# Puppet

[configure & manage]
IaC

→ Puppet is Ruby based

Deployment model {
Puppet Models
Standalone    Master-Agent

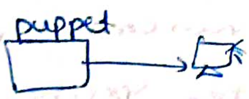
OS {
Puppet master supports Linux
" Agent " Linux & Windows

$$\begin{tikzcd}
 & \text{Programming} \arrow[dl, dash] \arrow[dr, dash] & \\
\text{Imperative} \arrow[d] & & \text{Declarative} \arrow[r] & \text{[specifying only final destination i.e. only what to do]} \\
\text{[Specifying what to do and how to do]} & & &
\end{tikzcd}$$

→ Why is Puppet ??
↳ puppet is declarative
↳ Increased productivity
↳ Consistency Delivery
↳ Simplicity
↳ Scalable

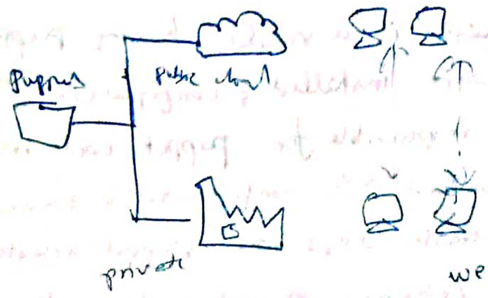
→ Use Case

Eg. to configure appl<sup></sup>n on single server like to install appl<sup></sup>n packages.



- Install Packages
- Manage firewalls ports
- modify configuration files
- Manage services

Eg: Setup complex infra in public / private cloud. In puppet we can mention VM's on public / private cloud.



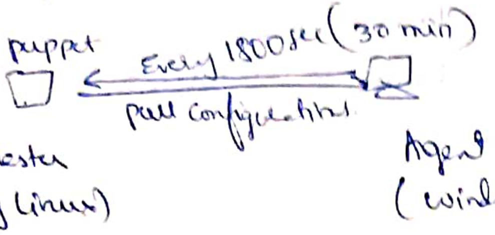
we can enable communication

→ How puppet works?

Deployment models:

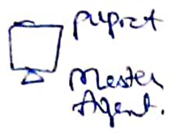
1) Master-Agent:

Agents are managed by puppet agents installed on master. Agent node checks regularly & updates for every (1800 sec). Agent pulls some necessary code from master & required actions are performed known as pull configuration



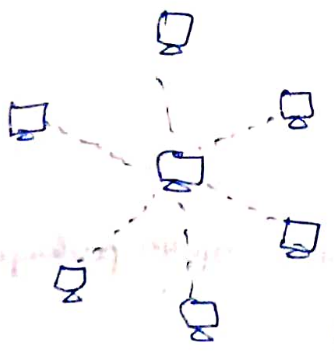
used for Production

2) Stand-alone



used for
[Dev/Test/POC]

→ Pull vs Push (Deployment models)



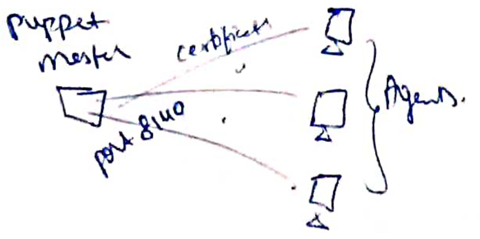
push:- Master server pushes the configuration & softwares to individual servers after verifying inventory & establishing secure connection. The master runs commands remotely on client & configurat initiated by master node

Eg: of such tools are Ansible, saltstack.

pull: The individual servers contact master server & establishing a connection after specifying inventory specific master & download the software then configure part is initiated by agents servers

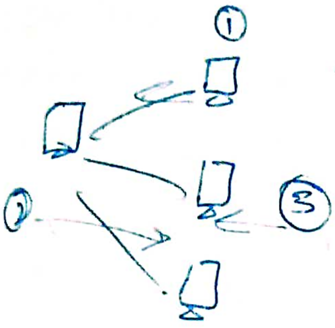
Eg: puppet, chef.

→ Execution flow:



puppet master is a node having puppet master software installed & configured. This machine is responsible for puppet code management & contains diff configuration in environment & administration logs into puppet master to create /change puppet codes. Here we have multiple agents setup with puppet agents installed on them the communication b/w master & agent is by secure certificates. Puppet Master allow us a secure connection b/w master & agent via port 8140. It should be open port on master.

communication is done in 3-step process:-



① facts
Once the connection is setup b/w agent & master the agent sends data to puppet master server called facts. This info ~~factes~~ include hostname, ipaddress, & file name....etc.

Puppet uses this facts & compiles the list with the configurations to be applied to agent. This is known as ② Catalog. This could change as a package installation upgrades / removals, ~~file~~ server reboot. ip config. etc...

is diff b/w running & desired state.

The Agent uses Catalog to apply required configu changes on the nodes. If no drift in configuration, then agent is not performs any changes & leaves the server on same.

After receiving catalog from master, puppet agent responds immediately to the changes by executing the configuration drifts. Once done the node reports back to the puppet master, indicating configuration has been applied & completed. ③ Reports Puppet provides flexibility to integrate this reports with 3rd party tools with puppet API's.

→ Ex
Suppose we have 500 Redhat servers, we need to create user 'priyanka'. So login to each server & adding user by running a command to create user is definitely not a practical approach.

So better approach is to write a script & let script perform the execution on a large number of servers.

But now we have 500 server with diff OS like CentOS, Redhat, Ubuntu... So command to add user might be diff accd to OS. So here we can use Configuration Management tool like Puppet.

↳ with puppet code used to

add →

```
user { "~~yog~~priyanka":
  ensure => "present";
}
```

to delete

```
user { "priyanka":
  ensure => "absent";
}
```

Eg: # NTP Package Installation

Install package
```
package { "ntp":
        ensure => "present",
    }
```

Configure File
```
#NTP file configuration
file { "/etc/ntp.conf":
   ensure => "present",
   content => "server 0.centos.pool.ntp.... \n";
  }
```

Start Service
```
#NTP service startup
 Service { "ntpd":
    ensure => "running",
   }
```

→ Puppet Resources

are the building blocks of puppet. Resources are all inbuilt functions th runs at the back end & to perform underlying operations in puppet.

They contain individual operations that can be performed with Puppet such file resource type. To work with files & directories. Service RT to manage appl^n services & user RT to manage users etc.

↳ Class

A combination of different resources & operations can be grouped together into a single unit called Class. Think of it as multiple small operations working towards a single larger goal such as everything required to setting up web services on web server.

↳ Manifest

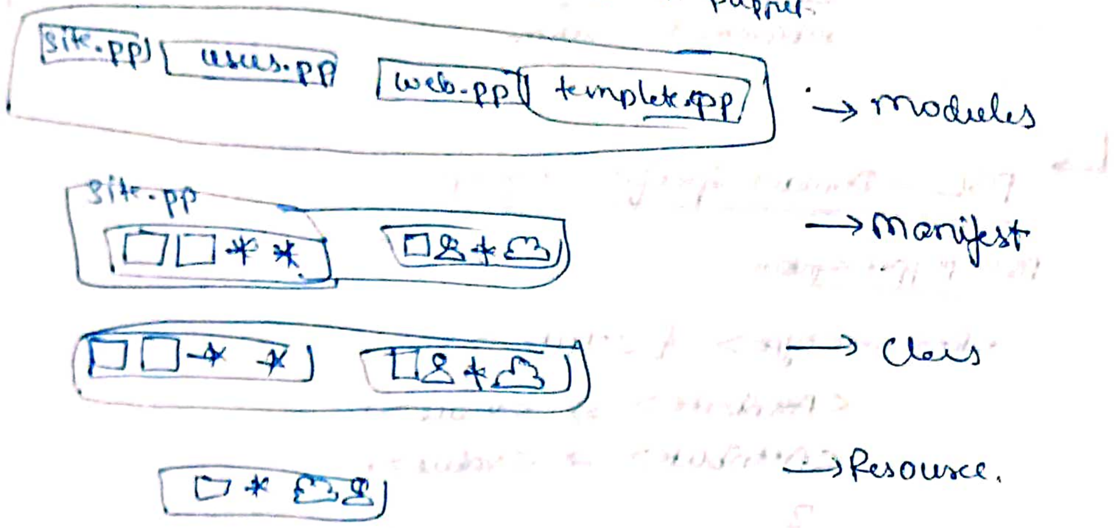
" is a directory containing puppet DSL files with .pp extension

.pp → puppet program. The puppet code consists of definations or declarations of Puppet Classes.

↳ Modules

" are collection of files & directories such as Manifests, class definations and any other dependent files that work together & follow specific directory structure. They are the re-usable & sharable units in puppet. Eg: mysql module to install & configure mysql to Jenkins module to manage.....

→ think resources as functions that performs individual tasks such as creating a user. Multiple such functions are usually organized into Class. We develop classes & files. In this case it happens to be Python file named user.py. Think of this a manifest in Puppet. Multiple such related python files and any dependent packages required by them can be packaged into a package in python. These can be shared & re-usable files with other or can be uploaded online which you can relate to modules in puppet.



→ Puppet Resource

All the operations on puppet agent are performed with the help of puppet resources. Puppet ships with multiple inbuilt resources which can be used to automate almost any IT operations & tasks.

Simply.... Puppet resources are ready made tools that are used to perform various tasks & operations on any supported platforms. Puppet Resources are completely compatible with any standalone, VM or any cloud env.

Puppet ships with multiple inbuilt resources types some of them are packages, files, service, user & group.

There are 3 resource types.

1) Core or built-in resource types which are pre-built puppet resource types shipped with puppet software. These prebuilt resource types are always available to be used in puppet Architecture. These are maintained by Puppet team. 2) $\boxed{.pp}$ Puppet resource types are written in Declarative language which could be combination of existing resource types. These are known as defined resource types & 3) $\boxed{.rb}$ Puppet provides us with flexibility to write our own complete customized resource types known as custom resource types in puppet

Some prior exposure to Ruby Language is highly recommended.

Commands

→ puppet ~~Help~~ OR puppet --help ↳ help command.

→ puppet help resource OR puppet resource --help
↳ list all actions

→ puppet resource --types ↳ list all resource types.
(resource ↳ subcommand; --types ↳ actions)

↳ DSL - Domain Specific Language.

DSL - puppet-syntax

```
<Resourcetype> { <Title> :
        <Attribute> => <value>,
        <Attribute> => <value>,
      }
```

Eg:

```
file   { "/etc/ntp.conf":
    ensure  => "present",
    content => "server pool.ntp.org\n";
      }
```

see Eg from backside

Code Creation process

create (1) — Check (2) — Test (3) — Run (4) → puppet apply

Check ↓ puppet parser validate

Test ↓ puppet apply --noop

Summary

→ puppet Resource ↳ inbuilt functions

→ puppet Resource types ↳ ③ - core / builtin, defined & custom

→ commands - (some commands)

puppet resource --types ↳ list of resource types

puppet describe ↳ detailed info about resource types

~~puppet~~

Puppet parser ↳ used to perform syntax checks on puppet programs

→ puppet code create process with self-contained deployment model for simplicity.

puppet help more

Usage: puppet <subcommand> [options] <action> [options]

creating directory

```
# mkdir /var/tmp/demo
# cd /var/tmp/demo
# ls -lst
# vi demouser.pp

    user { "priya":
         ensure => "present",
         }

:wq → Exit
# cat demouser.pp
```

to check syntax error:

```
# puppet parser validate demouser.pp
# clear
```

→ dry demo apply.

```
# puppet apply demouser.pp --noop
```

→ real apply

```
# puppet apply demouser.pp
```

→ to check user

```
# id priya
user id = ---(priya)   gid= (   )   groups=---
```

→ for deleting

```
# userdel priya
# id priya
user not found
```

then apply again

```
# puppet apply demouser.pp
# id priya
id = -----
# clear.
```

lets change uid & shell of existing user.

```
# grep -i priya /etc/passwd
# vi demouser.pp
    user { "priya":
         ensure => "present",
         uid => "7777",
         shell => "/bin/sh",   }
```

```
:wq

# puppet parser validate demouser.pp
# puppet apply demouser.pp --noop
# puppet apply demouser.pp
# cat grep -i priya /etc/passwd
```

lets create some test file

```
# vi ~~demouser.pp~~ demofile.pp
file { "/var/tmp / testfile" :
            ensure => "present",
            owner => "priya",
            group => "priya",
             mode => "0777",
             }

        :wq

 # cat demofile.pp

 # grep -i priya /etc/passwd
 # puppet parser validate demofile.pp
 # puppet apply demofile.pp --noop
 # ls -lst. /var/tmp/testfile.
 # date
 current date.

 # clear...
```

Code to install some package

```
#vi telnet.pp
package {
```