

Reference Architecture Overview - Neal Anders

***** Based on time constraints the bulk of this document has been AI generated and should not be used verbatim without further review. *****

Reference Architecture Overview

Goal: Migrate a desktop-based .NET application to a cloud-native architecture on Azure, introducing Python microservices for modular, scalable functionality.

1. Application Layer (Cloud-Hosted)

- .NET Components
- Reuse existing .NET logic that's core to business workflows.
- Refactor into containerized .NET Web APIs or .NET 8 minimal APIs.
- Deploy to Azure Container Apps or AKS (Azure Kubernetes Service).
- Python Microservices
- Build new or replacement functionality as containerized Python FastAPI/Flask services.
- Focus on compute-intensive or data science workloads, async processing, or areas where Python's ecosystem shines.
- Use shared message formats (JSON/gRPC schemas).
- API Gateway
- Use Azure API Management or Azure Front Door to route traffic and manage versioning, throttling, and security across .NET and Python services.

2. Data & Integration Layer

- Azure SQL Database for structured transactional data.
- Azure Blob Storage for documents, reports, or binary assets.
- Cosmos DB if low-latency global access or NoSQL is needed.
- Azure Service Bus or Event Grid for interservice communication and event-driven workflows.
- Azure Functions for lightweight event handlers and glue logic between services.

3. Security & Identity

- Azure Active Directory (AAD) for authentication and single sign-on.
- Azure Key Vault for secrets, certificates, and connection strings.
- Private endpoints and VNET integration for service isolation.

4. DevOps & Observability

- GitHub Actions or Azure DevOps Pipelines for CI/CD.
- Build → Test → Containerize (.NET/Python) → Push to Azure Container Registry (ACR) → Deploy to AKS or Container Apps.
- Application Insights and Log Analytics for metrics, tracing, and alerting.
- Use Feature Flags for incremental feature enablement.

5. Client & User Interaction

Option 1: Create a web front-end (React, Blazor, or Angular) connecting via APIs.

Option 2: Maintain a thin desktop client that calls APIs for all data/compute operations.

Option 3: Transitional hybrid model — existing client UI, backend logic progressively replaced by cloud endpoints.

Three-Phase Migration Timeline

Phase	Focus	Duration (Estimated)	Key Deliverables
Phase 1: Discovery & Foundation	Assessment, architecture design, and environment setup	4–6 weeks	<ul style="list-style-type: none">- Application inventory and dependency map- Cloud architecture design & roadmap- Azure subscriptions, networks, ACR, DevOps pipelines- Proof-of-concept containerization for one .NET component
Phase 2: Pilot & Strangler Migration	Incrementally introduce cloud services, starting with least-coupled modules	8–12 weeks	<ul style="list-style-type: none">- Deploy API Gateway & first Python microservice- Containerize main .NET service(s)- Implement CI/CD pipelines & monitoring- Integrate AAD & Key Vault- Data sync mechanism between local and cloud databases

Phase 3: Full Cutover & Optimization	Transition fully to cloud, deprecate desktop logic	8–10 weeks	<ul style="list-style-type: none"> - All business logic now cloud-hosted - Desktop client refactored to thin client or web UI - Load/performance tests passed - Cost, performance, and security reviews - Operational runbooks, rollback plan, post-mortem lessons learned
--------------------------------------	--	------------	---

Optional Phase 4: Continuous Improvement

- Introduce autoscaling, global distribution, and CI-driven environment promotion (Dev → Staging → Prod).
- Regular architecture reviews to optimize for cost and maintainability.
- Introduce data analytics, AI, or ML features (Python) once stable.

Key Success Factors

- Start with bounded contexts (modular decomposition before rewrite).
- Avoid big-bang migrations — use the strangler façade pattern.
- Establish strong DevOps and observability early — they'll make scaling and debugging easier later.
- Keep the team aligned around business outcomes, not just tech deliverables.



