IDUG

2021 EMEA **Db2** Tech Conference

D02
Modern Db2 job automation - maintenance with Ansible -

Markus Fraune, ITGAIN

Edinburgh, Scotland | October 17 – 21, 2021

ITGAIN

Agenda

- Ansible and AWX basics
- Getting Db2 into the game
- Backup Sample
- Pros and Cons / where to use

Infrastructure as Code (IaC) (1|2)

- Manage and provision systems through machine-readable definition files with automatic pipelines, rather than physical hardware configuration or interactive tools
- For virtual AND bare-metal servers including configuration resources
- Mostly stored in a version control system (svn, gitlab, github, git…)
- declarative definitions and scripts, rather than manual processes
- Two approaches: declarative (functional) vs. imperative (procedural)

Infrastructure as Code (IaC) (2|2)

Tool	Released by	Method	Approach	Written in	Comments
Chef	Chef (2009)	Pull	Declarative and imperative	Ruby	
Otter	Inedo	Push	Declarative and imperative	-	Windows oriented
Puppet	Puppet (2005)	Pull	Declarative and imperative	C++, Clojure, Ruby	
SaltStack	SaltStack	Push and Pull	Declarative and imperative	Python	
CFEngine	Northern.tech	Pull	Declarative	C	
Terraform	HashiCorp (2014)	Push	Declarative	Go	
Ansible / Ansible Tower	Rad Hat (2012)	Push	Declarative and imperative	Python	

Configuration as Code (CaC)

- Sometimes used as a synonym for IaC or as a part of it (they overlap at some point)
- “only” for application configuration data
- Separate from IaC to be able to deploy config without new software (has its own pipelines)
- Uses its own repositories and branches
- Goal is to keep it simple and make configuration resistant going from development to production
- Tools (example): Ansible, Consul, Terraform, Vagrant, Packer, most IaC Tools

History of Ansible

- 2012: Michael DeHaan started developing Ansible tool
- 2013: Company Ansible Inc. (originally AnsibleWorks, Inc.) was founded by Michael DeHaan, Timothy Gerla and Saïd Ziouani
- 2015: Red Hat acquired Ansible
- 2016: First AnsibleFest (annual conference)
- 2017: AWX (Upstream Project for Ansible Tower) got Open Source

Ansible Nodes

- Control Node
  - machine where Ansible is installed and command like ansible or ansible-playbook can be run on (laptop, shared desktop, server)
  - Python installation needed
  - Windows is not supported
  - Can have more than one
- Managed Node
  - Systems that will be managed by ansible
  - Mostly called hosts
  - Ansible is not installed on a managed host

Ansible Inventory (1|2)

- One (or more) yaml files that contain lists of hosts
- Sometimes called the “hostfile”
- Contains information like IP addresses of managed hosts
- Nodes can be organized in (nested) groups
- Default Groups all and ungrouped (hosts with no other group than all)
- Hosts can and will be in more than one group
- Typically hosts grouped by what (database, webservice), where (local region, datacenter) and when (stages like dev, prod, test)

Ansible Inventory (2|2)

```
all:
  hosts:
    nogroup.example.com:
  children:
    webservers:
      hosts:
        foo.example.com:
        bar.example.com:
    dbservers:
      hosts:
        dbone.example.com:
        dbtwo.example.com:
        dbthree.example.com:
    east:
      hosts:
        foo.example.com:
        dbone.example.com:
        dbtwo.example.com:
    west:
      hosts:
        bar.example.com:
        dbthree.example.com:
```

```
prod:
  children:
    east:
test:
  children:
    west:
```

Ansible Variables

- Can be placed in almost every stage of ansible
- variables will be overridden from next “deeper” part (the last listed variable will override all others)
  - Part of that order:
    - Role Defaults
    - Inventory
    - Group Vars
    - Host Vars
    - Playbook Vars
    - Role Vars
    - Extra vars (when running ansible or ansible-playbook)

    (full list on next slide (hidden))

Ansible Inventory Vars

```
atlanta:
  hosts:
    host1:
      db2_port: 50001
      db2_max_connections: 100
    host2:
      db2_port: 50002
      db2_max_connections: 250
```

Ansible Group Vars (inventory file)

```
atlanta:
  hosts:
    host1:
    host2:
  vars:
    db2_port: 50003
    db2_max_connections: 500
```

Ansible Group Vars (<group>.yml)

All.yml

```
db2_port: 50000
db2_max_connections: 100
```

Atlanta.yml

```
db2_port: 50003
db2_max_connections: 500
```

Ansible Host Vars (dbone.example.com.yml)

```
db2_svcename: 50000
db2_max_connections: 100
db2_database_name: SAMPLE
db2_admin_user: db2adm1
db2_day_of_backup: 1
db2_hour_of_backup: 18
```

Ansible Playbook

- Repeatable and re-usable ”plays” to execute specific tasks to a set of hosts
- Can be used for simple configuration management or complex environment orchestration
- Rule of thumb: if a task is going to be executed more than once it will fit perfectly in a playbook
- Can include other playbooks and/or roles
- No difference in result if executed for the first or 10time: result must be the same (Idempotency)

Ansible Modules

- Discrete unit of code that can be executed from cli or wihtin a paybook/role task
- Gets executed on the managed node, return values will be collected
- Sometimes referred as „task plugins“ or „library plugins“
- Commonly used modules:
  - command
  - shell
  - service
  - copy
  - Template

https://docs.ansible.com/ansible/2.9/modules/modules_by_category.html

Ansible Modules Examples

```
- name: restart webserver
  service:
    name: httpd
    state: restarted

- name: reboot the servers
  command: /sbin/reboot -t now
```

Ansible Plugins

- Discrete pieces of code that extend Ansible‘s core functionality
- Are usually executed on the control node
- Used for
  - Connecting to an inventory
  - Logging output
  - Switch to another user (become)

Ansible Roles

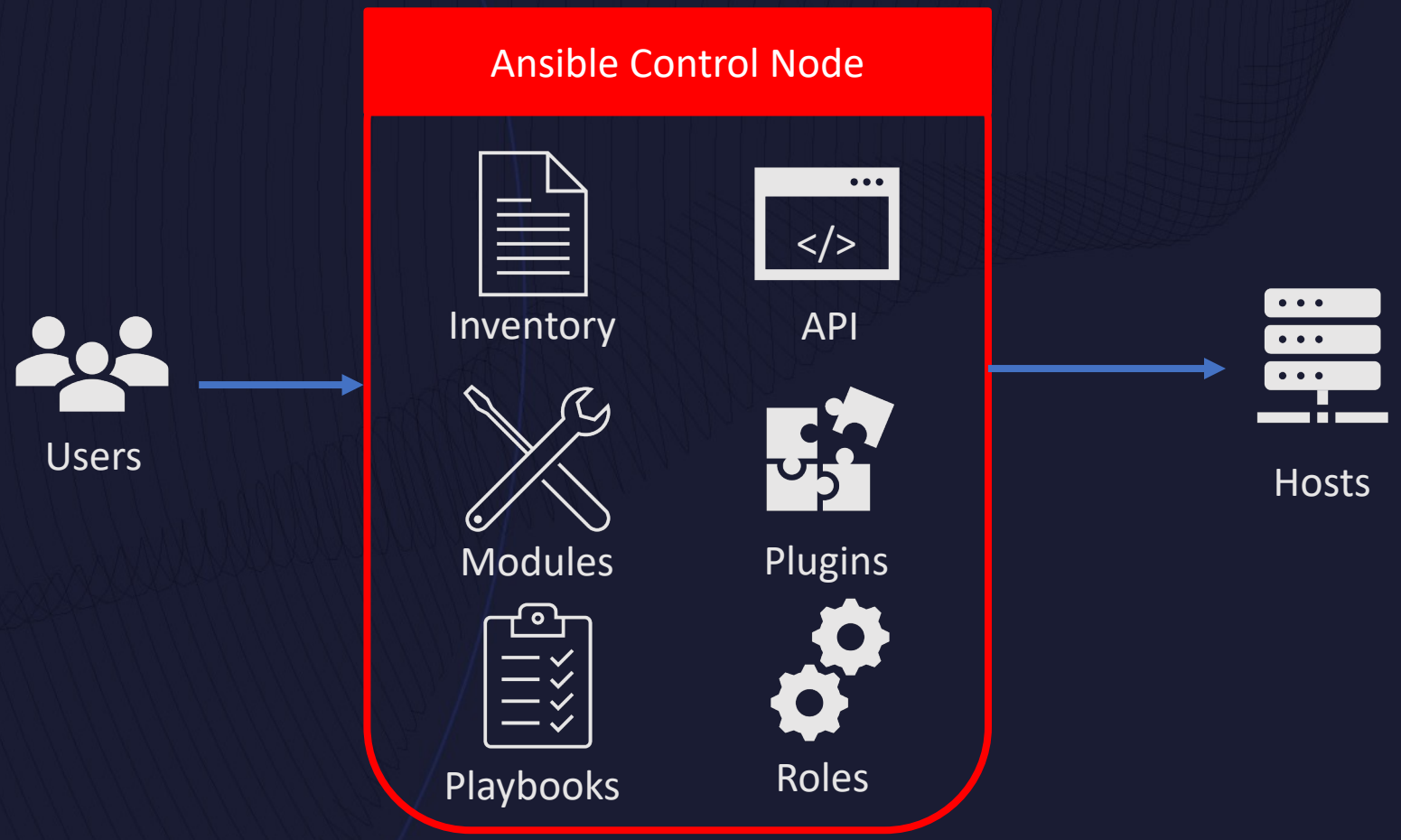
- Group a set of tasks, vars, template, files etc. to a role
- Has a known file structure
- Easily re-usable and share-able
- Can’t be executed without a play
- Examples for roles:
  - webserver
  - db2server
  - fileserver
  - db2_backup_scripts
- A play for a given set of hosts can include several roles

Ansible Roles Directory Structure

```
# playbooks
site.yml
webservers.yml
fooservers.yml
roles/
    common/
        tasks/
        handlers/
        library/
        files/
        templates/
        vars/
        defaults/
        meta/
    webservers/
        tasks/
        defaults/
        meta/
```

- Minimum is one directory per role
- Not used directories can be omitted
- Within every directory a main.yml will be called per default
- Tasks/main.yml is the default task file to be executed first (if not explicitly specified)

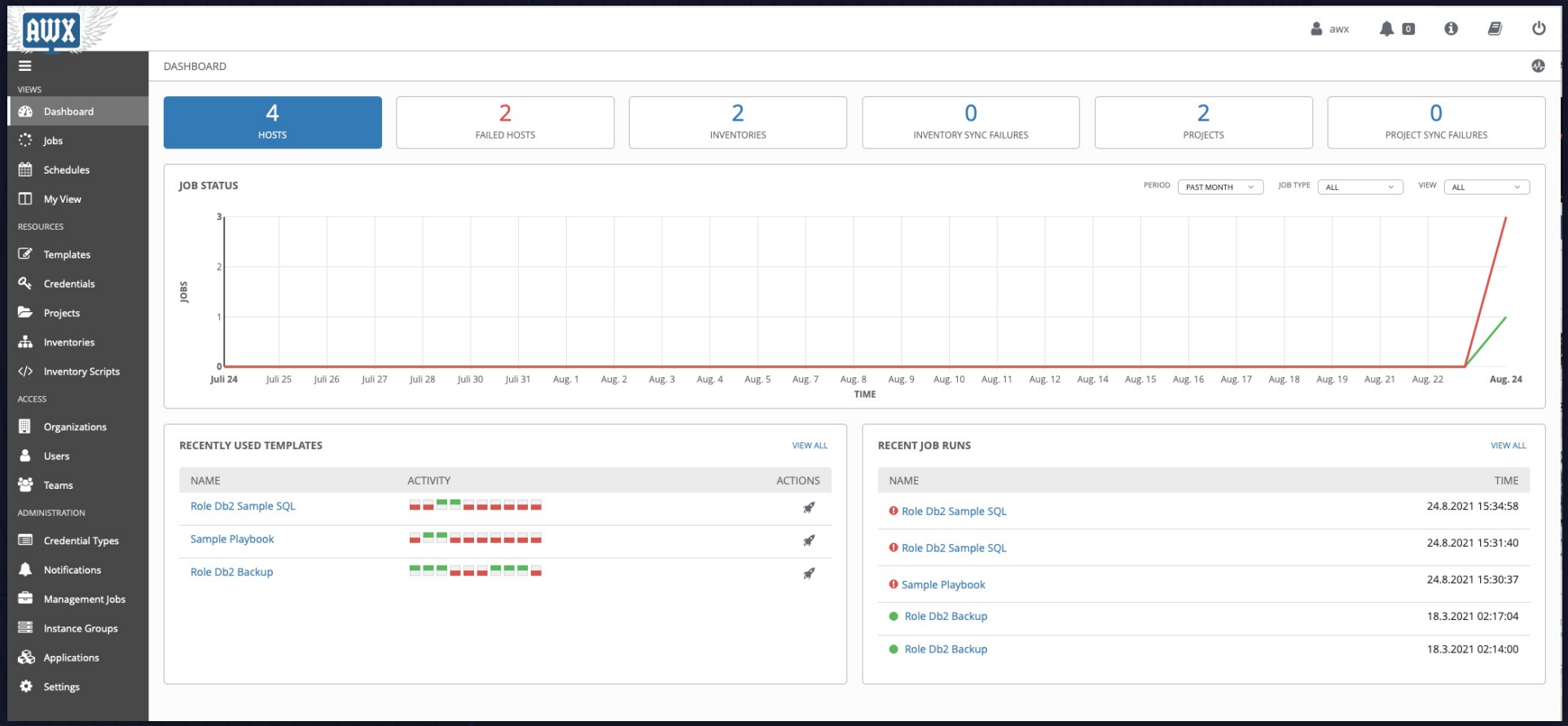
Ansible Architecture (Basic)



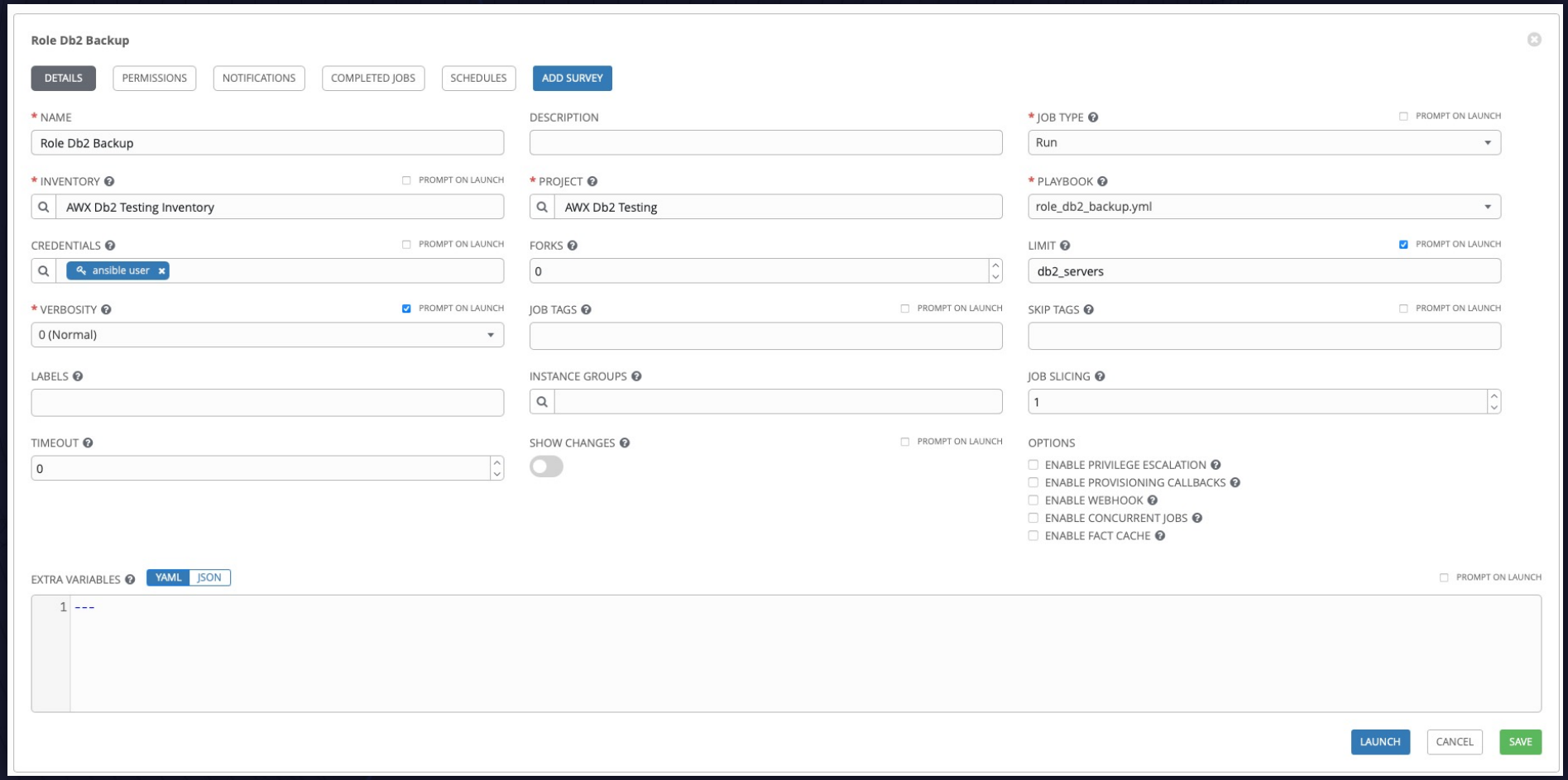
AWX Basics

- Open Source Community Project (The AWX Project) https://github.com/ansible/awx/
- Upstream for Ansible Tower (Enterprise Offering by Red Hat)
- Projects
- Job Templates
- Job Scheduler
- Workflows

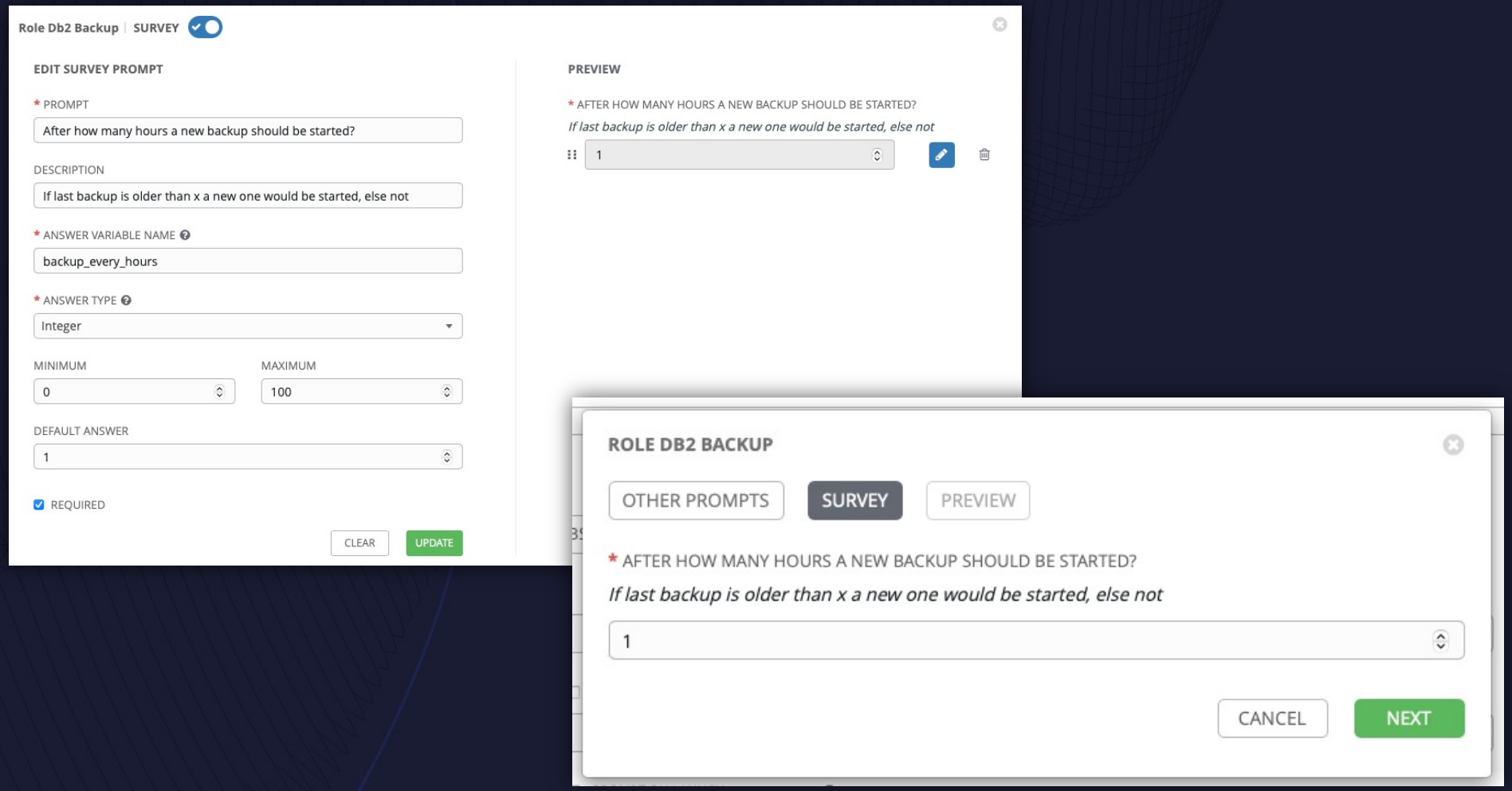
AWX Dashboard



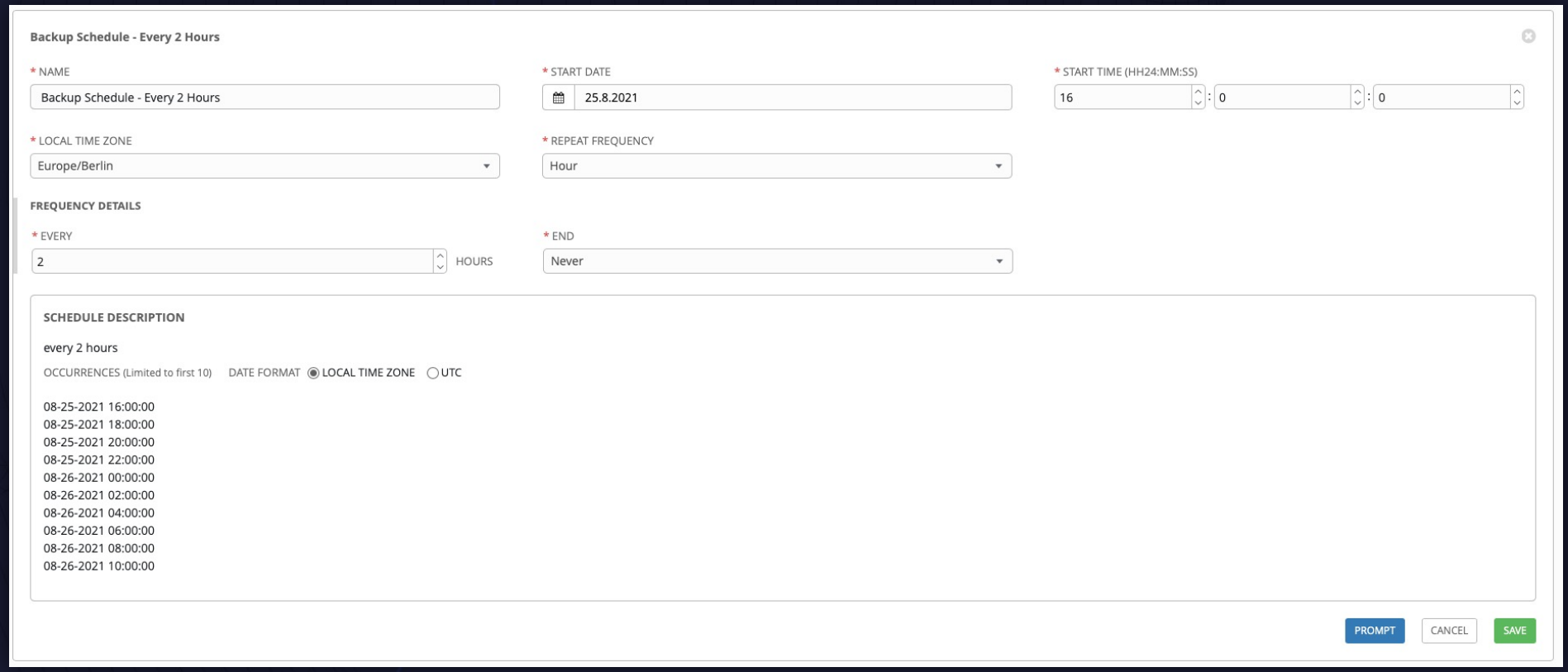
AWX Job Templates



AWX Job Templates - Survey



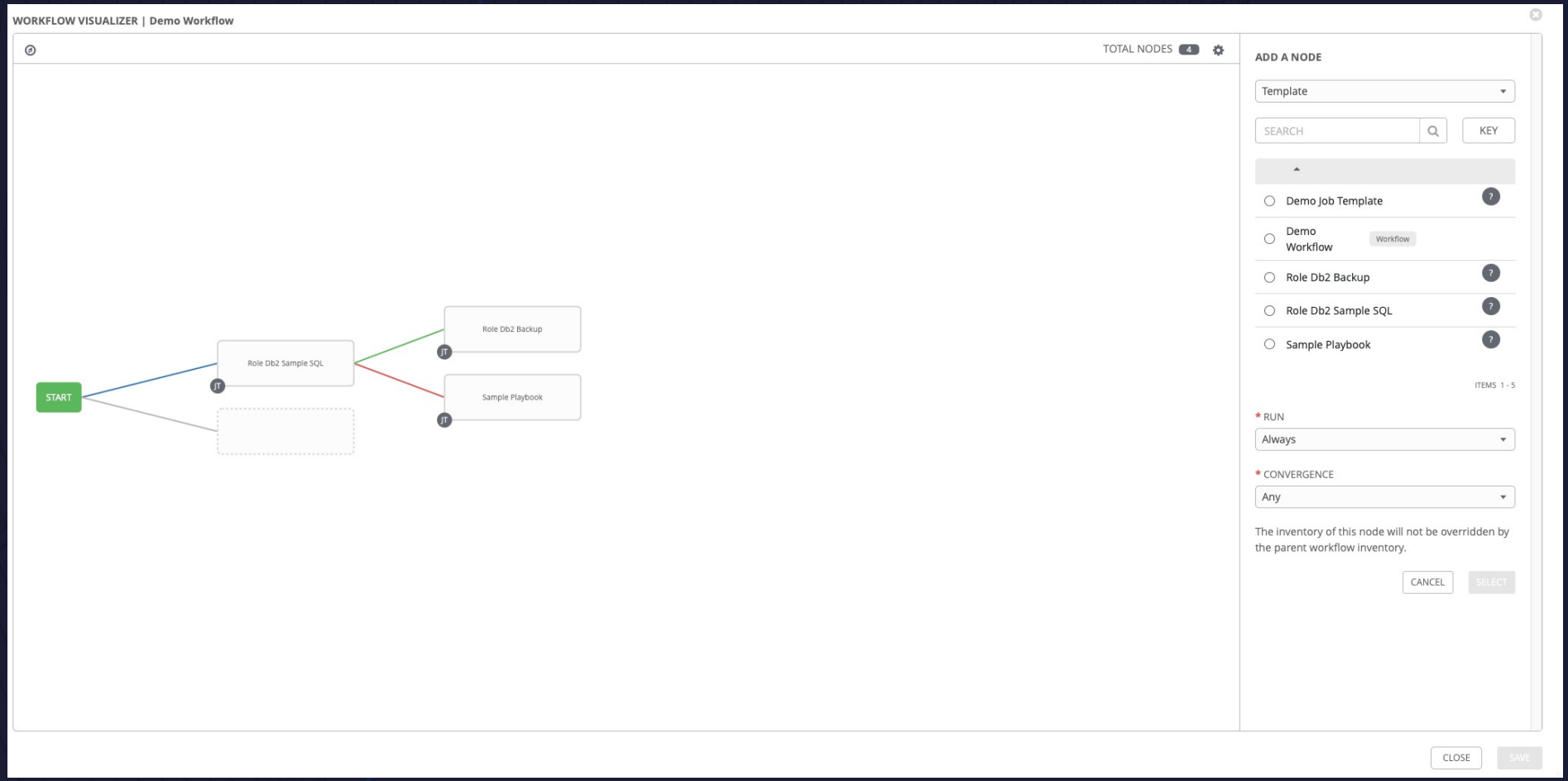
AWX Job Scheduler - Add Schedule



AWX Job Scheduler - thinks to know

- „No“ limit in number of schedules for template
- Schedule sticks to a template, can’t start more than one template
- A schedule starts the template with its predefined settings, no prompt or extra variables can be specified
- Mostly you create a template “for” a schedule, like a template to backup database on Monday with a schedule that starts this template on Mondays
- Jobs may start later than given times of the schedule because of load of AWX System

AWX Workflows



Ansible and Db2 - What do I need?

- Ansible Development Environment
  - System / Container running Ansible (Control Node)
  - System / Container to test your playbooks on (like db2 container, “empty” Alpine Linux container) -> Can be role / playbooks as well
- Repository to store and version your files (e.g. gitlab, github…)
- Running AWX K8S Environment (e.g. with minikube in a VM)
- SSH Access from four Control Node and AWX system to Managed Nodes
- Code Editor with a linter (e.g. Visual Studio Code)

Ansible and Db2 - basic host file (db2srv1.mydomain.org)

```
db2_app_user: db2_tech_u
db2_inst_user: db2inst1
db2_inst_con: server_encrypt
db2_dbname: proddb
db2_db_path: "/db2/{{ db2_inst_user }}/{{ db2_db_name }}/dbpath"
db2_data_path: "/db2/{{ db2_inst_user }}/{{ db2_db_name }}/data"
db2_db_pagesize: 32
```

Ansible and Db2 - Server Provisioning

Add db2 app users to System and add software dependencies

```
- name: Add user '{{ db2_app_user }}' with a bash, groups 'admins’ and 'developers'
  ansible.builtin.user:
    name: "{{ db2_app_user }}"
    home: "/db2/homes/{{ db2_app_user }}"
    shell: /bin/bash
    groups: admins,developers
    append: yes

- name: Upgrade all packages on the Server
  apt: upgrade=yes

- name: Install the latest versions of each component
  apt:
    name:
      - gcc
      - python
    state: latest
```

Ansible and Db2 - Db2 Provisioning (1|3)

- Bring db2 software package onto the system

```
- name: Download db2_install_package
  get_url:
    url: http://software_repo.mydomain.org/db2/luw/11.5.6/db2_install_package.tar.gz
    dest: /tmp/db2_install_package.tar.gz
```

- Unarchive db2 software package

```
- name: Unarchive a file that is already on the remote machine
  ansible.builtin.unarchive:
    src: /tmp/db2_install_package.tar.gz
    dest: /tmp/db2_install_package
    remote_src: yes
```

Ansible and Db2 - Db2 Provisioning (2|3)

- Install Db2

```
- name: Copy Response File
  ansible.builtin.template:
    src: respone.file.template
    dest: /tmp/response.file
- name: Install Db2 Server
  command: /tmp/db2_install_package/install/db2setup -r /tmp/response.file
```

- Create Db2 Instance

```
- name: Create Db2 instance {{db2_inst_user }}
  command: /opt/…/instance/db2icrt -a {{ db2_inst_con }} -u db2fenc1 {{db2_inst_user }}
```

Ansible and Db2 - Db2 Provisioning (3|3)

- Create Database

```
- name: create database {{ db2_dbname }} in instance {{ db2_inst_user }}
  become: true
  become_method: su
  become_user: "{{ db2_inst_user }}"
  shell: bash -lci 'db2 create database {{ db2_dbname }} on {{ db2_data_path }} dbpath on {{ db2_db_path }} page size {{ db2_db_pagesize }}k '
```

- Grant App User

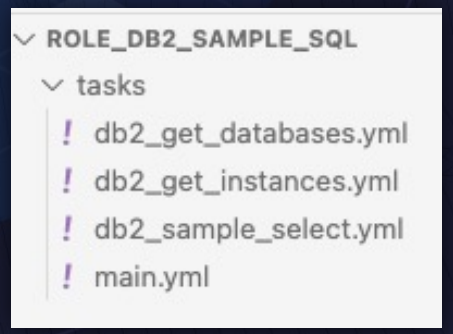
```
- name: grant App user
  become: true
  become_method: su
  become_user: "{{ db2_inst_user }}"
  shell: bash -lci 'db2 +o connect to {{ db2_dbname }} && db2 "grant connect, createtab, dataccess on database to user {{ db2_app_user }}"'
```

Ansible and Db2 - Role Sample SQL (1|4)

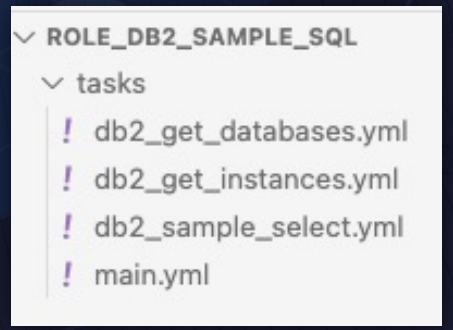
```
# main.yml - get all local installations
- name: get local db2 installations
  shell: db2ls |awk '/V/ {print $1}’
  register: db2_install_path_list
  changed_when: "db2_install_path_list.rc < 0"
- name: debug output
  debug:
    msg: "myscript has error: {{ db2_install_path_list.stderr }}"
  when: db2_install_path_list.stderr | length > 0
  failed_when: db2_install_path_list.stderr | length > 0
- name: include db2_get_instances
  include_tasks: db2_get_instances.yml
  with_items: "{{ db2_install_path_list.stdout_lines }}"
```

Ansible and Db2 - Role Sample SQL (2|4)

```
# db2_get_instances.yml
- name: get local instances for path {{ item }}
  shell: "{{ item }}/instance/db2ilist"
  register: db2_instance_list
  changed_when: "db2_instance_list.rc < 0"
- name: debug output
  debug:
    msg: "myscript has error: {{ db2_instance_list.stderr }}"
  when: db2_instance_list.stderr | length > 0
  failed_when: db2_instance_list.stderr | length > 0
- name: include db2_get_databases
  include_tasks: db2_get_databases.yml
  with_items: "{{ db2_instance_list.stdout_lines }}"
```



Ansible and Db2 - Role Sample SQL (3|4)




```
# db2_get_databases.yml
- name: set fact db2 instance vara
  ansible.builtin.set_fact:
    db2_instance: "{{ item }}"
- name: get local databases for instance {{ item }}
  become: true
  become_method: su
  become_user: "{{ db2_instance }}"
  shell: bash -lci 'db2 list db directory|awk -v RS= '/Indirect/' grep "Database name"|cut -d "=" -f 2|sort|uniq'
  register: db2_database_list
  changed_when: "db2_database_list.rc < 0"
- name: debug output
  debug:
    msg: "myscript has error: {{ db2_database_list.stderr }}"
  when: db2_database_list.rc > 0
  failed_when: db2_database_list.rc > 0
- name: include db2_sample_select
  include_tasks: db2_sample_select.yml
  with_items: "{{ db2_database_list.stdout_lines }}"
```

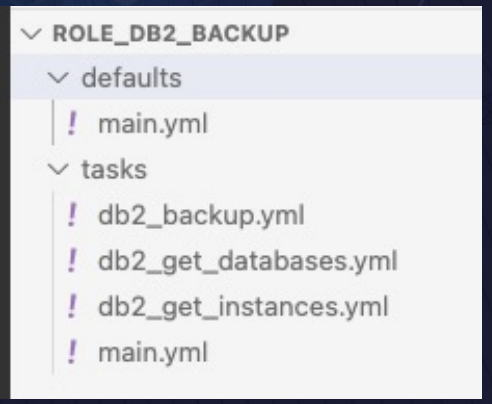
Ansible and Db2 - Role Sample SQL (4|4)

```
# db2_sample_select.yml
- name: sample select for database {{ item }} at instance {{ db2_instance }}
  become: true
  become_method: su
  become_user: "{{ db2_instance }}"
  shell: bash -lci 'db2 +o connect to {{ item }} && db2 -v "select * from
syscat.tables" '
  register: sql_output
  changed_when: "sql_output.rc < 0"
  failed_when: sql_output.rc > 4
- name: debug output
  debug:
    msg: "mysql has a non zero rc ({{ sql_output.rc }}): {{ sql_output.stdout }}"
  when: sql_output.rc > 0
  failed_when: sql_output.rc > 0
```

Ansible and Db2 - Role Backup (1|5)

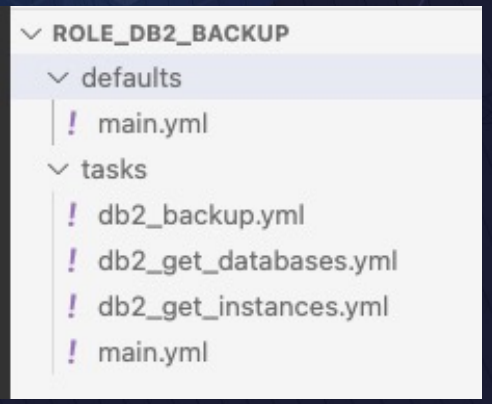
- In comparison to sample sql role:
  - Added defaults/main.yml for default variables
  - Include in “db2_get_databases.yml” is now db2_backup and not “db2_sample_select”

```
# defaults/main.yml
backup_every_hours: 1
```



Ansible and Db2 - Role Backup (2|5)

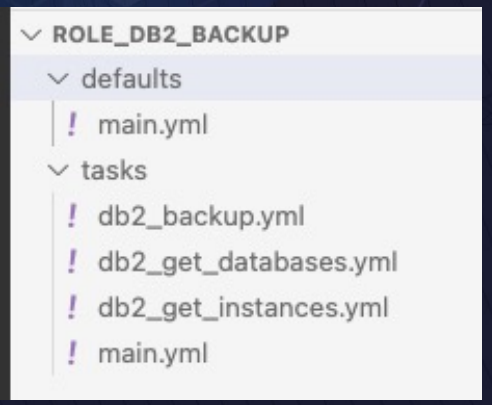
```
- name: check if database is active
  become: true
  become_method: su
  become_user: "{{ db2_instance }}"
  shell: bash -lci 'db2 list active databases | grep -i {{
item }}'
  register: list_active_db_output
  changed_when: "list_active_db_output.rc < 0"
  failed_when: "list_active_db_output.rc > 3"
- name: debug list active db output
  debug:
    msg: "List Active datasbases showed {{ item }} as active
database, no offline backup possible"
  when: "list_active_db_output.rc == 0"
```



Ansible and Db2 - Role Backup (3|5)

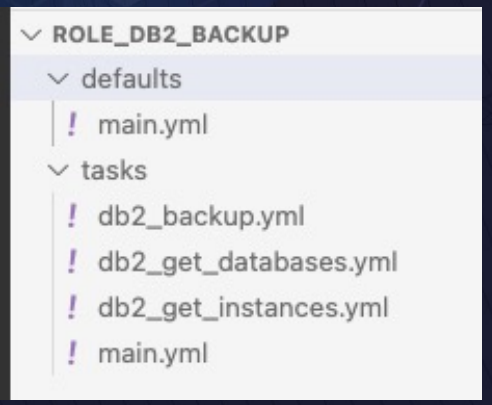
```
- name: check if last db backup timestamp is older than {{ backup_every_hours }}
hours
  become: true
  become_method: su
  become_user: "{{ db2_instance }}"
  shell: bash -lci 'db2 +o connect to {{ item }} && db2 -x "select
timestamp(LAST_BACKUP) as last_backup from table(mon_get_database(-1)) as t where
LAST_BACKUP >= current timestamp - {{ backup_every_hours }} hours"'
  register: backup_old_enough
  changed_when: "backup_old_enough.rc < 0"
  failed_when: "backup_old_enough.rc > 3"
  when: "list_active_db_output.rc == 1"

- name: debug check last backup ts
  debug:
    msg: "Last Backup TS of {{ item }} is not old enough - {{
backup_old_enough.stdout }}"
  when: backup_old_enough is not skipped and "backup_old_enough.rc == 0"
```



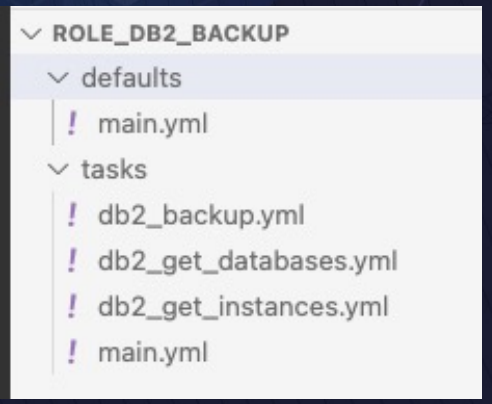
Ansible and Db2 - Role Backup (4|5)

```
- name: block backup database {{ item }} at instance {{ db2_instance }}
  become: true
  become_method: su
  become_user: "{{ db2_instance }}"
  block:
    - name: try to backup {{ item }} at instance {{ db2_instance }}
      shell: bash -lci 'db2 -v backup db {{ item }} to /dev/null’
      register: backup_output
      changed_when: "backup_output.rc == 0"
      failed_when: "backup_output.rc > 3"
      when: "list_active_db_output.rc == 1 and backup_old_enough.rc == 1"
      ignore_errors: yes
  rescue:
    - name: print out backup stdout
      debug:
        msg: "backup has a non zero rc ({{ backup_output.rc }}): {{ backup_output.stdout }}"
```

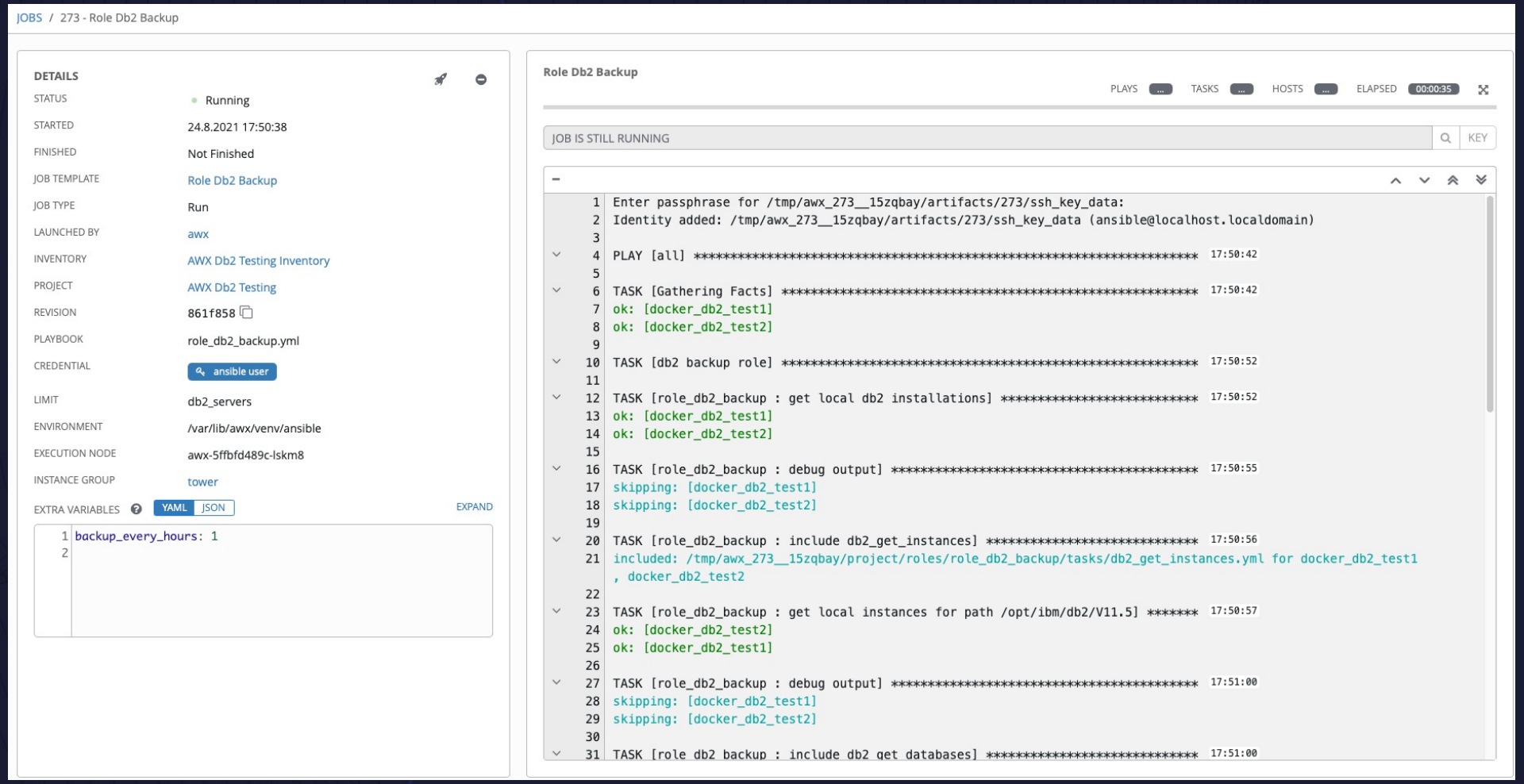


Ansible and Db2 - Role Backup (5|5)

```
- name: print backup output
  debug:
    msg: "backup was successfull. output: {{ backup_output.stdout }} "
  when: backup_output is succeeded and backup_output is not skipped
```

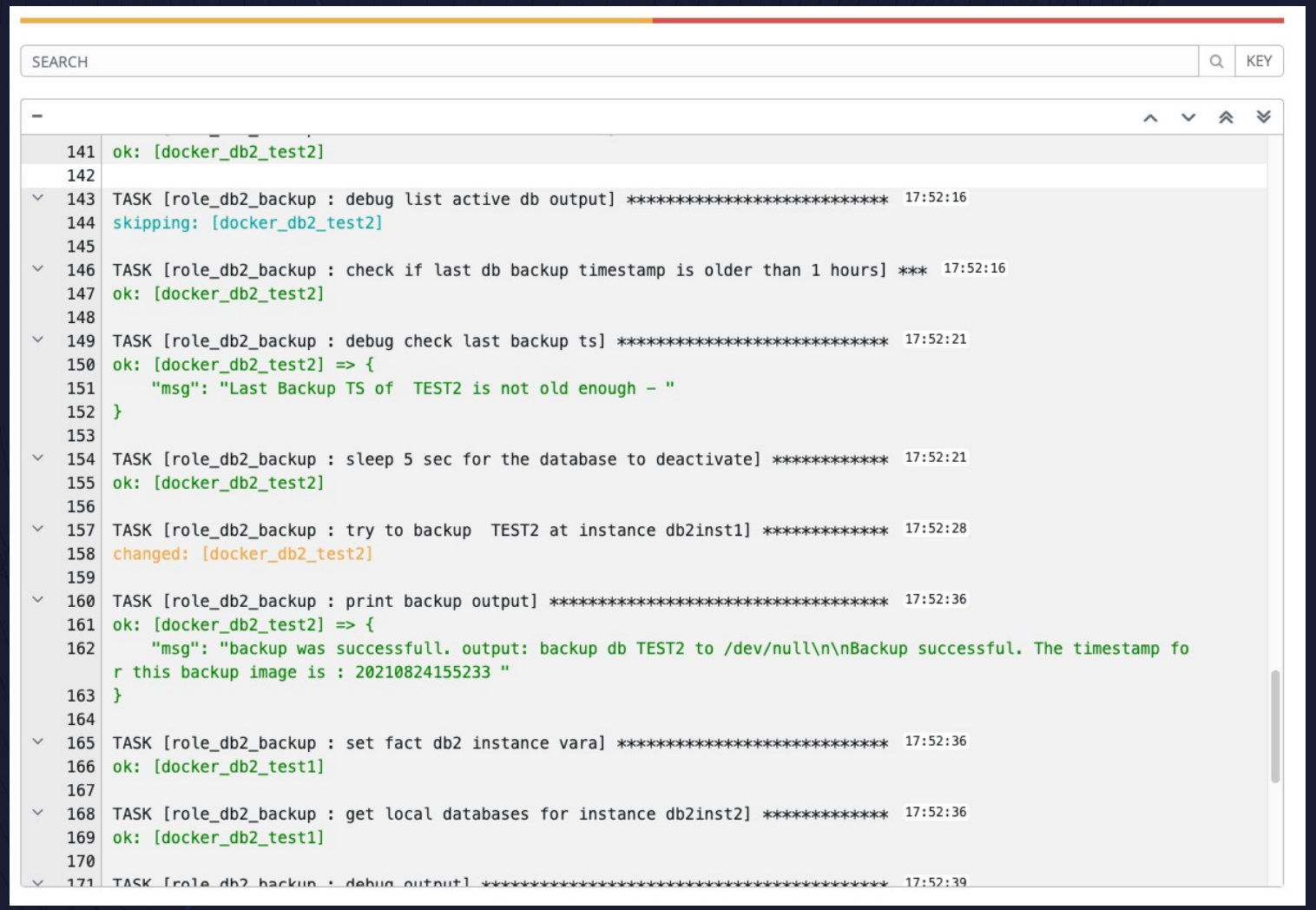


Ansible and Db2 - Backup Template (1|4)

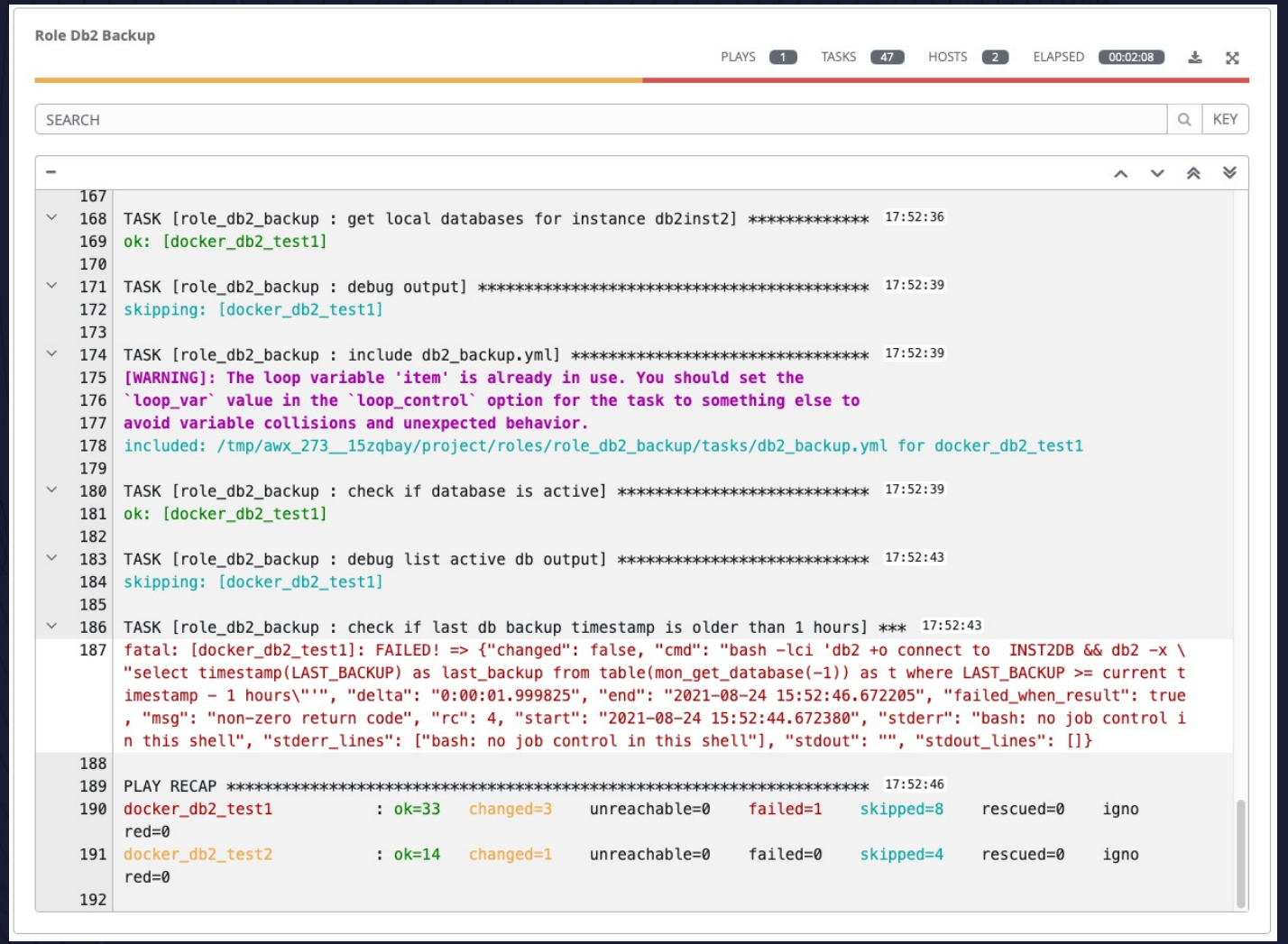


Ansible and Db2 - Backup Template (2|4)

Ansible and Db2 - Backup Template (3|4)



Ansible and Db2 - Backup Template (4|4)



Ansible and Db2 - What to use it for?

- Repeating jobs
- Repeating activities
- If you have more than 1 system you want to run jobs on
- Existing Scripts
  - Deployment / Copy to managed node
  - Execute via schedule on managed node
  - add crontab via role to execute scripts locally
- New Tasks
  - Create plays and roles for new requirements
  - Enhance and reuse existing plays/tasks/roles

Ansible and Db2 - Pros / Cons

- Cons
  - New / Something to learn
  - Slightly higher complexity
- Pros
  - Fast learning curve
  - Existing scripts can be reused
  - Repeatable and checked-in jobs and configuration lowers the risk of a failure
  - Easy to execute on a high number of hosts
  - Shareable scripts
  - Tons of pre-existing roles and plays available online

Session: D02
Speaker: Markus Fraune
Company: ITGAIN
Email Address: markus.fraune@itgain.de
Twitter: @maggusf

Please fill out your session evaluation!