Configuration Management

Pieter Lexis
lieter_
pieterlexis

Master SNE – LIA

February 15, 2016

whois

Pieter Lexis

- OS3 graduate
- System admin by training
- Coder by accident
- DNS(SEC) nerd

Introduction

Configuration Management

Configuration Management (CM) is an Information Technology Infrastructure Library (ITIL) version 2 and an IT Service Management (ITSM) process that tracks all of the individual Configuration Items (CI) in an IT system which may be as simple as a single server, or as complex as the entire IT department. In large organizations a configuration manager may be appointed to oversee and manage the CM process. In ITIL version 3, this process has been renamed as Service Asset and Configuration Management.

Configuration Management

Configuration Management

Configuration
on Technology
Infrastructure
rvice Manage-
ment (ITSM) p
ual Configura-
tion Items (CI)
ple as a single
server, or as c
In large orga-
nizations a co
ted to oversee
and manage t
s process has
been renamed
Management.

Configuration Management

The techniques and policies to track hardware, infrastructure and software, and the configuration thereof. What changed, when did it change and who changed it.

DevOps I

DevOps

DevOps (a clipped compound of ”development” and ”operations”) is a culture, movement or practice that emphasizes the collaboration and communication of both software developers and other information-technology (IT) professionals while automating the process of software delivery and infrastructure changes. It aims at establishing a culture and environment where building, testing, and releasing software, can happen rapidly, frequently, and more reliably.

DevOps II

“Because we are lazy, we want to do something only once”

“The developers and administrators are repsonsible together”

“Software and full systems should be testable”

Phrases, phrases everywhere

Hip terms to describe this:

- Automation
- Treat systems as cattle, not pets
- Deploy early, deploy often
- DevOps
- Infrastructure as Code

Elements

Orchestration

- What runs where?
- What depends on what?
- How can we move things around?

Configuration

- System settings
  - `sysctl`
  - Memory limits
- Software configuration
  - Settings
  - Data files
- Miscellaneous
  - Backups (muy importante!)
  - Monitoring
  - Logging

Monitoring I

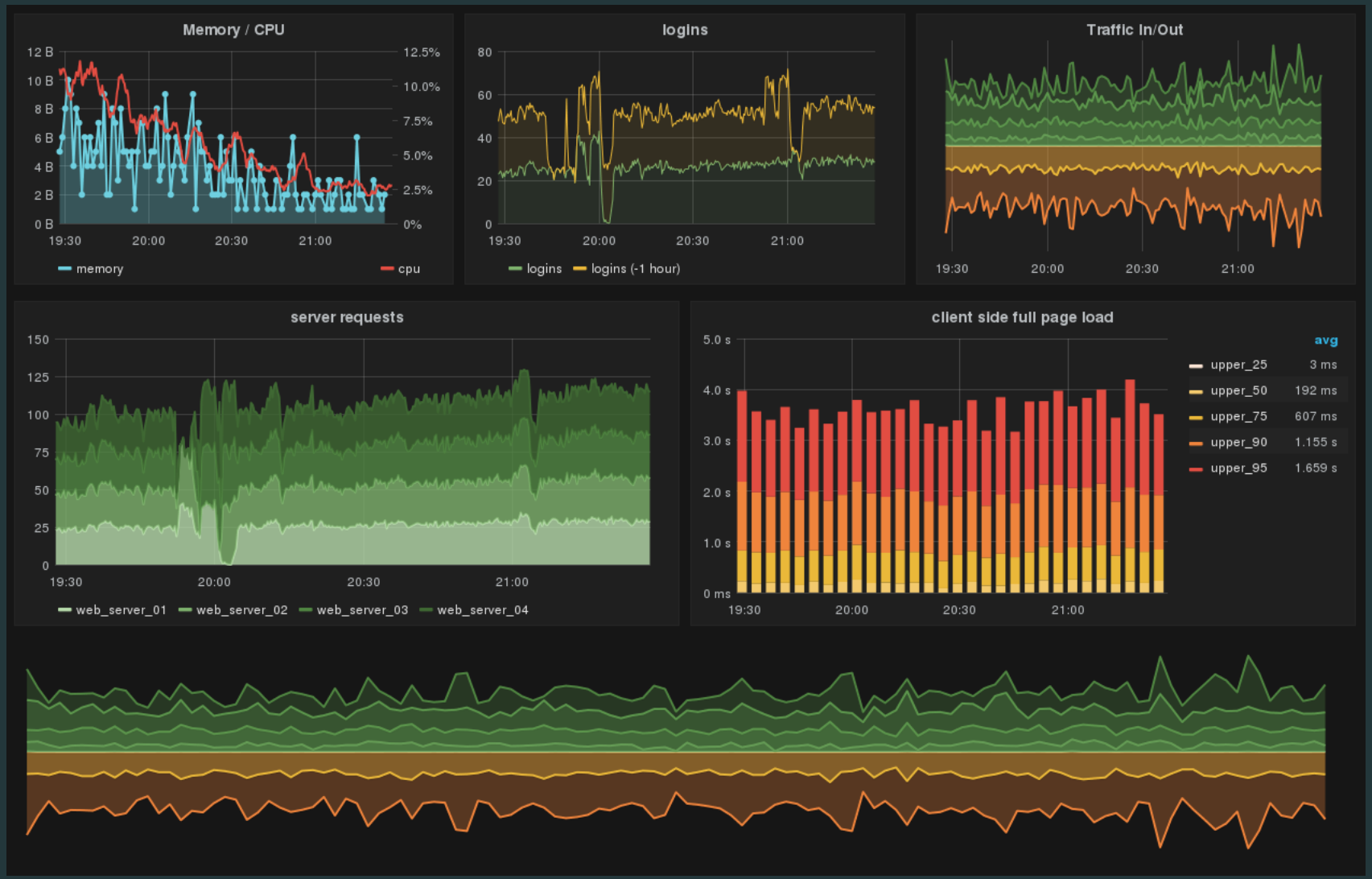
The most important thing in operations

- System metrics
- Application metrics
- Log files
- Business logic monitoring
- Visualizations

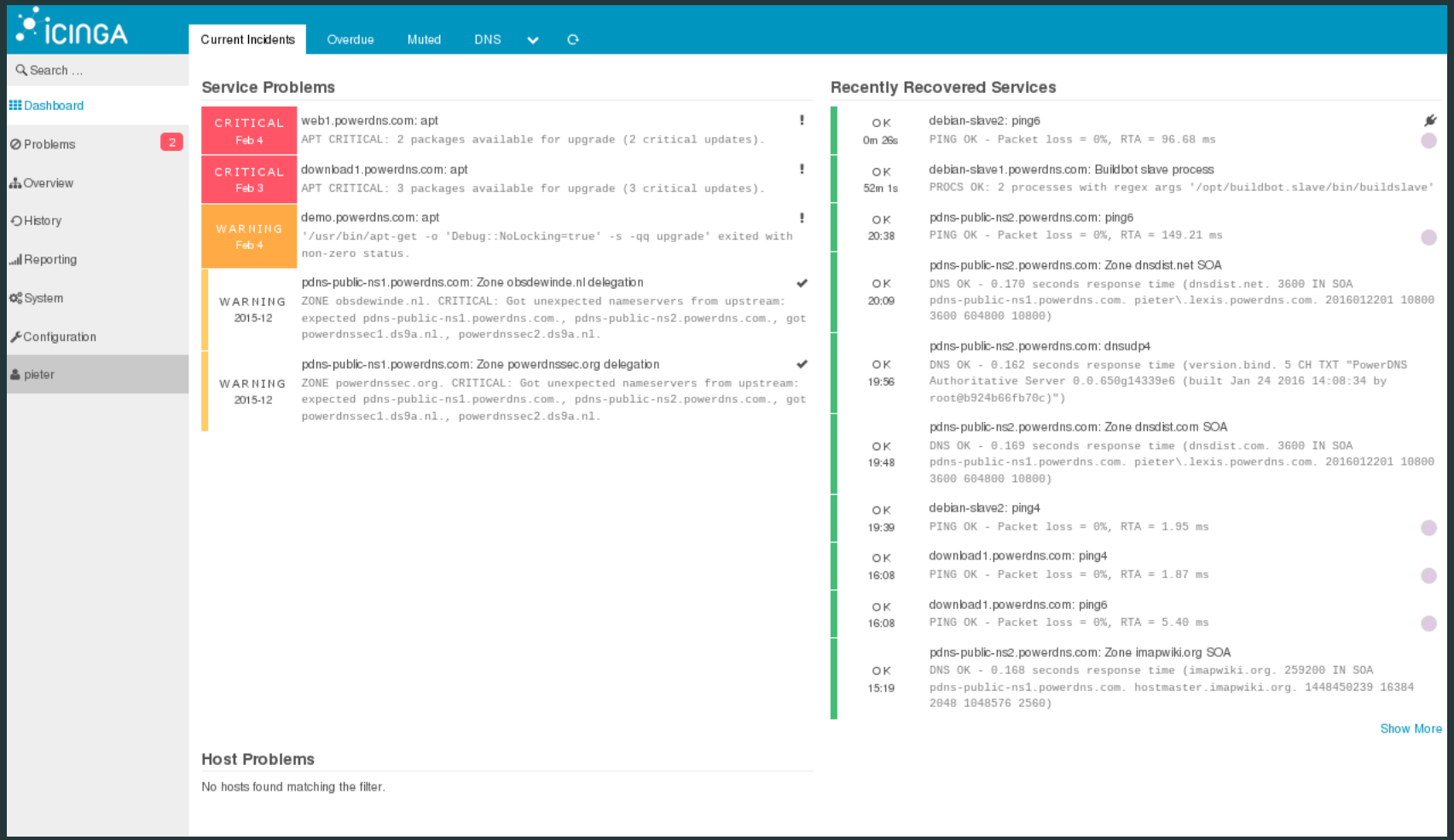
Monitoring II

- ElasticSearch, Logstash, Kibana (ELK)
- Graphite
- InfluxDB
- Collectd
- Nagios/Icinga

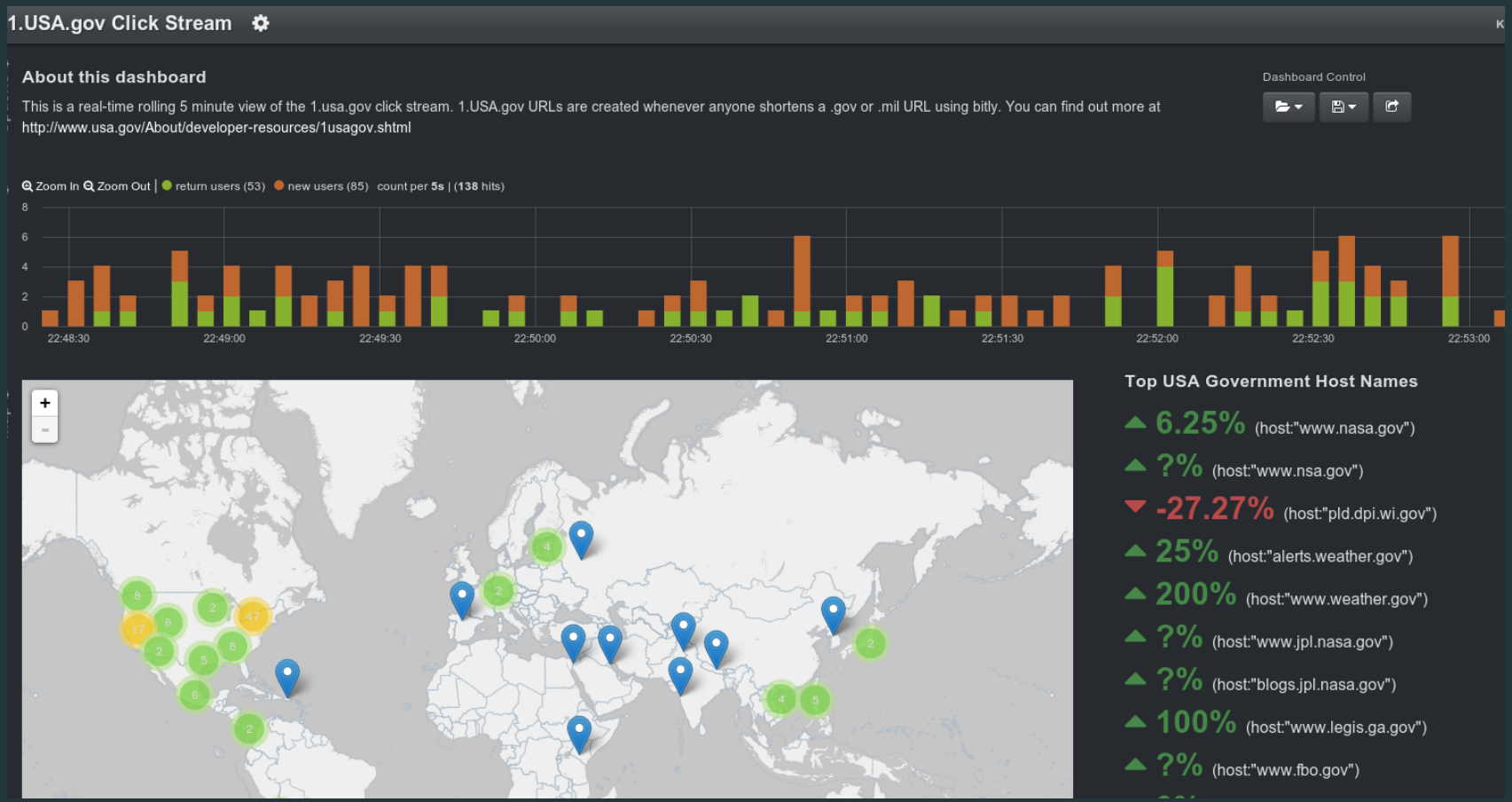
Eyecandy – Grafana



Eyecandy – Icinga



Eyecandy – Kibana



Tools

First off

“Practice is just around the corner”

— Maarten van Steen

Small history

- cfEngine (1993)
- bcfg2 (2004)
- Puppet (2005)
- Chef (2009)
- Salt (2011)
- Ansible (2012)
- cfgmgmt (2016)

Elements

- Inventory and Facts
- Files and Templates
- Command runners
- Domain Specific Language (DSL)
- Omnipotence
- Code re-usability

Tools – Ansible

Intro

- Written in Python
- YAML as DSL
- Jinja2 for templates
- “agentless” – Only SSH access required
- Push-based

In a **playbook**, **host**s have one or more **role**s that are described using **task**s.

Playbook

```
---
- hosts: webservers
  vars:
    http_port: 80
    max_clients: 200
  remote_user: root
  tasks:
  - name: ensure apache is at the latest version
    yum: name=httpd state=latest
  - name: write the apache config file
    template: src=/srv/httpd.j2 dest=/etc/httpd.conf
    notify:
    - restart apache
  - name: ensure apache is running (and enable it at boot)
    service: name=httpd state=started enabled=yes
  handlers:
    - name: restart apache
      service: name=httpd state=restarted
```

Role

```
- name: Install pgdg repository (Debian/pgdg)
  apt_repository: repo="deb http://apt.postgresql.org/pub/repos/apt/ {{
  ↪   ansible_distribution_release }}-pgdg main" update_cache=yes
  when: postgresql_flavor is defined and postgresql_flavor == "pgdg"

- name: Install PostgreSQL (Debian)
  apt: name=postgresql{{ '-' ~ postgresql_version if postgresql_version is defined
  ↪   else '' }}

- name: Create conf.d
  file: path={{ postgresql_conf_dir }}/conf.d state=directory owner=postgres
  ↪   group=postgres

- name: Set conf.d include in postgresql.conf
  lineinfile: line="include_dir 'conf.d'" dest={{ postgresql_conf_dir
  ↪   }}/postgresql.conf backup=yes
  notify: Reload PostgreSQL
  when: "{{ postgresql_version | version_compare('9.3', '>=') }}"

- name: Install pg_hba.conf
  template: src=pg_hba.conf.{{ ansible_os_family | lower }}.j2 dest={{
  ↪   postgresql_conf_dir }}/pg_hba.conf owner=postgres group=postgres mode=0400
  ↪   backup=yes
  notify: Reload PostgreSQL

- name: Ensure PostgreSQL is running
  service: name={{ postgresql_service_name }} enabled=yes state=started
```

Template

```
##
## This file is maintained by Ansible - CHANGES WILL BE OVERWRITTEN
##

{% if postgresql_pg_hba_local_socket is not defined or postgresql_pg_hba_local_socket %}
# "local" is for Unix domain socket connections only
local   all             all                                     peer
{% endif %}
{% if postgresql_pg_hba_local_ipv4 is not defined or postgresql_pg_hba_local_ipv4 %}
# IPv4 local connections:
host    all             all             127.0.0.1/32            md5
{% endif %}
{% if postgresql_pg_hba_local_ipv6 is not defined or postgresql_pg_hba_local_ipv6 %}
# IPv6 local connections:
host    all             all             ::1/128                 md5
{% endif %}

# Entries configured in postgresql_pg_hba_conf follow
{% if postgresql_pg_hba_conf is defined %}
{% for line in postgresql_pg_hba_conf %}
{{ line }}
{% endfor %}
{% endif %}
```

Tools – Puppet

Intro

- Written in Ruby
- Puppet DSL
- Embedded Ruby (ERB) as template language
- Requires agent on nodes

Modules contain **manifest**s and other data. The manifests describe **resource**s with dependencies that are compiled into a **catalog** that is shipped to the **node**.

Manifest

```
case $operatingsystem {
  centos, redhat: { $service_name = 'ntpd' }
  debian, ubuntu: { $service_name = 'ntp' }
}

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

service { 'ntp':
  name      => $service_name,
  ensure    => running,
  enable    => true,
  subscribe => File['ntp.conf'],
}

file { 'ntp.conf':
  path    => '/etc/ntp.conf',
  ensure  => file,
  require => Package['ntp'],
  source  => "puppet:///modules/ntp/ntp.conf",
}
```

Template

```
<%# Non-printing tag ↓ -%>
<% if @keys_enable -%>
<%# Expression-printing tag ↓ -%>
keys <%= @keys_file %>
<% unless @keys_trusted.empty? -%>
trustedkey <%= @keys_trusted.join(' ') %>
<% end -%>
<% if @keys_requestkey != '' -%>
requestkey <%= @keys_requestkey %>
<% end -%>
<% if @keys_controlkey != '' -%>
controlkey <%= @keys_controlkey %>
<% end -%>

<% end -%>
```

Tools – Salt

Intro

- Written in Python
- Modules can be written in Python
- Reactive orchestration
- Jinja2 as template language
- Both agentless and running an agent is supported

A **minion** is brought into a **state** by the content of the **master**'s **salt state**

Example

```
vim:
  pkg.installed

/etc/vimrc:
  file.managed:
    - source: salt://edit/vimrc
    - mode: 644
    - user: root
    - group: root
```

Tools – Chef

Intro

- Written in Ruby
- Configuration in Ruby

cookbooks have **recipe**s that contain the **resource**s that will configure the **node**s

Cookbook

```
package 'apache2'

service 'apache2' do
  supports :status => true
  action [:enable, :start]
end

template '/var/www/html/index.html' do
  source 'index.html.erb'
end
```

Thank you!