Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«Пермский национальный** **исследовательский политехнический университет»**

Факультет: Прикладной математики и механики

Кафедра: Вычислительной математики и механики

Направление: 09.04.02 «Информационные системы и технологии»

Программа: «Информационные технологии и системная инженерия»

Отчёт по лабораторным работам

по дисциплине

«Программирование с использованием

облачных платформ GOOGLE

APPS/AMAZON/AZURE»

Выполнила:

студент гр. ИТСИ-17-1м

Власова Екатерина Александровна

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

*(подпись)*

Принял:

*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*(должность, ФИО руководителя)*

*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*(оценка) (подпись)*

*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*(дата)*

**Пермь 2018**

**Содержание**

[Лабораторная работа №1 3](#_Toc532790942)

[Лабораторная работа №2 7](#_Toc532790943)

[Лабораторная работа №3 11](#_Toc532790944)

[Тестирование нагрузки (подходы и инструменты) 14](#_Toc532790945)

# **Лабораторная работа №1**

**Цель:** получить навыки работы с docker, docker swarm, socker compose, haproxy. Т.к. во всех платформах имеется возможность запуска  docker контейнеров.

**Ход работы:**

1. Создаём новое приложение Node.js – в файле index.js прописываем следующее (рисунок 1):

* **nano index.js**

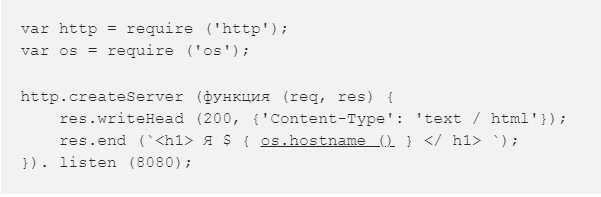


Рисунок 1. Текст index.js

1. Docker - изуем приложение, для этого создаём файл Dockerfile со следующим кодом (рисунок 2):



Рисунок 2. Текст Dockerfile

1. Создаем docker-образ нашего приложения, вводим в консоли следующую команду (рисунок 3):

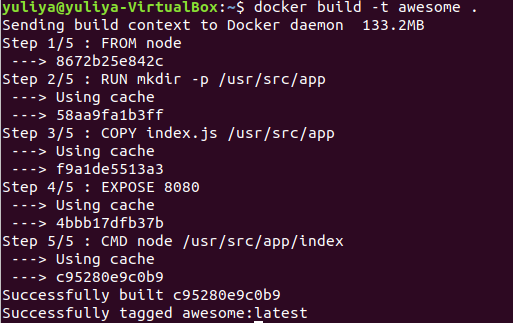
* **docker build -t awesome . **

Рисунок 3. Вызов команды docker build –t awesome .

1. Для нашего HTTP-сервера мы будем использовать HAProxy , это означает, что нам нужно создать контейнер с HAProxy, который будет прослушивать порт 80 и загружать баланс для разных контейнеров Node.js на порт 8080. Чтобы создать наши контейнеры (Node.js приложения и HAProxy), мы будем использовать Docker Compose, напишем наш docker-compose.yml файл (рисунок 4):

* **nano docker-compose.yml**

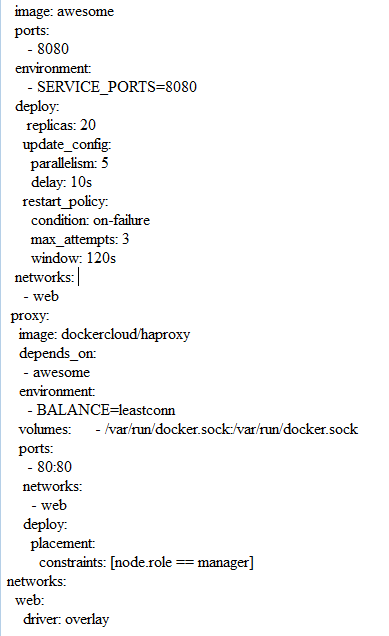


Рисунок 4. Текст docker-compose.yml

Чтоб включить docker swarm:

* **docker swarm init**

1. Cоздаём stack с названием prod из файла **docker-compose.yml:**

* **docker stack deploy --compose-file=docker-compose.yml prod**

1. Следующей командой увидим запущенные сервисы (рисунок 5)

* **docker service ls**

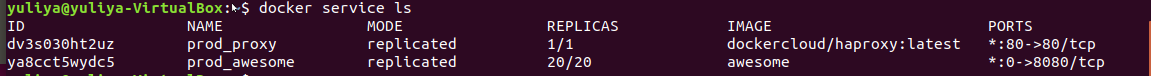


Рисунок 5. Запущенные сервисы

Когда мы откроем браузер и впишем <http://localhost> , то у нас выйдет такой результат – контейнер со своим идентификатором (рисунок 6).

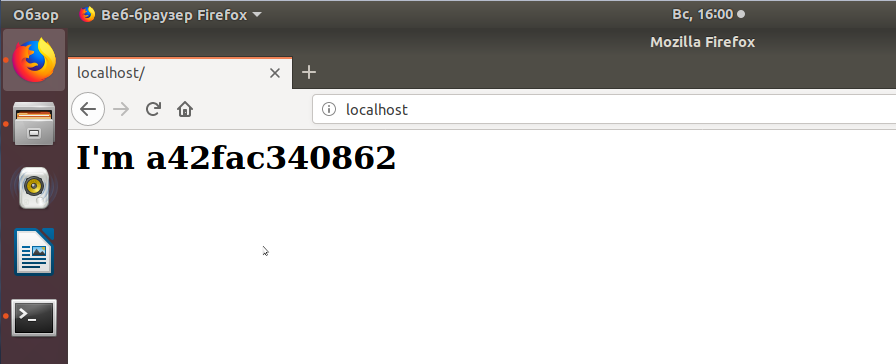


Рисунок 6. Страница в браузере localhost с идентификатором контейнера

Чтобы наверняка убедиться в существовании контейнера с идентификатором a42fac340862, в консоли сделаем проверку (рисунок 7) и, да, такой контейнер существует.

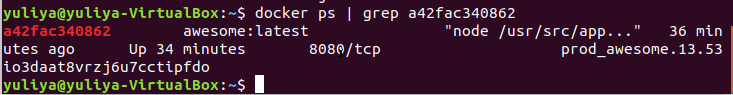


Рисунок 7. Проверка существования контейнера по идентификатору

# **Лабораторная работа №2**

**Задание:** Реализовать веб-приложение с минимальным пользовательским интерфейсом, которое загружает\удаляет файлы по протоколу Amazon S3 в mail ru cloud platform.

**Цель:** получить навыки работы с протоколом S3.

**Ход работы:**

1. Регистрируемся в MAIL.RU CLOUD SOLUTIONS.
2. Создаём аккаунт для старта работы с Объектным хранилищем. Мой аккаунт – Jula.

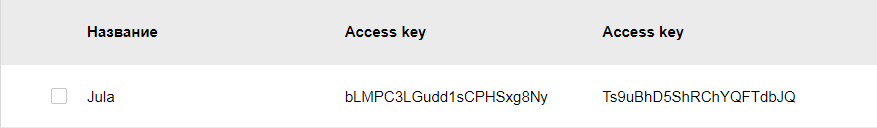


Рисунок 8. Зарегистрированный аккаунт Jula

1. Создаём бакет. В Объектном хранилище все данные хранятся в отдельных бакетах, в которых можно организовывать иерархию своих файлов. Первый созданный бакет называется firstbaket.

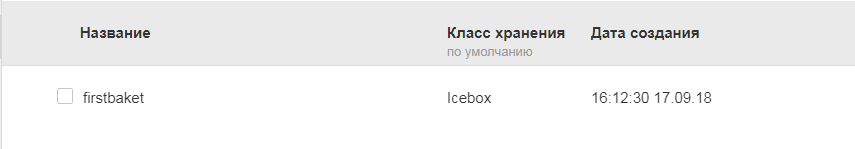


Рисунок 9. Первый созданный бакет firstbaket

1. Для прохождения аутентификации к сервисам Cloud Storage вам понадобятся ключи доступа известные как Access key ID и Secret access key – на рисунке 1 изображён только Access key ID, второй секретный ключ показывают лишь при создании нового Access key, потом он становится не доступен.

* Далее создаём файл учётных данных в ~/.aws /credentials в директории C:\Users\Boss\
* Добавляем ключи доступа в файл, выглядит файл так (рисунок 3). Для безопасности второй секретный ключ частично замазан.

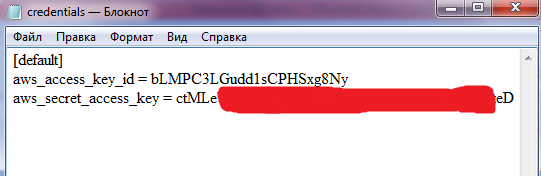


Рисунок 10. Файл credentials

1. После установки SDK загружаем пакет AWS, используя require (рисунок 4).

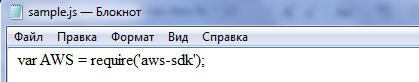


Рисунок 11. Загрузка пакета AWS

1. Загружаем учетные данные (credentials) и создаём клиента (рисунок 5).

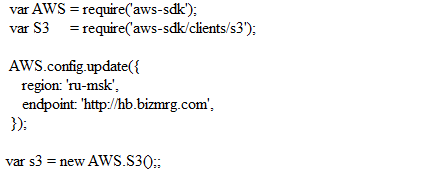


Рисунок 12. Загрузка учётных данных и создание клиента

1. Создав клиента S3, можем выполнять API-запросы:

* Получить тело объекта (рисунок 6)

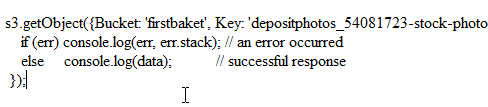


Рисунок 13. Код для получения тела объекта

* Положить объект (рисунок 7)

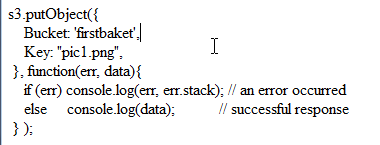


Рисунок 14. Код для загрузки объекта

На рисунке 8 видно, что без проблем загрузились объекты – pic1.png и example.js.

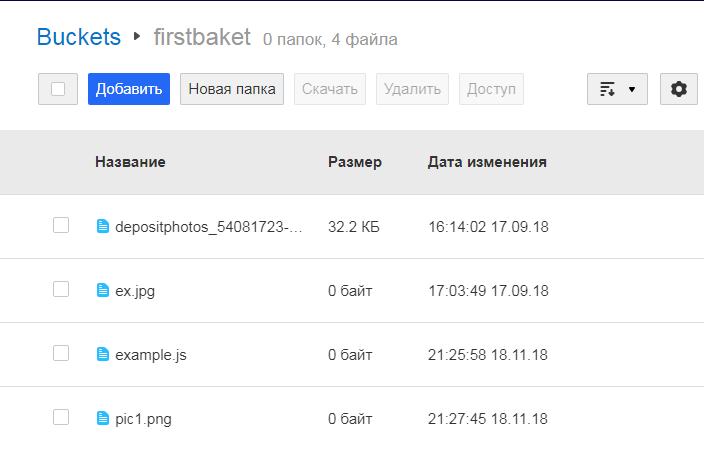


Рисунок 15. Результат работы sample.js – загрузка файла на mail.ru cloud solutions

На рисунке 9 можно увидеть результат работы sample.js. Заметим, что в sample.js в получении был указан файл «depositphotos\_540817…», на консоли видим Body этого объекта.

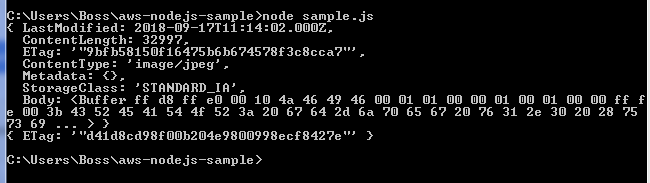


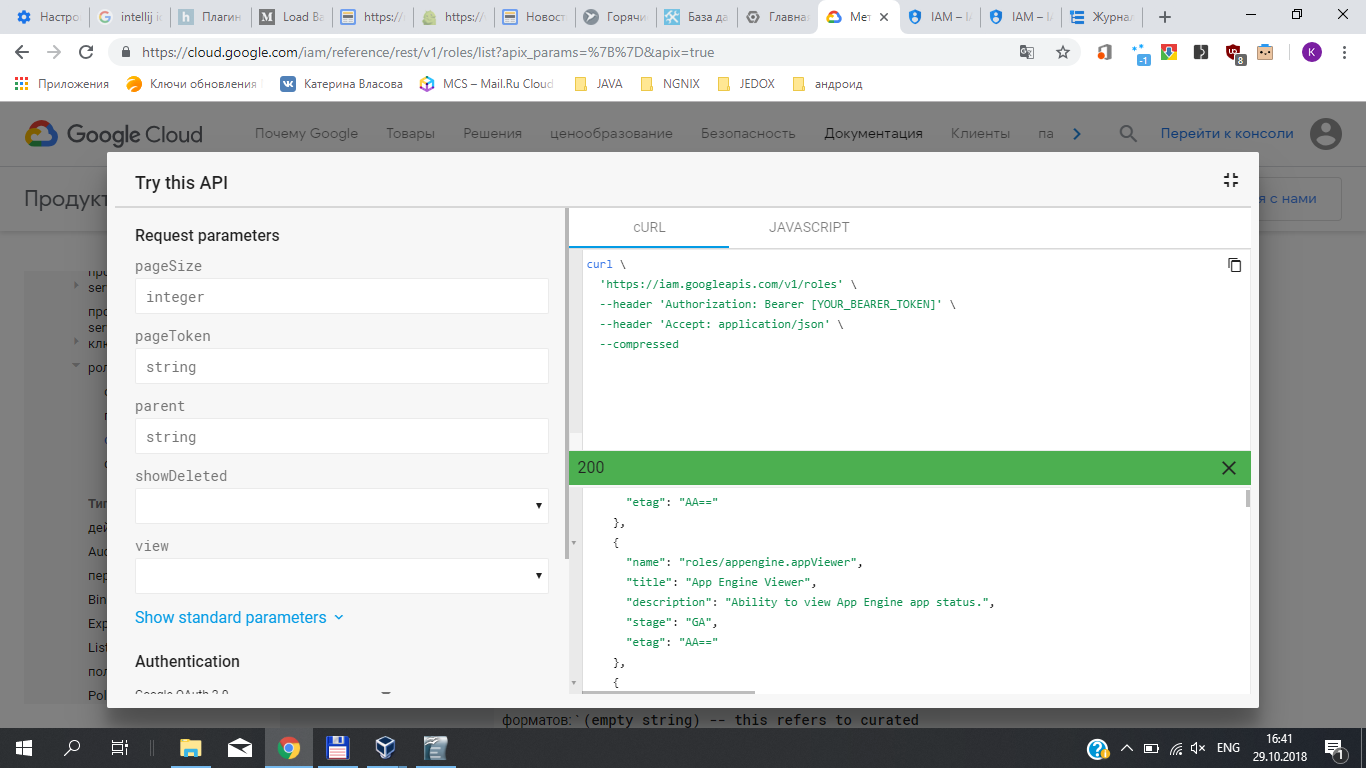
Рисунок 16. Результат работы sample.js – получение тела объекта с на mail.ru cloud solutions

# **Лабораторная работа №3**

**Задание:** реализовать веб-приложение с минимальным пользовательским интерфейсом, которое позволяет управлять созданием\удалением пользователей в  google cloud platfom.

**Цель:** получить навыки работы с протоколом управления пользователями и их полномочиями в google cloud platform.

**Ход работы:**

****

Javascript прописывается следующий код:

*<script src="https://apis.google.com/js/api.js"></script>*

*<script>*

*/\*\**

*\* Sample JavaScript code for iam.roles.list*

*\* See instructions for running APIs Explorer code samples locally:*

*\* https://developers.google.com/explorer-help/guides/code\_samples#javascript*

*\*/*

*function authenticate() {*

*return gapi.auth2.getAuthInstance()*

*.signIn({scope: "https://www.googleapis.com/auth/cloud-platform"})*

*.then(function() { console.log("Sign-in successful"); },*

*function(err) { console.error("Error signing in", err); });*

*}*

*function loadClient() {*

*return gapi.client.load("https://content.googleapis.com/discovery/v1/apis/iam/v1/rest")*

*.then(function() { console.log("GAPI client loaded for API"); },*

*function(err) { console.error("Error loading GAPI client for API", err); });*

*}*

*// Make sure the client is loaded and sign-in is complete before calling this method.*

*function execute() {*

*return gapi.client.iam.roles.list({})*

*.then(function(response) {*

*// Handle the results here (response.result has the parsed body).*

*console.log("Response", response);*

*},*

*function(err) { console.error("Execute error", err); });*

*}*

*gapi.load("client:auth2", function() {*

*gapi.auth2.init({client\_id: YOUR\_CLIENT\_ID});*

*});*

*</script>*

*<button onclick="authenticate().then(loadClient)">authorize and load</button>*

*<button onclick="execute()">execute</button>*

*{*

*"roles": [*

*{*

*"name": "roles/accesscontextmanager.policyAdmin",*

*"title": "Access Context Manager Admin",*

*"description": "Full access to policies, access levels, and access zones",*

*"stage": "BETA",*

*"etag": "AA=="*

*},*

*{*

*"name": "roles/accesscontextmanager.policyEditor",*

*"title": "Access Context Manager Editor",*

*"description": "Edit access to policies. Create, edit, and change access levels and access zones.",*

*"stage": "BETA",*

*"etag": "AA=="*

*},*

*{*

*"name": "roles/accesscontextmanager.policyReader",*

*"title": "Access Context Manager Reader",*

*"description": "Read access to policies, access levels, and access zones.",*

*"stage": "BETA",*

*"etag": "AA=="*

*},*

*{*

*"name": "roles/androidmanagement.user",*

*"title": "Android Management User",*

*"description": "Full access to manage devices.",*

*"stage": "GA",*

*"etag": "AA=="*

*},*

*{*

*"name": "roles/appengine.appAdmin",*

*"title": "App Engine Admin",*

*"description": "Full management of App Engine apps (but not storage).",*

*"stage": "GA",*

*"etag": "AA=="*

*},*

*{*

*"name": "roles/appengine.appViewer",*

*"title": "App Engine Viewer",*

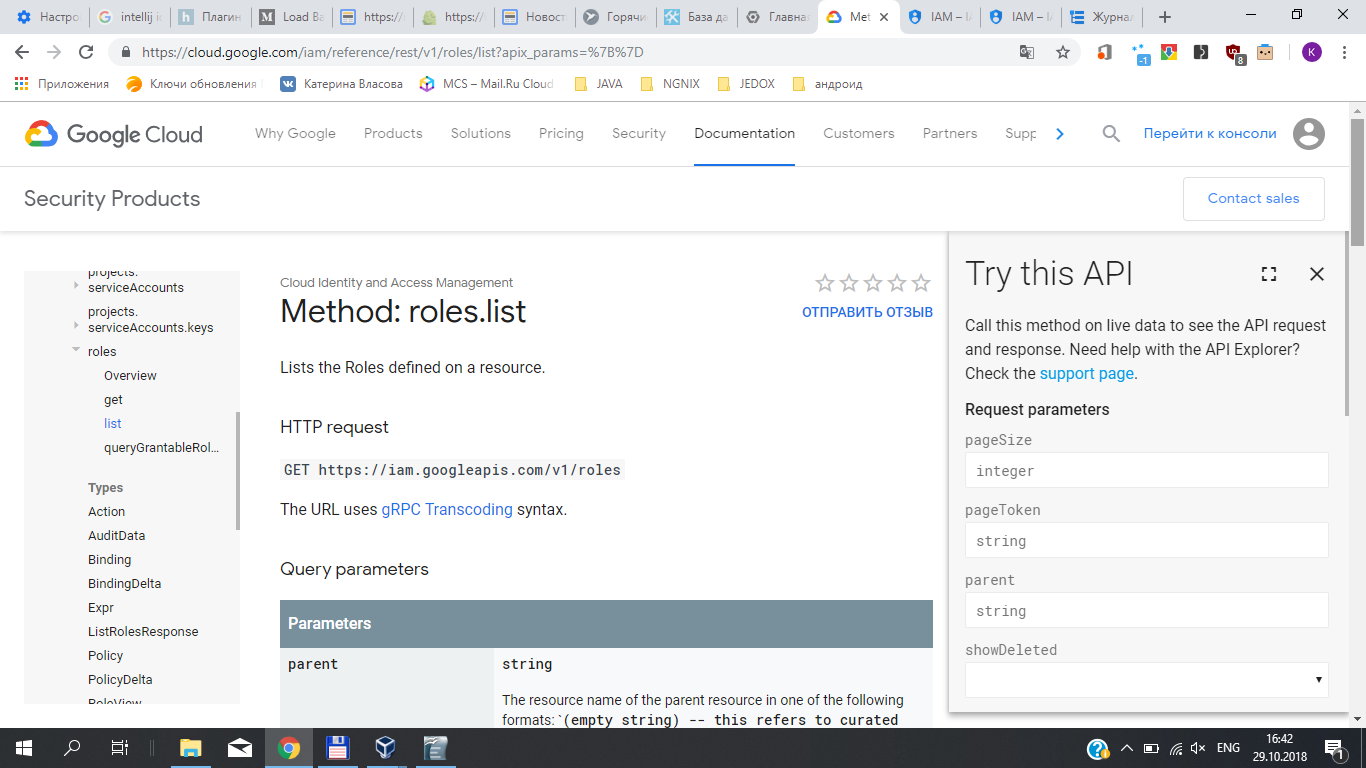
*"description": "Ability to view App Engine app status.",*

*"stage": "GA",*

*"etag": "AA=="*

*},*

*…......*

****

# **Тестирование нагрузки (подходы и инструменты)**

Существует два типа тестирования нагрузки:

* load testing («нагрузочное тестирование») - обозначает измерение производительности системы в условиях ожидаемой нормальной загруженности
* stress testing («стрессовое тестирование») - позволяет определить минимальные величины ресурсов, необходимых для работы приложения, оценить предельные возможности и найти «узкие места» системы. Кроме того, целью стрессового тестирования может являться обнаружение ошибок в серверном приложении, а также тестирование способности системы к сохранению целостности данных в аварийных ситуациях и к восстановлению после таких ситуаций.

Оба типа тестирования часто объединяют под общим наименованием performance testing («тестирование производительности»)

**Средства тестирования Web-серверов:**

Поведение виртуальных пользователей определяется сценариями, задающими порядок обращения к ресурсам, время задержки между запросами для одного пользователя, а также некоторые другие параметры.

1. Стандартные тестовые наборы.

В группу стандартных тестовых наборов входят SPECweb99 от Standard Performance Evaluation , WebBench компании eTestingLabs , WebStone, разработанный MindCraft, а также TPC-W от Transaction Processing Performance Council.

*Первые три пакета позволяют измерять производительность сервера в ответ на запросы статических HTML-страниц или динамического содержимого, генерируемого с помощью CGI-скриптов или API-интерфейса Web-сервера (ISAPI или NSAPI). Поддерживается обработка запросов с использованием SSL. TPC-W в большей мере предназначен для тестирования систем электронной коммерции и потому включает доступ к базам данных, а также позволяет оценить показатель цена/производительность.*

Общий подход, реализованный в рассматриваемых средствах, включает установку в тестируемой информационной системе стандартного содержимого — HTML-страниц, CGI-скриптов и т.д. Это позволяет получать унифицированные показатели производительности различных аппаратно-программных платформ и сравнивать их возможности между собой. В случаях, когда требуется оценка производительности и настройка конкретной информационной системы, такой подход неприменим.

1. Microsoft Web Application Stress Tool

Это относительно простое приложение для среды Windows распространяется бесплатно. Тестовый сценарий может быть создан вручную или записан с помощью Web-браузера и затем отредактирован. Для каждого запроса фиксируется запрашиваемый URL и время предварительного ожидания (delay, think time). Уровень нагрузки (stress level) регулируется путем задания количества нитей, осуществляющих запросы к серверу. Задается также число сокетов на нить (stress multiplier).

Среди возможностей Web Application Stress Tool: аутентификация виртуальных пользователей и пользовательских сеансов; передача данных с использованием протокола SSL; возможность группирования URL и задания относительной доли запросов для каждой группы; использование для тестирования Web-сервера нескольких централизованно управляемых клиентских машин. К недостаткам можно отнести отсутствие возможностей статистической обработки результатов тестирования.

1. OpenSTA

Приложение с открытым кодом для платформ Windows. Как и Web Application Stress Tool, оно позволяет создавать тестовые сценарии при помощи браузера на специализированном языке Script Control Language.

Поддерживается аутентификация пользователей и установления соединений по протоколу SSL. Основное внимание при разработке данного средства было уделено применению распределенной обработки и открытых стандартов, в частности CORBA. Распределенная обработка повышает реалистичность тестов и дает возможность тестирования высокопроизводительных систем, например, решений для электронной коммерции. В случае распределенной системы одна из машин (repository host) осуществляет сбор и хранение результатов; кроме того, на каждом из узлов тестирующей системы выполняется сервер имен OpenSTA name server. Каждый из узлов распределенной тестирующей системы выполняет свою группу заданий (task group), описываемую сценарием или набором сценариев. Результаты тестирования, включающие времена откликов и коды ответа для каждого запроса, представляются в табличной или графической форме.

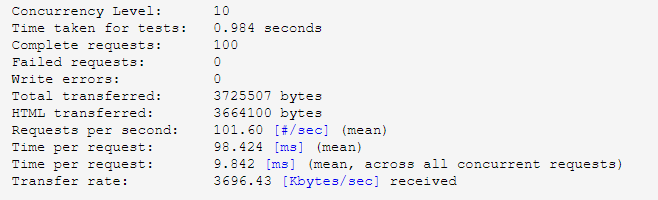
1. Apache HTTP server benchmarking tool

Бесплатный. Самый часто используемый, т.к входит в состав Apache.

*ab [options] [http[s]://]hostname[:port]/path* ,

где основные необходимые options: -c concurrency - количество одновременных запросов к серверу (по умолчанию 1); -n requests - общее количество запросов (по умолчанию 1).

В результате команды получаем такой отчет:



Плюсы:

* Есть везде, где есть Apache;
* Не требует никакой дополнительной настройки;
* Очень простой инструмент.

Минусы:

* Очень простой инструмент;
* Тестирует только производительность веб-сервера: опрашивает только один URL, не поддерживает сценарии нагрузки, невозможно имитировать пользовательскую нагрузку и оценить работоспособность проекта со всех сторон - как с точки зрения инфраструктуры, так и с точки зрения разработки.

1. Joe Dog Siege

Бесплатный. Немного сложнее ab и необходимые задачи выполняет гораздо лучше. В файле-сценарии задаются URL-ы и запросы тестирования. Если сценарий большой по объему, то можно вынести все запросы в отдельный файл и в команде указать этот файл при тестировании. В команде указывается количество пользователей , количество повторений и задержка между хитами.

Результат можно выводить в log-файл или сразу в консоль в режиме реального времени. Также можно взять из access-log веб-сервера URL-ы, по которым ходили реальные пользователи и эмулировать нагрузку реальных пользователей.

Плюсы:

* Многопоточный;
* Можно задавать как количество запросов, так и продолжительность (время) тестирования - т.е можно эмулировать пользовательскую нагрузку;
* Поддерживает простейшие сценарии

Минусы:

* Ресурсоемкий;
* Мало статистических данных и не очень хорошо эмулирует такие пользовательские сценарии, как ограничение скорости запросов пользователя;
* Не подходит для серьезного и масштабного тестирования в сотни и тысячи потоков, т.к он сам по себе ресурсоемкий, а на большом количестве запросов и потоков очень сильно нагружает систему.

1. Apache JMeter

Бесплатный. Основные возможности: написан на Java; HTTP, HTTPS, SOAP, Database via JDBC, LDAP, SMTP(S), POP3(S), IMAP(S); консоль и GUI; распределенное тестирование; план тестирования – XML-файл; может обрабатывать лог веб-сервера как план тестирования; визуализация результатов в GUI.

Результаты выводятся в графическом виде.

Плюсы:

* Кроссплатформенный, т.к написан на Java;
* Очень гибкий, используется много протоколов, не только веб-сервер, но и базы;
* Управляется через консоль и gui интерфейс;
* Использование напрямую логов веб-сервера Apache и Nginx в качестве сценария c возможностью варьирования нагрузки по этим профилям;
* Достаточно удобный и мощный инструмент.

Минусы:

* Ресурсоемкий;
* На длительных и тяжелых тестах часто падает по разным причинам;
* Стабильная работа зависит от окружения и конфигурации сервера.

1. Tsung

Бесплатный. Основные возможности: Основные возможности: написан на Erlang; HTTP, WebDAV, SOAP, PostgreSQL, MySQL, LDAP, Jabber/XMPP; консоль (GUI через сторонний плагин); распределенное тестирование (миллионы пользователей); фазы тестирования; план тестирования – XML; запись плана с помощью Tsung recorder’а; мониторинг тестируемых серверов (Erlang, munin, SNMP); инструменты для генерации статистики и графиков из логов работы.

С помощью собственных скриптов, которые обрабатывают логи работы, можно выводить различные отчеты по тестированию.

Плюсы:

* Т.к на писан на Erlang, то хорошо масштабируется, зависит от выделяемых ресурсов;
* Может выполнять роль распределенной системы, на большом количестве машин;
* Большое количество тестируемых систем - не только веб-серверы и БД, но и, к примеру, XMPP-сервер: может отправлять сообщения, тесты с авторизацией;
* Управление через консоль, но, благодаря поддержке плагинов, можно управлять и с помощью стороннего плагина с gui-интерфейсом;
* Наличие в комплекте инструмента Tsung Recorder - своего рода, proxy-сервер, через который можно ходить по тестируемому сайту и записывать сразу как профиль нагрузки;
* Генерация различных графиков тестирования с помощью скриптов.

Минусы:

* Нет gui-интерфейса;
* Только \*nix системы.

1. WAPT

Платный. Основные возможности: Windows; платный (есть триал на 30 дней / 20 виртуальных пользователей); запись плана тестирования из десктопных и мобильных браузеров; зависимости в планах тестирования (последующий URL в зависимости от ответа сервера); имитации реальных пользователей (задержки между соединениями, ограничение скорости соединений).

Отчет можно вывести как таблицей, так и графиком.

Плюсы:

* Очень гибкий, большое количество настроек и тестов;
* Эмуляция медленных каналов соединений пользователей;
* Подключение модулей;
* Запись сценариев тестирования прямо из браузера, как с десктопного, так и с мобильного;
* Генерация различных графиков тестирования с помощью скриптов.

Минусы:

* Доступен только для Windows;
* Платный.

1. PureLoad

Представляет собой Java-приложение и поддерживает платформы Windows, Sun Solaris, Red Hat Linux. Поддерживаются безопасные соединения с использованием SSL и пользовательские сеансы. Реалистичность имитации пользователей повышается путем обработки поступающих с сервера данных и генерации динамически изменяемых параметров. Управление системой осуществляется посредством консоли PureLoad Console.

Каждому виртуальному пользователю соответствует легковесный процесс, так называемая «рабочая нить» (worker thread). Виртуальные пользователи исполняют задания, объединенные в сценарии. Сценарии описываются в формате XML.

Для создания нагрузки PureLoad может использовать распределенные вычисления. Каждый из узлов вычислительной системы, осуществляющей тестирование сервера, исполняет «рабочий процесс» (worker process), которому соответствует копия виртуальной машины Java.

**Общие подходы к тестированию Web-серверов:**

В настоящее время применяются два способа обеспечения независимости запросов: использование многопоточности и создание распределенных систем.

Многопоточность в той или иной форме применяется во всех рассмотренных средствах тестирования. Данный метод не позволяет воспроизводить и в явной форме задавать характер потока запросов (равномерный, пульсирующий и т.д.) и его статистические характеристики (средняя величина интервала времени между последовательными запросами, дисперсия и др.). Величина выделяемого нити кванта времени (а следовательно, и величина интервала между запросами) зависит от производительности вычислительной системы и ее загруженности, а также от алгоритмов разделения времени, применяемых операционной системой. Непредсказуемые колебания интенсивности генерируемого потока запросов не позволяют достоверно оценить стабильность работы сервера. Увеличение числа нитей повышает уровень независимости запросов, однако конкуренция между нитями снижает уровень нагрузки, что делает данный подход неприменимым для тестирования высокопроизводительных серверов.

Применение распределенных вычислений позволяет добиться большей реалистичности тестирования, так как статистические характеристики генерируемого потока запросов приближаются к характеристикам, наблюдаемым в условиях эксплуатации сервера. Кроме того, становится возможным увеличение интенсивности нагрузки. Однако применение данного подхода предъявляет значительно более высокие требования к архитектуре системы и планированию тестов. В частности, необходимо распределять задания между узлами вычислительной системы и осуществлять централизованное управление, как это видно на примере PureLoad.

Так, при выборе инструмента тестирования производительности нужно помнить, что процесс настройки Web-сервера с целью достижения оптимальных параметров функционирования может потребовать многократного прогона тестов, поэтому система тестирования должна обладать возможностями по обработке результатов тестов, их хранению и сравнению. Уровни этих возможностей для рассмотренных систем различаются.