Puppet:

Puppet is a dynamic and idempotent server configuration management tool. Puppet supports both push and pulls mechanism. Puppet infrastructure generally contains 1 (or more) puppetmaster servers, along with a special agent package installed on each client node.

Pros:

1. Puppet is not associated with any particular language or web framework. Its users manage Rails apps, but also PHP applications, Python and Django, Mac desktops, or AIX mainframes running Oracle.
2. Strong compliance automation and reporting tools.
3. Advanced GUI that can manage nodes in real time
4. Larger community base which offers multiple implementations for any given module (ex: nginx, ntp, jenkins), each with their own quirks, strengths, and deficiencies.
5. Puppet supports multiple platforms.
6. Puppet integrates with wide variety of cloud providers including AWS, Rackspace, Openstack, Azure, HP Cloud.
7. Puppet uses a declarative language that is similar to JSON or XML.
8. Puppet's has over 80 organizations including Google, Red Hat, Siemens, lots of big businesses worldwide, and several major universities including Stanford and Harvard Law School. It has been through the ringer (meaning it has been tested in more large environments)
9. Puppet has its own DSL which is a subset of Ruby. It's not a Turing-complete language and was specifically designed to be accessible to sysadmins. Easy to learn.
10. Puppet internally creates a directed graph of all of the resources to be defined in a system along with the order they should be applied in. This is a robust way of representing the resources to be applied and Puppet can even generate a graph file so that one can visualize everything that Puppet manages.
11. Puppet users explicitly call out the package manager to be used.
12. Puppet gets a win for having a real noop mode.
13. Puppet is market leader out of all configuration management tools.
14. High availability is achieved through adding more puppet servers.

Cons:

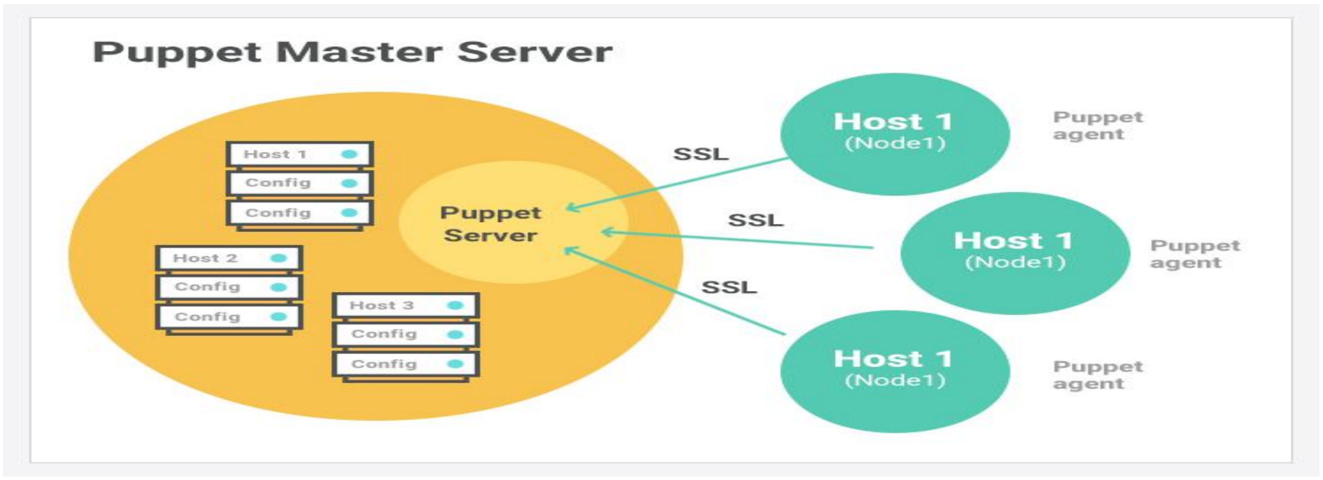
1. Resources defined in a Puppet manifest are not applied in order of their appearance (ex: top->bottom) which is confusing for people who come from conventional programming languages (C, Java, etc). Instead resources are applied randomly, unless explicit resource ordering is used. Ex: "before","require", or chaining arrows.

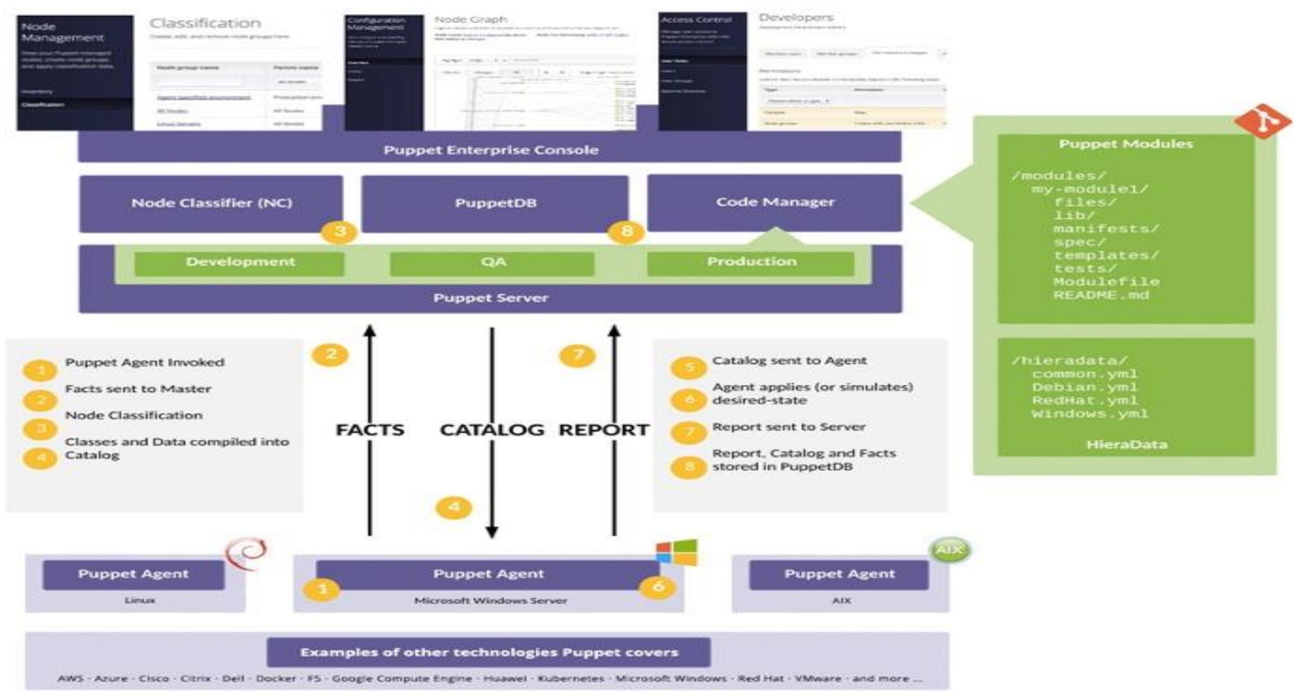
2. Creation of a directed graph of all of the resources can lead to circular dependencies.
3. Puppet is built upon Ruby and the Ruby ecosystem of tools for testing. Adding complex functionality is done through Ruby modules
4. All agents need to have puppet agent installed.
5. Puppet failure notification configurations achieved through extra effort.

Pricing:

The annual costs (as of December 2016) are $120/node for Puppet Enterprise (with a minimum 10 nodes, free below the threshold)

Architecture:





Ansible:

Ansible uses agentless architecture and easiest to implement. It is also dynamic and idempotent server configuration management tool. Ansible has neither a special master server, nor special agent executables to install. The executor can be any machine with a list (inventory) of the nodes to contact, the Ansible playbooks, and the proper SSH keys/credentials in order to connect to the nodes.

Pros:

1. Easy to implement because no agents to be installed and works well with vSphere/Packer.
2. Easy to learn.
3. Ansible follows a push workflow. It can also work in pull mechanism.
4. Playbooks are applied in top-to-bottom execution order.
5. High availability is achieved through secondary instances of tower.
6. Simple architecture and faster deployments.
7. Tower supports email notifications.

Cons:

1. Ansible pull doesn't scale very well, you still need to set up a master if you want to constantly enforce state/audit changes and at that point, you'll probably want to buy Tower.

2. The number of modules available on Ansible Galaxy is about on par with what is available at PuppetForge.
3. Ansible provides less abstraction when mentioning OS's package managers.
4. Ansible has semi-working dry-run (noop) mode.
5. Ansible has no resource dependency graph built internally.
6. The client machine requires Python 2.5+ installation.
7. The reports from clients are collected by setting cron jobs.
8. This platform is not mature as compared to puppet and chef.

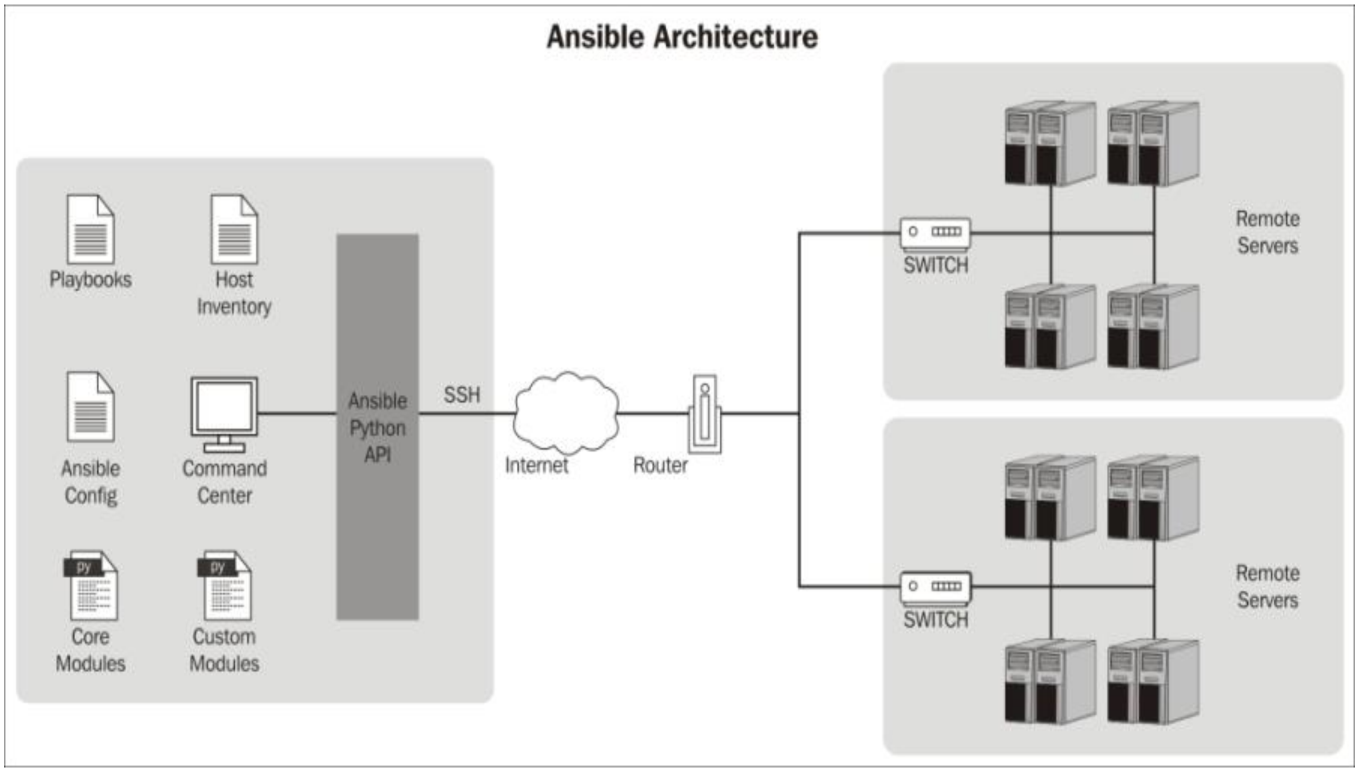
Pricing:

The self-support offering starts at $5000/year (100 nodes) and $10000/year (250 nodes)

The standard offering starts at $10000/year (100 nodes) and custom pricing over 100 nodes with 8x5 support.

The premium offering starts at $14000/year (100 nodes) and custom pricing over 100 nodes with 24x7 support.

Architecture:



Chef:

Chef can build, deploy and manage infrastructure as code. It is dynamic and idempotent server configuration management tool.

Pros:

1. With chef 12, high availability is achieved through hosting chef server on AWS cloud with elastic IP attached.
2. The order of execution is sequential.
3. Chef integrates with wide variety of cloud providers including AWS, Rackspace, Openstack, Azure, HP Cloud.
4. Code driven approach (more flexibility).

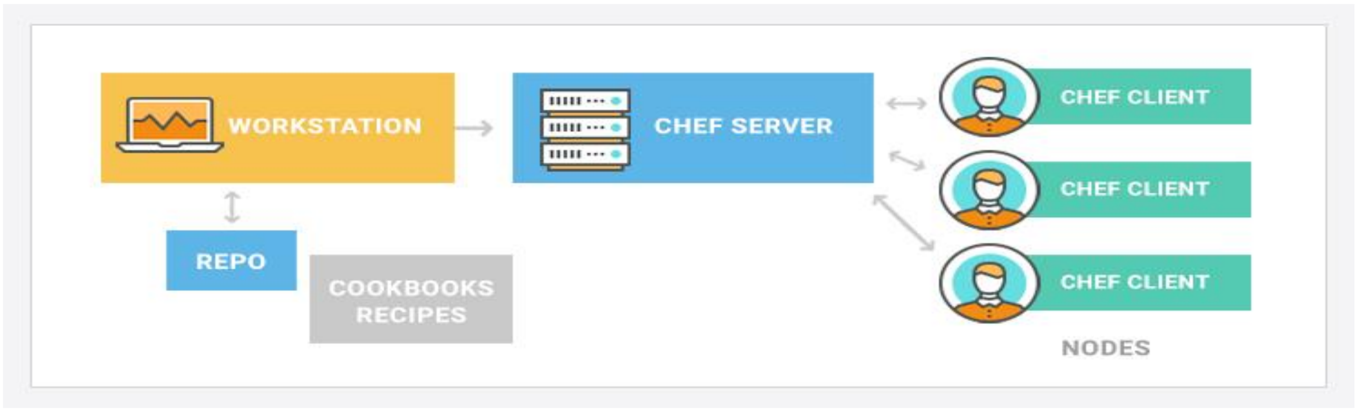
Cons:

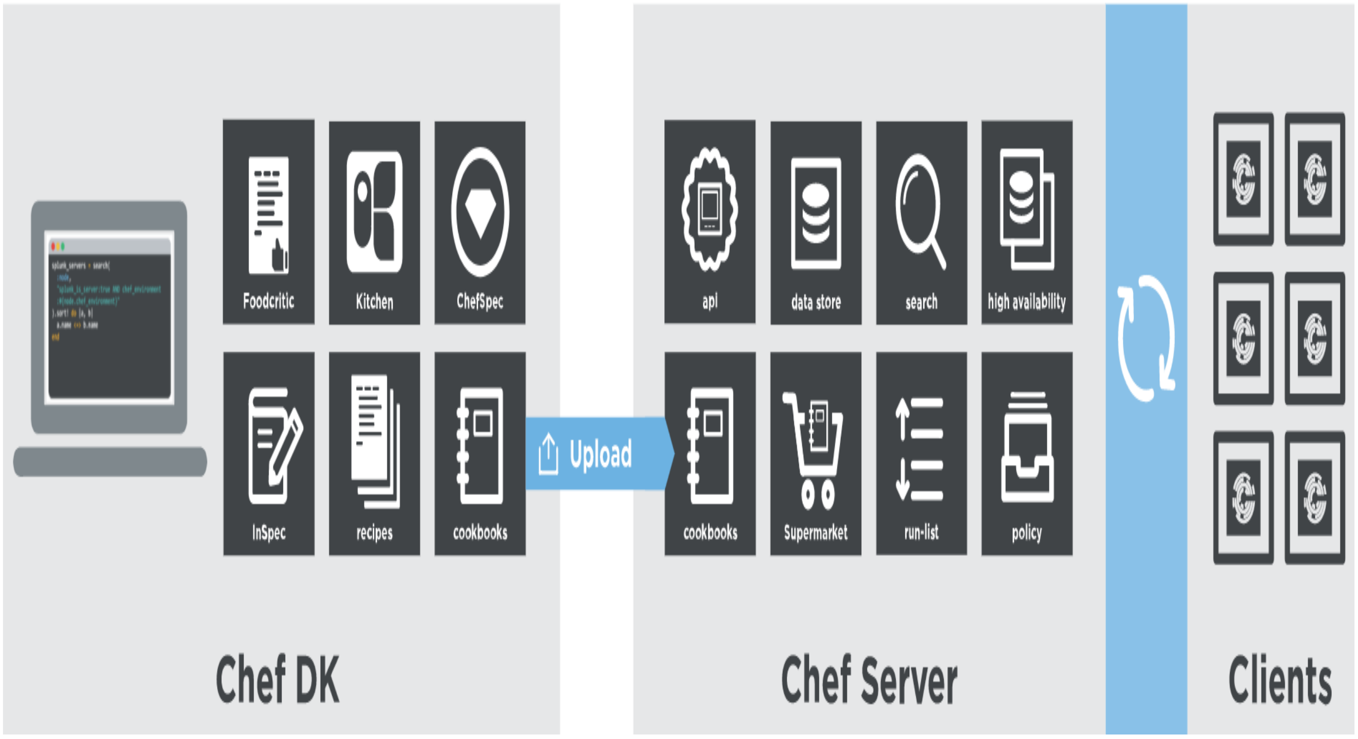
1. Chef supports push mechanism with additional extension called chef push jobs and it is not supported when running chef with the hosted chef server.
2. Initial setup is not as smooth as other configuration management tools.
3. Steep learning curve when compared to other configuration management tools.
4. Chef uses imperative language which is like programming the control of nodes.
5. Chef is much younger project than Puppet, so the developer community is smaller compared to puppet.
6. Additional piece called chef workstation is required to control configurations from master to agents.
7. Chef has no resource dependency graph, so there are chances of having dependency conflicts from other cookbooks which are written by other teams.

Pricing:

Hosted chef starts at $72 node/year with 12x5 support.
Chef automate start at $137 node/year with 12x5 support.

Architecture:





Questions need to be considered before taking decision:

1. **Are you going to run nodes in-house, cloud or hybrid?**
2. **What is the growth of infrastructure?**
3. **What about stability of the tool?**
4. **Who are writing the code?**