# Migrating PostgreSQL

# To

# The Cloud

PGDay UK 2018
London

Chris Ellis - @intrbiz

Ian Hiddleston

chris@intrbiz.com

ian@originalsyn.co.uk

# Hello!

- I’m Chris Ellis
- IT Jack of all trades: Dev, Architect, DBA
  - DBA’s are the original devops
- Been using PostgreSQL for ~13 years
- Delivery focused, makes stuff happen

- And I’m Ian Hiddleston
- Network Engineer and Server Wrangler
- Used Linux since RH7 (not RHEL7)
- Never used PostgreSQL in anger before
- Proactively lazy
- Good at getting things through audits

# About The Project And This Talk

- Worked on smart energy analytics, with PostgreSQL as its core data store
  - Complex schema, lots of functions, lots of data processing done in DB
  - Reasonable data volume: ~5TB, few billion row tables
  - Batch data loads, quite IO intensive
- When I joined, product was run out of a traditional Data Centre
  - Dedicated and customized DB boxes, plus a failed OpenStack deployment
  - Other parts of the company already running in AWS - pressure and then a project for us to
- We led the successful migration to AWS
  - Fair amount of replatforming in the process too
  - Focus on: keeping performance, security, no customer impact
- This talk focuses on what we did, our issues and what we learnt

# Is The Cloud For You?

- Public Cloud: How long is a piece of string
- The cloud is not one thing, you can use it and abuse it in many ways
  - The concepts are probably the bigger goal
  - Just running stuff in the public cloud is probably missing the point
  - You will need to change your application, factor this into your costings / plan
- Sometimes it just not all about the money
  - However no excuse for not looking at the TCO of cloud vs other hosting options
  - There are several myths around cloud costing
    - Reduced staffing cost - not seen this in any org who's gone to the cloud
    - Be aware of hidden costs - easy to overlook network transit costs
  - Huge positives around developer flexibility, scaling that are difficult to calculate
  - Be skeptical and critical of your account managers

# Managed (RDS) vs DIY (EC2)

- You lose control: PostgreSQL is deployed for you, one way
- You get ‘support’
- Restricted extension list
- Limited storage options
- Can be expensive
  - Especially for high perf
- More suited to simple, low usage or proof of concept applications

- Freedom to deploy PostgreSQL your way, as you want it
- You need to support it
- Use the extensions you want
- Need to solve failover and replication
- More flexibility and options for storage
  - Especially for high perf
- Plenty of companies are running PostgreSQL like this in the public cloud

# Data, Data, Data

- Back your plan with data!  Know your stats
  - Benchmark your old setup
  - Benchmark your cloud options
- The cloud makes it easy to experiment
  - Load your DB into RDS, give it a test run
  - Take an EC2 instance, put PostgreSQL on it, load your DB, give it a test
- We found our DB couldn’t even load into RDS
  - Even when we managed it, performance was terrible
- We found nvme SSDs can be trixy buggers
  - Can have very different performance characteristics
  - AWS favour deep queues
- AWS make it difficult for you to publish benchmarks :(
- You need benchmarks to define success

# PostgreSQL In The Cloud: One Way

- We settled on using EC2 instances with local SSDs
  - Storage latency and performance
  - Frankly: EBS sucks!
    - It's latent
    - Quite low throughput for modern standards
    - Great for your server root image, not so great for data
    - Loads of people end up having to software RAID multiple EBS volumes
- Ended up with the largest I3 instance
  - 64 CPU cores
  - 488GiB RAM
  - 15TB (or a bit less) local nvme SSDs

# PostgreSQL In The Cloud: One Way

Ephemeral!

# PostgreSQL In The Cloud: One Way

- Sends a chill down your spine: DBA's usually like reliable storage
  - Not something that can be lost at the drop of a hat
- We RAIDed the 8 SSDs
  - Wear leveling
  - With dm-crypt over the top for audit / compliance
- We had to place all our trust into PostgreSQL (well that's easy)
  - Reliance on streaming replication
  - Reliance on backups
- Given our application, this was an easier pill to swallow
  - Majority of our data was batch loaded and could easily be replayed
- But this compromise reverberated across our stack

# Replication And Failover In The Cloud

- We used 1 primary DB and 2 replica DBs
  - Streaming replicate between them
  - Our batch data loads went direct into primary DB
  - Application traffic load balanced across replica DB
  - We wanted two read replicas, in different AZs for availability
  - Very rarely read from primary db
- We looked at various replication options
  - Patroni was primary choice
    - However…
- We ended up using PL/Proxy for failover
  - Stateless tier of PostgreSQL servers routing function calls to replicate DBs or primary DB
  - Only possible because our pattern from start had been all queries are function calls
  - Opens up the possibility of sharding PostgreSQL in the future
- Frankly: I'd love the day when PostgreSQL core handles failover out the box
  - When was the last time you ran a single node in production?

# Backup In The Cloud

- We'd been using pgbackrest for a year or so in our DC in production
  - One of the only PostgreSQL backup tools which worked (well) for our size of dataset
- No obvious reason to change tooling
- Our DB churns quite a bit per day
  - ~ 300GB compressed per day
  - During loads we push a lot of WAL
  - We had 25TB storage which gave us about 1 months of backup
- This costs quite a lot of standard EBS
  - Cold Storage (sc1) is your friend
    - So is RAID 0
    - 16TB limit per volume
- During our migration pgbackrest added support for S3
  - Still need time to try it
- Get your data out of the cloud, very different failure modes
  - Someone gaining control of 1 account can be as devastating as a hurricane
  - Isolate your accounts
  - Replicate to another region
  - Get your backups into another provider
- One of the best ways to move your data for a migration...

# Migrating Your Data

- We needed to shift a reasonable volume of data into the cloud
  - Don’t underestimate GB/s of a van
  - VPN, Direct Connect
- We settled on using OpenVPN, was easy for us to setup on each site
  - Beware of T2 instances running out for CPU credits
- Our batch data file archive was uploaded directly into S3 from our DC
  - You need to make sure this process is robust and resumable

- We used pgbackrest, our backup solution to migrate our database
  - Straight over our VPN link
  - pgbackrest uses multiple levels of compression and multiple connections
  - Took about 24h to restore our 5TB database like this
    - Beware WAL replay, may be faster to load immediate and replay batch in cloud
  - pg_base_backup over VPN connection was very slow
  - However pg_base_backup was usually the fastest way to build a replica once in AWS

# Keeping In Sync

- We were lucky, our app was mostly batch loaded
- However we still needed to catch up now and then
  - We used pgbackrest over a VPN alot
  - Despite compression, etal, still slow
    - We churn alot
- We had one or two load processes which took data from other systems
  - Don’t underestimate how the latency of a VPN will screw your system over
  - In the end we had to run some of these in the DC and copy over a backup
- Made it really easy to keep in sync
  - We didn’t need any customized solutions to replicated data from our prod DB to our new set up
  - PostgreSQL 10’s logical replication would be ideal for this
- Thankfully we had only a handful of tables which were real-time
  - They were small
  - Easy to sync over
    - KISS - SQL dump / import

# When It All Goes Wrong

- Last thing you want the day before going live is your DB servers to be crashing with kernel panics!
  - Especially hard CPU lockups
  - Ended up taking us 6 weeks to track down the root cause
  - Started to get interesting when we triggered it across multiple Ubuntu kernel versions
  - AWS support were great, but couldn’t provide us with VM memory dumps etc
  - Ultimately our issue, but had to debug it blind
  - Mostly tracked down by: luck, trial and error

# Which Would You Pick: The Revenge Of Ephemeral!

- Instance: i3.16xlarge
- vCPU: 64 @ 2.3GHz (Broadwell E5-2686 v4)
- Memory: 488 GiB
- Disk: 8x NVMe SSD @ 15.2TB
- RAID: RAID 0
- Function: PostgreSQL Master

- Instance: i3.16xlarge
- vCPU: 64 @ 2.3GHz (Broadwell E5-2686 v4)
- Memory: 488 GiB
- Disk: 8x NVMe SSD @ 15.2TB
- RAID: RAID 5
- Function: PostgreSQL Master

# When It All Goes Wrong

- So: RAID 5 or RAID 0
  - We initially used RAID 5: because that's what we'd have done in a DC
- Ephemeral really is ephemeral
  - Any issues with any SSD in the system
    - Boom, bye bye instance
    - AWS will tell you, you may not be listening...
  - AWS may decide an instance needs replacing at any time
  - Stopping or hard rebooting an instance will delete your data
- Turned out there was a subtle race condition in the md RAID 5 module
  - Needed specific conditions to trigger
  - We seem to have been the 2nd people in the world to have hit it
  - It took the first people about 6 months of VM dumps to track it down
  - We just switched to RAID 0 instead

# Finally Flipping That Switch

- Plan and prep this from day 1
  - Have a backup plan
  - Have a backup backup plan
- Again our batch processing saved us
  - Really easy to parallel run our stacks
- Allowed us to build a full side by side stack and validate it
- You'll definitely need a maintenance window
- You'll probably need some downtime
- How you keep your data in sync will affect your synchronization window on switchover

- Very early on we put a proxy layer into the cloud
- This proxied to our DC
- Updated all DNS to point to these proxies rather than DC
  - Drop your DNS TTLs for your switch over
- Go live was simply a case of switching proxy ruleset
- Then doing a final synchronization of a few small database tables

# Thank You!

# Questions?

chris@intrbiz.com

ian@originalsyn.co.uk