Конспект лекций по курсу “Deployment and life cycle of modern software” («Развёртывание и жизненный цикл программного обеспечения»)

Лекция 1. Внедрение и развёртывание программного обеспечения. Процессы и роли при развёртывании программного обеспечения (2 часа)

Жизненный цикл (ЖЦ) разработки программного обеспечения – проектная деятельность по разработке и развертыванию программных систем.

В соответствии с базовым международным стандартом ISO/IEC 12207 все процессы ЖЦ ПО делятся на три группы:

1. Основные процессы:
    - приобретение;
    - поставка;
    - разработка;
    - эксплуатация;
    - сопровождение.
2. Вспомогательные процессы:
    - документирование;
    - управление конфигурацией;
    - обеспечение качества;
    - разрешение проблем;
    - аудит;
    - аттестация;
    - совместная оценка;
    - верификация.
3. Организационные процессы:
    - создание инфраструктуры;
    - управление;
    - обучение;
    - усовершенствование.

В таблице 2.1 приведены ориентировочные описания основных процессов ЖЦ. Вспомогательные процессы предназначены для поддержки выполнения основных процессов, обеспечения качества проекта, организации верификации, проверки и тестирования ПО. Организационные процессы определяют действия и задачи, выполняемые как заказчиком, так и разработчиком проекта для управления своими процессами.

Таблица 1.1. Содержание основных процессов ЖЦ ПО ИС (ISO/IEC 12207)

Процесс (исполнитель процесса)	Действия	Вход	Результат
Приобретение (заказчик)	• Инициирование • Подготовка заявочных предложений • Подготовка договора • Контроль деятельности поставщика • Приемка ИС	• Решение о начале работ по внедрению ИС • Результаты обследования деятельности заказчика • Результаты анализа рынка ИС/ тендера • План поставки/ разработки • Комплексный тест ИС	• Технико-экономическое обоснование внедрения ИС • Техническое задание на ИС • Договор на поставку/ разработку • Акты приемки этапов работы • Акт приемно-сдаточных испытаний
Поставка (разработчик ИС)	• Инициирование • Ответ на заявочные предложения • Подготовка договора • Планирование исполнения • Поставка ИС	• Техническое задание на ИС • Решение руководства об участии в разработке • Результаты тендера • Техническое задание на ИС • План управления проектом • Разработанная ИС и документация	• Решение об участии в разработке • Коммерческие предложения/ конкурсная заявка • Договор на поставку/ разработку • План управления проектом • Реализация/ корректировка • Акт приёмочных испытаний

Продолжение таблицы 1.1

Разработка (разработчик ИС)	• Подготовка • Анализ требований к ИС • Проектирование архитектуры ИС • Разработка требований к ПО • Проектирование архитектуры ПО • Детальное проектирование ПО • Кодирование и тестирование ПО • Интеграция ПО и квалификационное тестирование ПО • Интеграция ИС и квалификационное тестирование ИС	• Техническое задание на ИС • Техническое задание на ИС, модель ЖЦ • Подсистемы ИС • Спецификации требования к компонентам ПО • Архитектура ПО • Материалы детального проектирования ПО • План интеграции ПО, тесты Архитектура ИС, ПО, документация на ИС, тесты	• Используемая модель ЖЦ, стандарты разработки • План работ • Состав подсистем, компоненты оборудования • Спецификации требования к компонентам ПО • Состав компонентов ПО, интерфейсы с БД, план интеграции ПО • Проект БД, спецификации интерфейсов между компонентами ПО, требования к тестам • Тексты модулей ПО, акты автономного тестирования • Оценка соответствия комплекса ПО требованиям ТЗ • Оценка соответствия ПО, БД, технического комплекса и комплекта документации требованиям ТЗ

Для поддержки практического применения стандарта ISO/IEC 12207 разработан ряд технологических документов: Руководство для ISO/IEC 12207 (ISO/IEC TR 15271:1998 Information technology - Guide for ISO/IEC 12207) и Руководство по применению ISO/IEC 12207 к управлению проектами (ISO/IEC TR 16326:1999 Software engineering - Guide for the application of ISO/IEC 12207 to project management).

Позднее был разработан и в 2002 г. опубликован стандарт на процессы жизненного цикла систем (ISO/IEC 15288 System life cycle processes). К разработке стандарта были привлечены специалисты различных областей: системной инженерии, программирования, управления качеством, человеческими ресурсами, безопасностью и пр. Был учтен практический опыт создания систем в правительственных, коммерческих, военных и академических организациях. Стандарт применим для широкого класса систем, но его основное предназначение - поддержка создания компьютеризированных систем.

Согласно стандарту ISO/IEC серии 15288 в структуру ЖЦ следует включать следующие группы процессов:

1. Договорные процессы:
   - приобретение (внутренние решения или решения внешнего поставщика);
   - поставка (внутренние решения или решения внешнего поставщика).
2. Процессы предприятия:
   - управление окружающей средой предприятия;
   - инвестиционное управление;
   - управление ЖЦ ИС;
   - управление ресурсами;
   - управление качеством.
3. Проектные процессы:
   - планирование проекта;
   - оценка проекта;
   - контроль проекта;
   - управление рисками;
   - управление конфигурацией;
   - управление информационными потоками;
   - принятие решений.
4. Технические процессы:
   - определение требований;
   - анализ требований;
   - разработка архитектуры;

- внедрение;
- интеграция;
- верификация;
- переход;
- аттестация;
- эксплуатация;
- сопровождение;
- утилизация.

5. Специальные процессы:
   - определение и установка взаимосвязей исходя из задач и целей.

Одним из базовых понятий методологии проектирования ИС является понятие жизненного цикла ее программного обеспечения (ЖЦ ПО). ЖЦ ПО - это непрерывный процесс, который начинается с момента принятия решения о необходимости его создания и заканчивается в момент его полного изъятия из эксплуатации.

Основным нормативным документом, регламентирующим ЖЦ ПО, является международный стандарт ISO/IEC 12207 (ISO - International Organization of Standardization - Международная организация по стандартизации, IEC - International Electrotechnical Commission - Международная комиссия по электротехнике). Он определяет структуру ЖЦ, содержащую процессы, действия и задачи, которые должны быть выполнены во время создания ПО.

Структура ЖЦ ПО по стандарту ISO/IEC 12207 базируется на трех группах процессов:

- основные процессы ЖЦ ПО (приобретение, поставка, разработка, эксплуатация, сопровождение);
- вспомогательные процессы, обеспечивающие выполнение основных процессов (документирование, управление конфигурацией, обеспечение качества, верификация, аттестация, оценка, аудит, решение проблем);

- организационные процессы (управление проектами, создание инфраструктуры проекта, определение, оценка и улучшение самого ЖЦ, обучение).

Разработка включает в себя все работы по созданию ПО и его компонент в соответствии с заданными требованиями, включая оформление проектной и эксплуатационной документации, подготовку материалов, необходимых для проверки работоспособности и соответствующего качества программных продуктов, материалов, необходимых для организации обучения персонала и т.д. Разработка ПО включает в себя, как правило, анализ, проектирование и реализацию (программирование).

Эксплуатация включает в себя работы по внедрению компонентов ПО в эксплуатацию, в том числе конфигурирование базы данных и рабочих мест пользователей, обеспечение эксплуатационной документацией, проведение обучения персонала и т.д., и непосредственно эксплуатацию, в том числе локализацию проблем и устранение причин их возникновения, модификацию ПО в рамках установленного регламента, подготовку предложений по совершенствованию, развитию и модернизации системы.

Управление проектом связано с вопросами планирования и организации работ, создания коллективов разработчиков и контроля за сроками и качеством выполняемых работ. Техническое и организационное обеспечение проекта включает выбор методов и инструментальных средств для реализации проекта, определение методов описания промежуточных состояний разработки, разработку методов и средств испытаний ПО, обучение персонала и т.п. Обеспечение качества проекта связано с проблемами верификации, проверки и тестирования ПО. Верификация - это процесс определения того, отвечает ли текущее состояние разработки, достигнутое на данном этапе, требованиям этого этапа. Проверка позволяет оценить соответствие параметров разработки с исходными требованиями. Проверка частично совпадает с тестированием, которое связано с идентификацией различий между действительными и ожидаемыми результатами и оценкой соответствия характеристик ПО исходным требованиям. В процессе реализации проекта важное место занимают

вопросы идентификации, описания и контроля конфигурации отдельных компонентов и всей системы в целом.

Управление конфигурацией является одним из вспомогательных процессов, поддерживающих основные процессы жизненного цикла ПО, прежде всего процессы разработки и сопровождения ПО. При создании проектов сложных ИС, состоящих из многих компонентов, каждый из которых может иметь разновидности или версии, возникает проблема учета их связей и функций, создания унифицированной структуры и обеспечения развития всей системы. Управление конфигурацией позволяет организовать, систематически учитывать и контролировать внесение изменений в ПО на всех стадиях ЖЦ. Общие принципы и рекомендации конфигурационного учета, планирования и управления конфигурациями ПО отражены в проекте стандарта ISO 12207-2.

Каждый процесс характеризуется определенными задачами и методами их решения, исходными данными, полученными на предыдущем этапе, и результатами. Результатами анализа, в частности, являются функциональные модели, информационные модели и соответствующие им диаграммы. ЖЦ ПО носит итерационный характер: результаты очередного этапа часто вызывают изменения в проектных решениях, выработанных на более ранних этапах.

Внедрение программного обеспечения — процесс настройки программного обеспечения под определенные условия использования, а также обучения пользователей работе с программным продуктом.

При внедрении программного обеспечения требуется действие в трех следующих плоскостях работ.

Первая из них — это выделение критических, с точки зрения общего результата, процедур в деятельности организации. Когда набор таких процедур определен, необходимо в первую очередь использовать ИТ-решение для автоматизации операций внутри именно этих процедур. Таким образом, разработанное ИТ-решение автоматически становится жизненно

важным и востребованным для организации, а также будет обеспечена публичность процесса внедрения.

Вторая плоскость работ — это по своей сути расширение нормативной базы организации путем включения в нее регламентов, описывающих порядок выполнения процедур автоматизируемых процессов. В противном случае есть опасность возникновения рассогласования между автоматизированными процедурами и остальными процессами организации.

Третья — это выполнение работ по общей стандартизации существующей деятельности организации, когда выделяются лучшие практики выполнения процедур и включаются в ИТ-решение по принципу наибольшей полезности для большинства участников. Процент таких процедур относительно общего объема автоматизации может быть невелик, но это придает процессу построения решения вес в организации за счет увеличения его «полезности»

Развёртывание ПО – деятельность, заключающаяся в установке, настройке и подготовке программного обеспечения к использованию.

Основной процесс развёртывания ПО состоит из нескольких взаимосвязанных операций с возможными переходами между ними. Эти операции могут затрагивать как поставщика ПО, так и заказчика. Поскольку каждый программный продукт уникален, точную и универсальную процедуру развёртывания очень сложно описать. Следовательно, «развёртывание» должно интерпретироваться как общий процесс, который должен быть выстроен в соответствии со специфичными требованиями или характеристиками.

Операции развёртывания:

1. Выпуск версии (Release).

   Стадия выпуска версии выполняется при завершении очередного цикла разработки ПО. Она включает все действия по подготовке системы к сборке и передаче заказчику. Таким образом, во время этой операции должны быть собраны требования для проведения последующих операций на стороне заказчика.

2. Установка и активация (Install and activate).

   Активация – запуск исполняемого компонента ПО. Для простых систем это запуск бинарного файла или команды на исполнение. Для сложных систем, во время этой операции должны быть запущены все необходимые подсистемы. Рабочая копия ПО может устанавливаться как в целевом окружении, так и в тестовом окружении, окружении для разработки или же аварийном окружении для восстановления.
3. Деактивация (Deactivate).

   Деактивация – операция обратная активации, заключается в завершении работы всех подсистем приложения. Деактивация обычно требуется для выполнения других операций развёртывания, например обновления. Деактивация также может предшествовать полной остановке и удалению устаревших или неиспользуемых систем.
4. Адаптация (Adapt).

   Адаптация – процесс модификации установленного ПО. Она отличается от обновления тем, что инициируется внутренними событиями окружения, например сменой сервера или дополнительной настройкой компонентов, а не поставщиком ПО.
5. Обновление (Update).

   Операция обновления заключается в замене всех компонентов предыдущего экземпляра ПО на новый выпуск. Операция обновления может запускаться встроенным механизмом обновления ПО под контролем пользователя или в полностью автоматическом режиме.
6. Удаление (Uninstall).

   Удаление – операция обратная установке, заключается в удалении всех компонентов ПО из целевого окружения. Может требовать изменения конфигурации другого ПО в окружении.

Сложность и изменчивость программных продуктов способствовали появлению специализированных ролей для координации и проведения операций развёртывания. Для настольных приложений, конечные

пользователи часто выступают в роли инженеров по внедрению ПО, когда они устанавливают программный пакет на своей машине. Процесс развёртывания программного обеспечения для предприятий обычно вовлекает намного больше участников с различными ролями, состав участников и необходимых ролей меняется по мере того, как приложение переходит из стадии тестирования в стадию эксплуатации.

Участники развёртывания ПО в тестовых окружениях:

- Разработчики ПО (Software developers);
- Инженеры по выпуску версий ПО (Release engineers);
- Менеджеры по выпуску версий (Release managers);
- Координаторы внедрения (DevOps engineers).

Участники развёртывания ПО в целевых окружениях:

- Системные администраторы (System administrators);
- Администраторы баз данных (Database administrators);
- Координаторы внедрения (DevOps engineers);
- Менеджеры внедрения со стороны заказчика (Operations project manager).

Лекция 2. Методология непрерывной поставки программного обеспечения «Continuous Delivery» (4 часа)

2.1 Непрерывная интеграция

Непрерывная интеграция (CI, англ. Continuous Integration) — это практика разработки программного обеспечения, которая заключается в выполнении частых автоматизированных сборок проекта для скорейшего выявления и решения интеграционных проблем. В обычном проекте, где над разными частями системы разработчики трудятся независимо, стадия интеграции является заключительной. Она может непредсказуемо задержать окончание работ. Переход к непрерывной интеграции позволяет снизить трудоёмкость интеграции и сделать её более предсказуемой за счет наиболее раннего обнаружения и устранения ошибок и противоречий.

Непрерывная интеграция является одним из основных приёмов экстремального программирования.

Требования к проекту:

- Исходный код и всё, что необходимо для сборки и тестирования проекта, хранится в репозиториях системы управления версиями;
- Операции копирования из репозиториев, сборки и тестирования всего проекта автоматизированы и легко вызываются из внешней программы.

На выделенном сервере организуется служба, в задачи которой входят:

- получение исходного кода из репозитория;
- сборка проекта;
- выполнение тестов;
- развёртывание готового проекта;
- отправка отчетов.

Локальная сборка может осуществляться:

- по внешнему запросу;
- по расписанию;

- по факту обновления репозитория и по другим критериям.

В случае сборки по расписанию (англ. daily build — рус. ежедневная сборка), они, как правило, проводятся каждой ночью в автоматическом режиме — ночные сборки (чтобы к началу рабочего дня были готовы результаты тестирования). Для различия дополнительно вводится система нумерации сборок — обычно, каждая сборка нумеруется натуральным числом, которое увеличивается с каждой новой сборкой. Исходные тексты и другие исходные данные при взятии их из репозитория (хранилища) системы контроля версий помечаются номером сборки. Благодаря этому, точно такая же сборка может быть точно воспроизведена в будущем — достаточно взять исходные данные по нужной метке и запустить процесс снова. Это даёт возможность повторно выпускать даже очень старые версии программы с небольшими исправлениями.

Преимущества CI:

- проблемы интеграции выявляются и исправляются быстро, что оказывается дешевле;
- немедленный прогон модульных тестов для свежих изменений;
- постоянное наличие текущей стабильной версии вместе с продуктами сборок — для тестирования, демонстрации, и т. п.;
- немедленный эффект от неполного или неработающего кода приучает разработчиков к работе в итеративном режиме с более коротким циклом.

Недостатки CI:

- затраты на поддержку работы непрерывной интеграции;
- потенциальная необходимость в выделенном сервере под нужды непрерывной интеграции;
- немедленный эффект от неполного или неработающего кода отучает разработчиков от выполнения периодических резервных включений кода в репозиторий. В случае использования системы управления версиями исходного кода с поддержкой ветвления, эта проблема может решаться созданием отдельной «ветки» (англ. branch) проекта

для внесения крупных изменений (код, разработка которого до работоспособного варианта займет несколько дней, но желательно более частое резервное копирование в репозиторий). По окончании разработки и индивидуального тестирования такой ветки, она может быть объединена (англ. merge) с основным кодом или «стволом» (англ. trunk) проекта.

2.2 Непрерывная поставка

Непрерывная поставка (англ. Continuous Delivery) – язык программирования шаблонов, использующийся в программной разработке для улучшения процесса поставки ПО.

Такие техники, как Автоматизированное тестирование, Непрерывная поставка и Непрерывная интеграция позволяют обеспечить высокие стандарты поставки ПО и его легкое развертывание на тестовой среде. В результате поставщик сервиса имеет неоднократную возможность быстро и надежно внедрять доработки, а также исправлять ошибки с минимальным риском для клиента.

Непрерывная поставка является одной из техник в экстремальном программировании (Extreme Programming, XP), но на уровне компании развивается до самостоятельной дисциплины.

Технология непрерывной поставки предполагает максимальную автоматизацию такого жизненного цикла разработки ПО:

- Разработка;
- Тестирование;
- Создание установочного пакета для клиента;
- Развёртывание.

Процесс непрерывной поставки позволяет удешевить производство качественного ПО, добавляет предсказуемость и прогнозируемость в этот процесс.

Конвейеры развертывания могут быть организованы по разному, в зависимости от технологических особенностей поставки релизов, однако

фундаментальные принципы непрерывного развертывания одинаковы в любых ситуациях. Пример конвейера развертывания приведен на рис. 2.1.

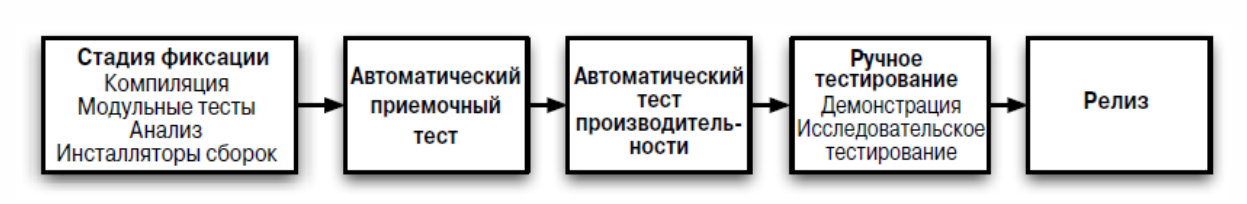

Рис. 2.1 – Пример конвейера развёртывания

Каждое изменение конфигурации, рабочей среды, исходного кода или данных приложения инициирует создание нового экземпляра конвейера развертывания. Начальная стадия конвейера – создание бинарных кодов и инсталляторов. На остальных стадиях выполняется ряд тестов бинарных кодов с целью проверки их пригодности к выпуску. Каждый тест увеличивает вероятность того, что текущая комбинация бинарных кодов, конфигураций, рабочей среды и данных будет работать безупречно. Релиз-кандидат, прошедший все тесты, считается готовым к выпуску.

Конвейер развертывания основан на процессе **непрерывной интеграции**. В сущности, непрерывное развертывание является результатом логического развития принципа непрерывной интеграции.

Концепция конвейера развертывания преследует три взаимосвязанные цели. Во-первых, конвейер обеспечивает видимость каждой части технологии (сборка, тестирование и поставка релиза) для каждого участника процесса. Во-вторых, конвейер улучшает обратную связь, делая каждую проблему идентифицируемой и решаемой на возможно более ранней стадии процесса. И наконец, конвейер позволяет развернуть любую версию приложения в любой среде с помощью полностью автоматизированного процесса.

2.3 Антишаблон: развертывание ПО вручную

Большинство современных приложений сложно развертывать вследствие того, что в них много взаимозависимых изменяющихся частей. Многие организации развертывают ПО вручную. Это означает, что процесс развертывания разбивается на ряд атомарных стадий, каждая из которых

выполняется ответственным исполнителем или командой. На каждой стадии нужно принимать сложные решения, в результате чего повышается вероятность ошибки. И даже если не произойдет ошибок, различия в принципах управления разными стадиями приводят к нестабильным результатам. Эти различия не могут повлиять на процесс развертывания положительным образом.

Основные признаки данного антишаблона следующие:

- Продуцирование большого объема документации, в которой подробно описывается, что нужно сделать и что может получиться неправильно.
- Решение о работоспособности релиза принимается исключительно на основе результатов ручного тестирования.
- Частые обращения к команде разработчиков для получения объяснений, почему в день поставки релиза приложение работает не так, как предполагается.
- Частые изменения процедуры поставки релиза.
- Конфигурации разных сред развертывания (тестовой, отладочной и рабочей) существенно отличаются друг от друга. Например, серверы приложений работают с разными пулами соединений и структурами файловых систем.
- Пробные запуски релизов занимают много времени.
- Результаты установки релизов непредсказуемые. Часто нужно возвращаться назад и разбираться с возникшими проблемами.
- Задержка персонала до поздней ночи для решения проблем, возникающих непосредственно в ходе поставки релиза.

Альтернатива развёртыванию вручную

Со временем технологии развертывания все больше автоматизируются. В идеале при ручном развертывании ПО в средах разработки и тестирования и в рабочей среде должны решаться две задачи: первая – выбор версии и предопределенной среды, вторая – щелчок на кнопке ‘‘Установить’’. Выпуск пакета ПО должен быть одним автоматизированным процессом, создающим инсталляторы.

Почему автоматизация развёртывания обязательна:

- Когда развертывание автоматизировано не полностью, часто возникают ошибки. Единственный вопрос – насколько они существенны. Даже если тесты великолепные, тяжело отследить источники ошибок.
- Когда развертывание не автоматизировано, оно не может выполняться как надежный, часто повторяющийся процесс. В результате тратится много времени на обнаружение и устранение ошибок развертывания, а не приложения.
- Процесс ручного развертывания должен быть документирован. Поддержка документации – сложная задача. На ее решение тратится много времени, и требуется вовлечение многих людей. Тем не менее, документация чаще всего неполная и всегда устаревшая. В то же время в автоматизированном процессе в качестве документации служат сценарии развертывания. Естественно, они всегда полные и самые новые (иначе процедура развертывания не работала бы).
- Автоматизация развертывания способствует сотрудничеству участников процесса, потому что все, что необходимо для развертывания, явно отображено в сценарии. В то же время при ручном развертывании документация предполагает определенный уровень знаний читателя и часто представляет собой не более чем “записную книжку”, в которую разработчик записывает отдельные важные сведения. Другим людям тяжело читать такой ‘‘конспект’’.
- Следствие вышесказанного: ручное развертывание зависит от квалификации исполнителя, и когда он увольняется или уходит в отпуск, вы оказываетесь в затруднительном положении.
- Развертывание вручную – весьма скучный процесс, тем не менее, требующий высокой квалификации. Когда квалифицированные специалисты делают скучную, рутинную (хотя и сложную) работу, они неизбежно совершают ошибки самых разных типов. Автоматизация развертывания освобождает высокооплачиваемых специалистов для более интеллектуальной работы.

- Единственный способ проверки процесса ручного развертывания состоит в его повторении. Часто это дорогостоящий процесс, требующий много времени. Автоматизированное развертывание дешевле, и его легче проверить.
- Иногда приходится слышать, что ручной процесс легче поддается аудиту, чем автоматизированный. Подобные мнения всегда приводят нас в замешательство. В ручном процессе нет гарантий того, что исполнитель строго придерживался документации. Полный контроль можно получить только над автоматическим процессом. Что может быть более надежным и полным документом, чем сценарий развертывания?

Автоматизированное развертывание следует использовать всегда; оно должно быть единственной технологией выпуска ПО. Сценарий развертывания можно запустить в любой момент, когда это необходимо. Один из принципов, пропагандируемых в данной книге, состоит в том, что один и тот же сценарий развертывания должен использоваться в любой среде. Благодаря этому в критический день поставки релиза используется сценарий, протестированный сотни раз. Если возникают проблемы с поставкой релиза, можно быть уверенным, что они связаны с конфигурацией среды, а не сценарием.

Иногда ручное развертывание проходит довольно гладко, но чаще всего это не так. Общепризнанно, что процесс развертывания может содержать ошибки и существенно задерживать выпуск. Поэтому стоит избавиться от источников ошибок и неопределенностей, присущих этому процессу.

2.4 Антишаблон: развертывание в среде производственного типа только после завершения разработки

В этом сценарии приложение впервые развертывается в среде производственного типа (например, в отладочной среде) только после того, когда команда разработчиков доложит, что они уже почти все сделали и можно попробовать установить приложение. Основные признаки данного антишаблона следующие.

- Если в процессе разработки ПО участвовали тестировщики, они уже протестировали систему в среде разработки.
- Администраторы впервые знакомятся с приложением при развертывании в отладочной среде. В некоторых организациях установкой приложений в отладочной и рабочей средах занимаются разные команды техподдержки. В этом случае администратор, который будет устанавливать приложение, впервые увидит его в рабочей среде в день поставки релиза.
- Производственная среда весьма дорогая, поэтому доступ к ней строго ограничивается. Кроме того, часто она еще не готова в нужный момент, а иногда даже случается так, что никто не позаботился о том, чтобы создать ее.
- Команда разработчиков собирает в единый комплект все инсталляторы, конфигурационные файлы, процедуры переноса баз данных и документацию, чтобы передать комплект команде, которая будет заниматься развертыванием. Естественно, весь комплект не тестировался ни в отладочной, ни в рабочей средах.
- Команда разработчиков и команда развертывания почти не взаимодействуют друг с другом.

Когда приходит время развернуть приложение в отладочной среде, собирается команда, которая сделает это. В некоторых случаях члены команды имеют все необходимые знания, но часто в крупных организациях команда развертывания делится на несколько специализированных групп, занимающихся администрированием баз данных, промежуточным программным обеспечением, веб-приложениями и т.п. Многие стадии еще не тестировались в отладочной среде, поэтому часто возникают ошибки. В документации пропущены важные стадии. В сценариях и документации заложены ошибочные предположения о конфигурации, версиях и параметрах целевой среды, в результате чего развертывание терпит крах. Команда развертывания вынуждена угадывать намерения команды разработчиков.

Часто плохое взаимодействие разных команд приводит к тому, что при развертывании завязывается оживленный обмен телефонными звонками и

электронными письмами для устранения неполадок. Дисциплинированная команда попытается включить коммуникацию с другими командами в план развертывания, однако их усилия редко оказываются эффективными. Когда напряженность ситуации доходит до предела, коммуникация между командами разработки и развертывания прекращается, потому что у команды развертывания не остается для нее времени.

В процессе развертывания нередко обнаруживается, что неправильные предположения о рабочей среде заложены в проект системы. Например, недавно мы участвовали в развертывании приложения, в котором данные кэшировались в файловой системе. На компьютере разработчика приложение работало безукоризненно, однако в кластерной среде возникли серьезные проблемы. Решение подобных проблем может занять много времени, причем развертывание нельзя считать успешным, пока все такие проблемы не будут решены.

Когда приложение переходит в отладочную среду, часто обнаруживаются новые ошибки. К сожалению, исправлять их нет времени, потому что крайний срок поставки релиза быстро приближается, а его откладывание недопустимо из коммерческих соображений. В результате наиболее критичные ошибки наспех ‘‘заштопываются’’ и включаются в список известных дефектов, подлежащих устранению в следующей версии.

Альтернатива

Оптимальное решение состоит в интеграции процедур тестирования, установки и поставки релиза в один процесс. В результате, когда придет время развернуть релиз в рабочей среде, все будет готово и почти все риски будут устранены, потому что вы уже прорепетировали развертывание в ряде тестовых сред, приближающихся к рабочей. Убедитесь в том, что все люди, вовлеченные в процесс поставки, работают совместно с самого начала проекта.

Интенсивное применение непрерывной интеграции как средства тестирования ПО и процедур развертывания – ключевой элемент подхода

Непрерывной поставки. Диаграмма последовательности процесса непрерывной поставки приведена на рис. 2.2.

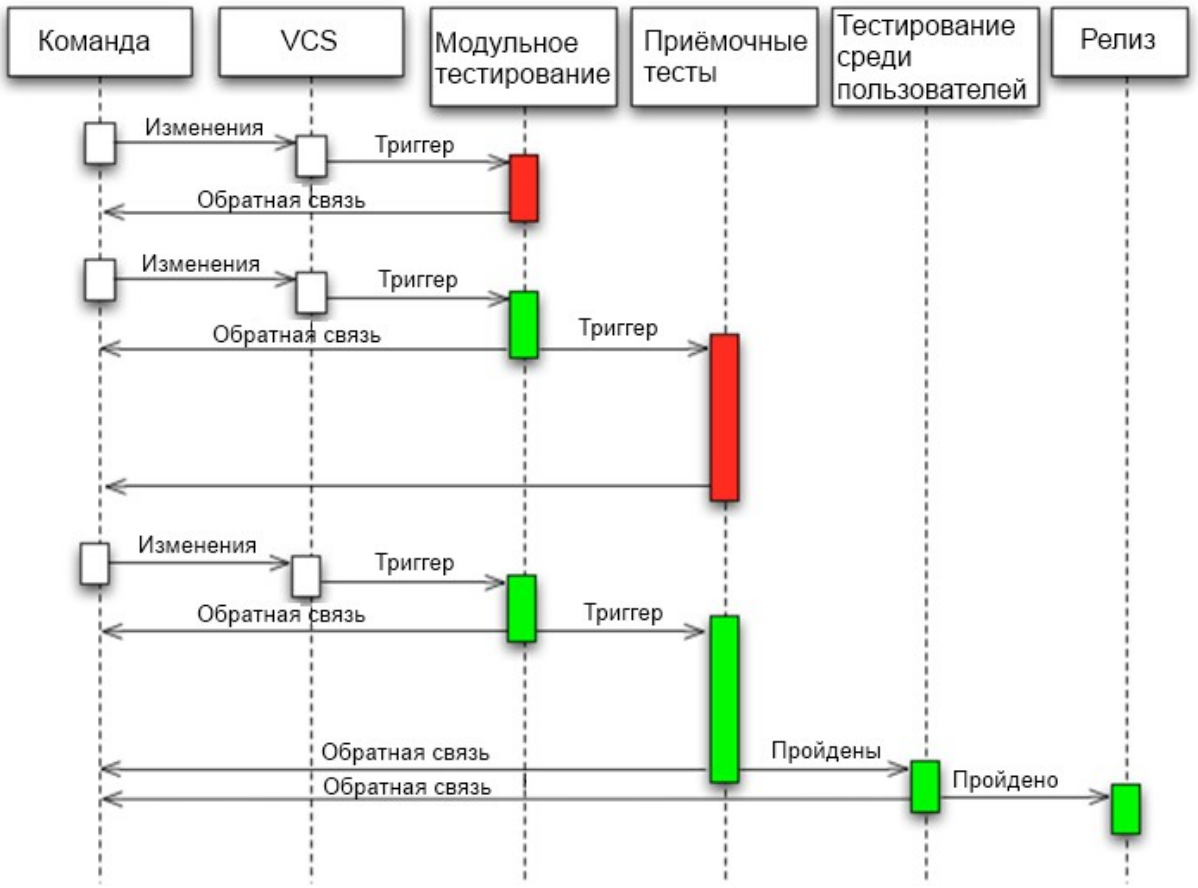

Рис. 2.2 – Диаграмма последовательности процесса непрерывной поставки

2.5 Преимущества непрерывной поставки

Главное преимущество подхода, состоит в том, что он позволяет превратить выпуск новых версий в повторяемый, надежный и предсказуемый процесс, в результате чего существенно уменьшается продолжительность цикла поставки.

Основные преимущества непрерывной поставки:

- Уменьшение количества ошибок;
- Гибкость развёртывания;
- Снижение стресса от ручных операций.

Лекция 3. Инструментарий развёртывания программного обеспечения (4 часа)

3.1 Системы контроля версий

Система контроля версий (Version Control System, VCS) — это система, регистрирующая изменения в одном или нескольких файлах с тем, чтобы в дальнейшем была возможность вернуться к определённым старым версиям этих файлов. Для примеров в этой книге мы будем использовать исходные коды программ, но на самом деле под версионный контроль можно поместить файлы практически любого типа.

Если вы графический или веб-дизайнер и хотели бы хранить каждую версию изображения или макета, то пользоваться системой контроля версий будет очень полезно. VCS даёт возможность возвращать отдельные файлы к прежнему виду, возвращать к прежнему состоянию весь проект, просматривать происходящие со временем изменения, определять, кто последним вносил изменения во внезапно переставший работать модуль, кто и когда внёс в код какую-то ошибку, и многое другое. Вообще, если, пользуясь VCS, вы всё испортите или потеряете файлы, всё можно будет легко восстановить. Вдобавок, накладные расходы за всё, что вы получаете, будут очень маленькими.

3.l.1 Локальные системы контроля версий

Многие предпочитают контролировать версии, просто копируя файлы в другой каталог (как правило, добавляя текущую дату к названию каталога). Такой подход очень распространён, потому что прост, но он и чаще даёт сбои. Очень легко забыть, что ты не в том каталоге, и случайно изменить не тот файл, либо скопировать файлы не туда, куда хотел, и затереть нужные файлы.

Чтобы решить эту проблему, программисты уже давно разработали локальные VCS с простой базой данных, в которой хранятся все изменения нужных файлов. Схема локальной VCS приведена на рис. 3.1.

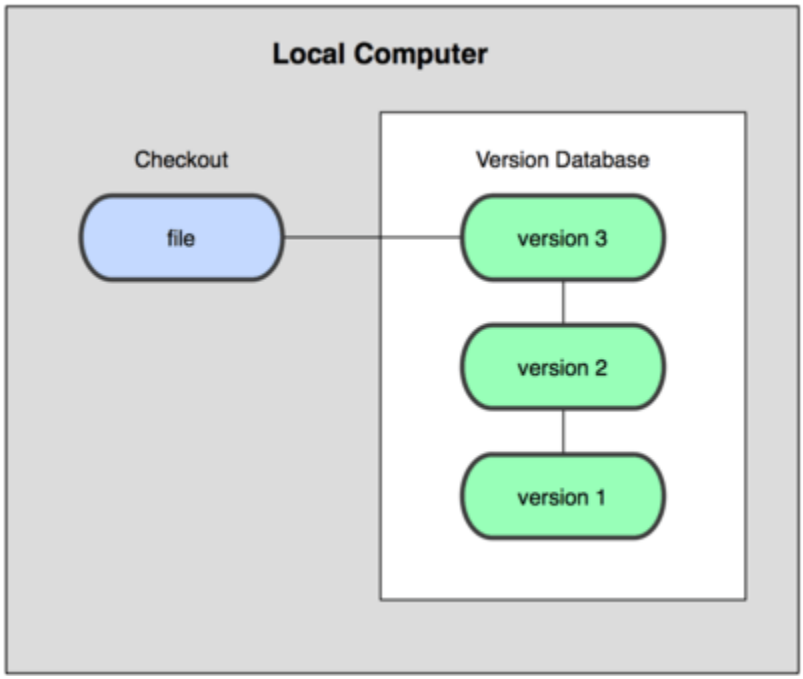

Рис. 3.1. Схема локальной VCS

Одной из наиболее популярных VCS такого типа является rcs, которая до сих пор устанавливается на многие компьютеры. Даже в современной операционной системе Mac OS X утилита rcs устанавливается вместе с Developer Tools. Эта утилита основана на работе с наборами патчей между парами версий (патч — файл, описывающий различие между файлами), которые хранятся в специальном формате на диске. Это позволяет пересоздать любой файл на любой момент времени, последовательно накладывая патчи.

3.1.2 Централизованные системы контроля версий

Следующей основной проблемой оказалась необходимость сотрудничать с разработчиками за другими компьютерами. Чтобы решить её, были созданы централизованные системы контроля версий (CVCS). В таких системах, например CVS, Subversion и Perforce, есть центральный сервер, на котором хранятся все файлы под версионным контролем, и ряд клиентов, которые получают копии файлов из него. Много лет это было стандартом для систем контроля версий. Схема централизованного контроля версий приведена на рис. 3.2.

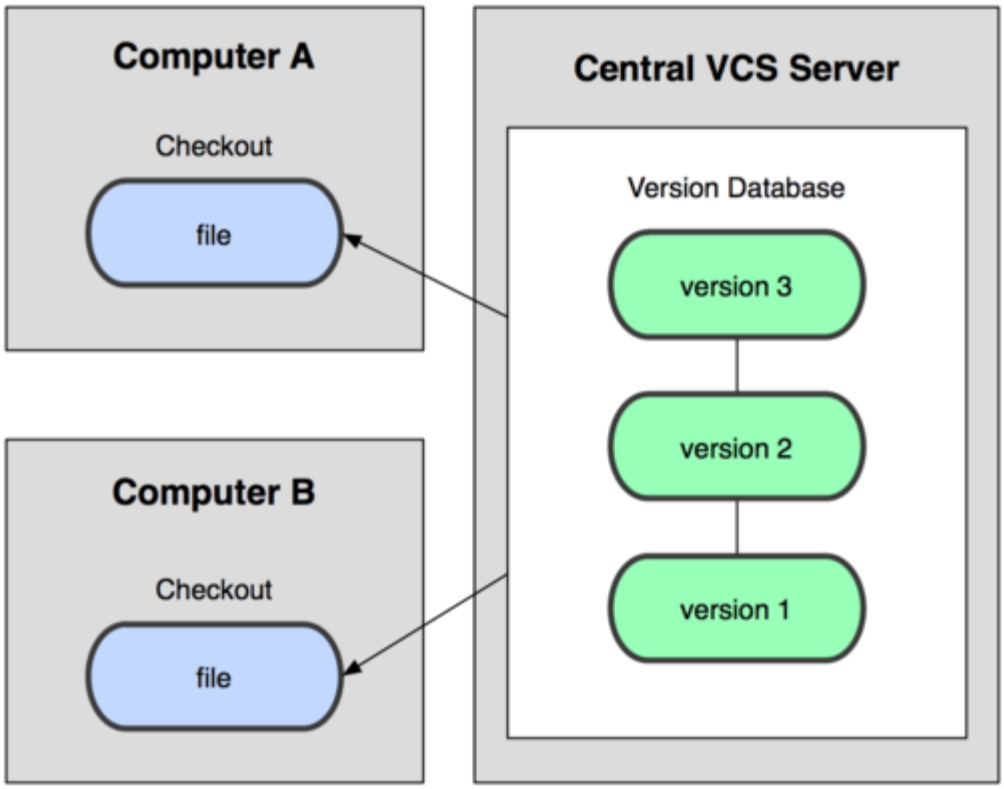

Рис. 3.2. Схема централизованного контроля версий

Такой подход имеет множество преимуществ, особенно над локальными VCS. К примеру, все знают, кто и чем занимается в проекте. У администраторов есть чёткий контроль над тем, кто и что может делать, и, конечно, администрировать CVCS намного легче, чем локальные базы на каждом клиенте.

Однако при таком подходе есть и несколько серьёзных недостатков. Наиболее очевидный — централизованный сервер является уязвимым местом всей системы. Если сервер выключается на час, то в течение часа разработчики не могут взаимодействовать, и никто не может сохранить новой версии своей работы. Если же повреждается диск с центральной базой данных и нет резервной копии, вы теряете абсолютно всё — всю историю проекта, разве что за исключением нескольких рабочих версий, сохранившихся на рабочих машинах пользователей. Локальные системы контроля версий подвержены той же проблеме: если вся история проекта хранится в одном месте, вы рискуете потерять всё.

3.1.3 Распределённые системы контроля версий

И в этой ситуации в игру вступают распределённые системы контроля версий (DVCS). В таких системах как Git, Mercurial, Bazaar или Darcs клиенты не просто выгружают последние версии файлов, а полностью копируют весь репозиторий. Поэтому в случае, когда "умирает" сервер, через который шла работа, любой клиентский репозиторий может быть скопирован обратно на сервер, чтобы восстановить базу данных. Каждый раз, когда клиент забирает свежую версию файлов, он создаёт себе полную копию всех данных. Схема распределённой VCS приведена на рис. 3.3.

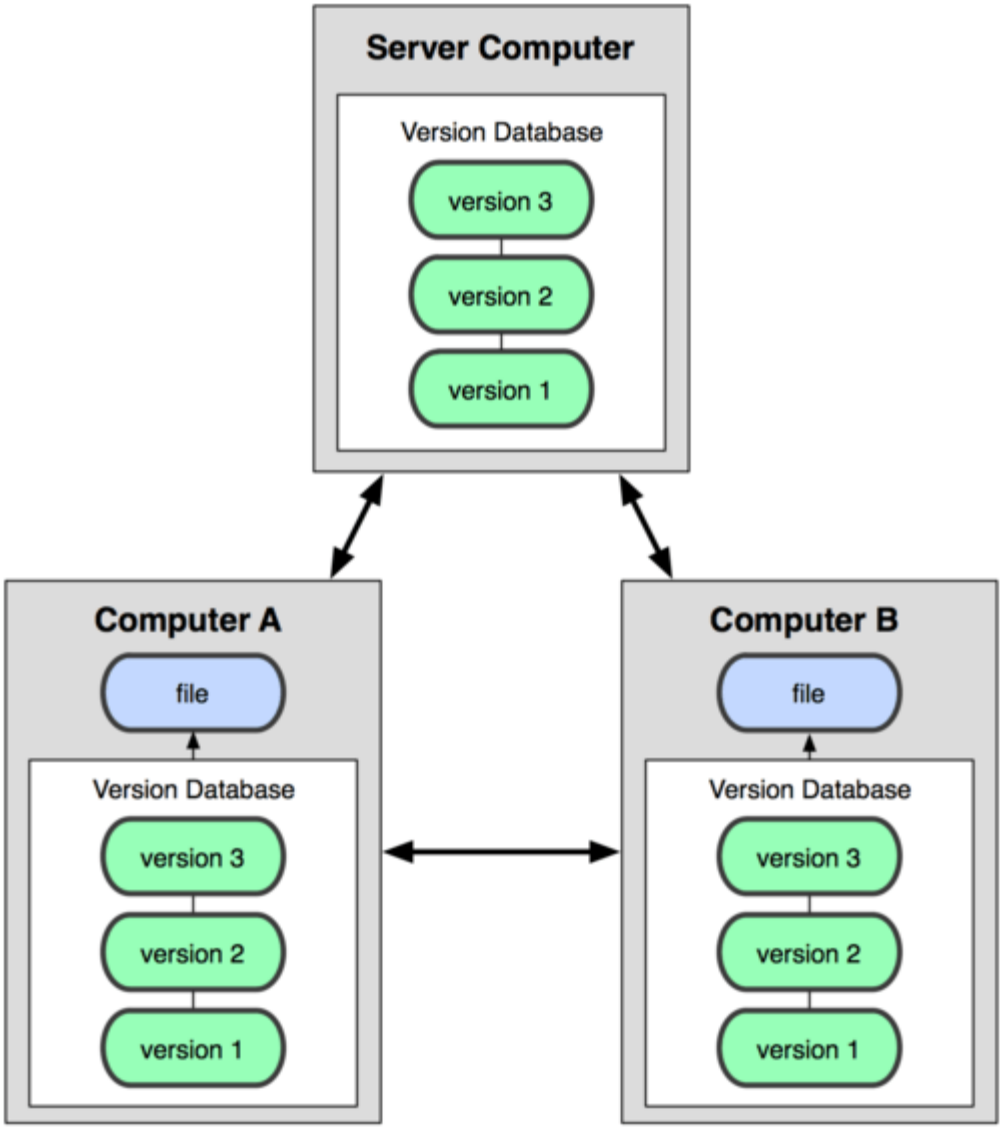

Рис. 3.3. Схема распределённой системы контроля версий

Кроме того, в большей части этих систем можно работать с несколькими удалёнными репозиториями, таким образом, можно одновременно работать по-разному с разными группами людей в рамках одного проекта. Так, в

одном проекте можно одновременно вести несколько типов рабочих процессов, что невозможно в централизованных системах.

3.2 Системы сборки приложений

Автоматизация сборки — этап написания скриптов или автоматизация широкого спектра задач, применяемого разработчиками в их повседневной деятельности. Включает в себя такие действия, как:

- компиляция исходного кода в бинарный код;
- сборка бинарного кода;
- выполнение тестов;
- разворачивание программы на производственной платформе;
- написание сопроводительной документации или описание изменений новой версии.

Исторически так сложилось, что разработчики применяли автоматизацию сборки для вызова компиляторов и линковщиков из скрипта сборки, в отличие от вызова компилятора из командной строки. Довольно просто при помощи командной строки передать один исходный модуль компилятору, а затем и линковщику для создания конечного объекта. Однако, при попытке скомпилировать или слинковать множество модулей с исходным кодом, причём в определенном порядке, осуществление этого процесса вручную при помощи командной строки выглядит слишком неудобным. Гораздо более привлекательной альтернативой является скриптовый язык, поддерживаемый утилитой Make. Данный инструмент позволяет писать скрипты сборки, определяя порядок их вызова, этапы компиляции и линковки для сборки программы. GNU Make также предоставляет такие дополнительные возможности, как например, «зависимости» («makedepend»), которые позволяют указать условия подключения исходного кода на каждом этапе сборки. Это и стало началом автоматизации сборки. Основной целью была автоматизация вызовов компиляторов и линковщиков. По мере роста и усложнения процесса сборки разработчики начали добавлять действия до и после вызовов компиляторов, как например, проверку (check-out) версий копируемых объектов на тестовую систему. Термин «автоматизация сборки» уже включает в себя управление и действия

до и после компиляции и линковки, так же как и действия при компиляции и линковке.

В последние годы решения по управлению сборкой сделали ещё более удобным и управляемым процесс автоматизированной сборки. Для выполнения автоматизированной сборки и контроля этого процесса существуют как коммерческие, так и открытые решения. Некоторые решения нацелены на автоматизацию шагов до и после вызова сборочных скриптов, а другие выходят за рамки действий до и после обработки скриптов и полностью автоматизируют процесс компиляции и линковки, избавляя от ручного написания скриптов. Такие инструменты чрезвычайно полезны для непрерывной интеграции, когда требуются частые вызовы компиляции и обработка промежуточных сборок.

Примеры утилит для сборки приложений:

- Make
- CMake
- Ant
- Maven
- Gradle

Apache Maven — утилита для автоматизации сборки проектов на основе описания их структуры в файлах на языке POM, являющемся подмножеством XML (англ. Project Object Model).

Maven, в отличие от другого сборщика проектов Apache Ant, обеспечивает декларативную, а не императивную сборку проекта. То есть, в файлах проекта pom.xml содержится его декларативное описание, а не отдельные команды. Все задачи по обработке файлов Maven выполняет посредством их обработки последовательностью встроенных и внешних плагинов.

Maven стал популярен благодаря автоматическому управлению зависимостями, которое позволяет автоматически скачивать и использовать необходимые проекту библиотеки в процессе сборки. На сегодняшний день

все крупные проекты используют автоматическое управление зависимостями в том или ином виде.

В файле pom.xml задаются зависимости, которые имеет управляемый Maven-ом пакет от других программных пакетов. Эти зависимости Maven разрешает, то есть, сначала он проверяет, находятся ли необходимые файлы в локальных каталогах или в локальном Maven-репозитории. Если зависимость не может быть локально разрешена, Maven пытается связаться с конфигурированным для него Maven-репозиторием в интранете или в Интернете и копировать необходимые данные оттуда в локальный репозиторий. По умолчанию Maven использует Maven Central Repository, но разработчик может конфигурировать и другие публичные Maven-репозитории, такие, как Apache или JCenter.

Поиск зависимостей (open-source-библиотек и модулей) ведётся по их координатам (groupId, artifactId и version). Эти координаты могут быть определены с помощью специальных поисковых машин, например, Maven search engine[10]. Введя в такой машине, например, поисковый признак «pop3», вы получите, среди прочих, и строку с groupId="com.sun.mail" и artifactId="pop3". Чаще же результаты будут менее вразумительны и только опытным путем вы сможете определить, что заказали именно тот jar-файл, который хотели.

Принадлежащие компании и расположенные в её интранете репозитории обычно реализуются с помощью менеджеров репозиториев Maven (Maven Repository Manager), таких как Apache Archiva, Nexus (ранее Proximity), Artifactory, Codehaus Maven Proxy или Dead Simple Maven Proxy.

Maven базируется на plugin-архитектуре, которая позволяет применять плагины для различных задач (compile, test, build, deploy, checkstyle, pmd, scp-transfer) для данного проекта, без необходимости их в явном виде инсталлировать. Это возможно потому, что информация поступает плагину через стандартный вход, а результаты пишутся в его стандартный выход. Это позволяет писать плагины для взаимодействия со средствами построения проекта (компиляторы, средства тестирования, и т. п.) для любого другого языка.

Количество доступных плагинов в настоящее время очень велико и включает, в том числе, плагины, позволяющие непосредственно из Maven запускать web-приложение для тестирования его в браузере; плагины, позволяющие тестировать или создавать банки данных; плагины, позволяющие генерировать Web Services. Задачей разработчика в такой ситуации является найти и применить наиболее подходящий набор плагинов.

Утилита сборки Gradle является логическим продолжением идей Maven, но ориентирована на более гибкий процесс разработки. Гибкость достигается использованием языка Groovy для описания структуры проекта и процесса сборки вместе со специализированным Gradle DSL. Gradle позволяет использовать Maven репозитории для подключения зависимостей. Как и Maven Gradle базируется на плагинной архитектуре.

В отличие от Apache Maven, основанного на концепции жизненного цикла проекта, и Apache Ant, в котором порядок выполнения задач (targets) определяется отношениями зависимости (depends-on), Gradle использует направленный ациклический граф для определения порядка выполнения задач.

Gradle был разработан для расширяемых много-проектных сборок, и поддерживает инкрементальные сборки, определяя, какие компоненты дерева сборки не изменились и какие задачи, зависимые от этих частей, не требуют перезапуска.

Основные плагины предназначены для разработки и развертывания Java, Groovy и Scala приложений, но готовятся плагины и для других языков программирования, например C/C++.

3.3 Системы Continuous Integration

Системы CI - серверное программное обеспечение, позволяющее автоматизировать сборку, тестирование и развёртывание ПО, а также дающее возможность управлять конфигурацией сборок и задач через веб-интерфейс.

Основные возможности систем CI:

- Сборка приложений с использованием популярных утилит сборки: Make, CMake, MSBuild, Ant, Maven, Gradle и т.п.;
- Запуск задач автоматизации по событию, расписанию или вручную;
- Тестирование приложений с выдачей отчётов по пройденным и проваленным тестам;
- Автоматизация рутинных действий при помощи скриптов на языках Bash/Python/Perl и т.п.
- Автоматизация развёртывания приложений;
- Автоматический выпуск релизов приложений с сохранением информации о выпуске, исходном коде, пройденных тестах и списке изменений по сравнению с прошлой версией;
- Предварительное тестирование кода перед коммитом. Предотвращает возможность коммита программного кода содержащего ошибки, нарушающие нормальную сборку проекта, путём удалённой сборки изменений перед коммитом;
- Грид-сборка проекта. Предоставляет возможность производить несколько сборок проекта одновременно, производя тестирование на разных платформах и в различном программном окружении;
- Интеграция с системами оценки покрытия кода, инспекции кода и поиска дублирования кода.

Популярные системы CI:

- Teamcity
- Jenkins
- Travis CI

В последнее время становятся популярны облачные сервисы сборки, такие как Travis CI, предоставляющие свои ресурсы для сборки приложений клиентов.

3.4 Системы управления конфигурацией распределённых приложений

Администрирование распределённых приложений осложнено внесением изменений в конфигурацию многих подсистем. На сегодняшний день

существует несколько основных систем, позволяющих упростить процесс конфигурации распределённых приложений: Chef / Puppet / Ansible.

Puppet — кроссплатформенное клиент-серверное приложение, которое позволяет централизованно управлять конфигурацией операционных систем и программ, установленных на нескольких компьютерах. Написано на языке программирования Ruby. Наряду с Chef отмечается как одно из самых актуальных средств конфигурационного управления по состоянию на 2015 год.

Puppet позволяет просто настроить и впоследствии быстро управлять практически любой сетью на базе любой операционной системы Red Hat Enterprise Linux, CentOS, Fedora, Debian, Ubuntu, OpenSUSE, Solaris, BSD, Mac OS X и Microsoft Windows.

Система Puppet достаточно популярна в среде IT-компаний, в частности, её используют Google, Яндекс, Fedora Project, Стэнфордский университет, Red Hat, Siemens IT Solution.

Узлы сети, управляемые с помощью Puppet, периодически опрашивают сервер, получают и применяют внесённые администратором изменения в конфигурацию. Конфигурация описывается на специальном декларативном предметно-ориентированном языке.

Chef — система управления конфигурациями, написанная на Ruby (клиентская часть) и Erlang (серверная часть), с использованием предметно-ориентированного языка для описания конфигураций. Используется для упрощения задач настройки и поддержки множества серверов и может интегрироваться в облачные платформы, такие как Rackspace и Amazon EC2, для автоматизации управления текущими и автоматизации процесса настройки новых серверов.

Пользователь Chef создаёт определенные «рецепты» с описанием того, как управлять серверными приложениями (например, Apache, MySQL или Hadoop) и их настроек.

«Рецепт» — это описание состояния ресурсов системы, в котором она должна находиться в конкретный момент времени, включая установленные пакеты, запущенные службы, созданные файлы. Chef проверяет, что каждый из ресурсов системы настроен правильно и пытается исправить состояние ресурса, если оно не соответствует ожидаемому.

Chef может работать как в режиме клиент-сервер, так и в режиме автономной конфигурации, называемом «chef-solo». В режиме клиент-сервер клиент посылает на сервер различные свойства хоста, на котором он расположен. На стороне сервера используется Solr для индексирования свойств и предоставления API для запроса информации клиентом. «Рецепты» могут запрашивать эти свойства и использовать полученные данные для настройки хоста.

Обычно используется для управления Linux-узлами, но последние версии поддерживают Windows.

Ansible — Open Source программное решение для удаленного управления конфигурациями. Название продукта взято из научно-фантастической литературы: в романах американской писательницы Урсулы Ле Гуин ансиблом называется устройство для оперативной космической связи.

Ansible берет на себя всю работу по приведению удаленных серверов в необходимое состояние. Администратору необходимо лишь описать, как достичь этого состояния с помощью так называемых сценариев (playbooks; это аналог рецептов в Chef). Такая технология позволяет очень быстро осуществлять переконфигурирование системы: достаточно всего лишь добавить несколько новых строк в сценарий.

Преимущества Ansible по сравнению с другими аналогичными решениями заключаются в следующем:

- на управляемые узлы не нужно устанавливать никакого дополнительного ПО, всё работает через SSH (в случае необходимости дополнительные модули можно взять из официального репозитория);

- код программы, написанный на Python, очень прост; при необходимости написание дополнительных модулей не составляет особого труда;
- язык, на котором пишутся сценарии, также предельно прост;
- низкий порог вхождения: обучиться работе с Ansible можно за очень короткое время;
- документация к продукту написана очень подробно и вместе с тем — просто и понятно, она регулярно обновляется;
- Ansible работает не только в режиме push, но и pull, как это делают большинство систем управления (Puppet, Chef);
- имеется возможность последовательного обновления состояния узлов (rolling update).

3.5 Развёртывание настольных приложений

Для быстрого развёртывания настольных приложений поставщиками ПО используются специальные технологии запуска и обновления. Самые популярные носят названия Java Web Start (Oracle) и Click Once (Microsoft).

Java Web Start (часто JavaWS) — технология компании Sun Microsystems, позволяющая запускать приложения на Java из браузера. Основана на протоколе Java Network Launching Protocol (JNLP). В отличие от апплетов приложения Web Start запускаются не в окне браузера и не имеют с ним прямой связи.

Различия между Java Web Start и аплетами:

- Технология Java Web Start используется для доставки обычных приложений, написанных на языке Java и начинающихся с вызова метода main, содержащегося в одном из классов.
- Приложения, созданные с помощью технологии Java Web Start, не запускаются внутри браузера. Они отображаются вне браузера.
- Приложения, созданные с помощью технологии Java Web Start, можно запустить с помощью браузера, однако механизм, лежащий в основе этого процесса, совершенно отличен от запуска аплетов. Браузеры тесно связаны с системой поддержки выполнения

программ, написанных на языке Java, которая запускает аплеты. Технология Java Web Start гораздо самостоятельнее. Браузер просто запустит внешнюю программу, как только загрузит дескриптор приложения, созданного с помощью технологии Java Web Start. Для этого используется тот же механизм, что и в приложениях Adobe Acrobat или RealAudio. Даже конкурирующие поставщики браузеров не могут вмешаться в работу этого механизма.

- После загрузки приложения, созданного по технологии Java Web Start, оно запускается вне браузера.
- Технология Java Web Start обеспечивает более мощную поддержку кэширования и автоматического обновления программ по сравнению с технологией Java Plug-In. (В будущем эти два подхода объединятся, с тем чтобы использовать одни и те же средства управления процессом развертывания программ.)
- Механизм «песочниц» в технологии Java Web Start более гибок и позволяет неподписанным приложениям получать доступ к локальным ресурсам.

Протокол JNLP описывает запуск приложений Java Web Start. JNLP состоит из набора правил, определяющих, как конкретно реализуется запускающий механизм. Файлы JNLP включают такую информацию, как месторасположение jar архивов, имя главного класса приложения. Правильно сконфигурированный браузер передает JNLP файлы среде JRE, которая загружает приложение на компьютер клиента и запускает его.

Для того чтобы подготовить приложение к доставке с помощью технологии Java Web Start, необходимо запаковать его в один или несколько JAR-файлов. Затем нужно подготовить дескрипторный файл в формате JNLP (Java Network Launch Protocol — сетевой протокол запуска приложений на языке Java). Теперь разместите файлы на Web-сервере. После этого необходимо убедиться, что ваш Web-сервер распознает тип MIME в каталоге application/x-java-jnlp-file в файлах с расширением .jnlp (браузеры используют тип MIME для распознавания программы, которую следует запустить).

ClickOnce — технология Майкрософт для развёртывания приложений, основанных на фреймворках Windows Forms или Windows Presentation Foundation. Она подобна технологии Java Web Start для Java Platform.

ClickOnce позволяет пользователю устанавливать и запускать Windows приложение, кликая по ссылке на веб-странице, либо в сетевом окружении. Основной принцип ClickOnce — простое развёртывание Windows-приложений пользователем. Кроме того, ClickOnce нацелена на решения трёх других проблем, связанных с обычной моделью развертывания:

- сложность в обновлении развертываемого приложения;
- воздействие приложения на компьютер пользователя;
- необходимость административных полномочий для установки приложения.

ClickOnce-приложения изолированы друг от друга, и одно приложение не может повлиять на работу других.

Лекция 4. Развёртывание приложений в облачной инфраструктуре (2 часа)

Инфраструктура как услуга (IaaS, англ. Infrastructure-as-a-Service) предоставляется как возможность использования облачной инфраструктуры для самостоятельного управления ресурсами обработки, хранения, сетями и другими фундаментальными вычислительными ресурсами, например, потребитель может устанавливать и запускать произвольное программное обеспечение, которое может включать в себя операционные системы, платформенное и прикладное программное обеспечение. Потребитель может контролировать операционные системы, виртуальные системы хранения данных и установленные приложения, а также обладать ограниченным контролем за набором доступных сетевых сервисов (например, межсетевым экраном, DNS). Контроль и управление основной физической и виртуальной инфраструктурой облака, в том числе сети, серверов, типов используемых операционных систем, систем хранения осуществляется облачным провайдером.

Платформа как услуга (PaaS, англ. Platform-as-a-Service) — модель, когда потребителю предоставляется возможность использования облачной инфраструктуры для размещения базового программного обеспечения для последующего размещения на нём новых или существующих приложений (собственных, разработанных на заказ или приобретённых тиражируемых приложений). В состав таких платформ входят инструментальные средства создания, тестирования и выполнения прикладного программного обеспечения — системы управления базами данных, связующее программное обеспечение, среды исполнения языков программирования — предоставляемые облачным провайдером.

Контроль и управление основной физической и виртуальной инфраструктурой облака, в том числе сети, серверов, операционных систем, хранения осуществляется облачным провайдером, за исключением разработанных или установленных приложений, а также, по возможности, параметров конфигурации среды (платформы).

Платформа приложений как услуга (aPaaS, англ. Application Platform as a Service) – облачный сервис, который предоставляет окружения для разработки и развёртывания веб-приложений, обычно на специфичной платформе и технологии. Одним из примеров aPaaS является платформа CloudFoundry, или сервис Jelastic. Jelastic предоставляет средства автоматизации развёртывания приложений на платформе Java, включая сервера приложений, базы данных и средства балансировки нагрузки.

Docker — программное обеспечение для автоматизации развёртывания и управления приложениями в среде виртуализации на уровне операционной системы, например LXC. Позволяет «упаковать» приложение со всем его окружением и зависимостями в контейнер, который может быть перенесён на любой Linux-системе с поддержкой cgroups в ядре, а также предоставляет среду по управлению контейнерами.

Программное обеспечение функционирует в среде Linux с ядром, поддерживающим cgroups и изоляцию пространств имён (namespaces); существуют сборки только для платформы x86-64.

Для экономии дискового пространства проект использует файловую систему Aufs с поддержкой технологии каскадно-объединённого монтирования: контейнеры используют образ базовой операционной системы, а изменения записываются в отдельную область. Также поддерживается размещение контейнеров в файловой системе Btrfs с включённым режимом копирования при записи.

В состав программных средств входит демон — сервер контейнеров (запускается командой docker -d), клиентские средства, позволяющие из интерфейса командной строки управлять образами и контейнерами, а также API, позволяющий в стиле REST управлять контейнерами программно.

Демон обеспечивает полную изоляцию запускаемых на узле контейнеров на уровне файловой системы (у каждого контейнера собственная корневая файловая система), на уровне процессов (процессы имеют доступ только к собственной файловой системе контейнера, а ресурсы разделены средствами LXC), на уровне сети (каждый контейнер имеет доступ только к

привязанному к нему сетевому пространству имён и соответствующим виртуальным сетевым интерфейсам).

Набор клиентских средств позволяет запускать процессы в новых контейнерах (docker run), останавливать и запускать контейнеры (docker stop и docker start), приостанавливать и возобновлять процессы в контейнерах (docker pause и docker unpause). Серия команд позволяет осуществлять мониторинг запущенных процессов (docker ps по аналогии с ps в Unix-системах, docker top по аналогии с top и другие). Новые образы можно создавать из специального сценарного файла (docker build, файл сценария носит название dockerfile), возможно записать все изменения, сделанные в контейнере в новый образ (docker commit). Все команды могут работать как с docker-демоном локальной системы, так и с любым сервером docker, доступным по сети. Кроме того, в интерфейсе командной строки встроены возможности по взаимодействию с публичным репозиторием Docker Hub, в котором размещены предварительно собранные образы контейнеров, например, команда docker search позволяет осуществить поиск образов среди размещённых в нём, образы можно скачивать в локальную систему (docker pull), возможно также отправить локально собранные образы в Docker Hub (docker push).

В своем ядре docker позволяет запускать практически любое приложение, безопасно изолированное в контейнере. Безопасная изоляция позволяет вам запускать на одном хосте много контейнеров одновременно. Легковесная природа контейнера, который запускается без дополнительной нагрузки гипервизора, позволяет вам добиваться больше от вашего железа.

Платформа и средства контейнерной виртуализации могут быть полезны в следующих случаях:

- упаковка вашего приложения (и так же используемых компонент) в docker контейнеры;
- раздача и доставка этих контейнеров вашим командам для разработки и тестирования;

- загрузка этих контейнеров в ваши целевые окружения, как в дата центры, так и в облака.

Быстрое развёртывание приложений

Docker прекрасно подходит для организации цикла разработки. Docker позволяет разработчикам использовать локальные контейнеры с приложениями и сервисами. Что позволяет интегрироваться с процессом постоянной интеграции и развёртывания (continuous integration and deployment workflow).

Например, разработчики пишут код локально и делятся своим стеком разработки (набором docker образов) с коллегами. Когда они готовы, отравляют код и контейнеры на тестовую площадку и запускают любые необходимые тесты. С тестовой площадки они могут оправить код и образы на продакшен.

Простое развёртывание

Основанная на контейнерах docker платформа позволят легко портировать вашу полезную нагрузку. Docker контейнеры могут работать на вашей локальной машине, как реальной так и на виртуальной машине в дата центре, так и в облаке.

Портируемость и легковесная природа docker позволяет легко динамически управлять вашей нагрузкой. Вы можете использовать docker, чтобы развернуть или погасить ваше приложение или сервисы. Скорость docker позволяет делать это почти в режиме реального времени.

Высокие нагрузки и больше полезных нагрузок

Docker легковесен и быстр. Он предоставляет устойчивую, рентабельную альтернативу виртуальным машинам на основе гипервизора. Он особенно полезен в условиях высоких нагрузок, например, при создания собственного облака или платформа-как-сервис (platform-as-service). Но он так же полезен для маленьких и средних приложений, когда вам хочется получать больше из имеющихся ресурсов.

Docker состоит из двух главных компонент:

- Docker: платформа виртуализации с открытым кодом;
- Docker Hub: платформа-как-сервис для распространения и управления docker контейнерами.

Docker использует архитектуру клиент-сервер. Архитектура Docker показана на рис. 4.1. Docker клиент общается с демоном Docker, который берет на себя тяжесть создания, запуска, распределения ваших контейнеров. Оба, клиент и сервер могут работать на одной системе, вы можете подключить клиент к удаленному демону docker. Клиент и сервер общаются через сокет или через RESTful API.

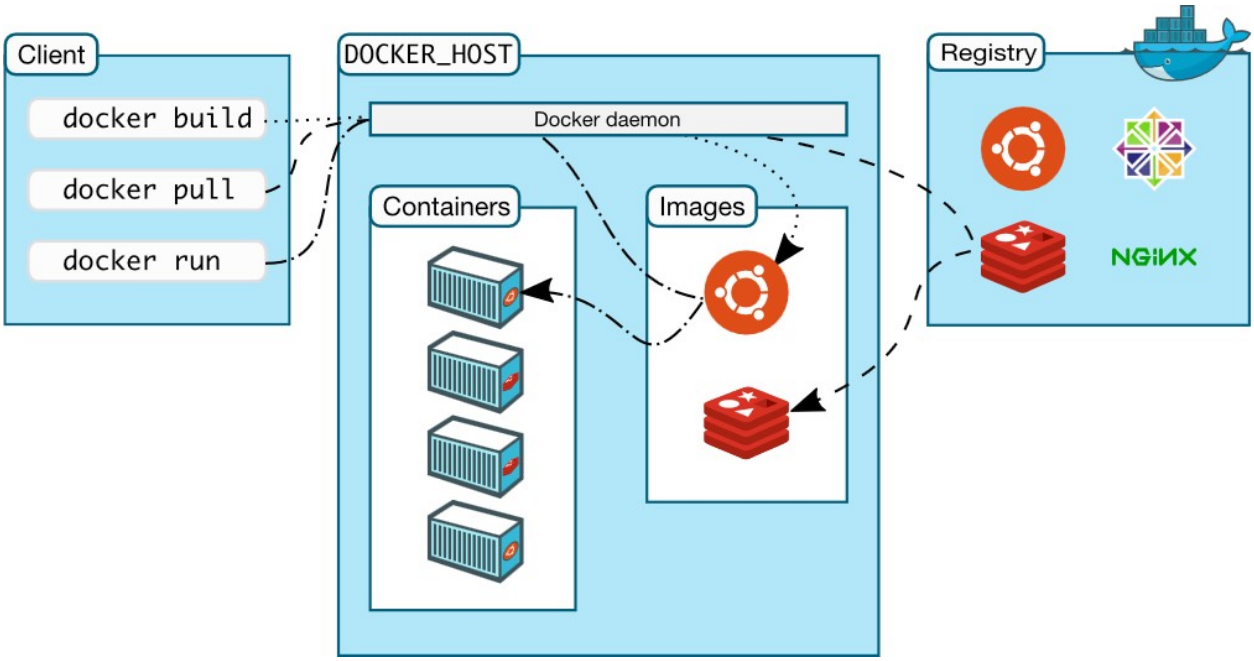

Рис. 4.1 – Архитектура Docker

Как показано на диаграмме, Docker-демон запускается на хост-машине. Пользователь не взаимодействие с сервером напрямую, а использует для этого клиент.

Docker-клиент, программа docker — главный интерфейс к Docker. Она получает команды от пользователя и взаимодействует с docker-демоном.

Чтобы понимать, из чего состоит docker, вам нужно знать о трех компонентах:

- образы (images)

- реестр (registries)
- контейнеры

Docker-образ — это read-only шаблон. Например, образ может содержать операционную систему Ubuntu с Apache и приложением на ней. Образы используются для создания контейнеров. Docker позволяет легко создавать новые образы, обновлять существующие, или вы можете скачать образы созданные другими людьми. Образы — это компонента сборки docker-а.

Docker-реестр хранит образы. Есть публичные и приватные реестры, из которых можно скачать либо загрузить образы. Публичный Docker-реестр — это Docker Hub. Там хранится огромная коллекция образов. Как вы знаете, образы могут быть созданы вами или вы можете использовать образы созданные другими. Реестры — это компонента распространения.

Контейнеры похожи на директории. В контейнерах содержится все, что нужно для работы приложения. Каждый контейнер создается из образа. Контейнеры могут быть созданы, запущены, остановлены, перенесены или удалены. Каждый контейнер изолирован и является безопасной платформой для приложения. Контейнеры — это компонента работы.

Лекция 5. Методология DevOps (2 часа)

DevOps (слияние англ. слов Development (разработка) и Operations (ИТ-операции)) – это новая методология разработки ПО. Она сосредоточена на коммуникации, сотрудничестве и интеграции между подразделениями разработки и эксплуатации. DevOps – это ответ на взаимную зависимость разработчиков и персонала ИТ-операций. Данная методология помогает организациям ускорить производство программных продуктов и предоставление своих сервисов.

Например, в производственной системе Toyota, роль профессионала DevOps схожа с должностью главного инженера. Этот сотрудник отвечает за успех проекта, но формально не руководит различным командами, задействованными на данном проекте. Деятельность специалиста DevOps требует знания используемых технологий и управленческой модели, чтобы убеждать менеджеров проектов принимать эффективные решения. Этому способствует авторитет «главного инженера», признанный руководством предприятия.

Большинство идей для DevOps взяты из методологии управления системами предприятия и движения Agile (операции и инфраструктура).

Термин "DevOps" был популяризирован на конференции «Дни DevOps» ("DevOps Days") в 2009 году в Бельгии. После этого «Дни DevOps» проводились в Индии, США, Бразилии, Австралии, Германии и Швеции.

Обычно новые методологии разработки (такие как Agile), внедряются в организациях в рамках отдельных департаментов:

- Отдел разработки (англ. Development/Software Engineering)
- Отдел ИТ-операций (англ. Technology Operations)
- Отдел обеспечения качества (англ. Quality Assurance)

Деятельность по разработке и внедрению ПО ранее не требовала глубокой интеграции между департаментами. Но на сегодняшний день необходимо тесное сотрудничество всех вышеуказанных отделов.

DevOps это нечто большее, чем просто развёртывание программного обеспечения - а именно совокупность процессов и методов, поддерживающих связь и сотрудничество между департаментами. Компаниям, которые выпускают релизы очень часто, желательно знать и применять методологию DevOps. Руководство Flickr приняло решение внедрить у себя DevOps для выполнения серьёзного бизнес-требования – 10 развёртываний в день. Этот ежедневный цикл мог бы требовать более частых развертываний в организациях, которые производят многофункциональные приложения с большим количеством модулей. Такой подход схож с методологиями непрерывная интеграция и непрерывная поставка и часто ассоциируется с методологией Lean Startup. Эта тема обсуждается в рабочих группах, профессиональных ассоциация и блогах с 2009 года. Рис. 5.1 демонстрирует DevOps как пересечение деятельности отделов Разработки, ИТ-операций и Обеспечения качества.

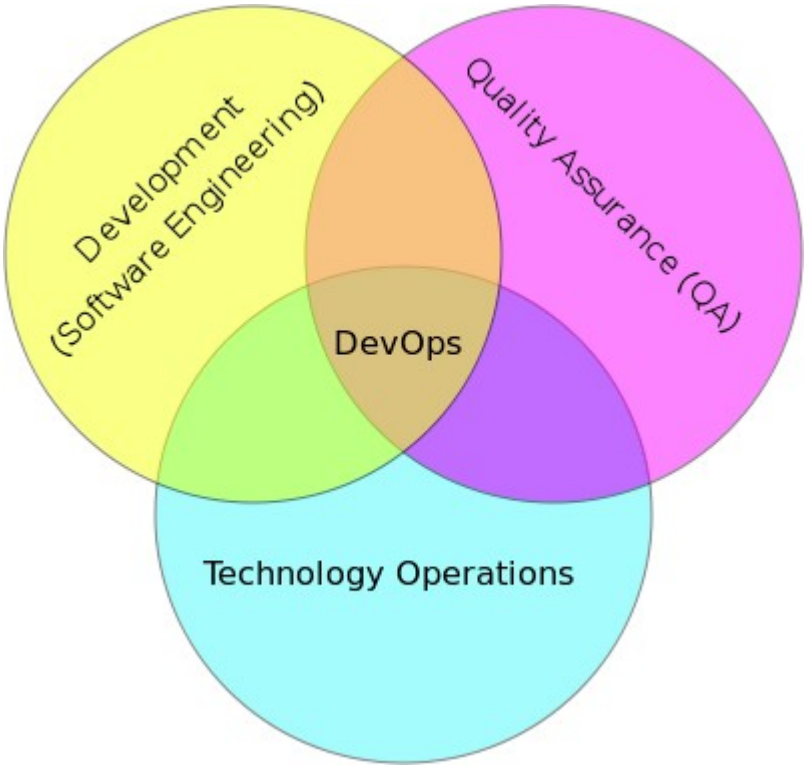

Рис. 5.1 – Место методологий DevOps среди других дисциплин

DevOps внедряется с целью повышения надежности, безопасности и ускорения цикла разработки и цикла развертывания. При этом, ключевое значение имеют следующие процессы:

- поставка продукта

- тестирование качества
- добавление нового функционала
- минорные релизы

Всеобщее признание DevOps обеспечивается за счёт таких факторов:

- Использование методологии Agile и связанных с ней процессов разработки;
- Требование увеличения частоты выпуска релизов от владельцев приложений и бизнесов;
- Широкое распространение виртуализированной и облачной инфраструктура, предоставляемой внутренними и внешними провайдерами;
- Распространение автоматизации дата-центров и инструментов конфигурационного управления.

DevOps часто описывается как более сплочённое и продуктивное сотрудничество между разработчиками и ИТ-персоналом. Это улучшение связи и сотрудничества повышает эффективность и снижает производственные риски, связанные с частым изменениям. Некоторые предприятия уже начинают применять показатели для оценки влияния DevOps на ROI и эффективность разработки.

Лекция 6. Мониторинг работы программного обеспечения (2 часа)

Средства контроля/мониторинга позволяют следить за процессами, происходящими в компьютерной системе. При этом возможны два подхода: наблюдение в реальном режиме времени или контроль с записью результатов в специальном протокольном файле. Первый подход обычно используют при изыскании путей для оптимизации работы вычислительной системы и повышения ее эффективности. Второй подход используют, когда мониторинг выполняется автоматически и (или) дистанционно, о последнем случае результаты мониторинга можно передать удаленной службе технической поддержки для установления причин конфликтов в работе программного и аппаратного обеспечения.

В любой сети, где есть больше, чем один сервер, очень полезно бывает иметь перед глазами полную картину происходящего. В крупных сетях, где количество хостов переваливает за несколько десятков, следить за каждым в отдельности — непосильная задача для администраторов. Для облегчения задачи наблюдения применяются системы мониторинга.

6.1 Zabbix

Zabbix — свободная система мониторинга и отслеживания статусов разнообразных сервисов компьютерной сети, серверов и сетевого оборудования.

Система Zabbix состоит из нескольких частей, и при большой нагрузке и наблюдении за очень большим количеством хостов позволяет разнести эти части на несколько раздельных машин.

Архитектура

- Zabbix сервер — это ядро программного обеспечения Zabbix. Сервер может удаленно проверять сетевые сервисы, является хранилищем, в котором хранятся все конфигурационные, статистические и оперативные данные, и он является тем субъектом в программном

обеспечении Zabbix, который оповестит администраторов в случае возникновения проблем с любым контролируемым оборудованием;

- Zabbix прокси — собирает данные о производительности и доступности от имени Zabbix сервера. Все собранные данные заносятся в буфер на локальном уровне и передаются Zabbix серверу, к которому принадлежит прокси-сервер. Zabbix прокси является идеальным решением для централизованного удаленного мониторинга мест, филиалов, сетей, не имеющих локальных администраторов. Он может быть также использован для распределения нагрузки одного Zabbix сервера. В этом случае, прокси только собирает данные, тем самым на сервер ложится меньшая нагрузка на ЦПУ и на ввод-вывод диска;
- Zabbix агент — контроль локальных ресурсов и приложений (таких как жесткие диски, память, статистика процессора и т. д.) на сетевых системах, эти системы должны работать с запущенным Zabbix агентом. Zabbix агенты являются чрезвычайно эффективными из-за использования родных системных вызовов для сбора информации о статистике;
- Веб-интерфейс — интерфейс является частью Zabbix сервера, и, как правило (но не обязательно), запущен на том же физическом сервере, что и Zabbix сервер. Работает на PHP, требует веб сервер (напр. Apache).

Возможности Zabbix:

- Распределённый мониторинг вплоть до 1000 узлов. Конфигурация младших узлов полностью контролируется старшими узлами, находящихся на более высоком уровне иерархии;
- Сценарии на основе мониторинга;
- Автоматическое обнаружение;
- Централизованный мониторинг лог-файлов;
- Веб-интерфейс для администрирования и настройки;
- Отчетность и тенденции;
- SLA мониторинг;

- Поддержка высокопроизводительных агентов (zabbix-agent) практически для всех платформ;
- Комплексная реакция на события;
- Поддержка SNMP v1, 2, 3;
- Поддержка SNMP ловушек;
- Поддержка IPMI;
- Поддержка мониторинга JMX приложений из коробки;
- Поддержка выполнения запросов в различные базы данных без необходимости использования скриптовой обвязки;
- Расширение за счет выполнения внешних скриптов;
- Гибкая система шаблонов и групп;
- Возможность создавать карты сетей.

6.2 Nagios

Nagios — программа с открытым кодом, предназначенная для мониторинга компьютерных систем и сетей: наблюдения, контроля состояния вычислительных узлов и служб, оповещения администратора в том случае, если какие-то из служб прекращают (или возобновляют) свою работу.

Первоначально Nagios была разработана для работы под Linux, но она также хорошо работает и под другими ОС, такими как Sun Solaris, FreeBSD, AIX и HP-UX.

Возможности Nagios:

- Мониторинг сетевых служб (SMTP, POP3, HTTP, NNTP, ICMP, SNMP);
- Мониторинг состояния хостов (загрузка процессора, использование диска, системные логи) в большинстве сетевых операционных систем;
- Поддержка удаленного мониторинга через шифрованные туннели SSH или SSL;
- Простая архитектура модулей расширений (плагинов) позволяет, используя любой язык программирования по выбору (Shell, C++,

Perl, Python, PHP, C# и другие), легко разрабатывать свои собственные способы проверки служб;

- Параллельная проверка служб;
- Возможность определять иерархии хостов сети с помощью «родительских» хостов, позволяет обнаруживать и различать хосты, которые вышли из строя, и те, которые недоступны;
- Отправка оповещений в случае возникновения проблем со службой или хостом (с помощью почты, пейджера, смс, или любым другим способом, определенным пользователем через модуль системы);
- Возможность определять обработчики событий произошедших со службами или хостами для проактивного разрешения проблем;
- Автоматическая ротация лог-файлов;
- Возможность организации совместной работы нескольких систем мониторинга с целью повышения надёжности и создания распределенной системы мониторинга;
- Включает в себя утилиту nagiostats, которая выводит общую сводку по всем хостам, по которым ведется мониторинг.

6.3 Munin

Munin - это приложение для мониторинга серверов и обычных клиентских компьютеров под управлением Linux, написанное на языке Perl. Программа создает вывод изменений характеристик системы в виде графиков, встроенных в html страничку. По умолчанию осуществляется мониторинг использования файловой системы, памяти, процессора, активности сетевых служб и др. В принципе, вам должно этого хватить. Если же нужно отслеживать какие-нибудь специфические параметры, то можно добавить дополнительные плагины из уже созданных или написать самому.

В состав Munin входят пакеты как для сервера (munin), так и для клиентов (munin-node). Серверную часть нужно устанавливать только на самом сервере, клиентскую, как на сервере (если вы хотите анализировать и его), так и на всех клиентских машинах.

Munin поддерживает плагины, позволяющие расширить спектр объектов мониторинга.

Лекция 7. Операционные системы для развёртывания распределённых приложений. Введение в Linux (2 часа)

Linux - общее название Unix-подобных операционных систем, основанных на одноимённом ядре. Ядро Linux создаётся и распространяется в соответствии с моделью разработки свободного и открытого программного обеспечения. Поэтому общее название не подразумевает какой-либо единой «официальной» комплектации Linux; они распространяются в основном бесплатно в виде различных готовых дистрибутивов, имеющих свой набор прикладных программ и уже настроенных под конкретные нужды пользователя.

Дистрибутивы на основе Linux имеют широкое применение в различных областях: от встраиваемых систем до суперкомпьютеров, надёжно удерживают лидирующие позиции на рынке серверов, как правило, в составе комплекса серверного программного обеспечения LAMP.

Самая популярная ОС для смартфонов и планшетных компьютеров — Android, также основана на ядре Linux.

Также растёт использование Linux в качестве настольной системы для дома и офиса.

Дистрибутивы Linux пользуются популярностью у различных государственных структур: Федеральное правительство Бразилии хорошо известно своей поддержкой Linux, а российские военные разрабатывают свой собственный дистрибутив Linux.

Linux-системы представляют собой модульные Unix-подобные операционные системы. В большей степени дизайн Linux -систем базируется на принципах, заложенных в Unix в течение 1970-х и 1980-х годов. Такая система использует монолитное ядро Linux, которое управляет процессами, сетевыми функциями, периферией и доступом к файловой системе.

Драйверы устройств либо интегрированы непосредственно в ядро, либо добавлены в виде модулей, загружаемых во время работы системы.

Отдельные программы, взаимодействуя с ядром, обеспечивают функции системы более высокого уровня. Например, пользовательские компоненты GNU являются важной частью большинства Linux-систем, включающей в себя наиболее распространённые реализации библиотеки языка Си, популярных оболочек операционной системы, и многих других общих инструментов Unix, которые выполняют многие основные задачи операционной системы.

Графический интерфейс пользователя (или GUI) в большинстве систем Linux построен на основе X Window System.

В Linux-системах пользователи работают через интерфейс командной строки (CLI), графический интерфейс пользователя (GUI), или, в случае встраиваемых систем, через элементы управления соответствующих аппаратных средств. Настольные системы, как правило, имеют графический пользовательский интерфейс, в котором командная строка доступна через окно эмулятора терминала или в отдельной виртуальной консоли.

Большинство низкоуровневых компонентов Linux, включая пользовательские компоненты GNU, используют исключительно командную строку. Командная строка особенно хорошо подходит для автоматизации повторяющихся или отложенных задач, а также предоставляет очень простой механизм межпроцессного взаимодействия.

Программа графического эмулятора терминала часто используется для доступа к командной строке с рабочего стола Linux. Linux-системы обычно реализуют интерфейс командной строки при помощи оболочки операционной системы, которая также является традиционным способом взаимодействия с системой Unix. Дистрибутивы, специально разработанные для серверов, могут использовать командную строку в качестве единственного интерфейса.

На настольных системах наибольшей популярностью пользуются пользовательские интерфейсы, основанные на таких средах рабочего стола как KDE Plasma Desktop, GNOME и Xfce, хотя также существует целый ряд

других пользовательских интерфейсов. Самые популярные пользовательские интерфейсы основаны на X Window.

«X Window» предоставляет прозрачность сети и позволяет графическим приложениям, работающим на одном компьютере, отображаться на другом компьютере, на котором пользователь может взаимодействовать с ними. Другие графические интерфейсы, такие как FVWM, Enlightenment и Window Maker, могут быть классифицированы как простые менеджеры окон X Window System, которые предоставляют окружение рабочего стола с минимальной функциональностью.

Оконный менеджер предоставляет средства для управления размещением и внешним видом отдельных окон приложений, а также взаимодействует с X Window System. Окружение рабочего стола включает в себя оконные менеджеры, как часть стандартной установки: (Metacity для GNOME, KWin для KDE, Xfwm для Xfce с 2010 года), хотя пользователь при желании может выбрать другой менеджер окон.

В апреле 2011 года семейство операционных систем на базе ядра Linux — четвёртое по популярности в мире среди клиентов Всемирной паутины (включая мобильные телефоны). По разным данным, их популярность составляет от 1,5 до 5%. На рынке веб-серверов доля Linux порядка 32% (64,1% указаны как доля Unix). По данным TOP500, Linux используется на 96 % самых мощных суперкомпьютеров планеты.

Можно выделить несколько основных областей, где нередко можно встретить Linux:

- Серверы, требующие высокого аптайма.
- Компьютеры нестандартной архитектуры (например, суперкомпьютеры) — из-за возможности быстрой адаптации ядра операционной системы и большого количества ПО под нестандартную архитектуру.
- Системы военного назначения (например, МСВС РФ) — по соображениям безопасности.

- Компьютеры, встроенные в различные устройства (банкоматы, терминалы оплаты, мобильные телефоны, маршрутизаторы, стиральные машины и даже беспилотные военные аппараты) — из-за широких возможностей по конфигурированию Linux под задачу, выполняемую устройством, а также отсутствия платы за каждое устройство.
- Массовые специализированные рабочие места (например, тонкие клиенты, нетбуки) — также из-за отсутствия платы за каждое рабочее место и по причине их ограниченной вычислительной мощности, которой может не хватать для проприетарных ОС.
- Старые компьютеры с ограниченными ресурсами быстродействия и оперативной памяти, для них используются быстрые рабочие окружения или оконные менеджеры, не требовательные к ресурсам (например, LXDE, Openbox, Xfce, Fluxbox).

Большинство пользователей для установки Linux используют дистрибутивы. Дистрибутив — это не просто набор программ, а ряд решений для разных задач пользователей, объединённых едиными системами установки, управления и обновления пакетов, настройки и поддержки.

Самые распространённые в мире дистрибутивы:

- Debian GNU/Linux — один из старейших дистрибутивов, разрабатываемый обширным сообществом разработчиков. Служит основой для создания множества других дистрибутивов. Отличается строгим подходом к включению несвободного ПО;
- Ubuntu — дистрибутив, основанный на Debian и быстро завоевавший популярность. Поддерживается сообществом, разрабатывается Canonical Ltd. Основная сборка ориентирована на лёгкость в освоении и использовании, при этом существуют серверная и минимальная сборки;
- openSuse — дистрибутив, разрабатываемый сообществом при поддержке компании Novell. Отличается удобством в настройке и обслуживании благодаря использованию утилиты YaST;

- Fedora — поддерживается сообществом и корпорацией RedHat, предшествует выпускам коммерческой версии RHEL;
- CentOS — дистрибутив Linux, основанный на свободных исходных текстах коммерческого дистрибутива Red Hat Enterprise Linux компании Red Hat, и совместимый с ним. Срок поддержки каждой версии CentOS составляет 7лет. Новая версия CentOS выходит раз в 2 года и каждая версия регулярно обновляется (каждые 6 месяцев) для поддержки новых аппаратных средств. В результате это приводит к безопасной, легко обслуживаемой, надёжной, предсказуемой и масштабируемой Linux среде.

Помимо перечисленных, существует множество других дистрибутивов, как базирующихся на перечисленных, так и созданных с нуля и зачастую предназначенных для выполнения ограниченного количества задач.

Каждый из них имеет свою концепцию, свой набор пакетов, свои достоинства и недостатки. Ни один не может удовлетворить всех пользователей, а потому рядом с лидерами благополучно существуют другие фирмы и объединения программистов, предлагающие свои решения, свои дистрибутивы, свои услуги. Существует множество LiveCD, построенных на основе Linux, например, Knoppix. LiveCD позволяет запускать Linux непосредственно с компакт-диска, без установки на жёсткий диск.

Система управления пакетами — набор программного обеспечения, позволяющего управлять процессом установки, удаления, настройки и обновления различных компонентов программного обеспечения. Системы управления пакетами активно используются в различных дистрибутивах операционной системы Linux и других UNIX-подобных операционных системах.

Программное обеспечение представляется в виде особых пакетов, содержащих помимо дистрибутива программного обеспечения набор определённых метаданных, которые могут включать в себя полное имя пакета, номер версии, описание пакета, имя разработчика, контрольную сумму, отношения с другими пакетами. Метаданные сохраняются в системной базе данных пакетов.

Наиболее известные системы управления пакетами Linux:

- RPM — система управления пакетами, изначально разрабатываемая компанией Red Hat для операционной системы Red Hat Linux. Ныне RPM применяется во множестве дистрибутивов операционной системы Linux, например, Fedora, RHEL, ASP Linux, ALT Linux, Mandriva, openSUSE;
- DEB / dpkg — система управления пакетами, используемая в операционной системе Debian и различных дистрибутивах, основанных на ней, например Ubuntu;
- Pacman — официальный менеджер пакетов в дистрибутиве Arch Linux. Является мощной системой управления пакетами и в то же время простой в изучении. Позволяет с легкостью управлять и настраивать под себя пакеты, вне зависимости от того, из официального ли они репозитория Arch или собранные самостоятельно.

Для работы с DEB пакетами существует множество инструментов, но пожалуй самый простой и часто используемый, это apt-get, входящий в стандартный набор инструментов. apt-get позволяет не только с легкостью устанавливать новые пакеты в систему, но и показывать какие пакеты доступны для установки и скачивать их с интернета в случае необходимости. Для установки приложения, введите в командную строку:

> apt-get install имя_пакета

Для удаления:

> apt-get remove имя_пакета

Несмотря на то, что содержимое пакетов может хранится на сервере в интернете или где нибудь на диске, APT хранит локальную базу данных со списком всех доступных для установки пакетов и ссылок где их брать. Эту базу необходимо периодически обновлять. Для обновления базы APT используется команда:

> apt-get update

Очень часто программы изменяются (выходят обновления, патчи, системы безопасности и т.д.), можно также использовать APT для обновления устаревших пакетов (программ) в системе. Для обновления нужно сначала обновить список пакетов, а потом уже установить все обновления. Для этого выполняется следующая команда, которая сразу сделает все необходимое:

> apt-get update; apt-get dist-upgrade

Стандартная программа для управления этими пакетами — dpkg, часто используемая с помощью apt и aptitude. Deb-пакеты могут быть преобразованы в другие пакеты, и наоборот, с помощью программы alien. Создают пакеты deb обычно с помощью утилит dpkg — в частности, dpkg-buildpackage. Основы создания пакетов описаны в Руководстве нового сопровождающего Debian и Справочнике разработчика Debian.