

Ansible

Playbooks

View the Lesson

Intro to Playbooks

Playbooks are a completely different way to use ansible than in ad-hoc task execution mode, and are particularly powerful. **Playbook'lar, ansible'ı kullanmak için ad-hoc'tan/geçici görev yürütme modundan tamamen farklı bir yoldur ve güçlüdür.**

Simply put, playbooks are the basis for a really simple configuration management and multi-machine deployment system, unlike any that already exist, and one that is very well suited to deploying complex applications.

Playbooks lar, basit bir yapılandırma yönetiminin ve çoklu makina dagitim sisteminin temelidir. Digerlerinden farkli olarak, karmasik uygulamari dagitmak icin daha uygundur.

Playbooks can declare configurations, but they can also orchestrate steps of any manual ordered process, even as different steps must bounce back and forth between sets of machines in particular orders. They can launch tasks synchronously or asynchronously.

Playbooks lar yapılandırmaları bildirebilir ve farklı adımların belirli emirleri makine grupları arasında ileri geri gitmesi gerekse bile işlem adımlarını organize edebilir. Görevleri eşzamanlı veya eşzamansız olarak başlatabilirler.

Playbooks are expressed in **YAML** format and have a minimum of syntax, which intentionally tries to not be a programming language or script, but rather a model of a configuration or a process.

Playbook'lar YAML formatında ifade edilir ve sade bir yazım formatına sahiptir. Bunun bilinçli olarak, bir programlama dili veya komut dosyası değil, bir yapılandırma veya süreç modeli olması amaçlanmıştır.

Each playbook is composed of one or more plays in a list. **Her playbook, içerisinde bir veya daha fazla play olan bir listedir.**

i The goal of a play is to map a group of hosts to some well defined roles, represented by things ansible calls tasks. At a basic level, a task is nothing more than a call to an ansible module. **i** **Play'ın amacı, ansible tarafından görev olarak adlandırılan şeyleri, bir grup HOSTS lara eşleştirmektir. Basit manada, bir task, bir ansible modüle yapılan çağrıdan başka bir şey değildir.**

By composing a playbook of multiple ‘plays’, it is possible to orchestrate multi-machine deployments, running certain steps on all machines in the webservers group, then certain steps on the database server group, then more commands back on the webservers group, etc.

Birden çok play’dan oluşan bir playbook oluşturarak, webservers grubundaki tüm makinelerde belirli adımları çalıştırarak, ardından database server grubunda belirli adımları, ardından tekrar webservers group grubunda daha fazla komutu çalıştırarak çoklu makine dağıtımlarını düzenlemek mümkündür.

Here’s a playbook, first-playbook.yml that contains two plays: (First play starting from line-2, and the second play starts from line-15)

Burada iki play içeren first-playbook.yml adlı bir playbook var: (İlk play 2. satırdan, ikinci play 15. satırdan başlıyor)

```
---
- hosts: webservers
  remote_user: root
  tasks:
    - name: ensure apache is at the latest version
      yum:
        name: httpd
        state: latest
    - name: write the apache config file
      template:
        src: /srv/httpd.j2
        dest: /etc/httpd.conf

- hosts: databases
  remote_user: root
  tasks:
    - name: ensure postgresql is at the latest version
      yum:
        name: postgresql
        state: latest
    - name: ensure that postgresql is started
      service:
        name: postgresql
        state: started
```

Note:

- You can use this method to switch between the host group you’re targeting, the username logging into the remote servers, whether to sudo or not, and so forth. Plays, like tasks, run in the order specified in the playbook: top to bottom.

Bu yöntemi, sudo olsun yada olmasın, hedeflediğiniz host gruplar arasında veya remote server'lara girmek için kullanıcılar arasında geçiş yapmak için kullanabilirsiniz. Play'lerde tasks'ler gibi, playbook'da belirtilen sırayla yukarıdan aşağıya doğru çalışır.

Let's Practice - 1

Plays, like tasks, run in the order specified in the playbook: top to bottom.

Select one:

True **Correct**

False

Hosts and Users

For each play in a playbook, you get to choose which machines in your infrastructure to target and what remote user to complete the steps (called tasks) as.

Bir playbook'daki her play için, altyapınızdaki hangi makineleri hedefleyeceğinizi ve adımları (tasks olarak adlandırılır) hangi remote user'ın tamamlayacağını seçersiniz.

The `hosts` line is a list of one or more groups or host patterns, separated by colons, as described in the Patterns: targeting hosts and groups documentation. The `remote_user` is just the name of the user account:

`hosts` satırı, hedeflenen hosts yada dokümantasyonlarda tanımlandığı gibi, ":" ile ayrılmış bir veya daha fazla grup veya hosts modelinin bir listesidir. `remote_user`, yalnızca kullanıcı hesabının adıdır:

```
---
- hosts: webservers
  remote_user: root
```

Note:

- You must ensure that the host or group is first defined in the inventory file we created earlier.

host'un veya grubun daha önce oluşturduğumuz envanter dosyasında tanımlandığından emin olmalısınız.

- The host defined in the inventory file must match the host used in the playbook and all connection information for the host is retrieved from the inventory file.

Envanter dosyasında tanımlanan host, playbook'da kullanılan host'la eşleşmelidir ve host için tüm bağlantı bilgileri envanter dosyasından alınır.

Let's Practice - 2

You must ensure that the host or group is first defined in the configuration file we created earlier.

Select one:

True

False **Correct**

Modules

Modules (also referred to as “task plugins” or “library plugins”) are discrete units of code that can be used from the command line or in a playbook task. Ansible executes each module, usually on the remote target node, and collects return values.

Modüller ('görev eklentileri' veya 'kütüphane eklentileri' olarak da adlandırılır) komut satırından veya bir playbook görevinde kullanılabilen ayrı kod birimleridir. Ansible, genellikle remote target node'da her modülü çalıştırır ve dönüş değerlerini toplar. (mesela, değişikliğin yapılıp yapılmadığı gibi donus değerini toplar)

You can execute modules from the command line: **Modülleri komut satırından çalıştırabilirsiniz:**

```
$ ansible webservers -m service -a "name=httpd state=started"
$ ansible webservers -m ping
$ ansible webservers -m command -a "/sbin/reboot -t now"
```

Each module supports taking arguments. Nearly all modules take key=value arguments, space delimited. Some modules take no arguments, and the command/shell modules simply take the string of the command you want to run.

Her modül argüman almayı destekler. Neredeyse tüm modüller key=value değişkenlerini alır, boşlukla ayrılıyor. Bazı modüller argüman almaz ve komut/kabuk modülleri çalıştırmak istediğiniz komutun string'ini alır.

From playbooks, Ansible modules are executed in a very similar way:

Playbook'larda, Ansible modülleri komut satırındakine çok benzer bir şekilde yürütülür:

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

All modules return JSON format data. This means modules can be written in any programming language. Modules should be idempotent, and should avoid making any changes if they detect that the current state matches the desired final state. When used in an Ansible playbook, modules can trigger `change events` in the form of notifying `handlers` to run additional tasks. Tüm modüller JSON formatında veriler döndürür. Bu, modüllerin herhangi bir programlama dilinde yazılabileceği anlamına gelir. Modüller bağımsız olmalı ve mevcut durum istenen durumla eşleşirse herhangi bir değişiklik yapmaz. Modüller, bir Ansible playbook'ında kullanıldığında, ek görevleri çalıştırmak için `handlers`leri bilgilendirerek `change events`'leri (degisiklikleri) tetikleyebilir.

Let's Practice - 3

Which of the followings are correct about Ansible modules?

Select one or more:

- ☐ You can execute modules from the command line or in a playbook task. **Correct**
- ☐ Modules should avoid making any changes if they detect that the current state matches the desired final state. **Correct**
- ☐ Each module supports taking arguments. **Correct**
- ☐ Modules should not be idempotent
- ☐ Modules ("task plugins" or "library plugins") are discrete units of code. **Correct**



Tasks

Each play contains a list of tasks. Tasks are executed in order, one at a time, against all machines matched by the host pattern, before moving on to the next task. It is important to understand that, within a play, all hosts are going to get the same task directives. It is the purpose of a play to map a selection of hosts to tasks.

Her play bir task listesi içerir. Task'ler, bir sonraki task'e geçmeden önce host modeliyle eşleşen tüm makinelerde her seferinde bir tane olarak sırayla uygulanır. Bir play'da tüm hostların aynı task emirlerini alacağını anlamak önemlidir. (Yani, bir gruptaki tüm hostlar bu taskleri uygulayacak demektir.) Bir play'in amacı, bir grup hosts'u tasklere eşleştirmektir.

When running the playbook, which runs top to bottom, hosts with failed tasks are taken out of the rotation for the entire playbook. If things fail, simply correct the playbook file and rerun.

Yukarıdan aşağıya doğru çalışan playbook, başarısız görevleri olan host'ları tüm playbook rotasyonundan çıkarır. İşler başarısız olursa, playbook dosyasını düzeltin ve yeniden çalıştırın.

 **Note:** The goal of each task is to execute a module, with very specific arguments. Variables can be used in arguments to modules. 

Not: Her görevin amacı, çok özel argümanlarla bir modül yürütmektir. Değişkenler, modüllerin argümanlarında kullanılabilir.

Every task should have a `name`, which is included in the output from running the playbook. This is human readable output, and so it is useful to provide good descriptions of each task step. If the name is not provided though, the string fed to 'action' will be used for output.

Her görevin, playbook'un çalıştırılmasından elde edilen çıktıda bulunan bir adı olmalıdır. Bu, insan tarafından okunabilen bir çıktıdır ve bu nedenle her bir görev adımının tanımlarını iyi yapmak önemlidir. Yine de tanımlama sağlanamazsa, çıktı için "action'e beslenen/gonderilen string kullanılacaktır.

Here is what a basic task looks like: **Temel bir görev şöyle görünür:**

```
tasks:
- name: make sure apache is running
  service:
    name: httpd
    state: started
```

Let's Practice - 4

Which one is incorrect about Ansible tasks?

Select one:

Every task should have a name, which is included in the output from running the playbook.

- ☐ All hosts are going to get the different task directives within a play. **Correct**
- ☐ The goal of each task is to execute a module, with very specific arguments.

Handlers

As we've mentioned, modules should be idempotent and can relay when they have made a change on the remote system. Playbooks recognize this and have a basic event system that can be used to respond to change. **Modüller bağımsız olmalıdır ve remote sistemde bir değişiklik yaptıklarında geçiş yapabilirler. Playbook'lar bunu tanır ve değişime yanıt vermek için kullanılabilecek temel bir event sistemine sahiptir.**

These 'notify' actions are triggered at the end of each block of tasks in a play, and will only be triggered once even if notified by multiple different tasks. **Bu "notify"(bildirim) eylemleri, bir Play'deki her görev bloğunun sonunda tetiklenir ve birden fazla farklı task tarafından bildirilse bile yalnızca bir kez tetiklenir.**

For instance, multiple resources may indicate that apache needs to be restarted because they have changed a config file, but apache will only be bounced once to avoid unnecessary restarts. **Örneğin, birden fazla kaynak, bir yapılandırma dosyasını değiştirdikleri için apache'nin yeniden başlatılması gerektiğini gösterebilir, ancak gereksiz yeniden başlatmaları önlemek için apache yalnızca bir kez geri döner.**

Here's an example of restarting two services when the contents of a file change, but only if the file changes: **Bir dosyanın içeriği değiştiğinde iki hizmeti yeniden başlatma örneği, ama yalnızca dosya değiştiğinde,**

```
- name: template configuration file

template:
  src: template.j2
  dest: /etc/foo.conf
notify:
  - restart memcached
  - restart apache
```

The things listed in the **notify** section of a task are called **handlers**.

Bir görevin bildir/notify bölümünde listelenen şeylere işleyici/handlers denir.

Handlers are lists of tasks, not really any different from regular 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.

Handlers'lar/İşleyiciler, normal tasklerden farklı olmayan, benzersiz/unique bir adla referans edilen ve bildirimciler tarafından bildirilen tasklerdir. Hiçbir şey bir handler'i bilgilendirmese, handlers çalışmayacaktır. Bir işleyiciye/handler'a kaç görevin bildirildiğine bakılmaksızın, belirli bir play'da tüm görevler(taskler) tamamlandıktan sonra yalnızca bir kez çalışacaktır.

Here's an example handlers section: Örnek bir işleyiciler bölümü:

```
handlers:
  - name: restart memcached
    service:
      name: memcached
      state: restarted
  - name: restart apache
    service:
      name: apache
      state: restarted
```

Let's Practice - 5

If nothing notifies a handler, it will not run.

Select one:

- ☒ True Correct
- ☐ False

Inventory File

Ansible works against multiple managed nodes or “hosts” in your infrastructure at the same time, using a list or group of lists known as **inventory**. Once your inventory is defined, you use patterns to select the hosts or groups you want Ansible to run against.

Ansible, envanter olarak bilinen bir liste veya listeler grubunu kullanarak aynı anda altyapınızdaki birden çok yönetilen nodes veya hosts'a karşı çalışır. Envanteriniz tanımlandıktan sonra, Ansible'in uygulamasını istediğiniz host'ları veya grupları seçmek için kalıpları kullanırsınız.

The default location for inventory is a file called `/etc/ansible/hosts`. You can specify a different inventory file at the command line using the `-i < path >` option. You can also use multiple inventory files at the same time, and/or pull inventory.

Envanter için varsayılan konum, `/etc/ansible/hosts` adlı bir dosyadır. `-i <PATH>` seçeneğini kullanarak komut satırında farklı bir envanter dosyası belirtebilirsiniz. Aynı anda birden fazla envanter dosyası kullanabilir ve/veya envanter çekebilirsiniz.

Formats, Hosts, and Groups

The inventory file can be in one of many formats, depending on the inventory plugins you have. The most common formats are `INI` and `YAML`. A basic `INI` `etc/ansible/hosts` might look like this:

Envanter dosyası, sahip olduğunuz envanter eklentilerine bağlı olarak birçok biçimde/formatta olabilir. En yaygın formatlar `INI` ve `YAML`'dir. Basit bir `INI` `etc/ansible/hosts` şöyle görünebilir:

```
mail.example.com
[webservers]
foo.example.com
bar.example.com
[dbservers]
one.example.com
two.example.com
three.example.com
```

The headings in brackets are group names, which are used in classifying hosts and deciding what hosts you are controlling at what times and for what purpose.

Koseli parantez içindeki başlıklar, hostları sınıflandırmada ve hangi hostları hangi zamanlarda ve hangi amaçla kontrol ettiğinize karar vermede kullanılan grup adlarıdır.

Here's that same basic inventory file in `YAML` format:

YAML formatındaki aynı temel envanter dosyasına örnek:

```
all:
  hosts:
    mail.example.com:
  children:
    webservers:
      hosts:
        foo.example.com:
        bar.example.com:
    dbservers:
      hosts:
        one.example.com:
        two.example.com:
        three.example.com:
```

There are two default groups: all and ungrouped. The all group contains every host. The ungrouped group contains all hosts that don't have another group aside from all. Every host will always belong to at least 2 groups (all and ungrouped or all and some other group). Though all and ungrouped are always present, they can be implicit and not appear in group listings like group_names.

İki default grup vardır: tümü ve gruplandırılmamış. All grup, her hostu içerir. ungrouped grup, başka bir grubu olmayan tüm hostları içerir. Her host her zaman en az 2 gruba ait olacaktır (all ve ungrouped veya all ve some other group). all ve ungrouped her zaman mevcut olsa da, ustü kapalı olabilirler ve grup_adı gibi grup listelerinde görünmeyebilirler.

Hosts in multiple groups

You can (and probably will) put each host in more than one group. For example a production webserver in a datacenter in Atlanta might be included in groups called [prod] and [atlanta] and [webservers]. You can create groups that track: Her host'u birden fazla gruba koyabilirsiniz. Örneğin Atlanta'daki bir veri merkezindeki bir production web server'i, [prod] ve [atlanta] ve [webservers] adlı gruplara dahil edilebilir. Aşağıdakileri şekilde gruplar oluşturabilirsiniz:

- What - An application, stack or microservice. (For example, database servers, web servers, etc).
Ne - Bir uygulama, yığın veya mikro hizmet. (Örneğin, veritabanı sunucuları, web sunucuları vb.).
(What dan kasit makineye grup ismi belirlerken makinenin ne amacla olusturulduguna bakip ve ona gore bir grup adi belirlemektir.)
- Where - A datacenter or region, to talk to local DNS, storage, etc. (For example, east, west).
Nerede - Yerel DNS, depolama vb. ile konuşmak için bir veri merkezi veya bölge (Örneğin, doğu, batı).
(Where den kasit makinenin nerede olusturulduguna bakip ve ona gore bir grup adi belirlemektir.)
- When - The development stage, to avoid testing on production resources. (For example, prod, test).
Ne zaman - Üretim kaynakları üzerinde test yapmaktan kaçınmak için geliştirme aşaması. (Örneğin, ürün, test).
(When den kasit makinenin ne zaman olusturulduguna bakip ve ona gore bir grup adi belirlemektir.)
- Extending the previous YAML inventory to include what, when, and where would look like: Önceki YAML envanterini neyin, ne zaman ve nerede görüneceğini içerecek şekilde genişletebiliriz:

```
all:
  hosts:
    mail.example.com:
  children:
    webservers:
      hosts:
        foo.example.com:
        bar.example.com:
    dbservers:
      hosts:
        one.example.com:
        two.example.com:
        three.example.com:
  east:
    hosts:
      foo.example.com:
      one.example.com:
      two.example.com:
  west:
    hosts:
      bar.example.com:
      three.example.com:
  prod:
    hosts:
      foo.example.com:
      one.example.com:
      two.example.com:
  test:
    hosts:
      bar.example.com:
      three.example.com:
```

You can see that `one.example.com` exists in the `dbservers`, `east`, and `prod` groups. `one.example.com`'un `dbservers`, `east` ve `prod` gruplarında bulunduğunu görebilirsiniz.

Bir makine, birden fazla grupta bulunabilir.

You can also use nested groups to simplify prod and test in this inventory, for the same result: Aynı sonuç için bu envanterde prod ve test'i basitleştirmek için iç içe grupları da kullanabilirsiniz:

```
all:
  hosts:
    mail.example.com:
  children:
    webservers:
      hosts:
        foo.example.com:
        bar.example.com:
    dbservers:
      hosts:
        one.example.com:
        two.example.com:
        three.example.com:
  east:
    hosts:
      foo.example.com:
      one.example.com:
      two.example.com:
  west:
    hosts:
      bar.example.com:
      three.example.com:
  prod:
    children:
      east:
  test:
    children:
      west:
```

Let's Practice - 6

What are the most common inventory file formats?

Select one or more:

- ☐ JAVA
- ☐ YAML **Correct**
- ☐ INI **Correct**
- ☐ JSON
- ☐ Python

ad-hoc commands

An Ansible ad-hoc command uses the `/usr/bin/ansible` command-line tool to automate a single task on one or more managed nodes. Ad-hoc commands are quick and easy, but they are not reusable. So why learn about ad-hoc commands first? `Ad-hoc commands demonstrate the simplicity and power of Ansible`. The concepts you learn here will port over directly to the playbook language.

Ansible'da ad-hoc komutu, bir veya daha fazla managed node'larda tek bir görevi otomatikleştirmek için `/usr/bin/ansible` komut satırı aracını kullanır. Geçici komutlar hızlı ve kolaydır, ancak yeniden kullanılamazlar. Öyleyse neden önce geçici komutları öğrenelim? Geçici komutlar, Ansible'in basitliğini ve gücünü gösterir. Burada öğrendiğiniz kavramlar doğrudan playbook diline aktarılacaktır.

Ad-hoc commands are great for tasks you repeat rarely. For example, if you want to power off all the machines in your lab for Christmas vacation, you could execute a quick one-liner in Ansible without writing a playbook. An ad-hoc command looks like this: `Geçici komutlar, nadiren tekrarladığınız görevler için mükemmeldir. Örneğin, Noel tatili için laboratuvarınızdaki tüm makineleri kapatmak istiyorsanız, Ansible'da bir playbook yazmadan daha hızlı bir tek satırlık işlem ile gerçekleştirebilirsiniz. Geçici bir komut şöyle görünür:`

```
$ ansible [pattern] -m [module] -a "[module options]"
```

Use cases for ad-hoc tasks: `Geçici görevler için case'leri kullanın:`

Ad-hoc tasks can be used to reboot servers, copy files, manage packages and users, and much more. You can use any Ansible module in an ad-hoc task. Ad-hoc tasks, like playbooks, use a declarative model, calculating and executing the actions required to reach a specified final state. They achieve a form of idempotence by checking the current state before they begin and doing nothing unless the current state is different from the specified final state. `Geçici görevler, sunucuları yeniden başlatmak, dosyaları kopyalamak, paketleri ve kullanıcıları yönetmek ve çok daha fazlası için kullanılabilir. Herhangi bir Ansible modülünü geçici bir görevde kullanabilirsiniz. Playbooklar gibi Ad-hoc taskleri de, belirli bir nihai duruma ulaşmak için gereken eylemleri hesaplayan ve yürüten bildirimsel bir model kullanır. Başlamadan önce mevcut durumu kontrol ederek ve mevcut durum belirtilen son durumdan farklı değilse hiçbir şey yapmaz.`

Let's Practice - 6

Which ones are correct about Ansible ad-hoc commands?

Select one or more:

- ☐ Ad-hoc commands demonstrate the simplicity and power of Ansible. **Correct**
- ☐ Ad-hoc tasks can be used to reboot servers, copy files, manage packages and users, and much more.
- ☐ Ad-hoc commands are great for tasks you repeat rarely. **Correct**
- ☐ An Ansible ad-hoc command uses the /usr/bin/ansible command-line tool **Correct**
- ☒ ~~Ad-hoc commands are reusable.~~
- ☐ Ad-hoc commands are quick and easy **Correct**
- ☒ ~~You can not use every Ansible module in an ad-hoc task.~~