Введение в DevOps

Обзор инструментов для автоматизации процесса запуска операционной системы. Введение в AWS. Обзор утилит от компании Hashicorp (Packer, Vagrant).

[Введение](#_tqv6hpz19jrb)

[Введение в DevOps](#_2cmpbb5ivi45)

[Continuous Integration](#_xs86rtqoarpy)

[Введение в AWS](#_ecyqk419ztil)

[Packer](#_xvi2olvasfmf)

[Установка Packer](#_kzq7laxuq8cx)

[Создание образа](#_tr75cj1uwnni)

[Provision](#_6m57n096osfl)

[Vagrant Boxes](#_bwg8elh0zt7)

[Домашнее задание](#_qk61dk4of60m)

[Дополнительные материалы](#_y937sk8fclye)

[Используемая литература](#_uvp6qax5r1ok)

# 

# Введение

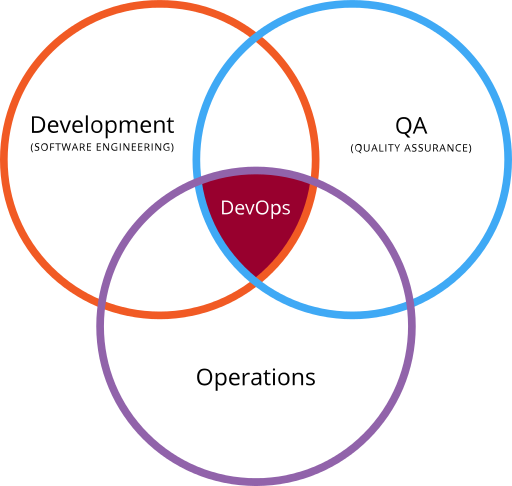
Сегодня мы поговорим об автоматизации – автоматизации установки FreeBSD и автоматизации с помощью FreeBSD, научимся работать с утилитами от компании Hashicorp: Packer и Vagrant. Компания Haschicorp занимается разработкой утилит для DevOps: Vagrant для регистрации виртуальных машин, Terraform для автоматизации облаков, Vault – утилита для хранения ключей шифрования, паролей, Nomad – scheduler для управления жизненным циклом задач, запускаемых на группе серверов, Consul – это децентрализованный отказоустойчивый discovery-сервис и т. д.

Сначала разберем Packer, затем коснемся AWS (Amazon Web Services) и научимся делать автосборку под VirtualBox с использованием Vagrant.

# Введение в DevOps

Все чаще звучит слово DevOps, его можно увидеть в заголовках вакансий. Что это за профессия такая, что за человек такой? На самом деле нет такой профессии DevOps, как и нет методологии Agile. Agile – это «гибкий», это не методология «Agile», а гибкая методология. Похожая ситуация и с DevOps. DevOps – это методология, подход, культура.

В 2008 году на Agile Toronto Conference впервые прозвучал термин «Agile Infrastructure», а в 2009 уже и термин «DevOps». Часто можно встретить следующую диаграмму:



By Devops.png: Rajiv.Pantderivative work: Wylve – This file was derived from Devops.png:, CC BY 3.0, <https://commons.wikimedia.org/w/index.php?curid=20202905>.



Автор: McCow – собственная работа, CC BY-SA 4.0, <https://commons.wikimedia.org/w/index.php?curid=50241422>.

Это распространенное мнение, что DevOps – это пересечение Development, Operations и QA. Но следует различать разработчиков и инженеров, работающих по DevOps, хотя такой инженер помимо навыков системного администрирования и должен уметь писать скрипты, и даже может владеть тем или иным языком программирования помимо shell.

На практике по DevOps работает build engineer, в задачи которого входит проектирование архитектуры и инфраструктуры, он решает вопросы выбора архитектуры, инструментов, осуществляет взаимодействие в команде и между командами, в т. ч. с разработчиками. Системный инженер, который работает по культуре DevOps, умеет автоматизировать свою работу, знает облачные вычисления и облачные платформы, использует утилиты и умеет писать скрипты для автоматизации, знает основы безопасности, системы виртуализации, знает, что такое Continuous Delivery, Continuous Deployment, и понимает, как построить эти процессы.

Как уже было сказано, инженер может знать один из языков программирования, помимо shell или bash (как правило, это Python).

В DevOps приходят либо программисты, которые умеют немного администрировать, либо системные администраторы которые умеют хорошо писать скрипты и знают Management Tools (Puppet, Ansible, Chiev и т. д).

## Continuous Integration

Что такое **Continuous Integration?** После того, как разработка нового ПО завершается, включается механизм **ci/cd**.

**Continuous Delivery** – это то, как мы доставляем продукт. В разработке имеется несколько стадий: разработчики пишут код, который попадает в GitHub, далее же его требуется быстро доставить до тестового (стейджингового) окружения.

Ранее активно применялась методология Waterfall (водопад), и каждая итерация от начала разработки до выкладки в production занимала 3–4 месяца. Сейчас в разработке ПО все чаще получает распространение методология Agile, и итеративные циклы разработки стали короткими, как правило 2-недельными и даже недельными. Теперь требуется тестировать, конфигурировать и выпускать ПО довольно быстро и часто. То есть еженедельно повторяется цикл: установить разработанное ПО на dev-серверы, сконфигурировать, затем на стейджинговые серверы, сконфигурировать, проверить, потом на production-серверы, и также сконфигурировать, проверить. Все это невозможно без автоматизации и подразумевает использование соответствующих утилит (в частности, таких как Packer, Vargant и т. д.). DevOps-подход объясняет, как мы будем доставлять ПО к деву, к стейджу и на продакшн.

**Continuous Delivery** – помимо автоматизации и развертывания нужно решить и другие вопросы: как мы выстроим процесс доставки, автоматизацию, какие шаги мы должны предпринять, какие тесты выполнить, как забирать исходные коды и доставлять на продакшн.

**Continuous Deployment** – помимо перечисленных выше стадий цикла разработки есть еще одна, последняя: отправка на продакшн, которая также должна выполняться не в ручном режиме, а автоматически.

**Continuous Integration** – автоматизация сборки и проведения всех стадий тестирования.

Подход DevOps – это не только утилиты. DevOps также позволяет описывать и формировать культуру взаимодействия и работы в определенных командах.

Довольно подробно раскрывает культуру и философию DevOps книга Devops Handbook (на английском языке). В частности, раскрываются такие подходы, как командная ответственность и работа в команде, как устраиваются митинги. Например, ответственность в DevOps – командная, распределенная. Ошибка – это не ошибка одного человека, а ошибка команды.

Теперь разберем утилиты, которые позволяют решать ряд задач DevOps на примере утилит от Hashicorp.

# Введение в AWS

AWS – Amazon Web Services. Можно испытать бесплатно и использовать без оплаты в течение года. Но требуется определенная осторожность, так как легко «попасть» на неожиданные платные услуги и траты (см. в примерах ниже).

Регистрация: <https://aws.amazon.com/ru/free/>.

Для создания аккаунта нужна кредитная карта с 1 долларом на счету.

Уже на главной странице мы видим доступные сервисы:

* Облачные вычисления.
* Виртуальные машины.
* Облачные вычисления.
* Базы данных.
* Хранение данных (storage) и CDN (доставка данных).
* И многое другое...

После регистрации можно попробовать нужные решения: например, можно запустить инстанс, выбрать тип ОС и создать инстанс. Подключиться можно только по ключам, по паролю – невозможно. После старта инстанса всегда можно посмотреть состояние, логи, имеется внутренний IP-адрес и публичный, динамический DNS.

Далее мы разберем пример по созданию образа для AMI с помощью Packer и упакуем его в Vagrant Box.

# Packer

Разберем, что такое Packer и пресобранные образы.

**Что такое пресобранные образы (Pre-backed machine images)?**

Пресобранные образы – это образы виртуальных либо других инстансов (если говорим об облачных решениях), с помощью которых мы можем разворачивать виртуальные машины.

**Какие есть проблемы с пресобранными образами?**

Если вы собираете образ для виртуальной машины, его необходимо поддерживать, постоянно актуализировать и пересобирать либо вручную, либо с помощью соответствующих утилит. Если в компании ведется активная разработка, этот процесс потребуется автоматизировать

**Зачем использовать Packer?**

* Packer – утилита для создания образов машин и контейнеров для Docker для множественных платформ из единого конфигурационного источника.
* Образ машины – единый статичный модуль, который содержит преконфигурированную ОС и установленное ПО, которое используется для быстрого создания и запуска машин.
* Packer только создает образы. Ничего более. Он не пытается управлять ими никоим образом. После создания билда все зависит от вас: запустить или уничтожить его.

**Преимущества использования Packer:**

**Быстрый деплоймент инфраструктуры (Super Fast Infrastructure Deployment)**. Пакер используется для быстрого деплоймента инфраструктуры. Как вы только собрали образ, настроили и сконфигурировали, это можно делать быстро, причем не только для production, но и для dev и для разработчиков, для нужд системного администратора.

**Мультипровайдерность (Multi-provider portability).** К примеру, у вас может быть production в Amazon Web Services, стейдж в Open Stack, разработчики разрабатывают в виртуальных машинах, например в VirtualBox и VMWare. Это четыре разные платформы, под которые мы можем создавать виртуальные машины из одного шаблона, которые будут идентичны и одинаково настроены, иметь одинаковый набор ПО. Packer создает образы, идентичные для множества платформ. Вы можете запускать production в AWS, staging/QA в частном облаке, таком как OpenStack, и development в десктопных решениях виртуализации, таких как VMWare и Oracle VirtualBox.

**Стабильность.** С помощью Packer можно устанавливать ПО на машины, и во время сборки вы точно будете знать, что ПО последнее, стабильное и проверенное. Если что-то пошло не так в момент сборки, packer завершается с ошибкой, и вы об этом будете сразу оповещены.

**Хорошая тестируемость**

После того, как вы собрали машину, вы можете ее запустить и сразу выполнить тесты – либо ручное тестирование, либо Smoke Testing, либо функциональное тестирование.

**Use cases:**

**Сontinuous Delivery**

Добавьте Packer в середину вашего continuous delivery pipeline. Он может использоваться для создания новых образов машин для множественных платформ при каждом изменении Chef/Puppet. Как часть pipeline, вновь созданные образы могут быть запущены и протестированы, проверяя работу инфраструктуры. Если тесты пройдены, вы можете быть уверены, что образы будут работать, когда будут развернуты. Это приводит к новому уровню стабильности и тестируемости при изменении инфраструктуры.

Таким образом, вы можете генерировать новую виртуальную машину, которую протестировали по какому-либо триггеру. По триггеру запускается Packer, машина пересобирается, и можно сразу проверить.

**Разделение Dev/Prod**

Packer помогает поддерживать Development, Staging и Production похожими настолько, насколько возможно. Packer можно использовать для одновременного создания образов или многопроцессорных платформ. Таким образом, если вы используете AWS для Production и VMWare для разработки, вы можете одновременно создавать как машину AMI, так и машину VMWare с использованием Packer из того же шаблона. Добавьте к этому Continuous Delivery, и вы получите довольно гладкую систему для последовательной работы от разработки вплоть до production.

Можно использовать одинаковые Dev и Prod: вы можете генерировать в различных средах и собирать образы, и быть уверены, что они будут одинаковы.

**Создание Appliance/Demo**

Packer удобен в создании демо разрабатываемого продукта. Как только разрабатываемое ПО меняется, вы можете автоматически создавать Appliance с пресобранным ПО. Потенциальные пользователи могут начать работу с разрабатываемым ПО в выбранном самостоятельно окружении. Упаковка ПО со сложными требованиями никогда не была так проста.

**Терминология**

* **Шаблоны (Templates)**: JSON-файлы, содержащие шаги по сборке.
* **Билдеры (Builders):** платформы, под которые собираются конфигурации.
* **Провижинеры (Provisioners):** утилиты автоматизации, чтобы установить ПО.
* **Пост-процессоры (Post-processors):** действия, которые должны быть произведены после того, как образ будет собран.

**Поддерживаемые платформы**

* Amazon EC2 (AMI);
* CloudStack;
* DigitalOcean;
* Docker;
* Parallels;
* QEMU;
* VirtualBox (OVF);
* VMWare (VMX).

**Поддерживаемые Provisioners (провижинеры)**

* Ansible Remote;
* Chef Client;
* ChefSolo;
* Converge;
* File;
* PowerShell;
* Puppet Masterless;
* Puppet Server;
* Salt Masterless;
* Shell;
* Shell (Local);
* Windows Shell;
* Windows Restart.

Ansible – работает только под Linux.

Chef – используется клиент-серверная модель.

Chef Solo – безсерверная модель.

Puppet в клиент-серверном исполнении. Похожая на Chef (обе системы написаны на Ruby), Puppet Agent подключается к Puppet Server и забирает изменения.

Puppet Masterless: в машину, которую собирает Packer, загружается манифест и выполняется, после чего делается Puppet Apply.

Salt – агенты называются миньонами; она также может быть в бессерверном исполнении.

File – обычное копирование с локального места на удаленное.

Поддерживаются любые скрипты, от PowerShell до bash, как с мастером, так и без мастеров.

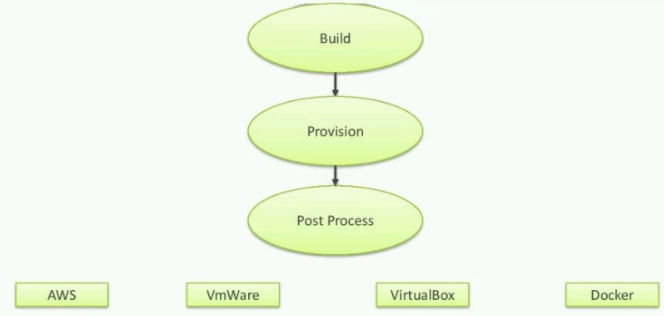
Windows Shell – обычный cmd.

Windows Restart – провижинер только для Windows. Если есть критическое обновление, необходимо перезапустить машину, чтобы установить обновление. Как только она перезапустится, будет продолжаться сценарий сборки.

**Поддерживаем пост-процессоры (Post Processors)**

* Amazon Import;
* Artifice;
* Atlas;
* Compress;
* Checksum;
* Docker Import;
* Docker Push;
* Docker Save;
* Docker Tag;
* Google Compute Export;
* Manifest;
* Shell (Local);
* Vagrant;
* Vagrant Cloud;
* vSphere.

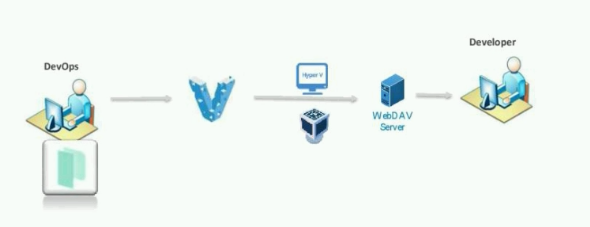
**Workflow**



Что случится после того, как мы поднимем и настроим машину:

1. Build – сборка. Включаем вирт-машину, деплоим образ, устанавливаем ОС.
2. Provision – настраиваем.
3. Post Process. Мы можем одновременно выгрузить:
   1. AWS;
   2. VMWare;
   3. VirtualBox;
   4. Docker.

**Building**



Путь от инженера до разработчика. Инженер с помощью Packer готовит образ, который собирает в Vagrant Box. Он, в свою очередь, выгружается на WebDAV-сервер, откуда разработчик может его развернуть, например, в свою виртуальную машину.

## Установка Packer

Packer устанавливается на многие платформы. Во FreeBSD:

|  |
| --- |
| $ pkg search packer  $ pkg install packer |

После установки проверим:

|  |
| --- |
| $ packer  usage: packer [--version] [--help] <command> [<args>]  Available commands are:  build build image(s) from template  fix fixes templates from old versions of packer  inspect see components of a template  push push template files to a Packer build service  validate check that a template is valid  version Prints the Packer version |

## Создание образа

Создается шаблон в формате json. Требуется образ ISO (в примере мы готовим во FreeBSD образ Ubuntu 16 для запуска в AWS), на основе которого будет подготовлен шаблон. На стадии build Packer сам подключится к виртуальной машине (VirtualBox или VMWare) либо удаленно, сымитирует клавиатурный ввод (ввод команд, нажатие кнопок, Enter, ожидание), на стадии Provision система будет перезагружена, после чего Packer подключится по ssh и сконфигурирует образ. Затем будет подготовлен образ, после чего созданная машина или инстанс должны быть удалены, а из образа уже можно будет за очень короткое время разворачивать готовые машины и инстансы. Например, можно подготовить Vagrant Box на стадии постпроцессора.

Пример создания образа для AWS. Необходимо зарегистрироваться в AWS. Чтобы собрать AMI-образ c помощью Packer, потребуется builder: amazon-ebs.

Пример шаблона aws.json:

|  |
| --- |
| {  "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-xenial-16.04-amd64-server-\*",  "root-device-type": "ebs"  },  "owners": ["099720109477"],  "most\_recent": true  },  "instance\_type": "t2.micro",  "ssh\_username": "ubuntu",  "ami\_name": "packer-example {{timestamp}}"  }]  } |

Запускаем билдер:

|  |
| --- |
| packer build \  -var 'aws\_access\_key=YOUR ACCESS KEY' \  -var 'aws\_secret\_key=YOUR SECRET KEY' \  example.json  ==> amazon-ebs: amazon-ebs output will be in this color.  ==> amazon-ebs: Creating temporary keypair for this instance...  ==> amazon-ebs: Creating temporary security group for this instance...  ==> amazon-ebs: Authorizing SSH access on the temporary security group...  ==> amazon-ebs: Launching a source AWS instance...  ==> amazon-ebs: Waiting for instance to become ready...  ==> amazon-ebs: Connecting to the instance via SSH...  ==> amazon-ebs: Stopping the source instance...  ==> amazon-ebs: Waiting for the instance to stop...  ==> amazon-ebs: Creating the AMI: packer-example 1371856345  ==> amazon-ebs: AMI: ami-19601070  ==> amazon-ebs: Waiting for AMI to become ready...  ==> amazon-ebs: Terminating the source AWS instance...  ==> amazon-ebs: Deleting temporary security group...  ==> amazon-ebs: Deleting temporary keypair...  ==> amazon-ebs: Build finished.  ==> Builds finished. The artifacts of successful builds are:  --> amazon-ebs: AMIs were created:  us-east-1: ami-19601070 |

Произойдет подключение к AWS, будет создан временный инстанс, будет установлена операционная система, затем будет готов AMI-образ. Для вас будет указан другой AMI ID, нежели чем в примере.

После выполнения приведенного выше примера в вашей учетной записи AWS появится связанный с ней AMI. AMI хранятся в S3 в Amazon, поэтому, если вы не хотите, чтобы с вас списывалось около 0,01 доллара США в месяц, удалите AMI, сначала отменив его на странице AWS AMI management. Затем удалите связанный snapshot на странице AWS snapshot management page.

## Provision

Следующий этап – Provision.

Добавьте в описанный пример aws.json следующие строки:

|  |
| --- |
| {  "variables": ["..."],  "builders": ["..."],  "provisioners": [{  "type": "shell",  "inline": [  "sleep 30",  "sudo apt-get update",  "sudo apt-get install -y redis-server"  ]  }]  } |

Задержка в 30 секунд важна: при подключении по ssh она требуется, чтобы проверить, что система установилась корректно.

Выполните валидацию

|  |
| --- |
| $ packer validate |

и запустите билдер так же, как в предыдущем примере. Действия аналогичны предыдущему примеру, но также после установки будет перезапущена ОС в инстансе, Packer подключится по ssh и установит необходимое ПО.

## Vagrant Boxes

Packer также позволяет получить результаты билдера (такие как образ AMI или простой VMware) и превращать его в Vagrant Box. Это делается с использованием постпроцессоров (post-processors).

Теперь модифицируем наш шаблон, чтобы использовать Vagrant post-processor для упаковки AWS AMI в Vagrant box с использованием vagrant-aws plugin (<https://github.com/mitchellh/vagrant-aws>).

**Включение постпроцессора**

Изменим aws.json, добавив:

|  |
| --- |
| {  "builders": ["..."],  "provisioners": ["..."],  "post-processors": ["vagrant"]  } |

Также выполните валидацию.

Запускаем:

|  |
| --- |
| $ packer build -only=amazon-ebs aws.json |

Для более подробного использования Packer доступна подробная документация на сайте: <https://www.packer.io/docs/index.html>.

# Домашнее задание

1. \* Собрать образ какой-нибудь системы через packer и разместить его на AWS (учтите, что бесплатный аккаунт выдается только на год и с ограничениями).

Технологии и ПО для дальнейшего изучения:

1. CI/CD.
2. Jenkins.
3. AWS, GCE, Azure.
4. Terraform.
5. Puppet, Ansible и т. д.

# Дополнительные материалы

1. The DevOps Handbook.
2. Лутц М. Изучаем Python.
3. <https://aws.amazon.com/ru/free/> AWS Free
4. <https://www.vagrantup.com/> Vagrant
5. <https://www.packer.io/> Packer
6. <https://www.packer.io/docs/index.html> Документация Packer
7. <https://github.com/mitchellh/vagrant-aws>
8. <https://gist.github.com/leonardofed/bbf6459ad154ad5215d354f3825435dc>
9. <https://github.com/vmbrasseur/devops-learning-resources>

# Используемая литература

Для подготовки данного методического пособия были использованы следующие ресурсы:

1. By Devops.png: Rajiv.Pantderivative work: Wylve – This file was derived from Devops.png:, CC BY 3.0, <https://commons.wikimedia.org/w/index.php?curid=20202905>
2. Автор: McCow – собственная работа, CC BY-SA 4.0, <https://commons.wikimedia.org/w/index.php?curid=50241422>
3. <https://www.packer.io/intro/getting-started/build-image.html>
4. <https://ivirt-it.ru/hashicorp-nomad-installation-guide-and-example/>
5. <https://www.consul.io/>
6. <https://habrahabr.ru/post/278085/>