The hour at which to run the cron job. Optional; if specified, must be between 0 and 23, inclusive.

minute

*(**Property:** This attribute represents concrete state on the target system.)*
The minute at which to run the cron job. Optional; if specified, must be between 0 and 59, inclusive.

user

*(**Property:** This attribute represents concrete state on the target system.)*
The user who owns the cron job. This user must be allowed to run cron jobs, which is not currently checked by Puppet.
This property defaults to the user running Puppet or root.
The default crontab provider executes the system crontab using the user account specified by this property.

Resource type: File

Description

Manages files, including their content, ownership, and permissions.
The file type can manage normal files, directories, and symlinks; the type should be specified in the ensure attribute.
File contents can be managed directly with the content attribute, or downloaded from a remote source using the source attribute; the latter can also be used to recursively serve directories (when the recurse attribute is set to true or local). On Windows, note that file contents are managed in binary mode; Puppet never automatically translates line endings.
Autorequires: If Puppet is managing the user or group that owns a file, the file resource will autorequire them. If Puppet is managing any parent directories of a file, the file resource will autorequire them.

Example:

```
file { '/etc/inetd.conf':
  ensure => link,
  target => '/etc/inet/inetd.conf',
}
```

ensure

*(**Property:** This attribute represents concrete state on the target system.)*
Whether the file should exist, and if so what kind of file it should be. Possible values are present, absent, file, directory, and link.

34. present accepts any form of file existence, and creates a normal file if the file is missing. (The file will have no content unless the content or source attribute is used.)
35. absent ensures the file doesn't exist, and deletes it if necessary.
36. file ensures it's a normal file, and enables use of the content or source attribute.
37. directory ensures it's a directory, and enables use of the source, recurse, recurselimit, ignore, and purge attributes.
38. link ensures the file is a symlink, and **requires** that you also set the target attribute. Symlinks are supported on all Posix systems and on Windows Vista / 2008 and higher. On Windows, managing symlinks requires Puppet agent's user account to have the "Create Symbolic Links" privilege; this can be configured in the "User

Rights Assignment" section in the Windows policy editor. By default, Puppet agent runs as the Administrator account, which has this privilege.

Puppet avoids destroying directories unless the force attribute is set to true. This means that if a file is currently a directory, setting ensure to anything but directory or present will cause Puppet to skip managing the resource and log either a notice or an error.

There is one other non-standard value for ensure. If you specify the path to another file as the ensure value, it is equivalent to specifying link and using that path as the target:

Resource Type: exec

Description

Executes external commands.

Any command in an exec resource **must** be able to run multiple times without causing harm — that is, it must be *idempotent*. There are three main ways for an exec to be idempotent:

Example

```
exec { 'tar -xf /Volumes/nfs02/important.tar':
  cwd     => '/var/tmp',
  creates => '/var/tmp/myfile',
  path    => ['/usr/bin', '/usr/sbin',],
}
```

cwd

The directory from which to run the command. If this directory does not exist, the command will fail

path

The search path used for command execution. Commands must be fully qualified if no path is specified. Paths can be specified as an array or as a ':' separated list.

creates

A file to look for before running the command. The command will only run if the file **doesn't exist.** This parameter doesn't cause Puppet to create a file; it is only useful if **the command itself** creates a file.

Resource Type: package

Description

Manage packages. There is a basic dichotomy in package support right now: Some package types (e.g., yum and apt) can retrieve their own package files, while others (e.g., rpm and sun) cannot. For

those package formats that cannot retrieve their own files, you can use the source parameter to point to the correct file.
Puppet will automatically guess the packaging format that you are using based on the platform you are on, but you can override it using the provider parameter; each provider defines what it requires in order to function, and you must meet those requirements to use a given provider.
You can declare multiple package resources with the same name, as long as they specify different providers and have unique titles.

Example
```
package { 'ntp':
ensure => 'installed',
}
```

ensure
*(**Property:** This attribute represents concrete state on the target system.)*
What state the package should be in. On packaging systems that can retrieve new packages on their own, you can choose which package to retrieve by specifying a version number or latest as the ensure value. On packaging systems that manage configuration files separately from "normal" system files, you can uninstall config files by specifying purged as the ensure value. This defaults to installed.

Resource Type: notify

Description
Sends an arbitrary message to the agent run-time log.

```
notify { 'resource title':
 name     => # (namevar) An arbitrary tag for your own reference; the...
 message  => # The message to be sent to the...
 withpath => # Whether to show the full object path. Defaults...
 # ...plus any applicable metaparameters.
}
```

```
package { 'apache2':
       provider=>'apt',
       ensure=>'installed'
}
 notify { 'Apache2 is installed.':
}
service { 'apache2':
       ensure=>'running'
}
 notify { 'Apache2 is running.':
}
```

name
*(**Namevar:** If omitted, this attribute's value defaults to the resource's title.)*
An arbitrary tag for your own reference; the name of the message.

(↑ Back to notify attributes)

message
*(**Property:** This attribute represents concrete state on the target system.)*
The message to be sent to the log.
(↑ Back to notify attributes)

withpath
Whether to show the full object path. Defaults to false.
Valid values are true, false.

Resource Type: service

Descritption:
Manage running services. Service support unfortunately varies widely by platform — some platforms have very little if any concept of a running service, and some have a very codified and powerful concept. Puppet's service support is usually capable of doing the right thing, but the more information you can provide, the better behaviour you will get.

```
service { ntpd:
ensure => 'running',
enable => true,
}
```

ensure
*(**Property:** This attribute represents concrete state on the target system.)*
Whether a service should be running.
Valid values are stopped (also called false), running (also called true).

enable
*(**Property:** This attribute represents concrete state on the target system.)*
Whether a service should be enabled to start at boot. This property behaves quite differently depending on the platform; wherever possible, it relies on local tools to enable or disable a given service.
Valid values are true, false, manual, mask.

Resource Type: user

Description
Manage users. This type is mostly built to manage system users, so it is lacking some features useful for managing normal users.
This resource type uses the prescribed native tools for creating groups and generally uses POSIX APIs for retrieving information about them. It does not directly modify /etc/passwd or anything.

```
user { 'agent1':
   ensure          => 'present',
```

```
    home            => '/sbin',
    uid             => '2',
    shell           => '/sbin/nologin',
    password        => '$1$HTQUGYUGYUGwsxQxCp3F/nGc4DCYM/',

}
```

ensure

*(**Property:** This attribute represents concrete state on the target system.)*
The basic state that the object should be in.
Valid values are present, absent, role.

home

*(**Property:** This attribute represents concrete state on the target system.)*
The home directory of the user. The directory must be created separately and is not currently checked for existence.

uid

*(**Property:** This attribute represents concrete state on the target system.)*
The user ID; must be specified numerically. If no user ID is specified when creating a new user, then one will be chosen automatically. This will likely result in the same user having different UIDs on different systems, which is not recommended. This is especially noteworthy when managing the same user on both Darwin and other platforms, since Puppet does UID generation on Darwin, but the underlying tools do so on other platforms.

shell

*(**Property:** This attribute represents concrete state on the target system.)*
The user's login shell. The shell must exist and be executable.
This attribute cannot be managed on Windows systems.
Requires features manages_shell.

Linux users have their passwords stored as hash in /etc/shadow file. Puppet passes the password supplied in the user type definition in the /etc/shadow file.
Generate your hash password using openssl command:

```
#openssl passwd -1
 #Enter your password here
 Password:
 Verifying - Password:
 $1$HTQUGYUGYUGwsxQxCp3F/nGc4DCYM
```

The previous example generate this hash: 1HTQUGYUGYUGwsxQxCp3F/nGc4DCYM/
Add this hash password to your class as shown (do not forget the quotes)

```
user { 'test_user':
  ensure   => present,
  password =>'$1$HTQUGYUGYUGwsxQxCp3F/nGc4DCYM/',
 }
```

Resource Type: group

Manage groups. On most platforms this can only create groups. Group membership must be managed on individual users.

```
group { 'sysadmin':
ensure => present,
gid    => '5000',
}
```

ensure

*(**Property:** This attribute represents concrete state on the target system.)*Create or remove the group.
Valid values are present, absent.

gid

*(**Property:** This attribute represents concrete state on the target system.)*
The group ID. Must be specified numerically. If no group ID is specified when creating a new group, then one will be chosen automatically according to local system standards. This will likely result in the same group having different GIDs on different systems, which is not recommended.

22. The PUPPET EXCERCISE

Now we are going to work on an exercise which would give us a detailed understanding of Puppet workflow and its influence on implementing “Infrastructure as Code”

1. Puppet Setup.

1.1 Setup puppet server

The following are the set of commands to setup a puppet server.

Installing puppet package manager:
#yum install -y http://yum.puppetlabs.com/puppetlabs-release-el-6.noarch.rpm

Installing puppet server:
#yum install -y puppet-server

Creating an environment with the the name “Production” and its file structure
#mkdir -p /etc/puppet/environments/production/{modules,manifests}

Configuring the environments
#cd /etc/puppet/environments/production
#echo "modulepath = /etc/puppet/environments/production/modules" >
/etc/puppet/environments/production/environment.conf
echo "environment_timeout = 5s" >> /etc/puppet/environments/production/environment.conf

Open puppet.conf and define the puppet server under [main] section and define the environmentpath and modulepath
[main]
certname = puppetmaster

[master]
environmentpath = $confdir/environments
basemodulepath = $confdir/modules:/opt/puppet/share/modules

In order to communicate with the server one requires to Stop Firewall,
/etc/init.d/iptables stop

After making all the configurations changes, we need to start the puppet server to apply those changes.
service puppetmaster start

1.2 Setup puppet agents
We are taking two nodes where we can make the configuration changes through puppet manifests. Those are named wiki(centos machine) and wikitest(ubuntu).

Setting up wiki
Installing puppet package manager:
#yum install -y http://yum.puppetlabs.com/puppetlabs-release-el-6.noarch.rpm
Install puppet
#yum install -y puppet

Update the puppetmaster info in puppet.conf so that the agent identify the puppet server.

vi /etc/puppet/puppet.conf
[main]
server = puppetmaster

Once we done with changing the puppet configuration file , the agent is needed to request for a ssl certificate from the puppet server.
Run the following puppet command to request cert
puppet agent --verbose --no-daemonize –onetime

Setting up wikitest
Login to wikitest to install puppet

Download the “puppetlabs-release” package for your OS version.
wget https://apt.puppetlabs.com/puppetlabs-release-trusty.deb

Install the package by running
sudo dpkg -i puppetlabs-release-trusty.deb

Updating and installing puppet
sudo apt-get update
sudo apt-get install puppet

Open the agent configuration file and update the info of puppet master
vi /etc/puppet/puppet.conf

[agent]
server = puppetmaster

To make puppet ready from the next system bootings run the following command
puppet agent –enable

Requesting for SSL certificate
puppet agent --verbose --no-daemonize –onetime

Login to puppet server to sign node certs

To list the ssl certificate requests , use the command below.
puppet cert –list

The puppet master needs to accept the requests from agents through the command,
puppet cert sign wiki
puppet cert sign wikitest

1.3 Testing node definitions.

Login to puppetserver
We can define the nodes and their desired state in nodes.pp file in the main manifests file. In the following example we are defining the nodes for the production environment .

```
# vi /etc/puppet/environments/production/manifests/nodes.pp
node 'wiki'{
file { '/info.txt':
ensure => 'present',
content => inline_template("This file was created by puppet at <%= Time.now %>\n"),
        }
}

node 'wikitest' {
file { '/info.txt':
```

```
ensure => 'present',
content => inline_template("This file was created by puppet at <%= Time.now %>\n"),
        }
}
```

We can observe the presence of a new file '/info.txt' on the nodes when we run this manifest file from the agents.

L'ogin to both puppet agent, wiki & wikitest
puppet agent --verbose --no-daemonize –onetime

The command will pull all the configuration changes specified in the manifest files in the puppet master.

On the nodes check for file using the command
cat /info.txt

2. Manifests.

In this section, we are going to discuss some of the most renowned resources like **file, packages** and **service**.

2.1 Managing Files

For managing the files in puppet, we have a resource type 'file'. We use different attributes for efficient use of 'file' resource.

Login to puppet server

/etc/puppet/environments/production/manifests/nodes.pp

```
node 'wiki' {
file { '/info.txt':
ensure => 'present',
content => inline_template("Created by Puppet at <%= Time.now %>\n"),
}
}

node 'wikitest' {
file { '/info.txt':
ensure => 'present',
content => inline_template("Created by Puppet at <%= Time.now %>\n"),
}
}
```

Go to both the nodes and execute to apply changes
puppet agent --verbose --no-daemonize --onetime

2.2 Managing Packages & Services

Package ' resource type is to manage the software installations through the attributes like ensure and name.

‘ service’ resource type is to manage the services like start, stop and restart .

```
node 'wiki' {
file { '/info.txt':
ensure => 'present',
content => inline_template("Created by Puppet at <%= Time.now %>\n"),
}
package { 'ntp':
ensure => 'installed',
}'
service { ntpd:
ensure => 'running',
enable => true,

}
}

node 'wikitest' {
file { '/info.txt':
ensure => 'present',
content => inline_template("Created by Puppet at <%= Time.now %>\n"),

}
package { 'ntp':
ensure => 'installed',
}
service { ntp:
ensure => 'running',
enable => true,
}
}
```

2.3 Selectors

For this exercise, we can use selectors to choose a variable based on the facts or other variables. The following will define different package names based on OS type which can be known from the facts.

```
$ntpservice = $osfamily ? {
'redhat' => 'ntpd',
'debian' => 'ntp',
default => 'ntp',
}

node 'wiki' {
file { '/info.txt':
ensure => 'present',
content => inline_template("Created by Puppet at <%= Time.now %>\n"),

}

package { 'ntp':
ensure => 'installed',
```

```
}

service { $ntpservice:
ensure => 'running',
enable => true,
}
}

node 'wikitest' {
file { '/info.txt':
ensure => 'present',
content => inline_template("Created by Puppet at <%= Time.now %>\n"),
}

package { 'ntp':
ensure => 'installed',
}

service { $ntpservice:
ensure => 'running',
enable => true,
}
}
```

2.4 Using classes

The classes can be defined in module's manifests file or in main manifests's nodes.pp file and can be called from the main manifests file.

```
node 'wiki' {
class { 'linux': }
}

node 'wikitest' {
class { 'linux': }
}

class linux {
$ntpservice = $osfamily ? {
'redhat' => 'ntpd',
'debian' => 'ntp',
default => 'ntp',
}

file { '/info.txt':
ensure => 'present',
content => inline_template("Created by Puppet at <%= Time.now %>\n"),
}

package { 'ntp':
ensure => 'installed',
}
```

```
service { $ntpservice:
ensure => 'running',
enable => true,
}
}
```

2.5 Variables

This section deals with usage of variables and calling them from the resources. We are using a variable '$admintools' to pass the package names to the package resource in the manifest file.

```
node 'wiki' {
class { 'linux': }
}

node 'wikitest' {
class { 'linux': }
}

class linux {

$admintools = ['git', 'nano', 'screen']

package { $admintools:
ensure => 'installed',
}

$ntpservice = $osfamily ? {
'redhat' => 'ntpd',
'debian' => 'ntp',
default => 'ntp',
}

file { '/info.txt':
ensure => 'present',
content => inline_template("Created by Puppet at <%= Time.now %>\n"),
}

package { 'ntp':
ensure => 'installed',
}

service { $ntpservice:
ensure => 'running',
enable => true,
}
}
```

3. Modules

We can create the manifest files in the main manifests nodes.pp file. But we may need to work on so many resources of different resources. For simple understanding of this structure we can use create different modules for specific kind of changes.

3.1 Creating modules

To create a module in the production environment we can use the following set of commands with the name 'imran-mediawiki'

```
# cd /etc/puppet/environments/production/modules
# puppet module generate imran-mediawiki --environment production
# mv imran-mediawiki/ mediawiki
# cd mediawiki
# ls -ltr
# cd manifests
# vi init.pp
```

The 'init.pp' is a file where we place our puppet code.

3.2 Customizing Modules

After creating the module we need to customize it to get a desired state in the nodes.

Login to puppetmaster

```
# cd /etc/puppet/environments/production/modules/mediawiki/manifests

# vi init.pp
class mediawiki {
$phpmysql = $osfamily ? {
'redhat' => 'php-mysql',
'debian' => 'php5-mysql',
default => 'php-mysql',
}

package { $phpmysql:
ensure => 'present',
}
}
```

Open nodes.pp file and invoke mediawiki class

```
# cd /etc/puppet/environments/production/manifests

# vi nodes.pp
node 'wiki'{
class {'linux': }
class {'mediawiki':}

}

node 'wikitest' {
class {'linux': }
class {'mediawiki':}
}
```

.....Rest of the file.....

Run puppet agent from both the nodes to verify
puppet agent --verbose --no-daemonize –onetime

3.3 Conditionals
The conditionals can be used to choose the variable based on the facts or other variables that enable the puppet master to prepare a catalog with the required information.
Here, we are choosing a package name for different Os family. There are two kinds of conditional statements in this example selectors and if conditionals.

Login to puppetmaster

cd /etc/puppet/environments/production/modules/mediawiki/manifests
vi init.pp

```
class mediawiki {
$phpmysql = $osfamily ? {
'redhat' => 'php-mysql',
'debian' => 'php5-mysql',
default => 'php-mysql',
      }
package { $phpmysql:
ensure => 'present',
       }
if $osfamily == 'redhat' {
package {'php-xml':
ensure => 'present',
}
}
}
```

Run puppet agent from both the nodes to verify
puppet agent --verbose --no-daemonize –onetime

4. Puppet Forge Modules.

Puppet forge is a collection of different modules developed by professionals across the world. We can pull them from the puppet forge and can use them to extend the power of puppet.

4.1 Downloading from puppet forge

https://forge.puppet.com/

4.2 Apache module

Login to puppetmaster

We are going to download a puppet module from the puppet forge named ‘puppetlabs-apache’ . This module can be used across puppet nodes of different OS types.

cd /etc/puppet/environments/production/modules
The command to downloading the module is below

puppet module install puppetlabs-apache --modulepath /etc/puppet/environments/production/modules/

We can call this module's classes from the init.pp file

```
# ls -ltr
# cd mediawiki/manifests/
# vi init.pp

class mediawiki {
$phpmysql = $osfamily ? {
'redhat' => 'php-mysql',
'debian' => 'php5-mysql',
default => 'php-mysql',
}
package { $phpmysql:
ensure => 'present',
}
if $osfamily == 'redhat' {
package {'php-xml':
ensure => 'present',
}
}
class { '::apache':
docroot => '/var/www/html',
mpm_module => 'prefork',
subscribe => Package[$phpmysql],

}
class {'::apache::mod::php': }
}
```

Run puppet agent from both the nodes to verify
puppet agent --verbose --no-daemonize –onetime
service httpd status (centos)
service apache2 status (ubuntu)

4.3 VCSREPO MODULE
To pull the code from git repository we need to install the puppet module 'vcsrepo'.
Login to puppetmaster

cd /etc/puppet/environments/production/modules
To download the module, run the following command.
puppet module install puppetlabs-vcsrepo --modulepath /etc/puppet/environments/production/modules/
The command will download the module to /etc/puppet/environments/production/modules/

```
# ls -ltr
# cd mediawiki/manifests/
# vi init.pp
```

```
class mediawiki {
$phpmysql = $osfamily ? {
'redhat' => 'php-mysql',
'debian' => 'php5-mysql',
default => 'php-mysql',
}
package { $phpmysql:
ensure => 'present',
}'
if $osfamily == 'redhat' {
package {'php-xml':
ensure => 'present',
}
}

class { '::apache':
docroot => '/var/www/html',
mpm_module => 'prefork',
subscribe => Package[$phpmysql],
}
class {'::apache::mod::php': }
vcsrepo { '/var/www/html':
ensure => 'present',
provider => 'git',
source => "https://github.com/wikimedia/mediawiki.git"
revision => 'REL1_23'
}
}
```

Run puppet agent from both the nodes to verify
puppet agent --verbose --no-daemonize –onetime

4.4 Resource Ordering
The resource ordering enables the puppet master to prepare a catalog that will run the resources in specified manner.

subscribe — Applies a resource **after** the target resource. The subscribing resource refreshes if the target resource changes.

vi init.pp

```
class mediawiki {
$phpmysql = $osfamily ? {
'redhat' => 'php-mysql',
'debian' => 'php5-mysql',
default => 'php-mysql',
}
package { $phpmysql:
ensure => 'present',
}
if $osfamily == 'redhat' {
package {'php-xml':
```

```
ensure => 'present',
}
}
class { '::apache':
docroot => '/var/www/html',
mpm_module => 'prefork',
subscribe => Package[$phpmysql],
}
class {'::apache::mod::php': }
vcsrepo { '/var/www/html':
ensure => 'present',
provider => 'git',
source => "https://github.com/wikimedia/mediawiki.git",
revision => 'REL1_23',
}

file {'/var/www/html/index.html':
ensure => 'absent',
}
}
```

Run puppet agent from both the nodes to verify

```
# puppet agent --verbose --no-daemonize –onetime
```

This puppet run is gonna remove the index.html file after trying vcsrepo so ultimately it will fail. For this will use resource ordering.

Login to puppetmaster

```
# cd /etc/puppet/environments/production/modules//mediawiki/manifests
# vi init.pp

class mediawiki {
$phpmysql = $osfamily ? {
'redhat' => 'php-mysql',
'debian' => 'php5-mysql',
default => 'php-mysql',
}
package { $phpmysql:
ensure => 'present',
}
if $osfamily == 'redhat' {
package {'php-xml':
ensure => 'present',
}
}
class { '::apache':
docroot => '/var/www/html',
mpm_module => 'prefork',
subscribe => Package[$phpmysql],
}
class {'::apache::mod::php': }
```

```
vcsrepo { '/var/www/html':
ensure => 'present',
provider => 'git',
source => "https://github.com/wikimedia/mediawiki.git",
revision => 'REL1_23',
}
file {'/var/www/html/index.html':
ensure => 'absent',
}'
File['/var/www/html/index.html'] -> Vcsrepo ['/var/www/html']
}
```

Run puppet agent from both the nodes to verify

```
# touch /var/www/html/index.html
# puppet agent --verbose --no-daemonize –onetime
```

4.5 MYSQL MODULE

Login to puppetmaster

```
# cd /etc/puppet/environments/production/modules
# puppet module install puppetlabs-mysql --version 3.9.0 --modulepath

/etc/puppet/environments/production/modules/
# ls -ltr
# cd mediawiki/manifests/
# vi init.pp
class mediawiki {
$phpmysql = $osfamily ? {
'redhat' => 'php-mysql',
'debian' => 'php5-mysql',
default => 'php-mysql',
}
package { $phpmysql:
ensure => 'present',
}
if $osfamily == 'redhat' {
package {'php-xml':
ensure => 'present',
}
}
class { '::apache':
docroot => '/var/www/html',
mpm_module => 'prefork',
subscribe => Package[$phpmysql],
}
class {'::apache::mod::php': }
vcsrepo { '/var/www/html':
ensure => 'present',
provider => 'git',
source => "https://github.com/wikimedia/mediawiki.git",
```

revision => 'REL1_23',
}
file {'/var/www/html/index.html':
ensure => 'absent',
}
File['/var/www/html/index.html'] -> Vcsrepo ['/var/www/html']
class {'::mysql::server':
root_password => 'training',
}'
}
Run puppet agent from both the nodes to verify
puppet agent --verbose --no-daemonize –onetime

4.6 FIREWALL MODULE

Login to puppetmaster

cd /etc/puppet/environments/production/modules
puppet module install puppetlabs-firewall --modulepath
/etc/puppet/environments/production/modules/
# ls -ltr
# cd mediawiki/manifests/
# vi init.pp
class mediawiki {
$phpmysql = $osfamily ? {
'redhat' => 'php-mysql',
'debian' => 'php5-mysql',
default => 'php-mysql',
}
package { $phpmysql:
ensure => 'present',
}
if $osfamily == 'redhat' {
package {'php-xml':
ensure => 'present',
}
}
class { '::apache':
docroot => '/var/www/html',
mpm_module => 'prefork',
subscribe => Package[$phpmysql],
}
class {'::apache::mod::php': }
vcsrepo { '/var/www/html':
ensure => 'present',
provider => 'git',
source => "https://github.com/wikimedia/mediawiki.git",
revision => 'REL1_23',
}
file {'/var/www/html/index.html':
ensure => 'absent',
}

File['/var/www/html/index.html'] -> Vcsrepo ['/var/www/html']
class {'::mysql::server':
root_password => 'training',
}
class { 'firewall': }
firewall { '000 allow http access':
port => '80',
proto => 'tcp',
action => 'accept',
}
}

Run puppet agent from both the nodes to verify

puppet agent --verbose --no-daemonize –onetime

5. Setup MediaWiki Site.

Now since our mediawiki server is deployed to wiki and wikitest, lets go ahead check that out. Open up a browser and enter wiki server IP, you should see the mediawiki webpage as shown below.

Follow the screenshots to setup mediawiki site.