



Automating DPK installs with GitLab and Ansible

Paul Houghton — Senior Database Administrator, Student Systems

SESSION 5514

22 October 2019



UNIVERSITY OF
CAMBRIDGE
Information Services

www.uis.cam.ac.uk

Presenter



- Paul Houghton
- Senior DBA
- Joined the University of Cambridge in 2009
- PeopleSoft Applications DBA
- Blog: <https://curiousdba.netlify.com>

Facts 'n' figures

Founded **1209**
810 years old

31 autonomous Colleges

100+ Departments,
Faculties, Schools & Non-
School Institutions

26,000+ students
11,000+ employees

109 Nobel Prize winners

The University houses over **15**
million books in over **100**
libraries

PeopleSoft at Cambridge

Implementation began

2001

Campus 9.0 upgrade

2008

De-customisation project

2017

Campus 9.2 upgrade

2017

Improvement Programme

2017+

Campus 9.2 PUM 12, PT 8.57.04

Now

Overview

1

8.55 upgrade
Challenges and
solutions: Ansible!

3

Making Things Even
Better: Gitlab!

2

The results What
worked well, and
what didn't.

4

More Results!

Do you use additional automation?



8.55 Upgrade



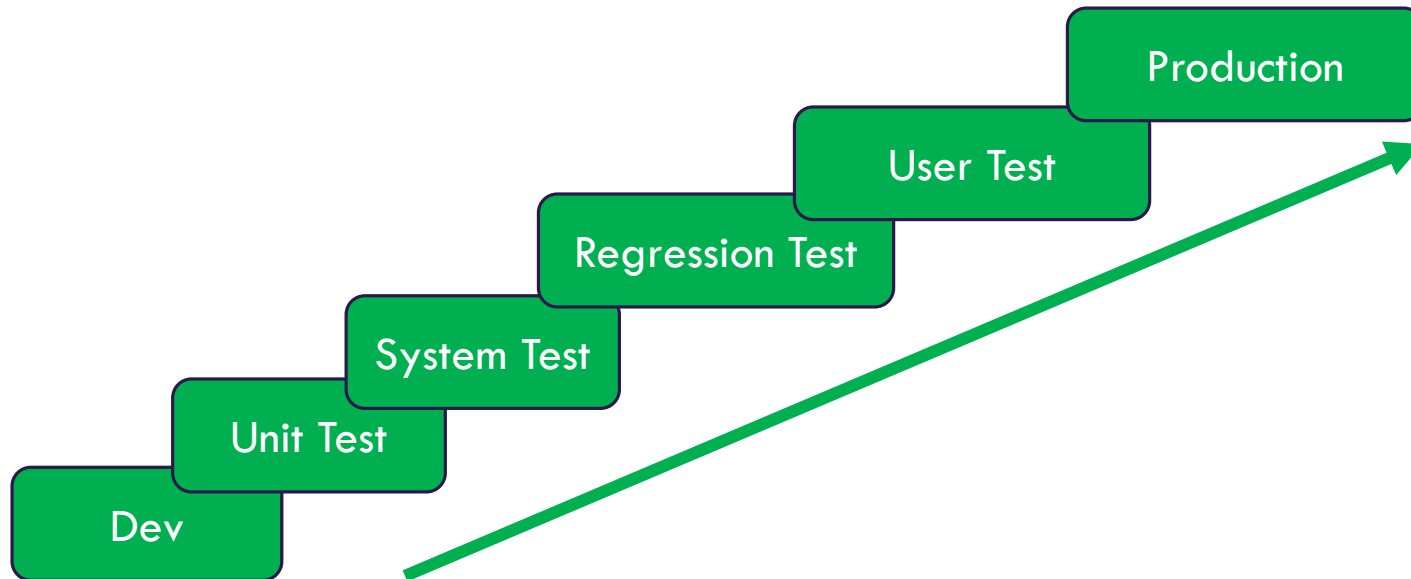
8.55 Upgrade – The Challenges

- A big change:
 - From shared hardware to about 100 VMs.
 - From installing separate software to one controlling installer: Puppet Deployment Packages (DPKs).
- Some things stay the same:
 - Preserve custom configuration where necessary.
 - Security patches still have to be applied.
 - Some things don't quite work out of the box, on our site.

Configuration Management

A quick way to mess up all your systems with one easy command?

Configuration needs to be controlled in the same way as code.
Take advantage of testing!



Why Ansible? Why not puppet?

Puppet is the obvious choice, because Oracle use it in the DPK.

- Why Ansible?

- Designed for this purpose (Like puppet).
- Agentless (Unlike puppet – easier bootstrapping).
- Colleagues are already using it.
- Can control all VMs from a central management server.



A N S I B L E

- Why not Puppet?

- Oracle might start using something else.
- It can be wrapped in another product anyway.



puppet

Why Ansible? Why not puppet?

Puppet is the obvious choice, because Oracle use it in the DPK.

- Why Ansible?

- Designed for this purpose (Like puppet).
- Agentless (Unlike puppet – easier bootstrapping).
- Colleagues are already using it.
- Can control all VMs from a central management server.



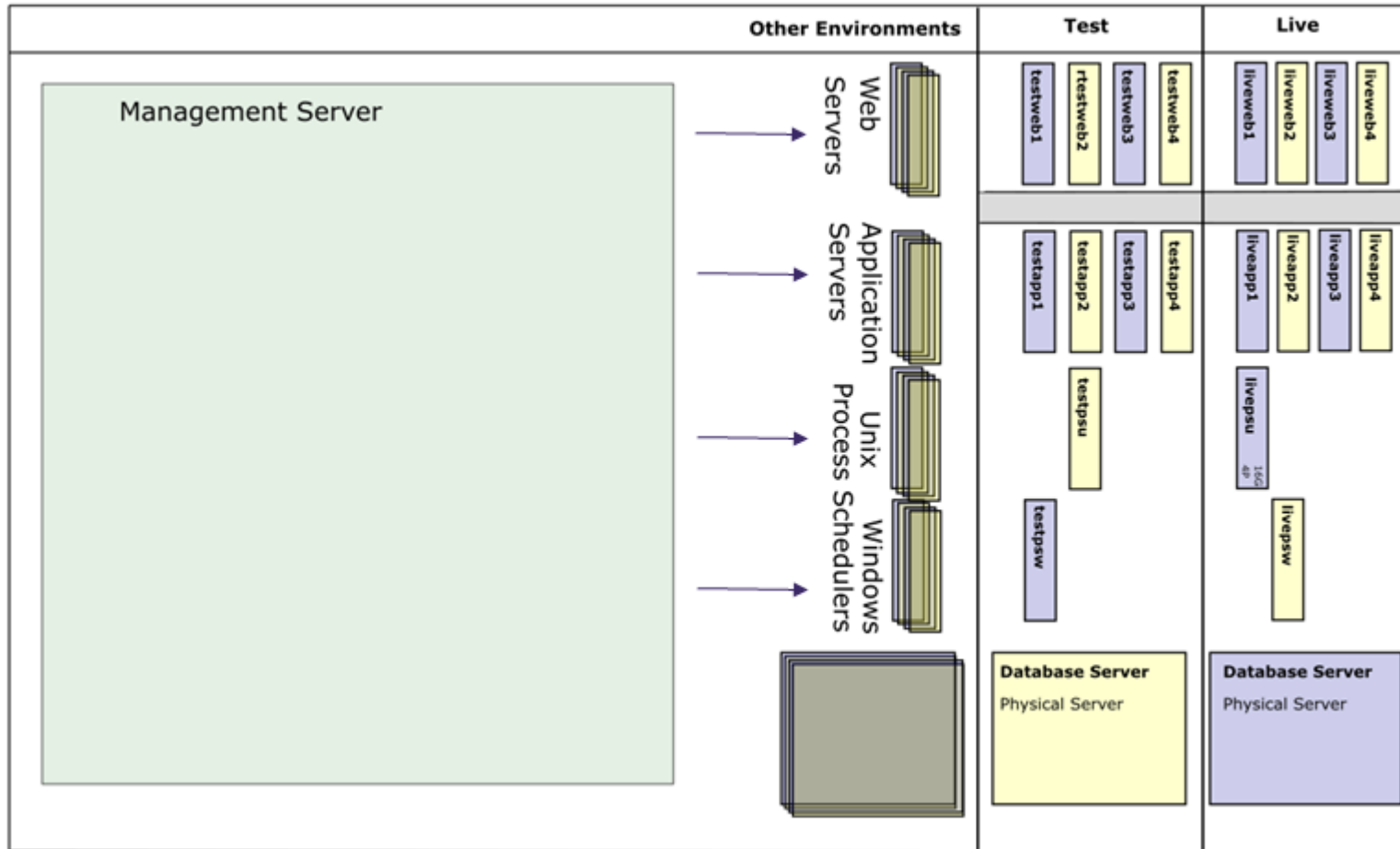
A N S I B L E

- Why not Puppet?

- Oracle might start using something else.
- It can be wrapped in another product anyway.



Ansible – Chosen Structure



Ansible – Chosen Structure

```

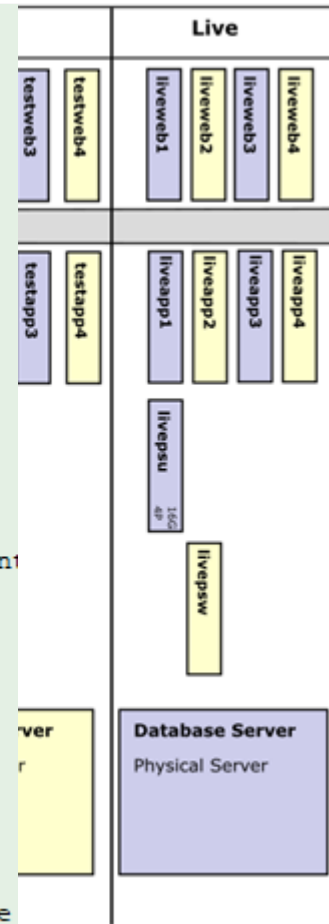
inventories/
  production      # inventory file for production servers
  staging         # inventory file for staging environment

group_vars/
  webservers.yml  # here we assign variables to particular groups

library/          # if any custom modules, put them here (optional)

camsis-vms.yml    # Delete and recreate VMs
camsis-java-cpu.yml # Apply Java critical patch
camsis-web-cpu.yml # Apply Weblogic critical patch
camsis-start.yml  # Start an environment
camsis-stop.yml   # Stop an environment

common/           # this hierarchy represents a "role"
  tasks/          #
    main.yml      # <-- tasks file can include smaller files if warranted
  handlers/       #
    main.yml      # <-- handlers file
  templates/      # <-- files for use with the template resource
    ntp.conf.j2   # <----- templates end in .j2
  files/          #
    bar.txt       # <-- files for use with the copy resource
    foo.sh        # <-- script files for use with the script resource
  vars/           #
    main.yml      # <-- variables associated with this role
  defaults/       #
    main.yml      # <-- default lower priority variables for this role
  
```



Secrets and how to deal with them

- What is a secret?
 - Password or private key.
 - Things we want to keep secret!
- Don't want them in version control.
- Don't want them in logs.
 - Ansible has a `no_log` parameter which is useful.
- Started by keeping them in the inventory. It became unwieldy.
- Ansible Vault or regpg?
 - My colleague wrote regpg, so I used that!

The Results



Benefits

- Time saved.
 - Apps DBA time to build an environment from 1 day down to 1 hour waiting for the script to run.
 - Apps DBA time applying critical patches.
 - Tester time finding mistakes in the build (3 days per environment per cycle).
 - Developer time diagnosing mistakes in the build.
 - User time wasted when build problems reach production.
- Repeatability – the surprise superpower!
 - No problems on a single server.
 - We can confidently fire up a new VM.
 - We can confidently create a new environment.
 - Bugs are only fixed once forever.

What Worked

- All the state is in Ansible.
- No state is kept on the VM.
- No upgrades – only installs.
- Inventory file per environment.
- Version control of the source – but being careful not to check in secrets.
- Starting small and incrementally improving.
- Learning from others.
- Learning from mistakes.

Mistakes I learned from

- Mistakes are good. We can learn from them. Lean.
- Chasing the project. Need to sort out automation first.
- Not having a strategy for secrets.
 - Keep the secrets out of version control.
 - Don't let secrets end up in log files either.
- Not having a strategy for branching: dozens of feature branches!
 - Git Flow or similar.
- Waiting for somebody else.

I rediscovered the power of mass production



Bell Aircraft Corporation: 1944.

How can I make things even better

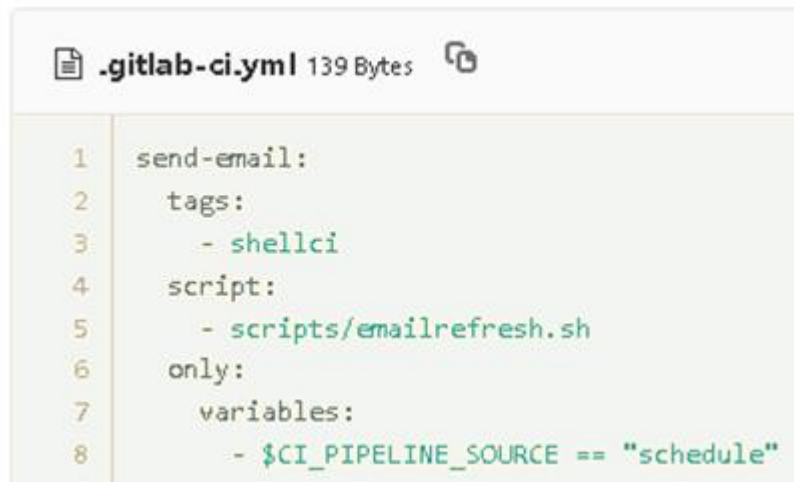
- I was surprised at how beneficial the automated VM build was.
- I was keen to see what other benefits could be gained.
- The University created a Gitlab instance for itself.
- Gitlab runner – can schedule things to happen.

What GitLab brings

- GitLab Runner.
- Installed on a VM previously used by Jenkins.
- Job details integrated with the version control.
- Understands environments.
- GUI web interface.

Use case 1 – Refresh email

- Process to automate:
 - Check a wiki page every day.
 - If there is a refresh tomorrow email the team.
 - If nobody objects run the refresh.
- This is tedious repetitive work. Computers are good at it, humans are not!



```
1  send-email:
2    tags:
3      - shellci
4    script:
5      - scripts/emailrefresh.sh
6    only:
7      variables:
8        - $CI_PIPELINE_SOURCE == "schedule"
```

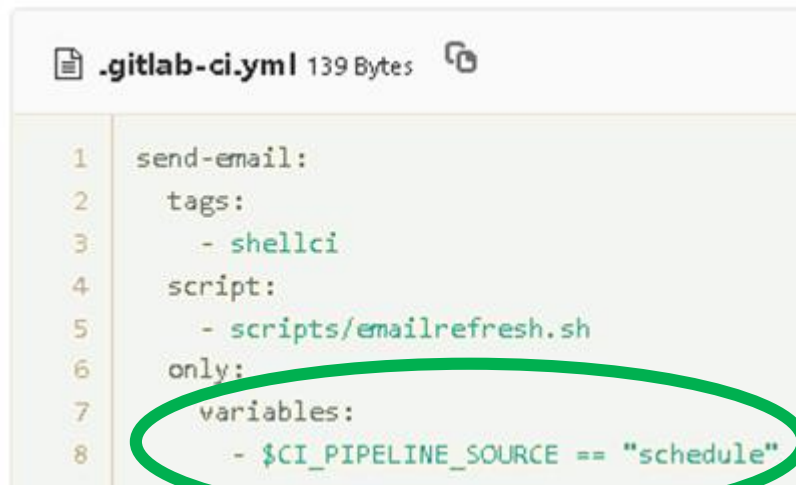
Upcoming Refresh Dates

Edit the table. Ensure it stays in markdown format. Dates must be in YYYY-MM-DD format.

Date	Environment	Comments
2019-08-22	CSPRTST	Project Testing
2019-09-09	CSPREV	Release 6 2019
2019-09-11	CSDAT	Release 6 2019

Use case 1 – Refresh email

- Process to automate:
 - Check a wiki page every day.
 - If there is a refresh tomorrow email the team.
 - If nobody objects run the refresh.
- This is tedious repetitive work. Computers are good at it, humans are not!



```
1  send-email:
2    tags:
3      - shellci
4    script:
5      - scripts/emailrefresh.sh
6    only:
7      variables:
8        - $CI_PIPELINE_SOURCE == "schedule"
```

Upcoming Refresh Dates

Edit the table. Ensure it stays in markdown format. Dates must be in YYYY-MM-DD format.

Date	Environment	Comments
2019-08-22	CSPRTST	Project Testing
2019-09-09	CSPREV	Release 6 2019
2019-09-11	CSDAT	Release 6 2019

Use case 2 – Management Server

- We used to have to remember to do a git pull whenever we made a change.
- We want to keep inventories in different repo to the playbooks.
- We want several repositories with playbooks.
- Solution - Gitlab runner scripts assemble the different repositories into the correct locations on the management server once they are committed.
- This example is from the inventory files.



The screenshot shows a code editor window with a file named `.gitlab-ci.yml` (74 Bytes). The file contains a YAML configuration for a job named `deploy-inventories`. The configuration specifies that the job should run on the `shellci` runner and execute the `scripts/copy.sh` script.

```
1  deploy-inventories:
2    tags:
3      - shellci
4    script:
5      - scripts/copy.sh
```

Use case 3 – VM Build Ansible playbooks.

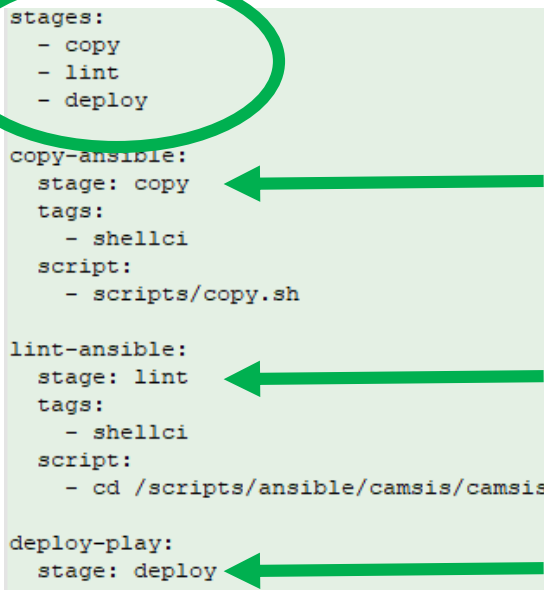
- Pipeline Stages

```
stages:
- copy
- lint
- deploy

copy-ansible:
  stage: copy
  tags:
  - shellci
  script:
  - scripts/copy.sh

lint-ansible:
  stage: lint
  tags:
  - shellci
  script:
  - cd /scripts/ansible/camsis/camsis-ansible/$CI_COMMIT_REF_SLUG && ansible-lint --force-color camsis-vms.yml || true

deploy-play:
  stage: deploy
  tags:
```



Use case 3 – VM Build Ansible playbooks.

```
deploy-play:
  stage: deploy
  tags:
    - shellci
  script:
    - cd /scripts/ansible/camsis/camsis-ansible/$CI_COMMIT_REF_SLUG && ansible-playbook
  environment:
    name: play
    url: https://camsis.cam.ac.uk/psp/adminplay
  when: manual
  only:
    - Development

deploy-ops:
  stage: deploy
  tags:
    - shellci
  script:
    - cd /scripts/ansible/camsis/camsis-ansible/$CI_COMMIT_REF_SLUG && ansible-playbook
  environment:
    name: ops
    url: https://camsis.cam.ac.uk/psp/adminops
  when: manual
  only:
    - Development
```


Use case 3 – VM Build Ansible playbooks.

```
deploy-play:
  stage: deploy
  tags:
    - shellci
  script:
    - cd /scripts/ansible/camsis/camsis-ansible/$CI_COMMIT_REF_SLUG && ansible-playbook
  environment:
    name: play
    url: https://camsis.cam.ac.uk/psp/adminplay
  when: manual
  only:
    - Development

deploy-ops:
  stage: deploy
  tags:
    - shellci
  script:
    - cd /scripts/ansible/camsis/camsis-ansible/$CI_COMMIT_REF_SLUG && ansible-playbook
  environment:
    name: ops
    url: https://camsis.cam.ac.uk/psp/adminops
  when: manual
  only:
    - Development
```

Use case 3 – View from GitLab

Run Pipeline Clear Runner Caches CI Lint

All 47 Pending 0 Running 0 Finished 47 Branches Tags

Status	Pipeline	Triggerer	Commit	Stages	
passed	#7493 latest		Development 121edbd7 Add sshkey.yr		01:00:39 1 day ago
passed	#7488		Development 338ef3f7 Correct installer name	deploy-ops deploy-play	00:56:19 2 weeks ago

Use case 3 – Gitlab has the logs!

```
TASK [Gathering Facts]
*****

ok: [csplaywin]

TASK [psw-start : Make sure services are running]
*****

ok: [csplaywin] => (item=ORACLE ProcMGR V12.2.0.0_V$2015)
changed: [csplaywin] => (item=TUXEDO 12.2.2.0.0_V$2015 Listener on
Port 3050)
changed: [csplaywin] =>
(item=PeopleSoft_C__Users_admin_psft_pt_8.57)

PLAY RECAP
*****

csplayapp1      : ok=5    changed=2    unreachable=0
failed=0        skipped=0    rescued=0    ignored=0
csplaypsu       : ok=4    changed=1    unreachable=0
failed=0        skipped=1    rescued=0    ignored=0
csplayweb1      : ok=3    changed=1    unreachable=0
failed=0        skipped=0    rescued=0    ignored=0
csplaywin       : ok=2    changed=1    unreachable=0
failed=0        skipped=0    rescued=0    ignored=0

Job succeeded
```

deploy-play

Retry

Duration: 60 minutes 3 seconds

Timeout: 2h (from project) ?

Runner: CamSIS Shared on dba-
ci1.internal.admin.cam.ac.uk] (#126)

Tags: shellci

Commit 121edbd7

Add sshkey.yml

✓ Pipeline #7493 for Development

deploy

✓ deploy-play - passed

→ ✓ deploy-play

✓ deploy-play

What worked

I was pleasantly surprised!

- Communicating with colleagues about what I am trying to achieve.
 - Colleagues in my team have embraced the new way of working.
 - Colleagues in other teams have changed their working practices to support what I am doing.
- Finding pain points for others in the team and addressing them:
 - Lack of notification of changes.
 - Downtime around refreshes.
- Allowing myself to make mistakes.

What doesn't work

Repository Layout

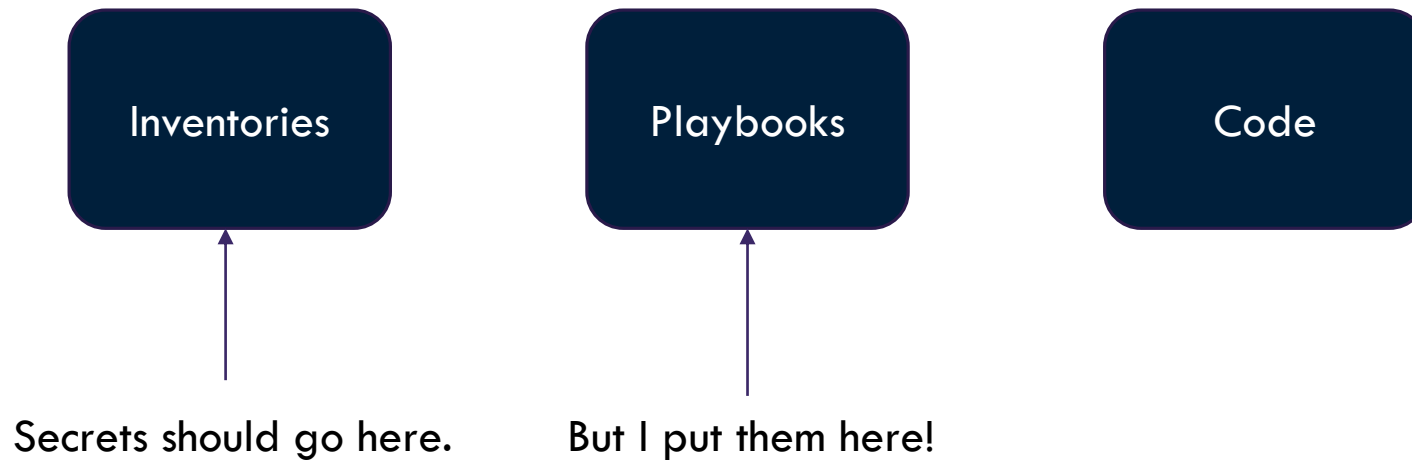
Inventories

Playbooks

Code

What doesn't work

Repository Layout



What doesn't work

Management server has access to all VMs.

- Can I restrict access somehow?
- Maybe use password authentication rather than key based for ansible ssh?

Gitlab is focussed on development rather than operations.

- I want buttons I can give to service desk, or managers for self service access to environment refreshes (for example). Gitlab doesn't seem to offer this. (Rundeck?)

The future

Learn from the mistakes identified above

Can I automate the following:

- Take a copy of production (Yes)
- Rebuild the VMs at the correct version of tools (Yes)
- Apply the tools release to the database (If required)
- Apply the Campus Solutions image (If required)
- Apply our customisations
- Run an automated test (Full test or read only sanity check)

SUMMARY

- Have a vision.
- Communicate it to inspire colleagues.
- Work towards it incrementally.
- Advertise success to colleagues.
- Accept mistakes are a part of the process.
- Use whatever tools are available and are useful.
- Rinse and repeat!

Questions



Presenter

Paul Houghton

Senior Database Administrator

University of Cambridge

psh35@cam.ac.uk

<https://curiousdba.netlify.com>

**ALL ALLIANCE PRESENTATIONS WILL BE AVAILABLE FOR
DOWNLOAD FROM THE CONFERENCE SITE**



THANK YOU!

