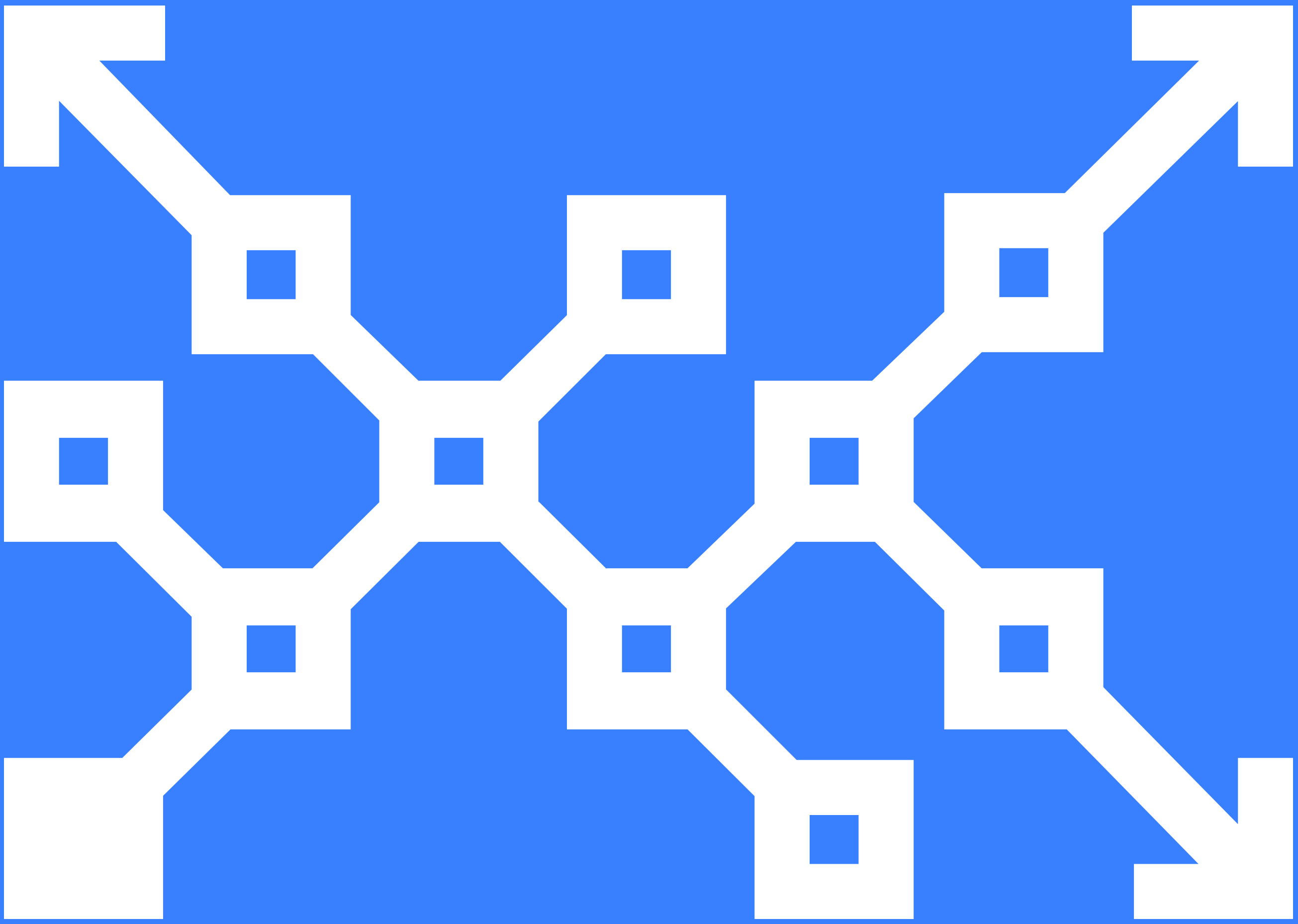
WHITE PAPER

Comparing Puppet and Ansible



puppet

Contents

Summary

Comparing Ansible and Puppet is a bit like comparing apples and oranges. While they are often mentioned in the same sentence, and both regarded as configuration management tools, they actually target a very different set of tasks. Puppet is focused on the holistic, ongoing management of your data center, systems, and applications. Ansible is primarily focused on task execution and orchestration.

IT organizations find Puppet Enterprise is best suited to:

- Scaling the IT organization's ability to contribute to the infrastructure configuration.
- Protecting the infrastructure from unplanned or unwanted changes by the multiple teams or contributors attempting to manage the same configuration.
- Enforcing and proving internal and regulatory configuration policy compliance.
- Monitoring and reporting on the current infrastructure state.
- Modeling complex and diverse configuration data sets.
- Testing infrastructure changes prior to deployment.

Ansible is often best suited for:

- Rapid adoption by small- to medium-sized IT teams.
- Multi-host, orchestrated, and event-based maintenance tasks.
- Multi-state application build deployments.

Ansible and Puppet capability comparison

Below are a set of important capabilities that organizations look at when deciding on a tool. These capabilities will be discussed throughout this document when discussing challenges modern IT organizations are facing today.

Capability	Puppet	Ansible
Desired state conflict detection • **What it is:** The ability to know when two bits of infrastructure code are attempting to manage the same configuration differently. • **Why it matters:** When many individuals and teams are managing the same infrastructure, it's important to make it impossible for different individuals to manage the same resource differently.	✓	X Multiple Ansible playbooks can silently manage the same resource differently. This can be true even for the same playbook in a single run.
Learning Curve • **What it is:** How quickly your team can learn the tool and begin using it for meaningful work. • **Why it matters:** The quicker a tool can be used to do meaningful work, the faster you'll realize return on investment (ROI). The quicker a tool can be learned, the faster your new team members can become fully productive.	– Puppet's domain specific language is a fully featured programming language specifically designed for managing configurations. It is written and read as a programming language. Puppet agents have to be distributed to existing systems before the tool can become useful.	✓ Ansible's data-driven playbooks are simpler to learn and write. Ansible's agentless approach means there's nothing to install on the systems you want to manage before the tool can become useful.

Curated and Supported Pre-built Content • **What it is:** Pre-built infrastructure code that is reviewed to be of high quality and supported by the respective company. • **Why it matters:** Finding which modules are high quality is an important part of discovering reusable content. Having support for mission critical downloadable infrastructure code ensures problems will be addressed quickly.	✓ Puppet reviews popular community maintained modules and adds high quality to our Approved Modules program. Puppet also builds and supports several modules under the Supported Modules program.	X While Ansible has a plethora of content available on its Galaxy, Ansible does not review and identify high quality roles, leaving that as an exercise to you. Ansible also does not support Galaxy roles.
Continuous enforcement • **What it is:** The ability to continuously ensure a configuration is in the its proper state. • **Why it matters:** Drift can happen on any mutable system for myriad reasons. When a change is made, it's critical to know the system will not change again until a change is deliberately made in line with business policy or strategy.	✓ Puppet's agent- based approach means the agent can continuously monitor for changes without checking into a central master for every run. Complete idempotence is a requirement for continuous running. Idempotence is built into the core of Puppet's framework.	– Ansible can be scheduled to run playbooks from a version control repository on a regular schedule using cron, but it complicates access control and forfeits direct control of changes. Continuous running requires complete idempotency. To get idempotency, Ansible modules must have it added as a feature.
Multi-host orchestration • **What it is:** The ability to coordinate configurations across multiple hosts. • **Why it matters:** Many infrastructure and business applications have multi-tiered, distributed components that need to be managed in a particular order.	✓	✓

Multi-state orchestration • **What it is:** Orchestrated processes where a single system or configuration can go through multiple states during the process. For example: a load balancer being drained in the beginning of the process and repopulated at the end. • **Why it matters:** Large-scale web applications often do phased deployments behind a load balancer. Being able to remove a subset of systems from the load balancer, updating the applications, and re-adding the systems to the load balancer is important for a successful rollout of these applications.	X Puppet's orchestration is entirely driven by the desired end state. A single system cannot have different states at different times of a deployment.	✓
Internal and regulatory policy compliance • **What it is:** Enforcement of configuration policies that comply with regulatory and internal policies. • **Why it matters:** Being able to confidently know the current configuration state of the infrastructure is a critical practice for guaranteeing compliance. Being able to prove compliance at any moment creates confidence in management and auditors.	✓ Only Puppet Enterprise can ensure a configuration can be enforced in just one single state. Puppet's declarative approach means Puppet can identify if different parts of the Puppet code base are attempting to manage the same configuration differently.	– Both products are capable of applying compliance-related configurations. However, Ansible and other imperative tools can silently overwrite a compliant configuration with a non-compliant value later in the same run, or in a later run of a different playbook.
Situational awareness • **What it is:** Ability to know the exact state of the infrastructure, be aware of when changes occur, and know whether a change was intentional or happened out of band. • **Why it matters:** The ability to identify when and where change occurs, and the type of change, is critical to knowing whether someone has bypassed the change management process.	✓ Puppet can tell you whether the system changed between Puppet runs (default run is every 30 minutes), or whether the change was the result of the Puppet code changing.	X Ansible can change systems, but it has no way of knowing whether the change was needed because of a playbook update or whether it was needed because of configuration drift (changes made by something/someone other than the configuration management tool).

Unmanaged resource purging	✓	✗
• **What it is:** Ability to remove configurations that shouldn't be present on the system. • **Why it matters:** For certain configuration types, it's important to know that only the configurations that are supposed to be there are the only ones there. For example, old or unauthorized firewall rules should be automatically removed from the system. SSH keys and Apache vhosts are other examples.	Puppet's declarative approach means Puppet knows every configuration being managed by the entire Puppet code base. Puppet can identify configurations on the system that are not managed by Puppet, and take necessary action to remove them.	Ansible cannot identify configurations of a certain type that are not being managed by Ansible. So Ansible cannot purge unmanaged firewall rules, SSH keys, system users, etc., from the system.

Scaling your organization

Larger IT organizations often have many individual contributors and specialized teams all working together to manage the infrastructure. Operating at scale is a challenge that will never go away, and the tools you choose can make the difference between working effectively as a team and perpetuating — or even reinforcing — siloed thinking, turf wars, and disjointed processes.

The tool you use needs to address several challenges:

- **Detecting conflict.** Any infrastructure tool needs to understand when multiple teams or contributors intend to manage the same configuration differently. This needs to be known *prior* to any work being done on the system.
- **Making dependencies transparent.** Infrastructure configuration can often be complex, with layers of dependencies. It's critical that a tool not only makes the order of operations clear, but also make clear the dependencies that exist between multiple configurations.
- **Establishing a single source of truth.** Many tools are good at managing parts of your infrastructure. IT teams will often use one set of tools for managing and orchestrating application deployments with another set for enforcing core infrastructure and middleware. It's important that your tools can *never* conflict, while simultaneously ensuring that all underlying infrastructure is in its intended state prior to deploying application services.

Detecting conflict

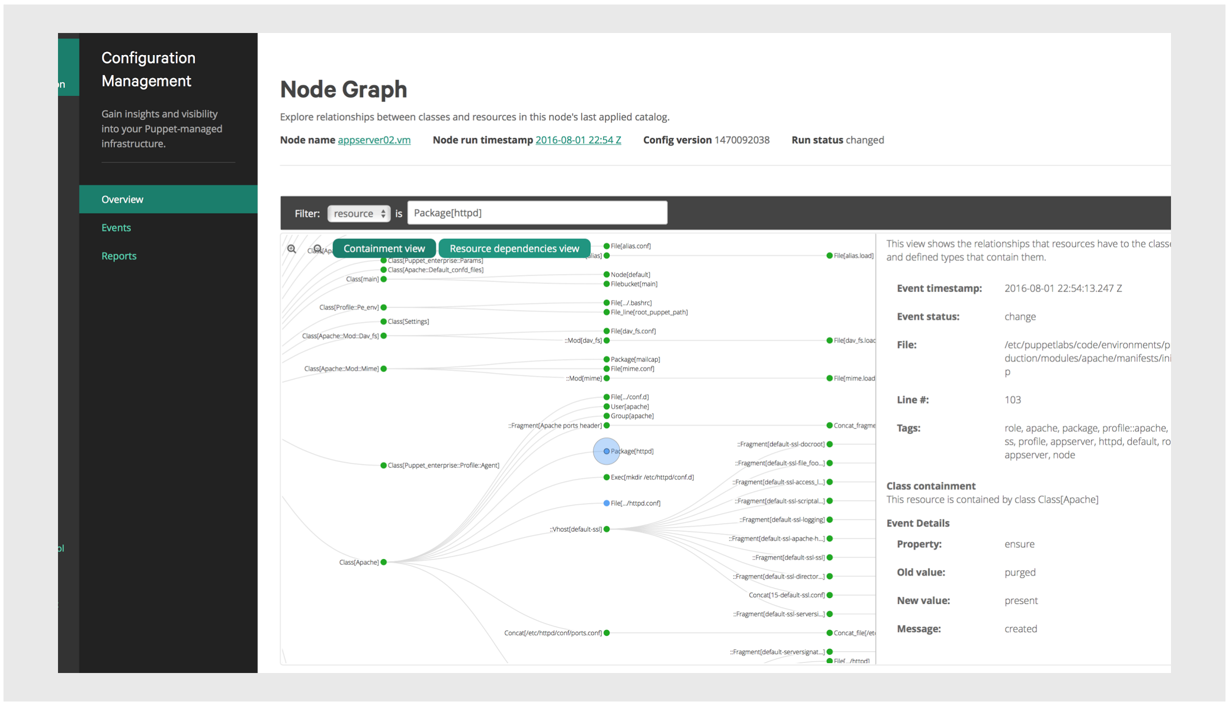
Puppet has desired state conflict detection built right into how Puppet works. Unlike nearly every other infrastructure management tool, Puppet does not blindly execute a series of commands as it reads them. Puppet reads and understands every line of infrastructure code *prior* to doing any work. As a result, Puppet has the intelligence to know every desired configuration for every system, and every dependency between configurations, even if the dependencies span multiple systems.

This deep knowledge means Puppet can identify if two different parts of the infrastructure code are attempting to manage the same configuration differently. This is useful when multiple teams are writing infrastructure code, or if you have multiple contributors on the same team.

Transparent dependencies

When you're managing hundreds or thousands of configurations on a single system, or managing hundreds of different services across a multitude of business applications, the need to understand how they relate is paramount. This is especially true when different teams manage different application services and parts of the system's overall configuration.

As more individuals and teams begin contributing to the infrastructure codebase, the code can quickly become a collection of black boxes. Puppet gathers *all* the infrastructure code assigned to a system to generate a complete model for how the system should look. Puppet Enterprise shows you the resulting model so you can see how each piece of infrastructure code contributes to a system's desired state. The black box is opened, and everyone who deals with infrastructure can see the state of things, including the dependencies.


Configuration Management
Gain insights and visibility into your Puppet-managed infrastructure.
Overview
Events
Reports
Containment view
Resource dependencies view
Class[main]
Class[Puppet_enterprise::Params]
Class[Apache::Default_confd_files]
Class[Profile::Pe_env]
Class[Settings]
Class[Apache::Mod::Dav_fs]
Class[Apache::Mod::Mime]
Class[Puppet_enterprise::Profile::Agent]
Class[Apache]
File[alias.conf]
Node[default]
Filebucket[main]
File[.../.bashrc]
File_line[root_puppet_path]
File[dav_fs.conf]
::Mod[dav_fs]
Package[mailcap]
File[mime.conf]
::Mod[mime]
File[.../conf.d]
User[apache]
Group[apache]
::Fragment[Apache ports header]
Package[httpd]
Exec[mkdir /etc/httpd/conf.d]
File[.../httpd.conf]
::Vhost[default-ssl]
Concat[/etc/httpd/conf/ports.conf]
File[alias.load]
File[dav_fs.load]
File[mime.load]
::Fragment[default-ssl-docroot]
::Fragment[default-ssl-file_foo...]
::Fragment[default-ssl-access_l...]
::Fragment[default-ssl-scriptal...]
::Fragment[default-ssl-logging]
::Fragment[default-ssl-apache-h...]
::Fragment[default-ssl-ssl]
::Fragment[default-ssl-director...]
Concat[15-default-ssl.conf]
::Fragment[default-ssl-serversi...]
::Fragment[default-serversignat...]

Node Graph

Explore relationships between classes and resources in this node's last applied catalog.

Node name appserver02.vm **Node run timestamp** 2016-08-01 22:54 Z **Config version** 1470092038 **Run status** changed

Filter: resource is Package[httpd]

This view shows the relationships that resources have to the classe
and defined types that contain them.

Event timestamp: 2016-08-01 22:54:13.247 Z

Event status: change

File: /etc/puppetlabs/code/environments/p
duction/modules/apache/manifests/ini
p

Line #: 103

Tags: role, apache, package, profile::apache,
ss, profile, appserver, httpd, default, ro
appserver, node

Class containment
This resource is contained by class Class[Apache]

Event Details

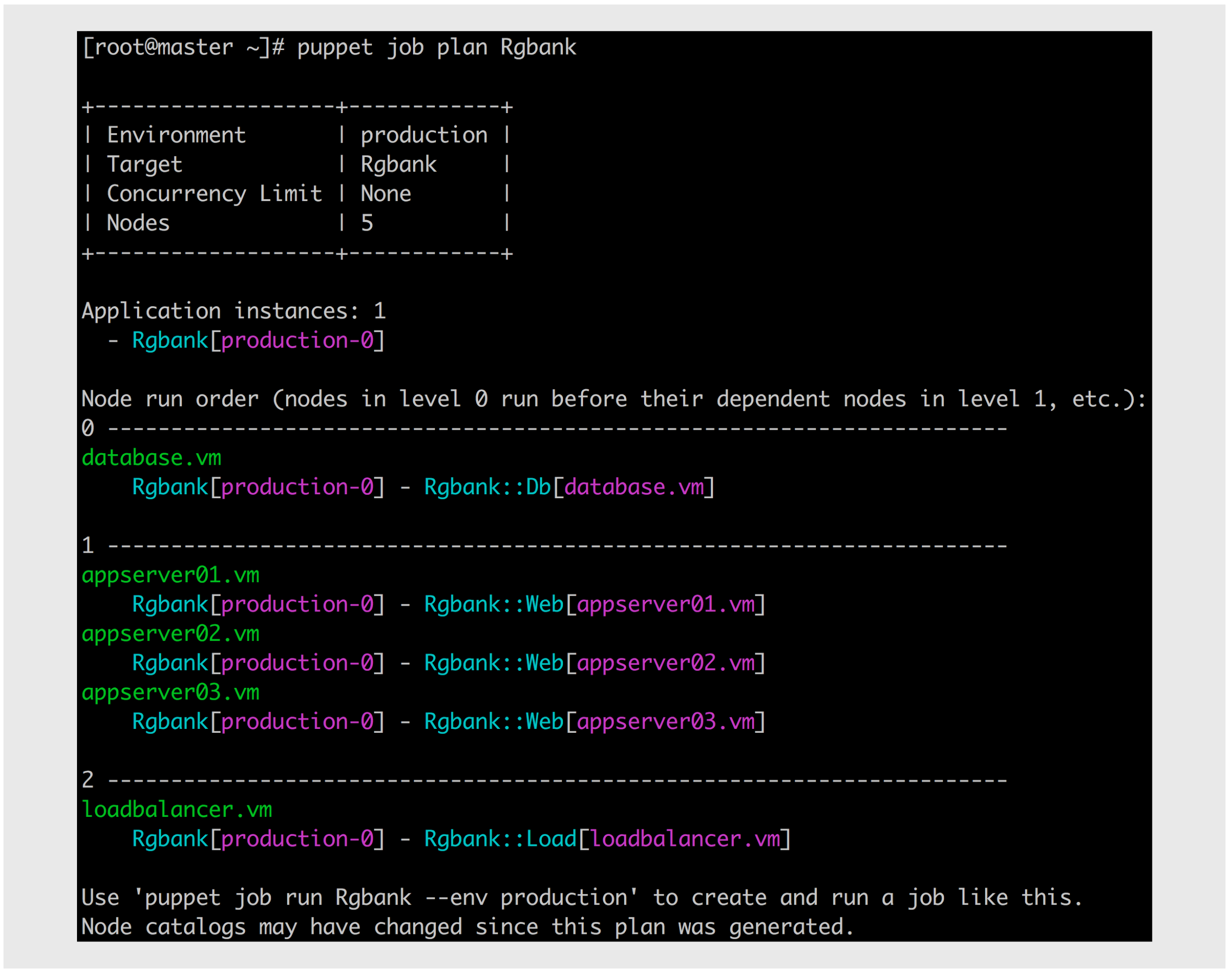
Property: ensure

Old value: purged

New value: present

Message: created

Puppet uses the information about dependencies between system configurations and infrastructure services to automatically determine the order in which everything should be managed. This dramatically simplifies how you need to think about your infrastructure: now you need to concern yourself only with the immediate services and configurations your piece of the puzzle depends on.



```
[root@master ~]# puppet job plan Rgbank

+-------------------+------------+
| Environment       | production |
| Target            | Rgbank     |
| Concurrency Limit | None       |
| Nodes             | 5          |
+-------------------+------------+

Application instances: 1
  - Rgbank[production-0]

Node run order (nodes in level 0 run before their dependent nodes in level 1, etc.):
0 -----------------------------------------------------------------------
database.vm
    Rgbank[production-0] - Rgbank::Db[database.vm]

1 -----------------------------------------------------------------------
appserver01.vm
    Rgbank[production-0] - Rgbank::Web[appserver01.vm]
appserver02.vm
    Rgbank[production-0] - Rgbank::Web[appserver02.vm]
appserver03.vm
    Rgbank[production-0] - Rgbank::Web[appserver03.vm]

2 -----------------------------------------------------------------------
loadbalancer.vm
    Rgbank[production-0] - Rgbank::Load[loadbalancer.vm]

Use 'puppet job run Rgbank --env production' to create and run a job like this.
Node catalogs may have changed since this plan was generated.
```

Single source of truth

You need a single source of truth that clearly outlines the way any given infrastructure component is managed to properly manage your infrastructure over time. With Puppet, you have just one code base per environment — and that code base is also your documentation, easily accessible and comprehended by anyone on your team. The precision of this executable documentation stands in sharp contrast to confusion inherent in the way other tools work: by employing a collection of playbooks, each running at various times across a subset of infrastructure systems, and often with multiple playbooks running at different times on the same system.

Continuously enforcing infrastructure state

It's never enough to just make an infrastructure change. You must be sure that it will remain as you set it until you're ready to change it again. That kind of confidence depends on multiple different capabilities coming together. You need one tool that can make a change; continuously monitor and enforce that change over time; report on the current state; and do all this at potentially huge scale, both in terms of numbers of systems and the amount of information collected.

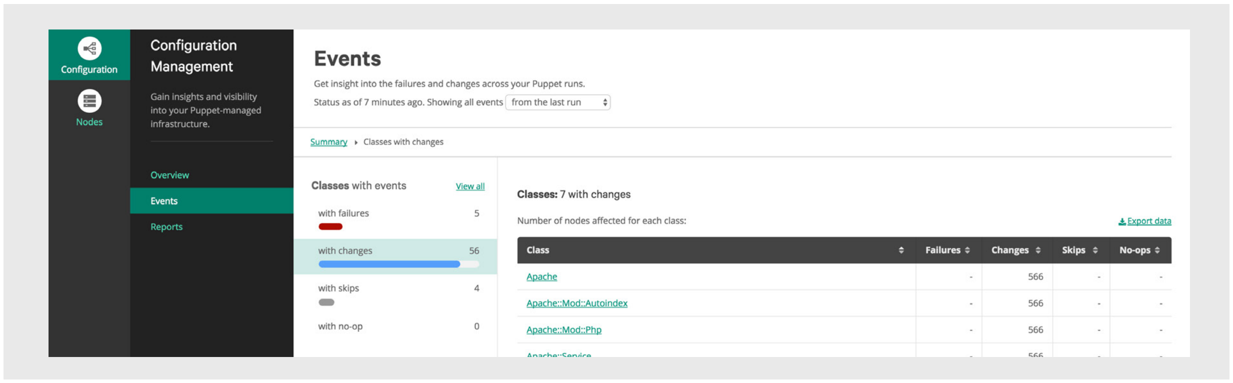
Enforcement over time

When you deploy something, you need to know it will remain in the state you deployed it in until you're ready for it to change. Puppet not only deploys new configurations and services, but continuously enforces their correct state. If configuration drift is detected, Puppet will automatically correct the drift, and report that the drift and its correction took place.

Monitoring and reporting

One of the great things about managing your infrastructure as code is the ability to replicate work across hundreds or thousands of systems without any increase in human effort. However, effective monitoring at this scale is critical. In addition to monitoring change, it's important to quickly identify where and why change occurred. You must continue to verify that systems remain in the state you left them in, so you can be confident that unauthorized changes have not occurred.

Puppet Enterprise includes powerful reporting tools, so you'll always know when a single configuration has changed across 1,000 systems, or when one system experiences 1,000 changes. Puppet Enterprise tells you at a glance, in real time, how many systems are in the desired state, which required correction, and how many are out of communication. Puppet Enterprise can inform you whether a change was caused by configuration drift or a change to the infrastructure code. This ability to report on infrastructure changes, the reasons for change, and the current state at any scale is unique to Puppet.



Gain insights and visibility into your Puppet-managed infrastructure.

Events

Get insight into the failures and changes across your Puppet runs.

Status as of 7 minutes ago. Showing all events from the last run

Summary ▸ Classes with changes

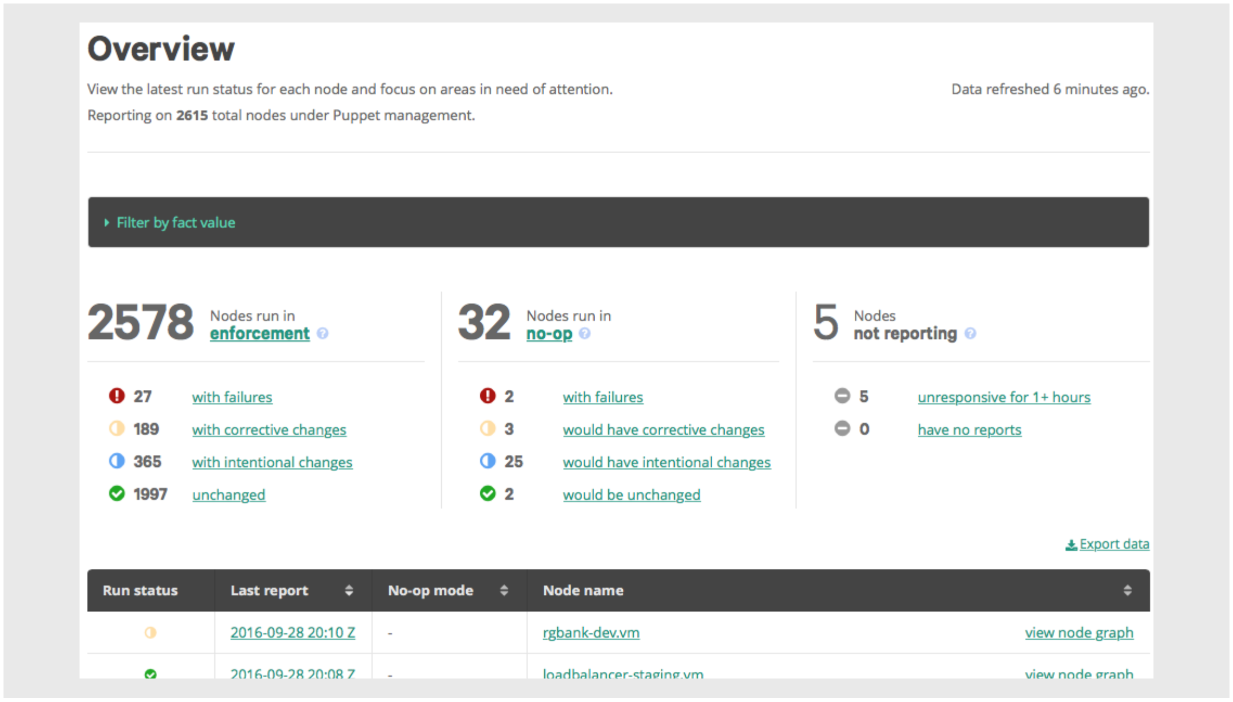
Classes with events

with failures	5
with changes	56
with skips	4
with no-op	0

Classes: 7 with changes

Number of nodes affected for each class:

Class	Failures	Changes	Skips	No-ops
Apache	-	566	-	-
Apache::Mod::Autoindex	-	566	-	-
Apache::Mod::Php	-	566	-	-
Apache::Service	-	566	-	-


Overview

View the latest run status for each node and focus on areas in need of attention. Data refreshed 6 minutes ago.

Reporting on **2615** total nodes under Puppet management.

Filter by fact value

2578	Nodes run in **enforcement**	**32**	Nodes run in **no-op**	**5**	Nodes **not reporting**		
27	with failures	2	with failures	5	unresponsive for 1+ hours		
189	with corrective changes	3	would have corrective changes	0	have no reports		
365	with intentional changes	25	would have intentional changes				
1997	unchanged	2	would be unchanged				

Export data

Run status	Last report	No-op mode	Node name	
◐	2016-09-28 20:10 Z	-	rgbank-dev.vm	view node graph
✓	2016-09-28 20:08 Z	-	loadbalancer-staging.vm	view node graph

Puppet Enterprise also tells you whether a system change was a corrective change or an intentional change. When Ansible or Puppet Enterprise make a change to a system, it's for one of two reasons: 1) The system's configuration was changed since its deployment by something other than the automation tool, or 2) the automation tool's instructions changed. We call these changes "corrective changes" vs "intentional changes."

Only Puppet Enterprise can distinguish between the two, informing you if a system change was required because someone or something changed the system since the last time Puppet checked the system's state, or if the system change was required because the Puppet code changed.

Infrastructure at scale

If you have a large number of systems to manage, you know how important it is for a tool to work well at scale. A single Puppet server can manage up to 30,000 systems. Customers like Walmart have successfully scaled Puppet Enterprise to manage more than 100,000 systems running a combination of Unix, Linux, and Windows.

Puppet's core strength is in how it's built

What makes Puppet unique is the way it works. No other tool generates anything like the Puppet graph, which details every configuration for a system, every service in an environment, and the relationships between all of them. This inherent grasp of infrastructure details and interdependencies gives Puppet a unique set of capabilities that are unmatched by any tool in the industry, including Ansible.

- Puppet is designed to be run continuously, making changes whenever drift is detected. That gives you the confidence of knowing your systems are always correctly configured.
- Puppet's domain specific language is powerful, yet puppet code is easy to read and understand. That makes it your executable documentation. You'll always be able to answer the question, "How is my environment configured?" and you'll never wonder what happened to the documentation. Puppet code is centralized so you know all the Puppet code applied to a system, as opposed to Ansible which may have multiple playbooks run against a single system.
- Puppet is model-driven and declarative. You define the end state you want your systems or applications to be in, using Puppet code, instead of listing out all the commands needed to get there. Because Puppet understands the complete model, Puppet detects desired state conflicts, and makes sure one team's code doesn't overwrite another team's code.
- Puppet gives you a true simulation mode built into the core of how Puppet works. This simulation mode, also called "no-op," allows you to preview any changes your code will make without any actual impact to your systems.
- Puppet gives you situational awareness. The Puppet graph gives you complete visibility into the status of your environments. Puppet understands the difference between a corrective change and an intentional change.
- Defining your security and compliance policies in Puppet code gives you full visibility into your compliance with your organization's policies, in real time. Puppet prevents configuration conflict, so one piece of Puppet code can't negate code earlier in a run. Plus, Puppet reports demonstrate that compliance automatically.

Operating mode

Puppet is a true configuration management solution, automating the initial deployment of your systems and continuing to manage them over time throughout the lifecycle of every system in your infrastructure. By default, Puppet agents retrieve and apply assigned configurations every 30 minutes. Each time an agent applies its configuration, it will first compare the desired state in your Puppet code to the actual state of the system.

Only when configuration drift is detected will the agent actually make changes to a given system. When no configuration drift is detected, Puppet simply reports that the current state of the system matches the desired state in your code.

This property of leaving correctly configured systems untouched is called idempotency, and it is core to how Puppet works. Because Puppet is idempotent, Puppet code can be repeatedly applied to a system without affecting the system's availability. Every time Puppet executes — whether it makes changes or not — it generates a report of all activities, showing when changes were made, and why. These reports can be used to prove compliance and to validate that security settings were applied at a given point in time. Puppet customers know what's going on in their infrastructure at all times, because Puppet provides the visibility and awareness you get when your desired configurations are continually enforced.

By contrast, Ansible is primarily focused on orchestration across a diverse set of systems. Where Puppet regularly enforces desired state, as well as executing infrastructure dependencies in the correct order, Ansible executes steps in the correct order without regularly enforcing desired state.

Puppet has two primary modes of operation:

1. A "self-healing" mode in which the Puppet agent regularly checks for configurations that have drifted out of their desired state, correcting any drift found, and enabling the system to self-heal.
2. A direct mode in which changes to the Puppet code are enforced on systems when you are ready.

Ansible has only a push mode, and therefore lacks the continuous self-healing capabilities that are core to the way Puppet works.

Where Puppet is fully idempotent, only portions of Ansible code are idempotent. This means that launching the same job twice on a target system can cause damage to your infrastructure, and because Ansible doesn't regularly enforce desired state, it can't automatically correct the problem. Ansible does provide reporting, but its reporting is not as comprehensive as Puppet's, so cannot provide the same degree of situational awareness.

Simulation mode

Simulation mode, also known as 'no-op' mode, is native to Puppet's operation. All configurations that you define within your Puppet code can *always* be run in a simulation mode. This allows you to evaluate your system's current state versus the desired state in your puppet code, and have Puppet report back the differences. This is extremely powerful functionality that gives you the capability to perform 'dry-runs' and know exactly what is going to happen to the target system before taking action. Customers find this functionality extremely useful in brownfield situations while attempting to bring an unmanaged system under Puppet control. You can gain confidence in the changes you're about to make without causing any impact to your target system.

Ansible does not have a simulation mode available for all tasks. Because Ansible allows you to launch tasks that are simply system commands, it has no way to predict or simulate the result of the actions you've defined. This results in lower confidence in the changes that will occur on a target system. For a greenfield environment this may be an acceptable risk, however once attempting to manage existing systems and application deployments the absence of this feature becomes challenging.

Model-based approach

Puppet does more than just automate a series of infrastructure processes; it is built to let you model your infrastructure. Then Puppet makes sure your infrastructure stays in the modeled state.

Ansible is designed largely around tasks. Typically, a playbook is created for each deployment task you want to automate. Within Ansible, there is no concept of a configuration model, and each playbook is its own island. This can lead to multiple playbooks managing the same configurations, ultimately causing deviation in your standards across systems.

Ansible doesn't provide the same guardrails you get when you automate with Puppet. Puppet's model-based approach ensures that a given configuration is managed only once, ensuring that a single desired state is enforced, versus the "last playbook run wins" effect of automation with Ansible. Puppet effectively and efficiently prevents unintentional configuration changes and ensures a consistent state throughout your infrastructure.

When you describe the complete configuration of your systems with Puppet, you start at the endpoint: What do you want your systems to look like? You don't need to list out all the processes to make something happen; you don't have to remember the right order. Puppet abstracts all that complexity away, leaving you to simply define the desired state for your systems and the relationships between configurations and infrastructure services.

You document in code how you want your infrastructure to work, and Puppet makes the correct configuration a reality. Puppet doesn't just manage all system configuration — it also continually enforces the desired state on a regular cadence.

This brings you a number of benefits. Puppet can detect desired state conflicts, and prevent them; detect logic errors and prevent execution of faulty logic; detect dependency cycles and fail appropriately for these; automatically apply the correct order of operations on your infrastructure; and handle failures gracefully, so infrastructure operations can proceed without a hitch.

Desired state conflict detection

The model Puppet generates for each system provides an intelligence that imperative tools do not have. For example, since Puppet compiles a list of every configuration for a system prior to doing any work on that system, Puppet knows if you're trying to manage the same configuration differently in different parts of your code base. We call this capability "desired state conflict detection."

When you don't have desired state conflict detection, different parts of the infrastructure code can silently configure the same configuration differently. You can't operate with the confidence that your systems are configured as you need them to be.

Logic error detection

Since Puppet reads all of your infrastructure code prior to configuring the system, logical errors can be caught early. Non-model-driven tools like Ansible find logical errors during configuration of the systems. If an error is found, the configuration run halts, leaving the system in a partially configured, partially known state. Again, you can't operate with full confidence in these conditions.

Dependency cycle detection

Puppet compiles the relationships between all configurations on any given system, and the dependencies between infrastructure services across systems. Puppet will detect if there is a dependency cycle between two or more configurations. This is very useful when working with large, complex infrastructure code bases.

In contrast, imperative tools like Ansible do not understand the relationships between configurations, and simply execute code in the order the tool reads it. This can cause problems when there are large number of Ansible playbooks exposing many roles, where calls between roles can cause an unexpected order of operations.

Automatic order of operations

Compiling the relationships between all configurations on a system enables Puppet to automatically determine the order of operations. This makes understanding the code base much simpler. When you're writing Puppet code, you need to focus only on the immediate dependencies of the configurations you're working with. Puppet will automatically figure out the order in which to configure the system, regardless of how many configurations are being managed.

Failure handling

Puppet handles failure automatically and gracefully. Because Puppet understands the relationships between all configurations (including the absence of relationship), when a failure occurs, Puppet automatically skips managing anything that depends on the failed configuration. At the same time, Puppet continues to manage any configurations that have no relationship to the failed configuration.

Testing

When you're writing code for automation or configuration management, it is extremely important to have a prescriptive way to test and validate the code's functionality. Puppet provides multiple tools for doing just that. Using tools like rspec-puppet, you can create full testing suites for any modules you write. This lets you validate proper functionality of the code and test future changes to that code.

Other Puppet tools such as Beaker let you move from unit testing with rspec to actual acceptance testing. During the process of writing Puppet code, you can spin up new nodes using Beaker, apply your Puppet code, and inspect the resulting system to ensure your configurations or applications are being set up properly. You can also include Puppet code development in a full continuous integration and continuous delivery workflow, giving you complete confidence in the quality and functionality of any new or changed configurations you release to your infrastructure.

When you use Puppet Enterprise, you get additional tools like Code Manager that let you directly integrate your version control system to Puppet. Now, as new code is written and merged, it's immediately available for application to your infrastructure.

Ansible does not provide this same level of testing capabilities, so cannot provide the same level of confidence in the code you're using to configure or deploy your infrastructure.

Configuration data management

While most of your infrastructure can be managed as code, there is some information that doesn't make sense to hard-code. Usually this is information that can vary between sites, environments, operating systems, etc. For working with this kind of variable information, Puppet Enterprise includes a configuration data management tool called Hiera.

Hiera is a hierarchical key/value pair management system that enables you to assign configuration values, segmented by infrastructure criteria. Hiera makes it easy to assign NTP servers per site, or application build versions per operating environment, without modifying the logic in your Puppet code. This makes Puppet code highly reusable.

Ansible uses global variables assigned to playbooks in Tower, but has no equivalent hierarchical data store such as Hiera. Assigning data to different segments of your infrastructure is much easier to manage with Puppet.

In addition to Hiera, Puppet Enterprise's web UI provides role-based access control, so you can define parameters that other people in your organization are allowed to manage. This lets teams outside of IT tweak configurations in a safe manner, self-serving instead of having to rely on IT for every configuration.

Questions to answer

As you consider which are the right tools for your organization, it helps to ask some questions about how your organization is running now and where you think you are going. The questions below should help you in your decision making.

How many contributors to the infrastructure code base will we have?

If the answer is more than a few, you need to plan how you're going to scale everyone's contributions to the infrastructure code base. You will want to make sure your tool of choice can:

- Prevent a single configuration from being managed multiple ways by different contributors.
- Integrate with existing pipelines and testing practices.
- Be easily approachable (i.e., easy to read and learn) from a diverse set of background disciplines.

What are our audit requirements?

If you need to be able to prove the infrastructure is always in its desired state, your new tool has to be great at both managing state over time *and* continuously reporting when systems are in the desired state; when drift is detected and remediated; and when new infrastructure changes are deployed. Compliance audits go much more smoothly when you can audit infrastructure changes — including change approvals — in the infrastructure code base itself.

Do we already know everything we need to manage?

When bringing brownfield systems under management, you often don't know on Day One exactly what you need to manage. So it's important to choose a tool that can manage anything from kernel parameters to cloud environments, and that can introspect the current configuration of any system you want to bring under management.

How many nodes do we eventually want to manage?

Scale should, of course, be a consideration from the beginning. It's important to have a tool that will handle everything you might eventually throw at it. Using Puppet's direct control features, teams can continuously enforce the configuration of up to 30,000 nodes from a single Puppet server. Companies like Walmart have scaled Puppet Enterprise to 100,000 nodes or more.

How complex are our deployments?

Many times, the complexity of a deployment is in managing the relationships between systems, rather than in managing the same system through multiple states. Puppet's application orchestration capability builds an environment model to understand the complex web of relationships between application services. Puppet uses this understanding to intelligently determine things like order, concurrency, and how to pass configuration information between dependent services. This frees you from managing the complexities of the deployment process so you can focus just on infrastructure services and their immediate dependencies.

Puppet <-> Ansible Glossary

Puppet Enterprise	Ansible
Module	Role
Resource type	Module
Resource provider	N/A (Ansible doesn't have this concept)
Manifest	Playbook
Forge	Galaxy

We want to hear from you

We hope we've helped you understand the differences between Ansible and Puppet Enterprise, and that we've given you some good questions you and your team can ask yourselves to determine which solution is right for you. If you have any concerns or questions this document didn't answer, or if you find anything in this document is inaccurate, please reach out to your Puppet representative. We will be happy to help.