



Ansible on IBM Power and IBM PowerVC Updates

Stuart Cunliffe

CTO Systems Lab Services Europe (Power and Cognitive)

email: s_cunliffe@uk.ibm.com

Twitter: [@StuCunliffe](https://twitter.com/StuCunliffe)

slack: [@Stu Cunliffe](#)

Agenda



- Ansible Overview
 - Architecture
 - Engine, Tower
- Terminology
 - Inventory
 - Configuration file
 - Modules
 - Playbooks and Roles
- Provisioning via PowerVC
 - Upgrading AIX LPARs using Ansible and NIM
- Managing clients
 - Simple Playbooks
 - Advanced Playbooks
- Ansible Tower

- PowerVC Updates

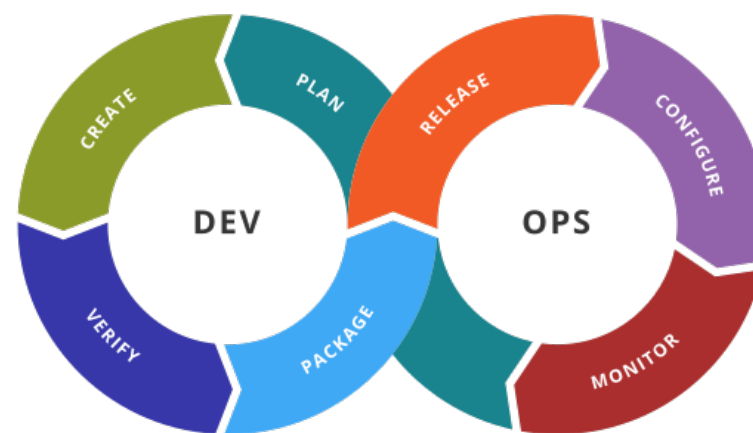
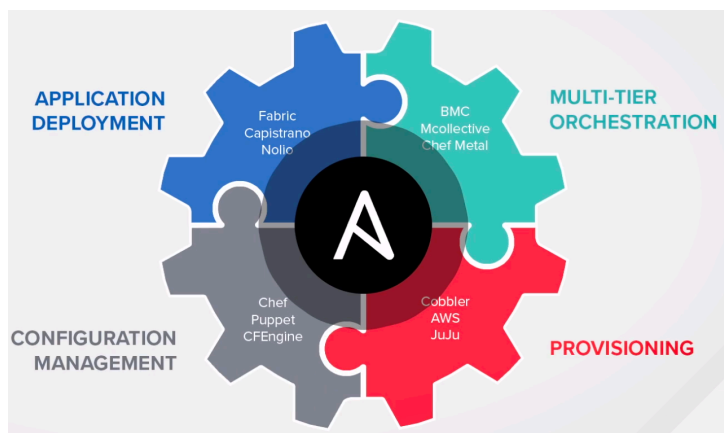
Stuart Cunliffe
email: s_cunliffe@uk.ibm.com
Twitter: [@StuCunliffe](https://twitter.com/StuCunliffe)
slack: [@Stu Cunliffe](#)

Ansible Overview

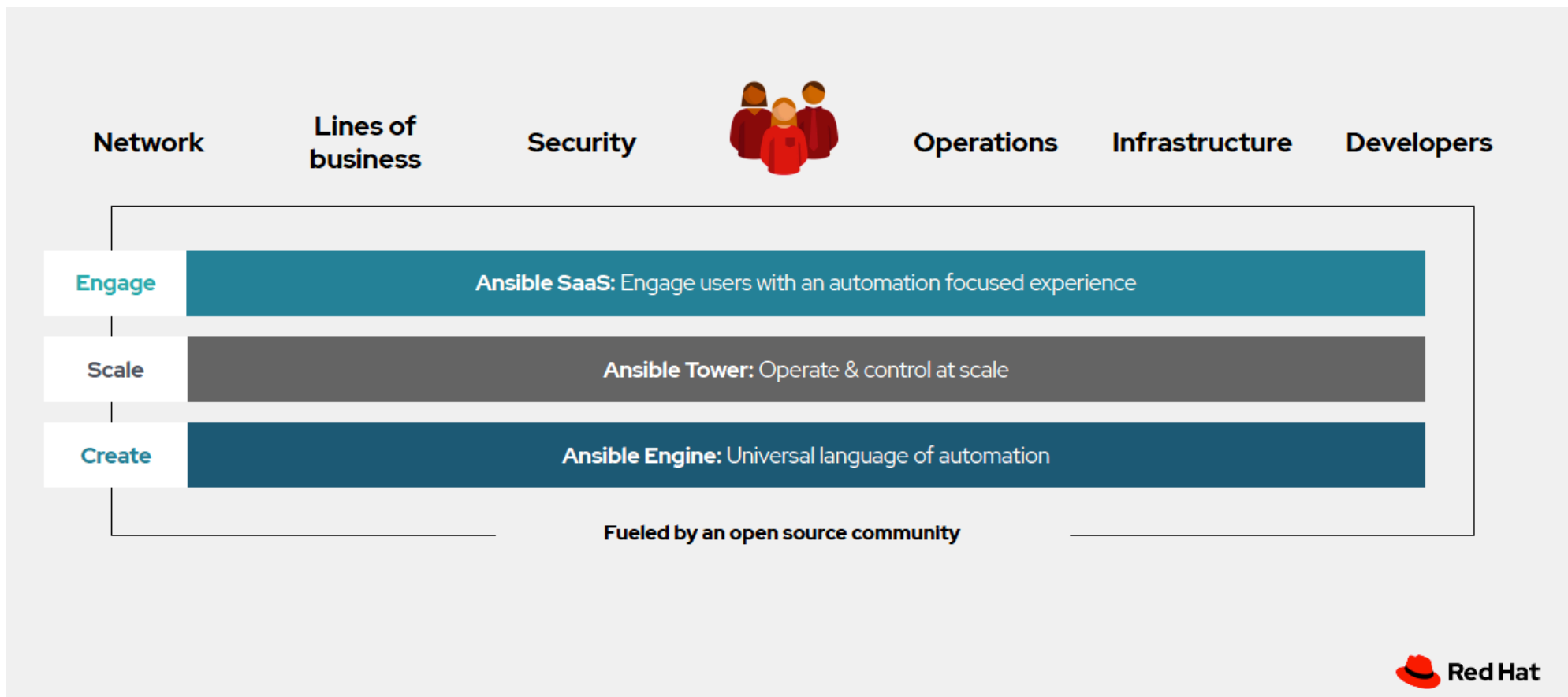


“Ansible is an open source automation tool for provisioning, orchestration, system configuration and patching”

First developed by Michael DeHaan and acquired by Red Hat in 2015.



Ansible Overview



Ansible Overview – key points

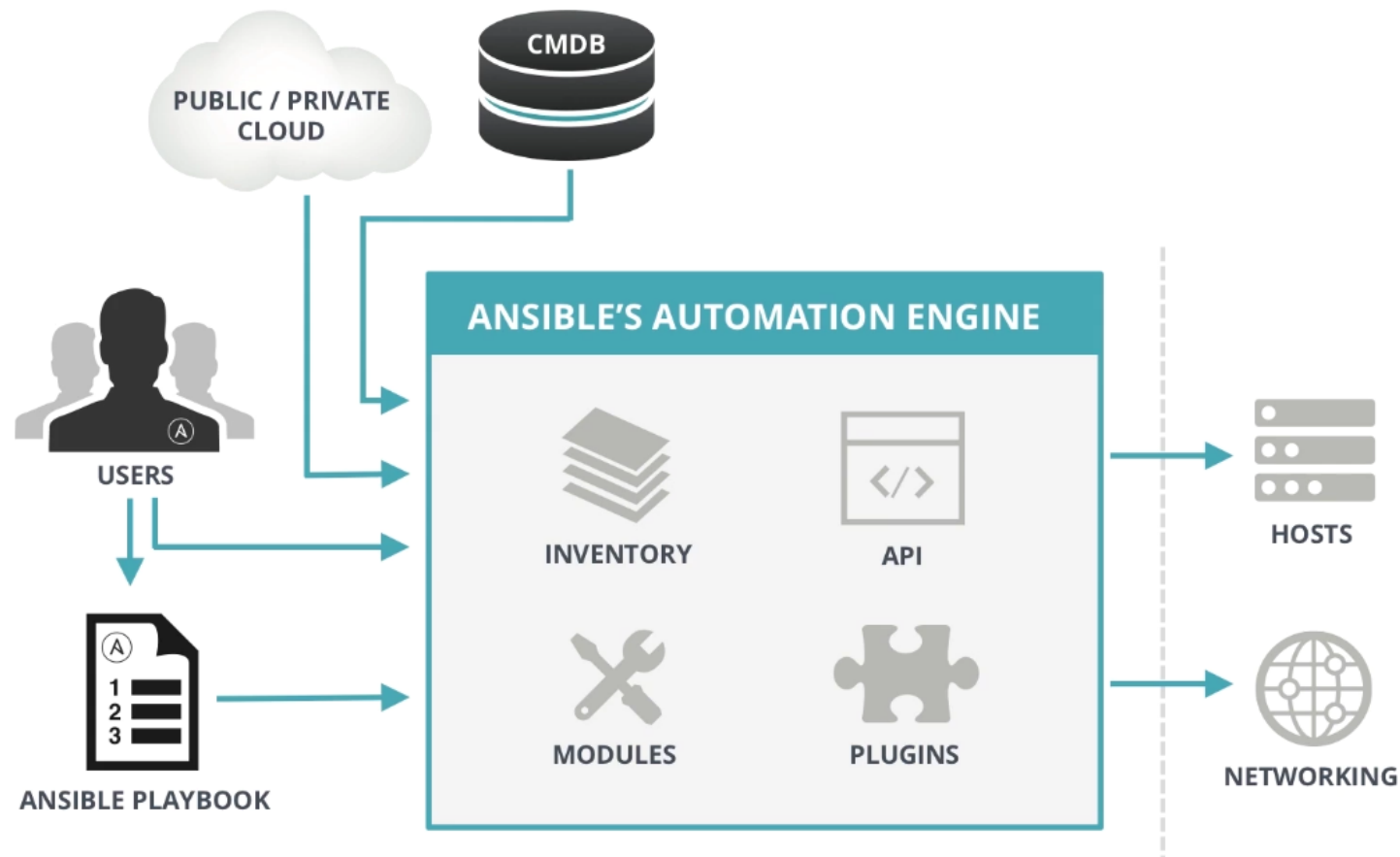


1. The Ansible Engine Controller runs on RHEL or Ubuntu
2. The Engine can manage a large number of clients (via an inventory)
3. It does not require an agent on the clients
4. Uses SSH to communicate with the clients
5. The clients can be AIX, IBM i, RHEL, Ubuntu, SLES, Centos, Fedora, network switches, storage controllers etc.....
6. Human readable automation
7. No special coding skills needed
8. Uses modules to perform tasks, these tasks can be called from the command line or playbooks
9. It is idempotent
10. Simple to get started

Architecture



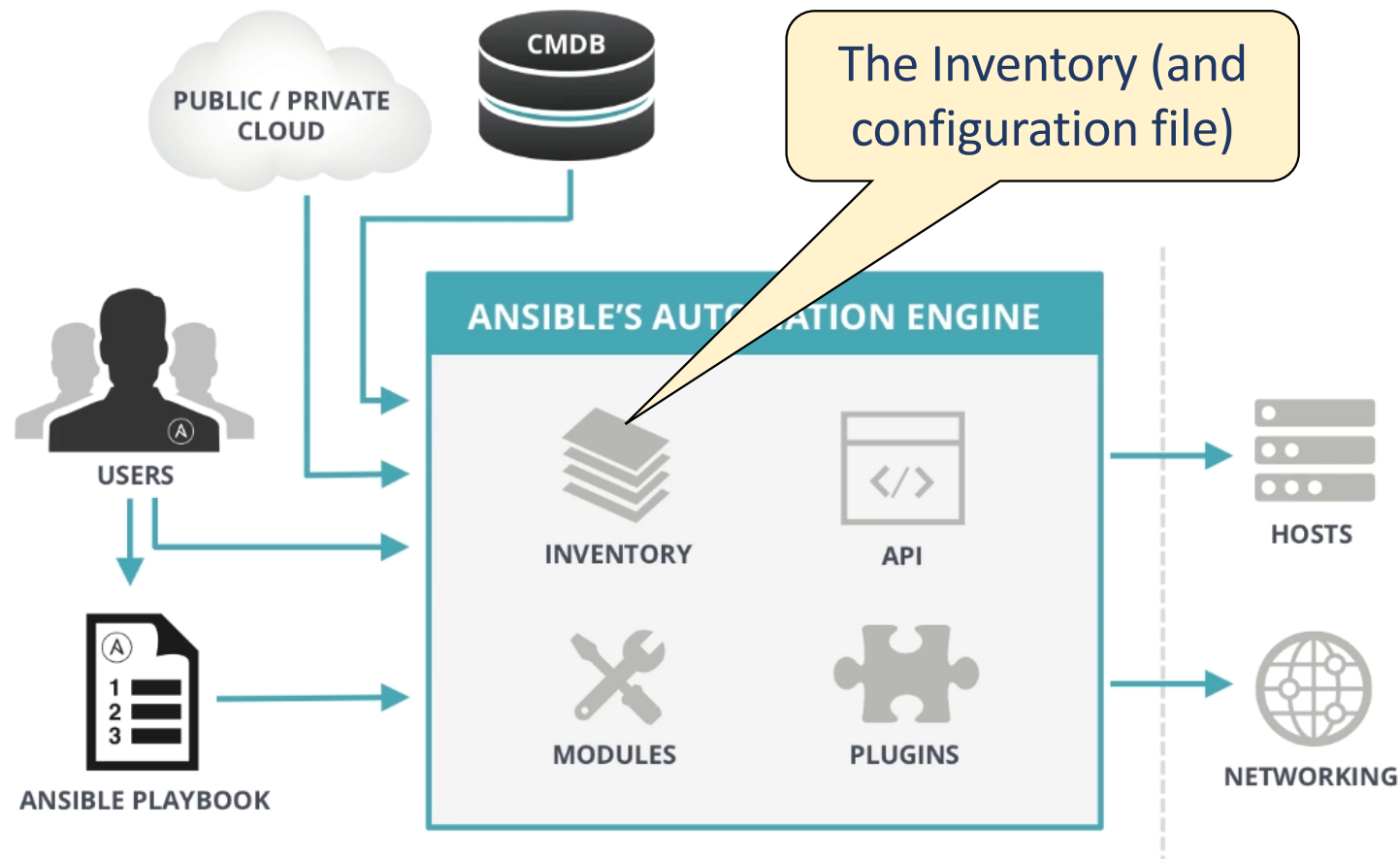
1. Ansible Engine
2. Inventory
3. Modules
4. Playbooks
5. Client hosts



How Ansible works



1. Ansible Engine
2. Inventory
3. Modules
4. Playbooks
5. Client hosts



How Ansible works – The Inventory



1. The client inventory file is a configurable list of VMs/clients that ansible can control.
2. It is written in an INI or YAML format, lists host and groups.
3. Can be static or dynamic.

Static Inventory example

```
# cat /etc/ansible/hosts
[managedClients]
[RHEL_Dev]           ← Group Name
lab-rhel-1
lab-rhel-2          ← Client Name

[AIX_Dev]
lab-aix-1
lab-aix-2

[Dev:children]      ← Collection of groups
RHEL_Dev
AIX_Dev
```

How Ansible works – The Inventory



So we can list the files in the inventory by using 'ansible-inventory'

```
# ansible-inventory --graph
@all:
|--@Dev:           ← Collection of groups
| |--@AIX_Dev:    ← Group Name
| | |--lab-aix-1
| | |--lab-aix-2  ← Client Name
| |--@RHEL_Dev:
| | |--lab-rhel-1
| | |--lab-rhel-2
|--@local:
| |--localhost
```

How Ansible works – The Inventory



We can use the inventory file to configure some connection options to the clients.

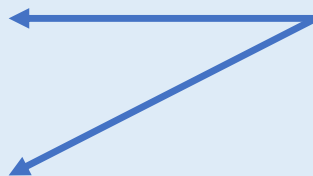
Static Inventory example with connection variables

```
# cat /etc/ansible/hosts
[managedClients]
[RHEL_Dev]
lab-rhel-1 ansible_user=ansible
lab-rhel-2 ansible_port=222

[AIX_Dev]
lab-aix-1 ansible_host=10.1.1.1
lab-aix-2

[Dev:children]
RHEL_Dev
AIX_Dev
```

Client unique variables



How Ansible works – The Inventory



We can use the inventory file to configure some connection options to the clients.

```
# ansible-inventory -list
....
"hostvars": {
  "lab-aix-1": {
    "ansible_host": "10.1.1.1"
  },
  "lab-rhel-1": {
    "ansible_user": "ansible"
  },
  "lab-rhel-2": {
    "ansible_port": 222
  }
}
....
```


How Ansible works – The Inventory



We can use the inventory file to configure group connection options to the clients.

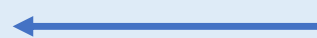
Static Inventory example with group connection variables

```
# cat /etc/ansible/hosts
[managedClients]
[RHEL_Dev]
lab-rhel-1 ansible_user=ansible
lab-rhel-2 ansible_port=222

[AIX_Dev]
lab-aix-1 ansible_host=10.1.1.1
lab-aix-2

[Dev:children]
RHEL_Dev
AIX_Dev

[AIX_Dev:vars]
proxy=proxy.labs.uk.ibm.com
```



Variable applies to whole group

How Ansible works – The Inventory



We can use the inventory file to configure group connection options to the clients.

```
# ansible-inventory -list
....
"hostvars": {
  "lab-aix-1": {
    "ansible_host": "10.1.1.1",
    "proxy": "proxy.labs.uk.ibm.com" ←
  },
  "lab-aix-2": {
    "proxy": "proxy.labs.uk.ibm.com" ←
  },
  "lab-rhel-1": {
    "ansible_user": "ansible"
  },
  "lab-rhel-2": {
    "ansible_port": 222
  }
}
....
```

Both clients in the group have picked up the new connection variable

How Ansible works – The Inventory



We can configure a dynamic inventory (on-prem or public cloud).

PowerVC/OpenStack Dynamic Inventory example

```
# ansible-inventory -i ./openstack.yml --graph
@all:
|--@meta-original_host_828422A_2177C0W:
| |--Lab-Ansible-64
| |--Lab-Chef-65
| |--Lab-DNS1-52
| |--Lab-GDR-51
| |--Lab-NIM-26
| |--Lab-PowerVC-57
| |--Lab-Proxy1-29
| |--Lab-RM-GUI-31
| |--Lab-VMRM-50
....
```

How Ansible works – The Inventory



We have a number of ways to tell Ansible which inventory file to use, in precedence:

1. the '-i' flag on the command line (you can call more than one inventory file if needed)
2. The ANSIBLE_INVENTORY environment variable
3. Using "inventory=xxx" in the ansible configuration file
4. If all else fails, the default is /etc/ansible/hosts

Method to check which inventory file you are using

```
# ansible -v -a "echo Inventory File is {{ inventory_file }}" localhost
Using /etc/ansible/ansible.cfg as config file
....
- Inventory
  - File
  - is
  - /etc/ansible/hosts
....
```


How Ansible works – The ansible config file



Ansible looks for a configuration file to determine a number of parameters. As with the inventory file, a number of configuration files can be defined for different projects.

Nearly all parameters in `ansible.cfg` can be overwritten in playbooks or during ansible calls.

Example `ansible.cfg` file

```
# cat /etc/ansible/ansible.cfg
[defaults]
inventory    = /etc/ansible/hosts
library       = /usr/share/ansible/plugins/modules
module_utils  = /usr/share/my_module_utils/
remote_tmp    = ~/.ansible/tmp
local_tmp     = ~/.ansible/tmp
sudo_user    = root
ask_sudo_pass = True
ask_pass     = True
remote_port = 22
.....
```

How Ansible works – The ansible config file



The active configuration files uses the following locations, in precedence:

1. The ANSIBLE_CONFIG environment variable
2. ./ansible.cfg - within the current directory
3. ~/.ansible.cfg. - home directory
4. If all else fails, the default is /etc/ansible/ansible.cfg

Method to check which configuration file you are using

```
# ansible --version
ansible 2.9.6
config file = /etc/ansible/ansible.cfg
configured module search path = [u'/root/.ansible/plugins/modules', u'/usr/share/ansible/plugins/modules']
ansible python module location = /usr/lib/python2.7/site-packages/ansible
executable location = /usr/bin/ansible
python version = 2.7.5 (default, Jun 11 2019, 14:33:56) [GCC 4.8.5 20150623 (Red Hat 4.8.5-39)]
```

How Ansible works – The ansible config file



We can display all the current Ansible values. There are approx. 190 configuration options:

Display configuration parameters

```
# ansible-config dump
ACTION_WARNINGS(default) = True
AGNOSTIC_BECOME_PROMPT(default) = True
ALLOW_WORLD_READABLE_TMPFILES(default) = False
ANSIBLE_CONNECTION_PATH(default) = None
ANSIBLE_COW_PATH(default) = None
ANSIBLE_COW_SELECTION(default) = default
ANSIBLE_COW_WHITELIST(default) = ['bud-frogs', 'bunny', 'cheese', 'daemon', 'default', 'dragon', 'elephant-in-
snake', 'elephant', 'ey
ANSIBLE_FORCE_COLOR(default) = False
ANSIBLE_SSH_CONTROL_PATH(default) = None
ANSIBLE_SSH_CONTROL_PATH_DIR(default) = ~/.ansible/cp
ANSIBLE_SSH_EXECUTABLE(default) = ssh
....
```

How Ansible works – The ansible config file



If no config file exists or if a parameter hasn't been set, Ansible uses all default settings. We can see which values are not default:

Display non-default configuration parameters

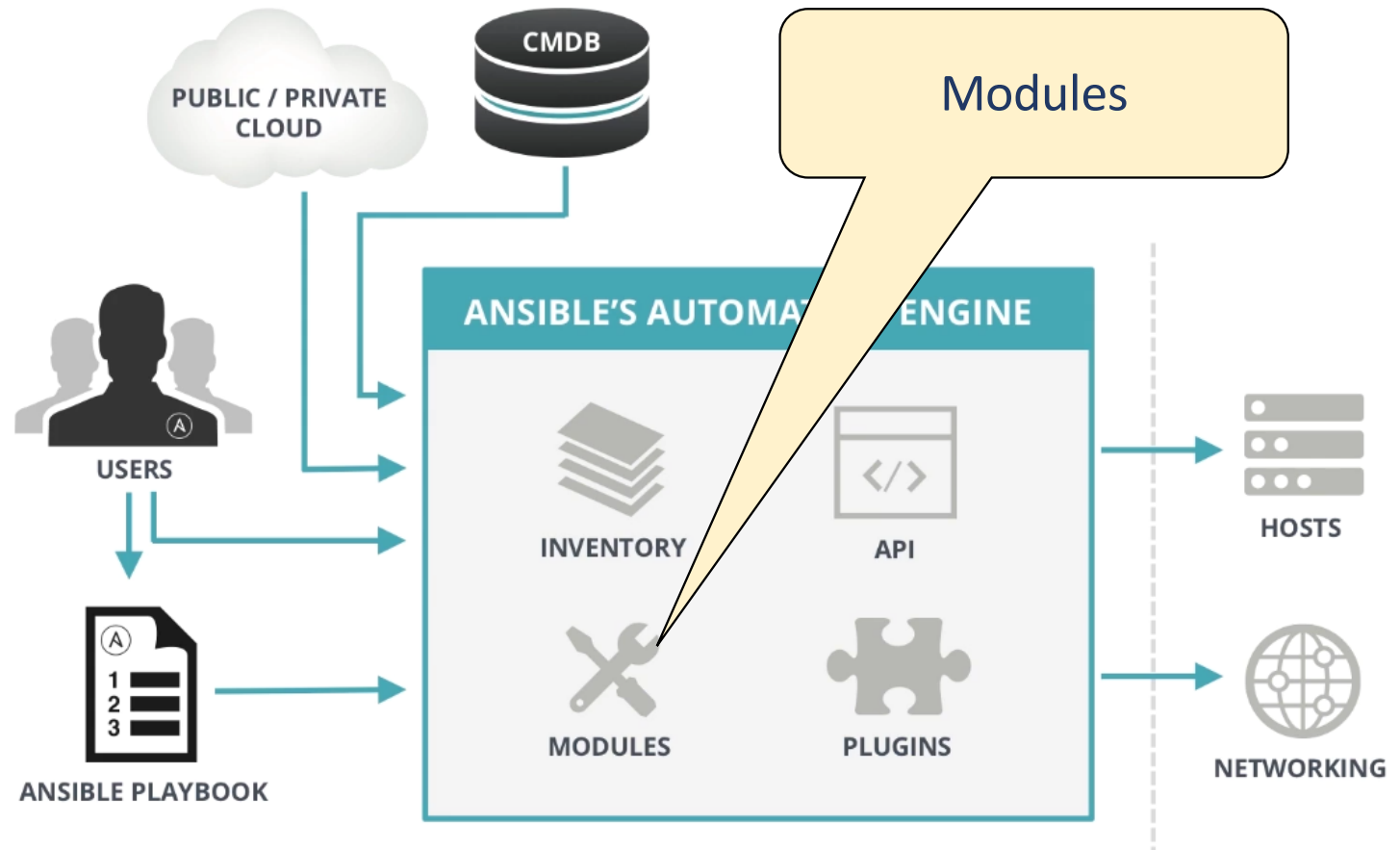
```
# ansible-config dump --only-changed
DEFAULT_HOST_LIST(/etc/ansible/ansible.cfg) = [u'/etc/ansible/hosts']
DEFAULT_LOAD_CALLBACK_PLUGINS(/etc/ansible/ansible.cfg) = True
DEFAULT_STDOUT_CALLBACK(/etc/ansible/ansible.cfg) = yaml
HOST_KEY_CHECKING(/etc/ansible/ansible.cfg) = False
INTERPRETER_PYTHON(/etc/ansible/ansible.cfg) = auto_silent
```

NOTE: It is very important to ensure you are using the correct configuration and inventory files. Although when we call modules and playbooks we can specify the hosts this isn't mandatory. Calling the wrong inventory could cause significant issues.

How Ansible works



1. Ansible Engine
2. Inventory
3. Modules
4. Playbooks
5. Client hosts



How Ansible works – Modules



Modules are the core of Ansible

1. They perform the real work by executing on the clients.
 - ✓ Ansible engine connects to your clients
 - ✓ It pushes out the module along with parameters
 - ✓ The module is then executed on the client
 - ✓ The module is then removed from the client
2. Ansible comes with thousands of modules covering server, network, storage, files, DB etc.
3. Can be written in Python, Perl, Ruby, Bash, etc. – that return JSON format
4. You can write your own modules
5. Command line syntax: *'ansible -m <module_name> -a <attributes>'*
6. They are idempotent (that word again)....

Dictionary definition:

“denoting an element of a set which is unchanged in value when multiplied or otherwise operated on by itself”

“For Ansible it means after 1 run of a playbook to set things to a desired state, further runs of the same playbook should result in 0 changes. Idempotency means you can be sure of a consistent state in your environment.”

How Ansible works – Modules (idempotency)



ANSIBLE

Add a logical volume – first run

```
# ansible lab-aix-1 -m aix_lvol -a "lv=testlv size=10M vg=rootvg"
PLAY [Ansible Ad-Hoc] *****
TASK [aix_lvol] *****
changed: [lab-aix-1]
PLAY RECAP
*****
lab-aix-1      : ok=1  changed=1  unreachable=0  failed=0  skipped=0  rescued=0  ignored=0
```

← During the first run a change occurs. The LV is created.

Add a logical volume – second run

```
# ansible lab-aix-1 -m aix_lvol -a "lv=testlv size=10M vg=rootvg"
PLAY [Ansible Ad-Hoc] *****
TASK [aix_lvol] *****
ok: [lab-aix-1]
PLAY RECAP
*****
lab-aix-1      : ok=1  changed=0  unreachable=0  failed=0  skipped=0  rescued=0  ignored=0
```

← During the second run a change does NOT occur. The LV already exists.

How Ansible works – Modules



Ansible comes with thousands of ‘core’ modules, divided into categories:

https://docs.ansible.com/ansible/latest/modules/modules_by_category.html#modules-by-category

Module Index

- | | |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <ul style="list-style-type: none">• All modules• Cloud modules• Clustering modules• Commands modules• Crypto modules• Database modules• Files modules• Identity modules• Inventory modules• Messaging modules• Monitoring modules | <ul style="list-style-type: none">• Network modules• Notification modules• Packaging modules• Remote Management modules• Source Control modules• Storage modules• System modules• Utilities modules• Web Infrastructure modules• Windows modules |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

How Ansible works – Modules



Some modules are fairly generic across different architectures/device types:

- ✓ **ping** module - Try to connect to host, verify a usable python and return “pong” on success

Some modules are specific to OS types e.g. Linux:

- ✓ **systemd** module – Manage services (uses systemctl command)

Other modules have been written for specific OS distribution e.g. AIX:

- ✓ **aix_devices** modules – Manage AIX devices (uses chdev, lsdev, etc. commands)

How Ansible works – Modules



ANSIBLE

As well as Ansible's website we can also use the Ansible Engine server to show modules, how they are supported, options available etc.

Using 'ansible-doc' to review a module

ansible-doc aix_lvol

> AIX_LVOL (/usr/lib/python2.7/site-packages/ansible/modules/system/aix_lvol.py)

This module creates, removes or resizes AIX logical volumes. Inspired by lvol module.

* This module is maintained by The Ansible Community

OPTIONS (= is mandatory):

- copies

The number of copies of the logical volume.

Maximum copies are 3.

[Default: 1]

type: int

= lv

The name of the logical volume.

type: str

- lv_type

The type of the logical volume.

[Default: jfs2]

type: str

.....

Shows the location of the module and support level.

The "=" indicates mandatory parameters.

How Ansible works – Modules



ANSIBLE

An example module.

aix_lv module

```
# cat /usr/lib/python2.7/site-packages/ansible/modules/system/aix_lv.py
#!/usr/bin/python
# Copyright: (c) 2016, Alain Dejoux <adejoux@djouxtech.net>
# GNU General Public License v3.0+ (see COPYING or https://www.gnu.org/licenses/gpl-3.0.txt)
...
author:
  - Alain Dejoux (@adejoux)
module: aix_lv
short_description: Configure AIX LVM logical volumes
description:
  - This module creates, removes or resizes AIX logical volumes. Inspired by lv module.
.....
if this_lv is None:
    if state == 'present':
        if lv_size > this_vg['free']:
            module.fail_json(msg="Not enough free space in volume group %s: %s MB free." % (this_vg['name'], this_vg['free']))
            mklv_cmd = module.get_bin_path("mklv", required=True)
            cmd = "%s %s -t %s -y %s -c %s -e %s %s %s %sM %s" % (test_opt, mklv_cmd, lv_type, lv, copies, lv_policy, opts, vg, lv_size,
pv_list)
.....
```

Should include information about the author

We can read the underlying commands being called

How Ansible works – Modules



What happens if we call an invalid module?

Calling an AIX module on a Linux client

```
# ansible lab-rhel-1 -m aix_lvol -a "lv=testlv size=10M vg=rootvg"
PLAY [Ansible Ad-Hoc]
*****

TASK [aix_lvol]
*****
fatal: [lab-rhel-1]: FAILED! => changed=false
  ansible_facts:
    discovered_interpreter_python: /usr/bin/python
  msg: 'Failed to find required executable lsvg in paths: /usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin'

PLAY RECAP
*****
lab-rhel-1      : ok=0  changed=0  unreachable=0  failed=1  skipped=0  rescued=0  ignored=0
```

The AIX command 'lsvg' was not found on the client.



How Ansible works – Modules



AIX specific modules inc. in Ansible library

System modules

- `aix_devices` – Manages AIX devices
- `aix_filesystem` – Configure LVM and NFS file systems for AIX
- `aix_inittab` – Manages the inittab on AIX
- `aix_lvg` – Manage LVM volume groups on AIX
- `aix_lvol` – Configure AIX LVM logical volumes
- `mksysb` – Generates AIX mksysb rootvg backups
- `installp` – Manage packages on AIX

Other AIX specific modules

e.g. <https://github.com/ansible/community/wiki/AIX> & <https://github.com/aioxss/ansible-playbooks/tree/master/library>

 [aix_flrtvc.py](#)

 [aix_nim.py](#)

 [aix_nim_updateios.py](#)

 [aix_nim_upgradeios.py](#)

 [aix_nim_vios_alt_disk.py](#)

 [aix_nim_vios_hc.py](#)

 [aix_nim_viosupgrade.py](#)

 [aix_suma.py](#)

 [suma.py](#)

How Ansible works – Modules (IBMi)



ANSIBLE

IBMi specific modules from GitHub: <https://github.com/IBM/ansible-for-i/tree/master/library>

IBMi and Ansible: https://mediacenter.ibm.com/media/t/1_fdz7x3vi

[_init_.py](#)

[ibmi_at.py](#)

[ibmi_cl_command.py](#)

[ibmi_copy.py](#)

[ibmi_device_vary.py](#)

[ibmi_display_subsystem.py](#)

[ibmi_end_subsystem.py](#)

[ibmi_fetch.py](#)

[ibmi_fix.py](#)

[ibmi_fix_imgclg.py](#)

[ibmi_get_nonconfigure_disks.py](#)

[ibmi_host_server_service.py](#)

[ibmi_iasp.py](#)

[ibmi_install_product_from_savf.py](#)

[ibmi_job.py](#)

[ibmi_lib_restore.py](#)

[ibmi_lib_save.py](#)

[ibmi_message.py](#)

[ibmi_object_authority.py](#)

[ibmi_object_find.py](#)

[ibmi_object_restore.py](#)

[ibmi_object_save.py](#)

[ibmi_reboot.py](#)

[ibmi_save_product_to_savf.py](#)

[ibmi_script.py](#)

[ibmi_script_execute.py](#)

[ibmi_sql_execute.py](#)

[ibmi_sql_query.py](#)

[ibmi_start_subsystem.py](#)

[ibmi_submit_job.py](#)

[ibmi_sync.py](#)

[ibmi_synchronize.py](#)

[ibmi_tcp_interface.py](#)

[ibmi_tcp_server_service.py](#)

[ibmi_uninstall_product.py](#)

[ibmi_user_and_group.py](#)

How Ansible works – Core Modules & Power



AIX collection modules – due June 2020:

Targeted Use case	Applicable Module(s)
Service & Technology Level Updating	suma
Interim Fix Patching (AIX)	emgr
Update Policy Scheduling	suma
System Vulnerability Security Scan (AIX/VIOS)	flrtvc
Open Source Package Management (Yum/ PiP)	geninstall, yum*, pip*
Centralized AIX/VIOS Patch Management (NIM)	nim_suma, nim_flrtvc
Centralized AIX/VIOS Backup & Restore (NIM)	nim
Alternate Disk Copying for VIOS	nim_vios_alt_disk

How Ansible works – Modules



What happens if we don't have OS specific models?

Don't panic there are a number of 'command' modules we can run.

Commands modules

- `command` – Execute commands on targets
- `expect` – Executes a command and responds to prompts
- `psexec` – Runs commands on a remote Windows host based on the PsExec model
- `raw` – Executes a low-down and dirty command
- `script` – Runs a local script on a remote node after transferring it
- `shell` – Execute shell commands on targets
- `telnet` – Executes a low-down and dirty telnet command

How Ansible works – Modules



✓ **command** module – Execute commands on targets

Simple 'command' module example

```
# ansible lab-aix-1 -m command -a "cat /etc/motd"
lab-aix-1 | CHANGED | rc=0 >>
*****
*
*
* Welcome to AIX Version 7.2!
*
*
* Please see the README file in /usr/lpp/bos for information pertinent to
* this release of the AIX Operating System.
*
*
*****
```

How Ansible works – Modules



- ✓ **script** module – Runs a local script on a remote node after transferring it

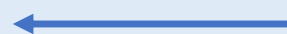
Simple 'script' module example

```
# cat ./show_date.sh
#!/bin/sh
date
```



Script on the Ansible Engine

```
# ansible lab-aix-1 -m script -a "./show_date.sh"
lab-aix-1 | CHANGED => {
  "changed": true,
  "rc": 0,
  "stderr": "Shared connection to lab-aix-1 closed.\r\n",
  "stderr_lines": [
    "Shared connection to lab-aix-1 closed."
  ],
  "stdout": "Fri May 15 08:31:21 CDT 2020\r\n",
  "stdout_lines": [
    "Fri May 15 08:31:21 CDT 2020"
  ]
}
```



Script is copied over and
executed on the client

How Ansible works – Modules (setup and facts)



ANSIBLE

- ✓ **setup** module – Gathers facts about remote hosts (~1000 lines for a AIX LPAR)

Setup module

```
# ansible lab-aix-1 -m setup
lab-aix-1 | SUCCESS => {
  "ansible_facts": {
    "ansible_all_ipv4_addresses": [
      "10.1.1.10"
    ]
    ...
    "fcs0": {
      "attributes": {
        "max_xfer_size": "0x100000"
        "state": "Available",
        "type": "C3-T1 Virtual Fibre Channel Client Adapter"
      }
    }
    ....
    "ansible_nodename": "lab-aix-1",
    "ansible_os_family": "AIX",
    "ansible_processor": "PowerPC_POWER8",
    "ansible_processor_cores": 8,
    "ansible_processor_count": 1,
    "ansible_product_name": "IBM,9119-MHE",
    "ansible_product_serial": "21Cxxxx",
    ....
  }
}
```

Thousands of facts about h/w, OS, network and storage devices etc. can be gathered.

These can be used to filter which clients to run a task against in a playbook.

How Ansible works – Basic Demo

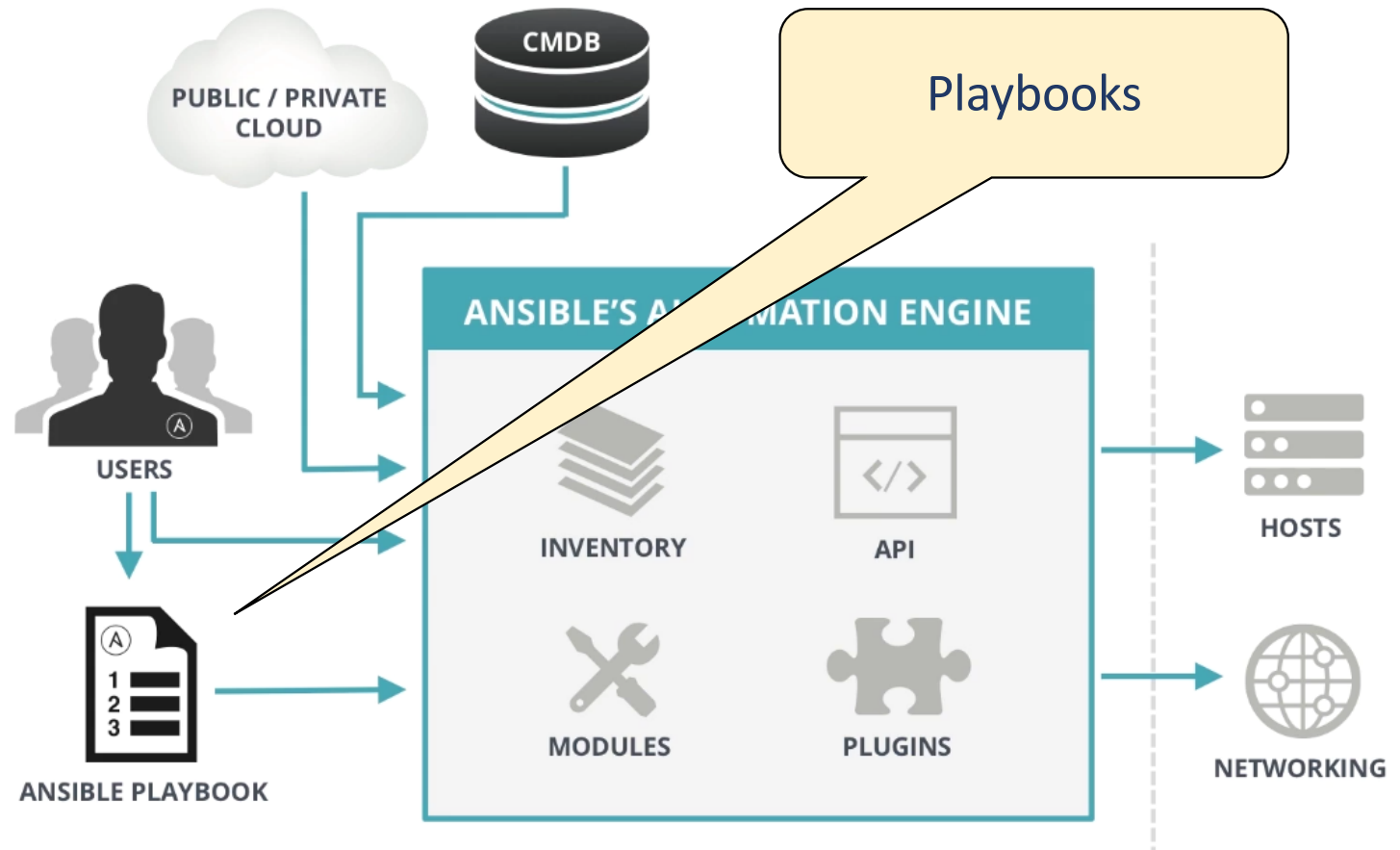


ANSIBLE

How Ansible works



1. Ansible Engine
2. Inventory
3. Modules
4. Playbooks
5. Client hosts



How Ansible works – Playbooks



Modules might be the core, but Playbooks are how we drive Ansible

- ✓ Playbooks are Ansible's configuration, deployment, and orchestration language.
- ✓ They are the instruction manual describing the configuration you want your remote clients to enforce.
- ✓ Written in YAML format, so should be readable.

Basic playbooks:

Used to manage configurations of and deployments to remote machines.

Advanced playbooks:

They can sequence multi-tier rollouts involving rolling updates, and can delegate actions to other hosts, interacting with monitoring servers and load balancers along the way.

How Ansible works – Playbooks



ANSIBLE

A playbook consists of 'plays', which in turn consist of 'tasks', which contain 'modules'.

Simple playbook

```
# cat AIX_LV_backup_playbook.yml
```

```
---  
- name: Create backup LV  
  gather_facts: no  
  hosts: AIX_Dev
```

```
  tasks:
```

```
    - name: Create LV (backuplv) size 10MB  
      aix_lvol:  
        vg: rootvg  
        lv: backuplv  
        size: 10M  
        state: present
```

Task

Play

Playbook

How Ansible works – Playbooks



ANSIBLE

A playbook consists of 'plays', which in turn consist of 'tasks', which contain 'modules'.

Simple playbook

```
# cat AIX_LV_backup_playbook.yml
```

```
---
```

```
- name: Create backup LV
```

```
gather_facts: no
```

```
hosts: AIX_Dev
```

```
tasks:
```

```
- name: Create LV (backuplv) size 10MB
```

```
  aix_lvol:
```

```
    vg: rootvg
```

```
    lv: backuplv
```

```
    size: 10M
```

```
    state: present
```

Define the 'play'

Do not gather facts

Which hosts to run the play against. 'All' will run it against all clients in the inventory

Define the 'task'

The name of the module to call for this task

Module parameters to use for this task

How Ansible works – Playbooks



A playbook consists of 'plays', which in turn consist of 'tasks', which contain 'modules'.

Simple playbook

```
# ansible-playbook AIX_LV_backup_playbook.yml
```

```
PLAY [Create backup LV]
```

← The name of the 'play'

```
*****
```

```
TASK [Create LV (backuplv) size 10MB]
```

← The name of the 'task'

```
*****
```

```
changed: [lab-aix-1]
```

```
changed: [lab-aix-2]
```

← Completed on 2 clients

```
PLAY RECAP
```

```
*****
```

```
lab-aix-1      : ok=1  changed=1  unreachable=0  failed=0  skipped=0  rescued=0  ignored=0
```

```
lab-aix-2      : ok=1  changed=1  unreachable=0  failed=0  skipped=0  rescued=0  ignored=0
```

How Ansible works – Playbooks (tasks and tags)



We can list the tasks in a playbook without actually running it:

Task in a playbook

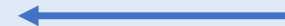
```
# ansible-playbook ./AIX_multiple_task_v4_all_tasks.yml --list-tasks
```

```
playbook: ./AIX_multiple_task_v4_all_tasks.yml
```

```
play #1 (all): mksysb & expect.man client install TAGS: []
```

```
tasks:
```

```
  Check if installp directory exists TAGS: []
  Make /installp directory if missing TAGS: []
  Copy installp files into /installp TAGS: []
  Create mksysb LV (mksysblv) with 5GB TAGS: []
  Create mksysb filesystem using mksysblv TAGS: []
  Mount mksysb filesystem /mksysb/images TAGS: []
  Install expect.man client from file TAGS: []
  Display expect.man LPP install result TAGS: []
  Generate a mksysb on /mksysb/images TAGS: []
```



All the tasks are listed
but not executed

How Ansible works – Playbooks (tasks and tags)



We can also ‘tag’ tasks with identifiers :

Task and tags in a playbook

```
# cat ./AIX_multiple_task_v4_all_tasks.yml
```

```
tasks:
```

```
- name: Check if installp directory exists
```

```
  stat:
```

```
    path: "{{ install_dir_name }}"
```

```
    register: file_details
```

```
    tags: install
```

```
....
```

```
- name: Copy installp files into /installp
```

```
  copy:
```

```
    src: '/tmp/{{ install_lpp_name }}'
```

```
    dest: '{{ install_dir_name }}/{{ install_lpp_name }}'
```

```
    tags: install, copy
```

```
....
```

```
- name: Create mksysb filesystem using mksysblv
```

```
  aix_filesystem:
```

```
    device:  "{{ filesystem_lv_name }}"
```

```
    filesystem: "{{ filesystem_fs_name }}"
```

```
    state: present
```

```
    register: fs_result
```

```
    tags: filesystem, backup
```

```
....
```

We can add ‘tag’
names to each task.

How Ansible works – Playbooks (tasks and tags)



We can also ‘tag’ tasks with identifiers, and list them:

Task and tags in a playbook

```
# ansible-playbook AIX_multiple_task_v4_all.yml --list-tasks

playbook: AIX_multiple_task_v4_all.yml

play #1 (all): mksysb & expect.man client install  TAGS: []
tasks:
  Check if installp directory exists                TAGS: [install]
  Make /installp directory if missing               TAGS: [install]
  Copy installp files into /installp                TAGS: [copy, install]
  Create mksysb LV (mksysblv) with 5GB             TAGS: [backup, filesystem]
  Create mksysb filesystem using mksysblv          TAGS: [backup, filesystem]
  Mount mksysb filesystem /mksysb/images           TAGS: [backup, filesystem]
  Install expect.man client from file               TAGS: [install]
  Display expect.man LPP install result             TAGS: [install]
  Generate a mksysb on /mksysb/images              TAGS: [backup, mksysb]
```


How Ansible works – Playbooks (tasks and tags)



We can then just run certain tasks, by giving a tag:

List copy tasks only

```
# ansible-playbook AIX_multiple_task_v4_all.yml --list-task -t copy
playbook: AIX_multiple_task_v4_all.yml
play #1 (all): mksysb & expect.man client install  TAGS: []
tasks:
  Copy installp files into /installp  TAGS: [copy, install]
```

Run 'copy' tasks only

```
# ansible-playbook AIX_multiple_task_v4_all.yml -t copy
PLAY [mksysb & expect.man client install]
*****
TASK [Gathering Facts]
*****
ok: [lab-aix-1]
TASK [Copy installp files into /installp]
*****
changed: [lab-aix-1]
PLAY RECAP *****
lab-aix-1      : ok=2  changed=1  unreachable=0  failed=0  skipped=0  rescued=0  ignored=0
```

How Ansible works – Playbooks (variables)



We can define variables from within the playbook

Playbook variables example

```
# cat Install_VMRM_agent_v1.0.yml
```

```
.....
```

```
vars:
```

```
source_dir: /root/VMRM_Code
```

```
target_dir: /tmp
```

```
aix_code: ksys.vmmon.rte
```

```
rhel_code: vmagent-1.3.0-1.0.el7.ppc64le.rpm
```

← Variable defined in the playbook

```
- name: Copy VM agent code - AIX
```

```
copy:
```

```
src="{{ source_dir }}/{{ aix_code }}"
```

```
dest="{{ target_dir }}/{{ aix_code }}"
```

← Copy module called

```
- name: Copy VM agent code - RHEL
```

```
copy:
```

```
src="{{ source_dir }}/{{ rhel_code }}"
```

```
dest="{{ target_dir }}/{{ rhel_code }}"
```

← Different variables used

How Ansible works – Playbooks (variables)



ANSIBLE

We can 'include' variables from an external file. There is a 'priority' order of var definition
Imported variables example

```
# cat OSlevel_check.yml
```

```
---
```

```
- hosts: all
```

```
tasks:
```

```
- name: Load AIX specific variables
```

```
  include_vars: AIX.yml
```

← We include an external variables file

```
- name: Check OS
```

```
  command: "{{ os_check_command }}"
```

← The command modules needs a variable called 'os_check_command'

```
# cat AIX.yml
```

```
---
```

```
# variables for script
```

```
os_check_command: "oslevel -s"
```

```
args_variable_name: "AIX_OS"
```

← The 'os_check_command' is defined in this variable file and passed back to the main playbook.

How Ansible works – Playbooks (conditions)



We can run tasks against 'facts' gathered from the clients, for example OS type

Playbook 'when' example

```
# cat OSlevel_check.yml
---
- hosts: Dev
  tasks:
    - name: Load AIX specific variables
      include_vars: AIX.yml
      when: ansible_distribution == "AIX"
    - name: Load RHEL specific variables
      include_vars: RHEL.yml
      when: ansible_distribution == "RedHat"
    - name: Load Ubuntu specific variables
      include_vars: Ubuntu.yml
      when: ansible_distribution == "Ubuntu"
    - name: Check OS
      command: "{{ os_check_command }}"
      register: os_check_result
      args:
        creates: "{{ args_variable_name }}"
```

Include a different variable file depending on the clients OS type

The relevant OS command is passed back

How Ansible works – Playbooks (Roles)



As we start out with Ansible we tend to create one or two large playbooks

Although this is a good start we may want to reuse file and avoid repeating code.

Roles, import and includes are a good way to do this.

Roles allow us to automatically load certain variables, tasks and handlers based on a known file structure. These can then be shared amongst other uses and projects.

How Ansible works – Playbooks (Roles)



Creating a role:

```
# ansible-galaxy init db-server-role
- Role db-server-role was created successfully
```

Directory structure of a role:

```
# tree
.
├── db-server-role
│   ├── defaults
│   │   └── main.yml
│   ├── files
│   ├── handlers
│   │   └── main.yml
│   ├── meta
│   │   └── main.yml
│   ├── README.md
│   ├── tasks
│   │   └── main.yml
│   ├── templates
│   ├── tests
│   │   ├── inventory
│   │   └── test.yml
│   └── vars
│       └── main.yml
```

If main.yml playbooks exist within the role, the tasks, handlers, variable etc. listed within will be added to the play that called it.

How Ansible works – Playbooks (Roles)



ANSIBLE

Why do we need roles?? If we look at our OpenStack playbook that creates AIX, Linux or IBMi VMs, its complex:

```
# ansible-playbook playbooks/VM_build.yml --list-tasks
play #1 (localhost): Build new VM via PowerVC/OpenStack TAGS: []
tasks:
  Prompt for new VM Name TAGS: [VM_Create]
  Set VM Variables TAGS: [VM_Create]
  Display VM Name TAGS: [VM_Create]
  VM_network_list : Retrieve list of all networks TAGS: [VM_Create, VM_Network]
  VM_network_list : Generate Network list TAGS: [VM_Create, VM_Network]
  VM_network_list : Debug - Output Network list TAGS: [VM_Create, VM_Network]
  VM_network_list : Display Network list TAGS: [VM_Create, VM_Network]
  .....
  VM_image_list : Retrieve list of all OS Distributions TAGS: [VM_Create, VM_Images]
  VM_image_list : Filter OS Distribution list TAGS: [VM_Create, VM_Images]
  .....
  VM_flavor_list : Retrieve list of all public flavors TAGS: [VM_Flavor, always, never]
  ....
  VM_name_list : Retrieve list of all VMs TAGS: [VM_Create, VM_List]
  VM_name_list : Retrieve VM list TAGS: [VM_Create, VM_List]
  ....
  VM_create_vm : Create a new VM instance TAGS: [VM_Create]
  VM_create_vm : Print VM's public IP address TAGS: [VM_Create]
```

Each group
of tasks is in
its own role

65 tasks in total

How Ansible works – Playbooks (Roles)



These roles can be used multiple times from other playbooks, other users or other projects:

```
# cat playbooks/VM_build.yml
```

```
---
```

```
- name: Build new VM via PowerVC/OpenStack
```

```
tasks:
```

```
- name: List Available Networks
```

```
  import_role:
```

```
    name: VM_network_list
```

```
    tags: VM_Create, VM_Network
```

```
- name: Pick Network for VM
```

```
  import_role:
```

```
    name: VM_network_pick
```

```
    tags: VM_Create
```

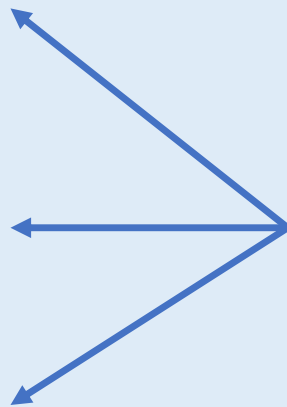
```
- name: List VM images
```

```
  import_role:
```

```
    name: VM_image_list
```

```
    tags: VM_Create, VM_Images
```

```
.....
```



Within the tasks we import each role

How Ansible works – Other features



Handlers

Handlers are lists of tasks, that are referenced by a globally unique name, and are notified by notifiers. If nothing notifies a handler, it will not run. Regardless of how many tasks notify a handler, it will run only once, after all of the tasks complete in a particular play.

Blocks

Blocks allow for logical grouping of tasks and in play error handling. Most of what you can apply to a single task can be applied at the block level, which also makes it much easier to set data or directives common to the tasks.

Vaults

Ansible Vault is a feature of ansible that allows you to keep sensitive data such as passwords or keys in encrypted files, rather than as plaintext in playbooks or roles. These vault files can then be distributed or placed in source control.

Galaxy

Ansible Galaxy refers to the Galaxy website, a free site for finding, downloading, and sharing community developed roles.

<https://galaxy.ansible.com/home>

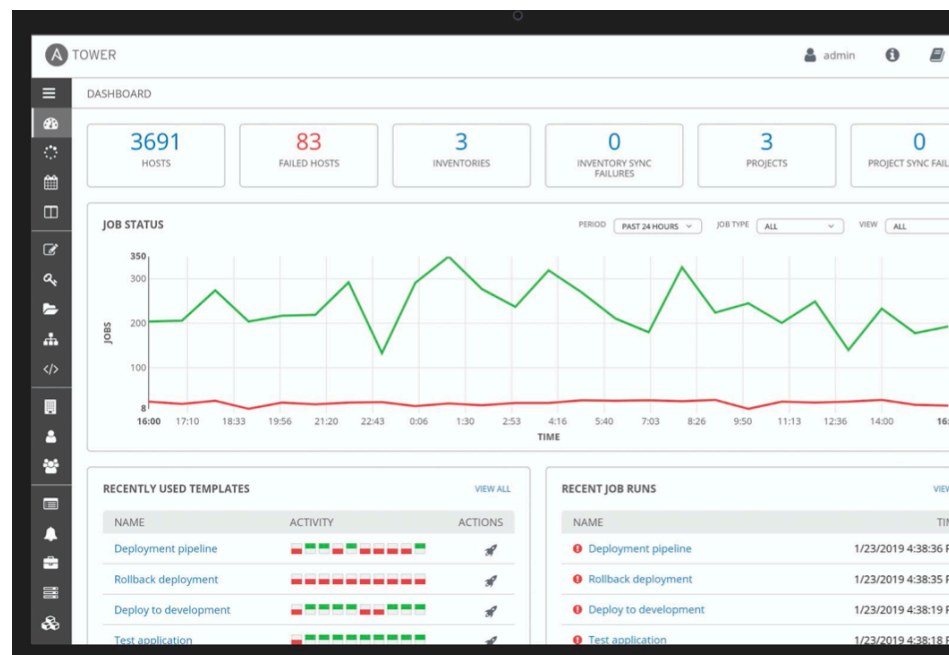
Ansible Tower



ANSIBLE

Ansible Tower is a UI and RESTful API allowing you to scale IT automation, manage complex deployments and speed productivity.

- Role-based access control
- Deploy entire applications with push-button deployment access
- All automations are centrally logged
- Powerful workflows match your IT processes



Ansible Tower - Projects



Project

A project is a logical collection of Ansible Playbooks, represented in Ansible Tower.

You can manage Ansible Playbooks and playbook directories by placing them in a source code management system supported by Ansible Tower, including Git, Subversion, and Mercurial.

The screenshot shows the Ansible Tower interface. At the top left, there is a logo with the letter 'A' in a circle followed by the word 'TOWER'. Below this is a dark sidebar menu with a hamburger icon at the top. The menu items are: VIEWS (Dashboard, Jobs, Schedules, My View) and RESOURCES (Templates, Credentials, Projects). The 'Projects' item is highlighted. The main content area is titled 'PROJECTS' and features a sub-header 'PROJECTS 6' with a badge. Below this is a search bar with the text 'SEARCH' and a magnifying glass icon. The main area displays a list of projects, each with a radio button, a name, and a 'MANUAL' status tag. The visible projects are 'AIX', 'Azhar_Project', and 'General'.

Project Name	Status
<input type="radio"/> AIX	MANUAL
<input type="radio"/> Azhar_Project	MANUAL
<input type="radio"/> General	MANUAL

Ansible Tower - Credentials



Credentials

Credentials are utilized by Ansible Tower for authentication with various external resources:

- Connecting to remote machines to run jobs
- Syncing with inventory sources
- Importing project content from version control systems
- Connecting to and managing devices

The screenshot shows the Ansible Tower web interface. On the left is a dark sidebar with navigation options: VIEWS (Dashboard, Jobs, Schedules, My View), RESOURCES (Templates, Credentials, Projects, Inventories, Inventory Scripts), and ACCESS (Organizations). The 'Credentials' option is highlighted. The main content area is titled 'CREDENTIALS' and shows a table of 6 credentials. At the top of the table is a search bar with a 'KEY' button. The table has columns for NAME, KIND, and OWNERS.

NAME ^	KIND	OWNERS
Ansible_Tower_localhost	Machine	azhar , Azhar_Organization
Azhar_LPAR_Credential	Machine	azhar
git-hub	Source Control	admin , Lab Services UK&I
PowerVC_Credential	OpenStack	azhar , Azhar_Organization
PowerVC (ibm-default)	OpenStack	admin , Lab Services UK&I
root	Machine	admin

Ansible Tower - Inventory



Inventory

Inventory is a collection of hosts clients (just like the with the engine) with associated data and groupings that Ansible Tower can connect to and manage.

- Hosts (nodes)
- Groups
- Inventory-specific data (variables)
- Static or dynamic sources

The screenshot shows the Ansible Tower web interface. The top navigation bar includes the 'TOWER' logo and a user profile icon. A dark sidebar on the left contains a menu with sections: VIEWS (Dashboard, Jobs, Schedules, My View), RESOURCES (Templates, Credentials, Projects, Inventories, Inventory Scripts), and ACCESS. The main content area is titled 'INVENTORIES' and features a search bar with a 'KEY' button. Below the search bar is a table listing various inventory sources.

NAME	TYPE	ORGANIZATION
Ansible_PowerVC_Inventory	Inventory	Azhar_Organization
inventory_localhost	Inventory	Azhar_Organization
Old domain VMs	Smart Inventory	Lab Services UK&I
PowerVC_Inventory	Inventory	Lab Services UK&I
VUG_demo	Inventory	Lab Services UK&I

Ansible Tower - Templates

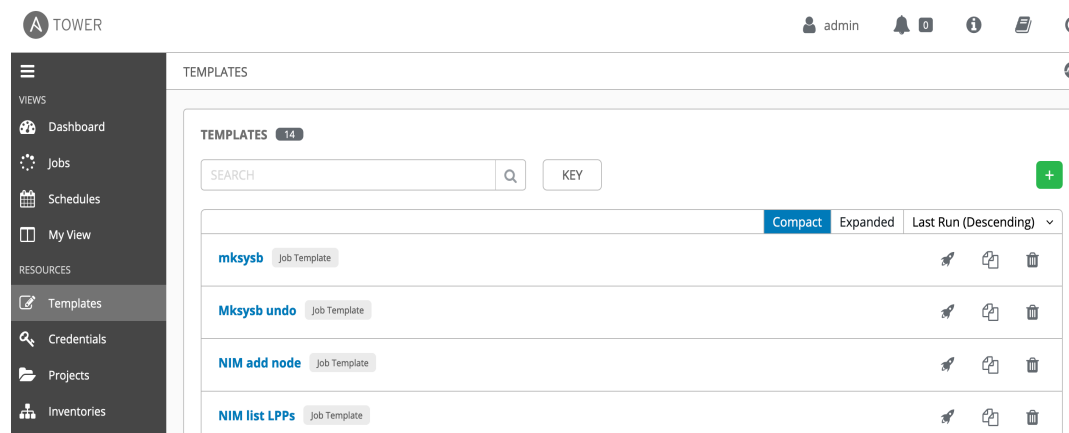


Job Templates

Everything in Ansible Tower revolves around the concept of a Job Template. Job Templates allow Ansible Playbooks to be controlled, delegated and scaled for an organization. Job templates also encourage the reuse of Ansible Playbook content and collaboration between teams.

A Job Template requires:

- An Inventory to run the job against
- A Credential to login to devices.
- A Project which contains Ansible Playbooks



PowerVC Updates – version 1.4.4.1



- Active backup of PowerVC using powervc-backup, even when operations are in progress.
 - Image sharing across projects. Set the image visibility of PowerVC images from private to public.
 - View user information for operations within a project.
 - Configuring initiator port groups (IPGs) to define the set of VIOS ports to be used for attachment when using NPIV.
 - Deploy a virtual machine using existing boot and data volumes.
 - Change (and retain) the availability priority of a virtual machine.
 - Pin virtual machines using soft or hard options.
 - Recall virtual machines to the source host following a maintenance mode activation or automate remote restart.
 - Pure Storage technical preview as an integrated driver via the CLI.
 - Infinidat storage as a registered pluggable driver.
 - IBM PowerVC FlexVolume Driver version 1.0.2 is available.
 - Red Hat® OpenShift Container Platform 4.3 support with PowerVC.
-
- PPC64 support to be withdrawn – you can migrate to ppc64le or x86_64.
 - Guest OS support for SLES BE and Ubuntu 14.0.4 has been withdrawn.
 - Cloud-init versions update to 19.1 to support SLES 15, RHEL 8.0 and 8.1.

PowerVC Updates – sharing images across projects



Image sharing across projects. We can now set the image visibility of PowerVC images from private to public. Setting the visibility of an image to public makes it available or shareable across the projects.

The screenshot shows the PowerVC interface for an image named "IBMi-image7_3". The image is currently set to "private" visibility. A dialog box is open, prompting the user to change the visibility to "public". The dialog box contains the following text:

Select the checkbox below to change the visibility of image **IBMi-image7_3**. Setting image to public will make it visible to the project administrators.

Set image **IBMi-image7_3** to public.

Buttons: OK, Cancel

Image: IBMi-image7_3			
Refresh	Deploy	Delete	Export
Information			
Name:	IBMi-image7_3		
State:	Active		
ID:	7bc7fd7c-cee1-49ea-b91e-1691		
Description:			
Visibility:	private		
Captured VM:	IBMi-image7.2-273a30e0-00000		
Created:	13 May 2020 13:12:16 British Su		
Last updated:	13 May 2020 13:12:27 British Su		
Specifications			

PowerVC Updates – user information in operation





View user information for operations within a project – The Message tab now lists user information for every operation along with other details. This helps to understand when a user has performed a specific operation. The user name is listed as system for operations that are performed by PowerVC services internally.

Messages

 Refresh  Delete  Delete All

Filter 

 No filter applied

Type	Timestamp	Message	User
		4a0f-94e6-80d2677f11e8 . Monitor deployment progress on the Virtual Machines page.	
 Information	16/03/2020 16:10	The status of deploy request b49d2c4d-280a-4a23-8c86-5269e8894b1e is completed.	spurwayd
 Success	16/03/2020 16:10	The volume AIX_7.2_TL3_S-436195f0-00000419-boot-0 has been successfully created.	spurwayd
 Information	16/03/2020 16:15	Deploy of virtual machine AIX 7.2 TL3 SP3 DB2 v11.5 on host Yorkshire was successful.	spurwayd

PowerVC Updates – Initiator Port Groups



With IPG, a subset of ports can be selected for a given volume attachment. The ports in a given IPG should match all of the VIOS and fabric settings of the storage connectivity group. For example, if you have selected VIOS redundancy of minimum 2 for a storage connectivity group, then the IPG should have ports from 2 Virtual I/O Servers for a host.

Storage Connectivity Group: Single_path

 Refresh |  Edit |  Delete

Overview

Health

Members


VIOS	Host	State	RMC State	Total FC Ports
Yorkshire-VIOS1	Yorkshire	● Running	● Active	2

Total: 1

Fabrics

Initiator Port Groups

Filter

Initiator Port Group Name	Host:VIOS	Fibre Channel Port	WWPN
Yorkshire_VIOS1_FCS1	 Yorkshire:Yorkshire-VIOS1	fcs1	21000024ff0ffe17

PowerVC Updates – Deploy using existing volumes



We can now choose to create an image without any volumes attached. While deploying this image, you can select the existing volumes in the Deploy template. This prevents cloning of volumes and avoids any data redundancy.

Create an image with no LUN listed

Create Image from Volumes

* Image name:

RH_CoreOS_4.3.18

Description:

Red Hat CoreOS v 4.3.18

* Hypervisor type:

PowerVM

* Operating system:

RHEL

Endianness:

Big Endian

Little Endian

 Add Volume  Remove Volume |  Move Up  Move Down

Order	Name	Size (GB)	Storage Template	Storage Provider
Select the volumes that will compose the created image. The boot set must contain at least one volume.				

Create

Cancel

PowerVC Updates – Deploy using existing volumes



We can now choose to create an image without any volumes attached. While deploying this image, you can select the existing volumes in the Deploy template. This prevents cloning of volumes and avoids any data redundancy.

Deploy 'blank' image and attached the existing LUN. This will create the VM and attach chosen LUNs.

Deploy RH_CoreOS_4.3.18

▼ Image Volumes



 Edit Storage Template

Order	Name	Size (GB)	State	Health	Storage Template	Storage Provider	Boot Set
No items to display							

Total: 0 Selected: 0

▼ Additional Volumes

 Add Volume  Remove Volume |  Move Up  Move Down

Order	Name	Size (GB)	State	Storage Template	Storage Provider	Description	Boot Set	Sharing Enabled
1	 RH_CoreOS_4.3	18	 Available	V7K1_RackH primary pool	V7K1_RackH	Red Hat CoreOS v4.3.18 boot disk	Yes	No

PowerVC Updates – Pin VMs



You can choose to soft pin or hard pin a virtual machine to the host where it is currently running. When you 'Soft Pin' a virtual machine for high availability, PowerVC automatically migrates the virtual machine back to the original host once the host is back to operating state. The 'Hard Pin' option helps you to restrict the movement of the virtual machine during remote restart, automated remote restart, DRO and live partition migration.

The screenshot shows the PowerVC interface for a virtual machine named "VM: Lab-RM-GUI-31". The "Overview" tab is selected, and a dialog box is displayed in the foreground. The dialog box contains the following text:

Select an option below to pin virtual machine **Lab-RM-GUI-31** to the current host.
[? Learn more...](#)

Soft Pin Hard Pin

At the bottom of the dialog box, there are two buttons: "OK" and "Cancel".

In the background, the "Overview" tab shows the following information:

- Excluded from automated remote restart:
- Storage connectivity group:
- Operating system:
- Power state:
- Task state:
- Hypervisor host name:
- Hypervisor partition name:
- Enforce affinity score check during migrations: No
- Secure boot: Disabled
- Pin state: Unpinned

PowerVC Updates – Migrate inactive VMs



We now have the ability to migrate inactive VMs

The screenshot displays the PowerVC interface for a virtual machine named "VM: ITSO-A-1". The "Overview" tab is active, showing various actions: Refresh, Start, Stop, Restart, Delete, Capture, Resize, Migrate, and Edit Details. The "Migrate" button is highlighted with a red dashed box. Below the actions, the "Information" section provides details about the VM:

Information	
Name:	ITSO-A-1
Description:	VM RM HA - Test AIX VM
State:	Shutoff
Health:	OK
Host:	Lancashire
Owner:	
Created:	4 January 2019 15:39:20 Greenv
Expiration date:	None
Memory:	4 GB (Dedicated)

A "Migrate" dialog box is open, prompting the user to select a destination host for the virtual machine "ITSO-A-1". The dialog includes a "Host" dropdown menu currently set to "Selected by placement policy" and "Migrate" and "Cancel" buttons.

PowerVC Updates – Recall VMs



PowerVC automatically recalls virtual machines that were moved during host maintenance mode or automated remote restart to the source host that has recall option enabled.

The screenshot shows the PowerVC management interface for a host named "Yorkshire". The "Overview" tab is selected, displaying various host capabilities. A dialog box is overlaid on the interface, prompting the user to enable the "Recall Virtual Machines" feature for the host. The dialog box contains the following text:

Select the checkbox below to recall virtual machines for the host **Yorkshire**.

Enable Recall Virtual Machines feature for the host **Yorkshire**

Buttons for "OK" and "Cancel" are visible at the bottom of the dialog box.

In the background, the "Recall enabled:" property is currently set to "No" and is highlighted with a red dashed box. Other visible properties include "Memory region size (MB): 256", "Live Partition Mobility capable:", "Active Memory Expansion capable:", "Physical Page Table ratio capable:", "Dedicated processor capable:", "Remote restart enabled:", "Excluded from automated remote restart:", "Excluded from DRO:", and "Enterprise pool member:".

PowerVC Updates – Availability Priority



PowerVC allows you to update "Availability Priority" setting of a virtual machine and retains it even after live partition migration (LPM) or remote restart of virtual machines to a new host.

The screenshot shows the PowerVC interface for a virtual machine named "Lab-VMRM-50". The "Specifications" section is visible, showing various settings. A dialog box is overlaid on the interface, prompting the user to enter the availability priority for the virtual machine. The dialog box contains a question mark icon, the text "Enter availability priority for virtual machine Lab-VMRM-50", a text input field containing the value "127", and "OK" and "Cancel" buttons. The "Availability priority" setting in the specifications table is highlighted with a red dashed box, showing the value "127" and a pencil icon.

Specifications	
Minimum memory (M)	
Maximum memory (MB):	18,432
Minimum processors:	1
Maximum processors:	4
Availability priority:	127
Processor mode:	Shared



ANSIBLE



Stuart Cunliffe
email: s_cunliffe@uk.ibm.com
Twitter: [@StuCunliffe](https://twitter.com/StuCunliffe)
slack: [@Stu Cunliffe](#)