Puppet in the cloud

jethrocarr.com

15 mins!!

Puppet handling experience

- Currently running a small environment for sailthru.com / carnival.io (Personalised marketing SaaS) of around 200 GNU/Linux hosts using AWS and Puppet.
- In my previous role, used to do the same sort of thing for a much bigger environment at Fairfax (stuff.co.nz, smh.com.au and others).
- Big fan of DevOps tools and techniques!
- (I also blog stuff at jethrocarr.com, sometimes about all these things)

Why does config management matter?

- Quick, reliable, repeatable provisioning of servers.
- Makes automatic provisioning possible.
- Consistency (and auditability).
- Because writing a beautifully tuned config file for an service isn’t so exciting once you’re up to the thousandth copy. ;-)

But I have Kubernetes/Mesos/ECS/Docker Swam now - why do I care?

¯_(ツ)_/¯

OK you have my interest - tell me more human!

Puppet Crash Course

- Puppet expresses infrastructure as code in it’s own DSL

```
# ensure mysql service is running
service { 'mysql':
  ensure => running,
}

# install php5 package
package { 'php5':
  require => Exec['apt-update'],        # require 'apt-update' before installing
  ensure => installed,
}

# ensure info.php file exists
file { '/var/www/html/info.php':
  ensure => file,
  content => '<?php  phpinfo(); ?>',    # phpinfo code
  require => Package['apache2'],        # require 'apache2' package before creating
}
```

Puppet Crash Course ++

- All servers run a Puppet Agent.
- These servers connect to a Puppet Master / Puppet Server periodically to download the latest manifests (configuration).
- If the server is not inline with what the manifests states, Puppet changes things to bring it inline (default: every 30mins).

This has failed to deliver on “interesting”

OK now for some actually interesting stuff...

- Most of us tend to run more than one kind of server.
  (ie: webserver, dbserver, fileserver)

- We naturally don’t want an exploited server of one type being able to access all the config for server of another.

- Puppet has a CA & cert model for trust relationships - the Puppet master signs the certs for each client. This signing process sets up a trust relationship.

- This guarantees that server “db1” is actually “db1” and not infact “www1”.

And all was well...

- In the “good ol days” we racked and stacked boxes.
- Running a process to register servers to the Puppet Master manually wasn’t that hard compared to cutting your hands apart on cheap cage nuts

```
pet: ~ # puppet agent --server puppetmaster --waitforcert 60
puppetmaster: ~ # Puppet cert --sign myclient
```

- The system worked!

But it’s not exactly cloud scale

Let’s make it cloud scale!

- Let’s enable auto signing on the Puppet master to sign those annoying certs so servers can boot and get their config immediately without admin involvement.
-
- And instead of hostnames, let’s use roles so we can just use generated hostnames without requiring upfront config.
-
- Something like “webserver_catpix” so we know it’s exact purpose?

- We can put it on the EC2 tag so it can self-discover at boot!

So our workflow now looks like this:

1. Server boots generic Linux OS image.

2. “Hey my EC2 Tag or Userdata says I’m a webserver”

3. Connects to Puppet to sign cert.

4. Puppet master signs it’s cert and gives it the configuration for the requested role.

5. Server goes on to do great things.

problem?

So our workflow now looks like this:

1. Server comes online.
   Server gets owned by an attacker
2. “Hey my EC2 Tag or Userdata says I’m a webserver”

3. Connects to Puppet to sign cert.
   Attacker executes Puppet cert sign request with the role dbserver.
4. Puppet master signs it’s cert and gives it the configuration for the requested role.
   Attacker now has config for db server including SQL root password.
5. Server goes on to do great things.
   Attacker pwns all the things

Puppet happily runs as non-privileged user

“www-data” just pwned your DB server

Autosigning is very dangerous if you get it wrong

1. You have to authenticate that the server is legitimately asking for the role it’s supposed to have!

2. Puppet offer various autosigning options - I’ll sum it up as “don’t use anything but policy based autosigners”.

3. Policy based autosigner involves puppet running an executable you provide to validate all autosign requests and to instruct it yay/nay.

Like actually 100% not kidding RTFM here if deploying Puppet - some very important notes:
https://puppet.com/docs/puppet/latest/ssl_autosign.html

Safe auto signing fundamentals.

1. Biggest headache is that EC2 user data is readable by any user on the server. And if you’re using Tags and IAM roles, those are readable by any user on the server as well.

2. Solution - design a process that can’t be repeated. For example using a single-use token to sign initially, or some out of band validation such as using the tags, or Lambda or ASG notifications.

3. Recommend thinking about your architecture and the ways an attacker might try to get past your autosigner.

Reference implementations

Puppet doesn’t link to any handy policy autosigners that would actually help you do this securely in their docs - this is something crying out for an official reference implementation.

https://github.com/carnivalmobile/carnival-autosign-aws-puppet
An autosigner I wrote built around the model of a trusted AWS tagging setup (ie servers cannot self-tag).

https://github.com/danieldreier/autosign
A different approach (maybe better?) using JWT tokens.

The next hurdle - Trusted facts

1. Puppet collects facts about a server “I’m a m4.xlarge!”, “I have 2GB RAM!”, “I run Plan9 as my OS!”.

2. These are NOT trusted - you don’t want to use them in your decision making about what config you provide a system, since they could be changed post-signing!

3. But you can define Trusted Facts at time of signing and have them baked into your certificate making them unchangeable.

More important reading:
https://puppet.com/docs/puppet/latest/lang_facts_and_builtin_vars.html#trusted-facts
https://puppet.com/docs/puppet/latest/ssl_attributes_extensions.html

Untrusted Facts example - dangerous

```
$role_class = "role::${facts['role']}"

if defined("$role_class") {
  include $role_class
}
```

Trusted Facts example

```
$role_class = "role::${trusted['extensions']['pp_role']}"

if defined("$role_class") {
  include $role_class
}
```

OK yes, that is kinda interesting. What else can you tell me?

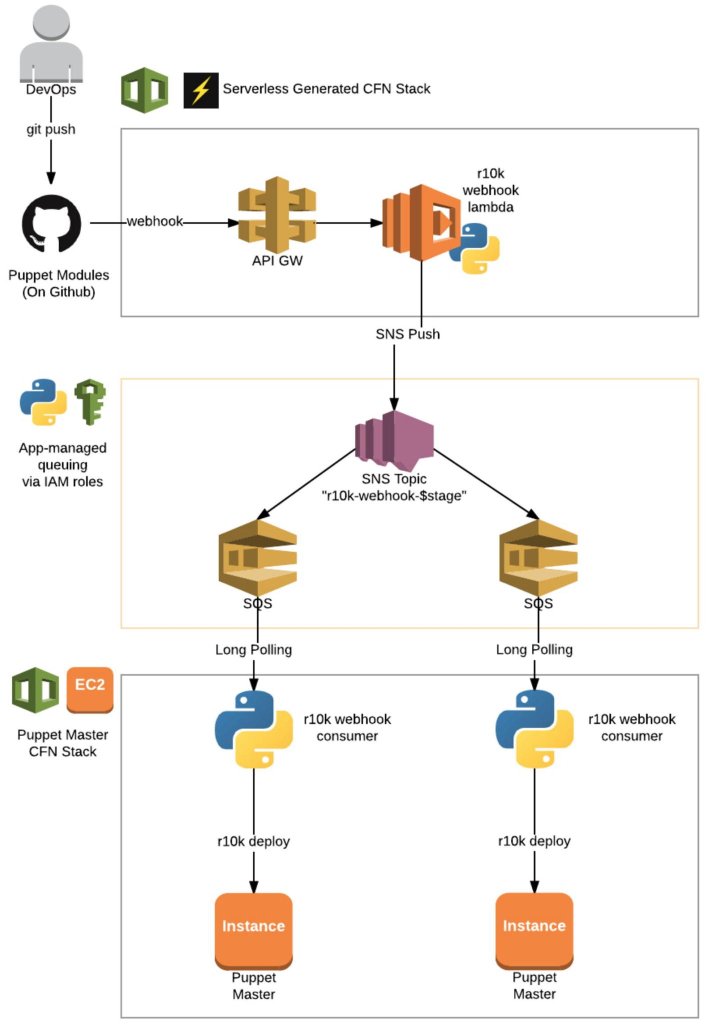
Puppet CI/CD

- In it’s simplest form, a Puppet Master just serves up configuration to clients from a directory on the Puppet Master itself.
- One popular solution for getting code from Engineer to Puppet Master, is a tool called “r10k”.
- This tool makes it easy to split apart your Puppet manifests into modules, each in their own independent git repo.
- And it allows Git branches to be mapped to environments.

R10k + Carnival r10k Webhook

R10K
https://github.com/puppetlabs/r10k

Webhook + Service for distributing updates to multiple Puppet masters:
https://github.com/carnivalmobile/carnival-r10kwebhook



High Availability

Actually not that hard

- You just need n+1 Puppet Masters :-)

- Puppet Manifests need to be synced to all Puppet Masters
  Solution: Use shared storage (eg EFS) or a solution like carnival-r10kwebhook which can keep all the Puppet Masters in sync from Github.

- Certificates that have been signed need to be distributed to all the Puppet masters.
  Solution: Make your autosigner sync them, or just use shared storage. I use EFS (make sure to provision IOPS!)

Bake for autoscale reliability

- Puppet config is code - and code can have bugs.
- Recommend not relying on Puppet for *directly* provisioning autoscale systems under load.
- Instead, bake Puppet and into an AMI using Packer. Packer can call Puppet to apply config to the AMI, so the resulting image can immediately boot with everything it needs and then

Masterless Puppet is a thing

Removing the Puppet masters makes things easier... Right?

https://www.jethrocarr.com/tag/pupistry/
https://github.com/jethrocarr/pupistry

A lot of caveats - for most uses, I’d recommend just sticking to the tried & tested Puppet Master -> Puppet Agent approach.

Out of time! But some takeaways:

- Very, VERY, carefully review the autosigning process you choose for your organisation. Think like an attacker- what could they do with it?

- Make sure it’s as easy and reliable to release infrastructure as it is code - fast and reliable continuous deployment pipelines are a must.

- HA isn’t as hard as you might think - so do the work and make sure all components (eg especially certs!) are redundant. You don’t want to lose your config management in an incident!

- Combine Puppet with AMI baking for extra deliciousness

Thank you

www.jethrocarr.com

Thanks to my employer for their support - www.carnival.io