

[PGCONF.EU 2026]

The Primary Is Down. Are You Sure?

The most critical decision-making time

Furkan Sahin

Ubicloud

Valencia, 20 October 2026

[THE QUESTION]

We cannot reach the primary



The primary is down. Are you sure?

Managed Postgres on local NVMe

Local NVMe disks for performance.

01

The data lives on the machine

02

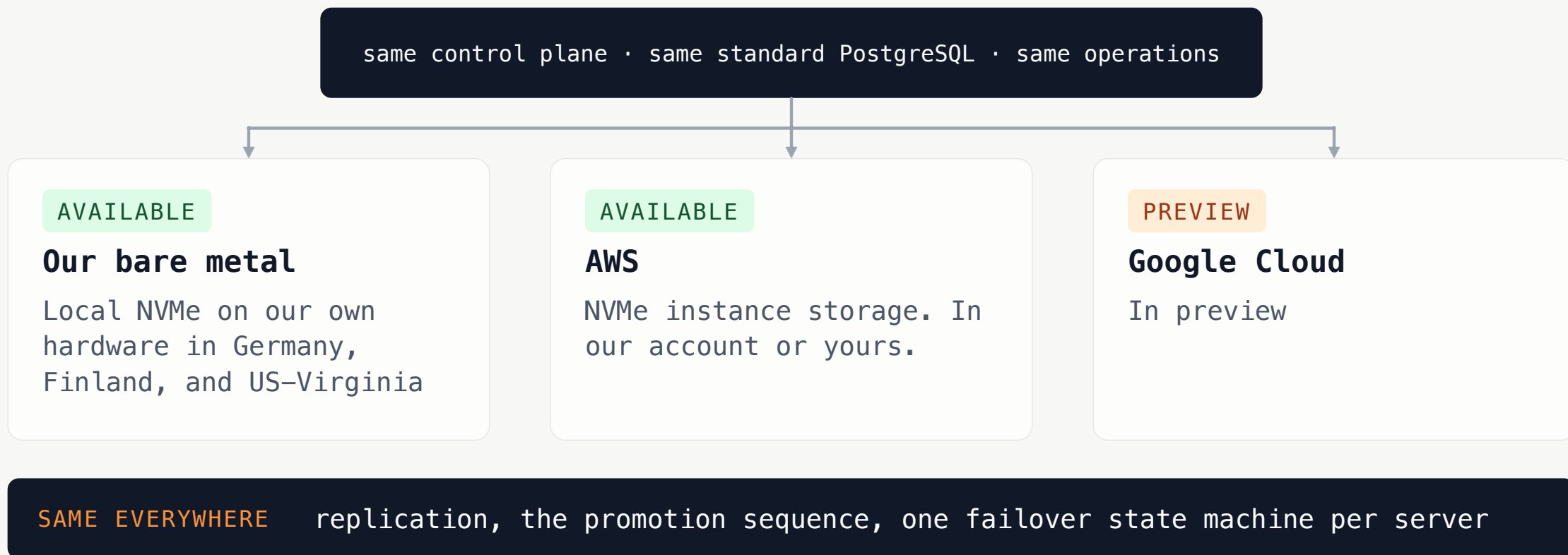
When the machine dies, its disk goes with it

03

So we rely on standbys and the WAL archive, and failover has to work

[THREE PLACES]

One Postgres on three different platforms.



[THE TEAM]

Where we come from

Heroku Postgres

Citus Cloud

Azure Postgres

Crunchy Bridge

So failover should work the same everywhere, right?

But no!

one step in the sequence is different

PER PROVIDER its own lockout mechanism, and on AWS and GCP a second NIC

Also per provider, after the promote: the DNS record type and the old VM's cleanup.

[SEVEN STEPS]

One failover sequence



Same code on bare metal, AWS and GCP. The standby waits for the fence step to finish.

How we detect a failure

Monitor

every ~5 s, over SSH, through a forwarded Unix socket

```
pulse[:reading] == "down" && pulse[:reading_rpt] > 5 && Time.now - pulse[:reading_chg] > 30
```

SSH works but Postgres is silent 6 failed checks in a row, over more than 30 s

SSH session fails

flag raised on the first failure

Confirm

fresh SSH, SELECT 1, then “redo in progress” in a recent log?

Picking the standby

Candidates	only standbys idle in “wait”
------------	------------------------------

```
[slot_score, lsn2int(it[:lsn]), recycle_score, type_score, rank_score]
```

- | | |
|-----------------|---|
| 1. Slot | its own replication slot on the current primary |
| 2. WAL | the most received: <code>pg_last_wal_receive_lsn()</code> |
| 3. Tie breakers | not being replaced, not a fallback type, newer family |

```
lsn_diff(lsn, target[:lsn]) > 80 * 1024 * 1024 # 80 MB or ~5 WAL files
```

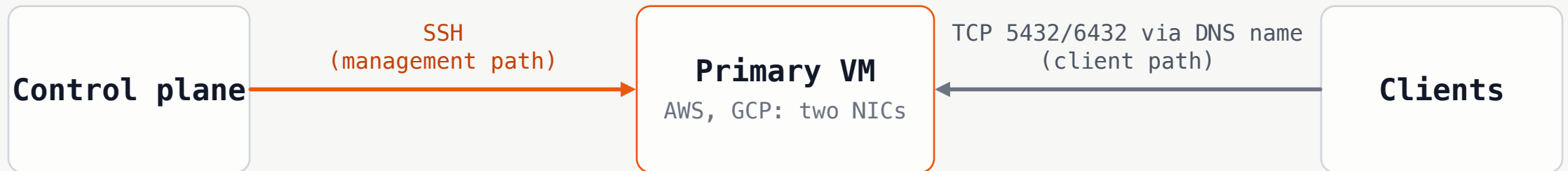
80 MB guard	one async standby only
-------------	------------------------

Promote and switch DNS

DNS only moves new connections.

- 1 The standby polls every 1 s for “locked out”
- 2 `SELECT pg_promote(true, 300)`
- 3 Flip the primary in our database, mark the old one for destroy
- 4 DNS: new record, TTL 10 s
- 5 Reconfigure all servers, fill the WAL archive

Is it really down?



- | | |
|---|------------------------|
| 1. Guest wedged | both paths dead |
| 2. sshd or management path broken | clients still use it |
| 3. Partition between control plane and VM | looks like case 2 |
| 4. Client path broken, SSH fine | we see up, no failover |

What happens with two primaries

Writes	commits on the old primary after the promote are lost, we destroy it with no rewind
Replication	a standby that kept following the old primary can fork off and must be rebuilt
Archive	same bucket, timeline IDs prevent overwrites, old-timeline WAL can keep arriving

“...the original server on that timeline is still running and archives the completed version of the same segment later...”

`xlog.c`

What a fence has to do

If you cannot make sure it is down,
make sure it stops accepting writes.

Sessions

cut the open ones
and the new ones

Replication

no standby follows it
past the fork

Outbound

no WAL reaches the archive

No guest needed

works on a wedged VM

Reboot

still fenced after a reboot

Confirmed, in time

if no fence worked in time,
page a human, don't promote

In-guest tools need a healthy guest

pg_stop and hba both run over SSH into the guest.

```
timeout 10 sudo pg_ctlcluster :version main stop -m immediate  
ExecStop=/usr/bin/pg_ctlcluster --skip-systemctl-redirect -m fast %i stop
```

“immediate”	run as root under systemd, it becomes a fast stop
pg_hba	matches new connection attempts, open sessions keep writing, replication still allowed
Reboot	start.conf is auto, a stopped cluster starts again at the next boot

How our lockout works

Environment	Lockout mechanisms
Bare metal	pg_stop, hba, host_routing
AWS	pg_stop, hba, detach_nic
GCP	pg_stop, hba, delete_nic

```
resource.lockout_mechanisms.each do |mechanism|  
  bud Prog::Postgres::PostgresLockout,  
    {"mechanism" => mechanism}  
end
```

One child job per mechanism but all in parallel
each one reports lockout_succeeded or lockout_failed

Locked out	when every child has finished, whatever they reported
-------------------	---

All failed	the standby still promotes, fail-open on purpose
-------------------	---

[STEP 4 · THREE TOOLBOXES]

Fencing in three environments

A fence can only use what the environment gives us.

Bare metal

we own the host

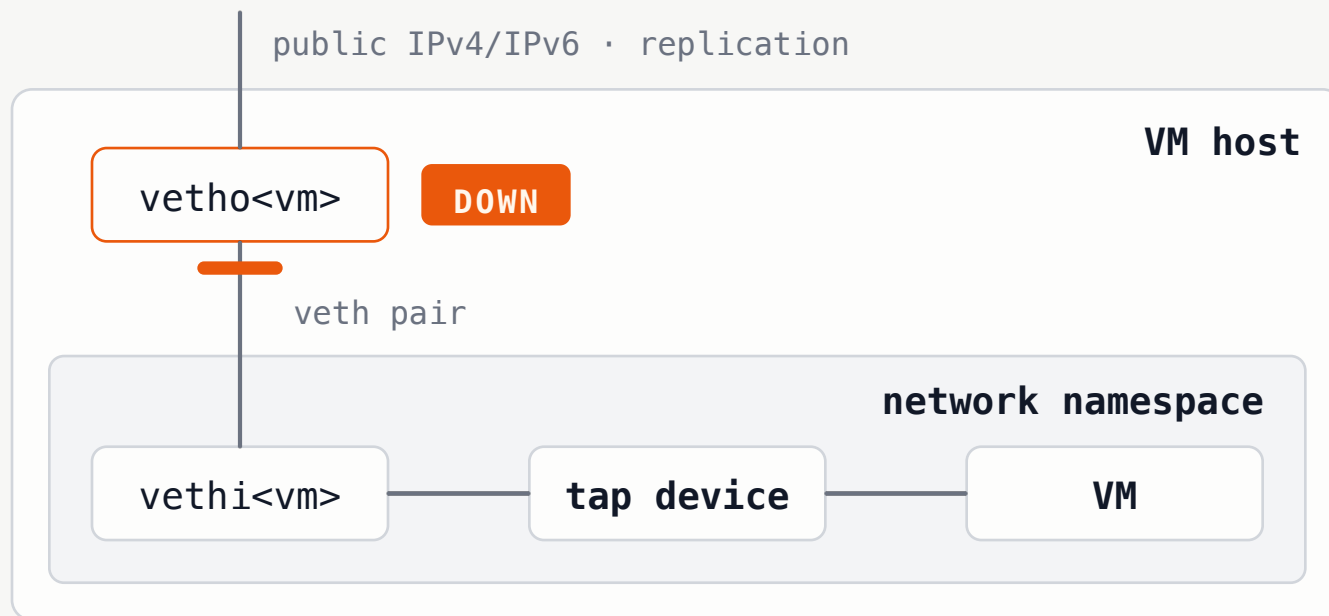
AWS

the EC2 API

GCP

the Compute Engine API

Fencing on bare metal



```
timeout 10 sudo ip link set vetho<vm> down
```

Runs on

the VM host, over SSH to the host

Cuts

clients, replication, WAL archive, our SSH into the VM

Needs from the guest

nothing

Limit

the host has to answer

Fencing on AWS: the problem

stop-instances --force	slow, waits for a graceful shutdown first
-------------------------------	---

Security groups	stateful, tracked connections stay
------------------------	------------------------------------

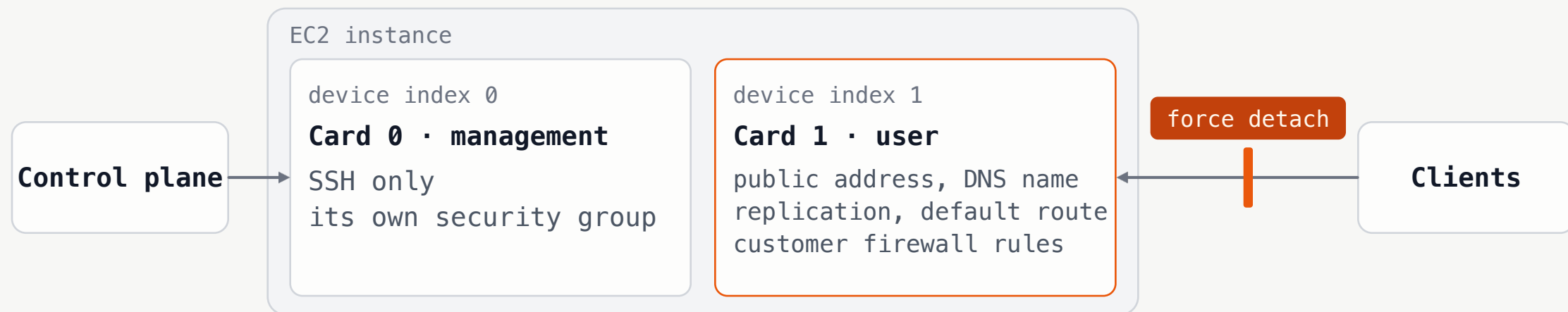
Network ACLs	more complicated, extra resources to clean up
---------------------	---

Detach the NIC	
-----------------------	--

	<i>"You can't detach a primary network interface from an instance."</i>
--	---

	AWS EC2 User Guide
--	--------------------

Fencing on AWS: two NICs



```
client.detach_network_interface(  
    attachment_id: network_interface.attachment.attachment_id,  
    force: true,  
)
```

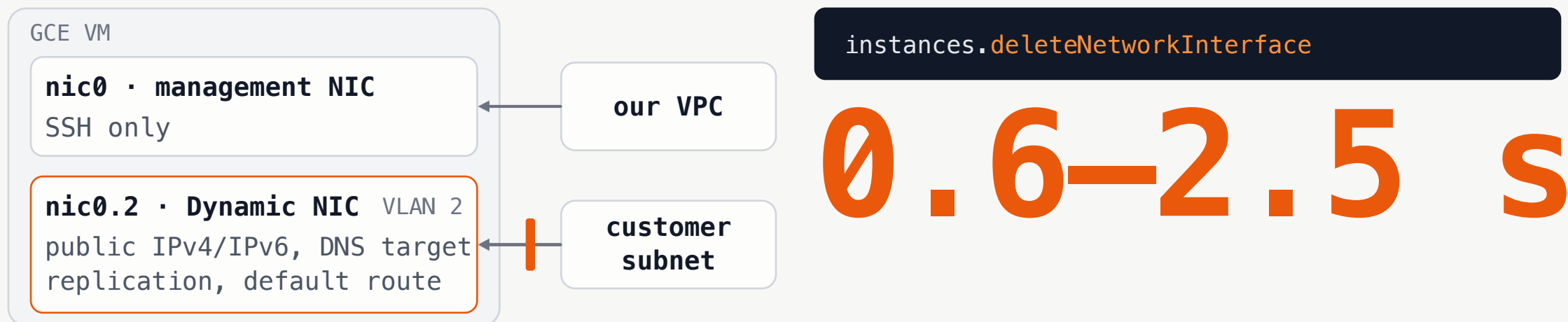
After the fence

card 0 stays, SSH and cleanup still work

Fencing on GCP: what doesn't work

Option	Why it is not a fence
Firewall deny	only new connections, existing ones persist
Remove the external IP	replication continues, our SSH is lost
Stop	the guest gets up to 120 s to shut down first
Reset	cut in about 1 s, but back in 12.6–14.6 s, and Postgres starts as a primary
Second regular NIC	only at instance creation

Fencing on GCP: Dynamic NICs



After the fence

`nic0` stays, SSH and cleanup still work

Egress

no internet and no Google APIs, so no WAL reaches the archive

The three fences compared

Fence	Without the guest	Open sessions	Replication, WAL	Fenced after reboot
Metal veth down on the host	yes (host)	cut	cut	veth stays down
AWS force-detach the user NIC	yes (EC2 API)	cut	cut	NIC stays detached
GCP delete the Dynamic NIC	yes (Compute API)	cut	cut	NIC is gone

The WAL nobody archived

- 1 `archive_mode = on`: no archiver on the standby `STANDBY pg_wal`
- 2 The walreceiver marks each finished segment `.done`
- 3 The archiver only picks up `.ready`
- 4 Checkpoints after the promote recycle `.done` segments

A promoted standby “will not archive any WAL or timeline history files that it did not generate itself.”

PostgreSQL docs

ARCHIVE

EXAMPLE

...92
archived by the old primary

...93 to ...96
marked `.done`

missing:
...93 to ...96

after the promote

00000002.history

...97 (end of WAL)

...97.partial

00000002...97 onward

Our first fix: backfill after promote

- 1 Promote
- 2 Backfill: `wal-g wal-push` the old-timeline segments still in `pg_wal` (newest 128)
- 3 Meanwhile: checkpoints recycle `.done` segments outside `wal_keep_size`

Why after	"so it adds nothing to the outage"
-----------	------------------------------------

Standbys	<code>wal_keep_size</code> 2GB (512MB on burstable)
----------	---

Result	the backfill can lose the race
--------	--------------------------------

Handing the segments to the archiver

- 1 Still in recovery: rename `.done` to `.ready` for complete received segments
- 2 `SELECT pg_promote(true, 300)`
- 3 Recovery ends, the archiver starts: history file first, then the oldest segment
- 4 `.ready` keeps each segment on disk until it is archived

```
File.rename(File.join(@path, "#{segment}.done"), File.join(@path, "#{segment}.ready"))
```

[NOT FAILING OVER]

When we do not fail over

SELECT 1 answers

over a fresh SSH connection

Crash recovery

“redo in progress” in a recent log

Upgrade in progress

the primary is stopped on purpose, its fenced state ignores the monitor’s flag

Async standby behind

more than 80 MB, or no sample of the primary

No healthy standby

nobody idle to promote

Failover already running

one at a time

Nobody to promote

retry the failover and restart Postgres every 5 s, mark for replacement, page at 5 min

NOT ON THIS LIST

“we could not fence it”: today we fail over anyway

[TAKEAWAYS]

Assumptions about your infrastructure

“It is down”

only on the path you check, and ours is SSH

“We stopped Postgres, so it is fenced”

only with SSH, a healthy guest and no reboot

“We can pull its NIC”

only a NIC you planned: not index 0 on AWS,
and on GCP, regular NICs are set at creation

“The archive has every segment”

only if the old primary archived it before it went away

Which of your failover assumptions are really assumptions about your infrastructure?

[THANK YOU]

Thank you. Questions?

ubicloud.com

github.com/ubicloud/ubicloud

open source

github.com/ubicloud/ubicloud/pull/6450

WAL fix
