

# INTRO TO PROMISE THEORY AND CFENGINE

BUILDING STABLE RELATIONSHIPS - CS444 LINUX  
ESSENTIALS

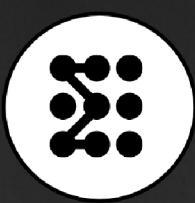
# WHO AM I?

- Nick Anderson
- Husband/Dad
- Sysadmin/Infrastructure Engineer
- CFEEngineer Doer of Things|Factotum @ Northern.tech

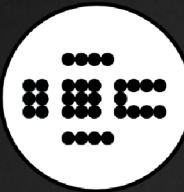


# Northern.tech

Securing the World's  
Connected Devices



Mender



CFEngine



Zener



# RELATIONSHIPS

*It's all about who you know. – They*

- How many things can you know?



DUNBAR

# DUNBAR

- Maximum number of stable relationships one can manage
  - ~ 150
- 2–3 intimate relationships
- 5–7 circle of our very closest friends
- 12–50 tribe
- 150–1500 battalion

# TRUST

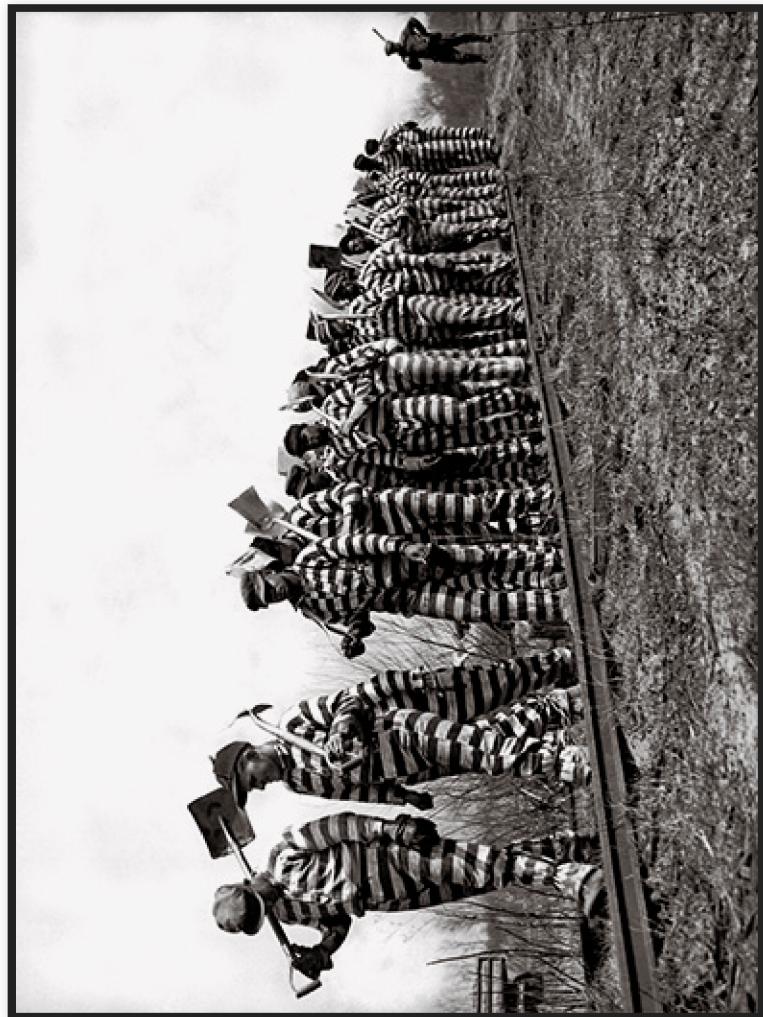
- The foundation of relationships
- Necessary for large systems like societies to function
- Built on repeated interactions over time



# SCALING

- Can technology scale your relationships?  
<https://www.youtube.com/watch?v=07lpED729k8>
  - TLDW; Kind of but no not really

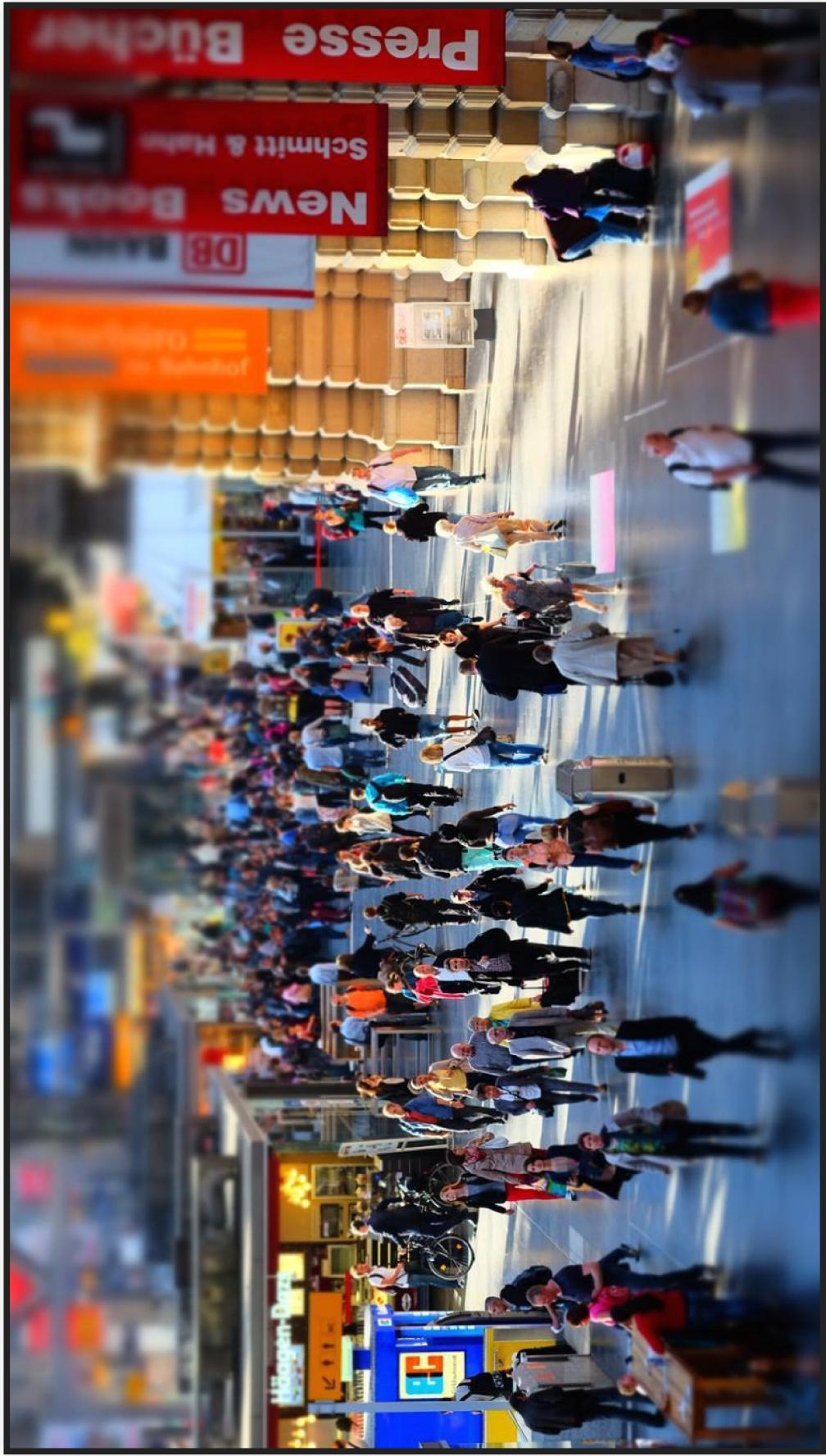
# IMPOSITION



# ACTIVITY

Form a circle

# PUBLISHED INTENT



# ACTIVITY

## Form a circle

# PROMISE THEORY

- Proposed by Mark Burgess in 2004
- Building reliable systems

# PROMISES

- Document intent
- **Agency** is required to make a promise
- Evaluated by user
- Allow us to model uncertainty

# APPLICATIONS OF PROMISE THEORY

- Team Cooperation
  - Team Leadership
- Economics
  - Brexit

# BENEFITS

- Identify points of failure
- Level set expectations

# CFENGINE

An implementation of promise theory



- 
- Autonomous
  - Experienced
  - Sophisticated
  - Flexible

# CFENGINE COMPONENTS



# **cf - agent**

cf - agent is the command you will use most often. It is used to run policy(code) and ensure your system is in the desired state. If you are running any CFEEngine command from the command line, there's a greater than 99% chance that this is it.

# **cf-monitord**

`cf-monitord` monitors various statistics about the running system. This information is made available in the form of **classes** and **variables**.

You'll almost never use `cf-monitord` directly. However the data provided by `cf-monitord` is available to `cf-agent`.

# **cf-execd**

`cf-execd` is a periodic task scheduler. You can think of it like cron on steroids.

By default CFEEngine runs and enforces policies every *five minutes*. `cf-execd` is responsible for making that happen.

# cf - serverd

cf - serverd runs on the CFEEngine server, as well as all clients.

- On servers it is responsible for serving files to clients.
- On clients it accepts cf - runagent requests cf - runagent allows you to request ad-hoc policy runs. I rarely use it.

# PROMISES

# ANATOMY OF A PROMISE

```
type:  
context::  
  "promiser" -> "promisee"  
    - - - - -  
  attribute1 => "value" ,  
  attribute2 => body;  
    - - - - -  
    - - - - -  
    - - - - -  
    - - - - -
```

## PROMISE ATTRIBUTES

- Separated by commas
- Vary by **promise type**
- Value is quoted string or unquoted object (function/body/bundle)

## EXAMPLE PROMISE

```
bundle agent main {
  files:
    linux:
      "/tmp/example" -> { "Instructor", "Students" }
      create => "true",
      touch => "true",
      action => warn_only;
}
```

warning: Need to touch (update time stamps) for '/tmp/multiple-attr'

# BUNDLES

- collection of promises
- logical grouping
- can have parameters
- **ARE NOT FUNCTIONS**

# A BUNDLE FOR APACHE WEB-SERVER MIGHT

- ensure the apache2 package is installed
- ensure the content in the config file is correct
- ensure content is present for serving
- ensure proper permissions of files
- ensure the httpd process is running
- ensure the httpd process is restarted when the configuration changes

# ANATOMY OF A BUNDLE

```
bundle type name
{
  type:
  context:
    "promiser" -> { "promisee" }
      attribute1 => "value",
      attribute2 => value;

  type:
  context:
    "promiser" -> { "promisee" }
      attribute1 => "value",
      attribute2 => value;
}
```

Bundles apply to the binary that executes them. E.g.,  
agent bundles apply to cf-agent while Server  
bundles apply to cf-serverd.

Bundles of type common apply to any CFEEngine  
binary.

**WHAT COMPONENT(S) USE THIS  
BUNDLE?**

## EXAMPLE 1

```
bundle common globals
{
    vars:
        "tool_path" string => "/srv/tools"
}
```

## EXAMPLE 2

```
bundle server my_access_rules
{
  access:
    "$(globals.tool_path)"
      admit_ips => { "192.168.0.0/24"  };
}
```

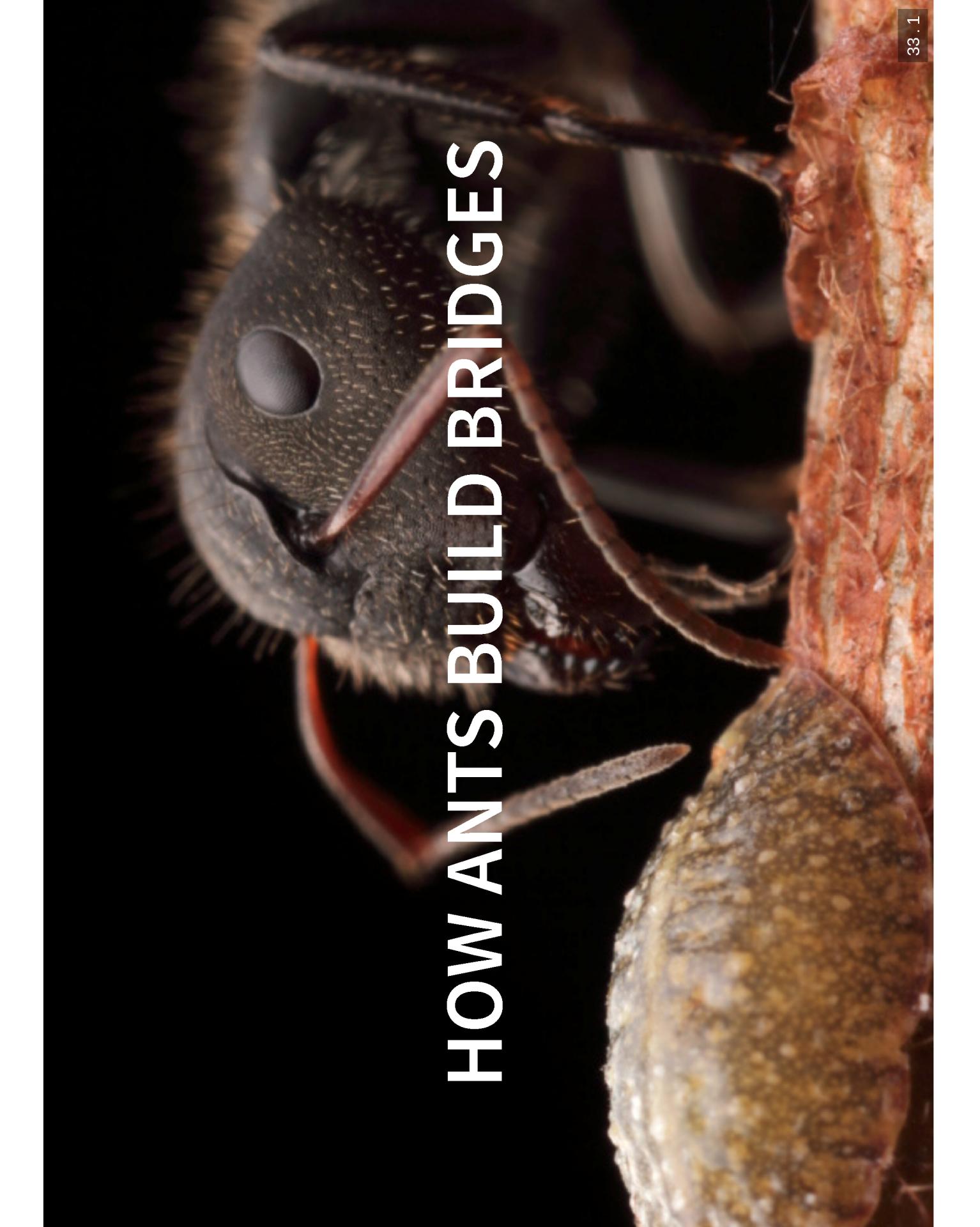
## EXAMPLE 3

```
bundle agent my_policy
{
    vars:
        "config[PermitRootLogin]" string => "no";
        "config[Port]" string => "22";
    files:
        "/etc/ssh/sshd_config"
        edit_line => set_line_based( "my_policy.config", " ", );
}
```

## EXAMPLE 4

```
bundle monitor measure_cf_serverd
{
    vars:
        "pid[cf-serverd]" string => readfile( "$(sys.piddir)/cf-serverd.pid" , 4k )
        "reg_stat[rss]" string =>"(?:[^\s+]*\s+){23}([^\s]+)(?:.*"
    measurements:
        "/proc/$(pid[cf-serverd])/stat" handle => "cf_serverd_vsize",
        stream_type => "file",
        data_type => "int",
```

**DO WE HAVE TIME, AND DO YOU WANT  
TO TRY YOURSELF?**



# HOW ANTS BUILD BRIDGES

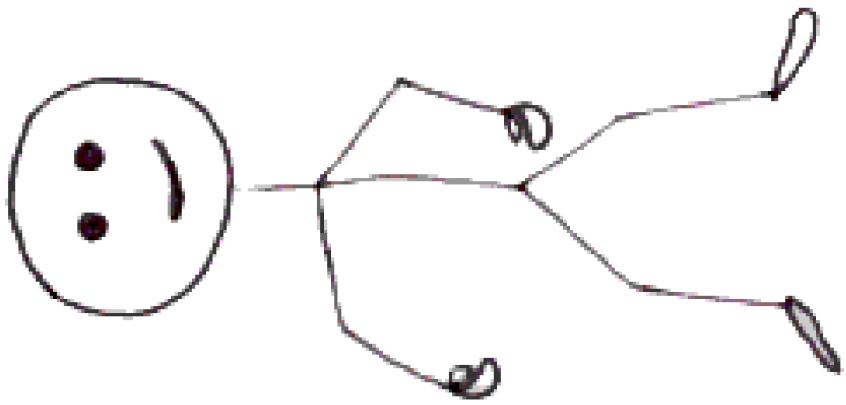
# QUESTIONS?

# CONNECT WITH ME

- Email: <[nick@cmdln.org](mailto:nick@cmdln.org)> | <[nick.anderson@northern.tech](mailto:nick.anderson@northern.tech)>
- Twitter: [@cmdln\\_](https://cmdln_.com)
- Careers at Northern.tech:  
<https://northern.tech/careers>
- Company Blog:
- My Blog: <http://cmdln.org>

# THANKS

This is  
my  
thank you  
dance!





# ADDITIONAL RESOURCES

# **ADDITIONAL RESOURCES**

## **CONTINUED**

## **Basic Concepts (part 1)**

Basic promise theory concepts (YouTube)

## **The rules of delegation (part 2)**

How delegation works (YouTube)

## **Scaling Cooperation (part 3)**

Scaling relationships (YouTube)

## **CFEngine Tutorial**

A comprehensive self directed tutorial to learn CFEngine 3.

## **CFEngine Zero to Hero**

A quick intro targeting some of the most important things to know getting started with CFEngine 3.

