NAME – Bhuvan Sawant
BATCH – T22
ROLL NO - 110

Assignment 10

AIM: To study Puppet Tools.

LO1: To understand the fundamentals of DevOps engineering and be fully proficient with DevOps terminologies, concepts, benefits, and deployment options to meet your business requirements.

LO6: To Synthesize software configuration and provisioning using Ansible.

THEORY:

What Is Puppet?

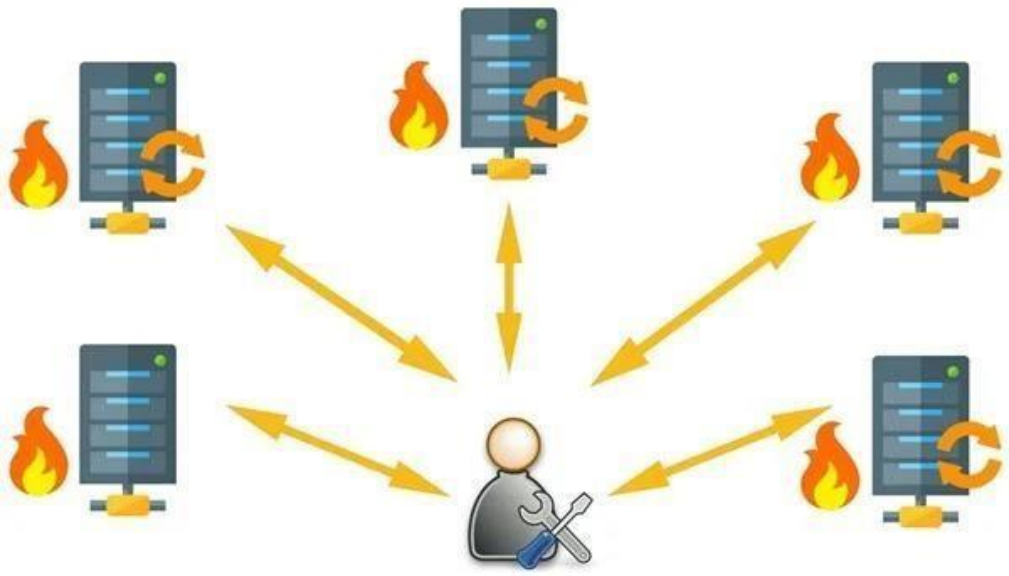
Puppet is a Configuration Management tool that is used for deploying, configuring and managing servers. It performs the following functions:

- Defining distinct configurations for each and every host, and continuously checking and confirming whether the required configuration is in place and is not altered (if altered Puppet will revert back to the required configuration) on the host.
- Dynamic scaling-up and scaling-down of machines.
- Providing control over all your configured machines, so a centralized (master-server or repo-based) change gets propagated to all, automatically.

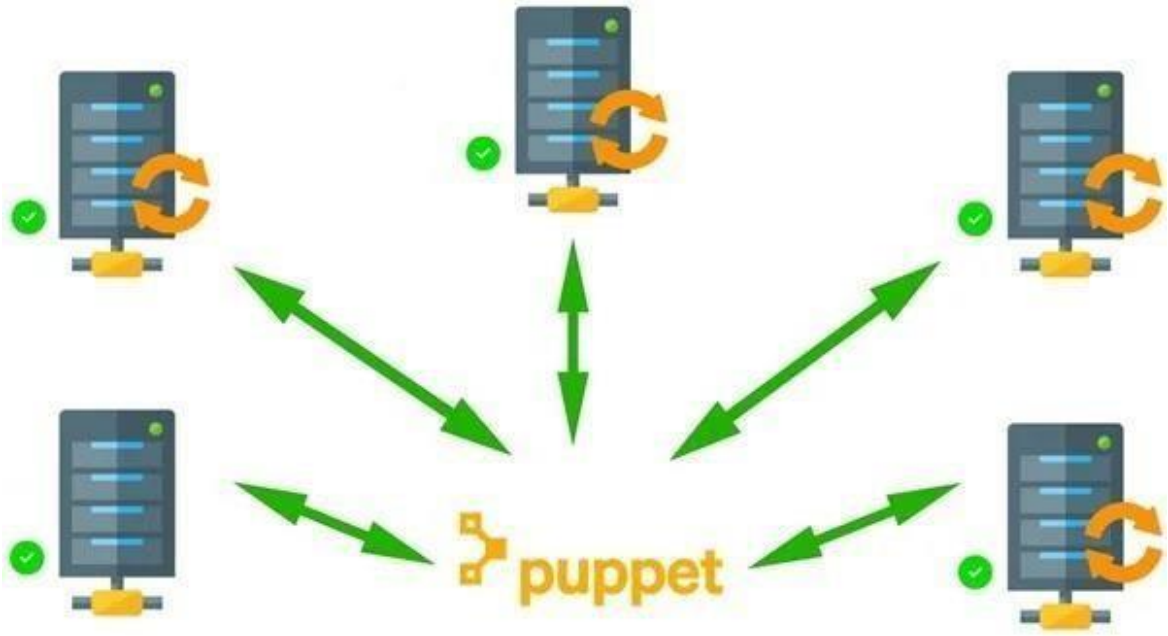
Puppet uses a Master Slave architecture in which the Master and Slave communicate through a secure encrypted channel with the help of SSL.

What Puppet can do?

For example, you have an infrastructure with about 100 servers. As a system admin, it's your role to ensure that all these servers are always up to date and running with full functionality.



To do this, you can use Puppet, which allows you to write a simple code which can be deployed automatically on these servers. This reduces the human effort and makes the development process fast and effective.



Puppet performs the following functions:

Puppet allows you to define distinct configurations for every host.

The tool allows you to continuously monitor servers to confirm whether the required configuration exists or not and it is not altered. If the config is changed, Puppet tool will revert to the pre-defined configuration on the host.

It also provides control over all the configured system, so a centralized change gets automatically effected.

It is also used as a deployment tool as it automatically deploys software to the system. It

implements the infrastructure as a code because policies and configurations are written as code.

How Puppet works?

Puppet is based on a Pull deployment model, where the agent nodes check in regularly after

every 1800 seconds with the master node to see if anything needs to be updated in the agent. If anything needs to be updated the agent pulls the necessary puppet codes from the master and performs required actions.

Let's explain it by an example:

Example: Master - Agent Setup:

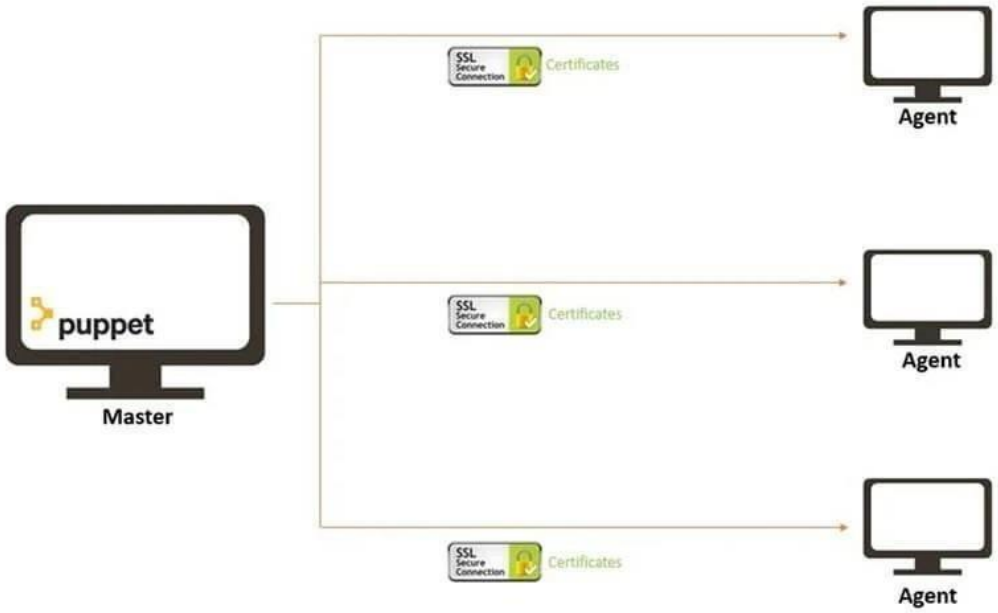
The Master:

A Linux based machine with Puppet master software installed on it. It is responsible for maintaining configurations in the form of puppet codes. The master node can only be Linux.

The Agents: The target machines managed by a puppet with the puppet agent software installed on them.

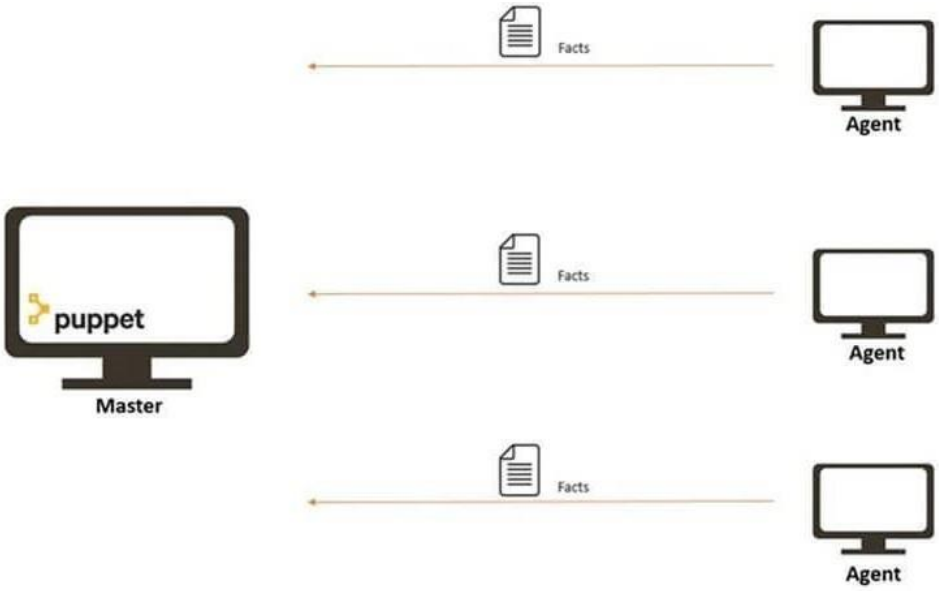
The agent can be configured on any supported operating system such as Linux or Windows or Solaris or Mac OS.

The communication between master and agent is established through secure certificates.



Communication between the Master and the Agent:

Step 1) Once the connectivity is established between the agent and the master, the Puppet agent sends the data about its state to the Puppet master server. These are called Facts: This information includes the hostname, kernel details, IP address, file name details, etc....



Step 2) Puppet Master uses this data and compiles a list with the configuration to be applied to the agent. This list of configurations to be performed on an agent is known as a catalogue. This could be changed such as package installation, upgrades or removals, File System creation, user creation or deletion, server reboot, IP configuration changes, etc.

Step 3) The agent uses this list of configurations to apply any required configuration changes on the node. In case there are no drifts in the configuration, Agent does not perform any configuration changes and leaves the node to run with the same configuration.

Step 4) Once it is done the node reports back to puppet master indicating that the configuration has been applied and completed.

Puppet Blocks

Puppet provides the flexibility to integrate Reports with third-party tools using Puppet APIs.

Four types of Puppet building blocks are

1. Resources
2. Classes
3. Manifest
4. Modules

Puppet Resources:

Puppet Resources are the building blocks of Puppet.

Resources are the inbuilt functions that run at the back end to perform the required operations in puppet.

Puppet Classes:

A combination of different resources can be grouped together into a single unit called class.

Puppet Manifest:

Manifest is a directory containing puppet DSL files. Those files have a .pp extension. The .pp

extension stands for puppet program. The puppet code consists of definitions or declarations of Puppet Classes.

Puppet Modules:

Modules are a collection of files and directories such as Manifests, Class definitions. They are the re-usable and sharable units in Puppet.

Benefits of Puppet (DevOps Tool)

Here are the benefits of Puppet for your glance

1. Infrastructure as code: Puppet enables to define your infrastructure as code (IAC) with easy of

coding few puppet scripts. IAC is an important part DevOps practices such as version control, peer review, automated testing and deployment, and continuous delivery.

2. Any downtime due to configuration issues is reduced to minimum, mostly avoided, as the speed of recovery is pretty fast.
3. Puppet provides a significant time savings with its speed of deployment.
4. Puppet is supported by a larger open source developer platform.
5. Unlike Scripting, Puppet works just fine even on a large infrastructure by automating the repetitive tasks easily.
6. It supports a wide number of platforms such as Microsoft Windows, Debian, BSD, etc.
7. An easy to learn language is being used by Puppet to define the configurations for a host.

Understanding Puppet Manifest Files: How to use and write Puppet Manifest Files? For the process of configuration management using Puppet, we will have to create a set of provisioning scripts or Puppet codes that will configure our systems. Puppet code is primarily composed of manifests. **What are Puppet Manifest Files?**

Puppet manifest files are the files where all the resources, i.e., services, packages, or files that need to be checked and changed, are declared. Puppet manifest files are created on the Puppet master, and they have the .pp extension. They consist of the following components:

- Files: They are the plain text files that are to be imported and placed in the target location.
- Resources: Resources represent the elements that we need to evaluate or change. Resources can be files, packages, etc.

- Node definition: It is a block of code in Puppet where all the information and definition of the client node are defined.
- Templates: Templates are used to create configuration files on nodes and can be reused later.
- Classes: Classes are what we use to group the different types of resources together.

Syntax of a Manifest File

Following is the syntax for writing a basic manifest file: resourcetype

{

'title':argument or attribute1 => value, argument

or attribute2 => value, }

Why do we need Puppet Manifest Files?

Writing manifest files is the closest to what we might consider as Puppet programming. Manifest files are used to define how resources should be configured. In them, we declare the resources, and they are fundamental for Puppet codes. Manifests are the building blocks of the complex Puppet modules.

Writing a Basic Manifest File with Examples

Let's start off by writing the most basic and smallest component of the manifest file, the resource.

file { 'intellipaat_file':ensure => present, path

=> 'intellipaat_file.txt', }

Note: When writing multiple resources, we have to keep in mind that the resources of the same type cannot have the same title. However, we can use the same title for different types of resources as that will not cause any conflicts.

It's also important to know that in the case of multiple resources, Puppet does not evaluate and execute the resources in the same sequence as they are defined. We have to explicitly define the priority and dependency between these resources.

For example: In the following code block, we have defined two types of resources: the first one is a package and the other is an exec package, which implies that it is a command. When we use a package resource, we need to make sure that it is installed. The before keyword is used to make sure that the package resource is executed before the command.

package { 'curl':ensure => 'installed' before

=> Exec['install script']

}

exec { 'install script':

command => '/usr/bin/curl'

}

When the exec package (lowercase) is used for declaring the resources, we use Exec (uppercase) for the already defined resources.

How to automate changes using Puppet in all infrastructure servers?

Example

Objective: To modify the etc/motd file in all infrastructure servers and to use Puppet automation to add the content of our choice in that file

Note: The /etc/motd file is a file available on the UNIX-like system. It consists of a message as

'The message of the day'.

Steps to implement the above example:

- Make sure that the file is present
- If present, then make the required changes
- Change the file permissions (optional)

Following is the code block of the manifest file to perform the above-mentioned operations:

file {'/etc/motd':name => '/etc/motd' ensure=> present, owner => 'root', group => 'root',

content => 'The testing content', }

}

In this example, the resource type is a file and the title is /etc/motd. The content that we have added to the file is 'The testing content'.

Conclusion: In this experiment, we learn puppet tool and understood how to use it in

infrastructure as a code service.