Puppet Enterprise vs Ansible Tower vs Chef Automate

Comparison table	Puppet Enterprise	Ansible Tower	Chef Automate
Product description	Puppet is an infrastructure automation platform that uses the Puppet domain specific language (DSL) to continuously enforce the state of configurations. Puppet also includes Tasks for performing ad hoc and one-off tasks that can be written in any language (i.e. Ruby, Python, Bash etc.). Puppet Enterprise includes all the capabilities of Puppet Platform 6 plus a number of enterprise-only features like PE console (the user interface), RBAC, Code Manager and more. Puppet's offerings also include Continuous Delivery for Puppet Enterprise, a CI/CD platform to automatically test and deploy Puppet code.	Red Hat's commercial Ansible offering is Ansible Tower, an infrastructure automation platform based on Ansible's open source configuration management tool. Ansible and Ansible Tower work by running orchestration jobs called playbooks (written in Yaml) which invoke Ansible modules to perform specific actions on target systems. Ansible Tower includes job scheduling and inventory discovery (which is comparable to PuppetDB / Puppet Facts).	Chef Automate uses a pure Ruby domain specific language (DSL) for writing system configuration called "cookbooks" that continuously enforce the desired state of a system. Chef Automate includes: Chef for infrastructure automation, Habitat for application automation and InSpec for compliance automation. InSpec is Chef's open source language for describing security & compliance rules that can be shared between software engineers, operations, and security engineers.

	Puppet Enterprise	Ansible Tower	Chef Automate
Continuous drift remediation	Yes	Yes	Yes
Report on intentional vs. corrective changes	Yes	No	No
Discover unmanaged packages and bring them under management	Yes	Yes	No Not by default, but there is a plugin for Ohai that collects package information for inventory purposes.
Job scheduling	Yes	Yes	No
Orchestration	Yes	Yes	Yes
Language	• The Puppet DSL is used to enforce the desired state of configurations; • Any supported language on the target system (ie. Ruby, Python, PowerShell, etc.) can be used to write Tasks for the execution of one-off, ad hoc tasks.	Playbooks are written in YAML. Modules can be written in any language supported on the target system.	Chef uses a Ruby DSL (a DSL based on Ruby). The Recipe DSL is a Ruby DSL that is primarily used to declare resources from within a recipe.

	Puppet Enterprise	Ansible Tower	Chef Automate
Agent-based	**Yes** (recommended for desired state enforcement, optional for tasks)	**No** Not by default, but there is a plugin for Ohai that collects package information for inventory purposes.	**Yes** (with the exception of InSpec that is based on an agentless mode)
Agentless remote tasks	**Yes**	**Yes**	**No**

Approach to configuration management: model-driven vs. task-based approach	Puppet's platform is based on a model-driven approach, and also offers a task-based approach.
	
	An example of model-driven approach:
	`package {'postgresql':`
	`  ensure => latest,`
	`}`
	
	`service {'postgresql':`
	` ensure => running,`
	`}`
		Ansible's platform is task-based by default, but it also allows users to adopt a model-driven approach
		
		An example of model-driven approach:
		tasks:
		 - name: ensure postgresql is at the latest version
		   yum:
		     name: postgresql
		     state: latest
		 - name: ensure that postgresql is started
		   service:
		     name: postgresql
		     state: started
			Chef's platform is based on model-driven or desired state enforcement approach only.
			
			An example of model-driven approach:
			package 'postgresql' **do**
			  action [**:start**]
			end
			
			service 'postgresql' **do**
			  action [:upgrade]
			end

	Puppet Enterprise	Ansible Tower	Chef Automate

Resource Abstraction Layer (RAL)	**Yes**
	Puppet code is executable documentation that can be used to easily interact with the RAL.
	
	`puppet resource user root`
	`user { ‘root’:`
	`  ensure   => ‘present’,`
	`  comment  => ‘System Administrator’,`
	`  gid      => 0,`
	`  groups   => [‘admin’, ‘certusers’, ‘daemon’, ‘kmem’, ‘operator’, ‘procmod’, ‘procview’, ‘staff’, ‘sys’, ‘tty’, ‘wheel’],`
	`  home     => ‘/var/root’,`
	`  password => ‘*’,`
	`  shell    => ‘/bin/sh’,`
	`  uid      => 0,`
	`}`
		No	**Yes**
			Chef Base Resources like "packages" are abstractions of platform-specific resource types. Chef doesn’t offer a CLI for easy access to all resource types.

	Puppet Enterprise	Ansible Tower	Chef Automate
Syntax checks	A Puppet manifest gets compiled into a catalog before execution so any syntax errors are identified beforehand. A manifest with errors will not be executed.	Ansible doesn't automatically check module invocation before execution. The playbook just stops when it encounters an error. In case a syntax error occurs the playbook is only partially executed.	Chef takes advantage of the Ruby ecosystem and has linters and parser validation for syntax checks.
Windows support	Windows is 100% supported, and Puppet DSL works across most OSes. In particular, Puppet Tasks supports task execution over WinRM.	Modules are OS dependent, and Windows requires different modules from modules used for other platforms. Execution over WinRM supported.	Windows is 100% supported, although the tool doesn't have support for execution over WinRM.
Inventory service	**Yes** The inventory service is PuppetDB and Puppet Facter. PuppetDB is populated during Puppet runs and can be queried via CLI using PQL (Puppet Query Language) or an API. Inventory info includes hosts, facts, managed and unmanaged packages and managed resources.	**Yes** The inventory service is included with Ansible Tower. The user can define inventories which are automatically synchronized with platforms like major public clouds, VMware and OpenStack	**Yes** Ohai detects data about your operating system. It can be used standalone, but its primary purpose is to provide node data to Chef. Ohai will print out a JSON data blob for all the known data about your system.

	Puppet Enterprise	Ansible Tower	Chef Automate
CI/CD tool to build, test and promote quality infrastructure as code, faster	**Yes** Puppet's offering includes Continuous Delivery for Puppet Enterprise that provides Puppet teams with a continuous delivery tool purposely built to make it easier and faster to deploy quality Puppet code.	**No** Although Ansible doesn't have a dedicated CI/CD solution, it integrates well with existing CI/CD solutions customers already have in use.	**Yes** Test Kitchen is a tool for testing cookbooks across any combination of platforms and test suites.
Supported CIS Benchmarks	**No** Puppet only has unsupported or third party modules available on the Forge.	**No** Ansible Galaxy and GitHub contain open source unsupported CIS benchmark roles / playbooks.	**Yes** A number of InSpec compliance profiles are part of Chef Automate.
ServiceNow integration	ServiceNow has created a Puppet plugin which uses the documented Puppet Enterprise APIs. This integration is provided by ServiceNow and not supported by Puppet. See the documentation here. Puppet can also provide customers with a custom-built integration using APIs or they can engage with one of our partners who have written one, such as AHEAD. Community modules exist which support some form of ServiceNow integration.	ServiceNow is not a featured integration on Ansible's website. Community modules exist supporting some form of ServiceNow integration.	ServiceNow has created a Chef plugin which uses the documented Chef APIs to integrate with it.

	Puppet Enterprise	Ansible Tower	Chef Automate
Splunk integration	Puppet has several native integrations with Splunk, including modules: • Splunk module for Puppet Enterprise • Splunk Alert Action for Puppet Enterprise • Splunk App and Add-On for Puppet Enterprise • Splunk IT Service Intelligence Module for Puppet Enterprise	Some integration. More details at: **https://www.ansible.com/integrations/devops-tools/splunk**	Some integration. More details at: **https://docs.chef.io/analytics_splunk.html**
Cloud support offering	On the Puppet Forge there are supported and auto-generated modules for AWS and Azure. AWS OpsWorks includes Puppet Enterprise, and Puppet Enterprise is also available on the AWS Marketplace with a BYOL (Bring Your Own License) option.	Ansible offers a library of cloud supported modules. For more info check out this page: **https://www.ansible.com/integrations/cloud** Ansible is also available on the AWS Marketplace, but is considered a competitor of AWS OpsWorks.	Chef offers AWS, GCE and Azure cookbooks. Chef Automate is available on the AWS Marketplace and is fully integrated within AWS OpsWorks.

Feature description and use cases

Continuous drift remediation

The typical use case for this feature is "infrastructure state enforcement," which helps prevent incidents and downtime due to manual changes and errors.

Report on intentional vs. corrective changes

The ability to easily get a report that shows what changes have been made intentionally by system administrators and what changes are corrective and due to errors and unauthorized changes is very useful, especially for compliance purposes.

Discover unmanaged packages and bring them under management

This capability enables the scaling of automation across the entire infrastructure. It also supports the use case of enforcing consistency across the entire infrastructure. For instance, you can easily check that you have the same package version across the whole infrastructure.

Job scheduling

Run tasks outside business hours. This capability enables the business to schedule one-off changes or repetitive ad hoc single changes.

Orchestration

This capability supports the use case of making changes on resources with dependencies that need changes to be made in a certain order. For instance, if you want to release a new version of an app that has lots of visitors and runs in production on a few instances, first you will need to take out one instance of a load balancer, upgrade that instance and put it back into the load balancer. Then you repeat the same process for the other instances, until all instances are upgraded. Orchestration allows you to do all those actions in the correct sequence, considering the dependencies.

Language

Puppet has a domain specific language (DSL) which brings pros and cons. The pro is that everyone can read Puppet language making it easier for other teams who don't know Puppet to read the reports. The con is that Puppet code has to be learned and it's not a language that everyone in Operations or Development will already know.

Note that:

- The Puppet DSL is an easy to read and understand declarative language to define and enforce the desired state of configurations.
- Puppet tasks can be written in any language including Bash, PowerShell, Python and Ruby.
- Ansible playbooks have to be written in a YAML format.
- Chef recipes are written using the imperative Chef DSL which resembles Ruby.

Agent-based & agentless

There are two architectural modes of a configuration management system: agent-based and agentless. Let's have a look at how those two modes bring benefits to some use cases more than others.

Agent-based use cases

Here you will find two examples of use cases where the agent-based architecture is the best choice.

1) State enforcement and task orchestration at scale

One advantage of the agent-based model for this use case is the scalability of it. Since the agent has administrative permission, it can run the configuration job without requiring extra credentials, which means you don't need to distribute credentials to every system. Let's assume you want to manage 100,000 servers centrally with Puppet. In this instance, an agent-based approach is more scalable because you don't need to create a central point which owns credentials for 100,000 servers. Furthermore if an agent can't access the master, because of network problems, it can still run Puppet with the last configuration settings present locally on the agent, which means it can still perform something like drift remediation.

The agent-based model also has the benefit of allowing you to orchestrate tasks across all the servers in short time. Also there is no limit to the number of machines that can be managed with an agent-based approach.

2) Self-service portal

Agent-based architecture has great benefits when you want to have an automatic and seamless configuration workflow for a self-service portal. When you provision systems (for instance using VRa/VRo) you can pre-install the agent on the system and also configure the role of the system (you can supply extra info to the agent with trusted facts). This makes the system provisioning faster and gives you control over day 2 operations too.

Agentless use cases

Here you will find two examples of use cases where the agentless architecture is the best choice.

Config management approach: model-driven vs. task approach

The model-driven approach is the best option for use cases such as compliance enforcement and drift remediation. The main reason for the suitability of the model-driven approach to those use cases is its idempotency. From a RESTful service standpoint, for an operation (or service call) to be idempotent, clients can make that same call repeatedly while producing the same result. In other words, making multiple identical requests has the same effect as making a single request. Note that while idempotent operations produce the same result on the server (no side effects), the response itself may not be the same (e.g. a resource's state may change between requests). Puppet and Chef are natively idempotent, while with Ansible you need to build your code to be idempotent.

Here's an interesting discussion on idempotence with Ansible: https://github.com/ansible/ansible/issues/35293

The task approach is the best option for use cases like performing one-off change (or an orchestrated series of changes).

1) Managing devices that don't allow the installation of an agent

If you want to manage network devices where you can't or don't want to install an agent (in some organizations agents can't be installed for security reasons), the agentless mode allows you to manage those devices like any other server but without any overhead.

2) One-off, ad hoc tasks

When you need to run a one-off single task on a machine or group of machines.

3) You need a simple and fast way to automate your infrastructure

If you want an easy and fast way to automate your infrastructure, the best option is going for an agentless product like Puppet Bolt or Ansible that don't require the installation of an agent to automate the management of your systems.

A question that we get frequently asked by our customers is: **How much scalability will I get with an agentless approach?** The number of machines you can manage simultaneously in an agentless mode depends on the power of the workstation/laptop from where you are running tasks and will always be limited to the maximum number of network connections supported by that workstation.

Resource Abstraction Layer

This feature allows you to use the same resource type (for example "package") across multiple platforms or providers. This way, resource declaration looks like this:

```
package { "mypackage":
     ensure => installed,
}
```

This code will work on all supported platforms. The provider layer will "translate" the abstract resource type "package" into operating system specific implementation, such as yum or deb on Linux and MSI or Chocolatey on Windows.

This allows you to model your configuration across heterogeneous environments (across different OSes) using the same code.

Syntax checks

This capability refers to the ability of the automation software to check for syntax errors before executing a manifest, and to not execute a manifest with syntax errors.

Windows support

Here you can find some of the frequently asked questions about automation software for Windows.

Why it is useful to use configuration management tools together with Windows tools like PowerShell DSC and SCCM?

DSC is a configuration language (Desired State Configuration) for Windows only. While PowerShell DSC enables the automation of Windows configurations, you still need a tool that allows you to deploy DSC scripts to target systems, test them, run them and report on the results. Puppet delivers all of that.

SCCM (System Center Configuration Manager) is a GUI (not infrastructure as code) that allows the user to orchestrate patch management, desktop client management, initial server configuration, licence reporting and asset tracking. Using a configuration management tool helps especially when the tool works across all major platforms and it makes sense in terms of tool rationalization.

Can Puppet do patching on Windows instead of SCCM?

Puppet alone can't replace SCCM for patching, but Puppet in combination with WSUS can be used to configure a WSUS client to patch systems. If you use Puppet and WSUS, you don't need to use SCCM for patching.

Inventory service

The inventory service generates lists of managed resources, packages, users, etc. useful for building reports for executives or security and auditing purposes, or to understand how to progress with system automation. For instance, if I want to install a new version of a package on all systems that have that package installed, I need to know where the package is installed. That's an example of how the inventory service can be useful.

CI/CD tools

CI/CD tools help you promote your code faster. Just like using a CI/CD tool to deploy applications makes you faster and allows you to achieve higher code quality in the deployed application, a CI/CD tool can also help you build, test and promote your infrastructure as code faster and make your systems more reliable.