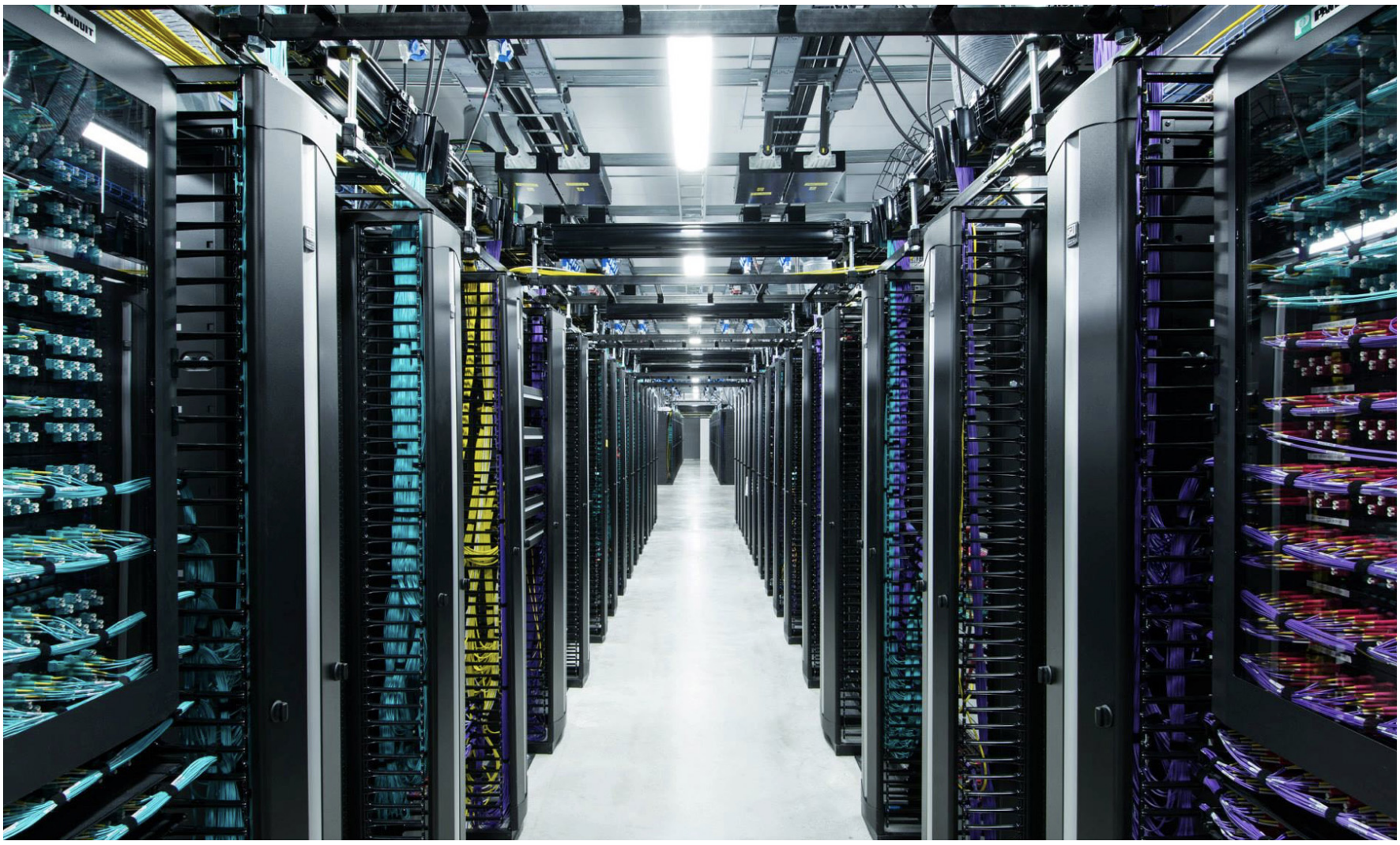
POSTED ON MAY 21, 2018 TO NETWORKING & TRAFFIC

Scaling the Facebook backbone through Zero Touch Provisioning



By Joe Hrbek Brandon Bennett David Swafford James Quinn

This new framework has already empowered our engineers to move faster, to solve problems more creatively, and to take a far more iterative approach to building our networks and our network deployment tools.

At Facebook, we have been growing our global wide-area backbone networks and edge network points-of-presence (POPs) for more than a decade. Today, these networks span multiple continents, both metro and long-haul optical networks, and two parallel IP backbone networks. All of these networks are rapidly expanding to support accelerating growth in both egress traffic (to internet) and even larger internal machine-to-machine traffic demands.

Our previous network provisioning systems proved inadequate for the scale and complexity of building these networks, so we created our own comprehensive, flexible workflow system with Zero Touch Provisioning. This new framework has already empowered our engineers to move faster, to solve problems more creatively, and to take a far more iterative approach to building our networks and our network deployment tools.

Challenges of rapid growth

Building backbone networks at Facebook's velocity and scale presents our network engineers with unprecedented and unique technical challenges. The fundamentally heterogeneous nature of geographically dispersed IP and optical backbone networks does not easily fit the conventional approaches of automating network builds often used in more homogeneous data center networks.

For many years, network engineers at Facebook approached these backbone growth challenges in a way not dissimilar from the rest of our industry. While we built tools to support the work of our network engineers, the overall approach was to write documentation (MOP, or method of procedure) and to hire more engineers to follow these increasingly detailed MOPs in our rapidly increasing deployment schedules. The tools, where they existed, were merely lines in a lengthy MOP for human engineers to execute manually.

Automation, but not enough. Then, a fresh approach with Vending Machine.

Facebook's rapidly growing backbone traffic demands strained our ability to keep up, and the manual approaches were no longer enough. Pushing to build network more quickly led to increased errors and escalating impacts to our services. We understood that we needed to shift to a more software-centric approach to building our backbone and automate the MOPs.

Initially, the backbone network team attempted to use an older Facebook system for interacting over console/serial connections to build data center servers and top-of-rack switches more than a decade ago. This console-based approach was state-of-the-art at the time, though provisioning over a console was not always fast or predictable. For a backbone network spread across the world, it was even more fragile.

This provisioning system had started small and focused, but, over time, many new network use cases, platforms, and roles were added, all within a single code path of if/else conditionals. At its apex, this single provisioning system was used to build nearly every Facebook network device, from top-of-rack switches all the way up to complex backbone MPLS and BGP peering routers. As code complexity grew, it became harder to troubleshoot, to fix when it broke, or to extend in solving new challenges.

Ultimately, these challenges drove Facebook's network engineers to develop a completely new approach for network deployment workflows. We called it Vending Machine, a name inspired by the machines that dispense candy and soft drinks. In the case of Facebook's Vending Machine, the input is a device role, location, and platform, and out pops a freshly provisioned network device, ready to deliver production traffic.

ZTP with Python agents

A particularly important challenge in this new Vending Machine system was finding an alternative to unreliable consoles for the automated device provisioning itself. As data center switch vendors began supporting Zero Touch Provisioning (ZTP) features using DHCP for their network platforms, we began working with our backbone IP router and optical equipment vendors to support similar features. This approach allowed us to reuse the DHCP auto-provisioning infrastructure we had built for our servers and to provision network equipment over far more reliable Ethernet management connections.

While a basic ZTP approach only applies configuration and upgrades software on a network device, we took a unique, more robust, and flexible approach: downloading a special piece of Python code in ZTP to be executed by the network device as our on-box agent. Once the ZTP process kicks off the Facebook Python agent on the network device, the agent performs several functions:

1. It downloads a package of instructions from our Vending Machine server that can be targeted for that specific platform, role, and location, and even for a specific device.
2. It follows these instructions to:
    1. execute any specified initial commands;
    2. download relevant files, including configuration, firmware, patches, or any other custom packages (such as Facebook's Open/R software); and
    3. perform further actions (loading configuration, upgrading code, rebooting, etc.).
3. It also sends logs back to Vending Machine servers through a REST communication channel to a HTTP server for status, logging, and debugging.

4. Finally, the agent reports completion of the build job back to Vending Machine and then terminates itself.

All these functions are executed on the network device in Python code written by Facebook's network engineers, which means we can readily extend and adapt this agent code, as well as handle any errors we find in our network vendor software or in our own workflows.

Vending Machine's next 'steps'

Loading a network device's software and configuration was important, but it only allowed us to replace a small part of our MOPs. Many manual steps remained to be automated. To obsolete our MOPs, Vending Machine needed to address every single line.

We built Vending Machine first and foremost as a flexible *workflow* framework where engineers could develop in parallel to iteratively automate each of the nonphysical *steps* involved in building a network device. We wrote Vending Machine to split out all the procedures into small, independent pieces of code (called *steps* in the system) that could be written in any programming language.

These steps are designed to be simple programs that focus on just one specific task. Using this simpler approach, an engineer could write and debug a step without any deeper knowledge of the overall Vending Machine system. This also meant that engineers could write and deploy steps independently, without redeploying the entire system, without any impact to other steps or workflows running in the system, and with minimal coordination between engineers writing and debugging different parts of the system.

Vending Machine queues steps to execute across a set of distributed servers. When each step runs, it:

1. Reads STDIN (standard input) encoded in JSON, including data identifying the device (hostname, IP address, etc.) and general job metadata.
2. Executes the work implemented in its step-specific code.
3. Outputs logs in a standard format to STDERR (standard error) to be stored with the parent job.
4. Returns a Unix exit code representing the success (0) or failure (any nonzero exit code) of the step. This behavior also catches any unhandled exceptions where a step fails.

When a step fails, Vending Machine will automatically queue the step to execute again, thus mitigating any transient failures during provisioning.

The simplicity of constructing steps as standalone binaries made it far easier for network engineers to develop and deploy new steps, and to iterate on their code while improving their specific network deployment workflows. It also made the overall Vending Machine system extremely flexible. It could be adapted to new use cases and features quickly, without the need for rewriting existing steps.

Our network engineers steadily developed more and more automated Vending Machine steps to replace what had been manual MOP steps in our network deployments:

- peering notifications
- updating inventory and management systems
- enabling operational monitoring
- waiting for a device to reboot
- performing address assignments

- generating configurations
- regenerating and pushing global iBGP and MPLS meshes
- performing a wide variety of link-level, device-level, and topology-level health checks and validations
- calling existing tool for draining/undraining of production traffic through the device

Vending Machine empowered our network engineers to link these automated *steps* together into steadily more comprehensive production network deployment *workflows* within the Vending Machine framework. These *workflows* define what order steps run in, as well as which steps should run in parallel with which other steps to speed the build process. Steps can be shared and reused across multiple workflows, or they can be isolated to a specific device role or platform.

These steps and workflows gradually replaced each and every line and section in our large MOPs, until the MOPs read just one line: "Run Vending Machine." Or, more literally: `vm configure <device-name>`. This same command can be used for every network device, role, and platform that we support in Facebook's networks. The complexity and variation of the network builds are hidden away, handled by the Vending Machine system and the workflows and steps constructed within it.

What's next for Vending Machine

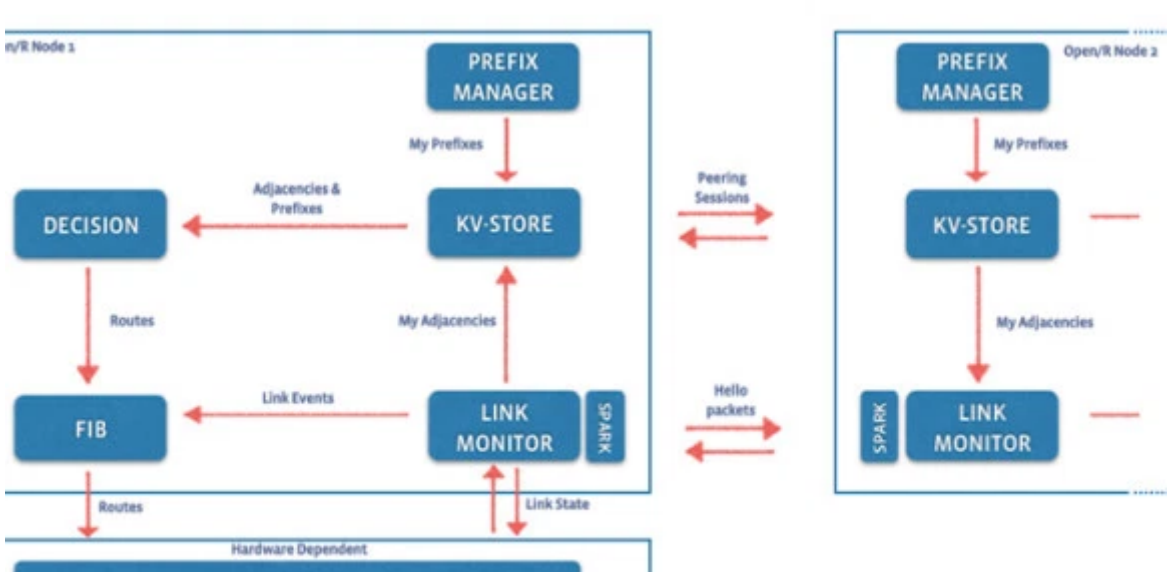
Vending Machine has given Facebook much greater confidence in our backbone network deployments and the reliability of the automation behind those deployments. This confidence is already making it possible for our engineers to begin pursuing even more ambitious goals:

- Orchestrating groups of Vending Machine device jobs to build or rebuild larger network topologies.
- Fully automating rebuilds of entire global planes in our Express Backbone network.
- Automating the clean, seamless upgrade of software revisions across our global backbone fleet.
- Implementing automation for the continuous rebuild of all of our networks in a healthy, structured way.

We'd like to thank the following engineers and acknowledge their contributions to Vending Machine: Ari Adair, Mayuresh Gaitonde, Chris Gorham, Zaid Hammoudi, Francisco Hidalgo, Robert Horrigan, Arsalan Khan, Matt Kirkland, Robel Kitaba, Pablo Lucena, Murat Mugan, James Paussa, Kyle Sugrue, and Andrew Sutters.

In whatever we build next, we will continue to empower our engineers to take creative and iterative approaches to solving the unique challenges of Facebook's global infrastructure. We know that this is the only meaningful way to meet the extraordinary demands of connecting the billions of people who use Facebook's services around the world.

Related Posts



Nov 15, 2017

Open/R: Open routing for modern networks

Jun 20, 2018

2018 Networking @Scale recap

Sep 25, 2019

Networking @Scale 2019 recap

Related Positions

Software Engineer (AI)

PARIS, FRANCE

System Test Engineering Architect, Portal

MENLO PARK, US

Solutions Engineer

MENLO PARK, US

Software Engineer, Ads Ranking

SEATTLE, US

Software Engineer, Machine Learning (IGTV)

SAN FRANCISCO, US

See All Jobs

Join Our Engineering Community

Available Positions

Software Engineer (AI)
PARIS, FRANCE
System Test Engineering Architect, Portal
MENLO PARK, US
Solutions Engineer
MENLO PARK, US
Software Engineer, Ads Ranking
SEATTLE, US
Software Engineer, Machine Learning (IGTV)
SAN FRANCISCO, US

See All Jobs

Stay Connected

Facebook Engineering

Like

@fbOpenSource

Follow

facebook research

Facebook Research

Like

Facebook Developers

Like

RSS

Subscribe

Open Source

Facebook believes in building community through open source technology. Explore our latest projects in Artificial Intelligence, Data Infrastructure, Development Tools, Front End, Languages, Platforms, Security, Virtual Reality, and more.

ANDROID

iOS

WEB

BACKEND

HARDWARE

Learn More