Автоматизация развертывания научного ПО в облачную инфраструктуру

М.Г. Щербаков[1,2], В.М. Соловьев[1], П.В. Ирматов[1]

Саратовский Государственный Университет[1], Grid Dynamics Consulting[2]

Проведено исследование возможности автоматизации развертывания научного программного обеспечения в облачной инфраструктуре. Показано, как упростить систему развертывания с использованием специализированного фреймворка Chef. Описан разработанный авторами программный комплекс, обеспечивающий развертывание высокопроизводительного вычислительного кластера в облачной инфраструктуре путем выполнения не более одной команды. В качестве примеров, разработана автоматизированная система развертывания фреймворка параллельного выполнения программ Hadoop и пакета конечно-элементного моделирования Elmer.

1. Введение

В современных условиях быстрого развития информационных технологий, в том числе программного обеспечения (ПО) для научных целей, необходимо обеспечение возможности апробирования ПО с минимизацией затрат времени и ресурсов на его развертывание. Поскольку среди вариантов, где можно развернуть ПО для оценки возможности его дальнейшего использования в рамках данных конкретных научных задач, зачастую имеется лишь единственный - персональный компьютер, то решение обычно затягивается. Особенно, если для этого необходима установка другой операционной системы (ОС). Кроме того, современные пакеты научного ПО могут требовать для работы кластеры высокопроизводительных вычислений. Апробирование нового научного ПО на специализированных вычислительных кластерах может представляться еще большими проблемами, как на бюрократическом уровне, связанными со всевозможными согласованиями, так и на техническом - проблемы совместимости совместимости с библиотеками и программами, уже работающими на кластере. Таким образом, возникает задача обеспечения возможности запуска ПО для целей апробирования и проведения научных расчетов на выделенных серверах под управлением совместимой с ПО операционной системой. По мнению авторов, решением поставленной задачи является использование облачных инфраструктур, то есть вычислительных ресурсов по заказу. На сегодняшний день существует уже достаточно большое количество общедоступных провайдеров облачных ресурсов, например: Amazon EC2, GoGrid, Rackspace. Облачные провайдеры предоставляют возможность загрузки виртуальных машин в их центрах данных и получения пользователями удаленного доступа с почасовой оплатой. Стоимость работы достаточна низкая, и на данный момент для не самых мощных компьютеров составляет порядка трех рублей в час. Пользователь, загружая необходимую для его экспериментов ОС, получает к ней удаленный доступ, например посредством протоколов RDP (Windows) или SSH (*nix). Далее пользователь работает с системой так, как будто бы это был его настольный компьютер. А именно, настраивает ОС и устанавливает необходимое ПО. Преимущество использования такого виртуального сервера в рамках задачи очевидно - устанавливается необходимая ОС (провайдеры предлагают на выбор большое число различных версий ОС, возможна загрузка собственного образа), система находится под полным контролем пользователя, не нужно никаких согласований и т.д. В том случае, если установка ПО вручную не является тривиальной задачей, тем более в случае распределенных приложений, и занимает значительное время, возникает вопрос автоматизации. Во-первых, это позволяет добиться безошибочного повторного развертывания ПО; во-вторых, позволяет сэкономить время и деньги. В случае несложных приложений, требующих за-

мены нескольких конфигурационных файлов в ОС, установки переменных окружения и установки приложения из пакетного репозитория, эта задача может быть достаточно легко решена путем написания bash скрипта в ОС *nix. Однако, когда список приложений для установки увеличивается, возникает связь между приложениями, тем более в случае установки распределенных приложений, быстро возрастает сложность кода скриптов. Отладка и поддержка таких скриптов отнимает значительное время, таким образом bash может быть использован в качестве инструмента настройки и установки приложений лишь для простых задач. В литературе [2] достаточно широко исследована работа по автоматизированному системному администрированию, включая анализ эквивалентности управления серверами и машины Тьюринга. На основе теоретических изысканий формируются фактические требования, предъявляемые к системе управления инфраструктурой серверов. В разное время были разработаны специализированные фреймворки, предназначенные именно для целей автоматизированного системного администрирования и установки приложений. Авторами рассмотрен достаточно новый фреймворк автоматизации развертывания приложений Chef Opscode [1] и показано, что фреймворк Chef удовлетворяет теоретическим требованиям, предъявляемым к инструменту автоматизированного управления серверами. Авторами также была разработана специальная утилита, которая совместно с Chef позволяет свести задачу развертывания распределенного приложения в облачной инфраструктуре к запуску одной команды с минимумом аргументов. Примеры и утилита находятся в свободном доступе [3], где рассмотрена автоматизация в *nix средах. Существует реализация Chef и для Windows, но она позволяет автоматизировать лишь ограниченное число задач.

2. Управление инфраструктурой серверов

Виртуальная машина в облачной инфраструктуре с точки зрения пользовательских программ и самого пользователя представляет собой ту же операционную систему, требующую настройки, администрирования, то есть управления, как и в случае физического сервера. Теоретические основы управления серверами базируются на практическом опыте [2]: *«Основываясь на проведенных нами исследованиях, не существует никакого инструмента написанного на каком-либо языке, способного предсказуемо администрировать инфраструктуру серверов без обеспечения детерминированного, повторяемого порядка изменений на каждом вычислительном узле. Среда времени выполнения для любого инструмента всегда работает в контексте целевой операционной системы; изменения могут повлиять на поведение самого инструмента, создавая круговые зависимости. Поведение таких изменений может быть сложно предсказуемо, следовательно, необходимо тестирование для проверки изменившихся вычислительных узлов. Как только изменения прошли тестирование, они должны быть реплицированы на рабочих вычислительных узлах в том же самом порядке, в котором они тестировались, из-за этих же круговых зависимостей.»*. Порядок изменений важен по следующим основным причинам:
– «круговая зависимость» — инструмент администрирования системы может выполнять код, который модифицирует сам этот инструмент и конечные пользователи могут не знать о таких зависимостях;
– непредсказуемость поведения в результате беспорядочных изменений на диске — сложнее проверить, чем предсказать поведение в результате детерминированного порядка.

2.1. Методы управления

Управление серверами (конфигурирование и администрирование) можно разделить на три категории: расходимость, конвергенция, и конгруэнция.
Расходимость — результат неверного управления. Расходимость характеризуется тем, насколько далеко отстоит существующая конфигурация сервера от требуемой или подразумеваемой. Симптомы расходимости включают в себя: непредсказуемое поведение сервера;

незапланированные нарушения работы; неожиданные проблемы с установкой пакетов и патчей; незакрытые проблемы безопасности; значительные временные затраты на устранение нарушений работы; высокая стоимость поиска источника нарушения работы и обслуживания. Причиной расходимости могут быть изменения на сервере «вручную» и другие беспорядочные изменения.

С **конвергенцией** большинство системных администраторов знакомятся, как только сталкиваются с расходимостью и начинают ручную синхронизацию критических файлов Затем они стараются автоматизировать этот процесс, котрорый и называется конвергенцией. В процессе конвергенции, вычислительный узел приближается к требуемому, предполагаемому поведению. Поскольку процессу конвергенции подлежит лишь некоторое подмножество файлов, а не система целиком, то все конвергентные инфраструктуры в некоторой степени содержат расходимость. Таким образом, основной проблемой администрирования инфраструктуры с изначальной расходимостью является отсутствие информации о том, когда сходимость, или конвергенция, будет завершена. В отсутствии такой информации, процесс никогда не будет завершен. Процедура развертывания программного обеспечения, уже пройденная успешно на одном из серверов, может провалиться на другом сервере из-за изначального расхождения в конфигурации. Установки ПО в этом случае превращается в итеративный процесс внесения изменений в инструмент конфигурирования и повторный запуск. Более того, необходимо убедиться в том, что внесенные изменения корректно работают на других серверах, где уже установлено это ПО. Не смотря на все вышеперечисленные недостатки, конвергенция обеспечивает большую надежность и предсказуемость, чем в инфраструктуре с расходимостью.

Конгруэнтность — управление серверами таким образом, чтобы их работа находилась в полном соответствии с предполагаемым, требуемым поведением. Конгруэнтность определяется состоянием битов на диске, а не поведением сервера, потому что состояние диска может быть однозначно определено, а поведение — нет [2]. По определению, расхождение от изначального состояния диска в конгруэнтной среде является симптомом сбоя кода, административных процедур или безопасности. В любом из этих трех случаев может не быть полной уверенности, какие области диска повреждены или изменены. В таком случае легче все три случая рассматривать как уязвимость в системе безопасности. Необходимо устранить причину сбоя, и переустановить вычислительный узел в конгруэнтном режиме. Этот режим позволяет восстановить сервер быстрее, чем каким-либо другим способом.

По мнению авторов, конгруэнтным инструментом также является инструмент, который хранит все инструкции по изменению системы, начиная (и включая) изначальную инсталляцию с образа. В случае выхода из строя вычислительного узла, все сделанные изменения на исходной системе должны быть повторены в таком же порядке на вновь созданной системе, используя те же инструкции. То есть, при соответствующем использовании инструментов конвергенции, а именно применении их на изначально идентичных системах, будет получена конгруэнтная система. Кроме того, все изменения в системе должны производиться с помощью инструмента и применяться одинаковым образом на вычислительных узлах, в противном случае в систему будет внесено расхождение с предполагаемым состоянием.

Для обеспечения конгруэнтного режима, инструмент управления сервером должен в том числе удовлетворять следующим требованиям.

1. Останавливать свою работу с отображением ошибки, в случае, если очередная инструкция не завершилась успехом. В качестве примера, покажем, что язык командной оболочки bash не обеспечивает такого поведения. Если инструкция

```
cd /storage/sources/
rm␣-fr
```

будет выполнена, например, с недостаточными привилегиями для смены директории, то мы получим следующий результат:

```
cd /storage/sources/
cd:␣permission␣denied:␣/storage/sources/
```

`rm -fr .`
Сообщение об ошибке может быть различным на разных ОС, однако результат одинаков: вместо удаления содержимого в `/storage/sources/`, будет удалено содержимое текущей директории.
2. Вносить изменения в одном и том же порядке.
3. Для большинства стандартных действий, таких как создание директории, пользователя, работа с сервисами и т.д., должны существовать специализированные ресурсы, облегчающие работу, и абстрагирующие от конкретной ОС.
4. Специализированные ресурсы должны быть идемпотентны [4].

С точки зрения программного кода, научное ПО ничем не отличается от какого-либо другого ПО, поэтому можно считать что автоматизированные системы установки и управления серверами одинаково применимы для любого вида программного обеспечения.

2.2. Установка и настройка приложений

Необходимые действия при установке приложения в ОС семейства *nix представляется более наглядным рассмотреть на конкретном примере, из которого выделить общие задачи. В качестве примера, рассмотрим Hadoop [5], фреймворк для запуска параллельных приложений с использованием технологии map-reduce. Кроме обеспечения параллельной работы приложений, Hadoop включает в себя распределенную файловую систему HDFS, высокопроизводительный координационный сервис для распределенных приложений Zookeeper, распределенную базу данных HBase и другое. Все перечисленные сервисы в общем случае могут устанавливаться на различные серверы, и некоторые из них имеют серверную и клиентские части. Сервисы не являются независимыми и взаимодействуют друг с другом, для чего должны быть созданы специальные конфигурационные файлы. В этих файлах указываются доменные имена узлов и порты, где запущены сервисы Hadoop, а также дополнительные настройки.

Hadoop написан на языке Java, и требует установленную Oracle JRE на сервере для своей работы. В простейшей конфигурации, в целях изучения основ Hadoop, на ОС не требуется никаких других пакетов, кроме JRE. Для использования Hadoop в режиме «production», система может потребовать специальных настроек (например, максимально допустимое число открытых соединений), настройка сетевого файрвола и т.д. В том случае, если требуется создание высокодоступного (Highly Available, HA) кластера Hadoop, необходима установка дополнительных пакетов в систему и их настройка. Если пакетов для данного дистрибутива нет, то потребуется сборка из исходного кода, что, в свою очередь, требует установленных пакетов компилятора и сборщика. Система установки должна работать без сбоев, то есть установка одного и того же приложения на один и тот же дистрибутив ОС должна заканчиваться успешно. В некоторых случаях необходимо обеспечить возможность работы приложения на другой ОС. В случае автоматизации, это означает обеспечение максимальной кроссплатформенности системы установки. При переходе на другую ОС, затраты на адаптацию системы установки должны быть минимальными, а код должен оставаться единым для установки на любую систему, следуя принципу DRY [6].

Обобщая, можно выделить следующие функциональные требования для инструмента управления серверами:
– повторяемая и предсказуемая работа на одной и той же платформе;
– работоспособность на всех необходимых платформах семейства *nix;
– установка пакетов в систему;
– внесение изменений в конфигурационные файлы системы;
– возможность отслеживания, на каких узлах установлен конкретный сервис.

С точки зрения программиста, для написания читаемого, кроссплатформенного кода необходимо обеспечение следующих требований:
– использование шаблонов для создания конфигурационных файлов;

– возможность доступа к системной информации (IP адреса, архитектура ОС и т.д.) независимым от ОС способом;
– возможность написания пользовательских функций;
– минимальное количество действий вручную для развертывания приложений на серверах;
– параллельная работа на различных серверах.

Для установки приложений в облачной инфраструктуре требуется их поддержка инструментом виртуализированных облачных сред. Кроме того, требование на минимальное количество действий вручную накладывает дополнительные функциональные ограничения на инструмент в плане работы с облачными инфраструктурами, а именно:
– автоматический запуск виртуальных машин облачного провайдера в соответствии с требованиями устанавливаемого ПО;
– ожидание готовности виртуальных машин;
– передача необходимых файлов и данных на виртуальные машины;
– активация установщика на виртуальных машинах (установка, конфигурация и запуск);
– параллельная работа с несколькими виртуальными машинами.

3. Фреймворк Chef

Выше были перечислены основные требования, предъявляемые к инструменту управления инфраструктурой серверов. Open Source продукт Opscode Chef удовлетворяет большинству из этих требований и является специализированным фреймворком по настройке и управлению серверами. Скрипты установки и настройки создаются на Ruby DSL и называются рецептами Chef. Абстрагируясь от ОС, в рецептах Chef предоставляются специальные **ресурсы**, позволяющие работать с системой. Например, ресурс `package` предназначен для установки пакетов в систему, `service` — для управления сервисами. Чтобы установить пакет «tar» в систему, в рецепте достаточно написать `package “tar"`; для запуска сервиса tomcat6:

```
service "tomcat6" do
  action :start
end
```

При выполнении кода запуска сервиса tomcat6, Chef сначала проверяет, не запущен ли уже сервис. Если сервис остановлен, Chef выполнит команду по его запуску, причем в зависимости от ОС команда может быть различной. Таким образом, фреймворк дает возможность программисту создавать скрипты автоматизированной установки, не привязываясь к конкретному дистрибутиву ОС. В некоторых случаях, однако, возникает необходимость выполнить различные действия в зависимости от ОС. Например, пакет Web сервера Apache в ОС RedHat называется httpd, а в Ubuntu - apache2. В этом случае можно использовать условное ветвление в зависимости от платформы:

```
package "apache2" do
  case node[:platform]
  when "centos","redhat","fedora","suse"
    package_name "httpd"
  when "debian","ubuntu"
    package_name "apache2"
  end
  action :install
end
```

Ресурсы в Chef обеспечивают идемпотентность. Однако, специальные проверки на полную идемпотентность отсутствуют. Например, следующий ресурс в Chef уже не будет идемпотентным:

```
bash "iptables␣startup" do
user "root"
  code <<-EOH
  echo ".␣/etc/iptables.sh" >> /etc/rc.d/rc.local
␣␣EOH
```

```
end
```
Таким образом, разработчик рецептов должен самостоятельно отслеживать и не допускать подобные ситуации.

На данный момент в Chef были доступны 24 ресурса, включая `package` и `service`, облегчающие конфигурирование системы и установку новых приложений. Так, Chef имеет ресурсы следующего предназначения:
– **HTTP Request**, отправка HTTP запросов;
– **Mount**, управление монтируемыми разделами;
– **Template**, создание файлов по шаблону.

Между ресурсами возможна передача сигналов. Это удобно, например, в следующем случае. Предположим, что в процессе выполнения рецепта изменяется конфигурационный файл Web-сервера. Для того, чтобы изменения вступили в силу, необходим перезапуск сервиса, однако нежелательно перезапускать сервис если конфигурационный файл не менялся. В Chef это можно реализовать следующим образом:

```
template "/etc/www/configures-apache.conf" do
  notifies :restart, "service[apache]"
end

service "apache"
```
Специальная инструкция `notifies` отправляет ресурсу `service "apache"` сигнал к действию, подлежащему выполнению.

Кроме рецептов, Chef оперирует также и другими сущностями. Все необходимые файлы для установки и конфигурации одного конкретного приложения, например Web сервера Apache, помещаются в единую структуру каталогов и такая структура называется Cookbook ("поваренная книга"). Типичное содержание каталога на примере Cookbook для установки Apache Tomcat имеет следующий вид:

```
.
|-- README.rdoc
|-- attributes
|   `-- default.rb
|-- definitions
|   `-- tomcat_app.rb
|-- files
|   |-- centos
|   |   `-- rightscale.repo
|   `-- default
|       |-- JVM-MANAGEMENT-MIB.mib
|       |-- dtomcat6
|       |-- logging.properties
|       `-- tomcat6
|-- libraries
|   |-- tomcat.rb
|   `-- tomcat_manager.rb
|-- metadata.json
|-- metadata.rb
|-- recipes
|   `-- default.rb
`-- templates
    `-- default
        |-- manager.xml.erb
        |-- tomcat-users.xml.erb
        `-- tomcat6.conf.erb
```
Имя корневого каталога, содержащей все эти файлы, и является именем "поваренной книги"для приложения. Рецепты хранятся в подкаталоге recipes. Кроме рецептов, Chef оперирует такими понятиями, как атрибуты (attribures/), определения (definitions/), файлы (files/), библиотеки (libraries/), шаблоны (templates/). **Атрибуты** в Chef можно рассматривать как некие глобальные переменные, причем во всей инфраструктуре, находящейся под управлением Chef сервера. Атрибуты содержат такие данные, как IP адреса, информацию об ОС, установленных пакетах и т.д. Во время запуска клиента Chef атрибуты формируют-

ся на клиенте и отправляются на сервер, где затем индексируются и становятся доступны для поиска. Это позволяет во время выполнения рецепта получить информацию о других серверах под управлением Chef, например, на каких еще виртуальных машинах выполняется или будет выполнен этот же рецепт. **Определение** в Chef позволяют создавать новый ресурс, содержащий в себе объединение нескольких других ресурсов. **Файлы** — файлы, которые требуется записать в систему без изменений. **Библиотеки** предоставляют возможность написания произвольного Ruby кода, расширяя возможности Chef. Например, это может быть библиотека для сбора данных с LDAP сервера для дальнейшего использования полученных данных в рецепте. **Шаблоны** — файлы со специальными переменными. Ресурс `template` передает в эти файлы значения подставляемых переменных, и записывает файлы в заданные места в ОС. Специальный файл `metadata.rb` содержит информацию о Cookbook в простом формате; `metadata.json` автоматически генерируется из `metadata.rb`, ручная модификация файла не требуется.

4. Утилита Clorun

Авторами была разработана утилита Clorun, основное предназначение которой состоит в уменьшении количества действий, необходимых для развертывания ПО, в том числе распределенного, в облачной инфраструктуре. На данный момент утилита работает только с облачным провайдером Amazon EC2, однако она может быть легко дополнена модулем сопряжения с другими облачными провайдерами, например Rackspace или GoGrid. Создание подобной утилиты показало принципиальную возможность сведения ручной работы к минимуму — программа автоматизирует все ручные действия, связанные с выбором и запуском виртуальных машин в облачной инфраструктуре, а также установку, настройку и запуск Chef клиента. Кроме работы в интеграции с Chef фреймворком, программа может быть легко адаптирована как для запуска произвольных bash скриптов, так и для работы с другими фреймворками. Требуемые изменения не должны затронуть больше, чем несколько строк конфигурации. Следовательно, утилиту можно считать универсальной для запуска виртуальных машин на Amazon EC2 и произвольного кода на этих машинах. Кроме запуска кластера виртуальных машин, утилита позволяет подобным же образом останавливать работу запущенных машин по окончании работ.

4.1. Принцип работы Clorun

Утилита представляет собой несколько файлов, написанных на Ruby, и конфигурационные файлы. Последние представляют из себя несколько файлов формата JSON (по количеству запускаемых машин), которые содержат информацию о запускаемых рецептах, а также некоторые дополнительные атрибуты, и также файл ec2.config, содержащий данные для запуска виртуальной машины на Amazon EC2 (тип машины, зона запуска, идентификационный номер образа). Принцип работы утилиты следующий:
– запрос виртуальных машин по количеству найденных JSON файлов в многопоточном режиме;
– запрос присвоенных IP адресов виртуальным машинам;
– создание JSON файлов конфигурации из уже существующих, включая в новые информацию о том, на каких IP адресах какие рецепты будут выполняться;
– ожидание перехода машин в статус "Running"и ответа по SSH протоколу;
– загрузка JSON файлов и заархивированных Chef рецептов по протоколу SFTP на каждую из машин в многопоточном режиме;
– выполнение на виртуальных машинах команды wget для загрузки рецептов, если они не были переданы по протоколу SFTP;
– распаковка рецептов на машинах;
– выполнение команды chef-solo для запуска Chef в режиме "только клиент". Пример кон-

фигурационного файла ec2.config имеет следующий вид:

```
--
:ami: ami-f95cba90
:instance_type: m1.small
:security_groups: mscherbakov
:keypair: mscherbakov
:key: /home/mike/keys/amazon/mscherbakov.pem
:availability_zone: us-east-1c
:EC2_URL:
:chef\textit{cooks}url: http://myserver/chef.tar.gz
```

Файл записан в формате YAML. Все параметры специфичны для облачного провайдера Amazon EC2, кроме `chef_cooks_url`. Этот параметр является необязательным. Если он присутствует, то должен содержать URL, по которому виртуальная машина сможет получить заархивированные рецепты Chef, выполнив команду wget.

Пример файла конфигурации машины формата JSON:

```
{
  "cloud": {
    "info": "Amazon EC2"
  },
  "recipes": [ "java6", "hadoop::namenode" ]
}
```

Атрибут `cloud` является необязательным, и лишь передает дополнительную информацию, к которой легко получить доступ во время выполнения рецепта. Атрибут `recipes` содержит массив рецептов для выполнения и является обязательным. Соблюдая формат JSON, возможно передать произвольные конфигурационные параметры, которые будут легко доступны из кода. Например, строка "Amazon EC2"может быть получена в коде рецепта следующим выражением:

```
node[:cloud][:info].
```

После получения информации об IP адресах всех запускаемых виртуальных машин, утилита Clorun перечитывает каждый из JSON файлов конфигурации (если их больше одного), дополняет их IP адресом, а также информацией о списке рецептов для запуска на каждой другой машине и ее IP адресом. Пример одного из полученных таким образом файлов имеет следующий вид:

```
{
  "cloud": {
    "info": "Amazon EC2",
    "name": "cassandra",
    "recipes": [ "cassandra": [ "204.236.196.147" ],
        "java6": [ "204.236.196.145", "204.236.196.147" ],
        "hadoop::namenode": [ "204.236.196.145" ] ],
    "roles": [],
    "ip": "204.236.196.145",
    "id": "i-c87175a0"
  },
  "recipes": [ "java6", "hadoop::namenode" ]
}
```

Модифицированные файлы сохраняются в директории `configs/<name_of_env>/`, где `<name_of_env>` — указанное параметром `-n` при запуске `clorun` имя кластера. Из полученных атрибутов достаточно легко получить IP адреса всех узлов, на которых будет выполнен конкретный рецепт. В коде рецепта это может выглядеть следующим образом:

```
cass_ips = node[:cloud][:recipes]["cassandra"] & \
    search(:node, "recipe:cassandra").map { |cfg| cfg["ipaddress"] }
```

В этом примере будет также произведена попытка поиска других узлов с рецептом `cassandra`, используя функцию search сервера Chef. Результатом будет объединение массивов IP адресов, полученных через переданные атрибуты утилитой Clorun, и найденными IP адресами на Chef сервере. В нашем случае Chef используется в режиме «соло», то есть без сервера, поэтому результат будет состоять только из IP адресов, переданных в виде атрибутов. Удобство именно такой записи в коде состоит в том, что позволяет без каких либо модификаций

работать рецепту в модели запуска Chef с сервером. Дополнительный атрибут «id», записанный в JSON файл, содержит идентификационный номер запущенной виртуальной машины, выданный облачным провайдером. Утилита использует этот id для остановки виртуальных машин.

После создания модифицированных конфигурационных файлов, утилита ожидает ответа серверов по SSH протоколу. Как только соединение установлено, в многопоточном режиме происходит передача файла конфигурации каждого из серверов из директории textttconfigs/<name_of_env>/, файла `solo.rb` и архива `chef.tar.gz` с рецептами Chef. Архив не передается, если `ec2.config` содержит атрибут `"chef_cooks_url"`. Файл `solo.rb` содержит конфигурационные параметры для Chef:

```
file\textit{cache}path "/root/chef"
cookbook_path "/root/chef/cookbooks"
role_path "/root/chef/roles"
```

После загрузки файлов, на сервере запускается Chef на выполнение рецептов. Утилита Clorun отображает на экране монитора стандартный вывод запуска Chef, и завершает свою работу только после окончания работы Chef на всех серверах.

Установка Hadoop, Elmer [8] или какого-либо другого научного программного обеспечения в разработанной системе развертывания сводится к написанию рецептов Chef и созданию конфигурационных файлов для работы утилиты Clorun. Разработанные авторами рецепты могут быть загружены с сайта [3], там же можно найти конфигурационные файлы утилиты Clorun для развертывания упомянутых приложений.

5. Заключение

В работе была представлена практическая реализация алгоритма автоматизированного процесса установки распределенного ПО в инфраструктуру облачного провайдера Amazon EC2, выполненная на базе Open Source фреймворка Chef и созданной авторами утилиты Clorun. Проведенные исследования показали возможность создания систем установки и управления инфраструктурой серверов, действующих без вмешательства человека в полностью автоматическом режиме, позволяя разворачивать несколько виртульных машин одновременно в облачной инфраструктуре. В случае сбоя по каким-либо причинам, не связанным с ошибкой в работе системы, самый простой способ разрешения — удалить виртуальный кластер, и попытаться создать его снова. Время создания кластера может варьироваться: около 5 минут для простой конфигурации серверов, около 10–15 минут для достаточно сложной, и дольше, например в случае пересылки большого объема данных для первоначального запуска. В работе использован Chef в качестве основного инструмента управления серверами, однако могут быть использованы и другие подобные фреймворки, удовлетворяющие теоретическим требованиям, например Puppet [7]. Для Puppet необходима дополнительная машина с установленным сервером Puppet, доступная для запускаемых клиентов. В дальнейшем планируется более детально исследовать соответствия теоретическим требованиям описанной системы установки на базе Chef, возможные сбои работы с облачной инфраструктурой и методы борьбы с ними. Кроме того, требуется дополнительное исследование на тему установки распределенных приложений со сложными связями, например, когда в процессе установки на одной из виртуальных машин потребуется, чтобы на другой уже был выполнен один из рецептов. Могут быть и более сложные ситуации, требующие распределенной блокировки.

Литература

1. Chef. Infrastructure Automation for the Masses: URL: http://www.opscode.com/chef (дата обращения: 14.12.2010).

2. Steve Traugott, Lance Brown. Why Order Matters: Turing Equivalence in Automated Systems Administration: URL: http://www.infrastructures.org/papers/turing/turing.html (дата обращения: 14.12.2010).

3. Scherbakov M. Public repositories of Mike Scherbakov: URL: http://github.com/mihgen (дата обращения: 14.12.2010).

4. Idempotence: URL: http://en.wikipedia.org/wiki/Idempotency (дата обращения: 14.12.2010).

5. Apache Hadoop: URL: http://wiki.apache.org/hadoop/ (дата обращения: 14.12.2010).

6. Эндю Хант, Дэвид Томас. Программист-прагматик. Москва: Лори, 2004. С. 22.

7. Docs: Introduction to Puppet: URL: http://docs.puppetlabs.com/guides/introduction.html (дата обращения: 14.12.2010).

8. Open Source Finite Element Software for Multiphysical Problems: URL: http://www.csc.fi/english/pages/elmer (дата обращения: 14.12.2010).