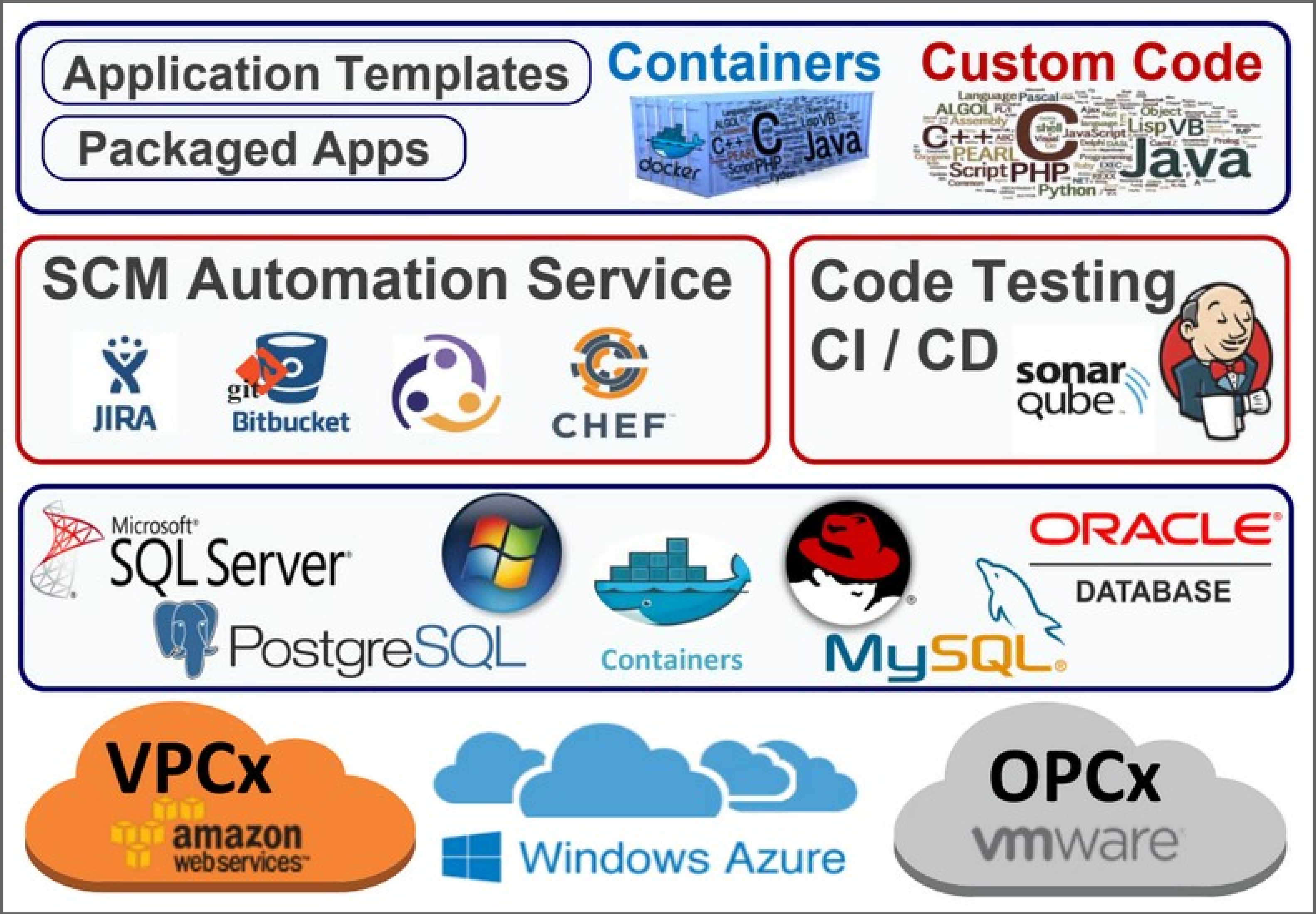
Configuration Management
through automation tools

PRESENTED BY:
Gratien D’haese
TC, HPE

Demo and Presentation available at https://github.com/gdha/cfgmgmt-intro

Source Control Driven Configuration Management and Application Deployment



Why Source Control Management?

- Pro
  - Consistency
  - Electronic record
  - Compliance
  - Policy Based Controls
  - Enterprise Application Blueprints
  - Integrated CI/CD
  - Open Source

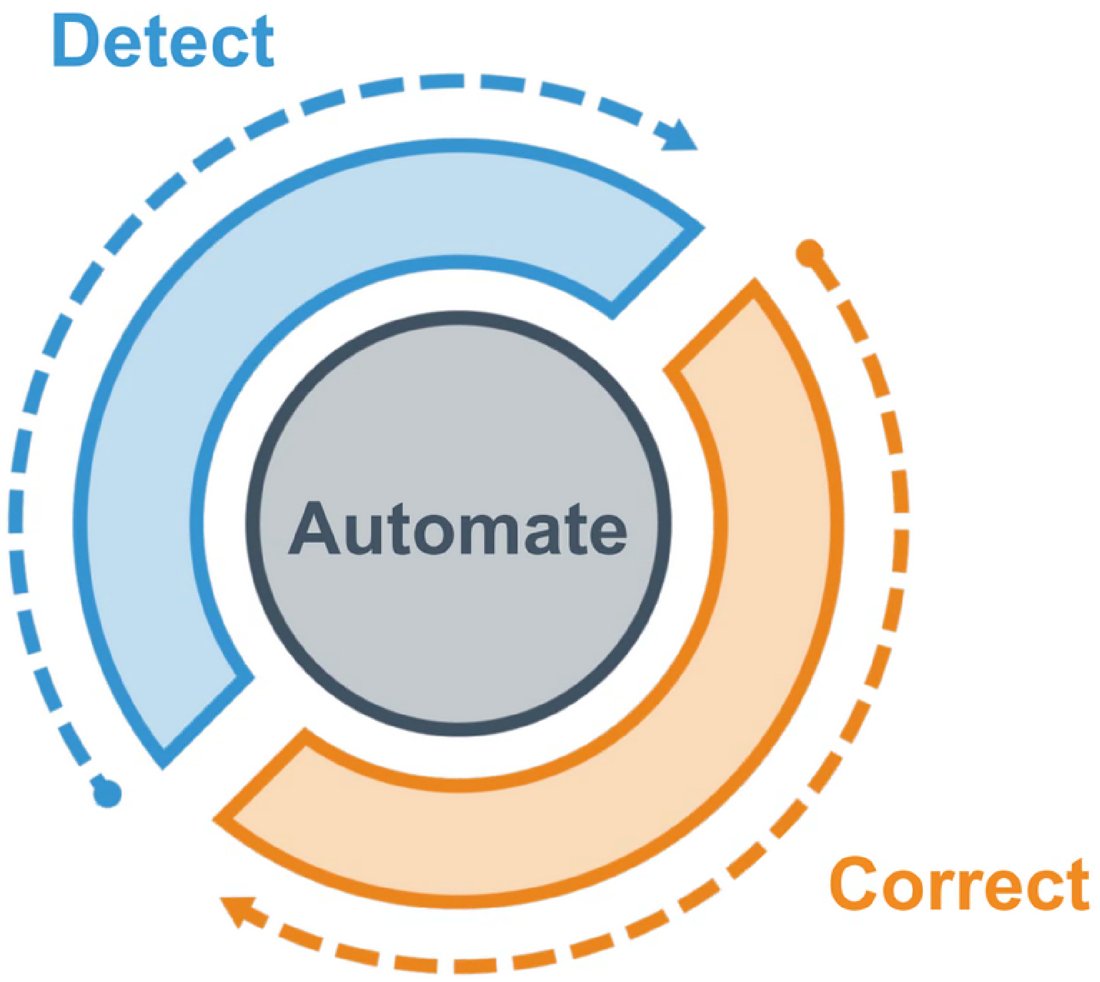
Why SCM? (2)

- Cons
  - Learning new standards
  - Learning new languages (ruby, inspec,...)
  - Learning new tools
    - Git (GitHub, GitLab, BitBucket)
    - Chef, puppet, ansible, salt
    - Jenkins
    - Docker
    - to name a few...

Advantages of configuration management

- Mass deployment
  - Avoid human errors during updates
  - Of course the source has to be correct
- Migrating from test to production
  - There is a world of difference between test <> prod
  - Use separated environments in the pipeline
- Application failure
  - Rolling back

Continuous Automation



1. **Detect**

   Gain visibility and develop baselines

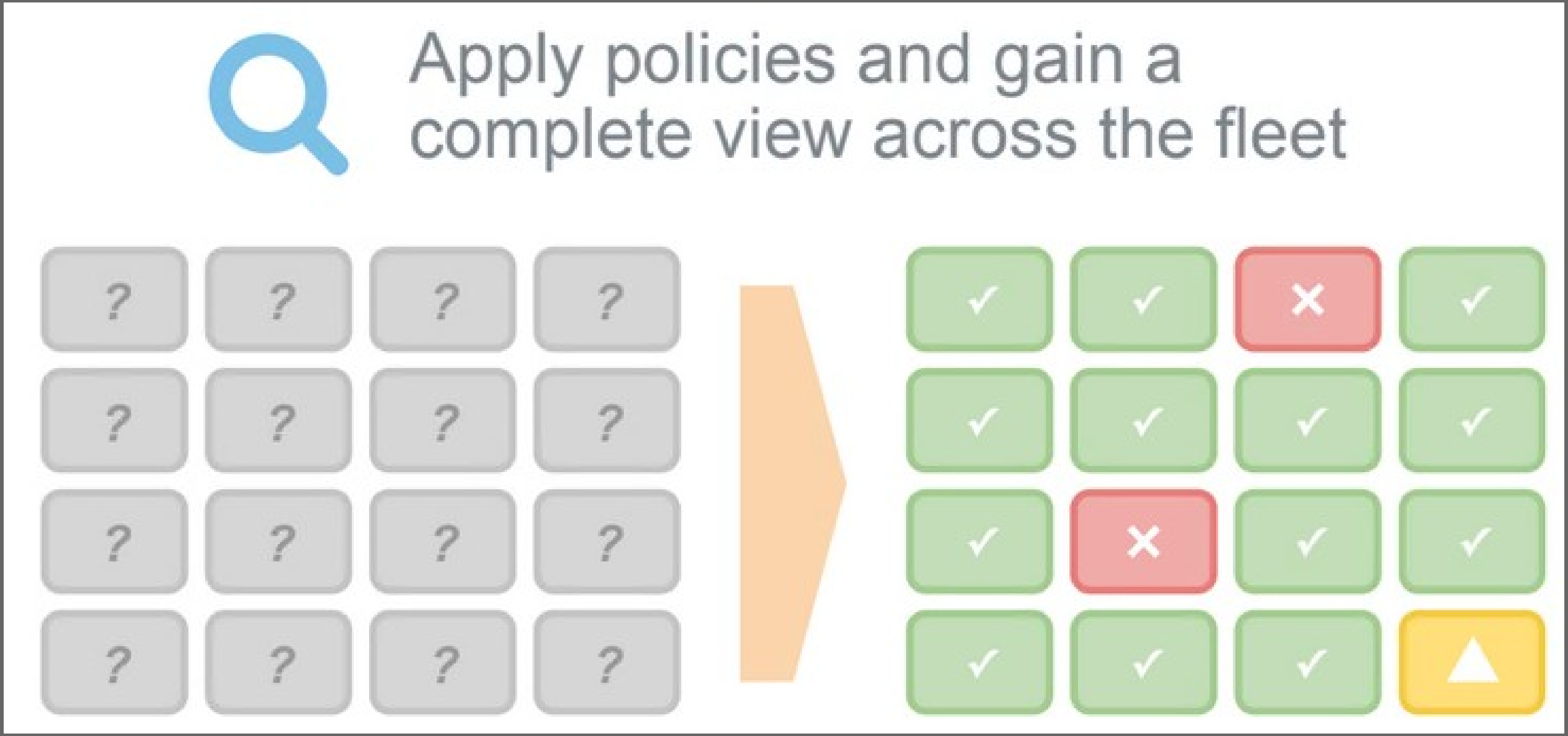
2. **Correct**

   Remediate priority issues

3. **Automate**

   Continuously detect & correct

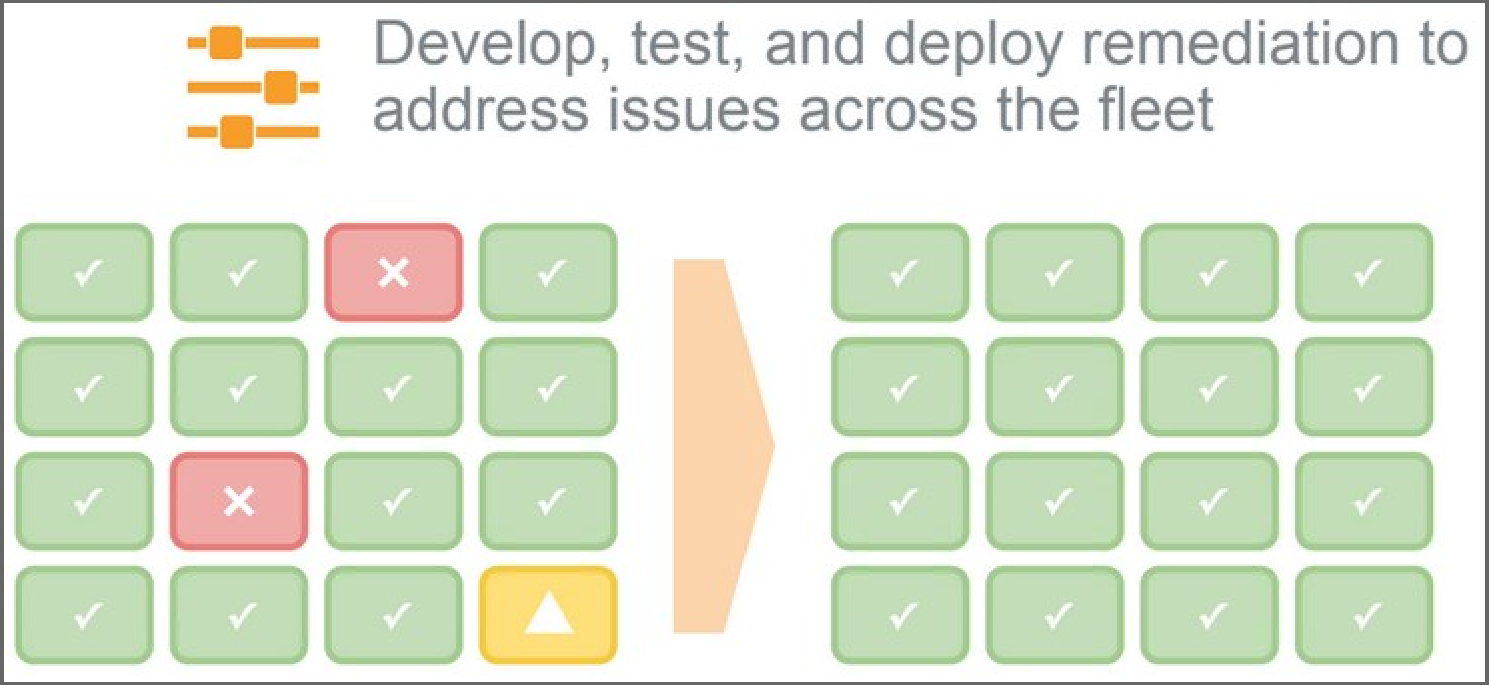
Continuous Automation: detect



Audit, risk assessment and policies

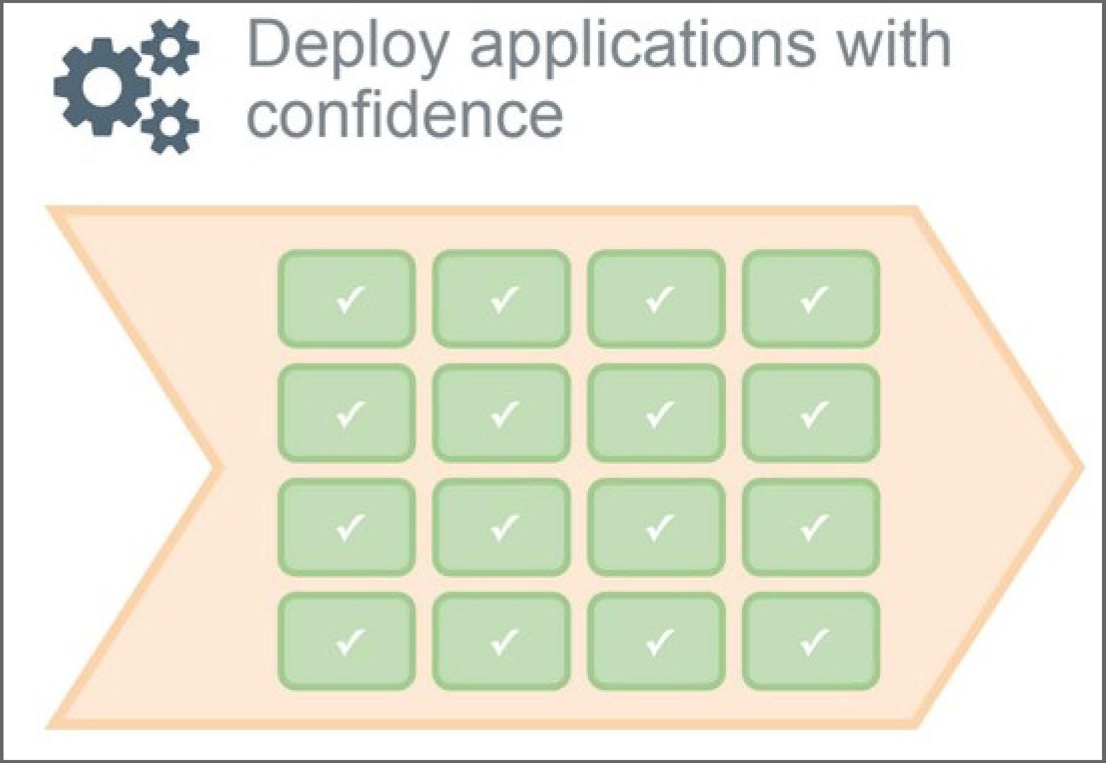
Continuous Automation: correct

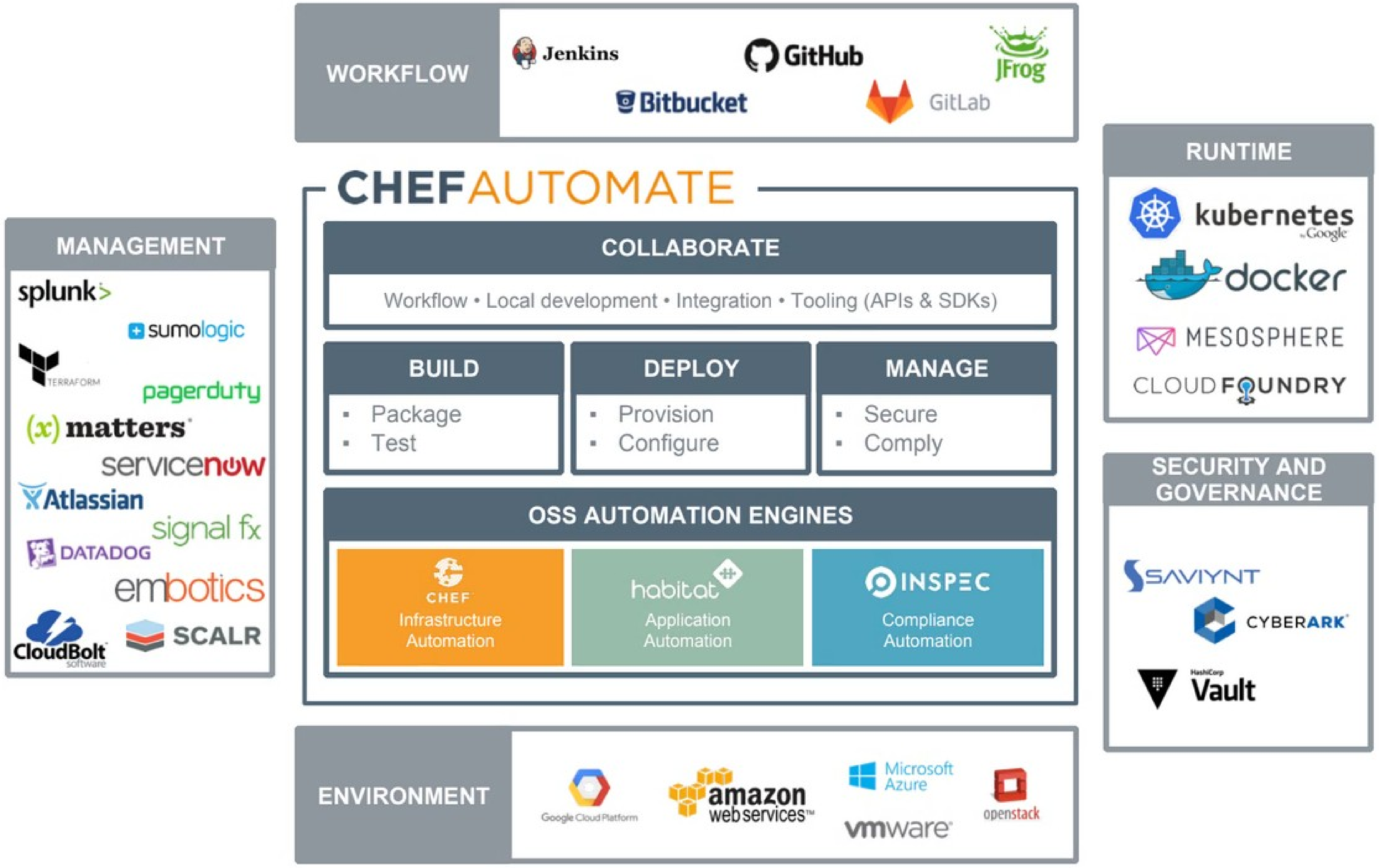
- Prioritize actions based on impact
- Close security holes
- Prove policies compliance


Develop, test, and deploy remediation to address issues across the fleet

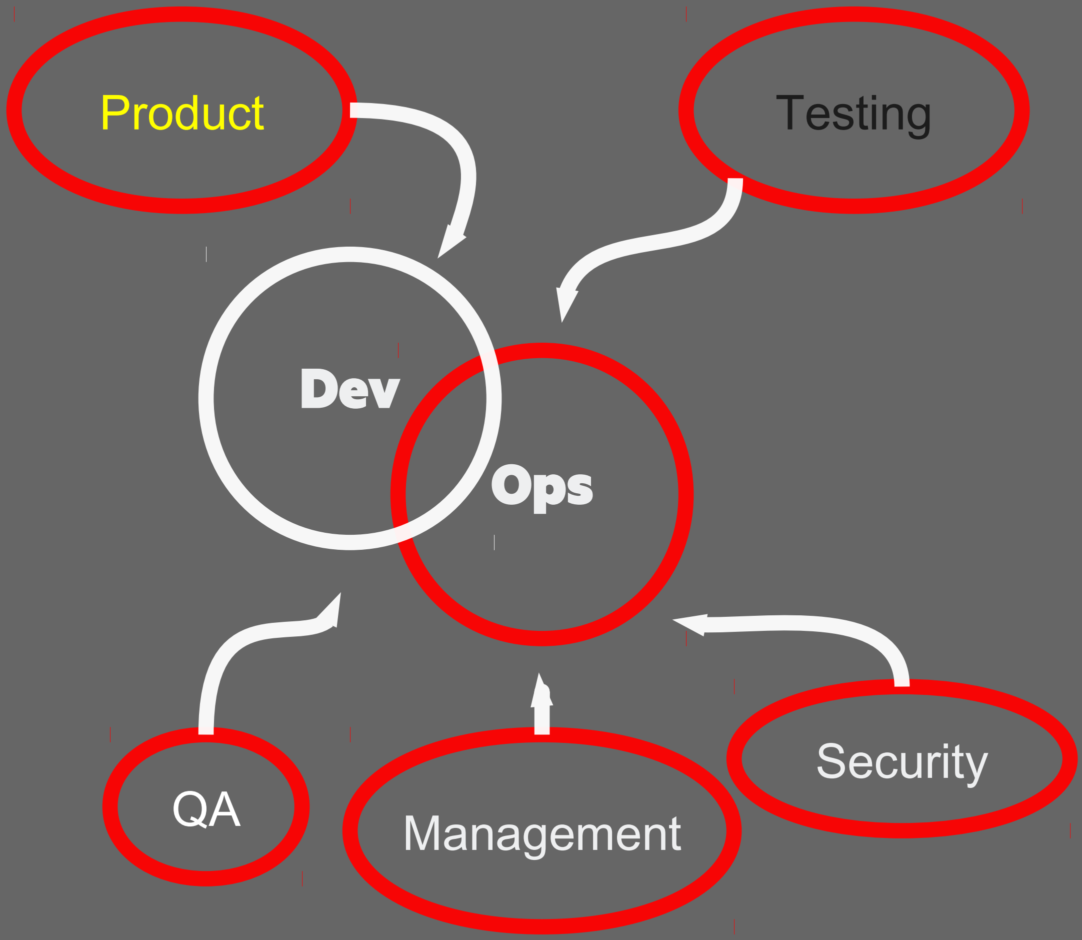
Continuous Automation: automate

- Increase speed while reducing risk
- Improve software change efficiency
- Maintain security and compliance




WORKFLOW
Jenkins
GitHub
JFrog
Bitbucket
GitLab
CHEF AUTOMATE
COLLABORATE
Workflow • Local development • Integration • Tooling (APIs & SDKs)
BUILD
Package
Test
DEPLOY
Provision
Configure
MANAGE
Secure
Comply
OSS AUTOMATION ENGINES
CHEF
Infrastructure Automation
habitat
Application Automation
INSPEC
Compliance Automation
MANAGEMENT
splunk>
sumologic
TERRAFORM
pagerduty
(x) matters
servicenow
Atlassian
signal fx
DATADOG
embotics
CloudBolt software
SCALR
RUNTIME
kubernetes by Google
docker
MESOSPHERE
CLOUD FOUNDRY
SECURITY AND GOVERNANCE
SAVIYNT
CYBERARK
HashiCorp Vault
ENVIRONMENT
Google Cloud Platform
amazon web services
Microsoft Azure
vmware
openstack

What is DevOps?



DevOps is a set of principles aimed to help teams work more efficiently and deliver better software faster

Chef vs Puppet vs Ansible vs Saltstack

Metrics	**Chef**	**Puppet**	**Ansible**	**Saltstack**
Availability	Yes	Yes	Yes	Yes
Ease of Setup	Not very easy	Not very easy	Easy	Not very easy
Management	Not very easy	Not very easy	Easy	Easy
Scalability	Highly scalable	Highly scalable	Highly scalable	Highly scalable
Language	DSL (ruby)	DSL (puppetDSL)	YAML (python)	YAML (python)
Interoperability	High	High	High	High
Price (100 nodes)	$13700	$11200 - $19900	$10000	$15000

Ansible

- Author: Michael DeHaan (released in 2012)
- URL: https://www.ansible.com/
- Acquired by Red Hat Inc in 2015
- Agentless
- All you need is OpenSSH and Python
- Playbooks use descriptive languages based on YAML and Jinja templates
- Support all UNIXes and Windows

Puppet

- Author: Luke Kanies
- URL: https://puppet.com
- Released in 2005
- Designed to manage systems (UNIX, Windows)
- Uses a Ruby DSL (domain-specific language) as declarative language
- Client – server architecture:
  - Client: puppet agent (pulls master for instructions)
  - Server: master

Chef

- Author: Adam Jacob
- URL: https://www.chef.cio
- Initial released in 2009
- Configuration management tool written in Ruby
- Supported on UNIX, Windows
- Recipes also written in Ruby
- Client – server model (or standalone chef-solo)

Salt

SALTSTACK®

- Author: Thomas S. Hatch
- URL: https://www.saltstack.com
- Released in 2011
- Python based configuration management
- Client – server based
  - Client: minion
  - Communication through SSH
  - YAML and Python

Using ansible

- Use a non-privileged user
- Use sudo rule to grant root permission on clients
- Exchange SSH keys for that non-privileged user
- Work from one master “client” (need python and ansible installed)
- Create /etc/ansible/host inventory file
- Ready to go

SSH Key exchanging for ansible

- Use a ansible playbook to:
  - Create /home/ansible/.ssh directory
  - Exchange SSH public key to clients
  - ansible-playbook playbooks/ansible-user-ssh.yml
    Permission denied (publickey,password).\r\n",
    "unreachable": true
  - Use the ‘-k’ option (to ask password)

I hear you thinking!

- How do I get the sudo rule installed if we do not have root access to all clients?

```
# cat /etc/sudoers.d/ansible
 Defaults        !requiretty
 ansible         ALL=(ALL) NOPASSWD:ALL
```

- Install this sudoers file through provisioning

Some basics of ansible

- Ansible uses an inventory file (INI style)
- Example of /etc/ansible/hosts (default file)

```
[newuat]
ITSGBHHLSP01527

[newprd]


[sbx]
ITSGBHHLSP00416
ITSGBHHLSP00417
ITSGBHHLSP00418
ITSGBHHLSP00419
```

Ansible command line usage

- Check simple stuff via command line
- For example:

```
$ ansible sbx -m shell -b -a "rpm -q rear" -o
ITSGBHHLSP00417 | CHANGED | rc=0 | (stdout) rear-2.00-6.el7.x86_64
ITSGBHHLSP00448 | CHANGED | rc=0 | (stdout) rear-2.00-6.el7.x86_64
ITSGBHHLSP00669 | CHANGED | rc=0 | (stdout) rear-2.00-6.el7.x86_64
ITSGBHHLSP00420 | CHANGED | rc=0 | (stdout) rear-2.00-6.el7.x86_64

$ ansible localhost -m setup
localhost | SUCCESS => {
    "ansible_facts": {
        "ansible_all_ipv4_addresses": [
            "10.180.5.67",
            ....
```

Ansible modules (-m)

- Command : default
- Setup : to view the system details
- Copy : copy files
- Fetch : retrieve files
- Service : stop/start system services
- Shell : run more complex shell commands (e.g. when using the pipeline)

ansible-galaxy

- Create a new project for ansible
- ansible-galaxy init ubuntu18

```
ubuntu18/
        |  README.md
        |- defaults
        |- files
        |- handlers
        |- meta
        |- tasks
        |- templates
        |- tests
        |- vars
```

Ansible: playbooks

```
$ cat playbooks/ansible-user-ssh.yml
 ---
- name: Prepare the ansible user .ssh sub-directory
  hosts: clients
  gather_facts: false
  vars:
    username: ansible
    groupname: ansible
  tasks:

    # Create for user ansible the .ssh directory
    - name: Create /home/{{ username }}/.ssh (if required)
      file: path=/home/{{ username }}/.ssh state=directory
mode=0700 owner={{ username }} group={{ groupname }}
recurse=yes
    - name: Copy public ssh key to /home/
{{ username }}/.ssh
      copy: src=/home/{{ username }}/.ssh/id_rsa.pub
dest=/home/{{ username }}/.ssh/authorized_keys mode=0744
owner={{ username }} group={{ groupname }}
```

The ansible magic can begin

- Once “ansible” non-privileged user exists on all clients
- And, secure shell password-less communication is possible
- You can now become a “power user”!
- The “-b” (become root) option grants you:

  ```
  ansible clients -i hosts -m shell -b -a
  "journalctl | tail -5"
  ```

Motd ansible playbook

- Remove the /etc/motd on clients

```
$ ansible clients -b -m file -a "path=/etc/motd
state=absent"
```

- Create a new /etc/motd

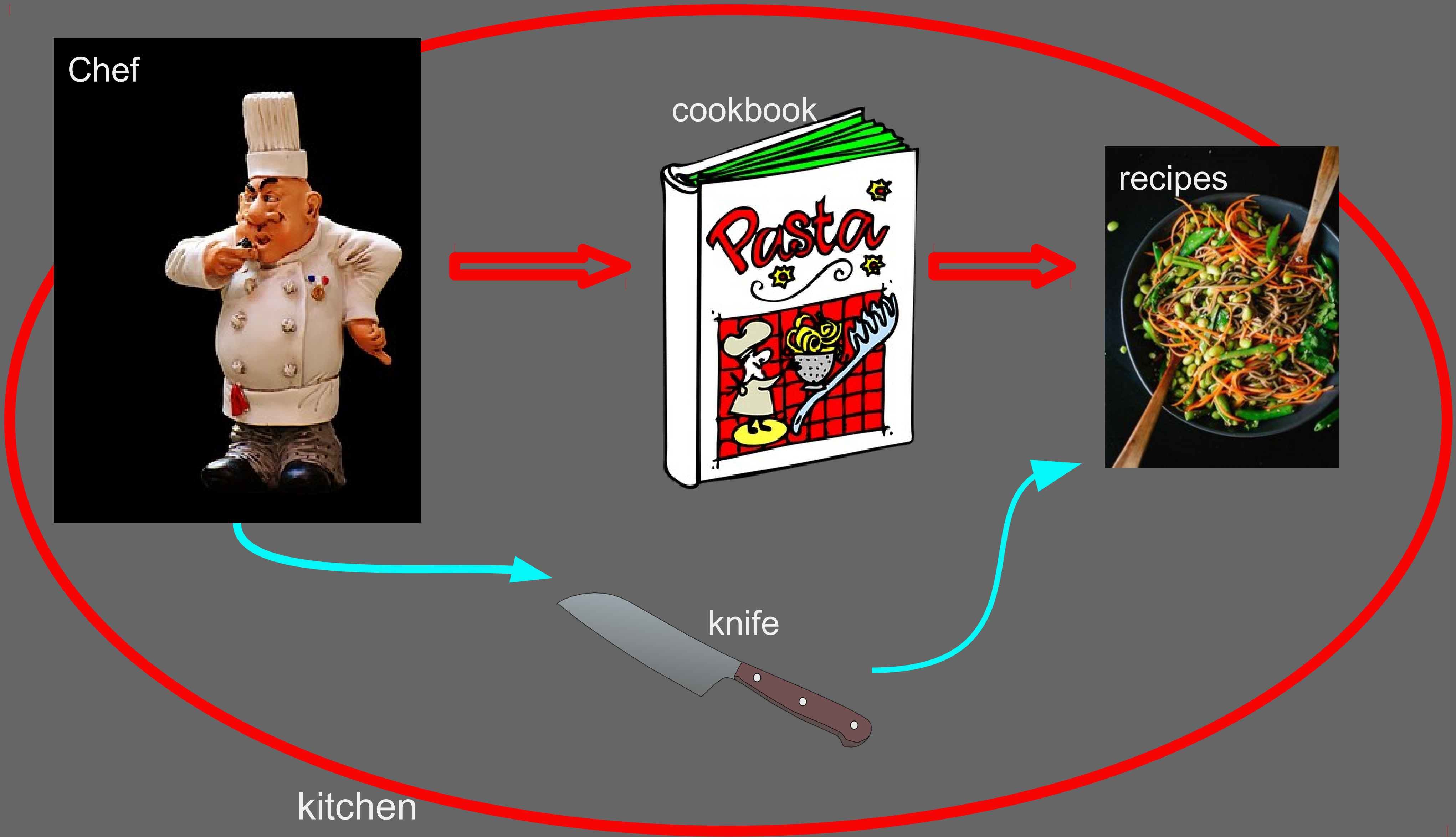
```
$ ansible-playbook playbooks/create_motd.yml
$ cat /etc/motd
Welcome to client1
```

- Rules should be idempotent
  - Means run as many times = result is the same
  - See our demo

Chef setup

- Chef Server part
- Chef clients (required to run recipes)
- ChefDK (Development Kit to write cookbooks)
- Chef Automate (GUI for compliance, pipeline view, nodes inspection) – talks to Chef Server
- Kitchen Test (to test your own cookbooks and is part of the ChefDK)

Chef terminology



Chef Concepts

- Cookbooks are collections of Recipes and associated Attributes defining a scenario
- Cookbooks are the fundamental unit of configuration and policy distribution in Chef
- Recipes are collections of Resources (written in Ruby)
- Attributes provide specific details of a Node (such as software installation and configuration)

Chef Concepts (continued)

- A Role is used to define patterns and processes that exist across Nodes
- A Run-list is an ordered list of Recipes or Roles that run in exact order
- A Data-bag is a global variable and could include sensitive data (e.g. encrypted password)
- A Node belongs to a specific Environment which controls which versions of cookbooks are used
  - A combination of Run-list and Role

Chef Resources

- It describes the desired state of an element of your infrastructure and the steps required to go to that state

```
file '/tmp/hello.txt' do
  content 'Hello World!'
end
```

Type
Name
Action

```
$ chef-client -z hello.rb
Converging 1 resources
Recipe: @recipe_files::./hello.rb
  * file[/tmp/hello.txt] action create
    - create new file /tmp/hello.txt
    - update content in file /tmp/hello.txt from none to 7f83b1
    --- /tmp/hello.txt  2019-04-03 17:52:32.394317658 +0200
    +++ /tmp/.chef-hello20190403-12410-a1rhyx.txt  2019-04-03
    @@ -1 +1,2 @@
    +Hello World!
```

Example Chef Resources

- Package

```
package 'tree' do
  action :install
end
```

- Service

```
service 'ntp' do
  action [ :enable, :start ]
end
```

- File

```
file '/tmp/hello.txt' do
  action :delete
end
```

- Mount

```
mount '/mnt/local' do
device '/dev/sdb1'
fstype 'ext3'
end
```

See also https://docs.chef.io/resource.html

Chef recipes

- Chef and recipes are written in ruby
- Knowledge of ruby is not a requirement
- A recipe is a collection of resources
- Each resource is executed in the order they are listed

Chef Cookbook

- A cookbook is a set of recipes
- Common components in a cookbook
  - README
  - Metadata
  - Recipes
  - Testing directories (spec + test)
- Install ChefDK to start writing cookbooks
- `$ chef generate cookbook NAME`

Cookbook NAME

```
NAME
|-- Berksfile
|-- CHANGELOG.md
|-- LICENSE
|-- README.md
|-- chefignore
|-- metadata.rb
|-- recipes
|   `-- default.rb
|-- spec
|   |-- spec_helper.rb
|   `-- unit
|       `-- recipes
|           `-- default_spec.rb
`-- test
    `-- integration
        `-- default
            `-- default_test.rb
```

Kitchen test (cookbook nginx_test)

- Kitchen provides a test harness to execute infrastructure code on one or more platforms in isolation (by default using vagrant)
- Kitchen create
- Kitchen converge
- Kitchen verify
- Kitchen destroy

DevOps in a nutshell

- What do you need to know/learn?
  - Basic OS knowledge (Linux, Windows)
  - Git
  - Python (optional)
  - Ansible (basics)
  - Automation tools of choice (or not) – Chef, Ansible, Puppet or Salt
  - Jenkins (usage)
  - Kitchen test (works with Chef, Ansible, Puppet, Salt)

Questions?

CONTACT:
gratien.dhaese@hpe.com