Setting the stage

Why are we here ?

- *"Plus ca change, plus c'est la meme chose"*

  "There is nothing new under the sun"
- *Our goals:*
  - *Take a critical look at our methods*
  - *Understand that we are not alone*
  - *Identify and disrupt bad habits*
  - *Do the same things better*

Old issues in maintaining computing infrastructures

- Service Configuration
  - Drift
    - it was working yesterday !
  - Reproducibility
    - Something happened...
  - Semantics
    - I did the thing, and then the thing happened
  - Maintenance
    - Not my problem
  - Portability
    - Now make it work over there

Old issues in maintaining computing infrastructures

- Monitoring
  - Service X went down - get it back to it's previous state asap
    - Ok, what was that previous state ?! (See: Drift)
    - What tasks need to be executed ?
- Disaster Recovery
  - Everything broke, fix it.
    - What's everything ?
    - How do we move from broken hardware to fixed hardware ?

”Newer” issues in computing infrastructures

- Complexity
  - Services depend on each other (recursively)
- Scale
  - Scale in services
  - Scale in teams
- Security
  - How can we test service hardening without breaking the services ?
  - … every time a service configuration is changed … ?

An example – the science gateway

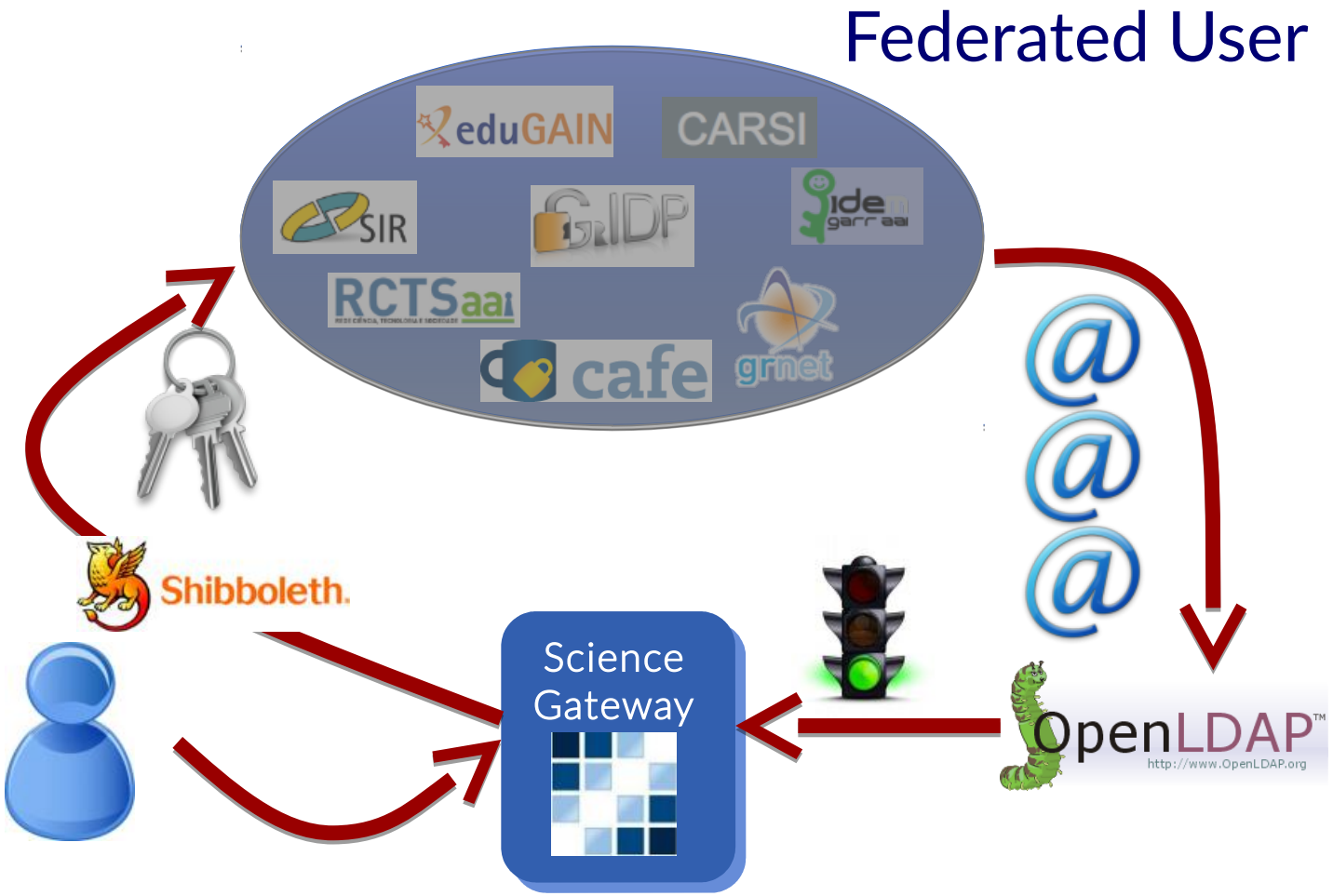


Image credit : Riccardo Rotondo (2013)

The issue in newer computing infrastructure

- Abstraction
  - What are we even running on ?
  - Does it even matter ?
- What is an ”application” ?
  - The Science Gateway is an ”application” of the cloud or grid infrastructure.

Here's the thing :

- Everything is code - Treat it like code !
  - Change control
  - Unit tests
  - Continuous Integration
- Code is developed and tested
- Services and configurations are put into Operation
- Enter DevOps

What is DevOps ?

AUG 2010

ASSESS

DevOps is a new movement seeking to achieve the business need for rapid delivery of software products while maintaining the stability of live environments. It uses two approaches: first, promoting closer collaboration between development and operations; second, applying practices shared with agile (collaboration, automation, simplicity, etc) to operations processes such as provisioning, change management, and production monitoring. It encompasses culture, processes, and tools - all supporting better communication, faster feedback and delivery, and more predictable outcomes.

https://www.thoughtworks.com/radar/techniques/devops

What is DevOps

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

What is DevOps ?

MAR 2012

ADOPT

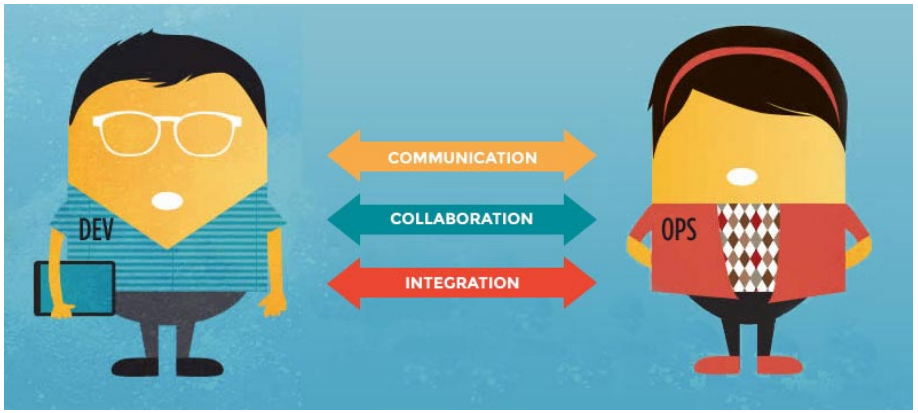
Improving the interactions and relationship between development and IT operations gives us more effective delivery and production systems that are more stable and maintainable. Creating a DevOps culture requires attention to team organization, work practices, reporting lines, and incentives - leading to joint responsibility for faster and safer delivery. We recommend adopting DevOps because we cannot see any situation where attention in this area will not have a positive benefit.

https://www.thoughtworks.com/radar/techniques/devops

How is this different to what I'm already doing ?

- Culture
- Automation
- Lean
- Measurement
- Sharing

http://newrelic.com/devops/lifecycle



http://newrelic.com/devops/what-is-devops

This is Not DevOps

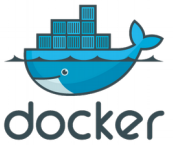
- A particular tool
  - There are lots : http://newrelic.com/devops/toolset
- A specific team or person
  - It's a culture of collaboration and transparency

Culture is learned – but can't be taught

- We'll be focussing on Ansible – *one* of the new tools in your toolbox
- Remember Maslow's Law !
  - https://en.wikipedia.org/wiki/Law_of_the_instrument
  - It's only going to be useful if used appropriately
  - It's just *one* tool

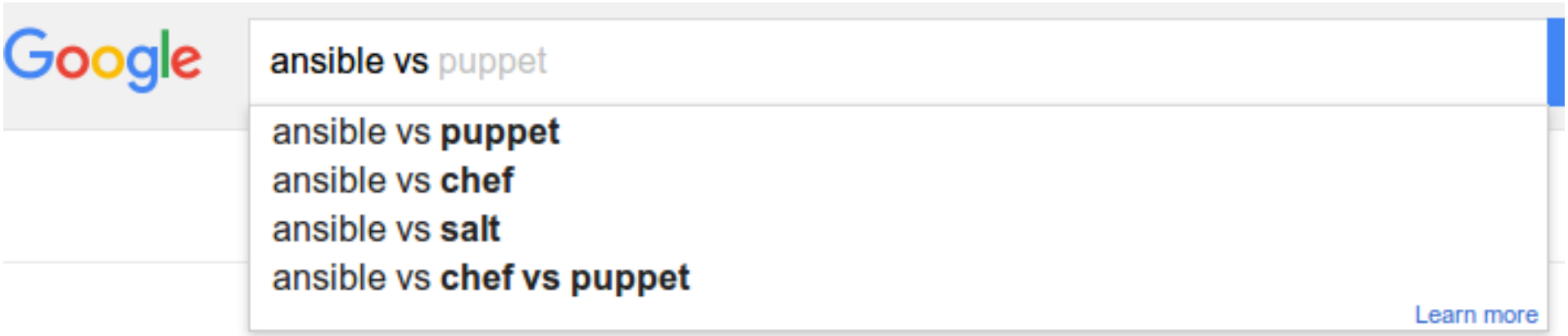
Things you'll need

- Languages : Markdown, Yaml, Jinja, JSON (Python)
- Change control : Git
- Containerisation tools (Docker)
- Test and Build services (Jenkins, Travis)
- Situational Awareness



"Kung fu has many schools, so does DevOps."

https://www.chef.io/solutions/devops/



KNOW YOUR KUNG FU

MONKEY

TIGER

VIPER

MANTIS

CRANE

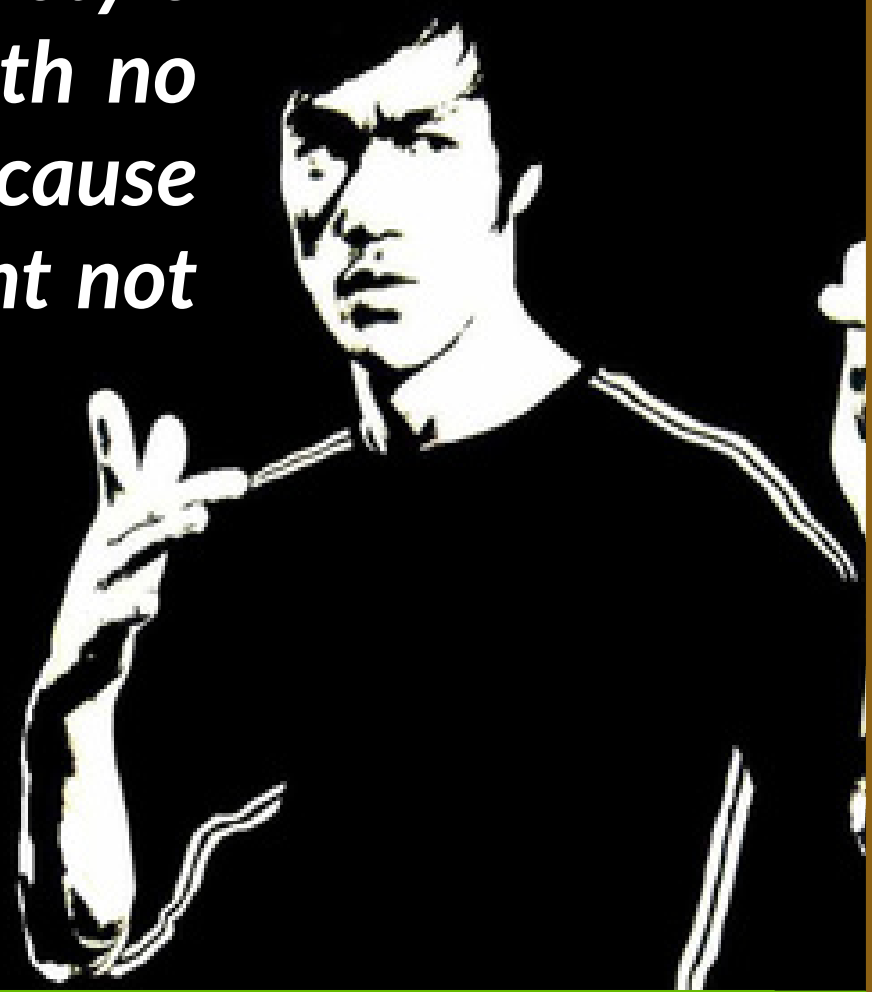
PANDA

Bruce Becker: Coordinator, Africa-Arabia ROC | bbecker@csir.co.za

”When people talk about fighting schools they say that Kung Fu, or Karate, or this other style is the best.

That is silly, and the problem becomes that the fighting style then becomes set in stone with no growth, and no adaptation, because what works well with me might not work for you.”

- Bruce Lee



Why Ansible ?

- There are many options available for developing better infrastructure
  - All of them are *much* better than none at all
  - There is no *right* tool – sometimes not even the *right tool for the job*
  - Sometimes you need to use the *right tool for you*

Ansible vs …

- The contenders
  - Ansible vs shell scripts
  - Ansible vs Puppet
  - Ansible vs Chef
  - ~~Ansible vs SaltStack~~ ← I don't know enough

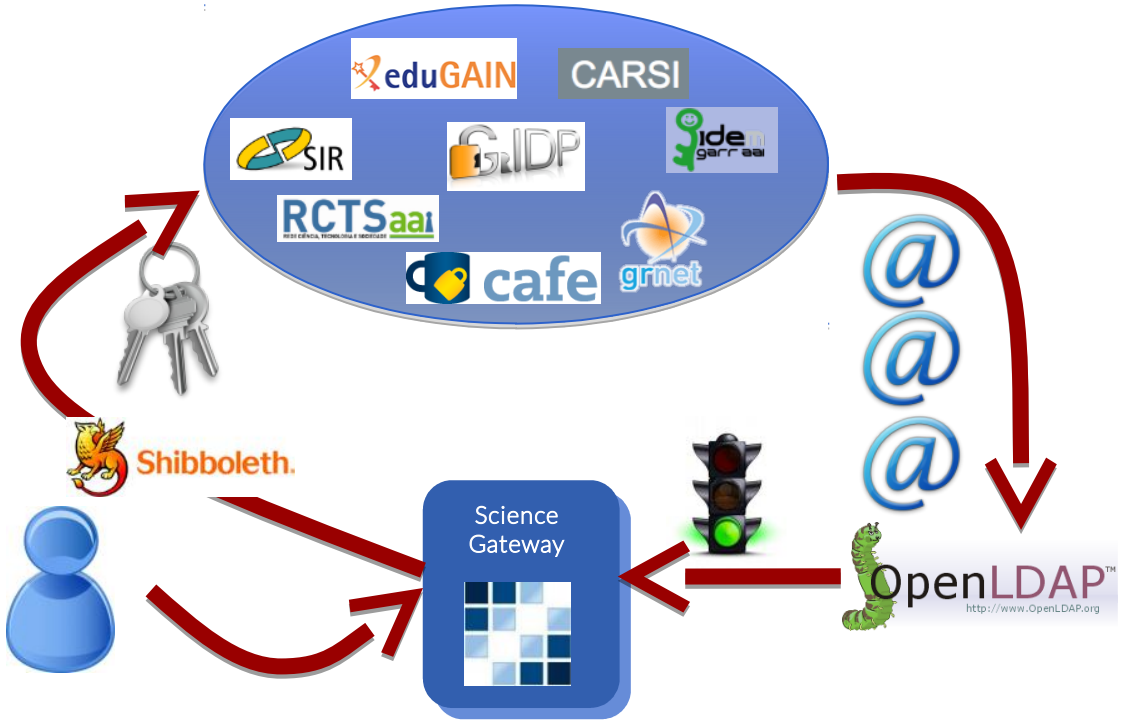
Ansible vs Shell scripts

https://valdhaus.co/writings/ansible-vs-shell-scripts/

Ansible vs Shell scripts

- Which method is most likely to end up in source control?
- Which method can be run multiple times safely with confidence?
- Which method can easily be run against multiple servers?
- Which method actually verifies (tests) your server for correctness?
- Which method can target certain servers easily (web, db, etc)?
- Which method supports easily templating your configuration files?
- Which method will grow to easily support your whole stack?

Remember the science gateway ?



You *can* do all of this with a shell script – but *should* you ?

Ansible : 1
Shell Scripts : 0

Shell script is *still better than nothing though*

The last easy decision you'll make

Remember

From here on, everything is *opinion* and *context matters*.

There is no canonical comparison between the tools

No use arguing :

YOU STILL HAVE EITHER BRUCE LEE OR CHUCK NORRIS ON YOUR SIDE !!!

Puppet

https://puppetlabs.com/

https://github.com/puppetlabs

- Puppet :
  - Old kid on the block
  - Ruby-based
  - Domain-Specific Language for internals
  - Master-client model
  - Model-Driven
    - Task and state dependencies computed by server
  - Agent-based

Pros

- Well-established support community through Puppet Labs.
- Most mature interface and runs on nearly every OS.
- Simple installation and initial setup.
- Most complete Web UI in this space.
- Strong reporting capabilities.

Cons

- For more advanced tasks, you will need to use the Ruby-based CLI, (you'll have to understand Ruby).
- Support for pure-Ruby versions (rather than those using Puppet's customized DSL) is being scaled back.
- Because of the DSL and a design that does not focus on simplicity, the Puppet code base can grow large, unwieldy, and hard to pick up for new people in your organization at higher scale.
- Model-driven approach means less control compared to code-driven approaches.

Chef

http://chef.io

https://github.com/chef

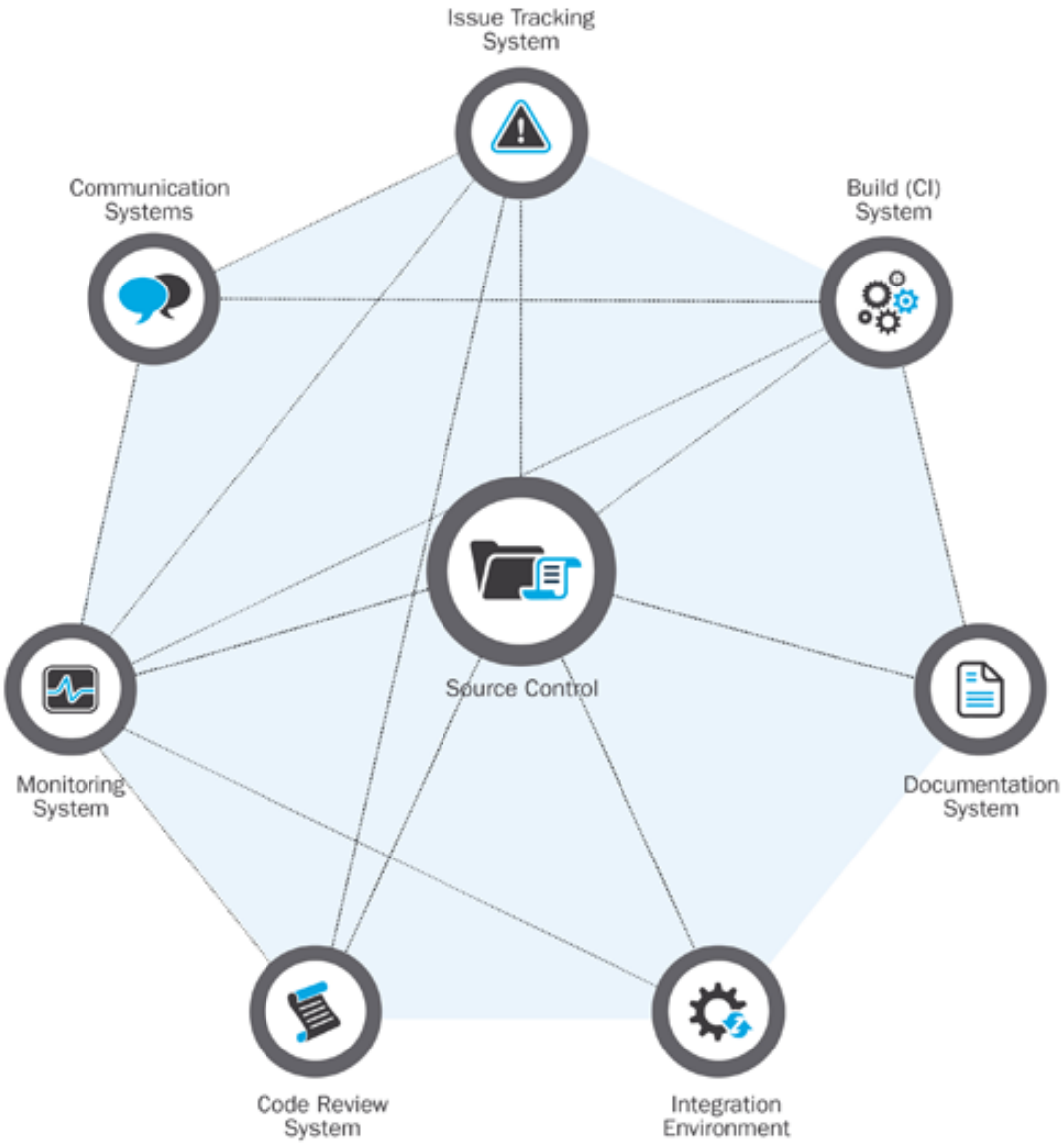
- Chef :
  - Also old kid on the block (2009)
  - Heavily dependent on Git (configuration)
  - Internals in Ruby
  - Master-client model
  - Procedural
    - Task and state dependencies computed by server
  - More ”Development” focussed than Puppet (?)

Pros Cons

- Rich collection of modules and configuration recipes.
- Code-driven approach gives you more control and flexibility over your configurations.
- Strong version control capabilities.
- 'Knife' tool (uses SSH for deploying agents from workstation) eases installation burdens.

- Steep learning curve if you're not already familiar with Ruby and procedural coding.
- Complex tool - can lead to large code bases and complicated environments.
- Doesn't support push functionality.

Take another look around



http://insights.sei.cmu.edu/sei_blog/2014/06/a-generalized-model-for-automated-devops.html

Development, Collaboration and Testing

- Any tool will build garbage if you tell it to !
- Benefits of DevOps tools : treat everything as *code* – you can apply software engineering principles to it:
  - Change control, versioning
  - Automated unit tests
  - Automated Integration
- Paradigm so widely adopted that everything is already available

What have we learned

- The same old issues and problems are now being addressed by *frameworks* instead of genius
- A shift in *culture* from top-down to flat can improve life for systems administrators, developers and users
- Sound software engineering principles can be applied to entire infrastructures
- A good knowledge of all the tools and a good situational awareness allows one to choose the right tools for the job

Next: Introduction to Ansible