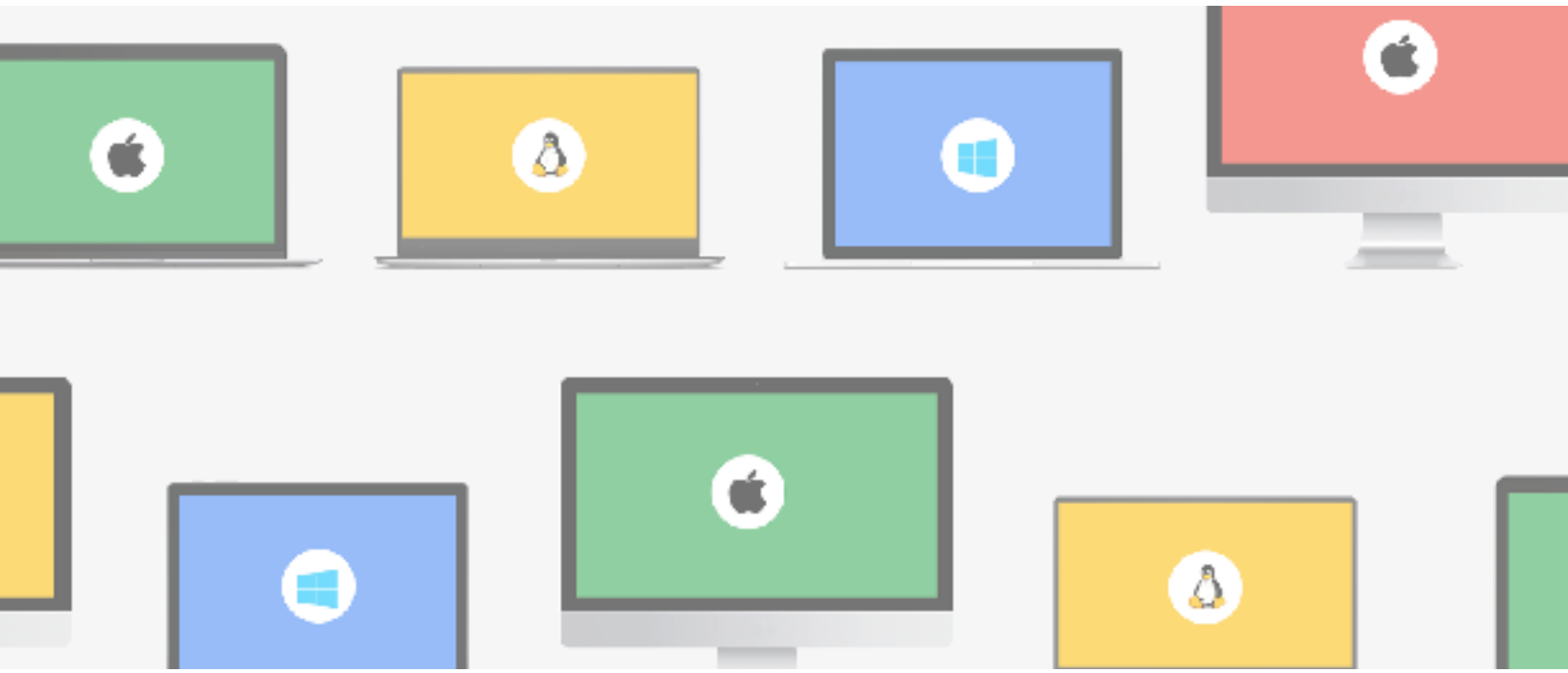
Google



Written by
Sam Mackness
Sarah Lucas

Published
November 2017

Fleet management at scale

How Google manages a quarter million computers securely and efficiently

Google

Introduction

Google's employees are spread across the globe, and with job functions ranging from software engineers to financial analysts, they require a broad spectrum of technology to get their jobs done. As a result, we manage a fleet of nearly a quarter-million computers (workstations and laptops) across four operating systems (macOS, Windows, Linux, and Chrome OS).

Our colleagues often ask how we're able to manage such a diverse fleet. Do we have access to unlimited resources? Impose draconian security policies on users? Shift the maintenance burden to our support staff?

The truth is that the bigger we get, the more we look for ways to increase efficiency without sacrificing security or user productivity. We scale our engineering teams by relying on reviewable, repeatable, and automated backend processes and minimizing GUI-based configuration tools. Using and developing open-source software saves money and provides us with a level of flexibility that's often missing from proprietary software and closed systems. And we strike a careful balance between user uptime and security by giving users freedom to get their work done while preventing them from doing harm, like installing malware or exposing Google data.

This paper describes some of the tools and systems that we use to image, manage, and secure our varied inventory of workstations and laptops[1]. Some tools were built by third parties—sometimes with our own modifications to make them work for us. We also created several tools to meet our own enterprise needs, often open sourcing them later for wider use. By sharing this information, we hope to help others navigate some of the challenges we've faced—and ultimately overcame—throughout our enterprise fleet management journey.

> "Everyone in Site Reliability Engineering's goal is to automate themselves out of a job. Don't worry—there will be a new job for you; something that isn't yet automated. Human beings do not exist to push buttons and turn cranks on things which should be automated."
>
> — Thomas Bushnell, Linux SRE at Google

[1] Since Chrome OS requires very little enterprise management, we don't cover it here. We also don't discuss mobile devices as the management systems and challenges are different and may be addressed in a future paper.

Google

Imaging at scale

The first stop a device makes when it enters the Google ecosystem is imaging. With nearly a quarter-million computers to image, we've had to find ways to reduce the complexity of our imaging process and cut down on the time it takes to image a machine.

No matter the platform, we always start with a basic vanilla image and package it with our configuration management tools. It's easier and faster to change network-based files than it is to regenerate a new image whenever we update a configuration tool.

We use Standalone Puppet[2]—which doesn't require connecting to Puppet configuration servers on the web—to apply configurations across our entire macOS, Windows, and Linux landscape. Our workflow entails declaring the desired machine state; Puppet then consistently runs checks to ensure that the computer is in the desired state. When a machine fails this check, Puppet returns the machine to the declared state. For example, if you declare that machines should have a 5-minute screen timeout and an employee disables their screen-lock, Puppet will enable the screen timeout the next time it runs.

> "Our package management and configuration management tools [allow us] to customize a single monolithic image for all of the Macs in the Google inventory."
>
> — Edward Eigerman, Mac SRE at Google

Master Puppet vs. Standalone Puppet

We've switched from standard Master Puppet mode to Standalone (Masterless) Puppet mode at Google for two main reasons:

- Standalone doesn't require a large infrastructure of Puppet configuration servers. Our hosts pull the cryptographically verified configuration files from a web host which serves the files, verifies the data locally, and then applies the configurations.
- Not having servers allows us to commit to our BeyondCorp access model, which does away with using internal networks for corp access.

Read more about our BeyondCorp effort at https://cloud.google.com/beyondcorp

[2] https://puppet.com

Google

Our approach to packaging our configuration tools with the image and distributing this image to computers varies by operating system.

On Mac, we use AutoDMG[3] to combine the base image from Apple with our configuration tools and then upload it onto our internal distributed file system (DFS). We created an app that pulls the image from our DFS and writes the image to machines attached in a target-disk mode. Our imaging time is down to 15 minutes, compared to the hour that it used to take when we used TFTP servers and PXE boot.

On Windows, we use Glazier[4]—a code-based imaging tool that we created in house and then open sourced. Glazier is made up of binaries that are configured through source-controlled and peer-reviewed text. Text files suit our typical use cases better than GUIs because they work with version control systems. Admins can see a complete revision history of the imaging environment, peer-review changes, and roll back the image if problems arise.

The image files are then distributed over HTTP(S). We chose this method because it's open and ubiquitous, has many freely available server implementations, can distribute data globally, and is highly secure (in the case of HTTPS).

On Linux, we use PXE to netboot a standard Ubuntu/Debian installer image. We have a system that automatically builds new OS install images on a schedule (in the form of compressed tar-format archives). These install images are then placed on an HTTPS server alongside Debian preseed files that automate the host setup portion of the installation.

Our installation process is integrated with our Puppet and host update infrastructure to ensure every host is configured as intended at install. This allows us to reinstall any host from the network in about 30 minutes without needing to distribute media or requiring another host to boot from.

We have a team dedicated to tracking the latest in consumer enterprise hardware, working with outside vendors and partners, as well as the internal Chrome OS hardware group. They monitor industry trends, attend advisory meetings with vendors, and run their own tests and focus groups with Googlers to ensure that our hardware offerings continue to meet everyone's needs.

[3] https://github.com/MagerValp/AutoDMG
[4] https://github.com/google/glazier

As a result of our retooling, our imaging processes are easy and fast enough for Googlers to reimage their own machines if they need to.

Getting software on computers

Since we aim to keep the image we install on new machines simple, we only preload mandatory management software onto machines. If a user needs specific software to do their job, we make this available to them through central software repositories. We use a combination of third-party and custom tools developed in-house to package and push software to these repositories in ways that are automatic and easily repeatable.

In 2010 we evaluated several commercially available software packaging and management solutions for macOS, but none of them fit our needs. Munki[5], a great open source software (OSS) tool, also fell short of our requirements because its only purpose is to fetch a manifest and catalog file from a simple web server. We needed the ability to dynamically generate these catalog and manifest files on a per-host basis, so we created and open sourced a solution called Simian[6]. Simian is a Google App Engine-hosted server, with a client powered by Munki.

We use Luggage[7] to create the Apple package installers and Munki to get the packages on Googlers' machines and push updates. Simian then works with Munki to deploy or update software to targeted users, hostnames, OS versions, groups, and more. Simian also lets us force-install updates on machines when necessary.

On Windows, we currently use Microsoft System Center Configuration Manager (SCCM). While SCCM has many features beyond packaging, it's not the best solution for us. Software needs are as diverse as our workers, so we need a tool that allows us to create reviewable packages in our codebase and push directly to our software repository. We've developed an internal tool on Linux called Rapture that does just this, and are working on switching

We provide end users with a catalog-style shopping portal where they can order licensed software. Once the request is approved, most software packages are automatically "pushed" to a user's machine and can be installed without tech support intervention.

[5] https://github.com/munki/munki
[6] https://github.com/google/simian
[7] https://github.com/unixorn/luggage

from SCCM to Rapture on Windows to drive more consistency between platforms and the infrastructure we use.

With Rapture, we can create union software repositories to group multiple repositories owned by different teams into one larger meta-repository. Using this system, we can publish one small set of repositories to all clients, that make use of server-side features like canaries and version controlling, without having to manage a complicated set of repositories on the client side.

Rapture also handles significant request load. Our hosts check in with Rapture every 15 minutes for new software updates. When things get busy, like during a new software release, Rapture regularly serves more than 75 gigabits per second of network traffic.

Of course, these backend processes are invisible to end users. We provide centralized software centers on all of our platforms where users can find the software they need and install it with just a few clicks. On Linux, since most of our users are much happier using CLI's, software can also be installed via APT. This self-service approach cuts down on the amount of time techs need to spend installing software on users' computers and makes it easy for users to quickly get the software they need, when they need it.

We often find that third-party tools don't fully suit our typical use cases. That's why we use open-source software whenever possible, or build our own tools and make them available for wider use.

Balancing usability with security

We try to give end users as much freedom as possible in managing their own machines and installing software. Granting end users this freedom, however, means that we need to take precautions to secure our fleet. Puppet is one tool we use towards this end. However, since Puppet isn't equipped to singlehandedly safeguard our fleet, we're taken steps to ensure that all of our devices are encrypted, have the latest OS version installed, and are free from malware.

Encrypting devices

The first step to securing our fleet was to fully encrypt all of our machines. To fit our needs, we used Apple's provided tools for key escrowing and created Cauliflower Vest[8] on App Engine. With Cauliflower Vest (an anagram for Filevault Escrow), we can forcibly enable encryption on users' machines and access recovery keys to unlock or revert volumes.

While we initially developed Cauliflower Vest for macOS, it also works with BitLocker recovery keys from Active Directory and LUKS on Linux. Users can retrieve their own recovery keys, so if they get locked out they don't have to wait for tech support to regain access.

> At Google, security and usability aren't necessarily mutually exclusive goals. We aim to design invisible and unobtrusive security solutions that users don't have to "work around".

Applying operating system updates

At Google, the state of your machine is a key factor in determining your level of access to internal systems. We use our Access Proxy and Access Control Engine[9] to enforce policies, like mandatory operating system upgrades, and restrict access to most corp resources until these policies are met.

To encourage users to install OS updates, we nag them with pop-up messages. The longer they wait to update, the more frequent the pop-ups become. If they wait too long to update, they will find their level of access degraded until these updates are applied.

If that isn't enough to get someone to upgrade, we built a tool that forces updates if too much time has passed since the last system update. The user receives pop-up notifications that their machine is about to reboot and upgrade so they're not force-updated without warning.

[8] https://github.com/google/cauliflowervest
[9] https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/45728.pdf

Google

Preventing malware with social whitelisting

While our software repositories allow Googlers to download the most popular software, we can't possibly review and package every piece of software employees need when they need it. We do allow users to download software from the internet, but only after it's gone through a social whitelisting process involving peer-based voting.

To this end, we use tools that provide local binary whitelisting systems at the kernel level: Santa[10] on macOS and Carbon Black[11] (formerly Bit9) on Windows. These tools run every time a binary launches, checking the hash of the binary and running it against local SQL database to see if the binary is allowed to run. If not, the tool blocks the binary from running.

When a user tries to install software that isn't whitelisted, they're served a pop-up that sends them to an internal website where they can request whitelisting approval. The tool notifies the user of any red flags with the software—for instance, if it looks like potential malware. If the software has no obvious problems, the user simply has to vote for the software in the tool and get another employee to vote for it. The software is then whitelisted and available for download by the user and anyone who voted for it.

Of course, there are some third-party tools that we prohibit due to potential security issues, so we maintain a list of banned software. Banned software can't be whitelisted with votes.

And if a piece of software becomes sufficiently popular, as measured by the percentage of installs across our fleet, we undertake a security review, and then package and deploy it to our software repositories.

After rolling out Santa (binary whitelisting for macOS) to our Mac fleet, we observed a 78% decrease in malware-related Mac reimage requests.

[10] https://github.com/google/santa
[11] https://www.carbonblack.com

Google

Applying a similar strategy at your company

Adopting a scaled enterprise fleet management approach did require some upfront investment and a culture shift toward automated, reviewable, and repeatable systems and processes. In return, we've benefited from lower maintenance and support costs, and increased job satisfaction for engineers, support staff, and our users.

Many of the tools mentioned in this paper are open source, making them affordable for companies of any size. Indeed.com is one good example of how a company much smaller than Google implemented a scalable strategy for securely managing their fleet of Macs. Before experimenting with Simian, their process for installing patches and updates was time-consuming and cumbersome. Their help desk had to manually apply updates using a 20+-task checklist and run various scripts manually on firstboot. It only took Indeed.com a few days to implement Simian with Munki, and their implementation was covered under the free App Engine usage tier.

The company further invested in this new strategy by using another tool to automatically upload the software package metadata and automate uploading/hosting the packages outside of the blobstore. According to Allister Banks, an IT Systems Administrator at Indeed.com, “Coworkers are very happy that they can self-serve to offer software to any customer, look at basic inventory items, and an item can be pushed globally in a super efficient manner.”

“Employees at indeed.com are very happy that they can self-serve to offer software to any customer, look at basic inventory items, and an item can be pushed globally in a super efficient manner.”

—Allister Banks, IT Systems Administrator at Indeed.com

When looking for ways to efficiently scale your fleet:

- Automate as many of the technical processes as possible.
- Give your users plenty of self-service options.
- Put automatic checks in place that prevent users from doing real harm.

Google

If you'd like to implement any of our open-source tools to manage your fleet, you can find a list of the tools with links to implementation instructions in the table below.

Google's open-source fleet management tools

Tool	Compatible with	Configure with	Setup and usage instructions
Glazier	Windows	N/A	https://github.com/google/glazier
Simian	macOS	https://github.com/munki/munki	https://github.com/google/simian
Cauliflower Vest	macOS, Windows, Linux	N/A	https://github.com/google/cauliflowervest
Santa	macOS	https://github.com/groob/moroz or https://github.com/zentralopensource/zentral/wiki	https://github.com/google/santa

About the authors

Sam Mackness, Engineering and Operations Manager

Sam leads the organization responsible for Google's corporate computing fleet. He is based in the Bay Area. Since joining Google in 2002, he has held roles in Hardware Operations, Global Production Infrastructure, and Corporate Engineering. Sam holds a BA in Political Science from the University of California, Irvine.

Sarah Lucas, Technical Writer

Sarah is a technical writer for Google's Corporate Engineering organization, based in NYC. Prior to joining Google in 2013, she was a freelance writer and content manager in the Metro Detroit area. Sarah holds degrees in English and Advertising from Michigan State University.

Contributors: Erin Pierce, Justin Hahn, Clay Caviness, Ofer Bar-Zakai, Matt LaPlante, Marga Manterola, Betsy Beyer, Kate Borger, Daniel Meltz, David Dorbin, Max Saltonstall