O'REILLY®

Special Edition

FIELD GUIDE TO THE Distributed Development Stack

ANDREW ODEWAHN

A Field Guide to the Distributed Development Stack

Andrew Odewahn

Beijing • Cambridge • Farnham • Köln • Sebastopol • Tokyo O'REILLY®

A Field Guide to the Distributed Development Stack
by Andrew Odewahn

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 .

Editor: Andrew Odewahn
Copyeditor: Amanda Kersey
Interior Designer: David Futato
Cover Designer: Edie Freedman

October 2014: First Edition

Revision History for the First Edition
2004-10-01: First Release

978-1-491-91658-2

[LSI]

Table of Contents

Introduction **1**
How to Contribute 3

The Cloud Is the Default Platform **5**
Traditional Cloud Providers 6
Hosted PaaS 6
Internal Services 7
For More Information 7

CI Servers Deploy Code, Not Ops **9**
Tools 10
Continuous Deployment 11

The Codebase Is in Git **13**
Tools 14

The Entire Application Runs Locally in Development **17**
Tools 17

The Environment Is Automated in the Code **19**

Tools 20
For More Information 21

The Monitoring Infrastructure Is Critical 23
Tools 24

Tests Done in Code, Not by a QA Department 27
Tools 27

Containerization for Production Services 29

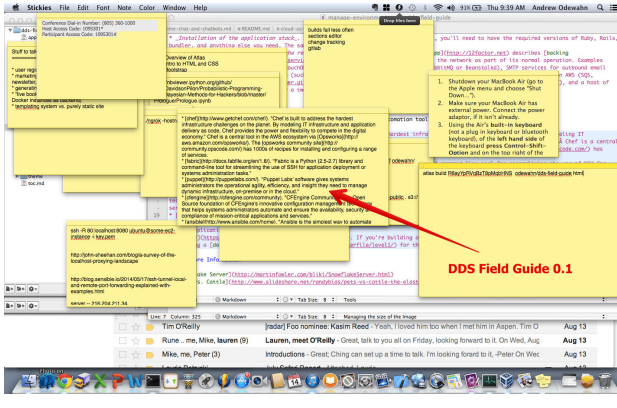
Real-time Chat and Chatbots 33
Chat 35
Chatbots 35
For More Information 35

Appendix: Contributors 37

Survey 39

Introduction

This project began while we were developing Atlas, O'Reilly Media's next-generation publishing tool. It seemed like every day we were finding interesting new tools in the DevOps space, so I started a "Sticky" for the most interesting-looking tools so I could explore them later.



At first, this worked fine. I was content to simply keep a list, where my only ordering criteria was "Huh, that looks cool. Someday when I have time, I'll take a look at that," in the same way you might buy an exercise DVD and then only occasionally pull it out and think "Huh, someday I'll get to that." But, as

anyone who has watched DevOps for any length of time can tell you, it's a space bursting with interesting and exciting new tools, so my list and guilt quickly got out of hand.

Once I reached the limits of the Sticky as a medium, I started to look for patterns in my list. Some were obvious. For example, many of the tools, like Ansible, Salt, or (to a certain extent) Dockerfiles, fit into a clear infrastructure-automation group pioneered by Chef, CFEngine, and Puppet. So, too, the many cloud services.

But where would something like CoreOS, Docker, or Mesos fit? As I thought about how to group them, they seemed somehow tied up with the notion of containerization, but that just seemed too narrow. Rather, these projects and tools were part of a much larger trend — enabling clustering and distributed computing—and containerization was just a piece. So, rather than group by technology, it made sense to me to group by trend—in other words, what did the tool enable, and why was that trend important?

Simultaneously, other people at O'Reilly were also exploring this same question, but from a different perspective. In "Everything is distributed," Courtney Nash, the chair of Velocity, was asking "how do we manage systems that are too large to understand, too complex to control, and that fail in unpredictable ways." In "Beyond the stack," Mike Loukides was thinking about how "a new toolset has grown up to support the development of massively distributed applications," and described the profound consequences that the shift from well-tended, internal servers to disposable VMs was having on the traditional "LAMP" stack. (As well as its hipster cousin, the MEAN stack.)

So, it's from this context that my Sticky list grew into this Field Guide to the Distributed Development Stack. The Guide is organized into buckets based on a general observation, such as:

- The cloud is the default platform
- CI servers deploy code, not ops

- The codebase is in git
- The entire application runs locally in development
- The environment is automated in the code
- The monitoring infrastructure is critical
- Tests done in code, not by a QA department

In addition to being a (hopefully) useful framework, the Guide is also meant to be a living resource. So, we've put the source on GitHub and invite you to contribute. If you feel like we've missed a tool (which we most certainly have, since new things are popping up every day) or a major theme, then fork the repo and send me a pull request. We'll be keeping this document up to date and republishing it as we watch this trend continue to grow. We'll use O'Reilly Atlas to pull in the contributions and periodically republish the guide.

This is still very much a work-in-progress, but I hope it will be a resource you'll add to your own Sticky collection.

How to Contribute

To contribute to the DDS field guide: *Fork this repo* Agree to the O'Reilly Contributor License Agreement *Add your tool / contribution* Submit a pull request

If your request is accepted, we'll add you to the Contributors page.

Making a larger contribution

If you want to make a suggestion or contribution that is larger than just a single tool, it might make sense to begin the conversation as a GitHub issue, rather than a pull request. For example, if you want to add a new theme, or want to add a major narrative section, it would be good to discuss that first to make sure it's suitable for the guide. While I certainly don't want to limit what people contribute in any way, it's also the case that this guide will be centrally curated by me and other O'Reilly contributors.

The Cloud Is the Default Platform

The accelerating transition to distributed, cloud-based platforms is one of the main drivers of the DDS trend. Organizations have adopted these services for a number of reasons: cost savings, increased speed for launching new projects, and scalability, to name just a few.

But, whatever the reasons for adoption, the default platform for many applications is increasingly assumed to be a transient, virtual, cloud-based platform, rather than a traditional server maintained by an internal IT group. Even in cases where the virtualization/PaaS solution is maintained in an internal cloud, the net effect is much the same.

The various platforms you're likely to encounter in this new world can be divided into three main groups:

- Traditional cloud providers. These allow you to quickly create storage or computing power as needed.
- Hosted PaaS services. These are value-added services built on top of raw hosting providers. For example, a PaaS might allow you to easily spin up a machine based on a particular stack when you deploy your application.

- Internal cloud and PaaS services.

These are tools and services you're likely to encounter here:

Traditional Cloud Providers

There are lots of hosting services. Here are some of the more popular:

- Amazon Web Services. AWS is probably the original model for pay-as-you-go infrastructure and remains one of the leading cloud platforms.
- Google Compute Engine. Google's cloud platform, which has become much more compelling now that they have open sourced their cluster management tools.
- Azure. Microsoft's cloud offering.
- Rackspace cloud. The cloud offering from Rackspace is mostly about compute power and storage. The API is well done, and the customer service is generally outstanding. Overall, though, it's not as full-featured as AWS.

Hosted PaaS

Hosted PaaS services add a layer on top of the raw offerings of hosting providers:

- Heroku. A PaaS service built on top of AWS. Unlike AWS, which gives you a raw machine, Heroku allows the developer to push an application into the service and have a corresponding application stack provision for the machine.
- OpenShift is a cloud-hosted PaaS solution developer by RedHat, the company behind RHEL Linux distribution.

- Digital Ocean. A lower-cost alternative to AWS that says it focuses on developers. It seems to be what a lot of developers use for side projects.
- Linode is a hosting service that offers SSD for really fast access.

Internal Services

These are tools that create virtual internal clouds (i.e., on premise). While they're technically running in your own internal datacenter, they enable the concept of scalable, on-demand resources:

- Open Stack. Open source software for building private and public clouds.
- VMWare vCloud Suite. A tool for running and managing VMWare images in your own data center.
- Mesos / Marathon / Chronos are a trio of technologies for managing and scheduling processes across a cluster of machines. Apache Mesos provides the core clustering technology for the stack. Marathon, from Mesosphere, is a distributed tool for starting, stopping, and managing individual jobs on a Mesos cluster. (So, it's like a distributed version of init or upstart). Chronos, developed by Airbnb's engineering team, is a distributed, fault-tolerant replacement for cron (the classic UNIX job scheduling tool) for scheduling when jobs will start.
- OpenShift Origin is an open source version of RedHat's Open Shift platform.

For More Information

You can find more important background at The Twelve Factor App.

CI Servers Deploy Code, Not Ops

Martin Fowler defines continuous deployment as "a software development practice where members of a team integrate their work frequently, usually each person integrates at least daily - leading to multiple integrations per day." This seminal article defines the key best practices as:

- Maintain a single source repository.
- Automate the build.
- Make your build self-testing.
- Everyone commits to the mainline every day.
- Every commit should build the mainline on an integration machine.
- Keep the build fast.
- Test in a clone of the production environment.
- Make it easy for anyone to get the latest executable.
- Everyone can see what's happening.
- Automate deployment.

The CI server executes a specific action on a repository when it receives a commit hook. For example, if a developer makes a commit against a repository called foo, the CI server might:

- Clone down a local copy of foo.

- Execute foo's test suites (see the section on application stacks for more about this).
- If the tests fail, send an alert to the development team and halt the process.
- If the test suite passes, deploy the code to a staging or even production server.

Tools

Here are a few of the CI servers you might encounter:

- Hudson. Hudson is a CI server from Oracle written in Java.
- Jenkins. Jenkins, a fork of Hudson, is one of a leading open source CI servers. It has a host of useful plug-ins for tasks like build tasks, error reporting, and repository management.
- Buildbot. Buildbot is an open source CI server based on Python.
- Travis. Travis is a hosted CI solution that is used primarily by the Ruby community, particularly Rails.
- Hubot. Hubot is a chatbot from GitHub. It allows you to easily create scripts that you can use inside your chatroom (e.g., Campfire or HipChat) to deploy new code, receive messages from the build server, or get messages from your monitoring tools when things go wrong.
- [Shippable] (http://shippable.com/) Shippable is a hosted CI solution built on Docker with Webhooks.
- [Werker] (http://wercker.com/). Werker is a hosted CI solution built on Docker with Webhooks.

Continuous Deployment

- [Distelli]
- [Capistrano]

The Codebase Is in Git

The version control system (VCS) is the heart of the process. At the most basic level, a VCS allows developers to keep track of all the changes made to a set of files and enables them to roll back to specific points in time in case something screws up. In some systems, like Subversion, the code is checked out and then checked back in from a central repository. If there is a conflict between two developers' files (for example, both of them edited the same line of code), then the two version must be merged. This can be a painful process.

In contrast, distributed version control systems (DVCS), like Git, are the heart of most new development processes. Rather than having a central, master copy that makes it difficult and expensive to merge a lot of contributions from developers, a DVCS makes it simple (well, simpler!) to have multiple people all working on the same codebase simultaneously in different *branches*, and these branches can be easily merged in a *master* branch.

While there are many different work styles, such as Git flow, the basic DVCS process is:

- There is an agreed-upon master repository, which is often on a public service like GitHub or BitBucket, or an internal server like GitLab or Mercurial.
- Each developer clones the master repository to his or her local machine.
- The developer creates a new branch, usually for a specific feature.
- The developer makes commits against the local copy.
- Once the feature is done, he or she merges the branch back into the master branch and pushes the change back to the master.
- Other developers pull from the master branch and merge their branch.
- The merged copy preserves the full version history of all the distributed copies.

In addition to these coordination functions, most version control systems also offer a feature called a *hook*. A hook is a process that fires once a specific event, like a commit, happens to the repository. Hooks can be defined in the repo itself, but also in the hosting service. For example, GitHub lets you define "service" hooks that are called whenever a specific event occurs. These hooks are the tie-in to the continuous integration (CI server).

Tools

Here are the key version control systems:

- Git. "Git is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency."
- Mercurial. "Mercurial is a free, distributed source control management tool. It efficiently handles

projects of any size and offers an easy and intuitive interface."

Hosting services provide a central point where you can manage and store all your code repositories. In addition to raw code storage, they usually offer features like issue tracking, collaborator management, and other process-oriented services.

The following table lists hosting services managed by a 3rd party. The pricing model is typically based on a block of repositories for a monthly fee.

- GitHub. One of the largest and most successful Git hosting services.
- BitBucket. Atlassian's Git hosting solution.
- GitLab.com. A hosting service based on the popular open source project GitLab HQ.
- Gitorious. Similar to GitLab, a hosted version of an open source tool that you can install and maintain yourself.

These are services that you can install and manage in your own environment:

- GitLab. "Project management and code hosting application."
- Gitosis. "software for hosting Git repositories"
- Gitorious. The self-hosted version of gitorious.org. (It's a Rails app.)

The Entire Application Runs Locally in Development

One of the key tenets of the DDS movement is that developers should have a simple way to install and run the entire app on their local machine. Being able to run it on their own system encourages creativity and flexibility and makes development much more fun and productive.

Vagrant is the key tool here. Basically, it takes the recipes you created with your environment tool (e.g., your Chef or Puppet files) and *provisions* (creates) a virtual machine that runs in a tool like Virtualbox or VMWare. Vagrant automatically maps a virtual drive from the virtual instance back to the host machine, allowing the developer to use his or her favorite editor / IDE but still run the application in an environment that matches the production environment as closely as possible.

Tools

The following table summarizes tools you will encounter:

- Vagrant. "Create and configure lightweight, reproducible, and portable development environments."

- VirtualBox. "VirtualBox is a powerful x86 and AMD64/Intel64 Virtualization product for enterprise as well as home use." Free and open source! This tool allows you to run a full image of another operating system (called the "guest") on your own machine (called the "host").
- VMware. One of the original virtualization solutions; Vagrant has a paid version that targets VMware fusion.
- ngrok. ngrok allows you to share applications running on your local machine to other users on the Internet. IT works by setting up a tunnel to ngrok, which then revers proxy to allow you to have a public URL. In the paid version (it's a donation model with a suggestion of $25, but you pan pay what you want), you can have it proxy to a custom domain.
- Vagrant Cloud. A service from the creator of Vagrant that allows you to share versioned Vagrant images.

Replicating third-party APIs locally

The develop "everything on localhost" approach breaks down somewhat when an application makes extensive use of third-party APIs. Clearly, you could not simply install Twitter or Facebook on your local machine. To get around this, there are a number of tools for mocking up the API results returned by these services. These include:

- Canned. "Server to respond with fake API responses, by using a directory of files for finding out what to say."
- WireMock. "WireMock is a flexible library for stubbing and mocking web services."

The Environment Is Automated in the Code

A key idea (maybe *the* key idea) of DevOps is that the environment in which your code will run should be modeled as code, and not be some separate thing that is a black box. (And, as we get a bit further down the stack, should be versioned with the code as well.) It seems pretty basic, but the idea is that you should have a "recipe" that allows you to recreate the environment at any moment. Some of the key parts of managing the environment include:

- *General configuration*. General configuration includes setting up the basic requirements for the app to even run, things like ensuring that whatever directory it will live in actually exists, creating any required users, setting up security groups, specifying where log files should be stored, exposing (or blocking) the proper ports, setting any required permissions, installing any license or cert files, and updating packages. In short, anything and everything that an app needs at the basic operating system level.
- *Installation of the application stack*. If you're writing a Rails app, for example, you'll need to have the

required versions of Ruby, Rails, bundler, and other stack-specific dependencies. The same goes for any other stack.
- *Installation and configuration of the required backing services*. The 12 Factor App describes backing services as "any service the app consumes over the network as part of its normal operation. Examples include datastores (such as MySQL or CouchDB), messaging/queueing systems (such as RabbitMQ or Beanstalkd), SMTP services for outbound email (such as Postfix), and caching systems (such as Memcached)." Backing services can also include third-party services, like Amazon AWS (SQS, dynamodb, etc.), GitHub, Twitter, and Parse. Ideally, the "code for a twelve-factor app makes no distinction between local and third party services."

Tools

The following table lists some of the configuration and environment automation tools you might encounter:

- Chef. "Chef is built to address the hardest infrastructure challenges on the planet. By modeling IT infrastructure and application delivery as code, Chef provides the power and flexibility to compete in the digital economy."
- OpsWorks. The OpsWorks community site has thousands of recipes for installing and configuring a range of services.
- Fabric. "Fabric is a Python (2.5-2.7) library and command-line tool for streamlining the use of SSH for application deployment or systems administration tasks."

- Puppet. "Puppet Labs' software gives systems administrators the operational agility, efficiency, and insight they need to manage dynamic infrastructure, on-premise or in the cloud."
- CFEngine. "CFEngine Community is the Open Source foundation of CFEngine's innovative configuration management technology that helps systems administrators automate and ensure the availability, security and compliance of mission-critical applications and services."
- Ansible. "Ansible is the simplest way to automate apps and IT infrastructure."
- Salt. "Fast, scalable and flexible software for data center automation, from infrastructure and any cloud, to the entire application stack"
- Docker. If you're building a Docker image, you can specify a lot of the dependencies by specifying a dockerfile for the container.

For More Information

- Snowflake Server
- Pets vs. Cattle

The Monitoring Infrastructure Is Critical

The monitoring infrastructure is perhaps the most foreign element in the DDS stack. Of all the parts of the software infrastructure, this was perhaps the most opaque. A disk would fill up, the monitoring system would alert the systems group, and they would quietly fix it without the developer being any the wiser.

However, as more of the traditional admin functions spread to other parts of the organization, there's an increasing need for developers to view the monitoring infrastructure as just another part of the app. In addition to providing the sorts of critical alerts on failures, many monitoring tools are tailored to the application stack and can be used to pinpoint performance bottlenecks.

Most of these systems have two components: a client and a server. The client is an agent that runs on the server you want to monitor; it is typically installed on the machine by a Chef or Puppet recipe (or Ansible or Salt or whatever tool you are using). Whether it's a daemon or a cron job, the client periodically reports back key metrics to the central server. The server provides the reporting interface, notification systems, and

other functions that are required to smoothly maintain a large number of systems.

Tools

Here are some monitoring tools you might encounter:

- New Relic. New Relic is an application monitoring tool that enables you to simply and easily report metrics from within your app. Once you do, however, you can get a wealth of information about the bottlenecks in your application.
- Scout. Scout is a hosted monitoring tool. It's very simple to set up, although it has far fewer plug-ins than Nagios.
- PagerDuty. PagerDuty is an alert system that's designed to allow you to create groups and roles that should be notified for a variety of configurable scenarios.
- loader.io. "Loader.io is a free load-testing service that allows you to stress-test your web-apps/apis with thousands of concurrent connections."
- Hubot. Hubot is a chatbot from GitHub. It allows you to easily create scripts that you can use inside your chatroom (e.g., Campfire or HipChat) to deploy new code, receive messages from the build server, or get messages from your monitoring tools when things go wrong.
- Nagios. Nagios is an open source monitoring tool that has been around for a long time. It has hundreds of client plug-ins that can report all manner of system performance metrics.
- Kale. Kale is Etsy's monitoring platform and is "designed to solve the problem of metrics overload."

- Graphite. Graphite is a tool for "scaleable realtime graphing." Once you have a data feed, graphite makes it simpler to get nice charts to spot anomalies.
- StatsD is a tool from Etsy. It's essentially a daemon process that can receive messages from your applications via a UDP port. StatsD receives and parses the message and then aggregates it so that it can be analyzed by another tool (like Graphite).
- Ganglia is a BSD-licensed monitoring tool that provide more granular detail than Nagios.
- InfluxDB is a time series, events, and metrics database.
- Grafana is an open source, feature-rich metrics dashboard and graph editor for Graphite, InfluxDB & OpenTSDB.

Tests Done in Code, Not by a QA Department

As described in the section on continuous deployment servers, running automated tests on each deploy or commit is an important way to ensure software quality.

Tools

The following table lists testing tools you're likely to encounter:

- CasperJS. "CasperJS is an open source navigation scripting and testing utility written in Javascript for the PhantomJS WebKit headless browser and SlimerJS (Gecko). It eases the process of defining a full navigation scenario and provides useful high-level functions, methods, and syntactic sugar for doing common tasks."
- PhantomJS. "PhantomJS is a headless WebKit scriptable with a JavaScript API. "
- Canned. "Server to respond with fake API responses by using a directory of files for finding out what to say."

Containerization for Production Services

Containerization is the idea that an application and all its dependencies can be packaged and shipped in a standardized way that is the same for any platform. This enables you, for example, to package a container you created and built on your development machine directly to a production server. Be aware that the container only needs the dependencies and code from your app; other systems (like, oh, the operating system) can be shared with the host environment via a Linux container. There are many, many tools in this rapidly emerging space, such as:

- CoreOS. CoreOS is a Linux distro built for running and manageing applications that are packaged as Docker images. The key components are:
  - Docker as the way you run apps.
  - etcd, which is a distributed key value database; it's sort of the "registry" you can use to share data between instances. It's bundled with the OS so that you can always count on it being there.
  - systemd, a distributed job system for scheduling and process management. I don't really

quite understand this yet but plan to dive in soon.
  - fleet. Fleet is a tool for managing processes on a CoreOS cluster.
- Deis. Deis is a self-hosted PaaS platform based on Docker and CoreOS. Basically, it allows you to create your own Heroku-like service based on buildpacks or "raw" Docker containers.
- Docker. Written in Go, Docker is an Open Source project that provides a clean and simple way to create system images based on a known filesystem, layer new elements onto those images, and then spin up running instances of what you've done. For example, you might start with a base images like "base/ubuntu", add a service like Redis, and then start the image to have a running Redis instance. Docker handles process management, networking, and other services for you, allowing you to focus on adding just the parts you need. There is also a company called (conveniently enough) Docker that maintains Docker, as well as providing a hosting service called the Index, where you can publish and maintain your images. Like GitHub, they have a "free for public/paid for private" model.
- Flynn.io. Flynn, like Deis, provides a self-hosted PaaS.
- geard is a tool for creating and managing Docker containers in OpenShift, RedHast's PaaS solution.
- kubernetes. Kubernetes is an open source version of Google's internal cluster management toolchain. You can use Kubernetes to run your own Docker containers on Google Compute Engine, Google's cloud platform.

- Mesos. Apache Mesos is a cluster resource manager that simplifies running applications on a shared pool of servers. Mesos supports containerized workloads via linux cgroups and now supports running tasks in Docker containers natively as of August 2014 with the release of version `0.20.0`.
- Panamax. "Panamax is a containerized app creator with an open-source app marketplace hosted in GitHub. Panamax provides a friendly interface for users of Docker, Fleet & CoreOS. With Panamax, you can easily create, share and deploy any containerized app no matter how complex it might be"

Real-time Chat and Chatbots

Once considered a relic of a bygone era, real-time chat systems are now a huge part of the workflow of most distributed teams. Like texting, chat allows developers to send each other messages in real time and is the place where most routine team communication takes place. (It can be the place where the team culture is formed and maintained.)

In addition to human participants, team chats can include a *chatbot*, which is a program that performs useful functions by monitoring the chatroom for special commands. The chatbot, which appears like anyone else in the room, can do routine tasks such as kicking off a build, locking the production server, reporting errors from the monitoring tools, or providing man-pages. And, programmers being programmers, the chatbot can also do things that reflect the culture of the team, like showing pictures of a pug, displaying a squirrel whenever you type "ship it," or providing responses in a Magic 8 Ball form when you ask it questions. The chatbot usually reflects the culture of the team

that's using it and can often become an informal mascot for the group.

Chat

These are some of the more popular chat services and resources:

- Campfire
- HipChat
- Slack
- Google Chat
- The Emoji Cheatsheet is a fun resource that lists text shortcuts that correspond to the emoji characters used in many chat services. For example, using the code ":boom:" creates a small explosion icon when it's rendered in chat.

Chatbots

These are resources for creating chatbots:

- Hubot is a Node.js chatbot engine from GitHub. It provides the essential services like logging into the char service, listening for commands directed its way, executing corresponding scripts, and returning output.
- Hubot Script Catalog is a directory of useful (and totally useless!) scripts that can be plugged into the Hubot framework.

For More Information

- Say "Hello" to Hubot is a blog post from GitHub, the company that developed the popular Hubot chatbot.

- ChatOps at GitHub is a video explaining the chatbot-based ops workflow at GitHub.

Appendix: Contributors

- Mike Loukides — @mikeloukides
- Courney Nash — @courtneynash
- Paco Nathan — @pacoid | https://github.com/ceteri
- Andrew Odewahn — @odewahn | https://github.com/odewahn

Survey

This field guide is a work in progress, and to make it a really useful tool, we really need your feedback. Please take a few minutes to fill out this short survey. Many thanks for your help.

http://oreil.ly/FGFeedback