**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