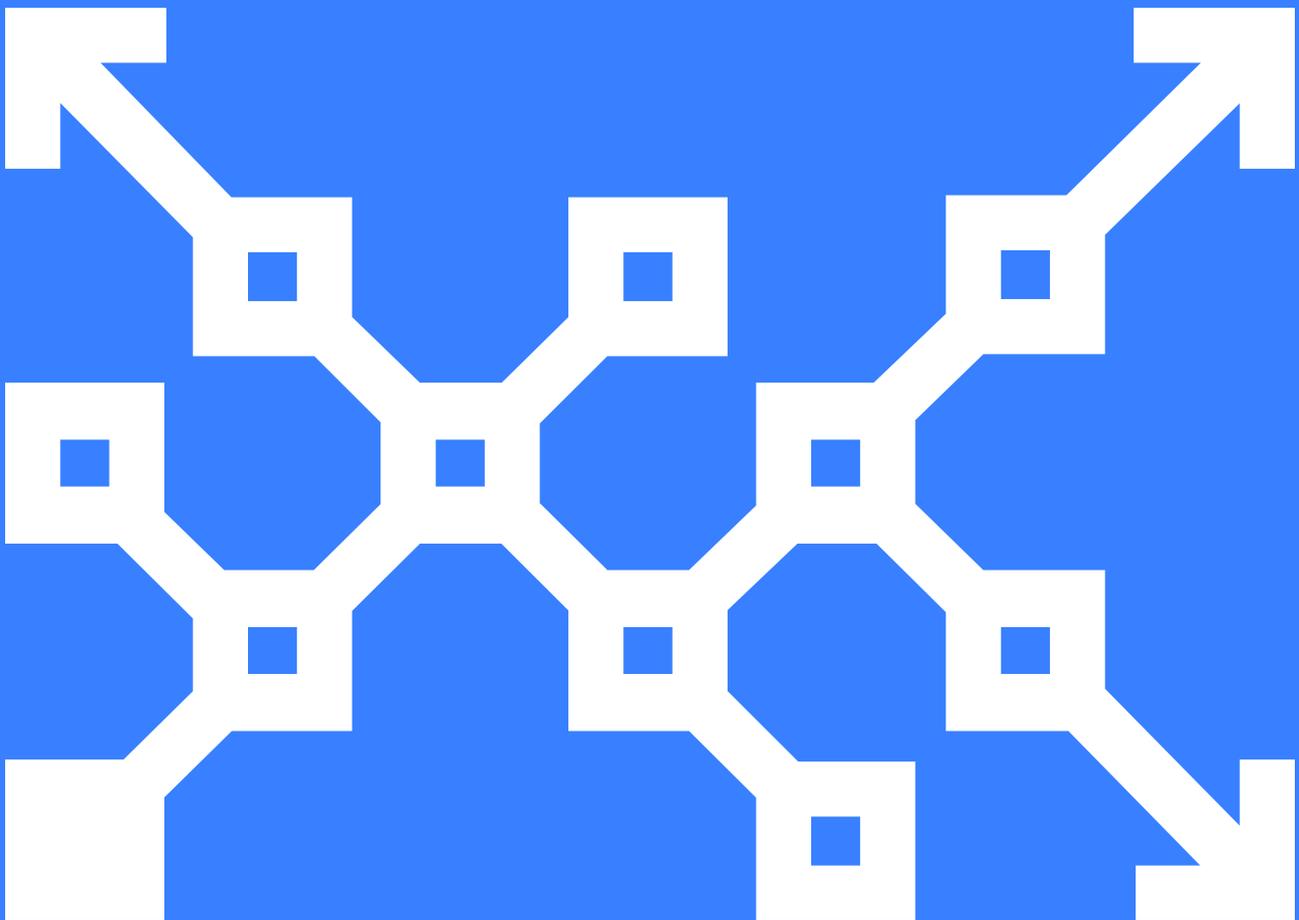


# Puppet Enterprise vs Chef Automate



# Contents

- 3 [Summary](#)
- 4 [Comparison: Puppet Enterprise vs. Chef Automate](#)
  - Organizational scale
- 5 [Change validation](#)
  - Time to value
- 6 [Application & infrastructure orchestration](#)
  - Infrastructure scale
- 7 [Windows support](#)
  - Continuous delivery
- 8 [Supported Content](#)
- 9 [Organizational Scale](#)
- 10 [Learning Curve](#)
- 11 [Intelligent Orchestration](#)
  - Environment and Node Modeling
- 12 [Continuous Delivery](#)
- 13 [Orchestration Control](#)
- 14 [How Puppet Works vs How Chef Works](#)
  - The Puppet process
- 16 [The Chef process](#)
- 17 [We want to hear from you](#)



# Summary

This document is intended to outline the main differences between Chef's enterprise product, Chef Automate, and Puppet's enterprise product, Puppet Enterprise. The first part of the document will discuss the business use cases of Puppet Enterprise versus Chef Automate, with the technical differences explained later in the document.

Chef Automate and Puppet Enterprise cover similar use cases, but with different approaches. This document will give you the information you need to decide which solution is right for you.

# Comparison: Puppet Enterprise vs. Chef Automate

Business need	Puppet Enterprise	Chef Automate
<p><b>Organizational scale</b></p> <ul style="list-style-type: none"> <li>• <b>What it is:</b> The ability of multiple IT teams to successfully contribute to, collaborate with, and benefit from an infrastructure automation solution.</li> <li>• <b>Why it matters:</b> Larger IT organizations often have many individual contributors and specialized teams working together to manage 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.</li> </ul>	<p style="text-align: center;"></p> <p>Puppet’s DSL is easily adopted by non-developers. Puppet’s built in intelligence helps multiple teams safely contribute to a growing infrastructure code base. For example, Puppet Enterprise features configuration conflict detection, preventing two pieces of Puppet code — potentially written by different teams — from managing the same configuration differently.</p> <p>Puppet Enterprise also includes the node graph, which shows a node’s resulting configuration model so teams can quickly understand how all the Puppet code comes together.</p>	<p style="text-align: center;">—</p> <p>Large Chef code bases are difficult for multiple teams to manage. Different Chef cookbooks can manage the same configuration in different ways. This means that you must run automated infrastructure testing in order to make sure that any node managed by several cookbooks is actually being managed correctly.</p> <p>The expertise required to be successful is more than most IT teams have on Day One of an infrastructure automation initiative.</p>

## Change validation

- **What it is:** Ensuring a change to a code base will have its intended effect, and no unintended side effects.
- **Why it matters:** Changes to logic that figures out what a configuration should be do not always perform as intended, and often (unintentionally) affect configurations that are outside the scope for the change. Unintended misconfigurations are the leading cause of service outages.



Puppet includes a simulation mode (no-op) built into the core of how Puppet works. This lets you do a test flight of changes before actually deploying them to ensure desired results will be achieved. It's not a feature of Puppet code. Any and all Puppet code supports no-op.

Puppet also provides Beaker, a Puppet code acceptance testing tool. Beaker is not necessary, however, to successfully validate Puppet code prior to node enforcement.

Chef has a “dry run” mode, but its ability to work depends on how the recipes in a cookbook are written. It is not built into the core of how Chef works. If a cookbook from the Chef Supermarket, for example, uses arbitrary Ruby code rather than sticking to Chef's Ruby DSL, then dry-run mode will not work and simulating changes will not be possible.

Chef Automate includes InSpec for performing automated testing and validation of Chef code, but requires software engineering skills to be used effectively.

## Time to value

- **What it is:** How quickly your team can learn the tool and begin using it for meaningful work.
- **Why it matters:** The more quickly 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 DSL is designed for simplicity. Rather than a programming language like Ruby, Puppet is a custom language designed to be picked up easily by people without a background in programming.

Puppet has a larger collection of pre-built modules of configuration code on the **Puppet Forge**, which means you can start automating more workloads on day one without needing to write code from scratch.

Chef requires development skills and a familiarity with Ruby, limiting adoption due to its high learning curve. Some organizations end up with workflows where operations teams are dependent on a dedicated development team to write code to their specifications, adding needless delay to the production lifecycle.

Chef has less pre-built content available than Puppet.

## Application & infrastructure orchestration

- **What it is:** Automatic assurance that changes are made across multiple nodes in an ordered, predetermined way.
- **Why it matters:** As IT teams manage more than just core infrastructure configurations and middleware with automation, the ability to make orchestrated change is critical to using automation to its full potential.



Puppet Enterprise includes application modeling: Puppet code models each distributed service that makes up the application, as well as each service's underlying infrastructure requirements.

Puppet Enterprise builds an environment graph of every application, every infrastructure service, and all infrastructure dependencies to intelligently determine the correct order of operations and cross-node information sharing. Plus, Puppet Enterprise is the single point of visibility and control.



Chef Automate provides a way for a pipeline project to have dependencies on other pipeline projects. To manage applications made up of several distributed services, each service needs to have its own pipeline project, and each project has to define its own immediate project dependencies. The complexity required to manage distributed applications and interdependent infrastructure services is difficult to maintain, and creates a high bar for IT organizations embarking on an automation initiative.

## Infrastructure scale

- **What it is:** Using a management tool to manage an ever larger number of nodes.
- **Why it matters:** The growth of things to manage will never decrease. That's true for servers, VMs, containers, and infrastructure services. In fact, containers and microservices dramatically increase the moving pieces that need to be managed. Having a tool that can manage the ever-increasing number nodes is critical for scaling your infrastructure.



A single Puppet Enterprise server can manage up to 30,000 nodes using **Direct Puppet** and up to 1,600 nodes in legacy enforcement mode. Adding additional compilation masters enables you to rapidly and reliably scale further as needed.



Because Chef Server does not compile environment and node graphs, a single Chef server can serve tens of thousands of nodes.

<h3>Windows support</h3> <ul style="list-style-type: none"> <li>• <b>What it is:</b> Support for Microsoft Windows operating systems.</li> <li>• <b>Why it matters:</b> Most IT organizations have a diverse set of operating systems with Microsoft operating systems almost always being one of them. Each operating system requires their own unique processes and point solutions. Having a tool that manages all your operating systems, including Windows, minimizes tool proliferation and reduces or removes siloed teams and processes.</li> </ul>	<p style="text-align: center;"></p> <p>Puppet Enterprise has extensive support for Microsoft Windows Server, 7, and Vista.</p> <p>In addition, Puppet Enterprise offers commercially supported modules to manage nearly every aspect of Windows from <b>Registry Keys</b> and <b>DSC resources</b> to <b>SQL Server</b> and <b>IIS</b>.</p>	<p style="text-align: center;"></p> <p>Chef Automate has extensive support for Microsoft Windows Server, 7, and Vista.</p>
<h3>Continuous delivery</h3> <ul style="list-style-type: none"> <li>• <b>What it is:</b> The ability to deploy each change at any time that the business is ready for it.</li> <li>• <b>Why it matters:</b> Continuous delivery improves an organization's ability to serve customers by lowering deployment risk, increasing deployment cadence, and decreasing cycle times. All of these improve time to market, market learning, ability to take risks, and ability to innovate.</li> </ul>	<p style="text-align: center;"></p> <p>Puppet Enterprise takes a partner and ecosystem approach, providing robust APIs and integrations with CD platforms such as <b>Jenkins</b>, <b>Bitbucket Pipeline</b>, and more. This enables you to leverage existing investments in CI/CD tooling and scale existing expertise.</p>	<p style="text-align: center;"></p> <p>Chef Automate includes the pipeline management features in the product itself.</p> <p>If you already use a CI/CD system, and add Chef Automate, you'll have pipeline tool sprawl, which silos expertise and sources of truth.</p> <p>All of Chef's orchestration features work through the pipeline features, so you must initially adopt continuous delivery in order to perform orchestration – something most IT organizations are not prepared for on Day One of their automation initiative.</p>

## Supported Content

- **What it is:** Pre-built downloadable infrastructure code that helps you quickly adopt and manage common technologies.
- **Why it matters:** Having pre-built content (Puppet modules and Chef cookbooks) available to download and manage common technologies rapidly increases the speed at which new software can be adopted, and existing infrastructure can be brought under management. In addition, when a problem arises with content you depend on, you need to know you can get help quickly.



Puppet not only provides commercial support with a SLA for many **Puppet Forge modules**, but also reviews and approves of **popular community modules**.

In addition, Puppet develops and supports several modules exclusively available to Puppet Enterprise customers.



Chef offers “best effort” support for a collection of cookbooks, with no SLA.

# Organizational Scale

Larger IT organizations often have many individual contributors and specialized teams that work together to manage 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 causing — siloed thinking, turf wars, and disjointed processes.

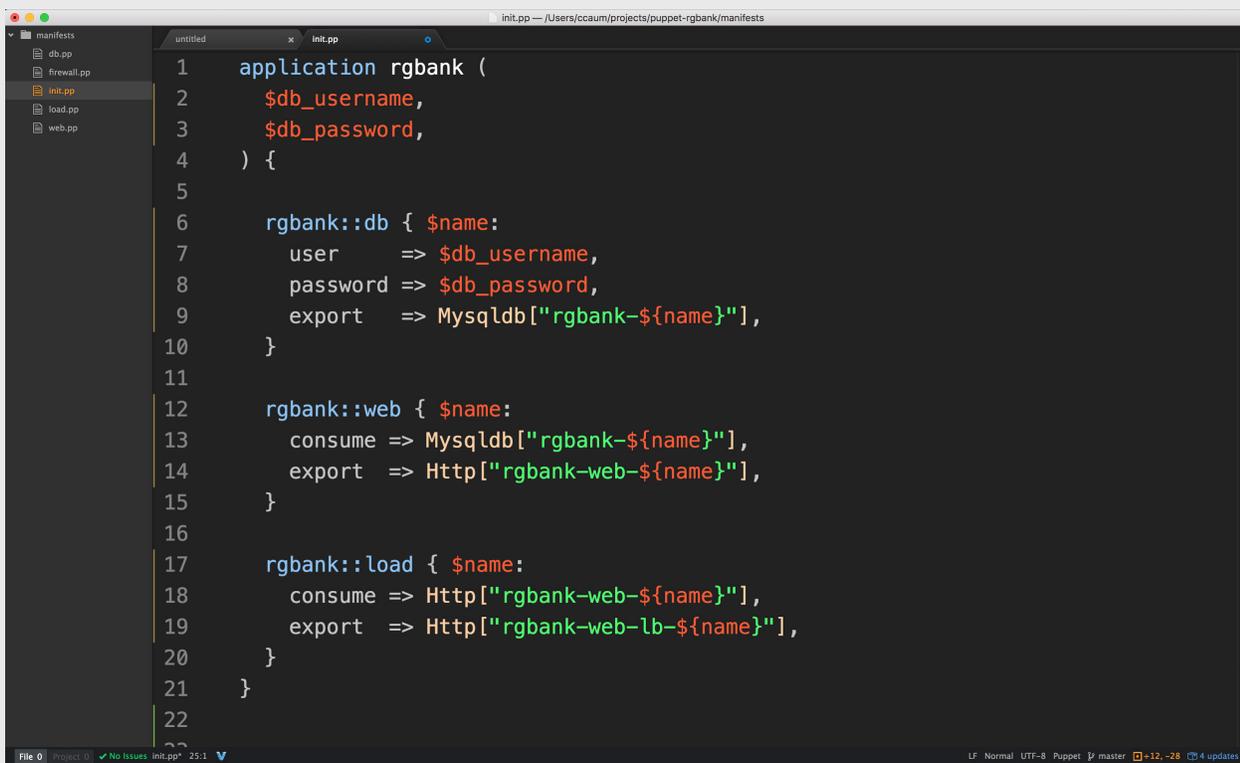
## The tool you use needs to address several challenges:

- **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 node.
- **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 makes 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, and 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.

## Learning Curve

Puppet's simple configuration domain-specific language (DSL) was designed to be easily read and written by people without a background in software development. Despite the language's simplicity, it features all the necessary features for defining modern configuration models such as loops, templates, and data validation.

Unlike Chef, Puppet does not require any experience with Ruby. If someone understands Bash, they can generally pick up Puppet.



```
1 application rgbank (
2     $db_username,
3     $db_password,
4 ) {
5
6     rgbank::db { $name:
7         user    => $db_username,
8         password => $db_password,
9         export  => Mysqldb["rgbank-{$name}"],
10    }
11
12    rgbank::web { $name:
13        consume => Mysqldb["rgbank-{$name}"],
14        export  => Http["rgbank-web-{$name}"],
15    }
16
17    rgbank::load { $name:
18        consume => Http["rgbank-web-{$name}"],
19        export  => Http["rgbank-web-lb-{$name}"],
20    }
21 }
22
```

# Intelligent Orchestration

Puppet builds a model of every configuration on every node for an entire environment, plus every infrastructure service distributed across multiple nodes within an environment. The model is used to intelligently determine the correct order of operations, what information needs to be shared between different services (credentials, locations, ports, etc.), and can show how all of this will be done so you can review prior to the actual orchestration.

## Environment and Node Modeling

When you're managing hundreds or thousands of configurations on a single node, or managing hundreds of different services across a multitude of business applications, it's paramount that you understand how all the pieces relate. This is especially true when different teams manage different application services, and different parts of the node'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 node to generate a complete model for how the node should look. Puppet Enterprise shows you the resulting model in an interactive graph so you can see how each piece of infrastructure code contributes to a node's desired state. The black box is opened, and everyone who deals with infrastructure can see the state of things, including all the dependencies.

The screenshot displays the Puppet Enterprise interface for a node configuration. At the top, it shows the run status as 'with corrective changes', the node run timestamp as '2016-11-23 22:13 Z', and the config version as '1479918601'. A filter is set to 'resource is Package[openssl]'. The main area shows a dependency graph with nodes like 'Package[openssl]', 'Package[ca-certificates]', 'Package[rubygems]', 'Package[ruby]', 'Vhost[dev\_web\_bfs]', and 'Package[ntp]'. The 'Package[openssl]' node is highlighted, and a detailed view is shown on the right. This view includes the class containment view, event timestamp (2016-11-29 22:15:12.182 Z), event status (corrective change), file path (/etc/puppetlabs/code/environments/production/modules/openssl/manifests/packages.pp), line number (5), tags (package, class, profile, openssl, ssl, profile:ssl, openssl:packages, default, node, packages), and event details (Property: ensure, Old value: 1:1.0.1e-42.e17.9, New value: 1.0.1e-51.e17.2.7, Message: ensure changed '1:1.0.1e-42.e17.9' to '1.0.1e-51.e17.2.7').

# Continuous Delivery

Continuous delivery is a set of practices that enable IT to have high confidence that every change is shippable with a push of a button, when the business is ready. These practices come together over time and usually involve multiple teams, processes, and tools. At Puppet, we believe these practices are independently valuable and can be used successfully without requiring continuous delivery on Day One. This is why Puppet Enterprise is made up of independently functioning capabilities such as node management, code management, and orchestration — capabilities that can be put together like LEGO blocks to build the right continuous delivery pipeline for you. The continuous delivery pipelines built with Puppet Enterprise are adaptable over time and can be adopted incrementally.

Puppet partners with several continuous delivery vendors such as CloudBees and Atlassian to integrate with the plethora of existing continuous delivery tools. This allows you to leverage existing in-house expertise and investments on your path to continuous delivery.

The screenshot shows the Jenkins interface for a pipeline named 'rgbank-web-component'. It includes a sidebar with navigation options like 'Back to Dashboard', 'Status', 'Changes', and 'Full Stage View'. The main area displays 'Pipeline rgbank-web-component' with 'Last Successful Artifacts' and 'Recent Changes'. Below this is a 'Stage View' table showing the duration of each stage for three builds: #15, #13, and #12.

	Lint and unit tests	Build and package	Deploy to dev	Promote to staging	Staging acceptance tests	Promote to production	Noop production run	Deploy to production
Average stage times:	1s	527ms	54s	1min 7s	1h 17min	12ms	1min 45s	1min 41s
#15 Sep 22 09:04 No Changes	1s	474ms	52s	1min 37s <small>(based on #14)</small>	2h 31min <small>(based on #14)</small>	14ms <small>(based on #14)</small>	1min 44s <small>(based on #14)</small>	1min 35s <small>(based on #14)</small>
#13 Sep 21 15:41 No Changes	1s	480ms	57s	1min 40s <small>(based on #12)</small>	3min 56s <small>(based on #12)</small>	11ms <small>(based on #12)</small>	1min 46s <small>(based on #12)</small>	1min 48s <small>(based on #12)</small>
#12 Sep 21 13:34 No Changes	1s	628ms	54s	5s <small>(based on #11)</small> <b>failed</b>				

Permalinks

- [Last build \(#15\), 6 days 3 hr ago](#)
- [Last stable build \(#15\), 6 days 3 hr ago](#)
- [Last successful build \(#15\), 6 days 3 hr ago](#)
- [Last failed build \(#12\), 6 days 23 hr ago](#)
- [Last unsuccessful build \(#12\), 6 days 23 hr ago](#)
- [Last completed build \(#15\), 6 days 3 hr ago](#)

## Orchestration Control

Orchestration of infrastructure change shouldn't require, nor be the sole responsibility of, a continuous delivery pipeline. Puppet Enterprise enables applications and services to be modeled and managed with the same Puppet code that's used to model your underlying infrastructure. Puppet Enterprise compiles a holistic environment graph of every application and infrastructure service, which models their dependency relationships.

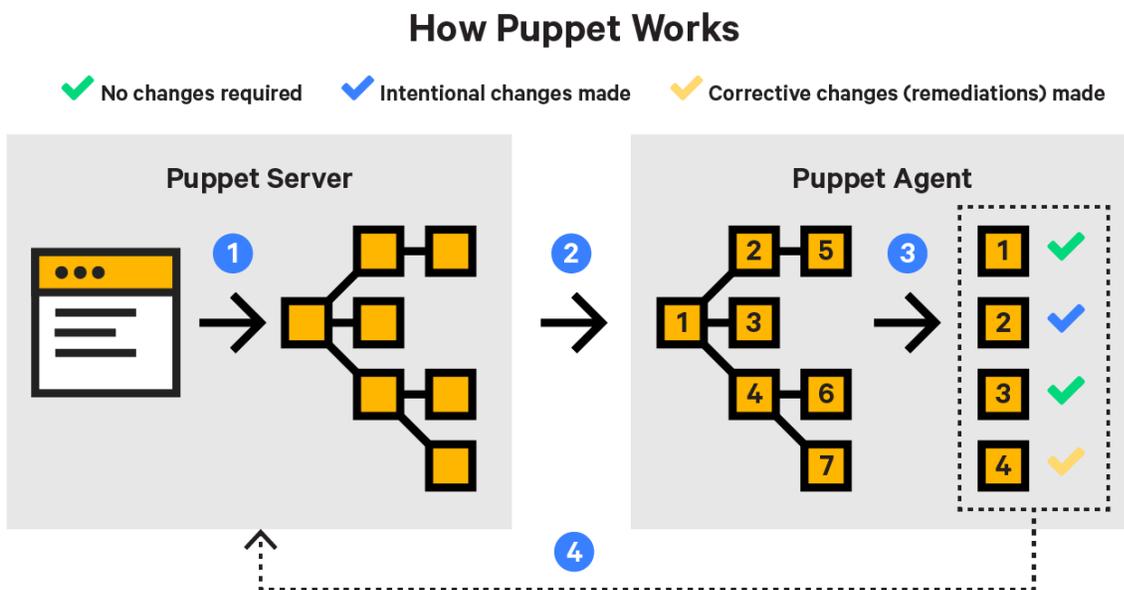
Using command line client tools, Puppet Enterprise users can direct change to as broad or as targeted a portion of infrastructure as required for that moment. Puppet Enterprise uses the environment model to automatically determine the order of operations and the information that needs to be passed between nodes. Puppet Enterprise can recognize and wait for a service to become healthy before continuing to the next step of the deployment.

```
[root@master ~]# puppet app show
Rgbank[production-0]
  Rgbank::Db[database.vm] => database.vm
  + produces MySQLdb[rgbank-production-0]
  Rgbank::Web[appserver02.vm] => appserver02.vm
  + produces Http[rgbank-appserver02.vm]
  consumes MySQLdb[rgbank-production-0]
  Rgbank::Web[appserver01.vm] => appserver01.vm
  + produces Http[rgbank-appserver01.vm]
  consumes MySQLdb[rgbank-production-0]
  Rgbank::Web[appserver03.vm] => appserver03.vm
  + produces Http[rgbank-appserver03.vm]
  consumes MySQLdb[rgbank-production-0]
  Rgbank::Load[loadbalancer.vm] => loadbalancer.vm
  + produces Http[rgbank-web-lb-production-0]
  consumes Http[rgbank-appserver01.vm]
  consumes Http[rgbank-appserver02.vm]
  consumes Http[rgbank-appserver03.vm]
```

# How Puppet Works vs How Chef Works

The open-source versions of Puppet and Chef are similar in many ways, but have critical technical differences in their approach. Puppet code is compiled on the Puppet server and results in a directed acyclic graph (DAG) containing every resource (configuration) for a node. The DAG is what makes Puppet unique among continuous configuration automation (CCA) solutions. It's also what makes Puppet smarter than any other solution. Using the DAG, Puppet can automatically determine the order of operations, intelligently react to failure, easily react to change events, perform simulation runs without requiring support from the Puppet code, and guarantee run reports are accurate.

## The Puppet process



The directed acyclic graph (DAG) contains every desired state and every dependency relationship for every resource, for every node. The dependency relationships are used to automatically determine the order in which the resources should be enforced. Note that the order of operations is perfectly deterministic, meaning that the pre-calculated order of operations will never change in future Puppet runs. The DAG will always be traversed in the same order. If no relationship is specified between two resources, Puppet defaults to the order resources that were declared in the Puppet code.

Whenever an infrastructure change is required (that is, there's new Puppet code), Puppet Enterprise goes through the following process for each node the change needs to be pushed to:

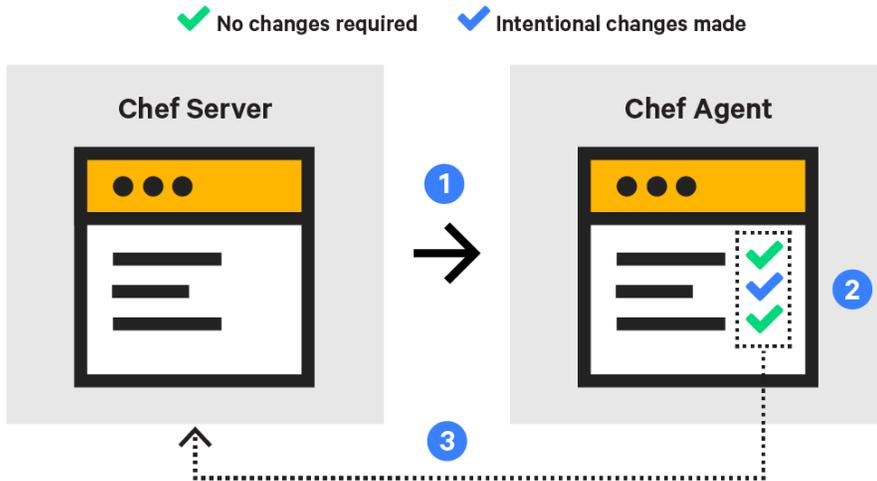
1. Read the Puppet code and compile the DAG into a **catalog**.
2. Send the catalog to the node.
3. Use the DAG to determine order of operations, and then enforce each resource.
4. Send the results to the Puppet server.

### Having a DAG provides numerous benefits:

- **Corrective change reporting.** Puppet knows whether a change was required because the node changed, or because the Puppet code changed. Puppet compares the current DAG with the DAG of the last Puppet run. If the DAG has not changed, yet the node does not match the desired state declared in the DAG, then an out-of-bound change occurred on the node and a remediation, or “corrective change,” needs to take place. Chef has no ability to know if a change is required due to a Chef recipe changing, or due to the node changing between Chef runs.
- **Simpler code.** Writing Puppet code is simpler, since you focus on immediate dependencies, not code order. This is particularly useful as the infrastructure code base scales to manage more and more infrastructure.
- **Desired state conflict detection.** Puppet detects if different sections of Puppet code are trying to manage the same resource and stops, alerting you to the error prior to doing any work on the node. This provides confidence that configurations truly are in the intended state.
- **Better failure handling.** When a failure occurs, Puppet can automatically skip the resources that were dependent on the failing resource, while continuing on to manage the rest of the node configurations that are unrelated to the failure.
- **True simulation mode.** No-op is built into the core of the model. Because only resources are allowed in the catalog (the DAG's artifact), no arbitrary code can be executed on the host. The purely stateful resource model means no-op runs can be trusted. Plus, no special conditionals are required in the Puppet code for no-op to work. With Chef, arbitrary Ruby can be run in a Chef recipe, which will not be respected by Chef's dry run without explicit support. Cookbooks with arbitrary Ruby code can make changes to a node even if Chef is run in dry-run mode.
- **Trusted reporting.** Reports are guaranteed to be accurate. Again, since arbitrary code cannot be run on the node, the events in the run report are guaranteed to be all the events that took place. If arbitrary Ruby code is executed in a Chef recipe, that code's changes cannot be reflected in the Chef report. You don't know the true state of your infrastructure.

## The Chef process

### How Chef Works



The Chef server is essentially a file server. It determines which cookbooks should be delivered to the agents, and which recipes should be run. All code executes on the agents themselves. The Chef agent simply executes as the recipes are read, one line at a time. Chef's imperative, procedural approach and lack of a DAG has disadvantageous, compared to Puppet's declarative, model-driven approach:

- Arbitrary Ruby code execution means no dependable way to simulate the changes Chef code would make.
- Configurations can be changed several different ways in the same Chef run, so you can't trust that the node is in its desired state at the end of the run.
- Relationships between components are unclear on complex systems.
- When writing cookbooks, you can't know whether a dependent configuration from another cookbook will be ready at run time.
- Reports cannot be trusted if arbitrary Ruby code was run outside of Chef's Ruby DSL.

## We want to hear from you

We hope we've helped you understand the differences between Chef Automate and Puppet Enterprise. 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.