

O'REILLY®

Modern System Administration

Building and Maintaining Reliable Systems



Early
Release

RAW &
UNEDITED

Jennifer Davis
with Chris Devers

Modern System Administration

Building and Maintaining Reliable Systems

With Early Release ebooks, you get books in their earliest form—the author’s raw and unedited content as they write—so you can take advantage of these technologies long before the official release of these titles.

Jennifer Davis
with Chris Devers



Modern System Administration

by Jennifer Davis with Chris Devers

Copyright © 2020 Jennifer Davis. All rights reserved.

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: Katherine Tozer

Interior Designer: David Futato

Cover Designer: Karen Montgomery

Illustrator: Kate Dullea

August 2020: First Edition

Revision History for the Early Release

- 2019-09-24: First Release
- 2020-01-10: Second Release
- 2020-04-09: Third Release
- 2020-06-26: Fourth Release
- 2020-09-14: Fifth Release
- 2020-12-11: Sixth Release

- 2021-03-05: Seventh Release
- 2021-06-30: Eighth Release
- 2021-10-06: Ninth Release
- 2021-11-23: Tenth Release
- 2022-02-09: Eleventh Release
- 2022-04-22: Twelfth Release

See <http://oreilly.com/catalog/errata.csp?isbn=9781492055211> for release details.

The O'Reilly logo is a registered trademark of O'Reilly Media, Inc. *Modern System Administration*, the cover image, and related trade dress are trademarks of O'Reilly Media, Inc.

While the publisher and the author have used good faith efforts to ensure that the information and instructions contained in this work are accurate, the publisher and the author disclaim all responsibility for errors or omissions, including without limitation responsibility for damages resulting from the use of or reliance on this work. Use of the information and instructions contained in this work is at your own risk. If any code samples or other technology this work contains or describes is subject to open source licenses or the intellectual property rights of others, it is your responsibility to ensure that your use thereof complies with such licenses and/or rights.

978-1-492-05514-3

Chapter 1. Introduction

A NOTE FOR EARLY RELEASE READERS

With Early Release ebooks, you get books in their earliest form—the author’s raw and unedited content as they write—so you can take advantage of these technologies long before the official release of these titles.

This will be the 1st chapter of the final book. Please note that the GitHub repo will be made active later on.

If you have comments about how we might improve the content and/or examples in this book, or if you notice missing material within this chapter, please reach out to the editor at vwilson@oreilly.com.

While the underlying concepts like managing capacity and security have remained the same, system administration has changed over the last couple of decades. Early administration required in-depth knowledge of services running on individual systems. Books on system administration focused on specific services on the systems from printing to DNS. The first conference dedicated to system administration, LISA, described large scale as sites for over 100 users.

Now, operations engineers are faced with an ever-growing list of technologies and third-party services to learn about and leverage as they build and administer systems and services that have thousands to millions of users. Software development is moving fast, and sysadmins need to move as quickly to accommodate and deliver value.

I wrote this book for all the experienced system administrators, IT professionals, support engineers, and other operation engineers who are looking for a map to understanding the landscape of contemporary operation tools, technologies and practices. This book may also be useful to developers, testers, and anyone who wants to level up their operability skills.

In this book, I examine the modernization of system administration and how collaboration, automation, and system evolution change the fundamentals of

operations. This book is not a “how-to” reference, as there are many quality reference materials to dig into specific topics. Where relevant, I recommend materials to level your skills in that area. I provide examples to guide a deeper understanding of the essential concepts that individuals need to understand, evaluate and execute on their work.

The focus is on tools and technologies in broad use currently, but progress is rapid with new tools and technologies coming into use all the time. These new tools may supplant today’s favorite tools with little notice. Don’t worry about learning the wrong tools; study the underlying concepts. Apply these concepts to evaluate and adopt tools as they become available.

At its core, modern system administration is about assessing and regulating risk to the business. It encompasses changes in how sysadmins collaborate with development and testing, deploy and configure services, and scale in production due to increased complexity of infrastructure and data generation.

Principles

The first part of the book focuses on the number of technical practices. These include:

- Version Control is a practice that enables the organization, coordination, and management of objects. It’s the foundation of automating software development and delivery with continuous integration and continuous deployment.
- Local Development Environment is a practice of standardizing on a set of tools and technologies to reduce challenges to collaboration and leverage work that has been done to set up an environment. It empowers teams to choose tools intentionally.
- Testing is a practice of getting explicit feedback about the impact of change. It’s another critical part of automation and continuous integration and continuous deployment.
- Security is the practice of protecting hardware, software, networks, and data from harm, theft, or unauthorized access.

You can't be the lone sysadmin anymore known for saying "no." The nature of the work may start at understanding operating systems, but it spans across understanding services across different platforms while working in collaboration with other teams within the organization and potentially external to your team. You must adopt tools and practices from across the organization to better perform your job.

You need to be comfortable with using the terminal and graphical interfaces. Just about every tool I'll cover has some aspect of command line usage. Being able to explore and use the tools helps you understand when problems arise with the automation. When you have to debug the automation, you need to know whether it's the tool or your use of the tool.

You can't ignore version control. For years, DORA's annual State of DevOps report has reported that the use of version control highly correlates to high IT performers.¹ Version control is fundamental to collaboration with other parts of the organization whether you're writing code to set up local development and test environments or deploying applications in a consistent and repeatable manner. Version control is also critical for managing your documentation whether it's README's embedded in a project repository, or as a separate project that spans content for the organization. You administer tests of the code you write, as well as the infrastructure that you build within version control.

You build and maintain virtual images and containers for use locally as well as within the cloud. All of this requires some understanding of how to read, debug, and in some cases write code in a particular language. Depending on the environment, Ruby, Python, or Go may be in use.

NOTE

While I include some code snippets in various languages, this book cannot cover the multitude of information that's important to learn a specific language. While you can (and should) specialize in a specific language, don't limit yourself to a single language as languages do have different strengths. Early Linux administration focused on bash or Perl scripts. Now individuals may additionally use Go, Python, or Rust. Folks who limit their ability to adopt other languages will hinder their employability as new tools evolve.

Whether you are collaborating on a project with development, or just within

your role-specific Operations team, you need to define and build development environments to replicate the work quickly that others have done. You then can make small changes to projects — whether they are infrastructure code, deployment scripts, or database changes — before committing code to version control and having it tested.

Modernization of Compute, Network and Storage

The second part of the book examines the contemporary landscape, or general conditions under consideration to lay a foundation for choosing the right options in alignment with requirements now and how to make changes as conditions evolve.

Compute

Virtualization technology set the stage for cloud computing, and containers further transformed the application infrastructure landscape. Serverless computing allows individuals to focus on application build and run in exchange for paying a hosted service for maintaining the server infrastructure as needed.

- Virtualization
- Containers
- Serverless
- Compute

Network

There are more than 20 billion connected devices as of 2021. This includes business and manufacturing robotics, cars, smart home devices, healthcare devices, security systems, phones, and computers. The more devices that need to communicate, the more network bandwidth is needed to enable connection between devices.

Storage

Storage choices have evolved and while storage is a commodity the data stored is not. The choices made about the data - how and where it's stored - impact what can be done with the data.

Infrastructure Management

The third part of this book covers managing infrastructure. Systems administration practices that work well when managing isolated systems are generally not transferable to cloud environments. Storage and networking are fundamentally different in the cloud, changing how you architect reliable systems and plan to remediate disasters.

For example, network tuning that you might handcraft with `ttcp` testing between nodes in your data centers is no longer applicable when your cloud provider limits your network capacity. Instead, balance the abilities gained from administering networks in the data center along with in-depth knowledge about the cloud providers limits to build out reliable systems in the cloud.

In addition to version control, you need to build reusable, versioned artifacts from source. This will include building and configuring a continuous integration and continuous delivery pipeline. Automation of your infrastructure reduces the cost of creating and maintaining environments, reduces the risk of single points of critical knowledge, and simplifies the testing and upgrading of environments.

Scaling Production Readiness

The fourth part of the book covers the different practices and processes that enable scaling system administration. As a company grows, monitoring and observability, capacity planning, log management and analysis, security and compliance, on-call and incident management are critical areas to maintain, monitor and manage risk to the organization.

The landscape of user expectations and reporting has changed with services such as Facebook, Twitter, and Yelp providing areas for individuals to report their dissatisfaction. To maintain the trust of your users (and potential users), in addition to improvements to how you manage and analyze your logs, you need to update security and compliance tools and processes. You also need to

establish a robust incident response to issues when we discover them (or worse when our users find them).

Detailed systems monitoring adds application insights, deeper observability, and tracing. In the past, system administration focused more on system metrics, but as you scale to larger and more complex environments, system metrics are less helpful and in some cases not available. Individual systems are less critical as you focus on the quality of the application and the impact on your users.

Capacity planning goes beyond spreadsheets that examine hardware projections and network bandwidth utilization. With cloud computing, you don't have the long lead times between analysis of need and delivery of infrastructure. You may not spend time performing traditional tasks such as ordering hardware, and "racking and stacking" of hardware in a data center. Instance availability is near instantaneous, and you don't need to pay for idle systems anymore.

Whether containerized microservices, serverless, or monolithic applications, log management, and analysis needs have become more complex. The matrix of possible events and how to provide additional context to your testing, debugging, and utilization of services is critical to the functioning of the business.

The system administrator role is a critical role that encompasses a wide range of ever-evolving skills. Throughout this book, I share the fundamental skills to support architecting robust highly scalable services. I'll focus on the tools and technologies to integrate into your work so that you can be a more effective systems administrator.

A Role by any Other Name

I have experienced a dissonance over the last ten years over the role "sysadmin". There is so much confusion about what a sysadmin is. Is a sysadmin an operator? Is a sysadmin the person with root? There have been an explosion in terms and titles as people try to divorce themselves from the past. When someone said to me "I'm not a sysadmin, I'm an infrastructure engineer", I realized that it's not just me feeling this.

To keep current with the tides of change within the industry, organizations have

taken to retitling their system administration postings to devops engineer or site reliability engineer (SRE). Sometimes this is a change in name only with the original sysadmin roles and responsibilities remaining the same. Other times these new titles encompass an entirely new role with similar responsibilities. Often it's an amalgamation of old and new positions within operations, testing, and development. Let's talk a little about the differences in these role titles and set some common context around them.

DevOps

In 2009 at the O'Reilly Velocity Santa Clara conference, John Allspaw and Paul Hammond co-presented "10+ deploys per day: Dev and Ops Cooperation at Flickr". When a development team is incentivized to get features delivered to production, and the operations team is incentivized to ensure that the platform is stable, these two teams have competing goals that increase friction. Hammond and Allspaw shared how it was possible to take advantage of small opportunities to work together to create substantial cultural change. The cultural changes helped them to get to 10+ deploys per day.

In attendance for that talk, Andrew Clay Shafer, co-founder of Puppet Labs tweeted out:

*Don't just say 'no', you aren't respecting other people's problems...
#velocityconf #devops #workingtogether*

—Andrew Clay Shafer (@littleidea)

Having almost connected with Shafer at an Agile conference over the topic of Agile Operations, Patrick Debois was watching Shafer's tweets and lamented not being able to attend in person. An idea was planted, and Debois organized the first devopsdays in Ghent. Later Debois wrote "And remember it's all about putting the fun back in IT"² in a post-write up of that first devopsday event. So much time has passed since that first event, and devopsdays has grown in locations³, to over 70 events in 2019 with new events started by local organizers every year.

But what is devops? It's very much a folk model that gets defined differently depending on the individual, team, or organization. There is something about devops that differentiates practitioners from nonpractitioners as evidenced by the

scientific data backed analysis performed by Dr. Nicole Forsgren in the DORA Accelerate DevOps Report.⁴

At its essence, I see devops as a way of thinking and working. It is a framework for sharing stories and developing empathy, enabling people and teams to practice their crafts in effective and lasting ways. It is part of the cultural weave of values, norms, knowledge, technology, tools, and practices that shape how we work and why.⁵

Many people think about devops as specific tools like Docker or Kubernetes, or practices like continuous deployment and continuous integration. What makes tools and practices “devops” is how they are used, not the tools or practices directly.

Site Reliability Engineering (SRE)

In 2003 at Google, Ben Treynor was tasked with leading a team of software engineers to run a production environment. Treynor described SRE as “what happens when a software engineer is tasked with what used to be called operations.”

Over time SRE was a term bandied about by different organizations as a way to describe operations folks dedicated to specific business objectives around a product or service separate from more generalized operations teams and IT.⁶ In 2016, some Google SREs shared the Google specific version of SRE based on the practices, technology, and tools in use within the organization in the Site Reliability Engineering book⁷. In 2018, they followed it up with a companion book “The Site Reliability Workbook” to share more examples of putting the principles and practices to work.

So what is SRE? Site Reliability Engineering is an *engineering* discipline that helps an organization achieve the *appropriate* levels of *reliability* in their systems, services, and products.

Let’s break this down into its components starting with *reliability*. Reliability is literally in the name “Site Reliability Engineer” so it makes sense. Reliability is a measurement of how well a system is performing. But what does that really mean? It is defined differently depending on the type of service or product that is being built. Reliability can be availability, latency, throughput, durability, or

whatever else your customer may be evaluating to determine that the system is “ok”.

Being an engineering discipline means that we approach our work from an analytical perspective to design, build, and monitor our solutions while considering the implications to safety, human factors, government regulations, practicality and cost.⁸

One of the strong evolution points from regular system administration work was the measurement of impact on humans. This work has been described as toil due to the work being repetitive and manual. Google SRE implemented a cap of 50% toil work, redirecting this work to development teams and management including on-call responsibilities when the toil exceeded the cap.⁹

By measuring the quality of work and changing who does the work, it changes some fundamental dynamics between ops and dev teams. Everyone becomes invested in improving the reliability of the product rather than a single team having to carry the brunt of all the support work of trying to keep a system or service running. SRE teams are empowered to help reduce the overall toil.

RESOURCES FOR EXPLORING SRE

Learn more about Google SRE from the [Site Reliability Engineering](#) and [The Site Reliability Workbook](#) books.

Read [Alice Goldfuss’s “How to Get into SRE”](#) and [Molly Struve’s “What It Means To Be A Site Reliability Engineer”](#) blog posts.

How do Devops and SRE Differ?

While devops and SRE arose around the same time, devops is more focused on culture change (that happens to impact technology and tools) while SRE is very focused on changing the mode of Operations in general.

With SRE, there is often an expectation that engineers are also software engineers with operability skills. With DevOps Engineers, there is often an assumption that engineers are strong in at least one modern language as well as have expertise in continuous integration and deployment.

System Administrator

While devops and SRE have been around for approximately ten years, the role of system administrator (sysadmin) has been around for much longer. Whether you manage one or hundreds or thousands of systems, if you have elevated privileges on the system you are a sysadmin. Many definitions strive to define system administration in terms of the tasks involved, or in what work the individual does often because the role is not well defined and often takes on an outsized responsibility of everything that no one else wants to do.

Many describe system administration as the digital janitor role. While the janitor role in an organization is absolutely a critical role, it's a disservice to both roles to equate the two. It minimizes the roles and responsibilities of each.

A sysadmin is someone who is responsible for building, configuring, and maintaining reliable systems where systems can be specific tools, applications, or services. While everyone within the organization should care about uptime, performance, and security, the perspective that the sysadmin takes is focused on these measurements within the constraints of the organization or team's budget and the specific needs of the tool, application, or service consumer.

NOTE

I don't recommend the use of devops engineer as a role. Devops is a cultural movement. This doesn't stop organizations from using *devops* to describe a set of tasks and job responsibilities that have eclipsed the role sysadmin.

I've spent a fair amount of time reading job requirement listings, and talking to other folks in the industry about devops engineers. There is no single definition of what a devops engineer does in industry (sometimes not even within the same organization!).

While engineers with "devops" in their title may earn higher salaries than ones with "system administrator"¹⁰, this reinforces the adoption of the title regardless of the lack of a cohesive set of roles and responsibilities that translate across organizations.

Having said that, "devops engineer" is in use. I will try to provide methods to derive additional context to help individuals understand how to evaluate roles with the title in comparison to their current role.

Finding Your Next Opportunity

One of the reasons you might have picked up this book, is that you've been within your position for awhile, and you're looking to your next opportunity. How do you identify positions that would be good for your skills and experiences and desired growth? Across organizations, different roles mean different things, so it's not as straightforward as just substituting a new title and doing a search. Often it seems the person writing a job posting isn't doing the job being described, as the postings will occasionally include a mishmash of technology and tools.

A danger to avoid is thinking that somehow there is some inherent hierarchy implied by the different roles even as some folks in industry or even within an organization assume this. Names only have as much power as we give them. While responsibilities are changing and we need to add and update our skills, this isn't a reflection of individuals or the roles that they have now.

There is a wide range of potential titles. Don't limit yourself by the role title itself, and don't limit your search to just "sysadmin" or even "sre" and "devops". From "IT Operations" to "Cloud Engineer" the variety of potential roles are diverse.

Before you even examine jobs, think about the skills you have. As a primer, think about what technical stacks are you familiar with? How familiar are you with the various technologies described in this book? Think about where you want to grow. Write all of this down.

As you review job reqs, as you note skills that you don't have that you'd like to have write those down. Compare your skill evaluation with the job requirements and work towards improving those areas. Even if you don't have experience in these areas, during interviews if you are able to clearly talk about where you are compared to where you want to be for those skills it goes a long way to showing your pursuit of continuous learning (which is a desirable skill).

PREPARING QUESTIONS PRIOR TO THE INTERVIEW

Logan McDonald, a Site Reliability Engineer at BuzzFeed, shares some questions to ask during an interview in this blog post [Questions I ask in SRE interviews](#). While she specifically targets the SRE interview, these are helpful questions for any kind of operations position to help qualify the direction and responsibility for the position.

Today, sysadmins can be devops engineers or site reliability engineers or neither. Many SRE skills overlap with sysadmin skills. It can be frustrating with years of experience as a sysadmin to see a lack of opportunities with the role *sysadmin*. If examined, often the roles advertised as SRE or devops engineer have very similar skills and expectations of individuals. Identify your strengths, and compare them with jobs requirements from positions that sound interesting. Map out your path and work on those skills.

-
- 1 DORA's annual State of DevOps report: <https://devops-research.com/research.html>
 - 2 <http://bit.ly/debois-devopsdays>
 - 3 DevOpsDays Events: <https://www.devopsdays.org/events/>
 - 4 DORA Accelerate DevOps Report: <https://devops-research.com/research.html>
 - 5 Effective DevOps, Davis, and Daniels
 - 6 The Many Shapes of Site Reliability Engineering: <https://medium.com/slalom-engineering/the-many-shapes-of-site-reliability-engineering-468359866517>
 - 7 Site Reliability Engineering book: <https://landing.google.com/sre/books/>
 - 8 Wikipedia: https://en.wikipedia.org/wiki/List_of_engineering_branches
 - 9 Stephen Thorne Site Reliability Engineer at Google, "Tenets of SRE": <https://medium.com/@jerub/tenets-of-sre-8af6238ae8a8>
 - 10 2015 DevOps Salary Report from Puppet: <http://bit.ly/2015-devops-salary>

Chapter 2. Version Control

A NOTE FOR EARLY RELEASE READERS

With Early Release ebooks, you get books in their earliest form—the author’s raw and unedited content as they write—so you can take advantage of these technologies long before the official release of these titles.

This will be the 5th chapter of the final book. Please note that the GitHub repo will be made active later on.

If you have comments about how we might improve the content and/or examples in this book, or if you notice missing material within this chapter, please reach out to the editor at vwilson@oreilly.com.

Imagine you share a bank account with a business partner, and you both have different ways of thinking about and managing money. The only shared visibility is the state of the bank account. It would be time consuming and frustrating to try to understand when and what was paid for with the added cost of overdraft fees or late fees from unpaid bills. Instead, if you shared a system where you had visibility into planned changes and accountability when changes took effect, you would reduce the friction of your partnership and focus more on achieving the outcomes of your business. It improves your collaboration, and allows you to do the work you actually want to do. Because your real work isn’t managing your money.

OK, so now replace “shared bank account with a business partner” with the system you need to manage. Your business partner is the rest of the team you have to work with (including the future you at 2am who has to deal with the state of the system). Every individual on a team may have different ways that they think of managing the state of those different files and unless you all come to a common way of working with the system, you will experience the pain and frustration of trying to fix the system any time there is conflict. Instead, if you adopt version control as a practice, leverage the tools present in your organization already (e.g., Git, Artifactory, GitHub, GitLab), you get visibility,

accountability, and alleviation of the pain of conflicts.

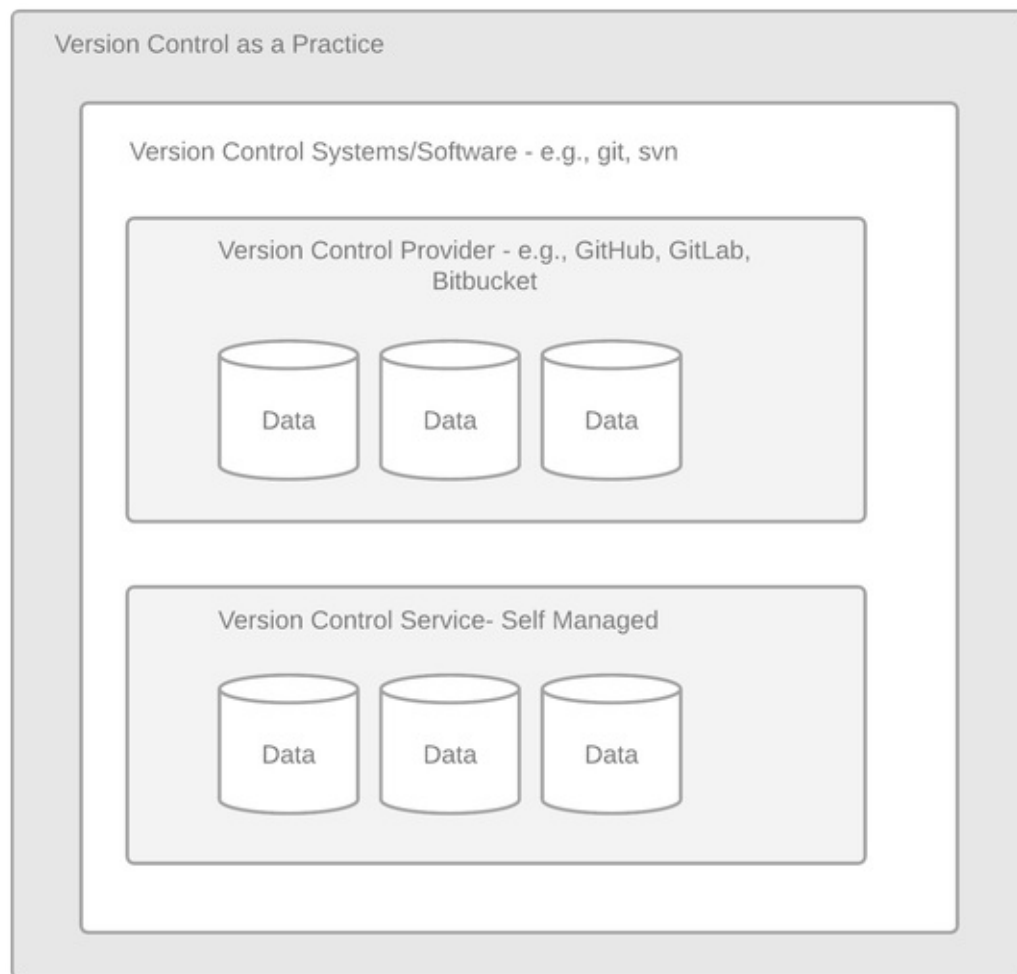
The system you are deploying is not just the source code or binary packages. It includes the set of configurations, scripts you use to deploy, and the processes that you have to get everything in place and monitor from end to end. The process of backing up and minimizing risk to your system as a whole is version control.

Because we (mistakenly) use the terms *source control* and *version control* interchangeably, we think of version control, the *practice*, as the domain of (only) software developers to maintain source code. But the practice of version control—managing and tracking change to configuration files, scripts, and build images—is crucial to system administrators to be able to create multiple environments with the same configurations, replicate and restore systems to original states, and apply published recommended practices to meet compliance standards.

In this chapter, I will explain the benefits of version control, explain the git model for managing change, and give recommendations for how to move your work into version control. I want you to feel enabled to adopt the language of version control and use it.

What is Version Control

Version control is a confusing topic, partly because we use the same words and abbreviations to mean different things. an overused term and means different things depending on the context of use. In the broadest sense (and the one I'm going to focus on in this chapter), version control is the *practice* of managing and tracking changes to data. It can apply to text files like source code or configuration files, and also build artifacts and images. Version control *systems* (VCS) — such as git which is a distributed system, and subversion (svn) which is a centralized system — are specific implementations of version control *software* that handle the features of repositories and branching differently. Hosted version control *providers*, such as GitHub and GitLab, are where the repositories that hold your data and changes are stored. Build artifact repositories are also version control *systems* but manage compiled binaries instead of text files.



Additionally, because people often use the terms source control and version control interchangeably, which think of version control, the practice, as the domain of software developers to maintain source code. But the practice of version control—managing and tracking change to configuration files, scripts, and build images—is crucial to system administrators to be able to create multiple environments with the same configurations, replicate and restore systems to original states, and apply published recommended practices to meet compliance standards.

Benefits of Version Control

Early days into my first official job as a system administrator with the keys to

the kingdom in hand (aka root password), my co-worker walked me through how to update a configuration. The first step was to make a backup copy of the configuration so I could get back to a known good place. Next, was to make the edits to the file. On some systems, this meant using `ed` which doesn't have all the conveniences of modern editors. Next was to restart the service and validate that it was in a good state. As I got used to all the various intricacies around managing the systems, I noticed a lot of old backup files sitting in the service directories with random naming patterns from `.bak` to `.bak.date`, `.date.bak`, `.name.date.bak`. It became really hard to know what actually mattered and whether we could clean up the files. Today, we don't have to configure systems directly and we can adopt version control systems without having to create our own backup naming scheme.

With a version control system, you get management and accountability for change with:

- A copy of each version
- Access controls for creation, deletion, and modifications
- History of changes including who is responsible for a change
- A process to prevent or handle conflict
- The ability to document changes

As a sole administrator for a given system, you can leverage version control to track the state of the system over time and document why certain settings were employed or changes were made. Practicing version control is the foundation for collaborating with others on your team and within the organization to help others understand how your systems are being managed by using a common language.

Leveraging version control with your infrastructure code, configuration files and system tools in addition to source provides:

- **Reproducibility** - You can deploy a specific version of a system or an environment with the scripts, configuration, and software artifacts.
- **Enculturation** - Onboarding new team members is facilitated by version control changelogs to show how you do things. Enculturation can increase

productivity and efficiency.

- **Visibility in Change Management** - You can provide access to other teams to your repositories to show intent and give them the ability to approve change even if they are not part of making the change and show what work is in progress or complete.
- **Accountability** - Use a version control system to track changes and the history of changes to systems provides an audit trail, making it possible to answer questions about who created each system and what purpose it was intended for. Accountability can decrease costs as systems can be audited to ensure they are still needed.

Organizing Infra Projects

There is no one right way to do project organization when it comes to choosing between one project per repo (multi-repo) or all projects within a single repo (mono-repo). Each method includes a set of trade-offs.

One trade-off is code organization. With multi-repos, you agree to one project per repo, but there is no holistic definition of what a project entails. Some projects line up well to the project definition but for other work that might not be so clear-cut. For example, think about this scenario: where would a single helper script for configuring a laptop reside?

It could be in its own repository, grouped with other random helper scripts, or grouped with all workstation related scripts.

How do individuals find this helper script or identify whether it exists already? In a mono-repo, there is a limited set of locations that the code can be found because everything is in one repo. With multi-repos, someone would have to know which repos to search.

A second trade-off is dependency management. With a mono-repo, you can lock your dependencies down to specific versions, which can be helpful when your projects need to have the same version of software. Yet, locking dependencies for software to a single version can be problematic if your projects require different versions of software.

A third trade-off is control, especially when separate functional teams need to collaborate on different projects and want to have different ways of working on the mono-repo. Work preferences can cause personal conflict between the different groups causing problems in code reviews, and merging code. Questions can also arise as to who “owns” the contents of a repository that multiple teams contribute to, and is therefore responsible for making changes, and verifying that these changes do not break things for other teams. Organizing material into smaller repos can minimize this problem, but steepen the learning curve for understanding which repos to work in for a given change requirement.

This is not a comprehensive list of trade-offs. Your team will have to decide whether a mono-repo or multi-repo is more beneficial and should include specific reasons why one method is preferred over the other. These decisions will vary for each team and each project, so you should document your chosen approach clearly enough that when it comes time to work with other teams, or bring on new members to your team, the people you begin working with will understand the approach you’ve agreed on.

Wrapping Up

Chapter 3. Local Development Environments

A NOTE FOR EARLY RELEASE READERS

With Early Release ebooks, you get books in their earliest form—the author’s raw and unedited content as they write—so you can take advantage of these technologies long before the official release of these titles.

This will be the 6th chapter of the final book. Please note that the GitHub repo will be made active later on.

If you have comments about how we might improve the content and/or examples in this book, or if you notice missing material within this chapter, please reach out to the editor at vwilson@oreilly.com.

You might wonder why talking about a development environment belongs in a book about contemporary system administration. While you may not do a lot of coding in your day to day environment, there are a fair amount of standard tools that are essential for productive system administration practices. In contemporary system administration, your working environment is the first system to manage; you can automate and manage the installation and configuration of your system to have a consistent and repeatable base. Further, this approach can be leveraged to reduce the context individuals on the team need to do for all of their day to day tasks.

That’s what this chapter is about: helping you manage the change in your tools and responsibilities to improve your system administration practices through the adoption of a codified local development environment (LDE). Chosen properly, your LDE can help you automate tasks more efficiently and make it easier to collaborate with your users and colleagues.

What Is a Local Development Environment

As a system administrator, you take on the responsibility for the reliability of production systems. Whatever your specific role is, whatever the system is, you need a safe way to simulate a realistic model of your production environment to tinker with the model and figure out workable processes. Ultimately, you want to identify the resilient and sustainable ways to operate in your production environment.

A local development environment(LDE) provides you the mechanism to minimize the risk to any customer facing system by providing the set of tools and technologies that permit you to develop code isolated from a live environment. To be clear, local doesn't refer to a specific geographical space; the environment can be located on your laptop or workstation, and it can also be a private sandbox on a remote system with a cloud provider.

A local development environment enables you to:

- Work offline.
- Debug code/configuration.
- Onboard new employees or team members with the relevant context necessary to do a specific task.
- Embrace policy compliance and recommended practices through codified standards.

The Components of an LDE

An effective LDE will be specific to the set of tasks and projects that your work requires including some combination of:

- An editor
- Programming languages
- Frameworks
- Libraries
- Applications

And of course, any configurations specific to any of these components. Let's review these components in more detail.

Editor

Sysadmins write code, scripts, infrastructure, documentation, and tests. The right text editor reduces your overhead, helping you catch problems in your code early, suggests code completions based on language semantics, formats your code based on team expectations, and integrates with other tools.

For example, you could craft a Dockerfile, the text file containing the build instruction for a docker container manually, looking up each build instruction. With a contemporary editor, you get suggested snippets corresponding to valid Dockerfile commands to quickly build out a new Dockerfile to ease the creation of a Dockerfile as you compose the file. For an existing Dockerfile, hovering over a command will give you a detailed description of what that command is doing.

What should you look for in an editor? While you're probably already familiar with one or more text editors, there are additional features that can make it worth learning another. As I mainly use Microsoft's open source, cross-platform editor, Visual Studio Code (VS Code), I'll be using it in my examples, but there are other editors with these characteristics. Some of the benefits I look for include:

- Integrated static code analysis
- Code completion
- Indented code to match team conventions
- Distributed pairing
- Integrated workflow with git

NOTE

Be open-minded about others trying and adopting different tools. For example, while vi or emacs may have all the features you want and need, they may not be the right choice for others who don't have that experience. Building and learning that context from scratch and all the unique mechanisms to operate the editor may not be the best use of their time; especially when there is so much to learn to be an effective system administrator.

Integrated Static Code Analysis

You can speed up development and reduce potential issues by adding static code analysis or linter extensions for the languages in use. For example for writing bash scripts, you can install shellcheck and the shellcheck extension. Then, as you write shell code, the editor will alert you of potential problems. In the following example, I wanted to find all the files with a .png extension in the current directory, so I wrote some shell code.

```
#!/bin/bash

for file_name in $(ls *.png)
do
    echo "$file_name"
done
```

VS Code with the shellcheck extension warned me that “Iterating over ls output is fragile.” I updated the code eliminating ls and used globs instead as recommended.

```
for file_name in *.png
do
    echo "$file_name"
done
```

Running a linter as you write code allows you to catch and fix potential problems as you are writing. There are linters for many types of files, from YAML to specific languages. Within the editor, you can customize the options for how the lint runs, allowing you to run linting as you type code or if that is too distracting after you save your updates.

Code completion

Code completion improves the coding experience by providing educated guesses about what you are trying to do. As you type, options will pop up with suggestions for autocompletion. Some languages have better completions automatically; you can add extensions to improve others.

Establish and validate team conventions

Many organizations use code linters to help enforce a consistent coding style, making it easier for teams to maintain a shared code repository. For example, rather than debate whether spaces or tabs are more readable, the team can standardize the text indentation and each individual can customize their editor to display their preferred indentation. Additionally, you can convert the amount of spacing currently in use within a file to conform to new requirements.

Distributed pairing

Multiple people can work on a project together, editing files and running tests with VS Code. Each participant maintains their customized environment with separate cursors while collaborating with the **Live Share** extension. As a result, pairing to build infracode together doesn't require individuals sitting in the same space, and everyone experiences editing with their preferred style.

Integrate workflow with git

As you work on a project, it's helpful to see the changes that you've made and whether you've committed those changes. This can prevent unfortunate surprises, such as forgetting to share your fixes for bugs back to the shared source control repository.

Programming Languages

While you might not develop applications, honing development skills in shell code and at least one additional language helps you to better collaborate and build functionality that improves your team's productivity as a whole.

Automating toil work—from building faster ways to open JIRA tickets with pre-populated meta information to scanning compute instances for systems out of compliance with required standards—frees up the team's time to focus on areas that require human thought and creativity.

Bash and PowerShell are reasonable choices in most environments, available on current versions of both Linux and Windows, and will be extremely handy day-in and day-out. Once a shell script starts getting longer than about 50 lines or needs complex data structures, it becomes harder to understand leading to a fragility in the way that a system is managed. No one wants to break the script.

In those cases, re-implementing a script as a utility in a general-purpose programming language can help.

Languages like Python, C#, Ruby, and Go can provide better error handling, a rich community of libraries, and additional debugging tools and utilities.

So how do you choose a specific language to invest time and energy in, either to study or to write a particular script? Think through these questions when choosing a language:

What languages is already in use within your organization or team? How much code in a specific language do you already have?

It can be beneficial to learn how to read whatever language(s) your development team uses. When the system isn't working as expected, it can be helpful to see whether the problem with the "as expected" is the code or tests of that code.

You can also leverage the fact that multiple people are able to support debugging or feature implementation as needed.

Sometimes the right choice is to go along with popular opinion; sometimes, the right choice is to buck the trend. It's ok to choose languages and technologies based on existing skills on the team.

Are there tools and technologies that you or your team would like to adopt, but they're implemented in a language that nobody on your team is familiar with?

You could adopt an alternative technology that uses a language your team already knows, or take it as an opportunity to broaden your skill set by learning a new language. New software often leverages contemporary languages, and a team that isn't learning is going to stagnate and be limited in their choices of software adoption.

Be sure to include the overall cost of using and supporting the tools into your decision. Some teams do better collaborating within a worldwide community of fellow-users as made possible with open source technologies, while other teams benefit more from access to commercial training and support. Neither of these options is inherently better, but choosing one that goes against your team culture will add to the complexity of successful adoption.

Even if your organization is happy with the languages in use, a signal of software health is evolution and change.

Consider the impact of different kinds of changes:

- New language versions can break backward compatibility with existing codebases.
- New libraries may simplify previously-complex chores, but require refactoring legacy code.
- Security patches require cessation of vulnerable features requiring legacy refactoring with urgency.

Any language popular enough to have an active development community will be in flux. When planning out what your team wants to work on in the future, your options will include alternatives.

Having documented reasons why the team has chosen a language is a useful artifact to have for reference material for future you as well as new future team members.

What languages are widely used by your industry peers?

Widely adopted languages within the industry will have more support resources with documented example code to community forums.

What challenges are you facing with previous decisions about language implementation?

Sometimes even though there is wide use of a particular language within your organization, it has associated challenges that may hinder a new project. Identifying and documenting your thinking process in your decision is part of the process of proposing the adoption of a new language.

It takes time and energy to refactor to new languages once a utility has been written; and the refactor could lead to needing to support two different tools at once. Even if your team is sticking with one primary language, languages themselves evolve, and it can be necessary to refactor legacy code in order

for it to keep working with new versions of the language or libraries your code relies on. For example, organizations trying to migrate from one infrastructure automation tool to another, often end up with both tools in use rather than a clean migration. Multiple tools with overlapping concerns add confusion and complexity to the environment.

In the end, there is no one right language to learn as a system administrator. Balance your experience and comfort with a specific language with the features and the rest of the team's skills.

NOTE

Sometimes, your operating system will include a version of the language. Often this is an outdated version, and you'll need to update to leverage the language's latest features. Changing the system included language, isn't a recommended practice. Instead, install the desired version separately and set execution paths appropriately to prefer the later version. The explicit external installation will help prevent system instability due to modifying software that the system might be using. It also helps eliminate undefined dependencies in environments.

Frameworks and Libraries

Depending on the languages you use, there may be a set of frameworks and/or libraries to use to support your work. Some examples include:

- AWS SDKs for specific languages
- PagerDuty API Client Libraries to manage your PagerDuty configurations
- Chatops automation frameworks in a specific language

These will be highly specific to your environment and needs and potentially problematic if functionality changes across versions of the library. Documenting explicitly what versions of these frameworks and libraries are in use and ensuring that they are codified into environments that people use, helps prevent lost time trying to debug why different results are occurring.

Other Helpful Utilities

Beyond your editor, languages, frameworks and libraries, there are applications

that can round out your LDE. Depending on your work, different tools will be useful. Areas to investigate for context within your environment for additional specific management tools include:

- Ticket or bug tracking
- Infrastructure and application monitoring
- Alerting
- Config management, container orchestration, and infrastructure provisioning
- Pipelines
- Artifact repositories
- Builds
- Source code
- Chat

You can codify all of these tools with infrastructure code into pre-built containers or virtual machines or leverage a remote system provided by a cloud provider.

Over time, you'll also build out sets of tools that you rely on regardless of your environment. I want to share some of my favorites. Many of the following recommended tools are cross-platform although some are UNIX-specific:

The Silver Searcher

The **Silver Searcher**, or *Ag* for short, levels up searching through code repositories. *Ag* is fast and ignores file patterns from `.gitignore`. It can be integrated with editors as well. When debugging errors or other “needles in the haystack” of code, it can be super helpful to search for a specific string to understand how it's called.

bash-completion

Modern shells provide command completion. This allows you to start typing the beginning of a command, hit TAB and see potential completions. *bash-*

completion extends this feature and allows you to add additional completion features. For example, this could be used to prepopulate resources you need access to. Extensions are shareable across the team.

cURL

Curl is a command-line tool and library to transfer data. You can use it to verify whether you can connect to a URL, which is one of the first validations when checking a web service, for example. You can also use it to send or retrieve data from a URL or retrieve HTTP headers to see specific server response codes.

Docker

Docker provides a mechanism to create isolated environments called containers. A Dockerfile encapsulates the OS, environment files, and application requirements. You can add a Dockerfile to a project and commit it to version control.

With Docker installed and access to a Dockerfile, then onboarding a new collaborator to a project can be as straightforward as running `docker run` to get a working test environment up. This test environment would even match more closely to a production environment if running production on containers.

gh

Using Git as version control and GitHub as the project repository, `gh` extends Git functionality that helps with GitHub tasks from the command-line.

For example, if I want to test out a PR submitted to a project, I can use **`gh pr checkout <issue-number>`** to checkout that specific pull request and do local testing in my environment prior to approving the PR to be merged.

git

As mentioned in the earlier version control chapter, Git is a distributed version control system.

HTTPIe

HTTPie is a command-line HTTP client to test, debug, and interact with APIs with JSON support and syntax highlighting.

jq

jq is a lightweight and flexible command-line JSON processor. Combined with cURL, you can process JSON output from the command line.

mkcert

mkcert makes locally trusted development SSL certificates.

shellcheck

ShellCheck is a utility that shows problems in bash and sh shell scripts. It can identify common mistakes and misused commands. You can ignore specific checks if they are not checks your team wants running against your code with a configuration file.

tmux

tmux is a terminal multiplexer allowing you to switch between several programs in one terminal.

tree

Tree is a utility that lists contents of a directory in a tree-like format. It can be helpful to visualize the structure of a file system, especially for documentation to show others what to expect to see. Sometimes, showing explicitly that expectation rather than just saying “in the current directory” can help uncover missed assumptions.

Wrapping Up

Wise admins are cautious about tinkering with production servers. The risk of introducing problems to your customer impacting service is high, and you have to remember to keep track of any changes, or run the risk of creating “snowflakes” that become difficult for teams to maintain. You need an environment that allows you to work on a realistic model of your production

systems, so that you can try out changes and not destabilize anything including your own laptop.

Even if you don't think of yourself as a programmer, a good local development environment provides the right text editor, programming languages, frameworks, libraries, and other applications so that you can experiment with and learn about your production system, and evaluate changes before pushing them live.

Consider that the first system you can manage is your own workstation. While a supportive local development environment is not required for most system administration, it can improve your effectiveness, automate your setup, and provide you with a repeatable platform for overseeing the production systems you need to oversee. Adopt what works for you, share with those working with you, and improve the foundations required for collaborative work.

FURTHER READING

- Thomas A. Limoncelli, “[Low-Context DevOps](#),” [login:](#), April 28, 2021

Chapter 4. Testing

A NOTE FOR EARLY RELEASE READERS

With Early Release ebooks, you get books in their earliest form—the author’s raw and unedited content as they write—so you can take advantage of these technologies long before the official release of these titles.

This will be the 7th chapter of the final book. Please note that the GitHub repo will be made active later on.

If you have comments about how we might improve the content and/or examples in this book, or if you notice missing material within this chapter, please reach out to the editor at vwilson@oreilly.com.

In my many conversations with other sysadmins, the sense I get is that sysadmins don’t think of themselves as testers. Whether we think of ourselves as testers or not, the reality is that we are using tests to give ourselves information on the state of tasks and explore our environments to understand them better. We want to prevent those terrible 2 am pages or at least gain the knowledge to remedy problems quickly. In this chapter, you’ll learn what tests to write to leverage automated testing and evaluate the effectiveness of tests and change them to suit your needs. Later in Chapter 11, Infrastructure Management, you’ll need these foundational concepts to apply testing to infracode and the practice of Infrastructure as Code.

You’re already testing

Have you ever run through installing a set of software on a non-production or non-live system, watching to see how the system responded and whether any gotchas could be user impacting? This manual testing approach is known as *exploratory testing*. The goal with exploratory testing is to help discover the unknown by experimenting with the system by looking at areas that may need more subjective analysis as to whether they are in a good state. In **Exploratory**

Testing Explained, James Bach defined exploratory testing as “simultaneous learning, test design, and test execution.” In contrast to scripted tests, exploratory testing is centered on your knowledge and perspectives and can be influenced by your personal biases. There is value in learning

You can level up your manual exploration and add some objectivity by adopting more rigor in your analysis — defining testing objectives with short feedback loops to inform your next steps. Then, working with software engineers and testers, you can help shape the testing to eliminate some manual testing for their scripts and infrastructure code (infracode). These scripted tests can:

- **Increase your team’s confidence in your code**

Tests help you reduce the fear of repercussions of making change. Rather than expecting everyone to execute flawlessly, you build safety nets that help you make change confidently.

- **Speed up delivery of working tools and infrastructure**

- **Tackle new projects.**

Other people can take on the responsibility of work you’ve completed when automated tests exist. You’ve created a safe space for individuals to learn and make change.

- **Document the expectations and context of the code.** Good tests can describe expected functionality well.

Testing helps you deliver a working product that includes the infrastructure and scripts that are part of the successful running of your system, eliminates single points of knowledge, and increases confidence that problems won’t easily make it to end-users.

NOTE

It’s helpful to have new team members explore products and processes as part of their onboarding. They can bring unbiased insight to level up quality to correct problems with the product and processes and clear up misunderstandings and inconsistencies that may already exist within the team.

Let's explore other common types of tests that you can script and leverage to build out automated testing: linting, unit, integration, and end-to-end tests.

Common Types of Testing

You can write more effective tools and infracode with testing. Writing tests to automate the testing project can be done in different ways. Let's examine the different types of testing — linting, unit, integration, and end-to-end.

Understanding the different types of testing, including their benefits and drawbacks, helps you create appropriately leveled, maintainable tests. As is no exact definition of these test types, there may be different interpretations of these tests depending on the team. For example, some Google teams **frame tests by size instead of type**.

Linting

Linters are a basic form of static analysis to discover patterns or style conventions problems. With linting, you can identify issues with code early, and you don't have to write specific tests. In addition, it can uncover logic errors that could lead to security vulnerabilities. Linting differs from formatting changes to code because it analyzes how code runs, not just its appearance.

TIP

You may get several warnings if you run a linting tool on an existing project. Changing working code with a large number of stylistic changes may frustrate your team, especially if there are reasons for those particular styles or conventions. Instead of immediately making changes, bring up the results and document the conventions explicitly by configuring the linter.

There are three key reasons to adopt linting in your development workflow: discover bugs, increase readability and decrease variability of your code.

- **Discover bugs** The best time to discover a bug is immediately after you create it, when your code is fresh in your mind and you have the clarity about what you intended to write. Adopting the process of linting while you code allows you and fix it with that known context. While you could run

the linter manually, many editors have linting plugins so you can receive near-instantaneous feedback about potentially problematic code. This allows you to fix issues as they arise instead of putting your code into a commit and submitting a pull request for review.

- **Increase readability** Consistent, readable code is easier to maintain, fix, and extend functionality. When you need to work on an existing codebase, chances are high that the original author will no longer remember the context, but if the code has been written clearly, then it is easier to get up to speed. Linters help enforce readability.
- **Decrease variability** Consistent standards and practices ensure the cohesiveness of code. Encoded style prevents arguments over team conventions so you can focus on discussing the changes that have impact for example specific architectural design or security fixes.

To implement the team standards and ignore or modify rules, you configure your linter with a configuration file. For example, the default line length configuration for Rubocop, the ruby language linter, is **80 characters**. Contemporary displays are much larger than traditional 72-character TTY displays, and your team may want to enable more characters per line. Creating or updating the Rubocop configuration file, `rubocop.yml`, within the project source code repository ensures that everyone checking out the project will run linting without getting a warning for the line length.

Example 4-1. Rubocop configuration to update characters per line test

```
Metrics/LineLength:  
  Max: 100
```

While individuals may have preferences about using 2 or 4 spaces and tabs instead of spaces within their code, your team can validate their code and configurations against the team standards within their editor. This way, code reviews can focus on implementation details rather than stylistic concerns.

A few examples of linters include:

- **ShellCheck** for shell scripts
- **jsonlint** for JSON

- **yamllint** for YAML
- **Black** for Python
- **Prettier** for CSS, HTML, JavaScript, Markdown, and other languages

Unit Tests

Unit tests are small, quick tests that verify whether a piece of code works as expected. They do not run against an actual instance of code that is running. Any resources that depend on calls to external resources are stubbed, i.e., database requests or calls to specific services. Stubbing makes unit tests super helpful for quick evaluation of code correctness because they are fast (generally taking less than a second to run). With unit tests, you aren't checking code on actual system instances, so you don't receive insight into issues due to connectivity or dependency issues between components.

Unit tests are generally the foundation of a testing strategy for a project as they're fast to run, less vulnerable to being flakey or noisy, and isolate where failures occur. They help answer questions about

- design,
- regressions in behavior,
- assumptions about the intent in code, and
- readiness to add new functionality.

When you write unit tests, ensure that it's testing your code. For example, when you write infracode to configure a file or create a directory, your unit test should be validating that you wrote code to configure a file or create a directory, not that your infracode platform knows how to execute on those tasks. Write tests that describe your desired outcomes that validate your code.

Examples of a unit in infracode might be a managed file, directory, or compute instance. The unit test to verify the example units would describe the file, directory, or compute instance requirements, including specific attributes. The unit test describes the expected behavior.

Integration Tests

Integration tests are tests that verify the behavior of multiple objects working together. The specific behavior of integration tests can vary depending on how your team views “multiple objects.” Integration tests can be as narrow as two “units” working together, or as broad as different, more significant components working together. Integration tests run against an ephemeral environment and don’t test every element of the project; they give insight into the project’s behavior at a broader scope.

Because integration tests don’t test everything, problem determination isn’t precise. You need the context of the environmental conditions to understand what went wrong and what caused the failure. Integration tests run in minutes due to the increased complexity in setting up potential infrastructure dependencies, including other services and software.

An example of what you might test in your *infracode* is that a database successfully installs, configures, and starts up appropriately, allowing for connections.

End-to-End Tests

Finally, *end-to-end tests* (E2E) are tests that verify the flow of behavior of a project functions as expected from start to finish on an ephemeral environment with realistic test data. E2E tests test all the application and services that were defined by the *infracode* and how they work together. As you can imagine, this could take quite a while to complete in order to provision and configure new instances and run through the testing suite. Additionally, end-to-end test failure is not isolated and deterministic to a single component. End-to-end tests check specific function output and require more frequent changes to the test code. For example, a test environment located in an availability zone on Amazon with network issues may have intermittent failures. The more flakey the tests, the less likely individuals will spend effort maintaining those tests, which leads to lower quality in the testing suite.

Even with these challenges, end-to-end tests are a critical piece of a testing strategy. They simulate a real user interacting with the system. Modern software can comprise many interconnected subsystems or services that are being built by

a different team inside or outside of an organization. Organizations rely on these externally built systems rather than expending resources into building them in house(which incidentally has even higher risk). System administrators often manage these boundaries where systems need to interconnect.

NOTE

Being able to identify and read the tests in your services product may help you identify tools or patterns that can help you to eliminate manual processes in your testing of infrastructure for those services.

Explicit Testing Strategy

One of the ways that the industry describes testing strategy is through the metaphor of the Test Pyramid. The “**Test Pyramid**” was coined by Mike Cohn in his book Succeeding with Agile in 2009. This pyramid serves as a visual representation of how to think about and plan a project’s testing strategy.

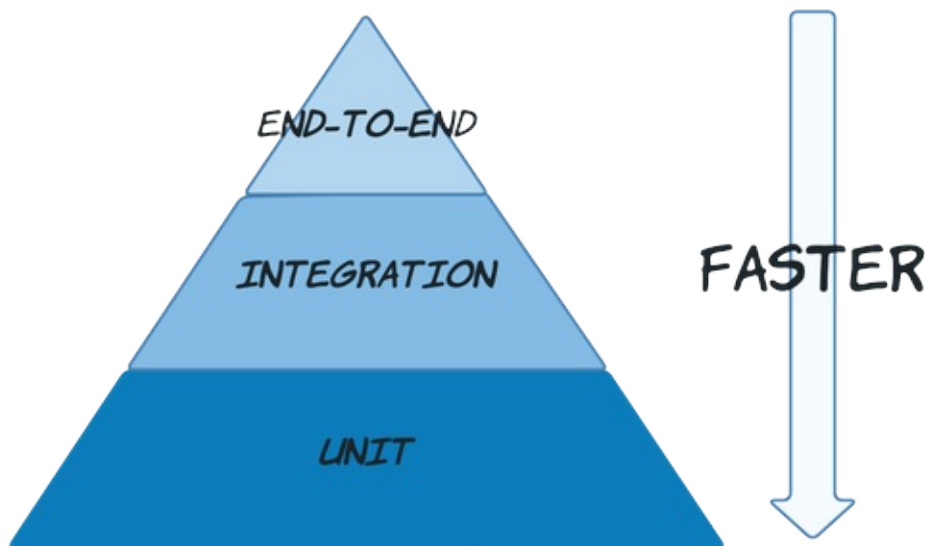


Figure 4-1. Test Pyramid Framework

The pyramid stresses the importance of the different types of tests while recognizing that tests have different implementation times and costs. As you move down layers, tests run faster as there is a decrease in the scope and

complexity of the test.

A good rule of writing tests is to push tests as far down the stack as possible. The lower in the stack it is, the faster that the test will run, and the faster it will provide feedback about the quality and risk of the software. Unit tests are closer to the code testing specific functions, where end-to-end is closer to the end-user experience, hence the pyramid shape based on how much attention and time you spend writing the particular type of tests.

You can examine the tests that exist for a project and qualify the strategy based on the number and type of tests. This informs you of potential gaps where additional testing is needed or tests that need to be eliminated.

Think of your tests as being building blocks. A unit test is 1x1. An integration test will test multiple components and range in multiple sizes for example 1x2 (to test 2 components) or larger. Now, your end-to-end tests will also vary in size, but don't have to get down into the nitty, gritty specifics of each component, especially if that component has already been tested by an earlier (and faster) test.

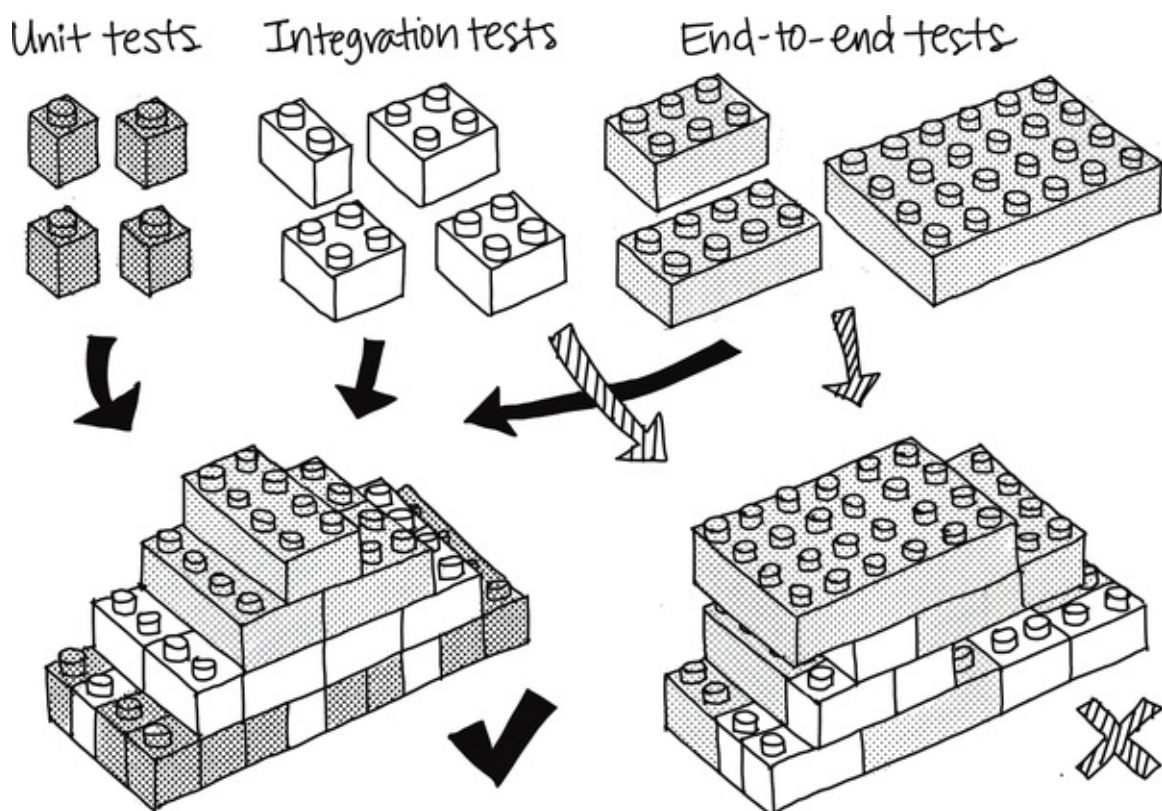


Figure 4-2. Assessing Automated Testing

In this image, you can get the idea of how to visualize the shape of your project's testing strategy based on the number and type of tests. In a healthy strategy, you've got mostly unit tests, with integration tests bridging components, and a few end-to-end tests. In an unhealthy situation, you've got lots of end-to-end tests that are overly specific and will end up taking a longer time to run.

Let's look at some different testing implementations in more detail to understand what could be wrong and steps to take; adding or removing tests. When you are examining a software project's testing implementation this can help you understand how much invisible work is being passed on to the system administration team to support a project.

When you have approximately equal number of tests at every level, this indicates that there are overlaps in testing, in other words, testing the same thing at different levels.

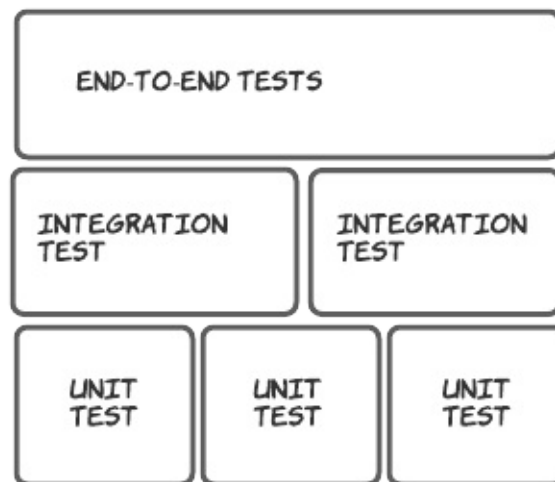


Figure 4-3. Testing Square

This may mean longer test times and delayed delivery into production. Identify the duplication in tests, and reduce those areas of testing within the end-to-end testing cycle.

When you have more end-to-end tests and fewer unit tests, this indicates that there is insufficient coverage at the lower levels of unit and integration tests.

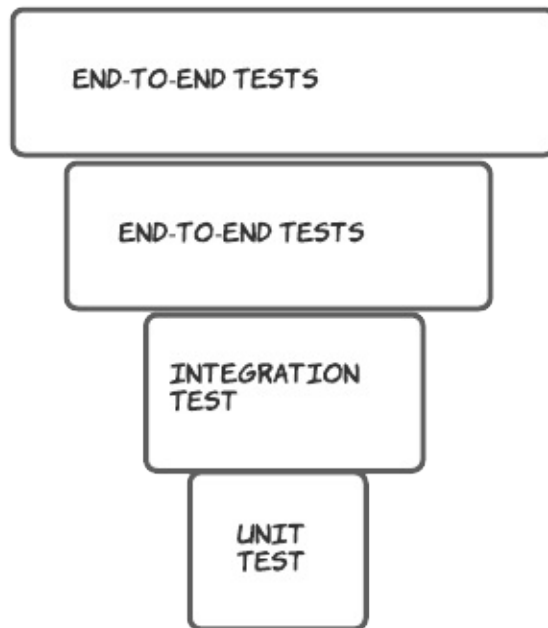


Figure 4-4. Testing Inverted Pyramid

This may mean longer test times, and delayed code integration as it will take longer to verify that code works as expected. Increasing the unit test coverage will increase the confidence of changes in code and reduce the time it takes to merge code leading to fewer conflicts!

When you have full coverage, but there are more end-to-end tests compared to integration tests, that may be an indication that there is insufficient integration coverage. You could also have more end-to-end tests than needed. End-to-end tests are more brittle so they will require more care and maintenance with changes.

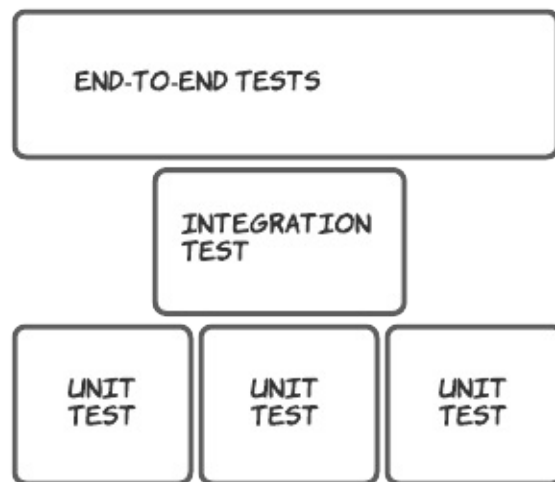


Figure 4-5. Testing Hourglass

These different strategies may also indicate the potential that more time is spent on maintenance of tests rather than developing new features.

That said, infrastructure code testing does not always follow these patterns. Specifically, infrastructure configuration code testing does not benefit from unit tests except when there may be different paths of configuration. For example, a unit test is beneficial when there are differences in platform requirements due to supporting different operating systems. It can also be beneficial where there are differences in environments between development, testing and production or making sure that production API keys don't get deployed in development and testing.

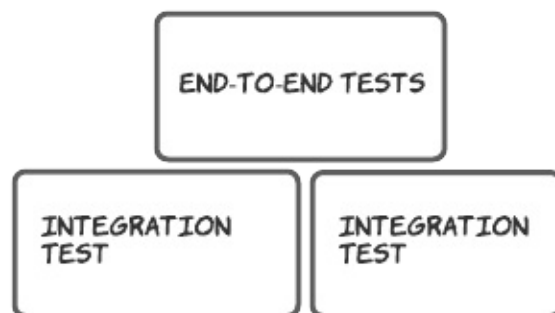


Figure 4-6. Modified Testing Pyramid for Infracode

So, while you want to push tests as far down the stack as possible, due to the nature of infrastructure code, integration testing might be as far down as it makes

sense to push tests.

Improving Your Tests; Learning from Failure

It's easy to write tests; it can be hard to write the right test. Practicing writing tests is the only way to improve your test writing, but how do you improve the quality of the tests that you write? Passing tests tell you that you haven't found a problem - yet. Let's talk about how you leverage the feedback you get from testing.

To really assess and understand how to adopt tests into automation, you need to understand how tests fail. Failing tests tell you more than "found a problem with your code". Examining why tests fail and the different kinds of feedback you are getting allows you to plan a roadmap and automate responses as possible.

You need to think about these as you create and update your test automation. Automation without the ability to act on the feedback you get back from the tests, just adds work; which detracts from the value you could be bringing to your customers and frustrates the team. You can plan how to assess the different outcomes of tests and implement controls around what can be automated and what needs human intervention.

When I think about test failures, there are four main types to plan for:

- **Environmental problems** are the most likely and fastest to resolve. Some examples of environmental problems include file permissions, network connectivity, hardware, or variance between the testing and live environments.
- **Flawed test logic** arises when the test isn't testing the code correctly whether it's due to evolution in specifications or initial miscommunication about the intent of the code.
- **Changing assumptions** are issues in the implementation of the test due to your assumptions about how something works. For example, you change the time when the tests run, and all of a sudden the tests are failing and there has been no change in the code.
- **Code defects** tend to be the least common source of test failures, but the

most complex to identify and resolve. When you believe you've ruled out the other possibilities, then it may be time to fire up a debugger and start looking for problems in the code.

TEST FAILURE ANALYSIS CASE STUDY

Chris Devers

Our test framework was reporting that web services weren't enabled when the deployment tool ran. It was unclear why the error was all of a sudden showing up, as the code for the web service setup hadn't been changed.

On investigation, it turned out that a broader system setup script unconditionally ran the web service setup script followed by a second tool to set up other services. The second tool re-enabled the web service, masking a bug in the setup script for the web service itself. When the system setup script was reorganized and the steps reordered, the web service was shut down by its own configuration script and never re-started.

Think about how this shows different types of test failures. Fixing the broader system setup script introduced an environmental problem that hadn't existed before. The rationale for fixing the dependency was good, but we made a false assumption that the individual steps were idempotent.

Ultimately, we traced the defect to the web service setup script that we might have discovered earlier with unit tests. On the other hand, even without unit tests, the integration test framework detected the problem before a customer noticed it, so this layered testing approach was still a win.

For an established project and code, defects are often blamed for test failures, but really they are at the bottom of the stack when thinking about "why did this test fail". You should look for code defects but after ruling out problems with the environment, test implementation or changes in assumptions so that you don't waste time editing and changing code if a fix might exist somewhere else.

Next Steps

Testing comes in more varieties than what I've covered in this chapter; for example, burn-in, performance, compliance, longevity, security, penetration, and

capacity testing to name just a few. And code can always be improved. Depending on your use case, you may find that you need to adopt a different set of testing into your strategy.

For example, long-lived monolithic databases can be prone to subtle resource allocation errors due to memory leaks. These problems can be hard to detect in shorter tests but in production will lead to service outages. In this case, you'd run a simulated workload on the testing environment for a period of days or even weeks to hopefully uncover problems before they arise in production.

Yet, if you are running in an ephemeral environment where resources are regularly restarted, it may not be necessary to spend the money on long-running environments to identify this kind of problem.

As your testing skills evolve, you'll discover ways to understand the quality of your systems and potential vulnerabilities. There are many resources dedicated to each of these different types of testing; seek out the information resources to address your specific context as you need it.

You don't need to start with a perfect strategy. Over time, you can iterate and improve as you uncover problems.

Wrapping Up

Testing is how you learn about your systems. It's how you learn to work with new things and evaluate that changes will work as expected. When testing systems, the "Test Pyramid" provides you with a model for organizing your test efforts, based on unit tests that show a small block of code functions the way you intended it to work, integration tests to evaluate the interactions between two or more blocks of code, and end-to-end tests that consider the system holistically and validate that your system works the way the customer expects it to work.

Test frameworks that skew toward many high level end-to-end tests are labor-intensive and difficult to identify specific causes for failures. On the other hand, a test suite that emphasizes unit tests lends itself to automation, provides clear rapid feedback, and is easy to extend.

When writing tests to validate your code, consider how to evaluate your tests to better understand the right tests to write. A passing test could mean that a change

to code is good, but it can also mean that the test isn't thorough enough. Failing tests can have many root causes, from external environmental factors that generate errors, to flaws in the tests themselves, to outdated assumptions that cause errors with code that used to work. These factors should be evaluated and eliminated before jumping to the conclusion that the code itself has a defect.

Chapter 5. Security

A NOTE FOR EARLY RELEASE READERS

With Early Release ebooks, you get books in their earliest form—the author’s raw and unedited content as they write—so you can take advantage of these technologies long before the official release of these titles.

This will be the 8th chapter of the final book. Please note that the GitHub repo will be made active later on.

If you have comments about how we might improve the content and/or examples in this book, or if you notice missing material within this chapter, please reach out to the editor at vwilson@oreilly.com.

Early on in my career as a Unix System Administrator, I felt total dread when I saw a large number of failed login attempts coming from external IPs that belonged to an IP range outside of the United States. Seeing the failures made me wonder about any malicious activity that we weren’t detecting. There were 2 people focused on security covering everything from the physical network to network and host intrusions for our Unix systems. Talking through these concerns with the security team, helped me to better understand the risk and motivations of the attackers, learn about the patterns of behavior and resources, and build up the relationship between teams.

In your organization, there may be a specialized team or you may be responsible for the security of your services. Regardless of your role, everyone within the organization has a part to play in security. It’s helpful to understand the concerns, especially where they overlap operations. In this chapter, I focus on sharing general security principles that apply to sysadmins so that you can collaborate. By the end of this chapter, you should be able to define security, explain threat modeling, and have a few methods for communicating security value during architecture planning.

What is Security?

Security is a specialized area of focus within an organization. Depending on the size and scope of an organization there may be a dedicated team, or teams focusing on the broad spectrum of responsibilities from physical systems, architecture planning to compliance adherence.

Security is the practice of protecting hardware, software, networks, and data from harm, theft, or unauthorized access. The ultimate purpose of security is to reduce the risk to people. Security is often viewed as being at odds with desirable features and user convenience, which can exacerbate implementation resistance.

It's impossible to release or manage a perfectly secure application or service when dependencies like underlying libraries, operating systems, and network protocols have security issues. Whether you are building software or deploying open-source or commercial software, plan a layered strategy to minimize vulnerabilities and reduce an attacker's opportunity to exploit them.

Security incidents are not a matter of if but when. They impact companies financially and reduce users' trust. The risks may be to networks, physical or virtual machines, applications, or the data being stored and processed. When the pager goes off, you don't want to discover compromised systems, data corruption, or defaced websites. How do you increase the security of your systems and services? Tackle security like other difficult problems. Break up the large task of "security" into smaller achievable pieces of work that the team iterates on. Allow feedback and learning to inform and modify the team's practice of working in collaboration with software and security engineers.

You can't have perfect security, but you can collaborate with other parts of the organization to establish acceptable levels of security. The amount of security work that every organization needs to do can not be distilled and assigned to one team, especially as the attacks evolve and become more costly and complex.

Collaboration in Security

With cloud services there is a shared responsibility for security. The more operational burden you hand off to the cloud provider, the more levels of

security are taken care of for you. For example, a cloud provider that manages the physical hardware is doing more than just purchasing a server and connecting it to the network for access. They are managing the physical access to that server.

For any service provider your organization uses, ask about and understand their security posture. It isn't helpful when someone leverages a vulnerability of your provider to tell your customers, "It was our provider's fault." You still lose money and the trust of your customers. At minimum find out how the provider handles notifications and the appropriate path of escalation for the discovery of security events.

Whether you use infrastructure, platform, or software-as-a-service from a provider, you are still responsible for some parts of security. Often there is an assumption that security is taken care of for you when you use cloud services, but your organization must configure account and access management, specify and configure endpoints, and manage data. For example, it doesn't matter if your cloud provider encrypts all data on disk if you configure it to have world wide public access.

NOTE

Different roles exist within security. Just because your organization has a "security team," it doesn't mean that they own all security responsibilities. This also doesn't mean you should do that work without recognition, especially if you are the one sysadmin managing and maintaining the systems. That's a path to burnout. Instead, surface necessary work so that your team can assess and prioritize as necessary.

Borrow the Attacker Lens

Taking a different perspective of the systems you manage can help you to improve security for your managed systems.

Threat modeling is a process by which you identify, prioritize, and document potential threats to your organization's assets (physical hardware, software, data) to help you build more secure systems. Assets are not always well understood or recognized, especially when you haven't designed or deployed the system or service yet. Sometimes the threat modeling process can help you identify explicit

data that increases risk to your organization without providing sufficient value, and therefore you shouldn't be storing it.

EXAMPLES OF DIFFERENT DATA ASSETS

Some of the many types of data that your company may collect includes personally identifiable information (PII), personal data, payment card information, and credentials.

- The National Institute of Standards and Technology (NIST) defines **personally identifiable information (PII)** as information that can identify an individual or that is linked or linkable to an individual. An example of PII is an individual's social security number.
- The European Commission defines **personal data** as any information that can directly or indirectly identify a living individual. An example of personal data is a home address.

PII is mainly used within the USA, while personal data is associated with the EU data privacy law; the General Data Protection Regulation (GDPR).

- Payment card information is data found on an individual's payment cards which includes credit and debit cards.
- User credentials are how your site verifies that an individual is who they say they are.
- Examining your data can help qualify your liability based on privacy and data retention laws and regulations.

Next, consider the different vectors of attack also known as the attack surfaces. Attack surfaces are all the potential entry points of intrusion for each asset specific to your organization. For example, look at the vulnerabilities of any endpoints, database connections, and network transports.

NOTE

There are a variety of different threat modeling tools available to help surface and examine

problems that might exist in your systems. If there isn't one in use within your environment, it might be a helpful area to understand vulnerabilities and areas for improvement. There is no one right way or tool, rather instigating the necessary discussions is beneficial.

- [NIST Common Vulnerability Scoring System Calculator](#)
- [Microsoft's Threat Modeling Tool](#)
- [Process for Attack Simulation and Threat Analysis \(PASTA\)](#)
- [OWASP Threat Modeling Control Cheat Sheet](#)

Ask yourself these questions:

- **Who are your attackers?** Attackers can be anyone. They may be internal or external from your organization. Based on the statistics coming from thousands of security incidents analyzed in the yearly [Verizon Data Breach Investigations Report \(DBIR\)](#), most attacks are external. There are the occasional internal rogue system administrators, but by and large internal security issues stem from system configuration errors or publishing private data publicly. In the next chapter, I'll cover some tools and technologies that will help reduce the number of errors that result in internal security incidents.
- **What are their motivations and objectives?** Attackers have different motivations and objectives for their activities. Financial gain motivates most attackers. Espionage and nation-state attacks are a growing threat with numerous¹ breaches occurring to gain intelligence and influence politics. Other motivations include for amusement, personal beliefs, ideology about a particular subject, or a grudge against your organization.

Examples of the objectives from obtaining different types of data include:

- With access to PII or personal data, attackers can apply for credit cards or sell information to marketing firms who specialize in spam campaigns.
- With access to payment card information, attackers can spend money fraudulently.
- With access to user credentials, attackers gains access to all the resources and services granted to the individual that can span multiple sites based on

reuse of credentials.

- **What kind of resources do they have to attack?** The attacker's resources include time, money, infrastructure resources and skills. Tools are evolving that reduce the knowledge required for an individual attacker to obtain their target assets (and ultimate financial gain). While you can't necessarily prevent every single attack, you can make them more expensive.
- **What are their opportunities to attack?** Opportunities are the windows of access to a particular asset. When a vulnerability or flaw in software is discovered and released, there is a window of time to exploit that vulnerability on unpatched systems and services. Successful mitigation requires awareness of necessary patching and adequate time and authority to complete the work.

In some cases, there may be assets outside of your responsibility that attackers leverage to get into production systems. Minimize these opportunities by tracking all assets and patching operating systems and software promptly.

TIP

Check out Ian Coldwater's talk from KubeCon + CloudNativeCon 2019 [Hello From the Other Side: Dispatches From a Kubernetes Attacker](#) for more on what you can learn by borrowing the attacker lens.

Check out the yearly [Verizon Data Breach Investigations Report \(DBIR\)](#) which provides in-depth analysis of thousands of security incidents and breaches, and provides insight into evolving security trends.

Design for Security Operability

Layer your strategies to reduce risk to services and applications, thereby limiting the attacker's opportunity and the scope of damage of a potential breach. This approach is known as defense in depth. Layering defenses means that if one defense fails, the blast radius of compromise may be contained.

For example, build defenses at the edges of your networks with firewalls and configure subnets to limit network traffic from approved networks. Locally on systems, lock down elevated privilege accounts. Additionally, recognize that

100% secure software is impossible, and assume zero trust. Zero trust means having no implicit trust in any services, systems, or networks even if you are leveraging cloud-native services.

It's important to participate in the early architecture and design process with an operability mindset, especially around security, to provide early feedback to reduce the overall development time required. Case in point — I had joined a relatively new team that was building a multi-tenanted service for an internal audience. I reviewed the architecture and realized that the code relied on having no MySQL root password. With hundreds of backend MySQL servers planned for this service, large numbers of unsecured services worried me.

Some of the potential attack vectors I thought about included:

- A misconfigured subnet could make these servers directly accessible to the broader internet.
- Malicious attackers that breached systems on the internal network could easily compromise unsecured systems.

Working with the security engineering team, I managed to get the work prioritized to repair this design defect. Identifying the issue before deployment to production felt great. However, there was the avoidable development cost to fix if implemented collaboratively to start.

Often, decision makers don't invite sysadmins into design meetings. It's important to foster and build relationships with the individuals designing and building the software. This allows you to provide early feedback that will reduce friction for change that comes later in the process.

One way to collaboratively uncover security requirements and prioritize work is to use the CIA triad model. This model provides a way to establish a common context and align values for feature work. CIA stands for Confidentiality, Integrity, and Availability.

- **Confidentiality** is the set of rules that limits access to information to only the people who should have it.
- **Integrity** is that assurance that information is true and correct to its original purpose, and that it can only be modified by those who should be able to.

- **Availability** is the reliable access to information and resources for the individuals who need it, when they need it.

In the case of the root password for the MySQL issue I described above, anyone with access would have been able to log in to the database management system and look at and edit any available data stored. A database breach is a confidentiality compromise. The modification of data by a non-authorized agent is an integrity compromise. Sysadmins can flag CIA issues as part of the acceptance criteria. Having intentional conversations about the design and tracking those conversations helps inform the decisions that the development and product teams make. This also adds a way of incorporating operability stories into work and prioritizing them appropriately. For web applications and web services, the [Open Web Application Security Project \(OWASP\)](#) provides a set of requirements and controls for designing, developing, and testing called the [Application Security Verification Standard \(ASVS\)](#).

NOTE

If you are finding it challenging to get executive support for your efforts to design and implement quality continuous integration and deployment mechanisms, reducing the impact of security vulnerabilities is an excellent use case.

Qualifying Issues

No matter how much effort the team takes to examine software and services from the attackers perspective and designing systems to incorporate a security mindset, there will still be security issues. Some issues may be discovered with your company's software, other times the problem will be with software that you are using either directly or indirectly. Vulnerabilities in publicly released software packages are tracked with [Common Vulnerabilities and Exposures \(CVE\) Identifiers](#). When quantifying the cost and potential impact, it's helpful to categorize them. One strategy is labeling an issue as a bug or a flaw.

Implementation bugs are problems in implementation that lead to a system operating in an unintended way. Implementation bugs can sometimes cause serious security vulnerabilities, for example, heartbleed². Heartbleed was a vulnerability in OpenSSL that allowed malicious folks to eavesdrop on presumed

secure communications, steal data directly from services and users, and impersonate those users and services.

Design flaws are issues where the system is operating exactly as intended, and the problem is with the design or specification itself. Design flaws can be super costly to repair, especially if other tools are building on or depending on the implementation as-is. Sometimes flaws are too expensive to change, and they carry specific warnings about use.

While you don't want to have metrics that incentivize behaviors that push for discovering flaws and bugs over other types of sysadmin work, it is crucial to surface the work that is in progress, especially when a compromise or security incident has been prevented.

TIP

Check out these examples of implementation bugs:

- MS17-010/EternalBlue
- CVE-2016-5195/Dirty CoW

Check out these examples of design flaws:

- **Meltdown**
- KRACK (WPA2 key reinstallation)

Wrapping Up

1 <https://blogs.microsoft.com/on-the-issues/2019/07/17/new-cyberthreats-require-new-ways-to-protect-democracy/>

2 Synopsys, Inc. "The Heartbleed Bug." Heartbleed Bug, heartbleed.com/.

Chapter 6. Baking Infrastructure

A NOTE FOR EARLY RELEASE READERS

With Early Release ebooks, you get books in their earliest form—the author’s raw and unedited content as they write—so you can take advantage of these technologies long before the official release of these titles.

This will be the 9th chapter of the final book. Please note that the GitHub repo will be made active later on.

If you have comments about how we might improve the content and/or examples in this book, or if you notice missing material within this chapter, please reach out to the editor at vwilson@oreilly.com.

It’s time to talk about cookies, the delightful small, sweet treat that is generally composed of some ratio of sugar, fat, and flour. You might just buy ready made cookies, bake them from prepackaged cookie dough, or assemble them from scratch from the ingredients you have in your kitchen. When baking your own cookies, you may run into a variety of problems that impact how your cookies turn out, from the texture and size to the color and taste. Challenges may come from the process you followed, the state of your ingredients, or the environment you are baking in including altitude or oven temperature. To resolve your problems, you can look to troubleshooting guides to modify your recipe based on the problem you are trying to solve. Generally you start with getting the basics right: having a quality recipe with known successful results and solid processes including measuring accurately, choosing and combining ingredients.

Building infrastructure is much like baking cookies. You can use services, buy prepackaged resources, or pull together your own from what you have available. All of the problems that can occur with your infrastructure (the process, resource state, or environmental conditions) can be remediated through scripting your infrastructure with infracode, creating the necessary recipes for your infrastructure. In this chapter, I will explain why you want to script your infrastructure and the different models you can adopt to look at your

infrastructure to plan your infrastructure project.

NOTE

This chapter will focus on infracode, which is the literal ruby, YAML or other language used to describe your infrastructure. In the next chapter, I'll talk about the model of Infrastructure as Code and all of the practices applied to your infracode.

Why script your infrastructure?

I've seen organizations where the pace of change in practices was essentially stagnant because there was always so much work to do that needed to be done immediately and there was no time to invest in scripting. Sometimes there is fear that automation will somehow take away the job.

To manage infrastructure automatically, you can write infracode, human- and machine-readable language to describe the hardware, software, and network resources to automate consistent, repeatable, and transparent management of resources.

Regardless of the type of infrastructure management automation tools you adopt into your organization, you can:

- Increase your speed at deploying the same infrastructure
- Reduce infrastructure risk by eliminating errors introduced through manual configuration and deploys
- Increase the visibility across the organization to governance, security, and compliance controls
- Standardize configuration, provisioning, and deployment tools

These outcomes might not map to specific business values so it is sometimes difficult to secure sufficient budget or support for an infracode project. And on some level this makes sense: It takes time to automate what you do manually, and there may be complexities that are not automatable. So in order to win support for your infracode project, you have to think about (and then communicate concisely) what it is you are doing, how you are doing it, and all

the corner cases that you take care of when you're driving the provisioning, configurations, and deploys manually. Instead of trying to do everything, think about the small specific parts that infracode can improve in order to create a vision that motivates your team and inspires stakeholder alignment, especially when there are competing priorities for the team's time. Let's look at a few ways that you could describe a vision with goals that align to business values.

Consistency

You deploy and configure systems in a uniform way which has been tested and documented.

Business Values: Consistency can increase productivity and efficiency of the team.

Scalability

Infracode streamlines the provisioning and deprovisioning process, allowing you to activate and deactivate fleets of systems as required, with minimal effort. This can take the form of easy manual scale-up and scale-down, fully automated cloud-native management, or any combination, allowing the system to dynamically respond to peaks and troughs in demand while also enabling humans the authority to govern the operation of the automation system.

Business Values: Scalability can increase revenue, add product differentiation, reduce always-on infrastructure costs, and increase user satisfaction.

Empowerment

You define layers of responsibility to allow different teams to have autonomy over their resource governance. You define how to share responsibility between infrastructure, security and application teams, enabling self-service within negotiated boundaries and maintaining overall visibility.

Business Values: Empowerment can decrease the friction of deploying new products while keeping spend within acceptable boundaries. This autonomy leads to increased revenue and differentiation in product development.

Accountability

Tracking infracode changes with version control, you have a history of changes to systems and an audit trail so anyone can answer questions about systems created.

Business Values: Accountability can decrease costs as you can deprovision systems that no longer should be in use.

Enculturation

Version control changelogs facilitate onboarding new team members. They can see how you do the work and can copy the same processes.

Business Values: Enculturation can increase productivity and efficiency.

Experimentation

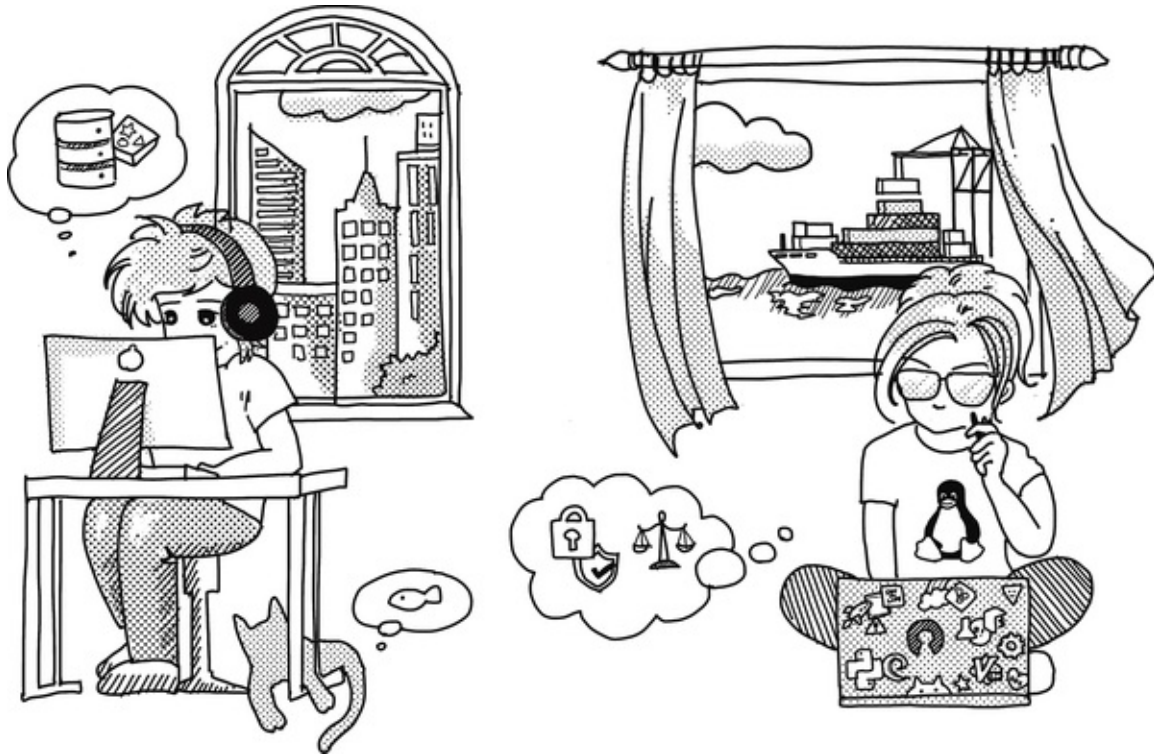
Infracode can allow people to spin up test environments easily, try out new technologies, and quickly push them to production when such experiments are successful.

Business Values: Experimentation can increase revenue and help the team focus on market differentiation.

You know your organization and its leaders best. Based on the company and larger organization objectives, define a project scope and goals that align with those objectives. Once you've got your project scope and goals in place, you can use a specific perspective to model your infrastructure to successfully land your project and goals.

Three Lenses to Model your Infrastructure

Think about the infrastructure that you are managing. You may have physical hardware, or separate compute instances with a variety of dependent services. Each compute entity will have an operating system, and may have several containers or virtual machines. Networking connects different entities, often with access control lists or policies that allow or restrict communication. Now, think about how you describe your infrastructure.



You could view your infrastructure from the furthest level; thinking about the broad set of instances, applications, and services, over their whole lifecycle from provisioning to removal from service. Or, you could view your infrastructure at the most detailed level; thinking about a single compute instance and exactly how it's configured, including its operating system and any software configurations and specific compliance policies.

NOTE

Technological advancement is much like a biological ecosystem, with various habitats, niches, and species. In the case of technology, some new tool comes along and fulfills a need, and the community adopts the practices, if not the technology. This morphs patterns of collaboration and communication, and other technology platforms change to mirror the community's new needs. I hope to show you general patterns here, but books reflect a point in time, and you may find newer tools and technology. Look at the documentation for the specific version of your chosen tool for up-to-date recommended practices.

With this perspective in mind, when choosing a tool, think about what lens fits your immediate need for infrastructure management, encoding your infrastructure to:

1. Build machine images
2. Provision infrastructure resources
3. Configure infrastructure resources

Code to Build Machine Images

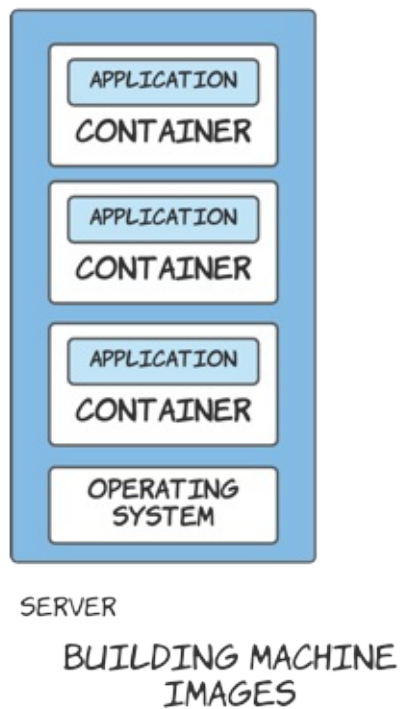
Early in my career, I deployed and maintained many physical systems. At one job, thankfully, I had a hard drive duplicator which allowed me to clone multiple drives at the same time from a single hard drive to speed up the process of deployment. I still had to update the configurations for each of the systems after installing the newly cloned drive, but it saved hours of operating system install and update time. This was a very manual process but faster than building the physical machine, installing the operating system via CDs, and then figuring out how to update the system while it was still potentially vulnerable.

This pattern is known as building from a *golden image*: a perfect, known good mold from which you create more imaged systems. Workflows today conceptually descend from this approach, where a machine image serves much the same purpose as golden images. With machine images, you automate system builds, harden the operating system to reduce vulnerabilities, pre-install any necessary and common tooling, and ultimately provision your compute resources from a more secure and robust base.

A key task for system administrators has been deploying physical computers, but what this entails has evolved. Now, the compute infrastructure you manage could include physical machines, virtual machines, and containers.

NOTE

Because technologies reuse many of the same concepts, infrastructure automation developers tend to reuse terminology, but this can create confusion when you need to be specific about the level of abstraction that you're using. For purposes of this explanation, I'm going to refer to "machines" and "machine images", with the understanding that in practice machine has a spectrum of meanings, from physical systems, virtual machines, to containers.



Examples of tools that build machine images include:

- Packer for multi-platform machine images
- EC2 Image Builder for Amazon Machine Images
- Buildah to build Open Container Initiative (OCI) container images

You may want to write code to build machine images if you need to:

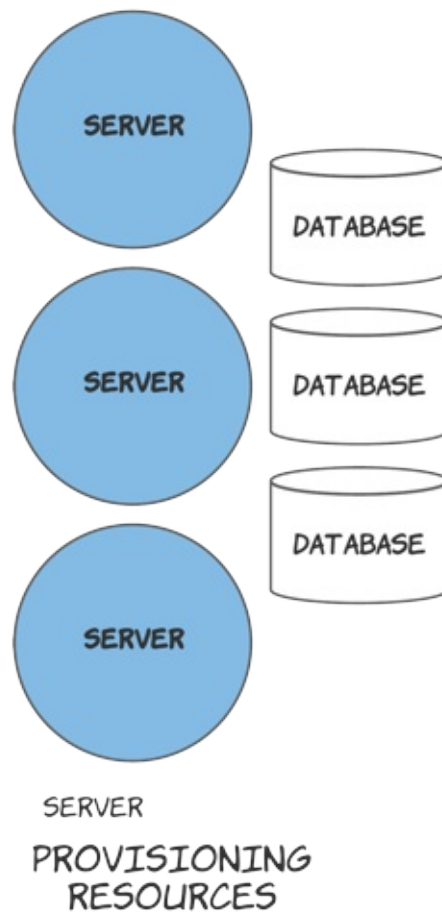
- Ensure systems have a common updated base image
- Install a set of common tools or utilities on all systems
- Use images that are built internally with known provenance of every software package on the system.

Code to Provision Infrastructure

When Cloud Native architectures were introduced, I relished the opportunity to have access to complex infrastructure quickly with simple APIs. No more racking and stacking, tracking cabling and configuring network ports in addition

to installing the application. Instead, installation of provider SDKs and tooling allowed me to quickly provision and configure the necessary infrastructure. Provisioning cloud resources through infracode allows you to

1. Specify the virtual machines, containers, networks, and other API enabled infrastructure needed based on your architecture decisions
2. Connect the individual infrastructure components into stacks
3. Install and configure components
4. Deploy your stack as a unit.



Examples of tools that provision infrastructure resources include:

- HashiCorp Terraform

- Pulumi
- AWS CloudFormation
- Azure Resource Manager
- Google Cloud Deployment Manager

Often writing valid infracode requires a lot of knowledge to successfully provision and configure infrastructure resources. Additionally, while cloud providers often offer fundamentally similar services there are subtle differences in capabilities. Trying to map one-to-one functionality between providers, especially with infracode, can be very frustrating because syntax and abstractions vary widely. If you have a multi-cloud architecture, you most likely will benefit from leveraging frameworks like Pulumi and Terraform which can deploy to multiple platforms.

WARNING

Infracode obfuscates the underlying “how does this work”. Humans work with these systems and must understand more than just deployment automation. When problems occur (and they will), you need to know where to debug.

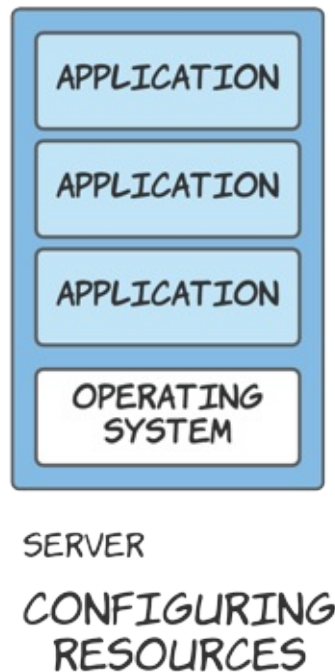
For example, if you use infracode to manage your DNS records for mail and forget SPF and DKIM records, mail delivery to most providers could be disrupted with mail from your domain not being delivered to all intended recipients. Checking for valid syntax doesn’t prevent operability mishaps in the code. Additionally, redeploying the infracode won’t catch the missing configurations.

You may want to write code to provision infrastructure if you have or need:

- Systems that are already partially using provisioning
- Multi-cloud support
- Multi-tier applications
- Repeatable environments for example a testing environment that is a smaller clone of the production environment

Code to Configure Infrastructure

Configuring infrastructure resources through infracode allows you to handle software and service configuration once hardware infrastructure is available.



Examples of tools that configure infrastructure resources include:

- **CFEngine,**
- **Puppet,**
- **Chef Infra,**
- **Salt,** and
- **Red Hat Ansible.**

Each of these options implements configuration management slightly different with different terminology to describe the building blocks that represent the abstractions of configuring infrastructure.

You may want to write code to configure infrastructure if you want to:

- Manage installation and configuration of software installed on systems

- Configure OS parameters
- Repeat installation and configuration of a system
- Automate repair of manual changes made directly to systems

Getting Started

I've shared three lenses which can help you narrow your infracode research. Depending on the infrastructure you want to manage, different tools have supporting features. This may lead you to reconsider your underlying technology choices too!

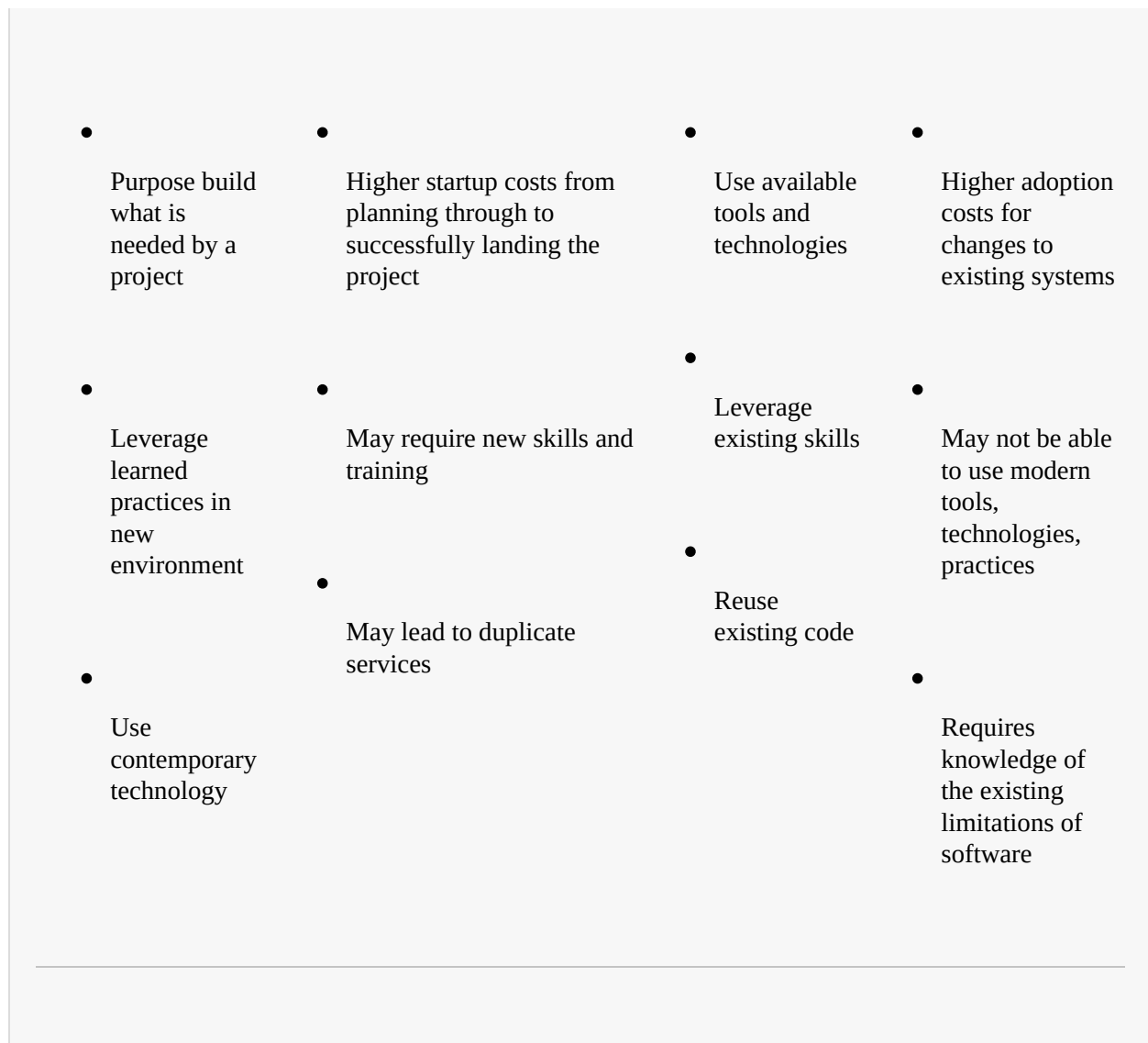
If infracode isn't in use, consider how the tool fits within the context of your environment. Apply the decision framework for choosing a programming language from Chapter 8 Local Development Environment to choosing infrastructure management tools. Selecting and implementing infracode platforms have long-lasting impacts for the team, if not the entire organization. It's difficult to retire technology that's still in use - difficult, but possible. The field is evolving quickly, and some tools may lock you into using a specific vendor's toolset which may not be an acceptable tradeoff. Deploying your chosen tool depends on whether you're adopting it for a new environment (greenfield deployment) or need it to solve struggles in an existing environment (brownfield deployment).

GREENFIELD VERSUS BROWNFIELD PROJECTS

Greenfield projects start from scratch and give you the opportunity to choose all the tools and technology in use. It involves end-to-end design, development, and deployment. There are tradeoffs to implementation that may impact how fast or modern your solution can be, and it might make more sense to adopt a brownfield approach rather than starting from scratch. Carefully assess before starting over.

Table 6-1. Comparing Greenfield to Brownfield

Pros of Greenfield	Cons of Greenfield	Pros of Brownfield	Cons of Brownfield
--------------------	--------------------	--------------------	--------------------



In a greenfield deployment, try to use the selected tooling for all the workflows where it is relevant. This will encourage the adoption of infracode habits and highlight any workflow issues right away. You may solve the issues by changing processes, tools or find that you need to re-scope the project.

In a brown-field deployment, prioritize workflows and gradually apply the new tool. Focus on one area for improvement at a time. For example, you might manage all SSH configuration with Puppet or Chef, and then move on to other parts of webserver configuration for a single server. Avoid areas where a particular aspect of your infrastructure is manually configured to start with as folks will get frustrated and try to bypass the automated processes in favor of being able to follow their normal paths of configuration.

Be wary of taking on too complex of a project at the beginning, or trying to force one tool to fix everything. For example, if you have numerous platforms but mostly Linux, focus on the Linux platform first before trying to adapt your infracode to be multi-platform and support Windows. You may find that you need completely different workflows and tools rather than trying to force a single tool for all platforms.

Often your infracode solution is a multi-prong one that accommodates the complexity of your infrastructure. This is ok. It's perfectly reasonable to adopt Packer to build machine images, Terraform for immutable ephemeral containers in the cloud, and Terraform with Chef for longer lived instances; you can devise a cohesive approach that weaves together these tools into a sustainable solution.

Wrapping Up

The purpose of infracode is to enable you to manage your infrastructure collaboratively as a team in a consistent, reliable and repeatable way. Current widely used infracode tools generally focus on one of three main use-cases: building machine images, provisioning infrastructure resources, and configuring existing infrastructure. With these guidelines, you will be able to create an infracode journey that is customized to your organization or team's needs, technology, strengths, and weaknesses.

Chapter 7. Managing Your Infrastructure

A NOTE FOR EARLY RELEASE READERS

With Early Release ebooks, you get books in their earliest form—the author’s raw and unedited content as they write—so you can take advantage of these technologies long before the official release of these titles.

This will be the 10th chapter of the final book. Please note that the GitHub repo will be made active later on.

If you have comments about how we might improve the content and/or examples in this book, or if you notice missing material within this chapter, please reach out to the editor at vwilson@oreilly.com.

Contemporary compute infrastructure platforms range from managed compute to container orchestrators such as Kubernetes. Storage, network and the software to be installed have also evolved. Likewise, there are many tools and practices available for managing these critical resources. Some have been around for years, and some are relatively new. This is true at an industry level as well as within individual organizations.

Once, I discovered 11 different active ways of managing different parts of the configuration and deployment for a single service. I’d successfully completed an upgrade with a shadow, so I attempted the next upgrade solo. Except, there was this one part of the process that wasn’t automated and depended on a developer generated package. It wasn’t available for the cluster, so while I followed the very thorough checklist and executed the various shell scripts, I had only upgraded part of the system (thousands of nodes) which put the entire system in to a precarious state.

It’s not sustainable to manage systems with complex steps that are individually managed by a human. In this chapter, I’ll introduce the models of Infrastructure

as Code (IaC) and Infrastructure as Data (IaD) and provide a recommendation on getting started so you too can navigate thorny infrastructure scenarios and incrementally adopt more contemporary (and sustainable) practices to improve and modernize your infrastructure management.

Infrastructure as Code

Let's start with the more well-known model, Infrastructure as Code (IaC). IaC is the process of taking time-tested recommended practices from software development and applying that to improve quality and visibility in the management of infrastructure. Infrastructure as Code is all of the practices applied to infrastructure code while infrastructure code is literally the ruby, YAML, or whatever you use to describe your infrastructure.

New practices will be adopted as software development evolves. Current practices include storing infrastructure code (infracode) in version control, code reviews, automated testing, and deployment automation:

Infracode

Infracode (Chapter 10) is human- and machine-readable language to describe the hardware, software, and network resources to automate consistent, repeatable, and transparent management of resources.

Version Control

In Chapter 7, I introduced the fundamental practice of version control; you can store infracode in version control for reproducibility, visibility into change management (how have my resources changed over time, who has access to what resources and when), and accountability (who made what change).

Code Review

Storing infracode in version control is great because it gives you insight about the changes made, but how do you introduce change and decide whether you want to incorporate that change into the system in a systematic and repeatable way?

Code review is the process of a peer looking over code (and in some cases prior to merging into the main branch of a version control repository). The goals of a code review include verifying the implemented solution, verifying that the problem was understood and solved, sharing knowledge about the requested change, providing opportunities for mentorship as the code creator or the reviewer, supporting the enforcement of coding standards and sometimes to help to catch bugs earlier.

Ultimately, code review is one of the ways that your code becomes the team's code and is a practice in navigating disagreements. Your team's code review practices will evolve as you learn from each other.

Automated Testing

As discussed in chapter 9, tests help to build confidence and eliminate some of the fear of making a change. The goals of testing infracode are to help you

1. assess risk
2. respond to and recover from problems quickly
3. improve your delivery processes

Executable automated testing is the only way to manage the needs of contemporary work supporting systems with the rapid pace of deployment, dependency vulnerability announcements, and evolution of infrastructure.

Deployment Automation

With the appropriate tests as validation gates, you can set up a build or continuous integration/continuous delivery or deployment (CI/CD) pipeline, describing the method of automating integrating code and automated builds. The steps or phases in a pipeline will be distinct subsets of tasks grouped by different stages.

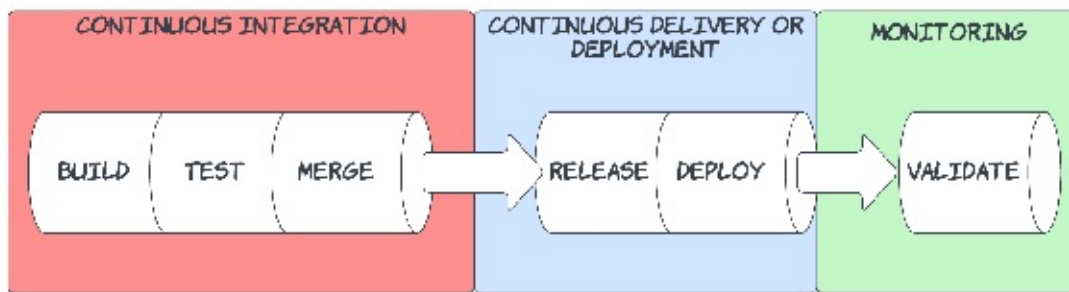


Figure 7-1. Example of Build Pipeline

Build

The project is compiled with the proposed changes.

Test

Scripted tests run against the project.

Merge

Changes are merged into the main branch.

Release

A version of the project is published to an artifact repository.

Deploy

A specific version of the project is deployed to a live environment; either upgrading an existing system or deploying to newly provisioned resources.

Validate

The live environment is validated.

NOTE

Applications built to run on serverless compute have significantly different architecture demands leading to different models in testing and monitoring. With a serverless application, you don't manage server infrastructure, so local testing may be different. You can use whatever continuous integration infrastructure used to promote change in the application through the various phases of the development lifecycle.

Additionally, because sysadmins aren't managing the hardware, promoting the application between environments is faster. Because so much of serverless relies on underlying cloud provided services when those services changes, it can impact how the application runs.

Realistically, your pipeline should model your build processes and this may mean not having these exact phases. You may have different pipelines per project, or you may have specific configurations that direct the flow based on the part of a single project.

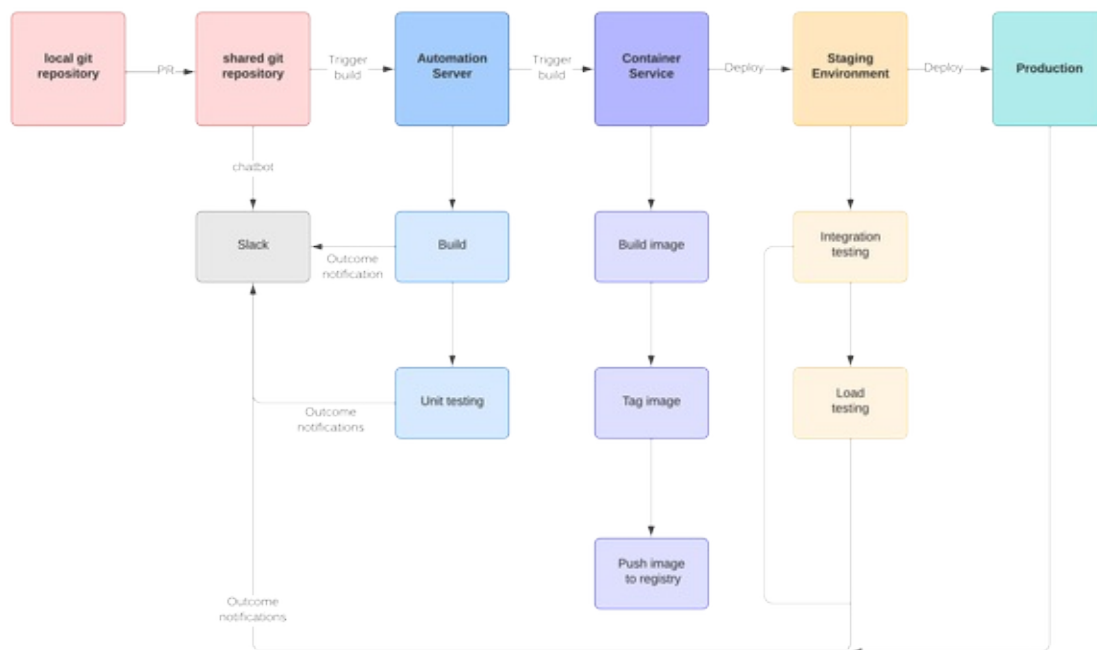


Figure 7-2. Example of CI/CD

In the Figure 2 example, let's follow the path of your pull request after you write, lint, and test your code on your local development environment.

You submit a pull request and the deployment automation software is triggered which sends a notification to the team chat, a Slack channel and triggers a build of the software with your proposed change. At this stage or in a later stage after tests, your team may review the code and approve or reject the PR.

The automation runs unit tests and sends the outcomes to Slack. On a successful code build with passing unit tests, automation kicks off a container build which builds, tags, and pushes the container image to an artifact registry.

A successful container image delivery to the registry triggers a deployment to the staging environment, an ephemeral environment that looks like production is provisioned to start a second round of testing which may include integration, load, and other tests. Finally, after successful testing, the image is deployed into production where further validation testing may occur.

At any point, failure triggers a signal to the team chat.

Examples of tools that support automated testing and deployment automation include:

- GitHub Actions
- CircleCI
- Jenkins
- Azure DevOps
- Google Cloud Build
- AWS CodePipeline

NOTE

Often IaC is conflated with Infrastructure as a Service(IaaS), but these are two different concepts. IaC can be used with on-prem hardware and cloud instances while IaaS is a service offered by a cloud provider and a service delivery model.

DECLARATIVE VERSUS IMPERATIVE INFRACODE

There are two main approaches that infrastructure management tools take with infracode: declarative and imperative. With declarative infracode, you describe the desired end state and the tool handles the implementation of the “how”. In contrast, with imperative infracode, you specify the procedure for achieving a task.

In practice, tools that confine sysadmins to either of these extremes end up being difficult to work with because of your resources. For example, a declarative framework might work for most common deployments while

being too limited to express what has to happen for particular scenarios. An imperative framework might provide better expressiveness for those edge cases, but it is too cumbersome for the common boilerplate scenarios where you just want to deploy a standard image with only a couple of minor tweaks through custom variables. The infracode tools that find widespread adoption tend to balance the declarative-imperative axis, providing straightforward and flexible ways to implement many deployment pipelines.

Treating Your Infrastructure as Data

In a 2013 blog post for the O'Reilly Radar site titled [The Rise of Infrastructure as Data](#), Michael DeHaan, creator of Ansible, said that “Infrastructure is best modeled not as code, nor in a GUI, but as text-based, middle-ground, data-driven policy” and coined the term “Infrastructure as Data” to expand on the concept of declarative infracode.

So should you think about infrastructure as data or as code? There is something extremely powerful in modeling infrastructure as data with a data model and code with infracode, much like light can be modeled as a wave or a particle.

Remember back to Chapter 5 and the importance of data and its value to a company. Part of your job is to keep data safe, managed, and available. When you create an infrastructure data model, you are recognizing the strategic value of all the fundamental resources required for your system to function at that point of time. Some of the concerns to consider:

- Where is your data model stored?
- Where is the metadata about your data model kept?
- How do you keep track of the changes in the data model?

WHAT ABOUT GITOPS

GitOps is a newer (2017) infrastructure management model (as compared to IaC and IaD) that arose from managing Kubernetes clusters. [OpenGitOps](#) is the community driven set of standards to describe the recommended

practices and principles around GitOps.

The v1.0.0 principles include

1. Declarative infracode.
2. Versioned and immutable state.
3. Resources pull approved configuration from the central repository.
4. Resources continuously reconcile actual state from the desired state.

Comparing this to the previous 2 models, GitOps is one approach to infracode, version control and deployment automation (also IaC) and immutable state (IaD). In other words, the model overlaps with the prior 2 models in a lot of ways.

Regardless, there is value in GitOps as a model if it helps you to adopt improved infrastructure management practices in your organization.

Getting Started with Infrastructure Management

With IaC and IaD models in mind, now you can think about how to incrementally adopt practices to improve and modernize your infrastructure management. In an organization with little to no current practices, adopting and modernizing your infrastructure is a large technical change that requires process change and skill updates in addition to the practices and technology. For example, if you're not creating and managing your infrastructure with code now, it can feel overwhelming to identify a place to start. In an organization with current practices, it may be difficult to try to understand all of the systems in place let alone how to make improvements.

A key component of success is to make sure that your team and stakeholders have a shared vision of the specific proposed parts of infrastructure management you want to improve. If you don't have alignment, you will be hard pressed to successfully complete your project.

Additionally, make sure that your project isn't "Automate all the things" and instead is a scoped specific project. Automating everything is a multi-quarter,

potentially multi-year project which means an extended time to reach success criteria. Examples of potential well scoped infrastructure management projects include:

- Improve time-to-deploy for development environments
- Help a geographically distributed infrastructure team collaborate asynchronously by turning real-time system configuration tasks into scheduled code changes at optimal times.
- Streamline onboarding, making it easier to accept part-time assistance for specific projects, while also facilitating cooperation among different teams.

Identification of the challenge is key; infracode is not an end unto itself even if that is what your leadership is saying and attempting to measure. Once you have a goal, break it down into smaller measureable objectives or milestones: version control, code reviews, automated testing, and deployment automation for your infracode, leveraging whatever is in place as necessary.

Start with version control.

In a team that has already adopted using version control software, decide where infracode will be located (with the project or in its own dedicated code repository).

If your team isn't using version control yet, revisit Chapter 7, and learn more about how to get started with version control.

Implement code review processes.

In a team that is already doing code review, assess and document the current process.

TIPS FOR CODE REVIEWS

Navigating the perils of reviewing other's work is not an easy task. When you reach that special flow state with another human, you develop language that communicates more than what is said. But, outside of the flow state, words can be taken so many different ways and adding more words won't necessarily solve the problem.

Here are a few tips I've learned over the years:

- Code doesn't need to be perfect. Before you assess someone's code, establish some understanding about the process of the review. Maybe you'll adopt some form of **conventional comments**, and only block merging for things that must be changed noting where something is just a suggestion. In other words, it's ok to merge code that improves the state of the system even if it's not 100% perfect.
- Don't forget to review the comments. Comments in the code should explain why code exists and not what the code is doing (except for complex things like regular expressions).
- Don't forget to praise the good stuff. Often reviews only focus on what's problematic; encourage appreciation for recommended practices.
- Don't use hyperbolic words like "always" and "never" because generally there will be some context that breaks the rules.

Most importantly, be kind. This isn't to say don't address problematic code. It's not especially kind to avoid saying the hard things that people need to learn by quality feedback. Instead, especially when there are problems, take the extra time to deliver the feedback thoughtfully.

Identify or choose your infrastructure management tool(s).

In the previous chapter, I introduced 3 different infracode models to help you identify and assess tools in use and choose additional inframanagement tools.

Implement single points of authority over elements of infrastructure.

Eliminate any areas where multiple tools are updating the same resources. Conflicts in updates will cause pain, frustration and needless paging.

Don't sacrifice collaboration.

Long-term success of an infrastructure management project requires considering the workflows the tool will encourage, and how those workflows will change the dynamics of your team. Once you roll-out an infrastructure

automation tool, that tool becomes the way relevant system changes will be made in the future. This means that everyone on the team needs to understand the tool well enough to use it in their day-to-day work.

If the infrastructure management project feels like it belongs only to a subset of the team, those people will become a bottleneck when the rest of the team comes to them asking to make changes they used to be able to do themselves. This leads to frustration on both sides. Be sure to build adoption by asking the whole team for their input and showing targeted demos that make real day-to-day struggles easier.

Check for single points of failure.

Remember that infrastructure includes the software to be deployed. Make sure applications are not defined with one person understanding the context of when and why the application runs.

For example, Cloud Native applications leveraging Serverless infrastructure should be defined in source control and deployed automatically to the cloud provider. Creating a standard project skeleton can serve as an example for engineers to start from. It can encapsulate specific security patterns to follow and help prevent the ad-hoc creation of resources without oversight.

Build quality in with automated tests.

Think about the infracode practices in use or planned. You can integrate testing into your initial plans or add them afterwards for existing infracode. Many times, automating infracode tests (infratests) at all is a big step.

NOTE

Recall from Chapter 9 that the challenge of writing tests for infracode is that it can be very easy to test the infrastructure platform in use rather than your code. Think about the test and whether it's verifying the code as written, or testing that the infrastructure management software is working. Unless it's an in-house developed system, trust that the software does what it is supposed to do. Even if you are working with an in-house developed configuration system, test that platform in its git project separately from your infracode project.

Let's revisit the key testing types of linting, unit, integration, and end-to-end

testing specifically with infracode.

Linting

Due to the nature of linters and evolution of recommended practices, linter versions can be especially sensitive. If one person has one version of lint software on their system and someone else has a different version they can have competing changes that influence how they write code causing needless conflicts when trying to work on the same project.

As with other tools in the environment, make sure that everyone is using the same version of linter software.

When your linter returns an error, this doesn't automatically mean that the code needs to be changed. It's important to examine the issues and identify whether they are real problems or areas where customizations to the lint configuration file need to be made.

Writing Unit Tests

With infracode unit tests, there generally is a specific package that maps out to testing the platform you are using. For example, Chef has Chefspec, and puppet has rspec-puppet.

Infracode can get complicated when you have specific customizations, for example, different operating systems, compute instances, or the environment that the system exists in test or production.

Valuable unit tests are going to test those inputs that change how the code runs so that you can have deterministic outputs. They help future sysadmins to modify your code and see issues early.

Generally, very simple infracode doesn't require unit tests! It's only when starting to use more complex patterns that it becomes really critical to have unit tests. Remember to assess the value of the tests because there are inherent costs to maintenance. Crufty tests can inhibit folks from collaborating!

Writing Integration Tests

Remember in Chapter 9, that integration tests are defined differently within organizations. Integration tests may be narrow (testing two components) or very broad (testing multiple components). Before you implement a bunch of tests against infracode, make sure your team is aligned in the goal of implementation.

For example, given infracode to configure a Datadog service, would you test an active configuration of Datadog or mock out a connection to the service and assume that it would work in different environments?

NOTE

Think about scripts that contain system commands which have different responses depending on external factors. When you want to do integration testing, you might want to control the output of the system command because you aren't testing the system command; you are testing the script you are writing that includes the system command. Mocking is a crucial technique that you can leverage to ensure reproducibility and focus your test.

Writing End-to-End Tests

E2E are tests that verify that a project's behavior functions as expected from start to finish on a production-like ephemeral environment which means that there is some amount of non-negligible cost associated with running them. There is a danger that if infrastructure isn't cleaned up from the tests, that you may spend money on unnecessary resources.

Identify the necessary skills required to be successful

Even the most experienced among us needs some level of training to bootstrap the successful adoption of technology.

If you haven't used version control before, that's the first skill to obtain, along with accounts on whatever version control system is in use.

Another example, for Terraform, you may need a combination of training in:

- HashiCorp Configuration Language (HCL), and
- Terraform, and
- Terraform implementation within your organization.

By following these guidelines, you will be able to adopt IaC and IaD that is customized to your organization or team's needs, technology, strengths, and weaknesses. Infracode should enable you to manage your infrastructure more easily and collaborate more effectively.

Wrapping Up

Learning culture, adopt practices and processes that will continue to bring meaningful improvements to the work

MORE RESOURCES

Learn more about Infrastructure as Code from the updated [Infrastructure as Code book by Kief Morris](#).

Chapter 8. Security and Infracode

A NOTE FOR EARLY RELEASE READERS

With Early Release ebooks, you get books in their earliest form—the author’s raw and unedited content as they write—so you can take advantage of these technologies long before the official release of these titles.

This will be the 11th chapter of the final book. Please note that the GitHub repo will be made active later on.

If you have comments about how we might improve the content and/or examples in this book, or if you notice missing material within this chapter, please reach out to the editor at vwilson@oreilly.com.

Defense in depth tells you to apply security practices at different layers in order to deter harm to your infrastructure. The security mindset improves the reliability, robustness, and general operability of your applications, tools, and services. Infrastructure as Code practices in modern system administration presents an opportunity to apply the security mindset in a scalable way. In this chapter, let’s examine some common example areas (identity and access, secrets, compute and network infrastructure) to consider when writing your infracode.

Managing Identity and Access

Depending on the length of time you have been administering systems, the operating systems in your environment and whether you have started using hosted services there may be a variety of different ways you’ve managed users and access including:

- Synchronizing `/etc/passwd` and preventing duplicate user ids,
- Managing LDAP, kerberos, or Active Directory,
- Managing identities in an `htpasswd` file,

- Running sql scripts to add users and grant roles to MySQL databases

These types of processes play an important role in contemporary system administration. Additionally, these processes have been augmented with new tools and technologies that facilitate automation, transparency and compliance.

Some of these processes may still remain as valid methods of managing user access. There are even more possible processes and technologies now to facilitate automation, transparency and compliance.

How should you control access to your system?

Identity and access management is how you configure roles and privileges for users, groups, and services and the underlying technology and processes that support the allocation and revocation of privilege.

There are three core elements of identity and access management:

- **Authentication** - A user is who they say they are.
- **Authorization** - A user has the privilege to do the requested action.
- **Activity Logging** - The recording of user actions via logging.

In addition to any in-house solutions you manage, external services implement identity and access management in their own way with different terminology and concepts. This can even apply to specific services offered by a given provider (e.g. compute instance versus database authentication and authorization). You'll need to read the specific documentation associated with the service you plan to use to understand exactly how authentication, authorization, and logging is done.

Examples of service providers and their identity services include:

- Amazon AWS Identity and Access Management (IAM)
- Google GCP Cloud Identity and Identity and Access Management
- Microsoft Azure Active Directory

If you are starting a new position, migrating to a different cloud, or using a new web service, don't assume that identity implementations are the same. Due to differences between services and providers, you can accidentally weaken your

system with misconfigurations.

Examples of how modern infrastructure identity practices change include:

- Instead of a single factor of authentication such as a password to log into a system, you might require multi-factor authentication(MFA) which requires multiple pieces of evidence to verify that the individual is who they say they are; usually with something they know like a password or PIN and something they have like a security token or card.
- Instead of synchronizing and centralizing `/etc/passwd` across many UNIX systems or binding them to an LDAP directory, you might rely on configuration infracode to ensure users have accounts only on the systems they need.

Identity and access management can get really complex. For example, in a hybrid scenario where you manage identities with a corporate user directory external from your service provider, you might have to manage trust relationships and federation between different services. This allows you to share authentication methods across services so that users can use existing credentials.

Complexity is also increased by the need for identity and access management in different domains, such as corporate identities within an organization, service identities to enable communication between applications, and consumer identities to access customer facing services.

Most likely, the set of tools you use for identity and access management for the variety of services you manage is more complex than it used to be. Leveraging infracode allows you to have consistent, repeatable, and testable configurations. You'll also need clear processes, especially around on-boarding and off-boarding of employees, to configure anything that doesn't integrate with automation.

Creating more developer friendly ways to manage provisioning of resources also creates the need for additional guardrails and audits. For example, one of the most common access misconfigurations with object storage services like AWS S3 is to configure full anonymous access to a bucket or allowing anyone to read or write to the bucket. How does this happen? Many how-to guides illustrate the concepts behind services by having developers immediately open up access to make it easy to focus on learning the service and don't explain exactly what

those configurations do. These patterns then get copied into live environments and create vulnerabilities. Providing example infracode snippets that reflect best practices can help make it easier for others and keep settings uniform across your organization.

You may need to audit for issues in your environment and educate other engineers within your organization to use specific technology. For example, you may wish to ensure everyone has MFA enabled for their accounts. You might set up automation that regularly scans for accounts missing MFA and notifies the account holder to remediate by adding MFA or deactivate the account.

You can leverage your infracode tools of choice to track, audit, and modify corporate and service identities to your systems, as part of your provisioning process. This ensures the settings you encode are applied uniformly, and when your needs change, the tooling makes pushing the change out easier.

Who should have access to your system?

Once you figure out how you control access to your variety of systems, then it's a matter of figuring out who should have access to your system.

When reviewing application or service documentation, you can often find guidance about expectations on running the systems including what accounts are needed and any associated permissions.

Other areas to identify include:

- Are elevated privileges required for individual or service accounts?
- Should there be time boundaries around access?
- Do users who have logged in require a different experience from a casual anonymous user?

You can minimize the scope of possible harm to your system by applying the principles of least privilege and segregation of duties when granting access to your systems. This ensures that a user or component only has access and authority to what they need, versus having root or Administrator accounts.

Putting this in another way, if an account is compromised then the harm that can be done to the system is limited to components of the system that the account

has access or authority over with those credentials.

Additionally, I examine what application programming interfaces (APIs) are available. Often, this is seen as the realm of developers but these are often the critical vectors of attack to your systems. Most modern web applications expose APIs to users in some form and many cloud providers have an API gateway service to configure and manage access to data and other backend services.

You can examine what application programming interfaces (APIs) are available. Often, this is seen as the realm of developers, but these are often the critical vectors of attack to your systems. Most modern web applications expose APIs to users in some form; check what your service is exposing and that it's intentional. In hosted services, the provider's API gateway is used to configure and manage access to systems, data and other backend services.

IAM and logging is analogous to the door locks, security cameras, and other physical controls of an on-premises datacenter or server closet and . Infracode is a practical necessity to ensure these "doors" remain appropriately "locked".

Managing Secrets

Engineers want to get work done as quickly as possible with the least amount of barriers, sometimes trusting the privacy of applications that don't have any notion of privacy or accidentally adding them to source control. Often you have incomplete visibility of what risks you have from exposed secrets, as there may be secrets embedded in code and different services require different processes

Secrets are subject to a bootstrapping problem: If I need to get access to a particular resource, how do I do it? If I need a password, how do I get that password? Early in my career, I remember being handed a carefully written sticky note and informed that it was critical to memorize the password and then destroy the note. Resetting root and Administrator passwords when anyone left the team, while also ensuring everyone remaining who needed access had access, was problematic.

In contemporary environments, you also need to keep more than host passwords secret from people who shouldn't have access to them. Secrets include passwords, mTLS certificates, bearer tokens, and API keys. Using infracode to

establish best practices around secret management can help you increase adoption and track your progress. Infracode also introduces new challenges for secret management, as the infracode tools require access to the secrets. Let's examine the tools and concerns that help to manage secrets.

Password Managers and Secret Management Software

Sometimes secrets need to be accessed or used by humans, sometimes by automated processes, and sometimes both. These access patterns dictate what type of interface is best, and so secret management software is usually tailored mainly for one use or the other.

When the primary concern is interactive use by humans, secret management software is usually called a password manager or privileged access management application. Using a password manager, you can generate and store strong, unique passwords. This helps prevent reusing passwords across sites, and enables sharing secrets across the team without resorting to insecure methods like writing them down or sending them over collaboration services or email. Some well-known password managers include:

- [1Password](#)
- [Lastpass](#)
- [KeePass](#)
- [BitWarden](#)
- [pass](#)

Secret management software for use by other applications is a key-value database with authentication and auditing features. Vendors add value to their secret management solution by integrating with different software ecosystems or supporting specific usage patterns.

Examples of secret management integrated with IaC include:

- Chef Infra with encrypted data bags and Chef Vault
- Puppet and the Hiera eyaml extension

- Ansible Vault
- Salt Stack with Pillar

These IaC platforms allow you to store secrets encrypted that is decrypted at run time on the configuration of your compute infrastructure.

Service provider-specific methods for storing secrets include:

- Amazon AWS Secrets Manager
- Google GCP Secret Manager
- Microsoft Azure Key Vault

Stand alone secret management tools that you can leverage within your code and configuration infracode include:

- Keywhiz
- Knox
- Confidant
- Hashicorp Vault

The primary purpose of a secret management platform is to allow you to decouple storage of secrets from the code or configuration that consumes the secrets. Besides the ability to support that decoupling, you should evaluate secret management software for other concerns, including:

- **Centralization** - all secrets are stored in one place reducing the risk of leaking secrets via storing it in the code or forgetting about their existence
- **Revocation** - marking a secret invalid and no longer trusted
- **Rotation** - updating credentials for an identity. This may include versioning of the secret allowing for progressive rollout of a new secret so that you don't create brittle interdependencies between secrets and applications.
- **Isolation** - ability to assign secrets to individuals or roles, so that the least amount of privilege is granted as needed. A single application doesn't need full access to all project secrets.

- **Inventory** - visibility of secrets being stored (separate from access of secret data itself) to eliminate secret sprawl.
- **Storage** - visibility and configuration of how and where secrets are stored and replicated.
- **Auditing** - interactions with secrets are logged and monitored.
- **Encryption** - secrets are encrypted at rest and during transit. Secrets shouldn't be written to disk or transmitted over networks in clear text.
- **Generation** - creation of new secrets.
- **Integration support** - usability with other services and ability to integrate with your own software.
- **Reliability** - secret access needs to be reliable. If the secret store is down, how do specific services and systems work?

Defending Secrets and Monitoring Usage

Monitoring access to and usage of credentials and other secrets is an important layer of your defense-in-depth strategy. Secrets can leak in many ways, so it's important to have mechanisms in place to detect and respond when that happens. Some ways that secrets get leaked include command history, debug logs, and the use of environment variables. Environment variables deserve special attention because they are available to the process and secrets there may be exposed through a process listing with no audit logs to trace exposure.

In 2020, **rogue activity** was detected within the Ubiquiti network and traced back to the misuse of an IT administrator's credentials that had been inside Lastpass. Lack of logging made it impossible to track what had been done by malicious attackers while they had access to the systems. Even if you assume that anyone that has access to your system should have access to all secrets at any time, think about the risk from third party services that ingest logs that may contain the secret in plain text. Consider the journey of a secret that is logged during a problem; for example, it may be ingested by Splunk, included in a PagerDuty alert, and sent through email and text messaging.

You want to know what systems are available (and should be!) as well as be able

to detect the use of credentials in unexpected ways (from different source IPs or at different times). Many applications and services provide account anomaly detection to enable you to see this unexpected behavior. This is a great opportunity to collaborate with your security team, if you have one.

To identify the breadth and depth of compromise, you need a comprehensive and clear data management strategy for audit logs. This includes separation of privileges so that administrative activities on systems are separate from administrative activities on the audit logs.

In traditional environments, you had to worry about managing user access. Now, you need to worry about service access as well. Tools and techniques have evolved, yet secret management is still problematic especially for machine to machine communication. Often you have incomplete visibility of what risks you have from exposed secrets, as there may be secrets embedded in code and different services require different processes. Access logs from secret management software can help with this problem: services that access secrets will have a certain pattern, which can help make anomalous access more visible. Also, you can audit which services or applications don't use the chosen secret management software; this may indicate places where secrets are accessed in a risky way. Infracode can help close those gaps.

Securing Compute Infrastructure

The efforts you must undertake to secure your compute infrastructure will depend on what types of services you use. For example, the cost of using managed services includes the service provider owning the responsibility of securing the infrastructure underlying those services. For virtual machines and containers that you choose to run, the service provider only provides the physical security and operating environment (hypervisor or container host) your workload is running on. Infracode can make it easier to secure the parts of the stack you remain responsible for and ensure its use. Operating systems and applications often default to open configurations prioritizing ease of use over security. For services that require operating system and application management, you can reduce the exposed attack surface by securing the configuration. This is a common compliance requirement under many regulations and standards,

including the Payment Card Industry Data Security Standard (PCI-DSS), ISO 27001, and the United States' Sarbanes-Oxley Act (SOX) and Federal Information Security Management Act (FISMA). Some resources that provide guidelines include:

- The Center for Internet Security (CIS) **implementation guides**.
- **The Security Technical Implementation Guides (STIGs)**

These peer-reviewed standards are available for a wide variety of operating systems, popular applications, and network devices. They are filled with detailed instructions for tightening all sorts of security-related settings, some of which may not be appropriate for your situation. Review standards and implement recommendations that make sense for your industry and environment.

Another key part of managing the security of compute infrastructure is patching the operating system(OS), installed packages, and applications. Patching can be difficult due to application dependencies on specific versions of OS or other packages, unsustainable deployment practices, or fear of compatibility and stability problems. Infracode can help to address all these concerns. If an application or package has specific requirements, the requirements can be reflected in the infracode. The automated, repeatable nature of infracode encourages frequent deployment and can enable testing of patches for critical systems. Automated testing can be implemented to test different versions of dependencies to expose the risk to patching and provide peace of mind to proceed with patching as needed.

You need to update a containerized application the same way you would need to update it if the application was running directly on a server. Most container images include a significant number of OS packages that will require periodic updates. You can use infracode to build new, patched container images, test, and deploy them.

The twelve-factor app recommends to ***explicitly declare and isolate dependencies*** which eliminates the implicit dependence on system-wide packages. By including a manifest with specific versions of applications, you can reproduce builds reliably without impacting the underlying operating system. Additionally, it provides a path to test builds with new versions by updating the manifest rather than relying on available upgrades from your

operating system vendor. If you isolate dependencies, remember that in addition to OS patching, you need to plan to keep your dependency manifest up to date as well, which includes rebuilding, testing, and redeploying your application.

Managing Networking

Network controls provide defense in depth for networked services. If an attacker is unable to communicate with a service, then they can't attack that service directly regardless of vulnerabilities or misconfigurations it may have. This basic insight led to the development of the classic two- or three-ring network layout. Sysadmins would create a trusted core to contain most of an organization's systems and configure firewalls to limit incoming access to that core from outer, less-trusted network zones. In this model, publicly-accessible systems such as web servers would go in the outermost zone, often called the demilitarized zone (DMZ).

This has been described as “candy bar network security”: crunchy on the outside, chewy on the inside. The idea is that attention is focused on the perimeter and assuming that anyone accessing internal resources is doing what they need to and needs a minimal friction experience.

The shortcomings of the classic two- or three-ring trust-based network become apparent when that network isolation starts being used as the primary defense for insecure systems or protocols. An attacker who is able to gain access to one system in the trusted core then enters a playground of insecure systems.

To combat the shortcomings of “candy bar security”, the industry has moved towards a zero trust architecture model. The key principles of zero trust is :

- no implicit trust is granted between entities based on their location
- required authentication and authorization
- protection is oriented around resources rather than network segments

In other words, each authorized and authenticated entity on the network (such as a server or a person's workstation) can communicate only with the services allowed based on established policies.

Infracode is a key enabler for moving toward a zero-trust network. Zero-trust ideas can be built into your network no matter what tech you use. In an on-premise, hardware-based environment, infracode allows the adoption of much finer-grained network segmentation along whatever boundaries make sense for your needs. Software-Defined Networking products take this flexibility a step further, being designed specifically to adapt their configuration quickly and easily as you add and remove servers and services. In the cloud, infracode makes it easier for you to integrate features such as AWS's Security Groups with the rest of your provisioning workflow. When new systems are being added, consider what services they need to communicate with, and restrict network communication to only those services. The initial effort of mapping these network dependencies is rewarded later by an easier to understand architecture with data flows explicitly documented in the infracode.

The dynamic nature of containerized and serverless workloads presents further challenges and opportunities for network segmentation. Most products and services have built-in or add-on features to enable zero-trust-style networking integrated with the workload orchestration. For example, Network Policies in Kubernetes can target specific pods according to the familiar selectors admins and developers use for everything else. If you want to utilize Network Policies in Kubernetes, it's important to make sure your chosen Kubernetes network plugin supports the features required to achieve your network security goals.

TIP

Check out these resources to learn more about zero trust:

- Forrester Research “No More Chewy Centers”
- Google BeyondCorp project
- [NIST SP 800-207](#) provides a formal reference

Recommendations for your Security Infracode

If your organization has little to no IaC practices currently, start with understanding the infracode practices in use or planned. Integrate security into

your initial plans or add them to your overall strategy.

1. Verify who has the access to run automation and infracode. Make sure that this privilege is limited to only what is necessary to perform those tasks and isolated from modification of the logging of those tasks.
2. Generate and store credentials safely.
3. Don't reuse user or service credentials. With identity and access management, it's possible to generate and revoke the credentials to be used as needed.
4. Check that provisioning infracode only grants the necessary privileges required to users and resources (e.g. virtual machines).
5. Check for resource configurations that can strengthen the integrity of the resources you are using.

For example, with this Terraform snippet you enable **uniform bucket-level access** and provide the key used to encrypt objects in a Google Cloud Storage bucket:

6. Add static code analysis to scan your infracode for security misconfigurations or missing best practices.

For example, **checkov** is an open source tool to scan infracode. Running a scan on the Cloud Storage bucket terraform example above returns the following:

```
terraform scan results:
```

```
Passed checks: 2, Failed checks: 0, Skipped checks: 0
```

```
Check: CKV_GCP_5: "Ensure Google storage bucket have encryption enabled"
```

```
    PASSED for resource: google_storage_bucket.static-assets
```

```
    File: /gcp_bucket.tf:1-7
```

```
    Guide: https://docs.bridgecrew.io/docs/bc_gcp_gcs_1
```

Check: CKV_GCP_29: "Ensure that Cloud Storage buckets have uniform bucket-level access enabled"

PASSED for resource: google_storage_bucket.static-assets

File: /gcp_bucket.tf:1-7

Guide: https://docs.bridgecrew.io/docs/bc_gcp_gcs_2

7. Scan your version control repositories for secrets. For example, **gitleaks** is an open source tool to detect hardcoded secrets within git repos. Hosted source control services like **GitHub** have started providing secret scanning services that alert repository admins and organization owners about potential leaks.

Chapter 9. Presenting Information

A NOTE FOR EARLY RELEASE READERS

With Early Release ebooks, you get books in their earliest form—the author’s raw and unedited content as they write—so you can take advantage of these technologies long before the official release of these titles.

This will be the 15th chapter of the final book. Please note that the GitHub repo will be made active later on.

If you have comments about how we might improve the content and/or examples in this book, or if you notice missing material within this chapter, please reach out to the editor at vwilson@oreilly.com.

Stories are the fundamental way that humans organize and make sense of information. Stories provide structure and purpose to data. Effective system administrators recognize the power of a good narrative and use different mediums to share messages effectively. They organize their information and communicate beyond text to tell a story with images, photos, graphs, charts, audio, and even video. So often when mentoring other sysadmins trying to make change occur in their organization, I find myself sharing some of these key concepts about data organization and presentation.

Show five clever people the same data and they’ll come up with ten interpretations of what it means. You can’t assume that others will draw the same conclusions that you do, unless you put in the effort to craft a narrative that will lead and influence people. One well known example of compelling narratives in our industry is The Phoenix Project from Gene Kim and company. They’ve influenced many practitioners sharing the **three ways of devops** through the story of Brent, the sysadmin that everyone views as a bottle neck. In this chapter, I share the skill of distilling information to convey meaning, and drive desired action through influencing people regardless of authority.

Know your audience

In the movies, the protagonist often has the ability to determine the next right step based off of a single query or dashboard that integrates all the necessary data. They can show the output and get support for their endeavors. In the real world, there is no single pane of glass possible that can provide this context and support. Additionally, you are competing for attention and acknowledgement that you have the supporting data for your conclusions.

TIP

Don't bury the lede. When you need help, tell people what specifically you are asking for so that they have the necessary context when listening to your pitch. Sysadmin and author, [Tom Limoncelli](#) offered examples of some of his introductory sentences:

- I'm here to ask for funding [or resources or money].
- I'm here to ask for a policy decision.
- I'm here to ask for advice [how to do something or who to talk to].
- I'm here to give a status update.

Executives have a number of stakeholders that they are responsible to and busy schedules. They also have a limited set of levers to control outcomes: providing resources, clarifying policy, and referring you to different resources.

People need insights into information that is relevant to their responsibilities, which can range from “nuts & bolts” details of the operation of a specific system, to “birds-eye” overviews of the activity in an overall environment. No single graphic or dashboard can aggregate information in a way that is useful for everyone. It is necessary to tailor each graph and dashboard to narrowly focus on the needs of a specific audience.

Presenting people with relevant, accurate, and timely information helps them carry out their duties effectively. If individuals aren't taking expected actions, this may be because they've been provided with information that is stale, vague, or inapplicable to them. If teams are overfocused on short-term speed and execution at the expense of long-term strategy, this could be an indication of a broken feedback loop.

TIP

When the team isn't taking time to reflect on how their work aligns with the organization's goals, make sure that this is not a reflection of the environment and broken feedback loops. It will be really hard to have desired impacts no matter how you modify your message in these cases.

I distinctly remember sitting in yet another meeting as a coworker tried to convey the importance of the work he was doing. He read sentences directly off the slides describing extremely boring maintenance work that talked of saving money that had already been spent. The large numbers from his measurements didn't alleviate the boredom or compel me to want to participate in the additional toil to achieve his project goals.

This experience reminds me of a Mark Twain quote: *Often, the surest way to convey misinformation is to tell the strict truth.* It's not enough to give people the cold facts and trust that they'll then be inspired to act in the way you want them to act; you have to demonstrate why those facts are compelling, how they relate to larger goals, and then create an emotional connection so people want to help your cause.

There are key questions to reflect on when you share your information to help you connect with your audience:

- Who are you communicating to?
- What is important to them?
- What do you want them to know or do?
- What do they already know?
- What is their preferred method of consuming information?
- How does your data make your point?

For example, your CTO may have many reports and need high level information distilled into scorecards. Leadership funds initiatives so you're wasting their time by going into the minutiae of your decisions. However, our peers need to be inspired and may want to explore the underlying data in order to give their

support to get a project done within a timely manner.

Choosing your channel

Once you've reflected on the questions about your audience, think about what you want them to do. Then decide if verbal or written communication is best — this will depend on your objective and type of message.

Verbal communication mostly happens in real time and gives you the opportunity to convey feelings along with facts. It's most useful when there is a component of emotion or sensitivity you want to communicate or if you need immediate feedback.

TIPS FOR SPEAKING

The more you present information through public speaking, the better you will get at it. Beyond practice, there are a few tips that I've learned over the years that may help you level up your speaking.

- Breathe

Especially if you are nervous, you may find yourself breathing faster or holding your breath. This comes across in your speech and affects your pace which can affect how well people can understand what you are saying. You can help yourself by adding cues to your notes to remind yourself to breathe while also leveraging those pauses for emphasis or laughter as appropriate for your contents.

- Vocabulary

Speech needs to sound more like conversation and use clear and natural words especially for technical talks. The environment of the room and the listener's experience and knowledge will all affect how they parse and understand what you are saying. Avoid jargon and acronyms, and make sure the audience understands any technical terms you need to use, taking a moment to define any potentially unfamiliar concepts.

- Pitch

Modulate your voice to create inflections to drive interest in your message. Practice this on different words to see how it changes the message. When you find the right fit, make notations to your presentation.

- Pace

The right pace for your talk varies depending on your audience. You may find yourself in the moment uncovering that some of your assumptions are incorrect. In general, for simple straightforward topics, it's ok to speed up the pace. For more complex topics, you want to slow down. When you have a mixed audience of beginners and experts this is where you can enter the dreaded middle ground of expectations where beginners may feel you went through the material too fast, while experts may feel you went through too slowly. Be thoughtful and consistent in your delivery as to who your audience is and you'll satisfy at least half of your audience.

- Authenticity

Match your expression to your words. Your body language and expressions convey information. Smiling can convey energy and engagement with your topic. If your message and manner don't match, it conveys a dissonance that is generally interpreted as dishonest. For example, when someone says "I'm so excited to share.." in a dull and disinterested voice, do you believe them?

Finally, in-person presentations are very different from virtual ones. When presenting to people, there can be an energy feedback loop that you tap into as you respond to the audience responding to your content. In front of the camera, it can feel draining. You can level up speaking to a camera by creating a virtual audience through setting up a side channel with live supporters who you can speak to rather than just a camera.

Most of the time written communication is asynchronous whether it's through proposals, design documentation, code, or reviews. For some communication like chat and messaging it can be either real time or asynchronous.

Written communication is a better choice when you want to focus on facts or need time to think before responding and have less urgency about getting a response. For more complex messages, it may be more meaningful.

Either of these communication methods can benefit from visualizations to complement the words that you use. The specific visualizations you choose are influenced by the type of information that you are sharing and stories you want to leverage. Regardless, both methods require time and effort to get right. You have to reflect on your purpose and ideas before you can convey your message effectively.

Choose your story type

You can use stories to reflect on the past to explain what happened and to look forward to provide direction now. Each type of story reveals information in a slightly different way, and choosing an effective story to present information drives your reader's reaction toward your desired outcome. Some example story types include:

- **Factoid**

Factoids distill data to interesting data points, highlighting the most common trends, or the noteworthy outliers. An interesting story may drive interest in exploring the rest of the data.

An example of a factoid is the total number of community members using a specific technology, or unique visitors to a website. Factoids are commonly used in dashboards for website stats or product newsletters.

- **Interaction**

Interactions show relationships between different data sets. Positive correlations between data sets move together: when one set moves up or down, the other trends in the same direction. Negatively correlated sets move in contrast to each other, with one moving down when the other moves up. Identifying a positive or negative relationship is useful, but doesn't explain why data sets move together. Be mindful that correlations may be spurious, where the connection is just a coincidence. An effective

story shows the correlation and establishes that the data is meaningfully linked.¹

An example of showing an interaction is having a graph showing MySQL query times and end to end request latency to better observe whether the performance is related to the workload, or if an increase in end-to-end latency is due to a problem in database configuration that has become a bottleneck.

- Change

Change stories are a way to describe how something changes over time. You can use change stories in capacity management and problem detection.

An example of showing change is having a graph that shows the growth of your current used capacity as it approaches the total capacity of your configured system over time. It can show the velocity (change in use from one point of time to another) and acceleration (slope between the lines) to provide how urgent it is to plan or increase capacity.

- Comparison

Comparison stories are a way to show the impact of data that tell different stories. An example of a comparison is showing the different performance characteristics between rolling out a managed relational database from a service provider versus a self managed MySQL instance in a scorecard. It could aggregate important metrics like cost (including the cost of in-house support), performance, scalability, and reliability.

- Personal

Personal stories connect to real-world experience. An example of showing a personal story is an incident summary that contextualizes technical issues with the experiences and choices that individuals made based on their understanding.

Presenting Data in Action

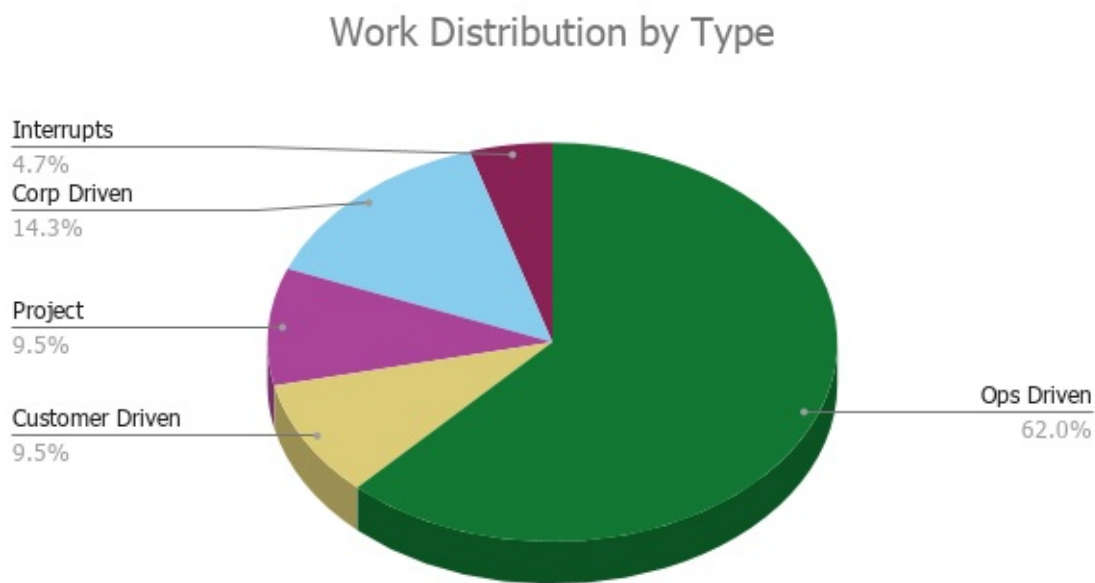
Let me share a couple of scenarios from my own career where presenting data to teams has been useful.

Charts Are Worth A Thousand Words

It was the dreaded quarterly planning time where the team assessed the previous quarter and committed to work in the next. I was new to the team, and I had few expectations. My co-workers expressed frustration because “they never had time to work on team projects to resolve technical debt because of customer interruptions”.

An undisclosed motivation for joining the team was that I had heard that there were challenges with visibility into the work queue and that requests were often delayed or incomplete with no notice. The manager had sought me out explicitly to bring engineering excellence and follow-through execution to the team.

After the planning meeting, I figured out what data to collect around the goals. I worked with the team to categorize the work based on incoming requests and operational debt. I wrote some perl code to query the internal bug API and based on the classification of requests created a few different dashboards to visualize the work. In the next retrospective, I presented a chart like this:

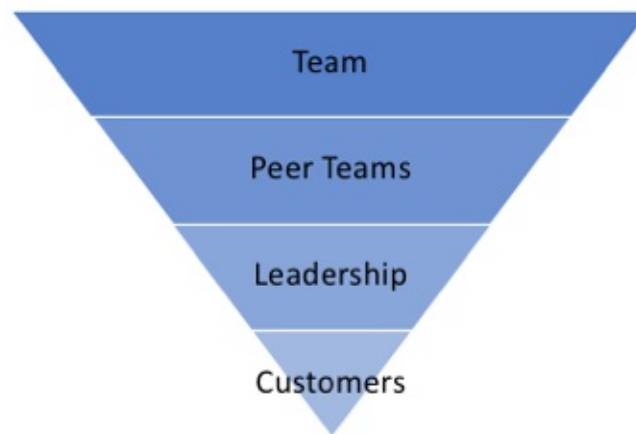


This chart showed that contrary to assumptions, the majority of completed work was driven by the team and not our customers. I could have written up a report, but this simple graphic was easily understood and combined with access to the

underlying data, influenced changes in how we prioritized work as a team and led to further improvements for customers in visibility into the work.

Telling the Same Story With a Different Audience

For projects where you need to focus on data analysis and presentation, think about who you are presenting information to and how to frame the data especially around the language in use. An example may help illustrate better what I mean here, but keep in mind that your team and organization may be different.



Let me explain the layers you're seeing in Figure.

- The team is the largest layer with a lot of shared language and context. They work together closely, sharing tools and processes. They might even have team slang or reuse specific terminology to mean something special in the context of their systems. They benefit from having all the information available when managing their systems, and improvements may help any individual's workflow when handling oncall or touching production systems in general.
- Peer teams may share some common terminology, but it's helpful to establish a shared context and understand their expectations. In some cases talking through monitoring plans may uncover that assumptions about concerns are incorrect.

- Leadership may understand some of the terminology depending on their background, but the further up in responsibility the more terminology may need to be translated to set the appropriate context and risk level especially if they are managing multiple teams and responsibilities.
- Finally, customers may share language, but it's going to be much harder to scale translation across all customers. This requires the most care in communication.

Customize the presentation of the data for each level based on this understanding. It may be too broad a project to try to implement a solution for all of these consumers. Add information about the specific audience to the monitoring plan.

Let me share a personal story from my experience. The announcement came out that a number of colocation facilities were to be closed in a few months to cut costs. This meant that our massively distributed database needed to shrink quickly while minimizing impact of latency and otherwise availability to our customers. I needed to think through what actions we could take as a team to limit how normal day to day actions like upgrades to software and onboarding new customers were impacted.

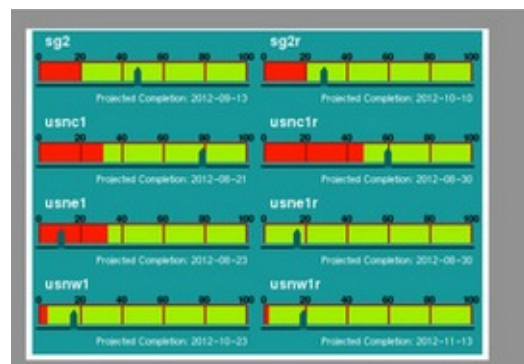
Based on the different timelines for each colocation, I could aggregate where each customer had data and what the best configuration would be to minimize latency impacts in addition to new projects and capacity constraints of the overall system.

I spun up a plan of migrations that balanced out speed, performance, and capacity. I wrote some perl to query the different APIs and javascript to visualize the information.

For the team, I created a table that allowed them to see tasks in progress (P), next prioritized tasks, and work that was complete©. This allowed the ops team to quickly identify whether requests to change a specific table required stopping a task or waiting until the task was complete. Non-impacted table changes could be completed as needed. Additionally, for a region that had work in progress, extra care needed to be taken with upgrades, potentially pausing the migration of data or redirecting customer traffic to the next colo to minimize disruption.

	REGION1	REGION2	REGION3	REGION4	REGION5	REGION6
TASK1	C	C	C	C	C	C
TASK2	-	C	-	P	C	C
TASK3	C	C	C	C	P	C
TASK4	C	-	-	C	C	P

The table of work in progress showed at a glance which regions were complete “C” and which were in progress. For a region that was complete, upgrades, compute and table deployments could be done with minimal coordination with the individuals working on migrations.



My manager didn’t need to know all the specifics. He just needed to know what work was in progress, if we were blocked, and whether we would finish on time.

For him, I created a set of gauges that showed how far we were and our projected completion for each co-location facility. I automatically generated the graphs daily based on the flow of work with updated projected completion dates. The expected work displayed as a red bar within the gauge based on the planned completion date.

Since this was a long-running project, it provided management with the necessary information required to re-prioritize any work and assign additional interrupt work as they could immediately see the impact. They could then communicate progress to all stakeholders for any other projects.

	REGION1	REGION2	REGION3	REGION4	REGION5
TABLE1	READY FOR USE	READY FOR USE	May 1	May 15	READY FOR USE
TABLE2	READY FOR USE	READY FOR USE	May 15	READY FOR USE	
TABLE3	READY FOR USE	READY FOR USE	READY FOR USE	READY FOR USE	
TABLE4	READY FOR USE	READY FOR USE	READY FOR USE	READY FOR USE	

Finally, every customer had their own set of tables. I provided visualizations that let them know exactly where their data was located, which tables would be updated and when they could expect that the tables would be ready in the new

colocation facilities.

The Key Takeaway

Having these different visualizations reduced the number of support and status requests allowing individuals from the team to focus on the work.

Adapt your message based on what your audience needs. Everyone doesn't need all the data collected. Focus your message on the information that matters to the individuals.

Be clear with what data is missing and impacts what individuals can learn from the data that is collected.

Know your visuals

The greatest value of a picture is when it forces us to notice what we never expected to see.

—John W. Tukey

In the previous two scenarios I showed a few ways to visualize data, but there are so many more different visualizations to choose from to transform your data into compelling stories. You can also use design principles to help your audience see what you want them to see.

Visual Cues

Visual cues can help you to display information that others can process without conscious thought. The four basic visual cues are color, form, movement, and spatial position.

1. **Color** You can imply relationships between two different metrics or points in time by varying the hue. You can imply quantity or strength by varying the saturation. You can adjust the temperature, or the perceived warmth or coolness of a color to focus attention. Warmer colors tend to advance into the foreground while cooler colors fade into the background. Be mindful that color should be used to enhance the conveyance of information but that new information shouldn't be expressed solely through the use of different

colors.

2. **Form** You can change length, width, orientation, size, and shape.
3. **Movement** Flicker and motion can call attention to specific areas of importance but can be distracting or annoying. You can also imply motion through the other visual properties rather than using motion directly.
4. **Position** You can use a 2-D position and spatial grouping.

Sometimes cues are not appropriate, if they mislead or hinder your audience's interpretation of your visualizations. For example, don't use different sized circles for categorical data if the magnitude difference of the categories aren't important.

TIP

Learn more about design principles from Robin Williams' The Non-Designer's Design Book.

Chart types

You can use different charts to visualize data. Some examples include:

- **Data Tables**

Data tables organize data into rows and columns. Tables can be a valuable tool to:

- **Plan**, such as itemizing a list of requirements for a proposal, brainstorming quarterly projects and elaborating on details that apply to each identified element such as proposer or length of time.
- **Document**, for example to lay out a list of options or provide comparisons between different tools and services.
- **Define** lists as a quick periodic review for tactical direction. Examples include top pages or sources for websites.
- **Explore** large sets to filter, display data and drill down into individual queries.

Tables can be an overwhelming way to present a large volume of data, so it is a good habit to complement tables with other visualisations that can draw attention to trends, outliers, and other patterns in the raw table data. Take a look at this example:

Table 9-1. Amazon DynamoDB Throughput Limits in Table format^a

	On-Demand	Provisioned
Per table	40K read request units and 40K write request units	40K read request units and 40K write request units
Per account	Not applicable	80K read capacity units and 80K write capacity units
Minimum throughput for any table or global secondary index	Not applicable	one read capacity unit and one write capacity unit

^a “Service, Account, and Table Quotas in Amazon DynamoDB”, Amazon, last modified December 15, 2020, <https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Limits.html#default-limits-throughput-capacity-modes>

Here, the table format is used in documentation to illustrate a comparison between the on-demand and provisioned Amazon DynamoDB throughput limits. The format works because there isn’t a lot of data, and it’s clear what is different.

Timestamp UTC	cache_status	client_ip_hash	content_type	geo_c:
2021-01-18 21:15:25	HIT	0a8b1069f217aed3f6f3307136f2e0a59d351ded15f339428af10cd011fb0a52	binary/octet-stream	cambr:
2021-01-18 21:15:25	HIT	dbdb4a3a06785feaa5e29dd1bf2466bb187ff3f967ddfe0e6827a5e5c92bf78e	application/octet-stream	ashbu
2021-01-18 21:15:25	HIT	4dbc8a020963bf0553e087fea64e85c875b4cdf1156e38141c27d67379629a40	application/octet-stream; charset=utf-8	louis:
2021-01-18 21:15:25	HIT	769a0fe4eb80fa5e2685ea60ac6a061ef8717eaa543fe17e3029113ba7c8dd12	text/plain; charset=utf-8	ashbu
2021-01-18 21:15:25	HIT	cbf0ab785c36706715ff57c25996a39327f49bf2add13d78a99830070f45ebef	application/octet-stream; charset=utf-8	dubli
2021-01-18 21:15:25	HIT	afca926e07a90d729928645dbfb566dda787599478a6b248c4c1ea506750c1c1	binary/octet-stream	london
2021-01-18 21:15:25	HIT	61cba6bf0889bcd8273eb5ca78245f636171ba83948cb30795b17130ed2aa962	application/octet-stream; charset=utf-8	san j
2021-01-18 21:15:25	HIT	794e7164d2f423d2a10173106e2b9c87601c2bea9b91fb78a1209c81881fdca5	text/html	pavas

Figure 9-1. Rubygems.org Raw Data in Table format²

In this example from the [Honeycomb play with live Rubygems.org data playground](#), a customized table visualization applies visual cues to the raw event logs in the data table. Rows have alternating colors to make it easier to read the table.

- **Bar charts**

Bar charts are useful for quantified categories of data that you want to compare when you have more than 2 or 3 categories. Compound bar charts extend the idea to visualize how the proportion of elements within a category contribute to the sum total for each bar. Bar charts are often displayed vertically, especially when representing time-series data, but a horizontal orientation can work better when using long category names.

For example, I've used them as ways to visualize system audits across multiple co-los to see the number of nodes running out of date operating systems. Similarly, using bar charts to display the disk consumption of a list of directories, partitions, or servers helps explain visually how storage capacity is being used.

- **Line charts**

Line charts plot changes in value and show patterns over time or relationships between two variables. Additional lines can be added to the chart to show trends between series. These are often the go-to for showing time-based trends, as well as differences between series.

Often the vertical axis will represent a statistic like the count, sum or average of a measured attribute across a dataset. On the horizontal axis, a continuous interval is used, for example, time.



Figure 9-2. Rubygems.org results in line chart format³

In this example from the [Honeycomb play with live Rubygems.org data playground](#), the raw data from the table before is visualized showing the cache hits, misses, errors, and passes over time.

- **Area charts**

Area charts are based on line charts and show quantitative data over time. Stacked area charts are useful to show part of the whole or cumulative values.

- **Heat maps**

Heat maps show data patterns through shading or color. One of the challenges of these kinds of graphs is making sure that the color schemes are accessible and don't create artificial gradients. Heat maps can also be problematic when there isn't a discernible pattern that can hinder

comprehension.

- **Flame graphs**

Flame graphs are a way to visualize profiled software and are helpful in debugging problems of resource exhaustion.

- **Tree maps**

Tree maps use tiles of varying sizes to illustrate proportions; they are in effect a two-dimensional compound bar chart. Tree maps are useful for showing the way that a total value is composed of many smaller elements. Tiles can be color-coded to convey additional information. For example, a tree map showing the used space on a hard drive would show large rectangles for files using a lot of disk space, and clusters of rectangles for directories; color coding of individual tiles can be used to indicate attributes such as file types, ages, or ownership.

ADDITIONAL RESOURCES FOR CHART VISUALIZATIONS

Learn more about the visualization of information from Edward Tufte's books [The Visual Display of Quantitative Information](#), [Envisioning Information](#), [Visual Explanations](#), and [Beautiful Evidence](#).

Learn about other charts from [AnyChart's "Chart Type: Chartopedia."](#)

Learn more about [Flame graphs](#) from the inventor Brendan Gregg.

Recommended Visualization Practices

In presenting information you control the narrative and provide a way to interpret the data. Contemporary tools allow us to explore the data available to us and interact, verify, or provide alternative narratives to explain what is happening.

Imagine you manage a cluster of load-balanced web servers. You might have a line chart of total errors with a different color line per server. Multiple lines can be visibly noisy but quickly show outliers in error types.

You might also have a graph per server that shows different shapes per error type. Different shapes show at a glance when a particular server was serving more errors, and whether the errors are associated with a particular type of error.

Apply these recommended practices when presenting visualizations:

- Distill your key points. Don't rely on text alone. Choose the right visualizations to support your key points.
- Use consistent colors in a dashboard with multiple charts and within a chart. Color directs focus. Lower the saturation for supporting or less important data. Limit the number of different colors in use. While color can be useful, charts need to be understandable even when reduced to greyscale.
- Graphs should always have labeled axes and a legend. Eliminate duplicate information within the graph, though. For example, if you are using bar charts and have labeled the categories, then a legend isn't useful.
- Include references to the sources of data. If something looks off about the chart, people can go back to the data to verify and dig deeper if needed.
- Design for the format. For presentations, lots of words will be hard to read and might obscure the most important message. For an on-call dashboard, more detail that provides clear and specific steps to take will be appreciated for those 2am pages.
- When visualizing a specific dataset, point out key observations using annotations and highlighting.
- Construct dashboards in a way that charts can explain each step of discovery. This is especially helpful if you need to rely on those dashboard for middle of the night on-call support.

TIP

See different visualizations of one dataset and how they change the message with [Nathan Yau's "One Dataset, Visualized 25 Ways" on Flowing Data](#).

Wrapping Up

An effective presentation of information provides audiences with the interpreted data and the context to understand and make decisions in a timely manner. Tell a compelling story that is tailored for the needs and interests of each audience by thinking about the nature of the data, the message you want to convey, and how best to express your interpretation of the data so that it makes sense for your readers or listeners. Remember that stories are at the heart of effective communication. When you're preparing to present information—to fellow sys admins, to leadership, to customers—ask yourself:

- Who is your audience? What do they care about, and what do they need?
- What is the nature of the data? What kind of story are you telling?
- What format will be most effective in reaching this particular audience? Should you present your information in writing, verbally, graphically, or as a multimedia presentation?
- What interpretation do you want the audience to understand? What context do they need to reach the conclusions you have in mind?
- What information does the audience need to understand the story you're telling? What information should be omitted because it distracts from the story? What information needs to be included, even if it might undercut your narrative, in order to allow the audience to reach conclusions that you might not have considered?
- Visualizations can be an effective way to convey meaning concisely. What type of visuals would be effective for telling your story?

You'll know your presentation is successful when your audience understands your message and is able to make decisions based on your information in a timely manner.

¹ “Beware Spurious Correlations,” Harvard Business Review, June 2015, <https://hbr.org/2015/06/beware-spurious-correlations>

² “Honeycomb’s Play with Live Rubygems.org” Honeycomb, honeycomb.io/play.

3 “Honeycomb’s Play with Live Rubygems.org” Honeycomb, honeycomb.io/play.

Chapter 10. Monitoring Theory

A NOTE FOR EARLY RELEASE READERS

With Early Release ebooks, you get books in their earliest form—the author’s raw and unedited content as they write—so you can take advantage of these technologies long before the official release of these titles.

This will be the 12th chapter of the final book. Please note that the GitHub repo will be made active later on.

If you have comments about how we might improve the content and/or examples in this book, or if you notice missing material within this chapter, please reach out to the editor at vwilson@oreilly.com.

Monitoring is the process of measuring, collecting, storing, exploring, and visualizing data from infrastructure (including hardware, software, and human processes). Monitoring helps you answer the “when” and “why” questions of your work, and it informs business decisions that support humans working in a sustainable manner(e.g., hiring so that your sysadmins are not constantly working at full capacity).

In this chapter, I focus on broad monitoring theory with the goal of providing you a framework to identify effective monitoring strategies. I will differentiate monitoring from observability, and explain the elements and steps of the monitoring process and how they work together. Understanding these mechanics at a high level will help you prioritize the different desirable outcomes monitoring makes possible, decide how and what you monitor, and increase visibility into your workflow, systems and teams, regardless of the tools you choose.

Why Monitor?

There are many reasons to monitor. All of them involve increasing visibility into

your systems. Visibility brings attention to weaknesses, fragility or risk, and helps you make better decisions. Some examples of achieving this visibility include:

- Problem discovery: You want to identify problems and know when and how those problems have been resolved (e.g., monitoring latencies of web requests and identifying when slow MySQL queries are impacting customers).
- Process improvement: You want to identify areas where your processes can be improved to make sure that your team is not overworked, increase accuracy and speed of task resolution, automate toil work, and improve overall efficacy (e.g., monitoring work queues to identify impact on the team).
- Risk management: You want to identify, evaluate, and prioritize potential problems (e.g., monitoring deployments of software, and adjusting automation or processes to reduce the frequency and severity of surprises).
- Baseline behaviors: You want to identify how the system behaves with normal traffic (e.g., monitoring data over a longer period to see your service trends to analyze different events that occur like holidays, weekends, and predictable news events like elections and sports events).
- Budget setting: You want to identify, evaluate, and prioritize infrastructure investment and enforce accountability related to spending (e.g., monitoring infrastructure spend to identify areas where different solutions may be more cost effective or set up constraints that enable engineers to test out new solutions without worrying about a surprise bill).
- Capacity management: You want to build sustainable capacity based on business demand. (e.g., monitoring infrastructure to identify when reserved instances will save money over ad-hoc instances).

Monitoring is so much more than implementing a single tool; it's identifying what you're trying to learn and desirable outcomes, and then assessing available tools and implementing practices that will best help you get there. Thinking about why you are monitoring, and establishing specific monitoring objectives encourages critical thinking around your business context so that you avoid

copying specific vendor-implied monitoring practices into your organization that aren't a good fit for your goals.

BE YOUR OWN AUTHORITY

A lot of practitioners tell us why and what to monitor, but I'm here to tell you that you are the best authority on your environment. Imagine for example that you are running a web service for your company. While it might be the same software in use at other organizations, the specifics about the web service vary between those organizations. You know your specific risks based on failures in different parts of the service as well as the different individuals that are responsible within your organization from development to support. All of these variables affect what needs to be monitored and the specific actions that need to be taken to derive the most business value while supporting the humans that run the software.

How Monitoring and Observability Differ?

Rudolf E. Kálmán introduced the concept of observability for linear dynamic systems in the 70s. Observability is a measure of how well you can see inside a system under observation with just the outputs. A system, in this case, is the collection of interrelated objects that are treated as a whole to model behavior. For example, a system may be a single host, container, or an entire distributed service.

Observability is not monitoring, and monitoring is not observability.

Observability is a property of a system; monitoring is a multi-step process of observing a system. Often, individuals think of monitoring as dashboards and production alerts. Framing monitoring in this manner leads people to define monitoring as a subset of observability. The problem with this definition then becomes: What do you call the other activities that you need to monitor?

You end up with overlapping terminology to cover all the potential use cases while also increasing the potential for misunderstanding. Monitoring has always been a broad process with a variety of different practices across organizations.

In some ways, it's a lot easier to think about the "unobservability" of a system.

Imagine for a moment that your customers experience a problem that your dashboards and alerts don't identify or explain. If your underlying data doesn't help you explain why and how the problem occurred, that indicates a lack of observability.

You can monitor the observability of your systems by assessing the variety of problems that occur, how often you are able to answer questions with existing data, and how often the final assessment of why a problem occurred is "I don't know."

NOTE

Terms are constantly evolving across teams, organizations, and the industry. Conflict arises in the monitoring community of practice over these terms signalling that there is a lack of shared context. For example, monitoring and observability and whether observability is a subset or superset of monitoring. Often these conflicts arise and build over vendors trying to be the perfect solution and in the process reusing words to mean different things.

Take time to build the shared context within the team around your use of monitoring terms. Then as you assess different vendor's monitoring offerings you will be better prepared to compare implementations and choose solutions that map to the way that your team works and thinks about monitoring.

Monitoring Building Blocks

To better communicate the process of monitoring, let's define some critical terms: events, monitors, metrics, logs and tracing.

Events

An event is a thing that happens, a fact that can be tracked. An event may be system, application, or service specific. Events occur regardless of whether they are being monitored.

Examples of events include:

- CPU utilization at a certain time
- The execution of specific code

- A sysadmin terminates an instance

Monitors

A monitor is a tool that defines and captures events of interest. They can be fixed or flexible.

Fixed monitors are specific functional checks against known issues that can't be customized by individuals at run-time. Examples of fixed monitors include event logs and CPU or memory gauges.

Flexible monitors can be changed ad-hoc. Tracing is an example of a flexible monitor that captures and records events. For instance, on a Linux system, you can run `strace` on a process to capture all the system calls made by that process. Flexible monitors are often used in diagnosing issues.

Monitors can be narrow or broad. Narrow monitors might define an event as a single instruction like a log that is triggered. Broad monitors might define an event as an aggregate of instructions, for example, a single web request that results in many system activities.

Monitors can be event-driven or sampled periodically. Event-driven monitors execute when the event occurs and aggregate over the reporting period. Periodic sampling monitors execute at a specific interval of time, collecting a statistically significant number of events.

Data: Metrics, Logs, and Tracing

Monitors collect data about configured events into three main types: metrics, logs, and tracing. They are collected from systems, devices, applications, and networks. You may be able to apply filters to limit the data collected or to sample in a way that represents the whole.

Most metrics are time stamped numeric values represented as a counter or gauge.

- A gauge is a value that reflects a point in time. A gauge doesn't tell you anything about the previously measured values.
- A counter is a cumulative value that reflects events since a point in the past.

When a counter reaches its upper or lower limit, it may roll over. Counters may be measured per time interval, and reset at the time interval. Counters may also be reset upon certain system events (such as reboots), or upon request. Counters that are measured per time interval and reset will not tell you anything about previously measured values.

Let's look at this difference between a gauge and a counter. A car's speedometer tells you how slow or fast you are driving. You use that information to guide your immediate actions by knowing whether you are traveling within posted speed limits. The car's odometer tells you how far you have gone. You use that information to guide preventative services like tire rotation and oil changes.

NOTE

Monitoring platforms may provide different metric types and implementations of these types may vary. Look carefully at the metric types, as the implementation will affect how the data about your events are collected and stored. Data reduced or aggregated too early may provide insufficient information for debugging purposes. Data that isn't reduced may lead to a flood of traffic that can impact network performance and the quality of service.

Logs are append-only records of events. Generally, logs are unstructured; the file format does not provide context or meaning to fields. Within a log, there is no implied relationship between records. Configuration changes of applications may alter which fields are displayed, affecting any scripts created to parse logs. Logs provide a lot more information than metrics but are more expensive to capture and store. Analyzing logs requires more specific tool customization.

Structured logs are structured in a key-value format that makes it easier for computers to process. Application configuration changes may affect which fields are displayed but won't impact existing scripts to parse logs. Event logs are structured logs that monitor broad events.

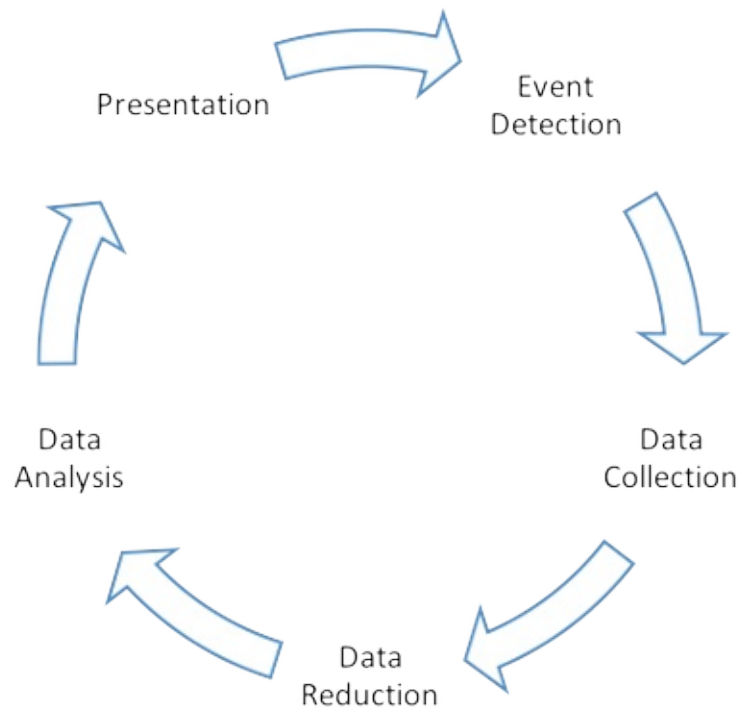
Tracing is a specialized form of logging to record a rich set of event data. Examples of tools that provide tracing include strace and tcpdump.

Distributed tracing is a specialized form of tracing that instruments an application to provide rich logs and metrics across different systems to connect contextual data across systems.

There are tradeoffs to consider between choosing metrics, logs, or traces. Metrics allow limited context to be associated with the data which minimizes the amount of resources required to store. Logs allow you to associate more context to the data you collect. Traces have the highest amount of context and require the most resources to store.

What does Monitoring look like?

The monitoring process includes a set of sequential steps: event detection, data collection, data reduction, data analysis, and presentation.



Let's look at these steps individually.

Event Detection

The first step in the monitoring process is event detection; events trigger monitors. Some monitors track the absence of expected events.

Data Collection

The second step in the monitoring process is data collection when monitors collect data.

Monitored data can be collected by:

- the monitored system pushing the data to the central monitoring server on a schedule or based on an event,
- the monitoring system signaling the server to push the data, or
- the monitoring system pulling data via a health check.

NOTE

Depending on the size of your environment and what you are measuring, a central server pulling data can create a scaling issue. This is one factor to consider when evaluating platform options if you have a larger environment.

The method of collection may create an observer effect; imagine the impact of a time-based collection strategy where every monitor checks at midnight. This frequency of monitoring can cause CPU or disk resource exhaustion, which increases latency and leads to unnecessary alerting.

The method of collection may change what you monitor and how you monitor it. For example, metrics are generally event-driven and aggregated over a period to compress data.

NOTE

If you have metrics that represent people, make sure you protect their privacy and obtain their consent in the collection of their data. With personal data and PII, you may have additional rules and regulations to follow, so when possible, avoid infringing user privacy by not tracking it in the first place.

Additionally, don't assume permanent consent, especially if you change the context or method of data collection. An example where you might need to think about this is telemetry data collected and logged from an individual's use of an application.

Data Reduction

In the third step, your monitoring platform aggregates and reduces the data. While this may happen to some degree at collection time, often it makes more sense to perform separately especially with distributed data.

Your monitoring agents collect data from many different sources. Your monitoring platform may aggregate, edit, sort, or compress the data down to its essential parts.

For metrics, sometimes the older data is aggregated for storage purposes while also providing some historical accounting to show differences against baselines. Older is contextual and could be weeks, months, or years.

For example, if you are monitoring request counts, you might not need 6 months of 5 minute interval data. Instead the count data could be aggregated so you have a baseline to compare against, but with reduced resolution and no ability to examine the original 5 minute intervals from 6 months ago.

Utilization over time of some metrics may be less useful. Storing metrics costs money, so aggregation is a balance of cost and usefulness.

Data Analysis

In the fourth step, you analyse the data to discover useful information about business and direct action.

During this analysis, you identify a set of service level indicators (SLIs) that help you measure the reliability of your system.

There are a few different ways to monitor for reliability including: availability, latency, throughput, and durability.

- Availability measures whether a system is operational and can perform the service as expected.
- Latency measures the time it takes to perform an action.
- Throughput measures the number of requests passing through the system.
- Durability measures long term data protection; that the stored data doesn't degrade or get corrupted.

Once you have SLIs, you can identify the achievable and appropriate levels of

reliability through setting service level objectives(SLOs). Because it is very difficult (and costly) to provide better reliability than what you depend on from external service providers, you must factor in those dependencies when setting your targets. Don't forget to factor in network and DNS.

Data Presentation

The fifth step in the monitoring process is the presentation of information. To transform data into information, you create visualizations. You collect charts into dashboards that cover areas of known bottlenecks and elevated risk. You create other ad-hoc visualizations to explore available data.

You may create charts based on real-time off-line data. For example, alerts should be as close to real-time data as possible to limit the impact of problems. Quarterly capacity planning for a Hadoop cluster may be the aggregation of various data sources and processed off-line.

Dashboards aggregate a set of visualizations to communicate information. The specific dashboards you create depend on what you envision people doing. People could be making a one-time strategic decision, determining day to day operational direction, or reviewing the system weekly or monthly to establish tactical direction.

These dashboards are products that drive action. Outcomes and information should feed back into the various team and organizational processes.

Monitoring for Sustainable Work

The monitoring process is iterative. Monitoring provides information to help you analyze what is happening and the supporting evidence to educate the team and drive changes. Sometimes these changes are to the compute infrastructure, other times it's to the human processes. People are part of the systems that you manage from development to support in production.

For example, I have been in environments where the average work load for the Ops team meant that we each worked at our full capacity. If any one took time off whether planned or unplanned this led to extra stress on the system which led to increased mistakes in resolving incidents, and frustration with one another in

the team. Monitoring helped us to establish that we needed additional people on the team based on our expected work load. This gave us extra capacity when everyone was available, but reduced friction when people needed time off.

Chapter 11. Developing On-Call Resilience

A NOTE FOR EARLY RELEASE READERS

With Early Release ebooks, you get books in their earliest form—the author’s raw and unedited content as they write—so you can take advantage of these technologies long before the official release of these titles.

This will be the 18th chapter of the final book. Please note that the GitHub repo will be made active later on.

If you have comments about how we might improve the content and/or examples in this book, or if you notice missing material within this chapter, please reach out to the editor at vwilson@oreilly.com.

The most critical responsibility of supporting a service or system is on-call and the management of impactful events. When you are constantly being paged to repair the system, you may not have the time or energy to repair the underlying infrastructure, software, or services effectively. In extreme situations, you may avoid thinking about the on-call experience when you are not on-call because it feels better to accomplish project work. In this chapter, I propose a framework for building resilience, investing early and regularly to prepare for on-call.

What is On-call?

On-call is a temporary rotating role assignment that may include being reachable outside of normal business hours (e.g., evenings, weekends, and holidays) to answer requests for support and handle discovered alerts. When you are on-call, you are one of the people responsible for this work that comes in for a specific length of time. Depending on the size and distribution of the team, on-call rotations may consist of 8 to 24-hour shifts for one to two weeks.

On-call duties vary widely within different organizations from failed application

services to power outages. You may be the person to respond to services going offline or provide escalation support. You may have to investigate why a website went offline in the middle of the night, or scramble to restore backups when a file server crashes. Some on-call is for the very rare issues “just in case”; in others, paging is so frequent that it feels like a full-time job. Often on-call and interrupt-driven work tends to merge into a single work queue.

Many contributing factors lead to unsustainable on-call practices that transform the sysadmin job into task based reactive work lacking growth opportunities. Two prime factors are misalignment in severity and priority assessment.

When individuals assess the severity of a problem too high, they may demand a fix for an issue even if a viable workaround exists; assessing too low, can lead to under-prioritizing a problem that affects many people.

Operations team may have difficulty assigning priority. Problematic practices include assigning all interrupts at a high priority automatically, failing to rank incoming issues, not combining duplicate reports that are the same issue, or failing to clean up known problems to eliminate the possibility of duplicate reports.

Ideally, the urgency of a request and impact of the problem is known and shared including:

- How many people are affected?
- Is there a satisfactory workaround?
- Is data at risk?
- What’s the business impact on your organization?
- What’s the business impact on your customer(s)?

Let’s talk more about what tools and techniques are available to help you improve your resilience through refining the on-call process.

Humane On-call Processes

I’ve been there. Late-night pages and interrupted sleep. Years of waking up in a

panic, wondering if I missed an alert? Skipped vacations and missed meals, or eating whatever cold pizza was left from the team huddle as we resolved a large revenue-impacting incident. Missing out on family and friends events, and relatives expecting that I would bail again. I have painful memories of on-call that have had long-term impacts on my relationships, mental, and physical health. I eliminated the very activities that could have helped, because I didn't see the path to a more sustainable experience.

It doesn't have to be this way. While you have an obligation to your company, you also have a responsibility to yourself and to your health. You can be a responsible and attentive worker who is on-call, while at the same time advocating for yourself, and maintaining relationships with your friends and family.

In the next few sections, I'm going to share my recommendations for a sustainable on-call, from the preparation steps you can take before on-call even begins all the way through your on-call shift and the handoff meeting. Compare your processes to what I describe here and adopt practices that help you.

Preparing for On-Call

During the weeks leading up to your first on-call shift, make sure you know about all the systems you're responsible for and the escalation path; in other words, who do you ask for help and when you should pull them in. Part of understanding your systems is knowing the availability expectations for the systems you are responsible for: in some cases, an outage of minutes or even seconds is a critical problem, while in others, an outage might not have a customer-visible impact, and it's enough to leave a note for someone to deal with it the next day.

Regardless of whether there is a formal process of participating with other on-call engineers (also known as shadowing on-call), ask if you can shadow others on the team. Shadowing allows you to see tools and processes in use, examples of how to respond and interact with the team, and assess the cost of the on-call experience to you.

Shadowing also helps you get a sense of paging frequency and the typical response norms:

- How are new incidents reported?
- Is there an email or SMS message, or a notification in a messaging service like a Slack channel, or a status report in a dashboard?
- Does a service ticket get generated?
- If so, does this happen automatically, or does someone need to file one manually?
- How promptly do requests need an acknowledgement?
- How quickly is a resolution required?
- If a solution requires specific expertise, what is the escalation procedure?
- When is it considered appropriate to escalate?
- After an issue has been resolved, what additional steps are taken to make sure the problem doesn't happen again?

Make sure your laptop and phone are charged and up-to-date with software requirements and that you can access the services you need from home and wherever you may be during your on-call shift: your favorite coffee shop, the soccer field or bike path. Depending on the nature of your on-call rotation, you should have the latitude to do these sorts of things, as long as you can receive and acknowledge requests promptly, and are prepared to help resolve problems as they arise.

Bookmark the different services that you need and make sure that you can log in and access them. Then, when you get paged, you don't want to be fumbling around trying to find where you need to go to learn more about the pages.

Configure your phone and other devices in your alerting service. Services have different escalation policy customizations, so make sure to enable more than just email. For example, I focus on alerts when I'm on-call, so I prefer to minimize the distraction of future alerts on the same issue while still enabling redundancy. For an expected response time of 15 minutes, I like email and SMS, with a 10-minute follow-up phone call if I haven't responded. This configuration gives me 10 minutes to respond to the SMS before I get another alert, which reduces potential duplicate alerts and gives me time to respond within 15 minutes.

While teams have a specific expected response time, you also can configure your preferences. It's essential to consider the requirements of the on-call rotation and response time and your way of working. Find the balance of being responsive while not getting frustrated by noisy notifications.

Check your company's expense policy and talk to your management about expensing additional charging cables for all your devices to help eliminate the dreaded "did I leave that cable" panic. For example, I like to have extra power cords for my laptop and phone in my on-call bag that ensures that I don't have to break down any part of my day-to-day setup or worry that I've forgotten a cable.

NOTE

Battery packs or power banks can give you extra time to resolve issues on your phone and laptop.

While you may not make and receive phone calls regularly, be prepared to have voice or video conferences during your on-call rotation with a hands-free headset so you can continue to type without sacrificing the quality of sound with the speakerphone.

A mobile HotSpot or Wi-Fi tethering device can support sustainable on-call rotations by enabling you to work from anywhere. Instead of being limited to the distance between your working station and the expected response time to resolve an issue, you can find an available spot and connect when you get paged. Having a mobile HotSpot allowed me to enjoy family picnics and log in from the park to resolve issues that often took only a few minutes.

A separate device allows you to use the phone to further alert on other issues or dial into conferences as needed. It increases the diversity of connection options — if your phone has service from one provider and the device gets service from another, you're more likely to have access to a viable signal.

One Week Out

The week before your on-call shift, you can notify any stakeholder teams depending on your work and update associated project tickets to share status

information. By updating the project tracking system with information about your upcoming on-call, you minimize the unplanned stress folks might have about specific work. Hopefully, proactive updates also reduce the project work interrupting on-call. If there are critical time-bound tasks, let your manager know and support delegation of those tasks. An up-to-date documented state of the project means others can chip in to keep the project moving forward if you get pulled into supporting a long-running incident.

If possible, send test alerts to confirm that you're enabled to receive alerts. Even if you have checked for past rotations, ensure configuration changes haven't eliminated your notifications. I have uncovered problems with alerting services blocking my phone provider, which saved me from dealing with failed system alerts and debugging why the phone provider was blocked.

Plan your snacks and meals ahead of time. Self-care is especially critical during an on-call shift. When and how often you'll get paged is unknown. While you can estimate what will happen based on past performance, it's not a guarantee. For the things that you can plan, this will help eliminate additional stressors when cascading failures occur. Energy bars can fill the gap, for example, when you have to start your day earlier than expected and need something quick to get your brain going.

NOTE

Relationship builder: Do you have family or friends who you can depend on to support you through on-call? Ask for help. Bring people into your experience. You don't have to be isolated and giving people the opportunity to help you can help build connections, especially if you reciprocate when you're no longer on call.

Plan for any additional coverage. Do you have a long commute or a regularly scheduled doctor's appointment? Do you need to drop your kids off at daycare or attend a soccer game? Do you need to take your pet to the vet? Talk to the secondary or, ideally, another engineer that can provide coverage. Remember to reciprocate support when others need it.

Configure these overrides in advance. On-call rotations need to factor in the real demands of personal life responsibilities and be flexible. A team that already

practices this will be more able to handle additional short-term demands from outages.

Connect with the rest of the on-call team. Ideally, there is a secondary, other escalation points of contact, and an incident manager. The point of this step is to give you additional confirmation that everyone is ready for your participation in on-call.

TIP

While you don't have a responsibility to reach out to everyone on-call, doing so helps build and sustain meaningful connections for successful, minimum drama rotations. There have been a few times where I've discovered that folks had planned a vacation, and this helped prevent holes in coverage.

It's also helpful when the on-call team is a virtual team composed of folks from different roles who may not have an awareness of the different skills that the individuals bring to the on-call rotation.

Connect to specialized engineers. While there might not be an official on-call if there are single points of responsibility within the organization, it's essential to have contact details for your security, network, or database engineer. If they are not part of the on-call rotation, identify under what conditions they should be notified as an escalation point of contact.

Talk to your family or roommates about upcoming on-call. set the expectations around what an event looks like and the expectations they may have of you. Set boundaries around acceptable behaviors (e.g., no hosting parties on your on-call weekends)

Preparatory work is necessary for going on-call. Make sure that time-allocated for the week doesn't focus on a project's progress to the detriment of on-call preparation.

The Night Before

Verify that your notification device is charged and not silenced or in do-not-disturb mode. Get enough sleep; restedness is a crucial component to being able to sustain alertness to a changing environment. If you're fatigued going into an

on-call rotation, it will hinder your effectiveness at sustained attention.

NOTE

I asked on Twitter “**Best recommendations for things to have/do when going on-call?**” and received many suggestions from other experienced sysadmins. One area that is often overlooked is preparing comfort for future you. A couple of suggestions:

- Sera [@tsdubz]. (2021, September 19) **Keep a warm hoodie/dressing gown near the bed for less cognitive load on those 2am wakeups**
- Yvonne Lam [@yvonnezlam]. (2021, September 19) **Tea/coffee beverage of choice set up and ready to make should you get paged in the night.**

Think about the accommodations (beverages, food, and/or clothing) that will provide comfort of convenience when your time is constrained to respond to an outage that may last for awhile.

Your On-Call Rotation

Throughout your on-call rotation, the overall process may vary based on your teams’ expectations, but a general approach includes:

- Receive Alert(s)
 - Acknowledge the alert(s)
 - Triage
 - Fix
- Improve On-Call Experience
 - Documentation
 - Monitoring
 - Assessing normal

When you receive an alert, the first action is to acknowledge the page. An acknowledgment lets folks know that you have received the alert and will help minimize further interruptions for the same issue.

Next, triage or assess the severity and urgency of the problem and, based on

these factors, route the alert to the appropriate action.

Finally, fix the problem that is being alerted. Fixing includes adjusting a noisy alert that pages with no expected action.

Assess your on-call readiness. High impact lengthy incidents and numerous frequent alerts are both concerning. It may be better for you and the team to hand off primary on-call to someone while you take a break.

NOTE

Assessing on-call readiness needs to be formalized within the team's processes. A few examples of what that would look like:

- If a team member gets a page after standard working hours that takes over an hour to resolve, then the team member automatically is granted that additional time to come in the next morning.
- If a page takes more than 8 hours to resolve during the work day or 4 hours to resolve after hours, then the team member automatically gets the next work day off.

Having explicit policies helps increase team resilience as individuals are more willing to be a member of the on-call rotation, which helps build the layers of redundancy required so that people can cover for each other when someone needs to take a break.

During the typical on-call workday, when not receiving an alert, the focus is on improving the on-call experience (versus working on project work). Workday tasks could be improving documentation or monitoring, or in learning more about what “normal” behavior looks like in your systems. Sometimes in the process of examining the live system, you’ll discover something that is impacting and requires fixing. Make sure that these discoveries are documented (in the work queue as well as the on-call handbook) and alerts configured.

On-Call Handoff

Ok, so the clock hits the magic hour, and you are no longer the designated on-call. You want to be done. But you’re not done yet. You still need to hand off to the next on-call engineer. Making this an official sync meeting will do two things. First, it will support the incoming engineers by informing them of the past week’s issues and any remaining open issues so they’re set up for success.

And second, it will give you a much-needed psychological release to have an explicit stopping point to the hyper alertness required of being on-call for production. It's a ritual of finality that tells your body it's okay, you can stop now, and it is glorious.

But it's also a ritual of beginning because it sets the starting point for when the next person needs to take on the mantle of hyper alertness. When it's time for you to start on-call, your colleagues should be handing off to you in the same way, otherwise you may stress more about expectations and whether something is already a problem depending on the state of the systems you are managing.

You may think, “ my environment doesn't have these concerns, my environment isn't that complex, we don't get paged a lot, etc..” But we're not trying to optimize for environments that are calm without regular issues; we're trying to create team processes that are sustainable regardless of the inevitable issues and incidents that may arise: data corruption, loss of data centers or cloud provider outages, security incidents. A clean handoff sets yourself and your team up for success when problems do arise because the team is already well practiced in how to hand off responsibility with ongoing issues so that individuals are well rested and at their best when tackling thorny or complex ongoing problems.

Part of the handoff includes a weekly review document. An example of information included:

- Time period
- Individuals who made up the on-call team for the time period
- Incidents and relevant links to more information about those incidents
- Open Incidents
- Resolved Incidents
- Incidents that were not captured by alerts
- Manual work
- Opportunities for automation and improvement
- Open questions; While there may no longer be an impact on consumers, there might still be unanswered questions.

- Call-outs for specific items that went well and what needs improvement.

The weekly review document is crucial. I can trust that the person before me has handled things and is supporting me through documentation, and the next person can trust that I will handle things and will support them through my documentation.

Handoff procedures are vital for effective collaboration across regions. A good practice is to have quick standup video conferences for shift transitions, where the people who are ending their day can bring the next group up to speed on what they've been working on. On an ongoing basis, sharing case notes in a ticketing system like Zendesk, or a chat system like Slack, can make it much easier for regional teams to be able to pick up where their colleagues left off. Additionally, searchable case notes lay the groundwork for internal and customer-facing documentation, as well as bug reports for the software team.

The Day After On-Call

Being on-call is done, but it doesn't mean that you're done doing work to improve on-call. While the events are fresh (either the same day as the handoff or the very next day), revisit the issues you filed. This is the best time to have those creative epiphanies to improve what you just experienced. Update necessary documentation, clean up any noisy alerts (which includes reducing the severity of alerts as appropriate) and record any project related work required for long term improvements. For any incidents, add relevant information to the incident report.

TIP

One way to help continuously improve on-call alerts is to have a regular alert review with your team to talk through the impacts and values of the alerts.

VARIABILITY IN THE ON-CALL EXPERIENCE

Chris Devers

Reading this chapter, I was struck by how the on-call experience where I

work differs from what this chapter describes. Most of the on-call work I've had in my career deals with humans as much as it was with the systems, e.g., "the news system just crashed and we're on the air in 17 minutes, help!". Yes, there's a technical aspect to the response, but there's a great deal of human interaction, talking to frustrated people and improvising solutions to get them to acceptable states of resolution.

My employer builds solutions for the media and entertainment industry, where on-site server deployments continue to play important roles. People working in this field need to deal with things like cameras, tape decks, satellite links, broadcast systems, and vast archives of media, and the servers we build help tie it all together so that the show, as they say, can go on.

Many of the systems we sell are physical servers, which our customers install and manage themselves, at their studios, offices, and data centers, wherever in the world they may be. Routine administration of these systems is the customer's responsibility, but if they run into problems, they can turn to us. Our tech support staff are, in effect, a team of consulting sysadmins, providing escalation assistance for the on-site admins at individual customer locations.

We're not a huge company. But we do have offices around the world, and this has been key to maintaining a sustainable approach to on-call work. If a broadcaster in India reports an overnight problem, the on duty team in Europe is ready to assist; if the problem extends past the end of the work day in Europe, the case is handed off to the Americas team as their work day begins. Similarly, global staffing allows holiday coverage. Regions adjust shifts to provide coverage so that we minimize impact to customers when our regional offices close for holidays, whether it's Lunar New Year in East Asia or Thanksgiving in North America. And when the COVID-19 driven shift to widespread remote work came along, we took this in stride, because we were already used to collaborating with remote colleagues and customers.

We do a shift rotation for weekend coverage that resembles the rotation described elsewhere in this chapter: people need to watch email and Slack notifications on their phones, and be ready to get on a laptop at a moment's notice. Or perhaps a customer has scheduled a weekend maintenance

window, and the engineer on duty knows in advance how their Saturday is going to unfold. But late-night investigations are rare, because cases are handed off regionally, just as they are during the work week.

We also encourage a close working relationship between our support and dev teams, which brings a variety of benefits. The support team, of course, are keenly aware of the customer pain points, but they also get excellent feedback about how to improve and extend the product. At the same time, it can be rewarding for developers when they see that the work that they're doing is meaningfully improving things, not only for customers, but also for support staff. This collaboration also helps distribute knowledge: if a particular individual is a recognized subject matter expert on a particular aspect of the product, it makes everyone's jobs easier when that person shares their bag of tricks. Obviously, having a lot of interruptions can make it difficult to focus and get things done, and everyone tries to be mindful of this. But when the benefits of such collaboration are recognized, it seems to be easier to get more people on board, and this can lead to a positive feedback loop: the support team level up, the escalations get less frequent, and the devs aren't consulted as often.

Each organization needs to craft an approach to on-call that is adapted to the problems you need to solve, and the resources you have to work with. In my case, working with a global team has led to a low-impact approach to on-call duty. Think about how your own organization may apply creative solutions to sustainable on-call coverage.

Monitor the On-Call Experience

Once again, monitoring is not just for production systems, and it's important to monitor the human systems. The on-call process itself requires monitoring in order to be aware of what is not working and proactively iterate on improvements. This is tied into advocating for yourself. To know whether on-call sucks and to provide that supporting information to management who can make change, you have to have monitoring that measures and presents that information in compelling ways. See presenting information and apply these improvements to how you share the measurements you make about on-call.

The first measurement includes monitoring work in progress, even if you're the solo on-call engineer and you don't need to explain your work to anyone. Ideally, work associated with alerts should come into a shared work queue. You want to be able to share visualizations over time of the work and when you make change, you want to be able to see the impact. By measuring first, you can establish the baseline, and can then observe the impact that changes (like more people on-call, specific improvements to code or infrastructure) can have on the work being measured.

Here are a few questions to think about and consider monitoring in your environment:

- How many on-call hours per time period?
- How many active on-call hours per time period?
- How often does an alert page?
- How often is it actionable? Does the alert self resolve?
- When was the alert last updated?
- When was documentation last updated?
- What is the impact of the failed system? Does it need to alert outside of hours?
- How much coverage is available? If an individual is paged out and resolving an issue, who takes the next page?
- How often does the person on-call get diverted from normal life activities, including: sleep, meals, and showers?
- How often are family gatherings and obligations interrupted? There are many activities that can't be rescheduled and are critical to healthy relationships.

Rather than just focusing on system time to recovery and time to discovery, these metrics help to classify and direct improvement in the on-call experience. During production meetings, it's helpful to talk about these metrics so that the team notes the necessary action items to improve the observed trends.

If your team has periodic retrospectives, think about the on-call progress. Potential remediations you can suggest may include updating the paging schedule and escalation policies. (If your team doesn't have retrospectives, I encourage you to suggest them.)

Wrapping Up

Supporting your system through participating in an on-call rotation is part of managing systems, but on-call can be handled in a humane way that is compatible with a healthy lifestyle that includes time with friends and family and engaging in activities outside of work that you are passionate about. You can take time to step back from your usual routine, and focus on how the systems you oversee can be managed in a more maintainable and sustainable way.

TIP

For more resources about on-call, check out:

- [Crafting sustainable on-call rotations by Ryn Daniels](#)
- [The On-Call Handbook by Alice Goldfuss and contributors](#)
- The Notifications chapter in *The Art of Monitoring* by James Turnbull

Chapter 12. Managing Incidents

A NOTE FOR EARLY RELEASE READERS

With Early Release ebooks, you get books in their earliest form—the author’s raw and unedited content as they write—so you can take advantage of these technologies long before the official release of these titles.

This will be the 19th chapter of the final book. Please note that the GitHub repo will be made active later on.

If you have comments about how we might improve the content and/or examples in this book, or if you notice missing material within this chapter, please reach out to the editor at vwilson@oreilly.com.

As we explored in the previous chapter, the purpose of on-call is to be aware of your systems so you can keep them healthy. But as much you strive to reduce risk, failure will happen — there will be incidents. Incident management begins when you detect a problem during an on-call rotation, but management often extends beyond on-call when other subject matter experts and teams are required for issue resolution. The aim of incident management is to minimize the impact of an incident.

You as an individual need the kinds of tools, techniques and practices that will not only get you through an incident with minimal suffering, but will also help you feel prepared ahead of time and able to react effectively when an incident occurs. You need good, clear communication across teams so that the appropriate subject matter experts can share their knowledge and minimize time to resolution. And you need a way to capture and apply what you learned from the incident to improve overall production, reduce future impacts to customers, and reduce the team’s toil.

In this chapter, I share the framework for collaborative and sustainable incident management from identifying incidents to post-incident reviews and identifying the actions required to improve the live environment.

NOTE

I am assuming your team has incident management and that you'll have some framework to apply what I'm sharing to improve your experience. If your team doesn't currently do incident management, then share this book or the Leading Sustainable Teams chapter with your leadership team.

What is an Incident?

The definition of incident varies across organizations: an incident may be anything that pages the on-call engineer or specifically security breaches. In this book, I define an incident as an *exception* to a *live* site, service, or software application that has an *impact*.

Let's break this definition down into the components starting with exceptions. Exceptions occur when the system doesn't behave in an expected way. Exceptions can be bugs in the code, failures in underlying systems (like DNS or the network), or misunderstanding in the project planning that led to a different implementation.

A live site, service, or application is something that is in use by clients or customers. In many cases, this is the production environment for a site or service but also includes applications installed on devices.

Impact is the qualitative effect that the exception has on the clients or customers. Sometimes, this impact may be visible externally. Other times, the impact isn't visible and a decision needs to be made about whether to disclose the incident or not.

Some examples of incidents:

- In October 2021, the loss of IP routes to Facebook DNS servers led to a global outage of over six hours to Facebook and its subsidiaries' sites. When the system went down, it also took down the system that controlled keycard access to the buildings and server rooms, so nobody could access the servers remotely and the sysadmins on-site couldn't get into the buildings and server rooms to do hands-on mitigations.
- In July 2020, an expired server certificate and a data outage prevented the

California Reportable Disease Information Exchange from accepting COVID lab results from external partners leading to discrepancies and under reporting of case information.

- In October 2019, Docker experienced an incident where the Docker Hub registry was down. Any organization that relied on directly pulling images from the registry would have experienced issues that relied on these images being available. Organizations that cached docker images or hosted their own registry would have minimized their impact.
- In May 2019, **Slack** started a deploy of a feature that prevented some customers from connecting to and using Slack. For organizations that were impacted, this was a complete outage.

As you can see from these examples, incidents can vary in degrees of external impacts. Additionally, incidents may be near misses that your customers have not (yet) observed.

What is Incident Management?

Managing an incident is more than how you respond to the impactful event and restore your system back to its operational state. Incident management is the process of planning, preparing, responding, investigating, and learning from the incident.

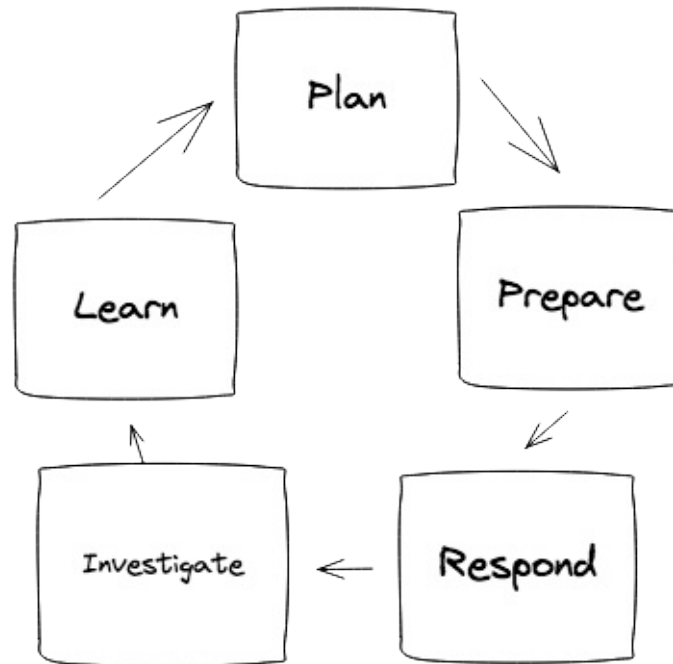


Figure 12-1. Incident Management Cycle

Outcomes of these different parts of incident management lead to:

- Reducing damage, costs, and recovery time
- Identifying code or process issues
- Repairing issues to prevent repeat incidents
- Documenting incidents
- Learning from the investigation

Each step in the incident management cycle shares basic principles that include clearly defined roles and responsibilities, as well as opportunities for continuous collaborative learning. Effective incident management may lead to data to support headcount requests, improved training, and promotion artifacts.

RECOGNIZING WHEN IT'S REALLY THE SYSTEM, NOT YOU.

In the last chapter, I talked about building your individual resiliency to support your production services during on-call. Yet, sometimes there are components of the work that are outside of your control and no amount of

individual resilience is going to support sustainable work.

When it comes to managing incidents there are some warning signs that your role has limited growth opportunities. A few of these signs are:

- Lack of transparency around failure,
- Blame and fear culture where folks are afraid to talk about mistakes,
- Repetitive incidents without improvement or long term correction.

There are other problematic issues, but these are especially harmful as they hinder learning, disrupt trust and relationship building, and promote burnout which can compound the impact of incidents. If you see these signals and can't change your work to make it more sustainable, find a new opportunity before you burn out because it is especially challenging to interview for a new job when you're already depleted.

Planning and Preparing for Incidents

While you can hope that nothing happens during your on-call rotation, inevitably something will happen so have a plan and regularly prepare for incidents. With contemporary systems, this requires collaboration and coordination within and across teams to communicate to the various stakeholders with a consistent and reliable response. In some organizations, an incident response team (IRT) or incident management team (IMT) is the ad-hoc temporary team that is created to coordinate and collaborate to resolve an incident.

What planning and preparation steps do you need? Consider these:

Set up and document communication channels.

During an incident, a team shouldn't be trying to figure out the process for how everyone will communicate, especially when individuals might not be in the same place or even time-zone.

There is no one right way for handling incident discussions.

One approach, is to create a single #oncall channel where on-call discussion occurs. When a significant incident is identified in the discussion channel, a

new #incident_NUMBER channel is created, keeping the primary #oncall channel uncluttered by the highly focused incident needs so that other potential problems aren't hidden. A problem with this approach is managing and tracking a lot of short-lived channels.

Another approach, is to create a single #on-call channel. When an incident occurs, discuss the incident in threads. This helps with organization and visibility of incidents, but it can also make the channel overwhelming, especially when incident-related threads stretch to hundreds of messages.

A third approach, a compromise of these two approaches: start with threads in the main #oncall channel and be mindful of the scope of the investigation. Break out into separate channels when it becomes necessary.

Choose a standard, and change the approach based on how it works for your team.

Train communication.

Being explicit about the expectations around communication during an incident reduces mistakes and time to resolution. Remember from Chapter 14, consider also the level of detail that is appropriate for your different audiences: internal teams working a case need to share unfiltered real-time information, but managers overseeing things might only want periodic status reports, and customers and other external stakeholders may only need a brief summary.

Create templates.

Templates help guide consistency across incident management and improve efficiency as individuals have a structure and layout to start from. They set expectations and standards.

TIP

People may chafe at templates with too many rows or fields. Make sure that templates focus on the minimal required information.

Maintain documentation.

On-call and incident handling documentation should be reviewed and updated regularly. Stale documentation that doesn't reflect the processes in use hinders organizational learning as well as frustrates engineers. Make sure to review the processes to handle alerts, disaster recovery, and other artifacts that might not seem to be documentation at first glance.

Document the risks.

What are the risks that you are exposed to, what's the probability of those risks occurring, and what are the associated impacts? The goal of incident management isn't to eliminate incidents, but to reduce risks in a way that lets your organization continue to make changes.

Imagine the failures that could happen and explain what would cause them. This also helps prepare backup plans and highlight any factors that could influence successful resolution.

TIP

Read more about risk from Marc Alvidrez in [Chapter 3](#) of the Site Reliability Engineering book from Google.

Practice failure.

Exercise and review your incident handling procedures so that you have the different steps ingrained. Much like testing in development and the live running of your system, practicing response to simulated failure is a very different experience than a live incident. But even with a practice run, you can still identify gaps in documentation and processes that will provide a much better experience for you when responding to an event at 2am.

Understand your tools

Your team will have a collection of tools, practices, and processes for incident management. Some examples of tools to be aware of include:

Table 12-1. Tool Categorizations

Category	Purpose
Monitoring	Measuring, collecting, storing, exploring, and visualizing data from infrastructure
Alerting	Manage on-call rotations and escalations, and notify designated on-call responders
Chat Service	Real-time communication that provides a place to share observations, links, and screenshots
Video Chat	Real-time communication to discuss and agree on approaches for incident response
Incident Tracking	Process, troubleshoot, and track the overall progress of the incident
Documentation	Categorization and aggregation of artifacts (incident management reports, incident research)
Issue Tracking	Process, troubleshoot, and track the overall progress of issues with your systems and software. This may or may not be the same tool used for tracking incidents.

Make sure that you have an account on each of these tools as necessary and the method for accessing each tool whether it's a special application that you install on your phone or a URL.

Clearly Define Roles and Responsibilities

Incident response teams vary across organizations. If your organization has an IRT, there may be different names for specific roles and more or less differentiation. A few significant functions (whether they have these names or not within your organization) are the incident commander, subject matter expert, liaison, and note taker.

- The incident commander (IC) is responsible for driving an incident to resolution. During an incident, there is always a single acting lead to coordinate the various activities. The responsibility may be passed from one individual to another throughout the resolution of the incident.
- The subject matter expert (SME) is the on-call engineer or the designated owner for a particular part of the service. There may be a number of subject matter experts required to resolve a specific incident.

- The liaison is responsible for communicating internally and externally about the status of a current incident. There may be multiple liaisons for handling the different messaging internally and externally for a specific incident depending on the scope of the incident.
- The note taker takes notes, filling in details about the important actions and followups that occur during the incident. This might be done through the use of software that responds to special commands or a chatbot. Handling incidents via a chat tool like Slack or a recorded video conference can fulfill this role too, because both of these provide timestamped transcripts of what was discussed. These notes are critical for providing the context for the narrative that will drive learning for the incident later.

If your organization doesn't have an official defined process for incident response, this may be an area to refine to help support sustainable on-call and incident management. This will require leadership buy-in.

Understand Severity Levels and Escalation Protocols

When you are paged while on-call, you need a reliable way to prioritize pages and identify an issue as an incident; understanding how your team assesses severity levels helps you decide what to do and who to tell.

Lower number severity levels generally indicate higher impactful incidents. An example of what this might look like:

Severity 1

A critical incident with high impact; for example, this could be a system completely compromised impacting all customers, a privacy breach due to a hacked system, or the loss of customer data.

Severity 2

A major incident with significant impact; for example, this could be a degraded system impacting some customers.

Severity 3

A minor incident; for example the system may be slower to respond but not

completely down.

When there is a common understanding of what severity levels mean to a team, communicating the severity level can quickly initiate appropriate escalation protocols that bring the right level of response.

The more severe the incident, the more important it is to have different people handling the different roles of incident management.

Responding to Incidents

Every team will have some sort of process (whether documented or not) on how incidents are handled. Reflecting and documenting explicitly what each part of the process looks like can help in improving coordination when you actually are handling an incident. Here is an example of documenting the process of managing your incident response with clear roles and responsibilities for the different parts of the incident management team.



Figure 12-2. Incident Response Cycle

Assess

The IC assesses the incident through the observed symptoms, scope of the problem, and potential risks based on the symptoms.

Act

1. The IC identifies possible actions and associated risks.
2. The IC makes a decision.

The IC says the decision out loud if on a call and in channel if on a chat

platform.

3. The IC obtains consensus on the decision.

The IC asks whether there are strong objections to the decision. The IC adjusts actions based on feedback, but ultimately the IC makes the final determination.

4. The IC delegates stabilization actions.

The assignments must be clear and specific with explicit timing information about when the individual will update the team with progress.

Sometimes, an individual might not have the skills to do the identified action. It could be a good time for the individual to learn with guidance from someone with the experience. If there isn't sufficient time or there are too many tasks, then the stabilization step should be handed off to someone more experienced.

Assignments should be adjusted based on feedback and required timelines. Depending on the severity of the incident, this may require pulling people on to the incident response team to complete the required tasks in a timely fashion.

Inform

Depending on the size of the team handling an incident, the IC may name an explicit liaison to handle updates. Liaisons shouldn't be actively investigating and repairing the system. Shifting contexts from debugging to communications to executing critical commands can exacerbate stress and increase mistakes.

When the live site is in a degraded state, clear, timely communication to customers requires skill. Poorly worded explanations can cause more problems than the actual outage.

The Liaison(s) sends regular updates to the team, customers, and executives.

The frequency and content of the communications will vary by audience. Updates should include what is happening and the steps taken. . As an

individual's expertise is no longer needed, the IC reduces the scope of the incident. The IC informs the incident response team who is still required to resolve the incident and encourages folks who are no longer needed to take a break.

Verify.

1. The IC checks that the subject matter experts completed stabilization actions.
2. The IC checks the outcomes of those actions. If there is a continued impact, they repeat this action loop starting from accessing the incident.

Learning from the Incident

After the incident has been resolved, collate information from all the participants of the incident response. The goal of this is not to place blame, but to uncover what happened and drive conversations. One way to help prevent blame is to make sure that the focus is on what happened and what people decided to do based on that information rather than trying to talk about what should have happened, or could have been done.

How deep should you dig?

Organizations vary in size and complexity. There may be regular incidents of varying degrees. Have you ever been in a post-incident review meeting where it felt like the goal was just to go down a checklist, rather than focus on the impact of the incident and how it was handled? I've definitely sat in my share of meetings thinking about strategies to avoid those meetings ever again. To learn from an incident, it requires being open to discovery and exploration rather than a strict checklist.

Every incident is a special snowflake; even when it looks like the same problem, maybe it's a different set of compute, storage, or networking, software, configuration or people. Depending on the maturity of your organization, the set of tools you have, and the people in the mix, there may be limited time to analyze most incidents. So, how do you figure out what to investigate and to

what degree? Really, it depends on what your team wants or needs to learn. Interesting incidents might include the complex incidents that involve multiple teams or large impacts, incidents due to new systems or features, or events where an incident was actually avoided.

THE DANGER OF COGNITIVE BIASES

There are a number of cognitive biases that can hinder you from identifying systemic causes of incidents including:

- *Anchoring* bias is relying on one piece of information or single source when making a decision, rather than considering things holistically. Checklists reduce anchoring bias by helping you make sure that important details haven't been overlooked.¹ Following a checklist isn't a sign of incompetence, it's an admission that even professionals make mistakes and seek to minimize them.
- *Availability* bias occurs when you are influenced by memorable or easily accessible events. One way to minimize this bias is to maintain a searchable library of previous incidents so that you can compare the current incident to ones that your team has handled in the past.
- *Confirmation* bias is relying on data that agrees with your pre-existing opinions and beliefs while filtering out evidence that doesn't. To counter this bias, look for and include countering evidence and include diverse perspectives and points of view.
- *Hindsight* bias is assuming that it was possible to predict that a particular event would occur.
- *Status quo* bias is a preference for things staying the same leading to a resistance to change.

To counter biases:

1. Obtain the facts first without assuming an immediate cause.
2. Flag potential causes
3. Look for and evaluate contradictory evidence.

4. Revisit the data

Aiding Discovery

The should've and could've can derail learning about what was done. This doesn't mean don't acknowledge mistakes. Mistakes need to be talked about and understood. Without psychological safety on a team, it can be really hard to admit to being wrong or having made a mistake. Sometimes individuals might feel that this was something they did wrong, when upon discussion it might be something problematic about the systems or misunderstanding about the impact of specific normally recommended actions.

Depending on your role in the discovery and investigation process, especially if you weren't part of the incident response team, questions to ask include:

- How did you get notified of the event?
- Has this type of incident occurred before?
- If the incident has occurred before, what was the past impact?
- What surprised you in the incident?
- Could this incident occur again?

You may discover varying perspectives of the system and what went wrong as well as hidden differences in how people make decisions about managing the systems.

Effective Documentation of Incident

Incident reports are artifacts for the team to help to spread knowledge and prevent stagnation where the team as a whole doesn't know specific knowledge obtained via the incident. Team artifacts should be stored in a central place. Depending on the organization, transparency of these artifacts may be useful to other teams.

Each artifact may have slightly different content based on the nature of the incident. The intended audience of the team incident report is the individuals on

the team, so these reports can be longer and more detailed than external or executive briefings.

An example template for a team incident report:

- Title
- Date
- Author(s)
- Summary of the incident
- Incident participants and their role(s)
- Impact
- Timeline
- Include graphs and logs that help support the facts described in the timeline.
- Lessons learned about what went well, and what needs improvement.
- Action Items - These should include who, what, type of action, and when. Others outside of the incident response team might think of additional action items after reviewing the narrative.

Everyone involved in the incident response should review the record of the incident and add information that might be missing including areas where they might have been confused or uncertain about next steps.

Team incident reports aren't the only artifacts of interest from incidents. In my experience, when the focus is creating a single artifact, it can feel like a way to direct blame, and leads to ingrained fear that hinders collaboration on learning what happened. A lot of data is generated, many graphs are examined, and many people may have been involved in getting the service back into a healthy state. Sift through all this information and compose the necessary artifacts; this may be an executive report for a CEO, customer communication, in addition to the team incident report.

Distributing the Information

After documenting the incident, you share back what you learned from the incident with the organization. This might be through email, updating a website, or presenting the information in a meeting. The post-incident meeting is a critical part of continuous learning in an organization.

Everyone heading to this meeting should have shared objectives to help align efforts. A post-incident meeting without shared objectives is often worse than no meeting at all. If there are misaligned incentives or individuals are not getting recognized for the value they bring to the process, this can lead to heroics or dismissal of the whole process.

Objectives shouldn't reflect an idealistic "perfect" world. For example, there is no way to prevent all incidents from ever occurring so having an objective to eliminate incidents isn't reasonable or attainable.

Instead, aim not to repeat the same incident in the same way. Other helpful objectives might include identifying specific areas where information about why something occurred isn't understood clearly and where single individuals knew specific information that wasn't known to the entire team. In other words, the outcome of this meeting should increase knowledge and identify areas of focus. Some documentation may need to get updated after information has been distributed to the larger group.

Next Steps

Often incident management success metrics are focused on improvements to mean time between failures (MTBF), mean time to failure (MTTF), mean time to detection (MTTD), and mean time to recovery (MTTR). These metrics were useful when reading hardware specifications to schedule optimum proactive replacements to avoid outages. These metrics are much less valuable when it comes to modern cloud-centric systems, because their focus on predicting hardware failure trends no longer apply, now that the focus has shifted from physical servers to virtualized compute. Additionally, averaging response times for different times of failures isn't providing useful and actionable information. Better success metrics can be uncovered through continuous collaborative learning from incident reports.

Success for the incident management process could include:

- Less people in incident response (folks feeling more confident in the process)
- More people attending incident reviews (folks feeling like the use of their time is valuable)
- More time allocated for event investigation

Wrapping Up

Incidents are exceptions to a production system that have an impact on the users of that system. It's unrealistic to think you can eliminate all incidents. Instead, focus on improving your incident response with deliberate and measured change. Consider how well your team responds to and learns from incidents.

You and your team can prepare for incidents by establishing processes for communication, training, and documentation. When an incident happens, communicate clearly to internal and external stakeholders, customers and the team, pull in the necessary subject matter experts, and learn from the outages to improve the systems.

Incident resolution needs to include shared learning from the incident; to identify where things went wrong and to consider changes to take to reduce future risks where patterns can be detected around events that affect your systems or across incidents that occur.

MORE RESOURCES

- Vanessa Huerta Granda, [“Making Sense out of Incident Metrics,”](#) Learning from Incidents, May 28, 2021
- John Allspaw, [Moving Past Shallow Incident Data](#), March 23, 2018
- Richard Cook, [“How Complex Systems Fail”](#)
- Lorin Hochstein, [GitHub Repository of Resilience Engineering Papers](#)

-
- 1 As an example, even highly trained surgeons benefit from checklists: (Surgical Safety Checklists) have proven to be an effective way to improve medical outcomes for surgery patients.

Chapter 13. Capacity Management

A NOTE FOR EARLY RELEASE READERS

With Early Release ebooks, you get books in their earliest form—the author’s raw and unedited content as they write—so you can take advantage of these technologies long before the official release of these titles.

This will be the 20th chapter of the final book. Please note that the GitHub repo will be made active later on.

If you have comments about how we might improve the content and/or examples in this book, or if you notice missing material within this chapter, please reach out to the editor at vwilson@oreilly.com.

Capacity management is the process of maximizing system output based on customer demand and produced business value while minimizing the costs to the humans supporting the systems.

Historically, sysadmins focused on tuning system utilization to maintain good latency for real-time access systems or to reduce job runtime on batch systems. In contemporary environments, sysadmins may focus on scaling resource pools in self-maintained data centers, applications in cloud services, or both for hybrid environments.

In this chapter, I define capacity and capacity management, and provide a framework by which to understand your capacity management planning process. This will help you prioritize the different engineering tasks involved in capacity management.

What Is Capacity?

Before I talk about capacity management, I need to talk about capacity. Capacity

goes beyond just the absolute value of CPU, disk or memory. Defining capacity also includes the measurable quantity of output producible while maintaining standards of quality and performance.

Capacity is not an exact measurement in systems, but rather an approximation based on the information that you have. Over time, accumulated experience with how your customers use your system will allow you to fine-tune the capacity indicators you use to understand your system's capacity.

There are different measurements of capacity, depending on the specific metrics that matter to the system you are supporting. And there are a few ways to define capacity when describing your systems. For instance:

- **Design capacity** When designing or evaluating the architecture of a system, you estimate the potential maximum output based on whatever tools you may have or previous experience. This estimation is the design capacity of that system. For example, you may benchmark a website and identify as a result that it supports 1000 concurrent user logins.
- **Production capacity** When your system is faced with actual normal working conditions, you will be able to measure the real maximum output possible (including all of the operating constraints) and this is your production capacity. When a system is live you have the data to drive observations based on the site's usage under normal working conditions to better qualify the capacity of the system as built. For example, users start experiencing impactful latency in your deployed system in production before hitting 1000 concurrent user logins. The site's production capacity is 800 concurrent user logins.
- **Effective capacity** When your system is under normal working conditions, and real world constraints are added (impacts due to seasonal or economic events), maximum output is your effective capacity. For example, during an after Christmas sales event, you notice that there are a number of cascading degradations in the system leading to an effective capacity of 300 concurrent user logins.

When describing capacity, be specific about which of these - design, production, or effective capacity - you are measuring or talking about. The capacity

constraints are the resources that limit the output of the system and can help you think through likely failure scenarios. These are sometimes called the bottlenecks and are generally where the system will fail first. Capacity constraints in your system might be limitations due to an underlying service dependency, specific hardware resource, or available individuals to do work. Based on the risk of the event, you can plan whether the constraint is acceptable or needs mitigation.

The Capacity Management Model

Capacity management is one area of engineering that sysadmins have the opportunity to focus on when not overwhelmed with toil work. Some parts of capacity work are day-to-day and other parts are medium to long term design and planning projects.

Operational cost reduction is not the goal of capacity management, though it may be an outcome of applying quality capacity management practices. The goal of capacity management is to balance resource costs and customer demands through:

- Gathering knowledge over time to guide growth and declines,
- Qualifying availability of people and resources to support new projects and changes in current projects, and
- Identifying periodic cycles from holidays, special events, site specific tax season in the US, elections.

With capacity management, it's crucial to understand the business value of the system you are managing. Failing to practice capacity management leads to missed deadlines, lost opportunities, and customer attrition.

The four components to capacity management are:

- resource procurement
- resource justification
- resource management

- resource monitoring



Figure 13-1. Capacity Management Model

Let's look at these different components in more detail starting with procurement.

Resource Procurement

Procurement processes vary based on the behaviors and structures of differently sized companies. Small companies may pay more for equipment or resources because of the size of their order, but may have fewer gating factors to approval, while larger companies may be oriented to have multiple groups involved, and approvals needed, before a requisition may begin in earnest.

When planning for the datacenter, include overhead and longterm hardware costs, and the supply-chain. In the cloud, you have increased reliability but with the possibility of unconstrained complex costs.

The complexity for setup within a data center versus cloud native varies widely. For example, compare the long lag times for hardware delivery and setup in the data center versus the near-immediate delivery from a cloud provider.

Regardless of your environment, ask yourself these guiding questions:

- How much performance and availability do you need? is it variable?
- Will the cost for static instances or servers be more than the cost of auto-

scaling options month over month? Year over year?

- Should capacity be built to handle the spikes in activity or regular load?

Justification

Understanding the procurement process that you need to navigate helps inform your justification process. If you have long delivery times, you may need to do the appropriate work to justify resource purchases well before you need them. If resources can be made readily available at a moment's notice you can delay justification until you are ready to do the necessary associated work around deploying the resources. As with procurement, the processes required within an organization to justify resource procurement vary in implementation - from something ad-hoc, to very formal with a review board assessing the strength of the proposal.

Even if the environment doesn't require a formal review process, it's important to have this information on-hand to better understand the decisions made including what was considered and ultimately discarded. Circumstances change, and maybe a formerly inappropriate solution becomes a better fit for a future project, or a new direction for the current project. And it can also be worth having a record of why a particular solution was discarded — maybe there's a fundamental flaw that others should steer clear of as well.

Items to document include:

- Describe your problem assuming no prior understanding of the circumstances.
- Describe the potential solutions.
- Explain your choice of solution. For example, "With this resource, I expect this amount of improvement in a specific measured value with an estimated increase in revenue or business value."
- Provide supporting data for answering why and how much?
- Address any other potential constraints and risks to success.

Management

Resource management covers the full lifecycle of resources from deploying to deprovisioning and varies based on the type of resources and how much automation is in use.

A resource's lifecycle helps you to plot a set of actions in alignment to business objectives.

With managed physical infrastructure hardware, you plan provisioning, configuration, deployment, and eventual retirement; you have to think about these concerns before even purchasing the hardware.

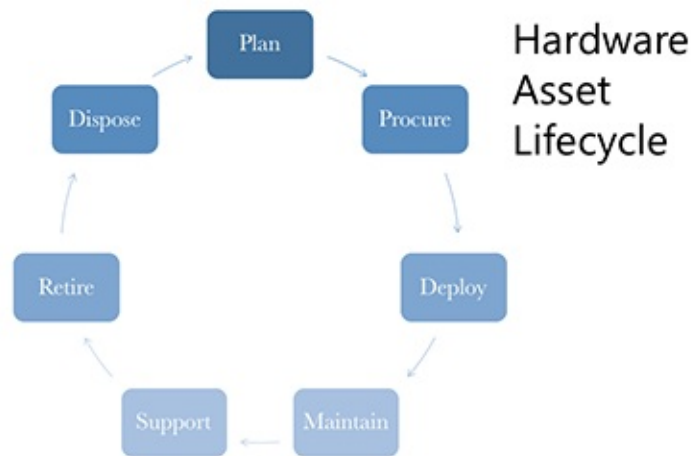


Figure 13-2. Physical Infrastructure Hardware Asset Lifecycle

The phases of the hardware asset lifecycle include:

1. Plan

You plan hardware purchases taking into account space, cooling, and power needs in addition to your currently owned hardware.

2. Procure

After identifying the hardware, you determine whether you are buying or leasing it based on availability of the hardware and vendor pricing aligned to your plan. Building strong relationships with vendors in servers, storage, and networking helps you get the best prices and necessary support for your hardware.

3. Deploy

Once equipment arrives, you need to verify the systems arrive as specced. A different team may be responsible for the physical deployment into the racks, or it may be part of your job responsibilities.

You install the required operating system and necessary updates. You may perform some amount of burn-in to verify that the system behaves as expected and that there are no component performance differences. Generally hardware follows a “bathtub curve”, where defective components show failures early in their lifecycle and can be identified through exercising the component.

Finally, you install and deploy necessary software and services to make the system live.

4. Maintain

You update the operating system and upgrade any hardware as necessary to support the required services.

5. Support

You monitor the hardware for issues and repair based on any expectations of services. This may mean coordinating support or physically swapping in new hardware as necessary.

6. Retire

You identify when the hardware is no longer needed and de-provision running systems. This may be a long process to identify any access to the system.

Sometimes, new hardware is being brought into service to replace older hardware. If the system architecture can be scaled up gracefully by adding new hardware and then scaled down by removing the older hardware, this allows for easier retirement and deployment processes with minimal impact to the end customer. If you have to completely shut down a system to remove hardware, this will cause some amount of end user impact.

NOTE

Consider the “bathtub curve”, the curve of a bathtub with steep sides and a flat bottom. This

model depicts observed hardware failures over time. Physical resources have three phases with higher failures at the beginning during the burn-in phase where defective products are discovered and at the end during wear-out.

Even if a system is still doing useful work, like an older car, it's necessary to evaluate whether the ongoing cost of repairs is more cost effective than replacing it and avoiding failure completely.

1. Dispose

Once you have retired software from the system and removed it from service (and if it is no longer useful within your organization in any other capacity), you have to dispose of the hardware. In addition to ensuring that no sensitive data remains on the system, you may need to be aware of specific laws and regulations around disposal.

When planning hardware requirements, it's common to think about a 3-5 year lifespan for non-specialized hardware. In part, this is due to advancements in the physical technology that improves the cost of running servers. It is also due to advancements in the system software, where older hardware might not support current operating systems.

NOTE

It's not uncommon for IT departments to be structured organizationally within finance leading to accounting depreciation schedules that trickle down into IT policy.

Organizations may use a different strategy for depreciation, and there may also be specific legal/tax guidelines to follow, but 3-5 year schedules dovetail with how the expense of expensive equipment is amortized over multiple years.

With specialized hardware like storage appliances, the lifecycle changes slightly in that the costs can range from the 10s of thousands to close to a million dollars. On top of that, maintenance and support are separate costs and longer-term investments.

There are challenges to implementing quality hardware management in your infrastructure strategy with staffing, tool availability, and the complexity of hybrid environments.

Operation engineering teams are often understaffed which can lead to not having enough time to spend on developing quality practices on managing hardware effectively. This could mean hardware arriving and delayed deployment or lack of retiring aging systems in a timely manner.

Another challenge is the lack of investment or availability in quality tools. Often spreadsheets are used to design datacenters (including cooling and power), manage vendor relationships and inventory (from the physical hardware itself to the cabling organization). This can hinder collaboration, communication, and knowledge transfer throughout the organization.

A hybrid environment where part of the infrastructure is on-premise and part is managed by a cloud provider adds additional complexity. This might be acceptable if there is not in-house knowledge for managing necessary services.

In the cloud, organizations still need to consider the lifecycle of assets. Physical racking and stacking and the physical security of the hardware are handled by the service provider. You also eliminate the need to maintain and dispose of physical systems, but every other phase is still present in some form.

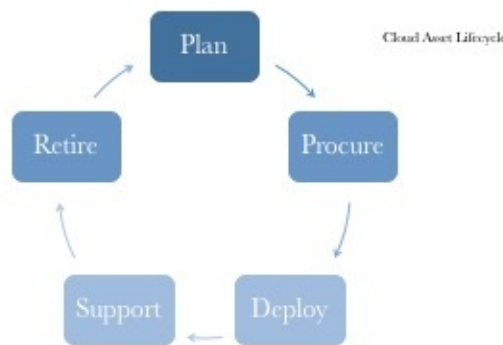


Figure 13-3. Cloud Asset Lifecycle

Plan

You focus on identifying specific cloud services to use (for example, specific machine types or reserving capacity versus on-demand) and budget forecasting.

Procure

Instead of having to plan for expenditures all at once, you set budgets per individual or team to align spending and leverage purchasing power across the organization. You build relationships with different cloud providers, and identify compatible services that align with business requirements.

Deploy

Instead of physically deploying servers, you write infrastructure code to provision, verify, and deploy necessary cloud resources programmatically.

Support

Through careful monitoring of systems in use, you identify areas for cost savings.

You assess, monitor, and repair security vulnerabilities in the software and underlying layers depending on the service in use.

You also may be the central contact with the service provider to coordinate support.

Retire

Rather than worrying about physical hosts for 3-5 years and maximizing their value, you can configure instances to only live as long as needed, eliminating cloud resources that are running and providing no value-add work. You can configure policies to shut down and de-provision resources that are no longer in use.

Migration to the cloud may ease some of the stress on operation engineering teams allowing more time to focus on the different practices involved in managing infrastructure. However, with the ease of quickly provisioning resources, visualization of resources in use is critical to prevent costly mistakes.

Serverless is a special type of cloud native compute, storage and networking. With serverless, the lifecycle of assets is simplified as many of the steps are handled by the provider. Sysadmins still have to plan what architecture and services are necessary to provide the customer's expected experience.

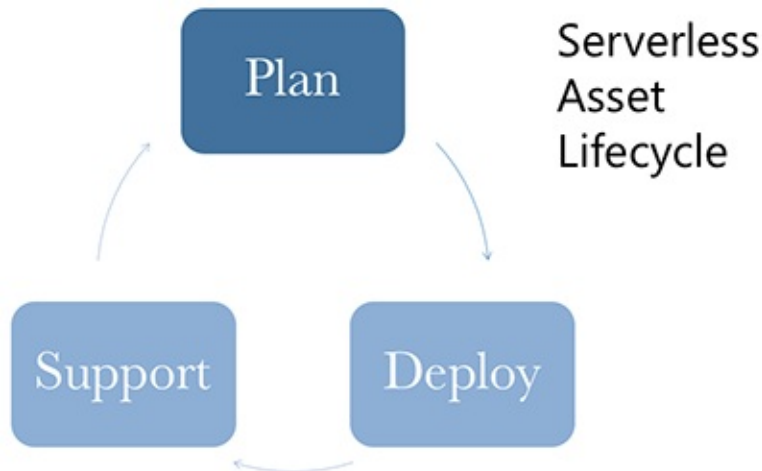


Figure 13-4. Serverless Asset Lifecycle

While the deployment of the underlying infrastructure servers is handled, individuals need to deploy the essential configurations and applications that will define what gets deployed when used. Finally, individuals need to instrument applications and configure monitoring and observability tools to ensure that you can see when there is a problem with the application and debug it.

Monitoring

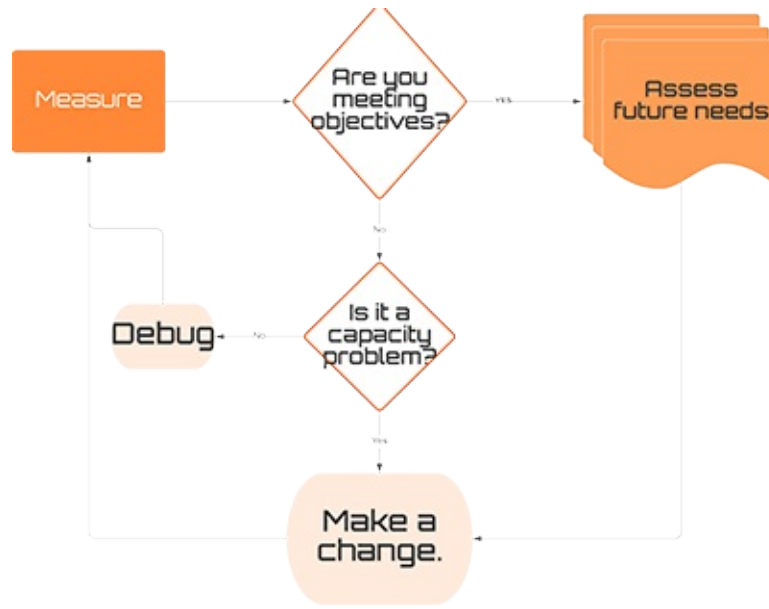
Resource monitoring is the monitoring of the specific resources in use with the goal of balancing resources costs, customer demand and business value.

This area of capacity management is covered in detail within the Monitoring chapters of this book.

The Framework for Capacity Planning

You should consider documenting capacity management components per environment, as underlying processes to follow will vary from team to team and across organizations. While I can't define what they will look like for you, I can provide a framework to guide you on what you can do next, once you understand the processes and policies in place within your environment.

Let's look at the capacity planning process framework:



1. Measure the current workload for all the components of the object in your system under evaluation.
2. Evaluate whether you are meeting the service level objectives based on demand.
3. If you are meeting objectives, spend time on assessing your future needs. (e.g. is there new compute technology that may replace current requirements?)
4. If you are not meeting the objectives, assess whether this is a capacity problem. Sometimes you need to resolve other issues before making changes to capacity. There may be optimizations possible in configuration tuning that will lead to performance improvements.
5. If it is a capacity problem, identify changes that can be made and apply one of the changes to see the impact of making that change. Make sure that information gleaned about the change is understood by the relevant team or teams to help guide future decisions. If you don't have enough information because you don't have the right measurements, make changes to your measurements.

How organizational strategies influence planning

There are three main strategies which organizations leverage to assess future

needs: lead, lag, and match. Any of these three strategies can help inform and prioritize action and reduce friction.

With the lead strategy, you add capacity as you receive indicators that system demands will increase over a given time frame. Often employed with on-premises resource management, this strategy compensates for not being able to make fast changes in the event that demand is higher than capacity since ordering and delivery of hardware can be highly variable. Overhead costs increase if demand doesn't materialize after capacity is increased.

With the lag strategy, you meet demand after it occurs. If you cannot fulfill demand in a timely manner, the lag strategy can increase the chance of losing customers and impact trust or confidence in the company.

The lag strategy isn't realistic for on-premises resource management for small companies, due to the long time it takes to order and receive hardware. In large companies, resource allocation to individual teams can be made from other teams within the organization. Resource-driven conflict occurs when popular projects that didn't do adequate capacity management "steal" resources, leading some teams to greatly exaggerate the estimations on resources that they need to accommodate for losing some portion of expected resources, which can reduce financial investment for other projects.

The match strategy attempts to compromise between the lead and lag strategy by incrementally increasing capacity with demand. For example, capacity may be preemptively expanded by a fraction of the forecasted future need, with the remainder waiting until the need actually appears.

JUST IN TIME (JIT) CAPACITY PLANNING

Another term for the lag strategy is "just in time". With the increased attention to Cloud Native architectures and Agile, JIT has been an increasingly popular approach to resource allocation.

With Just-in-Time Manufacturing rather than maintaining inventories of components ready to be assembled, parts are acquired as they are needed based on production demand. This reduces costs and minimizes unwanted surpluses, both of which increase profitability. But this efficiency relies on accurate predictions of future demand; incorrect predictions will disrupt the

pipeline.

Consider the economic effects of the COVID-19 pandemic on global supply chains. Shortages of products like toilet paper arose not because people started using more toilet paper; people didn't need commercial single ply toilet paper at the jobs and schools they were no longer going to, but they needed more household toilet paper which is produced differently than commercial grade toilet paper. It took time for paper manufacturers to retool their factories to shift production from commercial to domestic distribution, and in the meantime, retail shelves were empty even as warehouses of commercial toilet paper piled up.

As you evaluate your capacity planning needs, think about the variables that inform your predictions, and consider contingency plans for how you can respond to unexpected shifts in demand.

Do you need Capacity Planning with Cloud Computing?

Even when using cloud services, sysadmins need to develop an explicit capacity management strategy. While some cloud services offer dynamic scaling, it doesn't eliminate the need to conduct capacity management. Even if you only focus on resource management and monitoring, dynamic scaling has limitations which include:

- Time to spin up new resources.
- Resource ceilings set by the provider based on instance types chosen. CPU, network, and storage throughput are limited to what you choose in the initial configuration. While in some cases these can be changed, downtime may be required, depending on the cloud provider. Some limits require contacting the cloud provider to adjust, which can have varying times to resolve. Contrary to the idea that everything is API based and instantaneous in the cloud, service providers institute certain limits to better serve the average use case.
- Managed datastore configuration limitations. Cloud providers create tiered

offerings that simplify some of the management challenges of sizing databases, but you may need flexibility you hadn't anticipated. The more expensive the offering, the more of the fine-tuning with resource management they generally cover. You still have to select the specific functionality whether sharding, replicating, or load balancing and these choices can be very expensive. Right-sizing your resources follows the flow of the capacity planning process.

- Capacity limitations of the cloud provider itself. At certain levels of scale, the assumption that more resources can be added on-demand breaks down due to the real limits of how much hardware the provider has available.
- External dependencies may have additional limits or lack dynamic scaling functionality. Examples include gateways and proxies.

Cloud computing makes it easier to adjust dynamically to real demand. Engineering requirements can be more finely tuned to better approximate the variable nature of demand and inform staff of the impact of making changes to core infrastructure.

Service providers set varying limits for services. While the service provider handles scaling, individuals still have to be aware of the impact of dependent services and the limits across all of these services as well. Without oversight, it could be quite easy to run afoul of these limits; e.g., **75GB max limit on function and layer storage in AWS**

Wrapping Up

The future is unpredictable; deploying new resources can take time, but over provisioning costs money. Capacity planning is the art and science of matching your resources to anticipated future needs aligned with the demands of your organization, without constraining your system's potential or spending too much.

When considering the capacity of your systems:

1. Identify the need and justify how a particular resource will meet that need
2. Procure the resource, along with any overhead expenditure to maintain the resource

3. Monitor the resource
4. Manage the resource throughout its lifecycle

Capacity planning is important for all resources you oversee, including both physical and cloud-hosted systems, but the procurement characteristics are distinct. With hardware systems, it takes time to acquire and deploy new equipment, and it's generally harder to scale up or down as demand evolves. With systems built with cloud native services, scaling can be automated, but it's also easy to overspend if you aren't keeping an eye on things. Effective capacity planning requires ongoing assessment and adjustments to your processes.

Check out these additional resources on Capacity Management:

- Learn more about capacity planning for web sites from the [The Art of Capacity Planning: Scaling Web Resources in the Cloud](#) book by Arun Kejariwal and John Allspaw.
- Case study of capacity management from Capital One from Kevin McLaughlin at Velocity 2016 New York [“Is capacity management still needed in the public cloud?”](#)

About the Author

Jennifer Davis is an experienced engineering manager, operations engineer, international speaker, and author. Her books include *Modern System Administration*, *Effective DevOps*, and *Collaborating in DevOps Culture*. Jennifer has worked with a variety of companies, from startups to large enterprises, improving operability practices and encouraging sustainable work.

About the Contributors

Chris Devers has spent the last twenty years helping people get the most out of computers, so that they can spend their time on more important things, helping development teams focus their efforts on delivering software that solves real problems for real people. He lives in Somerville, Massachusetts with his wife, sons, and cat, and would usually rather be taking photos and bike rides.