Таким образом, спроектирована система управления афишей кинотеатров Республики Беларусь, которая позволит облегчить процесс управления афишей кинотеатра. Разработан интерфейс, который является понятным пользователю без обучения работы с системой. Присутствует очень подробная страница помощи. В случае выполнения неверных действий пользователю будет показано сообщение об ошибке, а в случае корректного выполнения – сообщение об успешном завершении операции.

ЛИТЕРАТУРА

1. Макфарланд, Д. Большая книга CSS3 / Д. Макфарланд. – 3-е изд. – СПб. : Питер, 2014 – 608 с.
2. Бенедетти, Р. Изучаем работу с JQuery / Р. Бенедетти, Р. Крэнли. – СПб. : Питер, 2012. – 512 с.
3. Спецификация Google – Material design [Электронный ресурс] / Google. Material design. – Режим доступа: http://www.google.com/design/spec/. – Дата доступа: 29.09.2015.
4. Плюсы и минусы CMS [Электронный ресурс] / Mywebblog.ru Блог Вебмастера. – Режим доступа: http://mywebblog.ru/sozdanie/plyusy-i-minusy-cms.html. – Дата доступа: 29.09.2015.

УДК 005

ВНЕДРЕНИЕ МЕТОДОЛОГИИ DEV OPS. ВИРТУАЛИЗАЦИЯ ОКРУЖЕНИЯ РАЗРАБОТЧИКА

Е.П. ЖИДЕЦКИЙ
(Представлено: Е.Р. СУХАРЕВ)

В рамках внедрения методологии DevOps в нашей компании был проведен ряд организационных изменений, пересмотрены зоны ответственности работников технического сектора, а также внедрены новые инструменты и методы в процесс создания и сопровождения программных продуктов. Одним из таких методов стала виртуализация окружения разработчика с использованием инструментов Vagrant и Ansible.

Термином «DevOps» обычно называют возникшее профессиональное движение, которое выступает за совместные рабочие отношения между разработчиками и ИТ-подразделением, в результате получая более быстрое выполнение планируемых работ, одновременно увеличивая надежность, стабильность, устойчивость и безопасность production-среды [1]. DevOps закономерно наследует наработки гибких методологий и определяет ряд принципов, общий смысл которых сводится к стиранию границ между разработчиками, тестировщиками, администраторами, специалистами по контролю качества и другими представителями операционного сектора, а также мотивации к реализации дерзких рискованных затей и получении максимально быстрой обратной связи от пользователей.

Для достижения поставленных целей требуется максимальная автоматизация рабочего процесса: тестирование, непрерывная интеграция, непрерывная доставка обновлений пользователям, проверка качества кода, развертывание production-окружений, мониторинг. Для реализации всех этих процессов существует целая масса готовых решений, обладающих своими плюсами и минусами. В соответствии с концепцией DevOps, зоны ответственности между различными отделами технического сектора компании размываются, что при выборе инструмента выводит критерий простоты освоения на единый уровень с функциональностью: DevOps-профессионал должен иметь возможность и уметь одинаково эффективно писать код, писать тесты к нему, разворачивать у себя окружение аналогичное тому, которое будет на production-стенде, изменять настройки этого окружения, постоянно находясь под присмотром автоматических средств контроля качества кода и покрытия тестами, и при этом превентивная годовая подготовка в специальном центре не предусматривается.

Проблема различных окружений у разработчика, тестировщика и на production-стенде известна всем. Еще лучше известны ее последствия: невоспроизводимые ошибки, фраза «Не знаю, у меня все работает», неожиданные падения production-стендов при отличной жизнеспособности сервиса под такими же нагрузками на тестовых площадках и др. Элементарно создание идентичных конфигураций на 5–20 машинах, что в наш век облачных сервисов даже не среднее количество, без автоматизации процесса превращается в работу, посильную только прожженному администратору старой закалки.

На самом деле решение всех этих проблем уже давно живет рядом с нами, и имя ему – виртуальная машина. Давно не секрет, что сети основной массы средних и крупных компаний целиком построены

на виртуальных машинах. У этого подхода есть масса преимуществ, основные из которых это цена и гибкость. С использованием виртуальных машин становится проще распределять ресурсы для различных узлов сети и контролировать уровни доступа, разворачивать новые узлы, клонируя виртуальные образы существующих машин.

В окружении разработчика виртуализация нашла свое применение сравнительно недавно. Да и сейчас, несмотря на массу известных положительных моментов, опытные разработчики не спешат перестраивать свой привычный подход к работе. Тем не менее, прогрессивные представители нашей профессии уже разработали несколько инструментов для упрощения работы с виртуальными машинами и ряд рекомендаций по их использованию, проверенных в «боевых» условиях.

Наиболее распространенным средством для управления виртуализацией окружения разработчика является Vagrant. Vagrant – это менеджер виртуальных машин, работает с VirtualBox, VmWare и несколькими другими менее популярными системами виртуализации [2]. Обладает простым консольным интерфейсом, и главное – работает по сценарию, описанному в конфигурационном файле в синтаксисе Ruby. Это значит, что один DevOps-специалист может написать сценарий на этом довольно распространенном языке программирования, поместить файл со сценарием под управление системы контроля версий, и вся команда разработчиков, работающая с ним над проектом, сможет ощутить на себе всю прелесть виртуализации, выполнив всего одну простую консольную команду: *vagrant up*. Если разработчик может поднять у себя виртуальную машину с окружением для разрабатываемого проекта, то он может поднять эту же машину и на тестовом стенде и на production-сервере, а это уже шаг от простого разработчика к DevOps-специалисту. Объективно, такой подход дает следующие преимущества:

– повторяемость окружения. У программиста, у тестировщиков и на production-стенде все одинаково;

– чистота host-машины. Работа на различных проектах требует различных окружений, если все ставить прямо на свою машину, то скоро там будет «зоопарк» из различных СУБД, версий интерпретаторов, сервисов и серверов. При использовании виртуальных машин, на host-машине остаются только любимая IDE и несколько легковесных UI-клиентов к сервисам, работающим на виртуальных машинах;

– простота подключения новых людей к проекту. В больших проектах всегда множество внешних зависимостей, новый человек сможет начать работать и видеть результаты своих действий, не изучая множество документации по установке и инструкций в wiki-проекта по конфигурированию экосистемы;

– простота переключения между проектами. Переключение между проектами сводится к следующей последовательности действий:

1) *vagrant suspend* в текущем проекте;
2) открытие нового проекта в IDE;
3) *vagrant up* в новом проекте;

– легко можно передать от тестировщика разработчику на исправление трудно воспроизводимую ошибку, просто заморозив виртуальную машину на моменте с ошибкой и передав образ.

Виртуальная машина при работе с Vagrant представляет собой некий начальный образ (box в терминах Vagrant), который разворачивается и конфигурируется. Большая коллекция базовых образов систем доступна в репозитории Vagrant [3]. Конфигурирование представлено в самом широком смысле этого слова: пробросы портов, настройка общих ресурсов, установка и настройка приложений (provisioning в терминах Vagrant). Установка и настройка приложений может выполняться как на низком уровне: исполнение shell-скриптов – так и с применением инструментов управления конфигурациями: Chef, Puppet, Ansible.

В нашей компании мы выбрали Ansible, как самый простой и достаточно функциональный инструмент управления конфигурациями [4]. Ansible обладает практически нулевым порогом вхождения для программиста. Все управление целевым сервером, или группой серверов, осуществляется через SSH протокол, не требуется открытие дополнительных портов и установка дополнительного программного обеспечения на обслуживаемых серверах.

Сценарии в Ansible описываются на языке YAML с использованием синтаксиса выражений шаблонизатора Jinja2. В сценарии (playbook в терминах Ansible) указывается конфигурация и множество серверов, к которым она применяется. Сами сервера представлены в так называемых inventory-файлах: файлы в ini-формате в которых указаны IP или DNS-имена серверов с группировкой по предназначению: сервера баз данных, сервера приложений и т.д. Как правило на каждое окружение: тестовые, production – создается свой inventory-файл. Конфигурация состоит из списка команд, за выполнение каждой команды отвечает конкретный модуль со своим списком параметров. Технически, модуль в Ansible – это одна команда в shell-оболочке. Основное различие заключается в том, что сценарий Ansible обладает свойством идемпотентности: если ранее конкретное изменение применялось, то оно не будет применено повторно –

а это значит, что можно спокойно вносить улучшения в сценарии и раз за разом запускать их на одном и том же множестве машин. Ansible «из коробки» поддерживает множество модулей, есть возможность писать собственные модули на языке Python. Чем выше версия Ansible, тем больше модулей он поддерживает, по этой причине рекомендуется всегда работать на последней стабильной версии.

Ansible позволяет писать сценарии в одном playbook-файле, в таком случае файл сценария представляет собой последовательность модулей, построенную по аналогии с обычным shell-скриптом. Сценарии, написанные таким образом, имеют большой размер, трудны в поддержке и абсолютно не пригодны для повторного использования. Есть еще более простой способ исполнения команды на требуемом множестве серверов – AdHoc [5]. Данный способ вообще не требует файла сценария, и может выполнить команду на серверах, содержащихся в inventory-файле, на лету из командной строки. Этот стиль является категорически плохим тоном и не рекомендуется к применению в «боевом» окружении, поскольку позволяет «сгоряча» накатить на множество серверов невоспроизводимое изменение. Любое изменение, не зафиксированное в сценарии под системой контроля версий, считается не воспроизводимым, поскольку не будет гарантированно применено к серверам, добавленным в систему в будущем, а значит, приведет к рассинхронизации системы и появлению потенциальных проблем поддержки.

В соответствии с лучшими практиками рекомендуется разделять наборы изменений на роли [6]. Под этим термином в Ansible, крайне не интуитивно, подразумевается некоторое установленное и настроенное средство: JRE, nginx-сервер, Tomcat, MongoDB и др. Создание роли предполагает описание процесса установки и настройки конкретного средства, определение множества возможных параметров, оказывающих влияние на процесс, определение зависимостей от других ролей. Формат описания роли строго регламентирован, используется определенная соглашением структура файловой системы.

Для распространения ролей существует специальный репозиторий Ansible Galaxy [7]. Стоит заметить, что модерация публикуемых в этом репозитории ролей оставляет желать лучшего. Существует система пользовательских отзывов, но и она не дает гарантии качества. Кроме этого, Ansible не является кросс-дистрибутивным средством, что создает дополнительные трудности в поиске готовой роли для применения в собственной среде. Методом проб и ошибок, мы пришли к собственноручной реализации ролей с минимально требуемой для повторного использования конфигурируемостью и распространению их через корпоративный репозиторий.

После освоения подхода по оформлению конфигураций в роли, файл сценария превращается в простой список ролей и серверов, к которым их требуется применить. При этом конкретные сервера для каждого окружения задаются в собственном inventory-файле. Конфигурацию такого формата легко расширять и поддерживать, она становится похожа на хороший объектно-ориентированный код.

В настоящее время практически во всех проектах нашей компании внедрена система виртуализации окружения разработчика с применением описанных выше инструментов и подходов. В результате этого, из оборота практически исчезли инструкции для новых разработчиков, а среднее время начала работы над новым проектом сократилось с 26 до 3 часов. Степень соответствия рабочего процесса методологии DevOps включает в себя множество параметров и почти не поддается объективной оценке, но положительная динамика очевидна, а значит, выбран правильный ориентир для развития.

ЛИТЕРАТУРА

1. Шарма, С. Освоение концепции DevOps в интересах непрерывной поддержки инноваций / С. Шарма // IBM developerWorks [Электронный ресурс]. – 2014. – Режим доступа: https://www.ibm.com/developerworks/ru/library/d-devops-continuous-innovation. – Дата доступа: 25.09.2015.
2. Providers [Electronic resource] / Vagrant docs. – 2015. – Mode of access: https://docs.vagrantup.com/v2/providers. – Date of access: 25.09.2015.
3. Vagrant boxes index [Electronic resource] / Vagrantbox.es. – 2015. – Mode of access: http://www.vagrantbox.es. – Date of access: 25.09.2015.
4. Поляков, В. Ansible / В. Поляков // Habrahabr [Электронный ресурс]. – 2013. – Режим доступа: http://habrahabr.ru/post/195048. – Дата доступа: 26.09.2015.
5. Introduction To Ad-Hoc Commands [Electronic resource] / Ansible documentation. – 2015. – Mode of access: http://docs.ansible.com/ansible/intro_adhoc.html. – Date of access: 26.09.2015.
6. Best Practices [Electronic resource] / Ansible documentation. – 2015 – Mode of access: http://docs.ansible.com/ansible/playbooks_best_practices.html. – Date of access: 26.09.2015.
7. Welcome to Ansible Galaxy [Electronic resource] / Ansible Galaxy. – 2015 – Mode of access: https://galaxy.ansible.com/intro. – Date of access: 26.09.2015.