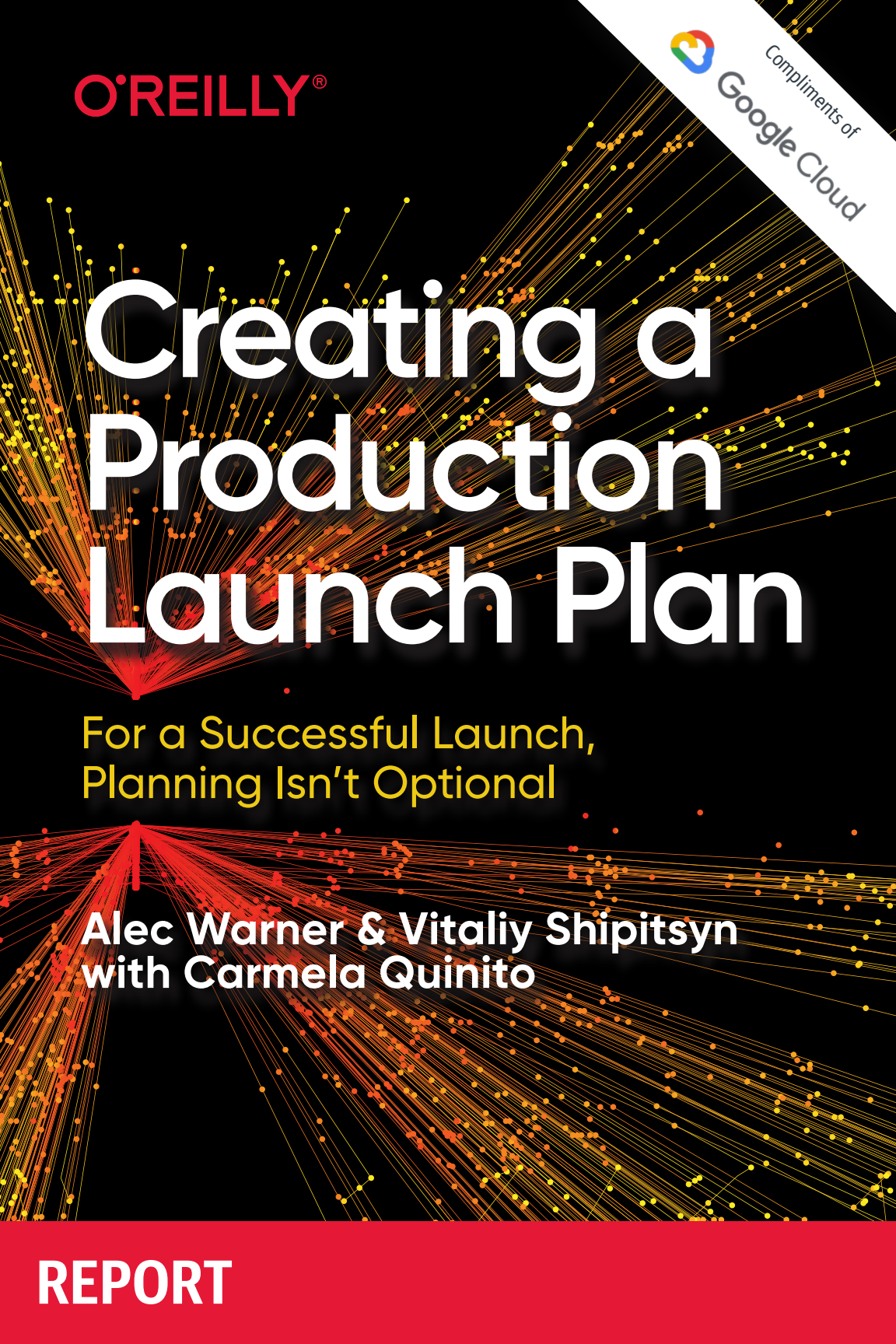
O'REILLY®
Compliments of
Google Cloud
Creating a Production Launch Plan
For a Successful Launch, Planning Isn't Optional
Alec Warner & Vitaliy Shipitsyn with Carmela Quinito
REPORT

Google Cloud

Want to know more about SRE?

To learn more, visit google.com/sre

Creating a Production Launch Plan

For a Successful Launch, Planning Isn't Optional

Alec Warner and Vitaliy Shipitsyn, with Carmela Quinito

Beijing • Boston • Farnham • Sebastopol • Tokyo O'REILLY®

Creating a Production Launch Plan
by Alec Warner and Vitaliy Shipitsyn, with Carmela Quinito

Printed in the United States of America.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (*http://oreilly.com*). For more information, contact our corporate/institutional sales department: 800-998-9938 or *corporate@oreilly.com*.

Acquisitions Editor: John Devins
Development Editor: Virginia Wilson
Production Editor: Christopher Faucher
Copyeditor: Rachel Monaghan
Proofreader: Rachel Head
Interior Designer: David Futato
Cover Designer: Karen Montgomery
Illustrator: Rebecca Demarest

November 2019: First Edition

Revision History for the First Edition
2019-11-13: First Release

978-1-492-07594-3

[LSI]

Table of Contents

Creating a Production Launch Plan

You are responsible for a cool new product with excellent feedback from user studies, and you can't wait to get it into end users' hands. You publish a spectacular press release and sit back, ready to enjoy the praise indicating that your product has surpassed people's dreams. Your product's hashtag shows a great reception, at first. Then you start noticing some complaints. Your operations team pages you that things are melting down.

What happened during the product launch? It could be that when the entire world took notice, the product's code collapsed under the heavy load; or that the infrastructure the product relies on responded to the shockwaves of load by rejecting requests, to protect from unexpected attacks; or that the people overseeing the launch were not prepared to execute on a coordinated response. A launch plan can prevent these kinds of issues by involving all relevant parties and processes to help ensure a controlled progression through the launch.

In this report, we describe the components of a launch plan at Google and offer practical methods you might apply to reduce production launch risks for your own products. We include lessons we have learned about what works well for launching Google products, including large products, new features for existing products, and even small products. The lessons we describe are adaptable for consumer-oriented services regardless of company size or a product's user base. How much control you have over the product's architecture, source code, and operational tuning affects how adaptable you may find these practices. Those with enterprise-oriented

services should find most of this advice relevant, though your implementation may differ.

We hope that after reading this report you will share our opinion that launch planning isn't optional. You know your situation best. We encourage you to choose the practices that are most relevant to you and adapt them to your circumstances.[1]

What Does a Launch Plan Look Like?

The launch plan is a document that communicates the scope and timeline of all proposed changes in production, assembles all relevant points of contact and mitigation, and records the actual launch progression along with any follow-up actions after the launch. It prepares your product for the critical step of transitioning from pre- to post-launch. We provide a template of a launch plan document in the appendix of this report.

Sometimes a launch consists of stages like Alpha and Beta, which permit you to pause and stabilize the product, and move on to the next stage after validating. Such launch stages, which we'll discuss later in the report, might each have a launch plan of their own.

The launch plan as we have defined it may seem obvious, or even superfluous, but we've found that it's essential for a successful launch. Why? We'll get into the benefits next.

Benefits of Launch Planning

Because launch planning is not free of cost, it's important to establish firmly what value it can bring, as well as which circumstances are more likely to deliver which benefits. This section covers just that. We believe that the organizations that choose to put effort into launch planning will reduce overall launch friction, and honing the launch practices we describe later in this report will lead to better launch outcomes on a shorter timeline and at lower costs.

1 Also see Chapter 27, "Reliable Product Launches at Scale," written by Google SREs, in *Site Reliability Engineering* (O'Reilly).

Managing Risks Instead of Hoping for Luck

No one wants publicity disasters or end user complaints to mar their product launches. You must be aware of all issues that might need attention in order to choose what actions to take; at the same time, you need to discern which issues are actionable and, among those, which actually deserve action.

Launch planning manages the risks that might lead to those unfortunate outcomes. Hope is not a strategy for launches; you might not have time for surprises once you initiate the launch, so don't count on luck getting you through!

Launch planning manages risk by helping you:

- Generate risk scenarios of potential launch failures or poor user experience.
- Build mitigation strategies for risk scenarios, and determine the relative mitigation costs.
- Prioritize mitigation work for the most impactful or disruptive risk scenarios, and accept the less likely risks.
- Focus resources toward mitigating launch risks earlier during product development, instead of trying to retrofit them closer to launch.

You have to decide where to spend the effort to gain meaningful improvements, and where the effort will be ineffective or even cost-prohibitive. Balance what risks you want to manage against what it will cost you in time, money, or staffing.

When you cannot anticipate confidently how a particular launch scenario might unfold, developing appropriate backup or fallback launch plans expands your options for managing related risks. This process might involve making a plan B—and sometimes a plan C. For example, if your product starts running out of resources, will you try to get more resources or start to throttle user adoption? And do you even have the option to do either? How quickly will you be able to throttle or completely disable the product if it starts degrading other systems? To lower the risk of being unprepared, allocate sufficient effort to evaluating the need for backup plans and developing these plans, and decide under what conditions you might invoke them.

Having tested and validated product behaviors and launch processes is another category of risk management benefits that launch planning brings. You probably already tested product behaviors before launch; what's easier to overlook are transitional states for the customers or their data when you're launching new product versions. For example, when all supported versions of your mobile client application have to upgrade their local database to a new format, you might want to retain the ability to revert to the original format in case a rollback is needed. Such one-time steps in the launch process may be harder to validate. After you identify where the launch could go wrong, you can do what's necessary to counter those risks.

Allowing Quick Adjustments

The preparations inherent to launch planning can enable your organization or team to make quick and informed adjustments to launch decisions or product attributes. This preparedness can be useful should something unexpected happen, whether due to external factors you cannot control, changes in business conditions, unanticipated technical limitations, or environmental events.

Launch planning allows for quick adjustments because:

- Analysis and preparations involved in managing launch risks help bring out technical parameters and capabilities of the product and the infrastructure.
- Knowledge of those parameters and capabilities can help you create action plans that are more likely to execute as expected.
- Organizing sound action plans is more likely to result in deliberate and optimal responses rather than knee-jerk reactions—especially under time pressure.

Adjusting quickly may be especially valuable:

- After the launch reaches the point of no return—when sufficient commitments have been made, and rolling them back to cancel the launch is no longer practical.
- Post-launch, after you gain new information that helps you decide what's optimal to do next. You can simply select the prepared and sound action plan to execute.

Being prepared for more than a single course of events also enables you to launch more cautiously, with pauses or adjustments. You don't have to be right on all launch predictions. Instead, observe how the events are unfolding, and pick which options fit the reality best. For example, if early during launch you observe aggressive user adoption (good) and concerns for future resource capacity arise (bad), you could pause further ramp-up until resources can be provisioned, if you've planned for this possibility.

Communicating Effectively

Being able to communicate with the right people at the right time can make a big difference during a launch. The number of people involved in or affected by a launch can vary. While a product feature might require just a few people, an entire product might require collaboration among multiple teams or organizations.

Launch planning can improve communication by helping you:

- Identify and keep track of all the parties who might have to communicate before and after a launch starts.
- Distribute and keep current the knowledge of what features will be launching and when.
- Convey anticipated demand from teams and infrastructure, as well as discover impractical or infeasible expectations.

At Google, we circulate the launch plan to all stakeholders, developer teams, and SRE teams involved in the launch. We look for feedback from any relevant teams and domain knowledge experts, and follow up as needed. We rely on this shared knowledge to prepare others for the decisions we might make during launch, and we provide them with the playbook of how the launch might unfold.

Reaching out to all relevant teams affected by the product also communicates what those teams should expect at launch. Communicating expectations can help prevent overloading of dependent services, avoiding alerts from unexpected shifts in service demand, saving time, and minimizing stress. Marketing and press teams are not technical, but may also require certain levels of readiness and hands on deck for managing issues during highly visible launches or at times of low staffing.

Improving Launch Processes Over Time

You want to be sure that launch planning pays off. While there's no doubt that planning is necessary, depth and rigor come at a cost, and not every launch requires the same amount.

Launch planning scales beyond improving any particular launch because:

- A systematic approach to launches permits improving processes over time. This might include collecting signals for the future—what worked well or not, what was impactful or not, and where you simply got lucky.
- Increasing the value and reducing the costs of the planning effort leads to more streamlined planning, permitting more rigorous planning applied cost-effectively to more launches.
- The abilities to evaluate the costs of a launch, build launch taxonomy, and provide signals for future planning are direct results of the planning in the first place.

At Google, defining the taxonomy of launch stages for a product formalized the progression through the product's commitment to its users. It also made team expectations more consistent across the company. After a sufficient number of launches, we began to define launch checklists that drive launch planning, properly assigning focus areas and degrees of rigor to specific launch stages.

Elements of a Successful Launch

Before we go into the details of how to build and execute on the launch plan, let's first establish what elements of products and organizations could contribute most to a successful launch. We'd like to share some of the lessons we've learned at Google. They have helped our teams get familiar with their products' relevant characteristics, ensuring that a product will operate well when launched and aligning it with business needs. Some of these elements may be best addressed early in the product development cycle.

Focus on the Product

To be well prepared, you will need to really understand the attributes of the product you're about to launch. These attributes

include the product's interactions with the environment, the expectations of its end users, and the development history of the product.

Start launch planning early

Some elements of launch planning should be included in product design and development. The architecture choices you make determine a product's scaling limits and deployment options. Throughout product development, try to engage with the teams who own dependent products, to determine their limitations. Evaluate the risks the product's needs might present to your company's infrastructure if the product is wildly successful.

The earlier you consider launch needs and start these interactions, the more opportunities you and other teams will get to accommodate requirements or advice. It's never too early to start a plan. In the worst case, you might have to revisit some issues when you have more of the product designed, implemented, or integrated. If you discover you started planning late, you can still choose what changes to make and determine whether to adjust the launch timeline.

Some feedback may force you to change the architecture of your product. You could uncover that the wrong or nearly obsolete technology was used, or at certain scaling the software may reach unacceptable limits. You can then decide whether to pivot right away or do something later; either way, you'll make fewer wrong assumptions, preventing some waste in development.

Get traffic projections

Estimate demand for the product using whichever means can reasonably forecast it for this particular launch. We express demand as traffic—the number of operations per unit of time (for example, requests per second or per hour).

How do you project demand for a product that has no users? You could ask customers about their estimated use cases. You could onboard a small number of users and measure their use cases, then project the demand outward by talking to customers who have not yet adopted the product's solutions. You could look at request rates on the old system your product is replacing, or at comparable product or feature rollouts in the past.

You will need traffic estimates for several key time ranges: the day of launch when the interest peaks (the first 24–48 hours), the near-

term range while the newness still exists (1 week), the medium-term range that permits you to evaluate the growth slope (1 month), and a long-term range that targets any perceived macro conditions (6–12 months).

Product features that significantly impact production infrastructure might need more time to prepare for launch. For example, if you expect to consume unusual amounts of bandwidth or memory, provisioning those resources may take time. High cost of projected resources might involve a longer approval process from the business side, or a longer time to physically provision resources. Even worse, off-the-shelf solutions could have scaling limitations, and if product demand might exceed them, significant architecture changes may be necessary. For example, if a relational database has an upper limit on transaction throughput, there's only so much scaling you can accomplish by adding more CPU or memory, which doesn't scale horizontally.

Accuracy of estimates is more important for the near-term horizon, because you have more time to adjust over the longer term, plus those estimates can rely on some history. Underestimating traffic for the day of launch by 100% can be disastrous, but doing the same for six months later is insignificant.

Longer-term estimates are useful for long-term capacity planning for your product or its dependencies. Production infrastructure supports many products, and will expand or contract the resource footprint in anticipation of demand. Short-term oversubscription or reserve resources support dynamic fluctuations, but this scenario either risks not having capacity for all services or wastes money on idle resources. Depending on your organization, the value of longer-term projections may be higher or lower.

Know the impact of all production traffic the product will add. Exposing a product to the world may amplify even the tiny traffic streams it adds to exceed what the product itself or any of its dependencies is provisioned for. For example, identify all remote procedure call (RPC) types your product sends, and the RPCs those RPCs might trigger, and so on. For all identified cases, you will want to obtain an explicit approval from the respective service owners, and record somewhere how much was approved in case you revise numbers significantly later.

Review the product's architecture

Thoroughly review the product's architecture from the perspective of functional stability. Get the diagrams of how the product's components interact and what they depend on. Every software entity has some API and an owner who ensures that API delivers its function. This point in launch planning is where you extract who the points of contact are, what service-level objectives (SLOs) are provided, what RPCs your product sends, and how much resources of various types those RPCs cost. You will use all of this information later.

Using these details, look for what might go wrong. Review the assumptions critically, interview component owners if needed, and record any questions or concerns for resolving definitively. The output of this process is a Risks and Mitigations table, which you give to the appropriate developer or operations teams to resolve *before* the launch date. These risks are systemic issues that code or configuration changes should eliminate, not action items for the launch day. Discoveries discussed in other sections on product analysis would also go into a Risks and Mitigations table (see Table 1 for an example entry).

Table 1. Example entry for the Risks and Mitigations table

Risk area	Risk description	Failure mode	Mitigation strategy	Status
Self-inflicted DoS attack by own clients	Clients that synchronize polling or compound retries can saturate networks.	Product becomes unavailable to other users.	Clients must add exponential backoff and jitter wherever polling or retries are used.	Low launch risk—DONE

Evaluate lifecycle expectations for the product's dependencies. If the product depends on a deprecated system, replacing it with a modern alternative before the launch will be cheaper, if time permits—more so if mitigation costs associated with a deprecated system are large.

It may seem too late to do an architecture review by the time you're planning the product launch, but that's not the case—it's never too late to uncover blind spots, and you can defer implementing the mitigations until after the launch, if appropriate.

Check resource needs

Armed with traffic estimates and an understanding of the product's architecture and RPC flows, you can look at accuracy of resource provisioning. You will need to identify distinct classes of operations

in terms of resource use. You could then rely on synthetic or replay performance testing to evaluate how the product scales with the relative traffic or data growth, and how it behaves under edge conditions, some of which we describe shortly.

Consider a video sharing service as an example. It needs two fundamental features—upload and play back a video—and they are vastly different. Playback requires selecting the video instance encoded for the streaming client and sending bytes to the client. Uploading involves receiving the entire video, doing content analysis for policy violations, sending it through a pipeline for generating encoded instances, and creating video metadata. The upload and view features thus have different CPU, memory, and storage needs. The rates of use differ too; for example, expect fewer uploads than views per video. Add up the resources for all uploads separately from the resources for all views. As traffic estimates change, you can properly adjust the right resource types.

Appropriately provision resources geographically. You could expect different countries with different consumer technology to upload and view videos differently. Users in some places might be uploading videos of higher quality. Others might have lower internet bandwidth, resulting in lower-resolution views and few uploads.

Next, consider possible failures for each class of operation and how they might impact resource consumption. Look for opportunities to reduce resource use. Resource use is not a launch risk but a quality attribute of the product's susceptibility to the exposure to the entire world's traffic. This section covers some examples; record the possible failures in the Risks and Mitigations table.

When operations fail, any resources consumed up to that point might be completely wasted, so look for opportunities to return errors as early as possible. For example, when uploading a video, check for the video stream's format validity continuously, instead of validating the video once it is uploaded in its entirety. Another risk is from workflows that may cause severe resource drains. If a bug in transcoding of uploaded videos to all supported formats creates garbage frames, the system may not be allowed to re-encode all videos at once but instead be rate-limited to the safe fraction of CPU and network resources not consumed by current uploaders and viewers.

Similarly, a product may require significantly more resources when operating on a cold cache. If pointers to stored videos are lost when the cache clears, all video view requests start going to the video metadata service. A sufficiently large traffic spike might result in an outage. Launch capacity planning must ensure that this fact is not lost after launch, and the product continues to reserve enough capacity to avoid exposure to cascading failures.[2] To help measure the needed capacity, you could upload a large set of videos in QA and then flush the cache to see how the system responds. If it does well, then you gain confidence that the same might happen in production at similar load levels.

Evaluate the user experience

Traffic estimation and resource provisioning set up your product for sustaining its launched services under load from user traffic. To make sure the product provides *good* service that satisfies end user expectations, you need to document the product's availability and latency goals—the SLOs.[3]

Review whether SLOs have been well defined for the product's core features. Pay attention to the balance of the costs and value of SLO metrics. Achieving low latency or high availability gets more expensive quickly, but it may not be necessary in most cases. A chat application might need good latency and to never lose characters when users are typing in interactive sessions, but a video playing service can afford a few dropped frames or a larger delay before starting playback.

If your organization hasn't adopted SLO practices, recall that users naturally grow to expect certain performance, and those expectations become the implicit and effective SLOs. Try to predict these SLOs the best you can, for the most critical use cases.

Consider whether dependencies are providing SLOs from which your product could still meet its own SLOs. For inadequate SLOs, you may have to relax your product's SLOs, negotiate with the dependency's owners for a better SLO, or replace the dependency with another.

---

2 See Chapter 22, "Addressing Cascading Failures," in *Site Reliability Engineering*.

3 We assume your organization has selected and adopted relevant practices; to learn more, see Chapter 4, "Service Level Objectives," in *Site Reliability Engineering*.

Finally, review whether you have monitoring metrics sufficient to measure whether product properties are meeting all relevant SLOs. Keep in mind that it may be necessary to modify your product to enable the telemetry needed for SLOs or any other monitoring needs.[4]

Create product resilience plans

Look for scenarios that could put your SLOs at risk, and what options you can choose from.

Review how the product will behave if some of its components are failing or overloaded, whether it's a temporary spike or a permanent rise. This review will improve the chances of sustaining the user experience you expect the product to provide. Functional mitigations include implementing load shedding and throttling of requests incoming to a service,[5] as well as stabilizing behavior of the UI when some of its backends are fully or intermittently failing.

Review disaster recovery for your data. The product should have backups, and the teams should have experience recovering the data fully. Data corruption risks, especially if they can go unnoticed, must be added to your Risks and Mitigations table.

Evaluate Operational Sustainability

After focusing earlier on whether the product's architecture and underlying infrastructure can sustain public demand, let's now consider the capabilities of teams building and supporting the product, before and after it launches.

As the end users begin to rely on the product's availability, and even more so if ecosystems begin to develop around the product, the cost of product outages to your company increases. This scenario may lead your company to tighten the availability objectives (SLOs) for the product, which imposes more operating constraints and raises operating costs. Therefore, you need to be clear about how this might evolve and who will support the product through it. Because the cost of tightened SLOs rises as they get closer to 100%, eventu-

4 Your organization probably has monitoring products with which to integrate; read more in Chapter 4, "Monitoring," in the *Site Reliability Workbook* (O'Reilly).

5 Read more in Chapter 21, "Handling Overload," in *Site Reliability Engineering*.

ally likely exceeding the value of the product, expect to maintain a cost/value equilibrium as the value of a product changes over time. This scenario also means that poor adoption of the product might justify reducing SLOs for the product.

Selecting the operations teams

The goal of hiring managers is to verify that the teams supporting the product after it launches have sufficient operating capacity to absorb it, and some reserve to accommodate unexpected or miscalculated toil.[6]

First, estimate the operational costs for the *specific* team to run *this* specific product. The product architecture defines the technologies and infrastructure capabilities, which translate into specific costs. What's more critical is whether these technologies align well to the team's *existing* practices. Does adding this product dilute or improve the operating efficiency? Poor alignment will increase the cognitive load on the team while reducing the reuse of the production expertise and hands-on skills. The difference can be drastic!

For example, if the new product uses the same technologies as all the other products a team supports, the added operational costs may be limited to a fraction of the costs for running the product's resource footprint. A product that doubles the team's resource footprint might increase toil by only 5–10%. If, on top of that, the new product uses different technologies, operational costs also add up proportional to the number of new technologies, perhaps growing by 40–80%. These numbers are made up, but they illustrate the point that a typical team operating at 80% capacity can absorb an increase in operational costs of 5–10% but not 40–80%. By aligning the technologies the products select with the technologies SRE teams operate, you improve the products' sustainability past launch.

Second, estimate how the operating costs might change with the product's growth. Use the same timeline as you did for traffic projections. Expect that the resource footprint will increase, the rate of feature developments may increase, and some of the technologies might change—you'll have discovered these scenarios when evaluat-

6 Read more in Chapter 5, "Eliminating Toil," in *Site Reliability Engineering*.

ing the product's architecture. Make sure the teams will not run out of capacity[7] to support the products on the projected timeline.

Finally, evaluate how efficient the teams are at handling their existing operational load.[8] Is there room to absorb unexpected discoveries immediately past launch, or is the team just getting by? Is the team's health on a stable trajectory, or might adding another product endanger the team as a whole?

When the product is operated by the developer team and doesn't have an SRE team, the same concerns generally apply, though instead of selecting the right team you will be assessing the needs to staff the team appropriately or make any other changes to mitigate identified risks.[9]

Impact of technical debt

Consider technical debt within the product as a factor affecting the operational sustainability.

Timeline pressure for time to market or in product dependencies may require technical problems to be worked around and deferred for later resolution. While workarounds accomplish the timeline objectives, the future still holds a problem. Launches typically follow a pattern of getting user feedback and delivering on the next set of features. Any delays for technical debt will extend the timeline for bringing more value to the users. As technical debt continues to accumulate through these cycles, so will the proportion of bugs, rollbacks, and product stability efforts. Technical debt diminishes the capacity of the operations teams to support future launches.

Prioritizing debt reduction might depend on business needs at the moment. If the organization can consistently afford to hold bug-fixing sprints and feature freezes, debt reduction will help reduce toil and manage product costs.

---

7 Read more in Chapter 17, "Identifying and Recovering from Overload," in the *Site Reliability Workbook*.

8 Read more in Chapter 29, "Dealing with Interrupts," in *Site Reliability Engineering*.

9 Chapter 20, "SRE Team Lifecycles," in the *Site Reliability Workbook* offers guidance relevant to organizations of all stages of maturity and growth.

Accommodate Business Needs

Considering a product in isolation—that is, separate from its business value and impact—is a mistake. The engineers and SREs must work with the business partners to optimally balance overall needs for the organization. Working together applies to launch plans too: launch planning teams must always consider business requirements, not only to avoid idealistic expectations but also to prevent those requirements from driving unilateral product decisions. Considering business requirements may be easier or harder to accomplish depending on the size of the organization and its structure, and the relative importance of the product. For example, your company's product managers could work with sales and executive units to learn how the products map to the company's roadmap. Product incidents incur risks to the business, such as revenue loss from an outage; therefore, having a method to quantify these risks is important so that the business can make a decision about their potential impact. Quantifying risks enables launch planners to make a case for the value of planning; if work is done ahead of time, future incidents can be mitigated.

Perfect is the enemy of good

Evaluating a product for launch planning might highlight that the product is not perfectly ready for launch. That's OK. Launching a perfect product is not the goal. Rather, you might want to launch the product as soon as it can practically meet its end users' goals, without exposing the end users to undue risks or damaging the company's reputation. Launching a product as soon as it meets user goals delivers value sooner, and reduces losses from mistakes unrelated to the product's reliability.

Sometimes you can't avoid a launch because of a committed date, despite uncovering significant product or infrastructure issues. Launch planning processes can be especially valuable here. By their nature, they will provide a framework for exposing all relevant risks, which can then be stack-ranked against the available time and the probability of remediation. If any best practices evolve from your organizational experience with launch planning, they might provide higher-confidence mitigation options, as well as helping identify unreliable options.

Management must support launch planning

Launch planning effectively targets reliability, but reliability is harder to accomplish and sustain without management support. When stakeholders and product managers consistently allocate resources to new features rather than more reliability, the balance inevitably shifts toward more risk. This situation would be further compounded by the lack of a launch plan—launch planning can indeed be viewed as an impediment to delivering features, especially when it delays launch or requires more development costs. As a result, safe and successful launches become harder to achieve, and the costs of planning per launch rise; launches are less frequent, less rigorous, and provide less improvement from launch to launch.

Work toward engineering and management agreeing on the costs and benefits of launch planning. You want management who will stand behind and defend launch planning against pushback for higher feature velocity or cost-cutting. If you don't have this support, you should absolutely still try to plan before launching; just be prepared for reduced budget and power to effect change.

Launch Structure and Execution

From all the principles and guidelines described earlier, you will be building a list of specific, ordered activities to execute in preparation for and on the launch day. At each step, you should know what's required, what to do, and how to tell that it was completed. Have a clear owner accountable for the entire launch plan.

You could use a simple text document or build a comprehensive software system to manage launch planning at scale. Starting with a simple spreadsheet to manage the details is often easy; you can have separate sheets for launch prerequisites, actions on the day of launch, actions for days after the launch, contact and escalation information, and miscellaneous information. You can move into a more complex solution after using spreadsheets for a number of launches. However you manage the launch planning content, remember to select technologies that allow teams to collaborate on the launch plan and share it widely.

If you already have generic launch templates, you might scale them down to match the needs of a particular launch, making informed

decisions about what is irrelevant but with confidence that you have not overlooked the important bits.

Launch Stages

If possible, break up launches into stages (the launch template in the appendix includes two stages). The breakdown has two dimensions: by product status or features, and by the amount of diverted user traffic. You can combine the options from either dimension.

You could make some product features available first, deferring the other features until later stages for practical or logistical reasons. At Google, we formally distinguish Early Access Preview (EAP), Alpha, Beta, and General Availability (GA) launch stages. EAP and Alpha have only some of the final product features, though the riskiest features tend to launch there for earliest evaluation by actual customers. Beta and GA have full features but differ in the degree of production maturity and product support available to the end users.

You could slowly divert user traffic to the new features, instead of launching 100% everywhere. Start by opening the product to a small proportion of your customers. As you monitor progression from 1% to 2%, 5%, and 10%, you are ready to pause and roll back in the case of any incident, or move toward 100%. Alternatively, you could open the product to some limited geography, expanding first to one region, then deploying to all remaining regions. The specific choices depend on how many users are available to start using your product; the number should be large enough to provide meaningful quality estimation. You might value the slow progression also because it builds monitoring data to evaluate product impact under the increasing load.

However you choose to progress through a launch, determine if it has a point of no return. What might reaching the point of no return look like? A simple case is when media companies have published the public announcement for your product—the announcements cannot be "unpublished." Subtler cases may include industry events that cannot be moved, or business contractual commitments to deliver specific product features. When such circumstances reduce your flexibility, launch planning can provide you with some countermeasures.

If possible, "dark" launch your product or its new components. At Google, the "dark" property means that we avoid visibility of the

new features, while exposing them to the users. To implement a dark launch, you could modify your public clients to make requests on your product without rendering the results to users. The product would throw away the results as late as possible, perhaps in the frontend or even on the client. This process permits a much slower rollout schedule for extra safety, and can still involve canary testing over a gradually increasing percentage of users. Importantly, you can push the point of no return significantly past most of the points of the launch plan. Be careful not to abuse the end user's trust and computing resources.

Launch Checklists

A launch checklist organizes the remaining tasks for the launch (see the appendix for a preproduction checks example). Think of a launch checklist as a useful planning tool as well as an accountability and verification tool. If the product consists of multiple components, you can break down the checklist by each component, providing easy access to its launch status.

Because each launch needs its own checklist, you might want to create a reusable process to bootstrap them. Following this advice, Google starts with generic checklist templates we've built over time, rather than building checklists from scratch every time. Launch engineers rely on their experience and interviewing product engineers to determine what elements are applicable, trim away everything else, and reset all status fields. The guidelines included with the remaining action items provide a self-sufficient path to completing the entire checklist.

Launch checklists eventually become historic references to all the artifacts related to a product launch. Subsequent launch stages of the same product can rely on the earlier launch checklists to expedite planning.

Launch Actions and Status

If you think of the launch plan as a series of action items, each action item has at least these properties:

Timing
: When the action item is needed and how long it is expected to take.

Owner
: The contact accountable for this action item and responsible for managing and verifying it.

Executor
: Who will actually perform the action item. If possible, assign someone different from the owner. This separation of responsibilities allows the executor to focus on the actions and the owner to focus on processes and verification.

Actions
: One or two actions to perform, simple enough not to need further breakdown of ownership, communication, or verification.

Status
: Whether this action item has been started, blocked, completed, verified, and so on. This property gives all observers the same situational awareness—the launch at any moment is in a clearly defined and clearly communicated state.

Verification
: How to determine that the actions occurred completely and successfully.

Rollback
: How to undo the actions if needed. Having a rollback on hand is a responsible action, and capturing it at the time of defining actions is convenient.

Associate other properties with each action item if they will be useful later on. For example, describe whether the action item is required or recommended, attach documentation that the executor can follow to resolve the action item, or reference any relevant policies or standards. Aim for whatever helps the launch succeed or provides input for launch reviews, post-launch analysis, or future launches. Your organization will be able to develop reusable launch templates and continuously improve them.

Creating launch actions is very much like general project planning, but we supply it as a set of specific minimal practices we've found to work well at Google, for small and large launches.

Launch Controls

Minimizing the amount of change needed during the launch is important. Ideally, you only have to do simple production changes such as a self-contained runtime configuration change, and not a more complex change like restarting your application servers. Before the launch, you already have deployed all necessary code to production and it is available for real live verification.

A more complex launch plan might require a sequence of changes. During the launch, you can imagine going through a series of production changes where more and more product components begin to act according to the launch plan. We find that engineers are often more capable of reasoning about these configuration changes as opposed to other launch methods, such as the rollout of a new application version. Configuration changes are easy to visualize, which also improves understanding and code review.

Documenting Launch Decisions

Many decisions will be made on the way to launching a product. You should document all relevant deliberate decisions so that you know who made the choices and why. Documenting decisions avoids the need to conduct archaeology. You are aiming for discoverability of reasons for decisions, to enable an informed response at launch and evolution afterward. The goals include consistency of shared knowledge accessible to people who do not directly interact. Remember that even simple production changes can have a large impact, and you must be ready to escalate quickly. People new to the company should be able to find what they need to know quickly.

While the discoverability is most important closer to the actual launch, the reasoning and historic context for decisions grow in value further in the future. Architecture and configuration nuances might make little sense years after the launch, possibly because the underlying infrastructure has changed significantly. An easy solution is to link decision documents into launch checklists.

Anywhere coordination is necessary, plan and document it. This document should include who should coordinate what with whom and under what circumstances, how to find substitute experts, and what to do if key people are not available. Don't overlook knowing the business decisions that need to be made about product behavior

at launch, the ordering of any stack-ranked fallback options, what traffic patterns to expect, and who decides whether escalations are needed. Documenting coordination is about handling planned and unplanned cooperation between people.

Note that the reason for tracking who made decisions is information, not blame. You want to know who can provide more context for a decision. You might care whether someone had the technical or organizational authority to make the decision, or whether the decision needs to be escalated to establish it as a plan of record. You do not need to know whom to blame when decisions lead to poor outcomes.

Insights into Launch Progress

Make sure you know how to determine the progress of the launch—that is, how to compare the actual and the expected conditions at each action item, and evaluate whether to advance the launch or look for corrective measures. You will likely rely on launch checklists and the documented decisions covered in the previous section. However, the point of monitoring launch progress is for the people familiar with the infrastructure and communications to know how to get the right answers quickly.

Include status monitoring and verification proportionately to the production impact of each step. The larger the scope or destabilizing capacity of an action is, the more nuanced you will want the insights to be, and the more proactive communication of potential issues you will want.

Timing of action items should be carefully reviewed in advance to ensure correct ordering and allow enough time for each item. The allocated time should include time to complete operations, perform verification, allow for alerts to occur, and gather metrics. A common mistake is to assume that a simple action does not need much time—but simple actions may cause changes that take time to verify. In some cases, you may need to allow time to see if user interactions cause alerts. Rollbacks and retries can multiply the projected times at least twofold. Allow enough time to progress through each step. If you need to cut the time short, you are making a trade-off for more risk—only do so intentionally.

Be in a known, stable, and previously documented state at all times during launch. Include overnight holds if necessary, to ensure stable

pauses and to allow engineers time to recover and reassemble. Overnight holds should identify clear state and metrics that are being watched for stability.

Progress tracking officially starts when the final decision to "go" with the launch is made. At that point, you should send a notification to the stakeholders and all other relevant contacts. This notification informs them that the launch really is occurring. Sending the notification is important because as the plans were changing, people may have heard different versions of what will launch and when. The notification should include a link to the launch plan and details of how to communicate with the launching team if necessary.

Create a tracking ticket specifically for this launch, and publish it in the launch plan. Anyone interested in details can follow along for status updates, which you will communicate to the team. The tracking ticket also leaves a documentation trail for launch-related events and issues, which can be easily reviewed if needed.

Managing Precise Time of Launch

The action items that actually launch the product should coordinate carefully with messaging, public announcements, press embargo lifts, and other external communications. Consider whether you want press releases before or after the product is available. Include the relevant action items at the right place in the launch plan.

Beware of promising more than what the infrastructure and SRE teams can realistically deliver. For example, launching at precise times like at 10:00 a.m. may be difficult, so you should carefully communicate the timing of product availability to the public. Data pushes usually drive launches, and they might be run only at fixed intervals. On top of that, data pushes may fail, delaying the update until an emergency push or next scheduled push succeeds. Unexpected problems may cause rollback of a successful push. As you develop the launch plan, you will become familiar with the available precision.

Be prepared for a scenario where the press embargo ends prematurely, or someone leaks the launch (accidentally or intentionally), or both. If any of these scenarios happen, you may be under pressure to launch sooner than planned. During launch planning, consider which parts of the launch can be accelerated, which must not be accelerated, and what the risks of such accelerations are.

Communicating schedule changes with all contacts is especially critical, because if they are expecting events on the wrong schedule, you cannot count on their assistance even if they are available. Scheduling is a good point to consider in assembling a command center during launch, as discussed in the next section.

What to Do on Launch Day

The launch plan should specifically call out what to do on launch day, and who will perform the planned actions. Before launch day, the launch plan should have been circulated to all relevant teams—developers, production, marketing, press—and all required action items on the launch checklist must be marked as completed. Now that you are ready for takeoff, it's time to go over what might help with launch execution!

Organizing a Command Center

Despite all the preparation, you must be ready to react to changing conditions in the moment. If that time comes, knowing who has the responsibility, authority, and expertise appropriate for the situation is critical. A production crisis is not the time to be looking up who those people might be.

You will want to include people from all essential roles. Subject matter experts—for the launching product and its main components—will be able to reason about the product's behavior in unexpected situations. Any other experts who were involved in technical choices and spotting problems during preparations for launch will be valuable. On-call engineers for the dependent products will be able to provide sufficient assistance as part of their on-call shift, but do keep them informed and reminded of the planned launch. Members of marketing and press can assess what's happening and communicate externally if needed. You may need executives on hand for highly visible launches. A natural way to assemble such an extensive list of people is by making notes as you are building the launch plan; you can then just review the contacts for availability near the launch day.

Make sure everyone accountable is available; ideally, you will have backups for people in these roles. A launch may turn into a production incident, and preparing ahead of time reduces time to mitigate.

If your company has incident management processes,[10] consider reviewing them. Capture phone numbers, pager numbers if applicable, and email addresses, and verify that people will be available across time zones. Depending on the combined risks and required coordination, you might decide to all be present in the same place, physically or virtually; this place is sometimes referred to as a "war room" or "command center." During a "boring" launch, people in the command center could just do their normal work from their laptops; they'll be ready to jump into face-to-face discussions if needed.

Document for others in advance how everyone will communicate during the launch. What conference rooms will they use? Will they fit enough people if more people need to join? In addition to physical locations, create a chat room for internal communications. If your company offers multiple communication mediums, publish the backup solution. At Google, we fall back to IRC if Google Chat is not available.

Before you start, consider making a PDF or even a hard copy of the launch plan to protect against any service outages at a critical time. If not the entire plan, you need at least the contacts and the day-of-launch actions with instructions in case you get stuck mid-launch.

Transitioning from "Before Launch" to "After Launch"

The transition from pre- to post-launch is the critical step during launch. You are already most likely committed to the launch; the press release is out, for example. You are potentially exposing new code to public traffic it has never experienced before. Fortunately, your launch preparation steps will have reduced the risks to a practical minimum. All you have to do is have action item executors and owners go through them one by one.

In a classic sense, the transition consists of the production changes that expose the product to public traffic. For staged launches, it could mean merely increasing the size of the public user base, such as going from Alpha to Beta. Some transitions could last for days, if gradual traffic ramp-up was planned.

10 Read about incident management at Google in Chapter 9, "Incident Response," of the *Site Reliability Workbook*.

You might notice that so far we have focused on things that happen in preparation for launch, and spent only a short segment covering the day of launch and the transition. Our goal in this approach was to represent the proportion of effort we allocate to planning, and the quality we strive for in that preparation. In our experience, lack of preparation tends to shift the costs toward the day of the launch, and maybe more days recovering from a launch gone poorly. These costs could take the form of an "all hands on deck" response or unrecoverable opportunity loss. Where the balance should be for a particular launch is up to you.

After the Launch

The launch typically doesn't end on launch day. Any abnormalities identified and mitigated during the launch need sustainable, permanent solutions.

Status should be actively or at least occasionally monitored until a reasonable demand cycle has completed, meaning that the initial traffic spikes have passed and more periodic traffic patterns can be expected to start to repeat. This cycle could be days or weeks for different organizations, depending on the business and the behaviors of its customers.

Multiple demand cycles with distinct periods could be overlaid when useful. For example, a day-night-day cycle captures the full 24-hour demand cycle of a product. A weekday–weekend cycle captures a regular week for many cultures, with spikes or drops during the weekdays or weekend. Sometimes a single spike of demand is the most telling, such as a flash sale at a retailer between 9:00 and 11:00, and you would not need to track an entire weekday–weekend cycle.

Not everything in the launch process needs to happen for every launch. There are always trade-offs between doing more work up front (an investment) and achieving some kind of benefit or business value. Next we'll look at a case study that demonstrates how to navigate those trade-offs. It covers the activities undertaken by the *Dauntless* team to achieve success, highlighting launch planning activities that were valuable.

Case Study: Lessons Learned from a Product Launch

Google and Phoenix Labs worked together to prepare for the product launch of *Dauntless*, a free-to-play role-playing game (RPG). Created by Phoenix Labs, *Dauntless* supports cross-play between different platforms such as the PlayStation 4, Xbox One, and the Epic Games store. Within a week of its launch the game had millions of players. How did the team manage the launch process and achieve a successful launch? How can you learn from this example when executing your own launch process?

Planning Ahead of Time

The plan began early, with a pairing of Google Cloud and Phoenix Labs to ensure the success of the launch. While the game was launched publicly in May 2019, planning itself began in mid-2018. Do not underestimate how valuable this lead time is—forgoing it is a key factor in launch failures. Most products have some kind of plan, even if it's business-focused; however, planners often neglect stakeholders until the last minute, which reduces the time to resolve issues. The *Dauntless* plan began long before the launch itself, providing a lot of time to execute the plan and enable a successful launch.

Holding a Scaling Session

The *Dauntless* team underwent a scaling review to evaluate how the game would operate in the production environment. One practical addition was to employ a "worst case" exercise—that is, imagining the worst-case scenario for demand for the game. If you expected a million players and 10 million showed up, what would happen? Would the game work? If not, what would break first in the game? This is commonly called a "10X exercise."[11] The goal is to expose

11 10X is a general rule of thumb based on Google's experience engineering software systems. In our experience, systems do not fare well when demand is 10 times higher than what the system was designed to handle. These systems typically require major engineering changes in order to service the heightened demand; this requirement is why this exercise is valuable, as it can deliver these pain points before the scenario occurs in production.

architectural pain points of a system by exploring where you expect the system to begin breaking down.

The result of this exercise was a series of risks similar to the Risk categories described in Table 1. The team grouped these risks into buckets according to their likelihood of occurrence. Risks that were likely to occur had mitigations or scaling plans. Less likely risks had a lower priority. Grouping risks served as an aid to the planning team to prioritize the work appropriately.

Load simulation

In the scaling review session, we were often relying on the experience of engineers, architects, and developers to point out likely pain points of the existing workload and its architecture. While pointing out pain points is useful, the humans involved are unlikely to brainstorm all problems ahead of time. In the case of *Dauntless*, the team also prioritized a production-scale load test to try to simulate its potential user base.

Load simulation can be costly. The *Dauntless* team had to run an instance of the game as well as pay for computing capacity to generate the game activity of thousands of players. Load simulation is not free from capital cost as well as operational costs; instead of working on the game, developers built testing infrastructure.

However, the benefit is typically worth the cost of such an exercise. A number of key learnings came out of the simulation. The advantage of the simulation is finding these potential problems ahead of launch day, discussing their impact on the business, and then prioritizing fixes to the product. Finding issues before users have access to the product can help provide a smooth user experience during the launch as well as after.

Denial of service

Like many internet-facing applications, *Dauntless* has a component that attempts to detect and mitigate denial-of-service (DoS) attacks. This component was triggered during the load simulation, causing the component to attempt to "mitigate" the load simulation. This attempt resulted in a partial production outage and halted the load simulation. Ideally the load simulation would produce no problems; in practice, that is rarely the case. The *Dauntless* team learned that the DoS component did function, they saw how the game operated

when the component was active, and they gained experience in configuring the component in a real incident. This practical experience is valuable in crafting improved procedures for future incidents.

Reviewing the Architecture Design

Google's Customer Reliability Engineering team offered a design review for the *Dauntless* architecture. The design review relies on Google's expertise in running large-scale systems. Google has done hundreds of reviews of production systems and has done design reviews with other Google Cloud Platform (GCP) customers. The intent of the review is to highlight areas that seem to contradict lessons learned from our experience with Google's production platform, as well as customers on GCP.

The result of the design review was a list of potential areas for development. We ranked these areas based on their potential impact to the product and business, and provided some prospective mitigations. The intent was not to fix every item called out; instead, we wrote short plans of action to mitigate risk. Prioritizing and planning in this way allows us to avoid costly development investments and complexity the product may not actually need. We can determine if the complexity and investment are needed later, after a load simulation or even after launch day.

Table 2 lists some of the items raised in the *Dauntless* design review.

Table 2. Risks raised during the design review exercise with notes and mitigation

Area	Risk	Notes	Mitigation
Game APIs did not have autoscaling enabled.	High	Large surges in player count are expected, and the game APIs should be able to scale up and down automatically without human intervention. This will enable automated operation of game APIs, instead of 24/7 human operation.	Team agrees that autoscaling should be enabled before launch day; will determine what blockers are necessary to be resolved prior.

Area	Risk	Notes	Mitigation
The Content Delivery Network (CDN) might go down, taking the game offline.	Low	*Dauntless*'s CDN vendor has pretty good uptime, historically speaking.	Mitigation (e.g., paying for two separate CDN vendors) tends to be expensive and operationally complex. CDN failure is rare. We accept this risk; no recommended actions.
Database: has the team executed a successful restore from a database replica?	Medium	Possible in theory to restore database state from a backup/replica, but this has never been done in production.	Action: try it and confirm we can do it. Document the process.
Database: concern that writes might overwhelm the throughput of the datastore without a sharding strategy.	High	Sharding the database can provide greater total throughput for the database layer.	Action: the *Dauntless* team committed to sharding some of the databases in order to sustain gaming performance.

Designing for Resilience

Dauntless's architecture helped the team remain agile in the face of unexpected events. Multiple Google Cloud regions host the *Dauntless* game world. Typically, players form a hunting party and are dispatched to a specific game server for the hunting portion of the game. *Dauntless* uses a latency test to find the closest server to the users to provide them with the best experience. However, sometimes the closest server is full and cannot accept additional players. Serving those users from the next closest server, sometimes in a different geographic region, then becomes a benefit. For example, a group of players on the East Coast of the US might play on a West Coast server.

This feature provides a strong failover and balancing capability to Phoenix Labs. If a region is having issues, is unstable, or is providing a bad user experience, Phoenix Labs can direct users to other nearby regions while still offering a suitable gaming experience. The work required to build this capability is notable—some of the regions are far apart (60–80 ms round trip), and this additional latency can impact gameplay. The *Dauntless* team had to make significant investments in their game platform to support this kind of experience at a satisfactory level.

Adapting Post-Launch

Once *Dauntless* launched, the launch process was not over; in fact, it had only just begun. Some organizations adopt a waterfall approach and think, "Hey, we did all this launch planning and we adopted all the critical mitigations, so everything will go fine on launch day, and if anything goes wrong it means we failed." This was not the approach the *Dauntless* team took to for launch day. They anticipated that things could go awry; the value of the launch planning is not in ensuring a perfect event-free launch, but instead in its ability to provide the necessary background and support so the organization can be agile when things go wrong.

Autoscaling

The design review had called out autoscaling—that is, automatically scaling capacity up or down with player demand—as an item that should be enabled prior to launch day. Autoscaling was appropriately staffed and deployed; however, an unforeseen scaling issue caused the autoscaler to malfunction during the launch. It turned out the autoscaler would have to be modified mid-launch in order to achieve its goals. Modifying the autoscaler is a daunting[12] task to undertake in the middle of a game launch, but a meeting of the minds between Phoenix Labs and Google produced a new autoscaler design that was deployed quickly to mitigate the situation.

Database

The "worst case," or 10X, review had called out the database as a bottleneck that scaled 1:1 with player count. As the game continued to gain players in the first couple of days, the database began to hit bottlenecks. The database has a primary with a number of secondary replicas. One problem the team experienced was that the replicas were struggling to keep up with the primary, causing them to fall further and further behind. Out-of-date replicas eventually become ineffective for serving users, leading to a loss of serving capacity, lack of resilience, and extra expense of paying for replicas that provide no value. The *Dauntless* team did not observe this behavior during the load tests, because replication was not enabled then to

12 Pun intended.

save on testing costs—leading to the first occurrence of this problem in production during the launch of the game.

The *Dauntless* team anticipated this problem in the design review of the game architecture, so they had some mitigation ideas planned. Only a subset of the database was experiencing a high write rate, and the team migrated these tables into a different storage system that could sustain that rate.

Login Queue

As players rapidly show up to play the game—at game launch, but also during other times of day or game events—having players log in all at once can lead to uneven demand on the game servers. Sometimes uncontrolled demand happens, which can cause instability for players already in the game. A lack of demand control from end user devices means that recovering from instability or routine game maintenance, like a software rollout, can be difficult. Phoenix Labs decided to provide a smooth gameplay experience for a preset number of players instead of a degraded experience for all players.

The Login Queue is a feature that Phoenix Labs can enable to control the speed at which players are admitted into gameplay. This feature enables the company to offer a consistent playable experience for players who have already joined the game, while communicating to queued players when they might be able to play. Controlling the speed is often better than providing an unstable gaming experience for all players. It allots room for the game operators to control the demand for gaming resources and to smooth out spikes in demand from players. Instead of having a poor gaming experience—players' games freezing, getting kicked out, or getting stuck at a loading screen—excess player demand goes into a queue that Phoenix Labs manages. The queue is not on all the time, but Phoenix Labs can enable it opportunistically during periods of instability or unexpected player demand in a region.

Lessons Learned

The *Dauntless* launch utilized a number of launch activities, including an architecture review, a scaling session, and post-launch modifications to the application architecture. We did not discuss some of the common launch activities, such as capacity planning and setting up a command center for launch day; in our experience many organizations are doing these activities already.

Dauntless benefited from the flexibility to adjust to unknown factors both during the launch and after, specifically with regard to demand for the game: how many concurrent players would there be for the game, and how would the game respond? Instead of spending significant time and resources trying to formulate perfect estimates or making the game perfectly responsive, the team focused on implementing the Login Queue to manage demand for the game resources.

We can highlight a few more useful practices. The scaling exercise and architecture review are exercises designed to explore what the team knows about an application and to aid in decision making about what mitigations to build into the application. Trading off features against launch velocity is an important exercise. Brainstorming potential problems and solutions is routinely cheaper than implementing such a solution. Often we can do the on-demand engineering work post-launch, to implement the discussed solutions. Planning exercises eliminate the surprise of potential problems we might encounter and allow us to offer predesigned solutions, reducing the stress of post-launch management.

Wrapping Up

In this report we organized the many different areas of launch planning, offering a selection of relevant examples based on lessons learned at Google. We hope these examples will be useful for you to build upon to develop launch practices that fit your specific needs. You could look at this report as an aid to help you determine where your product could use help on the way to its users. Use it to aid you in building a launch checklist, and follow the checklist to launch.

Aim to sustain launch planning practices over time by scaling the launch planning process to the size of the launch and the teams involved. Some products might launch fine with little planning,

while others could require long and rigorous planning efforts. Some organizations might have small operations teams. Others might have just one product and launch product features instead of entire products. Only you know what amount of planning will benefit each particular launch. By practicing launch planning, your organization can build up the degree of proficiency it needs.

Remember that for a successful launch, planning isn't optional. Plan your launches and launch successfully with confidence!

Appendix: Launch Plan Template

This is a generic template for a launch plan. Google does not have a single unified template, and often different business units will have their own customized versions for their product lines. Keeping that in mind, feel free to customize this template to your own business by adding or removing sections, processes, approvers, links to artifacts, and so forth.

Text formatted [*in italics*] is the text that the launch plan owner is expected to delete or replace with launch-specific text.

Status

[*Adjust this list as needed and remove stakeholders that are not relevant.*]

Stakeholder role	Email	Signoff
Marketing		
Security		
Site Reliability Engineering		
Product Manager		
Technical Program Manager		

Summary

[*Write a few sentences about what, why, and when you are launching. Describe the scope of the launch—include the list of changes, and what is not in scope.*]

Related Documents

[*Include links to documents relevant to this launch. Common documents include:*]

- Link: Privacy document
- Link: Design document
- Link: Test plan
- Link: Rollout plan

Testing Guidelines

- Environment: [*QA, staging, etc.*]
- Whitelisting: [*Instructions on how to whitelist a customer to access new functionality.*]
- What to test: [*Define a scope for what people should be testing and what to look for.*]
- Other information:
  - One-time manual and/or exploratory tests (new features): [*Link to QA test request*]
  - Number of manual regression tests added:
  - Number of integration/large tests added:
  - Unit test coverage %: [*If below required 70% coverage, include rationale for exception*]
  - Test signals: [*Unit tests, integration tests, continuous build status, etc.*]
  - Bug triage done: [*Yes*, *No*, *N/A*]

Checks for Trusted Tester Stage

[*Checks in this and the following sections cover many possible approvals, some of which will not be relevant for your launch. Feel free to remove rows that are not needed.*]

Check	Owner	Note
Dev code and test complete		[*Any exceptions or other concerns.*]
QA testing complete and no blocking bugs		
UX and PM signoff on feature		[*Any exceptions or other concerns.*]
All the dependent servers have the correct permission granted		

Checks for Production Stage

Check	Owner	Note
QA test complete and no production-blocking bugs		[*Any exceptions or other concerns.*]
Final signoff from non-eng partners (L10N, PR, legal, marketing, security, privacy, export, open source, etc.)		[*Localization of strings, help center articles, dev notifications, legal reviews.*]
Support page updated		
Early Access Program (EAP) or Trusted Tester Program (TTP) process		
Signoff from engineering partners (upstream and downstream dependencies)		
Production readiness (capacity planning, configurations, load testing, onboarding, etc.) complete		
Launch monitoring and alerts in place		
Product metrics verified		
Email to stakeholders with a heads up sent		

Checks for Post-Launch Stage

Check	Owner	Note
QA automation added to daily regression suite		
Experiment cleanup complete		

Rollout Schedule

[*Adjust these steps and percentages to the needs of your launch. If an experiment document exists, point to that; don't put dates in multiple locations.*]

Rollout step	Date	Note	Owner
Trusted tester		[*Required, 5+ days.*]	
Production 1%		[*Check basic health metrics and catch exceptions for ~1 day. Get SRE approval before each bump in traffic.*]	
Production 5%		[*Run long enough to get significant results for launch monitoring metrics.*]	
Production 20%			
Production 100%			
Experiment cleanup			

Launch Monitoring

[*List each team or component involved in the launch here:*]

- Component A (Owner team X)
  - [*List all dashboards and metrics you will be monitoring to ensure the launch is going as expected. These include basic service health metrics.*]
- Component B (Owner team Y)
  - [*List all dashboards and metrics you will be monitoring to ensure the launch is going as expected. These include basic service health metrics.*]

Plan for Emergency Rollback

[*Cover at least the following:*]

- Who are the decision makers?
- If rollback is needed and approved, how do we roll back?
- What is the impact of rollback?

Product Success Metrics

[*What logging is available to measure the success or failure of this feature?*]

About the Authors

Alec Warner is a senior site reliability engineer who has been at Google since 2007. His early work focused on traditional IT systems administration, while his later work focused on operating planet-scale storage systems. Today he works on Customer Reliability Engineering, assisting customers in operating reliably on Google Cloud Platform. Outside of work Alec likes space, hiking, climbing, and adventurous activities.

Vitaliy Shipitsyn is a staff software engineer in Site Reliability Engineering at Google. He works on the resilience of Google's production infrastructure, and on convergence and adoption of production best practices across Google. Vitaliy's experience includes several years of consulting for internal and external product launches at Google, helping engineering teams adopt SRE practices, and running services for BeyondCorp at Google. He obtained his MS degree from Ohio University in 2000.

Carmela Quinito is a technical writer for Site Reliability Engineering at Google. She writes documentation on products that help Google engineers manage their production systems. Prior to Google, she worked in technical support for 10 years. She holds a computer engineering degree from the University of the Philippines-Diliman.