NetApp Instaclustr

White Paper

LESSONS LEARNED

In PostgreSQL®

Table of contents

OVERVIEW

If ever there was a list of 'nightmare scenarios' that every business owner would agree upon, very near the top of that list would certainly be 'irrecoverable data loss.'

While many prudent companies have detailed business continuity plans and standard operating procedures in place to minimize the impact when something bad eventually does occur, for whatever reason, others simply do not—leading to a nightmare scenario.

In this white paper, Instaclustr Professional Services Consultant Perry Clark recounts a true story about such a scenario, and how he and his team was able to overcome the daunting challenges presented.

— Perry Clark

ABOUT *The author*

Norfolk, Virginia based Perry Clark is an Instaclustr Professional Services Consultant. He has 15 years of experience in Linux servers, services, and database administration.

Prior to joining Instaclustr, Perry served for many years as the Director of CloudOps and senior database administrator for an ERP/accounting software company. His duties included the administration of global cloud, onsite, and internal systems. He enjoys troubleshooting and performance tuning.

In his free time, he enjoys woodworking, fixing stuff, and restoring his 1965 Mustang Coupe. In the hot summer months, he and his wife, Carolyn, can be found boating around the lower Chesapeake Bay.

> **Ah, Houston, we've had a problem.**
>
> – Jim Lovell

THE *Incident*

"Send 0.5 BTC"

A couple of years ago I received a call from our customer service department with a panicked customer conferenced in. The customer encountered a message from their ERP Client application that their ERP Database "was not found."

We confirmed in their client application that the connection string (the server, port, database name, and user credentials) were correct, and there were no leading or trailing spaces, no typos, proper addresses, and so on.

Indeed, everything appeared to be perfectly fine.

The site administrator I was working with had been at this company for ages. He led their implementation of the ERP system, was responsible for the network and server infrastructure, and overall had a good understanding of how the ERP system worked. I always assumed he was a DBA or IT professional, so I was surprised to learn that he was a chemical engineer.

Given the gravity of the situation, it was time to escalate to deeper remote troubleshooting techniques.

We started a screen share from their workstation and attempted several other diagnostics. Ping returned the proper IP, and the server was responsive. Then I noticed that the workstation we were screen sharing from had PgAdmin4 which is a GUI based PostgreSQL® management and query tool; I recommended we launch that app.

Navigating in PgAdmin4 I could clearly see the database and connect to it as an admin user. The database names were there, so I selected it to check the contents of the schema and tables therein.

What I saw was not good.

Out of our applications' 600+ tables and 1300+ stored procedures, there was 1 simple table remaining titled "warning." Inspecting the contents of that table revealed a row named "warning_text—the content of that row was quite clear:

"Send 0.5 BTC to this address and go to this site abcxyz123lmnop456.onion to recover your database! SQL dump will be available after payment!"

At the time, .5 BTC was about $750 USD (...wish I knew then what I know now.)

I stated, *"This is ransomware. You will not be sending any bitcoin to that address. Your database is not there."* Ask me how I knew... well, that's a whole other story for another article.

About this time, the customer was getting what is termed a 'sinking' feeling. I reassured them that we still had a lot to troubleshoot and not to panic. I asked if we could do a remote session on the database server itself.

That is when the customer revealed: *"Oh, the other day all our Excel and Word documents on the shared drive were encrypted with ransomware but we restored them from a backup. We got hacked because one of our IT Consultants left MS Remote Desktop running unsecured on our main server."*

Well, that was certainly a helpful tidbit!

Now I was questioning the trustworthiness of their infrastructure. In my experience, once a server of infrastructure has been compromised it cannot be trusted—Are you under active attack? Who is attacking? Why us? What has been modified? What has been taken? Did they leave a backdoor? Where has the data gone and how will it be used?

The exercises and feeling one goes through to restore a sense of security are very similar to being burglarized.

With reasonably strong assurances that they had taken measures to resolve their security and infrastructure issues, I asked if they had any backups. *"Oh, yes we have nightly backups!"* was the reply.

"Great! Let's see what state those are in!" I said.

We browsed where the backups were stored and found that they were all extremely small in file size. Basically... ZERO size! Surely, we're in the wrong place and looking at the wrong thing.

Calmly, I asked *"Let's take a look at the backup script."*

Last year we said, 'Things can't go on like this', and they didn't. They got worse.

– Will Rogers

Back up plans

Upon inspection I immediately saw the cause of the problem. The script was calling an older version of pg_dump that was not compatible with the currently running PostgreSQL server version. This binary mismatch occurred when they upgraded their PostgreSQL major version and neglected to check and change the paths to the new version binary for pg_dump in their backup script.

Essentially, when the backup script was called, there was an error logged about the version mismatch that no one paid attention to. No backups had been made for months.

I thought about their remaining options to get them up and running again. From what I could tell, they had two choices, and neither was very appealing:

- Reimplement their ERP system from the ground up, losing much historical data and rebuilding customizations
- Find a backup and rebuild from there, and deal with the potential data issues

By now the CEO and CFO were anxiously and impatiently hovering over my contact on-site. It was too early to manage any expectations, but in my mind I was thinking they were bleak at best.

"Do you have ANY sort of 'good' backup—whole system snapshot, db backup, anything?" I inquired.

"Yes, I have a whole backup of the database directory from about 6 months ago."

Well, this was certainly not great news, but still better than I had expected. I was confident the backup was something like a 'SELECT – COPY – PASTE' from Windows Explorer.

The problem with that is the PostgreSQL server processes were likely running, connections to the database were still happening, and data was moving around. However, copying in this manner is guaranteed to cause a mess. In a perfect world, the proper way to create this backup would have been to use the pg_basebackup utility which can handle copying the files without data loss.

But, alas, we were clearly not in a perfect world. Had they been running on Instaclustr's Managed PostgreSQL service, pg_backrest could have been used to restore to a point in time before they experienced data loss.

By now, several hours had been spent troubleshooting. Feeling somewhat hopeful, I asked for a new database server with the exact same version of PostgreSQL that corresponds to whatever version this $PGDATA directory copy is. Knowing this would take some time, we decided to re-convene the following morning.

In the meantime, I transferred the 'backup' to one of my test servers to work out the viability of using this copied $PGDATA folder. I spent several more hours off-the-clock trying various things with the backup. There were even more problems, but there was a glimmer of hope as well.

> **...there are known knowns; there are things we know we know. We also know there are known unknowns; that is to say we know there are some things we do not know. But there are also unknown unknowns–the ones we don't know we don't know.**
>
> – Donald Rumsfeld

"Is it working yet?"

The next morning, we reconnected on the new server and I had a pretty good idea of what needed to be done. Once the PostgreSQL service was running we would shut it down and swap the $PGDATA directory with the 'backup' version and start up the server and cross our fingers.

At first, everything looked OK according to the server startup process, but when we connected to the database there were no databases listed (or not the ones we expected).

Hmm... something must be corrupted, or we need to re-create the relationship for the server to find the physical location of the database files. I understood the problem, but I needed to find out how that mapping worked. We could create new databases and access those—but something was preventing us from accessing the existing databases.

Clearly, it was time to study some PostgreSQL internals, and time was quickly dwindling.

The customer was burning up their Support hours, their team had put off entering new orders until the system was up and running, and their CEO and CFO were constantly asking *"Is it working yet?"*

I wasn't sure how long it would take to get this resolved and I increasingly felt that the CEO was attempting to blame me for not being able to quickly unwind the mess that they had created.

I knew that the $PGDATA directory location held the 'data files' for the server, but I didn't quite fully understand or appreciate the Data Directory Layout in relation to how the PostgreSQL server uses them.

So, step one was to confirm where PostgreSQL is expecting that directory to be found:

```
postgres=# show data_directory;
       data_directory
_____________________________
/data/erp_server
(1 row)
```

We knew this already because that is where we had copied the 'backup' files to. So, let's look in the /data/erp_server directory

```
drwx------ 27 postgres postgres 4096 Feb 21 11:40 base
drwx------  2 postgres postgres 4096 Feb 21 10:58 global
drwx------  2 postgres postgres 4096 Aug 16  2018 pg_clog
. . .
```

Ah-ha! The 'base' directory looks like it might have some interesting things in it, let's see what is in there:

```
drwx------ 27 postgres postgres   4096 Feb 21 11:40 -
drwx------ 19 postgres postgres   4096 Feb 21 10:57 --
drwx------  2 postgres postgres   4096 Feb 21 10:57 1
drwx------  2 postgres postgres   4096 Feb 21 10:58 12438
drwx------  2 postgres postgres   4096 Feb 21 11:40 12439
drwx------  2 postgres postgres  12288 Feb 21 10:57 16393
drwx------  2 postgres postgres  16384 Mar 20  2019 pgsql_tmp
```

These directory sizes looked exactly like what were expected considering the size of data on disk. This confirmed that we were on the right track. We need to recreate the relationship between these numbered directories and the PostgreSQL server.

But where do we set "/data/erp_server/base/16393 = production_erp"? And why is the directory a number?

On a normally working server if you run:

```
select oid, datname from pg_database order by 1;
```

you will get the mapping of the database name to the numbered directory in $PGDATA/base:

```
  oid   |          datname
--------+---------------------------
      1 | template1
  12438 | template0
  12439 | postgres
  16393 | production_erp
```

But our results were:

```
  oid   |          datname
--------+---------------------------
      1 | template1
  12438 | template0
  12439 | postgres
```

Clearly the entry that concerned us was missing. Is it enough to 'just' create the entry? We need to understand a bit more about the pg_database table, so let's look at the structure:

```
postgres=# select oid,* from pg_database limit 1;
-[ RECORD 1 ]-+------------
oid           | 12439
datname       | postgres
datdba        | 10
encoding      | 6
datcollate    | en_US.UTF-8
datctype      | en_US.UTF-8
datistemplate | f
datallowconn  | t
datconnlimit  | -1
datlastsysoid | 12438
datfrozenxid  | 579
datminmxid    | 1
dattablespace | 1663
datacl        |
```

Some of these are self-explanatory – oid = directory number, datname = friendly name. The fields that really concerned me were **datlastsysoid, datfrozenxid,** and **datminmxid**.

This led to several questions:

- How do we know what the appropriate values for those need to be for the database we are trying to recognize?
- What happens if we set it incorrectly?
- What happens if we ignore it?
- Will it magically rebuild or fix itself?

It was time to seek some additional help

> **A problem well stated is a problem half solved.**
>
> – John Dewey

A strong community

By this point I felt like we had gathered enough information about the situation and probable solution. Now, I just needed a little bit of help to figure out those last bits. Fortunately, PostgreSQL maintains **excellent documentation** and information about these columns. Reading that information led to the same questions.

I needed some practical, experience-based advice from the PostgreSQL Community, so I headed over to my company's PostgreSQL internal communications channel. This is where the PostgreSQL community really shines.

If you ask a question, you will get as deep an answer as needed—sometimes from the very people that develop the software. The PostgreSQL Community's commitment to share and spread their knowledge is amazing.

After several discussions and much exchanging of ideas and scenarios with a team of PostgreSQL experts all over the world, we determined the appropriate entries to create in the pg_database table. Then I ran some schema comparison tests and maintenance utilities; everything was checking out and their situation was certainly improving.

The time came to try connecting with the ERP Client, and...IT WORKED! At this point we performed a backup and restore of the database and made sure the restored copy was usable.

The next step was to have the sales department enter some sales orders and test the complete order-to-cash cycle, reports, etc. This uncovered quite a few data issues; data was missing.

The missing data issue appeared to be constrained to a set of related tables and functionality. Where there were parent/child relationships that had orphans or missing children. Depending on the relations, some missing records could be re-created from other transaction data. This led to iterations of restoring from backups and manipulating data via SQL and retesting the processes.

After a couple more hours of work, we were able to get the system stable enough to where processes were working as expected and new data was going to the proper places. There was still the issue of backfilling the gap in their data—a task which was divided up amongst their staff to recreate it from paper copies of invoices, purchase orders, shipping records, and so on.

AFTERMATH *And lessons learned*

It took us 4 days to get the system operating. Once everything was stable from a PostgreSQL server and data standpoint, the project was passed on to our Implementation team and accounting specialists.

The most important function of any system of record or accounting system is to produce accurate financial results and they had a bit of work to do to get there.

Once the dust had settled and the company was up and running again, I reflected on the many different lessons this experience taught me.

Troubleshoot methodically because there are no 'dumb' questions

While it may seem annoying or obvious at times, there is a good reason to start at the most basic troubleshooting steps. At the very least it's an annoying little checkmark on your list, but sometimes that seemingly obvious and annoying question is the actual solution. All of these troubleshooting steps help you paint a picture of the actual problem you are trying to solve.

If you get stuck, reach out to the community for help—the chances are that you are not the first person to ever run into a particular PostgreSQL problem!

> **The only mistake in life is the lesson not learned.**
>
> – Albert Einstein

Be flexible—sometimes the solution is not perfect

Thankfully, the company had made a 'kind of' backup at some point and retained it. Even though it was far from ideal, it probably saved them 3 months reimplementing their system and losing many years of transaction history.

Once you get a foothold towards a solution, be persistent

There were plenty of 'What-If' or 'If-This-Then-That' moments that took some reflection, consultation, and outside-the-box thinking. While my developers were relatively pessimistic about the outcome of this project, they were helpful with sanity checking some of the more complicated data rebuilding tasks.

Manage stakeholder expectations in a timely manner

While you may be feeling under pressure to resolve a problem quickly and efficiently, all stakeholders need to be appraised of the situation at all points in the process. If you are not comfortable having those conversations, then reach out to whoever is responsible for the customer relationship and ask them to relay what is going on in the process.

Reach out to the experts

Seek help from experts that deal with difficult problems every day. Instaclustr PostgreSQL Consultants and **Managed PostgreSQL offerings** can eliminate the worry of running and maintaining a modern and fault tolerant and highly available PostgreSQL installation and mitigate problems before they are a concern.

And finally: Don't overestimate your PostgreSQL and security management abilities

The threat landscape is constantly evolving. Keeping up with (and understanding!) PostgreSQL vulnerabilities at **https://www.postgresql.org/support/security/** can be a significant part of your PostgreSQL administration duties. This is where a managed service can really help you out, as they are designed with security first by default. There are numerous capabilities and benefits to a managed PostgreSQL service that are much simpler to implement than if you were to configure them on your own.

NetApp® Instaclustr specializes in open source technologies for enterprises. Our managed platform streamlines data infrastructure management, backed by experts who ensure ongoing performance, scalability, and optimization. This enables companies to focus on building cutting edge applications at lower costs.

3-22may25