Google Cloud

Designing Cloud Teams
How to Build a Better Cloud Center of Excellence

April 2024

Table of contents

Executive summary

The faster your IT organization can migrate your workloads and modernize them in the cloud, the shorter your time to value. And yet, we observe our fastest customers outpacing our slowest customers by an order of magnitude (10x) when migrating similar technical solutions. Why is that? The technical skills of your staff play an important role for sure, as do the regulatory requirements of your industry. Still, they don't add up to explain the up to ten-fold difference in speed.

The single biggest lever to accelerating your time-to-value is the configuration of those IT teams whom you are trusting with the execution of your cloud strategy. Said differently: before you think about how to rearchitect your software you should think about rearchitecting your teams.

The standard approach to addressing this concern is to create a *Cloud Center of Excellence*. However, we find that this label suffers from having too many competing definitions across our industry and that, in practice, no such static, single, large, interdisciplinary team exists successfully. Instead, we advocate for the creation of at least two distinct teams:

1x Cloud Office

Mission: drive cloud adoption across the organization by empowering the right teams with the right skills, with the right resources, and with the right KPIs.

KPIs: VMs migrated, employees trained/hired, timeline accuracy, cloud cost % saved, etc.

1+ Cloud Platform Team(s)

Mission: build the shared tools and services that enable other internal IT teams (their users) to securely build and run their solutions.

KPIs: user sentiment, user engagement, user productivity

A dedicated **Cloud Office** project manages the implementation of the cloud strategy. This team's success is measured by how accurately it can adhere to the timeline, the scope and the cost estimated when the cloud strategy was finalized and the cloud adoption journey commenced. Its scope comprises eight discrete workstreams, which can be mapped to separate subteams if needed:

- Executive Sponsorship
- Cloud Teams Design
  (the subject of this whitepaper)
- Communication
- Hiring/Recruiting
- Training/Upskilling
- Cost Control/FinOps
- Portfolio Planning/Intake
- Contract Management/Procurement

The Cloud Office proactively drives awareness and amplifies demand for onboarding to your cloud environment across the business and prioritizes inbound demand.

One or more **Cloud Platform Teams** each design, build and operate a single platform like a product and serve the needs of their respective users – the business/application teams. These teams' successes are measured by the user engagement and user satisfaction with the platform. They follow agile principles and DevOps practices and are not beholden to a fixed timeline, scope or budget.

Cloud Platform Teams must build a viable platform for one persona and one critical user journey first, before branching out into broader feature sets. Common platforms in the cloud include, but aren't limited to:

- A foundational cloud platform
  (sometimes referred to as a "landing zone")
- A big data and AI platform
- A container run-time platform, and
- A CI/CD or DevOps platform.

These platform teams should operate in close proximity to their internal customer(s) to develop a deep understanding of their needs and constraints.

Each team can be broken down into discrete technical domains and corresponding subteams, if needed. For the foundational cloud platform team, for example:

- Identity and access management
- Data protection and encryption
- Shared VPC networks
- Architecture blueprints as code
- Trusted images (software supply chain)
- Shared logging and monitoring

The Cloud Office does not instruct the Cloud Platform Teams on what to build into their platforms and landing zones. The Cloud Office merely determines which internal customers should be prioritized by the Cloud Platform Teams.

These teams are not an exhaustive list. Additional enabling cloud teams may be desirable, e.g. when an application team or a platform team would benefit from a dedicated cloud operations team or from special expertise in domains such as security, user interface design or artificial intelligence.

The distinction between these teams is an essential one: they allow each team to stay small (5-10 people), to apply a different mindset (project vs. product), organize themselves in different ways and with different competencies, and are measured and held accountable by different KPIs.

Rather than provide a prescriptive template to how you should organize your cloud teams, we explore five design principles that are based on comprehensive empirical research inside Google and the DevOps community at large. They are:

- Team size (smaller is better)
- Team purpose (projects vs. products)
- Team types (apps vs. platforms vs. enablers)
- Team priorities (personas, user journeys and OKRs)
- Team environments (physical vs. virtual vs. hybrid)

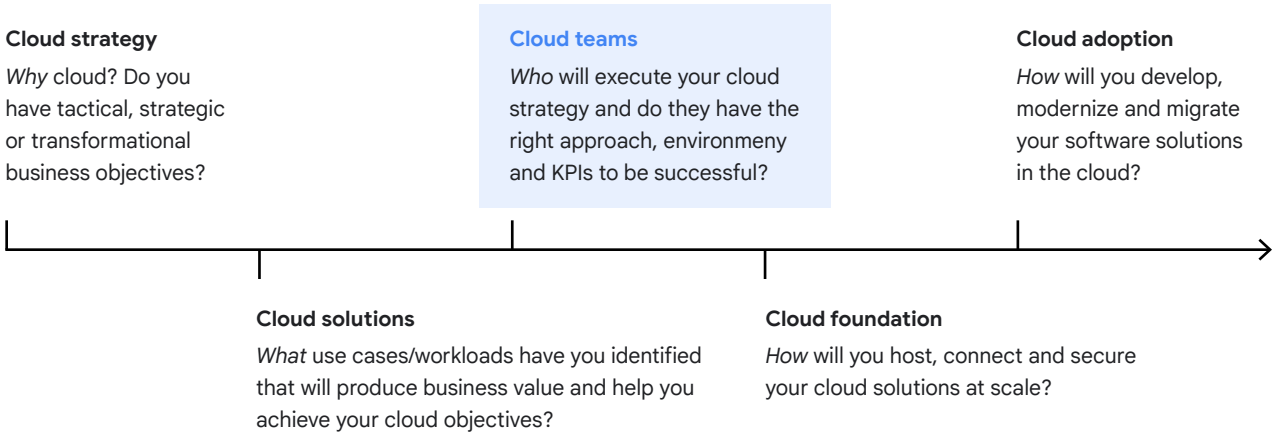
In due time, each platform team running on Google Cloud will learn to practice real-world agile principles, follow DevOps best practices and set up error budgets and conduct blameless postmortems in the spirit of Google's renowned Site Reliability Engineering methodology (SRE).

Your organization's cloud adoption journey provides a once-in-a-lifetime opportunity to transform some of those ways of working, one "cloud team" at a time, as they each move/modernize/invent their respective solutions in the cloud.

Deep dive

Understanding that a one-size-fits-all approach doesn't exist, we provide design principles and building blocks to help you customize your Cloud Teams. By emphasizing dedicated small teams, aligning team purpose and priorities based on user needs and creating collaborative environments, you can accelerate time-to-value while unlocking the full potential of your cloud strategy. This paper assumes that you have formulated your cloud strategy – the why of your cloud journey. We are proposing to effect change only where cloud solutions (the what) will be leveraged and to engage only those individuals who are directly associated with those use cases/workloads.

When to think about designing cloud teams



We define "team" (the who) as the molecular unit where real production happens, where innovative ideas are conceived and tested, and where employees experience most of their work. We are not concerned with your org chart and reporting lines. To build a more agile, cloud-native organization, you may want to consider formally changing your structure later to support, strengthen, and spotlight these new teams, but that is beyond the scope of the Cloud Teams Playbook.

Your Cloud Teams embody the early adoption of a new operating model. They form the template for the rest of your IT organization to follow, in due time, if and when their respective workloads shift to the Cloud.

Team size
(smaller is better)

When an organization faces a complex, novel problem, it can be tempting to throw lots of bright people at it. Add in a deadline, and the urge to create a big new "task force" or "working group" is even bigger. Soon, you'll find that everyone can't fit in the same room, and that there isn't enough time for everyone to speak up. The problem isn't the small room or the fact that everyone wants to share their opinion. The problem is that your team is too big.

A small team size maximizes human interaction and minimizes dependencies and distractions from outside the team. When adopting cloud technologies and ways of working, the majority of your people's time will be spent performing novel tasks and confronting first-in-kind technical challenges that produce a high level of cognitive load. More than time, focus is your cloud teams' scarcest resource. Each cloud team member should be dedicated to the cloud, meaning that they spend at least 80% of their work week implementing your cloud strategy. This ensures that the team is focused on and dedicated to the task at hand.

	Limit team size to max 5 people when	Limit team size to max 10 people when
Proximity	Team members are split across different rooms, different offices or different time zones and don't use a shared chat room	Everyone is physically located in the same room (or neighborhood in an open office) or everyone works remotely and shares a team chat room
Familiarity	Relationships are new and still forming. Each member's strengths and weaknesses and distinct personality traits are not yet revealed	Relationships are already established and trusted. Team members have prior experience working with other members
Psychological Safety	Individual team members hold back on asking possibly silly questions or openly admitting to mistakes in front of their team members	Everyone feels comfortable taking interpersonal risks between team members without fearing repercussions
Mission	The team's mission is unclear and/or objectives are not measurable and don't yet have full buy-in from each team member	The team's mission is clear and familiar to everyone, likely because they have solved a similar challenge in the past together and the objectives are measurable
R&Rs	Roles and responsibilities are not clear to everyone and still fluid. Role titles and RACI matrices are contentious	Each member's role and responsibilities are intuitively clear to everyone and leverage each member's individual strengths
Transparency	Key information is shared only verbally and opportunistically. Internal documents are shared only when explicitly requested and don't allow edit or comments from others	Everyone has easy access to the same information. Key information is always captured for posterity either in a group email or chat room. Documents allow inline commenting

Under no circumstances should a single cloud team have more than 10 members. In fact, when first forming your cloud teams, they should likely have no more than 5 members. Starting small is essential to establish trust, clear communication, and a shared understanding of roles and responsibilities.

This is not to say that your organization's cloud strategy must be fully implementable by no more than ten people. It merely means that you need to create multiple discrete teams, each with no more than ten members. When deciding how to split an existing team in two, consider the following three questions:

Distinct

Is each subteam's name and mission sufficiently clear, so that new work is easily routed to the correct team?

Dedicated

Does each subteam have their own OKRs/KPIs and deliverables and can they achieve them without being blocked by adjacent subteams?

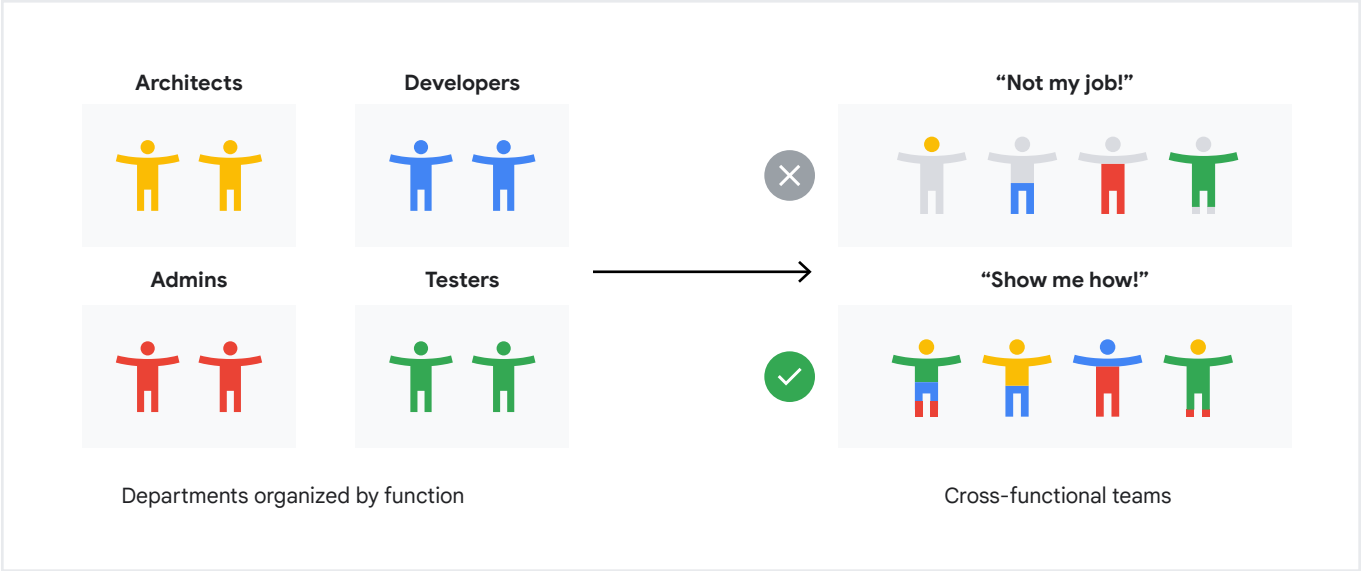
Decoupled

Could each subteam potentially deliver its value abstracted behind a software user interface or an API?

Cross-functionality

Cloud computing has enabled a new degree of cross-functionality. Before the cloud, being a network technician or a database administrator required very different skill sets. In the cloud, every function is an API call or a command line prompt away, so most cloud administrators are as skilled at provisioning a VPC network as they are at deploying a SQL database.

This revolution invites us to pack more productivity into a single small team. Instead of forcing team members to decide between being “architects” and “database administrators”, encourage your database administrators to develop architecture skills, and ask your architects to develop skills in the databases they support. This requires a shift in mindset. The team won’t succeed if everybody volunteers to work only on what they’re comfortable with. That’s the lethargic “not my job!” mindset. Instead, it requires an agile mindset of “show me how!” This is what we call a “cross-functional team”.



Of course, there will always be somebody who knows the most about something or excels at a particular task. But the goal is to spread expertise as broadly across the team as possible. This ensures that a) the team is fully utilized and productive without idle wait time, and b) that the team isn’t blocked when a team member goes on holiday or falls ill.

Finally, consider whether the team will need to interact with other teams or stakeholders, e.g., the CISO or your enterprise architect. Every time the team seeks an outside stakeholder for input or an approval, it introduces wait time and the flow of information degrades, ultimately slowing down the velocity of the team immensely. Include those individuals or one of their delegates as core members of the team, until such time when their expertise has been sufficiently absorbed by other team members or – better – their expertise has been captured in code or architecture blueprints.

In summary, being able to fit all your stakeholders and practitioners into one (metaphorical) room and allowing them to fully focus on one shared objective is the single biggest predictor of implementation speed and often the most obvious difference between a digital native organization and a traditional enterprise. If you can prioritize only one Cloud Teams design principle from our playbook, this is the one.

How many people need to be involved in order to fully execute your company's cloud strategy?

Will all team members be able to dedicate their time (80% or more) to execute your company's cloud strategy?

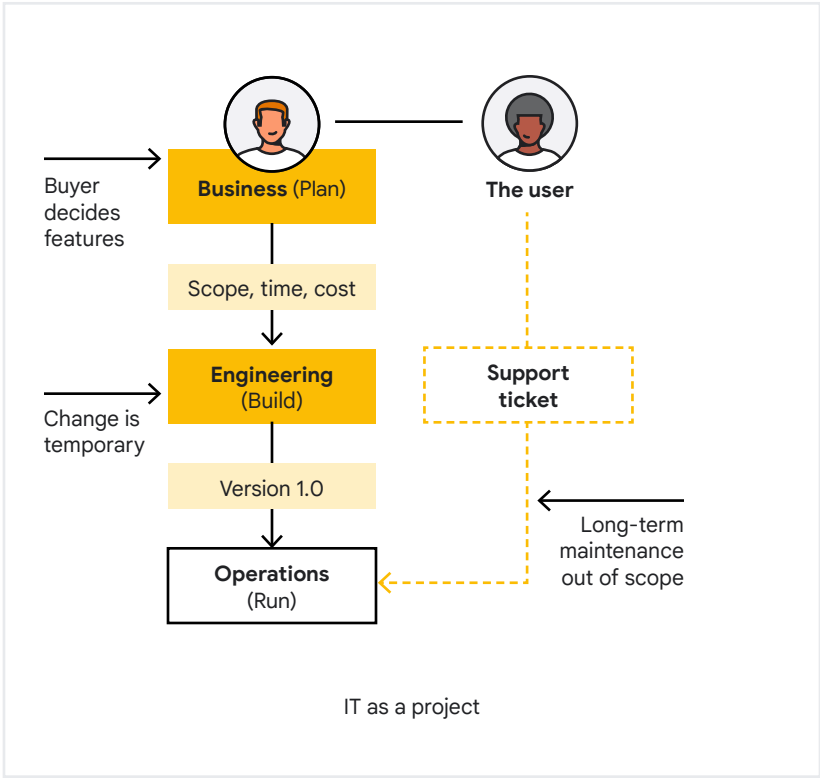
Which of your IT practitioners are most eager to grow into the role of Cloud Architect and/or Cloud Engineer?

Team purpose
(Projects vs. products)

After team size, the next most fundamental consideration is a team's purpose. Specifically: is the team running a project or is it building a product (meaning: software artifact)? In this section, we will explain why the distinction is so important and why their respective goals would be at odds with each other, if we were to confuse or blend the two.

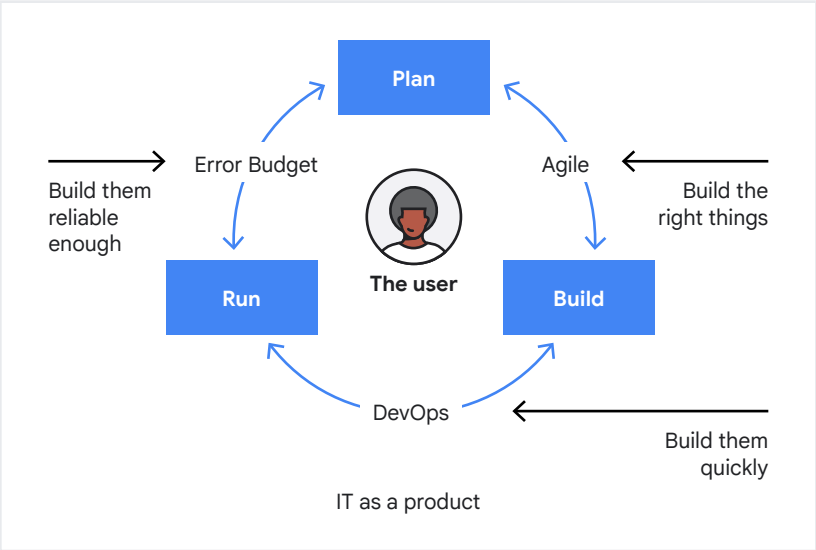
Project teams are optimized for efficiency and repeatability, with a well understood definition of "done" and the ability to move on from one batch of work to the next. They are ideal for large-scale migrations in which dozens or hundreds of applications are moved and ultimately handed back to their respective owners.

Conversely, project teams are not fit for developing new software solutions. The requirements are defined by what the budget owner thinks the user needs. The engineering team gets one shot to deliver a version 1.0 on time, in scope, in budget, and then they hand their work over to a separate operations team whose primary objective is to meet their own SLAs. The user's sole avenue for feedback is to file a support ticket, to which the operations team typically has only a binary response: either the solution is demonstrably broken or it's "working as intended."

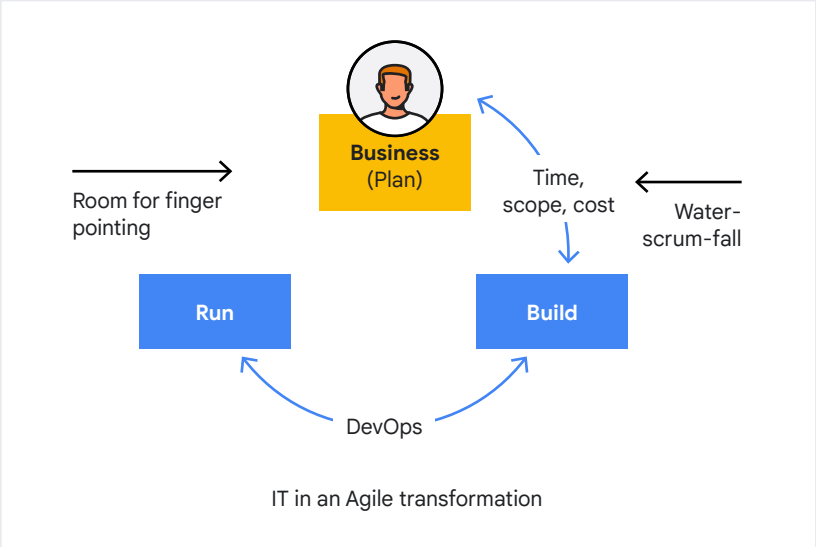


IT as a project

On the other hand, product teams are essential when there is no "before" or "after," and change is constant. The architect or the business does not dictate the requirements – rather it's the user who's the ultimate authority on informing what needs to be built, with a product manager continuously studying and learning from their users and advocating on their behalf. On a product team, instead of fixed deadlines and milestones, there are only backlogs of user stories that flow into sprints of incrementally feature-rich, usable software. A product team does not abandon its work after version 1.0 and remains dedicated to the solution, continuously expanding its functionality and refining it in perpetuity (or until it is deprecated).



A cautionary aside: a feature team can have some of the same characteristics as a product team – with the exception that requirements are still determined by an outside stakeholder - usually the budget holder - rather than actual customer feedback. This risks the product manager acting as a project manager, rather than as a customer advocate.



Be very deliberate about when you are running a project and when you are building a product! Even the smallest, most talented team with the most manageable scope of work and fullest decision-making autonomy will fail if it is confused about its mission.

	Project	Product
Purpose	Deliver a specific outcome	Solve specific user needs
Scope determined by	Requirements set by budget owner	Hypotheses through studying user needs
Lifespan	Until last milestone is achieved	Until product is deprecated
Work allocation	Fixed per person, documented with RACI	Fluid, ownership negotiated per user story
Delays addressed by	Assigning more people	Reprioritizing backlog, reducing work-in-progress
Team assignment	Part-time or full-time, fixed duration	Full-time, open ended
Success criteria	Time + scope + cost	User happiness + engagement

Building the right features (Agile)

Knowing what to build requires a rich understanding of your users or internal customers (personas), and a clear and shared documentation of their critical user journeys. Getting this right ensures that you don't squander your valuable engineering talent on building features, solutions and platforms that nobody ends up using. Establishing user-centric success metrics, for example using Google's H.E.A.R.T. framework, will help drive the right prioritization of your feature backlog.

	Happiness	Engagement	Adoption	Retention	Task success
Goal	Users actually enjoy using your product	Users explore multiple parts of your product	Users discover and try your product	Users keep coming back to your product	Users can complete tasks effortlessly
Example metrics	Surveys Net promoter score App store reviews	Number of visits per user per week Number of photos uploaded per user per day Number of shares	Upgrades to the latest version New subscriptions created Purchases made by new users	Number of active users remaining over time Renewal rate or failure to retain (churn) Repeat purchases	Search result success Time to upload a photo Profile creation complete

Building those features quickly (DevOps)

Being able to build software features quickly requires a high degree of automation and autonomy within your software engineering teams. Google Cloud's DORA program and its managed services for continuous integration and continuous delivery (CI/CD), can help you improve your engineering excellence along four key indicators: lead time for changes, deployment frequency, change fail rate, and failed deployment recovery time.

01

Change lead times

How long does it take to go from code committed to code successfully running in production?

Low: 1 week - 1 month
Medium: 1 week - 1 month
High: 1 day - 1 week
Elite: <1 day

02

Deployment frequency

How often does your organization deploy code to production or release it to end users?

Low: 1 week - 1 month
Medium: 1 week - 1 month
High: 1 day - 1 week
Elite: On-demand

03

Change failure rate

What percentage of changes to production or releases to users result in degraded service?

Low: 64%
Medium: 15%
High: 10%
Elite: 5%

04

Failed deployment recovery time

How long it takes to recover from a failed deployment?

Low: 1 month - 6 months
Medium: 1 day - 1 week
High: Less than 1 day
Elite: Less than 1 hour

In addition to the research findings, the DORA Quick Check aids cross-functional teams' ability to identify their biggest constraints and provides a guide for practitioners during transformational efforts and continual improvement.

Building those features reliably enough (Error budgets)

Google's Site Reliability Engineering (SRE) methodology is recognized as the industry benchmark for its focus on scalable, reliable, and efficient large-scale IT operations, along with its promotion of a blameless culture and the concept of error budgets.

Error budgets establish an explicit and quantifiable agreement between IT and the business about how important the reliability of a software innovation truly needs to be, and how much failure is acceptable. To quantify it, we establish service level indicators (SLIs) and service level objectives (SLOs) along the critical user journeys (see User journeys). When things do break, blameless postmortems help everyone discover the systemic root cause and to collectively learn from it for the future.

Kind	Indicator	Description
Request / response	Availability	% valid requests served successfully
	Latency	% valid requests served faster than a threshold
	Quality	% valid requests served without degrading quality
Data processing	Freshness	% valid data updated more recetnly than a threshold
	Coverage	% valid data processed successfully
	Correctness	% valid data producing correct output
Storage	Durability	% of data stored intact

01
Product management defines an availaibility target

02
The actual uptime is measured by the monitoring system

03
The difference between these two numbers in the "budget" of how much "unreliability" is remaining

04
As long as the uptime measured is above the target, new feature releases can be pushed

It is important that you first adopt and continuously improve SRE practices inside your cross-functional product teams. Only later, once these teams have reached an advanced level of cloud maturity, should they consider splitting out the “run” responsibilities into discrete SRE teams. Splitting these responsibilities too early carries the risk of merely rebranding what is otherwise a traditional IT operations team, without any of the benefits of the SRE methodology.

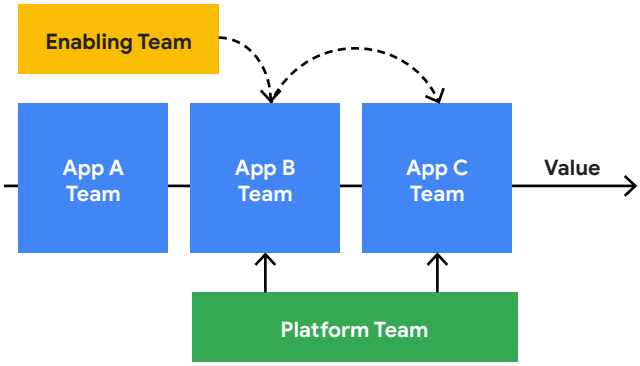
How do you determine which features to develop today? Who decides?

Which KPIs do you measure today when a team develops a software artifact?

How does your organization typically respond to timeline delays?

Team types
(apps vs. platforms vs. enablers)

The central design principle that should inform the composition of your Cloud Team(s) is the timely creation of user value. The larger your organization, the greater the temptation (and/or regulatory requirement) will be to centralize and standardize, at the expense of the fast flow of user value creation. We will dive into different team types and their many possible governance models and outline their benefits and potential risks to be considered as your IT organization evolves over time. For a more comprehensive exploration of the topic, and beyond the context of cloud adoption, we recommend Mathew Skelton and Manuel Pais' seminal publication, Team Topologies.



App teams

This cross-functional team type is your default choice for building the right things, building them quickly and building them reliable enough (see Team Purpose above). App teams enjoy a high degree of autonomy and comprise all the skills necessary to plan, build and run their solution. The quick feedback loops within the team and between the teams and its users allows for rapid iteration and improvement.

Due to their autonomous nature, they inherently run the risk of duplicating tools and processes and implementing repeatable patterns inconsistently between App Teams. As your organization establishes more and more App Teams, this becomes increasingly inefficient.

Platform teams

Platform teams provide shared tools and technologies across multiple App Teams. Their priority is to unburden the App Teams from performing engineering work that doesn't add user value while providing a delightful developer experience.

Standardization is a byproduct, not the purpose. App Teams should have the autonomy to self-determine whether a given platform sufficiently suits their needs or whether to maintain their own underlying infrastructure. If a platform does not suit the App Team's needs or provides a poor developer experience, it can do more harm than good. A different way of thinking of this trade-off is through the architectural lens of single tenancy vs. multi-tenancy. The cardinal mistake that any platform team can make is to assume that if they built it they (the App Teams) will come.

Platforms run on top of other platforms, meaning: one Platform Team can be another Platform Team's user. The most common platforms that organizations build in the cloud are 1. a foundational Cloud Platform (also referred to as "landing zones") 2. a big data and AI platform, 3. a container run-time platform and 4. a CI/CD or DevOps platform. Platform requirements should emerge organically. Successful organizations do not set out to "build a data platform", but build delightful data and AI use cases, and then over time recognise the need for bundling the underlying capabilities in a big data and AI platform.

The CNCF Platforms Working Group has developed a maturity model to help organizations assess and improve their platform engineering capabilities. This Platform Engineering Model can be used by organizations of all sizes and levels of maturity. It provides a framework for understanding the key components of platform engineering and for developing a plan to improve their capabilities. The model can also be used to measure progress and identify areas where further improvement is needed.

Enabling teams

Enabling teams are single-function, specialized teams who know things rather than build things. These two aspects set them apart from App Teams and Platform Teams. Their purpose is to perform research, analysis and experimentation (e.g., in areas like user interface design, IT security and artificial intelligence) and provide insights and best practices to the App Teams and Platform Teams. Their approach can be as light-touch as writing documentation and handbooks or go as far as embedding one of their members into a cross-functional App Team or Platform Team for a finite amount of time.

The impact of an Enabling Team is measured in traditional project-centric KPIs, such as:

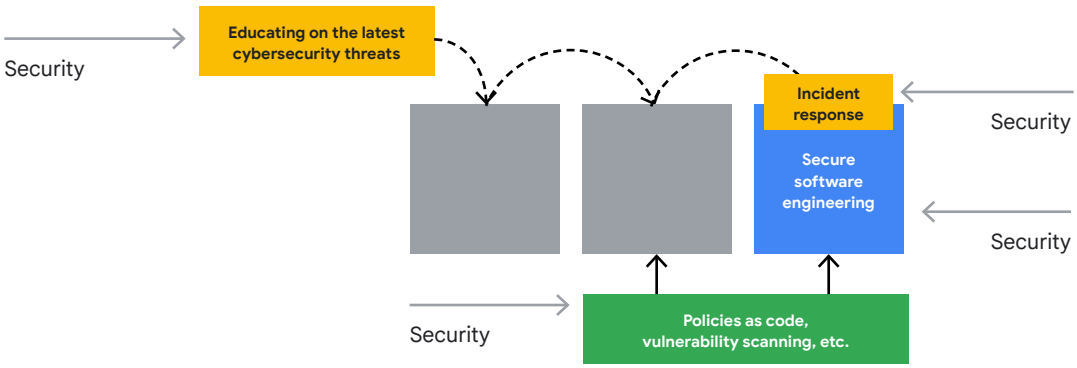
- Quantity and audience size of training workshops delivered
- Quantity and adoption rate of architecture blueprints designed
- Quantity of hackathons facilitated
- Quantity and turn-around time of support tickets answered

The advantage of forming a discrete Enabling Team, rather than simply embedding that expertise inside an App Team or Platform Team from the beginning, is two-fold: 1) not every App Team and Platform Team needs a certain specialization at all times, 2) most App Teams and Platform Teams are too busy burning down their backlog and fixing bugs that they don't have the extra bandwidth to research and experiment without a clear outcome. Mixing product-centric and project-centric KPIs within the same team produces conflicting priorities.

The risk of every Enabling Team is that they don't experience the real-world and long-term consequences of their own recommendations and end up giving poor advice. Their purpose is not to act as a governing authority or to grant permission or sign-off to App Teams and Platform Teams. Enabling Teams must prove their value to their internal customers and rely on them to recommend their expertise to other App and Platform Teams within the organization.

Different team types produce different outputs

We can further illustrate the differences between these three team types by example of an individual “Security Engineer”. This same individual with the same skills will perform different duties and have different goals, depending on which team type we staff them in.



A Security Engineer within an App Team is first and foremost a regular software engineer who “shifts left” on security and practices DevSecOps. They have an elevated understanding of ensuring that security is built into the whole application CI/CD process.

That same Security Engineer within a Platform Team acts as a kind of 2nd line defense against security vulnerabilities and attacks. They will implement security controls and policies as code, will deploy vulnerability scanning and intrusion detection systems, and provide trusted “golden” images for virtual machines and containers.

This same Security Engineer within an Enabling Team will design the policies and controls. They are usually the first to acknowledge new common vulnerabilities and exposures (CVE) and collaborate with the App and Platform teams on remediating them quickly and effectively. They might form a Red Team. Give them a pager, and they are on-call for incident response.

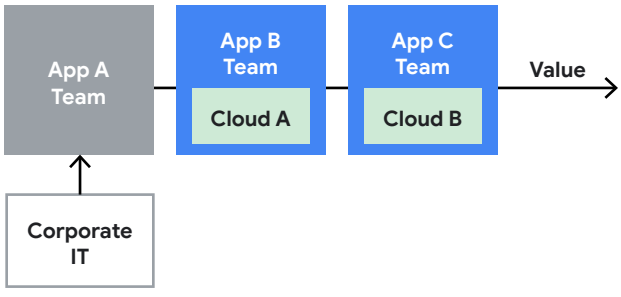
And their various governance models

The structure and governance of Cloud teams adapt to the organization's cloud maturity and needs. These governance models are dynamic, continuously evolving to meet user demands and align with the guiding principles outlined in this whitepaper. As application teams adopt cloud technologies at scale, governance models typically progress through a journey, transitioning from experimentation to a mature, optimized approach. We'll explore five distinct governance models, ranging from simple to complex, highlighting their benefits and limitations.

Experimental governance model

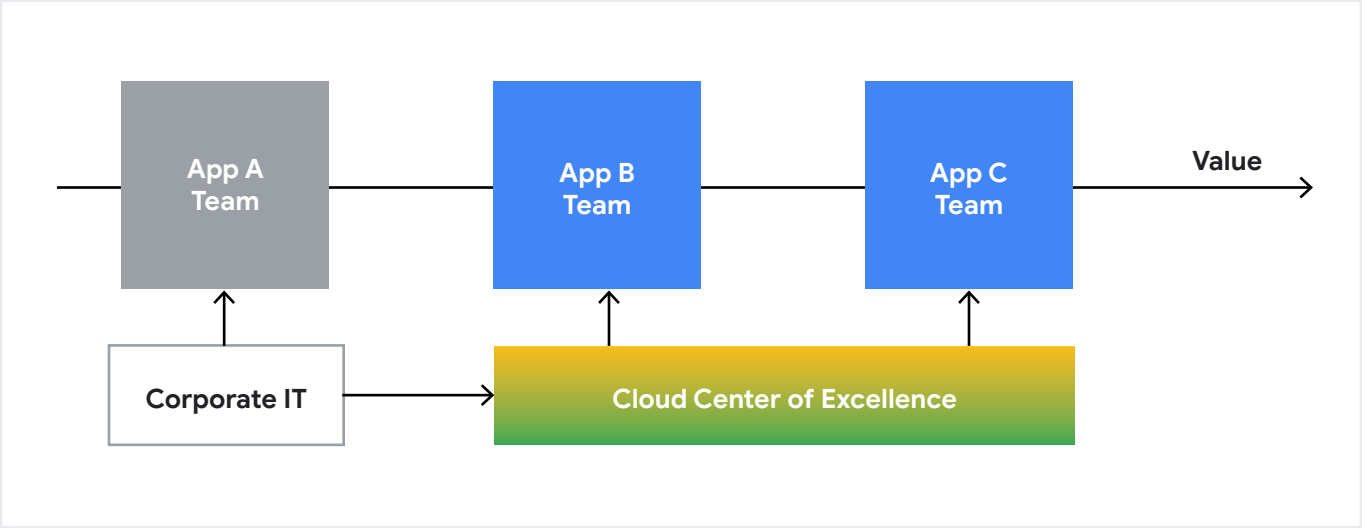
In this model, App teams have full access to experiment with cloud technologies. They also independently govern their cloud technologies, including aspects such as costs, security postures and controls, Identity and Access Management, and more. This model offers maximum autonomy for App teams, resulting in rapid time-to-value for their users. Additionally, it facilitates rapid learning and proof of concepts for cloud technologies, such as novel generative AI solutions.

However, there are constraints associated with this model. For instance, there may be inconsistencies in security postures and controls across different App teams. Additionally, App teams might encounter limited access to enterprise networks or external data stores due to the requirement that utilized cloud technology must be approved by the organization's compliance and security guidelines. This can potentially restrict the use cases.



Cloud center of excellence (CCoE) governance model

In this governance model, a dedicated team centrally drives the cloud strategy, builds, and manages the Cloud Platform. The governance setup is effective as a starting point for an organization's cloud journey, as the dedicated team spearheads cloud adoption and accumulates consolidated knowledge of cloud security and operating models.



Nevertheless, there are constraints to this model. One notable challenge is the presence of competing goals within the same team, specifically **project** and **product** goals. It can be challenging to simultaneously build the cloud platform, drive the cloud strategy, and support App teams. Consequently, the CCoE team may face pressure to cater to everyone's needs, making it difficult to scale beyond a handful of App teams.

Governance model for scaled, self-serve cloud adoption

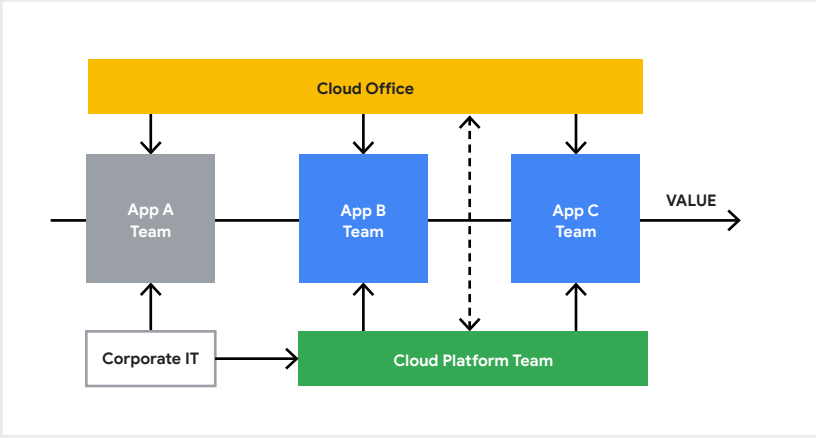
In this governance model, a central **Cloud Office** is established as an enabling team for App and Platform teams. This team drives the cloud strategy and fosters cloud demand within the organization as a project – meaning: until the final milestone of the cloud strategy is completed. Its responsibilities encompass eight discrete workstreams which can be mapped to separate subteams accordingly if needed:

- Executive Sponsorship
- Cloud Teams Design (the subject of this whitepaper)
- Communication
- Hiring/Recruiting
- Training/Upskilling
- Cost Control/FinOps
- Portfolio Planning/Intake
- Contract Management/Procurement

Given possible resource limitations, not every Cloud adoption journey can or wants to afford a fully dedicated Cloud Office. Designing to budget is key. Be aware however, if you are expecting quick time to value, it's absolutely essential to have ownership of the activities described above. This is where your cloud strategy needs to match your ambitions.

Additionally, a **Cloud Platform Team** provides the shared tools and services as self-service required by App teams. The foundational cloud platform can be broken down into discrete technical domains and corresponding subteams, if needed:

- Identity and access management
- Data protection and encryption
- Shared VPC networks
- Architecture blueprints as code
- Trusted images (software supply chain)
- Shared logging and monitoring

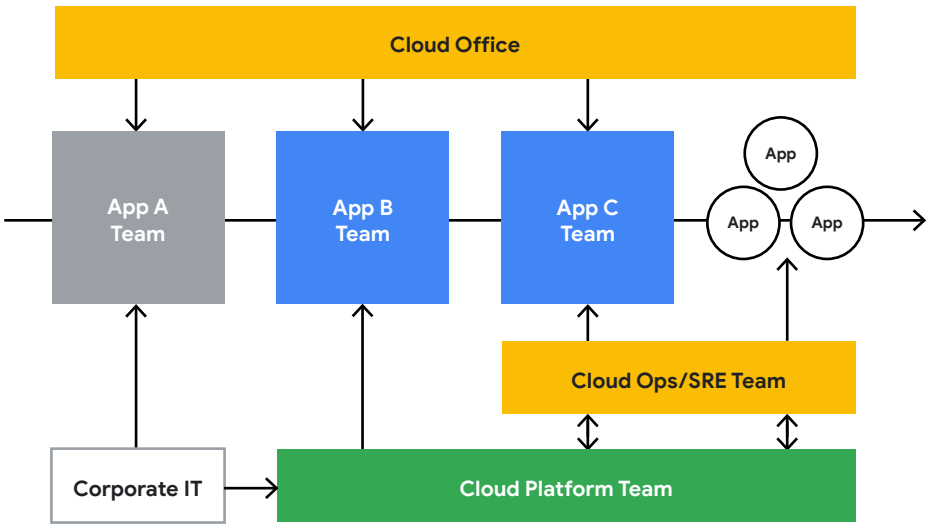


This model offers the advantages of a dedicated team focused on cloud strategy implementation and adoption, alongside platform teams serving user needs and operating platforms as products. This structure ensures a clear separation between project and product goals and KPIs. However, this governance model is not suited for apps (and teams) that require external operations support. This limits the scaling of applications in the organization that do not have dedicated cross-functional app teams.

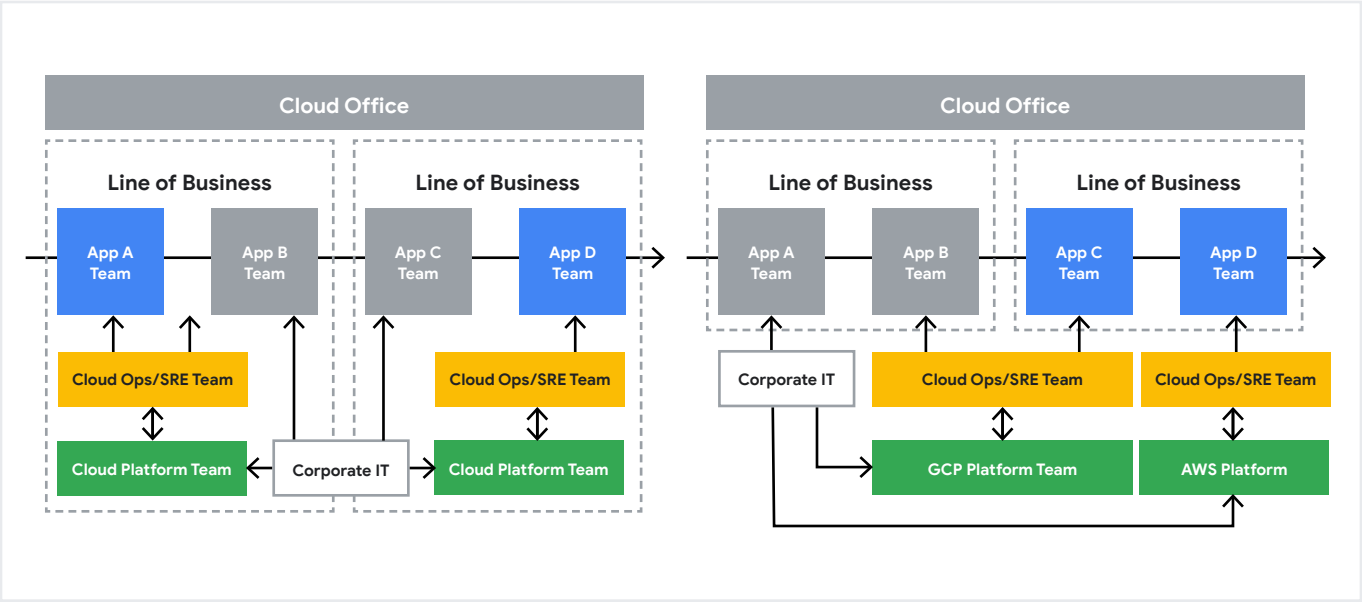
Governance model for scaled cloud adoption with full operations/SRE support

For scaled cloud adoption, establish a **Cloud Operations/SRE team** in addition to your Cloud Office and Platform Teams. This Cloud Operations/SRE team serves as an enabling team, offering expertise to App teams and the Platform team to ensure continuous product uptime. This team can also operate applications that don't have a dedicated application team. Optionally, App teams with well-instrumented SLIs and stable SLOs may outsource operations to the dedicated Cloud Operations/SRE team, as long as they remain within their error budget. The 'operation' work should be capped to a maximum of 50% to ensure the team has the bandwidth to advise App and Platform teams to improve the reliability of their products.

This model offers the advantage of having a dedicated team driving cloud adoption, while also maintaining a clear separation between project and product goals and KPIs. Moreover, the inclusion of a dedicated team capable of operating selected cloud applications without the need for a dedicated development team facilitates scaling within the organization.



Governance model spanning multiple business units



For very large organizations or those with very independent lines of business, it can be very hard for a single foundational Cloud Platform Team and a single Cloud Operations/SRE team to cover the needs of all their users, without breaking the first two design principles of the Cloud Teams playbook: keeping those teams small and staying close to their users.

There is nothing wrong with having two or more foundational Cloud Platforms and corresponding cloud teams in parallel, so long as each delivers enough value to their users to justify their cost. In fact, when organizations acquire or merge with other organizations, this is the default scenario.

Alternatively, if your organization uses more than one cloud service provider (multi-cloud), you may consider splitting your teams along these lines, so that each may deeply specialize in the product language of that respective cloud provider. An App Team will express a preference for a service of one cloud service provider or another, and they will leverage the platform and enlist the services of the cloud operations team that map to that cloud provider.

How many platforms do you intend to operate in the cloud? Which are they?

Are your app teams free to reject a shared platform and run their own stack?

Which areas of cloud expertise does your IT org have that are best bundled in an enabling team?

Team priorities
(personas, user journeys and OKRs)

Personas

We've all heard the advice to "start with the user and work backwards." When large enterprises develop internal platforms, this can be harder than it sounds. The actual user of your platform is another employee in another department, but their needs are represented by their department head. They, in turn, may interact with a "head of product X" who prioritizes the feature backlog and delegates architecture design to a platform architect, who in turn gives guidance to the engineers on how to build it… The end result is often a costly platform that nobody uses, unless forced to. On the flip side, compelling platforms most often begin as something that an App Team initially built for themselves and only generalized later. Most of Google's internal tools and platforms started this way. Some of their open source equivalents are listed here: https://opensource.google/projects. You can't get any closer to your user when the user is you.

When a discrete platform team is not its own user and doesn't regularly and comprehensively "dogfood" its own platform, it is crucial for the team to have a clear and shared understanding of whom they are building for, articulated as a user persona.

Hello, my name is
Jonas

And I am a
Application Engineer

My skills and experience:

- I am an experienced Java developer and DevOps practitioner
- I have minimal cloud platform experience and usually interact with the prod environment through my continuous deployment pipeline

My objective is:

- I want to deploy an application that meets performance and availability requirements

My biggest pain points are:

- Waste and manual work in existing process, e.g. performing checks and controls manually
- No prod-like test environments
- No canonical definition of performance and availability requirements across applications

An effective persona description includes key technical skills and skill gaps, what their main objectives in their own job are, and which common pain points they frequently experience. User personas are the foundation for prioritizing the platform backlog and for developing a platform that is easy enough to use, to make the user more productive and to ensure a delightful experience.

A single platform team may need to serve more than one persona. During sprint planning, the platform team needs to be absolutely clear about which persona they are catering to in each user story ("As [persona], I..."). If not all personas can be served equally well, the platform team would be well advised to make one persona productive and happy, before expanding to the next persona. Alternatively, consider splitting the platform team in two (or more), so that each may fully focus on their respective personas.

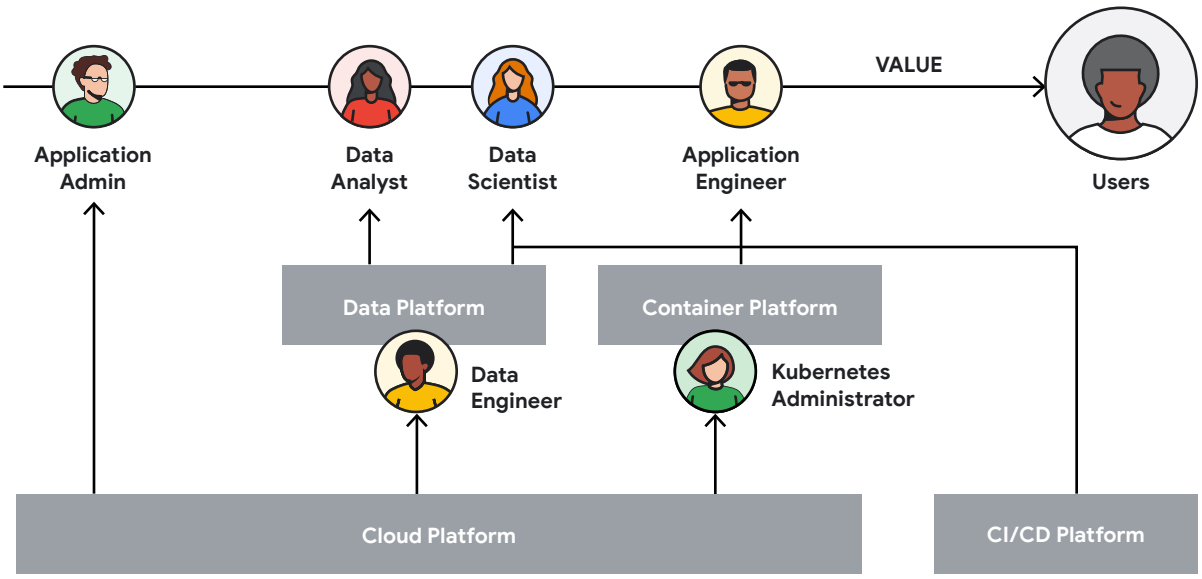
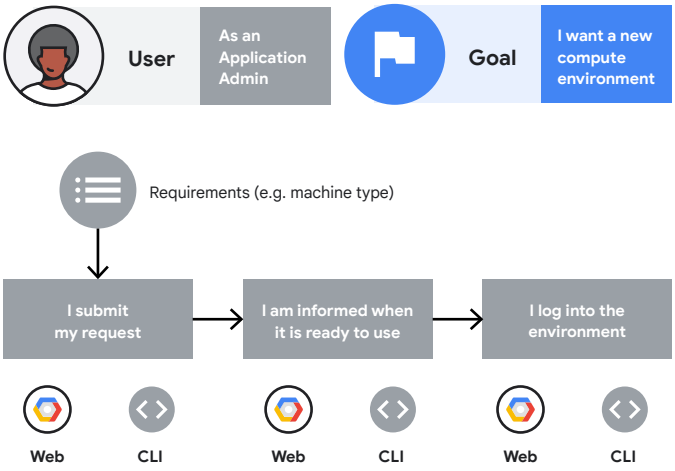


Illustration of how different common platforms in the cloud serve different personas

User journeys

User journeys are an essential part of any user experience design. They map out the steps that a certain user persona takes to complete a goal, from the beginning to the end. A critical user journey (CUJ) is one that is essential for the user to achieve and therefore is critical to the success of the overall platform. This analysis is not about replicating all user journeys, but about understanding the most important ones and ensuring that they are as efficient and effective as possible. These user journeys are critical because they are either very common (toothbrush journeys) or very important to get right (pivotal journeys), or both.

Critical user journeys express user intent, not features. Users care about accomplishing their goals and getting something done, not about features or the specific steps involved. As such, they don't change much over time and can serve as a north star for many months or even years. Every critical user journey consists of a goal and one or more tasks. The goal describes what the user intends to accomplish, such as "I want to remember an upcoming meeting". The tasks describe the specific steps that the user needs to take to achieve the goal, such as "I open my calendar". As a user completes each task, they get one step closer to achieving their goal.



Example critical user journey for creating a new compute environment

Critical User Journeys are a great methodology to prioritize the product backlog. They inform what the App/Platform Teams have to get right to make the product as useful and usable for their users.

To systematically improve user experience you should continuously measure the metrics outlined earlier (see H.E.A.R.T. framework) along the Critical User Journey. This includes regularly validating their importance to users and ensuring the ease of completing tasks within them.

Discover	Onboard	Migrate/Deploy	Run	Optimize
I **hear** about the platform at an internal tech talk / demo session I **read** about the platform on our internal wiki/Confluence I **touch and feel** the platform through interactive training labs and/or sandboxes	I **request access** to the platform I am **informed** when my access has been granted I **log in** to the environment via a command line (CLI) or web browser	I **redirect** my continuous delivery pipeline I **redeploy** my VM I **dump & import** (ETL) my database content	I **monitor** the health of my application I **review** the logs for specific events and application behaviours I am **alerted** to critical issues and recieve assistance in triaging whether it is an app issue or a platform issue	I **predict the cost** of cloud resources for my application I **review** the actual cost of my app at the end of each month I am **informed** of underutilized cloud resources

Example critical user journeys for a foundational Cloud Platform / landing zone

Critical User Journeys' success requires cross-functional commitment and a long-term perspective to allow for iteration and learning cycles that will shift the culture and development process within an organization.

Objectives and key results (OKRs)

OKRs are a tried and true framework of setting ambitious goals that are meaningful and inspiring along with key results that are specific, measurable, attainable, relevant and timebound (S.M.A.R.T.). Popularized by John Doerr, and famously adopted by Google since its early founding years, OKRs allow organizations of all sizes to set a direction and an expectation of the distance of travel in that direction, while federating most of the complexity and nuance closer to where the best information is. OKRs:

- Foster disciplined thinking (the major goals will surface)
- Communicate accurately (let everyone know what's important)
- Establish units of progress (shows how far we are along)
- Focuses efforts (keeps the org in step with each other and fosters coordination)

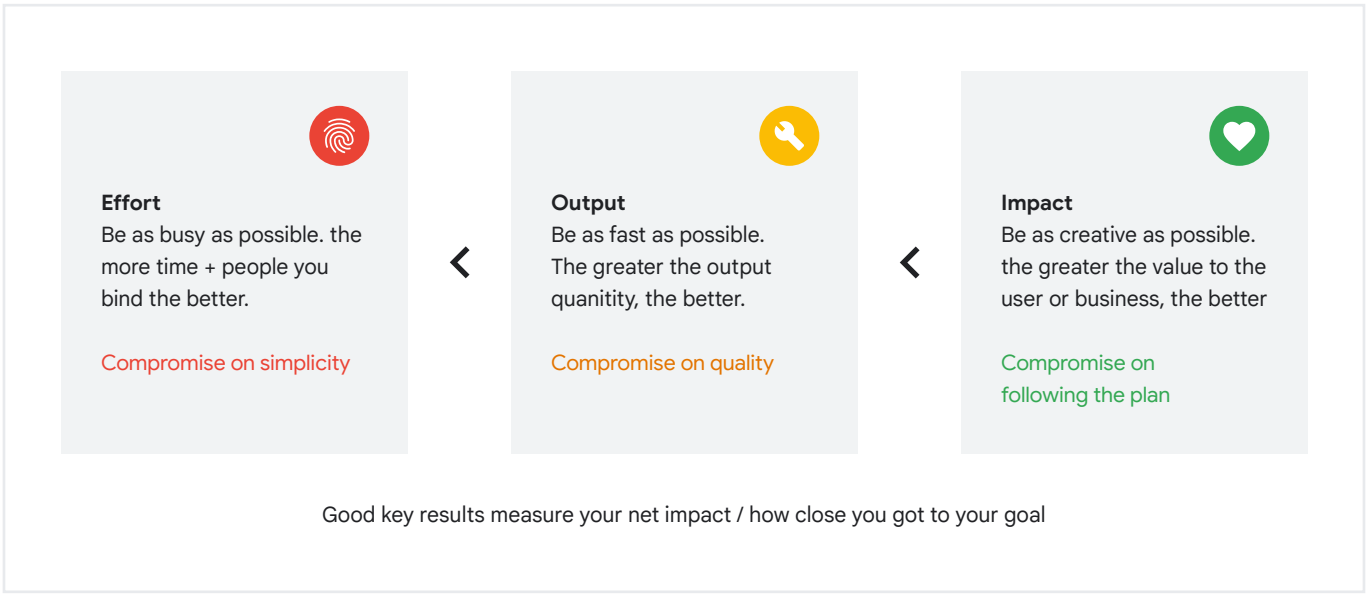
The main caveat with OKRs is that they are hard to implement in only parts of the organization and without full buy-in from the entire reporting chain. Since we are concerned only with optimally governing your Cloud Teams and not your entire organization, we must be deliberate about using OKRs primarily as a communication tool and not as a performance management tool (including incentive setting).

As a communication tool, it is essential that each Cloud Team share their OKRs and their progress transparently, frequently and proudly. Their OKRs are like an advertising billboard to all adjacent Cloud Teams. Finding a central place to manage and publish all Cloud OKRs is essential, as is the frequent and public review of progress, else they will quickly be forgotten and be considered little more than a toilsome thought exercise.

Tip #1:
Start with a blank sheet of paper, not with your org chart. The goal hierarchy of your cloud strategy (and corresponding key results) must be devised as one leadership group effort, without regard for reporting lines, departments, tribes, workstreams etc. Only once you have a single, shared, complete and concise view of your goal hierarchy should you begin to answer who will take ownership of which goal and/or key result. In some organizations, mapping clear ownership might prove to be a seemingly impossible task. While OKRs alone won't fix this, they will expose where a lack of clear ownership will delay or derail your efforts further downstream. Think of OKRs as a canary in your cloud coal mine.

Tip #2:
Focus on the first two levels of your OKRs (i.e., the most important objectives). These are all you need to successfully capture the essential aspects of your cloud strategy and ensure that your Cloud Teams set the right priorities and measure what really matters. No matter how many levels deep you go, don't cascade your OKRs down to the individual contributor level.

Tip #3:
Try to articulate key results that measure impact, over output, over effort. In the early stages, most teams will naturally gravitate towards articulating how busy they will be with activities that they can anticipate and are familiar with. The result will be little more than a todo list, and something that adjacent Cloud Teams and your leadership will neither understand nor appreciate. It's not clear what difference all this effort will make. Leaning towards impact-oriented key results will require leaving their comfort zone and takes practice and encouragement.



An example OKR for the foundational Cloud Platform Team that seeks to gain widespread adoption across the IT organization might look like this: **Offer all IT employees an easy to use and safe Google Cloud environment to develop and deploy their applications**

- KR1: 100 employees use Google Cloud on a monthly basis (MAU), as measured by logins
- KR2: 20 Google Cloud projects have been created and have seen activity in the last 30 days
- KR3: user satisfaction is >= NPS 30, as measured by a monthly pulse survey

An example OKR for the Cloud Office team that seeks to create and accelerate internal demand for the cloud might look like this: **Transform our IT to embrace cloud as the new default technology and way of working**

- KR1: 50 IT practitioners are upskilled in using Google Cloud, as measured by passing the Professional Cloud Architect certification
- KR2: Internal FinOps email newsletter about cloud cost optimization tips is read by 100 recipients per month
- KR3: 100% of additional cloud-first headcount is hired
- KR4: 40% of entire IT org staff has a positive sentiment towards the cloud strategy, as measured by a monthly pulse survey

For a deep dive on how we set OKRs at Google, we recommend Rick Klau's presentation on YouTube.

How do you know who your users are and how do you share this understanding across teams today?

How do you determine which features to build / bugs to fix first?

How do you communicate the value (impact) that other stakeholders may expect from a team?

Team environments
(physical vs. virtual vs. hybrid)

When considering team environments, it is relevant to differentiate between physical and virtual team environments.

Physical

There can be significant differences in productivity of high-performance and low-performance work spaces. Some of the key factors that contribute to lower performance workspaces include a lack of key social connections, unclear communication norms, and insufficient meeting spaces.

Google has established design principles for high-performance workspaces that focus on what is already built and not on high-cost customized spaces. These principles include:

Group:
Adjust open workspaces to a smaller 'neighborhood' size. The neighborhood should be designed for teams, meaning around 5-10 people. Putting a few teams together works well, with a limit up to 32 people.

Boundary:
Protect and separate neighborhoods so teams can work together without disruption. This means creating noise and traffic barriers between neighborhoods. As there might be several teams in the neighborhood, there should also be some way to subdivide these spaces.

Adjacency:
It is important to differentiate between noise-generating communal spaces and protected focus spaces. Noise and traffic-generating spaces should be located away from 'focus' spaces to protect team productivity. Having the team that gets work done together is a key to success and these boundaries need to be protected.

Meeting rooms:
There should be sufficient collaboration space to meet the needs of users. Understanding the real capacity and need of meeting rooms will help to decrease the competition for these spaces.

Virtual

Team chat applications have experienced exponential growth over the last recent years. While real-time communication is advantageous, the real paradigm shift that team chat introduced was the centralization of all topical and project communication in one place. This allows team members to join or leave channels at any time, as well as re-trace the conversation history to gain context and understanding.

Unlike email, where the sender decides who receives information, team chat gives the recipient the power to decide what is relevant to them. This is a fundamental shift from opt-out to opt-in.

In addition to verbal communication, video calls also provide nonverbal cues that are essential for effective communication. Eye contact and body language can help to determine whether a colleague is attentively listening, distracted, or in disagreement. Registering other team members' attentiveness also makes it easier to give equitable speaking time to everyone who wants to share their ideas. This is especially important when collaborating with colleagues that you don't know well or with whom you don't have a trusted relationship.

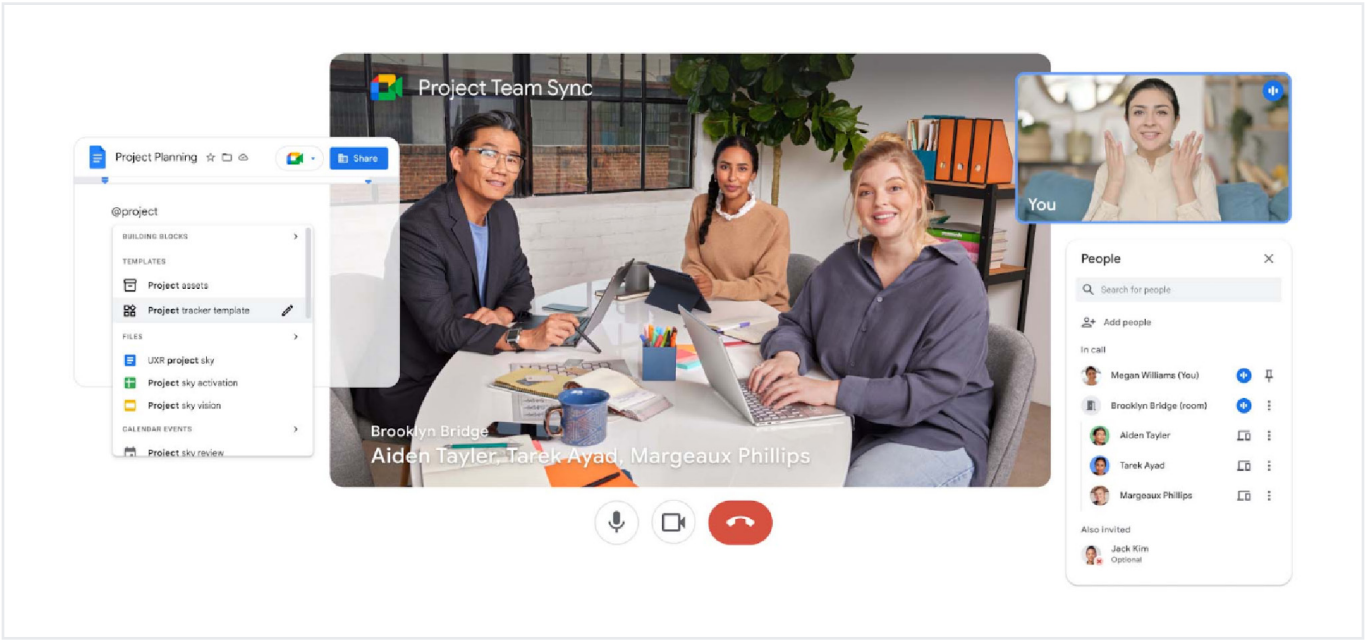
At Google, we are so convinced of the positive power of being seen that we've made it the default setting in Google Meet. Unlike other video conferencing solutions, we give you the option to turn off your camera before joining a meeting, rather than the other way around. We also worked hard to make sure Google Meet can display up to 49 participants' video feeds simultaneously.

Hybrid

The workplace as we once knew has irrevocably changed. Hybrid working environments, offering a mix of in-person and virtual work, are now the standard. Surveys illustrate the widespread adoption of flexible models, with 90% of organizations allowing some or all employees to determine their hours and work location. This empowers employees to work when and where they are most productive.

However, the benefits of the hybrid model are not without challenges. Employees can feel isolated from their teams, and the lack of in-person interaction may diminish trust. Research conducted with MIT further highlights that spatial proximity matters to increase collaboration between multiple departments as 'tacit' knowledge is best exchanged through face-to-face encounters.

Hence, the physical office still holds immense value, albeit with an evolving role. The ideal hybrid environment seamlessly blends physical and digital spaces. In-office employees should be enabled to collaborate and build connections while giving employees outside the office the tools they need to stay productive and feel connected to the team.



When a company strikes the right balance, the outcome is a workplace where employees feel empowered, respected, and well-equipped to thrive – whether they connect digitally or in the hallways of the office.

Tip #1:
Hybrid work offers productivity gains, but don't neglect the importance of belonging and trust. Plan in-office activities and personal interactions to foster these essential elements.

Tip #2:
Strategically arrange office 'neighborhoods' to maximize collaboration. Place teams with the most interdependencies near each other for optimal in-office time.

How well is everyone seen and heard in meetings? (literally and figuratively)

Is everyone from the same team in close proximity to one another? (virtually and physically)

Google Cloud