

# Развертывание проекта



Азамат  
Искаков



## **Азамат Искаков**

ИТ архитектор в Expertonica LLP





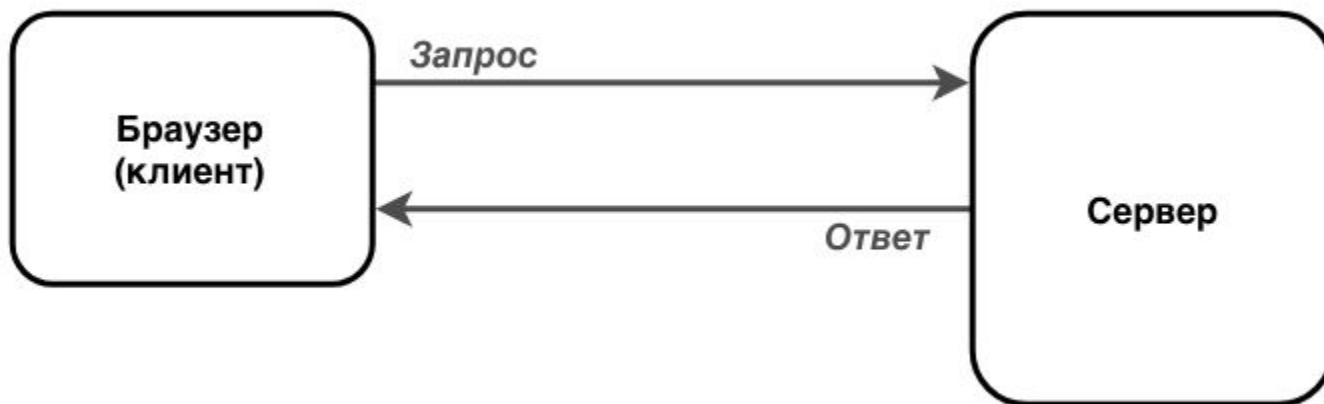
# План занятия

1. [Связывание хостинга файлов и запуска веб-приложения](#)
2. [Использование различных дополнительных файлов для работы веб-приложения](#)
3. [Работа со статикой](#)
4. [Деплой на собственном сервере](#)
5. [Деплой на платформах](#)

---

# **Связывание хостинга файлов и запуска веб- приложения**

# Веб-сервер



## Зачем отдельный веб-сервер

	manage.py runserver	webserver+wsgi	webserver+asgi	облачное решение
Удобство разработки	+			
Скорость		+	+	+
Безопасность		+	+	+
Использование в production		+	+	+
Легкость обслуживания				+

---

# Основные веб-серверы (On-premise)

- Nginx Web Server
- Apache HTTP Server
- Caddy
- Lighttpd Web Server

---

# NGINX

Быстрый веб-сервер/http-прокси

## Установка:

```
$ sudo apt-get install nginx  
$ sudo service nginx start  
$ wget localhost:80 -O -  
Welcome to nginx
```

Логи запросов и ошибок: `/var/log/nginx/`

Конфигурационные файлы (`nginx.conf` и дополнительные):

- `/etc/nginx/`
- `/usr/local/nginx/conf`
- `/usr/local/etc/nginx`



---

# Управление nginx

```
$ nginx -s сигнал
```


Где сигнал может быть одним из нижеследующих:

- `stop` – быстрое завершение;
- `quit` – плавное завершение;
- `reload` – перезагрузка конфигурационного файла;
- `reopen` – переоткрытие лог-файлов.

# Конфигурация nginx

Конфигурация nginx представляет из себя набор директив, в том числе вложенных друг в друга (то есть в контекст):

```
events {  
    ...  
}  
http {  
    # default server  
    server {  
        listen 80;  
        location / {  
            root /data/www;  
        }  
    }  
}
```



# **Использование различных дополнительных файлов для работы веб- приложения**

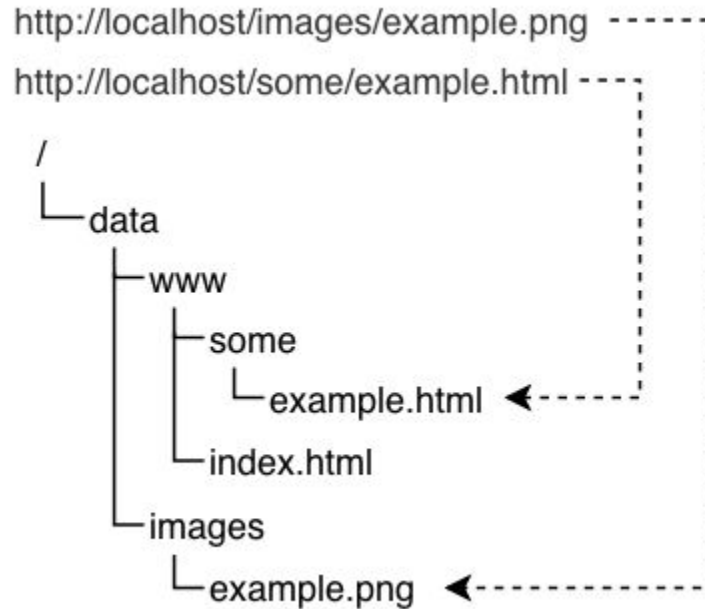
# Раздача статического содержимого

Например, у нас есть такая конфигурация nginx (обратите внимание на директивы `location`):

```
http {  
    server {  
        listen 80;  
        location / {  
            root /data/www;  
        }  
        location /images/ {  
            root /data;  
        }  
    }  
}
```

# Раздача статического содержимого

Тогда следующие URL будут получать указанные файлы:



---

## Раздача статического содержимого

При изменении конфигурации nginx, нужно её применить.

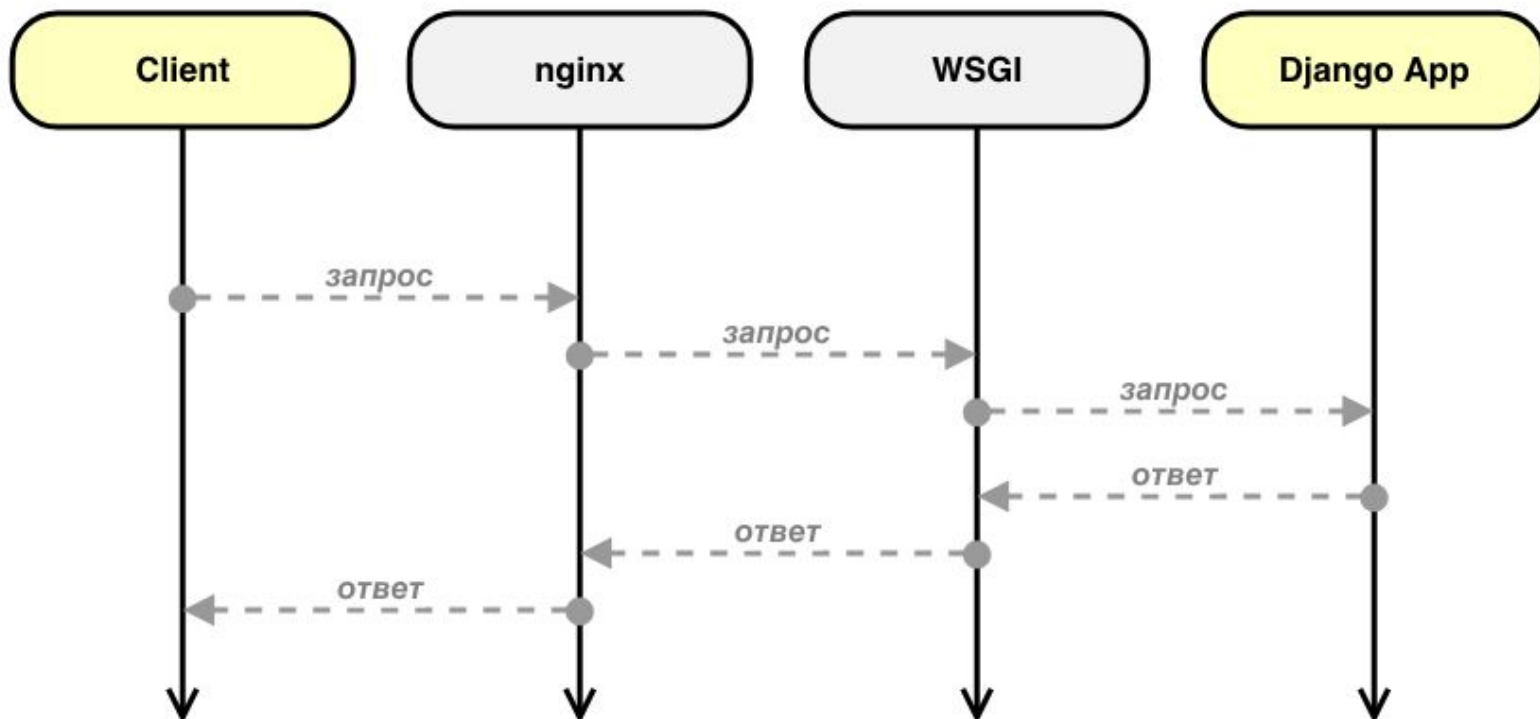
Для этого отправим уже запущенному nginx сигнал `reload`:

```
$ nginx -s reload
```

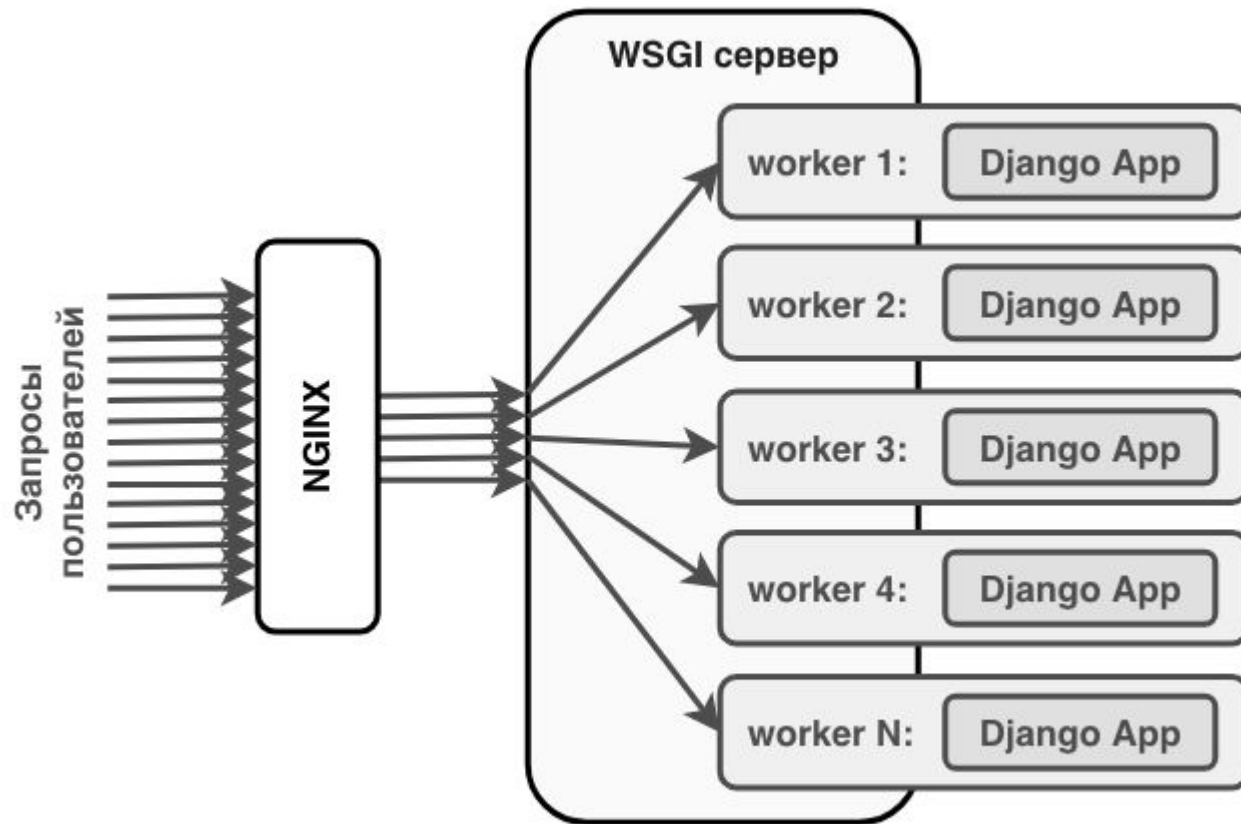
# WSGI

WSGI (Web Server Gateway Interface) – интерфейс шлюза web-сервера.

Упрощенная схема прохождения запроса от клиента(браузера) к приложению Django:



# Зачем wsgi-сервер





---

# gunicorn

Де-факто стандартный WSGI сервер для python - [gunicorn](https://docs.gunicorn.org/en/latest/run.html).



Чтобы запустить ваш Django проект необходимо в корне проекта ввести команду (работает только в UNIX системах):

```
$ pip install gunicorn
```

```
$ gunicorn <пакет с файлом wsgi>.wsgi -b 0.0.0.0:8000
```

Например:

```
$ gunicorn my_youtube.wsgi -b 0.0.0.0:8000
```

Подробнее: <https://docs.gunicorn.org/en/latest/run.html>



# Работа со статикой

# Статические файлы

- Добавим `django.contrib.staticfiles` в `INSTALLED_APPS`
- Укажем `STATIC_URL`

`settings.py`

```
DEBUG = True

INSTALLED_APPS = [
    ...
    'django.contrib.staticfiles',
    ...
]

STATIC_URL = '/static/'
```

# Статические файлы в продакшене

До сих пор наши настройки статики хорошо работали в дебаге (DEBUG = True), но в продакшене (DEBUG = False) все сломается. Почему так?

Дело в том, что в дебаге Django берет на себя заботу о статике, это удобно - нам не надо дополнительные инструменты устанавливать, запускать и настраивать, но Django работает со статикой медленно.

В продакшене статикой будет заниматься nginx. Для этого статику для nginx надо подготовить - говорят “собрать” статику.

```
STATIC_ROOT = os.path.join(BASE_DIR, 'static')
```

STATIC\_ROOT - переменная в settings.py, которая говорит Django, куда же надо собрать статику.

# Статические файлы в продакшене

Настройка сбора статики сделана, но как же собрать статику?

Для этого в Django есть специальная команда:

```
$ python manage.py collectstatic
```

```
168 static files copied to '.../static'
```

# Один сервер для приложения и для статики

Пути для статики в `settings.py`:

```
STATIC_URL = '/static/'  
STATIC_ROOT = os.path.join(BASE_DIR, 'static')
```

Соберем всю статику на сервере в одном каталоге:

```
$ python manage.py collectstatic
```

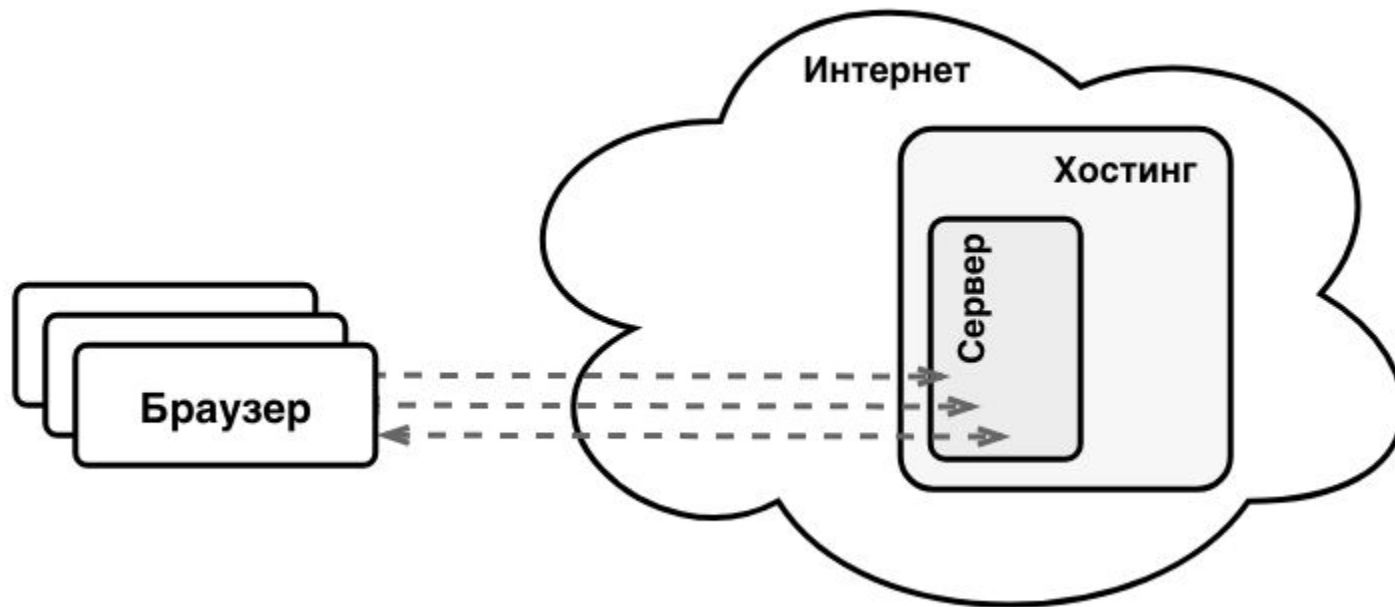
Укажем путь к статике для nginx:

```
server {  
    ...  
    location /static/ {  
        root путь_до_проекта_равный_BASE_DIR;  
    }  
}
```



# Деплой на собственном сервере

# Сервер и хостинг





# Переменные окружения

Как быть с настройками DEBUG и SECRET\_KEY? Ведь при разработке они одни, а на боевом сервере - другие. Да и другие настройки (пароли, токены и пр.) могут быть одними при разработке и другими в продакшене.

Как раз для таких случаев и нужны переменные окружения:

```
import os

SECRET_KEY = os.getenv('SECRET_KEY', default='the-best-secret-key')
DEBUG = os.getenv('DEBUG', default='True') == 'True'
```

# Запуск Django проекта

В настройках надо указать `ALLOWED_HOSTS = '*'`, чтобы разрешить обращения к нашему серверу со всех адресов. Более точно про эту настройку можно почитать в документации:

<https://docs.djangoproject.com/en/3.2/ref/settings/#allowed-hosts>

```
$ python manage.py migrate
$ python manage.py collectstatic
$ DEBUG=False SECRET_KEY=some-secret gunicorn
my_project.wsgi -b 0.0.0.0:8000
```

# Настройка nginx

Настроим поведение nginx с помощью файла  
/etc/nginx/sites-available/default

```
server {  
    listen 80;  
  
    server_name ip_вашего_сервера;  
  
    location /static/ {  
        root путь_до_проекта_равный_STATIC_ROOT;  
    }  
  
    location / {  
        proxy_pass http://127.0.0.1:8000;  
    }  
}
```

## Резюме

1. Подготовить проект (сделать настройки в settings.py)
2. Скачать подготовленный проект на сервер (git clone ...)
3. Установить все зависимости (pip install -r requirements.txt)
4. Сделать миграции (предварительно установить и настроить БД при необходимости) (python manage.py migrate)
5. Собрать статику (python manage.py collectstatic)
6. Установить gunicorn (если еще не установлен) (pip install gunicorn)
7. Запустить проект (DEBUG=False ... gunicorn [my\\_proj.wsgi](#) -b 0.0.0.0:8000)
8. Проверить доступность через браузер по адресу [http://ip\\_вашего\\_сервера:8000](http://ip_вашего_сервера:8000) (может выглядеть страшно, так как статика еще не раздается)
9. Установить nginx (apt install nginx)
10. Сделать настройку nginx (файл /etc/nginx/sites-available/default)
11. Перезагрузить nginx (nginx -s reload)
12. Убедиться в доступности сайта по адресу [http://ip\\_вашего\\_сервера](http://ip_вашего_сервера)



# Деплой на платформах

---

# Основные cloud серверы для приложений Django

- Google Cloud App Engine
- AWS Elastic Beanstalk
- Microsoft Azure Web App
- IBM Cloud App Service
- Oracle Application Container Cloud Service

---

# Как выбрать облачного провайдера для своего сайта

- Задержка отклика в мс в целевом регионе
- Поддержка Django приложений
- Легкость настройки и запуска
- Средства мониторинга
- Возможности масштабирования и балансировки
- Бесплатные функции и объем бесплатных средств \$ для новичков
- Цена

# Пример запуска Django приложения на Google Cloud App Engine

Google Cloud Platform isb-2007-bot

Search products and resources

App Engine Dashboard

LEARN SHOW INFO PANEL

⚠ Your app is currently disabled [ENABLE IN SETTINGS](#)

Version: 20191219t003851 (100%) [isb-2007-bot.ew.r.appspot.com](#)  
Region: europe-west

Summary

Count/sec

RESET ZOOM 1 hour 6 hours 12 hours 1 day 2 days 4 days 7 days 14 days 30 days

⚠ No data is available for the selected time frame.

CLOUD SHELL Terminal (isb-2007-bot) x +

Open Editor

```
Welcome to Cloud Shell! Type "help" to get started.
Your Cloud Platform project in this session is set to isb-2007-bot.
Use "gcloud config set project [PROJECT_ID]" to change to a different project.
a_iskakov1989@cloudshell:~ (isb-2007-bot)$
```



# Настройка app.yaml

```
# [START django_app]
runtime: python38

handlers:
# This configures Google App Engine to serve the files in the app's static
# directory.
- url: /static
  static_dir: static/

# This handler routes all requests not caught above to your main app. It is
# required when static routes are defined, but can be omitted (along with
# the entire handlers section) when there are no static files defined.
- url: /*
  script: auto
# [END django_app]
```

# Настройка app.yaml

Настроить requirements.txt проекта.

Развернуть приложение в облаке:

```
python manage.py collectstatic
```

```
gcloud app deploy
```

Проверить работу приложения по адресу:

[https://PROJECT\\_ID.REGION\\_ID.r.appspot.com](https://PROJECT_ID.REGION_ID.r.appspot.com)

---

## Домашнее задание

- Выполнить оставшиеся домашние задания.
- Повторить материал модуля.
- Приступить к выполнению дипломного задания!

**Задавайте вопросы и  
пишите отзыв о лекции!**

**Азамат Искаков**