SPONSORED BY

puppet

GEEK GUIDE

# Cloud-Scale Automation with Puppet

LINUX™ JOURNAL

# Table of Contents

About the Sponsor ..... 4

Introduction ..... 5

Getting Started with Puppet ..... 8

The Puppet Master ..... 12

Scaling Up and Out ..... 16

Automate Your Cloud Resources ..... 19

Puppet Enterprise ..... 26

Resources ..... 29

**JOHN S. TONELLO** is the Director of IT and Communications Manager for NYSERNet, New York's regional optical networking company, serving the state's colleges, universities and research centers. He's been a Linux user and enthusiast since building his first Slackware system from diskette more than 20 years ago.

**GEEK GUIDES:**
Mission-critical information for the most technical people on the planet.

***Copyright Statement***
© 2017 *Linux Journal*. All rights reserved.

This site/publication contains materials that have been created, developed or commissioned by, and published with the permission of, *Linux Journal* (the "Materials"), and this site and any such Materials are protected by international copyright and trademark laws.

THE MATERIALS ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT. The Materials are subject to change without notice and do not represent a commitment on the part of *Linux Journal* or its Web site sponsors. In no event shall *Linux Journal* or its sponsors be held liable for technical or editorial errors or omissions contained in the Materials, including without limitation, for any direct, indirect, incidental, special, exemplary or consequential damages whatsoever resulting from the use of any information contained in the Materials.

No part of the Materials (including but not limited to the text, images, audio and/or video) may be copied, reproduced, republished, uploaded, posted, transmitted or distributed in any way, in whole or in part, except as permitted under Sections 107 & 108 of the 1976 United States Copyright Act, without the express written consent of the publisher. One copy may be downloaded for your personal, noncommercial use on a single computer. In connection with such use, you may not modify or obscure any copyright or other proprietary notice.

The Materials may contain trademarks, services marks and logos that are the property of third parties. You are not permitted to use these trademarks, services marks or logos without prior written consent of such third parties.

*Linux Journal* and the *Linux Journal* logo are registered in the US Patent & Trademark Office. All other product or service names are the property of their respective owners. If you have any questions about these terms, or if you would like information about licensing materials from *Linux Journal*, please contact us via e-mail at info@linuxjournal.com.

## About the Sponsor

### Puppet

Puppet is driving the movement to a world of unconstrained software change. Its revolutionary platform is the industry standard for automating the delivery and operation of the software that powers everything around us. More than 33,000 companies—including more than 75 percent of the Fortune 100—use Puppet's open source and commercial solutions to adopt DevOps practices, achieve situational awareness and drive software change with confidence. Based in Portland, Oregon, Puppet is a privately held company with more than 470 employees around the world. Learn more at http://puppet.com.

# Cloud-Scale Automation with Puppet

JOHN S. TONELLO

## Introduction

If you're in charge of an IT operation these days, you're sure to be feeling pressure to take advantage of the cloud—pressure that might be coming from yourself, your boss, your CIO or even your customers. If you're not feeling the pressure, you will soon. Charting a path to shift resources and services to on-premises and off-site clouds takes some real thought and, ideally, some familiar tools to help you get there safely and in one piece.

A recent study found that 70% of organizations have at least one application living in the cloud, perhaps email

or web hosting or even a financial system. But that's relatively low-hanging fruit compared with, say, moving your development environment off site or hosting your proprietary apps on someone else's hardware. For that, you need to maintain a sense of control and visibility, and be able to do it from afar. After all, you can't just walk into a cloud-vendor's data center, roll in a crash cart and check on your servers.

If you've begun the process of moving resources to off-premises clouds, you're already hyper-aware of security, and you've thought hard about how best to ensure cloud VMs adhere to your company's standards and compliance policies. If you're not confident your data and servers will be safe and secure, you're not going to be able to sleep at night knowing that what you've spun up in the cloud is as bullet-proof as the servers you spin up on-premises.

If you've looked into any of the prominent cloud vendors—including AWS, Azure, Google Compute Engine and Rackspace—you know that each one provides the framework to ease your pain. Each vendor makes it easy (perhaps too easy) to create a cloud account, log in, start spinning up VMs and spend money. However, each cloud vendor has its own way of doing things, and if you dive in too quickly, you can run the risk of becoming over-reliant on a single vendor's framework. Instead of giving yourself more flexibility, you've actually done the opposite and put yourself at risk of vendor lock-in. Sure, it was easy to get your stuff into the cloud, but how easy is it to move it or get it out?

If you're just dipping your toe in with a few developer test environments or sandboxes, none of this really matters. You can spin up VMs, tear them down and throw caution

Fortunately, a growing number of tools can help you erase the line between cloud and on-premises server management, and automation tools are gaining popularity as a way to manage infrastructure at any scale—and in any environment.

to the wind. But once you move past that phase and start thinking about a production-grade environment, you really need to imagine ways your cloud infrastructure can match—and hopefully exceed—the performance of your on-premises infrastructure in every way that matters.

Fortunately, a growing number of tools can help you erase the line between cloud and on-premises server management, and automation tools are gaining popularity as a way to manage infrastructure at any scale—and in any environment. These tools are reliable, predictable and relatively simple, which means you and your organization will save valuable staff time and energy. They also help you take your team and your DevOps efforts to the next level, helping you think about tasks in new ways.

The leading automation tool is Puppet, which has shaped its approach around the idea of solving these problems and doing it in a way that's platform-agnostic, vendor-neutral and straightforward. It allows you to automate

the provisioning, configuration, deployment and ongoing management of your infrastructure and, unlike some other automation tools, the applications that run on it. So, no matter where your applications live, Puppet is a good option for helping you transition to the cloud or create a more modern DevOps environment.

As with all tools I deploy, I like to know how other people are using them and whether there's an active community to call on. These are often the folks who can give you insight into your own problems and deployments, and contribute to the public knowledge base. Since Puppet has been around for nearly 12 years now, and it's used by more than 30,000 organizations (from Apple to small boutique shops), it has a mature community to draw from and offer you help.

## Getting Started with Puppet

I first began to look into automation after spending a few weeks spinning up new servers, a chore I've now come to dread. It's not that I'm installing systems on bare metal anymore—I've long since converted our company servers to VMware and KVM virtual machines—but the routine server initialization is a bore. For example, I need to install iptables-persistent on my Ubuntu machines, set the IPv4 and IPv6 firewall rules, set a static IP, add some users with sudo authority and edit the hosts file. None of those things is difficult, of course, but I've done it all a million times. Cloned VMs have some of these settings pre-built, but I still have to make some edits and update each server manually. A few linked LXC containers save some of that time, but they represent only a fraction of my infrastructure.

If you're managing hundreds or even thousands of VMs, it simply can't be done without effective automation tools.

Then, after everything's up and running, I still have manual chores to perform for as long as each server is in use. Some might delight in this caretaker role, but I find it tedious and full of just enough gotchas to make it a pain. Frankly, I want to spend my time doing more interesting things than babysitting Linux servers—and my boss wouldn't mind it either.

The same goes for my Windows machines. Once the initial OS install is done, I need to import firewall rules, turn off a bunch of services I don't need and, again, manage updates. It's all necessary, but tedious. That might translate into job security for some, but it's not the stuff that sparks my imagination, boosts company revenue or speeds products to market. In fact, it's time stolen from larger, more mission-critical projects that keep our business, well, in business.

If you're like me, you feel the same dread for these routine tasks and your boss, like mine, laments the sunk staff time needed to manage it all. Sure, I have documentation to follow to make sure I apply the same settings to everything, but mistakes happen to the best of us—even when managing only a dozen or so machines. If you're managing hundreds or even thousands of VMs, it simply can't be done without effective automation tools.

If you're completely honest with yourself, you know even your best home-grown scripts—written and maintained by the most wizardly developer on your team—really can't scale the way you need them to.

It was this mundane stuff that first drew me to search out Puppet for a solution. I wasn't building an all-new server farm; I just wanted to control the servers I already had and, if possible, pave the way for moving some servers and services to the cloud. Puppet offered me a way to get started with very little overhead—I could install, test and start using it quickly and make my implementation more sophisticated as my needs grew.

Of course, needs vary widely among small-, medium- and large-sized companies. If you're small, you can deploy Puppet and puppetize your resources fairly quickly and then start improving your DevOps culture and seeing real and valuable benefits. Larger organizations have more highly fractured operations and many dev teams. They're fundamentally more complicated, with lots more variation in their control groups, or security and compliance requirements.

If you're on the large end of the spectrum, Puppet definitely can help reduce complexity, but regardless of the size of your organization, it's good to start small by installing Puppet agents on all your servers and start making things consistent at the OS layer, then look at the management layer, then middleware and then finally, the application layers. As you slowly work up the stack, you and your team will gain expertise

and grow your code base, which you easily can share across your organization and teams. This last bit isn't an afterthought. Being able to share consistent, easily discernible code with everyone who needs it will change how you do your work—and change it for the better.

Puppet offers several ways to get started. The folks at Puppet provide the Learning VM, a self-contained CentOS-based .ova you can download for free and spin up in a VirtualBox (or other VM host). Puppet also has an online emulator that will get you started learning Puppet's domain-specific language (DSL). Check the Resources section at the end of this guide for more information.

**FIGURE 1. The online Puppet Emulator allows you to get a taste for manifests and DSL, the domain-specific language Puppet employs to automate servers.**

## The Puppet Master

It all begins with setting up a Puppet master on your favorite Linux or Windows OS, preferably on a VM that's on the same subnet as the servers you want to manage. The master can be on-premises or in the cloud, but keep in mind that you need to open port 8140 in your firewall so the Puppet master can communicate with the Puppet agent on each of your servers. That means you'll also need forward and reverse DNS name resolution, which, in a pinch, can be managed by editing each server's hosts file.

The Puppet website offers great documentation on how to install open source Puppet and Puppet Enterprise on Linux and Windows. The enterprise version offers nice tools for node management and an easy-to-use GUI, among other things, but you can get started with the open-source version.

Once the Puppet master host is configured, you can start adding nodes by installing the Puppet agent on any Windows or Linux server. The Puppet master communicates with nodes via the agent by setting up shared certificates, and once enlisted, your agent node servers and VMs can be managed by the master, which is where you do your Puppet DSL work. This is where the fun begins—you can start managing individual servers remotely.

That work often takes the shape of what Puppet calls manifests, simple text files that contain descriptions of what you want to do. Puppet uses DSL to describe the desired state of a resource. DSL is a configuration language, not a programming language, and it enables you to declare what you want the system to be, not how to do it. Puppet

takes care of the "how" stuff behind the scenes, doing it consistnetly every time.

Take the simple example of adding a new user to your system. On a Linux server, you might run `adduser` and manually enter the user's real name, user name, root directory, shell preference and the like. If that user leaves the company, you'll probably go back to the server and run `deluser`, and then check a few files and directories to make sure the user no longer has privileges on the system. If you wanted to give that user permission on multiple servers, you'd repeat the process on each and every one.

With Puppet, a single, simple manifest can do it all. To create that same user, this bit of DSL is all it takes:

```
user: { 'msmith':
 ensure        => present,
 comment       => 'Mary Smith',
 home          => '/home/msmith/',
 shell         => '/bin/bash',
 password      => '$1$v4K9E8Wj$gZIHJ5JtQL5ZGZXeqSSsd0',
}
```

If I save this in a file called msmith.pp and run it, the msmith user will be created:

```
# puppet apply msmith.pp
```

If I add the above snippet to my site.pp manifest, which manages the state of all my Puppet-controlled servers, I can

apply this manifest (and create the msmith user) to all my nodes or just a subset of them:

```
# puppet apply -t
```

Regardless of whether I'm running Linux or Windows servers, Puppet will create a new msmith user anywhere I want it. I didn't have to describe how to create a new user, just that I wanted the new user to exist.

Don't worry if you don't get the syntax in a single glance; Puppet maintains the Puppet Language Style Guide, which shows you good and bad ways to write your DSL. It's also a great way to feel more confident with what you're doing. Check the Resources section at the end of this guide for more information.

Although the Puppet agent is fairly lightweight, you can begin to get a sense of how amazingly powerful it is. With just a few lines of code from the master, a Puppet agent can take the steps it needs to take to create this user. If the manifest instead had `ensure => absent`, Puppet would remove the user from all my servers at once.

You can apply this same approach to the OS layer, network layer, middleware layer and the application layer. You're essentially managing your entire stack with executable documentation. Write it once, and it gets done everywhere you want it and nowhere you don't. And everyone who views the code can understand what it does, because the DSL is clear and straightforward. Without even trying, you've taken a big step forward in your DevOps transition.

Of course, you can write your own manifests and build your own modules, but you can get up and running even faster by taking advantage of the more than 4,600 free modules available in the Puppet Forge.

Of course, you can write your own manifests and build your own modules, but you can get up and running even faster by taking advantage of the more than 4,600 free modules available in the Puppet Forge. It's where you'll find Puppet and user-contributed modules to install everything quickly—from MySQL or WordPress to full OpenStack deployments.

As with your own .pp files, pre-made modules use the Puppet DSL to describe the system (or application features) you want. With modules, you leverage your Puppetfiles, which live in the branches of your environment. Since you start with a basic production branch, adding a simple module command to /etc/puppetlabs/code/environments/production/Puppetfile makes the module and its classes available to your Puppet master, so it can, in turn, deploy what you need to all your nodes.

To declare (Puppet's term for use) a module from Puppet Forge, just add lines like this to your Puppetfile:

```
mod 'puppetlabs/apache', :latest
mod 'hunner-wordpress', '1.0.0'
```

The first example will install the Puppet module for Apache and grab the very latest version of the module. The second entry will make a WordPress module (made by a contributor named Hunner) available on your target node. Note that the version number here (1.0.0) is the version of the module, not the version of WordPress.

These examples just scratch the surface of what's available in Puppet Forge. It's worth spending some time browsing through all the possibilities.

If you've managed routine aspects of system administration, you know this sort of automation really can be a time-saving boon. It also can help you make all your systems more standards-compliant. Instead of managing each server separately, which invariably leads to configuration drift, you now can manage everything in exactly the same way and apply accepted IT standards, not custom one-offs. And instead of fighting fires and pouring time into routine tasks, you can start developing higher-order solutions for your enterprise.

## Scaling Up and Out

It's one thing to manage existing servers on a relatively small scale with Puppet, but the benefits of automation really start to pay off when you begin to use it to develop on- and off-premises cloud deployments. The benefits compound at cloud scale.

If you're currently manually building out development environments and then painstakingly replicating them on, say, AWS, you know there are a lot of moving parts to juggle. You have security implications and firewall

rules top of mind, and you also just want things to work in production in the same predictable way you got them to work in development. With Puppet, you can define an entire dev stack by applying all the modules you need in easy-to-understand manifests and deploy them anywhere. Adding a fully compliant production node based on your development stack—created from scratch or built from existing modules—is as easy as setting up an agent on a new server and applying the changes.

It's also important to know that using an automation tool like Puppet will do more for your team and your organization than just make tasks easier. It will do that, of course, but it will begin to change how you look at your resources, your development efforts and your business processes. For me, it's a bit like handing my shop over to a sort of digital chiropractor, who can help me snap all the once-disparate bones of my operation into place. Honestly, for the first time since I started managing servers, using Puppet makes me feel confident that everything is in place and as it should be.

Of course, we're the humans in this mix, and we tend to try to use new tools in old ways. It's sort of like the joke about the guy who was transitioning from a typewriter to a computer who kept putting Wite-Out on the screen every time he mistyped a word. Humans often resist change and blanket the new with the old, but adopting cloud infrastructures without changing how you think about the tasks—and the business outcomes—is a bit of a non-starter.

If you're transitioning to the cloud and you're not thinking about and using automation from the beginning, you're really just doing the same old things on someone else's computers.

If you're transitioning to the cloud and you're not thinking about and using automation from the beginning, you're really just doing the same old things on someone else's computers. In other words, the benefits of cloud infrastructure only truly come with end-to-end automation that's managed by a team that recognizes the broad value of a simplified, yet standardized, approach to its work.

With Puppet, you can more confidently (and quickly) deploy resources to your DevOps teams, write manifests that suit your business needs, configure (and reconfigure) VMs with the versions of software, security and resources you require, and move away from a one-off mentality. That means fully standards-compliant VMs, bare-metal servers and containers that are ready to use in moments, and that need less manual care and feeding. You can keep track and make universal changes knowing the Puppet master is keeping everything the way you want it. At the same time, your team is getting used to much greater standardization around the mundane stuff.

That's why I found puppetizing my resources so appealing. I could start small and grow as my needs grow. As you build out your cloud infrastructure—regardless of whether

you use AWS, Azure, Google Compute Engine, OpenStack, Rackspace, VMware or any off-premises provider or a mix—you're creating a framework for your entire operation, avoiding vendor lock-in, making everything more standards-compliant, and saving gobs of time—regardless of where everything physically lives.

## Automate Your Cloud Resources

When you begin to dig deeper into Puppet and become more sophisticated with your deployments, you soon recognize that your hardware—either on-premises or off—begins to fade into the background. Of course, all that compute, storage and network equipment is critical to you and your customers, but managing it is less about unboxing and manual configuration and more about defining what you need via code. That's the core of *Infrastructure as Code*.

As the number of systems you're trying to manage balloons from dozens to hundreds or even thousands, Day 2++ management becomes your greatest concern—that is, what happens after your VMs are up and running. On Day 1, you know that Puppet has defined everything the way you want it, but are the VMs really being underused? Are they being crushed by demand? It can be difficult to tell—even if you have monitoring tools in place.

Fortunately, you can take advantage of Facter, a part of the Puppet framework that provides Puppet (and you) with information about any agent node—everything from details about the underlying kernel to the system's time zone.

Here's a sample of some of the output when you run

Facter on any Puppet node (including the master):

```
# facter

...
is_virtual => true
kernel => Linux
kernelmajversion => 3.10
kernelrelease => 3.10.0-327.36.3.el7.x86_64
kernelversion => 3.10.0
load_averages => {
 15m => 0.11,
 1m => 0.09,
 5m => 0.06
}
memory => {
 swap => {
   available => “1.00 GiB”,
   available_bytes => 1073737728,
   capacity => “0%”,
   total => “1.00 GiB”,
   total_bytes => 1073737728,
   used => “0 bytes”,
   used_bytes => 0
 },
 system => {
   available => “1.03 GiB”,
   available_bytes => 1105281024,
   capacity => “72.20%”,
   total => “3.70 GiB”,
```

```
    total_bytes => 3975270400,
    used => "2.67 GiB",
    used_bytes => 2869989376
  }
} ...
```

This information is clearly readable and easy to understand. For example, you can see that Puppet recognizes that the node is a Linux VM with about 4GB of memory that's currently under very light load. This can be easily read and understood by the rest of your DevOps team too, which can dramatically improve your communication. Not that you're all going to walk around speaking DSL to each other, but the simple syntax and straightforward language can help you and your team spend less time trying to figure out someone's cryptic one-off notes and more time collaborating—and getting things done.

You probably can imagine how you might use this information to track and auto-scale your environment. By creating an auto-scaling group that manages a bunch of VMs on Azure, for example, you could use the Facter information to spin up new instances automatically, based on CPU and memory use. If these particular nodes are web servers, your Puppet auto-scaling group could deploy IIS or Apache, connect the new node to a load balancer and keep everything up to date. If the CPU use drops below a certain threshold, you could just as easily destroy the node, save money on your cloud platform costs and nearly eliminate the need to keep track of underutilized VMs manually.

Puppet modules for Azure, AWS, Google Compute

Engine and VMware are available to help you manage this natively. The modules take advantage of each platform's API, enabling you to create, stop, restart and deploy virtual machines and other resources.

For example, you can install the Azure Puppet module from any Windows or Linux system; placing your credentials in a manifest—including your Azure subscription ID, tenant ID, Client ID and password—gives Puppet authority to do its thing.

Once the module is installed, you now can create VMs with existing Azure parameters—for example, creating an Azure classic virtual machine:

```
azure_vm_classic { 'virtual-machine-name':
 ensure           => present,
 image            =>
'b39f27a8b8c64d52b05eac6a62ebad85__Ubuntu-14_04_2-LTS-amd64
➥-server-20150706-en-us-30GB',
 location         => 'West US',
 user             => 'username',
 size             => 'Medium',
 private_key_file  => '/path/to/private/key',
}
```

Or, create an Azure Resource Manager VM:

```
azure_vm { 'sample':
 ensure         => present,
 location       => 'eastus',
 image          => 'canonical:ubuntuserver:14.04.2-LTS:latest',
```

```
 user           => 'azureuser',
 password       => 'Password_!',
 size           => 'Standard_A0',
 resource_group => 'testresacc01',
}
```

These are the most basic; you can add a wide array of other properties, including setting domain names, address spaces and storage. By using Facter values, you also can create dynamic, node-specific properties. The same module can help you manage storage accounts, resource groups, template deployments, and the existing modules for AWS, Google Compute Engine and VMware offer similar capabilities for VM deployment.

On AWS:

```
ec2_instance { 'name-of-instance':
 ensure            => present,
 region            => 'us-east-1',
 availability_zone => 'us-east-1a',
 image_id          => 'ami-123456',
 instance_type     => 't1.micro',
 monitoring        => true,
 key_name          => 'name-of-existing-key',
 security_groups   => ['name-of-security-group'],
 user_data         => template('module/file-path.sh.epp'),
 tags              => {
   tag_name => 'value',
 },
}
```

And, on GCE:

```
gce_instance { “sample-agent-${value}”:
   ensure                  => present,
   zone                    => ‘us-central1-f’,
   startup_script          => ‘pe-simplified-agent.sh’,
   block_for_startup_script => true,
   metadata                => {
     ‘pe_role’             => ‘agent’,
     ‘pe_master’           => ‘puppet-test-enterprise-master-instance’,
     ‘pe_version’          => ‘3.3.1’,
  },
}
```

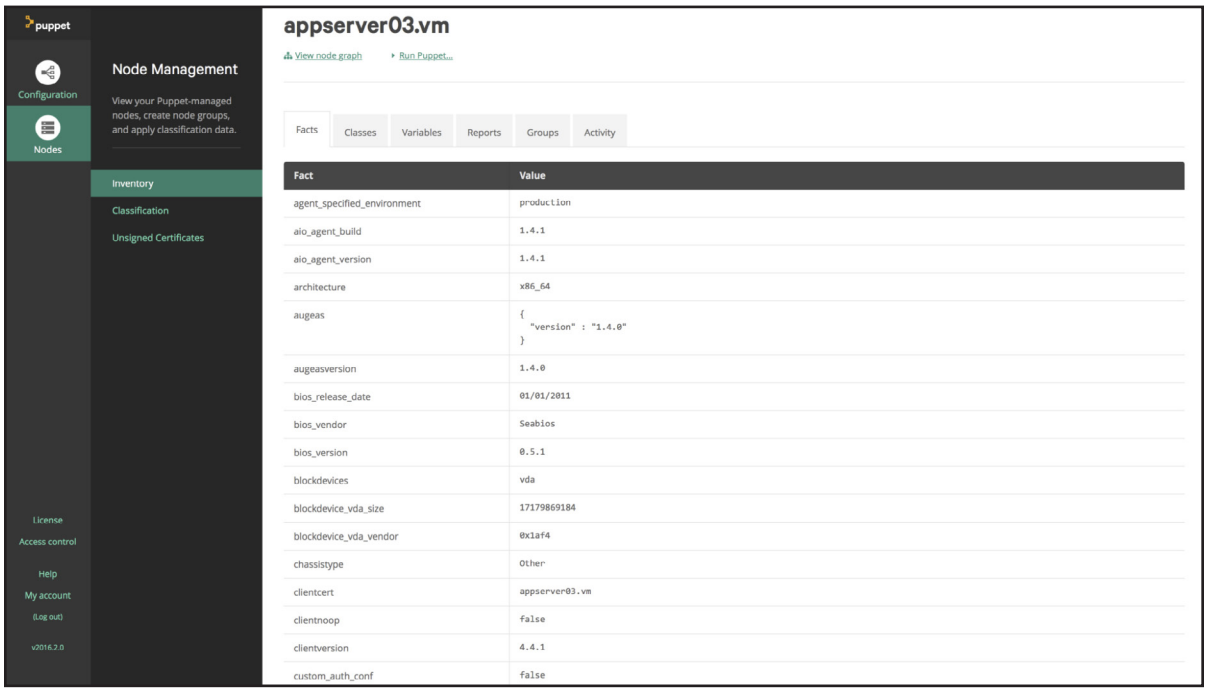


**FIGURE 2.** Puppet Enterprise in Azure offers a clean GUI that quickly reveals details about your managed nodes, including facts gathered from Puppet's Facter.

Typically, when you deploy Azure (or AWS or GCE) VMs, you order them through each vendor's control panel, perhaps setting a few templates to manage geographic location, instance size and the like. By using Puppet, you instead could define all those parameters in manifests and deploy them at the same time you deploy packages, users, firewall rules and other configurations you want. By pinning specific versions of packages, you can keep them from automatically updating and avoid changes that might break your stack. When you're ready to deploy updates, update a single manifest and all the changes happen across your infrastructure.

Puppet Enterprise is available in the Microsoft Azure Marketplace, and you can get a taste for how it works by

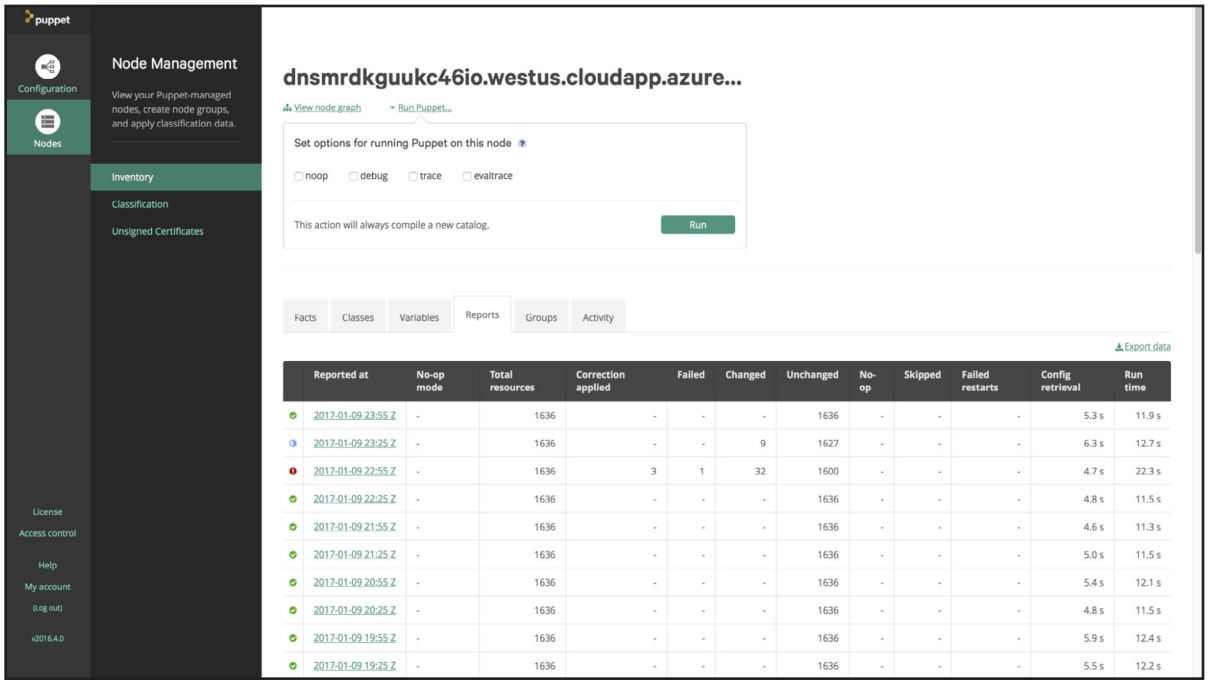


**FIGURE 3.** The sample Azure VM, managed by Puppet Enterprise in Azure, enables you to run Puppet in a variety of modes and see reports of activity.

taking it for a test drive. The example provisions a Puppet Enterprise VM and walks you through a live, guided tour.

## Puppet Enterprise

So far, I've primarily been describing the capabilities of open source Puppet, which can get you very far along. But, you can get even more economies of scale if you transition to Puppet Enterprise to manage server provisioning and application deployments. The enterprise version can help you move beyond tasks like tweaking OS settings, adding users and deploying applications, and on to large-scale shaping of hundreds or even thousands of bare-metal servers and VMs.

The browser-based interface extends what you've already done with raw manifest files, but it gives you quite a bit of additional flexibility and control. It also can shorten your Puppet learning curve.

For example, Puppet Enterprise can discover your infrastructure and provision machines automatically based on policies you define. Instead of having to write and run scripts manually yourself, Puppet Enterprise does much of the work for you, saving time and reducing errors. It also can help you manage all your cloud infrastructure with the same platform you use for your physical infrastructure, and even launch and manage Docker containers. By using a single platform for all these tasks, you're changing how you work—for the better—and truly tuning your DevOps efforts.

The Puppet Enterprise Web UI also offers tools for situational awareness. For example, it can figure out—and show you—all the VMs where an infrastructure change has happened. The event inspection tool will show you

those changes grouped into different contexts, such as configuration roles, so you can dive into the changes you expected—or didn't. With Puppet Enterprise's Node Graph, you can actually see dynamic models generated by an ever-growing number of Puppet modules. These situational awareness capabilities are particularly useful for ever-changing cloud infrastructures.

At the same time, the Puppet Enterprise Orchestrator allows you to model your server, VM or network configurations so you can deploy infrastructure and applications without having to figure out the correct order of operations. Puppet Enterprise automatically determines the order and figures out the credentials, network addresses and other information it needs to discover and securely share infrastructure and application services.

The Puppet Learning VM can give you a taste of how the GUI looks and works, and it includes the Orchestrator. It's

**FIGURE 4.** **Puppet Enterprise's Node Graph gives you graphical views of nodes and modules.**

a free download that allows you to install and run Puppet Enterprise for free on up to 10 nodes. It's a great way to get a taste for its capabilities, and if you're new to Puppet, it's another way to get your head around how automation works.

Whether you're looking to automate the management of a few servers, manage your cloud deployments across multiple vendors or begin building the foundation for a DevOps framework in your IT shop, Puppet has a lot to offer. Like many powerful tools though, you can't expect it to solve all your problems and enable you to save hundreds of staff hours on the first day. Take time to get familiar with its capabilities by starting small, perhaps managing existing servers and resources before moving on to larger-scale

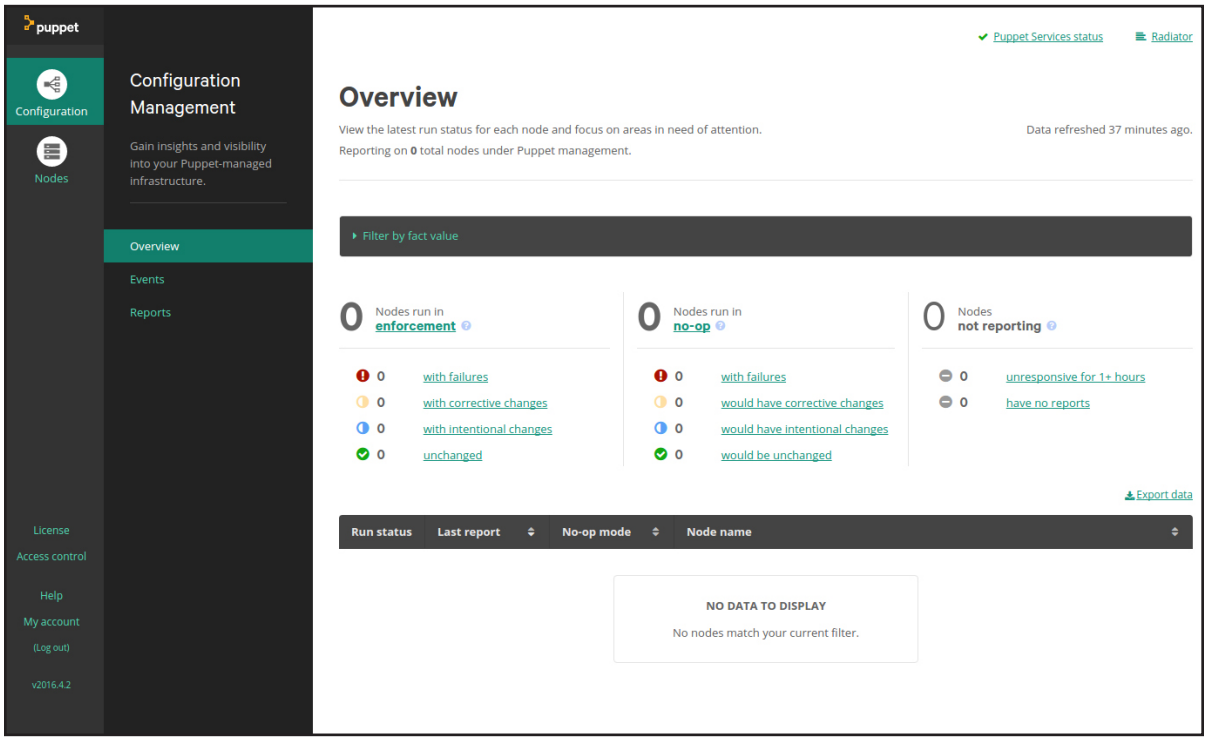


**FIGURE 5.** The main Puppet Learning VM GUI dashboard shows how nodes are configured and how policies are being enforced.

efforts. In the end, you're sure to find ways to reduce the number of manual tasks you and your staff have to perform, speed projects into production, show real business value and feel more confident about all the outcomes—at any scale.■

## Resources

Puppet Learning VM: https://puppet.com/download-learning-vm

Puppet Online Emulator: https://puppet.com/product/emulator

Puppet Documentation: https://docs.puppet.com

Puppet Forge: https://forge.puppet.com

The Puppet Language Style Guide:
https://docs.puppet.com/guides/style_guide.html

AWS Module: https://forge.puppet.com/puppetlabs/aws

Azure Module: https://forge.puppet.com/puppetlabs/azure

Test-Drive Puppet on Azure: https://puppet.com/test-drive-azure

Google Compute Engine Module:
https://forge.puppet.com/puppetlabs/gce_compute

Automated Provisioning:
https://puppet.com/product/capabilities/automated-provisioning

Try Puppet Enterprise:
https://puppet.com/download-puppet-enterprise