Cloud Migration Roadmap (from Desktop WPF)

Purpose

A concise, actionable plan to evolve the current WPF 'PackageConsole' into a secure, scalable cloud application. Keep this as a living reference as you continue desktop development.

1) Current State Summary

- App: .NET 8 WPF desktop
- Data: per-user SQLite '*_packages.db' on central share; MasterDB as aggregated view
- Files: INI/templates/exports on network share
- Auth: Windows user context; 'Admin Mode' inside app
- Known constraints: UNC/WAL quirks; manual 'Refresh Master' aggregation

2) Target Cloud Architecture (two proven paths)

Option A — Python-first (fastest admin/CRUD)

- Backend: Django + Django REST Framework (DRF)
- DB: Azure Database for PostgreSQL (managed)
- Storage: Azure Blob Storage (INI, exports, tooltips.json)
- Auth: Azure AD (Entra ID) SSO via django-allauth/python-social-auth
- Admin: Django Admin out-of-the-box for quick admin UI
- Pros: Very fast MVP, batteries-included admin and filters

Option B — Modern SPA + API

- Backend: FastAPI + SQLAlchemy 2.0 + Alembic + Pydantic
- Frontend: Next.js (React) with server-side rendering
- DB/Storage/Auth: Azure Postgres, Blob, Azure AD
- Pros: Best UX, clean API separation, scalable frontend

Alternative (.NET) — if you stay all-in .NET

- Backend: ASP.NET Core 8 Web API; Frontend: Blazor or Next.js
- DB: Azure SQL or Postgres; Auth: Azure AD

3) Phased Roadmap

Phase 0 — Decisions and Setup

- Choose stack: Django (fastest) or FastAPI+Next.js (best UX); confirm Azure
- Provision dev Azure resources: Postgres, Blob, App Registration
- Define environments: Dev, Test, Prod

Phase 1 — Domain & Schema

- Model: packages; optional submissions/history; tooltips; feedback
- Create migrations; add indexes (AppKeyID, SubmittedBy/On, AppName)

Phase 2 — API Foundation

- Endpoints: list/filter packages, details, tooltips get/update, export CSV
- AuthN/Z: Azure AD (OIDC), Admin/User roles
- Observability: structured logs

Phase 3 — Data Migration Tool

- Enumerate central '*_packages.db' SQLite files
- Read PackageMetadata; apply merge rule (latest by SubmittedOn per AppKeyID)
- Bulk insert into Postgres; validate counts vs desktop grid

Cloud Migration Roadmap (from Desktop WPF)

Phase 4 — UI MVP

- Django: start with Admin + thin list page; or DRF + small template
- FastAPI path: Next.js grid with filters, paging, sorting, export
- Use DB-driven distinct SubmittedBy list

Phase 5 — Replace Master Aggregation

- Use direct DB queries/views for 'Master' (no merge job)
- If precompute needed: materialized view or scheduled worker

Phase 6 — Files, Tooltips, Exports

- Store tooltips/templates in Blob; add caching
- Generate CSV on-demand; optionally persist to Blob via SAS

Phase 7 — CI/CD, Ops, Hardening

- Dockerize; GitHub Actions to build/test/deploy; run migrations
- App Insights, Key Vault, per-env configuration

4) Data Model Migration Notes

- Decide: single 'packages' vs packages + submissions for history
- Normalize lookups (vendors, installer types) if helpful
- De-duplicate via AppKeyID + latest SubmittedOn, or keep all and compute 'current' via a view

5) Authentication & Authorization

- Azure AD (Entra ID) OpenID Connect
- Roles: Admin (write/tooltips), User (read/export)
- Group-based role mapping in Azure AD

6) Storage and Files

- Blob containers: config (tooltips.json, templates), exports
- Access via SAS or managed identity; cache tooltips in API

7) Background Jobs

- Prefer query-time computation; add jobs only if needed
- If needed: Celery + Redis (Python) or Azure Functions Timer

8) CI/CD and Environments

- GitHub Actions: lint/tests, build, push, deploy; run DB migrations
- Dev/Test/Prod resource separation; infra as code later

9) Risks and Mitigations

- Data shape & duplicates: define merge rules; validate with reports
- Auth complexity: test Azure AD early with a sample app
- Performance: server-side paging, selective columns, proper indexes
- Cost control: start small on Postgres and scale

10) Hybrid Bridge (optional)

- Build API first; point current WPF grid to API for reads
- Validate data and filters before web UI cutover

11) Success Criteria (MVP)

- Users: filter/view packages in a browser with paging
- Export CSV from server; Admins edit tooltips/config

Cloud Migration Roadmap (from Desktop WPF)

- Data parity with desktop; central share retired
- One-click deploy; monitoring & error logs available

12) Next Steps

- Pick stack: Django (fastest) or FastAPI+Next.js (best UX)
- Confirm Azure AD tenant and Admin groups
- Provide access to central '*_packages.db' for migration script