GESTION DES ERREURS
PRESENTATION

notions avancées

- Par défaut, tout code retour d'une commande distinct de 0 ou tout statut d'échec d'un module provoquera l'arrêt du play.
- Pour les tâches, **ignore_errors** permet d'ignorer le statut failed.

Code : *ignore_errors au niveau d'une tâche*

```
---
- name: do some stuff
  command: /bin/foo
  ignore_errors: yes
```

Code : *ignore_errors au niveau d'un playbook*

```
- hosts: all
  ignore_errors: yes
  tasks:
  - name: ...
```

- Pour les hosts, **ignore_unreachable** permet d'ignorer une machine injoignable.

Code : *ignore_errors au niveau d'une tâche*

```
---
- name: do some stuff
  command: /bin/foo
  ignore_unreachable: yes
```

Code : *ignore_errors au niveau d'un playbook*

```
- hosts: all
  ignore_unreachable : yes
  tasks:
  - name: ...
```


Pour aller plus loin : https://docs.ansible.com/ansible/latest/user_guide/playbooks_error_handling.html

GESTION DES ERREURS
ERREURS & HANDLERS

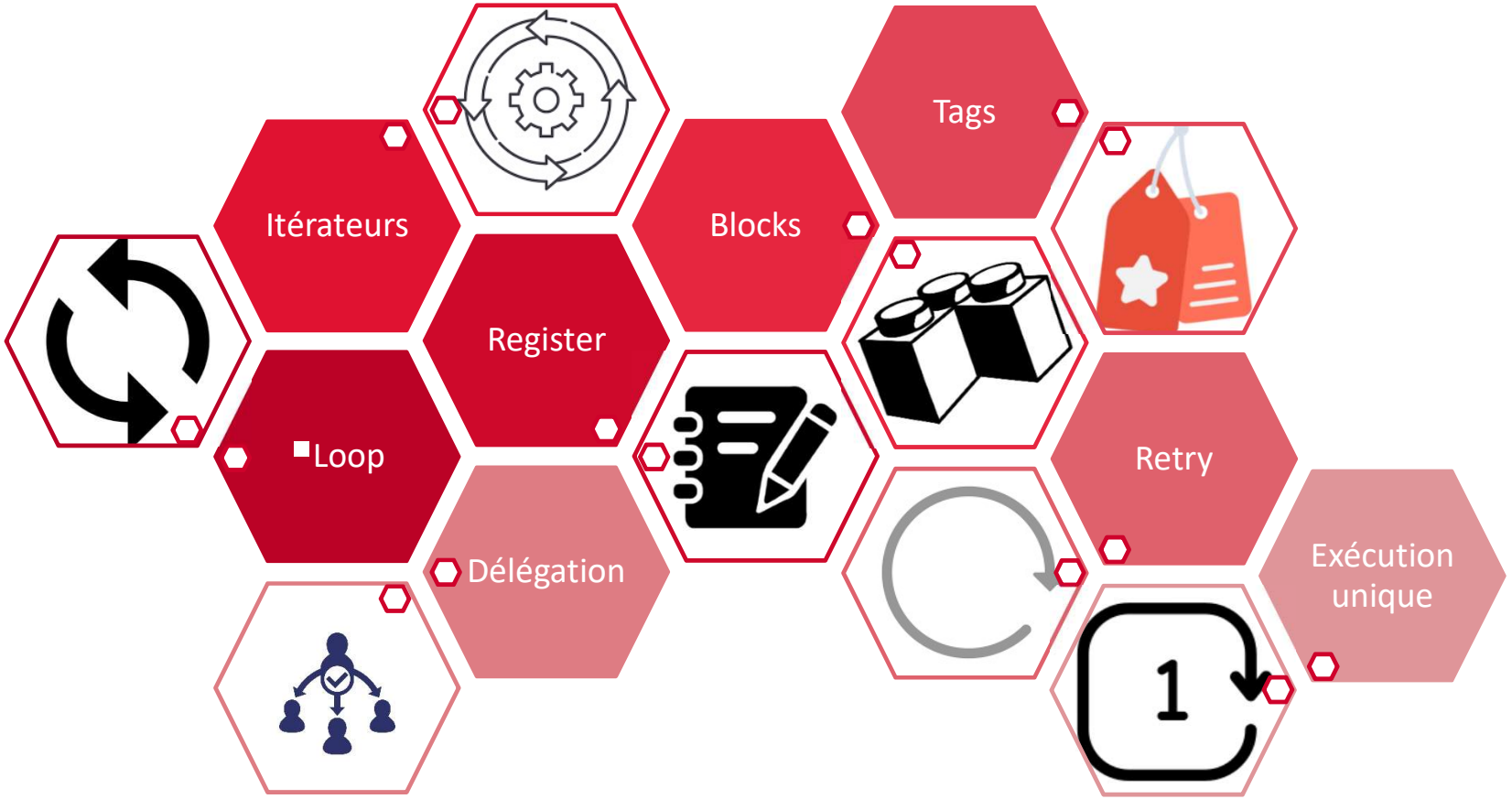
notions expert

- Les handlers sont lancés une fois toutes les tâches réalisées.

- Dans le cas où une tâche déclenche un notify et qu'une tâche ultérieure échoue, le handler ne sera jamais exécuté ce qui peut donner lieu à un état bancal de la machine.

- Pour remédier à cette situation, il est possible de forcer l'exécution des handlers après un échec :
  - Avec **l'instruction « force_handlers: true »** déclarée au niveau du playbook
  - Via **la ligne de commande --force-handlers**

USAGES AVANCÉS
SOMMAIRE



USAGES AVANCÉS
LOOP

- Boucle simple

Code : *boucle sur une tache*

```
- name: Ensure services are started
  service:
    name: "{{ item }}"
    state: started
  loop:
    - service1
    - service2
```

Code : *boucle avec variable externalisée*

```
vars:
  services:
    - service1
    - service2

- name: Ensure services are started
  service:
    name: "{{ item }}"
    state: started
  loop: "{{ services }}"
```

A noter : il est possible de boucler sur des structures complexes en utilisant item.monattribut

Les versions antérieures d'Ansible (<2.5) s'appuyaient sur une syntaxe différente pour réaliser les boucles à l'aide des mots clés **with_* (with_items, with_files, with_sequences, …)**

Pour aller plus loin : https://docs.ansible.com/ansible/latest/user_guide/playbooks_loops.html

USAGES AVANCÉS
LOOP & REGISTER

- Boucle avec register

Code : *boucle sur une tache*

```
- name: Display items
  shell:
    echo "{{ item }}"
  loop:
    - service1
    - service2
  register: echo_results
```

A noter : le contenu de la variable liée à register est sensiblement différent au sein d'une boucle

L'ordre dans la liste finale correspond à l'ordre défini dans le champ loop

Code : *boucle avec variable externalisée*

```
"echo_results": [
        {
            "changed": true,
            "start": "...",
            "end": "...",
            "delta": "...",
            "cmd": "echo \"service1\" ",
            "item": "service1",
            "invocation": {
                "module_args": "echo \"service1\"",
                "module_name": "shell"
            },
            "rc": 0,
            "stdout": "service1"
            "stderr": "",
        },
        {
           "changed": true,
            "start": "...",
            "end": "...",
            "delta": "...",
            "cmd": "echo \"service2\" ",
            "item": "service2",
            "invocation": {
                "module_args": "echo \"service2\"",
                "module_name": "shell"
            },
            "rc": 0,
            "stdout": "service2"
            "stderr": "",
        }
    ]
```

USAGES AVANCÉS
LOOP & CONDITIONS

- Boucle avec condition

Code : *boucle sur une tache*

```
- name: Ensure Nginx is installed if enough space on root
  apt-get:
    name: nginx
    state: latest
  loop: "{{ ansible_mounts }}"
  when: item.mount == "/" and item.size_available > 100000000
```

A noter : la condition est testée pour chaque élément

- Ansible parcourt l'ensemble de la liste même si certaines itérations ne sont pas exécutées du au conditionnement

Pour aller plus loin *: https://docs.ansible.com/ansible/latest/user_guide/playbooks_conditionals.html#using-conditionals-in-loops*

USAGES AVANCÉS
ITÉRATEURS (1/2)

- Possibilité d'écrire des boucles dans les tâches Ansible

Code : *exemple d'un play contenant sans itérateur*

```
---
- name: install nginx
  package:
    name: nginx

- name: install vim
  package:
    name: vim

- name: install tomcat7
  package:
    name: tomcat7

- name: install git
  package:
    name: git
```

Code : *exemple d'un play contenant un itérateur*

```
---
- name: install package
  package:
    name: "{{ item }}"
  with_items:
  - nginx
  - vim
  - tomcat7
  - git
```

USAGES AVANCÉS
ITÉRATEURS (2/2)

- Plusieurs types:
  - **with_items**
    - tableau simple
    - item => l'entrée
    - loop.index => l'indice
    - loop.first, loop.last
  - **with_dict**
    - dictionnaire
    - item.key
    - item.value
  - **with_together** notions expert
    - Parcours simultané de plusieurs tableaux
    - item.0 => élément du premier tableau
    - item.1 => élément du second tableau
  - **with_nested** notions expert
    - parcours de la combinatoire des tableaux fournis

Pour aller plus loin : https://docs.ansible.com/ansible/latest/collections/ansible/builtin/together_lookup.html

USAGES AVANCÉS
BLOCK

- Ansible permet de regrouper plusieurs tâches au sein d'un seul bloc logique avec l'instruction **block**
- De la sorte, certaines directives (become*, when, ignore*, …) s'appliquent à l'ensemble des tâches du bloc
- La gestion des erreurs s'applique aussi à l'ensemble du bloc grâce aux instructions **rescue** pour gérer la remédiation et **always** qui sera déclenché aussi bien lorsque le bloc termine en succès qu'en échec.

Code : #1 *exemple d'un play sans bloc / remédiation*

```
tasks:
  - name: tache#1
    debug:
      msg: 'tache#1'
    changed_when: yes
    notify: event__demo_block
  - name: tache#2 KO
    command: /bin/false

handlers:
  - name: on event__demo_block
    listen: event__demo_block
    debug:
      msg: 'handler event__demo_block'
```

Code : #2 *exemple d'un play utilisant une remédiation*

```
tasks:
  - name: demonstration block et gestion d'erreur
    block:
      - name: tache#1
        debug:
          msg: 'tache#1'
        changed_when: yes
        notify: event__demo_block
      - name: tache#2 KO
        command: /bin/false
    rescue:
      - name: tache#remediation
        meta: flush_handlers
    always:
      - name: tache#always
        debug:
          msg: "tache#always«

handlers:
  - name: on event__demo_block
    listen: event__demo_block
    debug:
      msg: 'handler event__demo_block'
```


Pour aller plus loin : https://docs.ansible.com/ansible/latest/user_guide/playbooks_blocks.html

USAGES AVANCÉS
TAG (1/2)

- Permettent de labéliser différents types d'objet
  - Rôles
  - Tâches
  - Inclusions

- Permettent de filtrer le périmètre d'exécution
  - Par inclusion
  - Par exclusion

Code : *exécuter un playbook avec seulement les tags*

```
$ ansible-playbook … --tags "configuration,packages"
```

Code : *exécuter un playbook à l'exception de qq tags*

```
$ ansible-playbook … --skip-tags "configuration,packages"
```

Code : *exemple de tâches avec tag*

```
tasks:
- name: Install the servers
  yum:
    name:
    - httpd
    - memcached
    state: present
  tags:
  - packages
  - webservers

- name: Configure the service
  template:
    src: templates/etc/service.conf.j2
    dest: /etc/service.conf
  tags:
  - configuration
```

Pensez à utiliser les options
--list-tags pour avoir la liste des tags d'un playbook
--list-tasks pour avoir la liste des tâches associées à un tag
(voir la doc pour plus de précision)

Pour aller plus loin : https://docs.ansible.com/ansible/latest/user_guide/playbooks_tags.html

USAGES AVANCÉS
TAG (2/2)

- Exécuter uniquement les tâches du tag ‘install’

Code : *exemple d’une ligne de commande*

```
$ ansible-playbook all –i host site.yml -t install
```

- Exécuter uniquement les tâches des tags ‘install’ et ‘configure’

Code : *exemple d’une ligne de commande*

```
$ ansible-playbook all –i host site.yml -t install,configure
```

- Il existe des tags built-in :
  - never : jamais exécuté, sauf si explicitement mentionnées via –t / --tags
  - always
  - tagged
  - untagged

USAGES AVANCÉS
RETRY

- La capture d'un résultat permet de faire plusieurs tentatives sur une tâche

Code : *exemple d'un play contenant un retry*

```
- name: do some stuff
  command: grep -q toto /etc/passwd
  register: my_cmd_result
  until: my_cmd_result.rc == 0
  retries: 5          # number of attempts
  delay: 10           # in seconds
```

until est obligatoire. En cas d'absence de la condition d'arrêt, Ansible force retries à 1

Il existe un module (**wait_for**) qui adresse potentiellement les même use case que **retry**.
wait_for et retry ont toutefois qq différences notables.

wait_for	retry / until
wait_for est un module en tant que tel. Il s'applique principalement pour l'attente d'un provisionning infra ou du reboot d'un système	Retry / until n'est pas un module à part entière. Il peut donc s'appliquer à n'importe quelle tâche.
Le message d'erreur est customisable	Il n'est pas possible d'avoir un message d'erreur customisé

Pour aller plus loin : https://docs.ansible.com/ansible/latest/user_guide/playbooks_loops.html

USAGES AVANCÉS
DÉLÉGATION

notions avancées

- Elle permet de lancer une commande concernant une machine sur une autre machine

- Cette fonction est généralement utilisée dans deux cas de figure
  - Lors des phases de provisionnement des machines
  - Lors des déploiements à chaud (sans interruption de service)

Code : *exemple contenant une délégation*

```
- name: activer les alertes nagios pour les serveurs web
  hosts: webservers
  tasks:
  - name: activer les alertes nagios
    nagios: action=enable_alerts service=web host= {{ inventory_hostname }}
    delegate_to: my.nagios.server
```

variable built-in désignant le serveur de l'inventaire

la tache s'exécute autant de fois qu'il y a de serveurs dans le groupe webservers, mais au lieu de s'exécuter sur chacun des serveurs, elle s'exécute sur le host my.nagios.server

Pour aller plus loin : https://docs.ansible.com/ansible/latest/user_guide/playbooks_delegation.html

USAGES AVANCÉS
EXÉCUTION UNIQUE

notions avancées

- Permet d'exécuter une tâche une seul fois sur une des machines du play

Code : *exemple d'un play contenant une exécution unique*

```
---
- name: forcer un timestamp unique pour un groupe de serveur
  set_fact:
    mytstamp: "{{ ansible_date_time.iso8601_basic_short }}"
  run_once: yes
```

- Si le play est configuré en 'serial', le run_once est exécuté une fois par groupe d'itération

Pour aller plus loin : https://docs.ansible.com/ansible/latest/user_guide/playbooks_strategies.html

RÉCAPITULATIF

Comment conditionne-t-on une itération d'un loop?

Quels sont les itérateurs disponibles?

Comment fonctionnent les tags?

A quoi sert le delegate_to?

Comment fonctionne la gestion des erreurs avec un block?

Comment fait-on pour exécuter une tâche une seule fois?