ChatGPT

Usage-Based Billable Entities: Aligning SaaS Pricing with Customer Usage

Authors: Dinis Cruz and ChatGPT Deep Research

Introduction

Software-as-a-Service (SaaS) companies have traditionally relied on fixed subscription plans – charging customers a monthly or annual fee for access to a product. However, this model often misaligns cost with actual value delivered. Customers can end up paying for unused capacity, while heavy users may consume far more resources than their subscription fee covers. As a result, an emerging best practice in SaaS pricing is the **usage-based** or **consumption-based** model, where **billable units** of usage form the core of pricing. In a usage-based model, customers pay only for what they actually use, measured in discrete units (e.g. API calls, data volume, compute hours, etc.), rather than a flat rate unconnected to activity [1] [2]. This approach aims to align revenue with value – delivering fairness to customers and sustainability to providers. In this white paper, we explore the key concepts of usage-based billable entities, the benefits of this model over fixed subscriptions, and practical considerations for implementing usage-based billing in a SaaS product. We also highlight real-world success stories of companies that have embraced usage-based pricing.

Pitfalls of Traditional Subscription Models

Many SaaS startups default to a **fixed subscription** model (e.g. per-user per-month plans). While simple, this approach comes with notable drawbacks:

- **Disconnect from Value:** Subscription fees often remain the same regardless of how little or how much a customer actually uses the service. Customers frequently feel they are paying even when not deriving new value. In fact, industry surveys show that unused or underutilized SaaS licenses are a major issue – 42% of IT professionals reported this as a top challenge, and nearly one-third estimated that 20–39% of their SaaS spend is wasted on underutilized subscriptions [1]. This waste breeds customer dissatisfaction.
- **Cross-Subsidization:** Providers of subscription services implicitly rely on light users to subsidize heavy users. A few "power users" may consume a disproportionate amount of resources (CPU, bandwidth, support time, etc.), eroding the provider's margins. Meanwhile, low-usage customers are still billed the full amount, effectively financing the heavy users' consumption. This imbalance is both unfair to customers and risky for the vendor's cost structure.
- **Churn and Trust Issues:** When customers sense they are overpaying relative to usage, they are more likely to churn (cancel service) or seek alternatives. The lack of transparency in flat pricing can erode trust – clients may wonder if they're getting their money's worth, especially after initial onboarding. Indeed, "finding unused or underutilized SaaS app licenses" is cited as a critical problem, and significant portions of SaaS budget can be wasted on shelfware [1].

In summary, static subscriptions can misalign incentives: the vendor benefits when customers use less (lower costs) despite paying the same fee, and the customer benefits when they over-use relative to their fee – a dynamic that is not sustainable long-term. These shortcomings are driving many SaaS companies to re-think their pricing approach.

The Shift to Usage-Based Pricing

Usage-based billing (also called consumption-based or metered pricing) directly addresses the above issues by tying costs to actual usage. Instead of a fixed fee, the customer's bill is calculated from their **consumption of measurable units** of the service [3] [4]. This model has several compelling advantages, which are causing a broad industry shift in its favor:

- **Aligned Value for Customers:** Users pay **only for what they use**, ensuring a direct correlation between cost and value received. This greatly reduces the feeling of overpaying for unused capacity. If usage drops, so do costs – which is particularly attractive in times when a customer needs to scale down. Conversely, if the customer derives more value and uses more, the cost scales accordingly. This fairness improves customer satisfaction and trust. It's telling that nearly a third of SaaS buyers believe roughly 20–40% of spend is wasted on under-use under subscription models [1] – a clear opportunity for improvement via pay-as-you-go.
- **Win–Win for Providers:** In a well-designed usage model, *heavy users pay more*, covering the extra resources they consume. The provider is not left footing the bill for power users. Every user contributes proportionally to the costs they drive, so low-usage customers are no longer subsidizing high-usage ones. This ensures the **economics remain sustainable** for the vendor while still giving high-usage customers the freedom to consume as needed. When properly priced, each unit of usage is profitable for the company, meaning **growth in usage directly drives revenue and margin**. Many SaaS vendors find that shifting to usage-based pricing improves their financial metrics: for example, a study of public SaaS companies found those with usage-based models grew revenue **38% faster year-over-year** than those with traditional pricing [5]. These companies also achieved higher net dollar retention of customers (on average 9% higher NDR) because existing customers tended to expand usage over time instead of churning [6]. Usage-based pioneers like Snowflake have reported exceptional NDR (~158%), indicating customers keep increasing spend as they find more value [7].
- **Lower Barrier to Adoption:** Usage-based pricing often makes it easier for new customers to start using a product. With no large upfront commitment or lock-in, a team can try the service on a small scale and see value before increasing usage. This **"land and expand"** dynamic is a hallmark of product-led growth. Indeed, under usage-based models, more individuals at a customer organization tend to get hands-on with the product, since they aren't limited by seat licenses. One analysis found usage-based software companies had **ten times more users per account** on average compared to seat-based subscription companies [8], because anyone interested can try the tool without procurement of a new license. This broad adoption can lead to virality within organizations and more organic growth.
- **Improved Customer Retention:** Because customers are continually seeing the direct relationship between their usage and their costs, they are constantly aware of the value they're getting. This can **reduce churn** – if they use less, their bill shrinks rather than them feeling compelled to cancel a contract. Conversely, if they continue using the service regularly, it's proof of ongoing value which justifies the expense. Over time, this tends to **increase lifetime value**. As noted, net retention rates often improve under usage-based models [6]. Customers feel the pricing is fair and thus stick around and grow usage if the product delivers results.
- **Greater Revenue Potential via Upsells:** Usage-based models naturally accommodate upselling. As customers grow and their needs expand, their usage (and bill) increases organically without a formal sales negotiation for a higher plan. Additionally, companies can introduce new **value-added features or higher-tier services** that are charged based on usage, capturing more revenue from power users. Research shows SaaS firms using consumption models often report net retention above 125% (meaning expansion exceeds churn) and higher valuations from investors [9] [10]. The flexibility and scalability of the model signals strong growth potential.

- **Market Trend and Expectations:** It's worth noting that the SaaS industry at large is moving in this direction. In 2018, only 27% of SaaS companies had some form of usage-based pricing, but by 2022 that figure had risen to 46% [11]. Surveys by OpenView Partners and others indicate nearly half of SaaS companies now offer usage-based options, and **a majority may do so by mid-2020s** [12]. This momentum reflects the competitive advantage such pricing can confer. Companies that stick strictly to inflexible subscriptions risk looking outdated or unaccommodating. Many customers now expect "pay-as-you-go" flexibility – much like they get with cloud infrastructure or utilities – in other software products.

In short, **consumption-based pricing ties pricing to genuine value creation**, creating a more transparent and symbiotic relationship between SaaS providers and customers. When done right, it can fuel faster growth and stronger customer loyalty while addressing the inefficiencies of the old subscription approach [5] [13].

Defining Billable Units (Value Metrics)

At the heart of any usage-based model is the concept of a **billable unit** of usage – sometimes called a **value metric** or consumption metric. This is the **measurable action or resource** that your service provides which you will charge for. Choosing the right billable units is critical; they should be **easy to understand, tied to customer value, and straightforward to meter**. Some examples include:

- **Cloud Infrastructure Units:** Cloud providers like AWS, Microsoft Azure, and Google Cloud pioneered usage-based billing by charging for low-level units such as compute time (CPU or VM hours), storage volume (GB stored per month), and network bandwidth (GB transferred). These units are familiar to technical users and directly reflect consumption of resources. Virtually all AWS services are metered this way – for instance, Amazon EC2 bills per second of VM runtime, Amazon S3 bills per GB of data stored and transferred, etc. Everything is literally "charged to the penny or micropenny" based on usage, aligning cost exactly with consumption [14]. This fine-grained approach has been key to cloud adoption, because startups can begin on AWS with tiny usage (and tiny bills) and scale up as they grow, paying more only when they use more. AWS even offers **free-tier usage allowances** for the first 12 months and certain always-free services to lower the initial barrier [15]. The transparency and granularity of these primitives (compute, storage, bandwidth) make them effective billable units that cover virtually all use cases.
- **Transactions or API Calls:** Many SaaS APIs and communications platforms charge per transaction or call. For example, **Twilio** (a communications API provider) bills per text message sent or call made. If you send 1,000 SMS messages, you pay 1,000 × (unit price per SMS). Twilio augments this with volume discounts and committed-use discounts for high-volume customers [16]. Similarly, payment processors like **Stripe** charge a small fee per successful transaction (e.g. a percentage of the payment amount or a flat fee per charge) rather than a flat monthly fee [17]. This per-transaction unit directly scales with the value the customer gets (each payment processed is clearly tied to revenue for the customer, making the fee easy to justify).
- **Data or Storage Volume:** For data-centric platforms, usage might be measured in data processed or stored. **Snowflake**, a cloud data warehouse, uses a consumption-based model where customers pay for the amount of data they store *and* the compute time (in Snowflake "credits") used for queries and processing [18] [19]. This model allows Snowflake's cost to a given customer to scale with that customer's usage of analytics – small teams with few queries pay very little, whereas large enterprises running massive workloads pay more, but in proportion to the value they derive. Another example is SaaS backup or logging services that charge per gigabyte of data stored or transferred.
- **Operational Events or Tasks:** Some applications define a billable action in terms of a higher-level task or event. For instance, **Zapier** – an automation platform – uses "tasks" (each time a

workflow runs an action, that's one task) as its billable unit. Zapier's pricing includes a certain number of tasks per month in each plan, and if you exceed that, you incur overage charges or move to a higher tier [20]. This is essentially usage-based, though implemented as tiered plans with limits. It works because a task is a meaningful unit of value to the customer (each task is some work automated) and it maps to Zapier's internal costs as well (each task consumes compute/API resources).

- **API Request Volume:** SaaS data services often charge by API usage. **Clearbit**, for example, provides a business intelligence API for enriching contact data, and it prices by the number of API requests (lookups) a customer makes [21]. Each API call returns value (information) to the user and has a roughly constant cost to Clearbit (in compute and data license fees), so per-request billing is logical. Customers can start small – e.g. a few hundred requests – and scale up usage as needed, paying more if and only if they are getting more data value.
- **User-Related Metrics:** In some cases, the value metric might still be related to users but in a more dynamic way than a simple seat license. For example, a service might charge per **active user** or per **user action**. An illustrative case is Slack's pricing: Slack traditionally charged per seat, but with a twist – it only bills for active users in a given month (crediting back inactive ones). This makes it somewhat usage-aligned (you don't pay for a user who never logged in that month). Other products might charge per 1,000 active end-users tracked, or per consumer transaction, etc., depending on what the software enables.

The key in defining billable units is to think **"what does our customer value, and what usage on our platform corresponds to that value?"** That is the metric you likely want to charge for. It should also correlate with your own **internal cost drivers**. In cloud infrastructure, the link is very direct (each GB or vCPU-hour consumed has a known cost). In higher-level SaaS, you might need to identify a few core **usage primitives** under the hood – e.g. computations performed, data stored, external API calls made – that drive your costs, and then package those into metrics that make sense to the user.

Often, a combination of **low-level primitives and business-level units** works best. For example, in an AI service that analyzes documents, you might have primitives like "characters processed" or "CPU-seconds used" (similar to how OpenAI bills per token for their language model API). But you could present higher-level units to customers like "documents analyzed" or "reports generated," as long as each of those can be translated to some underlying compute/storage cost. Similarly, a SaaS platform might have a small number of **core billable actions** (e.g. "graph created", "analysis run", "notification sent") each of which consumes some combination of CPU, bandwidth, etc. By metering these actions, you capture the usage in a way the customer directly connects to their activity. As the voice memo noted, *"think about the places in the business where we add value that are easy to explain, and where the user has a direct hand in the consumption"* – those are your candidate billable entities. Every time the user performs that action (or consumes that resource), they incur a charge. This transparency not only monetizes usage fairly, it also subtly **educates the customer on what parts of the service are most valuable** (since those have a price attached).

Building a Usage-Based Billing System

Implementing usage-based billing requires a strong operational foundation. Unlike a simple subscription where billing is the same each period, usage-based billing means the system must continuously **track and tally consumption events** for each customer. Here are key components and considerations for building a reliable usage-based billing system:

- **Instrumentation & Metering:** You need to instrument your application or service to log every billable event or usage unit per customer. This could be API gateways logging calls, databases measuring storage per account, or internal counters for actions taken. Accuracy is critical – every

unit must be counted. Modern usage-based companies invest in robust telemetry that feeds a **metering service**. For example, at scale, a usage engine might collect data from various microservices (for compute, storage, API calls, etc.), aggregate it, and produce a usage record stream [22]. Each record ties a usage amount to a specific customer and a timeframe. Aligning this metering closely with your monitoring/logging infrastructure can be efficient – essentially, your billing events are a subset of your application events.

- **Usage Attribution:** Multitenant systems must attribute usage to the correct **customer account or user**. This often means each request or action includes an account identifier in logs. A common pitfall is failing to track some background or asynchronous usage to the triggering customer – be sure to propagate customer IDs to any downstream processing (for instance, if a user's action kicks off a background job, that job's resource consumption should accrue to that user's bill). Designing your architecture with **usage tracking in mind up front** is ideal, as retrofitting can be complex.
- **Billing Engine & Pricing Rules:** Once usage data is collected, a billing engine applies pricing to compute charges. This could be as simple as multiplying units by a unit price, or as complex as applying tiered rates, volume discounts, or bundling into monthly plan charges. In early stages, some startups export usage data and calculate bills offline (e.g. in spreadsheets or a script). But as you scale, you may incorporate a dedicated billing platform or service. Many SaaS-specific billing solutions (Stripe Billing, Chargebee, open-source options like Lago, etc.) support usage-based pricing models. They allow you to define **rate plans, tiers, discounts, and billing cycles** to automatically convert usage records into customer invoices [23] [24]. Key features to consider in a billing system include: handling different units of measure, aggregating usage over the billing period, and managing any pricing promotions or caps.
- **Real-Time Visibility and Alerts:** Both you and your customers benefit from real-time visibility into usage. Providing customers with a **dashboard of their current usage and costs** prevents "bill shock" and builds trust through transparency [25] [26]. On your end, real-time monitoring can trigger alerts or safeguards – for example, if a customer suddenly uses an unusually high amount in a short time (which could indicate a configuration mistake or even a malicious usage spike), you might alert them or require confirmation to continue. Alerts for customers nearing a certain credit or usage limit are also a best practice [26] [27]. This keeps the billing relationship collaborative rather than punitive.
- **Billing Cycle & Invoicing:** Decide on a billing frequency (monthly is most common for SaaS). At the end of each cycle, the usage is totaled and the customer is billed. Some models also support mid-cycle **threshold billing** – for instance, if a customer exceeds a usage threshold, an invoice triggers immediately. This can protect the provider from extremely large unpaid usage accruals. In all cases, provide a **detailed invoice** that shows the breakdown of usage units and rates, so the customer can clearly see how their bill was calculated. This detail reinforces the value of what they consumed.
- **Payments and Credit Management:** One challenge with pure pay-after-use models is the risk of non-payment (a customer could rack up a large bill and then fail to pay). To mitigate this, many companies either **collect a payment method up front** and auto-charge it, enforce some credit limit, or adopt a **prepaid credit system**. Prepaid credits mean the customer pays in advance for a certain amount of usage credit (similar to buying phone top-up minutes). Each usage deducts from their credit balance [28]. If the balance runs out, the customer must purchase more to continue using the service [29]. This approach, used by some SaaS and API businesses, ensures the provider always has cash before delivering service, essentially eliminating credit risk. The downside is potential friction – service interruptions if credits lapse, and the customer having to predict usage upfront. Another approach is **postpaid with guard rails**, sometimes called progressive or threshold billing [30]. In this model, you might start new customers with a low usage threshold (say $50 worth of usage). As soon as they hit that, you charge their card and perhaps raise their limit. Over time as trust builds, you allow higher and

higher usage between charges. If a charge fails, the service can be paused at the threshold so you don't accumulate further losses [30]. This progressive model is effectively how cloud providers operate: new accounts are limited until they establish a history. Either way, thinking through **credit risk and cash flow** is important in usage-based billing. Requiring advance purchase of credits (or a credit card on file with auto-pay) is highly recommended to avoid scenarios where a customer consumes expensive resources and then disappears without paying [31].

- **Scalability and Performance:** A technical consideration – the volume of usage events can be very high for a successful product. Your metering and billing pipeline must handle potentially millions or billions of events (think of AWS monitoring billions of resource usage data points). This may involve batching data, using streaming processing, and ensuring your billing computations are efficient. Fortunately, modern cloud analytics tools and billing services are capable of this, but it's an area not to underestimate. Start with a system that can grow or use managed services for the heavy lifting (for example, some teams pipe usage events into a data warehouse and compute bills from there).

In essence, implementing usage-based billing is part technical challenge and part business process. It tightly couples with your product's analytics and monitoring. Many SaaS companies find that investing in a good billing infrastructure early pays off, because it enables rapid iteration on pricing, introduction of new pricing models, and clear insight into how customers use the product [24] [32]. In fact, the feedback from billing data can inform product decisions – e.g. which features are most heavily used (and thus most valuable). Companies like DigitalRoute provide specialized "usage engine" solutions to handle the data collection and mediation for usage-based models, highlighting the rise of tooling in this space [22]. Whether you build in-house or integrate a third-party solution, ensure accuracy, transparency, and scalability in your billing system – it's the backbone of trust in a usage-based relationship.

Successful Examples of Usage-Based Pricing

Usage-based billing is no longer theoretical – many of the world's most successful tech companies have built their business on this model. Below we highlight several real-world examples across different domains, illustrating how they define their billable units and structure their pricing. These cases show that usage-based models can be adapted to infrastructure platforms, APIs, SaaS applications, and more, providing flexibility and fairness in each context.

- **Amazon Web Services (AWS)** – *Cloud Infrastructure.* AWS is perhaps the poster child for usage-based pricing. From the start, AWS offered **on-demand pay-as-you-go pricing** for compute, storage, database, and other services. Nearly every AWS service is metered by the second, hour, or byte. For example, with AWS Lambda (serverless compute) you're billed per millisecond of execution time, and with Amazon S3 storage it's per gigabyte-month of data stored. AWS complements its on-demand rates with options for **volume discounts and reserved capacity**: customers can reserve instances for 1-3 years for a lower per-hour price, or use spot instances (excess capacity at big discounts) for non-critical workloads [33]. There are also bulk discount programs and committed spend agreements for large customers. Additionally, AWS's Free Tier gives new users limited free usage of many services for 12 months, introducing them to the platform's value at no cost. This combination of granular per-unit pricing with strategic discounts has enabled AWS to capture a massive market share [34]. It essentially proved that a pure consumption model at scale can not only work, but be highly profitable – AWS revenue scales directly with customer usage of cloud resources.
- **Snowflake** – *Data Warehousing.* Snowflake is a cloud data platform that surged in popularity with its usage-based model. Snowflake breaks its billing into two main metrics: **compute credits and**

storage. Compute credits are consumed based on the size of the compute cluster and the duration of queries/operations – effectively charging for CPU/time. Storage is charged per terabyte stored per month. Data egress (transfer between regions) also incurs usage fees. Importantly, Snowflake's compute credit price varies by edition (standard vs. enterprise edition have different $/credit rates), but within an edition it's purely pay-for-what-you-use [18]. This allows a tiny startup to use Snowflake at low cost (perhaps running only occasional small queries), while a large enterprise running heavy daily workloads might spend millions – yet each is paying proportionally for the resources they actually consume. Snowflake's impressive financial performance (e.g. 100%+ annual growth at points and very high retention) is often attributed to this model where customers easily increase spend as they ingest and query more data over time.

- **Microsoft Azure** – *Cloud Services.* Azure, like AWS, predominantly uses consumption-based pricing. One example is **Azure Virtual Machines**, which are billed per minute of uptime [35]. If you run a VM for 6 hours, you pay for 6 hours; if you turn it off, you stop incurring charges. Azure also offers reserved VM instances for 1 or 3 years at discounted rates and **spot VMs** for transient workloads at deep discounts [36]. Another interesting Azure example is **Azure Logic Apps**, which has a tiered pricing model where the unit cost of each action execution *decreases* at higher volume brackets [37]. This is a form of bulk discount implemented within a usage model – encouraging higher usage by making it cheaper per unit as you cross certain thresholds. Overall, Azure's rapid growth in the enterprise cloud market has been aided by the appeal of pay-per-use pricing (only pay for what you need, no wasted on-premise capacity), combined with flexible options to reduce cost at scale.
- **Twilio** – *Communications API.* Twilio provides APIs for SMS, voice calls, and other communications, and has always charged on a **per-message or per-minute** basis. If your app sends a text via Twilio, you might pay e.g. $0.0079 per SMS in the US. There's no monthly fee required – you can literally spend fractions of a cent if that's all you use. Twilio also offers **volume pricing** (the price per SMS drops if you send huge volumes) and **commitment deals** (e.g. commit to a certain spend for lower rates) for larger customers [16]. This fine-grained pricing was a huge enabler for Twilio's adoption among developers; anyone could try the API and spend just a couple of dollars to send test messages. As their usage grew (say, an app that suddenly needs to send millions of notifications), Twilio's revenue from that customer grew accordingly, and the customer still only paid for actual messages sent. Twilio's success in becoming a core infrastructure piece for many software products (as a communications layer) can be largely credited to this low-friction, usage-based business model which made communications billing scalable and predictable for developers.
- **Stripe** – *Payments Platform.* Stripe's pricing is a hybrid of **usage-based and value-based**: they charge primarily a percentage fee on each payment transaction processed (e.g. ~2.9% + 30¢ per credit card charge in the US). This is effectively a **per-transaction usage fee** that scales with the number of payments (and their size). Smaller products pay only a small amount when they make a sale, and big businesses processing millions in payments will pay Stripe a lot more, but in proportion to their own revenue. The consistency and simplicity of Stripe's per-transaction fee (with only a few factors like card type affecting it) has been cited as a key to its adoption [17]. There are no monthly gateways fees or setup costs – you integrate and pay as you use it. Stripe also has subscription billing services, but its core payments business is usage-priced, aligning well with customers' interests (if the business has no sales, they pay Stripe nothing; when they grow, Stripe earns more).
- **Clearbit** – *API Data Enrichment.* Clearbit's model shows usage-based pricing for a SaaS API service. Clearbit enriches lead or customer data via API, and **charges per API request** (or per credit, with each credit roughly equating to one lookup). For example, if you enrich 1,000 customer records, you spend 1,000 credits. This directly ties cost to how much data you pull. Clearbit typically sells packages of credits and allows them to be used flexibly, which is

essentially a prepaid usage model. This way, a new customer might buy a small bundle to start (low commitment) and a large enterprise might buy millions of credits at bulk rates. The value to the customer (insights for each record) is directly linked to each API call, so they can gauge ROI easily. The *"pay-per-API-call"* model is now common for developer-focused SaaS APIs, from email validation services to mapping and location APIs (Google Maps API, for instance, charges per map load or geocoding request). Clearbit's success in landing many B2B customers stems from the flexibility of this model – integrate the API and only pay for what you use, scaling up as your usage (and presumably your own business) grows [38].

- **Zapier** – *Workflow Automation SaaS.* Zapier uses a blend of subscription and usage charges. It offers **pricing tiers that include a certain number of tasks (automation runs) per month**, and if you exceed those, you pay an overage fee or move to a higher tier. For example, a "Professional" plan might include N tasks per month; using more than that triggers additional charges on a per-task basis [39]. They also provide a free tier (with 100 tasks/month and feature limitations) and a trial of premium features [40]. While the tiered approach provides predictability, the overage mechanism means that essentially customers are paying for each task beyond the included amount – a usage-based element. This ensures that high utilizers of Zapier's automation (who derive more value by automating a lot of work) contribute more revenue, while light users can opt for a lower-cost plan. Zapier reportedly reached over $140M ARR by 2022 with this strategy [41], demonstrating that even in application software, usage-aligned pricing drives growth.
- **Mailchimp** – *Email Marketing.* Mailchimp historically offered both subscription plans (priced by number of subscribers in your email list) and a **Pay As You Go credit system** for emails. The Pay-as-you-go option lets customers purchase email send credits (e.g. 5,000 emails) and then use them over time, one credit per email sent [42]. This is attractive for organizations that send emails infrequently or in seasonal bursts – they don't need a continuous monthly plan, they just consume credits when they send campaigns. Mailchimp's free plan (up to 500 contacts and 2,500 sends per month) also introduces users to the platform's value at no cost, after which they can either subscribe or buy credits to send more [42]. The flexibility of allowing true pay-per-email pricing (even if at a slightly higher rate than subscriptions) helped Mailchimp capture users who might otherwise hesitate to commit to a recurring plan. It exemplifies how even in markets where subscriptions were the norm, hybrid models with usage components can address more customer needs.

These examples – spanning cloud infrastructure, data, communications, fintech, automation, and marketing – all demonstrate the versatility and effectiveness of usage-based pricing in different contexts [43]. By charging customers based on actual usage, these companies have achieved massive scale and strong customer loyalty, since their pricing inherently **scales with customer success**. Moreover, they often combine usage billing with clever strategies like free tiers, volume discounts, and prepaid credit options to balance flexibility with revenue predictability. The common thread is that each identified a key **unit of value** (be it API call, credit, task, GB, or message) and built their business model around monetizing that unit in a fair, transparent way.

Pricing Strategy and Customer Considerations

Designing a successful usage-based offering is not just about the mechanics of metering – it also requires thoughtful **pricing strategy and customer communication**. Here are some best practices and concepts for setting up usage-based pricing in a way that resonates with customers and drives business growth:

- **Align Price with Cost (and Value):** A guiding principle is to ensure your usage prices reflect your internal costs *and* the value to customers. If a certain action is computationally expensive

for you (e.g. generating a complex AI model output), it should be priced higher to the customer as well. Conversely, if something is cheap for you to do (like storing an extra few MB of data), it should be inexpensive for the customer. This transparency builds trust – customers feel you're not gouging them arbitrarily. Cloud providers exemplify this by continuously lowering prices on commodities as they achieve economies of scale (passing savings on). That said, value to the customer can sometimes exceed your cost by a large margin, and you can capture some of that margin – pricing is not purely cost-plus. It's okay if some usage has a healthy profit margin as long as it's still a good value proposition for the customer. The key is to avoid scenarios where cheap-to-provide features are priced punitively high. **Be open about pricing** structure and how it ties to underlying factors. If possible, provide a breakdown (like AWS does with its detailed billing reports) so customers can see the link between usage and charges.

- **Simplicity and Clarity:** One risk in usage-based models is over-complicating the pricing with too many metrics or tiers. It's important to keep the pricing understandable. Ideally a customer should be able to answer "If I do X, about how much will it cost me?" without needing a PhD in billing. In practice, this means limiting the number of distinct billable metrics and providing tools like pricing calculators or examples. AWS, despite its complexity, offers a **pricing calculator** for this reason [44]. Many SaaS companies adopting usage pricing publish clear tables or docs (e.g. Twilio's pricing page shows per-text message fees, etc.). Avoid burying important details – make things like minimum charges, free allotments, or overage rates very visible. A transparent model helps customers plan their usage and builds confidence.
- **Free Credits and Trial Usage:** With usage-based services, it's a great strategy to provide new users with some **free credits or free usage tier**. Instead of a time-limited trial, usage-based products often say: "Here's $10 worth of usage free" or "Here are 1,000 free API calls." This accomplishes two things: (1) it lets the user explore the product's value at no cost, lowering the barrier to trying it out, and (2) it **sets the expectation that the service has a real cost per use**. The user is made aware from the start that continued use will require payment once the free credits are exhausted, which psychologically prepares them to convert to paying customers rather than expecting the product to be free indefinitely. OpenAI did this effectively by giving new users free credit to use their GPT API, after which developers were willing to pay as they had already seen the value. Many of the companies mentioned (AWS free tier, Zapier free tasks, Mailchimp free plan, etc.) employ a free usage allowance [40] [42]. The **conversion rate** from free usage to paid can be high if the product demonstrates its usefulness during the trial period. Ensure that converting from free to paid is seamless – for example, auto-charge the credit card on file once free credits are used up, or prompt the user to top-up with minimal friction.
- **Tiered and Volume Pricing Options:** While pure pay-as-you-go is simple, offering **volume discounts or tiered plans** on top of it can capture a wider range of customer preferences. Many enterprise buyers still like the idea of a committed plan or at least knowing their bill won't exceed a certain amount. You can address this by providing *hybrid models*: for instance, a customer can pre-commit to a certain monthly minimum usage to get a better unit rate (like buying a bucket of usage at discount). This is akin to AWS's reserved instances or purchasing credits in bulk at a cheaper rate. It rewards larger customers with economies of scale, which they expect. You can also have pricing tiers where beyond a certain usage, the incremental rate drops (graduated pricing). This way, the biggest customers automatically get better pricing, encouraging them to send even more volume through your service. Such structures should be designed so that **on-demand (no commitment) pricing is the highest unit cost**, and any commitment or high-volume gets a comparatively lower unit cost [33] [35]. This not only incentivizes more usage, but also ensures you're not leaving money on the table with small customers who are willing to pay a premium for the flexibility of no commitment.
- **Continuous Pricing Experimentation:** One surprising aspect of usage-based pricing is how much it can be tweaked and refined over time. You might discover that certain features are incredibly valuable to customers, and you can charge for them separately, or that a particular

usage metric is causing confusion and needs simplification. Don't be afraid to evolve your pricing model. For example, you could start with a single metric (say, API calls) and later introduce another billable metric for a premium feature (like data storage or number of projects). The voice memo suggests being **ready to experiment and change pricing**, as long as you remain transparent with customers. If you do change pricing for existing customers, best practice is often to grandfather their old rates for a period or give ample notice. But for new customers, you can certainly adjust as you learn. Use analytics: look at the distribution of customer usage and revenue. Are 20% of customers contributing 80% of usage? If so, are you comfortable with that revenue distribution? Do some usage types correlate with higher ROI for the client? You might adjust pricing to better capture that. The usage data you collect can guide these decisions scientifically.

- **Ensuring Profitability:** A well-designed usage model should ensure that **each unit of usage is profitable (or at least contribution positive)** for the provider. This often means engineering the service such that costs scale sublinearly or efficiently with usage. For example, using a serverless architecture can make much of your own cost base variable; you pay your cloud provider per request, and you charge your customers per request with a markup. Ideally, your fixed costs (e.g. engineering salaries) are covered by a base level of usage across customers, and beyond that each additional unit used by a customer has an associated cost that is lower than the price you charge for it. By architecting for minimal idle resources and leveraging platforms that allow elastic scaling (and shutting down of unused capacity), you can maintain healthy margins on usage. It's wise to model out your **unit economics**: if a user performs one billable action, what is the revenue from it and what are the direct costs (compute, etc.) to service it? This should remain in a favorable ratio. If it doesn't, you either need to adjust pricing or reduce costs (or perhaps that action shouldn't be offered as part of the product). The consumption model forces this discipline but in a good way – it prevents scenarios where a few users silently burn your server budget. Instead, you'll see it reflected in high usage and correspondingly high billing, or if not, you know something needs adjusting.
- **Customer Education and Support:** Finally, when rolling out usage-based billing, invest in educating your customers on how it works. Clear documentation, FAQ answers ("What happens if I run out of credits?" etc.), and responsive support are important, especially early on. Some customers might be wary of variable bills – you can alleviate this by offering budgeting tools (e.g. the ability to set a usage cap or receive alerts). Emphasize the **value proposition**: usage-based means no large upfront commitment, and it means if they use less, they automatically save money. Share case studies of how customers like the flexibility. Over time, as usage-based pricing becomes the norm across SaaS, this will be less of an issue, but currently there's still a mix of models in the market, so you want to be sure your customers fully understand and feel in control of their usage and spending.

Conclusion

Usage-based billable entities represent a powerful shift in SaaS business models – one that ties the success of the provider directly to the value delivered to the customer. By defining clear units of usage and charging accordingly, SaaS companies can create pricing that is fair, transparent, and scalable. Customers pay for what they actually need, avoiding the bloat and waste of oversized subscriptions. In turn, vendors enjoy more robust revenue growth as they capture expansion automatically through increased usage, and they build stronger retention as customers stick around and grow rather than churning to cut fixed costs [5] [13].

Transitioning to a usage-based model does require effort – from metering infrastructure to thoughtful pricing design – but the benefits have been proven in the market. Companies like AWS, Snowflake, Twilio, Stripe, Zapier, and many others have achieved massive scale and customer goodwill by

embracing consumption-based pricing [45]. These examples show that usage-based models can succeed across infrastructure, platforms, and end-user applications alike, as long as the **billable units align with real value** and the system is implemented with reliability and customer trust in mind.

For SaaS providers considering this model, the key concepts outlined in this paper – from identifying your value metrics and usage primitives, to building a solid billing pipeline, to iterating on pricing and policy – will help in crafting a strategy. Importantly, usage-based pricing is not "set it and forget it." It's a dynamic part of your product-market fit that you can refine over time. But done well, it creates a virtuous cycle: customers see direct value for cost, use the product more, derive even more value, and the resulting revenue fuels further product investment.

In an era where software value delivery can be measured in real time (API calls, computations, bytes, etc.), it is only logical that **pricing follows suit**. Usage-based billable entities provide the mechanism to do so. By charging in accordance with usage, you as a provider are essentially saying to customers: *"Our success grows with your success."* This alignment builds a healthier, more sustainable relationship for the long term. As the SaaS industry continues to evolve, usage-based pricing is poised to become even more prevalent, driving the next wave of growth for both companies and their customers [45] [46]. In summary, adopting usage-based billable entities is not just a pricing change – it's a strategic shift that places customer value at the center of your business model, where it rightfully belongs.

Sources: This white paper is based on insights from industry research and the authors' analysis, including Dinis Cruz's strategy at The Cyber Boardroom and public data on SaaS pricing trends. Key references and further reading on usage-based pricing include OpenView Partners' reports on SaaS metrics [5] [8], Stripe's guide to usage-based billing [2], and case studies of successful implementations [23] [45], as cited throughout this document.

---

[1] [5] [6] [7] [8] [13] [22] Why SaaS companies need usage-based pricing - DigitalRoute
https://www.digitalroute.com/opinions/moving-from-subscription-based-pricing-to-usage-based-pricing/

[2] [3] [11] [26] [27] What is usage-based billing? | Stripe
https://stripe.com/gb/resources/more/usage-based-billing-explained-how-it-works-and-how-to-optimize-its-benefits

[4] [9] [10] [12] [14] [15] [16] [17] [18] [19] [20] [21] [23] [24] [25] [32] [33] [34] [35] [36] [37] [38] [39] [40] [41] [42] [43] [44] [45] [46] Lago Blog - SaaS Usage-Based Pricing Models: 8 Best Examples
https://www.getlago.com/blog/usage-based-pricing-examples

[28] [29] [30] [31] Lago Blog - Prepaid Credits vs Progressive Billing for Usage-Based Billing
https://www.getlago.com/blog/prepaid-credits-vs-progressive-billing-for-usage-based-billing