Hekaton

Large Scale System Management with Python

Akron Linux Users Group - May 2009

who are you?

@mcrute
mcrute@gmail.com
mike.crute.org

AGINTERACTIVE

The Road Ahead

- python tutorial
- use-case
- first-try solution
- python solution

Python

here be snakes

From 50,000 Feet

- python basics
- the standard library
- useful modules
- examples

Why Python Rocks

- very newbie friendly
- multi-paradigm, do things your way
- simple tasks are simple
- runs anywhere
- awesome standard library

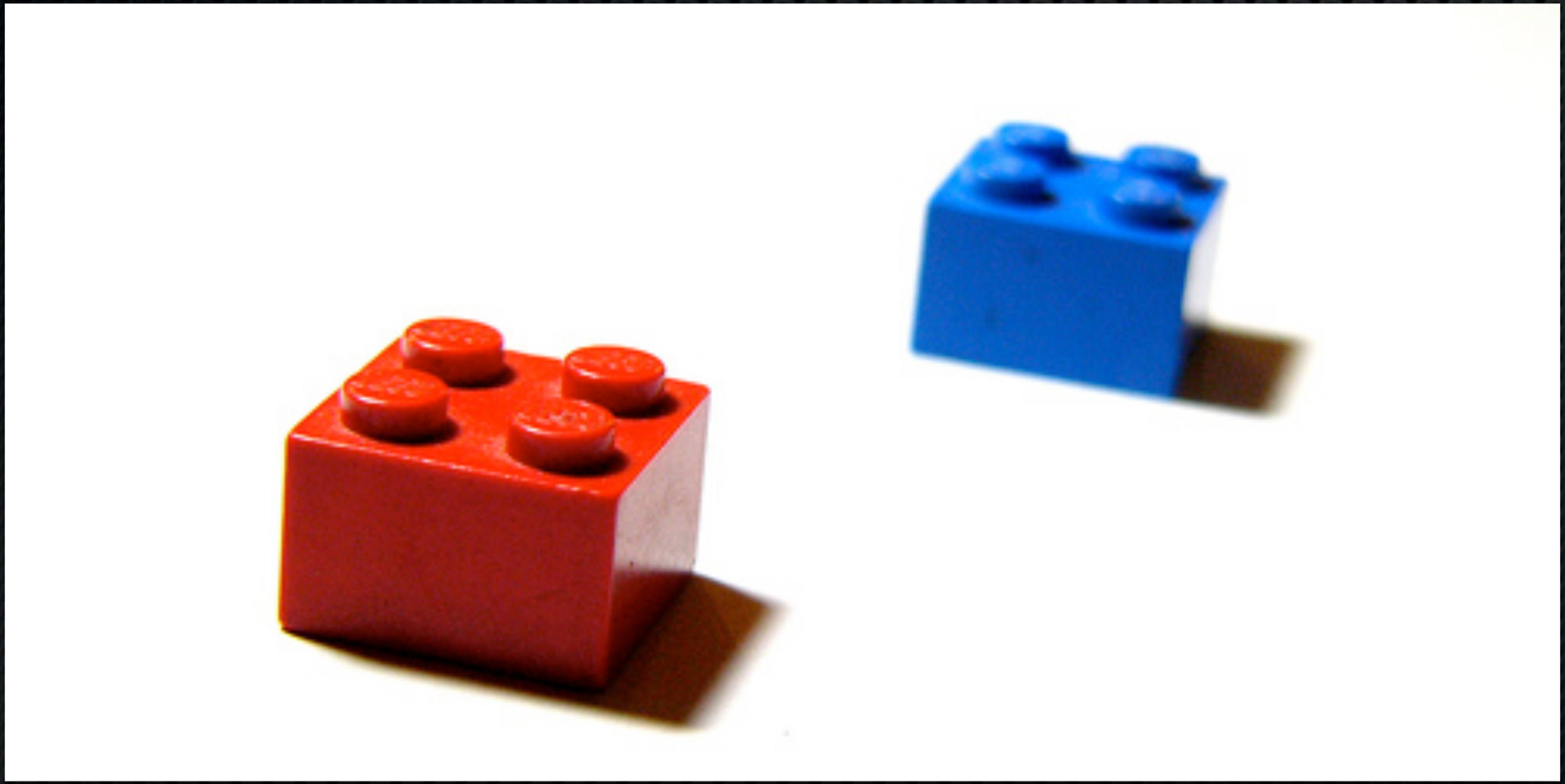
Code, Please

```python
import os
import shutil

def make_backup(to_backup):
    try:
        location = os.environ.get['BACKUP_LOCATION']
    except KeyError:
        raise Exception('Backup location not found.')

    if os.path.isdir(to_backup):
        shutil.copytree(to_backup, location)
    else:
        shutil.copyfile(to_backup, location)

make_backup('~/Documents')
```



The Basics

An introduction to python

Language Overview

- types
- standard constructs
- error handling
- classes
- some examples

Python is Simple

- no pointers
- no types
- everything is an object
- indentation matters

Poking About

- just run `python` for an interactive shell
- dir(*object*) to inspect it
- help(*object*) for some documentation

Types

- mutable types - list, dict, set, file
- singleton types - int, float, str, bytes, booleans, None
- immutable types - tuple
- user-defined subtypes

Tuples and Set

- tuples are immutable lists
  - created using ( )
  - can cast a list to a tuple
- sets are lists of unique members
  - support mathematical set operations like union
  - fronzenset equivalent to tuple

Dictionary Type

- called hash maps in many languages
- map keys to values
- do not preserve order
- keys must be hashable
- support iteration through views
- denoted by { }

Standard Constructs

Support for all the standard constructs

if Statements

```
if os.path.exists('/tmp'):
    # do something
elif os.path.exists('/test'):
    # do something else
else:
    # do something more
finally:
    # always do this
```

while Statements

```
while True:
    stuff = do_some_stuff()

    if stuff:
        break
```

for Statements

```
for i, word in enumerate(word_list):
    print('{0} => {1}'.format(i, word))
```

Classes

```
class User(object):
    _logged_in = False

    def __init__(self, username, name='John Doe'):
        self.username = username
        self.name = name
        self.password = name + username

    @property
    def logged_in(self):
        return self._logged_in

    def send_username(self, socket):
        self.login()

        if self.logged_in:
            socket.send(self.username)
```

Classes

```python
user = User('jdoe')
print(user.real_name) # John Doe
print(user.logged_in) # True
```

Error Handling

```
try:
    data_file = open('/tmp/foo', 'w')
    data_file.write(‘testing’)
except IOError:
    print(“Can't open file.”)
    raise
finally:
  if hasattr(data_file, ‘close’):
      data_file.close()
```

import Code

```
import os
import os.path
import os.path as path_module
from os.path import isdir, isfile
```

Standard Library

Important Modules

- os - operating system functions
- shutil - shell functions
- sys - system/python functions
- urllib - curl-like functions
- re - regular expressions
- optparse - getopt replacement

Useful Modules

- json - deal with json data
- configparser - parse ini files
- subprocess - launch subprocesses
- xml.etree - process xml
- datetime - deal with dates

Check out the docs

http://docs.python.org/library/

More Modules

Python Package Index

- repository for third-party modules
- easy to install
- modules for most common tasks
- even contains full applications
- analogous to CPAN

Get It

- http://peak.telecommunity.com/dist/ez_setup.py
- run this as root
- you now have easy_install
- run easy_install as root

easy_install

- python package manager
- downloads packages from pypi
- can also upgrade those packages

Removing Packages

Try not to think about it

Great Modules

- dateutil - parse dates
- pyyaml - deal with yaml files
- mysqldb - talk to mysql servers
- path - simple path manipulation
- LOTS more...

Get modules

http://pypi.python.org

Examples

Questions

Endeca

find the needle in the haystack

The Problem

Complex Environment

- 5 different code environments
- 3 different data environments
- must be kept in-sync
- servers managed by operations team
- application managed by content team

High Visibility

- everybody wants it
- business wants control
- drives our main sites

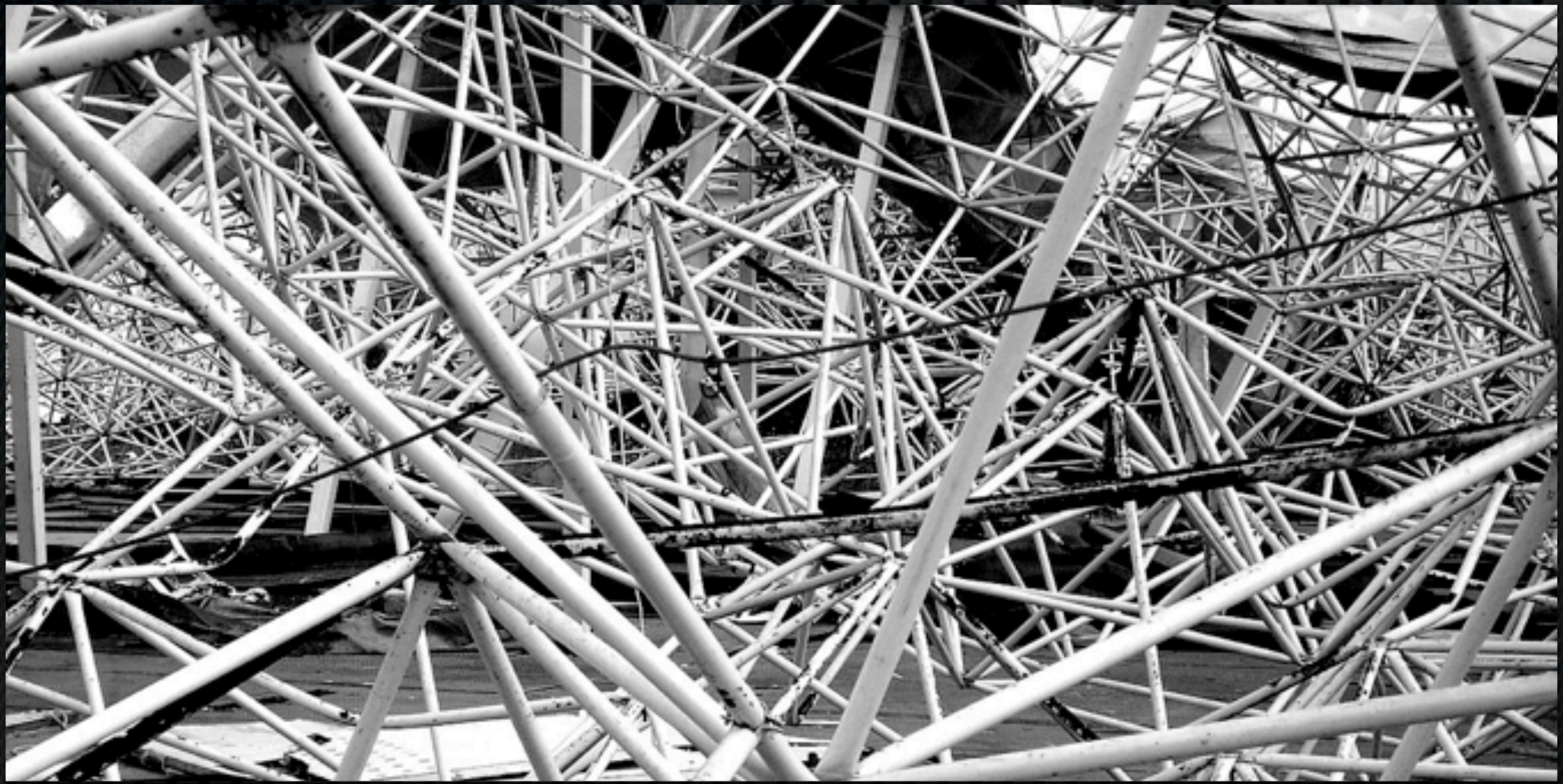
Limited Resources

- small core team
- automation is a must

Vendor Framework

- operational framework provided
- still evolving
- very robust in a single environment
- designed primarily for operations staff
- supplementation required

"The only way we can succeed is through ruthless automation."



First Attempt

Approach

- bash scripts
- mini-framework in bash

Benefits

- really quick to develop
- pretty simple
- good enough for now
- let us use vendor tools

Downsides

- gets complex quickly
- not easy to extend
- not everyone groks bash
- still required engineering involvement

End Result

It kinda sucked, so we dumped it.

The Solution

Approach

- python framework for management
- leverages our operations tools
- simple to use and extend
- understands web-enabled commands

Benefits

- reduced time to deploy features
- easier for others to understand
- easier to hide complexity
- solves our entire problem

Downsides

- took time to develop
- requires some knowledge of python
- more complex

Hekaton

Our little hero

Hekaton?

```
endeca | cut -dn -f2
deca == 10
deca * 10 == 100
100 | greek = hekaton
```

It is

- 100% python
- simple
- application framework
- ties into our webops framework

It isn’t

- actually an application
- terribly general purpose
- open-sourced

Simple Commands

```
@hekaton_command('show-config')
def show_config(info, sysargs):
    "print the loaded config"
    print(get_config())
    return 0
```

Complex Commands

```python
@hekaton_command('do-overlay')
class OverlayController(BaseCommand):
    "replace $WORKING_DIR definitions in control scripts"

    def run_command(self):
        for script in (self.info.appdir/'control').files('*.sh'):
            lines = script.lines()

            for i, line in enumerate(lines[:]):
                match = self._working_dir_def.match(line)
                if match:
                    replacement = self.working_dir.format(self.appname)
                    lines[i] = line.replace(match.group(1), replacement)

            script.write_lines(lines)

        return 0
```

Command-line

- everything is a sub-command
- provides useful help and usage
- tab completion for all

Web Commands

```
@hekaton_command('agi-merch-rule-moves')
class MerchRulesWSGIAppController(PipelineEnabledCommand):

    def __init__(self, request, response_class):
        self.request = request
        self.response = response_class()

    def run_web_command(self, environ, start_response):
        if not self.request.GET.get('action'):
            tmpl = get_template('merch_rule_moves.hjin')
            self.response.body = tmpl.render(**self.tmpl_vars)
        else:
            try:
                raw_action = self.request.GET.get('action', '')
                getattr(self, action_attr)()
            except AttributeError, exc:
                self.bad_request()

        return self.response(environ, start_response)
```

Environment Handling

- completely abstracted from commands
- paths are mangled automatically
- configuration provided per-environment
- hostnames provided per-environment

Scheduler

- works a lot like cron
- “free” with the framework
- easier to run hekaton commands
- easier to manage for us
- possibly not the best solution for everyone

Scheduler

```
AGI:
    all:
        scheduler:
            - job_name: build-indexes
              run_every: 1 hours
            - job_name: cleanup-logs
              run_every: '1 day at 3:00'

    production:
        scheduler:
            - job_name: mail-reports
              run_every: '1 week at 0:00'
              with_args: '-t 3'
```

Demo

Questions