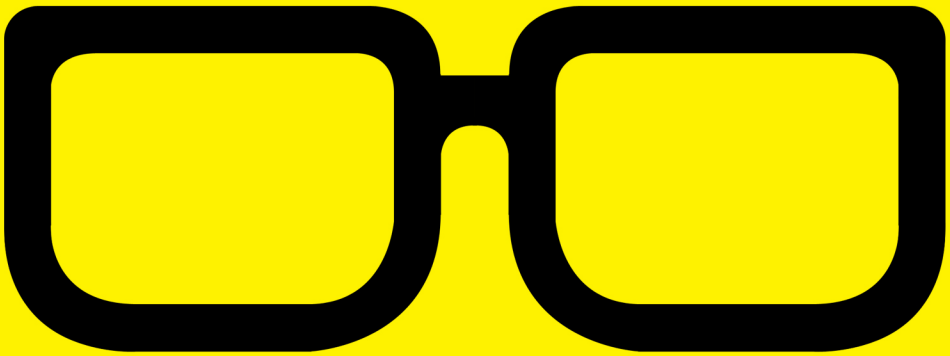
SPONSORED BY

IBM

GEEK GUIDE



The DevOps Toolbox

Tools and Technologies for Scale and Reliability

LINUX™ JOURNAL

Table of Contents

BILL CHILDERS is the Senior Development Operations Manager for a mobile device management company. Bill has worked in IT and DevOps since before the DevOps term was coined, and he has performed a variety of roles in software organizations: systems administrator, technical support engineer, lab manager, IT Manager and Director of Operations. He is the co-author of *Ubuntu Hacks* (O'Reilly and Associates, 2006), and he has been Virtual Editor of *Linux Journal* since 2009. He has spoken at conferences, such as Penguicon and LinuxWorld, and is enthusiastic about DevOps, IT and open source. He blogs at http://wildbill.nulldevice.net and can be found on Twitter at @wildbill.

GEEK GUIDES:

Mission-critical information for the most technical people on the planet.

About the Sponsor

The IBM Solution to Enable DevOps

IBM provides a total end-to-end DevOps solution including an open-standards-based platform that supports a continuous innovation, feedback and improvement lifecycle, enabling a business to plan, track, manage and automate all aspects of continuously delivering business ideas. At the same time, the business is able to manage both existing and new workloads in enterprise-class systems, and open the door to innovation with cloud and mobile solutions. This capability includes an iterative set of quality checks and verification phases that each product or piece of application code must pass before release to customers.

The IBM solution provides a continuous feedback loop for all aspects of the delivery process (e.g., customer experience and sentiments, quality metrics, service level agreements and environment data) and enables continuous testing of ideas and capabilities with end users in a customer-facing environment. The IBM DevOps solution consists of an open-standards-based platform that leverages hybrid cloud technologies, with end-to-end lifecycle capabilities and a set of services to enable organizations to improve their software delivery capabilities.

Visit http://ibm.com/devops.

The DevOps Toolbox

Tools and Technologies for Scale and Reliability

BILL CHILDERS
Senior Development Operations Manager
and Virtual Editor of *Linux Journal*

Introduction

When I was growing up, my father always said, "Work smarter, not harder." Now that I'm an adult, I've found that to be a core concept in my career as a DevOps engineer and manager. In order to work smarter, you've got to have good tools and technology in your corner doing a lot of the repetitive work, so you and your team can handle any exceptions that occur. More important, your tools need to

have the ability to evolve and grow over time according to the changing needs of your business and organization.

In this eBook, I discuss a few of the most important tools in the DevOps toolbox, the benefits of using them and some examples of each tool. It's important not to consider this a review of each tool, but rather a guide to foster thinking about what's appropriate for your own organization's needs.

Process and Documentation

Let me begin by discussing two things that are easy to overlook when talking about the DevOps toolbox: process and documentation. These two items are more about *how* an organization does its business, rather than *what* it uses to get that business done. As such, they can be thought of as check boxes on some manager's list, rather than tools that can help a DevOps organization on a daily basis. Once you think of process and documentation as tools to be wielded rather than simple prerequisites that may be forced upon you, you can start to use those tools to aid you and your team.

Process: Process can mean many things, but when distilled down to its basics, process is nothing more than a list of steps that an engineer or other team member can follow to ensure a consistent and successful outcome for any desired task. In the same way a DevOps engineer will write a script to automate a particular task, a process should act as a script for people. I like to refer to developing a process as "scripting in meatspace" for this reason. Whatever you do, make sure that your process is just enough for the task at hand.

The word "process" can make a lot of engineers twitch, and it's usually because those engineers have been forced into whatever process their organization has. When faced with a situation like this, a more effective approach can be to engage those engineers and make them part of the process creation and implementation work. Often, the people inside the process have the best insight and ideas as to how to streamline and optimize the workflow. Eventually, those engineers will become the biggest proponents of whatever process your team institutes, because they've helped invent it. A great side benefit of this is your process can become a living thing, continually evolving along with the growing and changing needs of your business. Process exists to assist your personnel. It's simply a tool to be utilized, like anything else.

Documentation: Documentation goes hand in hand along with process—without one, the other is almost useless. All the process in the world is difficult to follow without solid documentation, and documentation doesn't work if you have no process to back it up. In addition, documentation isn't something engineers have fun doing, so it is always last on the list to be delivered, if it's delivered at all.

Like process, the key to effective documentation isn't making it fancy, it's making it functional. It doesn't matter if the documentation has full screenshots and a table of contents. It doesn't matter if it's just comments in the source code or a bunch of notes on a wiki. If it's functional and meets the needs of your organization, then it's adequate to the task. The trick to documentation

is making sure it gets done. One way to ensure this is to implement a rule that a task isn't completed until it's documented.

Source Control

What Is Source Control, and Why Use It? Source control management (SCM) systems long have been tools used by programmers, but only recently have they been used by system administrators or DevOps personnel. Why should you use source control? Well, if you write scripts or other tools for internal use, they should be under source control, just like anything else your company develops. The benefits of source control are well documented elsewhere (see the Resources section), but here are a few key advantages source control provides to DevOps teams:

- Centralized storage of code.
- Allows for ease of auditing and backup.
- Enables easy rollback of changes should the need arise.

That last point is very interesting in a DevOps application. It's one thing to check scripts and code in to a source control system, but what if you check in your system *configurations* as well? Now you've got a great way to implement change control and have a built-in way to roll back any configuration changes if a change goes bad. When coupled with a configuration management system (more on that topic later), source control systems become extremely powerful.

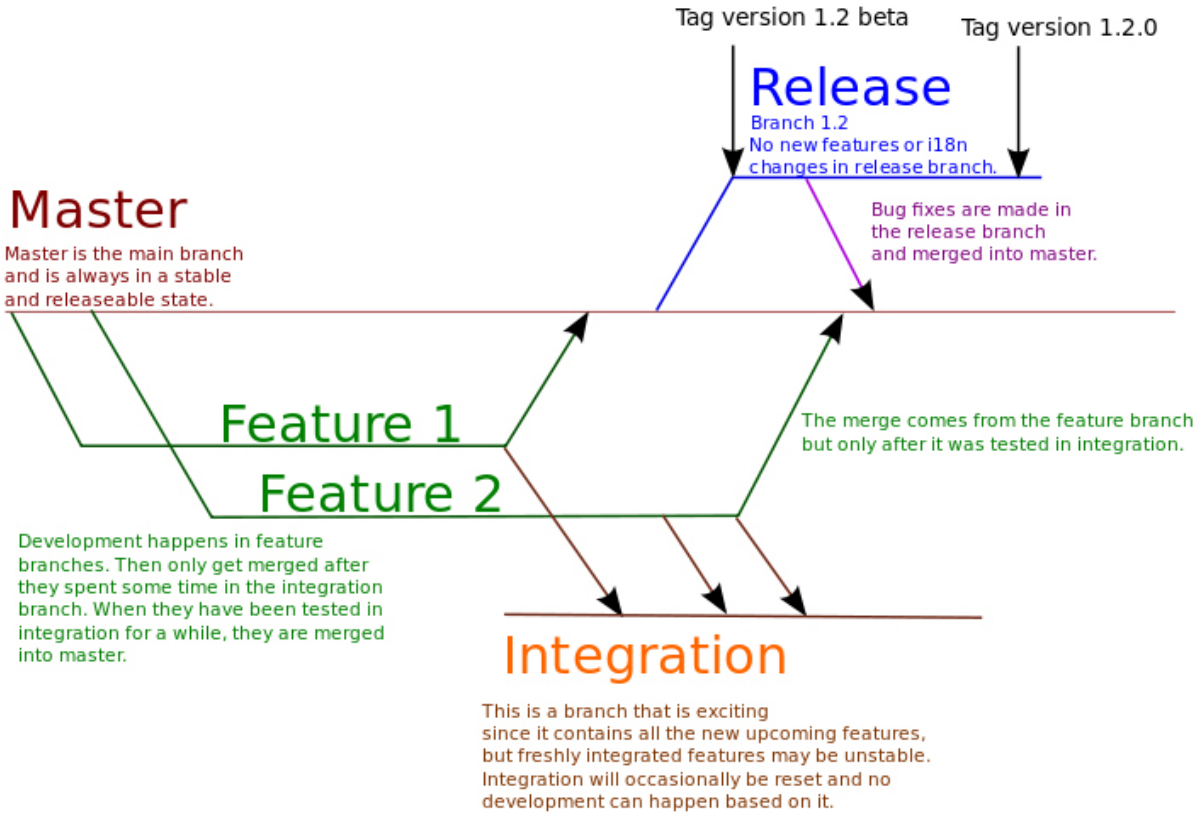


FIGURE 1. Git Branching Visualization

Code Review Processes: Code review is a great process to implement alongside source control. Code review is simple in concept: have another person check your work before it makes it into the main code repository and gets pushed to production. This process can be something as simple and informal as dropping an e-mail message with a code diff to another team member. It could be as involved as creating a branch in the source tree and filing a pull request against the branch, if you use the GitHub-style model. Whichever process you adopt, make sure it suits your holistic workflow and that your engineers support it. Your team's accuracy will improve, mistakes will drop, and uptime metrics will follow suit.

Source Control Systems: Source control isn't a new concept, and there are many possible options from which to choose. The odds are that your organization has one in place, so avoid re-inventing the wheel and try to leverage what your company already has. However, sometimes that doesn't work, so here's a list of some tools that are in wide use by DevOps organizations:

- Git: whether you're using Git on a server in-house or a hosted Git solution like GitHub or BitBucket, Git is a very powerful tool. It was designed by Linus Torvalds (the creator of Linux) to meet his particular needs, so it has a few design choices that cause it to stand apart from other SCMs. Git is open source and free to use, and it's a distributed SCM—it doesn't require a server to be functional. Atlassian, the makers of Bitbucket, also have a hosted product called Stash that provides Bitbucket's functionality on your own hardware, behind your firewall.

- Perforce: this is one of the more powerful SCMs out there, and it scales extremely well. If you work for a large company, there's a strong possibility Perforce is deployed internally. Perforce is free (as in beer) for use in open-source projects, but it's commercial software otherwise. Perforce is a centralized SCM and requires a server for use.

- Subversion: originally Subversion was founded by CollabNet, but it's now under the care of the Apache Foundation. Among open-source projects, Git largely has replaced Subversion, but it's still in fairly wide use.

Subversion is open source and free to use, although it is a centralized SCM and relies on a server to be functional.

- Mercurial: a distributed, open-source SCM that is mainly implemented in Python. It's got a lot of design features that make it easy for Subversion users to migrate to it, but it is unique in that it can handle both plain text and binary files. Atlassian's Bitbucket service can host Mercurial repositories for you, if you want to use a hosted Mercurial solution.

One warning: if you do decide to start checking configurations in to your SCM, make sure you don't make the SCM a choke point in your process, so that you can't deploy your configurations if you need to roll out a new software revision or config change. I've seen a lot of people on Twitter and IRC moaning from time to time, "GitHub is down! How are we going to work?" The point of a distributed SCM like Git is that you *can* continue to work without the server. Designing your workflow and process so you can't function in the event of a Git server going off-line (hosted or otherwise) is simply bad practice.

Configuration Management

Introduction to Configuration Management:

Configuration management (CM) software has been around for quite a few years, but cloud computing has brought a very real need to the forefront. The ability to manage a fleet of servers and ensure that all of them have the same configuration has been top priority for

many DevOps teams, and configuration management software has been the tool of choice. These technologies can install software packages, configure them for service and guarantee that those services are still installed, configured and in compliance with the policy set by the DevOps team.

A plethora of CM solutions exist, and each has its unique strengths. Configuration management systems frequently are chosen not only for their feature sets, but also according to the programming language in which they were written, so keep that in mind if extensibility is an issue for you.

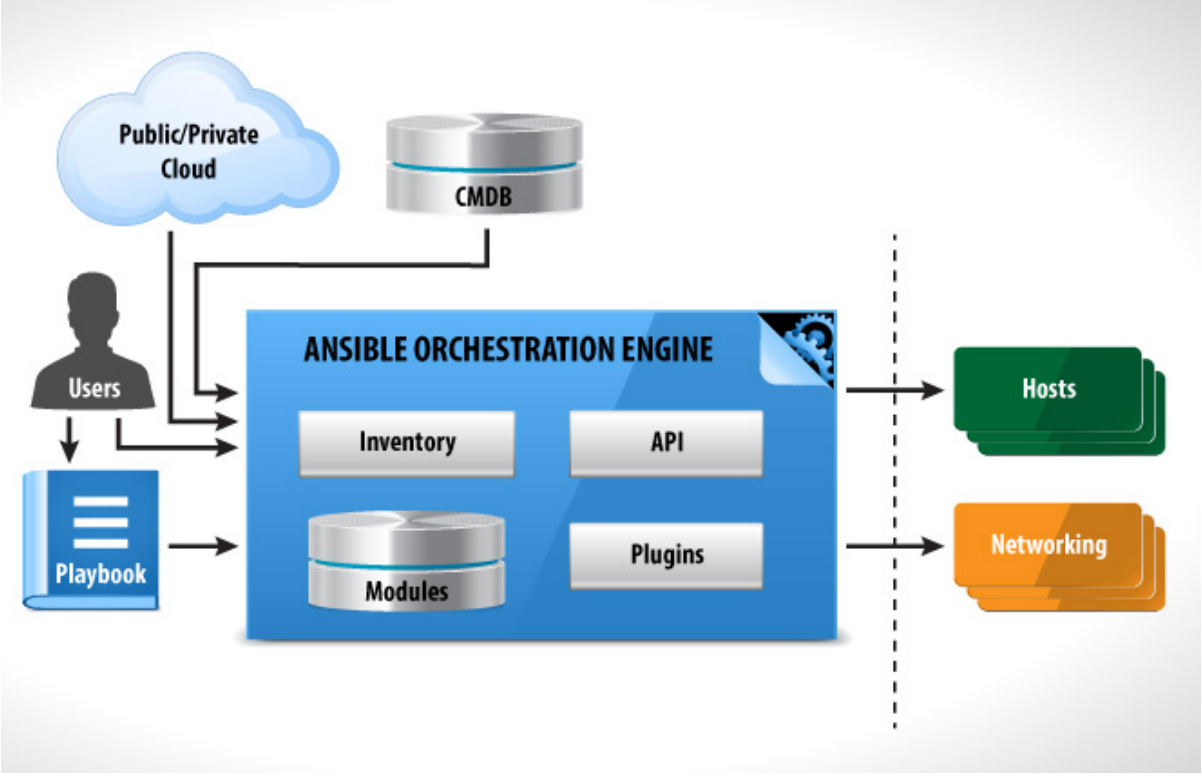


FIGURE 2. Block Diagram of Ansible's Workflow

Configuration Management Systems: The following is a list of some of the most popular configuration management systems at the time of this writing:

- Puppet: Puppet is one of the most popular CM packages out there, partially due to the fact that it's licensed as open source and partially due to the fact that it's written in Ruby. Puppet stores its information in "manifests", written by a system administrator in Puppet's own domain-specific language. Puppet also includes Facter, a program that will collect attributes about the system on which it's running and feed them back to Puppet so that those facts can be acted on. Puppet has excellent support, and many freely available add-on modules are available for it from Puppet Labs' Puppet Forge. Puppet can be run in standalone mode, but it's most frequently run with an agent installed on each client, talking to a centralized server. Puppet's supported operating systems include Linux and Microsoft Windows.

- Chef: Chef is as equally as popular as Puppet. Like Puppet, it's open-source and written in Ruby. Unlike Puppet, Chef's information is stored in "recipes" that are almost pure Ruby code. To make management simpler, recipes can be compiled into "cookbooks". Chef started out as an internal tool for the company Opscode, but it quickly grew beyond that. Chef also can be run in standalone mode (called chef-solo), but it's more commonly run in a client-server model. Chef primarily supports Linux, but has some Windows support as well.

Unlike Puppet and Chef, which have an agent running on each client, Ansible manages its nodes over SSH.

- Ansible: Ansible is an open-source CM system, written in Python. Unlike Puppet and Chef, which have an agent running on each client, Ansible manages its nodes over SSH. This means that there's no installation of agents or other prerequisites other than SSH with key-based authentication and Python. Not only does this save memory on the client machines, it also lowers network traffic since the clients don't have to contact the Ansible controller continually. Ansible's modules can be written in any language (such as Perl, Python or Bash), and Ansible uses YAML in files called "playbooks" to describe what happens on each node.

- Salt: Salt also is an open-source CM system, and like Ansible, it's written in Python. Salt sprang from a need for a very high-speed execution and data collection engine for system administrators. Salt uses a client-server model and supports Linux and Microsoft Windows.

Monitoring and Instrumentation

Part Alarm Clock, Part Instrument Panel—Your Monitoring System: Your operation's monitoring system

is just like the instrument panel in your car. It can tell you the state of the system, how it's performing, and what is broken. And just like the instrument panel in your car, it can tell you only information about what it's set up to display. Some cars have oil pressure gauges, while some have simple indicator lights for oil pressure. Your monitoring system, however, needs to be as detailed as possible to achieve your business goals.

There is no "one size fits all" monitoring system. Each one will have unique plugins, metrics and conditions to alert and track for every organization. The key to selecting and implementing the proper monitoring system is to find one that's able to scale and grow with your needs, while providing easy methods to plug in additional functionality when needed.

To be truly effective, you need to extend your monitoring system with new abilities from time to time. Did your data center's HVAC suddenly malfunction and blow freezing cold air into your disk array, killing several drives? You should be able to get an inexpensive temperature monitor in place to capture this condition and alert your team the next time the temperature dips below a certain threshold. Being able to continue to add and grow what system states you monitor is key to providing a reliable and stable service to your customers and end users.

An equally important part of any monitoring system is its ability to alert you when something *does* go wrong. Alerts should convey not only what went wrong, but they also should give DevOps personnel a possible action to do to remedy the situation, if needed. Configuring

your monitoring system's alerts can be touchy. If you set things too sensitive, your engineers will be flooded with information and become desensitized to the alerts. If everything's an emergency, then *nothing* is an emergency. However, if things are set too relaxed, real issues will slide under the radar untouched. If you continually re-evaluate your alerts and verify that each one is actionable, you can avoid this issue. Your engineers—and your customers—will thank you.

Monitoring Packages:

- Nagios: Nagios is one of the most popular monitoring systems in existence. Part of this is due to its open-source license, and part of this is due to its modular architecture. Nagios supports many operating systems, including Linux and Windows, but the real key to Nagios' success is in its plugin system. Known as NRPE (Nagios Remote Plugin Executor), the NRPE architecture allows developers to write modules for Nagios easily. Nagios also natively checks via other protocols, such as HTTP, ICMP and SNMP, but it does require an agent to run on each host being monitored to collect data about the host (such as disk space, CPU utilization and free RAM).

- SolarWinds: SolarWinds is more than a monitoring package; it's a modular suite of management software that can do server and application monitoring, virtualization monitoring and network storage management. It's commercial software, and it's

very powerful. SolarWinds comes pre-packaged with all manner of monitors for a typical enterprise environment. If your environment includes things like Oracle, Microsoft SQL Server or Exchange, SolarWinds may be exactly what you need. It's got the ability to act as a NetFlow collector and manage disk arrays like those from NetApp and EMC as well. SolarWinds aims to be a single pane of glass for an enterprise's monitoring needs, and it does a good job of it, but it's fairly expensive for that privilege.

- Zabbix: Zabbix is a scalable, open-source monitoring solution that can be installed atop a standard LAMP stack (Linux, Apache, MySQL, PHP). Zabbix is fairly simple to install and has a great network auto-discovery feature that can allow an engineer to get an instance running in a short amount of time. It can be run without an agent, if the hosts to be monitored support SNMP, but there's also an agent that optionally can be installed to get a richer set of monitoring data.

- Cacti: Cacti is primarily a data collection and graphing engine, although it has some basic monitoring and alerting abilities too. Cacti is an open-source solution powered by the RRDtool graphing engine. Owing to the fact that it's free to use and can scale quite well, it's often used by service providers to provide bandwidth statistics and other metrics to customers. However, unless your alerting needs are very basic, Cacti is best used as a collecting and trending engine.

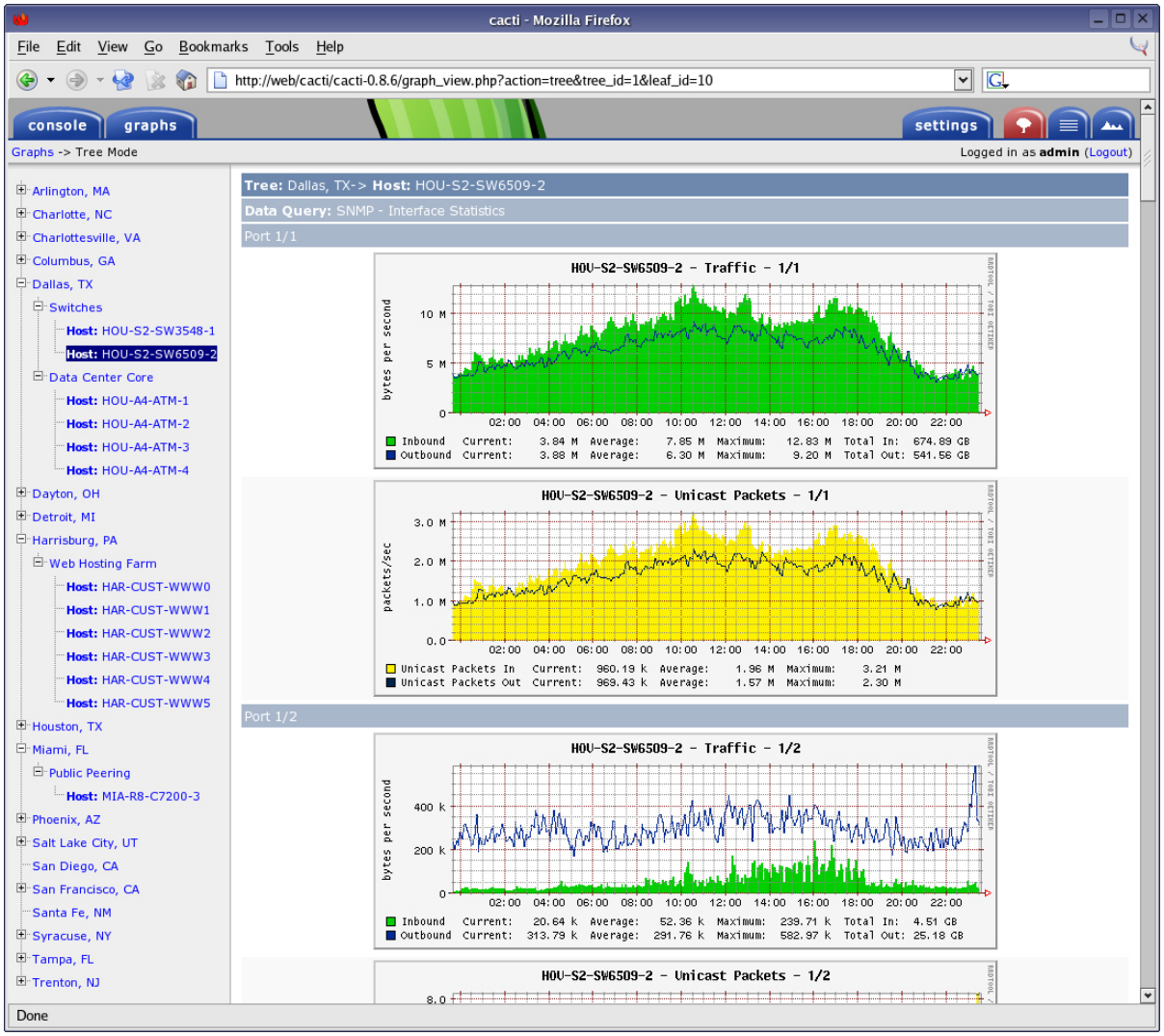


FIGURE 3. Cacti Screenshot

- Tivoli Monitoring: this monitoring package from IBM is geared at large enterprise operations who may have many different departmental needs. It's got the ability to monitor and trend not only on-premise systems in different geographies, but cloud servers as well. If you have SAP, DB2, MS SQL and other large enterprise-type services in your environment, Tivoli may be the answer

for you. It is commercial software and runs on most operating systems, including AIX, HP-UX, Linux, Solaris and Windows.

Sandboxes

A few other tools are helpful for DevOps personnel to iterate quickly and test new configurations in a disposable sandbox environment. A sandbox environment is a place where engineers can test a change or a new deployment quickly without disturbing the currently running production environment.

Desktop Virtualization: With laptops and workstations becoming more powerful through the years, it's feasible to run a low-capacity production analog on an engineer's local computer, inside a virtual machine or a group of virtual machines. This can not only be a huge cost savings, as the organization doesn't need to maintain a lot of test environments, but it's also a time savings, because engineers have full control over the environment while they are doing their work. Whether your engineers use VMware Workstation on Windows, VMware Fusion on Macs or the free VirtualBox software from Oracle on Windows, Mac or Linux, having a local virtual machine environment is extremely useful.

Rapid Prototype/Deployment Technologies: Once your engineers have a virtualization package on their local systems, it's best if they can spin up virtual machines quickly. This can be done via a number of legacy methods like kickstart, but that requires some network infrastructure that may not exist in your engineers' environment.

A new solution to this problem is Vagrant. Vagrant lets DevOps team members create a snapshot of what a particular machine in their environment is and then packages it up into an easily deployed container that can be pushed to any virtualization environment. If your production environment is in AWS, but you want to develop and test locally, that's not a problem. Vagrant can allow engineers to test their work locally without having to spin up an AWS node in the cloud. Once set up, it's as simple as issuing a `vagrant up` command to fire up the virtualized environment, and in a minute or two, the engineer is working on a system that resembles the real environment. It's great for engineers who work remotely—there's nothing like testing your changes while you sip a latte at Starbucks!

Conclusion

DevOps as a concept is all about iterating quickly, failing early to catch errors and bringing quality to the utter maximum. Having world-class tools at your engineers' disposal is *critical* to achieve this goal, and a huge selection of tools exist that can help drive your DevOps organization to new heights. However, in the end, the tools are only as good as the people, processes and methods that drive them. You must take the time to evaluate each and select the technologies that make the most sense for your company, team and each engineer. Feel free to take the DevOps Self Assessment at http://www.surveygizmo.com/s3/1659087/IBM-DevOps-Self-Assessment and see where you can improve your own organization's daily operations.■

Resources

“Source control is for everyone” by Michael Jansen:
http://www.embedded.com/design/prototyping-and-development/4006499/Source-control-is-for-everyone

“You’re Not Using Source Control? Read This!” by LornaJane:
http://www.lornajane.net/posts/2013/source-control-whitepaper

Wikipedia List of Revision Control Software:
http://en.wikipedia.org/wiki/List_of_revision_control_software

Git: http://git-scm.com

GitHub: http://github.com

Atlassian Bitbucket: https://bitbucket.org

Perforce: http://www.perforce.com

Apache Subversion: https://subversion.apache.org

Mercurial: http://mercurial.selenic.com

Open-Source Configuration Management Software (Wikipedia):
http://en.wikipedia.org/wiki/Comparison_of_open-source_configuration_management_software

Puppet Labs: http://www.puppetlabs.com

Puppet Forge: http://forge.puppetlabs.com

Chef: http://www.chef.io

Ansible: http://www.ansible.com

Salt: http://www.saltstack.com

Nagios: http://www.nagios.org

SolarWinds: http://www.solarwinds.com

Zabbix: http://www.zabbix.com

Cacti: http://www.cacti.net

Tivoli Monitoring:
http://www-03.ibm.com/software/products/en/tivomoni

Atlassian's Stash: https://www.atlassian.com/software/stash

VMware Workstation:
http://www.vmware.com/products/workstation

VMware Fusion: http://www.vmware.com/products/fusion

VirtualBox: http://www.virtualbox.org

Vagrant: https://www.vagrantup.com

DevOps Practices Self-Assessment: http://www.surveygizmo.com/s3/1659087/IBM-DevOps-Self-Assessment