accenture

Run PostgreSQL Like a Product

Operational Readiness for Databases

Roland Stirnimann

22.04.2026

Agenda

Image source: pexels.com

01 **Why Operational Readiness?**
The production reality: Choosing the right tools and partner

02 **Resilience**
Backup, recovery & high availability

03 **Observability**
Alerting, monitoring and logging visibility

04 **Security**
Authentication, authorization & compliance

05 **The Operating Model**
Governance and automation

01

Why Operational Readiness?

The production reality: Choosing the right tools and partner

Why Operational Readiness Matters

Organizations don't fail PostgreSQL – they fail to operate it!

01

Service Reliability & Availability

- PostgreSQL is production proven, failures happen at the ops layer
- Define your SLA requirements and map those to your target architecture
- Strive for an architecture that allows you to act proactively on potential challenges

02

Ecosystem Integration

- Do not reinvent the wheel.
- Integrate with existing services in your company (IAM, monitoring, backup, etc.)
- Focus on end-to-end automation capabilities
- Repeatable setups across platforms/stages

03

Tooling & Platform Maturity

- The tool zoo challenge
- PostgreSQL OSS and commercial ecosystem is rich but fragmented
- Consider automation integration
- Standardized toolchain reduces complexity

Accenture: Your Partner Across the Data Landscape

We are your one-stop shop. Get everything about PostgreSQL and related technologies from one partner.

CAPABILITY AREA	DESCRIPTION
PostgreSQL Core Expertise	Architecture, HA/DR, performance tuning, backups, lifecycle management, monitoring
Cloud Hyperscalers	AWS, Azure, GCP and OCI managed services
IAM & Security	Integration with enterprise IAM, encryption, access controls
Compliance & Governance	Regulatory controls, audit readiness, standardized policies, project management
Industry Expertise	Financial Services, Public Sector, Health & Life Sciences, Products, Resources, Communications, Automotive, Fashion
Application Development	Application development and modernization together with legacy database migration
AI	Most important topic in Accenture right now, everything is about AgenticAI

We scale with your demand on skills and people

02

Resilience

Backup, recovery & high availability

Define Your Recovery Objectives

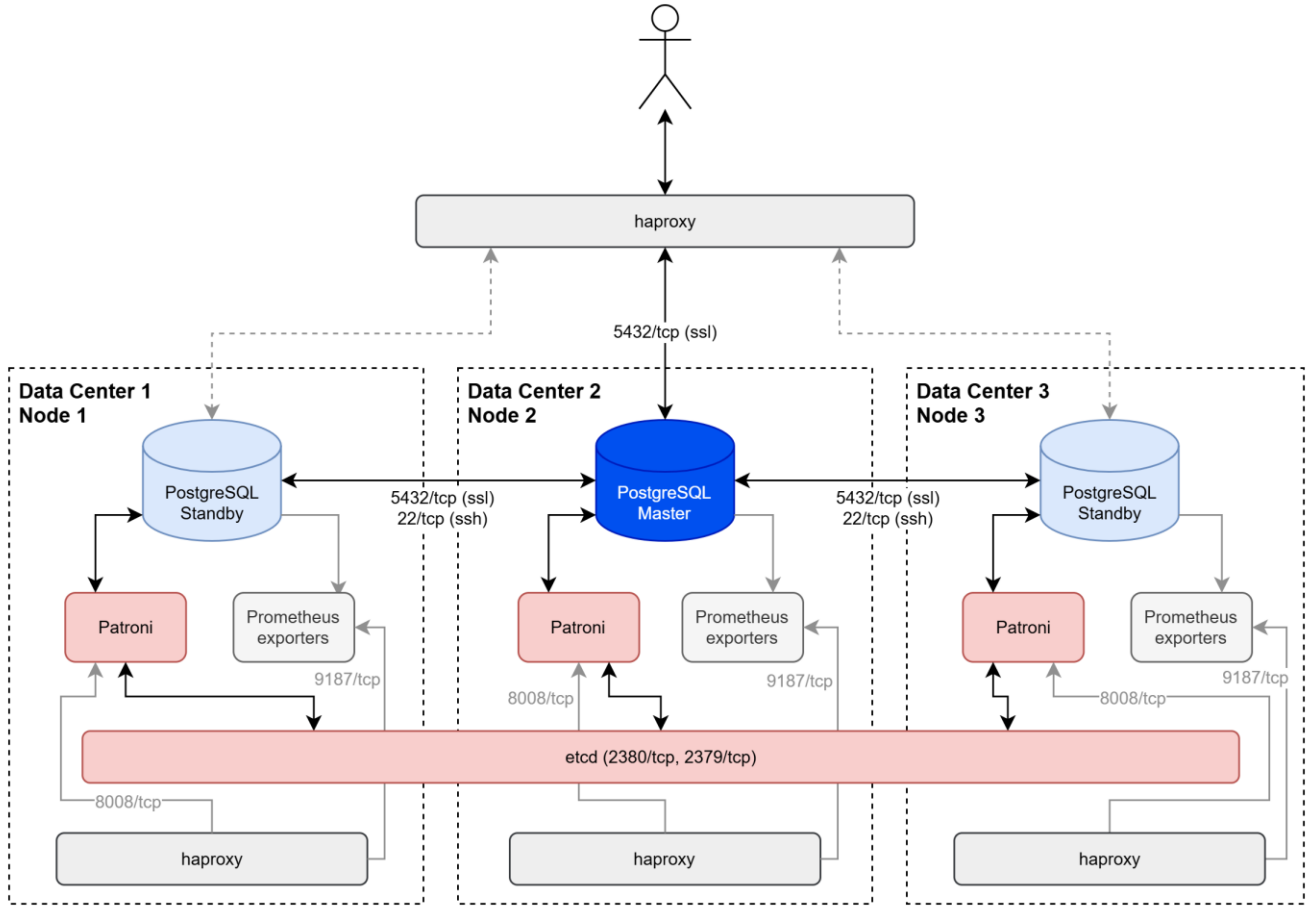
You cannot design resilience without knowing what you are protecting

- **RPO**: how much data can you afford to lose?
- **RTO**: how long can the business tolerate downtime?
- **Retention**: how long do we keep backup?
- **WORM** (Write Once, Read Many): how long must data be protected from any change or deletion?
- Recovery objectives must be agreed with the business
- Derive architectural decisions like backup frequency, WAL archiving, HA topology, disaster/recovery scenarios
- Document them in an SLA and make them visible, auditable, and signed off

High Availability with Patroni

The OSS standard for automated high availability management of your PostgreSQL streaming replication

- Patroni wraps PostgreSQL with a distributed key-value store to perform automated leader election and failover
- Physical replication at WAL level (sync or async)
- REST API for cluster state inspection and operations
- Prevents split-brain through consensus-based fencing
- Integrate with pgBackRest for replica rebuild
- Benefit from read replica(s)
- Alternative HA approaches
  - PostgreSQL on Kubernetes with CloudNativePG



Backup & Recovery with pgBackRest – Plan B if HA fails

DBA: “Our server has crashed”. *Boss*: “Where is the backup?” *DBA*: “On the server…” ☺

BACKUP

- Physical full, incremental and differential backups
- Retention policies including WAL archives handling
- Patroni integration
- Parallelization and encryption
- Backup integrity checks including page checksums
- Alternative backup approaches
  - Enterprise backup tool like CommVault
  - Barman in CloudNativePG on Kubernetes
  - pg_basebackup (pg_verifybackup for checks)
  - pg_dump for logical backups
  - Storage snapshots

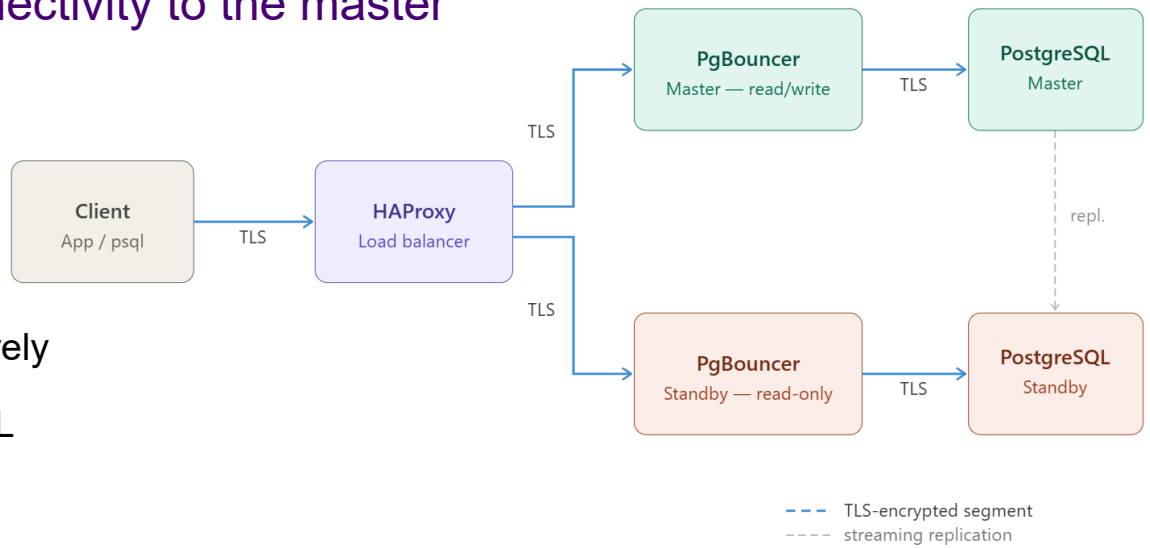
RECOVERY

- Testing and document your defined scenarios
  - Automated into a sandbox on a regular basis
  - Manual for hands-on practice
- Recovery use cases – in-place or out-of-place:
  - Reinstate a standby in a HA setup
  - Complete or point-in-time (PITR) instance recovery
  - Partial PITR via auxiliary instance and pg_dump
  - Import from a nightly pg_dump export
- Data propagation to dev stages
- Consider explicit page corruption tests – pg_dump to /dev/null

The Connection Layer

Guarantee secure, scalable and transparently routed connectivity to the master

- HAProxy acts as entry point for the client
  - Role detection via Patroni API and connection routing
  - Load balancing e.g. readonly session with multiple standbys
- PgBouncer understands and speaks the PostgreSQL wire protocol natively
  - Holds a pool of established and re-usable connections to PostgreSQL
  - Reduces load and number of connections
  - Trade-off: Consider the session state isolation principle since sessions are shared between clients
- Consider installing HAProxy on the database host for administrative connects
  - Proper TLS termination
  - Unified entry point (e.g. 443) and rewrite URLs to underlying services
    - *:443/metrics ➔ postgres_exporter on 9187
    - *:443/health ➔ Patroni REST API on 8008



03

Observability

Alerting, monitoring and logging visibility

What should you Measure?

Observability is a discipline – each layer serves a different operational purpose

ALERTING

Something is wrong right now, someone must act

- Near real-time
- Action-driven and reactive
- Time-critical (on-call 24/7)
- Keep the alert metric list short
- Stateful (auto-clear) and stateless alerts
- Examples:
  - PostgreSQL instance down
  - Free disk space below 5%

MONITORING

Monitoring informs decisions to respond to trends

- Proactive & trend-based
- Capacity & performance
- No immediate action (business hours)
- Large set of collected metrics
- Solve errors before they occur
- Examples:
  - Database growth rate
  - Connection count over time

LOGGING

Logs provide context: why did this happen?

- Severity-based
- Searchable & filterable
- Foundation for audit & analysis
- Detailed information about events
- High volume
- Examples:
  - Severity levels >ERROR from log
  - pgaudit log

The Observability Stack

Build your monitoring stack once, reuse it across every PostgreSQL instance

ALERTING

- **Prometheus Alertmanager**: grouping, deduplicating and routing alerts to the correct receiver based on the Prometheus time-series database
- **Grafana Dashboard**: visualizing alert metrics and status from Prometheus database

MONITORING

- **Prometheus Server**: Monitoring system that scrapes (pull) metrics from HTTP endpoints (exporters) and stores them as time series data.
- **Metric Exporters**: Exposes metrics from an application, database or host via an HTTP endpoint.
- **Grafana**: Visualizing metric data
- **PoWA**: Collecting and visualizing PostgreSQL performance and SQL metrics
- **pgwatch**: PostgreSQL monitoring solution

LOGGING

- **Logstash**: Collect, transform and send logs from your PostgreSQL instance
- **Elasticsearch / OpenSearch Core / Splunk**: Search and analytics engine for the collected logs
- **Kibana / OpenSearch Dashboards**: Visualize the log data stored in analytics engine
- **PostgreSQL extensions**: pgaudit and pgauditlogtofile

04

Security

Authentication, authorization & compliance

PostgreSQL Authentication: Who Are You?

From the connection request to the verified user – the first line of defense

01

pg_hba.conf: The PostgreSQL authentication rulebook

- Defines who connects, from where, to which database, and how
- Principle: start restrictive, open explicitly
- Order matters: first matching rule wins
- Pairing hostssl authentication methods with the verification of the client certificates (clientcert)

02

Password authentication

- Using scram-sha-256 for server-based password authentication
- Rotating passwords regularly for example with HashiCorp Vault
- Better implement enterprise identity integration: LDAP/Active Directory, GSSAPI/Kerberos for SSO or OAuth 2.0 with OIDC (PostgreSQL 18)
- Enterprise identity integration allows central user lifecycle management including password policies

03

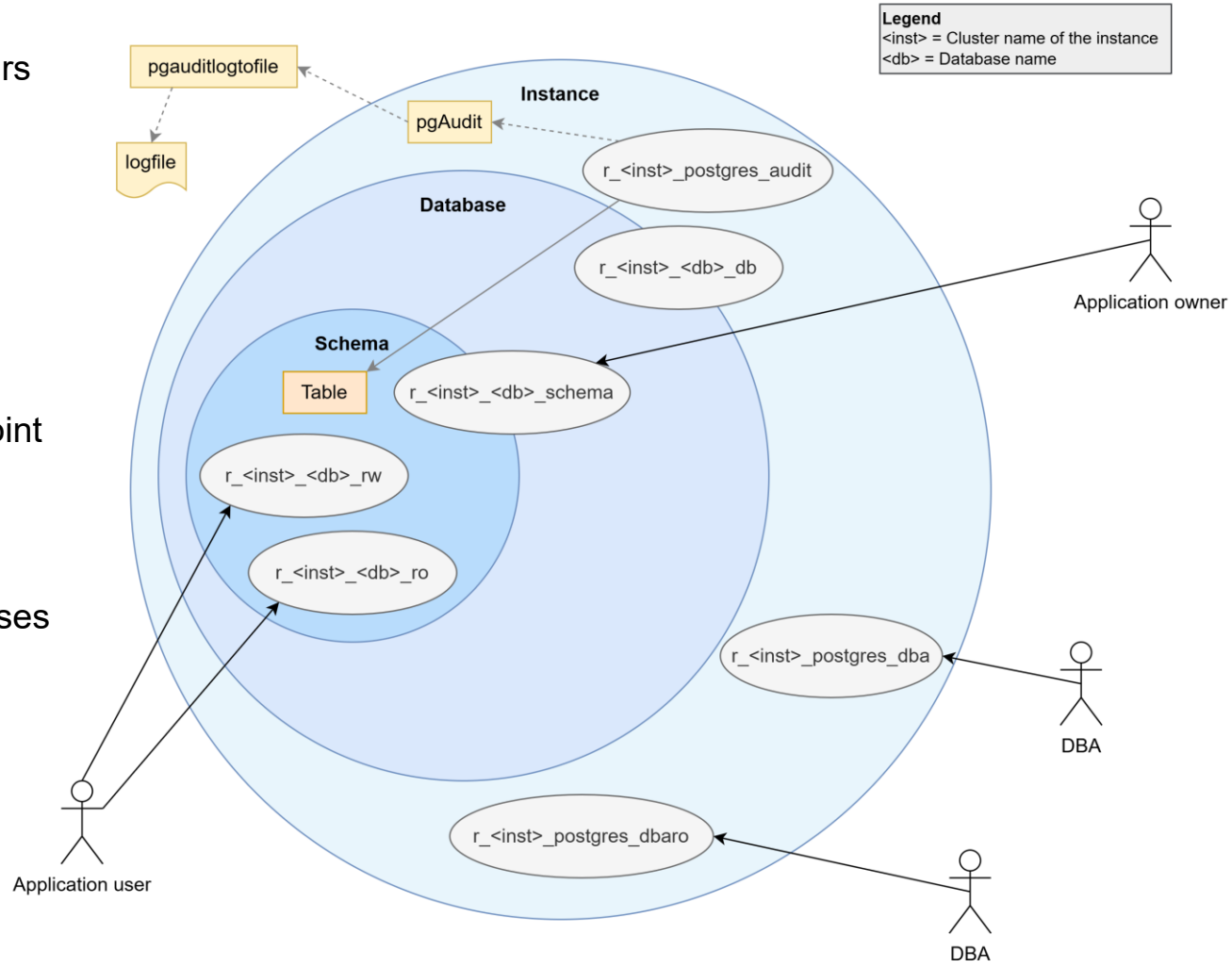
SSL/TLS Certificates

- Used to encrypt all client connections (ssl = on, hostssl in pg_hba.conf)
- Client certificate authentication for service accounts – no password to steal
- Implement automated certificate rotation for example with ACME

Authorization in PostgreSQL: Who Can Do What?

Defining and Enforcing Access Rights in the Database

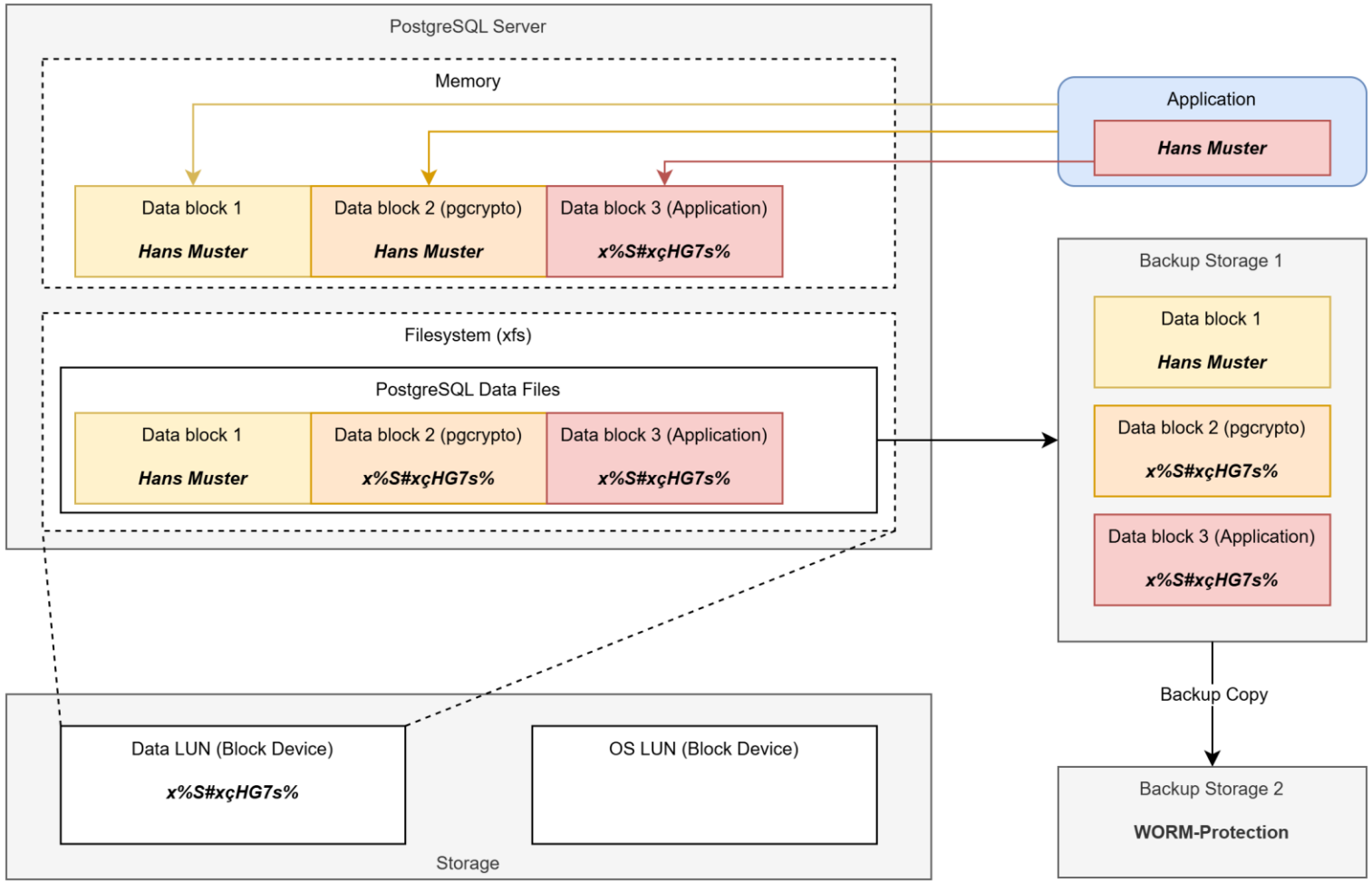
- RBAC (Role-Based Access Control) concept based on onion layers
  - Naming conventions
  - Default privileges to enforce grants on new objects
  - Enforce a role as object owner
- Row-level security to enforce user-specific visibility at the data
- Consider integration with enterprise wide IAM system (e.g. SailPoint IIQ, HashiCorp Vault)
  - Controlled privilege request flow
  - Automated provisioning (GRANT/REVOKE) into target databases



Security by Design: Hardening & Auditing PostgreSQL

Secure by Default, Auditable by Design

- Use the BSI Grundschutz Framework for building comprehensive security stance, use CIS Benchmarks for specific configuration recommendations
- Regular scanning of the defined security controls and reporting of any deviations
- Configure auditing powered by pgaudit extension
- Consider different encryption options
  - Storage-level encryption at rest
  - Filesystem-level encryption (LUKS)
  - Column-level encryption (pgcrypto)
  - Application-level encryption



05

The Operating Model

Governance, automation & the full picture

Operational Governance – Subject for the Ops-Concept

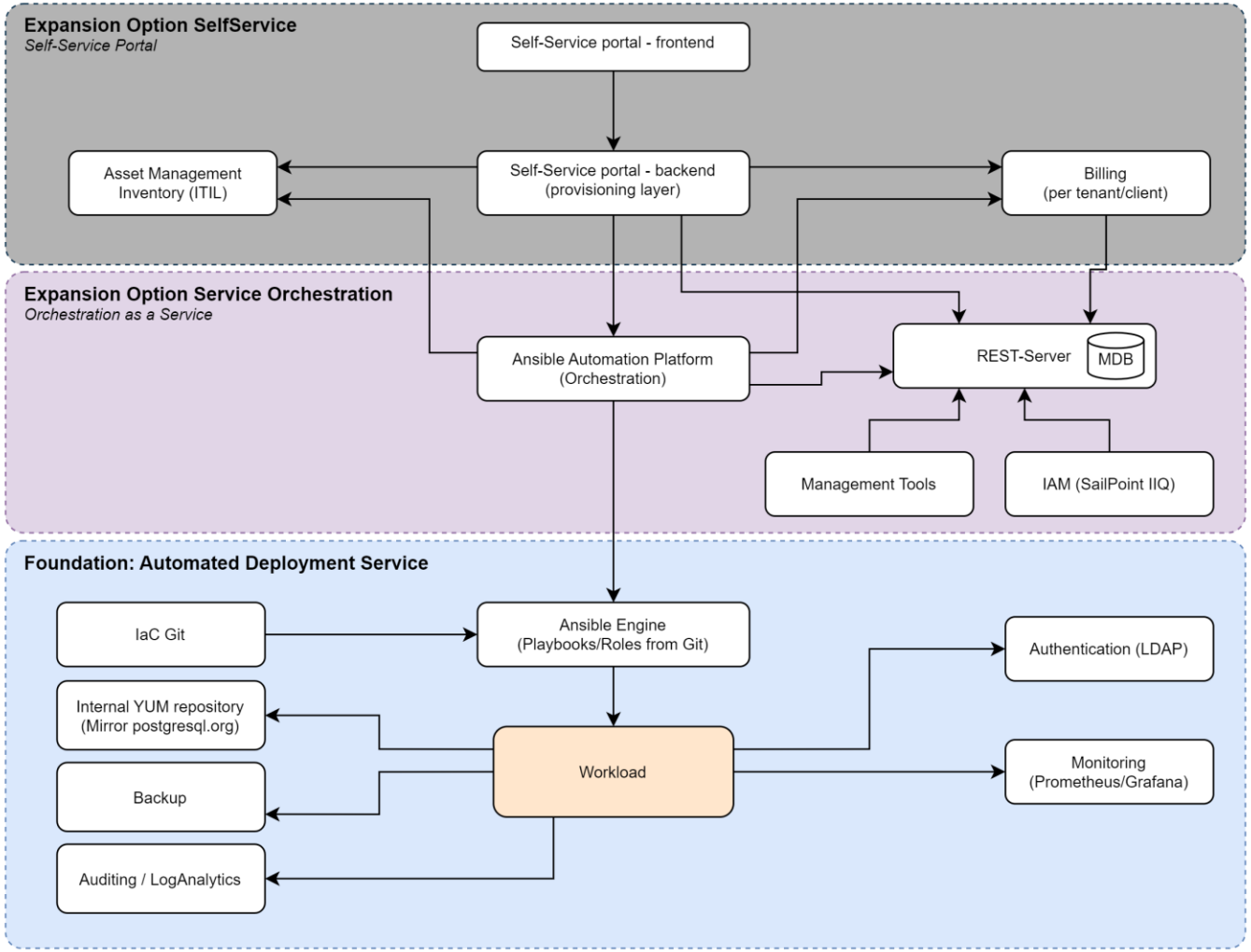
Operational governance turns PostgreSQL from a managed system into a trusted enterprise platform.

GOVERNANCE TOPIC	REQUIREMENTS
Roles and responsibilities	• Clear separation of duties between DBA, operator, security and application • Defined ownership for config standards, incident handling and access approvals
Configuration and change management	• Version-controlled baseline configurations • Standardized rollout of changes • Staging concept and environment consistency across prod, int and dev
Access and privilege governance	• Least-privilege principle for all roles and RBAC models for different purposes • Centralized IAM system • Regular review and revocation of unused privileges
Monitoring, auditing and evidence	• Continuous monitoring of availability, performance, capacity and security • Audit-ready logging of administrative commands and sensitive objects
Operational controls and resilience	• Definition of SLA/SLO and regular validation • Standardized backup and restore procedures (disaster recovery tests) • Clear escalation paths for operational incidents
Compliance and lifecycle management	• Aligned with internal security policies • Patch and upgrade governance • Documentation supporting audits and regulator reviews

Automation is Key – Ansible is your Friend

End-to-end Automation is a Mindset and requires Collaboration across Teams

- End-to-end automation involves the PostgreSQL provisioning and all surrounding components with one click!
- Reduce input parameters and derive parameters from running environment or properties store
- Depending on the environment size and changes, consider different expansion options from plain Ansible playbook to self-service portal including tenant based-billing
- Distinguish the automation workflows between:
  - Deployment: create (new or clone) and delete a PostgreSQL instance/database
  - Day-two configuration: enable/disable backup, change sync/async, change SSL config, add/remove extensions, enable HA, change VM-shape, etc.
  - Operation tasks: Unlock user, *Swiss Army Knife* command, restore/clone instance/database/table
- Early involvement of stakeholders during solution design!



Architecture and Automation Principles

Failed Automation Workflows must immediately clean up the Mess they produce

PRINCIPLE	DESCRIPTION
Write Cleanup Workflows	• Write create/delete or enable/disable workflows in parallel to revert failed tasks • Failed deployments trigger the cleanup to release resources and remove leftovers
Separate environments (stages)	• Stages allow proper testing and deployment pipelines from dev to prod • Separation can be physically or logically – the main thing is that it happens
Infrastructure as Code	• The code reflects your runbooks and is based on the metadata (database, vars-files, etc.) • Code is in Git and follows a defined branching strategy per stage, feature, fix
API-First Design	• Using APIs, e.g. creating VMs, adding instance to monitoring, starting workflows etc. • Avoid interruptions and manual intervention in the automation workflows
Metadata is Key	• Maintain the metadata about the environment and deployments as part of your workflows • Make sure that operational ad-hoc interventions are reflected in the metadata • Make the metadata available to the operation people, e.g. via simple CLI tool
Do not reinvent the Wheel	• Reuse available components in the company and integrate seamless with your service • Find a balance between using external libraries and do it your self – ask if it is needed. • Many external dependencies can be a pain as well during upgrades

Key Takeaways

01 AUTOMATE

02 STANDARDIZE

03 INTEGRATE

04 GOVERN

05 MEASURE

06 SECURE

Thank you