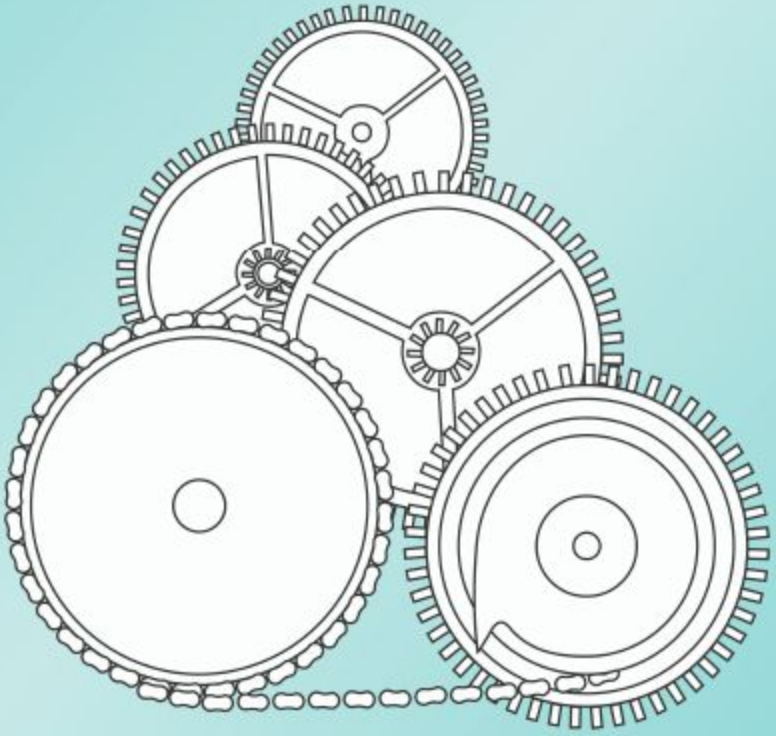
Управление ресурсами AWS при помощи Ansible

короткий гайд по автоматизации в облаках



Дэвид Клинтон

Bootstrap IT

Оглавление

Вступление

Разве вы не хотели бы иметь возможность просто взмахнуть волшебной палочкой, и чтобы все ресурсы вашей учетной записи AWS внезапно, как по волшебству, стали хорошо настроенной системой готовой удовлетворить все ваши потребности? Если у вас уже есть опыт работы с AWS, то вы знаете, насколько иногда сложно работать в консоли управления AWS, когда вы вручную управляете своими сервисами. И даже интерфейс командной строки AWS CLI, который является огромным шагом вперед, не всегда облегчает задачу.

В AWS существуют решения в виде собственного класса мощных инструментов, таких как CloudFormation или Elastic Kubernetes Service (***EKS***). Но ни один из этих вариантов не находится также близко к вашей существующей инфраструктуре и не использует столь же знакомый подход как Ansible. Если вы уже используете Ansible для своей локальной инфраструктуры, то подключение его к учетной записи AWS чаще всего может быть самым быстрым и безболезненным способом переноса инфраструктуры в облако.

Понимание преимуществ Ansible / AWS

Эта небольшая книга предназначена для быстрого ознакомления с декларативным подходом Ansible к работе с ресурсами AWS. Возможность «объявлять» конечные результаты конфигурации, которую вы хотите создать, а затем получить их, заставляя Ansible читать плейбук (***playbook***)— настоящая магия Ansible. При правильном планировании кажется удивительным насколько простым может быть выполнение сложных многоуровневых развертываний AWS.

Вот что мы сделаем:

- Установим инструменты и убедимся, что они могут взаимодействовать с учетной записью AWS;
- Воспользуемся HashiCorp Packer для создания кастомного AMI, который станет основой для более сложного развертывания Ansible;
- Организуем поиск документации, которая поможет вам в создании ваших собственных инфраструктурных проектов.
- Изучим способы более эффективной автоматизации процессов развертывания и управления с помощью скриптов динамической инвентаризации и плейбуков Ansible, которые могут перечислять и отключать работающие ресурсы.

Что же вам понадобится для извлечения максимальной пользы при прочтении этой книги? Вам необходимо быть знакомым с серверами, их рабочими нагрузками и вам обязательно понадобится учетная запись AWS и небольшой опыт работы с инструментами, которые предоставляет AWS. Если вы еще не уверены, что это вам подходит, или вы почувствуете пробелы в знаниях при прочтении этой книги, вы можете прочитать хотя бы несколько первых глав моих книг “Learn AWS in a Month of Lunches” или “Linux in Action” от издательства Manning. Или одну из множества моих статей, которые охватывают эти темы. Все это связано с моим сайтом Bootstrap IT[1].

Прямо сейчас мы займемся настройкой рабочей среды, через которую Ansible сможет общаться со всеми своими новыми «друзьями» в вашей учетной записи AWS.

Мне хотелось бы, чтобы вы могли легко выполнять задания дома, или в офисе, или пока вы едете в метро, или где бы вы этим ни занимались. Поэтому я создал веб-страницу по адресу bootstrapit.com/ansible[2], где я размещаю все фрагменты кода, используемые в книге. Вы также найдете ссылку на репозиторий github, в котором размещены некоторые из наиболее длинных плейбуков. Так вам не придется утомлять свои бедные и уставшие пальцы печатью.

---

[1] https://bootstrap-it.com

[2] https://bootstrap-it.com/ansible

Подготовка локального окружения

Как вы, наверное, уже знаете, Ansible — это инструмент управления конфигурацией и оркестровки, который позволяет вам писать файлы плейбука, декларирующие профиль программного обеспечения и идеальное состояние, которое вы хотите применить к целевому серверу. Эти серверы, известные как хосты, могут быть подготовлены практически для любых задач, которые вы только можете себе представить, с использованием практически любой комбинации программного обеспечения для работы практически на любой платформе.

В старые добрые времена, когда плейбук запускался на физическом сервере, Ansible использовал существующее SSH-соединение для безопасного входа на удаленный хост и создания вашего приложения. Но это не работает для рабочих нагрузок в AWS. Видите ли, поскольку инстансы EC2 и другая инфраструктура, которую вы хотите запустить, еще не существуют, не может быть «существующих» SSH-соединений. Вместо этого Ansible будет использовать Boto 3 - комплект разработки программного обеспечения (SDK), используемый AWS, который позволяет коду Python взаимодействовать с API AWS.

Использование AWS CLI для подключения Ansible

Для начала вам необходимо установить интерфейс командной строки AWS CLI. Мы не будем использовать сам интерфейс командной строки для чего-либо важного, но его установка предоставит нам все зависимости, которые нам дальше потребуются. Вы можете узнать, как установить AWS CLI на любую версию операционной системы на странице официальной документации AWS[3].

Работа с менеджером пакетов Python, PIP, - самый простой и популярный способ сделать это. Вот как можно установить сам PIP, а затем AWS CLI на машине Ubuntu:

```
1 $ sudo apt update
2 $ sudo apt install python3-pip
3 $ pip3 install awscli
```

Я должен отметить, что на момент написания книги Python 2 все еще существует… но это только пока. Поэтому иногда в вашей системе могут быть установлены обе версии: Python 2 и Python 3. Поскольку Python 2 скоро будет полностью устаревшим, вам, вероятно, не придется беспокоиться об указании python3 или pip3 в ваших командах: это должно работать автоматически.

После установки интерфейса командной строки запустите `aws configure` и введите свой идентификатор ключа доступа AWS (***access key ID***) и секретный ключ доступа (***secret access key ID***).

```
1 $ aws configure
2 $ cat .aws/credentials
```

---

[3] https://docs.aws.amazon.com/cli/latest/userguide/cli-chap-install.html

Вы можете получить ключи на странице Your Security Credentials в Консоли управления AWS. Вот пример как эти ключи должны выглядеть:

```
1 AccessKeyId: AKIALNZTQW6H3EFBRLHQ
2 SecretAccessKey: f26B8touguUBELGpdyCyc9o0ZDzP2MEUWNC0JNwA
```

Просто помните, что пара ключей, выданная root пользователю вашей учетной записи AWS, обеспечивает полный доступ ко всей вашей учетной записи AWS. Любой, кто сможет получить эти ключи, будет способен заказать услуги на шестизначную и даже семизначную сумму, поэтому будьте очень осторожны при их использовании и хранении. В идеале вам лучше ограничить риски, создав пользователя-с ограниченными правами в сервисе AWS Identify and Access Management (IAM) и использовать ключ, выданный этому пользователю.

Так зачем я всё это делаю? Ценность моего файла учетных данных AWS заключается в том, что Ansible достаточно умен, чтобы его искать, и, если в системной среде нет других ключей аутентификации, он будет использовать предоставленные. Скоро вы увидите, насколько это удобно. Однако вам следует знать о других способах управления аутентификацией для плейбуков Ansible, таких как использование ansiblevault или создание и последующий вызов файла aws_keys.yml. Но есть одна вещь, которую вам определенно НЕ следует делать — это хардкодить ключи в файлах плейбука, особенно если вы планируете загружать их в онлайн-репозиторий, такой как GitHub. Я быстро протестирую интерфейс командной строки, чтобы убедиться, что мы можем подключиться к AWS. Эта простая команда выведет список всех сегментов S3, которые у меня есть в этой учетной записи.

```
1 $ aws s3 ls
```

Теперь мы готовы установить ansible. Для этого я выберу pip3. Я мог бы с такой же легкостью использовать обычный репозиторий Ubuntu apt, но он, скорее всего, установит немного более старую версию. В зависимости от вашего сетевого подключения это займет некоторое время.

```
1 $ pip3 install ansible
```

Я убеждаюсь, что он установлен правильно, запустив ansible –version.

```
1 ansible –version
```

Здесь показана версия, которая была установлена:

```
2 ansible 2.8.5
```

Модули Ansible по умолчанию будут сохранены в одном из этих двух мест файловой системы:

```
3 config file = None
4 configured module search path =
5 ['/home/ubuntu/.ansible/plugins/modules',
6 '/usr/share/ansible/plugins/modules']
7 ansible python module location =
```

Здесь будут доступны другие модули и, что наиболее важно, исполняемый файл Ansible находится в каталоге /local/bin/ под домашним каталогом моего пользователя:

```
8 /home/ubuntu/.local/lib/python3.6/site-packages/ansible

9 executable location = /home/ubuntu/.local/bin/ansible
```

Мой пользователь здесь, кстати, называется ubuntu. Вы также можете видеть, что мы используем обновленную версию Python 3.

```
10 python version = 3.6.8 (default, Aug 20 2019, 17:12:48) [GCC 8.3.0]
```

Как я уже упоминал ранее, Ansible будет подключаться к AWS с помощью Boto SDK. Итак, нам нужно установить пакеты boto и boto 3. Для этого я также буду использовать PIP.

```
1 $ pip3 install boto boto3
```

Как только он будет установлен, мы сможем заняться по-настоящему важными делами. Мы приступим к ним в следующем разделе.

Тестирование Ansible с помощью простого плейбука

Это будет простая демонстрация концепции. Я создам пару файлов, продемонстрирую вам синтаксис, а затем запущу всё. Все примеры кода, которые вы увидите в этой главе, вы можете найти на странице bootstrap-it.com/ansible[4], скопировать и вставить.

Во-первых, я воспользуюсь любым текстовым редактором для создания файла hosts. Обычно файл hosts сообщает Ansible где он может найти удаленные серверы которые вы хотите подготовить, но в случае с AWS ресурсы, которые будут нашими хостами, еще не существуют, мы просто укажем Ansible на localhost, и boto будет обрабатывать соединения за кулисами. Вот как будет выглядеть содержимое этого файла:

```
1 [local]
2 localhost
```

Затем я создам файл плейбука, который я назову test.yml. Расширение yml указывает на то, что этот файл должен быть отформатирован с использованием синтаксиса языка YAML. Как вы можете видеть из текста файла, который я вставил чуть ниже, он начинается с трех тире, обозначающих начало файла, а затем тире с отступом, представляющей набор определений. Значением «хостов» может быть один или несколько удаленных компьютеров, но, как я уже сказал, мы оставим это на усмотрение локальной системы.

Следующий раздел включает задачи, которые мы хотим, чтобы Ansible выполнял. В этом примере будет использоваться модуль aws_s3 для создания новой корзины Amazon S3 (Simple Storage Service) в регионе us-east-1. Я дал ему столь странное имя, потому что корзины S3 требуют глобально уникальные имена - если имя, которое вы решили использовать, конфликтует с любым из уже существующих имен - операция завершится ошибкой.

```
1 ---
2       - name: Test s3
3                 hosts: local
4                 connection: local
5
6       tasks:
```

[4] https://bootstrap-it.com/ansible

```
7               - name: Create new bucket
8               aws_s3:
9                       bucket: testme817275b
10                      mode: create
11                      region: us-east-1
```

Я запускаю плейбук, вызывая команду ansible-playbook с помощью -i, чтобы указать файл hosts, а затем указываю на файл test.yml. Ansible должен выдать результат буквально через пару секунд. В случае успеха вы увидите «0» как значение «failed» и «1» как значение «ok».

```
1 $ ansible-playbook -i hosts test.yml
2 PLAY [Test s3] ****************************************************
3
4 TASK [Create new bucket] ******************************************
5
6 changed: [localhost]
7
8 PLAY RECAP *********************************************************
9 localhost: ok=1 changed=1 unreachable=0 failed=0 skipped=0
10 rescued=0 ignored=0
```

Если я еще раз проверю список корзин, я должен увидеть, что создалась новая:

```
1 $ aws s3 ls
2 2018-12-30 15:19:24 elasticbeanstalk-us-east-1-297972716276
3 2018-10-12 04:09:37 mysite548.com
4 2019-09-24 15:53:26 testme817275b
```

Это очень краткое введение в настройку окружения Ansible. Мы увидели, как использование Ansible с автоматически выделяемыми ресурсами Amazon, с традиционными хостами Ansible будет работать иначе. Вам понадобится другой набор инструментов аутентификации и управления. Мы пройдемся по процессу настройки среды Ansible и подключения ее к AWS, а затем запустили простой сценарий.

В следующей главе - вопреки тому, что вы могли ожидать - мы собираемся полностью игнорировать Ansible и вместо этого сосредоточимся на создании образов Amazon с использованием Hashicorp Packer. Конечно, в конце концов вы увидите, как это связано с Ansible.

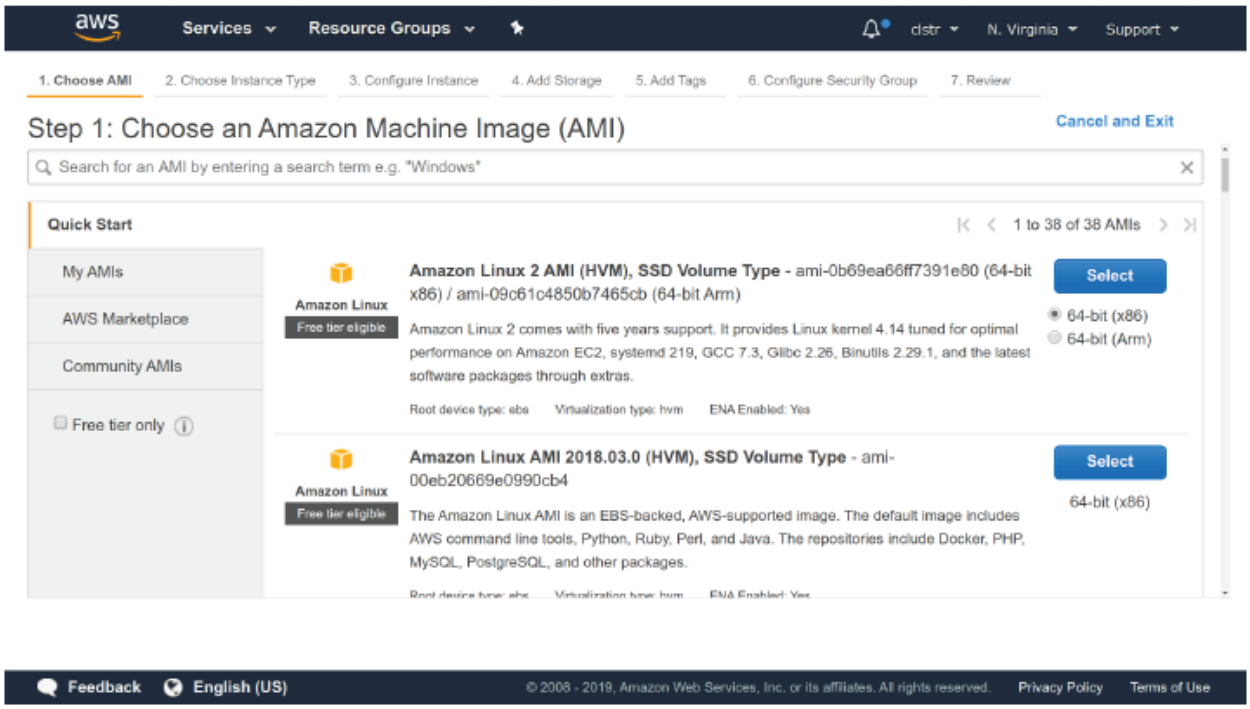
Использование HashiCorp Packer для создания EC2 Amazon Machine Images (AMI)

Чтобы полностью понять ценность HashiCorp Packer для Ansible/AWS, вам сначала необходимо понять, что такое Amazon Machine Image (AMI). Раньше, когда вы покупали себе новый сервер, вы присоединяли пустой жесткий диск или, что более вероятно, массив из нескольких пустых жестких дисков. Прежде чем вы смогли бы использовать свой сервер, вам было необходимо установить операционную систему, и только затем необходимое для работы приложение. На все это требовалось уделить значительное количество времени, особенно если вы где-то допустили ошибку при своей первой попытке.

Одним из поистине поразительных достижений облачных вычислений в целом и виртуализации в частности является возможность применять надежные и предсказуемые образы ОС к тому виртуального диска и запускать инстанс за считанные секунды - без ошибок.

Использование AMI AWS EC2

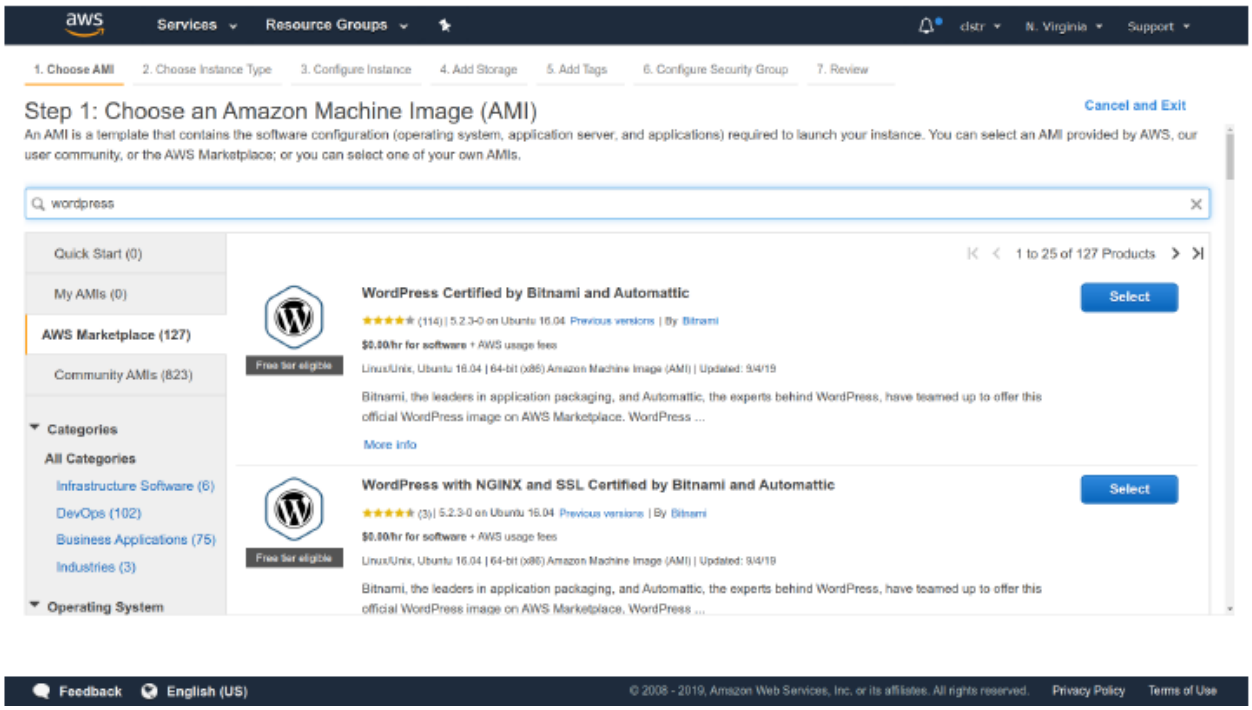
Пришло время запустить новый инстанс AWS EC2 из консоли управления используя образ (AMI) Ubuntu LTS. Вы можете пропустить некоторые шаги настройки конфигурации и сразу запустить его. Если вы в дальнейшем намерены повторять примеры за мной, то не забывайте отключать и удалять запущенные инстансы EC2 и другие ресурсы после того как они станут вам не нужны, так как за использование сервисов AWS может взыматься плата даже при наличии годового free tier.



Страница выбора AMI EC2 с вкладками коллекций, включая Quick Start и My AMI.

Есть тысячи других AMI доступных на AWS Marketplace, кроме того, вы можете создать свои создать образы. Например, вам требуется образ ОС Ubuntu с предустановленным WordPress, для которого потребуется лишь некоторая настройка на основе браузера после запуска. Вы можете

найти его на вкладках Marketplace или Community. Некоторые из них будут бесплатными (помимо эксплуатационных расходов EC2), а другие могут взимать плату за использование программного обеспечения.



Некоторые из доступных AMI с WordPress в AWS Marketplace

Если вы читаете книгу об Ansible, то, скорее всего, вы ищете эффективные способы предоставления стеков настраиваемого программного обеспечения. Вероятно, вас не будут интересовать какие-либо из доступных готовых образов. Конечно, вы можете запустить обычный стандартный AMI, самостоятельно настроить его и установить необходимо ПО, а после этого создать моментальный снапшот тома данных и зарегистрировать его как AMI. Но это утомительная рутинная работа, которая требует слишком много времени.

Почему бы вместо этого просто не объявить желаемое состояние для своего образа и позволить программному обеспечению позаботиться обо всех деталях, необходимых для превращения идеи в готовый AWS AMI? Вот тут-то и пригодится Packer, который мы рассмотрим далее в модуле.

Создание и запуск скрипта Packer

В этом разделе мы собираемся изучить правила синтаксиса, используемые для скриптов Packer, мы проверим, запустим скрипт и убедимся, что наш новый AMI полностью функционален, а затем немного глубже взглянем на то, где найти справку по синтаксису которая поможем вам в создании собственных образов.

Давайте вернемся на машину Ubuntu Ansible, и установим Packer, для этого мы будем использовать APT для установки упаковщика. Это займет всего несколько минут.

```
1 $ sudo apt install packer
```

После этого я добавляю код JSON в файл, который назову «*webserver*» — это простой веб-сервер, который мы будем создавать, я начну с официального Canonical Ubuntu AMI, а затем установлю Packer с помощью которого будет устанавливаться веб-сервер Apache. Когда мы закончим с этим,

у нас появится возможность запустить экземпляр на основе нашего нового образа, чтобы убедиться, что он работает. Для понимания полной картины вы можете изучить весь код файла:

```
{
        "variables": {
                "aws_access_key": "",
                "aws_secret_key": ""
        },
        "builders": [{
                "type": "amazon-ebs",
                "access_key": "{{user `aws_access_key`}}",
                "secret_key": "{{user `aws_secret_key`}}",
                "region": "us-east-1",
                "source_ami_filter": {
                        "filters": {
                                "virtualization-type": "hvm",
                                "name": "ubuntu/images/*ubuntu-bionic-18.04-amd64-server-*",
                                "root-device-type": "ebs"
                        },
                        "owners": ["099720109477"],
                        "most_recent": true
                },
                "instance_type": "t2.micro",
                "ssh_username": "ubuntu",
                "ami_name": "Apache Webserver {{timestamp}}"
        }
    ],
        "provisioners": [{
        "type": "shell",
        "inline": [
                "sleep 30",
                "sudo apt-get update",
                "sudo apt-get install -y apache2"
        ]
    }]
}
```

Теперь давайте рассмотрим синтаксис JSON по очереди. В этом файле три раздела: *переменные (**variables**)*, *строители (**builders**)* и *поставщики (**provisioners**)*. Переменные, которые я использую выше, могут содержать мои фактические ключи, но, как я уже упоминал в контексте скриптов Ansible, вам никогда не следует хардкодить ключи аутентификации в таких скриптах.

```
"variables": {
        "aws_access_key": "",
```

```
3       "aws_secret_key": ""
```

В данном случае я вообще не включаю никаких реальных значений, потому что, как мы видели с Ansible, Packer будет получать ключи, хранящиеся в файле учетных данных AWS, через системное окружение. Как и в случае с Ansible, у Packer есть и другие безопасные способы передачи учетных данных, в том числе в качестве аргументов командной строки или с использованием файла Shared Credentials.

В разделе «builders» устанавливается тип сборки «amazon-ebs». EBS - Elastic Block Store - это сервис AWS, который управляет томами виртуального хранилища, используемыми инстансами EC2 в качестве хранилища данных. Поскольку образы AMI загружаются на тома EBS, процесс сборки Packer характеризуется как «amazon-ebs». Эта строка, по сути, сообщает Packer, что весь последующий код следует интерпретировать в этом контексте.

```
1 "builders": [{
2       "type": "amazon-ebs",
```

Затем применяются ключи учетных данных, упомянутые выше, указывается регион us-east-1, а затем, базовый образ AMI, с которого мы начинаем.

```
1       "access_key": "{{user `aws_access_key`}}",
2       "secret_key": "{{user `aws_secret_key`}}",
3       "region": "us-east-1",
```

Если вам уже известен ID AMI, который вы ищете, вы можете определить его с помощью «source_ami»: и его ID. Вот как это будет выглядеть с использованием идентификатора образа Ubuntu 18.04, который мы видели в консоли управления AWS. Не забывайте про запятую.

```
1 "source_ami": ami-07d0cf3af28718ef,
```

В противном случае вы можете идентифицировать исходный AMI с помощью source_ami_filter, который будет искать среди тысяч доступных образов тот, который лучше всего соответствует параметрам фильтра. В этом примере указан тип виртуализации HVM и тип корневого устройства ebs. Сложнее всего добавить саму ОС. Здесь я указываю версию 18.04, которая использует «bionic» в качестве псевдонима, архитектуру md64 и используется в качестве сервера.

```
        "source_ami_filter": {
                "filters": {
                        "virtualization-type": "hvm",
                        "name": "ubuntu/images/*ubuntu-bionic-18.04-amd64-server-*",
                        "root-device-type": "ebs"
                },
                "owners": ["099720109477"],
                "most_recent": true
        },
```

Значение «owners» указывает Packer на необходимости поиска образа, принадлежащего компании Canonical, которая поддерживает Ubuntu.

Как только мы определили базовый AMI, мы укажем тип инстанса. Я выбираю t2.micro, потому что это будет бесплатно, если моя учетная запись будет по-прежнему имеет уровень бесплатного пользования (***free tier***). Имя, которое я хотел бы дать пользователю системы, - «ubuntu» - что-то вроде стандартной практики для виртуальных машин Ubuntu. Я бы хотел дать AMI удобочитаемое имя «Apache Webserver», за которым следует отметка времени при создании. Это упростит идентификацию моего AMI на экране, когда ваша учетная запись станет более загруженной.

```
                "instance_type": "t2.micro",
                "ssh_username": "ubuntu",
                "ami_name": "Apache Webserver {{timestamp}}"
        }
], 
```

В разделе «Provisioners» мы устанавливаем и настраиваем конкретные приложения, которые должны обслуживать наш инстанс. Наш пример представляет собой оболочку Bash, в которой мы ждем 30 секунд, чтобы убедиться, что все работает, затем синхронизируем локальный индекс APT с онлайн-репозиториями и, наконец, устанавливаем пакет веб-сервера Apache 2. Параметр «-y» автоматизирует ответ «yes» в процессе установки, что необходимо для обработки автоматических сценариев.

```
        "provisioners": [{
                "type": "shell",
                "inline": [
                        "sleep 30",
                        "sudo apt-get update",
                        "sudo apt-get install -y apache2"
                ]
        }]
}
```

Пусть Packer проверит синтаксис JSON:

```
$ packer validate webserver.json
Template validated successfully.
```

А теперь мы можем дать Packer возможность приступить к созданию нашего AMI:

```
$ packer build webserver.json
```

В процессе работы вы увидите множеством строк вывода на экране вашего терминала, чтобы не наблюдать за этим вы можете перейти на панель управления экземплярами EC2 в консоли управления AWS и посмотреть, как Packer запустил экземпляр EC2, в котором будет выполняться наша оболочка. Как вы увидите, скоро мы отключим этот инстанс навсегда, поскольку в нем больше нет необходимости. После этого на его месте вы увидите (щелкнув вкладку AMI на панели инструментов EC2), новый «ожидающий» AMI.

Вернитесь к терминалу, где, вероятно, будет завершена операция сборки Packer. Вот как должны выглядеть последние несколько строк вывода:

```
Build 'amazon-ebs' finished.

==> Builds finished. The artifacts of successful builds are:
--> amazon-ebs: AMIs were created:

us-east-1: ami-088229dc6681fe6d5
```

Обратите внимание на ID, присвоенный нашему новому AMI (в данном случае ami-088229dc6681fe6d5). Я возьму его и добавлю в команду AWS CLI «describe-images» для отображения важной статистики AMI с использованием форматирования JSON.

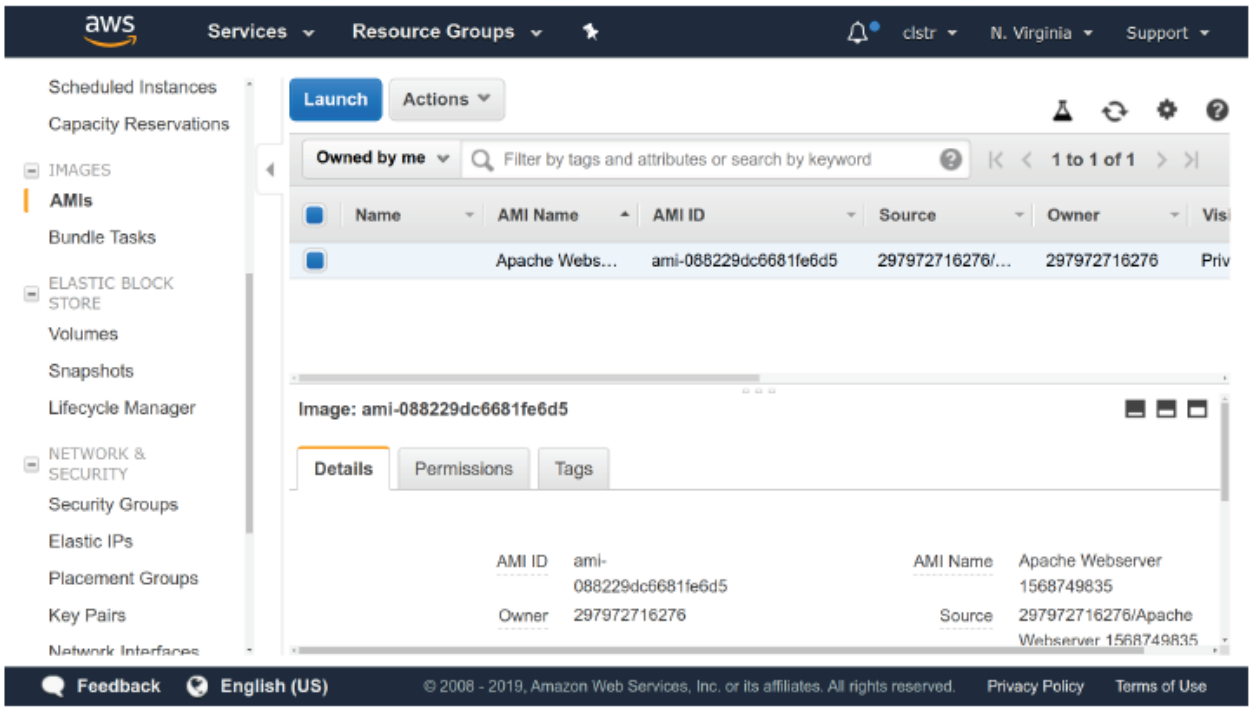
```
$ aws ec2 describe-images --image-ids ami-088229dc6681fe6d5
{
    "Images": [
        {
            "Architecture": "x86_64",
            "CreationDate": "2019-09-17T19:52:32.000Z",
            "ImageId": "ami-088229dc6681fe6d5",
            "ImageLocation": "297972716276/Apache Webserver 1568749835",
            "ImageType": "machine",
            "Public": false,
            "OwnerId": "297972716276",
            "State": "available",
            "BlockDeviceMappings": [
                {
                    "DeviceName": "/dev/sda1",
                    "Ebs": {
                        "DeleteOnTermination": true,
                        "SnapshotId": "snap-0ed35db4e3c586c2a",
                        "VolumeSize": 8,
                        "VolumeType": "gp2",
                        "Encrypted": false
                    }
```

```
                        },
                        {
                                "DeviceName": "/dev/sdb",
                                "VirtualName": "ephemeral0"
                        },
                        {
                                "DeviceName": "/dev/sdc",
                                "VirtualName": "ephemeral1"
                        }
                ],
                "EnaSupport": true,
                "Hypervisor": "xen",
                "Name": "Apache Webserver 1568749835",
                "RootDeviceName": "/dev/sda1",
                "RootDeviceType": "ebs",
                "SriovNetSupport": "simple",
                "VirtualizationType": "hvm"
        }
    ]
}
```

Здесь мы видим, что он создан для 8 ГБ диска, он не зашифрован и что корневой диск будет обозначен в инстансе, который мы запускаем с помощью этого AMI, как /dev/sda1. Это может пригодиться, если нам нужно работать непосредственно с «живым» инстансом . Наконец, обратите внимание на запись «Snapshot ID». Packer AMI будут генерировать эти снапшоты, и они не будут удаляться автоматически, даже если вы отмените регистрацию поддерживаемого им AMI. Их не так уж много, но, если вы будете создавать много подобных вещей, они будут накапливаться. Возможно, вы захотите удалить неиспользуемые снапшоты.

```
"DeviceName": "/dev/sda1",
"Ebs": {
        "DeleteOnTermination": true,
        "SnapshotId": "snap-0ed35db4e3c586c2a",
        "VolumeSize": 8,
        "VolumeType": "gp2",
        "Encrypted": false
```

После завершения создания AMI вы можете нажать большую синюю кнопку «Launch» на панели управления AMI в AWS (показанной на рисунке ниже) и настроить его для нового инстанса. Если вы хотите, чтобы пользователи имели доступ к вашему веб-серверу Apache, вам необходимо открыть хотя бы порт 80 с помощью группы безопасности. После запуска инстанса вы должны скопировать его общедоступный IP-адрес из панели управления инстансами и указать его в браузере. Вы должны увидеть приветственную страницу Apache.



Панель управления AMI EC2 с образом Apache для сборки Packer и видимой синей кнопкой «Launch»

Мы закончили с этим инстансом. Если у вас нет других планов, вы должны удалить его, а затем, при желании, также отменить регистрацию самого AMI (это может стоить вам нескольких пенни в месяц, поддерживая его на неопределенный срок).

Работа с Packer синтаксис JSON

В предыдущем разделе я показал вам, как создать и запустить простой AMI с веб-сервером на базе Ubuntu. Но что, если вы не хотите использовать Ubuntu? Или вам не нужен веб-сервер? Если бы вы после прочтения этой книги не умели ничего кроме создания и запуска простого AMI с веб-сервером на основе Ubuntu, тогда эта книга ничего бы не стоила, не так ли?

Я не могу показать вам все параметры конфигурации, предоставленных Packer, которые могут вам понадобиться. Вместо этого я дам вам информацию о ресурсах, где вы можете найти документацию, которая поможет вам в процессе написания собственных скриптов.

В первую очередь вам следует посетить главную страницу документации Packer по адресу packer.io/docs[5]. Там вы найдете ссылки на установку Packer и описание того, как работают основные инструменты. В частности, вы можете изучить страницы с информацией о *Builders* и *Provisioners*.

Например, в разделе *Builders* вы увидите ссылку на Amazon EC2[6], где вы найдете фрагменты кода JSON представляющие различные способы подключения к API AWS из скрипта Packer для создания AMI с EBS в качестве хранилища данных.

---

[5] https://packer.io/docs/

[6] https://packer.io/docs/builders/amazon.html

Точно так же параметры *Provisioners* включают, среди многих других, параметр Shell, который мы использовали в нашем собственном скрипте для вызова команд Bash во временном инстансе EC2.

Еще хорошим способом является поиск в Интернете с описанием того, что вы пытаетесь выполнить с добавлением слова «packer». В результате, велика вероятность, что вы найдете полезные примеры скриптов других людей. Особое внимание уделите результатам, которые ведут на публичные репозитории такие как GitHub[7] или форумы StackOverflow[8].

---

[7] https://github.com/

[8] https://stackoverflow.com/

Использование Ansible для управления ресурсами AWS

Работа в Ansible заключается во взаимодействии со скриптами, которые называются плейбуками. Все, что вы включите в такой скрипт, станет работающей инфраструктурой, когда он будет запущен и успешно выполнен. Плейбуки могут ссылаться на внешние файлы и ресурсы что позволяет грамотно организовывать данные и упростить высоконагруженный код особенно для сложных развертываний.

Всё это мы рассмотрим в этой главе. Но сначала давайте разберем относительно простой скрипт, который будет развертывать инстанс на основе Packer AMI, созданного в предыдущей главе.

Создание окружения для запуска AMI

Для успешного запуска инстанса EC2 требуется нечто большее, чем просто AMI. Вам также понадобится настроенная группа безопасности чтобы разрешить соответствующий доступ чтобы вы могли создавать сеансы SSH для администрирования.

Для начала вот как будет выглядеть полный плейбук, который я назову short-ec2-playbook.yml:

```
---
    - name: Launch an EC2 Instance
      hosts: local
      connection: local
      gather_facts: False
      tags: provisioning

      vars:
            instance_type: t2.micro
            security_group: webserver2
            image: ami-088229dc6681fe6d5
            keypair: mykey
            region: us-east-1
            count: 1

      tasks:

            - name: Create a security group
              local_action:
                    module: ec2_group
                    name: "{{ security_group }}"
                    description: Open ports 22, 80, and 443
                    region: "{{ region }}"
                    rules:
                          - proto: tcp
                            from_port: 22
```

```
                                to_port: 22
                                cidr_ip: 0.0.0.0/0
                              - proto: tcp
                                from_port: 80
                                to_port: 80
                                cidr_ip: 0.0.0.0/0
                              - proto: tcp
                                from_port: 443
                                to_port: 443
                                cidr_ip: 0.0.0.0/0
                    rules_egress:
                              - proto: all
                                cidr_ip: 0.0.0.0/0
                    register: basic_firewall

          - name: Launch EC2 Instance
                    local_action: ec2
                              group={{ security_group }}
                              instance_type={{ instance_type}}
                              image={{ image }}
                              wait=true
                              region={{ region }}
                              keypair={{ keypair }}
                              count={{count}}
            register: ec2
```

Поскольку вы быстро просмотрели файл, я надеюсь, вы заметили, что в разделе vars задан ряд важных переменных. Мы будет передавать значения переменным по мере их вызова в остальной части скрипта. Например, переменная instance_type принимает значение t2.micro, который является легким и недорогим типом инстанса EC2, который может использоваться бесплатно для учетных записей, у которых еще не истек free tier. Переменной image присваивается ID AMI, созданного с помощью Packer, region сообщит плейбуку, что нужно запустить наш инстанс в регионе Amazon us-east-1, а count означает, что нам нужен только один инстанс за раз:

```
vars:
        instance_type: t2.micro
        security_group: webserver2
        image: ami-088229dc6681fe6d5
        keypair: newcluster
        region: us-east-1
        count: 1
```

Но как насчет значений security_group и keypair? Откуда они берутся? Что ж, группа безопасности webserver2 будет создана в разделе tasks - просто убедитесь, что у вас еще нет группы

безопасности с таким именем в этом регионе. Если она уже существует, то просто измените имя группы в разделе vars.

А пара ключей? Здесь вам пригодится AWS CLI. Вы можете создать новый ключ и управлять закрытым ключом с помощью команды aws ec2 create-key-pair. Но если у вас уже есть пара ключей, и вы просто хотите воспользоваться ей, вы должны использовать эту команду:

```
$ aws ec2 describe-key-pairs
KEYPAIRS 15:af:af:27:4e:05:b2:bd:ad:01:c2:97:ce:03:21:77 mykey
```

Выходные данные для любых существующих ключей в вашей учетной записи не будут включать сам ключ, а будет достаточно идентификационной информации.

Кстати, если вы не уверены в ID образа, связанном с вашим AMI, вы можете быстро выполнить поиск по всем образам, принадлежащим вашей учетной записи, с помощью aws ec2 description-images --owners self. Этот вариант (по крайней мере работает на компьютерах с Linux) сузит вывод только до точной информации, которую вы ищете:

```
$ aws ec2 describe-images --owners self | grep ImageId
"ImageId": "ami-088229dc6681fe6d5",
```

Возвращаясь к нашему скрипту, первая из двух задач связана с созданием группы безопасности для нашего инстанса EC2. Задача сначала вызывает модуль ec2_action, применяет значение переменной security_- group, которую мы установили ранее, добавляет дополнительное описание и устанавливает регион в соответствии с нашими предпочтениями:

```
    tasks:

        - name: Create a security group
            local_action:
                module: ec2_group
                name: "{{ security_group }}"
                description: Open ports 22, 80, and 443
                region: "{{ region }}"
```

Затем плейбук добавит некоторые правила для группы безопасности. Открытие портов: 22 (SSH), 80 (незащищенный доступ HTTP) и 443 (безопасный доступ HTTPS) с любого клиентского компьютера в любом месте в Интернете (cidr_ip: 0.0.0.0/0). Исходящий доступ из инстанса возможен с использованием любого порта или протокола.

```
        rules:
            - proto: tcp
              from_port: 22
              to_port: 22
              cidr_ip: 0.0.0.0/0
            - proto: tcp
              from_port: 80
              to_port: 80
              cidr_ip: 0.0.0.0/0
            - proto: tcp
              from_port: 443
              to_port: 443
              cidr_ip: 0.0.0.0/0
              rules_egress:
            - proto: all
              cidr_ip: 0.0.0.0/0
              register: basic_firewall
```

Наконец, вторая задача соберет все части вместе (новую группу безопасности, указанный тип инстанса, образ, регион и key pair, а также количество инстансов, которые вы хотите подготовить) и запустит инстанс:

```
    - name: Launch EC2 Instance
        local_action: ec2
                group={{ security_group }}
                instance_type={{ instance_type}}
                image={{ image }}
                wait=true
                region={{ region }}
                keypair={{ keypair }}
                count={{count}}
                register: ec2
```

Предполагая, что у вас есть файл hosts, который мы использовали пару глав назад, вы будете готовы запустить свой инстанс. Вы можете получить предупреждение о зависимостях, но оно не будет критическим. Вот как будет выглядеть успешный запуск:

```
$ ansible-playbook -i hosts short-ec2-playbook.yml
/usr/lib/python3/dist-packages/requests/__init__.py:80:
   RequestsDependencyWarning: urllib3 (1.25.6) or chardet (3.0.4)
   doesn't match a supported version!
   RequestsDependencyWarning)

PLAY [Launch an EC2 Instance] *************************************

```

```
TASK [Create a security group] ************************************
ok: [localhost -> localhost]

TASK [Launch EC2 Instance] ****************************************
^Fchanged: [localhost -> localhost]

PLAY RECAP ********************************************************
localhost: ok=2 changed=1 unreachable=0 failed=0 skipped=0 rescued=0
ignored=0
```

Перейдите на панель управления инстансом EC2 в Консоли управления AWS, чтобы убедиться, что ваш инстанс работает. Как и раньше, вы можете скопировать публичный IP-адрес и загрузить страницу Apache. Поскольку вы указали свою пару ключей, вы также можете войти на сервер через SSH (заменив имя вашей пары ключей и IP-адрес вашего экземпляра на приведенные здесь):

```
ssh -i mykey.pem ubuntu@3.85.145.243
```

Создание более сложного устойчивого окружения

Этот пример, я нашел его в довольно старом публичном репозитории GitHub, принадлежащем человеку, который называет себя «arbabnazar» [9]. Я не знаю о нем ничего кроме того, что он создал этот репозиторий, но мне определенно нравится, как он создает скрипты Ansible.

Вы можете клонировать ресурсы Ansible WordPress «Arbabnazar's» с помощью этой команды, а затем перейти в каталог ansible-aws-vpc-ha-wordpress, который будет создан.

```
$ git clone https://github.com/arbabnazar/ansible-aws-vpc-ha-wordpress.git
```

Перечисление содержимого этого каталога покажет вам что-то вроде этого:

```
$ ls ansible-aws-vpc-ha-wordpress
LICENSE README.md ansible.cfg aws common hosts
site.yml wordpress
```

Обратите внимание, что три из этих записей (aws, command и wordpress) сами по себе являются каталогами, что говорит о том, что ресурсы могут быть довольно глубоко вложены в дерево иерархии. Если вы распечатаете содержимое этого файла site.yml, вы поймете, насколько глубока эта вложенность:

---

[7] https://github.com/

```
~/tree/ansible-aws-vpc-ha-wordpress$ cat site.yml
---
        - hosts: local
          connection: local
          gather_facts: no
          vars_files:
                  - aws/vars/tags.yml
                  - aws/vars/vpc.yml
                  - aws/vars/ec2_key.yml
                  - aws/vars/rds.yml
                  - aws/vars/webserver.yml
                  - aws/vars/elb.yml
                  - aws/vars/route53.yml
          tasks:
                  - include: aws/tasks/vpc.yml
                  - include: aws/tasks/ec2_key.yml
                  - include: aws/tasks/webserver.yml
                  - include: aws/tasks/rds.yml
                  - include: aws/tasks/elb.yml
                  - include: aws/tasks/route53.yml

        - hosts: webserver
          sudo: True
          remote_user: ubuntu
          gather_facts: True
          pre_tasks:
                  - include_vars: rds_info.yml
          roles:
                  - common
                  - wordpress
```

Вместо того, чтобы складывать все в один, очень длинный и запутанный файл, в этом проекте используется интуитивно понятная система файлов и каталогов. Файлы в основном разделены на два домена: переменные (aws/vars/…) и задачи (aws/tasks/…). Файлы vars содержат подробную информацию для управления широким спектром переменных. Например, файл aws/vars/route53.yml, показанный ниже, содержит домен, который будет использоваться для сайта (rbgeek.com). Очевидно, вы захотите отредактировать это значение в соответствии со своими потребностями.

```
---
domain: "rbgeek.com"
```

Я использовал команду Linux tree, чтобы отобразить всю иерархию файлов, клонированную командой git:

```
$ tree
.
├── LICENSE
├── README.md
├── ansible.cfg
├── aws
│   ├── README.md
│   ├── tasks
│   │   ├── ec2_key.yml
│   │   ├── elb.yml
│   │   ├── rds.yml
│   │   ├── route53.yml
│   │   ├── vpc.yml
│   │   └── webserver.yml
│   └── vars
│       ├── ec2_key.yml
│       ├── elb.yml
│       ├── rds.yml
│       ├── route53.yml
│       ├── tags.yml
│       ├── vpc.yml
│       └── webserver.yml
├── common
│   ├── README.md
│   ├── tasks
│   │   └── main.yml
│   ├── templates
│   │   ├── 10periodic.j2
│   │   └── 50unattended-upgrades.j2
│   └── vars
│       └── main.yml
├── hosts
├── site.yml
└── wordpress
    ├── README.md
    ├── files
    │   ├── php.ini
    │   └── www.conf
    ├── handlers
    │   └── main.yml
    ├── tasks
    │   ├── deploy.yml
```

```
43 |   ├── main.yml
44 |   ├── nginx.yml
45 |   └── php5.yml
46 ├── templates
47 |   ├── nginx.conf.j2
48 |   ├── virtualhost.conf.j2
49 |   └── wp-config.php.j2
50 └── vars
51 └── main.yml
52
53 13 directories, 36 files
```

Обратите особое внимание на наличие полезных файлов README.md в каждом каталоге, а также файлов конфигурации и файлов YAML в каталоге wordpress. Вот, например, то, что содержится в файле wordpress/tasks/nginx.yml:

```
1 ---
2       - name: Install the Nginx Repository
3         apt_repository:
4                 repo: 'ppa:nginx/stable'
5                 state: present
6
7       - name: Update the Repositories
8         apt:
9                 update_cache: yes
10
11      - name: Install Nginx
12        apt:
13                name: nginx
14                state: installed
15       notify:
16                - Restart Nginx
17
18      - name: Write the modified nginx.conf
19        template:
20                src: nginx.conf.j2
21                dest: /etc/nginx/nginx.conf
22                backup: yes
23        notify:
24                - Restart Nginx
25
26      - name: Disable Default Nginx Site
27        file:
28                dest: /etc/nginx/sites-enabled/default
29                state: absent
```

```
30        notify:
31                - Restart Nginx
32
33      - name: Delete the Default index page in Nginx
34        file:
35                dest: /var/www/html/index.nginx-debian.html
36                state: absent
```

Как вы можете заметить, это плейбук, который определяет состояние и конфигурацию программного пакета веб-сервера Nginx, который представит WordPress всему миру. Он станет важным элементом инфраструктуры, которую вы собираетесь запустить.

А может и нет. Это может работать не так, как вы ожидали. Я попытался запустить этот красиво оформленный проект Ansible с помощью этой команды:

```
1 $ ansible-playbook -i hosts site.yml
```

Но вместо мгновенного запуска WordPress я столкнулся с несколькими предупреждениями об устаревании и, наконец, с критической ошибкой. В чем была проблема? Моя первая подсказка заключалась в том, что после просмотра репозитория с исходным кодом я увидел, что многие файлы датированы еще 2015 годом. Это очень старый проект. Конечно, я уверен, что мы сможем решить все проблемы, но дело не в этом. В конце концов, было две причины, по которым я показываю вам этот проект, и ни одна из них не заключалась в том, чтобы научить вас развертывать WordPress с помощью Ansible.

Фактически, проект в этой главе показан для того, чтобы (1) дать вам пример того, как вы можете придать вашему развертыванию Ansible интеллектуальную и хорошо продуманную структуру. И (2) чтобы предупредить вас о некоторых потенциальных проблемах использования готовых решений Ansible, которые вы найдете в Интернете.

Какие потенциальные проблемы? А как насчет предупреждений? В них говорится, что мне следует избегать определенных конфигураций, которые больше не считаются передовыми методами администрирования. Одним из примеров является настройка `sudo: True` в файле sites.yml. Его следует заменить более безопасным методом `begin_user`. Старый код часто сопровождается старыми методами кодирования.

Все предупреждения можно проигнорировать или исправить. Но вызов модуля `ansbile_facts` был критическим. Вот сообщение, которое сопровождало сбой:

```
1 "msg": "This module has been removed. The module documentation for
2 Ansible-2.4 may contain hints for porting"}
```

Опять же, я уверен, что есть способы решить эту проблему и обновить проект. Но пока речь идет о процессе, а не о результатах.

Еще одна заметка о публичных репозиториях. НИКОГДА не запускайте код, который вы не писали сами, если вы точно не понимаете, что будет делать каждая строка. Я еще не слышал ужасных историй, но одной мысли о том, какой ущерб вредоносные сценарии Ansible могут нанести вашей учетной записи AWS, достаточно, чтобы не дать мне уснуть по ночам.

Вас предупредили.

Использование Ansible для управления жизненным циклом ресурсов AWS

В предыдущих главах вы видели, насколько легко можно автоматизировать создание сложных и надежно воспроизводимых развертываний на AWS. Однако отслеживать их - совсем другая проблема. Раньше, когда рабочие нагрузки Ansible выполнялись на ваших собственных серверах, вы выполняли задачи инвентаризации через файл hosts. Там, как объясняется в документации Ansible[10], вы можете отобразить свои удаленные хосты сложными способами, включая их организацию в группы.

Но здесь ничего из этого не сработает. Как я уже писал ранее, инстансы EC2, которые вы собираетесь использовать для следующего развертывания, еще не существуют, а это означает, что невозможно представить вашу доступную инфраструктуру, пока триггер не будет задействован. Итак, как вы должны визуализировать то, что у вас сейчас работает, с точки зрения Ansible?

Добро пожаловать в *динамическую инвентаризацию* (***dynamic inventory***).

Отслеживание с помощью динамической инвентаризации Ansible

Предполагая, что у вас установлен Boto и ваши учетные данные AWS доступны в системной среде (все, что мы установили ранее) вам нужно будет только загрузить два файла из репозитория Ansible с GitHub.

```
1 $ wget https://raw.githubusercontent.com/ansible/\
2         ansible/devel/contrib/inventory/ec2.py
3 $ wget https://raw.githubusercontent.com/ansible/\
4         ansible/devel/contrib/inventory/ec2.ini
```

В системах Linux вам также необходимо добавить разрешения на выполнение в файл ec2.py:

```
1 $ chmod +x ec2.py
```

ec2.py — это скрипт, который будет использовать ваши учетные данные AWS для инвентаризации всех запущенных ресурсов EC2. Это будут даже ресурсы, запущенные независимо от Ansible. Вся эта проверка может занять некоторое время и часто возвращает больше информации, чем вам необходимо. Таким образом, вы можете контролировать область поиска скрипта, используя второй файл: ec2.ini. По умолчанию ec2.py предполагает найти ec2.ini в том же каталоге. Если вам нужно, чтобы он находился в другом месте, вы можете обновить значение PATH в самом файле скрипта ec2.py.

В этом примере с настройками ec2.ini по умолчанию скрипт будет искать ресурсы учетной записи во всех регионах AWS, за исключением двух регионов (оба ограниченных региона: один принадлежит правительству США, а другой контролируется правительством Китая), которым присвоены значения в regions_exclude :

---

[10] https://docs.ansible.com/ansible/latest/user_guide/intro_inventory.html

```
# AWS_DEFAULT_REGION environment variable will be read to determine the region.
regions = all
regions_exclude = us-gov-west-1, cn-north-1
```

Во втором примере сценарий должен игнорировать все ресурсы, которые не используют тип экземпляра t1.micro или имеют тег env со значением staging:

```
# Retrieve only t1.micro instances OR instances with tag env=staging
instance_filters = instance-type=t1.micro,tag:env=staging
```

Вы можете указать скрипту ec2.py выполнить простую инвентаризацию с помощью команды ./ec2.py --list. Если в вашем аккаунте нет подходящих ресурсов, вы увидите простой вывод:

```
$ ./ec2.py --list
{
        "_meta": {
                "hostvars": {}
        }
}
```

Если у вас запущен инстанс EC2, вы получите несколько окон, заполненных метаданными, описывающими инстанс и его среду. Как только вы получите представление о том, что доступно, вы, вероятно, захотите отфильтровать вывод, чтобы быстро перейти к необходимой информации. Если, например, вы хотите видеть только ID AMI, вы можете просто передать вывод по конвейеру и отфильтровать его по строке ami.

```
$ ./ec2.py --list | grep ami
        "ec2_ami_launch_index": "0",
        "ec2_image_id": "ami-07d0cf3af28718ef8",
        "ec2_ami_launch_index": "0",
        "ec2_image_id": "ami-088229dc6681fe6d5",
"ami_07d0cf3af28718ef8": [
"ami_088229dc6681fe6d5": [
```

Ищете IP-адреса своих экземпляров? фильтр для ansible_host:

```
$ ./ec2.py --list | grep ansible_host
        "ansible_host": "3.85.192.121",
        "ansible_host": "54.198.179.176",
```

Отключение ресурсов EC2

Все хорошее в конце концов должно подойти к своему концу. Однажды вы захотите отключить используемые ресурсы и воспользоваться хорошим плейбуком на Ansible для работы — это звучит замечательно, не так ли? Вот как работает этот плейбук:

```
- hosts: local
        connection: local
        vars:
                region: us-east-1
        tasks:
                - name: Get EC2 info
                  ec2_instance_facts:
                          region: "{{ region }}"
                  register: ec2
        - debug: var=ec2

        - name: Shut Down EC2 Instances
          ec2:
                instance_ids: '{{ item.instance_id }}'
                state: absent
                region: "{{ region }}"
           with_items: "{{ ec2.instances }}"
```

Давайте разберем это шаг за шагом. Раздел vars определяет регион, в котором мы работаем.

```
        vars:
                region: us-east-1
```

Первая задача выполнит своего рода внутреннюю инвентаризацию всех ресурсов EC2 в указанном регионе:

```
        tasks:
                - name: Get EC2 info
                  ec2_instance_facts:
                          region: "{{ region }}"
                   register: ec2
```

Последняя задача применит состояние «absent» ко всем экземплярам с идентификаторами, соответствующими тем, которые были обнаружены при первоначальной инвентаризации:

```
        - name: Shut Down EC2 Instances
          ec2:
                instance_ids: '{{ item.instance_id }}'
                state: absent
                region: "{{ region }}"
           with_items: "{{ ec2.instances }}"
```

Все очень просто и аккуратно. Если вы сделали это самостоятельно, зайдите в консоль управления еще раз, чтобы убедиться, что все действительно отключено.

Заключение

Ничего не поделаешь. Я имею в виду: это все, что я могу вам рассказать об использовании Ansible для управления ресурсами на AWS. Конечно, это не конец вашей истории. Для реализации ваших собственных проектов потребуется множество настроек, обновлений и конфигураций. Вы столкнетесь с проблемами и будете пытаться придумать, как их решить.

Но я надеюсь, что я показал вам по крайней мере достаточно основ, чтобы вы смогли сделать первые шаги - и знали, куда обратиться за помощью, когда дела пойдут тяжело.

Если вы обнаружили в книге проблему (или много проблем), сообщите мне об этом[11]. Если вы нашли эту книгу полезной, напишите отзыв на Amazon. Эта информация поможет другим людям.

Будьте на связи и удачи!

Дэвид Клинтон

Bootstrap IT[12]

---

[11] https://bootstrap-it.com/?page_id=90

[12] https://bootstrap-it.com/ansible