PUPPET AND MGMT

BETTER TOGETHER

Felix Frank

CfgMgmtCamp.eu Gent 2017

Felix - trying to automate all the things since 2004

- ask me about Puppet
- ask me about Ansible
- ask me about mgmt (soon)
- ...or ask my employer if I can help with your project

Speaking of...

Competent colleagues

Cool projects

Prestigious customers

Steep learning curve

Informal culture

Plenty of creative opportunities

Flexible working hours

Nice little extras

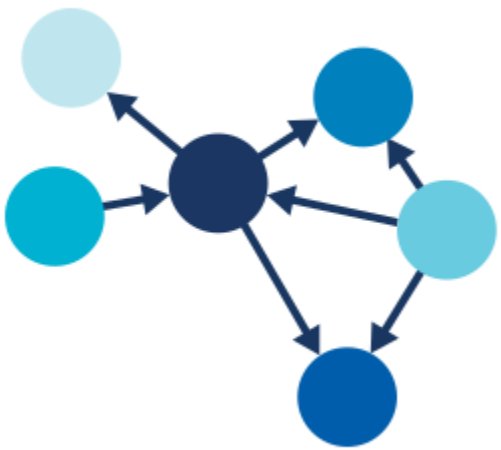
...we are hiring in Berlin

unbelievable-machine.com/en/careers

WARNING!

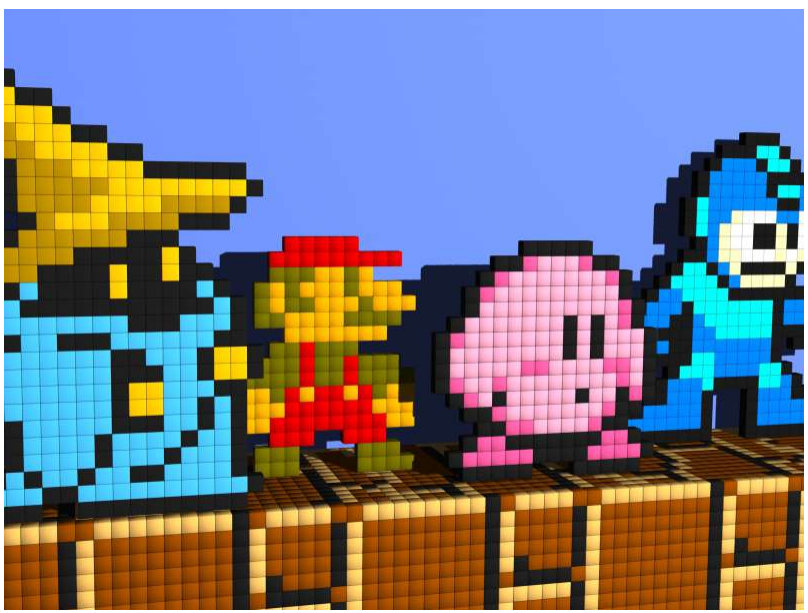
Major Puppet spoilers ahead

Welcome!



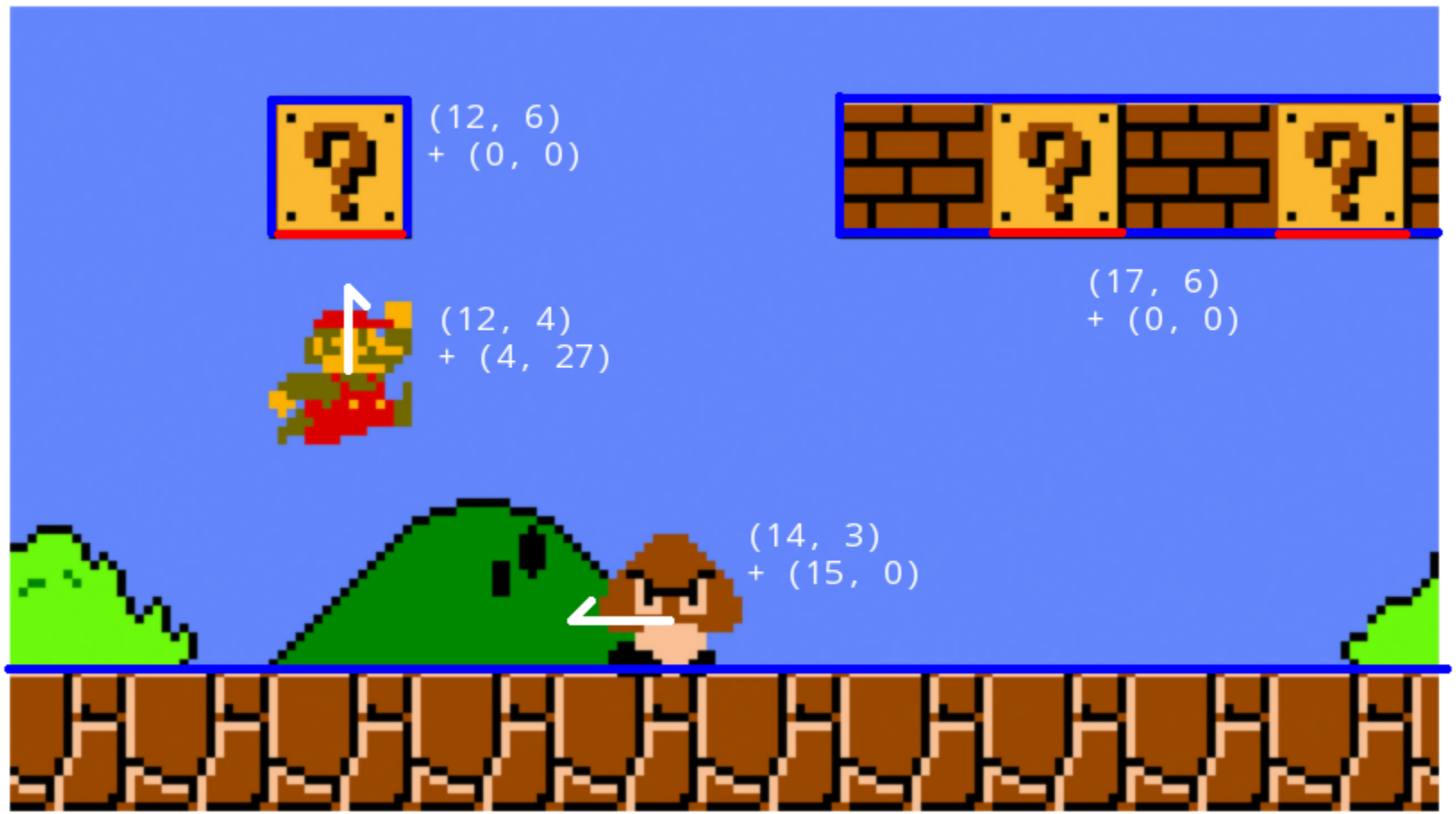
Many cool things to talk about

So let's talk about game development



http://7te.org/retro-game-collage-wallpaper.html

Games: stateful systems with *really* great UI

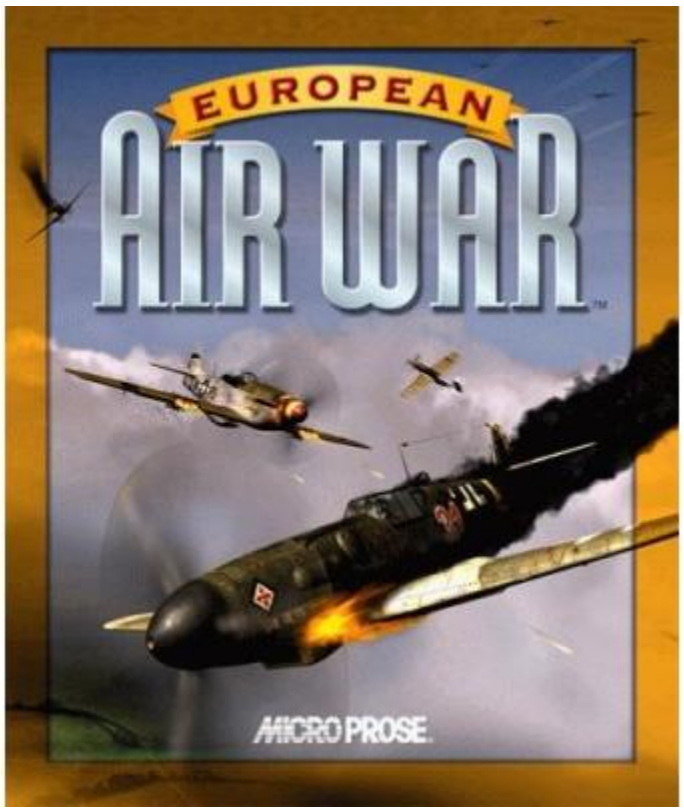


Game design lessons for utility software

- intuitive UI is everything
- do lots of play testing
- rapid prototyping

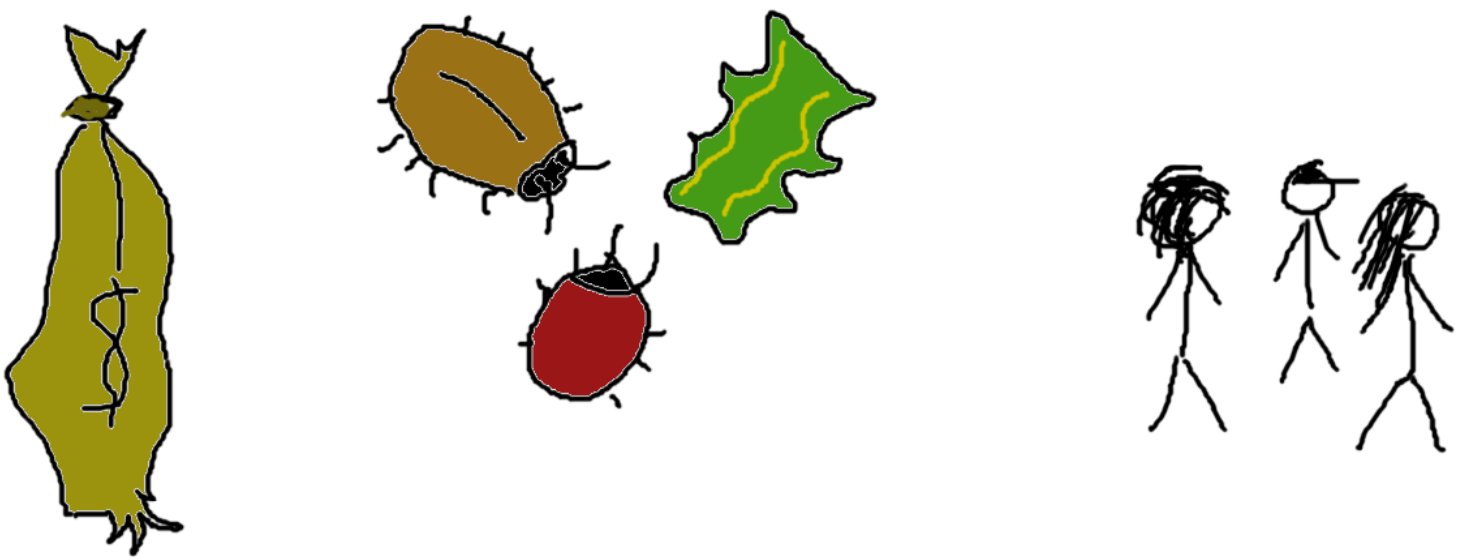
Larger game productions not unlike enterprise tools

e.g. European Air War



https://en.wikipedia.org/wiki/European_Air_War

It was doomed



New staff faced dozens of bugs

Hence first task:

add the

Cool Cam

feature

Ultimately an early gimmick feature saved the project

An early gimmick feature:

PUPPET SUPPORT IN MGMT

PUPPET+MGMT EXAMPLES

```
$ mgmt run --allow-tmp-prefix \
   --puppet 'package { "cowsay": ensure => installed }'
```

```
$ mgmt run --allow-tmp-prefix \
   --puppet 'file { "/etc/ntp.conf":
              content => template("/etc/ntp.conf.erb")
             }
             ~>
             service { "ntp": ensure => running }'
```

```
$ mgmt run --allow-tmp-prefix \
   --puppet 'class { "puppetdb": database => "embedded" }'
```

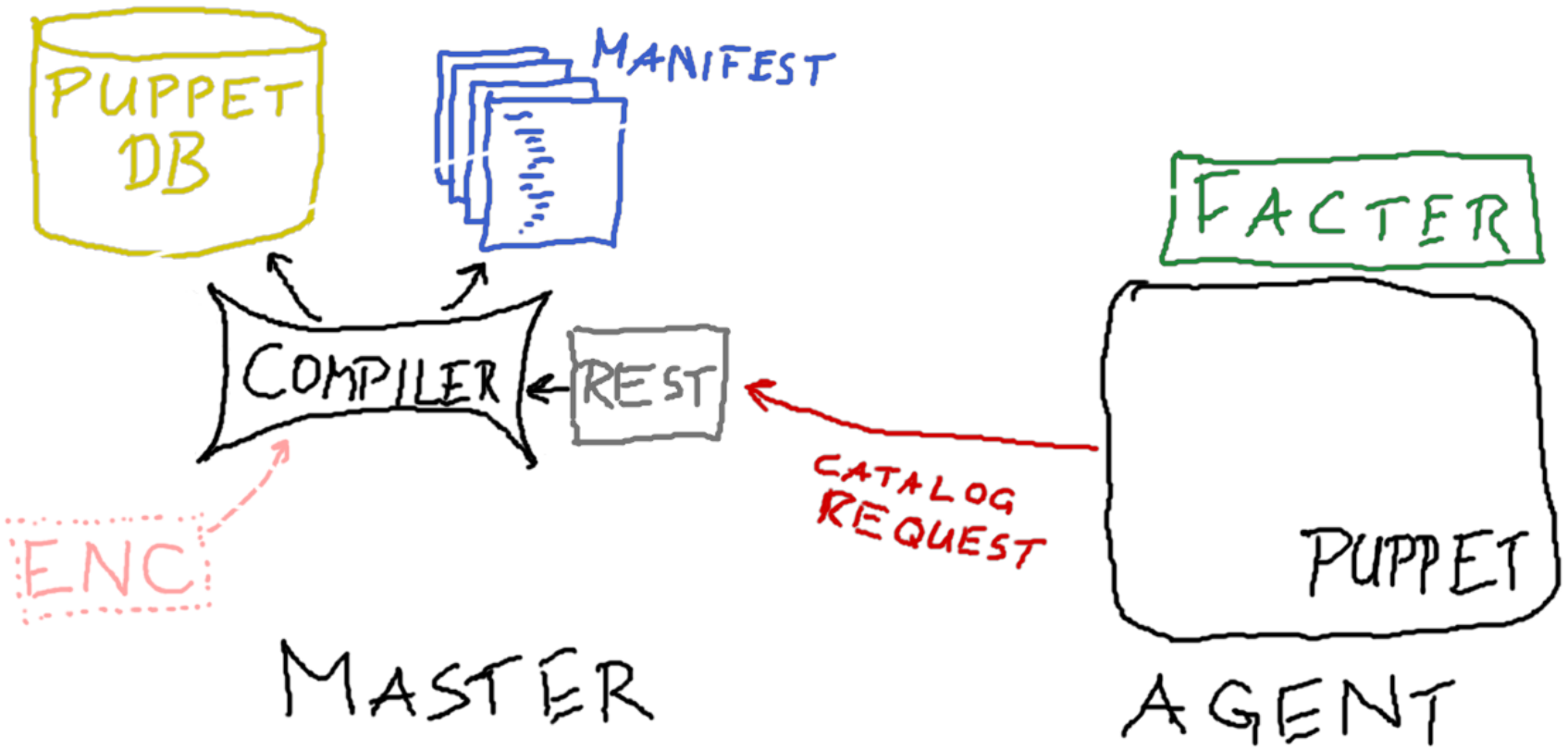
```
$ mgmt run --allow-tmp-prefix \
   --puppet /var/local/manifests/hardening.pp
```

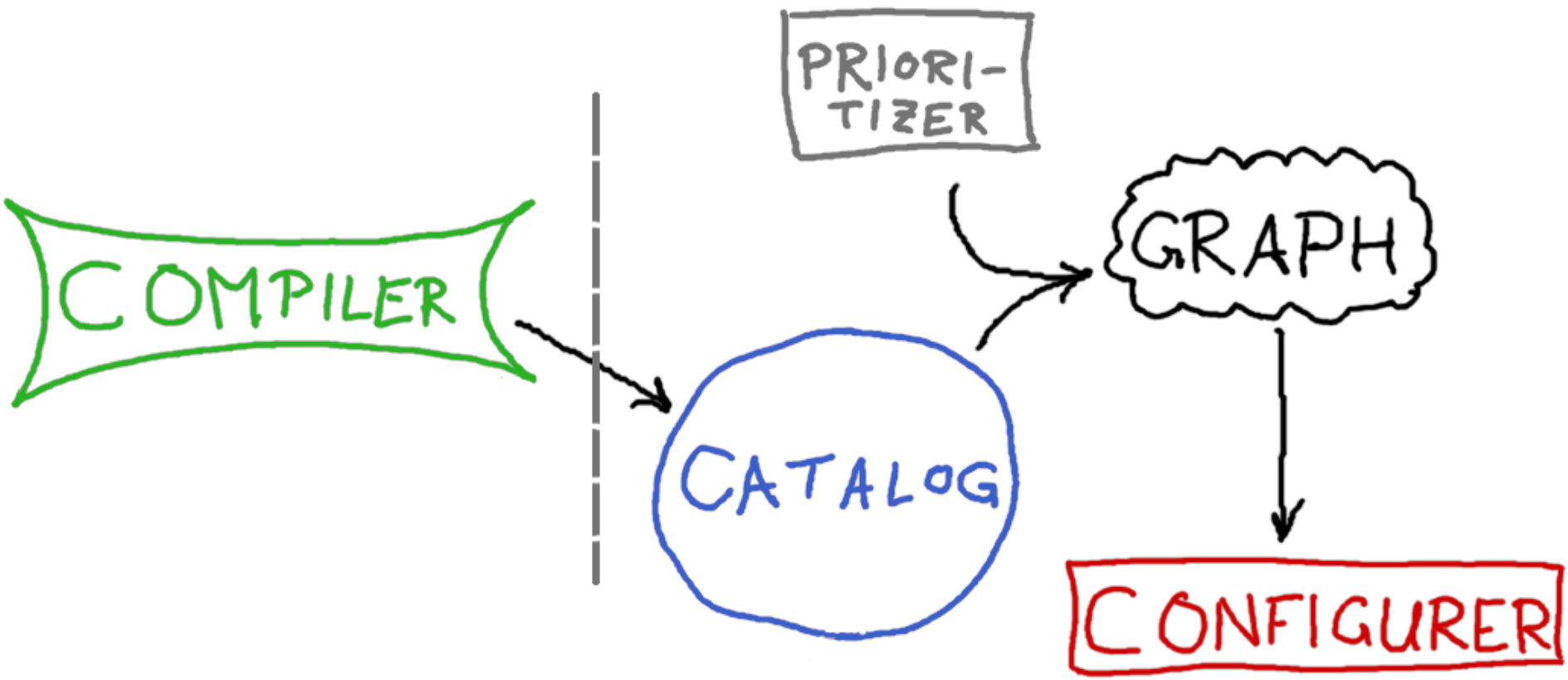
```
$ mgmt run --allow-tmp-prefix --puppet agent
```

So how does this work?

let's take a quick deep dive

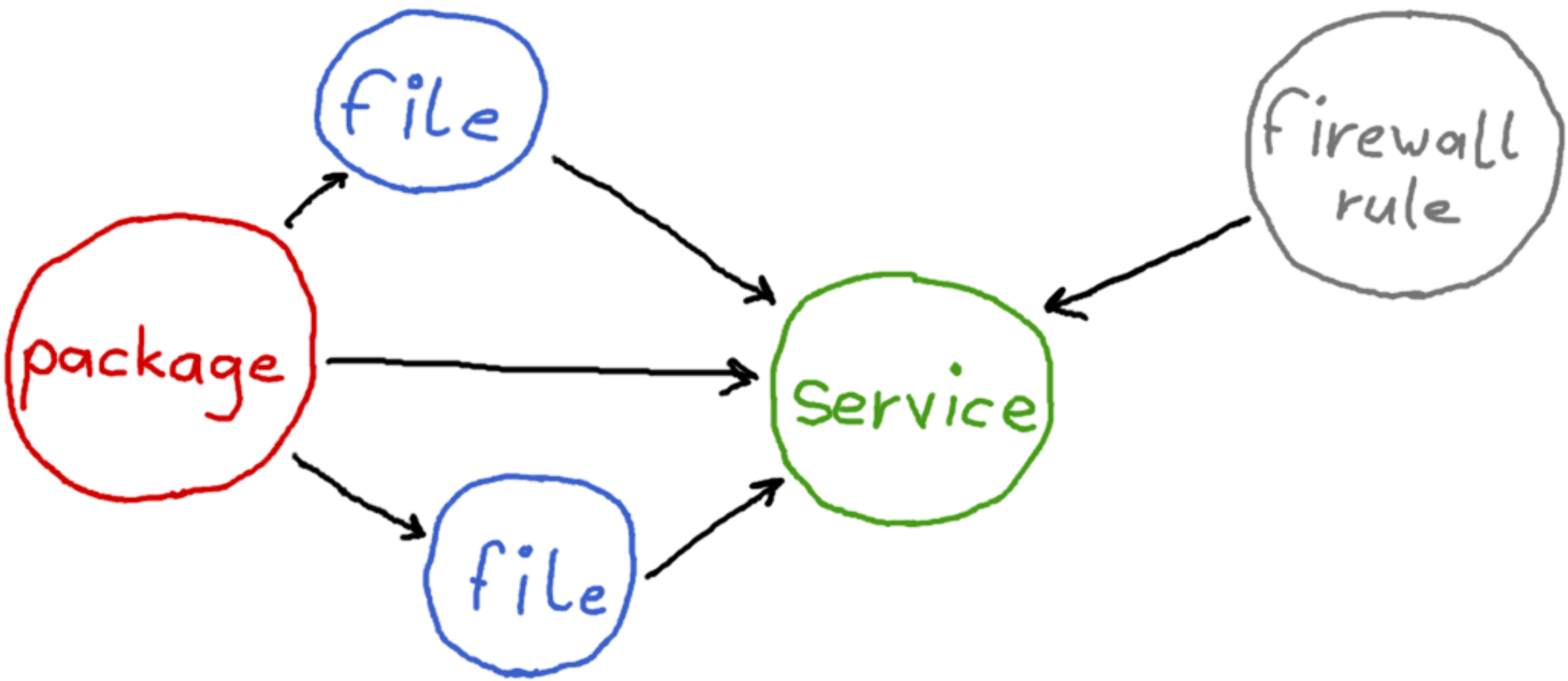
Two principal parts of Puppet




PRIORI-
TIZER
COMPILER
GRAPH
CATALOG
CONFIGURER

Side note:
the Puppet compiler
is not actually a compiler

catalog representation



On the wire, the catalog looks very similar to the manifest

```
{
  "tags": ["settings","fflaptop
  "name": "fflaptop.local",
  "version": 1486222644,
  "code_id": null,
  "catalog_uuid": "8bd0ee21-8ac
  "catalog_format": 1,
  "environment": "production",
  "resources": [
    {
      "type": "Stage",
      "title": "main",
      "tags": ["stage","main"],
      "exported": false
    },
    {
      "type": "Class",
      "title": "Settings",
      "tags": ["class","setting
      "exported": false
    },
```

```
# class { "demo":
#   arg => [
```

```
...
{
  "type": "File",
  "title": "/tmp/this-is-a-file",
  "tags": ["file","class"],
  "file": "/home/ffrank/.puppetlabs/
  "line": 4,
  "exported": false,
  "parameters": {
    "ensure": "present",
    "owner": "ffrank",
    "group": "www-data"
  }
},
{
  "type": "Exec",
  "title": "/usr/games/cowsay mooo",
  "tags": ["exec","node","fflaptop.l
  "file": "/home/ffrank/.puppetlabs/
  "line": 14,
  "exported": false
}
```

actual metaparameters:

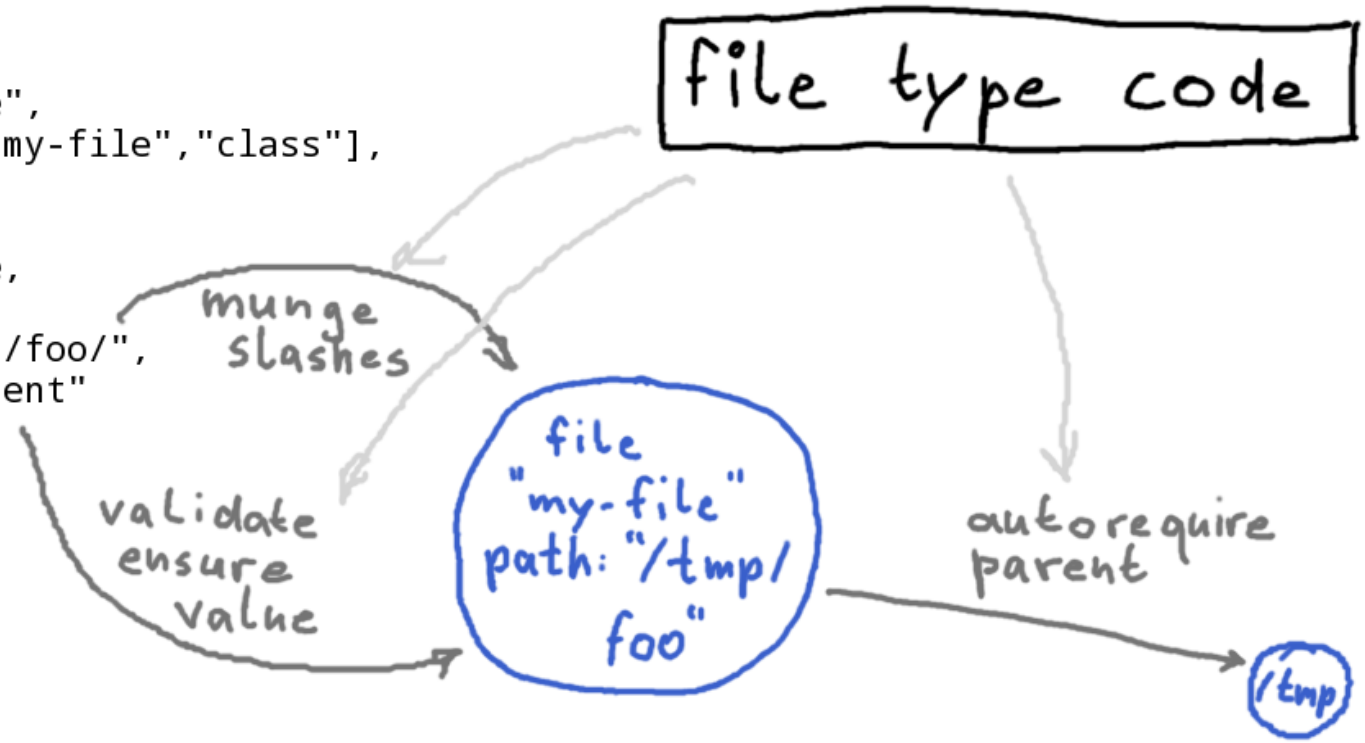
```
# object { "/tmp/something":
```

```
#    "complex", "value",

#   { sensible => false } ]
# }

{
  "type": "Class",
  "title": "Demo",
  "tags": ["class","demo"],
  "file": "",
  "line": 1,
  "exported": false,
  "parameters": {
    "arg": [
      "complex",
      "value",
      { "sensible": false }
    ]
  }
}
```

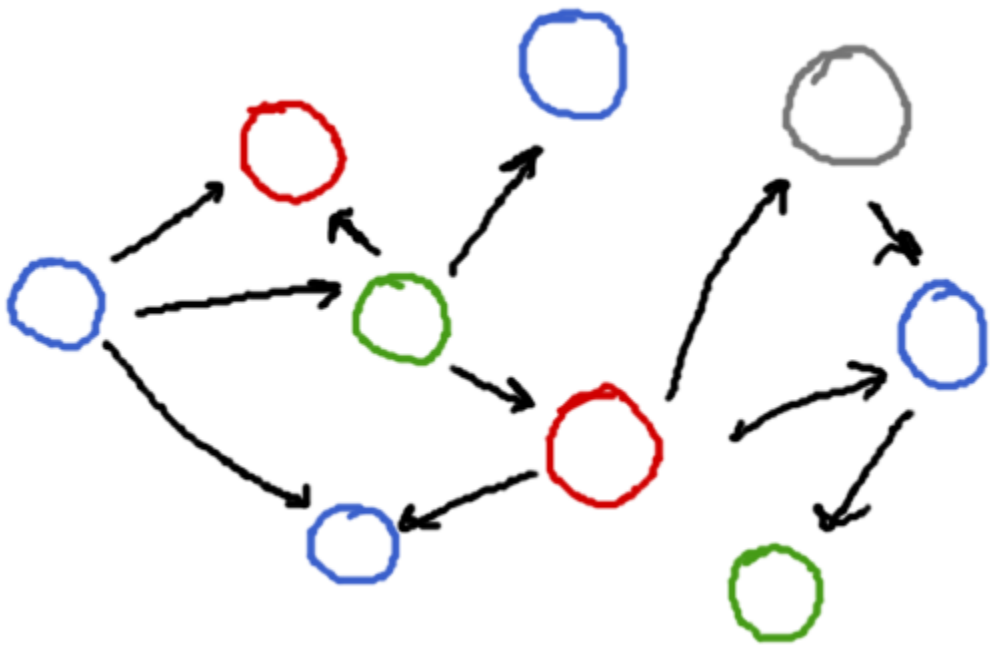
```
# object { '/tmp/something':

#   content => "things-to-store"
# }

{
  "type": "Object",
  "title": "/tmp/storage",
  "tags": ["object","class"],
  "file": "",
  "line": 1,
  "exported": false,
  "parameters": {
    "before": [
      "Service[ostor]"
    ],
    "content": "things-to-store"
  }
},
```

From catalog to Resource Abstraction Layer

```
{
  "type": "File",
  "title": "my-file",
  "tags": ["file","my-file","class"],
  "file": "",
  "line": 1,
  "exported": false,
  "parameters": {
    "path": "/tmp///foo/",
    "ensure": "present"
  }
}
```

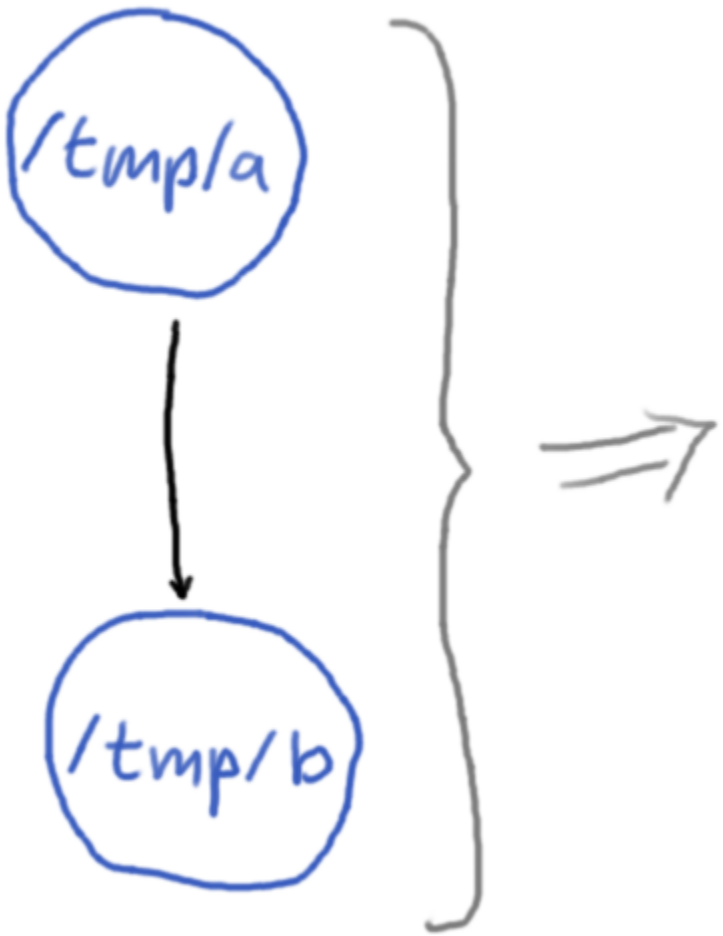


Finally the RAL catalog emerges



Puppet will hand this to the configurer

We remodel it into an mgmt graph instead




```
resources:
  file:
  - name: "/tmp/a"
    path: "/tmp/a"
    state: exists
    content: me first
  - name: "/tmp/b"
    path: "/tmp/b"
    state: exists
    content: then me
edges:
- name: File[/tmp/a] -> File[/tmp/b]
  from:
    kind: file
    name: "/tmp/a"
  to:
    kind: file
    name: "/tmp/b"
```

The **mgmt** interface you saw:

```
$ mgmt run --allow-tmp-prefix \
   --puppet 'package { "cowsay": ensure => installed }'
```

Internally, this invokes a **puppet** subcommand:

```
$ puppet mgmtgraph print \
   --code 'package { "cowsay": ensure => installed }'
```

This works courtesy of the ffrank-mgmtgraph module

Transformation

```
{
  "type": "File",
  "title": "/etc/ntpd.conf",
  "tags": ["file","class"],
  "file": "",
  "line": 1,
  "exported": false,
  "parameters": {
    "ensure": "present"
  }
}
```

```
file:
- name: /etc/ntpd.conf
  path: /etc/ntpd.conf
  state: exists
  content:
```

```
{
  "type": "File",
  "title": "/etc/ntpd.conf",
  "tags": ["file","class"],
  "file": "",
  "line": 1,
  "exported": false,
  "parameters": {
    "ensure": "present"
  }
}
```

```
file:
- name: /etc/ntpd.conf
  path: /etc/ntpd.conf
  state: exists
  content:
```

Getting from A to B

- translating type **File** to **file**
- using the **title** as **path**
- renaming the **ensure** parameter to **state**
- translating its vaule from **present** to **exists**

...let's convert that to code

```
module PuppetX::CatalogTranslation
  Type.new :file do
    spawn :name do
      @resource.title
    end

    spawn :path do
      @resource[:name]
    end

    rename :ensure, :state do |value|
      case value
      when :present, :file, :directory
        :exists
      when :absent
        :absent
      else
        raise "cannot translate file ensu
      end
    end
```

Getting from A to B

- ~~translating type **File to file**~~
- using the **title** as **path**
- renaming the **ensure** parameter to **state**
- translating its vaule from **present** to **exists**

Puppet name vs.

title

```
file { "the-ntp-configuration":
  path => "/etc/ntpd.conf",
  owner => "root",
}

file { "the-other-ntp-configuration-you-see":
  path => "/etc/ntpd.conf",
  mode => "0644",
}
```

This is the same file, twice!

Puppet knows this (it's not dumb).

This is how mgmt will see the former

```
file:
- name: the-ntp-configuration
  path: /etc/ntpd.conf
  content:
```

The name is chosen independently of the path.

In Puppet, **name** and **title** are distinct but related:

- the title is chosen freely
- the name gets its value from the **namevar** of the resource (for files: *path*)
- the namevar can use the title if not otherwise specified (as in *file { "/etc": }*)

Hence the rule:

```
spawn :path do
  @resource[:name]
end
```

It always picks up the actual path thanks to the namevar semantics.

This is much safer than looking at the **path** parameter.

I lied about that rule, by the way.

Here's what it actually looks like:

```
spawn :path do
  if @resource[:ensure] == :directory
    @resource[:name] + "/"
  else
    @resource[:name]
  end
end
```

mgmt has no **state=directory**.

It uses a trailing slash on the path.

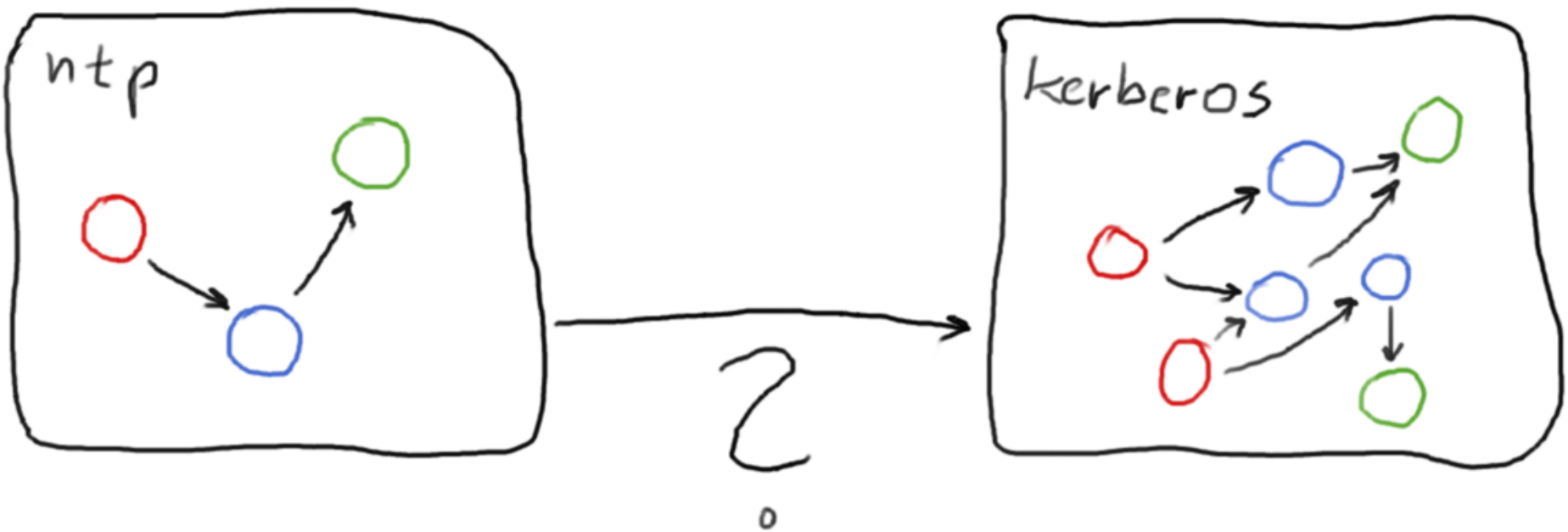
Here's the rule(s) for another resource type

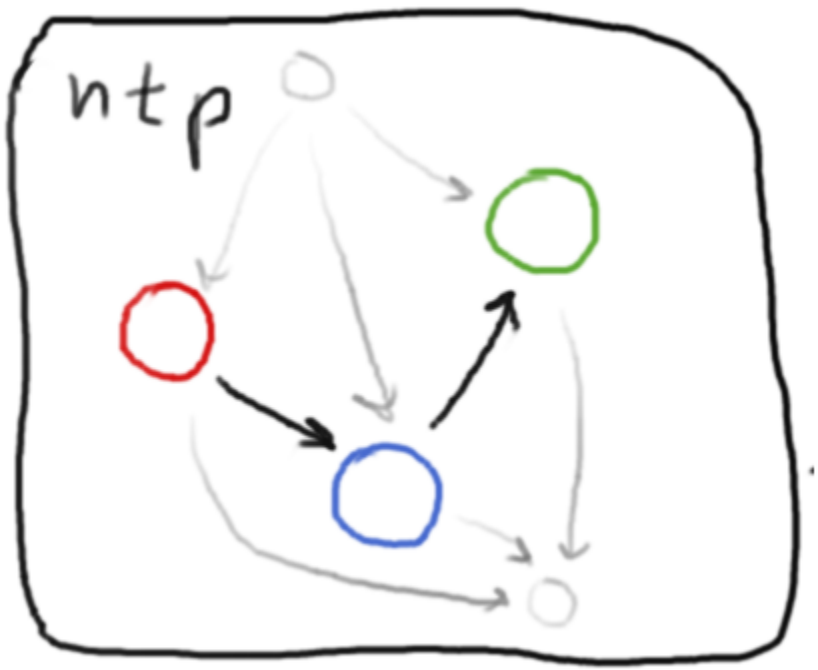
```
PuppetX::CatalogTranslation::Type.new :whit do
  emit :noop

  spawn :name do
    @resource[:name]
  end
end
```

What, you never heard of the **whit** resource?

Let's talk about containment

```
Class['ntp'] -> Class['kerberos']
```





edges to containers become edges to their **whit** boundaries

Too Weird; Didn't Listen?

the translation will just deal with classes, defines, and relationships between them

Advantages of this approach

- static validation by Puppet's agent engine
- Puppet magic like autorequire Just Works
- the usual convenience from Puppet's munging rules applies

on the flipside:

- a full agent installation on every node is needed
- getting an abstract catalog from a master using REST would work as well
- but then you lose all the advantages mentioned above

Perhaps **you** will find a better compromise?

All that being said, there are some general restrictions when running from Puppet code.

Puppet will always create a catalog using point-in-time input

- facts seen on the node
- values from PuppetDB
- manifest code

The catalog builder derives

- a consistent graph that represents
- the desired state of the (complete) system
- as of the time of requesting the catalog
- and which is suited
  to converge in a single transaction

mgmt can actually update the graph of a running agent

However, adding support for triggered Puppet graph rebuilds does not seem sensible

So what **can** you expect?

A sensible goal is to make it possible to run (the) most (popular) Puppet modules through mgmt.

For mgmt to become effective, however, a custom DSL is much more important.

That's because Puppet's DSL has some built-in assumptions.

Example: facts are static values

```
class { "apache": mpm_module => "prefork" }

$process_cap = Integer( $memorysize_mb / 125.0 - 10.0 )
class { "apache::mod::prefork":
  maxclients => $process_cap
}
```

In a VM, the memory size could change on the fly.

mgmt could receive an event about that and initiate action.

But Puppet cannot emit a structure that reflects this.

In summary

Support for Puppet manifests is neat, it eases testing right now and will help wider adoption.

On the other hand, a custom DSL for mgmt will allow actual new config management practices that a translated catalog cannot implement.

QUESTIONS

"The Cool Cam" originally published at http://thedailywtf.com/articles/The-Cool-Cam

Bonus content

But what about Puppet resources that mgmt does not have?

Consider the **puppet resource** command

```
$ puppet resource nagios_host mail01 ensure=absent
```

It allows simple resource management from the shell.

There's just one problem:

```
nagios_host { 'mail01':
  host_groups => [ "mailservers", "legacy" ]
}
```

Non-trivial values like hashes and arrays are not supported.

Solution

Yet another Puppet module that introduces the **puppet yamlresource** face:

```
$ puppet yamlresource nagios_host \
    mail01 '{ "ensure": "absent" }'
$ puppet yamlresource nagios_host \
    mail01 '{ "host_groups": [ "mailservers", "legacy" ] }'
```

Now the translator can emit an **exec** resource that makes Puppet do the legwork

```
PuppetX::CatalogTranslation::Type.new :default_translation do
  emit :exec

  catch_all

  spawn :name do
    @resource.type.to_s.capitalize + ":" + @resource[:name]
  end

  def command(resource)
    r_type = @resource.type.to_s
    r_title = @resource[:name]
    r_params = @resource.to_hash.reject { |attr,value|
      attr == :name
    }
    "puppet yamlresource #{r_type} '#{r_title}' " +
      "'#{Psych.to_json(r_params).chomp}'"
  end

  spawn :cmd do
    command(@resource)
  end
```

```
# puppet mgmtgraph print --code \
#   'nagios_host { "mail01": ensure => absent }'
  exec:
  - name: Nagios_host:mail01
    cmd: |-
      puppet yamlresource nagios_host 'mail01' '{
        "host_name": "mail01", "provider": "naginator",
        "ensure": "absent", "target": "/etc/nagios/nagios_host.cfg",
        "loglevel": "notice"}'
    timeout: 30
    shell: /bin/bash
    watchshell: /bin/bash
    ifshell: /bin/bash
    watchcmd: 'while : ;
      do echo "puppet run interval passed" ; /bin/sleep 1800 ;
      done'
    ifcmd: |-
      puppet yamlresource nagios_host 'mail01' '{
        "host_name": "mail01", "provider": "naginator",
        "ensure": "absent", "target": "/etc/nagios/nagios_host.cfg",
        "loglevel": "notice"}' --noop --color=false \
        | grep -q ^Notice:
    state: present
    pollint: 0
```

Starting lots of puppet processes though...well...

http://imgur.com/gallery/gsM3Lt5

The unbelievable Machine Company