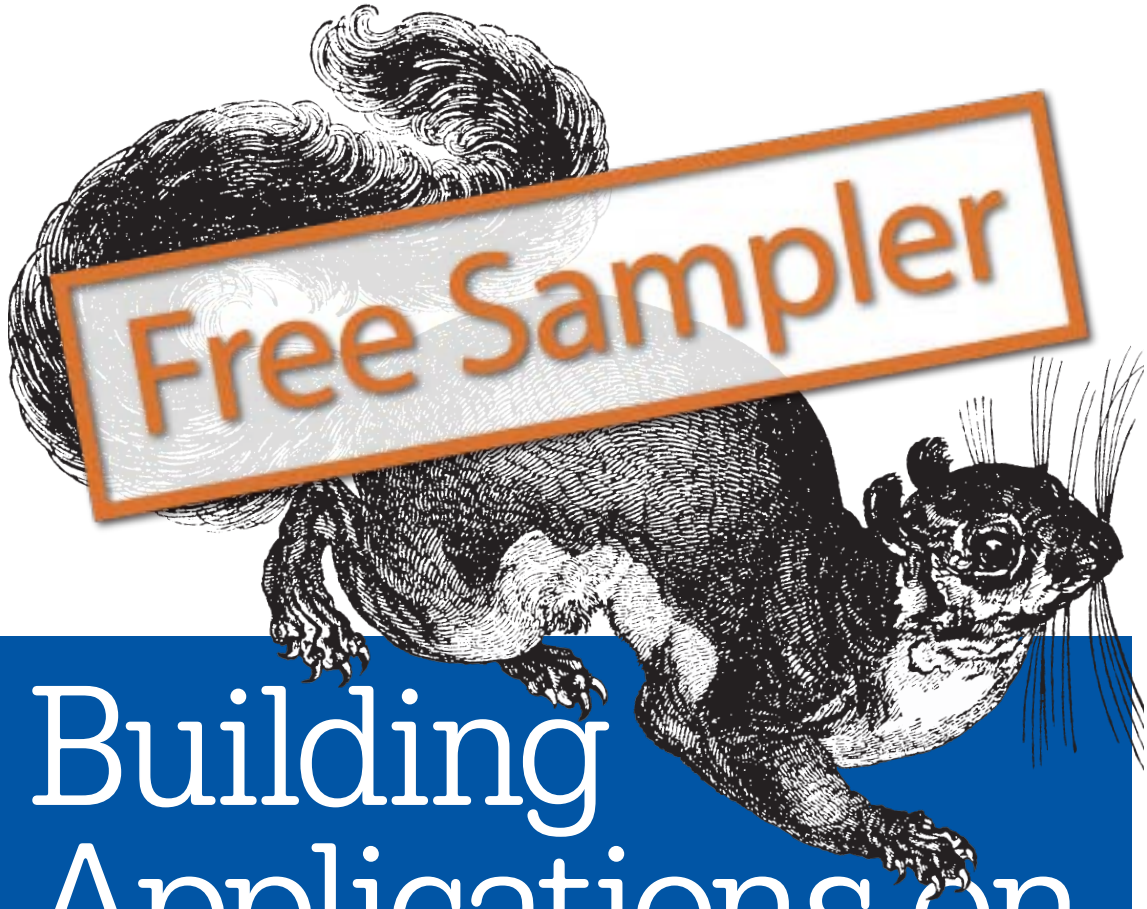
O'REILLY®



Building Applications on Mesos

LEVERAGING RESILIENT, SCALABLE, AND DISTRIBUTED SYSTEMS

David Greenberg

Building Applications on Mesos
by David Greenberg

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://safaribooksonline.com*). For more information, contact our corporate/institutional sales department: 800-998-9938 or *corporate@oreilly.com*.

Editors: Rachel Roumeliotis and Brian Foster
Production Editor: Nicholas Adams
Copyeditor: Rachel Head
Proofreader: James Fraleigh
Indexer: Wendy Catalano
Interior Designer: David Futato
Cover Designer: Randy Comer
Illustrator: Rebecca Demarest

December 2015: First Edition

Revision History for the First Edition
2015-12-04: First Release

See *http://oreilly.com/catalog/errata.csp?isbn=9781491926529* for release details.

978-1-491-92652-9

[LSI]

Table of Contents

CHAPTER 1

Introduction to Mesos

Let's take a trip back in time, to the year 1957. Computers that use transistors are starting to proliferate across universities and research laboratories. There is a problem, though—only one person can use a computer at a time. So, we have paper sign-up sheets so that we can reserve time slots on the machines. Since computers are so much more powerful than pencil and paper, they are in high demand. At the same time, since the computers are so expensive, if people don't use their whole reservations, then thousands of dollars of compute-time could be wasted! Luckily, the idea of operating systems already existed, more or less, at the time. A brilliant man named John McCarthy, who also invented LISP, had a great idea—what if all the users could submit their jobs to the computer, and the computer would automatically share its CPU resources among the many different jobs?

Jobs Became Programs

What we now call applications or programs used to be called jobs. We still can see this terminology in our shells, where, once we've backgrounded a process, we use the `jobs` command to inspect all the programs we've launched in the shell.

Once a single machine could be shared between jobs, we didn't need humans to supervise the sign-up sheets—now, everyone could use the machine, and share it more easily, since the machine could enforce quotas, priorities, and even egalitarian fairness (if so desired).

Fast-forward to 2010: with the falling costs of networked data transmission and storage, it's now possible to store every bit of information you can collect. To process all this data, you probably need to use Storm (a distributed real-time data processing system) and Hadoop. So, you get a whole mess of machines: a few for the Hadoop

JobTracker and Storm Nimbus (each with its own painstakingly crafted configuration), a few more for the HDFS NameNode and Secondary NameNode, 15 more that you'll install Hadoop TaskTrackers and HDFS DataNodes on, and 10 more that you use to run Storm Supervisors. At this point, you've managed to purchase 30 machines. However, if you decide that instead you'd like to use five of your Hadoop machines as Storm workers, you're in for painful task, because you now need to completely reprovision the Hadoop machines as Storm machines—a process that, as many practitioners can vouch, is not as easy as we'd wish.

It's 2011: the Berkeley AMP lab is in full swing. This is the group that today has brought us Spark, Tachyon, and many other distributed, scalable systems. One of the projects in the lab is called Mesos: it wants to be a platform for sharing your computer cluster between many disparate applications, such as Hadoop and MPI. In the Mesos paper,[1] the authors show promising results in sharing a single cluster of commodity computer hardware between many different applications, including MPI and Hadoop. This effectively solves the reprovisioning problem in clusters. Of course, companies like Google and Yahoo! are interested in the AMP lab, because they're working on similar problems. The team at Twitter take it a step further: they realize that Mesos solves a critical problem with which they've been struggling—how to efficiently wrangle the massive number of machines in their data centers—so, they manage to hire the lead author of the paper and architect of Mesos, PhD candidate Ben Hindman, so that he can tackle this problem at Twitter. Over the next several years, Mesos grows from being a simple research project into the core infrastructure that powers tens of thousands of servers at Twitter and many other companies.

So now that you know a little about Mesos (in the historical sense and context), you probably want to know what Mesos *is* (in the technical sense). Mesos is a system that allows you to harness all of the different machines in your computer cluster or data center and treat them as a single logical entity. With Mesos, the problem of having fixed sets of machines assigned to applications is no longer an issue. Instead, you can easily change which software is running on which machines, or even abdicate the responsibility of choosing which software runs on which machines to Mesos and allow it to make that decision for you, freeing up your time and mind to work on other (more interesting and more important) problems. For instance, in my company, Mesos allowed us to rapidly experiment with new distributed systems, such as Spark and Storm.

1 Although most people think of the NSDI 2011 paper "Mesos: A Platform for Fine-Grained Resource Sharing in the Data Center," by Ben Hindman, Andy Konwinski, Matei Zaharia, et al., originally Mesos was published as Nexus in "Nexus: A Common Substrate for Cluster Computing" by the same authors in 2009.

How to Use Mesos

In the simplest sense, Mesos is an orchestration platform for managing CPU, memory, and other resources across a cluster. Mesos uses containerization technology, such as Docker and Linux Containers (LXC), to accomplish this. However, Mesos provides much more than that—it provides real-time APIs for interacting with and developing for the cluster.

Many people wonder where Mesos sits in their stack. Is it part of their deployment infrastructure? Should their infrastructure team manage it? Or is Mesos an application development platform, like Heroku? Maybe Mesos should really belong to an application team… The answer to these questions is not straightforward, because Mesos provides functionality that crosses between Infrastructure as a Service (IaaS) and Platform as a Service (PaaS), in order to achieve greater efficiencies for your system. On the one hand, Mesos is infrastructure: it is the platform on which we deploy Hadoop and Storm and all the other PaaS goodies that we require for business. On the other hand, suppose that we're trying to develop an application that can process large-scale computations, similar to Hadoop or Spark. In that case, Mesos becomes the platform that we are building on. Suddenly, rather than treating it as a black box, we're very much concerned with its APIs and how to develop for it.

In practice, it's best to think of Mesos as serving a different purpose altogether. It allows you to stop focusing on individual machines, whether for developing your applications or for deploying them. Instead, it allows you to treat your cluster as a single, large group of resources. Mesos increases your agility for testing and deploying new distributed systems, as well as providing the platform to deploy in-house applications more quickly and efficiently than if you needed to involve your developer operations (DevOps) organization in those processes. For small projects, Mesos subsumes the many different systems that one might otherwise need to deal with—deployment tools like Ansible, Chef, and Puppet can now be relegated to basic roles in which they bootstrap Mesos. Mesos is the ultimate DevOps tool which thoroughly blurs the lines between the APIs the application runs against and the deployment of the application is a part of: Twitter has just three operators managing its tens-of-thousands-of-node Mesos clusters.

Let's now take a look at how Mesos fills two different roles: deployment system and execution platforms.

Mesos as a Deployment System

One way that you can think about Mesos is as a smart deployment system. First, let's look at a typical deployment system today, like Ansible or Chef. To use these tools, you write lists of tasks that should be performed on the various machines in your cluster. Tasks can modify files, install programs, and interact with supervision sys-

tems (like Supervisor and SystemD). Groups of tasks can be combined to describe a "role" or "recipe," which represents a higher-level concept, like setting up a database or web server. Finally, these groups of tasks are applied to an "inventory," or set of hosts, which then configures them according to the specification.

These tools are wildly more useful than the Bash, Perl, and SSH mashups of old; however, they suffer from fundamental limitations. On the one hand, they allow for configuration to be written in a structured, understandable form. They enable the use of best practices from software development—source control, encapsulation, and code reuse through libraries and plug-ins—to extend to system administration. On the other hand, they are fundamentally designed to make a fleet of machines conform to a static configuration. This is intentional: once you've run your Ansible or Puppet configuration, you want the cluster to be in the same, known consistent state. If, however, you want your cluster to dynamically reallocate resources or shift configurations depending on various external factors, such as the current load, these tools can't help you.

In this sense, it can be useful to think of Mesos as a deployment system. Distributed applications can be made into Mesos frameworks for use on a Mesos cluster. Rather than forcing you to use a static configuration description, each framework is essentially a role or recipe: the framework contains all the logic needed to install, start, monitor, and use a given application. Many frameworks are themselves platforms; when many applications follow the same general deployment pattern (like web servers), then a single framework can manage them all (like Marathon, a Mesos framework for stateless services). The power of Mesos is that since the framework is a live, running program, it can make decisions on the fly, reacting to changing workloads and cluster conditions. For instance, a framework could observe how many resources seem to be available, and change its execution strategy to better perform given the current cluster conditions. Because the deployment system runs constantly, it can notice and adapt to failures in real time, automatically starting up new servers as previous ones fail. Mesos frameworks are strictly more powerful than static descriptions used by more traditional deployment systems.

Thus, you can think of Mesos as supplanting much of the functionality that Ansible or Chef previously provided. You'll still need a traditional configuration manager to bootstrap the Mesos cluster; however, Mesos provides a more powerful platform for hosting applications: frameworks can automatically and dynamically adapt to failing machines and changing workloads, giving operators greater peace of mind and time to work on new initiatives.

Mesos as an Execution Platform

Another way to think of Mesos is as a platform for hosting applications. Today, you may use Heroku to run your web services, or you may manage a Hadoop cluster to

execute MapReduce jobs. In these examples, Heroku and the Hadoop cluster are the platforms for web services and MapReduce jobs. A Mesos cluster can also be a platform for higher-level applications. But if your system already works, why add Mesos to the mix? Mesos offers flexibility and mitigates the risk of being outpaced by technology development; with Mesos, launching a Spark cluster or switching to a newer technology is as simple as launching the framework and watching it bootstrap itself.

Consider the Heroku/PaaS scenario: Marathon can reliably launch your application, automatically starting up new instances when machines crash or go down for maintenance. Mesos-DNS and HAProxy can manage your load balancing for you (see Chapter 3). But once you've got Mesos running, why bother continuing to maintain your Hadoop cluster? After all, you could launch Hadoop on Mesos via Myriad, and you could even stop dealing with HDFS administration, since there's a framework for that (Mesos HDFS).

Thus, you can think of Mesos as an execution platform: rather than paying for third-party PaaS solutions and creating bespoke clusters for each big data analysis technology, you can have a single Mesos cluster. With your Mesos cluster as the foundation, you can easily launch whatever frameworks you require to provide whatever functionality is needed, now or as new requirements and technologies arise.

How This Book Is Structured

At this point, let's take a look at how the rest of the book will be structured. First, we're going to learn about Mesos itself. We'll cover the architecture of Mesos and its frameworks, and we'll learn how resources are allocated across the cluster. We'll also dive into configuration settings you'll want to know, such as how to provide service-level agreements (SLAs) to certain frameworks, and useful command-line settings to tune the overall performance and workload of your cluster.

Then, we'll learn about two existing open source Mesos frameworks—Marathon and Chronos—that provide application hosting and job scheduling functionality. We'll explore how to configure them, how to interact with them, and how to build deeper integrations. After learning about these frameworks, you will be able to host several common application architectures, such as web services and data pipelines, on Mesos, with high reliability and scalability.

In the later parts of this book, we'll go into great detail about how to build a custom framework for Mesos. We'll learn about the APIs that it exposes through the process of implementing a sample job scheduling framework in Java. In developing this framework, we'll delve into how to build on the core Mesos APIs, what some common pitfalls are and how to avoid them, and how to decide whether to build a new framework or adapt an existing system.

At the end of the book, we'll do a deeper dive into some advanced features of Mesos, such as its integration with Docker, its internal architecture, and several cutting-edge APIs, including persistent disk management for databases and the framework reservation system.

Summary

In essence, Mesos is an operating system: it manages your computers and harnesses them together into a single, logical unit. Companies use Mesos because of the reliability, flexibility, and efficiency it brings. Mesos clusters require very few administrators to ensure that even massive fleets of machines remain operational.

Frameworks are distributed applications that run on Mesos; for them, Mesos provides infrastructure and execution services. Frameworks can act as applications, like Hadoop or Storm, or as deployment systems, like Ansible or Chef. Mesos coordinates and collaborates with frameworks to allocate slices of machines in the Mesos cluster to various roles. This capability—brokering the division of machines between many frameworks—is how Mesos enables greater operational efficiency.

With this high-level view of Mesos in mind, let's start diving into Mesos concepts and get started with Mesos in practice!