Microsoft

Microsoft Marketplace Monetization Checklist for SaaS and Container Offers

This resource brings together two practical checklists to help software development companies successfully launch and monetize AI solutions, whether software as a service (SaaS) or container-based, on Microsoft Marketplace. Inside, you'll find actionable steps organized by the five pillars of the Azure Well-Architecture Framework, with guidance tailored for both solution types.

Each section is designed to help your team:

- Build trust with enterprise customers through robust security and compliance.
- Ensure reliability and scalability for seamless customer adoption.
- Optimize performance and cost to drive profitability.
- Operationalize your offer for ongoing growth, support, and marketplace success.

By following these checklists, software development companies can confidently move from idea to revenue. Get ready to unlock global reach, streamline sales, and engage new customers through the power of Microsoft Marketplace.

Learn more about building AI agent solutions with the Microsoft AI Envisioning Day video series

Watch now

Optimize your offer listing with compelling value propositions, high-quality visuals, and clear deployment instructions to boost customer engagement and conversion rates.

Learn how

Marketplace monetization checklist for SaaS offers

Building a successful **AI Agent SaaS offer** on Marketplace requires aligning your technical architecture with a clear monetization strategy and packaging that resonates with customers. This checklist is grouped by the five pillars of the Microsoft Well-Architected Framework, focusing on business and IT decision makers' needs for revenue and marketplace growth. It ensures your SaaS solution is technically sound and optimized for **pricing, tiering, and marketplace positioning.**

Security

- **Address privacy & compliance**
  - Identify the applicable legal, regulatory, industry and compliance objectives you must meet.
  - Ensure data residency and privacy policies meet enterprise standards.
- **Implement Microsoft Entra ID**
  - Enable secure single sign-on and automated user provisioning for transactable SaaS offers in Microsoft Entra.
- **Enforce secure licensing**
  - Protect revenue by ensuring only authorized users or tenants access paid features.
  - Integrate license checks in your app, or use Entra ID group claims/license assignments, to prevent usage beyond entitlements.
  - Securely call Marketplace APIs to validate subscription status on login/ report usage.
- **Brand & secure the authorization experience**
  - Apply company branding to Entra ID login screens to present a professional, trustworthy image.
  - Enable TLS/SSL across all endpoints and store secrets in Azure Key Vault for payment and usage endpoints.

Reliability

- **Thorough subscription testing**
  - Rigorously test the end-to-end Marketplace purchase and activation flow in a staging environment.
  - Simulate subscribe, upgrade, renew, and cancel scenarios using Preview Audience links.
- **Subscription lifecycle resilience**
  - Design for fault tolerance in commerce flows. Queue or retry Resolve and Activate API calls to ensure customer subscriptions are provisioned despite Marketplace delays.
  - Implement graceful job provisioning on unsubscribe to prevent "zombie" users from consuming services without payment.
- **Scale to honor paid usage**
  - Ensure your SaaS scales with customer adoption.
  - Implement backpressure and task queues for AI workloads to ensure high paying customers receive promised throughput without timeouts.
- **Usage metering accuracy**
  - Build robust monitoring for usage events to support usage-based revenue.
  - Duplicate critical metering information by logging usage to a database and sending it to the Marketplace metering API.
  - If an outage occurs, have a mechanism to reconcile missed usage data once recovered.
- **Dependable support & SLAs**
  - Publish a clear SLA outlining uptime and response targets, and implement processes to help meet those commitments.
  - Minimize downtime by using Azure's high availability features such as Availability Zones and multi-instance deployment.

Cost optimization

- **Define revenue models & pricing early**
  - Align architecture and cost with your chosen pricing model. Choose subscription, usage, or outcome-based pricing during solution design.
  - Outcome-based (say, $1 per report) pricing might mean sporadic spikes in usage, requiring scalable architecture but allowing you to pass cloud costs directly to customers.
  - Subscription models mean fixed income, so control costs to protect margin. Locking strategy early ensures you align engineering effort and Azure service choices with monetization.
- **Model your Cost of Goods Sold vs price**
  - Project costs and revenue using the Azure Pricing Calculator and Marketplace Value Calculator.
  - Verify that included usage for each plan tier does not exceed price.
- **Package tiers for upsell & MACC**
  - Offer packaging tiers that serve different customer sizes while also encouraging upgrades. For instance, a Basic plan with limited AI calls pushes heavy users to upgrade to Pro rather than pay overage.
  - Also consider a custom Enterprise plan for large customers, possibly aligning with Microsoft Azure Consumption Commitment (MACC), so customers can use committed spend.
- **Leverage Microsoft incentives**
  - Enroll in ISV Success and Marketplace Rewards incentives.
  - Eligible programs may provide benefits such as Azure credits, reduced marketplace fees, and co-marketing support as you scale.
  - These programs may help reduce customer acquisition costs.
- **Optimize trial to paid conversion**
  - Use cost-effective means to offer free trials or freemium without jeopardizing revenue.
  - Consider offering a time-limited trial (e.g., 14 days) with usage controls to manage costs, in accordance with Marketplace capabilities and your business objectives.
  - Monitor trial usage and require a credit card or restrict high-cost features if usage is excessive.

Performance efficiency

- **Align architecture with pricing model**
  - Design your solution to handle the usage pattern you charge for efficiently.
  - For example, if you sell per-user subscriptions, ensure the app scales for concurrent users (horizontal scaling on App Service or AKS).
  - If you charge per AI request, optimize the AI pipeline (caching results, batch processing) so you can serve more requests per second at lower cost.
- **Tiered service levels**
  - Implement resource isolation or QoS for different plan tiers: "Custom" customers might share a compute pool, while "Enterprise" customers get a dedicated instance or higher-priority thread pool.
  - This prevents lower-tier customers from affecting higher-tier customers' available resources.
- **Optimize cost of goods sold (COGS)**
  - Use Azure resources efficiently to keep COGS low. Choose right-sized SKUs and auto-scale rules so you're not over-provisioned for low usage periods.
  - For example, run your AI inference on serverless or schedule scaling around business hours if applicable.
- **Optimize costs with reserved instances & capacity**
  - Where appropriate, commit to 1- or 3- year reservations for Azure VMs, Azure Kubernetes Service (AKS) node pools, and Azure OpenAI Service to lock in discounted rates.
  - Reserved capacity ensures predictable pricing and resource availability for long-term workloads, reducing total cost of ownership while maintaining performance.
- **Monitor & tune usage patterns**
  - Continuously review Azure Monitor metrics and Application Insights telemetry to identify which features or times drive peak loads.
  - Use this data to refine your packaging. For example, if one feature is very resource-intensive, consider making it available in higher tiers only.

Operational excellence

- **Choose Marketplace offer type**
  - Decide whether to publish as a transactable SaaS offer or use a contact-me listing.
  - For monetization, transactable SaaS is preferred. Microsoft handles billing, and you get a broader reach and MACC eligibility.
  - Only use "Contact Me" if your sales process absolutely requires custom negotiation from day one.
- **Configure plans & pricing in Partner Center**
  - Set up your plans in Partner Center reflecting your monetization strategy. For SaaS, pick Flat rate vs Per user, then define regional pricing. Make sure your tiered plans have clear names like Starter, Pro, and Enterprise.
  - Utilize Private plans for special deals (target specific tenant IDs for enterprise discounts). Double-check all pricing and terms before publishing.
- **Implement fulfillment & metering APIs**
  - Integrate the SaaS Fulfillment API for automated subscription management. Resolve purchase tokens, Activate subscriptions, and handle Renewals or Cancel events reliably.
  - If using usage-based billing, connect to the Marketplace Metering API to report usage (e.g., number of AI tasks) in near-real-time so that Microsoft can bill appropriately.
- **Set up trial and upgrade paths**
  - In Partner Center, enable a free trial or a free tier plan if it's part of your growth strategy.
  - Ensure your app clearly guides trial users to convert (in-app notifications as trial expiry nears, seamless upgrade button that triggers a plan change via Marketplace).
  - Test the plan change flow thoroughly: when a user upgrades from "Basic" to "Pro," your backend should immediately unlock Pro features, and the billing should switch at the next cycle.
- **Plan for support & lead management**
  - Marketplace will generate customer leads for you (contact information of users who tried or purchased). Set up a process to ingest these customer leads into a CRM and follow up quickly.
  - Maintain a good support system: list two support methods (email, phone, etc.) on your offer. Providing responsive support during trials may help encourage conversions.
- **Leverage co-sell and Marketplace programs**
  - Once your SaaS offer is live and you have initial customers, engage Microsoft's co-sell program.
  - Ensure your listing is marked "IP Co-sell Incentivized" by meeting requirements (transactable, at least one Azure customer, etc.). This motivates Microsoft's sales teams to bring you into deals as they get quota credit.
  - Co-sell support may exponentially increase your sales pipeline at low cost, so operationalize it: have datasheets and a demo ready for Microsoft sellers to use.
- **Ongoing monitoring & optimization**
  - Post-launch, continuously monitor Marketplace analytics. Discover which plans are selling best, where drop-offs happen in the acquisition funnel, and the usage patterns of active customers.
  - Use Azure Monitor to track tenant-level usage and detect if any customer is hitting limits frequently—that's an upsell opportunity.
- **Regular update & compliance checks**
  - Treat the Marketplace offer as a living artifact. Update your offer listing frequently, with new screenshots and updated descriptions of the latest AI features or ROI stats.
  - Regularly review your privacy policy and terms are up to date. Marketplace certification will check them, so keeping them current avoids delays when publishing updates.
  - Stay compliant with any changes; for example, if Microsoft updates transactable offer requirements or fee structure, adapt quickly.

Marketplace monetization checklist for container offers

For AI solutions delivered as **Containerized Kubernetes Applications** via Microsoft Marketplace, monetization involves a slightly different path. You'll be selling a managed AKS deployment that runs in the customer's Azure subscription, with billing through Marketplace.

This checklist (grouped by Well-Architected pillars) helps ensure your container-based AI agent is packaged, priced, and deployed in a way that drives revenue and customer adoption.

Security

- **Container image security & trust**
  - Enterprise deployment of your container often depends on clear evidence of robust security and compliance practices.
  - Scan your container images for vulnerabilities and follow Microsoft's security baseline. No root access and minimal OS libraries.
  - Sign your images and publish a secure supply chain story to build trust in customer organizations.
- **Cluster permission model**
  - Design your Kubernetes manifest with least-privilege in mind.
  - Don't require cluster-admin and use Namespaces and RBAC.
  - Many customers will review this before purchase.
  - A security-compliant deployment may help accelerate deployment timelines.
- **Entra ID integration**
  - If your containerized app has a UI or API that users log into, integrate Entra ID auth just like a SaaS (multi-tenant or customer's tenant).
  - For purely backend agents, consider supporting managed identities or key Vault for any credentials. Showing you align with Azure security best practices can assure cautious customers that it's safe to deploy.
- **BYOL license handling**
  - If offering a Bring-Your-Own-License option, ensure you have a secure license activation inside the app. Only paying customers should get full functionality.
  - For example, require a license key for BYOL deployments and secure the validation.

Reliability

- **Deployment resilience**
  - Verify that your Helm chart/ARM template for the container offer deploys reliably on various cluster sizes and versions.
  - Test on AKS of different sizes, and handle insufficient quota gracefully. document required VM SKU or cores.
  - Azure Marketplace Container offers allow a test drive in a sandbox; use that to iron out kinks.
- **License enforcement on failures**
  - If using usage meters or periodic billing, ensure the agent handles it gracefully.
  - It should cache usage and retry, rather than disabling functionality. Reliability in usage reporting protects your revenue stream under adverse conditions.
- **Scalable architecture on cluster**
  - Design your Kubernetes app to scale within the customer's cluster so that it can handle the load as they increase usage.
  - Utilize HPA (Horizontal Pod Autoscaler) if appropriate, or let the customer know how to best scale your solution.
- **Transactional consistency**
  - If your container solution uses custom metrics for billing via the Marketplace Metering service, ensure exactly-once or idempotent reporting to avoid over- or under-charging.
  - For example, include a unique usage event ID if reporting via API so duplicates aren't counted.
- **Backup & recovery for state**
  - If offering a Bring-Your-Own-License option, ensure you have a secure license activation inside the app. Only paying customers should get full functionality.
  - For example, require a license key for BYOL deployments and secure the validation.

Cost optimization

- **Select the right offer billing model**
  - Azure Container offers can be transactable or BYOL. For monetization, plan on a transactable offer unless you have an existing license model you must honor.
  - Within transactable, choose a pricing dimension that maps to value: for example, an AI inference agent might charge per core-hour or per execution.
  - It's best to start with a straightforward metric (per core or per node) that aligns with cluster size growth. You can evolve to more refined custom metrics as you gather usage data.
- **Competitive pricing & packaging**
  - Research comparable solutions or DIY costs. Customers often compare running your agent vs. hiring staff or using another tool.
  - Price your container offer to be clearly compelling. For example, "estimated $500/month for typical use" should come in lower than the manual alternative's cost.
  - Consider offering a free trial period (you can publish an initial 30-day free plan or BYOL trial plan) to reduce adoption barriers, then convert to paid.
- **Core vs. custom meter trade-off**
  - Predefined meters like cores, nodes, and pods are easier to implement in Azure auto-reports, but might not directly reflect business value.
  - Custom meters let you charge per outcome, like per document processed, aligning cost with value.
  - However, custom metering requires you to instrument and report usage, introducing complexity. For launch, decide if the added complexity is worth it.
  - Many start with a core-based model and only later introduce outcome-based pricing if it gives a market edge.
- **Azure cost transparency**
  - Be mindful that customers will pay Azure infrastructure costs in addition to your software fee.
  - Optimize your solution's resource consumption to keep its overall cost reasonable.
  - For instance, if your agent processes data, allow customers to choose a lower-cost Azure SKU or scale down when idle.
  - In your offer description, provide transparency to help customers estimate costs.
- **Plan for MACC eligibility**
  - If your container offering is a Microsoft Azure Consumption Commitment-eligible, any spending on your solution counts toward the customer's MACC commitment. This may be a strong incentive for them to choose your solution over a non-MACC alternative."
  - Highlight "MACC eligible" in conversations. Operationally, ensure your offer is listed as such.

Performance efficiency

- **Choose the right billing metric**
  - Azure Container offers support various pricing meters (per core, per node, per pod, etc.). Select the meter that best aligns with how your app uses resources.
  - Example: if your AI agent uses significant CPU, per core or per node pricing ensures revenue scales with usage. If it's mostly one-pod dying work regardless of cluster size, per-pod or per-cluster instance might fit.
- **Optimize container resource usage**
  - Keep your container resource requests modest because many pricing models charge per core or node allocated.
  - By being efficient, you allow smaller customers to deploy on minimal clusters, expanding your addressable market.
- **Performance tiers via sizing**
  - Consider offering multiple deployment profiles to match packaging tiers.
  - For instance, a "Standard" edition of your container might scale to 2 pods and handle moderate data, while an "Enterprise" edition can scale to 10+ pods and handle large loads.
  - Configure this inside the Helm chart, which larger customers can adjust. This encourages customers to self-select into higher tiers as their performance needs grow.
- **Test multi-tenant efficiency**
  - If your containerized agent will be installed per-customer, ensure that running many instances across different customer clusters doesn't degrade Azure or your service calls.
  - Efficient algorithms and use of cloud-scale services ensure you can support growth in number of deployments without performance bottlenecks hitting your own backend.
- **Geo-distribution strategy**
  - If your solution involves data or users across regions, support deploying the container app in the customer's preferred region(s).
  - Performance efficiency includes minimizing latency for end-users. A snappy solution in their region is more likely to be adopted enterprise-wide.
  - Plan your container images and manifests to be region agnostic, and utilize Azure Arc to reach on-prem clusters.

Operational excellence

- **Prepare Kubernetes assets for Marketplace**
  - Package your solution following Microsoft's guidelines: create an Azure Application (CNAB/Helm chart) that defines how your container deploys on AKS.
  - Include health probes, proper resource limits, and an easy configuration. Consider exposing parameters for keys or small customizations.
- **Configure plans in Partner Center (Container)**
  - In Partner Center, define Container offer plans similar to SaaS; you might have a "Standard" and "Enterprise" plan with different pricing or deployment scales.
  - Use Plan visibility: Public plans for general availability, and Private plans for special deals or pilot customers. For instance, you could privately offer a discounted plan to a design partner.
  - Also, configure supported regions and any prerequisites in the plan description, to ensure customers pick the right option and ultimately convert.
- **Implement metering & telemetry**
  - Unlike SaaS, container offers often require you to report consumption.
  - Use the Marketplace Metering SDK or API to emit usage events for custom metrics.
  - Instrument your app with telemetry: have it log usage details to App Insights or Log Analytics so you can verify usage patterns and correlate with billing.
  - Set up alerts if usage reporting fails or if usage is unexpectedly low or high. This could indicate an issue or an upsell opportunity.
- **Guide customer deployment sizing**
  - Provide documentation or an onboarding script to help customers size their AKS cluster appropriately for your solution.
  - This falls into operational support, but it directly ties to monetization: if customers deploy too small and have bad performance, they may churn; if they deploy too large and pay too much, they may blame your solution for being costly.
- **Enable private offers & trials**
  - Similar to SaaS, you can leverage private offers & trials, like a custom price for a large client who will deploy five instances globally.
  - Set this up by creating a private plan in Partner Center and sharing only with that client's tenant. Operationally, be prepared to support these custom deals with dedicated TAM and deployment settings.
  - Also consider a trial plan. While containers don't have a built-in free trial toggle, you can publish a plan at $0 with an expiration or issue a promo code via private offer.
- **Continuous update & marketplace compliance**
  - Keep your container offer up-to-date. Azure and Kubernetes evolve; regularly test your solution on the latest AKS versions and update your Helm charts if needed (e.g., API deprecations).
  - Update your offer listing to reflect new features or Azure integration as you add them.
  - Monitor the Marketplace certification policies for containers as well. Staying compliant ensures your offer remains live and transactable; any lapse could pause new sales.
- **Customer success & support plan**
  - Although the solution runs in the customer's environment, provide strong customer success touchpoints.
  - Proactively reach out 1-2 weeks post-deployment to ask if they need help or tuning. You can even offer a free review of their configuration. This is operational overhead, but it pays back in higher retention and expansion.
  - Make it easy for customers to get help. Clearly document how to open support tickets, and ensure your support contacts in Partner Center are updated and monitored.
- **Leverage ecosystem programs**
  - Use App Advisor to help ensure your offer meets best practices. You can also leverage Microsoft offers that are built for software development companies.