Cloud cost unit economics

WHITEPAPER

Dan Bode
Sr. Director of Cloud Advisory

ust.com

Table of contents

Overview

The rise of cloud computing threw a wrench into how companies understand their profitability. On-demand pricing, real-time variable costs, multi-tenancy, complex pricing models, and hidden costs all require a new way of thinking about spending. This is made worse by the fact that in many organizations these costs are abstracted from the business and controlled by IT.

Whether you are a global technology company or a small family-owned convenience store, it is imperative to understand the per-unit costs for every item that you sell. This is unit economics. Understanding unit economics allows a business to better predict gross profit margins and break-even points, which are critical to making good business decisions.

Cloud Cost Unit Economics, part of the six principles defined by the FinOps Foundation, ensures that we can drive cloud spending directly to the business value that it creates. This can de-mystify cloud spend and let an organization realize the full value and business advantage that cloud computing promises.

It’s all about the data

To best understand unit costs, we need to generate as much data related to cost with as much context as possible. This data must be mapped back to the part of the organization that generated the cost, which could be a customer-facing application or an internal shared service. The process of identifying an enterprise’s technical products and what capabilities those services provide is part of setting up an internal **Service Catalogue**. For this paper, I will assume that work has been done and that all cloud spending can be attributed to a single **ProductId**.

This cost data and context are imported into a business intelligence tool where analysis can be performed. Since cloud cost information is stored as time-series data, we should consider what other time-series data we can correlate to costs.

Tracking cloud account costs

Partitioning each cloud account to a single **ProductId** is much simpler than figuring out how multiple products provisioned in the same account share costs. However, managing more accounts requires that we have automation in place for not only creating accounts, but also ensuring that all resources in each account are properly tagged with the relevant **ProductId**.

NOTE: While attributing each cloud account to a single **ProductId** may not be the right solution for each organization, it is worth considering as a principle since it is the easiest way to associate cloud costs. It also simplifies cloud account authorization and aligns with the goal of least privileged access (which is slightly out of scope but still worth mentioning).

Average cost per customer

Now that all cloud accounts emit cost data associated with a single product, we can start solutioning for Cloud Cost Economics. For this example, we will assume a customer-facing product running in AWS that we will label as 'A' and want the cost per customer as our unit cost. To calculate the cost per customer, we need the number of customers and the total cloud spend for our **ProductId**. The following tables show our cloud costs for all accounts associated with our product and customer counts.

ProductId	environment	April cloud costs
A	Dev	$500
A	QA	$1400
A	Prod	$1200
A	total	$3100

ProductId	Number of customers in April
A	80

The equation below shows a one-time calculation based on the unit measurement for customers.

(500 + 1400 + 1200) / 80 = 38.75 per customer

While a one-time cost calculation is a great starting point, it has limitations:

- It does not inform us if our cost per customer increases/decreases as we scale the business
- It does not consider cloud cost for shared services that product A depends on
- It assumes that each customer is contributing to cost equally

Observability data

Metrics and Distributed Tracing are two types of observability data published from a running application to an external data source as time-series data. Because metrics, trace, and cloud cost data are time-series and allow arbitrary labels to be set (like **ProductId**), observability data can be joined with cost data to provide additional context.

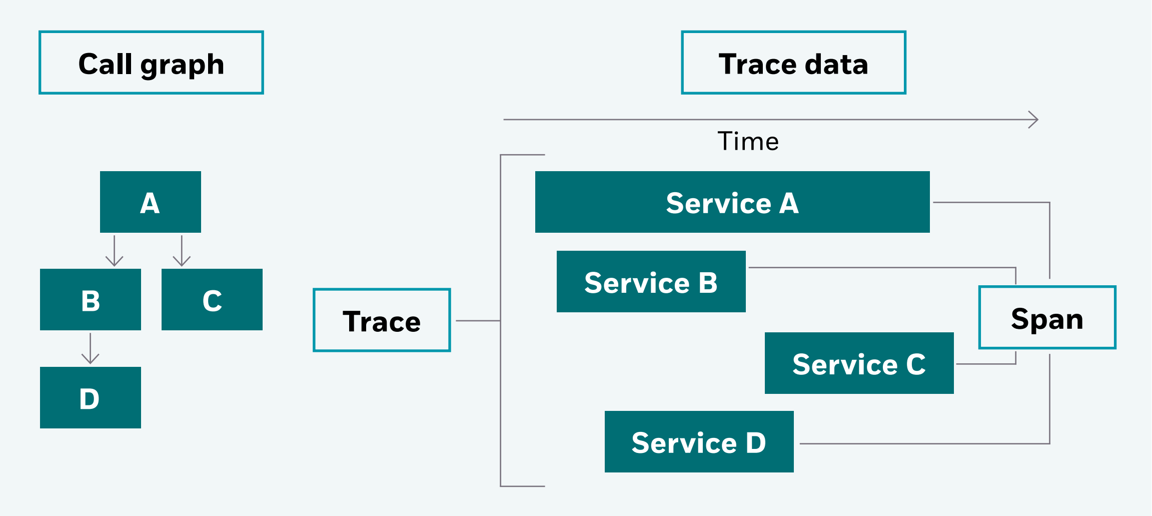
Metrics are time-series data published directly from application code. Each point (i.e., point in time) has a name, value, value type (counter, gauge, histogram) and arbitrary labels. For more details on metrics, I recommend looking at open telemetry since it is evolving as the industry standard.

Distributed Tracing is built on a set of standard headers that are propagated through service calls to construct a call graph. It provides us with a way to know, for each request, how the request was routed through multiple systems and how much time it spends in each system. Distributed Tracing has two parts:

Span – A unit of work where a single service has processed a transaction. Each span has a link to its parent span and a set of labels that can be assigned to it. If a span does not have a parent, it is the originating request.

Trace – A collection of spans.

In the diagram below, the left side shows a call graph showing how requests flow through multiple services. The right side shows how that data is expressed via spans and traces. You can see how each span measures the time spent to fulfill the requests in each service.



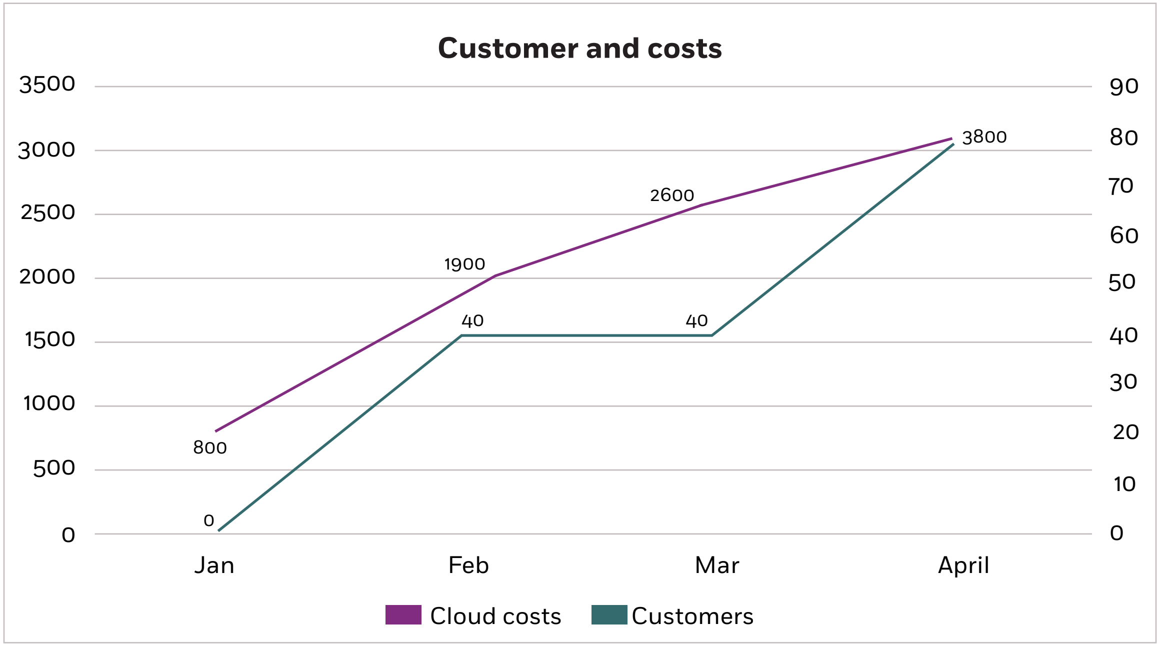
If you have not set up central observability in your organization, consider it part of your work toward Cloud Cost Unit Economics. It may even be easier than you expect. Tools like Istio and eBPF are evolving in the observability ecosystem to perform auto-instrumentation at the infrastructure layer (meaning that you can implement it once in your infrastructure layer without making code changes). Prometheus is becoming a standard format for creating metrics for many language ecosystems, rapidly increasing the number of existing metrics without requiring development efforts.

Costs per customer over time

Now that we understand how to use observability data to provide context to our cost data for analysis, we can publish a counter metric to track the number of active customers. This will help us understand how the number of customers impacts cloud costs over time. Consider the following cost and metric data.

ProductId	Environment	Costs			
		Jan	Feb	March	April
A	Dev	$200	$300	$400	$500
A	QA	$200	$800	$1400	$1400
A	Prod	$400	$800	$800	$1200
A	Total	$800	$1900	$2600	$3100

ProductId	Metric: Number of Customers			
	Jan	Feb	March	April
A	0	40	40	80

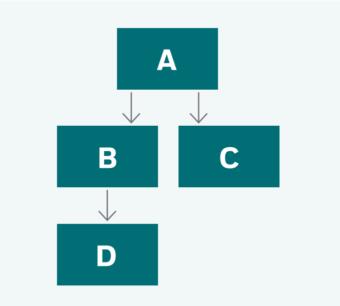


As you can see, there is a correlation between the number of customers and cost data, but it is not direct. Just a few notes about the data and resulting graph (and to show how complicated it can be), only some data is directly associated with the number of customers. In the above example, the costs related to Dev grow as our number of developers grows, and the costs related to QA grow as the end-to-end test suite is expanded. This complexity is precisely why I recommend getting the data into a business intelligence tool where it can be analyzed. For example, if we wanted to try to predict costs for May (and assume that we can break the numbers down per hour), we could use linear regression.

Shared services

So far, we have only looked at cloud costs of the accounts directly tagged with the **ProductId** for our customer-facing service. As usual, the reality is more complicated. It is common for multiple businesses with different cost units to leverage the same set of shared services. To understand these costs, we need to visualize our organization's service topology map to know how traffic flows across all services. As mentioned, implementing distributed tracing across the organization allows us to explore that service topology map.

Let us assume that we have implemented distributed tracing across our organization, and we can see the following service topology map for service A. To better understand the total cost of service A, we need to understand how it drives cost into B, C, and D.



The easiest way to calculate how much cost A drives into B and C is to take the total cost of B, C, and D and then divide them by the total number of services sending requests to them. Our distributed tracing data can provide this information.

For example, if service B has three other services calling it (A, E, and F) and total cloud costs of $900 in April, then we can assign $300 in cost to each service.

Adding this shared service cost to our total for A gives us a more accurate cloud cost per customer. However, we already have a problem. The owner of Product E feels he is stuck with a bill that is not proportional to his usage. Why is he responsible for 1/3 of the costs of B when he is sure he is responsible for less than a third of the traffic?

Not only does the distributed tracing data show how traffic is routed between services, it shows you how long each service spent processing each request. Looking at the data below, you can see a couple of options for how service B might want to do a chargeback to build a more accurate view of cost, either by number of requests or total wall clock time spent serving requests.

Request data for service B		
Calling Service	Number Spans from service	Accumulated Time of all Spans (in seconds)
A	1000	1100
E	910	1500
F	10	20

NOTE: This is just one example of how we can attribute costs that can help divvy out chargeback. If storage is the main factor driving cost, it makes sense to chargeback based on proportional use of total storage.

External SaaS (Software as a Service) costs

Although slightly out of scope for cloud costs, external SaaS products can be significant cost drivers that impact your cost-per-unit calculation. In my experience, costs related to the storage of logs and metric data can be one of the biggest expenditures after cloud costs.

To effectively calculate SaaS costs, you should understand how the vendor charges for usage (egress, storage, query) and ensure that each vendor can associate your costs back to multiple **ProductIds** so that you can effectively charge back. Many vendors have a concept of an organization and a way to create multiple access tokens within an organization where you can see cost data either per org or per token. Procurement governance should be in place to ensure that the external SaaS products you use align with your chargeback strategies required for accurately dividing costs.

Not all customers are created equally

So far, we can attribute cloud costs, shared service, and external service costs per technical product. Once we have these costs, we divide them by the number of customers and attribute the average cost to each. Like other measurements, we can start by applying the average cost, but we should also ask if there is a more accurate way to represent costs. Can we find a way to categorize our customers to determine if they are driving more costs into the system?

By adding customer context to our trace data, we can understand which customers are sending more requests to our system, the status code for those requests (like, are they respecting 429 retry headers), and how long each span runs for. This time-series data can help us build a much more fine-grained understanding of how individual customers are driving costs. There may be straightforward ways to categorize customers based on how they drive costs on your system. For example, customers using the APIs (application interfaces) may drive more cost than those using the UI (User Interface).

Product Owners use Customer Analytics Tools to supply insights into specific customer behaviors. It is worth looking at what tools your organization already uses and if the data generated can be correlated to specific costs. For example, an A/B test may find that new product features are increasing your cost per unit.

Other types of cost

While this paper focuses on cloud spending, this is just one of many costs that should be imported into our business intelligence tool so that the broader picture of cost can be understood. Other costs that need to be correlated to our **ProductId** are:

- Internal/External headcounts
- Data Center costs
- Licensing Costs
- Other Shared Services (Security, Development Tools)

Conclusions

Cloud Cost Economics allows a business to attribute its cloud spend against the business value that it creates with Unit Economics. Unit Economics is a process where you compare your costs associated with a particular unit of revenue. It might be cost per transaction, cost per ride, or even cost per can of soda.

Once you have selected a unit to measure cost against, figure out what data you can leverage to build the most accurate reflection of cost. The goal is to get as much correlated data as possible into an analysis tool. This requires that you know how to attribute costs and build out the most effective and fair chargeback systems you can.

It is going to be a journey, so think about the easiest way to divide costs (by taking an average) and how you can improve upon it to be as accurate as possible. The more accurate you can be, the better and more informed decisions you can make.

References

https://www.finops.org/wg/introduction-cloud-unit-economics/

https://www.paddle.com/resources/unit-economics

https://prometheus.io/docs/concepts/data_model/

https://opentelemetry.io/docs/specs/otel/metrics/data-model/

https://www.sumologic.com/blog/how-sumo-logic-monitors-unit-economics-to-improve-cloud-cost-efficiency/

https://www.dashcon.io/2018/agenda/rediscovering-the-hidden-capacity/

https://amplitude.com/blog/startup-fundraising-metrics

https://www.finout.io/blog/what-is-unit-economics

https://www.datadoghq.com/product/cloud-cost-management/

Together, we build for boundless impact

For more than 23 years, UST has worked side by side with the world's best companies to make a real impact through transformation. Powered by technology, inspired by people and led by our purpose, we partner with our clients from design to operation. Through our nimble approach, we identify their core challenges, and craft disruptive solutions that bring their vision to life. With deep domain expertise and a future-ready philosophy, we embed innovation and agility into our client's organizations—delivering measurable value and lasting change across industries, and around the world. Together, with over 30,000 employees in 30 countries, we build for boundless impact—touching billions of lives in the process. Visit us at:

ust.com

U ·
S T