A quantitative approach to cloud application architecture

Table of Contents

Introduction *3*

Overchoice *3*

Quantitative modeling *4*

Use patterns *4*

- Continuous 5
- Cyclical 5
- Surge 6
- Variable 6
- Proportional 6

Cloud application architecture patterns *7*

- API gateway pattern 7
- Sidecar pattern 8
- Comparison of the two patterns 9

Conclusions *10*

About the author *12*

Introduction

As cloud providers compete for the attention of IT architects and developers, the options to design, build, deploy, and run business-critical applications are expanding to an unprecedented number. Today, any of about 35 object storage solutions are available across cloud platforms. One leading vendor offers 25 database options, and another platform comprises more than 175 services across a diverse range of database, analytics, storage, machine learning, deployment, mobile, and Internet of Things (IoT) offerings. Selecting the right services from a cloud catalog can feel overwhelming, especially when you consider how cloud resources work together. How does any object storage solution pair with any database in terms of performance, scale, and cost? What is the net effect of integrating a set of services from different providers into a hybrid multicloud architecture?

Overchoice

Faced with so many combinations of resources, architects and developers suffer from overchoice, a term coined by Alvin Toffler in his book Future Shock to describe a cognitive process in which human decision-making becomes more difficult as the number of choices increases. The wider the variety within the assortment to select from, the more work is required to make an intelligent decision. To tip the scales in either direction, people must weigh options against each other, and the differences between them must be meaningful. But in an overchoice environment such as a large cloud services catalog, an abundance of non-optimal choices complicates decisions. Architects and developers struggle with the uncertainty that somewhere in the vast range of alternatives that the cloud provides, a better solution exists. The pressure to choose wisely increases as the time squeeze of continuous deployments forces more technical decisions to be made in shorter intervals of time from a growing panoply of options. When the limits of human decision-making are stretched, bad decisions take over and serious mistakes are made. Without guidance, if you rush to decide quickly about complex matters, you're likely to choose poorly.

Overchoice in the cloud market has a cure: quantitative modeling can narrow an overwhelming number of choices into a few simple options. At a basic level, you can model a use pattern as a random variable distribution to simplify technical decisions. In more complex scenarios, the approach can be applied to migration projects to help your team decide how to migrate traditional applications to the cloud and adapt the architecture to changing conditions as applications are migrated. A quantitative modeling approach can help you select, combine, and arrange cloud resources in a way that satisfies known and forecasted demand for services.

Quantitative modeling

A quantitative approach to assess cloud applications architecture begins with an understanding of the variables that underpin the demand for services in the cloud through sampling and measurement. Your goal is to collect and analyze data to predict the fitness of an architecture against nonfunctional requirements. To do so, use quantitative methods to create basic models, such as a line graph, to describe consumer behavior, and conduct a basic statistical examination of the data to gain insights. From there, you can support data-driven design decisions through data analysis, interpretation, and prediction.

Use patterns

Nearly every part of a modern cloud computing platform provides telemetry. No shortage of counters, gauges, meters, statistics, and precise instrumentation exists to measure a variety of data points about a cloud environment. In most environments, plots and graphs are as integral to the platform as metrics. But the interpretation of metrics, what the numbers mean, sometimes falls short of providing architectural insight. While metrics, plots, and graphs are readily available throughout a platform, not much attention is given to how to assess an architecture through patterns that emerge from the data. Recognizing use patterns in the data can help you address these oversights.

In the cloud, use patterns relate to workloads. Just as you can measure central processing unit (CPU) workload by the number of cores that are used as a percentage of the CPU and you can measure database workload by focusing on the number of queries that are run by a database over time, you can measure a cloud computing workload by the amount of use that is given to cloud resources in discrete periods. Give special attention to measuring workload to determine resource utilization because it has a cost associated with it. As a measurement of the amount of use that is given to a set of cloud resources, workload costs from one cloud provider to another can differ substantially. Similar resources cost less to use under one provider than they do under another. But a cloud computing workload might also refer to capabilities. From this perspective, a workload is a portable configuration of resources in support of a business process. In this sense, a workload as a configuration of capabilities can be moved somewhere else to do the same work.

In quantitative modeling, you focus on workloads not in terms of resource utilization or from the perspective of configuring resources in a portable way, but by using traffic data to create models that map into architectures. For example, you might use *request count*, which is the number of client requests that are made to the system over time. In general, you can recognize use patterns from these basic traffic data distributions: continuous, cyclical, surge, variable, and proportional.

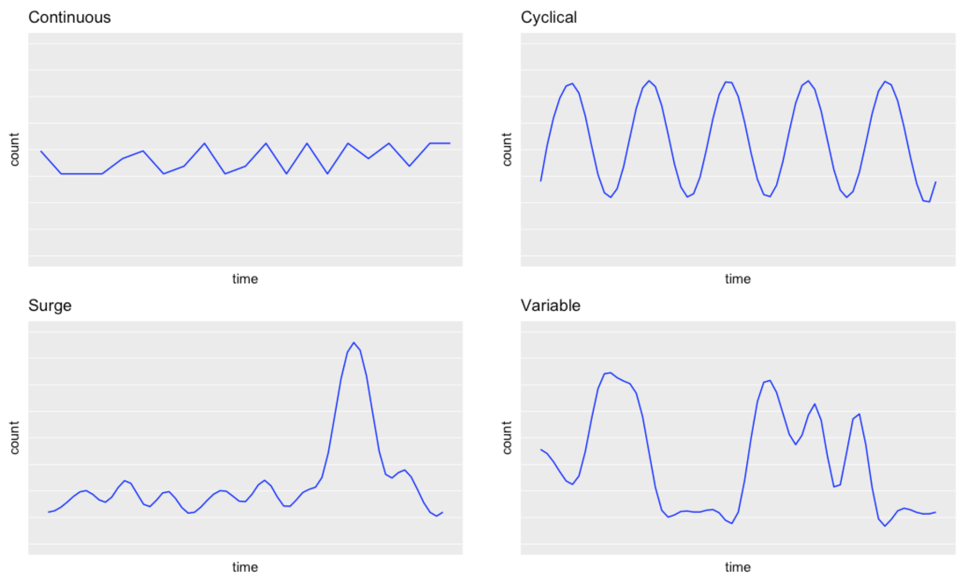

Figure 1. Basic traffic data distributions

Continuous

A continuous traffic data distribution has a request count that remains constant over time. Data points from a continuous distribution of requests for a service form a semi-straight line on a graph whose x-axis represents time and y-axis represents request count. Continuous traffic tends to stay within one standard deviation of a monthly average and typically originates from stable applications that generate a steady stream of traffic.

This kind of continuum comes from applications that poll the server at regular intervals. Polling takes place when an application repeatedly checks for a specific state in the service.

Cyclical

A cyclical traffic data distribution follows a pattern that repeats itself over time. Also called *periodic*, a cyclical traffic pattern is easily recognized on a line graph when you look at more than one period in which request counts repeat themselves. When cyclical traffic is associated with dates on a calendar, it is called *seasonal*. To be used in any practical way for quantitative modeling that analyzes cloud applications architecture, the length of a pattern in cyclical traffic distributions must be no longer than a year. Looking for multi-year patterns becomes impractical given the amount of data to analyze. Cyclical traffic fluctuates in four phases: peak, decline, trough, and expansion. For example, business processing that is associated with quarterly events might produce a cyclical model.

Surge

A surge traffic data distribution peaks over a short period and then drops. A one-time load on the system or a sudden increase in the demand for its services results in a surge. In a surge, request count grows at an increasing rate, peaks, and then declines at an increasing rate until it normalizes. Spikes in request count are brief, short lived, and highly noticeable on a graph. A two standard deviation move or higher above a monthly average is considered to be a surge. Unless they're accounted for, events that correlate to sharp, sudden traffic increases can be highly disruptive. A large data transfer to start a service typically produces a surge distribution model.

Variable

A variable traffic data distribution randomizes request count over time. In contrast to continuous or periodic traffic, variable traffic lacks a pattern and is, by definition, unpredictable. High variability in service demand presents challenges to the architecture that come from the difficulty to prepare for random events. At best, outliers in the data (lowest and highest request counts) might represent a minimum and a maximum that the system can be designed to handle. The unpredictable rise and fall of request count over random time periods is an indicator of instability in the demand for services. You might observe this behavior in the early stages of a new offering or product.

Proportional

A proportional traffic data distribution grows or decays along a linear, polynomial, or exponential curve. Linear traffic increases at a constant rate based on request count that changes over time by a fixed multiple. On a polynomial curve, traffic increases or decreases by a constant exponent. When traffic becomes polynomial, request count grows much more quickly than linear traffic. Under extreme expansion, exponential traffic doubles at every step, or period, that is measured. Exponential growth in traffic is not noticeably higher than linear or polynomial in the early stages. But when traffic becomes exponential, it's critical to notice in the early stages of growth, as traffic doubles at every step.

A quantitative approach to cloud applications architecture depicts a curve as a model. Data modeling changes the way that architects think about cloud application design problems because data analysis moves from being an afterthought in performance testing to validate scalability in an architecture to being a key driver of architectural decisions. Because a curve as a quantitative model describes an underlying business process, you can define architectural goals from the intrinsic features of a model. For example, a line graph model of traffic data provides insights that can answer key questions:

- What proportion of the business is exposed to a service that is on the receiving end of continuous traffic?

- Is the business in the initial stages of exponential growth and headed into rapid take up?
- Do some services differ in their rates of growth, and how is traffic changing between them?
- Does the system show diminishing returns as the benefits of adding resources fall away quickly (the more you put in, the less you get out)?

Cloud application architecture patterns

Consider a scenario in which a data-driven approach to cloud applications architecture can be integrated into an architectural design process. The following assessment reviews the popular API gateway pattern in relation to the Sidecar pattern. For this example, assume that the project has traditional services that are decomposed into microservices and deployed as containerized applications into Kubernetes.

API gateway pattern

Commonly used in microservice architecture, the API gateway pattern is adopted to hide the complexities of accessing microservice APIs. The gateway pattern shields consumer applications from a complex backend by exposing a single point of entry into the system. Generally, if a system exposes no more than one API, applications easily locate it, consume responses from it, and make adjustments for version changes as the one API evolves over time. But in cloud computing, APIs proliferate. As the number of APIs grows, consumer applications must locate and route calls to new endpoints, handling the differences in versioning and protocols between them. Without a gateway, the logic to interface with APIs replicates across consumer applications and application code becomes an area of maintenance challenges.

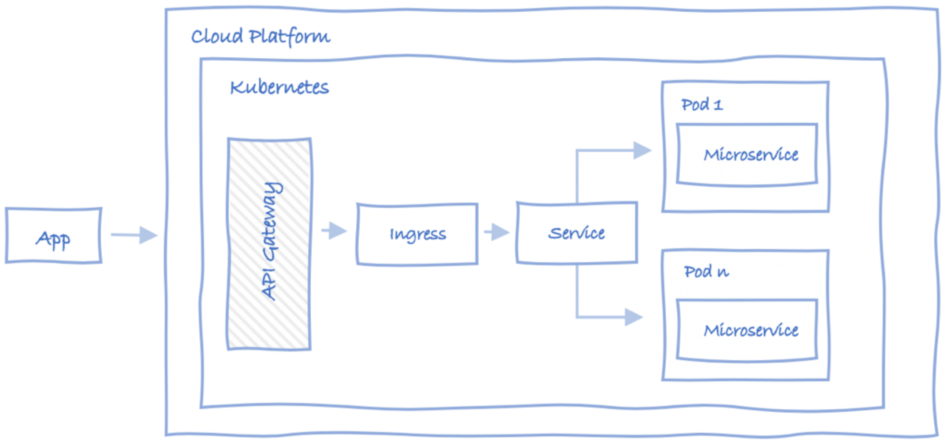

Figure 2. API gateway pattern

To solve this problem, the primary role that is played by the gateway in the system is to encapsulate all API access logic and become a common entry point that applications make API calls through. The gateway might also play a critical role in addressing common backend concerns such as authentication, authorization, logging, telemetry, and response scanning. Without a gateway, the logic to handle these common concerns ends up scattered throughout API implementations. When all traffic flows through a single point in the system, the gateway centralizes common concerns as it validates all requests uniformly on the way in and verifies responses consistently on the way out.

Sidecar pattern

The sidecar pattern is popular in service mesh environments as an alternative to a gateway. While a gateway sits in front of a complex backend and mediates between all consumer applications and microservice APIs, a sidecar colocates common concerns within each Pod that runs a microservice. In a service mesh, authentication, authorization, logging, and telemetry are distributed into sidecars that are attached, but not tightly coupled, to a primary microservice.

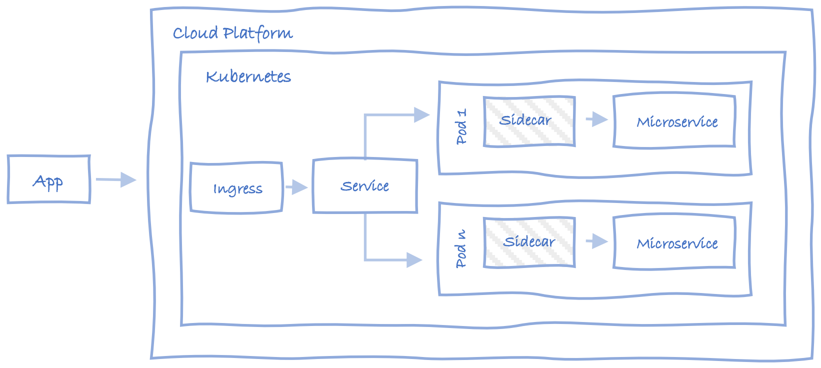

Figure 3. Sidecar pattern

This distribution of common concerns provides resilience in the system at run time by removing a single point of failure that a gateway represents in the architecture. If the gateway fails, the entire system suffers. The sidecar in a service mesh, which is also called a *proxy,* isolates failure so that requests can flow to other parts of the system that are unaffected. Sidecars are typically deployed alongside a microservice and can connect to and communicate directly with the microservice that they're attached to. Optionally, sidecars can communicate with a service mesh coordinator to accept, but not require, commands that drive behavior at runtime based on system-wide policy.

Comparison of the two patterns

To compare these two architectural patterns, look at the data through a quantitative lens. Traffic data for the traditional services reveals that some services are on the receiving end of a high level of continuous traffic with occasional surges, while most services receive a low level of cyclical traffic. Models that are created from traffic data provide two key insights:

- The underlying processes that drive demand for high, continuous traffic services originate from the same clients and authenticate with the traditional system by using the same credentials.
- The low level, cyclical traffic originates from a wide range of applications (web and mobile) that represent usage by a wide number of different users that authenticate with different credentials.

Given the insights from the data analysis and all things being equal, the demand for traditional services after they're decomposed and migrated into a microservices environment in the cloud is not expected to change. Traffic shifts from the traditional system into the cloud. If a high level of continuous traffic that includes occasional surges is routed through a single point of entry for all APIs, the shift places a disproportionate load on the gateway.

In this scenario, the gateway requires extra Kubernetes cluster resources such as container instances, CPU units, and memory to scale into traffic for specific services only. In addition, the common concerns that the gateway addresses, such as authentication and authorization for this portion of the traffic, become a series of duplicate security checks for the same set of credentials. Looked at quantitatively, the fitness of the gateway architecture is expected to be inadequate for the combination of use patterns that are identified in the data analysis. Given the data, the gateway architecture has the potential to introduce risks that include a negative impact on a portion of business-critical traffic that originates from a broad range of users.

Based on the use patterns, a quantitative approach to architectural decision-making favors the sidecar as a better fit for high-level, continuous traffic services. In this scenario, the extra cluster resources that are required to scale into the demand are allocated only to the specific services that require them. This decision has a more focused impact on cost in the cloud that is associated with the added resources, as the upgraded services can be optimized for cost and performance independently on an ongoing basis. The data analysis provides support for a decision to bypass the gateway for high, continuous traffic and to direct traffic through the sidecar/proxy instead to address common back-end concerns.

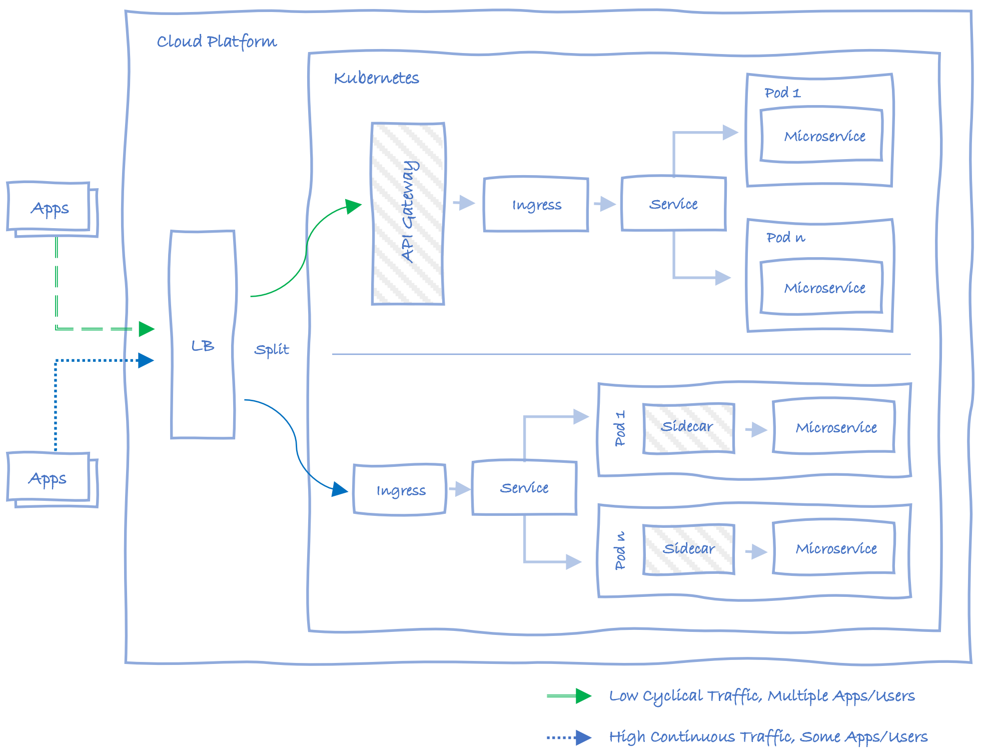

Figure 4. Hybrid architecture

Because low, cyclical traffic requires a broader range of security checks, such as authentication and authorization, you can anticipate that a hybrid architecture in which low, cyclical traffic flows through the gateway is a better fit for the use patterns in the analysis. A hybrid architecture that splits traffic through a load balancer (LB) and combines the use of a gateway and a sidecar sets up the system to scale proportionally for the two different traffic distributions.

Any resource that is allocated on a cloud platform is likely to exist in support of a business process in one way or another. Object storage, databases, IoT applications, messaging systems, and other resources in the cloud can often be combined in complementary ways, but in some cases, they provide functions that compete to solve problems in similar ways. In these cases, a quantitative approach takes the guesswork out of complex decisions that involve a wide range of options. The approach compares options in an objective way and differentiates alternatives by putting evidence forward in support of an architectural decision.

Conclusions

Quantitative modeling applies to business processes at every level: in research and development, in production planning and scheduling, in purchasing and inventory, and in marketing and finance, to optimize the allocation of money, labor, materials, machines, and time. At a fundamental level, metrics measure the business processes that matter most. One of the most fundamental processes that focuses the attention of a business is sustainable growth of customers, profits, and market share. As such, models that represent sustainable growth characterize the most intrinsic goals of a business. In cloud architecture, as in other mission-critical processes of a healthy business, quantitative modeling provides the formal description that you need to understand the variables that contribute to sustainable growth in the cloud.

Resources that are optimally allocated in the cloud support business growth in major ways. But as clouds grow, architectural flaws sometimes become apparent in the worst ways. Even when architecture follows accepted principles, best practices, and common design patterns, components are generally hand-picked with good intentions. Flaws in those selections reveal themselves when an implementation of an architecture fails as it is put to the unforgiving tests of the real world. Engineering teams scramble to stabilize a system by resetting it to an initial state while the variables that contribute to instability continue to operate beneath the surface.

Without a metrics-based model for a use pattern to evaluate early in the design process, cloud architecture remains disconnected from reality. And without an understanding of the relationship between variables that exert pressures on a system, engineering teams lack a well-defined model to analyze systemic issues. As a result, a standard for an architectural design process that misses quantitative modeling as input into design increases the probability of failure and exposes the business to unrecoverable risks.

To move beyond these limitations, cloud computing holds the greatest advantage over traditional IT in the dynamic allocation of resources to support the changing needs of the business. The ease with which resources are allocated in the cloud sets the stage for an architectural approach that relies more on data and analytics than on instincts, hunches, and hand-picked components. Data-driven architecture asks teams to pay more attention to metrics as critical input into design and to think about design problems based on the available data. A data-driven architecture that favors the highest probability of success and the lowest probability of failure increases confidence in architectural decisions that support business growth.

To be fair, metrics can’t tell the whole story no matter how you model them. Metrics can’t reasonably be expected to provide a basis for the right way to solve any problem in any situation. This limitation goes beyond accuracy in numbers; some critical aspects of cloud architecture are too difficult to measure. For example, quantitative modeling doesn’t take into account that a suboptimal, more expensive solution might support the best short-term tactic to be quick to market and continue on a path toward a longer-term strategy to reduce costs and scale in the future. Simple models can’t tell you that.

A quantitative approach to cloud architecture doesn’t replace long-term strategic thinking. Instead, quantitative thinking becomes a powerful enhancement to an architect’s repertoire of analytic skills that integrates data science and cloud architecture to make better decisions. As Alvin Toffler defined it, *overchoice*, the problem of selection from too many alternatives, becomes more acute with time. As competition for market share by cloud providers intensifies, the alternatives across cloud provider catalogs expand. In such an environment, good architectural decision-making depends on your ability to narrow a range of non-optimal choices into an optimal combination of cloud resources.

About the author

Alex Rodriguez is an experienced software architect at IBM Security Services. Alex is mainly interested in cloud application architecture, microservices, and quantitative methods as they apply to decision-making on software engineering projects. You can reach Alex at arodrigu@us.ibm.com.