Managing
Large-Scale PostgreSQL Deployments
OpenSCG Case Studies

Scott Mead
Sr. Architect
OpenSCG, Inc

OpenSCG™
# Table of Contents

Purpose.......................................................................................................................................................3  
Audience......................................................................................................................................................3  
Overview...................................................................................................................................................3  
Methodology..............................................................................................................................................4  
    Software..................................................................................................................................................4  
    Packaging............................................................................................................................................4  
    Deployment...........................................................................................................................................5  
Database development and evolution........................................................................................................5  
Testing.......................................................................................................................................................6  
    Database Engine Upgrades ...................................................................................................................6  
Availability................................................................................................................................................6  
Case study..................................................................................................................................................7
Purpose

This document is designed to provide insight into how the OpenSCG Production Data Operations (PDO) team manages large-scale (100+ instance) PostgreSQL deployments for our customers.

Audience

This document has been geared to deliver valuable information to anybody engaged in managing a large number of database instances spread across multiple continents, coasts, data-centers and clouds.

Overview

As virtualization and cloud-based hosting becomes a mainstay in day-to-day operations, system 'sprawl' is beginning to take place. IT professionals are taking advantage of the cost structure and self-service nature of public and private clouds in order to provide quicker turn-around and higher satisfaction to their users. Production infrastructures are now beginning to make their way into the cloud, this includes the database.

Traditionally, database infrastructures have been rigidly physical in nature; I have a set number of servers, those servers are attached to a SAN and that SAN has the following 10 spindles dedicated to tables X, Y and Z. Over time, data managers have become deeply proficient at 'scaling up'. A superdome with a room full of disk cages was not uncommon.

When scaling up, the brunt of the administration work is placed squarely on the shoulders of the hardware team and systems administrators, the DBA team simply logs in to one instance or service, makes their changes, and moves on. Configuration of a new host or service is straightforward, the administrator focuses some time on understanding the application and the workflow, and builds a configuration file for it. Then, they will install the database software, build the configuration and install the schema. The Deployment of new schema changes, although not simple, was again, straightforward.

Clouds don't scale-up, they scale-out. This is very counter to our industry methodologies, so, what is it that we as database professionals are going to encounter? That instance that we could learn and understand and know how to care for suddenly explodes into 50 or 100 copies of itself. We experience requests to pull data from servers in 10 data-centers all at once while simultaneously being required to deploy new schema to 25 development servers and their 50 QA counterparts, all with no downtime. With all of this going on, when do I take the time to look at my daily stats and notice that the users table on phlnnldb78 has a 500% increase in updates and I am now violating my response time SLA due to a blown index?

Managing a large number of database systems requires a strange combination of rigid structure
and intrinsic flexibility. The tools and the team must be able to reliably build repeatable structures and constructs across a large number of systems without losing the ability to adapt to emergent situations. Tools need to be able to provide enough information so that the administrator can ascertain (correctly) system state without having to login to each and every host; these tools must perform without severely impacting the performance of the production system.

**Methodology**

The general methodology behind managing a large number of systems can be boiled down to a few small steps:

- Minimize change
- Keep human intervention to a minimum
- A framework for periodic maintenance must exist that allows human intervention when required and the ability to convert manual processes and procedures into automated scripts easily
- All changes should be additive (backward compatibility maintained)
- Periodic control reports against all hosts and hardware
- Frequent control reports against database schema
- Simple software and schema deployment mechanism

**Software**

The version of the PostgreSQL database in use varies depending on which specific environment is being referenced, OpenSCG PDO is simultaneously managing versions 8.3 – 9.2. Some of these systems cannot be upgraded for legacy reasons and are frozen on older versions of the database.

Managing the PostgreSQL database software is the primary function of OpenSCG's PDO, packaging, deployment, upgrade, etc...

**Packaging**

When managing a large number of systems in production, installers or source-based installation is not optimal. Each of these options comes with many different variables that, in the end, introduce slight differences on each installed node. Packaging allows the administrator to control each aspect of installation and minimizes the per-node differences in an environment.

There are multiple types of packages created in order to keep management as simple and flexible as possible:

- Binary packages
- Configuration packages
- Schema packages
- Management package
Deployment

Deployment consists of many pieces

1. Deploy database software to the operating system

   All PostgreSQL deployments are managed via RPM (Redhat Package Manager). This customer uses a redhat-based operating system, and software packages to deploy all software. The OpenSCG PDO team builds a specific set of packages to conform to the customer deployment system. These packages must deploy completely and successfully to each host, whether dev / qa / prod in an unattended fashion. Once deployed, the database is active and listening to TCP / IP settings with a per-environment or per-host set of security configurations associated.

2. Deploy database configuration package

   The package that contains the database configuration (postgresql.conf, pg_hba.conf, etc...) is deployed

3. Deploy customer schema to database engine

   Once the database engine is active schema deployment takes place. OpenSCG PDO has developed special tools designed to deploy the schema to the system. The deployment mechanism generally managed centrally, however, we have also developed packages that allow for deployment to systems that do not have network connectivity to the central deployer.

4. Validation of Schemas

   Experience has shown that each database engine in the cluster will, at some time or another, be subject to manual changes. This could be for many reasons: manual analysis, errors in the 'gold-master' schema, errors in the application, bad data from the field, etc... When manual changes occur, they typically occur for good reason, however, many times, the exact change gets lost or forgotten. It is critical that the deployment system be able to determine what the schema version should be as well as provide delta showing what it actually is. If a deployment is designed to upgrade from version 1.1.10 to 1.1.11, it is assuming that the structure is at 1.1.10. If it is at an unknown variant, the upgrade could fail and result in application downtime.

Database development and evolution

Some shops have developers that write code, others, a single or group of development-oriented DBA[s] that engineer procedures, SQL and the schema. Whichever method is in use, someone with production experience needs to take a look at the schema from a 'deployability' perspective. Whether or not the code fits the problem is a different issue, and should be addressed in the development cycle, how to deploy the code needs to go through a person or persons that have an eye for locking behavior, approximate runtime based on input size, overall performance impact to the system, etc... Some of these issues can be identified by automation, be more often than not, they will slip through the cracks, and a simple column addition will bring along an SLA violation. The correct way around this is to get the DBA involved up-front and testing during the development phase. The purpose here is to take what schema the developers need and to make it fit into the deployment model (as discussed above). The deployment should go through the standard deployment system, even if some DBA intervention is
required during the deployment, getting the team involved early-on guarantees that, come deployment time, these issues will be ironed out.

**Testing**

When a large number of systems are involved, new releases get deployed very widely and the ability to revert, although present, isn't as simple as when only a few systems are involved. Automated and frequent regression tests are a **requirement** when dealing with large distributed systems. Tests should be written by the DB team and the development team to ensure that the system is covered from both sides.

**Database Engine Upgrades**

Upgrading the version of the PostgreSQL database engine is a simple procedure when the dataset is small and downtime is an option. In a large environment that has terabytes of data and low or no-downtime operations, special upgrade procedures need to be designed.

A few key elements:

- The packaging needs to allow for multiple versions of the database to be installed AND run simultaneously
- The DBA team must have full discretion and deployment control over the 'Management' package. This allows for management scripts and tools (like upgrade and replication utilities) to get deployed quickly

Major version upgrades of a database server are well known to be time-consuming and 'hair-raising' operations. The key ingredient is keeping your schemas in-line with the 'gold-master' (or within a tolerable limit). This allows not only development to get the upgrade procedure correct (the data access layer or sql statements) and the DBA's to write a consistent set of scripts to perform the upgrade (minimize change).

**Availability**

When a large number of critical systems are deployed, failure of each component must be considered. The pgHA project was built specifically to allow for failure of a single database in a large number of deployed systems. pgHA can manage either express notification and a simplified manual failover process or a fully-automated failover to the standby database.

In the event of a failure, a set of business rules are followed that can move production operations over to the standby, replica database and re-point the applications to the new database. At the same time, pgHA collects a number of statistics that allow for the administrator to debug the state of the systems and reason for failure.

Whichever mode is used, manual or automated failover, it is important that the system provide administrators with alerts and a debugging console. If the manual failover is in use, the administrator needs to be able to make the decision to 'pull-the-trigger' within a few seconds or minutes. PgHA has been purpose-built for this purpose.
Case study

OpenSCG's PDO team (Production Data Operations) is responsible for a combined total of 100 billion transactions per day across our customer base. The team manages multiple customers, many of whom are themselves moving to some combination of the public and private cloud.

One of these corporations is a US based, multi-billion dollar telecommunications company. This client's infrastructure is responsible for processing over 500 million transactions a day initiated automatically by cellular phones ('dumb' and smart phones alike). The primary persistence backend for this client is a PostgreSQL database. In order to provide isolation based on their customer's wishes and for scalability purposes, a group of database servers is deployed for each defined business unit. These requirements are based on many factors including handset features and customer wishes.

At the time of this writing the number of deployed instances was:

- Production: > 50
- Load Test: > 20
- Development: > 60
- Regression Test: > 60
- Quality Assurance: > 100

All of the Production hosts are physical hardware, whereas Dev / Regression and QA are 100% virtualized.

The development team follows a mostly agile development structure. This requires a large number of servers due to the number of branches being worked on at any time and the fact that builds are automatically generated on commit. The QA team is responsible for many permutations of testing and has many servers performing dual-duty as well as splitting single product tests across a number of hosts.

While development moves forward at a breakneck pace on new customer features, OpenSCG PDO is responsible for maintaining stability in a 24x7 production environment. This requires an understanding of all the changes taking place through Dev / Reg / QA and understanding their impact to production.

Challenges

Initially, this client was running with very little database support and a similar transaction pattern across their business units. This similar pattern made it easy to make changes to the core and deploy it to all environments as a single piece with a high guarantee of success. Eventually however, each system started to behave differently; standard database maintenance (indexing, etc...) that used to run in a few minutes in all environments was taking minutes to complete in some and days in others. The purely automated database deployment tool had a knack for dropping whole tables just to add new columns and everywhere you looked, you would find a slightly different version of a schema (plus a few manual modifications). To say the least, the databases were not sharing the load, the brunt of the work was being done on a single server that was not fully equipped to handle it while the others sat, mostly idle.
Improvements

Applying the OpenSCG, PDO set of methodologies to these database instances moved the client into having a true data-cloud.

- Per-table monitoring, alerts and thresholds enabled the data team to focus on building a data model that could accommodate the ever-changing footprint of the application
- Aggregate, environment based monitoring and alerting provided the client with a better way to quantify the efficiency of the application from a data-storage and utilization perspective
  - Performance enhancements were made based on this data to shift the load from a single, stressed to severs to a balanced load across multiple hosts
- Simplified failover with pgHA took the system from a 1 hour failover time to a 10 minute total failover time for all services (when in manual mode)
- Automated failover will bring this down to 2 minutes once deployed
- Rapid, Agile development is becoming simpler
  - Schema is reviewed prior to commit
  - Once commit takes place, dev / regression and QA get new copies
  - Production deployment is simplified and, due to the amount of previous deployment to dev and QA, done with low risk and high comfort
- On-boarding developers and admins is becoming simpler and simpler
- Per-database differences are rapidly disappearing, while the team still has the ability to react in an emergency

Summary

The methodologies tools and training developed by the OpenSCG PDO team have provided a database management strategy that allows for DBAs to experience the comfort afforded them by the old-hat, “single, big instance” strategy while taking advantage of horizontal scalability, distributed computing and cloud infrastructures. It's a subtle mindset shift from what we as DBAs are used to, but it makes a big difference in the way that we can effectively manage hundreds or even thousands of disparate database systems.