Deconstructing an Abstraction to Reconstruct an Outage

sinjo.dev
A familiar story
Add checkout endpoint #279

Open

Sinjo wants to merge 1 commit into main from add-checkout-endpoint
DB::ConnectionFailure – could not connect to server: Connection refused
Hi
Infra Engineer
Databases & Distributed Systems 😍
GoCardless
Deconstructing an Abstraction to Reconstruct an Outage
First:
Our cluster setup
API backend

Postgres
API backend

Postgres

Pacemaker

Pacemaker

VIP

Postgres

Repl

Repl

Postgres

Postgres

Pacemaker

Pacemaker
Postgres

API backend

Postgres

Pacemaker

VIP

Postgres

Postgres

Pacemaker

Pacemaker

Repl
Note:

One replica always synchronous
So...
So...Unfortunately...
Except it didn't
Our API was down
Fallback: fully manual setup
Postgres

Pacemaker

VIP

API backend

Repl
API backend

Postgres

Pacemaker

Postgres

Pacemaker

Postgres

Pacemaker

Repl

Repl
We're safe, for now...
But only one failure away from downtime
Mission:
Recreate the outage
There's a lot

We'll go step-by-step
1. RAID array loses disks

2. Kernel sets file system read-only

3. Pacemaker detects primary failure

4. Synchronous replica crash

5. Suspicious log on synchronous replica
1. RAID array loses disks
2. Kernel sets filesystem read-only
1. RAID array loses disks
2. Kernel sets filesystem read-only
3. Pacemaker detects primary failure
1. RAID array loses disks
2. Kernel sets filesystem read-only
3. Pacemaker detects primary failure
4. Synchronous replica crash
1. RAID array loses disks
2. Kernel sets filesystem read-only
3. Pacemaker detects primary failure
4. Synchronous replica crash
5. Suspicious log on synchronous replica
Suspicious log on synchronous replica

2023-02-24 17:23:01 GMT LOG: restored log file "00000002000000000000000000000003" from archive

2023-02-24 17:23:02 GMT LOG: invalid record length at 0/3000180
1. RAID array loses disks
2. Kernel sets filesystem read-only
3. Pacemaker detects primary failure
4. Synchronous replica crash
5. Suspicious log on synchronous replica
1. RAID array loses disks

2. Kernel sets filesystem read-only

3. Pacemaker detects primary failure

4. Synchronous replica crash

5. Suspicious log on synchronous replica
1. RAID array loses disks

2. Kernel sets filesystem read-only

3. Pacemaker detects primary failure

4. Synchronous replica crash

5. Suspicious log on synchronous replica
Everyone's favourite fault-injection tool
You know it well...
NAME
  kill – terminate or signal a process

SYNOPSIS
  kill [-s signal_name] pid ...
  kill -l [exit_status]
  kill -signal_name pid ...
  kill -signal_number pid ...

DESCRIPTION
  The kill utility sends a signal to the processes specified by the pid operands.

  Only the super-user may send signals to other users' processes.

  The options are as follows:
# on primary - hard kill

```
kill -SIGKILL <main_pid>
```
# on primary - hard kill
kill -SIGKILL <main_pid>

# on synchronous replica - subprocess crash
kill -SIGABRT <subprocess_pid>
We kept our expectations low...
...which was the right choice
1. RAID array loses disks
2. Kernel sets filesystem read-only
3. Pacemaker detects primary failure
4. Synchronous replica crash
5. Suspicious log on synchronous replica
1. RAID array loses disks
2. Kernel sets filesystem read-only
3. Pacemaker detects primary failure
4. Synchronous replica crash
5. Suspicious log on synchronous replica
Suspicious log on synchronous replica

2023-02-24 17:23:01 GMT LOG: restored log file "00000002000000000000000003" from archive

2023-02-24 17:23:02 GMT LOG: invalid record length at 0/3000180
What do we mean by "log"?
What we normally mean by logs
A different kind of log: binary logs
Some extremely boring SQL

```sql
INSERT INTO users VALUES ('codd');
INSERT INTO users VALUES ('lovelace');
INSERT INTO users VALUES ('turing');
```
Warning: simplifying lie ahead
A different kind of logs
(if they were textual)

INSERT INTO users VALUES ('codd');
INSERT INTO users VALUES ('lovelace');
INSERT INTO users VALUES ('turing');

Wrote 'codd' into table 'users'
Wrote 'lovelace' into table 'users'
Wrote 'turing' into table 'users'
Postgres calls these "Write Ahead Logs" (WALs)
But why bother doing that?
Crash safety
<table>
<thead>
<tr>
<th>id</th>
<th>username</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>codd</td>
</tr>
<tr>
<td>2</td>
<td>lovelace</td>
</tr>
<tr>
<td>id</td>
<td>username</td>
</tr>
<tr>
<td>----</td>
<td>----------</td>
</tr>
<tr>
<td>1</td>
<td>codd</td>
</tr>
<tr>
<td>2</td>
<td>lovelace</td>
</tr>
<tr>
<td>3</td>
<td>turing</td>
</tr>
<tr>
<td><strong>Index</strong></td>
<td><strong>Table</strong></td>
</tr>
<tr>
<td>-----------</td>
<td>-----------</td>
</tr>
<tr>
<td>id</td>
<td>id</td>
</tr>
<tr>
<td>1</td>
<td>username</td>
</tr>
<tr>
<td>2</td>
<td>codd</td>
</tr>
<tr>
<td></td>
<td>lovelace</td>
</tr>
<tr>
<td></td>
<td>turing</td>
</tr>
<tr>
<td>id</td>
<td>username</td>
</tr>
<tr>
<td>----</td>
<td>-----------</td>
</tr>
<tr>
<td>1</td>
<td>codd</td>
</tr>
<tr>
<td>2</td>
<td>lovelace</td>
</tr>
<tr>
<td>3</td>
<td>turing</td>
</tr>
</tbody>
</table>
We can **replay** this operation

```
INSERT INTO users VALUES ('codd');
INSERT INTO users VALUES ('lovelace');
INSERT INTO users VALUES ('turing');
```

Wrote 'codd' into table 'users'
Wrote 'lovelace' into table 'users'
Wrote 'turing' into table 'users'
## Index

<table>
<thead>
<tr>
<th>id</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>???</td>
</tr>
</tbody>
</table>

## Table

<table>
<thead>
<tr>
<th>id</th>
<th>username</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>codd</td>
</tr>
<tr>
<td>2</td>
<td>lovelace</td>
</tr>
<tr>
<td>3</td>
<td>turing</td>
</tr>
<tr>
<td>Index</td>
<td>Table</td>
</tr>
<tr>
<td>-------</td>
<td>-------</td>
</tr>
<tr>
<td>id</td>
<td>id</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>username</td>
</tr>
<tr>
<td></td>
<td>codd</td>
</tr>
<tr>
<td></td>
<td>lovelace</td>
</tr>
<tr>
<td></td>
<td>turing</td>
</tr>
</tbody>
</table>
Also:

replication
Suspicious log on synchronous replica

2023-02-24 17:23:01 GMT LOG: restored log file "000000020000000000000003" from archive

2023-02-24 17:23:02 GMT LOG: invalid record length at 0/3000180
WAL archival

Primary

archive_command

Replica

restore_command
Issue restoring WAL

→

Cause of failure to promote replica?
We already had those writes!
Just because something shouldn't happen doesn't mean it didn't happen.
Suspicious log on synchronous replica

2023-02-24 17:23:01 GMT LOG: restored log file "00000002000000000000000003" from archive

2023-02-24 17:23:02 GMT LOG: invalid record length at 0/3000180
I had zero experience working with binary formats.
None of it is magic
We can cheat: Postgres is open source
But!
These techniques also work on closed source software.
We just call that reverse engineering
Let's find the error

$ git checkout REL9_4_26 # we were running 9.4
$ git grep -n "invalid record length"

src/backend/access/transam/xlogreader.c:295: [...]
src/backend/access/transam/xlogreader.c:604: [...]
src/backend/access/transam/xlogreader.c:678: [...]

src/backend/access/transam/xlogreader.c:291-300:
{
    /* XXX: more validation should be done here */
    if (total_len < SizeOfXLogRecord)
    {
        report_invalid_record(state, "invalid record length at %X/%X",
                              (uint32) (RecPtr >> 32), (uint32) RecPtr);
        goto err;
    }
    gotheader = false;
}
Let's find the error

```c
src/backend/access/transam/xlogreader.c:291-300:
{
    /* XXX: more validation should be done here */
    if (total_len < SizeOfXLogRecord)
    {
        report_invalid_record(state, "invalid record length at %X/%X", (uint32) (RecPtr >> 32), (uint32) RecPtr);
        goto err;
    }
    gotheader = false;
}
```
Let's find the error

src/backend/access/transam/xlogreader.c:291-300:

```c
{
    /* XXX: more validation should be done here */
    if (total_len < SizeOfXLogRecord)
    {
        report_invalid_record(state, "invalid record length at %X/%X",
                                (uint32) (RecPtr >> 32), (uint32) RecPtr);
        goto err;
    }
gotheader = false;
}
```
Let's find the error

src/include/access/xlog.h:58:

#define SizeOfXLogRecord MAXALIGN(sizeof(XLogRecord))
Wouldn't it be convenient if we could make `total_len == 0`?
Let's find the error

```c
src/backend/access/transam/xlogreader.c:272-273:

record = (XLogRecord *) (state->readBuf + RecPtr % XLOG_BLCKSZ);
total_len = record->xl_tot_len;
```
Let's find the error
src/include/access/xlog.h:41-56:

typedef struct XLogRecord
{
    uint32         xl_tot_len;    /* total len of entire record */
    TransactionId xl_xid;        /* xact id */
    uint32         xl_len;        /* total len of rmgr data */
    uint8          xl_info;       /* flag bits, see below */
    RmgrId         xl_rmid;       /* resource manager for this record */
    /* 2 bytes of padding here, initialize to zero */
    XLogRecPtr     xl_prev;       /* ptr to previous record in log */
    pg_crc32       xl_crc;        /* CRC for this record */
}

/* If MAXALIGN==8, there are 4 wasted bytes here */

/* ACTUAL LOG DATA FOLLOWS AT END OF STRUCT */
typedef struct XLogRecord
{
    uint32 xl_tot_len;      /* total len of entire record */
    TransactionId xl_xid;   /* xact id */
    uint32 xl_len;          /* total len of rmgr data */
    uint8 xl_info;          /* flag bits, see below */
    RmgrId xl_rmid;         /* resource manager for this record */
    /* 2 bytes of padding here, initialize to zero */
    XLogRecPtr xl_prev;     /* ptr to previous record in log */
    pg_crc32 xl_crc;        /* CRC for this record */

    /* If MAXALIGN==8, there are 4 wasted bytes here */

    /* ACTUAL LOG DATA Follows AT END OF STRUCT */
} XLogRecord;
src/backend/access/transam/xlogreader.c:291-300:

```c
{  /* XXX: more validation should be done here */
  if (total_len < SizeOfXLogRecord)
  {
    report_invalid_record(state, "invalid record length at %X/%X",
                        (uint32) (RecPtr >> 32), (uint32) RecPtr);
    goto err;
  }
  gotheader = false;
}
```

What was that check doing?
What was that check doing?

Size the record says it is

src/backend/access/transam/xlogreader.c:291-300:

```c
{  /* XXX: more validation should be done here */
if (total_len < SizeOfXLogRecord)
{
    report_invalid_record(state, "invalid record length at %X/%X",
                        (uint32) (RecPtr >> 32), (uint32) RecPtr);
    goto err;
}
goheader = false;
}
```

Smallest possible size it can be
A different kind of logs
(if they were textual)

INSERT INTO users VALUES ('codd');
INSERT INTO users VALUES ('lovelace');
INSERT INTO users VALUES ('turing');

Wrote 'codd' into table 'users'
Wrote 'lovelace' into table 'users'
Wrote 'turing' into table 'users'
Let's see what they look like in practice
Some extremely boring SQL

```
INSERT INTO users VALUES ('codd');
INSERT INTO users VALUES ('lovelace');
INSERT INTO users VALUES ('turing');
```
Grab the binary log file, and...
A barely comprehensible wall of data 😅

<table>
<thead>
<tr>
<th>0</th>
<th>7ED00600</th>
<th>02000000</th>
<th>00000003</th>
<th>00000000</th>
<th>00000000</th>
<th>00000000</th>
<th>00000000</th>
<th>18A001B0</th>
<th>5A98B159</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>00000001</td>
<td>02000000</td>
<td>3C000000</td>
<td>00000000</td>
<td>01000000</td>
<td>10000000</td>
<td>00000000</td>
<td>C8460102</td>
<td>00000000</td>
</tr>
<tr>
<td>64</td>
<td>69CB39DE</td>
<td>00000000</td>
<td>7F060000</td>
<td>682F0000</td>
<td>7C2E0000</td>
<td>00000000</td>
<td>00000000</td>
<td>01000000</td>
<td>00000000</td>
</tr>
<tr>
<td>96</td>
<td>04000700</td>
<td>00000000</td>
<td>40000000</td>
<td>A9020000</td>
<td>20000000</td>
<td>80A0000</td>
<td>28000000</td>
<td>00000000</td>
<td></td>
</tr>
<tr>
<td>128</td>
<td>FCCF5539</td>
<td>00000000</td>
<td>7F060000</td>
<td>682F0000</td>
<td>04000000</td>
<td>00000000</td>
<td>01003E1F</td>
<td>00010002</td>
<td>00000000</td>
</tr>
<tr>
<td>160</td>
<td>0B18000B</td>
<td>636F6464</td>
<td>20000000</td>
<td>A9020000</td>
<td>0C000000</td>
<td>06010000</td>
<td>68000000</td>
<td>00000000</td>
<td></td>
</tr>
<tr>
<td>192</td>
<td>5867F1A1</td>
<td>00000000</td>
<td>BB000001</td>
<td>09802000</td>
<td>00000000</td>
<td>00000000</td>
<td>44000000</td>
<td>AA020000</td>
<td>00000000</td>
</tr>
<tr>
<td>224</td>
<td>24000000</td>
<td>0000A000</td>
<td>A8000003</td>
<td>00000000</td>
<td>16F29852</td>
<td>00000000</td>
<td>7F060000</td>
<td>682F0000</td>
<td>00000000</td>
</tr>
<tr>
<td>256</td>
<td>00400000</td>
<td>00000000</td>
<td>02003E1F</td>
<td>00010002</td>
<td>08180013</td>
<td>6C6F7665</td>
<td>6C616365</td>
<td>00000000</td>
<td>00000000</td>
</tr>
<tr>
<td>288</td>
<td>02000000</td>
<td>AA020000</td>
<td>0C000000</td>
<td>60010000</td>
<td>D8000003</td>
<td>00000000</td>
<td>5B36701C</td>
<td>00000000</td>
<td>00000000</td>
</tr>
<tr>
<td>320</td>
<td>FDE40802</td>
<td>C0980200</td>
<td>00000000</td>
<td>00000000</td>
<td>42000000</td>
<td>A8020000</td>
<td>22000000</td>
<td>0000A000</td>
<td>00000000</td>
</tr>
<tr>
<td>352</td>
<td>20010003</td>
<td>00000000</td>
<td>E5D39201</td>
<td>00000000</td>
<td>7F060000</td>
<td>6B2F0000</td>
<td>00400000</td>
<td>00000000</td>
<td>00000000</td>
</tr>
<tr>
<td>384</td>
<td>03003E1F</td>
<td>00010002</td>
<td>0818000F</td>
<td>74757269</td>
<td>6E670000</td>
<td>00000000</td>
<td>20000000</td>
<td>A8020000</td>
<td>00000000</td>
</tr>
<tr>
<td>416</td>
<td>00000000</td>
<td>60010000</td>
<td>50010003</td>
<td>00000000</td>
<td>439DD8EC</td>
<td>00000000</td>
<td>6AFC1002</td>
<td>C0980200</td>
<td>00000000</td>
</tr>
</tbody>
</table>
A barely comprehensible wall of data 😅

Hex

ASCII
Same data, rendered differently

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Hexadecimal</th>
<th>Character</th>
</tr>
</thead>
<tbody>
<tr>
<td>62</td>
<td>3E</td>
<td>&gt;</td>
</tr>
<tr>
<td>63</td>
<td>3F</td>
<td>?</td>
</tr>
<tr>
<td>64</td>
<td>40</td>
<td>@</td>
</tr>
<tr>
<td>65</td>
<td>41</td>
<td>A</td>
</tr>
<tr>
<td>66</td>
<td>42</td>
<td>B</td>
</tr>
<tr>
<td>Decimal</td>
<td>Hexadecimal</td>
<td>Character</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
<td>-----------</td>
</tr>
<tr>
<td>62</td>
<td>3E</td>
<td>&gt;</td>
</tr>
<tr>
<td>63</td>
<td>3F</td>
<td>?</td>
</tr>
<tr>
<td>64</td>
<td>40</td>
<td>@</td>
</tr>
<tr>
<td>65</td>
<td>41</td>
<td>A</td>
</tr>
<tr>
<td>66</td>
<td>42</td>
<td>B</td>
</tr>
</tbody>
</table>
Some good news
Some good news

We can see our users!!
How can we find xl_tot_len?
Some even more boring SQL

```
INSERT INTO repro VALUES ('A');
INSERT INTO repro VALUES ('AB');
INSERT INTO repro VALUES ('ABC');
INSERT INTO repro VALUES ('ABCD');
INSERT INTO repro VALUES ('ABCDE');
...```

Look for a field increasing by 1
Guesswork incoming!
The data we inserted
## A little help: ASCII codes

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Hexadecimal</th>
<th>Character</th>
</tr>
</thead>
<tbody>
<tr>
<td>62</td>
<td>3E</td>
<td>&gt;</td>
</tr>
<tr>
<td>63</td>
<td>3F</td>
<td>?</td>
</tr>
<tr>
<td>64</td>
<td>40</td>
<td>@</td>
</tr>
<tr>
<td>65</td>
<td>41</td>
<td>A</td>
</tr>
<tr>
<td>66</td>
<td>42</td>
<td>B</td>
</tr>
</tbody>
</table>
Notice anything?

The data we inserted
Notice anything?

The data we inserted

Familiar characters
Notice anything?

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Hexadecimal</th>
<th>Character</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>3F</td>
<td>?</td>
</tr>
<tr>
<td>64</td>
<td>40</td>
<td>@</td>
</tr>
<tr>
<td>65</td>
<td>41</td>
<td>A</td>
</tr>
</tbody>
</table>

Familiar characters

The data we inserted
Notice anything?

The data we inserted

Familiar characters
Notice anything?

Familiar characters

The data we inserted
Notice anything?

Familiar characters (hex)

The data we inserted (hex)

The data we inserted
Wouldn't it be convenient if we could make $total\_len == 0$?
We could import the Postgres structs and do this properly...
...or we *could* write a regex 🤔
Let's write a regex
wal_file_name = ARGV[0]
puts wal_file_name

wal_contents = IO.read(wal_file_name, encoding: "BINARY")
hex = wal_contents.unpack("H*").first
replaced = hex.gsub(/3f(000000.+41424300)/, "00\1")
bindata = [replaced].pack("H*")
File.write(wal_file_name + ".broken", bindata)
wal_file_name = ARGV[0]
puts wal_file_name

wal_contents = IO.read(wal_file_name, encoding: "BINARY")
hex = wal_contents.unpack("H*").first
replaced = hex.gsub(/3f(000000.+41424300)/, "00\1") # Replaces 'ABC' size
bindata = [replaced].pack("H*")
File.write(wal_file_name + ".broken", bindata)
And if we give it to a Postgres replica?
We reproduced the error!

2023-02-28 19:24:11 GMT LOG: restored log file "00000002000000000000000003" from archive

2023-02-28 19:24:11 GMT LOG: invalid record length at 0/3000148
Success 😄
Success, with a caveat...😔
This wasn't enough to reproduce the outage
1. RAID array loses disks

2. Kernel sets filesystem read-only

3. Pacemaker detects primary failure

4. Synchronous replica crash

5. Suspicious log on synchronous replica
1. RAID array loses disks
2. Kernel sets filesystem read-only
3. Pacemaker detects primary failure
4. Synchronous replica crash
5. Suspicious log on synchronous replica
6. ...
1. RAID array loses disks
2. Kernel sets filesystem read-only
3. Pacemaker detects primary failure
4. Synchronous replica crash
5. Suspicious log on synchronous replica
6. Backup VIP on synchronous replica
We added it to the cluster
Ran the repro script
Success
(no caveats)
😄
but... why?
Background: how Pacemaker schedules resources
2 relevant settings
By default: reschedule without penalty
API backend

Postgres

Pacemaker

VIP

Postgres

Pacemaker

Postgres

Pacemaker

Repl

Repl

Repl
Setting: default-resource-stickiness
By default: resources can run anywhere.
Setting: colocation
default-resource-stickiness = 100

&

colocation -inf: BackupVIP Primary
default-resource-stickiness = 100

&

colocation -inf: BackupVIP Primary
default-resource-stickiness = 100

&

colocation -inf: BackupVIP Primary
A very subtle semantic difference
"Avoid scheduling these together"
"Avoid scheduling these together"

"Literally never schedule these together"
default-resource-stickiness = 100
&
colocation -inf: BackupVIP Primary
default-resource-stickiness = 100

&

colocation -1000: BackupVIP Primary
Failover works properly
P.S. The WAL error was a red herring
Sorry
I know it was the most interesting part
and it would have been kinda cool
but it was part of the debugging process
1. RAID array loses disks

2. Kernel sets filesystem read-only

3. Pacemaker detects primary failure

4. Synchronous replica crash

5. Suspicious log on synchronous replica

6. Backup VIP on synchronous replica
What can we learn?
None of the stack is magic
None of the stack is magic 😁
None of the stack is magic 🤪
"It's just someone else's computer"
"It's just someone else's abstraction"
Read other people's code...
...and try to modify it
Automation erodes knowledge.
Game days are a partial fix
"What if we had to recover our database server manually?"
Don't stop questioning your repro
1. No magic in the stack
2. Automation *erodes* knowledge
3. Always *question* the repro
JSON over HTTP
Binary formats are coming to web development.
Protobuf over HTTP/2
Protobuf (e.g. gRPC) over HTTP/2
It's worth getting familiar
One last thing to ask of you
Most computing happens successfully.
The 0.00000001% * not a real statistic
Outsized negative impact
It's a shame not to learn.
"We noticed a problem."

"We fixed the problem."

"We'll make sure the problem doesn't happen again."
3 good examples
Intermittent downtime from repeated crashes

On Friday 18th November 2022 we experienced 13 minutes of downtime over a period of 32 minutes from 15:40 to 16:12 GMT.

At incident.io, our customers depend on us to be available even when everything else is down, so we take incidents like this seriously. We also sincerely believe in the power of transparency, and want to be as open as possible in these situations, sharing what details we can.

https://incident.io/blog/intermittent-downtime
On January 31st 2017, we experienced a major service outage for one of our products, the online service GitLab.com. The outage was caused by an accidental removal of data from our primary database server.

https://about.gitlab.com/blog/2017/02/10/postmortem-of-database-outage-of-january-31/
Please Share the difficult stories too
Thank you
Image credits


• Pouring Latte Art - Craft Coffee Spot - CC-BY - https://www.flickr.com/photos/195403219@N08/52200966448/

• microscope - Milosz1 - CC-BY - https://www.flickr.com/photos/mikolski/3269906279

• Hard Disk Guts - CC-BY - https://www.flickr.com/photos/mattandkim/97533589/

• Corsair ForceGT 180GB - CC-BY - https://www.flickr.com/photos/ruocaled/8173124575/
Image credits

• Server - The Noun Project (via WikiMedia) - CC0 - https://commons.wikimedia.org/wiki/File:Server_-_The_Noun_Project.svg

• Rope - Robo Android - CC-BY - https://www.flickr.com/photos/49140926@N07/6798304070/

• Stargazing - Max Delaquis - CC-BY - https://www.flickr.com/photos/115000114@N07/28861043652
Questions?

✌❤

sinjo.dev

@planetscaledata