Есть очень удобный сервис для тестирования скорости работы сайта — webpagetest.org. Мало того, что он самый удобный и информативный из всех, что я знаю, так его еще и локально можно установить себе на сервер. Этим я и займусь — установкой и настройкой локальной версии сервиса webpagetest для тестирования скорости работы сайта.

Содержание:

Введение

Любую работу по оптимизации сайта и сервера, на котором он работает, я начинаю с проведения тестов сайта в его текущем виде. Без этого невозможно оценить результат. Нужно всегда понимать, движешься ты в верном направлении или нет. Приведу простой пример из своего опыта, с которым пришлось столкнуться.

Был сайт, который работал на HTTP/1.1 с разделением самого сайта и картинок по разным доменам. Сайт на одном домене, картинки на поддомене. Сделано это было для того, чтобы побыстрее грузить сайт. Если кто не знает, то напомню, что протокол HTTP/1.1 мог одновременно устанавливать только 5 соединений. Вывод картинок на отдельный поддомен ускорял загрузку.

Я решил перевести сайт на HTTP/2.0 и разместить на одном домене. Протокол HTTP/2.0 снимал ограничение на количество подключений, так что весь сайт начинал грузиться сразу. Дополнительно появлялись приоритеты загрузки для разных типов данных, что теоретически тоже ускоряло загрузку сайта. Разделение на поддомены стало не нужным, а оно усложняло управление и администрирование, поэтому логичным было сделать сайт опять единым. Я

провел начальные тесты сайта на HTTP/1.1, потом перевел его на HTTP/2.0 и объединил домены.

По факту оказалось, что сайт стал грузиться чуть медленнее на большей части ключевых страниц. Суммарное время загрузки страницы уменьшилось, так что вроде бы все нормально, новый протокол реально ускорил загрузку сайта. Но вот начальная отрисовка и появление читабельной версии сайта стали медленнее. Я был удивлен таким раскладом, но когда сравнил внимательно результаты webpagetest до и после перехода, стало все ясно.

При разделении сайта на поддомены лесенка запросов и загрузок складывалась более оптимально, так что первый контент появлялся раньше, чем то же самое на HTTP/2.0. Это была **особенность конкретного сайта**, его структуры и верстки. В общем случае так быть не должно, а тут было именно так. Я пытался вручную скорректировать загрузку, используя современные механизмы HTTP/2.0 типа **server push**, но сделать результат лучше, чем был на HTTP/1.1 так и не получилось. В итоге просто откатился обратно.

Я привел этот случай для примера, зачем нужен webpagetest. Без него мне бы и в голову не пришло, что переход на HTTP/2.0 даст отрицательный результат. Теперь перед каждым изменением сайта, я внимательно тестирую ключевые страницы и сравниваю, как было и как стало. То же самое сделал при переходе на tls 1.3 и сжатие brotli. Но результат оценить не смог. Установленная версия webpagetest не поддерживает tls 1.3 и brotli. По заголовкам увидел, что использует gzip и tls 1.2. Надо внимательно разбираться с настройками и версиями используемых браузеров.

Краткая история, зачем все это нужно. Как пользоваться webpagetest рассказывать не буду, так как в интернете полно информации, в том числе с видео. Переходим к практической части.

Установка docker на Ubuntu 18

Далее расскажу, как установить свою персональную локальную версию webpagetest для регулярного использования. Это может быть нужно по нескольким причинам:

- Вы часто делаете тесты и вас не устраивает очередь ожидания в публичном сервисе.
- Вы хотите сами выбирать, где располагать тестовые локации с агентом. Очевидно, что они должны быть там же, где целевая аудитория сайта.

Сервер webpagetest состоит из следующих компонентов:

1. Сервер с веб интерфейсом.
2. Агент с браузерами для тестирования.

Отдельно устанавливается сервер с веб интерфейсом и отдельно агенты для тестирования. Сначала рассмотрю самый простой случай, когда агент и сервер установлены через **docker контейнер** на одном сервере. Для персонального использования этого будет вполне достаточно.

Если вы планируете более масштабное использование, то следите за тем, чтобы было достаточно ресурсов у агента. Если он будет тормозить по какой-то причине, результаты тестов будут искажены. Я лично это наблюдал, когда разворачивал на тестовом сервере и нагружал его посторонними задачами.

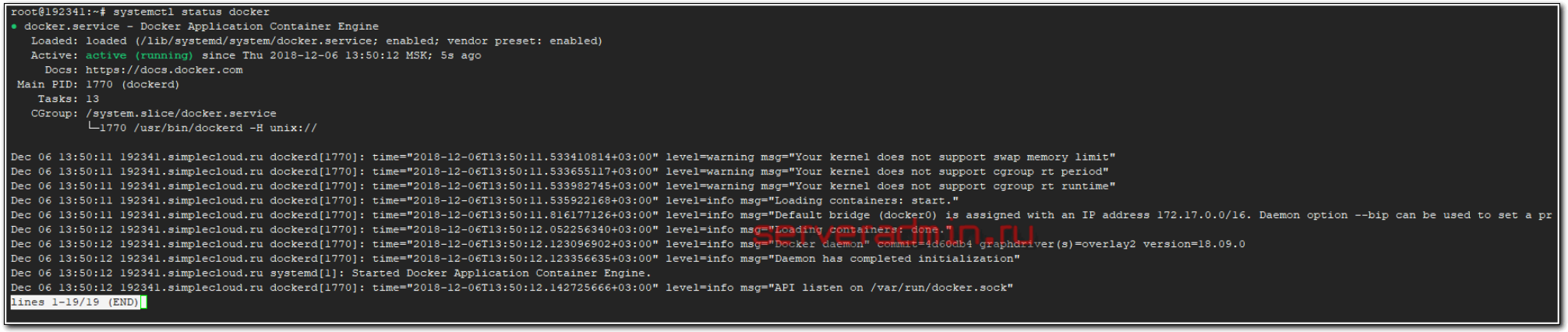
Как я уже сказал ранее, сервер и клиент будут в docker. Хостовая система будет **Ubuntu 18**, как наиболее подходящая для работы с докером. Но в целом это не важно, можете разворачивать на любой системе, где работает docker.

Если он у вас уже установлен, пропустите этот раздел. Установка docker на Ubuntu 18:

```
# apt update && apt upgrade -y
# apt install apt-transport-https ca-certificates curl software-properties-common
# curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
# add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu bionic stable"
# apt update
# apt-cache policy docker-ce
# apt install docker-ce
```

Проверяем работу docker:

```
# systemctl status docker
```



Локальная установка webpagetest

Самый простой вариант установки — использовать готовый образ. В нем предустановлено использование одного агента для тестирования. Все базовые конфиги подготовлены. Если агентов нужно больше, то надо будет вынести конфиги из образа и сделать вручную некоторые настройки. Сейчас выполним простую установку, а дальше я расскажу, как вынести конфиги и полностью управлять настройками.

Скачиваем официальные образы из docker hub:

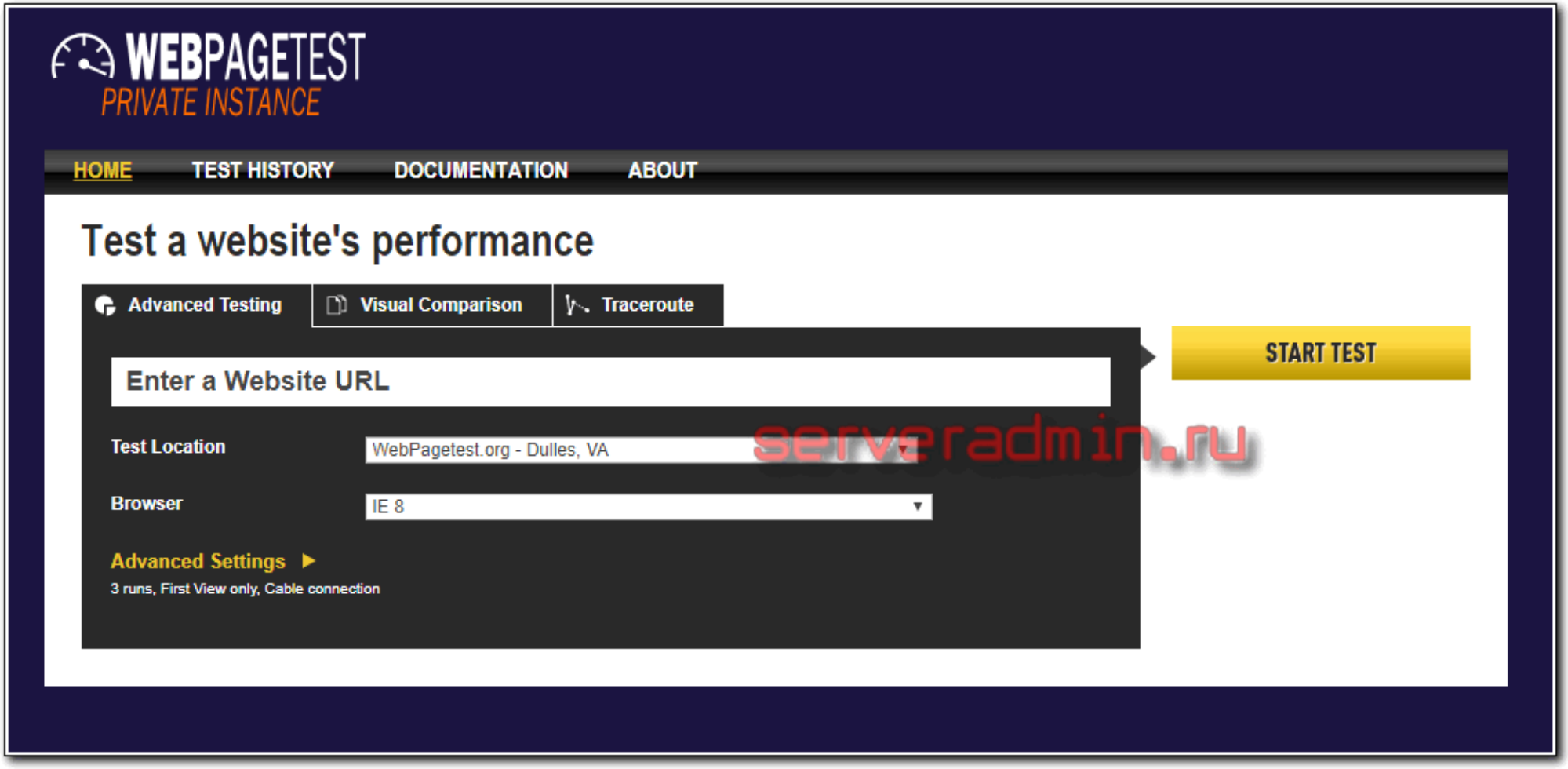
```
# docker pull webpagetest/server
# docker pull webpagetest/agent
```

Подробную информацию об образах можете посмотреть на docker hub — https://hub.docker.com/u/webpagetest/ .

Запускаем сервер webpagetest:

```
# docker run -d -p 80:80 --restart=always --name wpt-server webpagetest/server
```

Проверьте, что у вас 80-й порт никто не занимает. Если он занят, то используйте другой. И не забудьте настроить или отключить firewall. Через пару минут после запуска, переходите по адресу *http://89.223.28.57/* , где 89.223.28.57 — ip адрес сервера. Увидите веб интерфейс локальной версии webpagetest.



Теперь пройдите по адресу *http://89.223.28.57/install/* и проверьте выполнение всех основных условий работы. Должно быть примерно так.

PHP

- ✓ PHP version at least 5.3: **5.6.38**
- ✓ GD Module Installed: **yes**
- ✓ FreeType enabled for GD (required for video rendering): **yes**
- ✓ zip Module Installed: **yes**
- ✓ zlib Module Installed: **yes**
- ✓ mbstring available: **yes**
- ✓ curl Module Installed: **yes**
- ✓ php.ini allow_url_fopen enabled: **yes**
- ⚠ APC Installed: **NO** (optional)
- ✓ SQLite Installed (for editable test labels): **yes**
- ✓ Open SSL Module Installed (for "Login with Google"): **yes**
- ✓ xml Module Installed (for rss feeds): **yes**
- ✓ pcre Module Installed (for rss feeds): **yes**
- ✓ xmlreader Module Installed (for rss feeds): **yes**
- ✓ php.ini upload_max_filesize > 10MB: **20M**
- ✓ php.ini post_max_size > 10MB: **20M**
- ✓ php.ini memory_limit > 256MB or -1 (disabled): **512M**

System Utilities

- ✓ ffmpeg Installed with --enable-libx264 (required for video): **yes**
- ✓ ffmpeg Installed with scale and decimate filters(required for mobile video): **3.2.12-1~deb9u1,scale,mpdecimate**
- ✓ imagemagick compare Installed (required for mobile video): **yes**
- ✓ jpegtran Installed (required for JPEG Analysis): **yes**
- ✓ exiftool Installed (required for JPEG Analysis): **yes**

Misc

- ⚠ python 2.7 with modules (faster mobile video processing): **Missing python modules: SSIM** (optional)

Filesystem

- ✓ {docroot}/tmp writable: **yes**
- ✓ {docroot}/dat writable: **yes**
- ✓ {docroot}/results writable: **yes**
- ✓ {docroot}/work/jobs writable: **yes**
- ✓ {docroot}/logs writable: **yes**
- ⚠ {docroot}/tmp on tmpfs: **NO** (optional)

Test Locations

- ✗ Test_loc : Test Location
  - ✗ IE : Test Location - IE 8 - **No Agents Connected**
  - ✗ Test : Test Location - **No Agents Connected**
- ✗ Public_Dulles : WebPagetest.org - Dulles, VA
  - ✗ WPT_Dulles_IE8 : WebPagetest.org Dulles, VA - IE8 - **No Agents Connected**
- ✗ Appurify : Appurify Mobile Testing
  - ✗ Appurify_Mobile : Appurify Mobile - **No Agents Connected**

Все, необходимые для работы условия, выполнены. Теперь запустим агента. Не обязательно это делать на этом же сервере. Можете запустить на другом, только правильно укажите адрес сервера. Перед запуском агента, подгружаем модуль ядра для корректной работы ограничения скорости для теста.

```
# modprobe ifb numifbs=1
# docker run -d --network="host" --restart=always -e "SERVER_URL=http://localhost/work/" -e "LOCATION=Test" --cap-
add=NET_ADMIN --name wpt-agent webpagetest/agent
```

Подождите пару минут и проверяйте страничку *http://89.223.28.57/install/* , вы должны увидеть подключившегося агента.

На этом базовая установка локального сервера webpagetest готова. Можно начинать тесты. В качестве Test Location выбирайте Test Location (тавтология получилась), указывайте остальные параметры и проверяйте сайт.

Этой установки вполне достаточно для собственных нужд. Если же вам необходимо использовать кастомные настройки и добавлять агентов с разными настройками, то читайте дальше, как собрать свой собственный образ webpagetest сервера и агента.

Настройка своего сервера webpagetest

Создадим свои docker образы, на основе официальных, немного их изменив, вынеся конфигурации на хостовую машину. Для этого создадим 2 директории:

```
# mkdir ~/wptserver && mkdir ~/wptagent
```

Создадим dockerfile и конфиг для сервера:

```
# cd ~/wptserver && touch Dockerfile locations.ini
```

Содержимое Dockerfile:

```
FROM webpagetest/server
ADD locations.ini /var/www/html/settings/
```

Содержимое locations.ini:

```
[locations]
1=Test_loc
[Test_loc]
1=Test
label=Test Location
group=Desktop
[Test]
browser=Chrome,Firefox
label="Test Location"
```

Это базовая настройка для location, которая идет в дефолте. Сюда вы можете добавлять свои агенты, устанавливать их параметры. Подробнее об этом читайте в документации.

Приведенные настройки образа призывают использовать оригинальный образ, заменив файл locations.ini на тот, что вы создали. Собираем образ с сервером:

```
# docker build -t local-wptserver .
```

Делаем то же самое для агента.

```
# cd ~/wptagent && touch Dockerfile script.sh
```

Содержимое Dockerfile:

```
FROM webpagetest/agent
ADD script.sh /
ENTRYPOINT /script.sh
```

Содержимое script.sh:

```
#!/bin/bash
set -e
if [ -z "$SERVER_URL" ]; then
  echo >&2 'SERVER_URL not set'
  exit 1
fi
if [ -z "$LOCATION" ]; then
  echo >&2 'LOCATION not set'
  exit 1
fi
EXTRA_ARGS=""
if [ -n "$NAME" ]; then
  EXTRA_ARGS="$EXTRA_ARGS --name $NAME"
fi
python /wptagent/wptagent.py --server $SERVER_URL --location $LOCATION $EXTRA_ARGS --xvfb --dockerized -vvvvv
```

В последней строке указываются параметры запуска агента. Сейчас представлены дефолтные настройки, но вы их сможете изменять при необходимости. Подробнее об этом написано на github.

Делаем скрипт исполняемым и собираем образ:

```
# chmod u+x script.sh
# docker build -t local-wptagent .
```

Запускаем сервер и агент из наших новых образов. Сначала сервер:

```
# docker run -d -p 80:80 --restart=always --name wpt-server local-wptserver
```

Ждем немного и проверяем web интерфейс. Если все OK, запускаем агента.

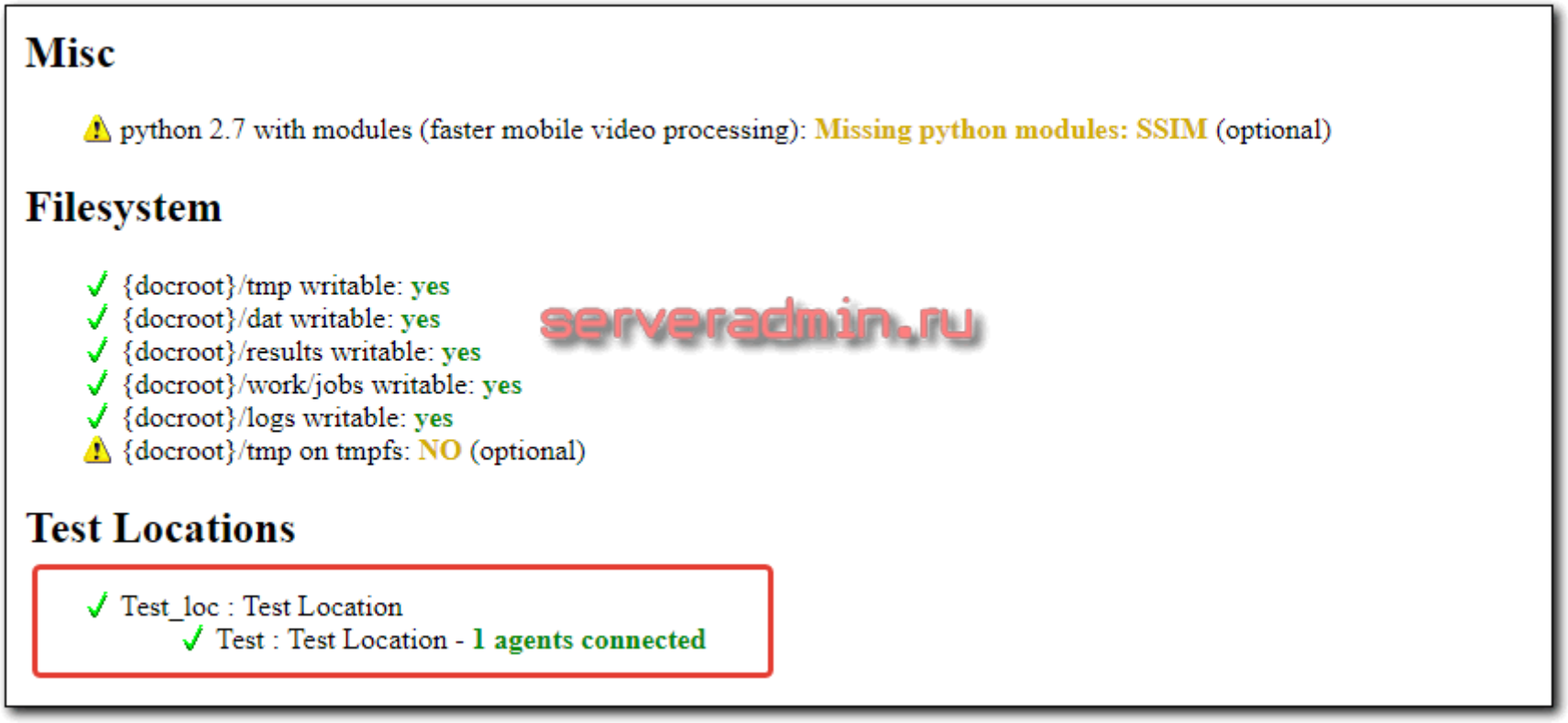
```
# docker run -d --network="host" --restart=always -e "SERVER_URL=http://localhost/work/" -e "LOCATION=Test" --cap-
add=NET_ADMIN --name wpt-agent local-wptagent
```

Проверьте на всякий случай, все ли правильно запустилось:

```
# docker ps
```

```
root@192341:~/wptagent# docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS                          NAMES
85973a900890        local-wptagent      "/bin/sh -c /script.…"   4 seconds ago       Up 3 seconds                                       wpt-agent
54df0606364b        local-wptserver     "docker-php-entrypoi…"   About a minute ago  Up About a minute   0.0.0.0:80->80/tcp, 443/tcp    wpt-server
root@192341:~/wptagent#
```


Проверьте, подключился ли агент.



Все верно. Одну локацию мы указали, ее и получили. Теперь вы сможете добавлять новые локации с новыми агентами и разными настройками.

На этом все. Вопрос локальной установки webpagetest я раскрыл.

Заключение

Не понравилась статья и хочешь научить меня администрировать? Пожалуйста, я люблю учиться. Комментарии в твоем распоряжении.

> Расскажи, как сделать правильно!

Я выбрал установку из контейнера из-за простоты и удобства. Здесь как раз тот случай, где docker оправдывает себя на все 100%. Я собирал все то же самое из пакетов, у меня даже работало, но это занимало в разы больше времени. Система многокомпонентная с огромным числом зависимостей и сервисов. Все это настраивать вручную очень хлопотно. Есть скрипт автоматической установки, но с ним тоже есть проблемы. Я немного повозился просто из любопытства. В итоге понял, что с докером проще, как в установке, так и в обновлении сервера.

При желании, к системе можно прикрутить https, авторизацию через basic auth. Не стал это рассматривать отдельно, так как тема стандартная. При желании, каждый сможет сам настроить. Под капотом у webpagetest обычный nginx в качестве веб сервера. Еще полезная ссылка — https://github.com/WPO-Foundation , профиль компании на github со всем софтом и инструкциями.

В целом, проект Webpagetest очень интересный и полезный, но на удивление, слабо освещен в интернете. Практически не нашел никакой информации. Я внимательно погуглил тему, но в итоге почти во всем разбирался сам с помощью инструкций на github и документации. В заключении маленький и полезный нюанс. По-умолчанию, webpagetest тестирует сайт в разрешении 1024 на 768. Не самое популярное на текущий момент. Я не стал ковыряться в потрохах сервера, чтобы изменить этот параметр, а поступил проще. Для установки разрешения экрана, просто добавьте следующий код на вкладке **Advansed settings -> Script**:

```
navigate https://serveradmin.ru
setViewportSize 1920 1080
```

Все параметры, которые могут быть использованы в скриптах, описаны отдельно в документации. На этом у меня все. В будущем будут еще статьи на тему ускорения сайтов.

Помогла статья? Есть возможность отблагодарить автора