Puppet-IPA

A FreeIPA Puppet module by James

Available from:

https://github.com/purpleidea/puppet-ipa/

Also available from:

https://gitorious.org/purpleidea/puppet-ipa/

This documentation is available in: Markdown or PDF format.

Table of Contents

1. Overview
2. Module description - What the module does
3. Setup - Getting started with Puppet-IPA
    - What can Puppet-IPA manage?
    - Basic setup
    - Advanced setup
    - Multi-master setup
    - Type setup
    - Client setup
4. Usage/FAQ - Notes on management and frequently asked questions
5. Reference - Class and type reference
    - ipa::server
    - ipa::server::host
    - ipa::server::service
    - ipa::server::user
    - ipa::client::deploy
6. Examples - Example configurations
7. Limitations - Puppet versions, OS compatibility, etc...
8. Development - Background on module development and reporting bugs
9. Author - Author and contact information

Overview

The Puppet-IPA module installs, configures, and manages a FreeIPA server, replicas, and clients.

Module Description

This Puppet-IPA module handles installation, configuration, and management of FreeIPA across all of the replicas and clients in your infrastructure.

Setup

What can Puppet-IPA manage?

Puppet-IPA is designed to be able to manage as much or as little of your FreeIPA infrastructure as you wish. All features are optional. If there is a feature that doesn't appear to be optional, and you believe it should be, please let me know. Having said that, it makes good sense to me to have Puppet-IPA manage as much of your FreeIPA infrastructure as it can. At the moment, it cannot rack new servers, but I am accepting funding to explore this feature ;) At the moment it can manage:

- FreeIPA packages (rpm)
- FreeIPA server installation (ipa-server-install)
- FreeIPA configuration files (/etc/ipa/)
- FreeIPA replica peering (ipa-replica-{prepare,install})
- FreeIPA replica topology (ipa-replica-manage)
- FreeIPA firewalling (whitelisting)
- FreeIPA host creation/modification/deletion (ipa host-{add,mod,del})
- FreeIPA service creation/modification/deletion (ipa service-{add,mod,del})
- FreeIPA user creation/modification/deletion (ipa user-{add,mod,del})
- FreeIPA client installation (ipa-client-install, ipa-getkeytab)
- And much more...

Basic setup

For a single host FreeIPA server, setup is quite simple and straight-forward.

```
node ipa1 {
    class { '::ipa::server':
        dm_password => 'changeme',  # pick a dm password
        admin_password => 'changeme2',  # pick an admin password
```

```
        shorewall => true,
    }
}
```

Please be careful to keep the the *dm* and *admin* passwords secret. For a better way to do this, please have a look at the advanced setup.

Advanced setup

Storing secrets in puppet is generally a bad idea. Because FreeIPA is usually the security cornerstone of your infrastructure, care was taken to ensure that this puppet modules does all that it can to survice even the harshest scrutiny.

If desired, the local FreeIPA server can securely generate *dm* and *admin* passwords, and encrypt them with your public GPG key. This way, you can have an automatic FreeIPA deployment without ever having the secrets touch puppet. For more information on this technique and how it works with puppet please read: Securely managing secrets for FreeIPA with Puppet.

Here is an example of how you can use the GPG parameters:

```
node ipa1 {
    class { '::ipa::server':
        # NOTE: email must exist in the public key if we use gpg_sendemail
        #email => 'root@example.com',
        gpg_recipient => '24090D66',    # use your own GPG key id here!
        #gpg_publickey => '...',    # alternatively, paste it here!
        gpg_keyserver => 'hkp://keys.gnupg.net',    # pick your own
        gpg_sendemail => false,     # optional choice you can make.
        shorewall => true,
    }
}
```

For information on how Puppet-IPA can be used for hybrid management of FreeIPA types, please read: Hybrid management of FreeIPA types with Puppet.

For information on additional advanced options, please see the reference section below for the specifics.

Multi-master setup

Puppet-IPA can be used to setup a cluster of FreeIPA servers. Each server in the cluster becomes a FreeIPA replica. There are some additional requirements and (unfortunately) limitations at this time. You will require:

- An additional IP address to be used as a VIP. (It can be a private address.)
- More than one server or virtual machine.

The limitation is that Securely managing secrets for FreeIPA with Puppet is not currently compatible with the automatic multi-master deployment. This is due to a limitation in FreeIPA, and it will hopefully be fixed in a future release. Help is appreciated!

```
node /^ipa\d+$/ {   # ipa{1,2,..N}
    class { '::ipa::server':
        vip => '192.168.123.253',   # pick an available VIP
        topology => 'ring',     # choose your favourite
        dm_password => 'changeme',  # pick a dm password
        admin_password => 'changeme2',  # pick an admin password
        # NOTE: Unfortunately the gpg_* options are not currently
        # compatible with automatic multi-master replication. This is
        # due to a limitation in FreeIPA, but should hopefully get
        # resolved in a future FreeIPA release. Help is appreciated!
        vrrp => true,           # setup keepalived
        shorewall => true,
    }
}
```

The server configuration that you choose must be identical on each FreeIPA server. As you can see in the above example, this is achieved by including a *node* pattern that matches ipa{1,2,...N} so that this is automatic.

Another point of interest is the *topology* parameter. This automatically figures out the correct relationships for all of the hosts in your cluster algorithmically, so that you can worry about other issues. Other algorithms can be written and included, and contributions are encouraged. You can also specify the topology manually if you want to be extremely hands on with your layout.

Type setup

Naturally you'll probably want to define FreeIPA types on your server. The most common ones are *host*, *service*, and *user*. These should be defined on all the server hosts. Some examples include:

```
# create a managed user
ipa::server::user { 'ntesla':
    first => 'Nikola',
    last => 'Tesla',
    city => 'Shoreham',
```

```
    state => 'New York',
    postalcode => '11786',
}

# create a host
ipa::server::host { 'nfs':  # NOTE: adding .${domain} is a good idea....
    domain => 'example.com',
    macaddress => "00:11:22:33:44:55",
    random => true,     # set a one time password randomly
    locality => 'Montreal, Canada',
    location => 'Room 641A',
    platform => 'Supermicro',
    osstring => 'CentOS 6.5 x86_64',
    comment => 'Simple NFSv4 Server',
    watch => true,  # read and understand the docs well
}

# create a service for the above host
ipa::server::service { 'nfs':
    service => 'nfs',
    host => 'nfs',          # needs to match ipa::server::host $name
    domain => 'example.com',    # used to figure out realm
}
```

For more information on FreeIPA type management, and the hybrid management feature that Puppet-IPA supports, please read: Hybrid management of FreeIPA types with Puppet.

Client setup

Getting a client host to enroll and work magically with the FreeIPA server is particularly easy. Simply include the IPA client *deploy* class:

```
# there is now a single "deploy" which can be used for both hosts and services!
include ipa::client::deploy
```

This will automatically pull down any host registration and service definitions that were defined on the Puppet-IPA server. If you want to be more specific about which you include, you can include each set of types separately:

```
# if you use fqdn's for the ipa:server:host $name's, then you can deploy with:
include ipa::client::host::deploy
```

or to only include services:

```
# pull down any defined FreeIPA services.
include ipa::client::host::deploy
```

For an NFS host (which is a FreeIPA client), you might want to use:

```
# and on the nfs server (an ipa client):
class { '::ipa::client::deploy':
    nametag => 'nfs',   # needs to match the ipa:server:host $name
}
```

All of this happens automatically through the magic of puppet and exported resources. See the examples for more ideas.

Usage and frequently asked questions

All management should be done by manipulating the arguments on the appropriate Puppet-IPA classes and types. Hybrid management is also supported for certain aspects of FreeIPA management. This is a stellar feature of Puppet-IPA. Please read: Hybrid management of FreeIPA types with Puppet.

Do I need to use a virtual IP?

Using a virtual IP (VIP) is strongly recommended as a distributed lock manager (DLM) for certain operations. For an article explaning the mechanism (but for a different puppet module), please see: How to avoid cluster race conditions or: How to implement a distributed lock manager in puppet

Remember that even if you're using a hosted solution (such as AWS) that doesn't provide an additional IP address, or you want to avoid using an additional IP, you can use an unused private RFC1918 IP address as the DLM VIP. Remember that a layer 3 IP can co-exist on the same layer 2 network with the layer 3 network that is used by your cluster.

Is it possible to have Puppet-IPA complete in a single run?

No. This is a limitation of Puppet, and is related to how FreeIPA operates. It is possible for a single FreeIPA server, but for many multi-host scenarios, including the multi-master FreeIPA case, you will require more than one run.

For example,

Can you integrate this with vagrant?

Yes, see the vagrant/ directory. This has been tested on Fedora 20, with vagrant-libvirt, as I have no desire to use VirtualBox for fun. I have written many articles about this on my technical blog. In particular, I would recommend: Vagrant on Fedora with libvirt (reprise).

Puppet runs fail with "Connection refused - connect(2)" errors.

You may see a "*Connection refused - connect(2)*" message when running puppet. This typically happens if your puppet vm guest is overloaded. When running high guest counts on your laptop, or running without hardware virtualization support this is quite common. Another common causes of this is if your domain type is set to *qemu* instead of the accelerated *kvm*. Since the *qemu* domain type is much slower, puppet timeouts and failures are common when it doesn't respond.

Will this work on my favourite OS? (eg: GNU/Linux F00bar OS v12 ?)

If it's a GNU/Linux based OS, can run FreeIPA, and Puppet, then it will probably work. Typically, you might need to add a yaml data file to the *data/* folder so that Puppet-IPA knows where certain operating system specific things are found. The multi-distro support has been designed to make it particularly easy to add support for additional platforms. If your platform doesn't work, please submit a yaml data file with the platform specific values.

Awesome work, but it's missing support for a feature and/or platform!

Since this is an Open Source / Free Software project that I also give away for free (as in beer, free as in gratis, free as in libre), I'm unable to provide unlimited support. Please consider donating funds, hardware, virtual machines, and other resources. For specific needs, you could perhaps sponsor a feature!

You didn't answer my question, or I have a question!

Contact me through my technical blog and I'll do my best to help. If you have a good question, please remind me to add my answer to this documentation!

Reference

Please note that there are a number of undocumented options. For more information on these options, please view the source at: https://github.com/

purpleidea/puppet-ipa/. If you feel that a well used option needs documenting here, please contact me.

Overview of classes and types

Please note that the most common, user facing classes and types are documented, while there may be many other undocumented classes and types that are used internally by other classes and types. These will be documented as time allows.

- ipa::server: Base class for server hosts.
- ipa::server::host: Host type for each FreeIPA client.
- ipa::server::service: Service type for each FreeIPA service.
- ipa::server::user: User type for each FreeIPA user.
- ipa::client::deploy: Client class to deploy types.

ipa::server

This is the main class to be used for IPA server installation. It is used for both simple standalone FreeIPA servers, and for complex multi-master scenarios.

`hostname` The hostname of the IPA server. This defaults to *$::hostname*.

`domain` The domain of the IPA server and of the cluster. This defaults to *$::domain*.

`realm` The realm of the cluster. This defaults to *upcase($domain)*.

`vip` The virtual IP address to be used for the cluster distributed lock manager. This option can be used in conjunction with the *vrrp* option, but it does not require it. If you don't want to provide a virtual ip, but you do want to enforce that certain operations only run on one host, then you can set this option to be the ip address of an arbitrary host in your cluster. Keep in mind that if that host is down, certain options won't ever occur.

`peers` Specify the peering topology manually in dictionary form. Each dictionary value should be an array of the peers to connect to from the originating key.

topology The topology algorithm to use when setting up mult-master cluster automatically. A few default algorithms are included with Puppet-IPA. They are: *ring*, *flat*. If you'd like to include an algorithm that generates a different topology, it is easy to drop one in! Please contact me with the details!

topology_arguments A list of arguments to pass to the topology algorithm. Not all topology algorithms support this, however it can be very useful so that it's easy to generalize certain algorithms in the same function in terms of these variables.

dm_password The dm_password in plaintext. It is recommended that you use a better mechanism for handling secrets. Please see: Securely managing secrets for FreeIPA with Puppet.

admin_password The admin_password in plaintext. It is recommended that you use a better method for handling secrets. Please see: Securely managing secrets for FreeIPA with Puppet.

gpg_recipient The GPG recipient ID of your public key goes here. This is a valid *-r* value for the *gpg* command line tool.

gpg_publickey This is the value of your GPG public key, or a *puppet:///* uri pointing to the file. This can't be used in conjunction with the **gpg_keyserver** option.

gpg_keyserver This is the value of the GPG keyserver of your choice. You can use a public one such as *hkp://keys.gnupg.net* or you can use your own private keyserver.

gpg_sendemail Do you want to mail out the *dm* and *admin* passwords encrypted with your GPG key or not? Defaults to *false*.

idstart The FreeIPA *idstart* value to use. This is the starting id for users created. This comes with a mostly sensible default, but recommendations are welcome.

idmax The FreeIPA *idmax* value to use.

email_domain The email domain to use. This defaults to *$domain*.

shell The default user shell to assign. This default to */bin/sh*.

homes The default home prefix. This defaults to */home*.

ntp Should we install an NTP server along with FreeIPA? This defaults to *false*.

dns Should we install a DNS server along with FreeIPA? This defaults to *false*. This must be set at install time to be used.

dogtag Should we install a cert server along with FreeIPA? This defaults to *false*. This is not currently managed by Puppet-IPA.

email The email address to associate with the FreeIPA server. This defaults to *root@$domain*. This is important for FreeIPA, and it is used to mail out encrypted passwords depending on your `gpg_sendemail` settings.

vrrp Whether to automatically deploy and manage *Keepalived* for use as a *DLM* and for use in volume mounting, etc... Using this option requires the *vip* option.

shorewall Boolean to specify whether puppet-shorewall integration should be used or not.

again Do you want to use *Exec['again']* ? This helps build your cluster quickly!

host_excludes This list matches and excludes certain *hosts* from being removed by puppet. The `host_excludes` are matched with bash regexp matching in: `[[ =~ ]]`. If the string regexp passed contains quotes, string matching is done:

```
$string='"hostname.example.com"' vs $regexp='hostname.example.com'
```

Obviously, each pattern in the array is tried, and any match will do. Invalid expressions might cause breakage! Use this at your own risk!! Remember that you might be matching against strings which have dots. A value of true will automatically add the `*` character to match all. For more information on this option, please read: Hybrid management of FreeIPA types with Puppet.

service_excludes This list matches and excludes certain *services* from being removed by puppet. The `service_excludes` are matched with bash regexp matching in: `[[ =~ ]]`. If the string regexp passed contains quotes, string matching is done:

```
$string='"hostname.example.com"' vs $regexp='hostname.example.com'
```

Obviously, each pattern in the array is tried, and any match will do. Invalid expressions might cause breakage! Use this at your own risk!! Remember that you might be matching against strings which have dots. A value of true will automatically add the `*` character to match all. For more information on this option, please read: Hybrid management of FreeIPA types with Puppet.

user_excludes This list matches and excludes certain *users* from being removed by puppet. The `user_excludes` are matched with bash regexp matching in: `[[ =~ ]]`. If the string regexp passed contains quotes, string matching is done:

```
$string='"hostname.example.com"' vs $regexp='hostname.example.com'
```

Obviously, each pattern in the array is tried, and any match will do. Invalid expressions might cause breakage! Use this at your own risk!! Remember that you might be matching against strings which have dots. A value of true will automatically add the `*` character to match all. For more information on this option, please read: Hybrid management of FreeIPA types with Puppet.

peer_excludes This list matches and excludes certain *peers* from being removed by puppet. The `peer_excludes` are matched with bash regexp matching in: `[[ =~ ]]`. If the string regexp passed contains quotes, string matching is done:

```
$string='"hostname.example.com"' vs $regexp='hostname.example.com'
```

Obviously, each pattern in the array is tried, and any match will do. Invalid expressions might cause breakage! Use this at your own risk!! Remember that you might be matching against strings which have dots. A value of true will automatically add the `*` character to match all. This particular option is difference than the other excludes because it only prevents un-peering of the listed hosts.

ipa::server::host

This is the main FreeIPA type that maps to host entries in FreeIPA. This gets set on the FreeIPA server (or servers) and with the associated *ipa::client* types and classes, will automatically setup the FreeIPA client associated with this host type. Here are a few examples:

```
# create a host
ipa::server::host { 'nfs':  # NOTE: adding .${domain} is a good idea....
    domain => 'example.com',
    macaddress => "00:11:22:33:44:55",
    random => true,     # set a one time password randomly
    locality => 'Montreal, Canada',
    location => 'Room 641A',
    platform => 'Supermicro',
    osstring => 'CentOS 6.5 x86_64',
    comment => 'Simple NFSv4 Server',
    watch => true,  # read and understand the docs well
}

ipa::server::host { 'test1':
    domain => 'example.com',
    password => 'password',
    watch => true,  # read and understand the docs well
}
```

domain The domain of the host. This defaults to the ipa server *$domain* variable.

server Where the client will find the IPA server. This has a sensible default.

macaddress The MAC address of the host.

sshpubkeys Leave this at the default for automatic public ssh keys to get transferred.

password The one time password used for host provisioning. Not to be used with `$random`.

random Generate the one time password used for host provisioning. Conflicts with `$password`.

locality Host description parameter for *locality*. Example: "*Montreal, Canada*".

location Host description parameter for *location*. Example: "*Lab 42*".

platform Host description parameter for hardware *platform*. Example: "*Lenovo X201*".

osstring Host description parameter for *os string*. Example: "*CentOS 6.5*".

comments Host description parameter for *comments*. Example: "*NFS Server*".

admin Should this client get the admin tools installed ?

watch Manage all changes to this resource, reverting others if this is true. For more information on this option please read: Hybrid management of FreeIPA types with Puppet.

modify Modify this resource on puppet changes or not ? Do so if true. For more information on this option please read: Hybrid management of FreeIPA types with Puppet.

ipa::server::service

This is the main FreeIPA type that maps to service entries in FreeIPA. This is set on the FreeIPA server (or servers) and with the associated *ipa::client* types and classes, will automatically setup the FreeIPA service associated with the correct host. Here is an example:

```
ipa::server::service { 'nfs':   # this $name should match nfs::server => $ipa
    service => 'nfs',
    host => 'nfs',      # needs to match ipa::server::host $name
    domain => 'example.com',    # used to figure out realm
}
```

service The service string. Common examples include: *nfs*, *HTTP*, *ldap*.

host The hostname where the service will reside.

domain The domain of the host where the service will reside.

realm The realm of the host where the service will reside.

principal Specify the desired principal of this service, overriding the principal that can be ascertained by using the above values.

server Where the client will find the IPA server. This has a sensible default.

pactype The service *pac type*. Bad values are silently discarded, `[]` is *NONE*.

watch Manage all changes to this resource, reverting others if this is true. For more information on this option please read: Hybrid management of FreeIPA types with Puppet.

modify Modify this resource on puppet changes or not ? Do so if true. For more information on this option please read: Hybrid management of FreeIPA types with Puppet.

comment A comment field that you can use however you want.

ipa::server::user

This is the main FreeIPA type that maps to user entries in FreeIPA. This is set on the FreeIPA server (or servers) and creates user entries which can be seen in the FreeIPA LDAP database, and which are then available on IPA clients. Here are a few examples:

```
# create a managed user
ipa::server::user { 'ntesla':
    first => 'Nikola',
    last => 'Tesla',
    city => 'Shoreham',
    state => 'New York',
    postalcode => '11786',
}

# create a user by principal but without the instance set
ipa::server::user { 'arthur@EXAMPLE.COM':
    first => 'Arthur',
    last => 'Guyton',
    jobtitle => 'Physiologist',
    orgunit => 'Research',
}
```

```
# create a user using a full principal as the primary key
# NOTE: the principal itself can't be edited without a remove/add
ipa::server::user { 'aturing/admin@EXAMPLE.COM':
    first => 'Alan',
    last => 'Turning',
    random => true,     # set a password randomly
    password_file => true,  # store the password in plain text ! (bad)
}
```

login The login of this user. In the principal, the pattern is: `login/instance@REALM`.

instance The instance of this user. In the principal, the pattern is: `login/instance@REALM`.

domain The domain associated with the user. Uppercase version can be used as the realm default.

realm The realm of this user. In the principal, the pattern is: `login/instance@REALM`.

principal Specify the desired principal of this user, overriding the principal that can be ascertained by using the above values.

first First name of user. Required.

last Last name of user. Required.

cn Full name of user. Defaults to: "$firstlast$".

displayname Display name of user. Defaults to: "$firstlast$".

initials Initials of user. Defaults to: "$first[0]last[0]$".

email Email address of user.

gecos Legacy field. Can be set manually if needed.

uid UID value for user. By default this is automatically assigned.

gid GID value for user. By default this is automatically assigned.

shell Shell for user. By default this is automatically assigned.

home Home dir for user. By default this is automatically assigned.

sshpubkeys Public SSH keys for the user.

random Set to *true* to have the user password auto generated.

password_file Save user password to a file. The file is in: *${vardir}/ipa/users/passwords/*.

password_mail Mail out a GPG encrypted password to the admin. This requires that the minimum requirements for this to work are set in the ipa::server class.

street The users street address.

city The users city.

state The users state or province.

postalcode The users zip or postal code.

phone The users phone number. Can be an array of numbers.

mobile The users mobile number. Can be an array of numbers.

pager The users pager number. Can be an array of numbers. Users with pager numbers are particularly cool.

fax The users fax number. Can be an array of numbers.

jobtitle The users job title. Silly titles are allowed.

orgunit The users organization unit, otherwise known as a department.

manager The users manager. Should match an existing FreeIPA user name.

carlicense The users car license string. FreeIPA created these fields, I just wrap them.

watch Manage all changes to this resource, reverting others if this is true. For more information on this option please read: Hybrid management of FreeIPA types with Puppet.

modify Modify this resource on puppet changes or not ? Do so if true. For more information on this option please read: Hybrid management of FreeIPA types with Puppet.

comment A comment field that you can use however you want.

ipa::client::deploy

Include this class to deploy the client host itself and any services. The necessary information will automatically get exported from the FreeIPA server. This class takes care of fetching this information through exported resources, and safely running *ipa-getkeytab* so that the admin sees an automatic process.

Examples

For example configurations, please consult the examples/ directory in the git source repository. It is available from:

https://github.com/purpleidea/puppet-ipa/tree/master/examples

It is also available from:

https://gitorious.org/purpleidea/puppet-ipa/source/examples

Limitations

This module has been tested against open source Puppet 3.2.4 and higher.

The module is routinely tested on:

- CentOS 6.5

It will probably work without incident or without major modification on:

- CentOS 5.x/6.x
- RHEL 5.x/6.x

It has patches to support:

- Fedora 20+

It will most likely work with other Puppet versions and on other platforms, but testing on those platforms has been minimal due to lack of time and resources.

Testing is community supported! Please report any issues as there are a lot of features, and in particular, support for additional distros isn't well tested. The multi-distro architecture has been chosen to easily support new additions. Most platforms and versions will only require a change to the yaml based data/ folder.

Development

This is my personal project that I work on in my free time. Donations of funding, hardware, virtual machines, and other resources are appreciated. Please contact me if you'd like to sponsor a feature, invite me to talk/teach or for consulting.

You can follow along on my technical blog.

To report any bugs, please contact me.

Author

Copyright (C) 2012-2013+ James Shubin

- github
- @purpleidea
- https://ttboj.wordpress.com/