

Managed Database as a Service

Whitepaper- Understanding Managed Database as a Service (MDBaaS)

Table of Contents

Executive Summary 5
Chapter 1 — The Evolution of Database Management 6
1.1 The stages 6
1.2 What actually changed 7
1.3 A note on the direction of travel 7
Chapter 2 — Getting the Terminology Straight 8
2.1 The terms 8
2.2 The distinction that matters: control plane and data plane 8
2.3 Where DBaaS sits among service models 9
Chapter 3 — What Is a Self-Managed Database? 10
3.1 Definition 10
3.2 What the customer owns 10
3.3 Advantages 10
3.4 Disadvantages 11
3.5 An honest position 11
Chapter 4 — What Is Managed DBaaS? 13
4.1 Definition 13
4.2 The shared responsibility model 13
4.3 Responsibility matrix 13
4.4 What the customer gains 14
4.5 What the customer gives up 15
Chapter 5 — Deployment Models Within Managed DBaaS 16
5.1 Multi-tenant public 16
5.2 Virtual private / dedicated network 16
5.3 Dedicated infrastructure 16
5.4 Private cloud managed DBaaS 16
5.5 Hybrid 16
5.6 Choosing 16
Chapter 6 — Who Provides Managed DBaaS? 18
6.1 Hyperscale cloud providers 18
6.2 Regional and sovereign cloud providers 18
6.3 Database vendors 18
6.4 Managed service providers and system integrators 18
6.5 The pattern 19
Chapter 7 — Managed DBaaS Is Not a Different Category 20
7.1 The correct taxonomy 20

7.2 The incorrect taxonomy, and why it persists 20
7.3 Applying the test 21
7.4 What follows from getting this right 21
Chapter 8 — Where Providers Genuinely Differ 22
8.1 Engine coverage 22
8.2 Control plane maturity 22
8.3 High availability architecture 22
8.4 Backup and recovery capability 22
8.5 Scaling behaviour 22
8.6 Security capability 22
8.7 Observability 23
8.8 SLA structure 23
8.9 Support model 23
8.10 Jurisdiction and compliance 23
8.11 Commercial model 23
8.12 Using these dimensions 23
Chapter 9 — Availability, Resilience, and Disaster Recovery 25
9.1 RPO and RTO 25
9.2 High availability patterns 25
9.3 Failure domains 25
9.4 What managed DBaaS changes 25
9.5 DR is still a shared responsibility 26
Chapter 10 — Security and Compliance Architecture 27
10.1 Encryption 27
10.2 Network security 27
10.3 Access control 27
10.4 Audit logging 27
10.5 Vulnerability and patch management 28
10.6 Certifications and attestations 28
10.7 The division, summarised 28
Chapter 11 — Why Regulated Industries Prefer Managed DBaaS 29
11.1 What changed 29
11.2 Sector requirements 29
11.3 The requirements these sectors share 30
11.4 The residual obligation 30
Chapter 12 — Why Sovereign Managed DBaaS Matters 31
12.1 Three distinct concepts 31
12.2 The Indian regulatory position 31

12.3 What sovereign managed DBaaS provides 32
12.4 Where Yotta fits 32
12.5 The honest caveat 33
Chapter 13 — Total Cost of Ownership 34
13.1 The cost categories 34
13.2 Comparative view 34
13.3 An honest statement of the economics 35
13.4 Building a defensible TCO model 35
Chapter 14 — Lock-In, Portability, and Exit 36
14.1 Separating the layers 36
14.2 Self-managed is not lock-in-free 36
14.3 Designing for portability 36
Chapter 15 — Migration Approach 38
15.1 Assess 38
15.2 Prioritise 38
15.3 Choose a method 38
15.4 Validate 38
15.5 Cut over and optimise 38
Chapter 16 — Choosing the Right Database Strategy 39
16.1 Choose self-managed when 39
16.2 Choose managed DBaaS when 39
16.3 A decision sequence 39
16.4 Mixed estates are normal 40
Chapter 17 — Why Enterprises Are Moving to Managed DBaaS 41
Chapter 18 — Common Misconceptions 42
Conclusion 43
Appendix A — Questions to Understand How Managed DBaaS Is Delivered 44
Appendix B — Glossary 47

Executive Summary

Ask ten enterprise architects to define "DBaaS", and you will get ten answers. Ask them how "Managed DBaaS" differs from "hyperscaler DBaaS" and most will describe two different products. They are not two products. They are the same operating model sold by different companies.

This confusion is not academic. It shapes RFPs, distorts vendor shortlists, and pushes organisations into architectures they did not intend to buy. A procurement team that believes "hyperscaler DBaaS" is a distinct and superior technology class will filter out capable providers before evaluation begins, not on capability, not on compliance fit, not on cost, but on a label.

The core argument of this paper is simple:

> Managed Database as a Service is a *service model*, not a *product family*. It describes a division of responsibility between a customer and a provider. Any organisation with the engineering depth to automate provisioning, high availability, backup, patching, monitoring, and recovery can deliver it. The provider's identity does not change what the model is.

There are two meaningful choices in front of an enterprise:

1. **Self-managed** — the customer owns the database end to end, wherever it runs.
2. **Managed DBaaS** — a provider owns the platform and its operations; the customer owns the data and the application.

Everything else — AWS RDS, Azure Database for PostgreSQL, Google Cloud SQL, Oracle Autonomous Database, MongoDB Atlas, EDB BigAnimal, Yntraa Cloud SutraDB, and FlexiDB — sits inside the second category. They share the managed operating model but differ materially in engine architecture, engine coverage, automation maturity, SLA structure, support model, commercial terms, and jurisdiction. Those differences affect workload fit even though they do not create a separate operating-model category.

This paper covers:

- how database operations evolved from manual administration to managed services
- what "self-managed" and "managed" actually mean, responsibility line by responsibility line
- the deployment models available under managed DBaaS, which is where genuine architectural difference lives
- who delivers managed DBaaS, and why the delivery organisation is not a category
- the dimensions on which providers *do* legitimately differ, and how to evaluate them
- why regulated industries in India like government, BFSI, healthcare, telecom, manufacturing ,etc. are moving to managed models
- what data sovereignty means under the DPDP Act and its 2025 Rules, and why it changes the shortlist
- total cost of ownership, including the costs that rarely appear in a spreadsheet

- lock-in, portability, and how to design an exit before you need one
- a decision framework, an evaluation checklist, and a glossary

Who should read this. CIOs and CTOs setting platform strategy. Enterprise and data architects designing for availability and compliance. Infrastructure and DBA leads evaluating operating models. Procurement and compliance teams writing cloud database RFPs. Product marketing and pre-sales teams who need a defensible technical narrative.

What we are not doing. This is not a competitive teardown, and it is not a product brochure. Where Yotta and Yntraa Cloud appear, it is as an example of a sovereign managed DBaaS provider, alongside others. The evaluation criteria in Chapter 8 and Appendix B are written to be applied to any provider — including ours.

YOTTA | yntraa CLOUD



Chapter 1 — The Evolution Of Database Management

Chapter 1 — The Evolution of Database Management

Understanding why managed DBaaS exists requires understanding what it replaced. Each stage in this progression solved a real problem and created a new one.

1.1 The stages

Stage 1 — Traditional on-premises (1980s to early 2000s)

Databases ran on physical servers in owned data centres. A dedicated DBA team handled installation, tuning, backup, and recovery. Capacity was procured in advance and sized for peak load, which meant most hardware sat idle most of the time. Provisioning a new database environment took weeks or months, gated by purchase orders, rack space, and change windows. High availability meant expensive clustering software and a standby site that was tested rarely, if ever.

What it solved: full control, predictable performance, data physically in your custody. *What it cost:* capital lock-up, slow delivery, deep dependence on a small number of specialists.

Stage 2 — Virtualisation (early to mid 2000s)

Hypervisors decoupled the database from the physical server. Provisioning dropped from months to days. Consolidation ratios improved hardware utilisation dramatically, and snapshots made rollback practical for the first time.

What it solved: hardware efficiency, faster environment creation, easier test/dev refresh. *What it did not solve:* the database itself was still installed, patched, tuned, backed up, and recovered by hand. Virtualisation moved the box; it did not move the work.

Stage 3 — Cloud infrastructure, IaaS (mid 2000s to mid 2010s)

Compute and storage became API-callable and consumption-priced. A database server could be running in minutes. Capital expenditure shifted to operating expenditure.

What it solved: elasticity, geographic reach, no capital procurement cycle. *What it did not solve:* still self-managed. Running PostgreSQL on a cloud VM is architecturally identical to running it on a VM in your own data centre. The bill changes; the operational burden does not. Many organisations discovered this the hard way, having assumed "moving to cloud" would reduce database operations effort. It did not, because they had moved the infrastructure model and left the operating model alone.

Stage 4 — Managed services (mid 2010s onward)

Providers began operating the software layer, not just the infrastructure beneath it. Provisioning, patching, backup, failover, and monitoring became provider-side automation exposed through a control plane. The customer stopped installing and started consuming.

What it solved: operational burden, consistency, speed of delivery, predictable recovery behaviour. *What it introduced:* a shared responsibility boundary that must be understood, and a dependence on the provider's engineering quality.

Stage 5 — Managed Database as a Service (current)

The mature form of stage 4 for databases specifically: multi-engine catalogues, self-service

provisioning, automated high availability across fault domains, point-in-time recovery, integrated observability, contractual SLAs, and lifecycle automation for version upgrades delivered as a product with a defined support model behind it.

1.2 What actually changed

The important shift across these five stages is not technological. It is the answer to a single question: **when this database fails at 3 a.m., whose pager rings?**

- On-premises: yours.
- Virtualised: yours.
- IaaS: yours.
- Managed services: the provider's, for the platform.
- Managed DBaaS: the provider's, against a contractual SLA, with defined recovery objectives.

Every other difference follows from that one.

1.3 A note on the direction of travel

None of these stages fully replaced the one before it. Substantial self-managed estates still run, and for good reasons covered in Chapter 3 and Chapter 16. But the centre of gravity for new workloads has moved decisively toward managed consumption, and the reasons are structural rather than fashionable: the skills market for senior DBAs is tight and getting tighter, database estates are growing faster than the teams that manage them, and regulatory expectations around patching cadence, encryption, auditability, and tested recovery have risen to a level that manual operations struggle to sustain across dozens or hundreds of instances.

YOTTA | yntraa CLOUD



Chapter 2 — Getting The Terminology Straight

CHAPTER 2 — GETTING THE TERMINOLOGY STRAIGHT

Before comparing models, we should agree on words. Most of the confusion this paper addresses comes from four terms used interchangeably when they mean different things.

2.1 The terms

Database as a Service (DBaaS) is an umbrella term for consuming database capability through a service interface rather than building and operating every underlying component directly. Provisioning is typically on demand, and access is provided through a database endpoint, API, console, or command-line interface. Pricing may be consumption-based, provisioned-capacity-based, subscription-based, or a combination of these models.

Hosted database Someone else owns the hardware and the facility; you still install, configure, patch, and operate the database. Classic colocation and traditional hosting fall here. *Hosted is not managed*. This distinction is routinely lost in vendor marketing and is worth policing in your own RFPs.

Managed DBaaS The provider operates the database platform provisioning, HA, backup, patching, monitoring, recovery, scaling, under a defined SLA. The customer operates the data, schema, queries, and application. This is the model this paper is about.

Fully managed / autonomous Marketing terms for higher automation within managed DBaaS: automatic index recommendations, self-tuning, automatic storage growth, automated minor version patching. Useful capabilities. Not a separate category. And worth testing rather than accepting “fully managed" has no agreed definition, so it means whatever the provider's datasheet says it means.

"Hyperscaler DBaaS" Managed DBaaS delivered by a hyperscale cloud provider. That is the entire definition. It is a statement about *who is selling*, not about *what is being sold*. There is no technical property that a database service acquires by being operated by a company above a certain revenue threshold.

2.2 The distinction that matters: control plane and data plane

Managed DBaaS is best understood as a split between two planes.

The control plane is the provider's automation layer: the APIs and console that create instances, orchestrate failover, schedule backups, apply patches, expand storage, and expose metrics. It is the product. When people say, "the provider manages the database", the control plane is what they mean.

The data plane is the running database engine and your data inside it. The engine is usually standard PostgreSQL, MySQL, MongoDB, Redis, or similar, the same software you could install yourself.

This split explains three things that otherwise seem contradictory:

1. **Why managed DBaaS does not imply proprietary technology.** The engine is typically the same open-source software running everywhere else. Your SQL, drivers, ORMs, and application code do not change.
2. **Why providers differ so much despite offering "the same" databases.** The

engine is common; the control plane is where each provider's engineering investment shows up, and control plane maturity varies enormously.

3. **Why some superuser operations are restricted.** The provider's automation must be able to rely on certain invariants replication topology, configuration parameters, and extension set. Managed services restrict what would break that automation. This is a real constraint and Chapter 3 treats it honestly.

2.3 Where DBaaS sits among service models

Model	Provider manages	Customer manages	Database example
On premises	Nothing	Everything	PostgreSQL on your own hardware
IaaS	Facility, hardware, virtualisation	OS, database, data, application	PostgreSQL on a cloud VM
Managed DBaaS (PaaS)	Through the database platform	Data, schema, queries, application	Managed PostgreSQL service
SaaS	Everything including the application	Data and usage	A CRM you log into

Managed DBaaS is PaaS applied to databases. That is its correct place in the taxonomy, and it holds regardless of who delivers it.



Chapter 3 — What Is A Self-Managed Database?

Chapter 3 — What Is a Self-Managed Database?

3.1 Definition

In a self-managed model, the customer owns the complete database lifecycle. The provider, if there is one, supplies infrastructure and stops there. Everything from the operating system upward is the customer's responsibility, whether the database runs on owned hardware, in a colocation facility, or on a cloud VM.

The location is irrelevant to the model. A PostgreSQL instance on an EC2 instance is self-managed. So is a PostgreSQL instance in your own rack. The bill differs; the responsibility does not.

3.2 What the customer owns

Installation and configuration — OS hardening, kernel and storage tuning, database installation, parameter tuning for the workload, extension management, connection pooling.

High availability — designing and building replication, configuring automatic failover, and fencing, managing quorum and split-brain protection, load balancing reads, keeping the standby genuinely usable.

Backup and recovery — choosing a backup strategy, sizing, and securing backup storage, implementing point-in-time recovery through WAL or binlog archiving, verifying backup integrity, and the step most often skipped - periodically restoring to prove recovery works.

Patching and upgrades — tracking CVEs across the database and OS, testing patches, negotiating change windows, executing minor patches and major version upgrades, handling extension and driver compatibility, planning rollback.

Monitoring and observability — deploying and maintaining a metrics stack, defining thresholds, building dashboards, managing alert routing and on-call rotation, retaining query performance history.

Scaling — capacity forecasting, vertical resize planning, read replica provisioning, sharding or partitioning design, storage expansion before it becomes an incident.

Disaster recovery — secondary site design, cross-site replication, documented runbooks, and periodic DR drills with evidence for auditors.

Security — network segmentation, TLS certificate lifecycle, encryption at rest and key management, privileged access control, audit logging, and satisfying auditors that all of it works.

3.3 Advantages

Complete control. Every parameter, every extension, every patch decision. For workloads with unusual requirements—such as exotic extensions, kernel-level tuning, custom storage engines, or tight coupling to a specific patch level—this can be decisive.

Deep customisation. Superuser access, filesystem access, custom builds, third-party plugins. Some legacy applications simply require it.

No platform lock-in. The database is a standard installation. Moving it means moving a

standard installation.

Change control on your timetable. Nothing changes unless you change it. In some regulated change-freeze environments this is genuinely valuable.

Cost efficiency at scale and at steady state. For very large, stable, predictable estates with an existing skilled team already on payroll, self-managed can be cheaper per unit, provided the team is fully loaded and the estate is genuinely stable.

3.4 Disadvantages

Skilled DBAs required, and they are scarce. Not one DBA but a team, because 24×7 coverage needs rotation, and expertise across PostgreSQL, MySQL, MongoDB, Redis, and Oracle is rarely found in one person. Senior database engineers are among the harder infrastructure roles to hire and retain in the Indian market, and losing one is an operational risk, not just an HR event.

Operational burden compounds with scale.

At small scale, databases can be managed individually. At enterprise scale, success depends on automation, governance, and repeatable operational processes.

Patching alone, across engines and environments, becomes a permanent workstream.

Slow provisioning. Days to weeks for a production-grade, HA-configured, backed-up, monitored instance. Development teams route around this, which is how shadow databases appear.

Higher risk concentration. Every safeguard depends on being correctly implemented and continuously maintained by your own team. The common failure modes are well documented and depressingly consistent: backups that were never restore-tested, standbys that had silently stopped replicating, patches deferred past the point of exposure, and a DR site that had not been failed over to since commissioning.

Undertested recovery. Most organisations discover their true RPO and RTO during an incident rather than during a drill.

Opportunity cost. Time senior engineers spend patching databases is time not spent on the systems that differentiate the business. This is the largest and least measured cost in the model.

3.5 An honest position

Self-managed is not obsolete and this paper does not argue that it is. It is the right answer for a defined set of situations such as deep engine customisation, extreme performance engineering, existing large-scale investment in tooling and people, and specific licensing or contractual arrangements. Chapter 16 sets out where it wins.

What has changed is that self-managed is no longer the *default*. It is now a deliberate decision that should be justified workload by workload, and the justification should be more substantial than "that is how we have always done it."


YOTTA | yntraa CLOUD
Chapter 4 — What Is Managed DBaaS?

Chapter 4 — What Is Managed DBaaS?

4.1 Definition

In managed DBaaS the provider assumes operational responsibility for the database platform under a defined service level agreement. The customer consumes a database endpoint and retains full responsibility for what is inside the database and how the application uses it.

The service is delivered through a control plane via console, API or CLI, that exposes lifecycle operations as automation rather than as tickets and runbooks.

4.2 The shared responsibility model

The single most common cause of failed managed DBaaS deployments is misunderstanding this boundary. Customers assume "managed" means "someone else worries about everything", then discover during an incident that query performance, schema design, and data correctness were always theirs.

The provider is responsible for the platform. The customer is responsible for the data and the application.

That sentence is the entire model. Everything below is elaboration.

4.3 Responsibility matrix

Responsibility area	Self-Managed	Managed DBaaS
Data centre facility, power, cooling	Customer / colo provider	Provider
Physical security	Customer / colo provider	Provider
Server hardware and lifecycle	Customer	Provider
Storage infrastructure	Customer	Provider
Network infrastructure	Customer	Provider
Hypervisor / virtualisation	Customer	Provider
Guest operating system	Customer	Provider
OS patching and hardening	Customer	Provider
Database installation	Customer	Provider
Database configuration baseline	Customer	Provider
Database minor version patching	Customer	Provider
Major version upgrade execution	Customer	Provider (customer schedules)
High availability architecture	Customer	Provider
Automatic failover	Customer	Provider
Replication management	Customer	Provider
Backup execution and retention	Customer	Provider
Backup integrity verification	Customer	Provider
Point-in-time recovery capability	Customer	Provider
Restore initiation and target choice	Customer	Customer
Disaster recovery infrastructure	Customer	Provider

Responsibility area	Self-Managed	Managed DBaaS
DR invocation decision	Customer	Customer (provider executes)
Platform monitoring and alerting	Customer	Provider
Capacity alerting	Customer	Provider
Encryption at rest	Customer	Provider
Encryption in transit (TLS)	Customer	Provider (customer enforces client-side)
Key management	Customer	Provider, or customer via BYOK
Network isolation and firewalling	Customer	Provider (customer configures rules)
Platform audit logging	Customer	Provider
Platform vulnerability management	Customer	Provider
Scaling execution	Customer	Provider (customer requests)
Capacity planning	Customer	Shared
Schema design	Customer	**Customer**
Indexing strategy	Customer	**Customer**
Query design and optimisation	Customer	**Customer**
Data model	Customer	**Customer**
Application connection handling	Customer	**Customer**
Database user and role management	Customer	**Customer**
Data classification and governance	Customer	**Customer**
Data accuracy and integrity	Customer	**Customer**
Application-level access control	Customer	**Customer**
Compliance of data content	Customer	**Customer**

The bolded rows never transfer. No provider, at any price, takes responsibility for a badly designed schema or an unindexed query scanning a hundred million rows. Managed DBaaS removes operational toil; it does not remove the need for competent data engineering.

4.4 What the customer gains

Time. Provisioning drops from weeks to minutes. Environments become disposable, which changes how teams test.

Consistency. Every instance is built the same way by the same automation. Configuration drift, the source of an enormous share of production incidents, largely disappears.

Tested recovery. Backups run on schedule, are verified, and point-in-time recovery is a documented, exercised procedure rather than a hopeful one.

Predictable availability. Failover is automated and measured in seconds to a couple of minutes, not in however long it takes to reach the right person.

Reduced patch exposure. Security patching follows a defined cadence rather than the gaps between other priorities.

Audit evidence. Encryption, access logs, patch records, and backup reports are produced as a matter of course — which is significantly easier to hand an auditor than a set of screenshots assembled the week before.

Skill leverage. Your database specialists move from routine maintenance to data architecture, performance engineering, and modelling work that actually differentiates.

4.5 What the customer gives up

An honest accounting matters here, because overselling this model is how disappointment gets manufactured.

Superuser access is usually restricted. The provider's automation depends on invariants that unrestricted superuser access can break. Most managed services grant an elevated role short of full superuser. Applications that genuinely need OS-level or superuser access will need a different model.

Extension and plugin catalogues are curated. Providers support a tested set. If you depend on an unusual extension, verify support before committing; this is one of the most common late-stage surprises in migrations.

Parameter tuning is bounded. Most workloads' relevant parameters are exposed through parameter groups; a small number are locked because the platform depends on them.

Version timing is negotiated, not unilateral. You control the maintenance window; you do not control the support calendar indefinitely. End-of-life versions will eventually be retired.

Dependence on provider engineering quality. You are trusting the provider's automation, support, and operational discipline. This is precisely why Chapter 8 and Appendix B exist, the correct response to this dependency is rigorous evaluation, not avoidance.



Chapter 5 — Deployment Models Within Managed DBaaS

Chapter 5 — Deployment Models Within Managed DBaaS

Managed DBaaS is one service model, but it is delivered through several deployment topologies. This is where genuine architectural difference lives, far more so than in provider branding and it is the part of the decision most often skipped.

5.1 Multi-tenant public

Database instances run on shared provider infrastructure, isolated logically and at the network layer. Fastest to provision, lowest entry cost, most elastic.

Fits: development and test, SaaS applications, workloads without dedicated-tenancy mandates, variable demand. *Consider:* noisy-neighbour effects at the storage or network layer, and whether your regulator or internal policy accepts shared infrastructure for this data class.

5.2 Virtual private / dedicated network

The database sits inside the customer's own virtual network with no public endpoint. Access is via private connectivity only. This is the default posture for most enterprise production workloads and should be treated as a baseline requirement rather than a premium feature.

5.3 Dedicated infrastructure

The customer's databases run on physically dedicated hosts or a dedicated cluster. No compute-layer tenancy sharing. Higher cost, predictable performance, and an answer to regulators and auditors who ask direct questions about tenancy.

Fits: BFSI core systems, government workloads, healthcare data, anything under a dedicated infrastructure mandate.

5.4 Private cloud managed DBaaS

The managed database platform is deployed inside the customer's own data centre or a dedicated private environment, and the provider operates it remotely. The customer gets the managed operating model without the data leaving their facility.

Fits: organisations with existing data centre investment, strict physical custody requirements, or workloads that cannot leave a defined boundary.

5.5 Hybrid

Production on one side, DR, or analytics replicas on the other; or a phased migration where both estates run in parallel for an extended period. Requires the provider to support replication across the boundary and to be clear about which side each responsibility sits on. Hybrid is common in practice and under-discussed in vendor material.

5.6 Choosing

Requirement	Indicated model
Lowest cost, fastest start	Multi-tenant public
Standard enterprise production	Virtual private / dedicated network

Requirement	Indicated model
Regulatory dedicated-tenancy mandate	Dedicated infrastructure
Data must not leave your facility	Private cloud managed DBaaS
Phased migration or split estate	Hybrid
Sovereignty and jurisdiction mandate	Any of the above, delivered in-country by an in-country entity

The last row is the subject of Chapter 12, and it is the one that most often reshapes a shortlist.



Chapter 6 — Who Provides Managed DBaaS?

Chapter 6 — Who Provides Managed DBaaS?

Managed DBaaS is a service model. Many organisations deliver it. Their categories differ; the model does not.

6.1 Hyperscale cloud providers

AWS, Microsoft Azure, Google Cloud, and Oracle Cloud Infrastructure offer managed relational and non-relational database services across a wide engine catalogue, integrated with their own compute, identity, and analytics stacks.

Characteristic strengths: very broad service catalogue, global footprint, deep ecosystem integration, large partner networks. *Characteristic considerations:* pricing complexity — particularly egress and cross-zone data transfer; support quality tiered by contract level; jurisdiction and parent-company legal exposure; limited commercial flexibility for mid-sized buyers.

6.2 Regional and sovereign cloud providers

Providers operating within a specific national jurisdiction like Yotta and others in India, OVHcloud in Europe, and comparable providers in other markets, deliver the same managed model with in-country infrastructure, in-country legal entity, and in-country support.

Characteristic strengths: data residency and jurisdictional clarity, lower latency to domestic users, local-language and local-time zone support, commercial flexibility, alignment with government procurement frameworks. *Characteristic considerations:* narrower global footprint; catalogue breadth varies by provider and should be verified against actual requirements rather than assumed.

6.3 Database vendors

MongoDB Atlas, EDB BigAnimal, Redis Cloud, Neo4j Aura, Confluent Cloud, and similar offerings are managed services operated by the company that builds the engine.

Characteristic strengths: deepest possible engine expertise, earliest access to new versions, engine-specific tooling. *Characteristic considerations:* single-engine focus, so a multi-engine estate means multiple vendor relationships; the underlying infrastructure is frequently rented from a hyperscaler, which means the residency question needs to be asked twice, once of the vendor and once of the infrastructure beneath them.

6.4 Managed service providers and system integrators

Large IT services organisations operate managed database platforms for enterprise clients, often built on open-source engines and often on infrastructure belonging to someone else.

Characteristic strengths: deep customisation, integration with wider managed IT contracts, willingness to take on legacy estates. *Characteristic considerations:* automation maturity varies widely; some are genuine platform products; others are staff augmentation with a portal on top. This is worth probing directly.

6.5 The pattern

Provider type	Example	Service model
Hyperscaler	AWS RDS	Managed DBaaS
Hyperscaler	Azure Database for PostgreSQL	Managed DBaaS
Hyperscaler	Google Cloud SQL	Managed DBaaS
Hyperscaler	Oracle Autonomous Database	Managed DBaaS
Sovereign / regional CSP	Yntraa Cloud SutraDB, FlexiDB	Managed DBaaS
Sovereign / regional CSP	OVHcloud managed databases	Managed DBaaS
Database vendor	MongoDB Atlas	Managed DBaaS
Database vendor	EDB BigAnimal	Managed DBaaS
Database vendor	Redis Cloud	Managed DBaaS
MSP / SI	Various enterprise platforms	Managed DBaaS

Ten entries. Four provider types. One service model.



Chapter 7 — Managed DBaaS Is Not A Different Category

Chapter 7 — Managed DBaaS Is Not a Different Category

This is the central chapter.

7.1 The correct taxonomy

```
Database Operating Models
│
├── Self-Managed
│   ├── On-premises hardware
│   ├── Colocation
│   └── Cloud IaaS (VM-hosted)
│
└── Managed DBaaS
    ├── Hyperscalers
    │   ├── AWS
    │   ├── Microsoft Azure
    │   ├── Google Cloud
    │   └── Oracle Cloud
    ├── Sovereign / regional CSPs
    │   ├── Yotta — Yntraa Cloud (SutraDB, FlexiDB)
    │   ├── OVHcloud
    │   └── Others by market
    ├── Database vendors
    │   ├── MongoDB Atlas
    │   ├── EDB BigAnimal
    │   └── Redis Cloud
    └── Managed service providers
```

Two branches. Many providers under one of them. "Hyperscaler DBaaS" is a node three levels down the tree, not a sibling of the tree's root.

7.2 The incorrect taxonomy, and why it persists

The framing this paper argues against looks like this:

```
├── Self-Managed Databases
├── DBaaS
├── Managed DBaaS
└── Hyperscaler DBaaS          ← presented as a distinct class
```

This is a marketing construct, and it works on exactly the misconception it creates. It performs three jobs for whoever publishes it:

1. It implies a capability tier that scale alone does not confer.
2. It makes "which provider" feel like a *category* decision rather than a *capability* decision, and category decisions are made earlier, faster, and with less scrutiny.
3. It removes providers from consideration before their capabilities are ever compared.

The test is simple. **A genuine category difference changes what the customer is responsible for.** Self-managed and managed DBaaS pass that test; the responsibility matrix in Chapter 4 is materially different between them. "Hyperscaler DBaaS" fails it. The responsibility matrix for AWS RDS and for a well-built sovereign managed PostgreSQL service is the same matrix. The provider changes; the model does not.

7.3 Applying the test

Ask of any proposed database category:

1. Does the responsibility boundary move? If not, it is not a category.
2. Does the customer's operational obligation change? If not, it is not a category.
3. Does the underlying engine change in a way that affects the application? Usually not — it is standard PostgreSQL, MySQL, MongoDB, or Redis in every case.
4. Is the difference in *capability* rather than in *kind*? Then it belongs in an evaluation scorecard, not in a taxonomy.

Scale, catalogue breadth, and global footprint are real differences. They belong in criterion four. They are reasons to prefer a provider for a specific requirement. They are not reasons to invent a category.

7.4 What follows from getting this right

For buyers: the shortlist is built on capability, compliance fit, support model, and commercial terms, not on brand tier. Providers that meet the requirements get evaluated on merit.

For architects: the design question separates cleanly into two decisions. First, self-managed or managed? Second, which provider and which deployment model? Conflating them produces bad architecture.

For marketing and pre-sales teams: the argument is not "we are as good as a hyperscaler." It is "we deliver the same service model, and here is where our capability, jurisdiction, support, and commercials fit your requirements better." That is a stronger position because it is a true one and it is verifiable.



Chapter 8 — Where Providers Genuinely Differ

Chapter 8 — Where Providers Genuinely Differ

Rejecting a false category does not mean claiming all providers are equivalent. They differ substantially, just not in kind. These are the dimensions that carry real weight, and they are the ones an evaluation should score.

8.1 Engine coverage

Which engines, which versions, how quickly new major versions are supported, and how long old ones are maintained? A provider supporting one engine deeply may serve you better than one supporting twelve shallowly or worse, depending on your estate.

8.2 Control plane maturity

The most under-examined dimension in most evaluations, and often the most consequential. Does the provider expose a complete API, or is the console the only real interface? Is there Terraform or equivalent infrastructure-as-code support? Can instance creation, scaling, backup policy, and restore be scripted end to end? A control plane that cannot be automated pushes work back onto your team, which quietly erodes the benefit you bought the service for.

8.3 High availability architecture

Synchronous or asynchronous replication? Automatic failover, and measured at what time? Failure domain separation — same rack, same availability zone, different zone, different region? Is there a documented failover time, and has the provider demonstrated it? "Multi-AZ" without stated replication semantics tells you very little.

8.4 Backup and recovery capability

Full and incremental cadence, retention range, point-in-time recovery granularity and window, cross-region backup copies, restore speed for a large database, self-service restore, and whether backups are integrity-verified rather than merely completed. Ask specifically how long a restore of your largest database would take. An untested restore time is an untested RTO.

8.5 Scaling behaviour

Vertical resize with or without downtime. Read replica provisioning and lag characteristics. Storage autoscaling. Connection pooling built in or bolted on. Whether scaling is self-service or a support ticket, and how long it takes.

8.6 Security capability

Encryption at rest and in transit as default rather than option. Customer-managed keys and BYOK. Network isolation model. Role-based access control granularity. Audit logging depth and export. Integration with enterprise identity. Vulnerability management and disclosed patch cadence.

8.7 Observability

Metrics available and retention. Slow query capture and query performance history. Log

access and export to your own SIEM. Alerting integration. Some providers expose rich telemetry; others expose CPU and memory and little else.

8.8 SLA structure

This deserves more scrutiny than it usually receives. Read carefully:

- What is the availability commitment, and is it for the single-instance or HA configuration?
- What counts as downtime, and what is excluded? Maintenance windows are frequently excluded.
- What is the measurement window — monthly or annual? Annual measurements hide a lot.
- What is the remedy? Almost always service credits, capped at a fraction of the affected month's spend. It is a commitment mechanism, not compensation.
- Is there a separate RPO and RTO commitment, or only availability? Many SLAs cover uptime and say nothing about recovery.
- Is there a support response-time SLA distinct from the availability SLA?

8.9 Support model

Included tier versus paid tier. Response and resolution targets by severity. Whether escalation reaches an engineer who knows the platform or a script. Time zone alignment and local-language capability. Whether a named technical account manager is available, and at what threshold.

8.10 Jurisdiction and compliance

Where data is stored, where it is processed, where backups live, which legal entity holds the contract, which law governs it, and which certifications and empanelment are in force. Chapter 12 develops this.

8.11 Commercial model

Pricing structure and predictability, egress and inter-zone data transfer charges, committed-use discounts, reserved capacity, currency of billing and consequent FX exposure, and contractual flexibility for mid-sized enterprises.



Chapter 9 — Availability, Resilience, And Disaster Recovery

Chapter 9 — Availability, Resilience, and Disaster Recovery

Availability is the reason most organisations move to managed DBaaS, and it is the area where terminology is loosest. This chapter defines the vocabulary an evaluation should use.

9.1 RPO and RTO

Recovery Point Objective (RPO) — the maximum acceptable data loss, expressed in time. RPO of five minutes means up to five minutes of committed transactions may be lost in a disaster.

Recovery Time Objective (RTO) — the maximum acceptable time to restore service.

Every database in the estate should have a stated RPO and RTO derived from business impact, not from what the current architecture happens to deliver. Most organisations have this backwards: they discover their RPO after an incident. A tiering exercise such as mission-critical, business-critical, standard, development with defined RPO/RTO per tier is the foundation of any serious resilience conversation.

9.2 High availability patterns

Single instance. No redundancy. Failure means downtime until restore completes. Appropriate for development and test only.

Single instance with automated backup. RPO equals the backup or WAL archive interval; RTO equals restore time, which for a large database can be hours.

Synchronous standby in a second failure domain. Transactions commit only when the standby confirms. RPO effectively zero. RTO in seconds to a couple of minutes with an automated failover. Some write latency costs, and the standby must be close enough for that cost to be acceptable.

Asynchronous replica. Lower write latency, non-zero RPO equal to replication lag. Suitable for read scaling and for DR across longer distances.

Multi-region. Protects against regional failure. Almost always asynchronous, so RPO is non-zero, and lag must be monitored as a first-class metric.

9.3 Failure domains

"Multi-AZ" or "high availability" means nothing until you know what the domains actually are. Ask directly: are the primary and standby in separate racks, separate power and cooling paths, separate availability zones, separate buildings, or separate cities? Two nodes on the same power distribution unit are not high availability regardless of what the console calls them.

9.4 What managed DBaaS changes

In self-managed, all the above is architecture the customer designs, builds, tests, and maintains. In managed DBaaS, it is a configuration choice at provisioning time, backed by provider automation and a contractual SLA.

That is a substantial reduction in risk provided you verify the semantics rather than trusting the label, and provided you still test. Which brings us to the point most often missed:

9.5 DR is still a shared responsibility

The provider builds and operates the DR capability. The customer decides the tier, defines acceptable RPO and RTO, decides when to invoke DR, validates that the application actually functions correctly after failover, and participates in drills.

Managed DBaaS does not remove the need for DR testing. It makes DR testing possible without a six-week project. Organisations under regulatory obligation to evidence periodic DR drills should confirm the provider supports non-disruptive drill capability and produces evidence artefacts. This is a concrete, auditable requirement worth writing into the contract.



Chapter 10 — Security And Compliance Architecture

Chapter 10 — Security and Compliance Architecture

Security is where the shared responsibility model is most frequently misunderstood, and the misunderstanding is expensive.

10.1 Encryption

At rest. Data files, backups, and snapshots encrypted with AES-256 or equivalent. Should be default and non-optional. Confirm that *backups* are encrypted — this is occasionally a separate setting and an easy audit finding.

In transit. TLS between application and database, and between replication peers. The provider enables it; the customer must enforce it client-side. A managed database offering TLS while the application connects without it is a customer-side failure, and it happens regularly.

Key management. Provider-managed keys are the default. Customer-managed keys or BYOK give the customer control over key lifecycle and, importantly, the ability to revoke access cryptographically. For regulated workloads, BYOK is frequently a requirement rather than a preference — confirm support and confirm which key management service integrations are available.

10.2 Network security

Private endpoints with no public exposure should be the production default. Beyond that: network-level access control lists, security groups scoped tightly, private connectivity to the application network, and no database directly reachable from the internet. If a managed database service defaults to a public endpoint, that default should be changed on day one.

10.3 Access control

Provider-side administrative access should be governed by least privilege, just-in-time elevation, and logged sessions. Ask the question directly: **under what circumstances can provider personnel access customer data, what approval is required, and is that access logged and available to the customer?** A provider that cannot answer this crisply has not thought about it enough.

On the customer side, database roles and privileges remain the customer's responsibility. Application accounts with superuser-equivalent privileges are a common and avoidable finding.

10.4 Audit logging

Platform audit logs — provisioning, configuration changes, access events, administrative operations — should be available to the customer and exportable to their own SIEM. Database-level audit logging, capturing DML and DDL as required, should be configurable. Retention needs to meet the longest applicable regulatory requirement, and log immutability matters for evidentiary value.

10.5 Vulnerability and patch management

The provider should publish a patch cadence, an emergency patch process for critical CVEs,

and a communication mechanism. Ask how quickly a critical database CVE was patched across the fleet the last time one was disclosed. The answer is informative, and it is verifiable.

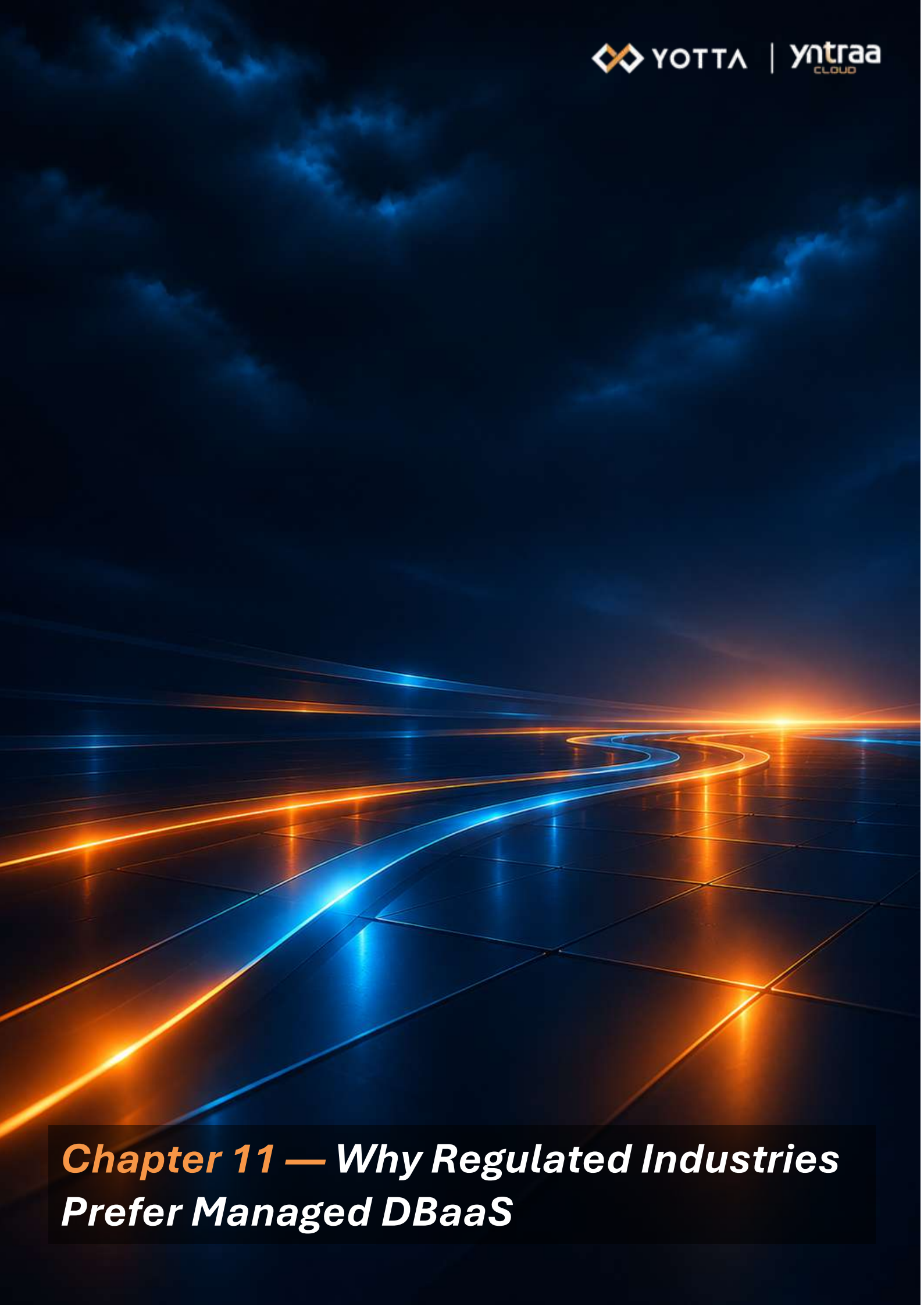
10.6 Certifications and attestations

Standards commonly relevant to enterprise and regulated buyers include ISO/IEC 27001 for information security management, ISO/IEC 27017 and 27018 for cloud-specific controls and cloud PII protection, SOC 1 and SOC 2 Type II reports, PCI DSS for cardholder data environments, and in India MeitY empanelment with STQC audit for public sector procurement.

Two cautions. First, certifications describe the provider's control environment; they do not make the customer compliant. Compliance is a joint outcome. Second, verify scope — a certification may cover the data centre and not the specific service, or one region and not another. Ask for the scope statement, not just the logo.

10.7 The division, summarised

Security domain	Provider	Customer
Physical and facility security	✓	
Infrastructure hardening	✓	
Encryption at rest	✓	
TLS availability	✓	
TLS enforcement in application		✓
Key management	✓ (or customer via BYOK)	✓ if BYOK
Network isolation capability	✓	
Firewall rule configuration		✓
Platform patching	✓	
Platform audit logging	✓	
Database user and role design		✓
Application-level authorisation		✓
Data classification		✓
Consent, retention, and erasure of personal data		✓
Regulatory compliance of data content		✓



Chapter 11 — Why Regulated Industries Prefer Managed DBaaS

Chapter 11 — Why Regulated Industries Prefer Managed DBaaS

Regulated sectors were, for a long time, the most resistant to managed database services. They are now among the fastest adopters. The reversal is worth understanding because it was driven by evidence rather than by fashion.

11.1 What changed

The original objection was that outsourcing database operations meant losing control, and losing control meant regulatory risk. Three things undermined that argument.

First, the audit evidence turned out to be better. Regulators increasingly ask for demonstrable, continuous evidence: patch records, encryption attestations, access logs, verified backups, tested recovery. Manual operations across a large estate produce this inconsistently and assembling it for an audit is a project. Automated platforms produce it as a by-product.

Second, the failure data pointed the other way. The incidents that damage regulated organisations are overwhelmingly operational rather than exotic: unpatched vulnerabilities, backups that could not be restored, misconfigured access, failover that did not work. These are precisely the failure modes that automation addresses.

Third, the skills gap became structural. Maintaining 24×7 expert database operations across multiple engines is genuinely difficult to staff and sustain, and the difficulty is not decreasing.

11.2 Sector requirements

Government and public sector. Data residency within national borders. Procurement through empanelled providers in India, MeitY empanelment following STQC audit is a practical prerequisite for public sector procurement. Auditability, defined SLAs, and long-term data retention.

Banking, financial services and insurance. The RBI's directive on storage of payment system data requires payment system data to be stored within India. Sector regulators impose expectations on outsourcing governance, business continuity, incident reporting, and audit access — including, frequently, the regulator's own right to audit the service provider. Transaction integrity, tested DR, and detailed audit trails are non-negotiable.

Healthcare. Patient data is sensitive personal data under the DPDP framework, and health records carry long retention obligations. Encryption, strict access control, comprehensive audit logging, and high availability for clinical systems are baseline.

Telecom. Licence conditions impose obligations on subscriber data handling and lawful interception support. Extreme scale, very high availability expectations, and specific retention requirements.

Manufacturing. Increasingly regulated through supply chain and quality frameworks. Traceability data, batch records, and IoT telemetry with long retention. Availability requirements tied directly to production line uptime — an hour of database downtime can be an hour of stopped production.

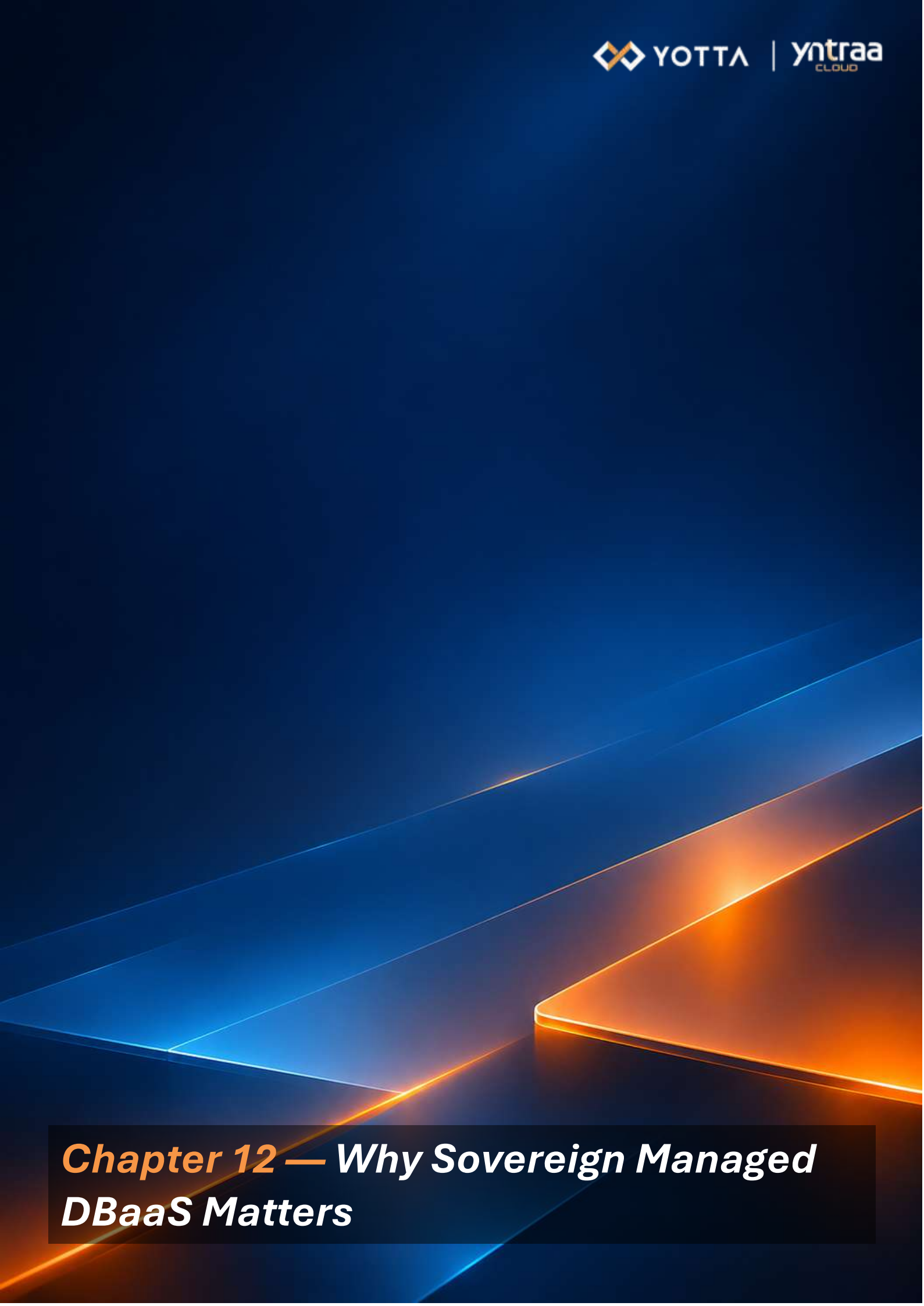
11.3 The requirements these sectors share

Requirement	How managed DBaaS addresses it
Data residency	Provider commits to storage and processing location, including backups and DR copies
Encryption	Default at rest and in transit, with BYOK where required
Auditability	Platform audit logs, configuration history, exportable evidence
Availability	Automated HA with contractual SLA
Disaster recovery	Provider-operated DR with defined RPO and RTO, and drill capability
Patch currency	Defined cadence and emergency patch process
Access control	Least privilege, logged administrative access, identity integration
Backup verification	Automated backups with integrity checks and tested restore
Incident reporting	Provider notification process aligned to regulatory timelines — in India, CERT-In directions require reporting of specified cyber incidents within six hours of noticing them
Long-term retention	Configurable retention meeting sector requirements

11.4 The residual obligation

Regulatory accountability does not transfer with operational responsibility. Under the DPDP framework the Data Fiduciary remains accountable for personal data even when a Data Processor handles it. The provider is a processor; the customer is the fiduciary.

This makes three things contractual necessities rather than nice-to-haves: a written data processing agreement defining the provider's obligations, audit rights sufficient to satisfy your regulator, and a defined breach notification process with timelines that let you meet your own reporting deadlines. Negotiate these before signing, not after an incident.



Chapter 12 — Why Sovereign Managed DBaaS Matters

Chapter 12 — Why Sovereign Managed DBaaS Matters

Sovereignty has moved from a niche procurement clause to a mainstream architectural requirement. This chapter explains what it actually means, because the term is used loosely.

12.1 Three distinct concepts

Data residency — where data is physically stored. The narrowest and most commonly claimed form. A global provider with an in-country region can offer residency.

Data sovereignty — which nation's laws govern the data. This depends on the legal entity holding the contract and its corporate parentage, not only on where the servers are. A subsidiary of a foreign parent may be subject to that parent's home jurisdiction under certain legal instruments, regardless of where the data sits.

Operational sovereignty — who can access the data and from where. Are support engineers accessing your systems from within the country? Is the operations team local? Can the provider technically or legally be compelled to grant access to a foreign authority?

A provider can satisfy the first without satisfying the second or third. Buyers who need sovereignty and evaluate only residency have not actually tested the requirement — this is one of the most consequential gaps in enterprise cloud procurement today.

12.2 The Indian regulatory position

India's data protection regime became operational with the notification of the DPDP Rules in November 2025, implementing the Digital Personal Data Protection Act, 2023. Key features relevant to database architecture:

- A **phased compliance timeline** of eighteen months from notification, with the substantive obligations landing in mid-2027. Organisations should treat the intervening period as the implementation runway.
- **Data Fiduciary accountability** for personal data, including where processing is delegated to a processor.
- **Breach notification** obligations, with affected individuals to be notified within seventy-two hours of the required Board notification.
- **Purpose-limited retention and erasure** — personal data must be erased once the purpose is served, with sector-specific default retention periods specified in the Rules.
- **Significant Data Fiduciary** obligations for designated entities, including annual data protection impact assessments, independent audits, and appointment of a Data Protection Officer based in India.
- **Cross-border transfer** permitted subject to conditions the government may prescribe, with certain categories of data to be notified as requiring storage in India.
- **Penalties** up to ₹250 crore for failure to maintain reasonable security safeguards.

Alongside this sit sector-specific requirements: RBI's payment system data storage directive, CERT-In's six-hour incident reporting directions and associated log retention requirements,

and MeitY empanelment with STQC audit as a practical gate for public sector cloud procurement.

The direction is unambiguous. Architectural decisions about where data lives and who can reach it are now compliance decisions with financial consequences.

12.3 What sovereign managed DBaaS provides

Jurisdictional clarity. Data stored and processed in India, under an Indian legal entity, governed by Indian law, with Indian courts as the forum. The question "which country's authorities could compel access to this data" has one answer.

Residency across the full data lifecycle. Not only the primary instance — backups, snapshots, DR replicas, logs, and telemetry. Ask specifically about backups and logs. This is where residency claims most often break down, and it is a straightforward contractual point to close.

Latency. Databases serving Indian users from Indian data centres remove tens of milliseconds per round trip compared with offshore hosting. For chatty applications making many sequential queries, this compounds into user-visible difference.

Local support. Engineers in the same time zone, working the same business calendar, contactable in the language your operations team uses. During a severity-one incident this is not a soft benefit.

Procurement alignment. MeitY empanelment, GeM availability, and familiarity with public sector contracting reduce friction in government and PSU procurement considerably.

Commercial terms. Billing in rupees removes FX exposure. Regional providers are typically more flexible on contract structure, committed-use terms, and support inclusions for mid-sized enterprises than global providers operating standardised global terms.

Ecosystem proximity. Direct connectivity to domestic exchanges and networks, and integration with the domestic partner and SI ecosystem.

12.4 Where Yotta fits

Yotta operates hyperscale data centre infrastructure in India and delivers managed database as a service through Yntraa Cloud — the SutraDB line for relational engines and FlexiDB for non-relational engines — under the same managed DBaaS service model described throughout this paper.

The infrastructure is Indian-owned and Indian-operated. Yotta's facilities comprise NM1 in Navi Mumbai, Asia's largest Tier IV data centre and India's only multi-tenant Tier IV facility certified under the Uptime Institute's Gold Tier Certification of Operational Sustainability; D1 in Greater Noida, a Tier III+ hyperscale facility; and G1 in GIFT City, Gujarat, built to IFSC-zone requirements for regulated financial workloads and data localisation. All three are carrier-neutral, with direct access to major telcos, ISPs, and cloud on-ramps. Edge facilities are planned across a further eight cities.

For a database platform, the practical consequences are these. Primary instances, replicas, backups, snapshots, and logs all remain within Indian borders. The contracting entity is Indian, and the governing law is Indian, which means the sovereignty question in 12.1 has a

single answer rather than a layered one. Operations and support are staffed in-country and in-time zone. Multiple facilities in geographically separated regions make cross-region disaster recovery possible without data leaving the jurisdiction — a configuration that is difficult to achieve with an offshore provider and an Indian residency mandate simultaneously. And Yotta has appointed a Data Protection Officer based in India, which is a specific obligation for Significant Data Fiduciaries under Rule 13 of the DPDP Rules.

The proposition is not that the service model is different. It is that the model is delivered with Indian residency across the full data lifecycle, under Indian jurisdiction, with in-country operations and support, and with commercial terms designed for the Indian enterprise and public sector market.

12.5 The honest caveat

Sovereignty is a requirement, not a virtue in itself. An organisation with no residency obligation, no regulatory exposure, and a genuinely global user base may reasonably prioritise global footprint over jurisdiction. The argument here is not that sovereign is always better. It is that where sovereignty is a requirement, it is a *hard* requirement - one that cannot be satisfied by a discount or a feature, and it should be tested properly rather than assumed away.


YOTTA | yntraa CLOUD
Chapter 13 — Total Cost Of Ownership

Chapter 13 — Total Cost of Ownership

Most database cost comparisons compare infrastructure prices and stop. That comparison is not merely incomplete — it is systematically biased, because it counts the costs that appear on an invoice and ignores the costs that appear on a payroll or in an incident report.

13.1 The cost categories

Infrastructure. Compute, storage, IOPS, network, backup storage, DR site capacity. Visible in both models. In self-managed on-premise, this includes capital cost amortisation, facility, power, cooling, and hardware refresh.

Software licensing. Database licences where applicable, plus backup software, monitoring tooling, HA and clustering software, and replication tools. Frequently significant in self-managed and frequently forgotten in comparisons. Under managed DBaaS, most of this is bundled.

Personnel. Fully loaded cost of DBAs and infrastructure engineers, including the multiplier needed for 24×7 coverage. A single DBA does not provide round-the-clock cover; a rotation requires three to five people. Add recruitment cost, training, certification, and the risk cost of attrition in a tight market.

Operations. Time spent on patching, upgrades, backup verification, monitoring maintenance, capacity planning, incident response, and DR drills. This is real cost even when it is nobody's line item.

Downtime. Expected unplanned downtime multiplied by cost per hour, which varies enormously by workload — a payments database and an internal reporting database are not comparable. Plus, planned maintenance downtime, which managed services reduce through online patching and rolling upgrades.

Provisioning delay. The business cost of environments taking weeks rather than minutes. Hard to quantify precisely, consistently underestimated, and often the largest number in the model for fast-moving product organisations.

Support. Vendor support contracts in self-managed; usually bundled or tiered in managed.

Opportunity cost. The highest-value item and the one never in the spreadsheet. Senior engineering time spent on database maintenance is time not spent on revenue-generating work.

13.2 Comparative view

Cost element	Self-Managed	Managed DBaaS
Compute and storage	Customer, often over-provisioned for peak	Included, elastic
Backup storage	Customer procures and manages	Included to retention limit
DR site capacity	Full duplicate infrastructure	Included in DR-enabled tiers
Database licensing	Customer	Usually bundled
Backup software	Customer	Included
Monitoring stack	Customer builds and maintains	Included

Cost element	Self-Managed	Managed DBaaS
HA / clustering software	Customer	Included
DBA team	Full team for 24×7	Reduced to data-focused roles
Patching effort	Recurring engineering time	Provider
Upgrade projects	Recurring project cost	Provider-executed
Incident response	Customer on call	Provider for platform
DR testing	Full project each cycle	Provider capability, customer validates
Provisioning lead time	Weeks	Minutes
Recruitment and retention	Significant and rising	Reduced
Unplanned downtime risk	Higher	Lower, SLA-backed

13.3 An honest statement of the economics

Managed DBaaS is not automatically cheaper on infrastructure. For a large, stable, predictable workload with an existing fully utilised DBA team, self-managed infrastructure cost per unit can be lower.

What managed DBaaS reliably reduces is **operational cost, operational risk, and time to deploy**. Whether total cost falls depends on the estate. It usually does for organisations with growing estates, constrained DBA capacity, high availability requirements, or compliance obligations — and the gap widens as the estate grows, because self-managed operational cost scales with instance count while managed cost scales with consumption.

13.4 Building a defensible TCO model

1. Model three years, not one. Year one favours self-managed if hardware is already sunk; years two and three tell the truth.
2. Include the full personnel multiplier for 24×7 coverage, not a single headcount.
3. Price downtime using your own workload's cost per hour, agreed with the business owner.
4. Include licensing for every tool the self-managed model requires and the managed model bundles.
5. Model the growth curve, not the current state. Estates grow; teams do not grow proportionally.
6. Include compliance effort — evidence collection, audit preparation, remediation.
7. State assumptions explicitly so the model can be challenged. A TCO model nobody can argue with is a model nobody believes.



Chapter 14 — Lock-In, Portability, And Exit

Chapter 14 — Lock-In, Portability, and Exit

The strongest argument against managed services is lock-in. It deserves a direct answer rather than a dismissal.

14.1 Separating the layers

Lock-in is not one thing. It operates at four layers, and they carry very different weight.

Engine lock-in. Low when the service runs standard open-source engines. Your SQL, drivers, ORMs, and application code are portable because the engine is the same software everywhere. A managed PostgreSQL service running community PostgreSQL presents the same wire protocol and the same dialect as PostgreSQL anywhere else. This is the layer that matters most to applications, and it is the layer where standard-engine managed services score well.

Control plane lock-in. Real but bounded. Provisioning scripts, Terraform providers, backup policies, and monitoring integrations are provider-specific and will need rework. This is typically weeks of engineering effort, not a rewrite.

Proprietary feature lock-in. This is where genuine lock-in accumulates. Provider-specific extensions, proprietary engine forks with capabilities not present upstream, serverless-only APIs, and tight coupling to the provider's identity, messaging, or analytics services. Each adopted proprietary capability is a deliberate trade of portability for benefit. Make the trade knowingly and record it.

Data gravity. The largest practical barrier at scale. Moving multiple terabytes takes time, bandwidth, and — critically — egress charges. Egress pricing is a material portability cost and should be modelled at contract time rather than discovered at exit.

14.2 Self-managed is not lock-in-free

The comparison is often made against an idealised alternative. In practice self-managed estates carry their own lock-in bespoke automation nobody else understands, tribal operational knowledge, hardware refresh cycles, perpetual licence commitments, and colocation contracts with notice periods. The freedom is real but partial.

14.3 Designing for portability

- Prefer standard open-source engines over proprietary forks unless the fork's capability is genuinely required.
- Keep application code free of provider-specific database extensions where a standard alternative exists.
- Maintain the ability to take a logical dump in a standard format and test it periodically.
- Define infrastructure as code so control plane migration is a rewrite of definitions rather than an archaeology exercise.
- Negotiate egress terms and data export assistance in the contract.
- Ask, before signing: what exactly do we get back, in what format, and how quickly, if we leave?

A provider confident in its service will answer that question directly. Reluctance to answer it is itself an answer.



Chapter 15 — Migration Approach

Chapter 15 — Migration Approach

Deciding to adopt managed DBaaS is the easy part. This chapter outlines a migration approach at the level a strategy document should carry; detailed engine-specific runbooks belong in separate documentation.

15.1 Assess

Inventory the estate: engines, versions, sizes, growth rates, extensions in use, and interdependencies. Classify each database by criticality tier with an agreed RPO and RTO. Identify blockers early — unsupported extensions, superuser dependencies, end-of-life versions, and applications with hard-coded connection assumptions. The extension audit in particular should happen in week one, not week ten.

15.2 Prioritise

Start with low-risk, high-learning workloads: development and test environments, then non-critical production. Build team confidence and platform familiarity before moving anything that pages someone. Sequence dependent applications together — migrating a database whose application still lives elsewhere introduces latency you did not plan for.

15.3 Choose a method

Backup and restore. Simplest, requires a downtime window sized to database volume. Fine for smaller and non-critical databases.

Logical replication / change data capture. Initial load followed by continuous replication until cutover. Downtime measured in minutes. The standard approach for production systems.

Dual write. Application writes to both old and new during transition. Maximum control, maximum application complexity. Use only where the other methods genuinely cannot meet the requirement.

Whichever method, the sequence is the same: schema first, then bulk data, then continuous sync, then validation, then cutover, with a documented and tested rollback path.

15.4 Validate

Row counts and checksums against source. Application functional testing against the migrated database. Performance testing under realistic load — query plans can differ across versions and configurations, and this is where surprises appear. Failover testing before go-live, not after. Backup and restore verification on the new platform.

15.5 Cut over and optimise

Cut over during a low-traffic window with rollback ready and the source retained read-only until confidence is established. Then optimise right-size instances based on observed rather than assumed load, tune parameters to the workload, review index usage, and configure alerting thresholds to your actual patterns. Migration is not finished at cutover; it is finished when the new platform is tuned and the old one is decommissioned.



Chapter 16 — Choosing The Right Database Strategy

Chapter 16 — Choosing the Right Database Strategy

There is no universal answer. There is a defensible decision process.

16.1 Choose self-managed when

- The workload requires superuser or OS-level access that no managed service grants.
- You depend on extensions, plugins, or custom builds outside every candidate provider's supported catalogue.
- Kernel, filesystem, or storage-level tuning is material to performance in a way managed parameter groups cannot express.
- A specific patch level must be held for extended periods for certification reasons.
- You have an existing, mature, adequately staffed DBA team with proven automation, and the estate is stable rather than growing.
- Existing perpetual licences and infrastructure investment have material remaining life.
- Legal or contractual constraints prevent any third party from operating the platform.
- The workload is genuinely enormous and steady, with per-unit economics that outweigh operational cost.

16.2 Choose managed DBaaS when

- DBA capacity is constrained, hard to hire, or hard to retain.
- The estate is growing faster than the team.
- Provisioning speed affects delivery velocity.
- High availability and tested disaster recovery are required but not currently reliable.
- Compliance requires demonstrable patching, encryption, audit logging, and recovery evidence.
- Multiple engines are in use, and expertise is stretched across them.
- Workload demand is variable and elasticity has value.
- Senior engineering time is better spent on differentiating work.
- Data residency and jurisdiction are requirements — where a sovereign provider is the appropriate choice.

16.3 A decision sequence

Step 1 — Can it be managed? Are there hard blockers: superuser requirement, unsupported extension, unsupported version? If yes, self-managed for this workload. Revisit at the next major version.

Step 2 — Should it be managed? Weigh operational burden, availability requirement, compliance obligation, team capacity, and growth trajectory. Most workloads that pass step one should be managed.

Step 3 — Which deployment model? Multi-tenant, virtual private, dedicated, private cloud,

or hybrid — driven by data classification, regulatory position, and performance requirement. Chapter 5.

Step 4 — Which provider? Score candidates on the twelve dimensions in Chapter 8 and Appendix B, weighted to your requirements. Filter first on hard requirements — jurisdiction, engine coverage, certification — then score the rest.

Step 5 — What is the exit? Before signing. Chapter 14.

16.4 Mixed estates are normal

Most enterprises will run both models for a long time, and that is a legitimate architecture rather than a transitional embarrassment. Legacy systems with deep customisation stay self-managed; new development goes to managed; migration happens opportunistically at version upgrades and hardware refreshes. What matters is that each decision is deliberate and documented, not inherited.



Chapter 17 — Why Enterprises Are Moving To Managed DBaaS

Chapter 17 — Why Enterprises Are Moving to Managed DBaaS

The shift is driven by several forces at once.

Automation as the operational default. Manual database administration does not scale to hundreds of instances across multiple engines. Automation is now expected and building it in-house duplicates work that platform providers have already done.

Cloud-native application patterns. Microservices multiply database instances. Architectures that once used one large, shared database now use many smaller ones, and the per-instance operational cost model that worked for five instances does not survive fifty.

Developer productivity and self-service. Development teams expect to provision an environment in minutes through an API. Where platform teams cannot deliver that, teams route around them, and shadow infrastructure is how unmanaged production databases come to exist.

AI and analytical workloads. Vector search and retrieval-augmented generation have pushed vector capability pgvector and equivalents into mainstream requirement lists. New workload patterns arrive faster than internal platform teams can support them, and managed platforms absorb that change more readily.

The security and compliance ratchet. Regulatory expectations around patch currency, encryption, auditability, and tested recovery have risen steadily and are not going to reverse. Meeting them manually across a large estate is expensive and fragile.

Open-source licensing shifts. Licence changes across parts of the database ecosystem in recent years have made "which entity supports this engine, under what licence, with what long-term commitment" a live procurement question. Managed services with clear engine support commitments reduce that uncertainty.

Skills economics. The supply of senior database engineers is not keeping pace with demand. This is a structural constraint, not a cyclical one, and it changes the build-versus-buy calculation permanently.

Sovereignty and jurisdiction. Covered in Chapter 12, and increasingly the factor that determines the shortlist rather than merely influencing it.

Chapter 18 — Common Misconceptions

Chapter 18 — Common Misconceptions

"Managed DBaaS means we lose control of our data." The customer retains full control of the data, schema, access rights, and encryption keys where BYOK is used. What transfers is control of platform operations. These are different things, and conflating them is the root of most resistance.

"Hyperscaler DBaaS is a more advanced technology than other managed DBaaS." It is the same service model, frequently the same open-source engines. Providers differ in catalogue breadth, footprint, automation maturity, jurisdiction, and support. Those differences belong in a scorecard, not a taxonomy. Chapter 7.

"Managed means we no longer need DBAs." It means DBAs stop patching and start designing. Schema, indexing, query performance, and data modelling remain the customer's responsibility permanently, and they are the higher-value half of the job.

"Managed DBaaS is always more expensive." Sometimes on infrastructure line items. Frequently not on total cost once personnel, licensing, downtime, and delay are included. Chapter 13.

"Managed DBaaS locks us in." Engine-level portability is high with standard open-source engines. Control plane and proprietary feature adoption create real but manageable lock-in, and self-managed carries its own. Design for exit at the start. Chapter 14.

"We can just lift and shift to cloud VMs and get the same benefit." That changes the infrastructure model and leaves the operating model untouched. Every operational responsibility remains yours. Chapter 1, stage 3.

"Our data is in India, so we are sovereign." Residency is necessary but not sufficient. Sovereignty depends additionally on the contracting entity, governing law, and who can access the data from where — including backups, logs, and support access paths. Chapter 12.

"The SLA guarantees we will not have downtime." An SLA defines a commitment and a remedy, and the remedy is almost always capped service credits. It is a mechanism for aligning incentives, not an insurance policy. Read the exclusions, and check whether recovery objectives are covered at all. Chapter 8.

"Managed DBaaS means we do not need to test DR." The provider operates the DR capability. Validating that your application works correctly after failover is yours, and most regulators expect evidence of periodic testing regardless of who operates the platform. Chapter 9.

Conclusion

Managed DBaaS is not defined by who delivers it.

Whether it comes from a hyperscaler, a sovereign cloud provider, a database vendor, or a managed service provider, the defining characteristic is the same: **the provider assumes responsibility for operating the database platform, and the customer retains responsibility for the data and the application.** That responsibility boundary is the category. Everything else is a capability difference.

This distinction has practical consequences. Organisations that treat provider identity as a category make architectural decisions on the wrong axis — filtering candidates before evaluating them, and mistaking brand tier for capability. Organisations that understand managed DBaaS as a single service model with many providers evaluate on the criteria that actually determine outcomes: engine coverage, control plane maturity, availability architecture, recovery capability, security posture, observability, SLA structure, support model, jurisdiction, and commercial terms.

For Indian enterprises and public sector organisations, jurisdiction has moved from a preference to a requirement. With the DPDP Act and its 2025 Rules now operational, sector directives on data storage and incident reporting in force, and procurement frameworks tied to empanelment, where data lives and who can reach it are compliance decisions with quantified financial consequences. Sovereign managed DBaaS exists to meet that requirement without giving up the operating model — the same managed service model, delivered under Indian jurisdiction with in-country operations and support.

The recommendation is straightforward. Decide first between self-managed and managed, workload by workload, on the basis of blockers and operational fit. Then choose a deployment model on the basis of data classification and regulatory position. Then evaluate providers on capability, compliance, support, performance, and commercial fit.

Evaluate the service. Not the label.

Appendix A — Questions to Understand How Managed DBaaS Is Delivered

These questions are intended to help you understand how a managed DBaaS service is designed, delivered, operated, and supported. They are a discussion and due-diligence guide, not a universal pass/fail checklist. Their relevance and weighting will vary by workload, deployment model, service tier, risk profile, and regulatory requirements. Customers should verify current capabilities, contractual commitments, and service availability directly with each provider.

Engine coverage

- Which engines and versions are supported?
- How quickly are new major versions made available after upstream release?
- How long are versions supported, and what is the end-of-life process?
- Are community editions or proprietary forks used? If forks, what is the divergence?

High availability

- Synchronous or asynchronous replication? Configurable?
- What are the failure domains — rack, zone, region?
- Is failover automatic? What is the measured failover time?
- Has failover time been demonstrated, and can we witness a test?

Backup and recovery

- Backup cadence, method, and retention range?
- Point-in-time recovery granularity and window?
- Are backups integrity-verified? How, and is evidence provided?
- Are backups encrypted, and where are they stored?
- Cross-region backup copies available?
- Estimated restore time for a database of our largest size?
- Is restore self-service?

Scaling

- Vertical scaling — downtime required?
- Read replicas — how many, what lag, how provisioned?
- Storage autoscaling available?
- Built-in connection pooling?
- Self-service or ticket-based, and what is the turnaround?

Security

- Encryption at rest — default, algorithm, key management?
- Customer-managed keys / BYOK supported? Which integrations?
- TLS enforced? Which versions and cipher policy?
- Private endpoints available? Public endpoint disabled by default?
- RBAC granularity and enterprise identity integration?
- Under what conditions can provider staff access customer data? What approval, what logging, is it visible to us?
- Vulnerability management and patch cadence? Emergency CVE process?

Observability

- Metrics exposed and retention period?
- Slow query capture and query performance history?
- Log access and export to customer SIEM?
- Alerting integrations?

SLA

- Availability commitment, and for which configuration?
- What is excluded? Is planned maintenance excluded?
- Measurement window — monthly or annual?
- Remedy structure and cap?
- Separate RPO and RTO commitments?
- Support response SLA by severity?

Support

- Tiers, inclusions, and cost?
- Response and resolution targets by severity?
- Escalation path — who is reached, and how quickly?
- Time zone and language coverage?
- Named technical account manager available? At what threshold?

Compliance and jurisdiction

- Where are data, backups, logs, and DR copies stored and processed?
- Which legal entity contracts, and under which governing law?
- Certifications held, and what is the scope statement for each?
- MeitY empanelment status if public sector procurement applies?
- Is a data processing agreement available? Does it meet DPDP processor obligations?
- Breach notification process and timelines?

- Audit rights — ours and our regulator's?
- Is non-disruptive DR drill capability available, with evidence artefacts?

Commercial

- Pricing model and predictability?
- Egress and inter-zone transfer charges?
- Committed use or reserved options?
- Billing currency?
- Contract term, notice period, and exit assistance?
- What data do we get back on exit, in what format, and how quickly?

Migration

- Which migration tools and methods are supported?
- Is migration assistance included?
- Is a proof of concept available?

APPENDIX B — GLOSSARY

Availability Zone — an isolated failure domain within a region, with independent power, cooling, and networking.

BYOK — Bring Your Own Key. Customer supplies and controls the encryption keys used by the service.

CDC — Change Data Capture. Streaming changes from a source database, commonly used in migrations.

Control plane — the provider's management and automation layer: APIs and console for lifecycle operations.

Data Fiduciary — under the DPDP Act, the entity determining the purpose and means of processing personal data. Accountable for compliance.

Data plane — the running database engine and the data within it.

Data Processor — under the DPDP Act, an entity processing personal data on behalf of a Data Fiduciary.

Data residency — the physical location where data is stored.

Data sovereignty — the legal jurisdiction governing the data.

DBaaS — Database as a Service. Umbrella term for consuming a database as a service.

DPDP Act / DPDP Rules — India's Digital Personal Data Protection Act, 2023 and the Rules notified in November 2025 operationalising it.

Failover — promotion of a standby to primary following a primary failure.

High Availability (HA) — architecture designed to continue operating through component failure.

IaaS — Infrastructure as a Service.

Managed DBaaS — a service model in which the provider operates the database platform under an SLA while the customer retains responsibility for data and application.

MeitY empanelment — certification by India's Ministry of Electronics and Information Technology, following STQC audit, enabling a cloud provider to serve government and public sector customers.

Multi-tenant — infrastructure shared across customers with logical isolation.

PaaS — Platform as a Service.

PITR — Point-in-Time Recovery. Restoring a database to a specific moment.

Read replica — a read-only copy used to scale read traffic.

RPO — Recovery Point Objective. Maximum acceptable data loss, measured in time.

RTO — Recovery Time Objective. Maximum acceptable time to restore service.

Self-managed — an operating model in which the customer owns the complete database lifecycle.

Shared responsibility model — the division of security and operational responsibility between provider and customer.

SLA — Service Level Agreement. Contractual commitment with defined remedies.

Sovereign cloud — cloud services delivered within a national jurisdiction, under that jurisdiction's law, by an entity subject to it.

STQC — Standardisation Testing and Quality Certification Directorate, the MeitY body auditing cloud providers for empanelment.

Vertical scaling — increasing the resources of an existing instance.

WAL — Write-Ahead Log. PostgreSQL's transaction log, used for replication and point-in-time recovery.

This whitepaper is published by Yotta for educational purposes. Product capabilities, certifications, and service levels should be verified against current documentation. Regulatory references are current as of publication and should be re-verified before reuse.