Microsoft

Create an Agent MVP in 30 Days

A Microsoft Foundry Developer Checklist

Is your agent development process bogged down in ambiguity, second-guessing, and redundant efforts?

This checklist is designed to help teams build a minimum viable product (MVP) agent in approximately 30 days. The steps are organized by the five pillars of the Microsoft Well-Architected Framework.

Security | Reliability | Performance Efficiency | Cost Optimization | Operational Excellence

Security

- **Embed content safety and privacy**
  - Integrate content safety filters at every stage of development. Remove unnecessary personal or confidential data from storage and logs to help support privacy requirements. This approach can help reduce the risk of exposing sensitive information and support user confidence.
- **Enable end-to-end encryption**
  - Encrypt data at rest with platform-level encryption, such as Azure-managed or customer-managed keys, for all databases, storage accounts, and artifacts.
  - Enforce HTTPS for all data in transit to protect data comprehensively in your MVP agent environment.
- **Adopt key-less authentication (Entra ID)**
  - Eliminate static API keys from your architecture. Use Microsoft Entra ID for authenticating calls to Foundry Tools and your own APIs.
  - This modern approach to managed identity simplifies security and compliance, removing the risks of leaked keys while providing fine-grained access control.
- **Implement role-based access control**
  - Grant permissions using Azure Role-Based Access Control (RBAC), so each team member and service has the permission they need. Leverage Foundry's built-in roles (User, Project Manager, Account Owner) for project and resource access.
  - Proper RBAC supports secure, scalable collaboration and is designed to align with enterprise compliance expectations.

Reliability

- **Leverage managed HA infrastructure**
  - For the MVP, deploy critical components on redundant instances to avoid a single point of failure. Consider deploying multiple agent service instances behind a load balancer to improve resilience.
- **Plan graceful failure and fallback**
  - Design the agent to fail gracefully—providing fallback responses—if an AI model or data source is unavailable.
  - For instance, keep a previous model version or a simplified rules-based response as a fallback when the latest model deployment encounters issues. That way, the agent still provides output instead of crashing.
- **Keep the architecture stateless**
  - Avoid local session or instance-specific data and build the MVP agent to be stateless, to enable easy scaling and recovery.
  - A stateless design means that any instance can be replaced or duplicated without impacting user experience, enabling seamless scale-out and future failover.
- **Set up health checks and recovery**
  - Implement basic health monitoring for your agent service (e.g., a heartbeat endpoint).
  - Configure the platform (App Service, Azure Kubernetes Service [AKS], etc.) to automatically restart or fail over if a health check fails.
  - Having automated health checks and self-healing policies in place can support quicker recovery from crashes, even during the MVP phase.

Performance Efficiency

- **Right-size compute resources**
  - Don't over-provision for the MVP. Start with the smallest setup that meets your latency requirements.
  - Select the appropriate Azure resources for your workload to meet performance targets (keep the link). Use GPU-powered VMs only when required for heavy model tasks, and opt for CPU-optimized instances for lighter inference to balance speed and cost. To support effective architecture planning, use App Advisor during the design phase to help right-size your solution.
- **Optimize for speed**
  - Evaluate fine-tuning options using the Foundry recommendations outlined here: Foundry fine-tuning considerations.
  - Assess whether Small Language Models (SLMs) meet your use case requirements by reviewing the guidance provided here: What Are Small Language Models (SLMs)?
- **Implement caching**
  - Introduce caching to optimize performance and reduce unnecessary processing.
  - For example, implement Azure Cache for Redis to temporarily store frequently accessed data—such as common responses or embeddings—and reuse it when appropriate. This approach can significantly reduce repeated calls to the model or external services, lower operational overhead, and improve response times for end users.
- **Enable automatic scaling rules**
  - Configure automatic scaling so the agent can handle load spikes without manual intervention.
  - Using Azure Container Apps or AKS, set up auto-scale triggers (e.g., CPU or response-time thresholds) to spin up additional instances under high load. This ensures consistent performance as usage grows, even during the MVP trial runs.

Cost Optimization

- **Analyze cost drivers early**
  - Identify key cost factors for your AI agent (data volume, number of queries, latency requirements, and any third-party API costs) and create a simple cost model.
  - Set an initial budget for the 30-day MVP and track against it so you have clear visibility into how each feature or usage pattern impacts Azure spend.
- **Use cost-efficient services**
  - Choose technology options that meet requirements efficiently to secure the best rates from providers. For example, run your MVP on consumption-based plans or lower-tier instances.
  - If you have batch training jobs, consider spot instances (preemptible VMs) to save money. Match resources to intended use — don't pay for an always-on high-end GPU if your agent's workload doesn't continuously require it.
- **Minimize idle resource waste**
  - Monitor Azure usage and shut down any unneeded processes. Set spending guardrails! Use Foundry and Azure Monitor tools to watch utilization and set alerts.
  - Consider pausing or scaling down dev/test virtual machines (VMs) or orchestration jobs during off-hours to help manage costs. This approach may prevent cloud costs from piling up when the agent is idle.
- **Automate for lower Total Cost of Ownership (TCO)**
  - Leverage automation and best practices to reduce ongoing costs. For instance, automate deployments and testing to cut manual effort, and use auto-scaling to avoid over-provisioning.
  - These optimizations can help save time and contribute to a lower total cost of ownership as your agent moves from MVP to production.

Operational Excellence

- **Embrace DevOps and agile iteration**
  - Work in rapid, iterative cycles rather than a long waterfall. From the start, involve developers, IT ops, and data scientists in planning so that requirements are clear.
  - Aim to deliver a basic functional agent in a short sprint, then improve it. This DevOps/MLOps mindset with early cross-team collaboration can help you catch mismatches early and continuously learn and adjust.
- **Automate CI/CD pipeline with GitHub Actions**
  - Set up a continuous integration/continuous deployment (CI/CD) pipeline using Foundry's integration with GitHub Actions. Automate builds, tests, and deployments so that every code change is quickly validated.
  - This reduces human error, provides quick feedback on what works, and gets new features or fixes into the agent faster – all critical in a 30-day MVP timeline.
- **Consider using PaaS and monitor from day 1**
  - Evaluate platform services (Azure OpenAI, Functions) to avoid server management, and enable logging plus Azure Monitor/App Insights from day one for proactive error and performance alerts.
- **Practice safe deployments and feedback**
  - Even in the MVP stage, deploy changes in a controlled way. Implement basic safe deployment practices. For instance, test new model versions with a small set of queries or internal users before rolling out to everyone, and have a rollback plan.
  - Continuously collect feedback after each update and use it to inform the next iteration. This approach ensures you improve the agent steadily without major setbacks or regressions.