CHEF & MCOLLECTIVE

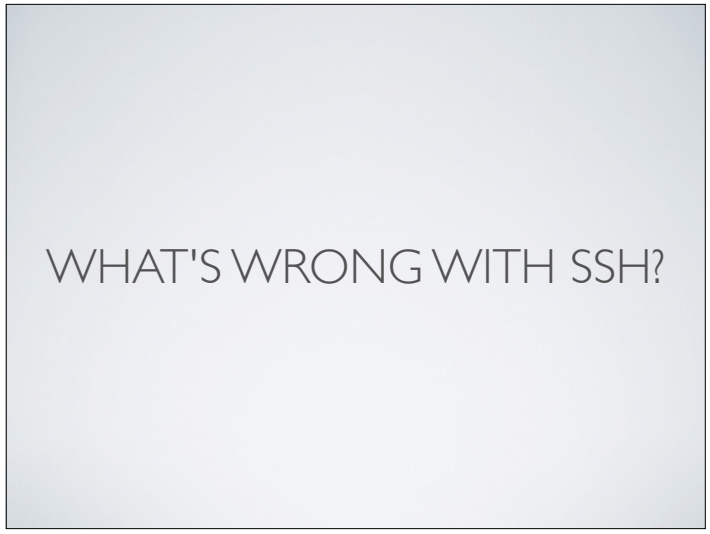
http://www.etsy.com/uk/listing/116941841/chef-marionette-made-to-order

---

WHO AM I?

- zts on Freenode IRC
- @zts on Twitter
- github.com/zts

You want finer control of chef-client runs than is offered by scheduling with a splay. You're happy with scheduled runs ensuring the state of the system, but sometimes want extra runs ahead of schedule. You want to build other tools that interact with chef-client on your nodes.



The problem with using SSH isn't anything it does, but all of the things it doesn't. This isn't immediately obvious in the simple case, particularly if you're a sysadmin.

REQUIREMENTS ANALYSIS

Objective:

Run chef-client,

as the root user,

on all database servers.

---

1. DISCOVERY

- Identifying the database servers

Chef-server keeps a record of our nodes, so discovery is easy - “knife ssh” makes good use of that.

2. TRANSPORT

- Connecting to an individual database server

SSH is a tempting transport due to its ubiquity, but is not without its drawbacks. For one thing, it's point to point. For another, its original intent is to provide a shell on a remote system. Limit to a single command => forcecommand is okay. Multiple?

---

3. AUTHENTICATION

- Logging in to a server

A secure connection alone is not enough - we only want to allow some users to do it. SSH has us covered here, too. If you require key-based authentication, a successful connection to an account can only be made by someone who has the authorised key.

4. AUTHORISATION

- Running chef-client as root

Back to “ssh doesn’t help us” territory. We need to run Chef as a privileged user, so the next step is to do that. You probably use `sudo` to do this.

---

5. AUDITING

- Log all the things

All the logs, all over the place - wtmp, messages, secure, chef, etc. Pulling them together is your responsibility.

So, SSH does its job well, but it only covers a few of the things we care about. It's not a big problem when you just want your sysadmins to kick Chef, but it quickly becomes a challenge to expose a wider range of fine-grained actions to a broader group of users and tools.

At the time, Rundeck looked like a pain to work with, and not very flexible. Capistrano/Fabric? Today - Juju/Saltstack?

"The Marionette Collective, AKA mcollective, is a framework to build server orchestration or parallel job execution systems."

– http://docs.puppetlabs.com/mcollective/

Released by R.I.Pienaar at UK Scale Camp in December 2009. Puppet Labs acquired the project in October 2010. Chef cookbook released July 2011, but other folks were using it before then. MCO docs already included ohai as a fact source, Nicolas Szalay had released some plugins (https://github.com/rottenbytes/mcollective 3/2011), Jonathan Weiss talked about building with mco and chef-solo (http://www.slideshare.net/jweiss/build-your-own-clouds-with-chef-and-mcollective)

MCOLLECTIVE IS…

- Node Discovery
- Message-oriented transport layer
- Authentication and Authorisation
- Host for server-side code
- Library and framework for client-side tools

1. DISCOVERY

Chef-server keeps a record of our nodes, so discovery is easy - “knife ssh” makes good use of that.

DISCOVERY

- Find nodes to operate on by matching various criteria

Find a set of nodes to operate on, using criteria including name, facts (think ohai data), and applied roles and recipes.

DISCOVERY

```
$ mco find -v
Discovering hosts using the mc method for 2 second(s) .... 6

illuminati.cryptocracy.com
monitoring.nat0.cryptocracy.com
mco-x509.nat0.cryptocracy.com
test-precise.nat0.cryptocracy.com
chef-11.nat0.cryptocracy.com
ci.nat0.cryptocracy.com

Discovered 6 nodes in 2.00 seconds using the mc discovery
plugin
```

The simplest possible discovery - everything in the collective.

DISCOVERY FILTERS

Identity: Exact or regex match node name

```
$ mco find -I '/^test-/'
test-precise.nat0.cryptocracy.com
test-centos.nat0.cryptocracy.com

$ mco find -I ‘test-precise.nat0.cryptocracy.com’
test-precise.nat0.cryptocracy.com
```

DISCOVERY FILTERS

Classes: Roles and recipes in the expanded run-list

```
$ mco find -C role.chef-client-11
chef-11.nat0.cryptocracy.com
test-precise.nat0.cryptocracy.com
mco-x509.nat0.cryptocracy.com
```

Square brackets aren't permitted in class names, so we munge these into "role.name" and "recipe.name".

DISCOVERY FILTERS

Agents: plugins loaded on the server

```
$ mco find -A chef
illuminati.cryptocracy.com
monitoring.nat0.cryptocracy.com
mco-x509.nat0.cryptocracy.com
test-precise.nat0.cryptocracy.com
chef-11.nat0.cryptocracy.com
ci.nat0.cryptocracy.com
```

Agents package up (generally related) actions. If you had an agent to manage, say, Postfix, you'd probably only want to install it on your mail servers. I might be missing the point on this one, but I see this as a proxy for configuration management classes.

DISCOVERY FILTERS

Facts: key/value data from the server

```
$ mco facts platform
Report for fact: platform

        centos                                  found 2 times
        ubuntu                                  found 3 times

Finished processing 5 / 5 hosts in 1247.10 ms

$ mco find -F platform=centos
test-centos6.nat0.cryptocracy.com
ci.nat0.cryptocracy.com
```

We use a subset of the ohai data by default.

DATA PLUGINS

```
$ mco rpc rpcutil get_data source=fstat query=/etc/issue

illuminati.cryptocracy.com
           atime: 2014-01-27 22:22:06
       atime_age: 521356
   atime_seconds: 1390861326
           ctime: 2013-02-06 17:27:58
       ctime_age: 31211004
   ctime_seconds: 1360171678
             gid: 0
             md5: 95a5a06e937cbbb554fd483e62c1c156
            mode: 100644
           mtime: 2013-01-25 11:31:25
       mtime_age: 32269197
   mtime_seconds: 1359113485
            name: /etc/issue
          output: present
         present: 1
            size: 26
            type: file
             uid: 0
```

Data plugins let you retrieve pieces of information from the system running mcollective. In contrast to facts, these are completely dynamic, and loaded on every request. You can use them for discovery, and you can use them within your agents.

DATA PLUGINS

```
$ mco rpc rpcutil get_data source=fstat query=/etc/issue \
                          -I illuminati.cryptocracy.com | grep md5
             md5: 95a5a06e937cbbb554fd483e62c1c156

$ mco find -S "fstat('/etc/issue').md5=95a5a06e937cbbb554fd483e62c1c156"
illuminati.cryptocracy.com
mco-x509.nat0.cryptocracy.com
chef-11.nat0.cryptocracy.com
```

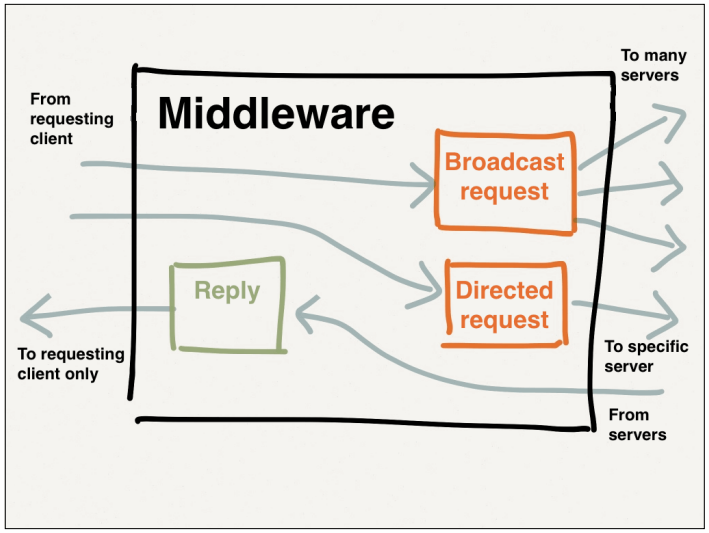
I've shown fstat as it's in the core packages. Other things you might use it for: sysctl values, size of the mailqueue, NTP stratum, number of running Apache processes, etc.

---

2. TRANSPORT

MIDDLEWARE

- Message-Oriented
- Client Requests
  - Directed to some nodes, or broadcast to all
- Server Responses
  - Directed to the requesting client



http://docs.puppetlabs.com/mcollective/overview_components.html

CONNECTOR PLUGINS

- Stomp-based protocol
  - ActiveMQ
  - RabbitMQ
- Alternatives are possible
  - non-production Redis connector

ActiveMQ is perhaps best supported, although RabbitMQ is generally known to work.

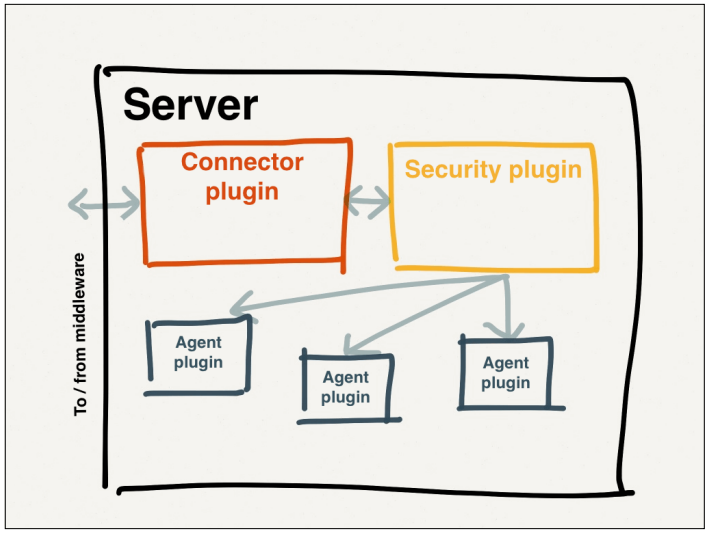
---

3. AUTHENTICATION

MESSAGE SECURITY

- Encode outbound messages
- Validate inbound messages

At its simplest, this isn't much - but the various available plugins offer plenty of choice.



From http://docs.puppetlabs.com/mcollective/overview_components.html

SECURITY PLUGINS

- PSK
  - Simple digest authentication with a shared key - not recommended for production.
  - Callers identified by local uid
- SSL
  - Public/private keypairs - one for all servers, one for each client
  - Callers identified by key
  - Signs but does not encrypt
- AES
  - Every server can have its own keypair
  - Encrypted payloads
  - Optional key distribution

These options are all included in the core packages. The SSL plugin is the perhaps most commonly used, and is the recommended option.

---

4. AUTHORISATION

Optional authorisation plugin can decide whether a request is permitted. ActionPolicy is a functional example plugin.
http://docs.puppetlabs.com/mcollective/simplerpc/authorization.html

5. AUDITING

Optional audit plugin - log to a local file, to an mcollective agent over the network, or anything else you want.

http://docs.puppetlabs.com/mcollective/simplerpc/auditing.html

---

A BRIEF TANGENT...

Pluggability

As you may have gathered, everything is implemented as plugins. While
the core plugins offer most of what you'll need, you can easily swap out any (or all) of them to fit your needs. Kind of like Chef, but better - enable replacement plugins with configuration, not monkeypatches.

PLUGIN TYPES

- Common
  - Connector
  - Security
- Client
  - Application
  - Discovery
- Server
  - Data
  - Facts
  - Agent
  - Authorisation & Audit
  - Registration

Actually, Agent plugin metadata (but not the implementation) must be installed on clients, too.

Connector -> redis

Security -> sshkey, x509

Discovery -> flatfile, mongodb, chef-server

Audit -> local file, centralrpclog

BONUS SECURITY PLUGINS

- sshkey
  - non-core
  - uses SSH keys for signing and validation
  - Optional key distribution
- x509
  - Implemented by Venda
  - Key+Certificate for each server & client
  - Payloads signed but not encrypted
  - Validates message is signed by caller's cert, and cert issued by trusted CA.

https://github.com/VendaTech/mcollective-x509-security.git

AGENTS, CLIENTS, & APPLICATIONS

Up to this point, I've talked about plumbing - the foundations we're going to build on. Sure, you can write your own security or connector plugins, but you probably won't. Agent, Applications and Clients are where we really start to express ourselves.

AGENTS

- Receive a message from a client
- Perform some action
- Return some data

Two parts: DDL and implementation. The DDL is a Ruby DSL used to describe available `actions`, their input variables, and the output they produce. It's not unlike the Resource part of an LWRP.

The agent implementation is plain old Ruby, although it is possible to implement specific actions in external scripts. MCollective will invoke the script with the names of an request file, containing the request data as JSON, and a reply file, which the script needs to populate with a JSON hash of the result.

An rspec harness is available, so you can write tests for your agents.

AGENTS - DDL

```
action "status", :description => "Is the chef-client daemon running?" do
    output :status,
           :description => "Status of the chef-client daemon",
           :display_as => "Status"

    summarize do
        aggregate summary(:status)
    end
    display :failed
end
```

```
module MCollective
  module Agent
    class Chef<RPC::Agent

      action "status" do
        out = ""
        err = ""
        exitcode = run("service chef-client status",
                       :stdout => out, :stderr => err)
        case exitcode
        when 0
          reply[:status] = "OK"
        when 1
          reply[:status] = "Missing"
        when 3
          reply[:status] = "Stopped"
        end
      end

    end # end of class Chef
  end
end
```

AGENTS - OUTPUT

```
$ mco rpc chef status
Discovering hosts using the mc method for 2 second(s) .... 8

 * [ ============================================================> ] 8 / 8



Summary of Status:

        OK = 6
   Stopped = 1
   Missing = 1


Finished processing 8 / 8 hosts in 747.93 ms
```

AGENTS

- https://github.com/scalefactory/sf-deploy
- https://github.com/youdevise/orc
- https://github.com/ripienaar/mc-plugins/tree/master/agent/libvirt
- https://github.com/puppetlabs/mcollective-puppet-agent

APPLICATIONS

- New 'mco' subcommands
- Unified help, error reporting, and option parsing
- Full control of the client (with helpers)

APPLICATIONS

```
class MCollective::Application::Echo<MCollective::Application
   description "Reports on usage for a specific fact"

   option :message,
          :description    => "Message to send",
          :arguments      => ["-m", "--message MESSAGE"],
          :type           => String,
          :required       => true

   def main
      mc = rpcclient("helloworld")

      printrpc mc.echo(:msg => configuration[:message], :options => options)

      printrpcstats
   end
end
```

http://docs.puppetlabs.com/mcollective/reference/plugins/application.html

APPLICATIONS

```
$ mco echo
The message option is mandatory

Please run with --help for detailed help

$ mco echo -m test

 * [ ============================================> ] 1 / 1


example.com
   Message: test
      Time: Mon Jan 31 21:27:03 +0000 2011


Finished processing 1 / 1 hosts in 68.53 ms
```

Automatic help for required argument. Helper methods provide same output as the "mco rpc" subcommand - but this is optional.

CLIENTS

- mcollective-client gem
- fine-grained control
- CLI parsing still on by default

This is what you'll want to use if you'd like to embed an mcollective client in another application - a web interface, your own command-line tools.

http://docs.puppetlabs.com/mcollective/simplerpc/clients.html

REGISTRATION

- Nodes announce themselves
- Agent consumes the announcements

There's also a registration system - nodes can announce their existence to the network, and an agent elsewhere can listen for those announcements and do something with them. One of the registration agents simply writes out the registration announcements into a directory, while another stores them in mongodb (where they can be used for discovery).

If you're using the Chef server, this might not sound very valuable. Using them to populate a database is a bit redundant, but they can be used to monitor that nodes haven't silently stopped running MCollective.

CHEF & MCOLLECTIVE

As I said earlier, MCollective is CM-agnostic. That said, the plugin system provides many opportunities for integration, not to mention the basic fact that we need to install and configure the thing. Let’s have a look at what's there today, and what I'm working on, and where we might go in the future.

CLASSES & FACTS

CLASSES

```
Configuration Management Classes:
   recipe.chef-client::config     recipe.chef-client::service
   recipe.chef_handler::default   recipe.extra_packages::default
   recipe.mcollective-x509-security::certificates
   recipe.mcollective-x509-security::default
   recipe.mcollective-x509-security::install_from_github
   recipe.mcollective::_install_client_pkg
   recipe.mcollective::_install_common_pkg
   recipe.mcollective::_install_server_pkg
   recipe.mcollective::client     recipe.mcollective::common
   recipe.mcollective::default    recipe.mcollective::puppetlabs-repo
   recipe.mcollective::server     recipe.sudo::default
   recipe.users::sysadmins        recipe.vt-gpg::default
   recipe.vt-gpg::library         recipe.x509::default
   role.base                      role.centos6
   role.chef-client-11
```

I find this pretty annoying, when I look at the full list from a node. I might reasonably want to discover nodes based on some of these recipes, but most of them are implementation details.

FACTS

- Ohai
  - mcollective runs ohai on request
- YAML
  - mcollective reads YAML from disk
  - Chef handler generates class list and facts yaml

Using ohai directly makes some requests much slower, but guarantees that fact data is (relatively fresh). Yaml makes requests more predictable, but data may be stale. Fact values are strings - no complex data.

FACTS

```
chef_environment => mcollective-dev
chef_packages.chef.version => 11.4.0
chef_packages.ohai.version => 6.16.0
cpu.total => 2
platform => centos
platform_version => 6.2
virtualization.role => guest
virtualization.system => kvm

cpu.1.mhz => 3411.482
memory.anon_pages => 121608kB
ohai_time => 1390840103.16891
```

Hands up: who has browsed through the full set of data returned by ohai? You might want it for use in a recipe, but it's less convincing when standing alone.

DISCOVERY

DISCOVERY - MC

```
$ mco facts chef_environment
Report for fact: chef_environment

        dev                                     found 1 times
        mcollective-dev                         found 2 times
        prod                                    found 3 times

Finished processing 6 / 6 hosts in 132.51 ms
```

In this example, we get the value of the “chef_environment” fact from every node that responded to our broadcast request. It appears we have 6.

DISCOVERY - CHEF

```
$ mco facts chef_environment --dm chef
Report for fact: chef_environment

        dev                                     found 1 times
        mcollective-dev                         found 2 times
        prod                                    found 3 times

Finished processing 6 / 7 hosts in 2002.09 ms


No response from:

   test-centos6.nat0.cryptocracy.com
```

Now, we get the list of nodes from the Chef server, before asking them for the fact. This time, we can see that a node we know about didn't respond.

DISCOVERY - CHEF

```
$ mco facts hostname --dm chef -C "role[base]" -I 'test-*'
Report for fact: hostname

        test-centos6                            found 1 times
        test-precise                            found 1 times

Finished processing 2 / 2 hosts in 144.28 ms

$ mco facts hostname --dm chef
   --do "chef_environment:prod AND platform_family:debian"

Report for fact: hostname

        chef-11                                 found 1 times
        illuminati                              found 1 times

Finished processing 2 / 2 hosts in 162.46 ms
```

We can also search by run list items, in familiar Chef syntax. Alternatively, use an arbitrary search query to find nodes.

SECURITY

- Basic configuration possible
- SSL/AES not turn-key

If you want to configure the SSL security method, you'll need to do a little work yourself. There's an attribute to set the PSK, which is alright for development.

SECURITY - SSL

```
search(:client, "name:*").each do |client|
  file "/etc/mcollective/ssl/clients/#{client.name}_public.pem" do
    owner "root"
    mode "0644"
    content client.public_key
    action :create
  end
end
```

There are two problems with this. First, you may not want all of our servers to operate as clients. Second, you'll want some (or all) of your admin users to be mcollective clients - they're completely ignored by this snippet. I'd like to see this improved, so that it is easy to deploy MCollective with the SSL security plugin if you're already using Chef Server.

SECURITY - X509

- Cookbook - mcollective-x509-security
- Uses x509 cookbook + chef-ssl
- Lacks tests, not on community site

https://github.com/zts/chef-cookbook-mcollective-x509-security.git

http://www.cryptocracy.com/blog/2013/04/20/very-simple-x509-pki-with-chef/

```
include_recipe "x509::default"

certs = node['mcollective-x509-security']['certs']

x509_certificate "mco-server-#{node['fqdn']}" do
  ca 'MCollective-CA'
  type 'server'
  key certs['server_key']
  certificate certs['server_cert']
end

x509_certificate "mco-client-#{node['fqdn']}" do
  ca 'MCollective-CA'
  type 'client'
  key certs['client_key']
  certificate certs['client_cert']
end

x509_ca_certificate "MCollective-CA" do
  cacertificate certs['ca_cert']
end
```

This recipe creates keys, self-signed certificates, and submits CSRs to the server. The chef-ssl tool is then used to produce signed certificates, which are installed on the following Chef run. MCO won't actually work until the signed certificates are installed.

AGENT - CHEF-CLIENT

- Start/Stop/Restart daemon
- Wake daemon (run now)
- chef-client service status

You saw the service status before - just says whether the daemon is configured, and what state it's in (no details about what Chef is actually doing).

AGENT - ISSUES

- Restarting services
- Concurrent runs
- Returning data

MCollective runs Chef, Chef restarts MCollective, request fails.
When chef already running, subsequent runs will wait - note agent timeout.
The 'min' formatter can be parsed somewhat easily, but writing out data directly would be preferable.

WHAT DOES PUPPET GET?

- Responses with #/changed resources, runtime, etc
- Data: Puppet status and result of most recent run
- Data: resources managed by puppet
- Manage individual resources

'Package' and 'Service' agents also exist with pluggable backends - implemented using Puppet, but we could do the same with Chef.
https://github.com/puppetlabs/mcollective-puppet-agent

COOKBOOK - GOOD

- Install from puppetlabs packages
- Most configuration from attributes
- Chef handler to generate fact and classes files
- Agent to manage chef-client

COOKBOOK - BAD

- Security configuration
- Configuration for authorisation and audit
- Plugin installation

Turnkey config of SSL plugin? LWRP for agent installation? Want to be able to easily add agents relevant to the roles on the host. Or install everywhere, and control activation?

ODDS & ENDS

Thinking back to the start of this talk, one of the problems I proposed was reconfiguring one tier of your infrastructure when another changes - eg, updating database firewall rules when additional webservers are added. How might we do that?

MCO_NOTIFY - LWRP

```
mco_notify “run chef” do
  agent "chef"
  action “wake_daemon”
  classes “role.haproxy“
  action :nothing
  subscribes :send, “service[httpd]”
end
```

Currently broken - a failure in conference-driven development. The `mco_notify` cookbook (github) was an early experiment in that direction. It offers an LWRP that can send a "fire and forget" request to a given MCollective agent.

MCO_NOTIFY - PROBLEMS

What happens if 20 nodes fire that request at the same time? Chef runs will pile up (subject to agent timeout?)

What if we wanted to return data? Perhaps stash stuff in node.run_state for retrieval by providers later in the run.

If we run another node too quickly, the Chef server/index might not have updated yet.

MCO_NOTIFY - HANDLER

Useful for audit logs? If you're using enterprise Chef, this would be reinventing the wheel.

MCO_SEARCH

Not even a proof of concept - use MCO discovery to find nodes. Would be useful for chef-solo - what use cases might exist for chef-server? Don't need to wait for server indexing, but shifts the problem to updating mcollective. With a data plugin loading state saved by Chef, this might be no big deal.

OTHER CONCERNS

- Packaging Philosophy
- MQ dependency

Chef: Omnibus, Puppet: build on the native system. How do we reconcile this? Unless you're building your own packages, you'll need to chef_gem mcollective-client. If you want an mcollective agent to load chef, you'll need to install the chef gem in the system ruby.
Or, only use CLI-based integration.

PARTING THOUGHTS

MCollective is great. It's not too hard to get started, and you only need to do that once. After that, it's incredibly simple to add things to suit your needs. Philosophically, it's a good fit for Chef.

You should definitely give it a try, work on building/improving Chef-related plugins, and share them with the world!

REFERENCES

- http://docs.puppetlabs.com/mcollective/
- https://github.com/zts/cookbook-mcollective
- https://github.com/zts/mcollective-chef-discovery
- https://github.com/zts/chef-cookbook-mcollective-x509-security
- https://github.com/puppetlabs/mcollective-puppet-agent