAL:  the database system is shutting down
2025-11-05 12:40:39.528 UTC [266226] FATAL:  the database system is shutting down
```

- And they end doing kill -9 everytime

```
2025-11-05 13:13:34.219 UTC [76] LOG:  server process (PID 2910) was terminated by signal 9: Killed
2025-11-05 13:13:34.219 UTC [76] DETAIL:  Failed process was running: START_REPLICATION SLOT "dms_slot" LOGICAL
00000943/A1004900 ("include-timestamp" 'on')
2025-11-05 13:13:34.219 UTC [76] LOG:  terminating any other active server processes
2025-11-05 13:13:34.220 UTC [76] LOG:  abnormal database system shutdown
2025-11-05 13:13:34.220 UTC [76] LOG:  [pg_stat_monitor] pgsm_shmem_shutdown: Shutdown initiated.
2025-11-05 13:13:34.338 UTC [76] LOG:  database system is shut down
```


PERCONA

Symptoms

- We had some process while it was hanging
  - Postmaster
  - Checkpointer
  - 2 walsender for patroni replicas
  - 2 walsender for extra logical replicas

- Can’t connect since postgres is stopping

- Only those logs are providing some clue

```
[76] DETAIL:  Failed process was running: START_REPLICATION SLOT "dms_slot" LOGICAL 00000943/A1004900
("include-timestamp" 'on')

[67893] LOG:  background worker "logical replication launcher" (PID 68289) exited with exit code 1
```


PERCONA

Basic reflexes

- Some idle replica ? wal_sender_timeout ? **=> Nope!**
- Is the replica lagging ? **=> Nope!**
- Any transaction ? **=> Nope!**
- Any k8s instability ? **=> Nope!**
- Any extension that might be bugged ? **=> Nope!**
- Any foreign data wrapper or other external call ? **=> Nope!**
- (slow network ? That too, but what can you do)

PERCONA

But what are those logical replicas ?

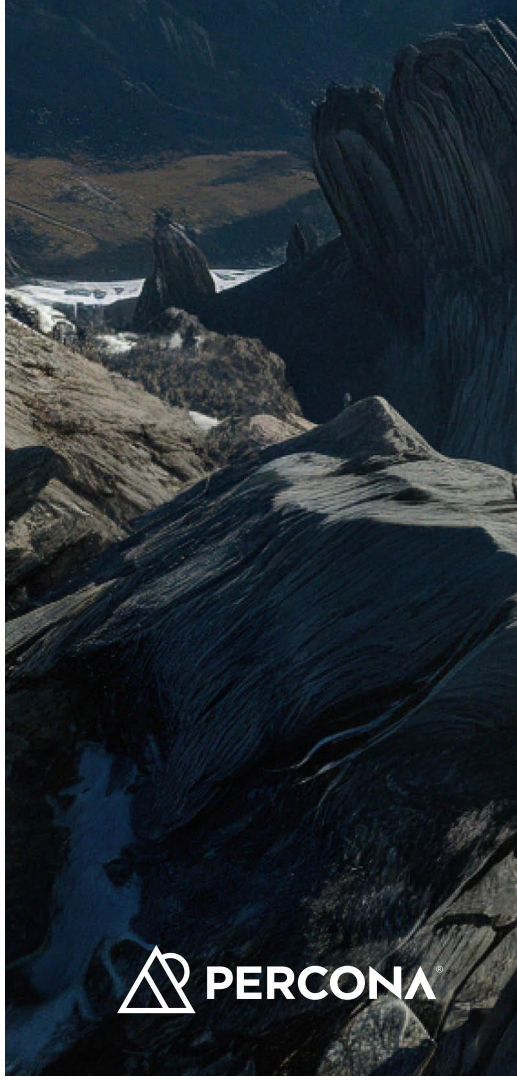
- Both are CDC systems
  - AWS DMS (Data Migration Service) using test_decoding, **which we found failing in logs.**
  - "**auditcdc**" using wal2json, and required for their application

- Does it work if we stop them ?
  - **Yes!**
  - (little lack of clarity if it required stopping both or only one, but I can't retry for them!)

Quick terminology

- Postgres is the **publisher**
- Their app is a **subscriber** connecting to postgres using a **logical replication slot**
- This app tracks data changes, this pattern is called **CDC** (=> Change Data Capture)

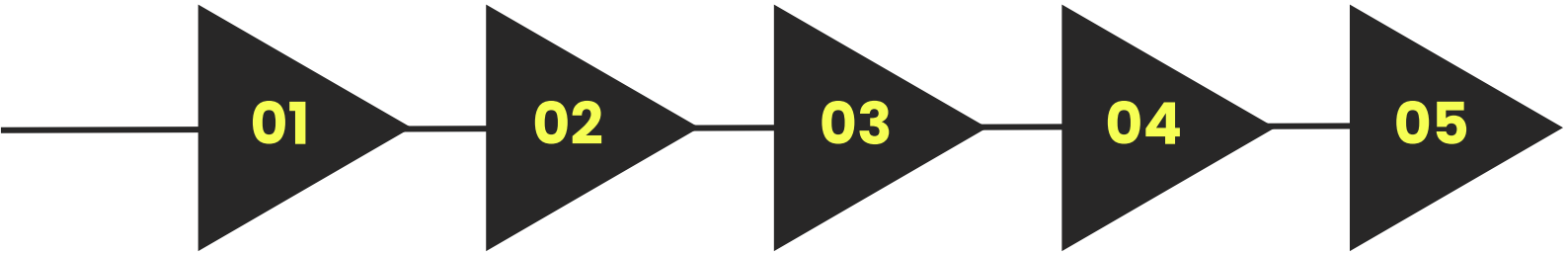
Ok sure, but even assuming the worst, how can a subscriber even impact postgres like that ?



Tracing!

PERCONA®

Tracing fun !



Getting the k8s worker

Starting debug pod on the worker

Install linux-tools

Getting the right PIDs!

And checking the pid mapping:

grep <podUID> /proc/{pid,potentiels}/cgroup

Strace, perf

perf record -a -g -F99 -p <pid> -o perf.data -- sleep 60

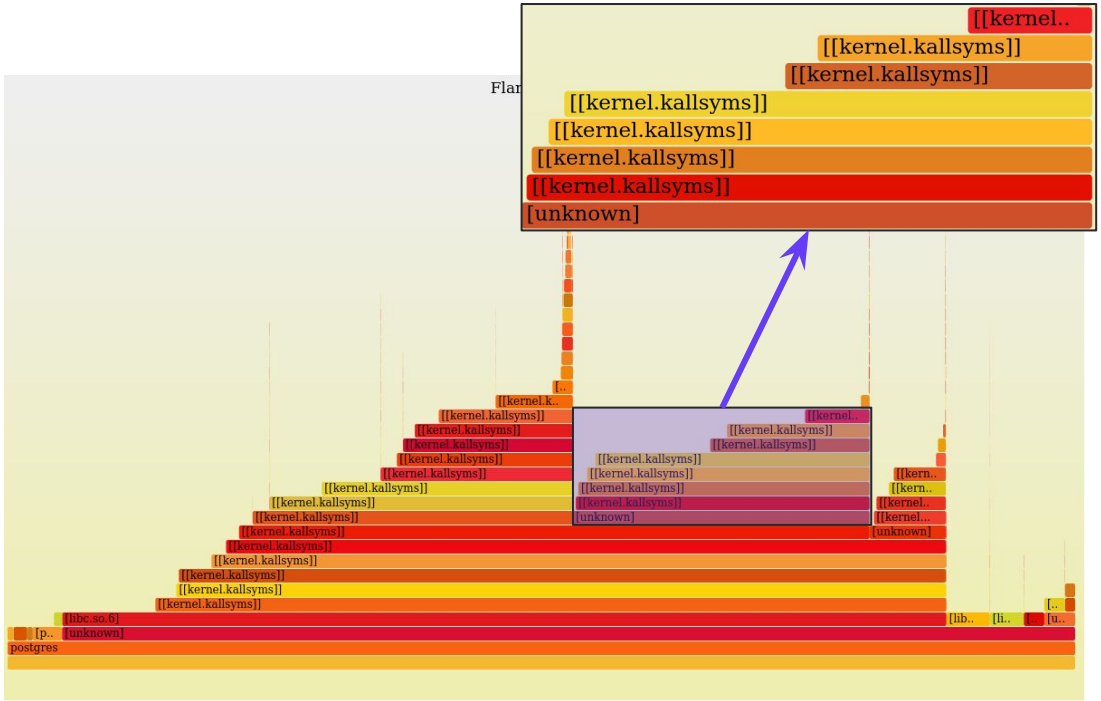
timeout 60 strace -tt -T -f -yy -o /tmp/strace.out -p <pid>

PERCONA

Tracing fun !

- Perf: nothing was visible at all.
- Empty result on the postmaster
- On logical replicas we only get:

=> it’s not helping



Stracing fun !

- Strace, we got results
- Postmaster
- Accepting connections which will get an error
- 148kb of output

```
2785772 epoll_wait(10<anon_inode:[eventpoll]>, [{EPOLLIN, {u32=964094112, u64=964094112
2785772 accept(7<TCP:[315909653]>, {sa_family=AF_INET, sin_port=htons(40307),
sin_addr=inet_addr("<applicationIP>")}, [128->16]) = 11<TCP:[334168843]>
2785772 getsockname(11<TCP:[334168843]>, {sa_family=AF_INET, sin_port=htons(5432),
sin_addr=inet_addr("<serverIP>")}, [128->16]) = 0
2785772 setsockopt(11<TCP:[334168843]>, SOL_TCP, TCP_NODELAY, [1], 4) = 0
2785772 setsockopt(11<TCP:[334168843]>, SOL_SOCKET, SO_KEEPALIVE, [1], 4) = 0
2785772 getpid()                                = 76
2785772 rt_sigprocmask(SIG_SETMASK, ~[ILL TRAP ABRT BUS FPE SEGV CONT SYS RTMIN RT_1], [U
2785772 clone( <unfinished ...>
2785772 <... clone resumed> child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETT
child_tidptr=0x7f7afaa1cc10) = 267023



2025-11-05 12:44:35.119 UTC [267023] FATAL:  the database system is shutting down
```


PERCONA

Stracing fun !

- Checkpointer?
- Is it waiting? (yes)
- `10000000ns = 10ms`

```
2785777 restart_syscall(<... resuming interrupted read ...>) = 0
2785777 clock_nanosleep(CLOCK_REALTIME, 0, {tv_sec=0, tv_nsec=10000000}, NULL) = 0
2785777 clock_nanosleep(CLOCK_REALTIME, 0, {tv_sec=0, tv_nsec=10000000}, NULL) = 0
2785777 clock_nanosleep(CLOCK_REALTIME, 0, {tv_sec=0, tv_nsec=10000000}, NULL) = 0
2785777 clock_nanosleep(CLOCK_REALTIME, 0, {tv_sec=0, tv_nsec=10000000}, NULL) = 0
2785777 clock_nanosleep(CLOCK_REALTIME, 0, {tv_sec=0, tv_nsec=10000000}, NULL) = 0
2785777 clock_nanosleep(CLOCK_REALTIME, 0, {tv_sec=0, tv_nsec=10000000}, NULL) = 0
2785777 clock_nanosleep(CLOCK_REALTIME, 0, {tv_sec=0, tv_nsec=10000000}, NULL) = 0


$ uniq -c strace.out.checkpointer
      1 2785777 restart_syscall(<... resuming interrupted read ...>) = 0
   2831 2785777 clock_nanosleep(CLOCK_REALTIME, 0, {tv_sec=0, tv_nsec=10000000}, NULL) =
      1 2785777 clock_nanosleep(CLOCK_REALTIME, 0, {tv_sec=0, tv_nsec=10000000},  <detach
```


PERCONA

Stracing fun !

- Physical replicas?
- Just responding errors to connections ?
- Only 3kb of output

```
233794 epoll_wait(5<anon_inode:[eventpoll]>, [{EPOLLIN, {u32=964563016, u64=964563016}}
233794 recvfrom(11<TCP:[334140050]>, "<removed>", 5, 0, NULL, NULL) = 5
233794 recvfrom(11<TCP:[334140050]>, "<removed>"..., 56, 0, NULL, NULL) = 56
233794 recvfrom(11<TCP:[334140050]>, 0x397e4153, 5, 0, NULL, NULL) = -1 EAGAIN (Resour
temporarily unavailable)
233794 sendto(11<TCP:[334140050]>, <removed>"..., 45, 0, NULL, 0) = 45
233794 recvfrom(11<TCP:[334140050]>, 0x397e4153, 5, 0, NULL, NULL) = -1 EAGAIN (Resource
temporarily unavailable)
233794 epoll_wait(5<anon_inode:[eventpoll]>, [{EPOLLIN, {u32=964563016, u64=964563016}}],
233794 recvfrom(11<TCP:[334140050]>, "\27\3\3\0/", 5, 0, NULL, NULL) = 5
233794 recvfrom(11<TCP:[334140050]>, "<removed>"..., 47, 0, NULL, NULL) = 47
233794 recvfrom(11<TCP:[334140050]>, 0x397e4153, 5, 0, NULL, NULL) = -1 EAGAIN (Resource
temporarily unavailable)
```


PERCONA®

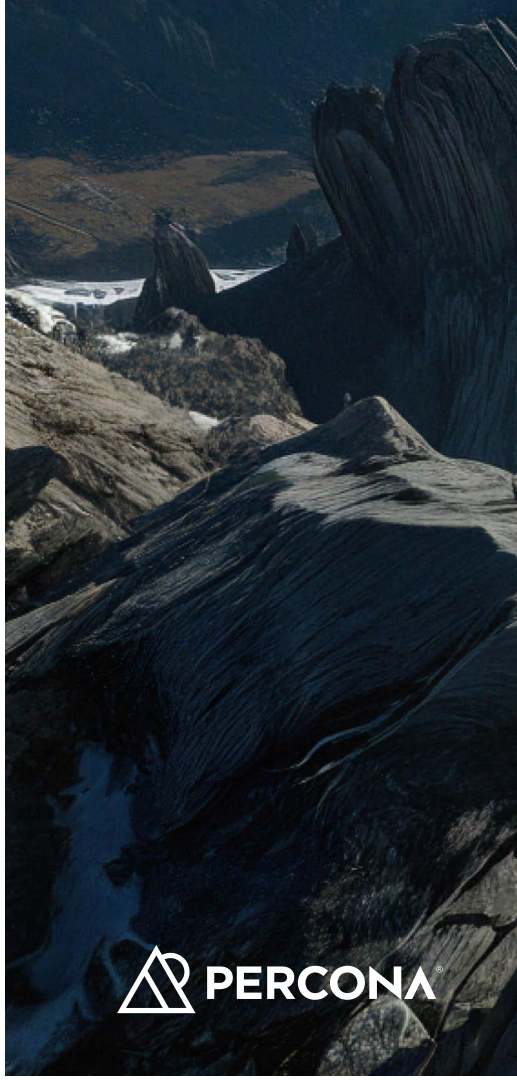
Stracing fun !

- Logical replicas?
- 3mb of output, so more activity
- recvfrom “d” ?
- Reconnecting?

```
212246 recvfrom(11<TCP:[334074242]>, "d", 1, 0, NULL, NULL) = 1
212246 recvfrom(11<TCP:[334074242]>, "<removed>"..., 8192, 0, NULL, NULL) = 38
212246 recvfrom(11<TCP:[334074242]>, 0x7fffc7b6780f, 1, 0, NULL, NULL) = -1 EAGAIN (Resource
temporarily unavailable)
212246 recvfrom(11<TCP:[334074242]>, 0x7fffc7b6b9ef, 1, 0, NULL, NULL) = -1 EAGAIN (Resource
temporarily unavailable)
212246 sendto(11<TCP:[334074242]>, "<removed>"..., 46, 0, NULL, 0) = 46
212246 recvfrom(11<TCP:[334074242]>, "d", 1, 0, NULL, NULL) = 1
212246 recvfrom(11<TCP:[334074242]>, "<removed>"..., 8192, 0, NULL, NULL) = 38
212246 recvfrom(11<TCP:[334074242]>, 0x7fffc7b6b9ef, 1, 0, NULL, NULL) = -1 EAGAIN (Resource
temporarily unavailable)
212246 sendto(11<TCP:[334074242]>, "<removed>", 23, 0, NULL, 0) = 23
212246 recvfrom(11<TCP:[334074242]>, 0x7fffc7b6b9ef, 1, 0, NULL, NULL) = -1 EAGAIN (Resou
temporarily unavailable)
212246 recvfrom(11<TCP:[334074242]>, 0x7fffc7b6780f, 1, 0, NULL, NULL) = -1 EAGAIN (Resource
temporarily unavailable)
212246 recvfrom(11<TCP:[334074242]>, 0x7fffc7b6780f, 1, 0, NULL, NULL) = -1 EAGAIN (Resource
temporarily unavailable)
212246 recvfrom(11<TCP:[334074242]>, 0x7fffc7b6b9ef, 1, 0, NULL, NULL) = -1 EAGAIN (Resource
temporarily unavailable)
212246 sendto(11<TCP:[334074242]>, "<removed>", 23, 0, NULL, 0) = 23
```


PERCONA

What is expected or not from code ?



Code analysis

- Stopping walsenders is part of the checkpointer shutdown
- Checkpointer shutdown is part of the postmaster shutdown
- So those processes are all expected

```
ShutdownXLOG(int code, Datum arg)
{
...
    /*
     * Signal walsenders to move to stopping state.
     */
    WalSndInitStopping();

    /*
     * Wait for WAL senders to be in stopping state.  This prevents commands
     * from writing new WAL.
     */
    WalSndWaitStopping();

    CreateCheckPoint(CHECKPOINT_IS_SHUTDOWN | CHECKPOINT_IMMEDIATE);

}

void
WalSndWaitStopping(void)
{
    for (;;)
    {
        int            i;
        bool        all_stopped = true;

        for (i = 0; i < max_wal_senders; i++)
        {
            ...
        }

        /* safe to leave if confirmation is done for all WAL senders */
        if (all_stopped)return;

        pg_usleep(10000L);        /* wait for 10 msec */
    }
```

Bugs ?

PERCONA®

Bugs ?

- Bug ? A worthy one fixed in 16.10! Most prod were in 16.8

  "Use replay LSN as target for cascading logical WAL senders"

  *"Using the latest flush LSN rather than the replay LSN could cause the **WAL senders to be stuck in an infinite loop preventing them to shut down**, as the startup process does not run when WAL senders attempt to catch up, so they could keep waiting for work that would never happen."*

PERCONA

Bugs ?

- Bug ? A worthy one fixed in 16.10! Most prod were in 16.8

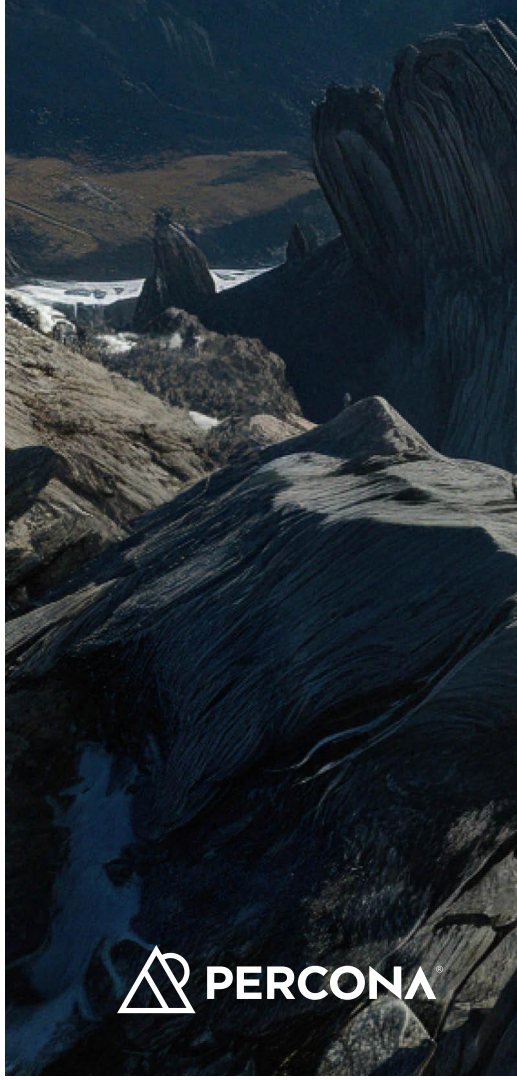
  *"Use replay LSN as target for **cascading** logical WAL senders"*

  *"~~Using the latest flush LSN rather than the replay LSN could cause the **WAL senders to be stuck in an infinite loop preventing them to shut down**, as the startup process does not run when WAL senders attempt to catch up, so they could keep waiting for work that would never happen.~~"*

PERCONA

Who has any idea of what to do after that ?

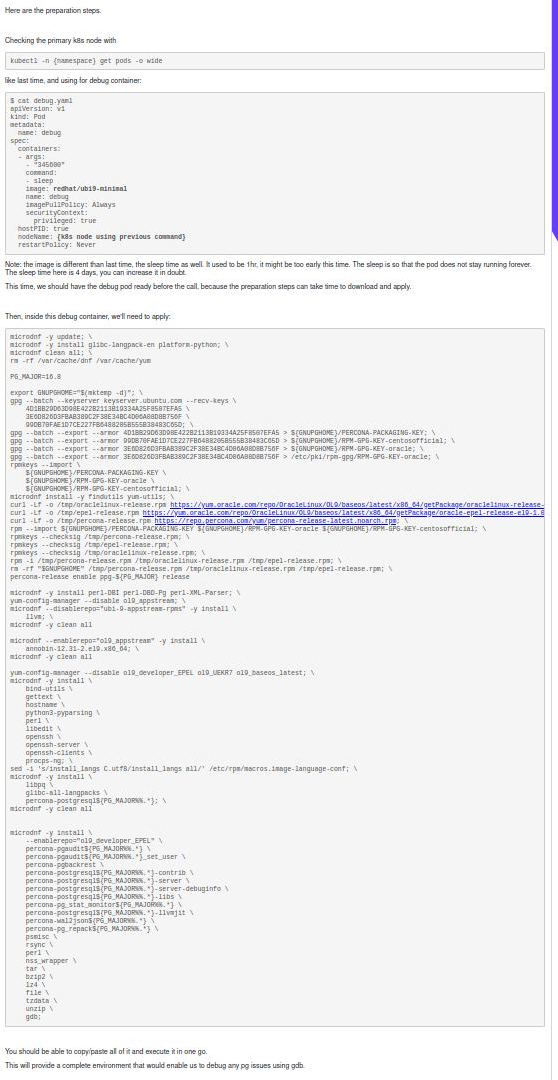
And I’m sure a local wizard will have one



GDB ?

- C debugger, to print and check variables
- Will need symbols installed to be able to access them
- **Solution:** make them deploy a debug pod installing every components and their symbols to launch GDB from there

  (And making sure to do the homework to provide actions to do on GDB on **screenshare** without control!)



Here are the preparation steps.

Checking the primary k8s node with

```
kubectl -n {namespace} get pods -o wide
```

like last time, and using for debug container:

```
$ cat debug.yaml
apiVersion: v1
kind: Pod
metadata:
  name: debug
spec:
  containers:
  - args:
    - "946800"
    command:
    - sleep
    image: redhat/ubi9-minimal
    name: debug
    imagePullPolicy: Always
    securityContext:
      privileged: true
  hostPID: true
  nodeName: {k8s node using previous command}
  restartPolicy: Never
```

Note: the image is different than last time, the sleep time as well. It used to be 1hr, it might be too early this time. The sleep is so that the pod does not stay running forever. The sleep time here is 4 days, you can increase it in doubt.

This time, we should have the debug pod ready before the call, because the preparation steps can take time to download and apply.

Then, inside this debug container, we'll need to apply:

```
microdnf -y update; \
microdnf -y install glibc-langpack-en platform-python; \
microdnf clean all; \
rm -rf /var/cache/dnf /var/cache/yum

PG_MAJOR=16.8

export GNUPGHOME="$(mktemp -d)"; \
gpg --batch --keyserver keyserver.ubuntu.com --recv-keys \
    4D1BB29D63D98E422B2113B19334A25F8507EFA5 \
    3E6DB26D3FBAB389DC2F3BE34BC4D86A8DDB756F \
    99DB70FAE1D7CE227FB6488205B555B38483C65D; \
gpg --batch --export --armor 4D1BB29D63D98E422B2113B19334A25F8507EFA5 > ${GNUPGHOME}/PERCONA-PACKAGING-KEY; \
gpg --batch --export --armor 99DB70FAE1D7CE227FB6488205B555B38483C65D > ${GNUPGHOME}/RPM-GPG-KEY-centosofficial; \
gpg --batch --export --armor 3E6DB26D3FBAB389DC2F3BE34BC4D86A8DDB756F > ${GNUPGHOME}/RPM-GPG-KEY-oracle; \
gpg --batch --export --armor 3E6DB26D3FBAB389DC2F3BE34BC4D86A8DDB756F > /etc/pki/rpm-gpg/RPM-GPG-KEY-oracle; \
rpmkeys --import \
    ${GNUPGHOME}/PERCONA-PACKAGING-KEY \
    ${GNUPGHOME}/RPM-GPG-KEY-oracle \
    ${GNUPGHOME}/RPM-GPG-KEY-centosofficial; \
microdnf install -y findutils yum-utils; \
curl -Lf -o /tmp/oraclelinux-release.rpm https://yum.oracle.com/repo/OracleLinux/OL9/baseos/latest/x86_64/getPackage/oraclelinux-release-
curl -Lf -o /tmp/epel-release.rpm https://yum.oracle.com/repo/OracleLinux/OL9/baseos/latest/x86_64/getPackage/oracle-epel-release-el9-1.0
curl -Lf -o /tmp/percona-release.rpm https://repo.percona.com/yum/percona-release-latest.noarch.rpm; \
rpm --import ${GNUPGHOME}/PERCONA-PACKAGING-KEY ${GNUPGHOME}/RPM-GPG-KEY-oracle ${GNUPGHOME}/RPM-GPG-KEY-centosofficial; \
rpmkeys --checksig /tmp/percona-release.rpm; \
rpmkeys --checksig /tmp/epel-release.rpm; \
rpmkeys --checksig /tmp/oraclelinux-release.rpm; \
rpm -i /tmp/percona-release.rpm /tmp/oraclelinux-release.rpm /tmp/epel-release.rpm; \
rm -rf "$GNUPGHOME" /tmp/percona-release.rpm /tmp/oraclelinux-release.rpm /tmp/epel-release.rpm; \
percona-release enable ppg-${PG_MAJOR} release

microdnf -y install perl-DBI perl-DBD-Pg perl-XML-Parser; \
yum-config-manager --disable ol9_appstream; \
microdnf --disablerepo="ubi-9-appstream-rpms" -y install \
    llvm; \
microdnf -y clean all

microdnf --enablerepo="ol9_appstream" -y install \
    annobin-12.31-2.el9.x86_64; \
microdnf -y clean all

yum-config-manager --disable ol9_developer_EPEL ol9_UEKR7 ol9_baseos_latest; \
microdnf -y install \
    bind-utils \
    gettext \
    hostname \
    python3-pyparsing \
    perl \
    libedit \
    openssh \
    openssh-server \
    openssh-clients \
    procps-ng; \
sed -i 's/install_langs C.utf8/install_langs all/' /etc/rpm/macros.image-language-conf; \
microdnf -y install \
    libpq \
    glibc-all-langpacks \
    percona-postgresql${PG_MAJOR%.*}; \
microdnf -y clean all


microdnf -y install \
    --enablerepo="ol9_developer_EPEL" \
    percona-pgaudit${PG_MAJOR%.*} \
    percona-pgaudit${PG_MAJOR%.*}_set_user \
    percona-pgbackrest \
    percona-postgresql${PG_MAJOR%.*}-contrib \
    percona-postgresql${PG_MAJOR%.*}-server \
    percona-postgresql${PG_MAJOR%.*}-server-debuginfo \
    percona-postgresql${PG_MAJOR%.*}-libs \
    percona-pg_stat_monitor${PG_MAJOR%.*} \
    percona-postgresql${PG_MAJOR%.*}-llvmjit \
    percona-wal2json${PG_MAJOR%.*} \
    percona-pg_repack${PG_MAJOR%.*} \
    psmisc \
    rsync \
    perl \
    nss_wrapper \
    tar \
    bzip2 \
    lz4 \
    file \
    tzdata \
    unzip \
    gdb;
```

You should be able to copy/paste all of it and execute it in one go.

This will provide a complete environment that would enable us to debug any pg issues using gdb.

GDB ?

- AWS EKS
- Just not possible
- No workarounds (right ?)

PERCONA

Desperation!

PERCONA®

Debugging from the source

- Thankfully, this was enough to justify adding a few print statements to their CDC app!

- After the query **`START_REPLICATION`**, the logical replication protocol can only send :
  - **`XLogData`** `messages` (=>DMLs)
  - "**`primary keepalive`**" messages
- And we can only send back
  - Standby status updates
  - Hot standby feedback messages

- https://www.postgresql.org/docs/current/protocol-replication.html#PROTOCOL-REPLICATION-START-REPLICATION

PERCONA

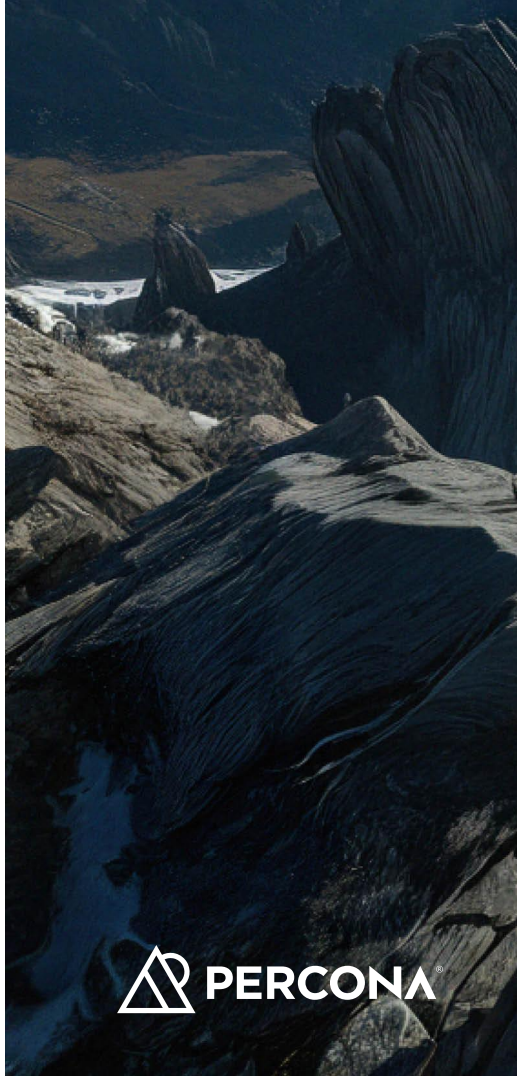
Timeline after a shutdown

- Latest **xLogData** received had server_lsn = 1/00000**BA0**
- Then we got “Primary **keepalive**”, with some LSN 1/00000**ED0**, 816 bytes more than 1/00000**BA0**
- Their app acknowledge this LSN as received, but kept 1/00000**BA0** as committed_lsn

```
StandbyStatusUpdate {
    last_committed_lsn: 1/00000BA0,
    last_received_lsn: 1/00000ED0,
    timestamp: …, request_primary_keepalive: true
}
```

- **Fair enough**: there was no data to commit, and it did update the LSN was received.
- **But it’s looping.** The CDC app kept sending the same, while postgres kept sending primary keepalive messages.

But what would pg_recvlogical do ?



What does pg_recvlogical do?

```
  walEnd = fe_recvint64(&copybuf[pos]);
+ pg_log_info("getting keepalive with endlsn %X/%X %lu, output_written_lsn %X/%X %lu, fsync_lsn %X/%X %lu (slot %s)",
+              LSN_FORMAT_ARGS(walEnd), walEnd,
+              LSN_FORMAT_ARGS(output_written_lsn), output_written_lsn,
+              LSN_FORMAT_ARGS(output_fsync_lsn), output_fsync_lsn,
+              replication_slot);
  output_written_lsn = Max(walEnd, output_written_lsn);

...

  cur_record_lsn = fe_recvint64(&copybuf[1]);
+ pg_log_info("getting xlogdata with cur_record_lsn %X/%X %lu, output_written_lsn %X/%X %lu, fsync_lsn %X/%X %lu (slot %s)",
+                   LSN_FORMAT_ARGS(cur_record_lsn), cur_record_lsn,
+                   LSN_FORMAT_ARGS(output_written_lsn), output_written_lsn,
+                   LSN_FORMAT_ARGS(output_fsync_lsn), output_fsync_lsn,
+                   replication_slot);

...
        if (verbose)
                pg_log_info("confirming write up to %X/%X, flush to %X/%X (slot %s)",
                                        LSN_FORMAT_ARGS(output_written_lsn),
                                        LSN_FORMAT_ARGS(output_fsync_lsn),
                                        replication_slot);
```

What does pg_recvlogical do?

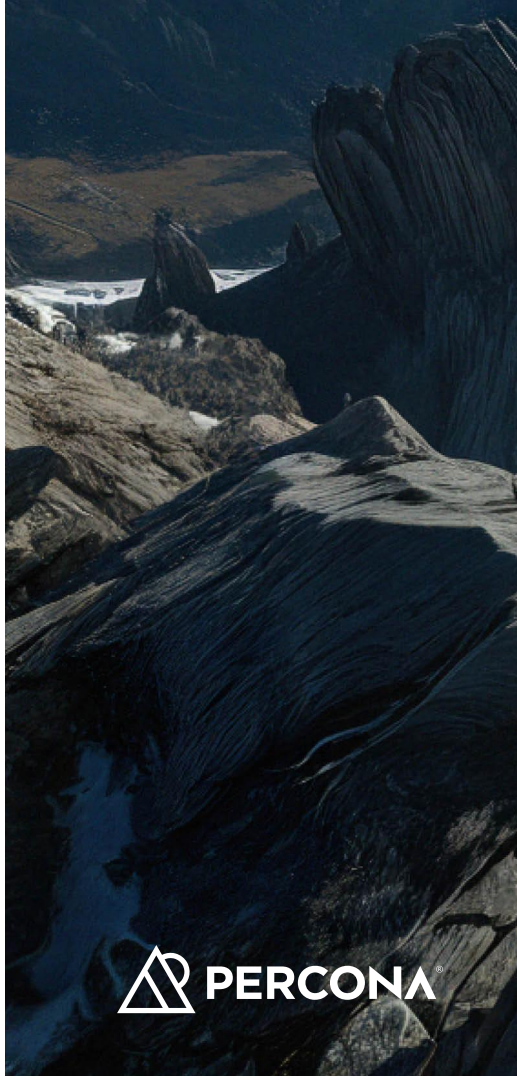
```
bash-4.4$ ./postgres/src/bin/pg_basebackup/pg_recvlogical --dbname=postgres --host=/run/postgresql -I '0/0' -S test -P wal2json
--option=format-version=2 --option=add-tables=public.tab --option=actions=insert --option=include-lsn=true -v --start -f /dev/null

pg_recvlogical: starting log streaming at 0/0 (slot test)
pg_recvlogical: streaming initiated
pg_recvlogical: confirming write up to 0/0 0, flush to 0/0 0 (slot test)
pg_recvlogical: getting keepalive with endlsn 0/7001548 117445960, output_written_lsn 0/0 0, fsync_lsn 0/0 0 (slot test)
pg_recvlogical: getting keepalive with endlsn 0/7001548 117445960, output_written_lsn 0/7001548 117445960, fsync_lsn 0/0 0 (slot test)
pg_recvlogical: getting xlogdata with cur_record_lsn 0/7001548 117445960, output_written_lsn 0/7001548 117445960, fsync_lsn 0/0 0 (slot test)
pg_recvlogical: getting xlogdata with cur_record_lsn 0/70015B8 117446072, output_written_lsn 0/7001548 117445960, fsync_lsn 0/0 0 (slot
test)
pg_recvlogical: getting keepalive with endlsn 0/70015B8 117446072, output_written_lsn 0/70015B8 117446072, fsync_lsn 0/0 0 (slot test)
pg_recvlogical: getting keepalive with endlsn 0/70015B8 117446072, output_written_lsn 0/70015B8 117446072, fsync_lsn 0/0 0 (slot test)
pg_recvlogical: confirming write up to 0/70015B8 117446072, flush to 0/70015B8 117446072 (slot test)
```

=> we would expect this, we flush based on the latest xlog received, but:

```
pg_recvlogical: getting keepalive with endlsn 0/70015F0 117446128, output_written_lsn 0/70015B8 117446072, fsync_lsn 0/70015B8 (slot test)
pg_recvlogical: confirming write up to 0/70015F0 117446128, flush to 0/70015F0 117446128 (slot test)
```

=> pg_recvlogical consider the latest LSN received as flushed. Even if it has no data associated

But how can we have increasing LSNs without any data to match it ?



What? Why?

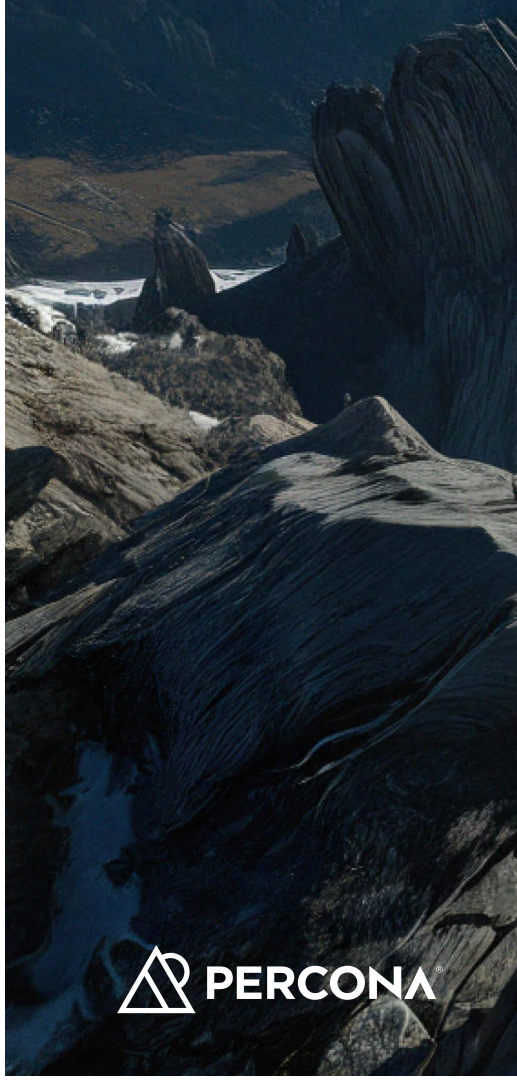
- Actually simple
  - CHECKPOINT
  - select pg_switch_wal()
  - Any DML on a table that is not published

- walreceiver.c (physical replica) will increase its LSN using the amount of bytes written in WAL, which is appropriate since it replicates **byte by byte**

- But **pg_recvlogical is different**: checkpoints, WAL switches are just not interesting for external systems

PERCONA

Final proof!



Reproducing the issue

```
/* Check the message type. */
if (copybuf[0] == 'k')
{
        int                     pos;
        bool            replyRequested;
        XLogRecPtr      walEnd;
        bool            endposReached = false;

        /*
         * Parse the keepalive message, enclosed in the CopyData message.
         * We just check if the server requested a reply, and ignore the
         * rest.
         */
        pos = 1;                        /* skip msgtype 'k' */
        walEnd = fe_recvint64(&copybuf[pos]);
        // output_written_lsn = Max(walEnd, output_written_lsn);
```

- Pg_recvlogical adjusts its “written_lsn” using the LSN received from keepalive messages
- Even if we don’t have any xlogdata to match the LSN

PERCONA

Reproducing the issue

```
pg_recvlogical: getting xlogdata with cur_record_lsn 0/30005D8, output_written_lsn 0/3000330, fsync_lsn 0/3000290 (slot test)
pg_recvlogical: getting keepalive with endlsn 0/30005D8, output_written_lsn 0/30005D8, fsync_lsn 0/3000290 (slot test)
pg_recvlogical: getting keepalive with endlsn 0/30005D8, output_written_lsn 0/30005D8, fsync_lsn 0/3000290 (slot test)
pg_recvlogical: getting keepalive with endlsn 0/3000610, output_written_lsn 0/30005D8, fsync_lsn 0/3000290 (slot test)
pg_recvlogical: confirming write up to 0/30005D8, flush to 0/30005D8(slot test)
pg_recvlogical: getting keepalive with endlsn 0/3000610, output_written_lsn 0/30005D8, fsync_lsn 0/30005D8 (slot test)
pg_recvlogical: confirming write up to 0/30005D8, flush to 0/30005D8(slot test)
<-- SHUTDOWN STARTED -->
pg_recvlogical: getting keepalive with endlsn 0/4000000, output_written_lsn 0/30005D8, fsync_lsn 0/30005D8 (slot test)
pg_recvlogical: confirming write up to 0/30005D8, flush to 0/30005D8(slot test)
pg_recvlogical: getting keepalive with endlsn 0/4000000, output_written_lsn 0/30005D8, fsync_lsn 0/30005D8 (slot test)

...

bash-4.4$ /usr/pgsql-17/bin/pg_ctl -D /var/lib/pgsql/17/data/ stop
waiting for server to shut down.............................................................. failed
pg_ctl: server does not shut down

2026-06-19 10:07:34.839 UTC [18182] LOG:  received fast shutdown request
2026-06-19 10:07:34.846 UTC [18182] LOG:  aborting any active transactions
2026-06-19 10:07:34.849 UTC [18182] LOG:  background worker "logical replication launcher" (PID 18189) exited with exit code 1
2026-06-19 10:07:34.850 UTC [18184] LOG:  shutting down
```

Fix!

- Just respect the protocol!
- Postgres sends the correct messages sequentially. The LSN sent should be assumed to be correct
- Make sure to **test the whole lifecycle of postgres** if you use custom-made subscribers!

PERCONA

Bonus

PERCONA®

```
yum install -y git gcc bison wal2json_17 make flex

git clone https://github.com/postgres/postgres.git
cd postgres/
git checkout REL_17_7
./configure --without-icu --without-readline --without-zlib
cd src/bin/pg_basebackup/

vim pg_recvlogical.c +469
=> comment the line

make

./pg_recvlogical --dbname=postgres --host=/run/postgresql -I '0/0' -S
test -P wal2json --option=format-version=2
--option=add-tables=public.t --option=actions=insert
--option=include-lsn=true -v --create-slot -f /dev/null


./pg_recvlogical --dbname=postgres --host=/run/postgresql -I '0/0' -S
test -P wal2json --option=format-version=2
--option=add-tables=public.t --option=actions=insert
--option=include-lsn=true -v --start -f /dev/null
```

```
psql
ALTER SYSTEM SET wal_level='logical';
pg_ctl -D /var/lib/pgsql/17/data restart


psql
create table t(id integer generated always as identity primary key);

-- once the slot is created
insert into t(id) values (default);

select pg_switch_wal();
exit;


pg_ctl -D /var/lib/pgsql/17/data stop

-- should hang
```


PERCONA