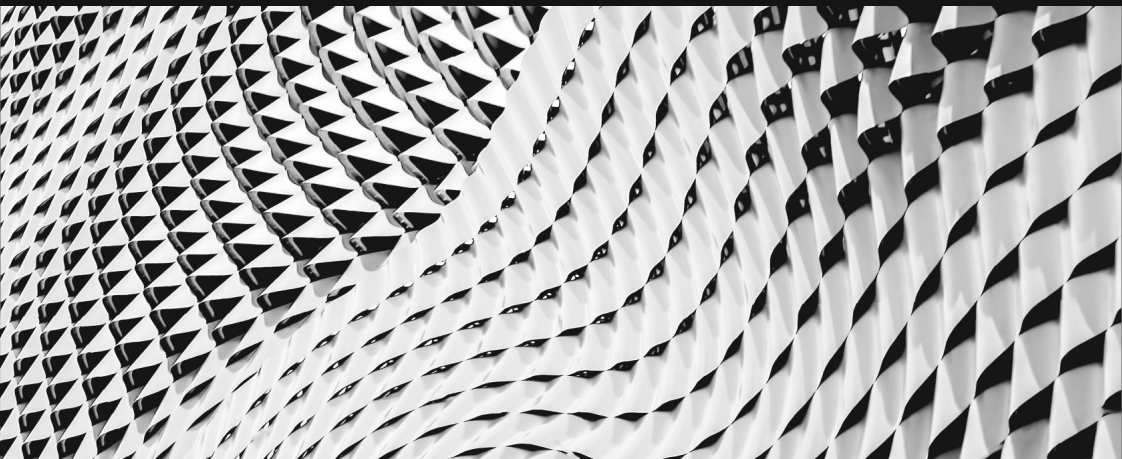


Ansible Best Practices

How to write, how to execute, and how to use in real life

Create



Treat your Ansible content like code

- Version control your Ansible content
- Iterate
 - Start with a basic playbook and static inventory
 - Refactor and modularize later

Do it with style

- Create a style guide for consistency:
 - Tagging
 - Whitespace
 - Naming of Tasks, Plays, Variables, and Roles
 - Directory Layouts
- Enforce the style
- Nice example: openshift-ansible Style Guide
example: <https://goo.gl/JfWBcW>



**CODE MUST BE
ORGANIZED**

USE GIT!

Use a clear structure

```
site.yml                # master playbook, calling others
webservers.yml         # playbook for webserver tier
deployonce.yml        # separate playbook for single-shot tasks
inventories/
  production/         # different stages via inventory
    hosts             # inventory file for production servers
    group_vars/
    host_vars/
  london/            # additional, alternative grouping if useful
roles/
  requirements.yml    # includes roles from some other place
  common/            # base line, company wide configuration
  webtier/
```

Start with one Git repository - but when it grows, use multiple!

At the beginning: put everything in one Git repository

In the long term:

- One Git repository per role
- Dedicated repositories for completely separated teams / tasks

New to git? Get your cheat sheet here: <https://opensource.com/downloads/cheat-sheet-git>

SO, WHAT DO WE HAVE?



Give inventory nodes human-meaningful names rather than IPs or DNS hostnames.

10.1.2.75

10.1.5.45

10.1.4.5

10.1.0.40

db1 ansible_host=10.1.2.75

db2 ansible_host=10.1.5.45

db3 ansible_host=10.1.4.5

db4 ansible_host=10.1.0.40



w14301.acme.com

w17802.acme.com

w19203.acme.com

w19304.acme.com

web1 ansible_host=w14301.acme.com

web2 ansible_host=w17802.acme.com

web3 ansible_host=w19203.acme.com

web4 ansible_host=w19203.acme.com

Group hosts for easier inventory selection and less conditional tasks -- the more the better.

```
[db]  
db[1:4]
```

```
[web]  
web[1:4]
```

```
[east]  
db1  
web1
```

```
db3  
web3
```

```
[west]  
db2  
web2
```

```
db4  
web4
```

```
[dev]  
db1  
web1
```

```
[testing]  
db3  
web3
```

```
[prod]  
db2  
web2  
db4  
web4
```

Use dynamic sources where possible. Either as a single source of truth - or let Ansible unify multiple sources.

- Stay in sync automatically
- Reduce human error
- No lag when changes occur
- Let others manage the inventory



VARIABLES

**JUST WORDS,
RIGHT?**

Proper variable names can make plays more readable and avoid variable name conflicts

```
a: 25
```

```
data: ab
```

```
data2: abc
```

```
id: 123
```

```
apache_max_keepalive: 25
```

```
apache_port: 80
```

```
tomcat_port: 8080
```

Avoid collisions and confusion by adding the role name to a variable as a prefix.

```
apache_max_keepalive: 25  
apache_port: 80  
tomcat_port: 8080
```

Know where your variables are

- Find the appropriate place for your variables based on what, where and when they are set or modified
- Separate logic (tasks) from variables and reduce repetitive patterns
- Do not use every possibility to store variables - settle to a defined scheme and as few places as possible



**MAKE YOUR PLAYBOOK
READABLE**

NO!

- name: install telegraf
yum: name=telegraf-{{ telegraf_version }} state=present update_cache=yes
notify: restart telegraf
- name: start telegraf
service: name=telegraf state=started

Yes!

- name: install telegraf
yum:
 - name: “telegraf-{{ telegraf_version }}”
 - state: present
 - update_cache: yes
 - enablerepo: telegrafnotify: restart telegraf

- name: start telegraf
service:
 - name: telegraf
 - state: started

Exhibit A

- hosts: web

tasks:

- yum:

name: httpd
state: latest

- service:

name: httpd
state: started
enabled: yes

PLAY [web]

TASK [setup]

ok: [web1]

TASK [yum]

ok: [web1]

TASK [service]

ok: [web1]

Exhibit B

```
- hosts: web                                PLAY [install and starts apache]
  name: installs and starts apache          *****

tasks:                                       TASK [setup]
  - name: install apache packages          *****
    yum:                                    ok: [web1]
      name: httpd
      state: latest

  - name: starts apache service            TASK [install apache packages]
    service:                               *****
      name: httpd                          ok: [web1]
      state: started
      enabled: yes                         TASK [starts apache service]
                                           *****
                                           ok: [web1]
```



POWERFUL BLOCKS

Blocks can help in organizing code, but also enable rollbacks or output data for critical changes.

```
- block:  
  copy:  
    src: critical.conf  
    dest: /etc/critical/crit.conf  
  service:  
    name: critical  
    state: restarted  
rescue:  
  command: shutdown -h now
```

Run





**PROPER
LAUNCHING**

Don't just start services -- use smoke tests

```
- name: check for proper response
  wait_for:
    port: 80
    timeout: 300
    state: present
    search_regex: "Hello World"
```

Ansible provides multiple switches for command line interaction and troubleshooting.

```
-vvvv  
--step  
--check  
--diff  
--start-at-task
```

Ansible has switches to show you what will be done

Use the power of included options:

`--list-tasks`

`--list-tags`

`--list-hosts`

`--syntax-check`

If there is a need to launch something without an inventory -
just do it!

- For single tasks - note the comma:

```
ansible all -i neon.qxyz.de, -m service -a  
"name=redhat state=present"
```

- For playbooks - again, note the comma:

```
ansible-playbook -i neon.qxyz.de, site.yml
```

THE RIGHT TOOLS



Try to avoid the command module - always seek out a module first

```
- name: add user
  command: useradd appuser

- name: install apache
  command: yum install httpd

- name: start apache
  shell: |
    service httpd start && chkconfig
    httpd on
```

```
- name: add user
  user:
    name: appuser
    state: present

- name: install apache
  yum:
    name: httpd
    state: latest

- name: start apache
  service:
    name: httpd
    state: started
    enabled: yes
```

If managed files are not marked, they might be overwritten accidentally

- Label template output files as being generated by Ansible
- Use the `ansible_managed**` variable with the comment filter

```
{{ ansible_managed | comment }}
```

COLLECTIONS AND GALAXIES



Roles enable you to encapsulate your operations.

- Like playbooks -- keep roles purpose and function focused
- Store roles each in a dedicated Git repository
- Include roles via roles/requirements.yml file, import via ansible-galaxy tool
- Limit role dependencies

Get collections from Galaxy, but be careful and adopt them to your needs

- Collections can contain roles, and other other code like modules as well
- Galaxy provides thousands of roles and collections
- Quality varies drastically
- Take them with a grain of salt
- Pick trusted or well known authors

ACCESS RIGHTS



Root access is harder to track than sudo - use sudo wherever possible

- Don't run as root
- For login and security reasons often request non-root access
- Use become method - so Ansible scripts are executed via sudo (sudo is easy to track)
- Best: create an Ansible only user
- Don't try to limit sudo rights to certain commands - Ansible does not work that way!

**DEBUG YOUR
PROBLEM**



Check logging on target machine

```
ansible-node sshd[2395]: pam_unix(sshd:session): session
  opened for user bob by (uid=0)
ansible-node ansible-yum[2399]: Invoked with name=['httpd']
  list=None install_repoquery=True conf_file=None
  disable_gpg_check=False state=absent disablerepo=None
  update_cache=False enablerepo=None exclude=None
```

How to keep the code executed on the target machine

Look into the logging of your target machine

```
$ ANSIBLE_KEEP_REMOTE_FILES=1 ansible target-node -m yum  
-a "name=httpd state=absent"
```

Execute with:

```
$ /bin/sh -c 'sudo -u $SUDO_USER /bin/sh -c  
"/usr/bin/python /home/bob/.ansible/tmp/..."'
```

Debugging tasks can clutter the output, apply some housekeeping

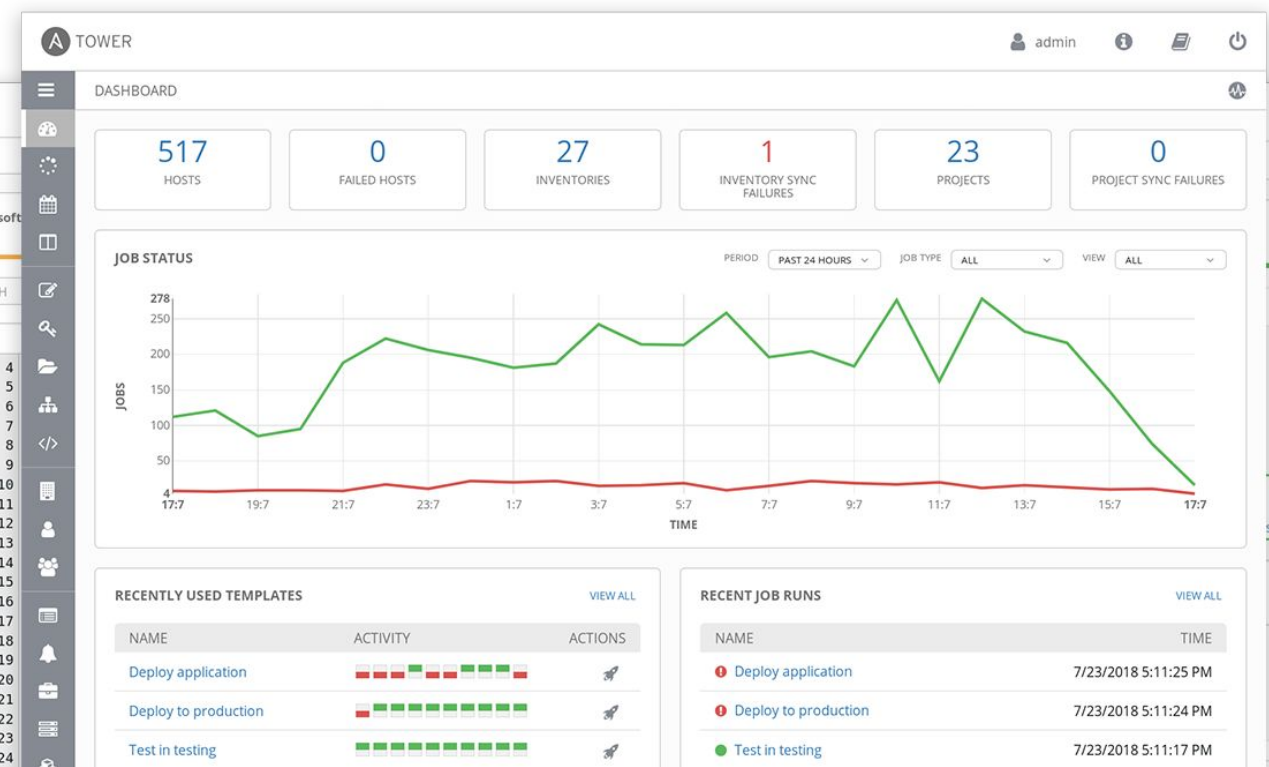
- name: Output debug message
debug:
 msg: "This always displays"
- name: Output debug message
debug:
 msg: "This only displays with ansible-playbook -vv+"
 verbosity: 2

GET TOWER TO ADOPT ANSIBLE IN YOUR DATA CENTER

Scale

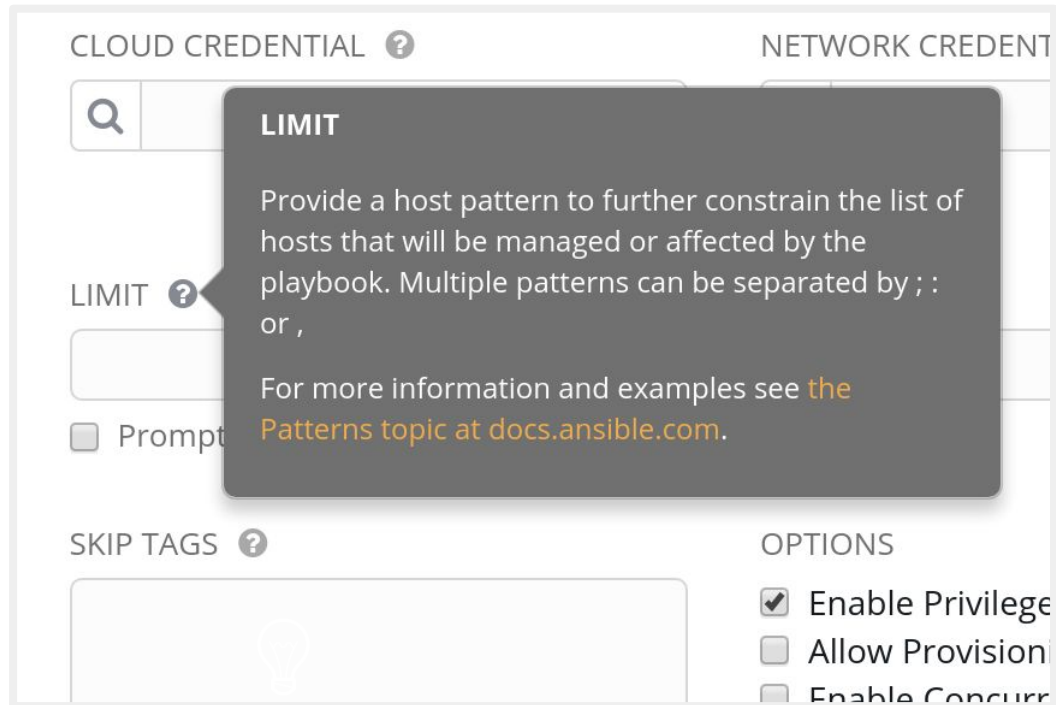


Simple: Use Tower.



- Tower was developed with Ansible in mind
 - Extends the limits of Ansible to meet enterprise needs:
- Scalability, API, RBAC, audits, etc.

Tower has inbuilt help

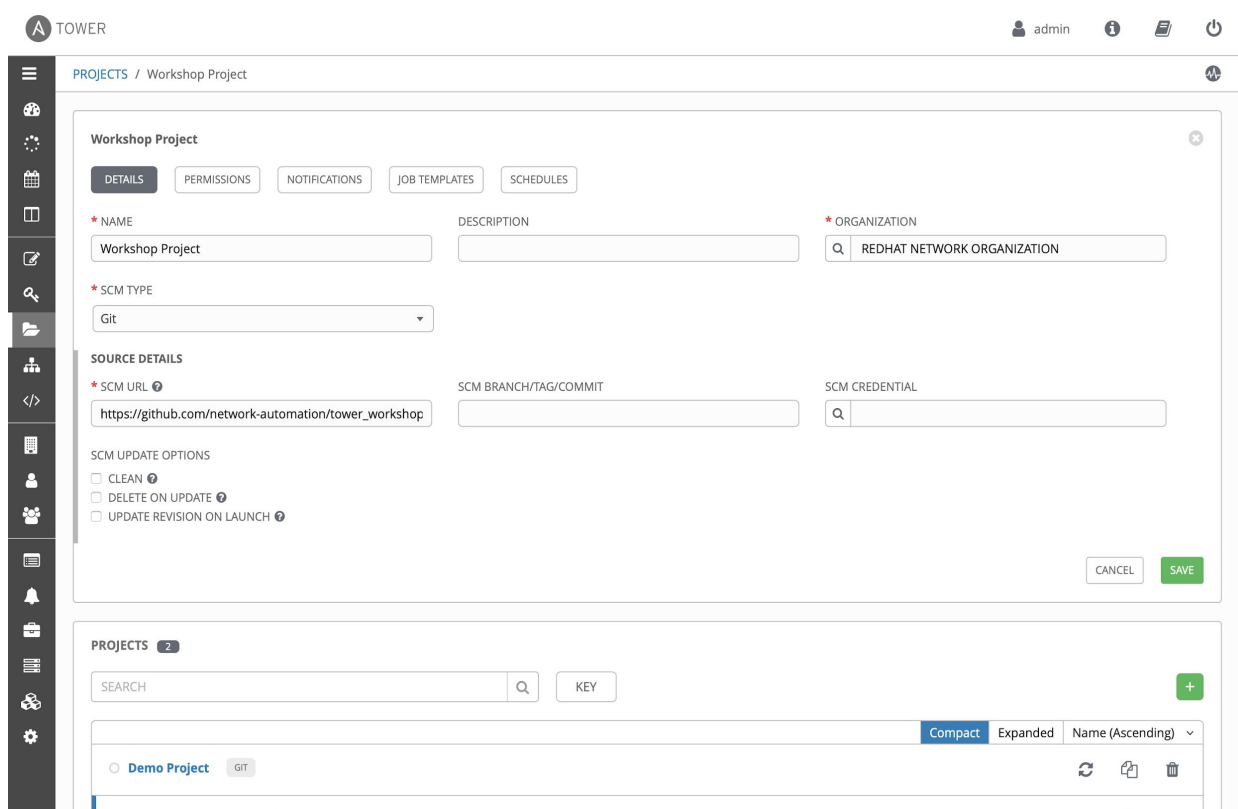


- Tower provides in-program help via questionmark bubbles
- Can include examples or links to further docs



BRANCHES, ANYONE?

Tower can import a repository multiple times with different branches



- Use feature or staging branches in your Git
- Import them all separately, address them separately
- Useful for testing of new features but also to move changes through stages

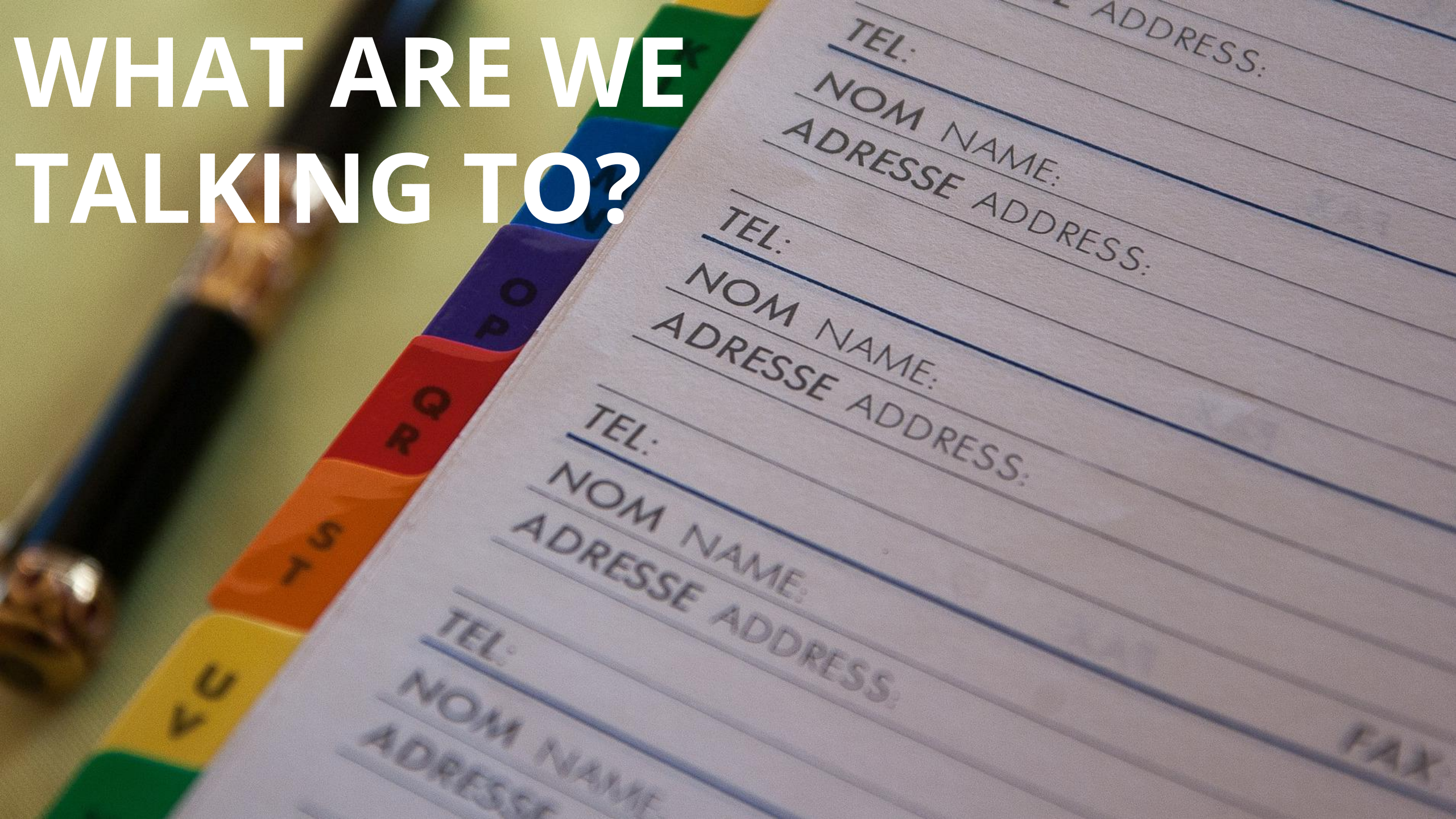


MANY, MANY ROLES

Tower automatically imports Roles during Project update

- Do not copy roles into your playbook repository, just create a roles/requirements.yml
- Tower will automatically import the roles during Project installation
- Mix roles from various sources
- Fix version in roles/requirements.yml to have auditable environment!

WHAT ARE WE TALKING TO?



TEL:

ADRESSE ADDRESS:

NOM NAME:

ADRESSE ADDRESS:

TEL:

NOM NAME:

ADRESSE ADDRESS:

TEL:

NOM NAME:

ADRESSE ADDRESS:

TEL:

NOM NAME:

ADRESSE

FAX:

Use dynamic & smart inventories

The screenshot shows the Tower web interface for the 'Workshop Inventory' page. The page is titled 'Workshop Inventory' and has several tabs: DETAILS, PERMISSIONS, GROUPS, HOSTS (selected), SOURCES, and COMPLETED JOBS. Below the tabs is a search bar with a 'KEY' button and a 'RUN' button. The main content area is divided into two columns: 'HOSTS' and 'RELATED GROUPS'. The 'HOSTS' column lists five hosts: ansible, rtr1, rtr2, rtr3, and rtr4, each with an 'ON' status indicator. The 'RELATED GROUPS' column shows the groups associated with each host: ansible is associated with 'control'; rtr1 with 'cisco' and 'dc1'; rtr2 with 'arista' and 'dc2'; rtr3 with 'dc1' and 'juniper'; and rtr4 with 'arista' and 'dc2'. At the bottom of the page, there are additional search fields and a table header with columns for NAME, TYPE, and ORGANIZATION.

- Combine multiple inventory types
- Let Tower take care of syncing and caching
- Use smart inventories to group nodes



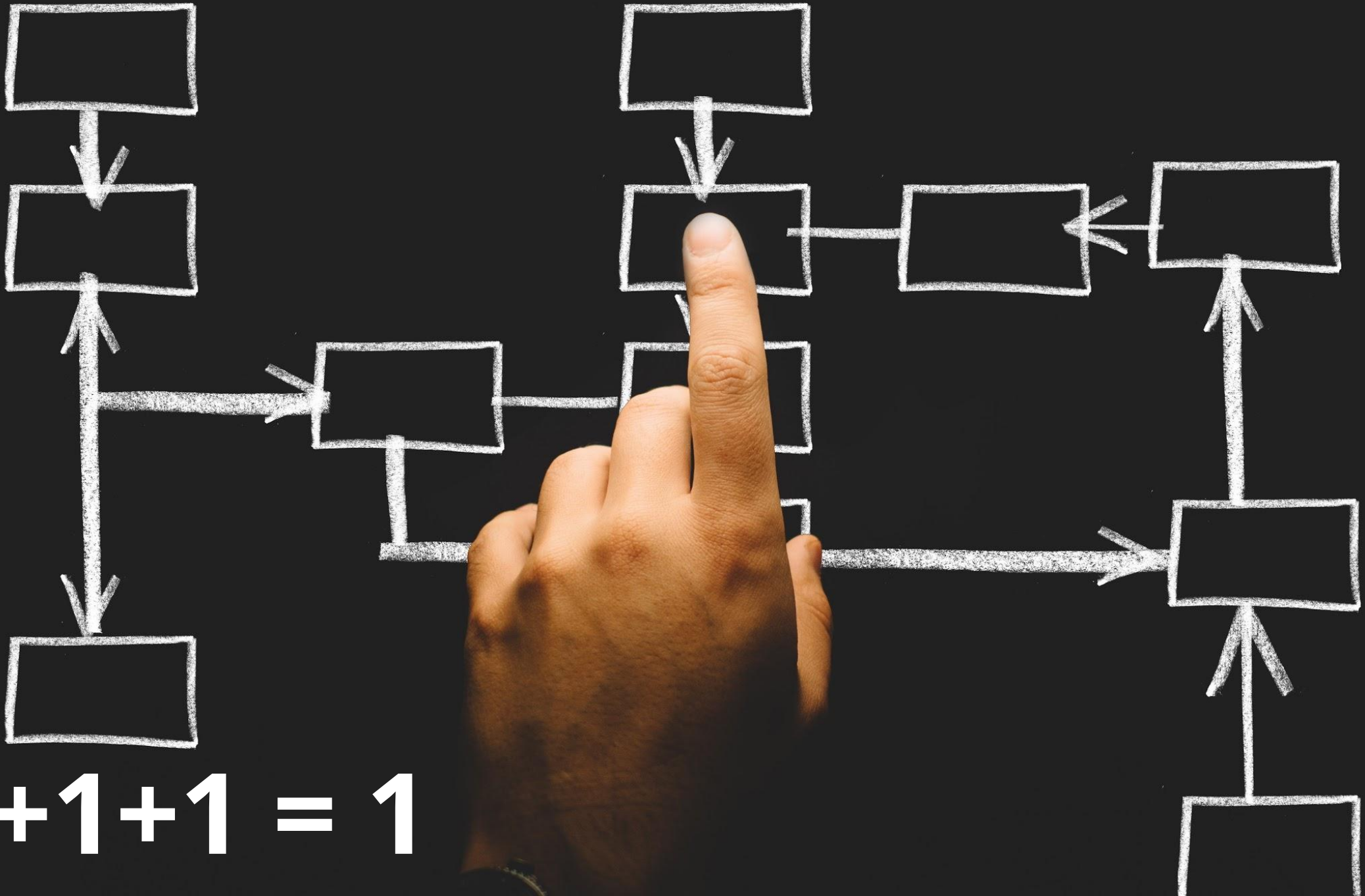
DOING GOOD JOBS

Tower job templates provide multiple options - use them wisely

The screenshot shows the Tower web interface for configuring a job template named 'Azure Linux VM Spinup'. The interface includes a sidebar with navigation icons and a top navigation bar with the 'TEMPLATES / Azure Linux VM Spinup' breadcrumb. The main content area is divided into several sections:

- DETAILS** (selected), PERMISSIONS, NOTIFICATIONS, COMPLETED JOBS, SCHEDULES, and EDIT SURVEY buttons.
- * NAME**: Input field containing 'Azure Linux VM Spinup'.
- DESCRIPTION**: Empty input field.
- * JOB TYPE**: Dropdown menu set to 'Run'.
- * INVENTORY**: Input field containing 'Prod', with a 'PROMPT ON LAUNCH' checkbox.
- * PROJECT**: Input field containing 'fest19-demo'.
- * PLAYBOOK**: Input field containing 'azure_spinup.yml'.
- CREDENTIAL**: Input field containing 'Azure-Service-Principal', with a 'PROMPT ON LAUNCH' checkbox.
- FORKS**: Input field containing '0'.
- LIMIT**: Empty input field.
- * VERBOSITY**: Dropdown menu set to '0 (Normal)', with a 'PROMPT ON LAUNCH' checkbox.
- JOB TAGS**: Empty input field, with a 'PROMPT ON LAUNCH' checkbox.
- SKIP TAGS**: Empty input field.
- LABELS**: Empty input field.
- INSTANCE GROUPS**: Input field containing 'Q'.
- JOB SLICING**: Input field containing '1'.
- TIMEOUT**: Input field containing '0'.
- SHOW CHANGES**: Input field set to 'OFF', with a 'PROMPT ON LAUNCH' checkbox.
- OPTIONS**: A list of checkboxes: 'ENABLE PRIVILEGE ESCALATION', 'ALLOW PROVISIONING CALLBACKS', 'ENABLE CONCURRENT JOBS', and 'USE FACT CACHE', all of which are currently unchecked.
- EXTRA VARIABLES**: A section with 'YAML' and 'JSON' tabs.

- Keep jobs simple, focussed - as playbooks or roles
- Add labels to them to better filter
- For idempotent jobs, create "check" templates as well - and let them run over night
- Combine with notifications - and get feedback when a "check" failed



$$1 + 1 + 1 = 1$$

Multiple playbooks can be combined into one workflow

JOBS / 363 - Wave Workflow Example

The screenshot shows a workflow management interface. On the left, a 'DETAILS' sidebar provides information about the job: 'STATUS: Running', 'STARTED: 10/10/2019 10:40:57 AM', 'FINISHED: Not Finished', 'TEMPLATE: Wave Workflow Example', and 'LAUNCHED BY: admin'. Below this, there are tabs for 'EXTRA VARIABLES' in 'YAML' and 'JSON' format, with an 'EXPAND' button. The main area displays a flowchart titled 'Wave Workflow Example' with 'TOTAL NODES: 7' and 'ELAPSED: 00:00:52'. The flowchart starts with a blue square node that branches into four parallel tasks: 'PS - Workflow', 'Configure Red Hat Enterprise...', 'Configure Firewall', and 'Configure Network Fabric'. These four tasks converge into a single 'PS workshop source' node. From there, the flow goes to a 'Rollback changes' node, and finally to a 'Check connectivity' node.

- Simple jobs, complex workflows
- React to problems via workflow
- Combine playbooks of different teams, different repositories
- Re-sync inventories during the play

A blackboard with a white border, resting on a wooden surface. The words 'WHO', 'HOW', 'WHEN', 'WHERE', 'WHAT', and 'WHY' are written in white chalk. A large, stylized question mark is drawn in the center of the board, with the words arranged around it.

DO ASK PROPER QUESTIONS

Use surveys to get variable values

* PROMPT

Please provide data

DESCRIPTION

data

* ANSWER VARIABLE NAME ⓘ

data

* ANSWER TYPE

Text

MINIMUM LENGTH

0

MAXIMUM LENGTH

1024

DEFAULT ANSWER

data

- Use good, meaningful variable names
- Provide a default choice
- Multiple choice > free text
- If answer not required - do you really need it at all?



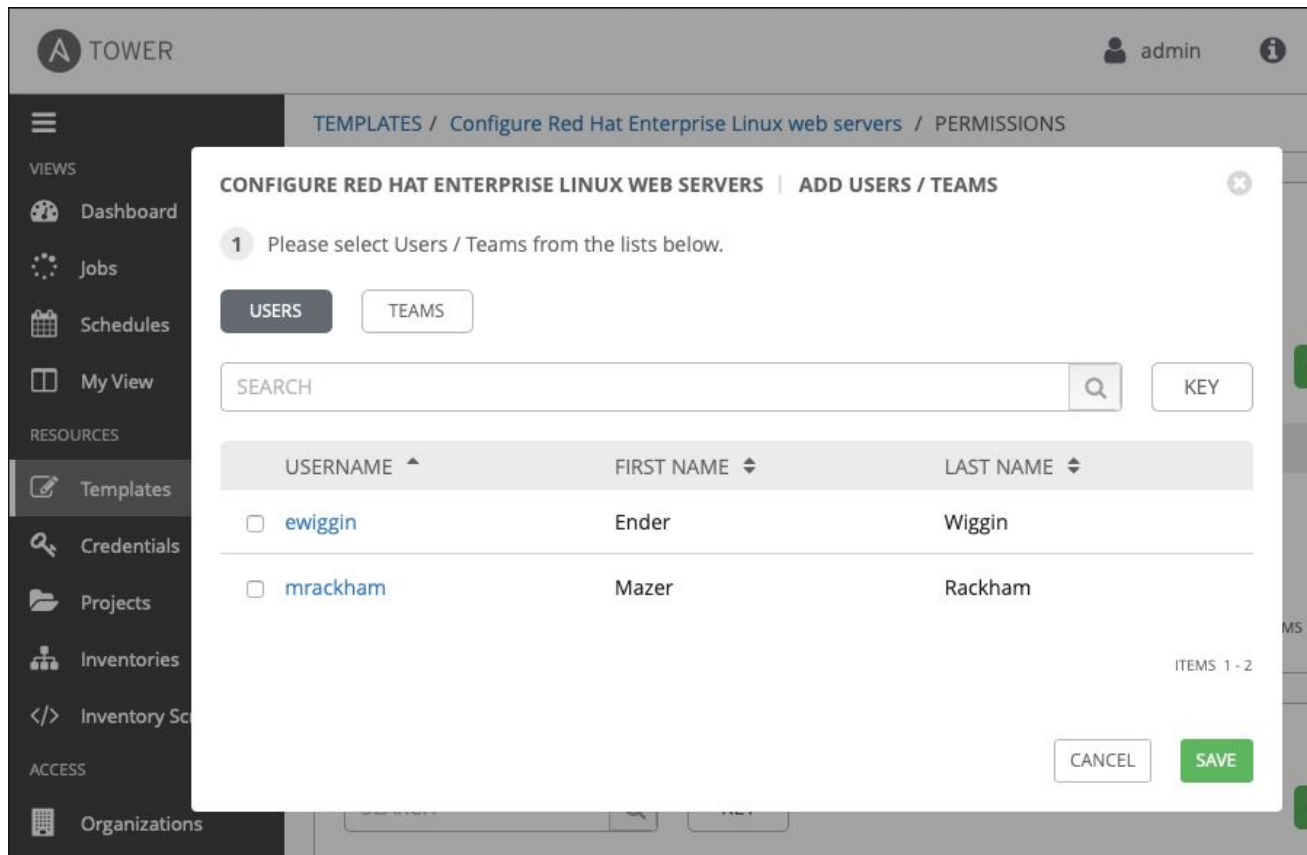
I ❤️
ANSIBLE

```
cat baby.yml
---
- name: baby
  hosts: parental_units
  roles:
    - eat
    - sleep
    - poop
    - love
```

Even I can run
a playbook

**A POWERFUL
TEAM**

Tower provides tenants, teams, and users - use them for separation



- Provide automation to others without exposing credentials
- Let others only see what they really need
- Use personal view instead of full Tower interface



**ONE KEY TO RULE
THEM ALL ...**

Tower credentials should only be used by Tower – not by others

The screenshot shows the 'Workshop Credential' configuration page in the Tower web interface. The page is titled 'Workshop Credential' and has two tabs: 'DETAILS' and 'PERMISSIONS'. The 'DETAILS' tab is active. The form includes the following fields:

- * NAME:** Workshop Credential
- DESCRIPTION:** (empty)
- ORGANIZATION:** REDHAT NETWORK ORGANIZATION
- * CREDENTIAL TYPE:** Machine
- TYPE DETAILS:**
 - USERNAME:** ec2-user
 - PASSWORD:** (empty)
 - Prompt on launch
- SSH PRIVATE KEY:** ENCRYPTED (with a search icon)
- SIGNED SSH CERTIFICATE:** HINT: Drag and drop private file on the field below. (with a search icon)
- PRIVATE KEY PASSPHRASE:** (empty)
- Prompt on launch
- PRIVILEGE ESCALATION METHOD:** (empty)
- PRIVILEGE ESCALATION USERNAME:** (empty)

- Set up a separate user and password/key for Tower
- That way, automation can easily be identified on target machines
- The key/password can be ridiculously ~~complicated~~ secure
- Store key/password in a safe for emergencies

NOTIFY YOURSELF!



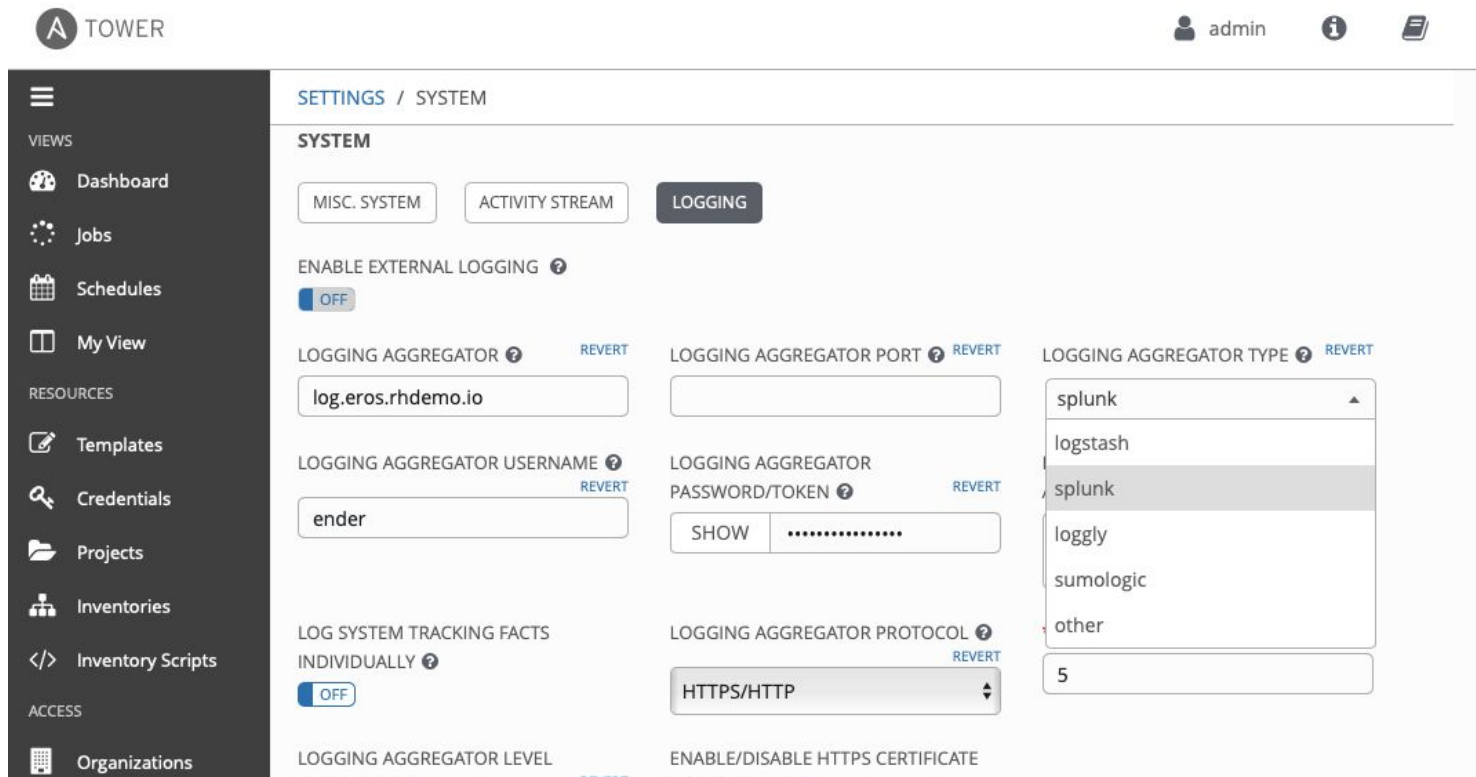
Tower can send notifications if a job succeeds, fails or always - as mail, IRC, web hook, and so on

- Let Tower notify you and your team if something breaks
- Send mails/web-hooks automatically to a ticket systems and monitoring if there is a serious problem



LOGS, ANYONE?

Send all logs from Tower to central logging

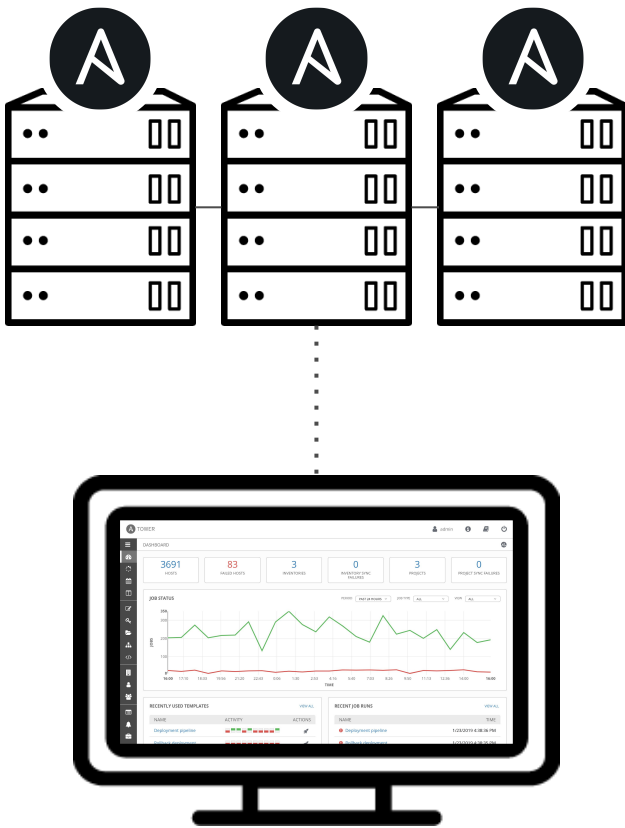


- Splunk, Loggly, ELK, REST
- Send results from Ansible runs
- but also from Tower changes



**ALWAYS KEEP
THE LIGHTS ON**

Tower can be easily set up HA - and for restricted networks, deploy isolated nodes



- Make Tower HA - it is easy! (Well, except the DB part maybe....)
- For distant or restricted networks, use isolated nodes

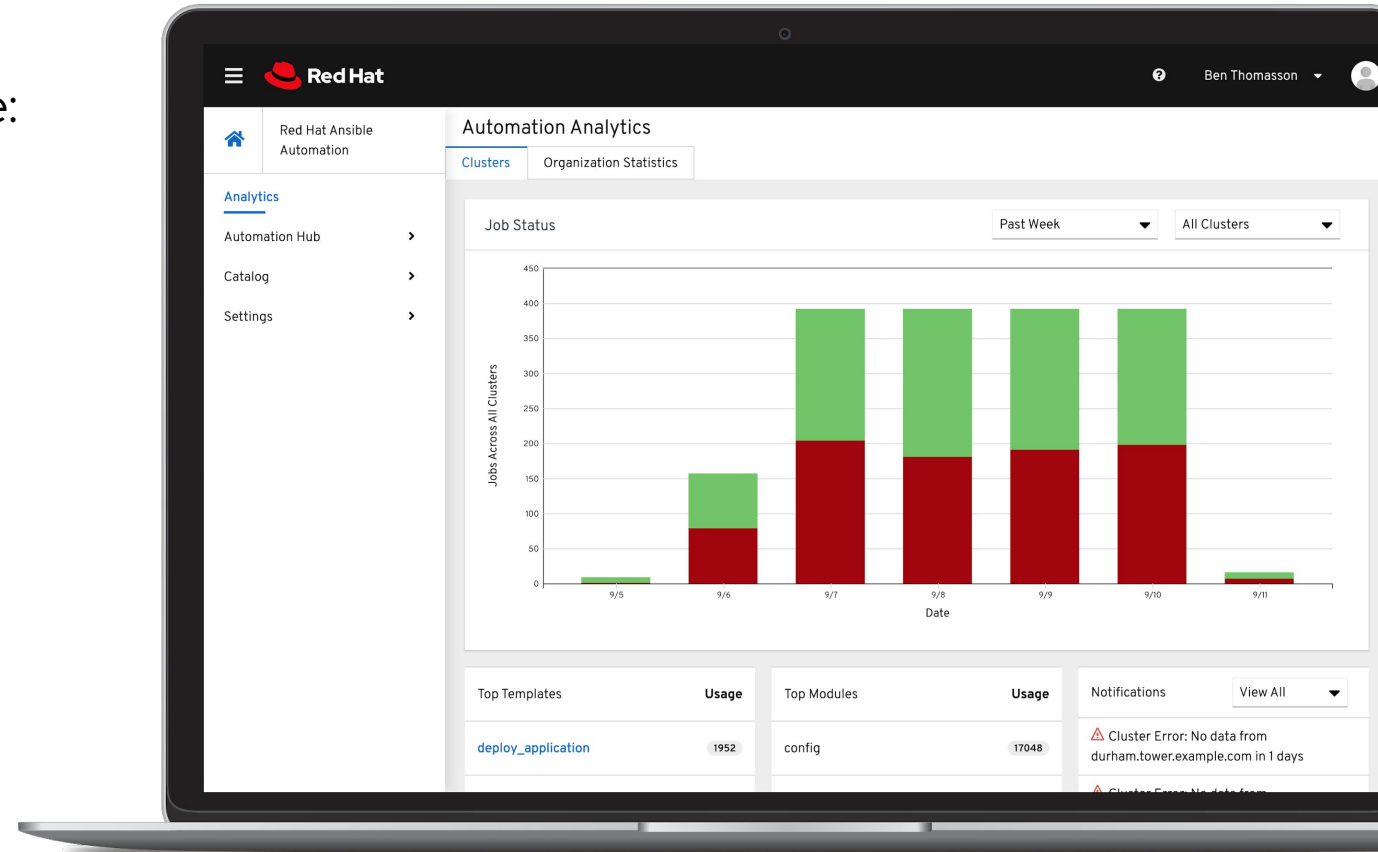


**CONNECT YOUR
AUTOMATION**

Analytics dashboard

Information across all clusters for an enterprise:


- Job Status graph
- Top Job Templates
- Top Modules




Health notifications

- Ansible Tower Cluster is down
- Node (within a cluster) is down
- Last time data was updated
- Near license count

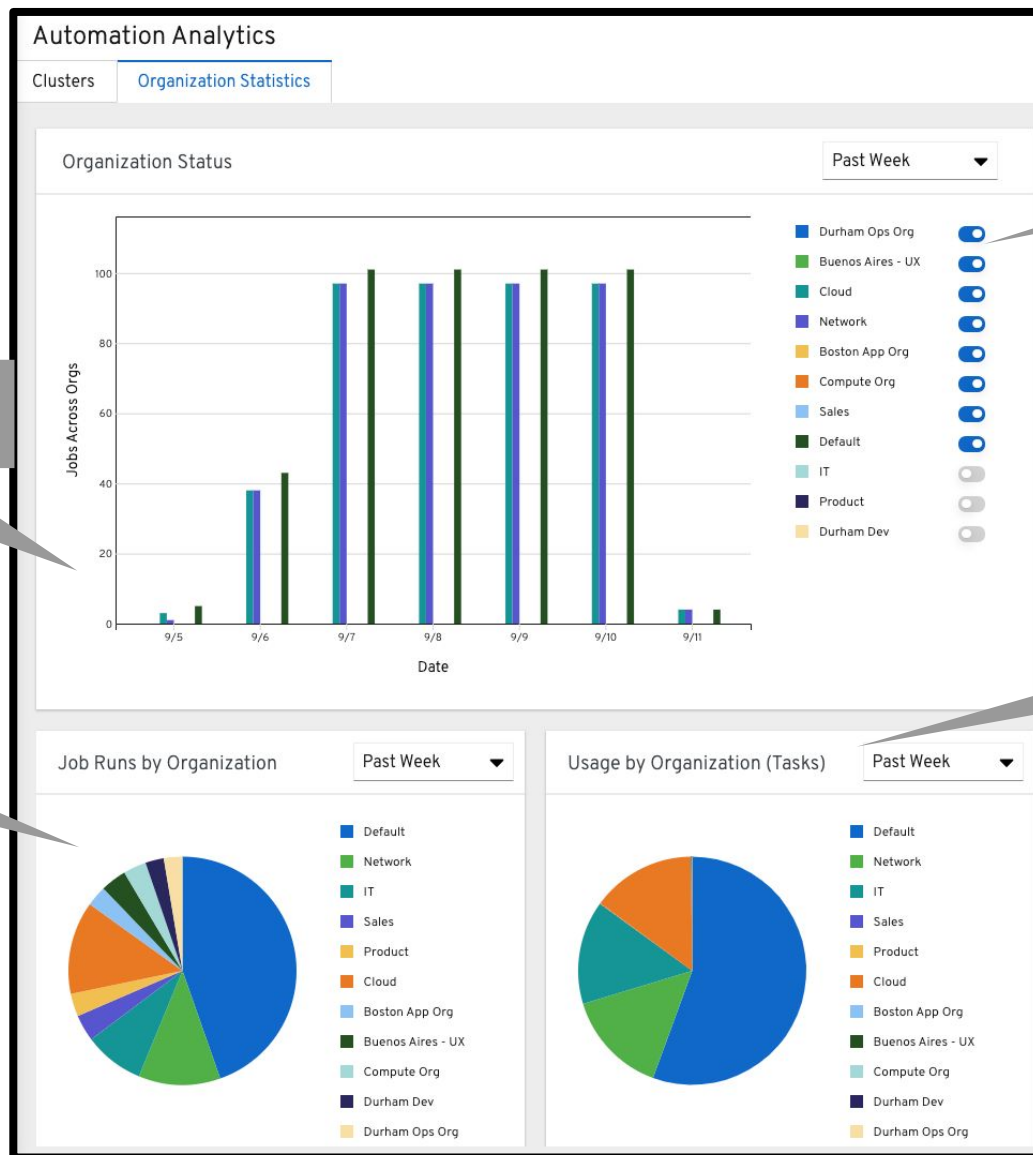
Notifications View All

 Cluster Error: No data from durham.tower.example.com in 1 days

 Cluster Error: No data from madrid.tower.example.com in 1 days

Notifications last updated 2019-09-11 07:42:12 UTC

Organizational statistics



Job Status by Organization

Job Runs by Organization

Filter by Organization

Usage by Organization

Thank you

Red Hat is the world's leading provider of enterprise open source software solutions. Award-winning support, training, and consulting services make Red Hat a trusted adviser to the Fortune 500.



[linkedin.com/company/red-hat](https://www.linkedin.com/company/red-hat)



[youtube.com/user/RedHatVideos](https://www.youtube.com/user/RedHatVideos)



[facebook.com/redhatinc](https://www.facebook.com/redhatinc)



twitter.com/RedHat