**What is Pact? (Brief Intro)**

“Pact is a contract-testing framework that ensures APIs between microservices work as expected.  
Each service publishes a **contract** — a JSON file describing the expected request and response — which is verified by the provider.  
This helps us catch integration issues early in CI/CD before deployment.”

📄 Example Contract (Simplified JSON)

**⚙️ 3. Path to Enable Pact for Onboarding Applications**

**Onboarding Phases:**

| **Phase** | **Key Activities** | **Key Tasks or Actions** | **Success Criteria** |
| --- | --- | --- | --- |
| **Phase 1** | Setup and configure Pact Broker for Wealth Service (see above) | - Deploy Pact Broker. - Configure database and security. | Pact Broker running and accessible; sample contract published and visible |
| **Phase 2** | Local Implementation | - Consumer: add Pact dependencies, write consumer tests, generate and publish contract manually. - Provider: add Pact dependencies, write verification tests, implement @State setup, verify locally. | Pact visible in Broker UI; Provider verifies contract successfully |
| **Phase 3** | Automate Pact workflow in GitLab CI/CD | - Modify .gitlab-ci.yml for consumer and provider pipelines to automate contract publishing and verification. - Integrate Pact Broker with CI/CD for automated testing and reporting. | Contract publishing and verification fully automated; pipeline fails on contract break |
| **Phase 4** | Multi-Environment Contract Promotion | - Integrate Pact Broker verification results into deployment pipelines to gate deployments based on contract compatibility. | Use Pact Broker's verification results to make deployment decisions, preventing incompatible versions from deploying |

**5. Infrastructure Setup Plan**

“Pact Broker requires a persistent database for storing contracts, verification results, and metadata. The open-source recommendation is PostgreSQL.”

**🔧 Infra Components**

| **Component** | **Purpose** |
| --- | --- |
| Pact Broker | Manages contracts & verifications |
| PostgreSQL (on-prem) | Backend DB for persistence |
| GitLab CI/CD | Publishes & verifies contracts |

**🧱 4. Pact Broker Options**

| **Option** | **Description** | **Pros** | **Cons** |
| --- | --- | --- | --- |
| **Pact Broker (Self-hosted)** | Open-source broker hosted on-prem or cloud | Full control, cost-effective, customizable | Infra setup & maintenance |
| **PactFlow (SaaS)** | Managed commercial service | No infra overhead, enterprise support | Paid license, limited infra control |

“We chose the **self-hosted Pact Broker** to retain control, ensure data residency, and align with internal DevSecOps policies.”

**☁️ 6. On-Prem vs AWS PostgreSQL Discussion**

Given our timeline and governance overhead for AMS onboarding, we recommend **on-prem PostgreSQL** as an interim solution.  
It allows us to move forward with Pact adoption immediately and later migrate to AWS Postgres once cloud onboarding is streamlined

**Pact Implementation Roadmap – Wealth Service**

**Phase 1 – Pact Broker Setup (Infrastructure)**

1. Prepare Kubernetes setup (namespace, secrets, configmaps).
2. Provision PostgreSQL DB *(decision: bundled Postgres vs managed Postgres)*.
3. Configure & Deploy Pact Broker (Helm/manifests, connect to DB).
4. Verify setup (pods/logs, DB connectivity, schema migration).
5. Setup Ingress/DNS (Broker accessible at pact-broker.wealthservice.company.com).

**Outcome:** Pact Broker running and accessible.

**Phase 2 – Local Implementation (Teams)**

**Consumer Side**

1. Add Pact dependencies.
2. Write consumer Pact tests & generate contracts.
3. Publish contract manually to Broker & verify in UI.

**Provider Side**  
4. Add Pact dependencies.  
5. Write provider verification tests with @State setup.  
6. Run local contract verification:

* Configure Broker connection.
* Write verification test (@Provider, @PactBroker, @State).
* Start provider locally & run ./gradlew pactVerify.
* Check logs & Broker UI for ✅.
* Fix mismatches until green.

**Outcome:** Contract visible in Broker; Provider successfully verifies locally.

**Phase 3 – CI/CD Integration (Automation)**

**Consumer Side**

1. Add Pact publishing to CI/CD pipeline (auto publish).
2. Run consumer Pact tests in pipeline.

**Provider Side**  
3. Add Pact verification stage in pipeline (./gradlew pactVerify).  
4. Ensure provider test env is available in CI/CD.  
5. Publish verification results back to Broker.

**End-to-End Flow**  
6. Gate pipeline on contract verification (fail if verification fails).

**Outcome:** Fully automated Pact lifecycle in CI/CD.

**Phase 4 – Multi-Environment Contract Promotion**

1. Define environments in Pact Broker (dev, qa, staging, production).
2. Tag published contracts with environment labels.
3. Promote contracts through lifecycle (e.g., dev → qa → staging).
4. Ensure provider verifies contract in each environment.
5. Configure CI/CD to move contracts forward only if prior env verification is ✅.

**Outcome:** Controlled promotion of contracts across environments with visibility in Broker.

**Phase 5 – Readiness**

1. Enable can-i-deploy checks in deployment pipeline.
2. Integrate Pact Broker environments & tags into gating rules.
3. Add mandatory can-i-deploy step before prod deployment.
4. Automate promotion workflow using Pact Broker verification results.
5. Define rollback strategy if deployment blocked.
6. Monitor readiness via Broker dashboard; alert on blocked deployments.

**Outcome:** Deployments to production only allowed when contracts are verified and compatible.

Thanks for joining.

I’d like to walk you through our **Pact enablement plan** for the onboarding applications. As our ecosystem grows, multiple services depend on each other, and ensuring their APIs stay compatible has become critical. Pact helps us automate this validation by enabling **consumer–provider contract testing**, where each microservice defines its expectations upfront and verifies them continuously.

In today’s discussion, I’ll briefly explain **what Pact is**, how it integrates into our existing **CI/CD pipelines**, and the **enablement plan** we’ve drafted for our applications.

I’ll then walk through the **infrastructure design** — particularly our choice of a **self-hosted Pact Broker** — and explain why we are proposing to start with an **on-prem PostgreSQL instance**. This approach allows us to move faster, maintain data within our internal network, and stay compliant with security and governance controls, while keeping a clear path open for future migration to AWS-managed Postgres once our AMS onboarding matures.

The goal today is to get approval to proceed with this setup so we can start establishing contract testing as a core part of our delivery lifecycle — improving quality, confidence, and collaboration across all our onboarding applications.”

“Thanks for joining. I’d like to walk you through our **Pact enablement plan** for the onboarding applications.

The idea is to introduce **consumer–provider contract testing** into our CI/CD workflow to improve integration reliability and reduce manual testing effort between microservices.

I’ll start by giving a brief overview of **what Pact is** and how it helps us ensure API compatibility between our systems, followed by the **enablement plan for our onboarding applications** — how Pact fits into our current pipeline and governance structure.

Then, I’ll walk through the **infrastructure setup**, the rationale behind choosing a **self-hosted Pact Broker**, and finally, why we are proposing an **on-prem PostgreSQL instance** instead of AWS Postgres at this stage.

We’ll also touch on the **expected scale**, **data retention approach**, and **deployment strategy**, ensuring the design aligns with our broader engineering and compliance standards.

The goal of today’s discussion is to get alignment and approval to proceed with the **on-prem PostgreSQL setup for the Pact Broker**, so that we can begin the rollout and start capturing integration contracts across our applications in a secure, governed, and auditable manner.”

*Pact Enablement Plan – Introduction*

* Objective: Introduce **consumer–provider contract testing** across onboarding applications
* Why Pact: Early detection of integration issues between microservices
* Scope: 10+ applications with ~20 APIs each (both consumer & provider roles)
* Plan:
  + Integrate Pact in CI/CD pipelines for automated verification
  + Centralize contract storage via **Pact Broker**
  + Retain 3 months of contract history for auditability
* Infra Proposal:
  + Start with **on-prem PostgreSQL** for Pact Broker (quick provisioning, full control, governance compliant)
  + Future-ready for AWS migration
* Goal: Approval to proceed with **on-prem Pact Broker setup** to begin enterprise rollout

Perfect 👍 Let’s connect everything — we’ll **revisit and finalize the DB estimation** part now, incorporating the new clarification:

🧩 Each of your **10 applications** is built **10–15 times per day**, and each build **publishes Pact contracts** to the **Pact Broker**.

Here’s the **updated, detailed, and presentation-ready estimation** considering your actual build frequency.

**🧮 Updated Pact Broker Database Estimation (10 Apps, 10–15 Builds/Day)**

**1. Base Parameters**

| **Parameter** | **Value** | **Notes** |
| --- | --- | --- |
| Number of Applications | 10 | Each acts as both consumer & provider |
| Contracts per consumer-provider pair | 1 | ~5 KB JSON per contract |
| Average number of provider dependencies per app | 5 | Typical for microservice architecture |
| Daily builds per app | 10–15 | CI/CD pipeline builds triggered frequently |
| Retention period | 3 months | Older contracts pruned automatically |

**2. Contract Publication Volume**

Each app consumes ~5 providers → 5 contracts per build.  
10 apps × 5 contracts = **50 contracts per build cycle**.  
At **10–15 builds per day per app**, total builds per day = 10 × 10–15 = **100–150 builds/day**.

Each build publishes 5 contracts →  
👉 **Daily contracts = 100–150 builds × 5 = 500–750 contracts/day**

**3. Storage Estimation**

| **Period** | **Contracts Stored** | **Size per Contract (5 KB)** | **With DB Overhead (~×5)** | **Total** |
| --- | --- | --- | --- | --- |
| **Daily** | 500–750 | 2.5–3.75 MB | ~12.5–18.75 MB | ~15 MB/day |
| **Monthly (30 days)** | 15,000–22,500 | 75–112.5 MB | ~375–560 MB | **~0.4–0.6 GB/month** |
| **3 Month Retention** | 45,000–67,500 | 225–337.5 MB | ~1.1–1.7 GB | **~1.5 GB total** |

🟩 **Steady-state DB size with 3-month retention:** ~1.5 GB  
🟦 Add 20–30% for indexes/logs ⇒ **~2 GB total**

**4. Yearly Growth Projection (If Not Pruned)**

| **Duration** | **Approx. Data Volume** | **Notes** |
| --- | --- | --- |
| **1 year (no pruning)** | ~6–7 GB | if 3-month pruning is disabled |
| **With pruning (rolling 3 months)** | Stable at ~2 GB | sustainable steady-state size |

**5. Recovery, Retention & Performance**

| **Aspect** | **Recommendation** | **Reason** |
| --- | --- | --- |
| **RPO (Recovery Point Objective)** | ≤ 1 hour | Contract data changes every build |
| **RTO (Recovery Time Objective)** | ≤ 1 hour | Quick restore from snapshot |
| **Backup Frequency** | Daily or post-build event | Minimal backup size |
| **Retention** | 3 months rolling | Keeps DB lean, easy to manage |
| **Expected DB Size (steady)** | 1.5–2 GB | including verification metadata |
| **Allocated Size** | 5 GB | recommended with 2–3× buffer |

**6. Infra Recommendations**

| **Component** | **Recommendation** | **Notes** |
| --- | --- | --- |
| **DB Engine** | PostgreSQL 14+ (on-prem) | Stable, open-source |
| **Storage** | 5–10 GB allocated | Scales easily |
| **Broker Server** | 2 CPU, 4 GB RAM | Sufficient for ~10 apps |
| **Backup** | Nightly snapshot + weekly full backup | Automated in CI/CD |
| **Monitoring** | Enable DB size and contract count metrics | Detect unusual growth |

**✅ Final Summary**

| **Metric** | **Estimate** |
| --- | --- |
| Applications | 10 |
| Builds per day per app | 10–15 |
| Contracts published per day | 500–750 |
| DB size (3-month retention) | ~1.5–2 GB |
| DB size (1 year, no pruning) | ~6–7 GB |
| Allocated storage | 5–10 GB |
| RPO / RTO | ≤ 1 hour |
| Backup | Daily |
| Retention | 3 months (rolling window) |

Would you like me to now create a **1-slide visual summary** (diagram + table + key takeaways) for this estimation so you can include it in your MD discussion deck?

**You said:**

yes pls

**ChatGPT said:**

Perfect 👍 Here’s how you can present this in a **single professional slide** for your MD — concise, data-backed, and visually structured for impact.

**🧮 Slide Title: Pact Broker Database Estimation & Retention Plan**

**🧩 Overview**

| **Parameter** | **Details** |
| --- | --- |
| **Applications** | 10 (each both Consumer & Provider) |
| **Build Frequency** | 10–15 builds per app per day |
| **Contracts per Build** | ~5 contracts published |
| **Total Contracts per Day** | 500–750 |
| **Average Contract Size** | 5 KB (JSON) |
| **Retention Period** | 3 months (rolling) |

**💾 Database Sizing & Growth**

| **Period** | **Contracts Stored** | **Estimated DB Size (with overhead)** |
| --- | --- | --- |
| **Daily** | 500–750 | ~15 MB |
| **Monthly (30 days)** | 15K–22.5K | ~0.4–0.6 GB |
| **3 Months (steady state)** | 45K–67.5K | **~1.5–2 GB** |
| **1 Year (no pruning)** | ~270K | ~6–7 GB |

📌 **Allocated DB Storage:** 5–10 GB (includes buffer for growth & indexes)  
📌 **Expected Steady-State Usage:** ~2 GB with 3-month retention

**⚙️ Infrastructure & Backup**

| **Component** | **Recommendation** |
| --- | --- |
| **DB Engine** | PostgreSQL 14+ (on-prem) |
| **Server Spec** | 2 vCPU, 4 GB RAM |
| **Backup Policy** | Daily incremental + weekly full backup |
| **RPO / RTO** | ≤ 1 hour |
| **Monitoring** | DB size, contract count, verification metrics |

**🧭 Key Takeaways**

* High CI/CD activity (10–15 builds/day/app) → ~500–750 new contracts daily
* Rolling **3-month retention** keeps DB size stable (~2 GB)
* **On-prem PostgreSQL** ensures data control & fast provisioning
* **5–10 GB** allocation recommended for safe scaling
* Ready for future migration to AWS RDS Postgres if needed

Would you like me to generate a **PowerPoint (.pptx)** version of this slide — with the tables, icons, and layout ready for your meeting? I can make it look boardroom-ready with your team’s context (onboarding, GS-style formatting).

Top of Form

Bottom of Form

’s continue the **DB sizing and on-prem setup estimation** for your **Pact Broker**.  
We’ll base this on your earlier note — *each application builds 10–15 times a day* — which means the number of Pact publications and verifications will be high.

**🧩 Pact Broker On-Prem Database Sizing Estimation**

**1️⃣ Inputs / Assumptions**

| **Parameter** | **Value / Assumption** | **Notes** |
| --- | --- | --- |
| Number of onboarding applications | ~20 | (Example — update with your actual count) |
| Builds per application per day | 10–15 | Each build publishes new consumer contracts |
| Consumers per provider | 1–3 | Average relationships |
| Pact JSON average size | ~100 KB | Typical Pact contract (small JSON) |
| Retention period | 6 months | Broker stores version history for traceability |
| DB Engine | PostgreSQL | Preferred open-source option |

**2️⃣ Storage Estimation**

**Daily Pact Uploads**

20 applications × 15 builds/day × 100 KB ≈ 30 MB/day

**6-Month Retention**

30 MB/day × 180 days ≈ 5.4 GB

Add indexing, metadata, provider verification data, logs (~50% overhead):

≈ 8 GB total for 6 months

You should **provision at least 20–30 GB** storage for future growth and safety margin.

**3️⃣ Performance & Scaling**

| **Component** | **Recommendation** | **Notes** |
| --- | --- | --- |
| **DB CPU/RAM** | 2 vCPU, 4–8 GB RAM | For 50–100 concurrent API calls |
| **Disk Type** | SSD storage | Faster JSON reads/writes |
| **Backup** | Daily logical dump | Retain last 7 days |
| **Archival** | Rotate old contracts monthly | Optional cleanup policy |

**4️⃣ On-Prem vs AWS PostgreSQL**

| **Aspect** | **On-Prem** | **AWS (PactFlow/Postgres RDS)** |
| --- | --- | --- |
| **Data Security** | ✅ Full control, stays within firm network | ❌ External data storage |
| **Integration** | ✅ Easy internal app integration (VPN free) | ❌ Requires network setup |
| **Maintenance** | ❌ Manual upgrades, patching | ✅ Managed by AWS |
| **Compliance** | ✅ Aligns with internal data policies | ❌ External compliance risk |
| **Cost** | ✅ One-time infra cost | 💰 Ongoing SaaS/RDS costs |

Since onboarding involves **sensitive client metadata**, on-prem PostgreSQL provides better **data control, auditability, and latency**.

**5️⃣ Infra Architecture (High-Level)**

+------------------------+

| CI/CD Pipeline |

| (publishes Pacts JSON) |

+-----------+------------+

|

v

+------------------------+

| Pact Broker API |

| (Dockerized service) |

+-----------+------------+

|

v

+------------------------+

| PostgreSQL (On-Prem DB)|

| - Stores contracts |

| - Version metadata |

+-----------+------------+

|

v

+------------------------+

| Onboarding Apps |

| (Consumer/Provider) |

+------------------------+