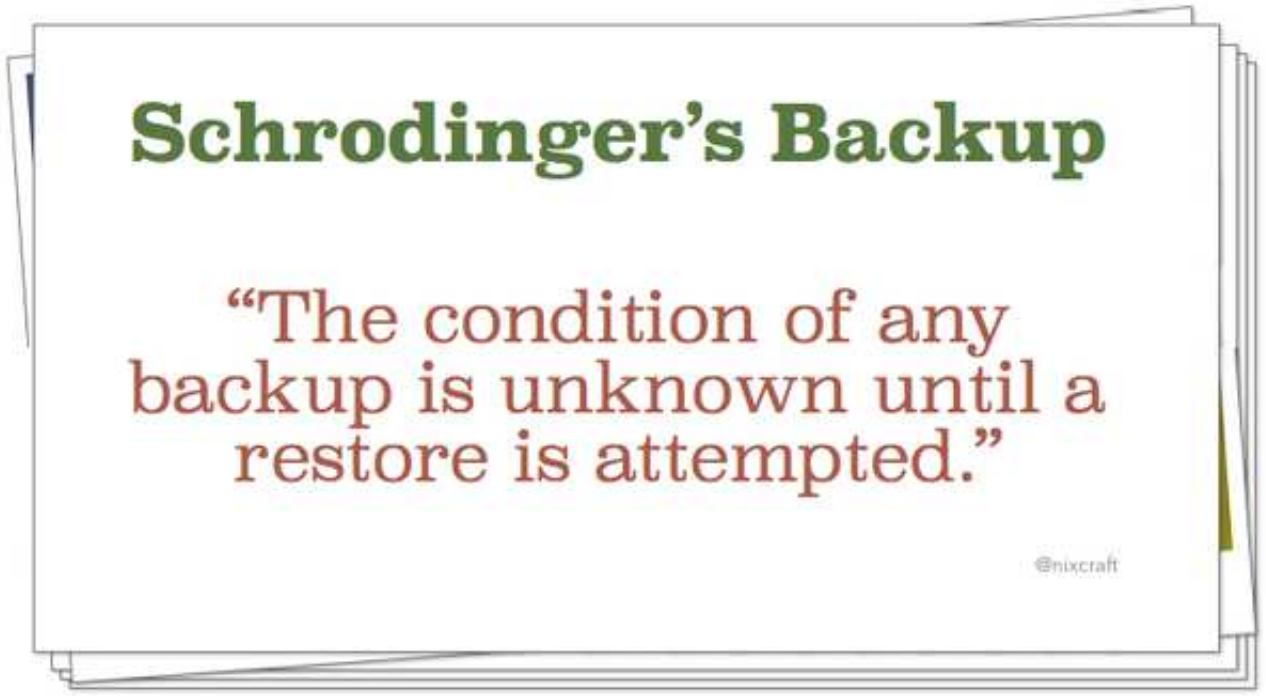
CS615 - Aspects of System Administration

Configuration Management

Department of Computer Science
Stevens Institute of Technology
Jan Schaumann
jschauma@stevens-tech.edu
https://www.cs.stevens.edu/~jschauma/615A/

To the backups!



HW Review

Use your words!

Entropy is the Enemy

The entropy of an isolated system never decreases.

Entropy is the Enemy

A static system is a useless system. A useful system is being used.

- data is processed; files are created, modified, removed
- software is added, upgraded, removed
- systems are created, copied, decommissioned
- instances / containers are even more short-lived, coming into existence and disappearing again as needed

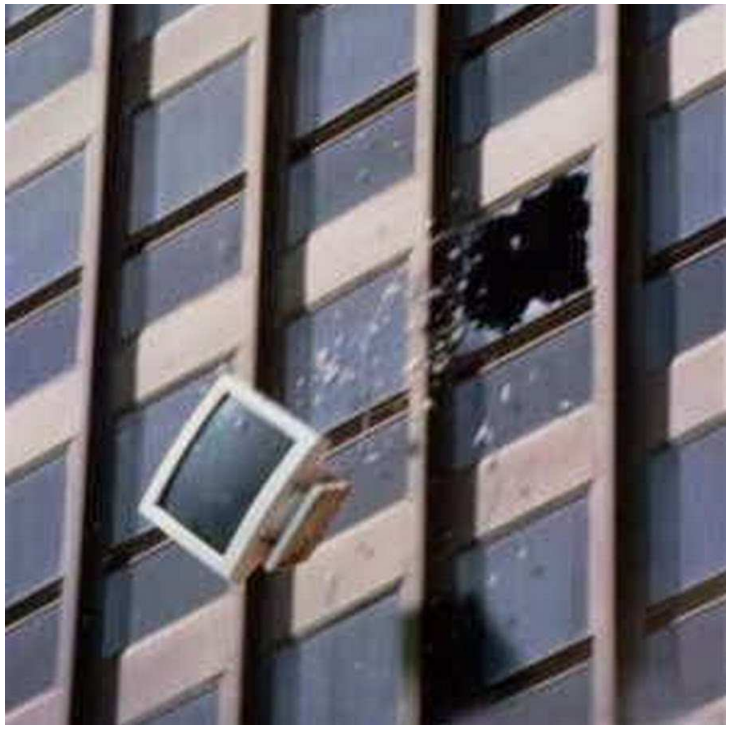
Single Systems are Fragile

Individual systems created and configured by hand are fragile. Our processes need to be repeatable, automated, reliable.

Recall previous lectures:

- OS installation
- package management
- multi-user basics
- automation
- recovery / restores

Reproducable



“Never trust a computer you can’t throw out the window.” – Woz

Evolution of Configuration Management

"I set up a server over here to do X. Replicate that setup on all the others."

Evolution of Configuration Management

"I set up a server over here to do X. Replicate that setup on all the others."

"I know how to do this! Watch me!"

```
$ ssh root@server1
# rsync -e ssh -avz / server2:/
```

"`/etc`? What's that?"

Evolution of Configuration Management

	shareable content	unshareable content
static data	`/usr` `/opt`	`/boot` `/etc`
variable data	`/home` `/var/mail`	`/tmp` `/var/run`

Every Sysadmin ever...

1. `scp(1)`

Every Sysadmin ever...

1. `scp(1)`
2. `rsync(1)`

Every Sysadmin ever...

1. `scp(1)`
2. `rsync(1)`
3. some sort of parallel `ssh(1)` of the above

Every Sysadmin ever...

1. `scp(1)`
2. `rsync(1)`
3. some sort of parallel `ssh(1)` of the above
4. switch to *pull*

Every Sysadmin ever...

1. `scp(1)`
2. `rsync(1)`
3. some sort of parallel `ssh(1)` of the above
4. switch to *pull*
5. add mutual authentication

Every Sysadmin ever...

1. `scp(1)`
2. `rsync(1)`
3. some sort of parallel `ssh(1)` of the above
4. switch to *pull*
5. add mutual authentication
6. but effectively ignore mismatches, because doing things the right way is difficult and inconvenient

Every Sysadmin ever...

1. `scp(1)`
2. `rsync(1)`
3. some sort of parallel `ssh(1)` of the above
4. switch to *pull*
5. add mutual authentication
6. but effectively ignore mismatches, because doing things the right way is difficult and inconvenient
7. switch to *push* with remote dæmon

Every Sysadmin ever...

1. `scp(1)`
2. `rsync(1)`
3. some sort of parallel `ssh(1)` of the above
4. switch to *pull*
5. add mutual authentication
6. but effectively ignore mismatches, because doing things the right way is difficult and inconvenient
7. switch to *push* with remote dæmon
8. write an inventory database

Every Sysadmin ever...

1. `scp(1)`
2. `rsync(1)`
3. some sort of parallel `ssh(1)` of the above
4. switch to *pull*
5. add mutual authentication
6. but effectively ignore mismatches, because doing things the right way is difficult and inconvenient
7. switch to *push* with remote dæmon
8. write an inventory database
9. deploy a well-known CM system

Every Sysadmin ever...

1. `scp(1)`
2. `rsync(1)`
3. some sort of parallel `ssh(1)` of the above
4. switch to *pull*
5. add mutual authentication
6. but effectively ignore mismatches, because doing things the right way is difficult and inconvenient
7. switch to *push* with remote dæmon
8. write an inventory database
9. deploy a well-known CM system

Finally: find something it can't do, goto 1.

Base configuration vs. service definition

Your servers have *unique*, yet predictable properties. E.g.:

- network configuration
- critical services: DNS, NTP, Syslog
- minimum OS / software version
- user management
- common service configuration (e.g. `sshd(8)`)
- ...

Base configuration vs. service definition

Different sets of servers have *shared* properties. For example, consider an HTTP server:

- minimum server software
- appropriate TLS specification
- shared TLS certificate and key
- database configuration
- static content (HTML / JS / CSS files)
- ...

Pets vs. Cattle

“Pets”:

- unique, cheerful hostnames
- single systems grown over time, lovingly configured by hand
- when sick, everybody is very concerned
- slowly nursed back to life

“Cattle”:

- predictable, boring hostnames
- almost identical to all others
- centrally managed, easy to recreate
- when sick, they get taken out back and shot
- quickly replaced by another

Service definitions

```
class syslog {
  include cron
  include logrotate
  package {
    'syslogng' :
      ensure  => latest ,
      require => Service['syslogng'];
  }
  service {
    'syslogng' :
      ensure => running ,
      enable => true;
  }
  file {
    '/etc/syslogng/syslogng.conf':
      ensure  => file,
      source  => 'puppet:///syslog/syslogng.conf',
      mode    => '0644',
      owner   => 'root',
      group   => 'root',
      require => Package['syslog-ng'],
      notify  => Service['syslog-ng'];

    '/etc/logrotate.d/syslog-ng':
      ensure  => file,
      source  => 'puppet:///syslog/logrotate-syslogng',
      mode    => '0644',
      owner   => 'root',
      group   => 'root',
      require => Package['logrotate'];
  }
}
```

Service definitions

```
package "ldap-utils" do
  action :upgrade
end

template "/etc/ldap.conf" do
  source "ldap.conf.erb"
  mode   00644
  owner  "root"
  group  "root"
end

%w{ account auth password session }.each do |pam|
  cookbook_file "/etc/pam.d/common-#{pam}" do
    source   "common-#{pam}"
    mode     00644
    owner    "root"
    group    "root"
    notifies :restart, resources(:service => "ssh"), :delayed
  end
end
```

Service definitions

```
bundle agent sshd(parameter) {
    files:
        "/tmp/sshd_config.tmpl"
            perms     => mog("0600","root","root"),
            copy_from => secure_cp("/templates/etc/ssh/sshd_config",
                                   "cf-master.example.com");

        "/etc/ssh/sshd_config"
            perms     => mog("0600","root","root"),
            create    => true,
            edit_line => expand_template("/tmp/sshd_config.tmpl"),
            classes   => if_repaired("restart_sshd");

    commands:
        restart_sshd::
            "/etc/rc.d/sshd restart"
}
```

CM Requirements

- software installation

CM Requirements

- software installation
- service management / supervising

CM Requirements

- software installation
- service management / supervising
- file permissions / ownership

CM Requirements

- software installation
- service management / supervising
- file permissions / ownership
- static files

CM Requirements

- software installation
- service management / supervising
- file permissions / ownership
- static files
- host-specific data

CM Requirements

- software installation
- service management / supervising
- file permissions / ownership
- static files
- host-specific data

- command-execution

CM Requirements

- software installation
- service management / supervising
- file permissions / ownership
- static files
- host-specific data

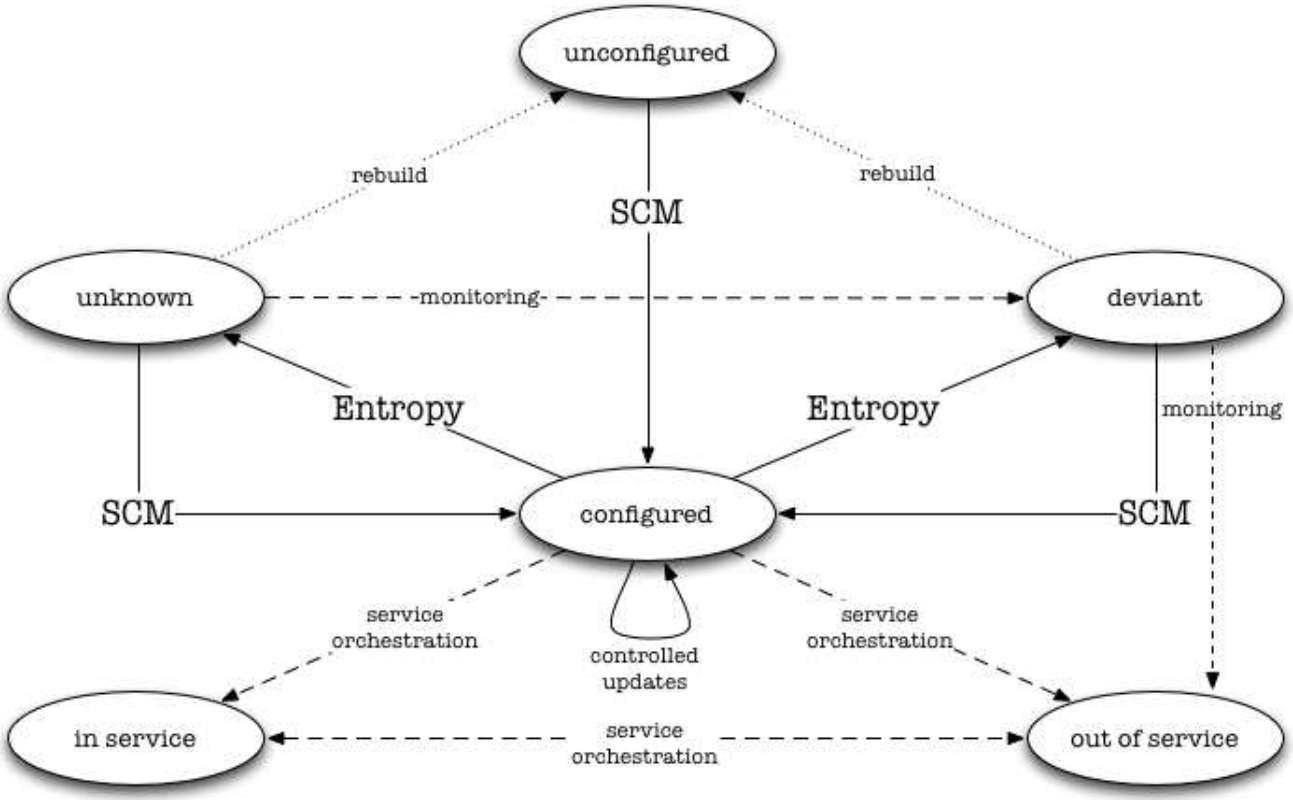
- command-execution
- data collection

One more layer of abstraction...

The objective of a CM system is not to *make changes* on a system.

The objective of a CM system is to *assert state*.

CM States

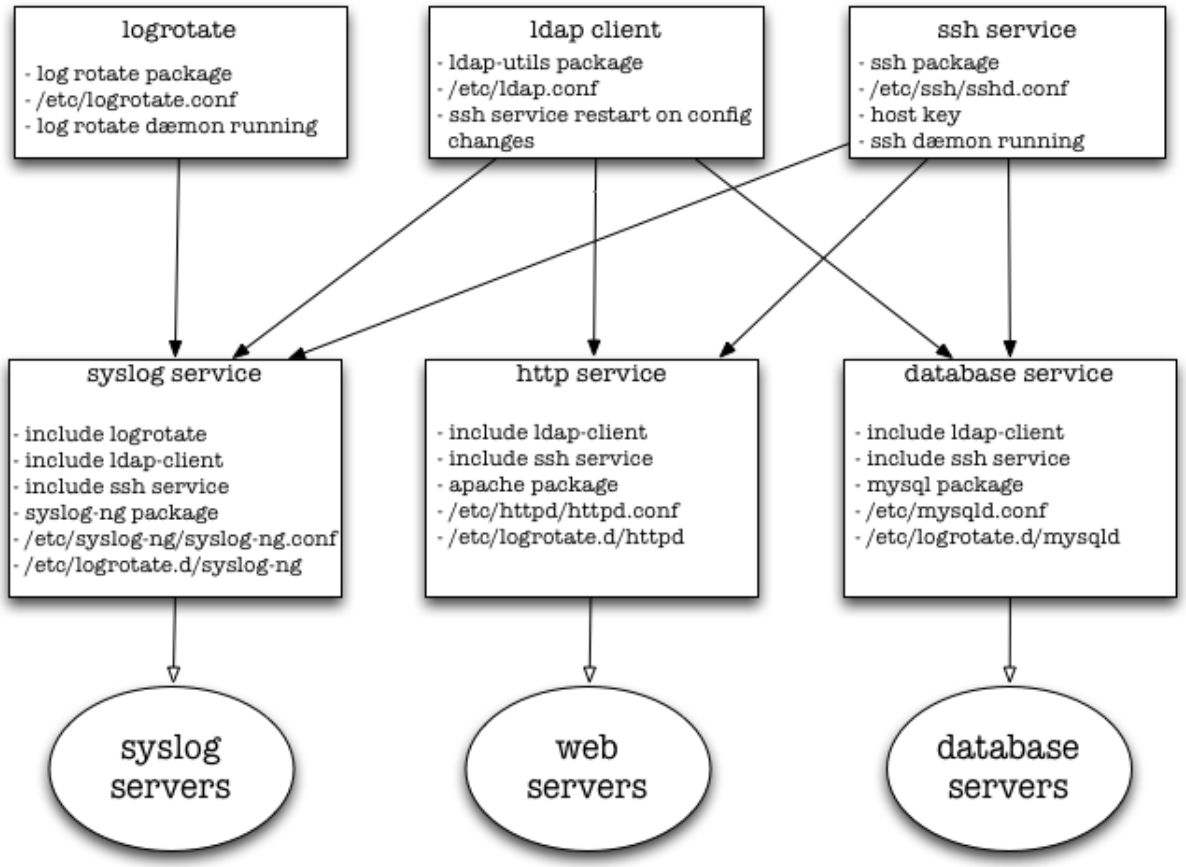

unconfigured
rebuild
SCM
rebuild
unknown
monitoring
deviant
Entropy
Entropy
monitoring
SCM
configured
SCM
service
orchestration
controlled
updates
service
orchestration
in service
service
orchestration
out of service

Circles around things

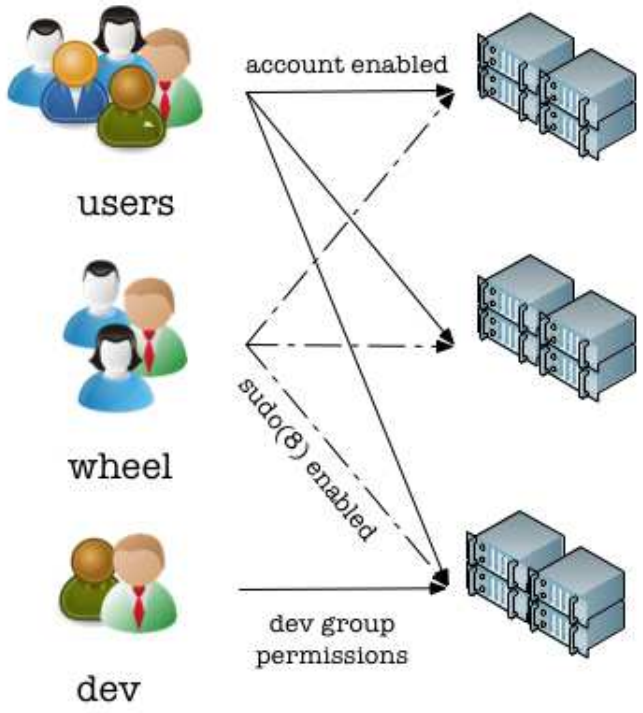
Group your resources into *sets*.

- functional groupings
- services
- users
- hosts

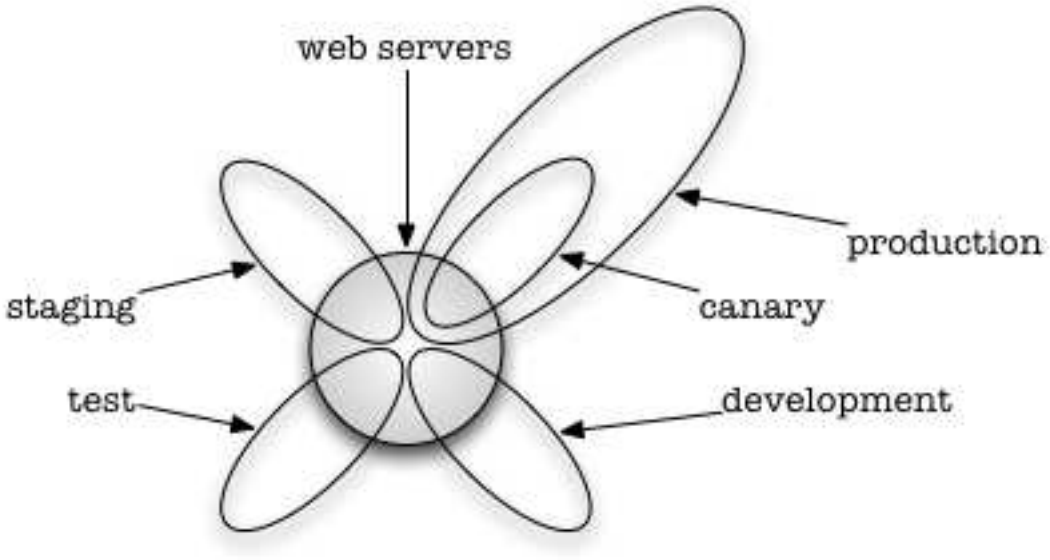
Circles around things



Circles around things



Circles around things



CMs configure complex systems

CM systems are complex themselves.

CM systems are inherently trusted.

CM systems can break everything. To the degree that you can't unbreak things afterwards.

Consider:

- staged rollout of change sets
- automated error detection and rollback
- self-healing properties
- authentication and privilege

Idempotence

CM systems assert state. For this, all operations must be *idempotent*.

$$f(f(x)) \equiv f(x)$$

$$||-1|| \equiv |-1|$$

Idempotence

CM systems assert state. For this, all operations must be *idempotent*.

$$f(f(x)) \equiv f(x)$$

$$||-1|| \equiv |-1|$$

```
$ cd etc
```

Idempotence

CM systems assert state. For this, all operations must be *idempotent*.

$$f(f(x)) \equiv f(x)$$

$$||-1|| \equiv |-1|$$

```
$ cd etc                                          # not idempotent
$ rm resolv.conf
```

Idempotence

CM systems assert state. For this, all operations must be *idempotent*.

$$f(f(x)) \equiv f(x)$$

$$||-1|| \equiv |-1|$$

```
$ cd etc                                                    # not idempotent
$ rm resolv.conf                                            # idempotent
$ echo "nameserver 192.168.0.1" > resolv.conf
```

Idempotence

CM systems assert state. For this, all operations must be *idempotent*.

$$f(f(x)) \equiv f(x)$$

$$||-1|| \equiv |-1|$$

```
$ cd etc                                                    # not idempotent
$ rm resolv.conf                                            # idempotent
$ echo "nameserver 192.168.0.1" > resolv.conf               # idempotent
$ echo "nameserver 192.168.0.2" >> resolv.conf
```

Idempotence

CM systems assert state. For this, all operations must be *idempotent*.

$$f(f(x)) \equiv f(x)$$

$$||-1|| \equiv |-1|$$

```
$ cd etc                                                  # not idempotent
$ rm resolv.conf                                          # idempotent
$ echo "nameserver 192.168.0.1" > resolv.conf             # idempotent
$ echo "nameserver 192.168.0.2" >> resolv.conf            # not idempotent
$ chown root:wheel resolv.conf
```

Idempotence

CM systems assert state. For this, all operations must be *idempotent*.

$$f(f(x)) \equiv f(x)$$

$$||-1|| \equiv |-1|$$

```
$ cd etc                                                  # not idempotent
$ rm resolv.conf                                          # idempotent
$ echo "nameserver 192.168.0.1" > resolv.conf             # idempotent
$ echo "nameserver 192.168.0.2" >> resolv.conf            # not idempotent
$ chown root:wheel resolv.conf                            # idempotent
$ chmod 0644 resolv.conf                                  # idempotent
```

Convergence and Eventual Consistency

Note: idempotence does not guarantee efficiency!

CM systems should ensure changes are:

1. idempotent (well, that part's on you)
2. only applied if needed
3. eventually consistent

This often requires complexity (oh no!), coordination with and awareness of other systems. *Service Orchestration* has developed as a separate, related discipline to help address this.

Distributed Systems

CM systems are *distributed* systems. As such, they are subject to the CAP Theorem:

Consistency: all systems managed by the CM are consistent within their respective service definition.

Availability: the services managed by the CM are kept available, even if no further updates or change sets can be retrieved.

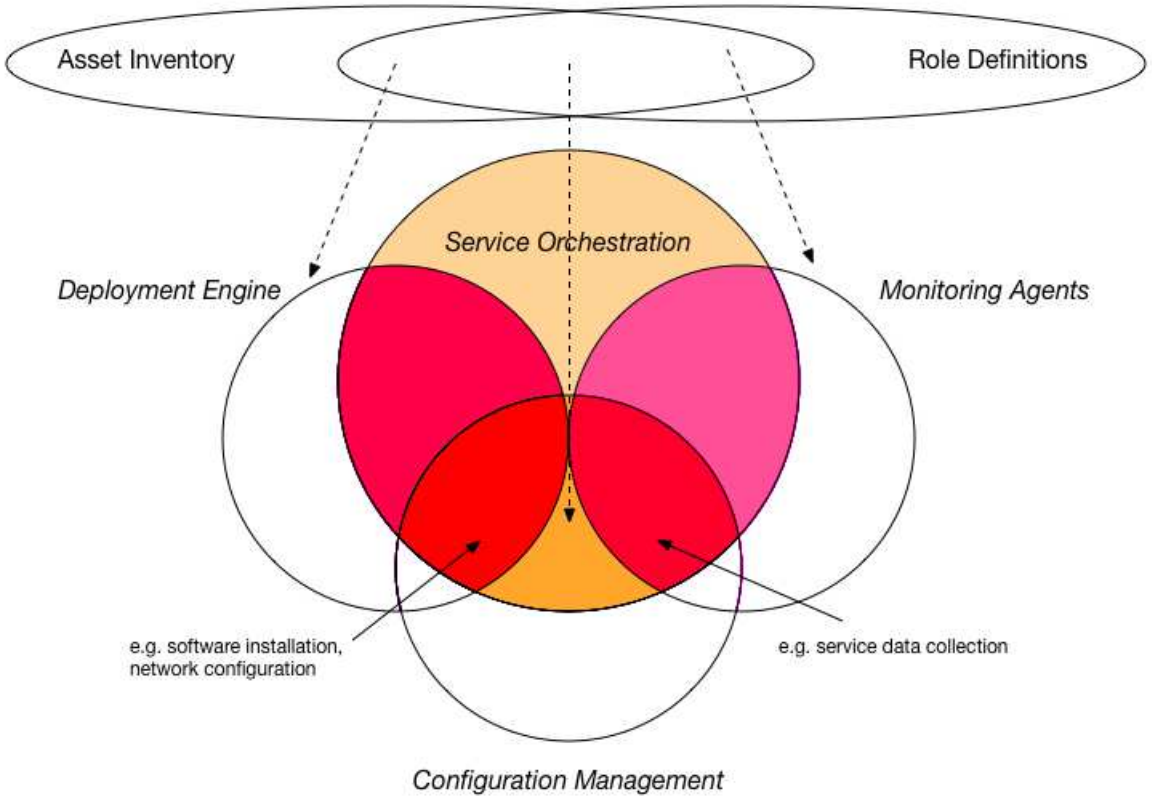
Partition tolerance: the CM system can (continue to) operate despite interruptions between its components; e.g. intermediate (coordinated) changes are not required.

More than just servers...

Configuration Management is not just for servers. You also need to manage configurations for:

- network equipment
- load balancers
- containers
- ...

Overlap with other systems



Reading

Additional topics to research:

- Service Orchestration
- Continuous Deployment / Continuous Integration
- Infrastructure as Code
- Information Technology Infrastructure Library (ITIL)

Relevant links:

- `http://www.infrastructures.org/bootstrap/recovery.shtml`
- `https://is.gd/paZ7qu`
- `https://blog.engineyard.com/2014/pets-vs-cattle`
- `http://markburgess.org/blog_cap.html`
- `http://markburgess.org/blog_cap2.html`
- `https://aws.amazon.com/opsworks/chefautomate/`
- `https://puppet.com/product/managed-technology/aws`