# **Azure Marketplace Offer Types & Pricing Models – Internal Guide**

Azure’s commercial marketplace supports multiple **offer types** – each with distinct pricing models and billing options. Choosing the right offer type and pricing model is critical to align with your solution’s delivery and revenue model. This guide summarizes how pricing is structured for the main offer types (**SaaS**, **Virtual Machine**, **Azure Container**, **Azure Managed Application**, and **Solution Template**), and provides best practices and constraints for each.

## **Offer Types Overview**

The table below compares key pricing model capabilities across the main Azure Marketplace offer types:

|  |  |  |  |
| --- | --- | --- | --- |
| **Offer Type** | **Supported Pricing Models** | **Billing Terms & Payment Options** | **Custom Metering Support** |
| **SaaS (Software as a Service)** | - Flat subscription fee - Per-user (seat-based) - Usage-based (metered) | Monthly or annual subscriptions; one-time upfront or recurring billing. **Supports 1-month to 5-year subscription terms** (flexible multi-year contracts). | **Yes** – supports custom usage meters via the Marketplace Metering API (pay-as-you-go charges). |
| **Virtual Machine (VM)** | - Usage-based (pay-as-you-go hourly) - BYOL (Bring Your Own License) | Hourly usage billing (billed monthly) for software fees on top of Azure infrastructure costs. Optionally 1-year or 3-year upfront billing for software (reservation). | **No** – cannot report custom usage metrics beyond the default VM runtime (no per-user or other custom meters). |
| **Azure Container** (AKS app) | - Usage-based (e.g. per vCPU-hour or custom metric) - Fixed monthly fee (subscription) | Month-to-month billing. Supports recurring monthly charges or purely usage-based billing. **No built-in multi-year terms** (annual/prepaid options not supported out-of-box). | **Yes** – supports usage metering (partners can emit custom usage events, e.g. API calls). *No native GPU-hour meter support* (GPU usage must be metered via custom metrics). |
| **Azure Managed Application** | - Flat rate (fixed management fee) - Usage-based (metered management fee) | Monthly billing (subscription) for the publisher’s managed services. **No multi-year term** in public plans (longer terms can be negotiated via private offers if needed). | **Yes** – can meter usage for management tasks (e.g. per resource managed). *Cannot meter underlying Azure resource usage* (those are billed separately to the customer). |
| **Solution Template** | - *Not directly priced* (non-transactable offer) | N/A – no direct price (essentially free from marketplace billing perspective). Deploys Azure resources into the customer’s subscription (customer pays for those at standard rates). | **N/A** – no marketplace metering (any usage charges come from the Azure resources deployed, such as VMs, not from the offer itself). |

**Note:** In the table above, “Custom Metering Support” refers to the ability for the publisher to define custom usage charges via the marketplace (using the metering API).

Below, each offer type is described in detail with its pricing models, billing considerations, and best practices/constraints.

## **SaaS (Software as a Service) Offers**

In a **SaaS** offer, the software is delivered outside of the customer’s Azure subscription (typically running in the publisher’s cloud environment). This isolation allows the most flexible pricing options among all offer types. Key points for SaaS offers:

* **Supported Pricing Models:** SaaS supports multiple pricing models, and **all plans within a SaaS offer can use one of the following models** (chosen per offer):
  + **Flat Rate Subscription:** A fixed price for the service, independent of the number of users. This can be a **monthly fee or an annual fee** (with multi-year options up to 5 years). For example, you might offer *Basic* and *Premium* plans at $X per month or $Y per year. A flat rate works well when the value provided isn’t strictly tied to user count or usage volume, offering simplicity to customers.
  + **Per-User (Per-Seat) Pricing:** A price **per user license** who will access the SaaS product, e.g. $20/user/month. This model scales revenue with the number of seats purchased. You can define minimum or maximum seat counts for a plan and offer tiered pricing (volume discounts) by creating different plans (e.g. an *Enterprise* plan for larger user counts at a lower per-user rate). Use per-user pricing if the service’s value (or cost) scales roughly linearly with the number of end-users. This is common for B2B SaaS apps where each additional user adds value.
  + **Usage-Based (Metered) Pricing:** Charges based on actual **consumption or transactions** measured from your application. SaaS offers allow custom metering dimensions via the Marketplace Metering API, enabling you to bill for units like API calls, data processed, messages sent, etc. This can be used standalone (a purely consumption-based model) or in combination with a base fee. For a purely consumption SaaS, you might set a base price of $0 and charge customers only for what they use (all revenue via meters). More often, publishers use a hybrid model: e.g. a base subscription that includes some usage allowance, plus overage charges for usage beyond that limit. Metered pricing is ideal when usage varies significantly per customer or you want a “pay-as-you-go” model (e.g. for SaaS APIs, analytics platforms charging per data volume, etc.).
* **Billing Terms & Frequency:** SaaS subscriptions offer flexible billing terms and frequencies:
  + You can offer **monthly or annual billing** options for a plan. Payment can be **monthly recurring** (pay-as-you-go each month) or **upfront** for a longer commitment.
  + SaaS supports **contract terms from 1 month up to multi-year commitments**. Specifically, you can configure plans with **1-month, 1-year, 2-year, 3-year, 4-year, and 5-year** subscription durations. Multi-year terms (anything beyond 1 year) are typically paid upfront in full at purchase (the customer is billed once for the entire term). For example, a 3-year plan would bill the customer one time for 36 months of service. Longer-term commitments often come with discounted rates to incentivize upfront payment.
  + It is possible to **combine billing term options within the same plan**. In Partner Center, you can specify prices for multiple term lengths (for instance, offer a monthly option and a 1-year option for the same plan). The customer can choose their preferred term (e.g. month-to-month vs. annual) at the time of purchase. Each term can have its own price point. **Important:** Only one billing frequency is allowed per term in a given plan (e.g. if a 1-year term is offered as both annual upfront and monthly installments, that would require two separate plans — typically, however, 1-year term is just paid upfront).
  + Customers are **billed through their Azure bill**. Monthly terms are charged each month, annual/multi-year terms are charged upfront at purchase (and subsequently on renewal). Any usage-based (metered) charges are metered and billed **monthly in arrears** (even for customers on an annual base plan).
* **Custom Metering Capability:** SaaS offers **fully support custom metered billing**:
  + You can define up to **30 custom meter dimensions** for a SaaS offer (each representing a billable unit of consumption, like “API calls” or “GB of data”).
  + Each meter has a price per unit in USD (you can override pricing per market if needed via a pricing file) and optionally an included quantity in the base plan (for flat plans).
  + **Reporting usage:** It is the publisher’s responsibility to **track the customer’s usage** for each meter and report the usage to Azure via the Marketplace Metering API. Microsoft will aggregate these usage events and bill the customer accordingly (usually monthly). Note that you should only report usage exceeding any included base amount (e.g. if a plan includes 1000 API calls free per month, only usage beyond 1000 should be reported for billing).
  + **Combination with flat or per-user plans:** You can enable metered billing on top of either a flat-rate or per-user base plan. For instance, you could have a per-user base fee and also charge for additional usage (though in practice many publishers who need both seat and usage charges might structure one of them as a base fee and the other via meters).
  + This capability allows hybrid models, but be cautious to keep it understandable (e.g. clearly communicate what usage is included vs. billed extra).
* **Key Constraints & Best Practices for SaaS:**
  + **Single Pricing Model per Offer:** All public plans in a given SaaS offer **must use the same base pricing model type** (either all are flat-rate or all are per-user). You cannot mix flat and per-seat plans within one SaaS offer listing. If you need to offer both a per-user and a flat option for the same product, you would need to create two separate offers (or use private plans for special cases).
  + **Pricing Model Lock-In:** After a SaaS offer is published, you **cannot change the pricing model type** of that offer. For example, you can’t switch an offer from per-user to flat-rate later without creating a new offer. Plan carefully and choose the model that fits your business from the start.
  + **Plan Variations:** You can create multiple plans (SKUs) within one SaaS offer to represent different editions or service tiers (e.g. Basic, Standard, Premium). These plans can vary by features, capacity, included usage, or user count limits. However, as noted, they will all share the same billing model (e.g. all per-user).
  + **User Limits:** If using per-user pricing, you have the option to set a **minimum and/or maximum number of users** that can be purchased for each plan. For instance, an Enterprise plan might require at least 100 seats. These limits are configured in the plan details and cannot be changed once the offer is published or in use.
  + **Free Trials:** You may offer a **free trial period (typically 30 days)** for a SaaS plan. This is configured per plan (as an option for a one-month free trial) and allows customers to start a subscription without charges for the first month. After the trial period, it converts to a paid subscription automatically. **Note:** Once a transactable SaaS offer is published with a trial enabled, you cannot remove that trial option for that plan, so enable it only if you intend to keep it.
  + **Cancellation and Renewal:** SaaS subscriptions will auto-renew by default at the end of their term (monthly or annual) unless the customer cancels. Customers can cancel at any time (the subscription will remain active until the end of the paid term). Multi-year subscriptions also auto-renew for the same term unless canceled. If you negotiate custom terms or pricing with a customer, you would typically use a **private offer** rather than relying on mid-term adjustments (since the platform doesn’t support changing the price or term mid-subscription).
  + **Selecting a Model:** Choose the pricing model that aligns with how your customers derive value from your service:
    - If customers value **predictable costs** or your costs are mostly fixed, a flat rate can be attractive.
    - If bigger customers inherently use more and get more value, **per-user** might align revenue with usage.
    - If usage is highly variable or you want a low entry cost, consider **usage-based** so customers pay in proportion to what they consume. You can also combine a base fee for commitment plus usage for overage to balance predictability and flexibility.
  + **Simplicity:** It’s generally best to **keep pricing simple and transparent**. Avoid too many complicated meters or a complex matrix of plans which might confuse customers on what they will be billed. Clear packaging (with perhaps one primary value metric) is easier to sell and for customers to understand. If you do have a complex usage model, provide examples to customers of how their bill is calculated.
  + **Technical Integration:** Remember that a SaaS offer requires integrating with the **SaaS fulfillment APIs** for provisioning, seat management, etc. This is separate from pricing, but it’s important operationally. Also, for metered billing, ensure your application is instrumented to measure usage accurately and handle high-scale reliably when reporting to the meter API.
  + **Use Cases:** SaaS offers are ideal for multi-tenant software solutions or services that the publisher runs as a service (e.g., SaaS apps, data feeds, APIs). They allow maximum flexibility in monetization since you’re not directly constrained by Azure resource billing.

## **Virtual Machine (VM) Offers**

In a **Virtual Machine offer**, you publish VM images (or a VM solution) that customers deploy into their **own Azure subscription**. The customer runs the software on Azure infrastructure (VMs) in their environment. Pricing for VM offers primarily revolves around software licensing fees on top of Azure VM costs. Key points for VM offers:

* **Supported Pricing Models:** There are two main models for VM offers:
  + **Usage-Based (Pay-as-You-Go):** You set an **hourly price for the software** running on the VM. The customer is charged per hour (or minute, metered by Azure) while the VM is running. This charge is in addition to the normal Azure infrastructure costs (compute, storage, networking) for the VM. For example, you might charge $0.10/hour for your software on a given VM type. If the VM runs 10 hours in a month, the customer pays $1 in software fees that month (plus whatever Azure charges for the VM itself). This model scales directly with usage and is the default for paid VM offers.
  + **BYOL (Bring Your Own License):** You list the VM image with **no software fee ($0/hour)**. This means the marketplace does not bill the customer for any software licensing; the expectation is that the customer has obtained a license through other means (direct from you or a license key they already possess). BYOL essentially makes the marketplace listing a convenience for deployment, not a point of sale. It’s often used when you want to allow existing customers to deploy your software via Azure Marketplace or if you have a separate licensing channel. Note that if you choose BYOL, your software should enforce license requirements (for instance, require a product key or login) because Azure won’t gate usage.
* **Billing Terms:**
  + **Hourly Billing:** By default, usage-based VM software charges are metered hourly and billed monthly in arrears. This means the customer’s Azure bill will show a marketplace charge based on the VM’s runtime each month. There is no long-term contract by default—if the customer shuts down or deletes the VM, the charges stop.
  + **Reserved Billing Options:** Azure Marketplace introduced **software reservation (long-term) options** for VMs, analogous to Azure Reserved Instances. This allows you to offer **1-year or 3-year upfront billing** for the VM software. If you enable this, the customer can choose to purchase a 1-year or 3-year “software plan” for your VM offer. They pay a single upfront price for that term, and in return, they are not charged hourly software fees for the duration (effectively locking in a discounted rate for committed use). This is optional to use, and not all VM offers have a reserved option, but it can be attractive for customers wanting to commit long-term. (For example, a VM software that costs $0.10/hr might offer a 1-year reservation for ~$876 upfront, which is equivalent to $0.10 \* 24\*365 with perhaps a discount.)
  + There are **no multi-year terms beyond 3-year for VM software** publicly (aside from custom private offers). For longer commitments or custom billing schedules, publishers can use private offers.
* **Custom Metering: No custom meters are supported for VM offers.** Unlike SaaS or other offer types, you cannot add additional metered dimensions to a VM offer. The only billable unit is the VM’s runtime (and optionally the number of cores or size, which is handled by setting price per VM size). This means you cannot directly charge per user, per transaction, or any metric other than hours via the VM offer.
  + **Implication:** If your pricing model requires something like usage-based events or per-user charges on a VM-deployed solution, you have to handle that outside the marketplace (either baked into the hourly price by estimating average usage, or by implementing a separate service or offer for those charges).
* **Key Constraints & Best Practices for VM offers:**
  + **Single VM vs. Multi-VM Solutions:** A VM offer is best suited for solutions that can run primarily on a single VM (or a scale set of identical VMs). The marketplace VM listing can deploy one VM (or a VM scale set), but if your solution requires multiple coordinated VMs or other Azure services (storage, database, etc.), consider using an Azure Application offer (Managed App or Solution Template) instead for a smoother deployment experience.
  + **Image Maintenance:** Ensure you keep your VM images up to date (with security patches, latest version of your software, etc.). Customers deploy directly from your images, so outdated images can lead to security vulnerabilities or poor user experience. Microsoft periodically scans marketplace images for issues and may unpublish offers that are insecure or very outdated.
  + **BYOL Considerations:** If you offer a BYOL edition:
    - Make it clear in the description that a separate license is required. Often publishers provide a link or instructions on how to obtain a license.
    - Use BYOL to serve existing customers or offer a trial that requires contacting sales for a key. BYOL offers should ideally still enforce that unlicensed usage is not permitted (since Azure won’t charge anything, the onus is on your software to permit or restrict use).
  + **Pricing per VM Size:** When configuring a usage-based VM offer in Partner Center, you will specify prices for each Azure VM size (SKU) that you want to support. It’s common to price **per vCPU or per core**. For example, you might set $0.05 per core/hour – so a 4-core VM would effectively be $0.20/hr and an 8-core VM $0.40/hr. This way, larger VMs running your software incur higher fees, aligning with likely greater usage or value. If your software doesn’t scale by cores, you can set a flat hourly price for all sizes, or limit which VM sizes the image is available on.
  + **No Free Trial Mechanism:** There isn’t a built-in free trial period for VM offers. A customer can try your VM by deploying it and incurring charges for only the time they use it (they could run it for a short time to evaluate, incurring minimal cost). If you want to let customers try the software without any marketplace charge, you’d need to either use a BYOL model (so the marketplace doesn’t charge) combined with a trial license key that expires, or create a separate free edition image.
  + **Combining with Azure Costs:** Remember that customers will see the **combined cost** of running your VM (Azure infrastructure + your software) in Azure cost estimates. Price your software reasonably relative to the Azure cost. If your software fee is extremely high compared to the VM cost, customers will evaluate if the value justifies it. Offering a range of VM sizes (including smaller ones) can allow cost-conscious trials or use.
  + **MACC Eligibility:** If your VM offer is a transactable paid offer and is IP co-sell eligible, the customer’s spending on your software can count toward their **Azure consumption commitment (MACC)**. This can be a selling point for enterprise customers with Azure spend commitments.
  + **Use Case:** Use a VM offer for **appliance-style software** where the customer essentially gets a VM with your software ready to use (examples: virtual appliances like firewalls, database servers, analytics or AI appliances). The customer benefits from quick deployment of a pre-configured VM, and you monetize via the hourly software fee seamlessly through Azure billing.

## **Azure Container Offers (Kubernetes Applications)**

An **Azure Container offer** allows publishers to sell containerized applications that customers deploy into their **Azure Kubernetes Service (AKS)** clusters. It’s essentially a managed Helm chart offering. From the customer’s perspective, purchasing a container offer is like subscribing to a software that will run on their AKS. Key points for Container offers:

* **Supported Pricing Models:** Container offers support two main monetization approaches, which can also be combined:
  + **Usage-Based Charging:** Similar to VM offers, you can charge based on usage metrics such as **per core/hour**, per pod-hour, or other custom-defined units. For example, you might charge for each vCPU the software consumes per hour on the cluster, or per execution, etc. The usage is measured while the containerized application runs in the customer’s cluster.
  + **Fixed Monthly Fee:** You can also set a **flat monthly price** for the container application, regardless of usage. This can be thought of as a subscription fee for having the software installed on the cluster.
  + Many publishers use a combination: e.g. a fixed monthly base fee for providing the software, plus usage-based fees for scaling or consumption beyond a baseline. The pricing model chosen should reflect how the containerized solution provides value or incurs cost.
* **Billing Terms:**
  + Container offers are inherently **month-to-month**. The platform as of now **does not support annual or multi-year prepaid terms for container offers** in the public marketplace. Customers subscribe and are billed monthly (or per usage, tallied monthly).
  + This means you cannot natively offer a 1-year or 3-year discounted contract for a container offer via the public listing. If longer commitments or prepaid deals are needed, those typically have to be handled via a private offer with custom billing arrangements or outside the marketplace.
  + Customers can cancel the subscription at any time; if they do, charges stop (either immediately or at the end of the billing period, depending on how it’s implemented — generally, it stops new charges immediately and they won't be billed further).
  + If using purely usage-based model (no base fee), the customer’s bill will only show charges when the app was actively used.
* **Custom Metering: Yes, container offers support custom metering** using the Marketplace Metering API, much like SaaS:
  + You can define custom usage meters for things like number of transactions, data processed, etc., and report usage from within your container app or an associated service.
  + **No native GPU meter:** One notable limitation is that Azure does not provide a built-in meter for GPU usage in AKS equivalent to a vCPU meter. If your solution utilizes GPU resources and you want to charge for that, you must implement a custom meter (e.g. count GPU-minutes as a custom metric).
  + In general, think carefully about what metric to use for billing that correlates with the value delivered. If your app mainly consumes CPU and memory, vCPU-hours could be a fair measure. If it processes transactions, maybe count per transaction.
  + All usage reported will be accumulated and billed monthly to the customer.
* **Deployment and Access:** When a customer “purchases” a container offer, they typically go through Azure Marketplace, which then triggers a deployment (often via Helm chart) to their specified AKS cluster. The software then runs in their cluster (in their subscription). From a pricing standpoint, it behaves somewhat like SaaS (a subscription tied to an entitlement), but technically the workload runs on the customer’s infrastructure (AKS).
  + The customer needs to have an Azure Kubernetes Service cluster to install the offer. The marketplace will guide them to select a cluster or create one during purchase.
* **Key Constraints & Best Practices for Container offers:**
  + **No BYOL:** Container offers are **always transactable** (or free). There is no BYOL licensing mode for containers. If the offer is listed as paid, it must transact through marketplace. (Alternatively, you could list a container offer as free which would just serve as a deployment mechanism with no charges, if you plan to handle licensing off-platform, but that would be akin to BYOL outside the marketplace’s knowledge.)
  + **CSP Channel:** Currently, container offers are **not available for Azure CSP resellers**. In other words, customers using a CSP-managed subscription cannot directly purchase container offers from marketplace. This is a limitation of the channel; as a publisher, you might work around this by offering your solution via a different mechanism for those clients or through private offers if/when that becomes supported.
  + **Simplify Pricing:** Given that Kubernetes apps can scale dynamically, consider tying your pricing to an easily measurable resource like CPU cores or nodes. For instance, charging per vCPU-hour means if the customer scales your app up or down, the cost adjusts naturally. If you charge a flat fee, be clear on what that fee covers (e.g. one deployment up to a certain size? Unlimited usage? etc.).
  + **Communicate Requirements:** Make sure to document any prerequisites (like needing an AKS cluster of a certain version or configuration) in your offer listing. This isn’t directly pricing, but it affects customer success with deploying your paid offer.
  + **Value Justification:** If charging a fixed monthly fee, ensure it’s clear what value they get regardless of usage (perhaps support, or the right to run the software). If charging per use, ensure high usage correlates with high value (so customers feel the cost scales reasonably).
  + **Monitoring & Metering:** Just as with SaaS, you’ll need a mechanism (often within the container or an accompanying sidecar or service) to report usage via the meter API. This should be secure and reliable. If the container can’t phone home (due to network restrictions), you may need a strategy to collect usage (maybe via an optional connected agent).
  + **Cancellation Handling:** When a customer unsubscribes, you should have logic to detect that and potentially disable or limit the application usage in their cluster (or at least stop reporting usage). Often, the contract termination will be communicated through an event that your service can catch via webhook or other means.
  + **Use Case:** Container offers are ideal for solutions that are deployed into a Kubernetes environment—such as add-ons to AKS (logging/monitoring agents, controllers), or applications that run on a customer’s cluster for data locality or other reasons. It provides a way to monetize Kubernetes-deployed software through Azure Marketplace.

## **Azure Managed Application Offers**

An **Azure Managed Application** (a type of “Azure Application” offer) enables publishers to provide solutions that get deployed into the **customer’s Azure subscription**, but **remain managed by the publisher**. This is essentially a way to sell a managed service for an Azure solution, with Azure Marketplace handling the billing for that management. Key points for Managed Application offers:

* **Supported Pricing Models:** The charging in a Managed Application offer is intended to cover the **management and support services** provided by the publisher (not the underlying Azure resources, which the customer continues to pay for separately). Two model options:
  + **Flat Monthly Management Fee:** A fixed monthly charge for the service of managing the application. For example, $500 per month to monitor and support the deployed solution. This fee does not typically depend on how many resources or usage – it’s a flat service retainer (unless you package different tiers of service as separate plans).
  + **Metered Management Fee:** Usage-based charges for management activities or quantities. For instance, charging per resource under management, per user of the managed app, or other metrics that represent the scope of management. You might have, say, $100 per month base plus $10 per virtual machine managed by the application, reported via a meter. The platform allows custom meters to be defined, similar to SaaS.
  + In practice, many managed services will use a flat fee if the environment size is known/stable, or a base + variable component to account for different sizes of deployment.
* **Billing Terms:**
  + **Monthly Subscription:** Managed app offers are billed on a monthly cycle. There is **no built-in support for annual or multi-year upfront contracts** for Managed Applications in the public marketplace. The customer pays month by month for your management. If longer-term or upfront payment is needed (perhaps for a discount), that would have to be handled via a private offer or separate arrangement.
  + The charges will appear on the customer’s Azure bill each month under marketplace charges. If using both a flat fee and meters, the flat fee is typically charged in advance for the month and the metered charges accumulate throughout the month and are charged in arrears.
  + Cancellation stops future charges; typically access to the managed resources by the publisher would be removed upon cancellation.
* **Scope of Charges (Important Constraint):** With managed applications, **Microsoft policy dictates that marketplace charges should only cover the management/service fees**, not the underlying software license value or Azure infrastructure:
  + The **customer is still responsible for all Azure resource costs** (VMs, storage, PaaS services deployed) in their subscription at regular Azure rates.
  + If your solution includes proprietary software that normally would have a license fee, you are **not supposed to charge that license through the managed app fee**. Instead, the recommended approach is to **include that software as part of the deployed resources** so that it is billed appropriately (e.g., include a VM from a marketplace VM offer that has its hourly price, or instruct the customer to acquire a license separately). Essentially, the managed app’s marketplace fee should strictly be for your operational involvement (support, management, updates, etc.).
  + Example: You offer a managed database service. The managed app deploys an Azure SQL or a SQL VM into the customer’s subscription. The customer pays Azure for that SQL resource (including any licensing that comes with it, or a marketplace VM software fee if that VM has one). Your managed application might charge $200/month for managing the database (backups, monitoring, support). If you also have proprietary software in the solution, you’d either charge for it via a VM meter or factor it into that $200 carefully following guidelines. Be transparent to avoid “double charging.”
* **Custom Metering:** Yes, managed apps can have custom meters just like SaaS:
  + These meters can be used for charging based on things like number of tickets handled, amount of data processed by the managed service, number of resources beyond a base count, etc.
  + **However**, as noted, do not use meters to charge for raw Azure usage. For example, don’t have a meter that essentially charges for VM hours or Azure storage, since the customer is already paying Azure for that. Meters should correspond to your management or additional value-add services (e.g., “monthly security scans per VM” or “GB of data analyzed by our service team” might be valid if that’s part of your managed service).
  + As with SaaS, you must emit usage events via the API from whatever system is tracking those metrics (which could be a central management system you operate).
* **Operational Model:** In a managed application, the resources (Azure resources) are deployed into a **locked resource group in the customer’s subscription**. The publisher (you) gets access (usually as a managed identity or through delegated access) to that resource group to perform management operations (like updating, monitoring, troubleshooting). The customer cannot directly alter certain parts of those resources without going through you (depending on how it’s set up), which ensures the integrity of the managed service. They see a resource in their subscription that represents the managed application, and behind the scenes the actual resources are contained within.
  + The customer essentially outsources the operation of those resources to you, the publisher, for the fee you charge.
* **Key Constraints & Best Practices for Managed App offers:**
  + **Clearly Define What the Fee Covers:** Be explicit in documentation about what services you provide for the monthly fee. For example: “This monthly fee covers 24/7 monitoring, patching, and technical support for the deployed application.” This helps justify the cost and avoids confusion about Azure vs. your charges.
  + **Don’t Duplicate Charges:** Ensure you *do not charge twice* for the same thing. If the customer is paying Azure for a VM, you should not charge a separate “VM usage fee” in the managed app. If your own software license is part of the solution, consider publishing a linked VM offer for it or include it as a line item via Azure (some advanced scenarios allow combining offers).
  + **Use Private Offers for Complex Deals:** If a customer wants a custom duration (e.g., annual pricing for a managed service) or custom pricing, you can create a private plan for that customer rather than bending the public offer rules. Private offers allow flexible billing schedules (like quarterly payments, custom start/end).
  + **Access Management:** After a customer purchases, ensure your operations team has the necessary access to the customer’s resources granted by the managed app. If the customer revokes your access (which they can do by removing the service principal or so), you may need to have a clause that they must cancel the marketplace subscription if they don’t want service, to avoid them being billed without you being able to provide service.
  + **Ideal Use Cases:** Use a Managed Application offer if you have a solution that **requires ongoing involvement from your team** to run or maintain. This is common for managed services, turn-key solutions where the customer doesn’t have the expertise or capacity to run it themselves. Examples: managed security services, managed backups, or complex software that needs babysitting where you handle the ops.
  + **Disentangling Software vs Service:** Think of a managed app as selling your **operations as a service**. If you also need to sell software IP, you might combine it with a VM offer internally. Many publishers deploy at least one custom VM (possibly a marketplace VM from their catalog) as part of the managed app so that they can capture the software license fee there, while the managed app plan covers the labor/management fee.

## **Solution Template Offers**

A **Solution Template** is an Azure Marketplace offer used to deploy a collection of Azure resources via an ARM template (in Azure portal, these are often listed as “Azure Applications” with the plan type being a solution template). **Crucially, solution template offers are non-transactable** – meaning **there is no direct cost or billing for the offer itself**. Instead, it’s a vehicle to set up Azure resources which will then incur standard Azure charges. Key points for Solution Template offers:

* **Pricing Model:** There is **no price for the solution template itself**. The publisher does not get paid through the marketplace for a solution template deployment. The marketplace listing is essentially a deployment script. All costs to the customer come from the Azure resources that are deployed:
  + If those resources include Azure services with costs (e.g. VMs, Azure SQL, App Service), the customer will pay Azure for those at normal rates.
  + If some resources involve third-party software (like a VM image that has licensing), the customer would be prompted to accept those licensing terms and potentially be billed for that usage (see next point about monetization).
  + The solution template plan type does **not allow attaching a price or meter** in Partner Center. It’s always listed as “Free” in pricing since the charges are outside of it.
* **Monetization via Component Offers:** If you as a publisher **want to monetize a solution template**, the typical approach is to **embed a paid component within the template**:
  + The most common method is to include a **Marketplace VM offer** as one of the resources in the ARM template. For example, you have a paid VM image (perhaps for your software) and the solution template deploys that VM (along with other infrastructure). During deployment, the customer will see that the VM has a cost, agree to it, and after deployment the customer will be billed for that VM’s software usage just as if they had deployed that VM offer directly. This way, you earn software fees via the VM.
  + You could also potentially use a managed application or container as part of the deployment, but typically it’s done with VMs or relying on Azure service charges.
  + If no paid component is included, the solution template deployment will only incur Azure’s costs. In such cases, effectively you are not charging anything via marketplace for your IP. This is fine if your goal is to make it easy for customers to deploy your solution while you plan to earn revenue through Azure consumption or separate support contracts.
* **Billing Terms:** Since solution templates have no price, there are no billing terms. It’s a one-time deployment action. After deployment, there is no ongoing subscription from marketplace’s perspective (though the deployed resources may run indefinitely and incur costs).
  + The customer doesn’t “subscribe” to a solution template; they just run it. If they want to run it again or get updates, they would basically redeploy or use a new version of the template manually.
  + There is also no concept of renewal or cancellation because there’s no subscription. The customer can of course delete the resources if they don’t want them.
* **Custom Metering:** Not applicable. The marketplace doesn’t track anything for a solution template beyond initiating the deployment. Any usage metering happens at the Azure resource level (e.g., Azure meters for VMs, storage, etc., which are billed in the Azure bill, not as a marketplace item).
* **Use Case & Purpose:** Solution templates are primarily used to provide **easy deployment of complex solutions**. They are often used for:
  + **Free or BYOL offerings:** If you want to allow customers to deploy your software but you handle licensing off-platform (or it’s a free/open-source solution), a solution template can automate the setup.
  + **Complex deployments:** Solutions that require multiple VMs, networking, storage, etc., can be packaged into a single ARM template for convenience.
  + **Trials or POCs:** You might provide a solution template for a trial deployment (perhaps deploying a limited version of your product, using BYOL licensing, and later upsell a managed service).
  + First-party Azure scenarios also use solution templates to help customers deploy reference architectures, etc., without any charge from the template itself.
* **Key Constraints & Best Practices for Solution Templates:**
  + **No Direct Revenue:** Be aware that a pure solution template offer **does not generate marketplace revenue for you** unless tied to a paid component. If your business model requires recurring revenue or any kind of service fee, a solution template alone is not sufficient.
  + **Monetize via included resources:** If you need to monetize, plan to include at least one **transactable resource**. The simplest is a **VM from a Marketplace offer** (which could be your own VM offer). Alternatively, instruct users to acquire a license separately (less seamless). Make sure any monetized component is clearly described so users know what they will be paying for.
  + **Consider Managed App for Ongoing Services:** If your scenario involves **ongoing management or support** and you want to charge for that, consider using a Managed Application offer instead of a solution template. Managed Apps are transactable and designed for recurring charges. Solution templates are best for *deploy-and-you-manage-it-yourself* scenarios.
  + **Customer’s Azure Costs:** List the Azure services that will be deployed and emphasize to customers that they will incur Azure usage charges for those. Providing an estimate or using Azure’s cost estimation in the portal can help set expectations (e.g., “This template will deploy 2× Standard\_D2\_v3 VMs, 1× SQL Database S1, etc. Typical Azure cost ~$X/hour”).
  + **No Update Mechanism:** If you update the template (new version), customers won’t automatically get those changes; they’d have to redeploy or manually update resources. There’s no built-in upgrade flow as there might be with managed apps. So for continuous improvement or management, again, solution template is hands-off.
  + **Simplicity and Reliability:** Ensure the ARM template is well-tested and robust, since any failure might frustrate users. Provide parameters to allow basic customization. Document any post-deployment steps the customer needs to do (especially if BYOL license activation is needed).
  + **Combining Offer Types:** It’s possible to have both a solution template and a managed app in one Azure Application offer (different plans), or evolve from template to managed app as needed. Think of a solution template as a starting point: if you see demand for a fully managed version, you might add a managed application plan to the offer in the future.

## **Selecting the Appropriate Offer Type & Pricing Model: Key Takeaways**

* **Align Offer Type with Deployment Model:** First determine how your solution will be delivered.
  + If it’s a multi-tenant cloud service you operate, go with **SaaS**.
  + If it’s a single VM image that the customer runs, **VM offer** is the straightforward choice.
  + If it’s a Kubernetes-based application, use **Azure Container offer**.
  + If it’s a full solution stack that the customer will run and perhaps you will manage, consider **Managed Application** (for managed service) or **Solution Template** (if just providing deployment).
  + This choice will narrow down the pricing models available.
* **Choose a Pricing Model that Matches Value Metric:** Within the chosen offer type, pick the pricing model that best aligns with how your customers perceive value and how you incur costs:
  + **Flat vs Per-User (SaaS):** Use flat for simplicity or whole-org value, per-user if scaling by users.
  + **Usage-Based:** Use metered models if usage varies significantly or you want to lower the entry cost. Ensure you have a way to measure usage accurately.
  + **Hybrid Models:** Don’t be afraid to combine a base fee plus usage overages to balance revenue stability and fairness for heavy users.
  + **BYOL vs Transact (VM):** Use BYOL if you have existing sales channels or want to avoid sharing revenue for certain customers; use transact (payg) for ease of procurement and to count towards Azure commitments.
* **Billing Term Flexibility:** Azure marketplace now allows up to 5-year contract durations for SaaS (and software plans for VMs). Offering multi-year options can attract customers who prefer long-term commitments (often at a discount). Use this feature if it aligns with your sales strategy, but remember multi-year plans lock in pricing, so set a sustainable price. For offer types that don’t support multi-year (Containers, Managed Apps publicly), handle those via custom deals if needed.
* **Private Offers for Special Cases:** If the standard models don’t fit a particular customer scenario (e.g., they want a unique pricing metric, a specific discount, or a different billing schedule like quarterly payments), you can create a **private plan/offer** for that customer. Private offers let you override pricing and terms on a per-customer basis without affecting your public plans.
* **Constraints to Keep in Mind:**
  + You cannot mix pricing model types within one offer (applies to SaaS, VM images must be all one model or separate SKUs for BYOL vs paid).
  + Once published, core aspects like pricing model and user seat limits usually cannot be changed without republishing a new offer or plan.
  + Some channels have limitations (e.g., Container offers not in CSP, multi-year not in CSP until updated, etc.).
  + Understand the **revenue share** and billing cycle: Marketplace charges typically incur a percentage fee to Microsoft (e.g., 3% for commercial marketplace transacted deals as of this writing), and payouts to you are handled through your partner center account. Also consider that customers may have Azure spend commitments (MACC) that your offer could contribute to if eligible — being MACC-friendly can be a selling point.
* **Best Practice – Customer Transparency:** Provide clear, concise information in your marketplace listing about how your pricing works. If you use metered billing, list what kinds of things will generate usage charges. If multi-year options are available, mention the discount or value for longer commitments. A well-informed customer is more likely to proceed with a purchase and less likely to be surprised (and frustrated) by the bill.

By understanding these offer types and their pricing frameworks, you can select the model that best fits your solution and business model while providing a smooth experience for customers via the Azure Marketplace. Always refer to the latest Azure Marketplace documentation for up-to-date capabilities (as features evolve, such as new term options, new offer types, etc.) and plan your offer strategy accordingly.