Тел.: (495) 363-37-53; (4012) 51-05-64; e-mail: info@1c-bitrix.ru, http://www.1c-bitrix.ru

1С-Битрикс: Управление сайтом

Руководство по настройке и администрированию

«1С-Битрикс: Веб-кластер»

Содержание

1. Кластеризация

Понятие "кластер" применительно к компьютерным вычислительным системам впервые стала применять компания DEC в 80-х годах прошлого века. С тех пор - по определению - "кластер" представляет собой группу компьютеров, серверов, объединенных в единый узел. При этом с точки зрения приложения все ресурсы кластера в общем случае предоставляются единой системой.

На практике кластерные системы используют для решения двух больших классов задач:

1. **Обеспечение высокой (в идеале - непрерывной) доступности сервиса** (так называемые HA - High Availability или Failover кластеры).
2. **Масштабирование доступных ресурсов в условиях возрастающей нагрузки** (HP - High Performance кластеры).

2. Что такое "1С-Битрикс: Веб-кластер"

Решение комбинации обеих описанных выше задач - обеспечение масштабируемой производительности и непрерывной доступности (отказоустойчивости) сервиса крайне важно для владельцев веб-проектов: сайт любого бизнеса в интернете должен быть всегда доступен для клиентов, и его страницы должны загружаться моментально - независимо от каких-либо внешних факторов (сбои и аварии на оборудовании, возросшая пиковая посещаемость и т.д.)

В противном случае посетители сайта, не дождавшись загрузки страниц с каталогом или с конкретным товаром, не сумев сделать заказ из-за проблем на сервере (если речь идет, например, об интернет-магазине), уйдут к конкурентам и, скорее всего, больше не вернутся. А поисковые системы из-за низкой скорости загрузки страниц ниже отранжируют их в результатах поиска (например, Google уже поступает именно так).

Крупные компании Enterprise сегмента прекрасно осознают не только практические задачи, решаемые "быстрым" сайтом, но и репутационный фактор: сайт компании - это ее "лицо" в интернете, и он просто обязан быть высокопроизводительным.

Сейчас подобные задачи успешно решаются с использованием, например, Oracle RAC (Real Application Cluster) или кластера Microsoft SQL Server в качестве базы данных.

Это, безусловно, хорошие и надежные (при соответствующей настройке и администрировании) решения. И платформа «1С-Битрикс» поддерживает работу с базами Oracle и MS SQL в старших редакциях продуктов. Однако их использование вызывает ряд сложностей для конечного пользователя:

- кластерные конфигурации Oracle или MS SQL решают только задачу резервирования и масштабирования базы данных, отдельно требуется решать подобную задачу для веб-сервера;
- высокая стоимость коммерческих продуктов;
- сложное решение – требуются высококвалифицированные системные администраторы с обязательным опытом работы с указанными системами.

Разработанный для платформы «1С-Битрикс» модуль «Веб-кластер» позволяет комплексно решить задачу масштабирования и резервирования всего веб-проекта (а не только базы данных), используя при этом свободное ПО (Apache, PHP, MySQL) и простые и понятные конфигурации системы, администрирование которой не потребует каких-либо узкоспециализированных навыков.

Отдельно отметим крайне важный момент: работа кластера поддерживается на уровне самой платформы "1С-Битрикс". Разработчикам, которые используют платформу для создания сайтов, не потребуется вносить какие-либо изменения в код существующих и новых проектов для того, чтобы обеспечить их масштабирование с помощью "Веб-кластера".

3. Цели и задачи, которые решает "1С-Битрикс: Веб-кластер"

Возможность одновременного масштабирования ресурсов и обеспечения отказоустойчивости позволяет разработчикам проектов на платформе "1С-Битрикс" решить с помощью модуля "Веб-кластер" несколько очень важных практических бизнес-задач.

3.1. Масштабирование веб-проекта, повышение производительности

Развитие веб-проекта неизбежно приводит к росту его популярности и посещаемости. И – как следствие – к повышению требований к выделяемым ресурсам (CPU, памяти, дисковому пространству).

До определенного порога задача решается простым увеличением ресурсов. С виртуального хостинга сайт переносится на VPS, затем – на свой выделенный сервер, затем – на более мощный сервер. Рано или поздно, каким бы мощным не оказался выделенный сервер, его ресурсов оказывается недостаточно. При этом каждая новая итерация переноса сайта на новые ресурсы сопровождается повторным конфигурированием каждой новой системы и временной недоступностью сайта в период переезда.

"1С-Битрикс: Веб-кластер" дает возможность гибко в реальном времени масштабировать именно те ресурсы (база данных или веб-сервер), которые востребованы в настоящий момент, просто добавляя новые машины по мере необходимости.

Ключевые особенности технологии:

- каждая машина ("нода") в кластере может быть разной сущности: виртуальный хост, VPS, выделенный сервер, инстанс в "облаке" (например, Amazon EC2);
- ноды могут быть распределены любым удобным образом – вплоть до нахождения в разных датацентрах;
- добавление нод различной мощности в среднем дает пропорциональный прирост производительности.

Пример: если ваш сайт работает на выделенном сервере с одним процессором Intel Quad Core, 4 Гб RAM и в пиках посещаемости может обрабатывать до 40 запросов в секунду, то добавление второго такого же сервера в веб-кластер повышает производительность примерно на 90-95% и позволяет обрабатывать в пиках до 77 запросов в секунду.

Подобный подход позволяет последовательно по мере необходимости наращивать ресурсы для проекта.

3.2. Резервирование всех узлов системы, обеспечение отказоустойчивости и непрерывной доступности

Вторая важная задача – обеспечение отказоустойчивости системы и минимизация времени простоя в случае аварий и плановых работ.

Выход из строя оборудования или авария в датацентре может привести к многочасовой недоступности сайта. А это чревато многими неприятными последствиями:

- потерянные заказы (если, например, речь идет об интернет-магазине);
- впустую потраченные деньги на медийную и контекстную рекламу;
- потеря репутации;
- потеря позиций в выдаче поисковых систем в случае переиндексации поисковым роботом в момент недоступности сайта.

Кластеризация всех составляющих сайта (веб-сервер и база данных) с помощью "1С-Битрикс: Веб-кластер" позволяет минимизировать время простоя системы. В зависимости от того, как именно реализована балансировка нагрузки между нодами кластера, время недоступности сайта может составить 5-15 минут, а в отдельных случаях и вовсе может быть сведено к нулю.

Как это работает: при аварии на отдельных нодах вышедшие из строя узлы кластера помечаются неактивными (OFFLINE) - будь то база MySQL, веб-сервер или сервер memcached, а нагрузка на систему автоматически распределяется между оставшимися узлами.

В пиках нагрузки может пропорционально возрасти время отдачи страниц сайта посетителям, однако в целом его работоспособность не будет нарушена. В штатном же режиме работы при наличии достаточного резерва производительности клиенты вообще не заметят каких-либо изменений.

Впоследствии после устранения причин аварии можно вновь ввести в работу вышедшие из строя ноды, тем самым распределив нагрузку между большим числом серверов.

3.3. Real-time backup

Частным случаем резервирования системы с помощью веб-кластера может быть создание резервной копии всех данных в реальном времени (real-time backup).

О необходимости регулярного создания резервных копий данных известно всем. При этом важно не только создавать бэкап, но и иметь возможность максимально быстро восстановить данные из него в случае аварии.

По данным исследования Strategic Research Institute 93% бизнес-компаний, не сумевших в течение 10 дней после потери данных восстановить их из бэкапа, прекращают свою деятельность и терпят банкротство в течение года, а 50% компаний, оставшихся без данных на такой срок – сразу же.

Поэтому крайне важным оказывается еще один аспект создания резервных копий – частота их обновлений. Если бэкапы создаются не очень часто, например, раз в неделю (а для больших – по объему данных – проектов, это, к сожалению, не редкость), то в случае аварии есть риск потерять все изменения (код разработчиков, данные самого сайта, пользовательские данные) вплоть до семи дней.

"1С-Битрикс: Веб-кластер" позволяет подключить одну из нод в качестве бэкап-сервера. Это может быть не очень дорогой сервер или, например, VPS. В настройках веб-кластера можно отметить, что данная нода используется только для бэкапа, в этом случае нода не будет участвовать в распределении нагрузки от посещений сайта.

При этом данные сайта (и файлы, и база данных) будут дублироваться в реальном времени. В случае аварии будут восстановлены актуальные данные на момент сбоя. Для максимально быстрого восстановления обращения к сайту могут быть переключены непосредственно на резервный бэкап-сервер.

3.4. Снижение стоимости трафика, простая система CDN

Перед владельцами сайтов, генерирующих большое количество "тяжелого" контента (аудио, видео, фото) встает еще одна серьезная задача: с одной стороны, есть потребность минимизировать стоимость траффика проекта, с другой стороны – важно обеспечить высокую скорость доступа для всех посетителей независимо от того, как они распределены территориально.

Ноды веб-кластера могут находиться в разных датацентрах (например, три сервера – в США, в Европе и в России). Использование дополнительных инструментов (например, базы GeoIP) позволяет автоматически определять регион посетителя сайта и перенаправлять его запрос на соответствующую ноду кластера (подобный механизм, регулирующий загрузку контента в зависимости от географического положения клиента, называется CDN - Content Delivery Network).

4. Практическая реализация (на примере Amazon Web Services)

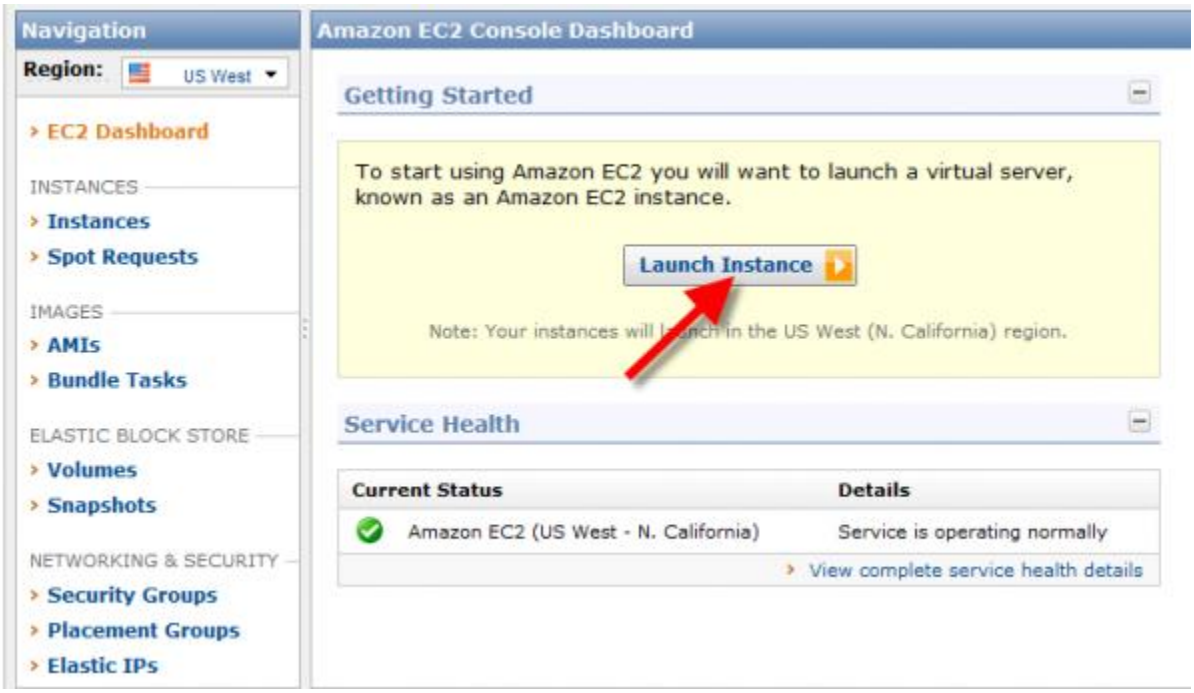
"1С-Битрикс: Веб-кластер" может быть развернут практически **на любой хостинговой платформе**: от выделенных физических серверов до виртуальных машин на "облачном" хостинге. Для достижения предоставляемой кластером высокой надежности и производительности требуется, как минимум, два сервера.

4.1. Создание виртуальных машин

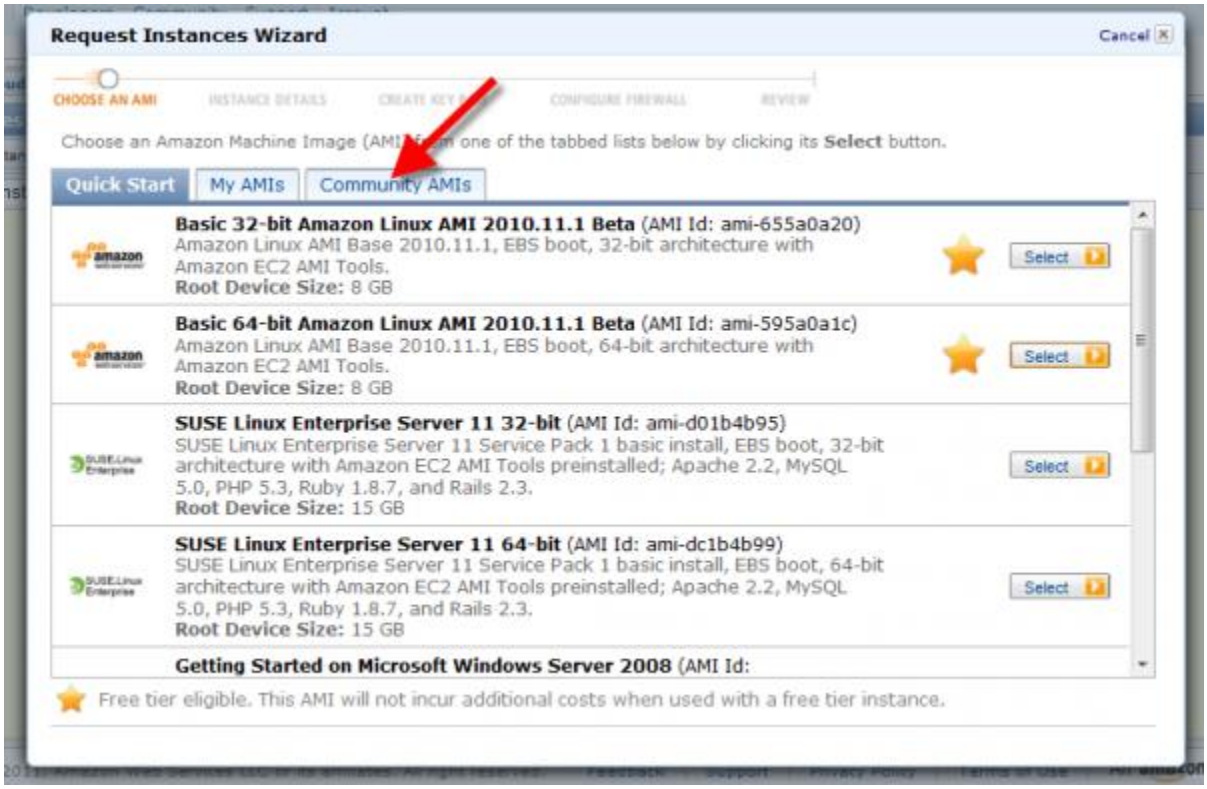
Мы опишем процесс создания и конфигурирования веб-кластера на примере настройки двух виртуальных машин на "облачном" хостинге Amazon Web Services. Он удобен тем, что мы можем в любое время практически моментально получить виртуальные серверы любой необходимой нам конфигурации.

На сайте создаем аккаунт:

Далее создаем две виртуальные машины. Выбираем пункт "Launch Instance":

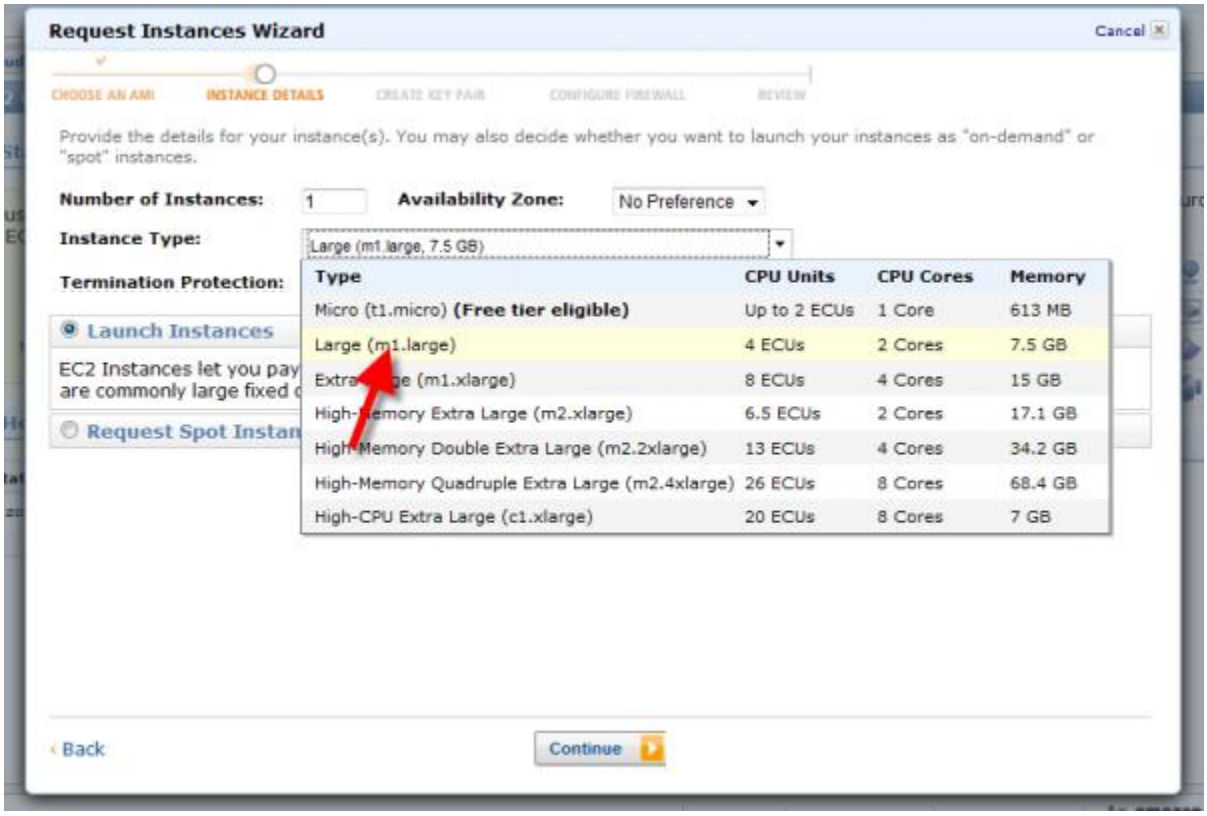


Переходим в раздел "Community AMIs". В ближайшее время будут выложены сконфигурированные для веб-кластера AMI-образы виртуальной машины «1С-Битрикс».

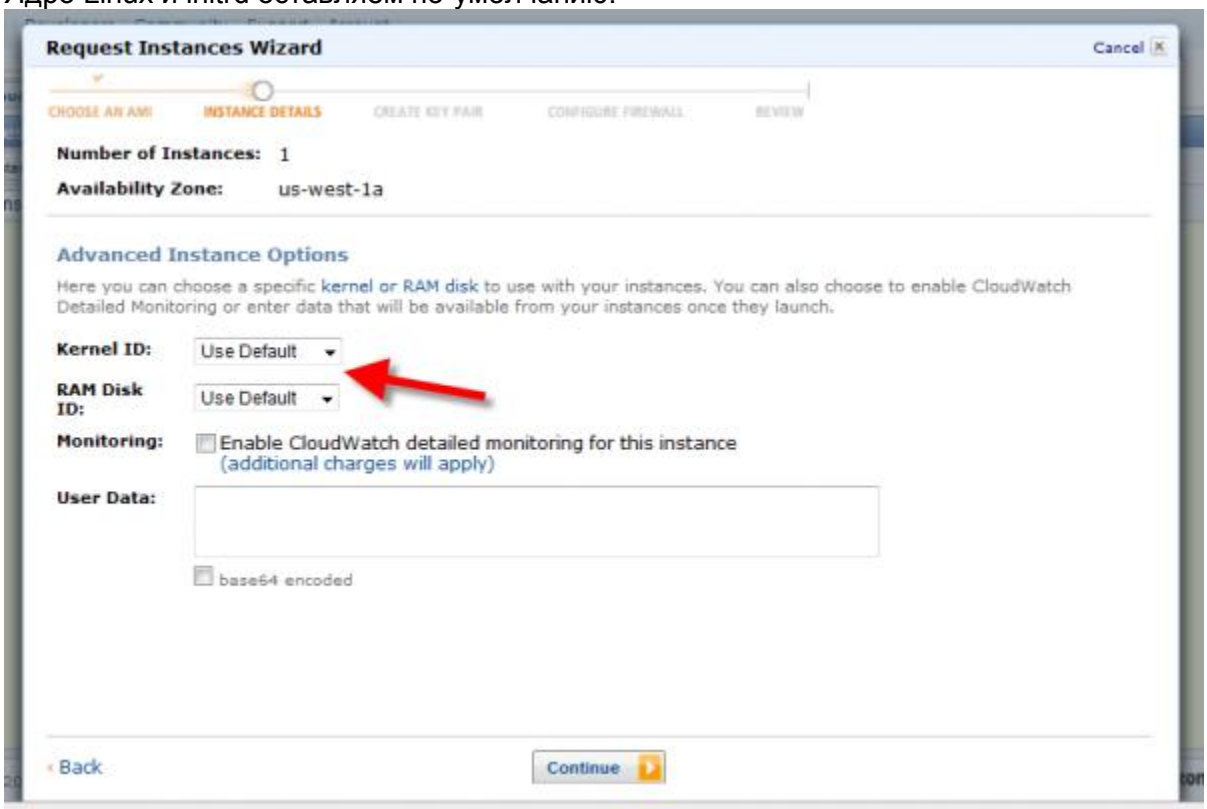


Выбираем образ ami-6d7f2f28, содержащий CentOS_5.4_x64. "1С-Битрикс: Веб-окружение - Linux" поддерживает также: Fedora 8-14 (i386), Red Hat Enterprise Linux 5 (i386, x86_64).

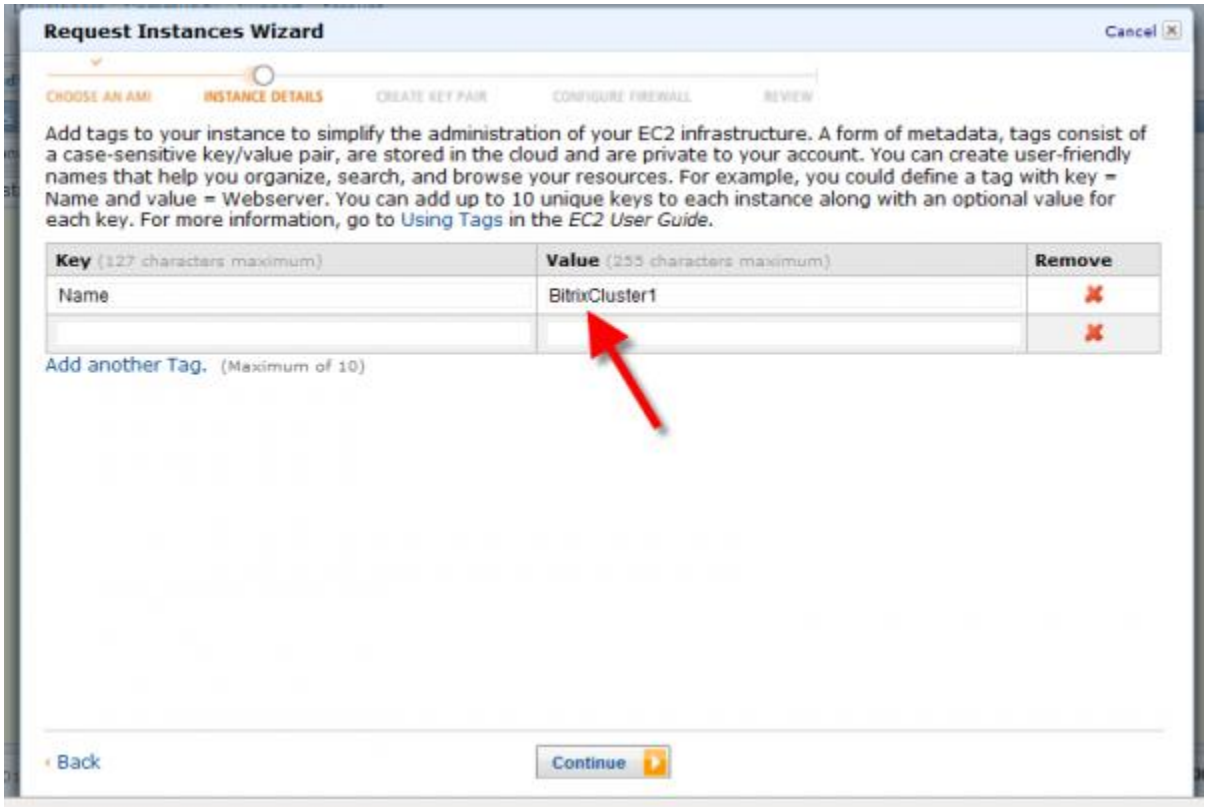
В зависимости от ожидаемой нагрузки выбираем аппаратную конфигурацию виртуальной машины. В данном случае выбираем конфигурацию "m1.large": 2 ядра примерно по 2 GHz и 7.5GB оперативной памяти:



Ядро Linux и initrd оставляем по-умолчанию:

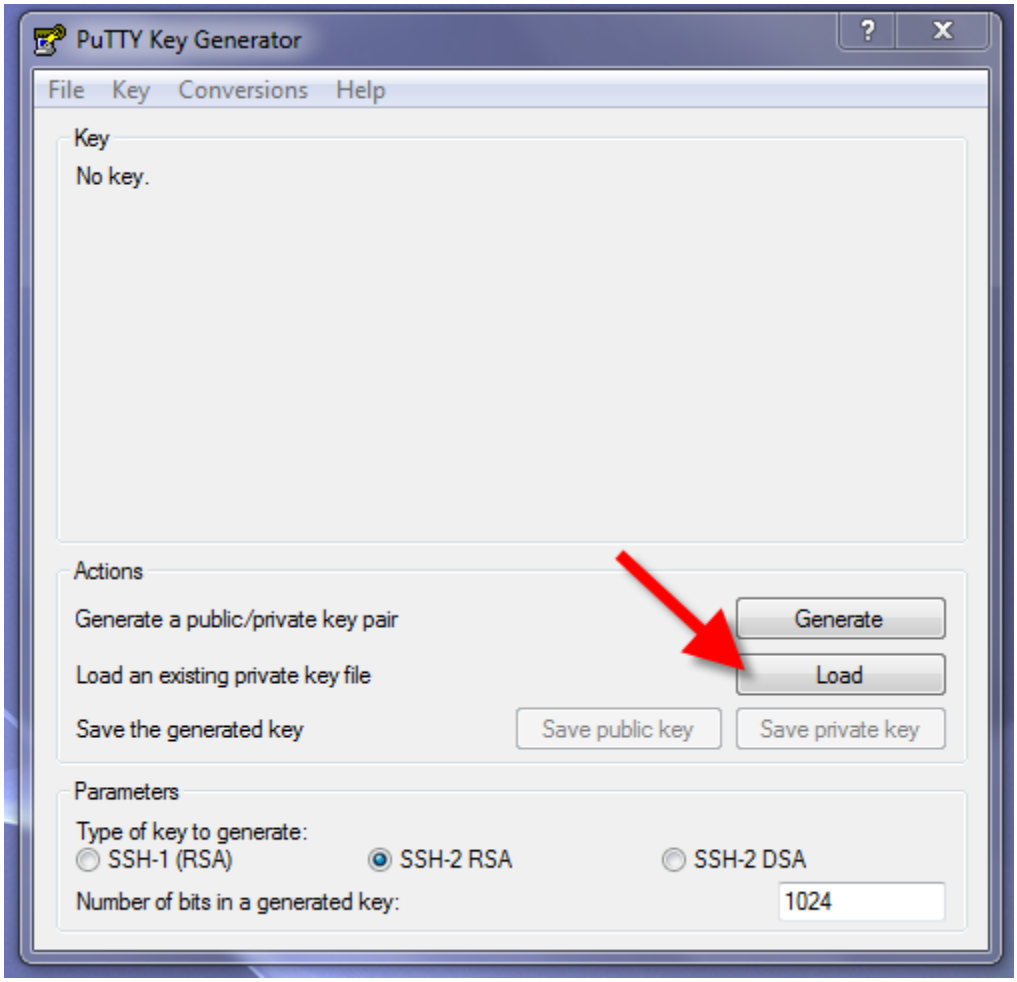


Даем машине понятное название, позволяющее отличить ее от других машин кластера:

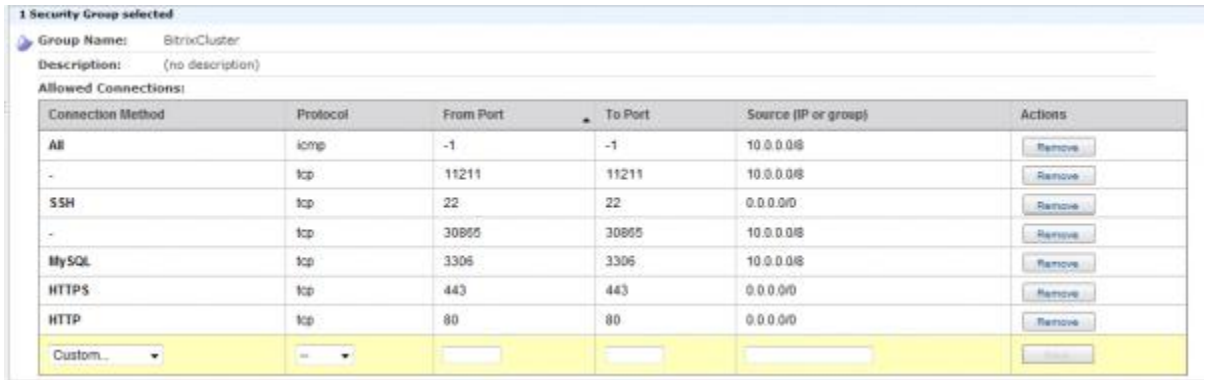


Создаем либо выбираем сделанную ранее пару ключей (для последующей авторизации по SSH):

Созданный закрытый ключ необходимо импортировать в PuTTYgen и пересохранить в формат закрытого ключа Putty (если Вы работаете на платформе Windows). Т.к. доступ на виртуальные машины осуществляется по умолчанию с использованием закрытого ключа, рекомендуется задать пароль для его защиты.

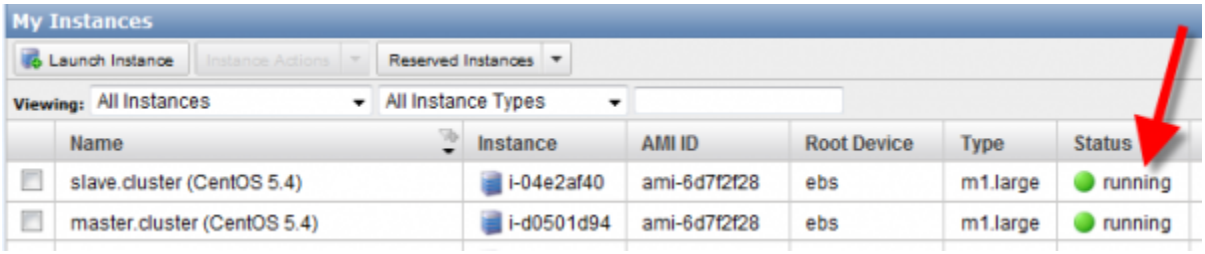


Создаем либо выбираем созданную ранее группу безопасности со следующими параметрами:



Порт 11211 используется memcached, 30855 - для csync2. После настройки кластера необходимо обязательно закрыть доступ извне кластера к портам memcached.

Аналогично запускаем вторую виртуальную машину:



Итог:
- Запущены две виртуальные машины требуемой производительности с установленной 64-х разрядной операционной системой CentOS 5.4.

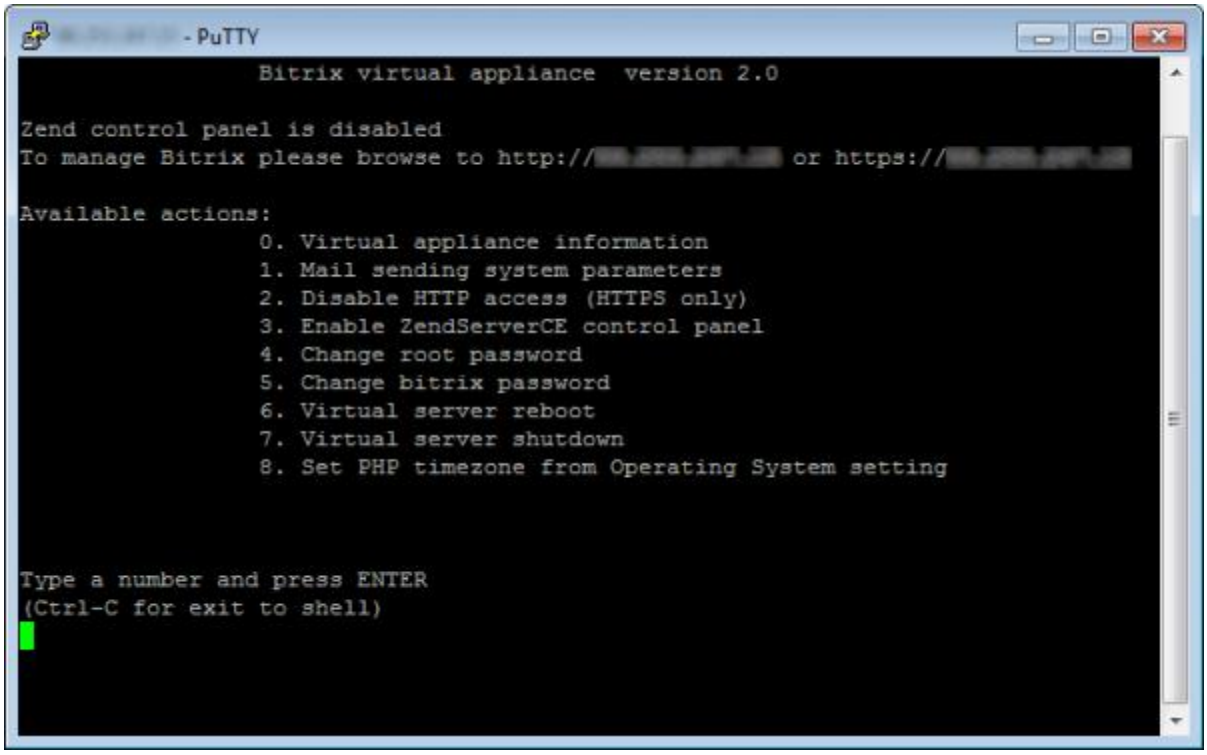
Для установки и настройки веб-платформы мы использовали бесплатный пакет "1С-Битрикс": Веб-окружение" 2.0 - Linux. Данный пакет предназначен для быстрой и простой установки и конфигурации всего стека ПО, используемого продуктами "1С-Битрикс" на Linux-платформах.

С помощью веб-окружения автоматически устанавливается и настраивается:

- mysql-server 5.*
- web-server (Apache 2.2.*)
- zend-server-ce-php-5.3
- mod-php-5.3-apache2-zend-server
- php-модуль geoip
- nginx
- memcached
- stunnel

Мы не ограничиваем использование той или иной веб-платформы. Можно использовать любой другой дистрибутив ОС, иные дистрибутивы и настройки серверного ПО. Однако мы рекомендуем использовать именно "1С-Битрикс: Веб-окружение":

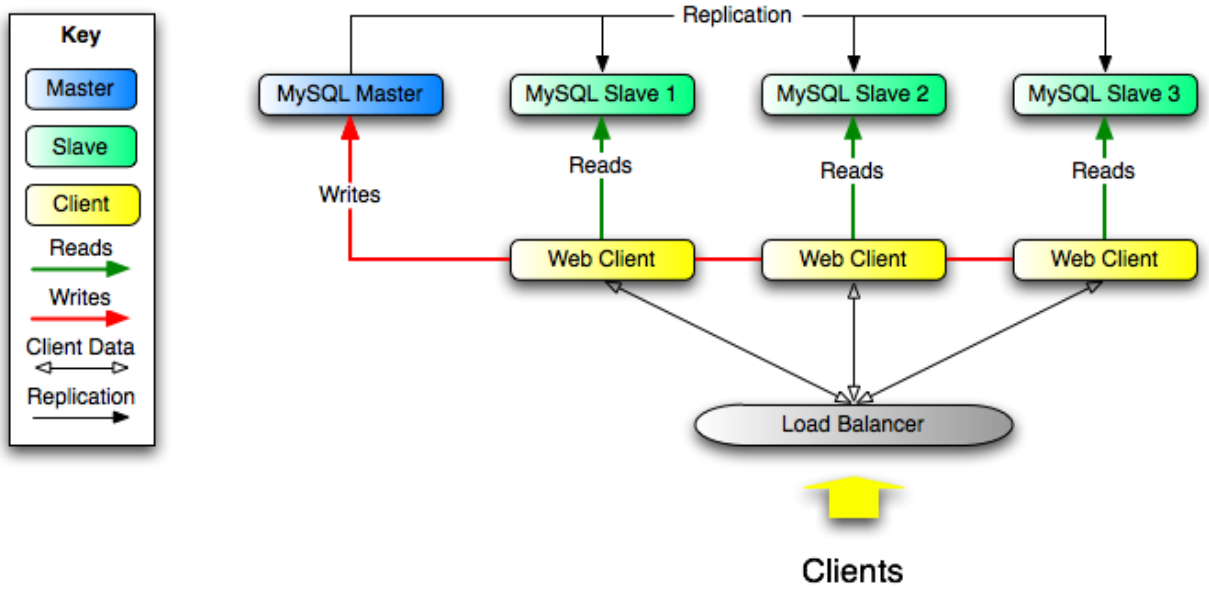
- как показывают тесты, прирост производительности на сервере, на котором установлено веб-окружение, по сравнению с "голым" сервером с настройками по умолчанию - как минимум, в 2-3 раза;
- требуется минимальное время для разворачивания системы - полностью настроенную среду можно получить в буквальном смысле в считанные минуты, использование веб-окружения по нашим подсчетам экономит 200-300 часов администрирования.
- Настройки веб-окружения сбалансированы под большие нагрузки.



4.2. Настройка репликации MySQL, аварийное переключение slave->master

Репликация MySQL используется для решения следующих задач:

1. Увеличение производительности СУБД путем подключения к ней серверов для адаптации к возрастающей нагрузке. Производительность растет за счет выделения одного сервера преимущественно для модификации данных (master) и остальных для чтения данных (slaves). Данное решение особенно эффективно для веб-приложений: у данной категории приложений большая часть запросов к СУБД - запросы на чтение.
2. Он-лайн бэкап - данные передаются в режиме он-лайн на резервные/slave-сервера. При отказе основного сервера на одном из резервных/slave серверов имеется "свежая" копия данных.
3. Организация эффективного резервного копирования - бэкап делается с резервного сервера без прерывания /замедления работы основного сервера СУБД. При необходимости, для осуществления целостного бинарного бэкапа СУБД (в особенности InnoDB), можно остановить на необходимое время резервный сервер, выполнить бэкап и запустить резервный сервер снова.
4. Обеспечение высокой доступности - при выходе из строя одного из серверов СУБД, система продолжает обрабатывать запросы.



Подробнее о примерах использования репликации MySQL читайте в официальной документации.

Надежность репликации

Для обеспечения максимальной надежности репликации рекомендуется установить параметры MySQL следующим образом:

innodb_flush_log_at_trx_commit = 1
sync_binlog = 1
sync_relay_log = 1
sync_relay_log_info = 1

Однако имейте в виду: установка именно таких параметров может привести к общему снижению производительности системы.

Для повышения производительности можно использовать такие параметры (чревато потерей данных нескольких транзакций в момент аварии на базе данных):

innodb_flush_log_at_trx_commit = 2
sync_binlog = 0

Динамический hostname

Если у настраиваемого сервера динамический IP-адрес/hostname, рекомендуется явно задать параметры: relay-log, relay-log-index.

Привилегии

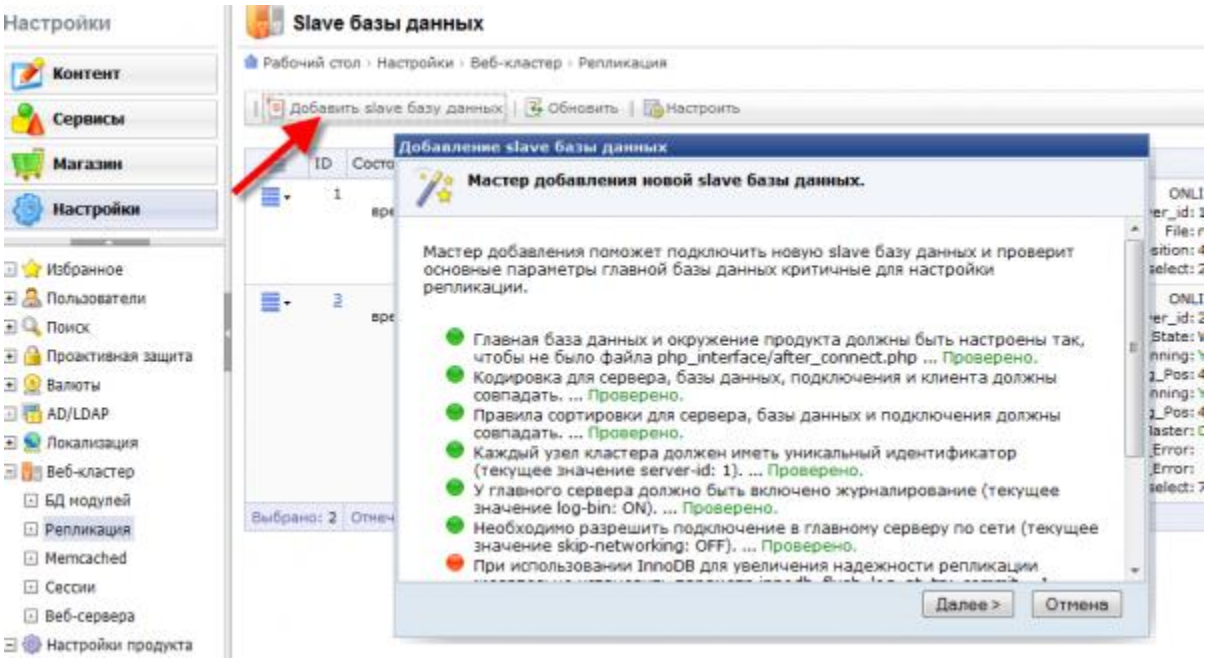
Для работы репликации учетные записи основного и резервных/slave-серверов должны иметь, кроме стандартных, также привилегии: SUPER, REPLICATION CLIENT, REPLICATION SLAVE.

Временные зоны

Если сервера СУБД кластера расположены в разных дата-центрах, необходимо настроить на них единую временную зону.

Настройка репликации в административном интерфейсе

При добавлении резервного/slave-сервера СУБД мастер настройки репликации проверяет все необходимые параметры:



После успешного добавления slave-сервера отображается его статус:

Статус
ONLINE server_id: 1 File: mysql-bin.000016 Position: 45050135 Com_select: 264856 (+466)
ONLINE server_id: 2 Slave_IO_State: Waiting for master to send event Slave_IO_Running: Yes Read_Master_Log_Pos: 45050135 Slave_SQL_Running: Yes Exec_Master_Log_Pos: 45050135 Seconds_Behind_Master: 0 Last_IO_Error: Last_SQL_Error: Com_select: 741 (+1)

Администрирование репликации

Репликация после настройки работает надежно и требует минимального администрирования. Тем не менее, рекомендуется периодически проверять ее состояние утилитами мониторинга операционной системы (nagios, zabbix, monit, linux-ha).

В маловероятном случае возникновения ошибки на slave-сервере рекомендуется его переинициализировать - заново залить на него данные с основного сервера. Для этого нужно его "Прекратить использовать", а затем "Начать использовать" в разделе «Настройки/Веб-кластер/Репликация».

Резервное копирование

Можно свободно останавливать slave-сервера, в т.ч. для осуществления логического и целостного бинарного резервного копирования средствами MySQL и операционной системы. При этом не прерывается работа основного сервера СУБД.

Переключение slave->master в случае отказа master

В случае отказа основного (master) сервера СУБД, необходимо вручную или автоматически скриптом переключить кластер на другой master-сервер СУБД. Для этого обычно slave-сервер, хранящий последние реплицированные данные, переводят в режим основного.

Общая схема этой процедуры такова:

1. Закрываем доступ клиентов к веб-приложению

Если используется двухуровневая конфигурация (фронтэнд nginx - бэкэнд apache и т.п.), рекомендуется на фронтэнде отключить доступ к бэкэнду (веб-приложению) и отдавать при обращении клиентов к кластеру информационную страницу о регламентных работах.

2. Ожидаем "досинхронизации" slave-серверов

Останавливаем на всех slave-серверах поток получения обновлений бинарного лога с основного (master) сервера:

```
STOP SLAVE IO_THREAD;
SHOW PROCESSLIST;
```

Ждем, пока от потока выполнения команд slave (SQL_THREAD) не появится сообщение "Has read all relay log; waiting for the slave I/O thread to update it", говорящее о том, что slave-сервер выполнил все команды из relay-лога в своей базе. Сразу останавливать slave командой 'STOP SLAVE' не рекомендуется, т.к. не все SQL-команды могут быть выполнены из relay-лога (по причине отставания и т.п.), а при переключении slave на новый мастер, relay-лог будет очищен и, возможно, потеряется часть "непроигранных" данных.

3. Подготовка нового master-сервера

Убеждаемся, что на slave-сервере, который мы хотим сделать master-сервером, бинарный лог ведется и не логируются запросы из master:

```
SHOW VARIABLES LIKE 'log_bin';
log_bin       | ON
```

```
SHOW VARIABLES LIKE 'log_slave_updates';
 log_slave_updates | OFF
```

Полностью останавливаем slave - потоки чтения бинарного лога и выполнения SQL-команд:

```
STOP SLAVE;
RESET MASTER;
```

Команда RESET MASTER необходима для очистки бинарного лога нового master, иначе, если в бинарном логе будут записи (устаревшие и т.п.), они проиграются на подключаемых к нему slave-серверах. Такое возможно, если сервер был master с включенным бинарным логом, потом стал slave и перестал использовать бинарный лог, потом снова переводится в режим master.

Итак, новый master подготовлен, у него очищен бинарный лог и он готов обрабатывать запросы.

4. Переключение slave-серверов на новый master-сервер

На всех slave-серверах выполняем:

```
STOP SLAVE;
CHANGE MASTER TO MASTER_HOST='#new_master_host_name#';
START SLAVE;
```

В момент выполнения на slave 'CHANGE MASTER ...' очищается relay-лог slave-сервера, а позиция с которой читается бинарный лог master-сервера устанавливается, если не задано иное, в значение по умолчанию: первый файл бинарного лога, 4 позиция (т.е. в самое начало бинарного лога master-сервера).

5. Переключаем веб-приложение на новый master-сервер

Необходимо настроить кластер на использование нового master-сервера. Для этого вписываем его характеристики ($DBHost) в файл '/bitrix/php_interface/dbconn.php'. Затем, если интервал синхронизации статического контента кластера достаточно велик, вручную запускаем процесс синхронизации на master-сервере: 'csync2 - x' (см. раздел «4.3.2. Синхронизация данных между серверами»).

6. Открываем доступ клиентов к веб-приложению

Если используется двухуровневая конфигурация (фронтэнд nginx - бэкэнд apache и т.п.), на фронтэнде убираем информационную страницу о регламентных работах и переключаем запросы на бэкэнд (веб-приложение).

В итоге кластер использует новый master-сервер.

7. Подключение вышедшего из строя master-сервера как slave-сервера

В административном разделе "Настройки/Веб-кластер/Репликация" нажимаем "Добавить slave базу данных" и следуем указаниям мастера. Происходит переинициализация slave-сервера: в него переносится текущая база данных с master-сервера и включается репликация.

4.3. Вертикальный шардинг – распределение модулей продукта в разные базы данных

В разделе «Настройки» - «Веб-Кластер» - «Шардинг» таблицы с данными некоторых модулей можно вынести в отдельную базу данных, перераспределив нагрузку, создаваемую запросами между различными серверами. Для этого нажмите кнопку "Добавить новую базу данных". После добавления подключения данные модуля можно перенести двумя способами.

Первый способ доступен только для MySQL. Он заключается в том, чтобы в списке баз данных модулей в меню действий выбранной базы данных выполнить команду "Начать использовать". Запустится мастер переноса данных.

Второй способ заключается в удалении модуля и последующей его установки. На первом шаге установки появится возможность указать базу данных для использования модулем. В этом случае таблицы модуля не будут перенесены.

На данный момент поддерживаются следующие модули:

- Веб-аналитика
- Поиск

Вертикальный шардинг - разделение одной базы данных веб-приложения на две и более базы данных за счет выделения отдельных модулей, **без изменения логики работы веб-приложения**.

4.4. Кластеризация веб-сервера

Использование нескольких веб-серверов для работы сайта означает, что любой запрос любого посетителя сайта может попасть на любой из узлов кластера (о том, как именно могут распределяться коннекты к разным веб-серверам мы расскажем далее в главе "Способы балансировки нагрузки между нодами веб-сервера").

Такой принцип работы порождает ряд вопросов.

1. На каждом веб-сервере должен находиться одинаковый контент (файлы). Например, картинка, загруженная через административный интерфейс на одном веб-сервере, должна быть доступна для отдачи посетителю через другой веб-сервер.

2. Пользовательская сессия должна быть "прозрачной" для всех серверов веб-кластера. Например, после авторизации на одном из серверов пользователь должен считаться авторизованных и для всех других серверов. И наоборот - окончание сессии на любом сервере должно означать ее окончание на всех серверах сразу.

Сразу отметим, что не все эти задачи могут быть решены в полной степени на уровне только лишь веб-приложения. Однако мы опишем возможные подходы, покажем на примерах конкретные реализации и расскажем о тех инструментах модуля "Веб-кластер" платформы "1С-Битрикс", которые помогают решить вопрос кластеризации веб-сервера.

Существует два возможных метода обеспечения синхронизации данных на серверах:

1. Использование общего дискового хранилища данных.
2. Использование инструментов синхронизации данных между локальными

хранилищами разных серверов.

О каждом из них мы расскажем в соответствующих главах. Кроме того, опишем возможность использования пула серверов memcached (что позволяет значительно сократить объем синхронизируемых файлов за счет вынесения кеша продукта с файловой системы в memcached) и различные варианты хранения пользовательских сессий.

Каждый веб-сервер кластера можно добавить в общий веб-кластер для мониторинга в административном интерфейсе в разделе "Настройки" - "Веб-кластер" - "Веб-сервера".

На каждом из серверов необходимо настроить страницу, на которой будет отображаться статистика веб-сервера Apache (с помощью модуля mod_status). Если используется "1С-Битрикс: Веб-окружение", необходимо:

1. Внести изменения в конфигурационный файл Apache (/etc/httpd/conf/httpd.conf) - добавить:

```
<IfModule mod_status.c>
ExtendedStatus On
<Location /server-status>
    SetHandler server-status
    Order allow,deny
    Allow from 10.0.0.1
    Allow from 10.0.0.2
    Deny from All
</Location>
</IfModule>
```

Директива *Location* обозначает адрес, по которому будет доступна статистика, строки *Allow from* определяют, с каких IP-адресов статистика будет доступна для просмотра. Можно указать IP всех веб-серверов кластера.

Перечитать конфигурационные файлы Apache с помощью команды:

```
# service httpd reload
```

2. Внести изменения в конфигурационный файл nginx (/etc/nginx/nginx.conf) - в первой секции *server* добавить:

```
Location ~ ^/server-status$ {
  proxy_pass http://127.0.0.1:8888;
}
```

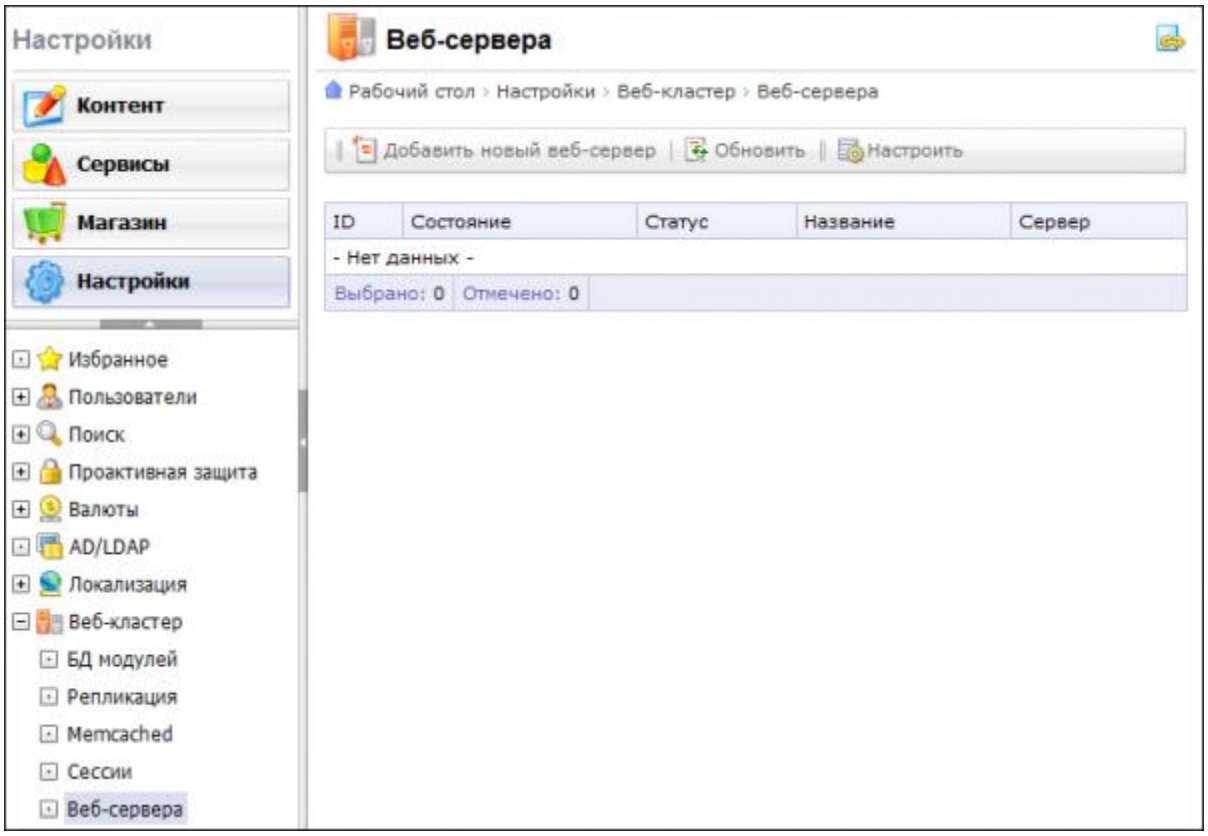
Перечитать конфигурационные файлы nginx с помощью команды:

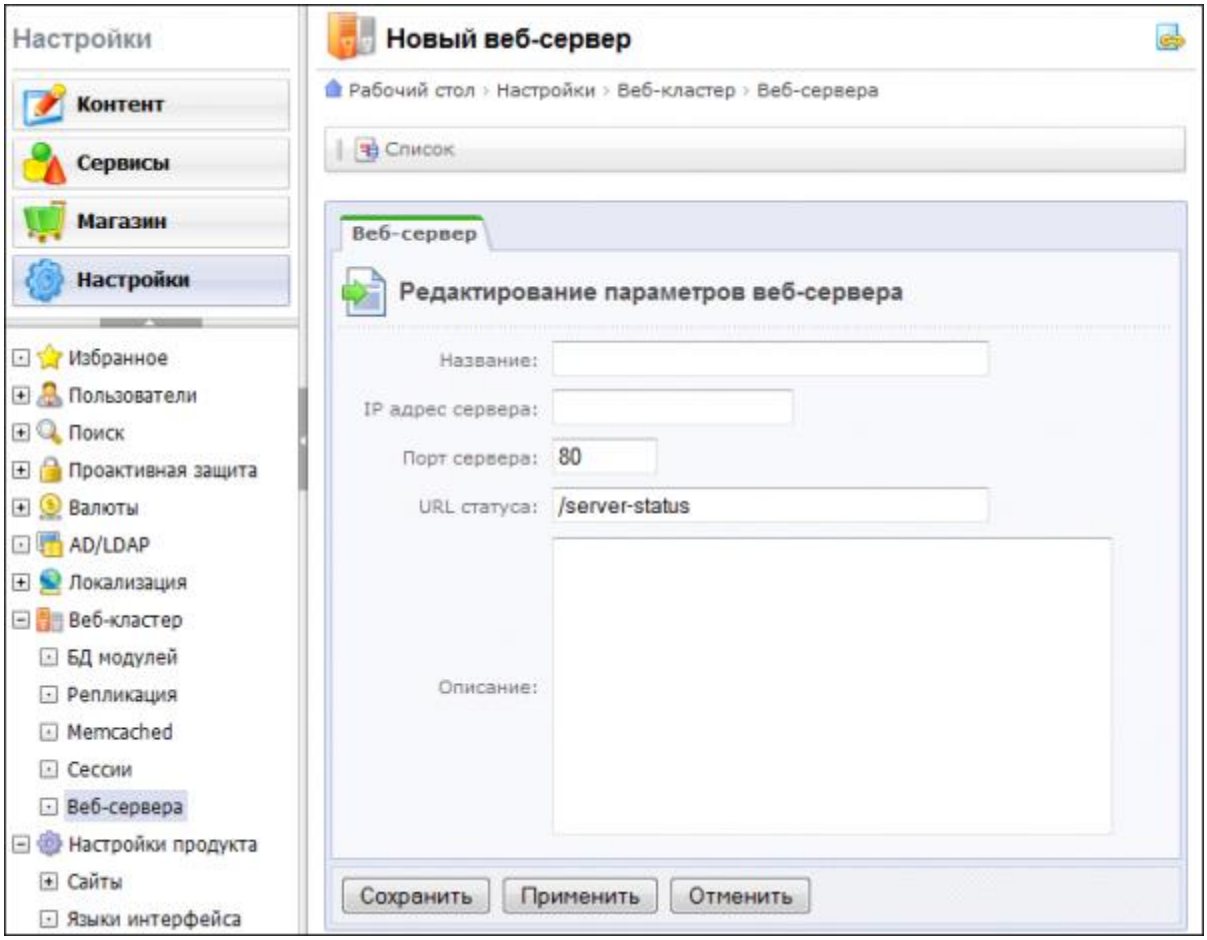
```
# service nginx reload
```

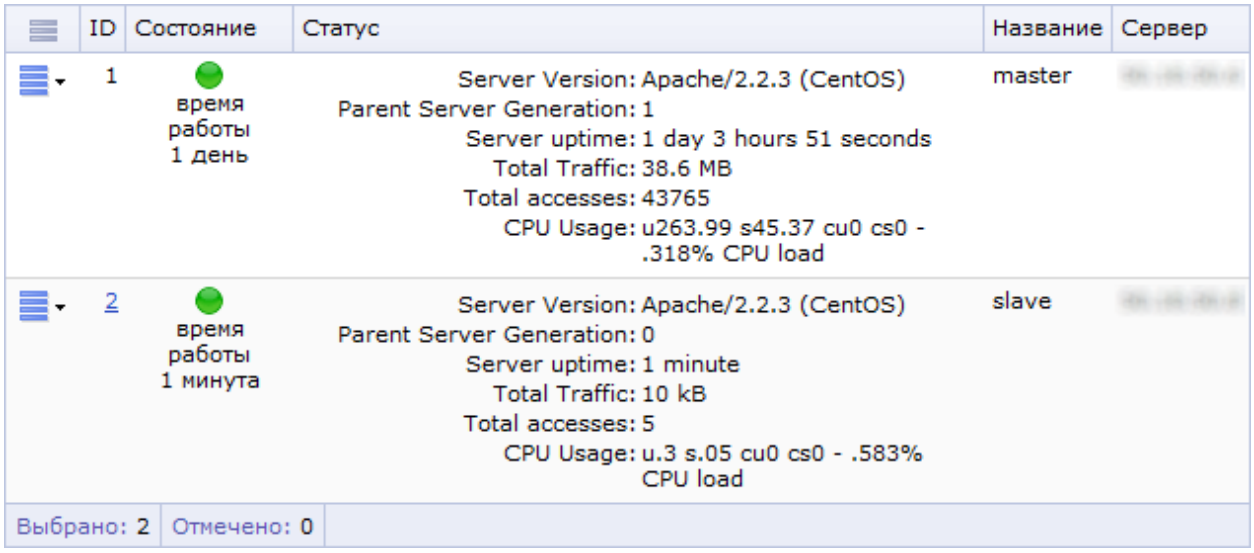
3. Отредактировать файл /home/bitrix/www/.htaccess, добавив после строки:

```
RewriteCond %{REQUEST_FILENAME} !/bitrix/urlrewrite.php$

...строку:

RewriteCond %{REQUEST_URI} !/server-status$
```

После внесения всех необходимых изменений адрес server-status'а можно добавить в конфигурацию кластера:







4.4.1. Общее файловое хранилище данных

Задачу организации общего файлового хранилища данных веб-кластера можно решить разными, хорошо известными способами, начиная от "расшаривания" папки с файлами продукта на одной из нод, заканчивая высокопроизводительными SAN/NAS-решениями уровня предприятия.

NAS-сервис веб-кластера на базе NFS, SMB/CIFS

Если веб-кластер создается на двух нодах, на одной ноде можно запустить сервер NFS/CIFS и обращаться к хранимым на нем файлам с других нод веб-кластера.

Для увеличения производительности можно организовать выделенный NFS/CIFS сервер, оптимизированный исключительно под файловые операции, используемый всеми нодами веб-кластера.

Для достижения большей производительности и надежности общего файлового хранилища веб-кластера рекомендуется обратить внимание на технологии: OCFS, GFS, Lustre, DRBD, SAN.

4.4.2. Синхронизация данных между серверами

Другой подход в организации нескольких серверов состоит в использовании локальных хранилищ (дисков) и постоянной синхронизации файлов между ними.

Существует множество инструментов, позволяющих решить эту задачу - начиная с простых утилит копирования файлов (ftp, scp) и заканчивая специализированным ПО для синхронизации данных между серверами (rsync, unison).

В нашем кластере для синхронизации данных между серверами мы будем использовать утилиту csync2.

Почему мы выбрали именно ее, какие ее ключевые особенности по сравнению с аналогами?

- Высокая скорость работы.
- Низкое потребление ресурсов (CPU, дисковые операции). Два этих фактора позволяют запускать процесс синхронизации максимально часто, поэтому данные на серверах становятся идентичными практически в "реальном времени".
- Простота настройки для обмена данными между любым количеством серверов.
- Возможность синхронизации удаления файлов.
- Защищенный обмен данными между хостами (SSL).

Подробная документация по использованию и конфигурированию csync2 доступна на официальном сайте разработчика: http://oss.linbit.com/csync2/ - мы же рассмотрим пример конкретной конфигурации для двух веб-серверов в кластере.

В некоторых дистрибутивах Linux (например, в Ubuntu) csync2 уже присутствует в репозиториях и доступен для установки из пакетов. Для CentOS это, к сожалению, не так, поэтому необходимо рассмотреть иные варианты установки (сборка из исходников, установка из доступных rpm).

Один из возможных вариантов установки:

1. Устанавливаем необходимые доступные пакеты (библиотеки rsync используются в csync2, из xinetd будет запускаться серверная часть csync2):

```
# yum install rsync librsync-devel xinetd
```

2. Устанавливаем libtasn1 - библиотеку для работы с X.509 (требуется для csync2):

```
# wget
http://ftp.freshrpms.net/pub/freshrpms/redhat/testing/EL5/cluster/x86_
64/libtasn1-1.1-1.el5/libtasn1-1.1-1.el5.x86_64.rpm
# wget
http://ftp.freshrpms.net/pub/freshrpms/redhat/testing/EL5/cluster/x86_
64/libtasn1-1.1-1.el5/libtasn1-devel-1.1-1.el5.x86_64.rpm
# wget
http://ftp.freshrpms.net/pub/freshrpms/redhat/testing/EL5/cluster/x86_
64/libtasn1-1.1-1.el5/libtasn1-tools-1.1-1.el5.x86_64.rpm
# rpm -ihv libtasn1-1.1-1.el5.x86_64.rpm libtasn1-devel-1.1-
1.el5.x86_64.rpm libtasn1-tools-1.1-1.el5.x86_64.rpm
```

3. Устанавливаем sqlite (2-ой версии) - именно эта версия используется в csync2:

```
# wget
http://ftp.freshrpms.net/pub/freshrpms/redhat/testing/EL5/cluster/x86_
64/sqlite2-2.8.17-1.el5/sqlite2-2.8.17-1.el5.x86_64.rpm
# wget
http://ftp.freshrpms.net/pub/freshrpms/redhat/testing/EL5/cluster/x86_
64/sqlite2-2.8.17-1.el5/sqlite2-devel-2.8.17-1.el5.x86_64.rpm
# rpm -ihv sqlite2-2.8.17-1.el5.x86_64.rpm sqlite2-devel-2.8.17-
1.el5.x86_64.rpm
```

4. Устанавливаем непосредственно csync2:

```
# wget
http://ftp.freshrpms.net/pub/freshrpms/redhat/testing/EL5/cluster/x86_
64/csync2-1.34-4.el5/csync2-1.34-4.el5.x86_64.rpm
# rpm -ihv csync2-1.34-4.el5.x86_64.rpm
```

5. После установки уже доступны сгенерированные SSL-сертификаты, однако, если их нужно сгенерировать заново, сделать это можно так:

```
# openssl genrsa -out /etc/csync2/csync2_ssl_key.pem 1024
# openssl req -new -key /etc/csync2/csync2_ssl_key.pem -out
/etc/csync2/csync2_ssl_cert.csr
# openssl x509 -req -days 600 -in /etc/csync2/csync2_ssl_cert.csr -
signkey /etc/csync2/csync2_ssl_key.pem -out
/etc/csync2/csync2_ssl_cert.pem
```

6. На одном из хостов необходимо сгенерировать файл ключей, который затем будет необходимо разместить на всех серверах, которые будут участвовать в процессе синхронизации:

```
# csync2 -k /etc/csync2/csync2.cluster.key
```

Файл csync2.cluster.key на всех машинах должен быть одинаковым и должен быть размещен в /etc/csync2/.

7. Пример конфигурационного файла /etc/csync2/csync2.cfg:

```
group cluster
{
  host node1.demo-cluster.ru node2.demo-cluster.ru;

  key /etc/csync2/csync2.cluster.key;

  include /home/bitrix/www;
  exclude /home/bitrix/www/bitrix/php_interface/dbconn.php;

  auto younger;
}
```

host - адреса серверов, для которых синхронизируются данные;
key - файл ключей;
include - директория, файлы которой синхронизируются;
exclude - исключения (не синхронизировать эти данные);
auto - способ разрешения конфликтов в ситуациях, когда один и тот же файл был изменен на нескольких серверах одновременно; "younger" - приоритет имеет сервер, где были сделаны последние изменения.

8. Обратим особое внимание на директиву *host*.

В конфигурационном файле csync2 может быть задано несколько групп синхронизации. csync2 автоматически игнорирует те группы, в которых не задано локальное имя машины (которое можно посмотреть, например, с помощью команды hostname).

Поэтому имейте в виду, что имена машин, данные которых синхронизируются, не должны изменяться. Для этого можно использовать файл /etc/hosts для указания постоянных IP для конкретных имен. А для того, чтобы задать фиксированное имя для каждого сервера, можно, например, указать его в каком-либо файле:

```
# echo "node1.demo-cluster.ru" > /etc/hostname
```

И в стартап-скрипте /etc/init.d/network предпоследней строкой (до exit) вписать:

```
/bin/hostname -F /etc/hostname
```

9. Если изначально каждый новый сервер создается уже с копией данных с других серверов (например, в случае использования Amazon EC2 удобно сделать snapshot имеющегося instance, а затем подключить его для нового instance), то можно быстро выполнить первую инициализацию базы файлов для синхронизации с помощью команды:

```
# csync2 -cIr
```

Сделать это нужно на каждом хосте.

10. Работа csync2 на каждом хосте состоит из двух частей - серверной и клиентской. Для запуска серверной части необходимо закомментировать (символом #) в файле /etc/xinetd.d/csync2 строку "disable = yes", перезапустить сервис xinetd и разрешить его запуск при загрузке системы:

```
# service xinetd restart
# chkconfig xinetd on
```

11. Клиентская часть - запуск csync2 с ключом "-x".

Можно определить (в зависимости от объема данных) необходимую частоту обновлений и прописать запуск csync2, например, через cron. Строка в /etc/crontab:

```
*/5 * * * * root /usr/sbin/csync2 -x >/dev/null 2>&1
```

...означает запуск csync2 каждые 5 минут.

4.5. Кластеризация кеша (memcached)

Для централизованного и надежного хранения данных кэша используется кластер серверов memcached, который активно используется в таких проектах как Youtube, Facebook, Twitter, Google App Engine.

Для запуска сервера memcached на "1С-Битрикс: Веб-окружение" (Linux) необходимо выполнить команды:

```
# chkconfig memcached on
# service memcached start
```

Улучшенная производительность за счет централизации

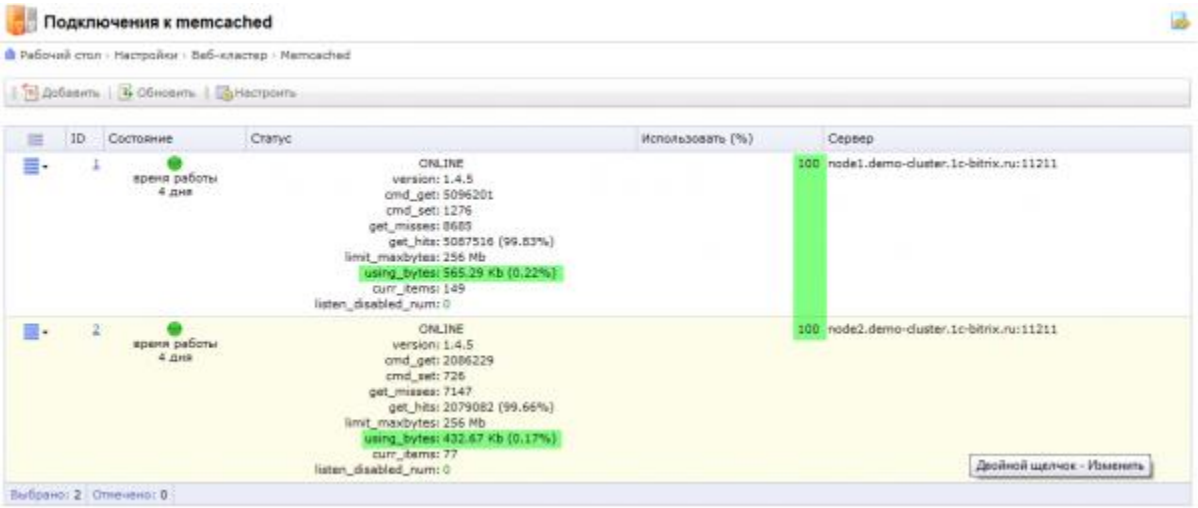
Для обеспечения максимальной производительности веб-кластера реализовано централизованное, разделяемое между нодами хранилище данных кэша - кэш, созданный в результате ресурсоемких вычислений нодой "A", будет использован нодой "B" и остальными нодами кластера. Чем больше нод, тем больший эффект даст использование централизованного кэша.

Улучшенная эффективность за счет вытеснения устаревших данных

За счет использования в сервере memcached алгоритма LRU, в кэше будут храниться только наиболее актуальные данные, а редко используемые - вытесняться. Это не позволит объему кэша безгранично разрастаться в размерах (например, из-за ошибок разработчиков при интеграции), что приводило не только к неэффективному использованию ресурсов системы, но и отрицательно сказывалось на скорости работы с кэшем за счет роста его объема.

В результате кэш веб-кластера будет автоматически поддерживаться в максимально эффективном состоянии: как по скорости доступа, так и по использованию ресурсов.

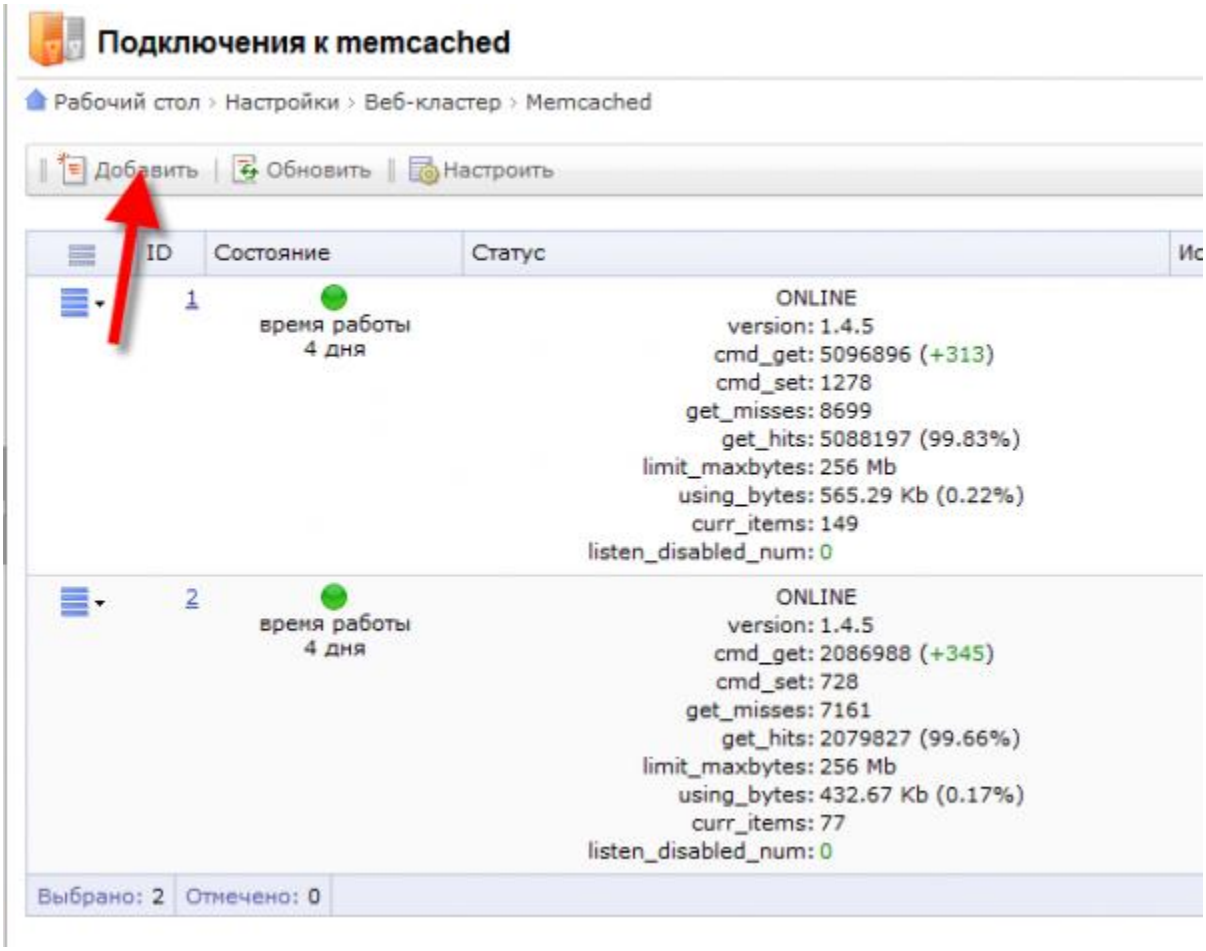
Рекомендуется периодически анализировать использование кэша приложением и подобрать оптимальный размер кэша:



Если сервера memcached имеют разный размер (обусловленный, возможно, разным объемом оперативной памяти на физических серверах), можно регулировать их использование через задание "веса" (колонка "Использовать").

Улучшенная надежность и масштабируемость

За счет кластеризации данных кэша на нескольких серверах memcached, выход из строя одного memcached сервера не выведет из строя подсистему кэширования. В случае увеличения объема данных кэша необходимо подключить к веб-кластеру дополнительный memcached сервер:

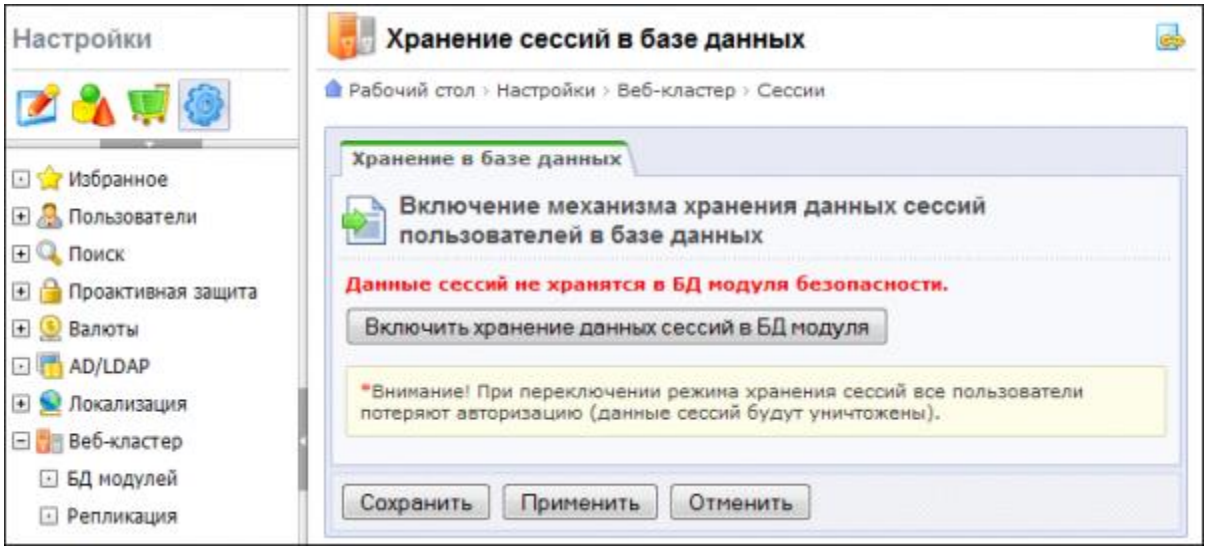


Безопасность

Необходимо открыть доступ к серверам memcached только для нод веб-кластера путем настройки файервола. При использовании хостинга Amazon Web Services это можно сделать через редактирование параметров группы безопасности (введите параметры, соответствующие вашей конфигурации веб-кластера):

4.6. Хранение сессий

По умолчанию данные о сессиях пользователей хранятся на файловой системе сервера. Информацию об этом можно увидеть в административном интерфейсе в разделе "Настройки" - "Веб-кластер" - "Сессии":

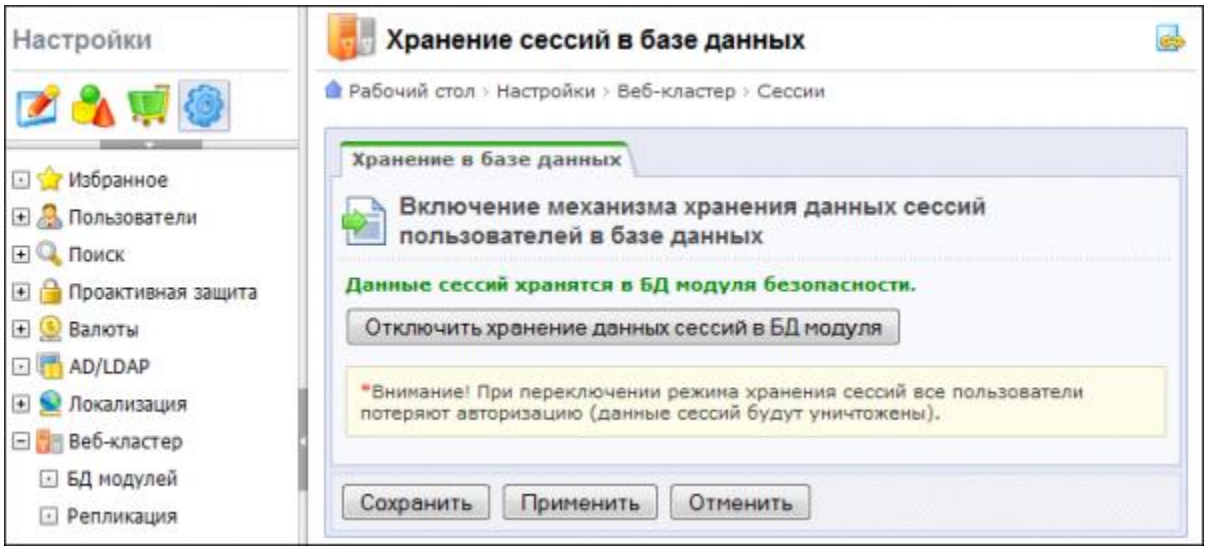


При использовании одного единственного веб-сервера такой способ хранения сессий наиболее удобен. Основной его плюс - наиболее высокая производительность (как показывают различные нагрузочные тесты, скорость генерации страниц сайта при хранении сессий в базе снижается на 3-5%).

Однако при работе с несколькими веб-серверами возможны ситуации, когда один запрос пользователя (например, непосредственно авторизация) попал на один сервер, а следующий или какие-либо другие запросы - на другие серверы, где посетитель еще не будет авторизован. Подобные ситуации будут вызывать целый ряд неудобств для посетителей сайта.

Пользовательская сессия должна быть "прозрачной" для всех серверов веб-кластера.

Для этого используется единое централизованное хранилище для данных сессий. В нашем случае - база данных. Включается хранение данных сессий в базе в разделе "Настройки" - "Веб-кластер" - "Сессии":



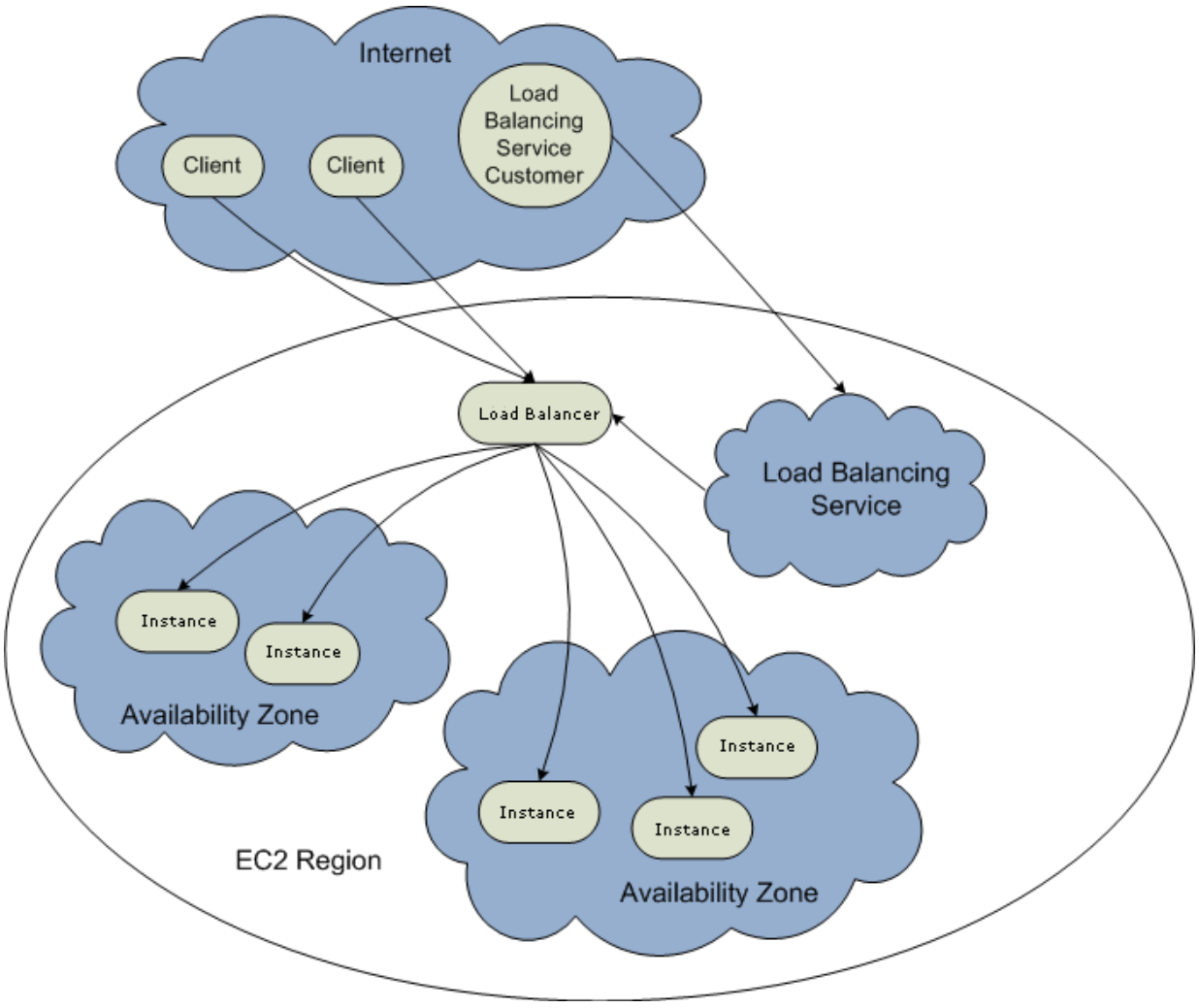
4.7. Способы балансировки нагрузки между нодами веб-сервера

Спектр инструментов, служащих для распределения и балансировки нагрузки между серверами очень широк. Существуют как дорогие аппаратные решения (например, Cisco CSS, Content Services Switch - различных моделей), так и более простые (но тем не менее весьма эффективные) программные средства.

4.7.1. Load Balancer (Amazon Web Services)

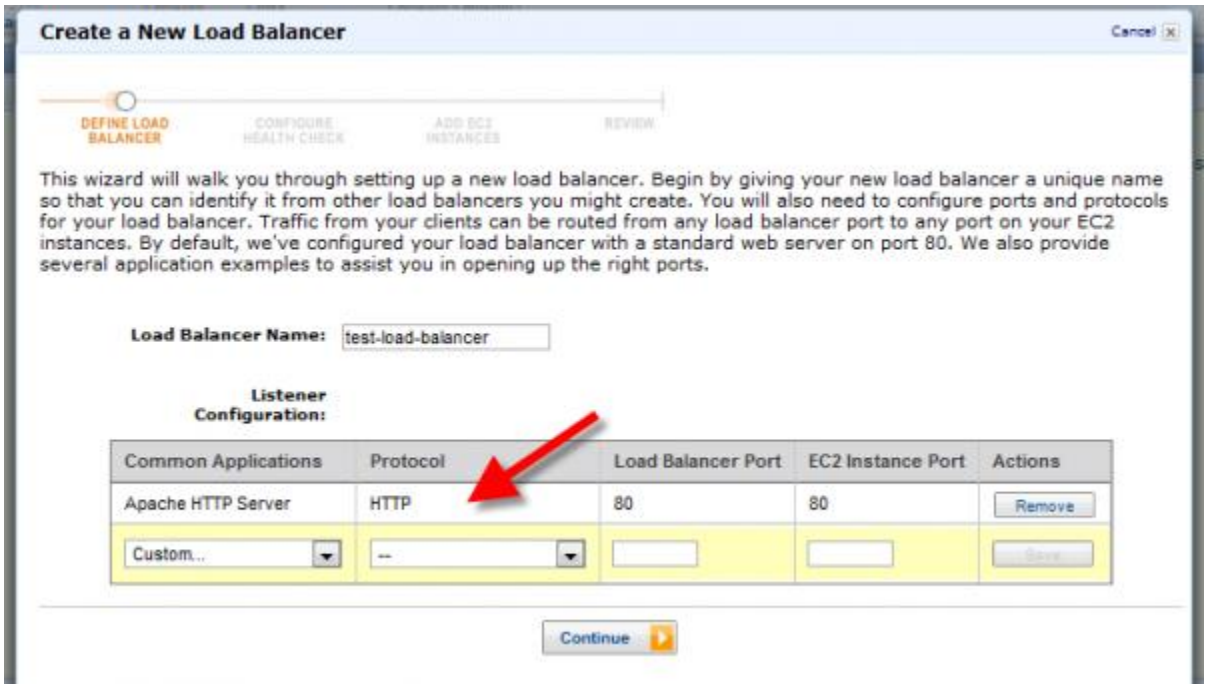
Балансировщик нагрузки AWS использует простой алгоритм балансировки (round robin) и настраивается достаточно просто. Следует обратить внимание, что DNS-имя балансировщика создается автоматически, и необходимо создать для него удобочитаемое CNAME:

my-load-balancer-1820597761.us-west-1.elb.amazonaws.com -> www.mydomain.com

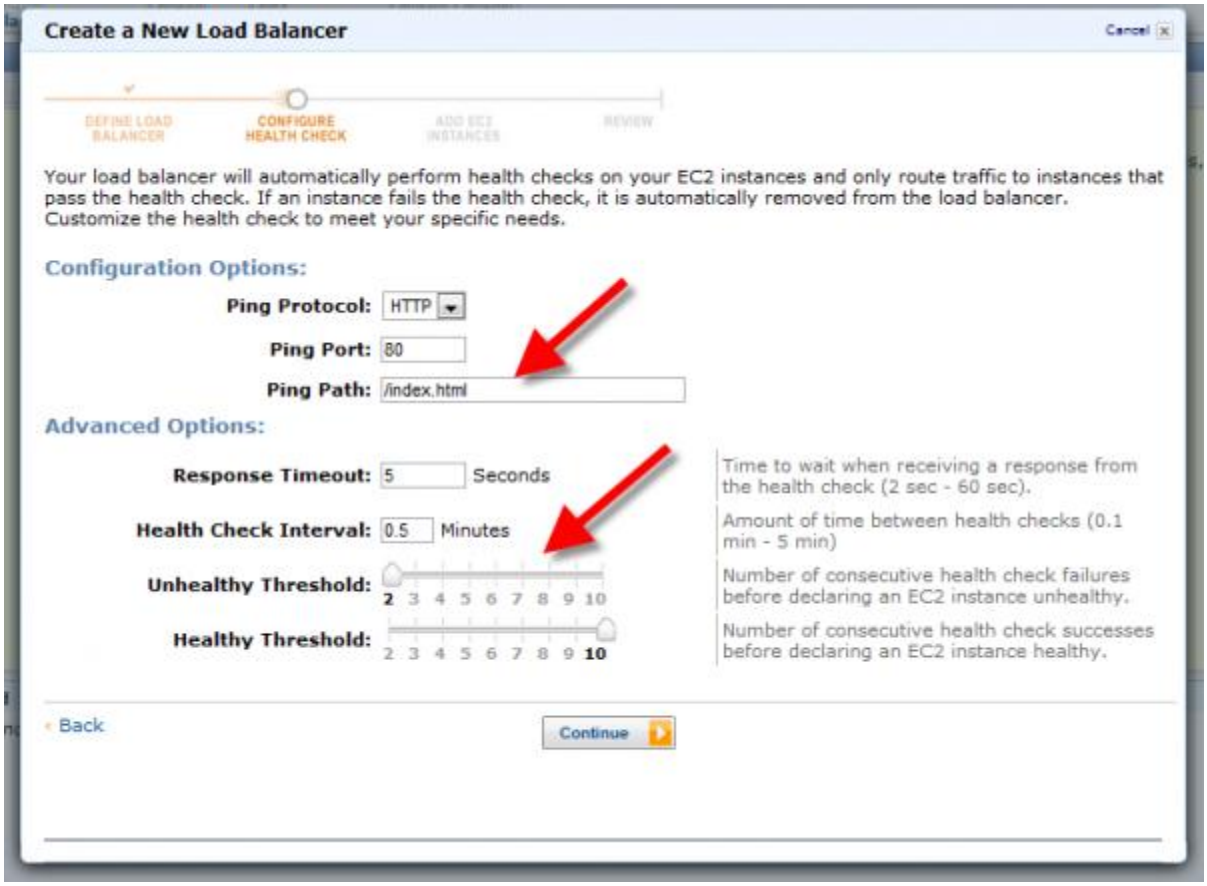


Добавляем балансировщик:

Балансировщик «слушает» порт 80 и перенаправляет запросы на 80 порт нод веб-кластера:

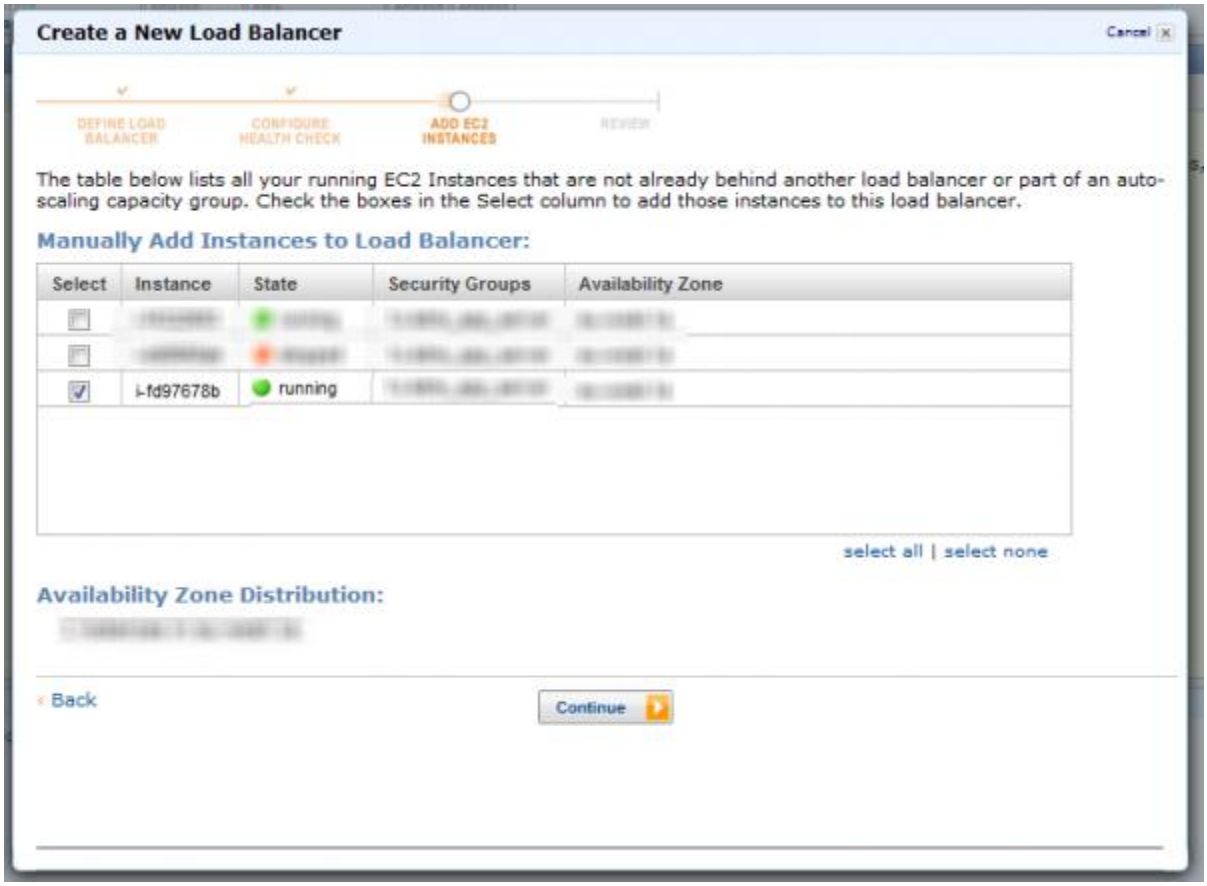


Конфигурируем страницу ноды для проверки и таймауты:



Рекомендуется в 'Ping Path' задать страницу, обращения к которой не фиксируются в модуле "Веб-аналитика". Остальные параметры выбираются в зависимости от ожидаемой нагрузки.

Выбираем ноды, которые будут находиться за балансировщиком. В связи с простым алгоритмом балансировки (round robin), рекомендуется подбирать ноды примерно одинаковой производительности. Если ноды располагаются в разных дата-центрах (availability zones), то рекомендуется их одинаковое количество в каждом дата-центре.



Если мы хотим, чтобы все запросы от одних и тех же клиентов обрабатывались одними и теми же нодами кластера, необходимо добавить привязку сессии посетителя к ноде веб-кластера. Для этого в настройках балансировщика включаем привязку "Enable Load Balancer Generated Cookie Stickiness".

Теперь, независимо от того, сколько нод кластера размещены за балансировщиком, они будут доступны по единому доменному имени: www.mydomain.com. В случае выхода ноды из строя/перегрузки, балансировщик перестает направлять на нее посетителей кластера.

Подробнее о балансировщике нагрузки читайте в официальной документации AWS.

4.7.2. DNS

Самый простой способ балансирования нагрузки и распределения запросов между веб-серверами - использование механизма round robin DNS.

Спецификация стандарта DNS позволяет указать несколько разных IN A записей для одного имени. Например:

```
www   IN A   10.0.0.1
www   IN A   10.0.0.2
```

В этом случае DNS сервер в ответ на запрос разрешения имени в IP-адрес будет выдавать не один адрес, а весь список:

```
# host www.domain.local
www.domain.local has address 10.0.0.1
www.domain.local has address 10.0.0.2
```

При этом с каждым новым запросом последовательность адресов в списке в ответе будет меняться. Таким образом клиентские запросы будут распределяться между разными адресами и будут попадать на разные серверы.

Использование round robin DNS - самый простой способ балансировки нагрузки, не требующий никаких дополнительных средств, однако он обладает целым рядом недостатков, поэтому мы рекомендуем использовать его только в том случае, если нет возможности применить какой-либо иной балансировщик.

Минусы применения round robin DNS:

- нет четкого критерия выбора IP из списка клиентом (может выбираться первый элемент, может кэшироваться последнее выбранное значение, возможны иные варианты) - все зависит от реализации конкретного клиента;
- нет механизмов определения доступности узлов и возможности задания их "весов" - в случае аварии на одном или нескольких узлах кластера нагрузка будет продолжать распределяться между рабочими и вышедшими из строя нодами;
- длительное время кэширования ответов DNS - в случае аварии и изменении тех или иных записей в DNS потребуется некоторое время (зависит от настроек DNS) для обновления данных на всех клиентах.

4.7.3. nginx

В данном разделе мы рассмотрим пример использования в качестве балансировщика http-сервера nginx. Для этих целей используется модуль ngx_http_upstream.

При использовании "1С-Битрикс: Веб-окружения" nginx уже установлен на серверах веб-кластера. И для распределения нагрузки можно использовать непосредственно один из серверов кластера. Однако для более простого, гибкого и удобного конфигурирования лучше использовать отдельный сервер с установленным nginx в качестве балансировщика.

В простейшем примере фрагмент конфигурационного файла nginx (/etc/nginx/nginx.conf), обеспечивающего распределение нагрузки между серверами, будет выглядеть так (помимо стандартных директив, определяющих, например, пути и формат log-файлов и т.п.):

```
http {
    upstream backend {
        server node1.demo-cluster.ru;
        server node2.demo-cluster.ru;
    }
```

```
    server {
        listen   80;
        server_name  load_balancer;

        location / {
            proxy_set_header X-Real-IP $remote_addr;
            proxy_set_header X-Forwarded-For
$proxy_add_x_forwarded_for;
            proxy_set_header Host $host:80;

            proxy_pass  http://backend;
        }

    }
}
```

В секции "upstream" задаются адреса серверов, между которыми будет распределяться нагрузка.

В DNS имя сайта указывается на тот IP, на котором работает сервер с nginx, распределяющем нагрузку.

Если не указаны какие-либо дополнительные параметры, запросы распределяются между серверами по принципу round robin.

Однако с помощью модуля ngx_http_upstream можно реализовать и более сложные конфигурации.

1. В некоторых случаях бывает удобно передавать все запросы от одного клиента на один и тот же сервер, а между серверами распределять только запросы разных клиентов.

Для реализации подобного функционала служит директива "ip_hash". Пример:

```
upstream backend {
    ip_hash;

    server node1.demo-cluster.ru;
    server node2.demo-cluster.ru;
}
```

В этом случае распределение запросов основано на IP-адресах клиентов.

2. Если для узлов кластера используются серверы разной конфигурации (разной мощности), бывает полезно задать "вес" для тех или иных хостов, направляя на них больше или меньше запросов.

Для этого служит параметр "weight" директивы "server". Пример:

```
upstream backend {
    server node1.demo-cluster.ru weight=3;
    server node2.demo-cluster.ru;
    server node3.demo-cluster.ru;
}
```

В такой конфигурации каждые 5 запросов будут распределяться следующим образом: 3 - на node1.demo-cluster.ru, 1 - на node2.demo-cluster.ru, 1 - на node3.demo-cluster.ru.

Обратите внимание: для серверов, использующих метод распределения "ip_hash", нельзя задать вес.

3. Можно настроить параметры, по которым будут определяться неработающие серверы.

max_fails=число - задает число неудачных запросов к серверу в течение времени, заданного параметром fail_timeout, после которых он считается неработающим также в течение времени заданного параметром fail_timeout;

fail_timeout=время

Пример:

```
upstream backend {
    server node1.demo-cluster.ru max_fails=3 fail_timeout=30s;
    server node2.demo-cluster.ru max_fails=3 fail_timeout=30s;
}
```

Если при запросе к серверу произошла ошибка, запрос будет передан на следующий сервер. Если произошло 3 ошибки в течение 30 секунд, то на 30 секунд сервер будет помечен неработающим, и в течение этого времени новые запросы не будут на него отправляться.

4.8. Добавление ноды веб-кластера

Задача: в связи с растущей нагрузкой необходимо добавить ноду к веб-кластеру. Фактически, необходимо запустить новый физический/виртуальный сервер и прописать его в настройках веб-кластерра.

Так как наш демо-кластер мы запускали в «облаке» Амазона, мы опишем последовательность действий именно для AWS. Однако и для любой другой среды (будь то VPS или несколько выделенных серверов) общая схема будет примерно такой же.

На облачном хостинге AWS необходимо:

1) Создать снепшот диска с данными приложения одной из нод.
2) Создать из снепшота новый диск, который будет хранить данные оригинальной ноды.
3) Запустить виртуальную машину с AMI, аналогичной оригинальной ноде.

4) Выбрать для виртуальной машины аппаратную конфигурацию в зависимости от ожидаемой нагрузки. В самом простом случае рекомендуется ее выбрать аналогичной оригинальной ноде.
5) Остановить новую ноду.
6) Отключить от нее диск.
7) Вместо отключенного диска подключить созданный в шаге 2 диск.
8) Запустить новую ноду.
9) Привязать при необходимости к новой ноде эластичный IP-адрес.
10) Настроить новую ноду - если используется csync2, то прописать в его настройки на всех нодах веб-кластера доменное имя новой ноды. Запустить на новой ноде необходимые сервисы: memcached, mysql-slave.
11) Рекомендуется в инициализационные скрипты новой ноды добавить ее привязку к эластичному IP-адресу. Иначе при ее останове/запуске нужно будет снова привязывать к ней IP-адрес вручную.
12) Теперь с новой нодой синхронизируется контент с текущих нод веб-кластера. Ее необходимо добавить в балансировщик нагрузки.
12-б) Если новая нода используется как mysql-slave, memcached-сервер - необходимо зарегистрировать сервисы в административном интерфейсе веб-кластера "Рабочий стол/Настройки/Веб-кластер".

4.9. Безопасность

В связи с тем, что веб-кластер использует дополнительные сервисы (централизованное кэширование, синхронизация) и запускается, как правило, на группе машин, рассмотрим особенности обеспечения информационной безопасности веб-кластера.

Балансирование нагрузки и защита от DDOS-атак

Рекомендуется открыть для публичного доступа 80 порт балансировщика нагрузки, ограничив внешний доступ к http портам машин (нод) веб-кластера. Это надежно защитит ноды, спрятанные за балансировщиком, от перегрузки (например, возникшей при проведении рекламной интернет-кампании), а также существенно снизит эффективность DDOS-атак.

Кластеризованный кэш

Необходимо закрыть от публичного доступа серверы memcached (tcp порт 11211), открыв доступ к ним с нод веб-кластера. Одно из решений - сконфигурировать файервол.

Сервис синхронизации контента нод кластера

Если для синхронизации контента используется утилита csync2, необходимо закрыть ее сервисы от публичного доступа (tcp порт 30865), открыв доступ к ним с нод веб-кластера.

Пример настройки файервола

Для ноды веб-кластера:
22 (tcp; ssh) - открыт для подсети администратора
80 (tcp; http) - открыт для подсети веб-кластера
443 (tcp; https) - открыт для подсети веб-кластера
3306 (tcp; mysql) - открыт для подсети веб-кластера
11211 (tcp; memcached) - открыт для подсети веб-кластера
30865 (tcp; csync2) - открыт для подсети веб-кластера

Для балансировщика нагрузки:
80 (tcp; http) - открыт для всех
443 (tcp; https) - открыт для всех

5. Нагрузочное тестирование кластера, анализ различных сценариев и выводы

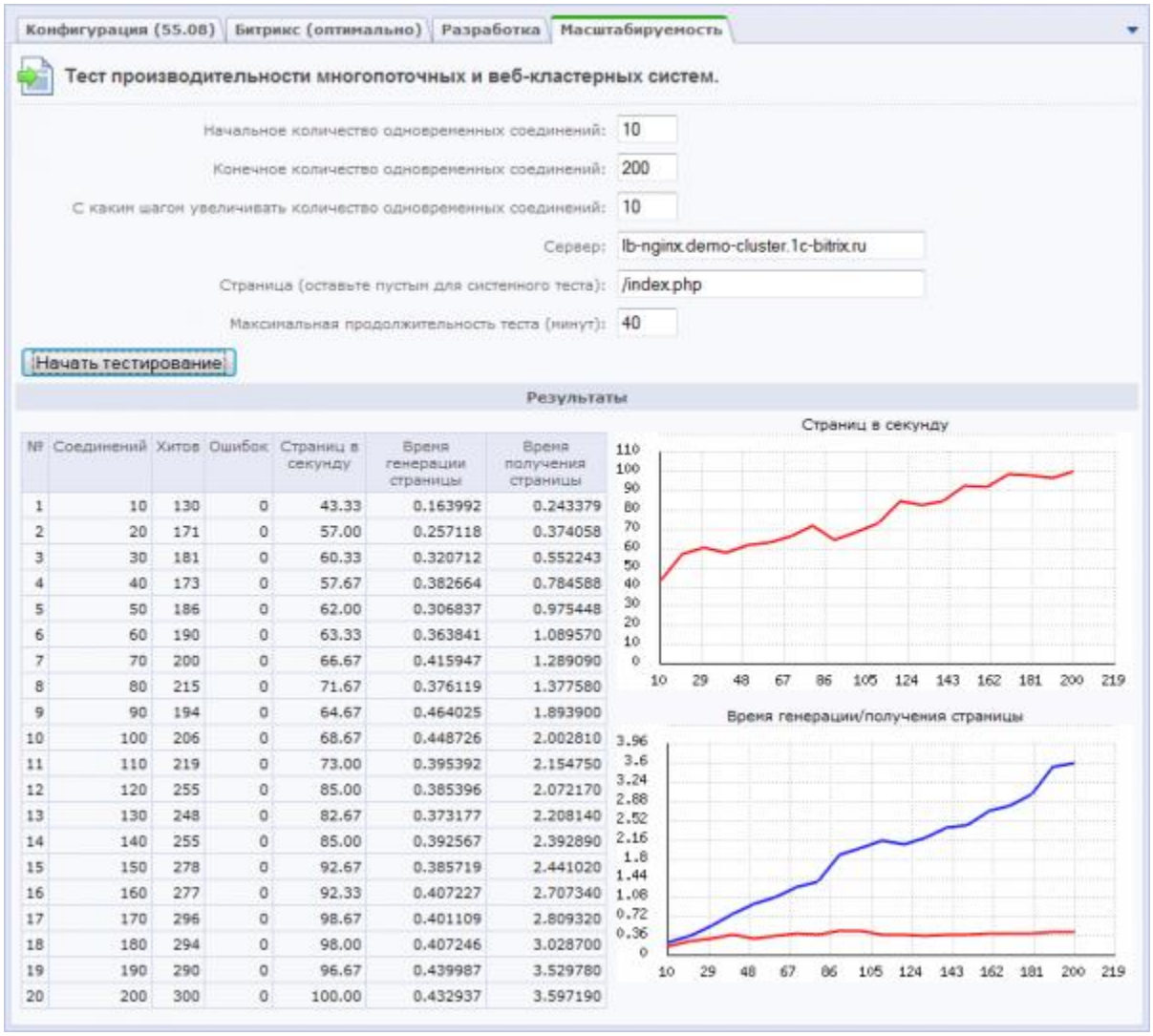
Существует множество утилит для проведения нагрузочных тестов веб-систем. От достаточно простых (ab, входящая в дистрибутив Apache, siege, httperf) до мощных инструментов, позволяющих задавать любые пользовательские сценарии и предоставляющих самую разнообразную статистическую информацию (JMeter, tsung, WAPT).

Начиная с версии 10.0 в модуле монитора производительности платформы "1С-Битрикс" доступен встроенный инструмент тестирования нагрузки.

Обратите внимание: для минимизации влияния самого тестирующего скрипта на результаты тестов рекомендуется запускать его не на самих тестируемых серверах, а на отдельном хосте.

Покажем, как работает этот инструмент и проведем несколько тестов.

1. Рост нагрузки.



№	Соединений	Хитов	Ошибок	Страниц в секунду	Время генерации страницы	Время получения страницы
1	10	130	0	43.33	0.163992	0.243379
2	20	171	0	57.00	0.257118	0.374058
3	30	181	0	60.33	0.320712	0.552243
4	40	173	0	57.67	0.382664	0.784588
5	50	186	0	62.00	0.306837	0.975448
6	60	190	0	63.33	0.363841	1.089570
7	70	200	0	66.67	0.415947	1.289090
8	80	215	0	71.67	0.376119	1.377580
9	90	194	0	64.67	0.464025	1.893900
10	100	206	0	68.67	0.448726	2.002810
11	110	219	0	73.00	0.395392	2.154750
12	120	255	0	85.00	0.385396	2.072170
13	130	248	0	82.67	0.373177	2.208140
14	140	255	0	85.00	0.392567	2.392890
15	150	278	0	92.67	0.385719	2.441020
16	160	277	0	92.33	0.407227	2.707340
17	170	296	0	98.67	0.401109	2.809320
18	180	294	0	98.00	0.407246	3.028700
19	190	290	0	96.67	0.439987	3.529780
20	200	300	0	100.00	0.432937	3.597190

4-ая вкладка "монитора производительности" - "масштабируемость".

Параметры проводимых тестов:

сервер - в нашем случае мы даем тестовую нагрузку на балансировщик (nginx), который распределяет нагрузку между двумя веб-серверами кластера (на каждом сервере работают и добавлены в кластер веб-сервер, MySQL, memcached);
страница - URL, на который отправляются запросы;
начальное количество одновременных соединений, конечное количество, шаг увеличения соединений - параметры нагрузки.

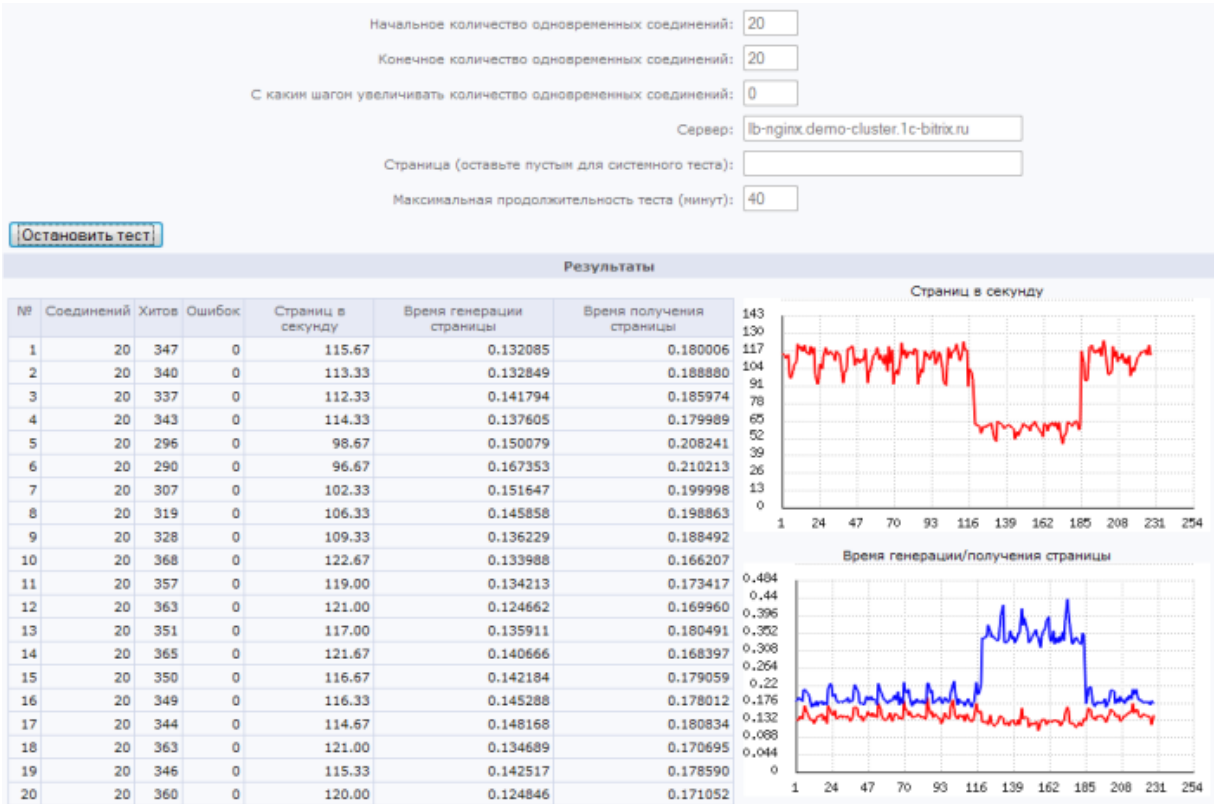
В нашем тестовом примере мы начинаем отправлять запросы на индексную страницу в 10 одновременных потоков и постепенно увеличиваем их количество до 200.

По второму графику мы видим следующую картину:

- вся система сбалансирована по нагрузке, с ее ростом не наступает деградация производительности системы (скорость генерации страниц сервером практически не изменяется, это видно на красном графике);
- увеличивается время отдачи страниц клиентам, растет очередь (синий график).

2. Второй тест - эмуляция аварии (отключения) сервера со slave базой данных MySQL.

Во втором тесте мы эмулируем отключение одной из машин кластера (той, на которой работает SLAVE база данных MySQL).



№	Соединений	Хитов	Ошибок	Страниц в секунду	Время генерации страницы	Время получения страницы
1	20	347	0	115.67	0.132085	0.180006
2	20	340	0	113.33	0.132849	0.188880
3	20	337	0	112.33	0.141794	0.185974
4	20	343	0	114.33	0.137605	0.179989
5	20	296	0	98.67	0.150079	0.208241
6	20	290	0	96.67	0.167353	0.210213
7	20	307	0	102.33	0.151647	0.199998
8	20	319	0	106.33	0.145858	0.198863
9	20	328	0	109.33	0.136229	0.188492
10	20	368	0	122.67	0.133988	0.166207
11	20	357	0	119.00	0.134213	0.173417
12	20	363	0	121.00	0.124662	0.169960
13	20	351	0	117.00	0.135911	0.180491
14	20	365	0	121.67	0.140666	0.168397
15	20	350	0	116.67	0.142184	0.179059
16	20	349	0	116.33	0.145288	0.178012
17	20	344	0	114.67	0.148168	0.180834
18	20	363	0	121.00	0.134689	0.170695
19	20	346	0	115.33	0.142517	0.178590
20	20	360	0	120.00	0.124846	0.171052

На сервере, на котором запущен master MySQL, в этот момент можем увидеть, что сервисы выключены:

	ID	Состояние	Название	Отставание (сек)	Статус	Использовать (%)
	1	время работы 8 дней	Главная база данных	0	ONLINE server_id: 1 File: mysql-bin.000021 Position: 55510328 Com_select: 473098 (+13438)	50
	3	нет подключения	slave #1	0	ERROR	100
Выбрано: 2	Отмечено: 0					

Небольшие пики на графиках – моменты синхронизации контента (у нас в тесте установлен частый период обновлений, его можно уменьшить).

Большой «провал» («всплеск» на синем графике) – момент отключения одной из машин кластера.

3. Третий тест - эмуляция аварии (отключения) сервера с master базой данных MySQL.

Отключение сервера с master базой данных MySQL отличается от предыдущего теста тем, что необходимо выполнить ряд действий для переключения базы данных slave в режим мастера.

Общая схема действий подробно описана в главе «4.2. Настройка репликации MySQL, аварийное переключение slave->master».

6. Варианты конфигурации веб-кластера для решения практических задач

Прежде всего, необходимо внимательно проанализировать (спрогнозировать) характер нагрузки на веб-приложение. Для сбора и анализа данных рекомендуется проактивно использовать известные инструменты мониторинга, такие как munin, zabbix, apache server-status, и т.п.

Ниже перечислены варианты конфигурации веб-кластера для различных типов нагрузки.

Высокая нагрузка на процессоры, невысокая/средняя нагрузка на СУБД, контент меняется редко

Необходимо снизить нагрузку на процессоры - подключаем к веб-кластеру ноду(ы). Ноды заводим под балансировщик нагрузки. В результате нагрузка на процессоры распараллеливается между нодами.

Рекомендуется также на каждой ноде запустить сервер memcached - это позволит еще больше снизить нагрузку на процессоры за счет того, что кэш будет создаваться один раз на одной из нод веб-кластера, а использоваться остальными нодами.

Для синхронизации контента подойдет csync2, nfs/cifs-сервер на одной из нод веб-кластера.

Высокая растущая нагрузка на СУБД

Для разгрузки СУБД необходимо организовать master-slave репликацию. Чем больше будет добавлено slave-нод, тем сильнее снизится нагрузка на основную master базу данных. Если slave-ноды разной мощности, рекомендуется адекватно распределить между ними нагрузку с помощью параметра "Процент распределения нагрузки".

В результате нагрузка на СУБД распределится между master и slave нодами веб-кластера. При дальнейшем увеличении нагрузки на СУБД рекомендуется:
1) Оптимизировать master и slave ноды. Т.к. master-сервер будет использоваться преимущественно для записи, его нужно, путем настройки параметров, оптимизировать для записи, а slave-ноды - оптимизировать для чтения.
2) Добавлять slave-ноды.

Большой объем кэша, кэш часто перестраивается, контент меняется часто, изменений много

Необходимо настроить эффективную работу с кэшем. Для этого запускаем на каждой ноде сервер memcached, выделяем каждому по несколько GB памяти (в зависимости от требований приложения).

Для синхронизации контента рекомендуется использовать выделенный nfs/cifs-сервер, либо ocfs/gfs или аналоги.