Deployment checklist¶

The Internet is a hostile environment. Before deploying your Django project, you should take some time to review your settings, with security, performance, and operations in mind.

Django includes many security features. Some are built-in and always enabled. Others are optional because they aren't always appropriate, or because they're inconvenient for development. For example, forcing HTTPS may not be suitable for all websites, and it's impractical for local development.

Performance optimizations are another category of trade-offs with convenience. For instance, caching is useful in production, less so for local development. Error reporting needs are also widely different.

The following checklist includes settings that:

- must be set properly for Django to provide the expected level of security;
- are expected to be different in each environment;
- enable optional security features;
- enable performance optimizations;
- provide error reporting.

Many of these settings are sensitive and should be treated as confidential. If you're releasing the source code for your project, a common practice is to publish suitable settings for development, and to use a private settings module for production.

Run `manage.py check --deploy`¶

Some of the checks described below can be automated using the **`--deploy`** option of the **`check`** command. Be sure to run it against your production settings file as described in the option's documentation.

Critical settings¶

`SECRET_KEY`¶

The secret key must be a large random value and it must be kept secret.

Make sure that the key used in production isn't used anywhere else and avoid committing it to source control. This reduces the number of vectors from which an attacker may acquire the key.

Instead of hardcoding the secret key in your settings module, consider loading it from an environment variable:

```
import os

SECRET_KEY = os.environ['SECRET_KEY']
```

or from a file:

```
with open('/etc/secret_key.txt') as f:
```

```
    SECRET_KEY = f.read().strip()
```

DEBUG¶

You must never enable debug in production.

You're certainly developing your project with **`DEBUG = True`**, since this enables handy features like full tracebacks in your browser.

For a production environment, though, this is a really bad idea, because it leaks lots of information about your project: excerpts of your source code, local variables, settings, libraries used, etc.

Environment-specific settings¶

ALLOWED_HOSTS¶

When **`DEBUG = False`**, Django doesn't work at all without a suitable value for **`ALLOWED_HOSTS`**.

This setting is required to protect your site against some CSRF attacks. If you use a wildcard, you must perform your own validation of the **`Host`** HTTP header, or otherwise ensure that you aren't vulnerable to this category of attacks.

You should also configure the Web server that sits in front of Django to validate the host. It should respond with a static error page or ignore requests for incorrect hosts instead of forwarding the request to Django. This way you'll avoid spurious errors in your Django logs (or emails if you have error reporting configured that way). For example, on nginx you might setup a default server to return "444 No Response" on an unrecognized host:

```
server {
    listen 80 default_server;
    return 444;
}
```

CACHES¶

If you're using a cache, connection parameters may be different in development and in production.

Cache servers often have weak authentication. Make sure they only accept connections from your application servers.

If you're using Memcached, consider using cached sessions to improve performance.

DATABASES¶

Database connection parameters are probably different in development and in production.

Database passwords are very sensitive. You should protect them exactly like **`SECRET_KEY`**.

For maximum security, make sure database servers only accept connections from your application servers.

If you haven't set up backups for your database, do it right now!

`EMAIL_BACKEND` and related settings¶

If your site sends emails, these values need to be set correctly.

By default, Django sends email from webmaster@localhost and root@localhost. However, some mail providers reject email from these addresses. To use different sender addresses, modify the `DEFAULT_FROM_EMAIL` and `SERVER_EMAIL` settings.

`STATIC_ROOT` and `STATIC_URL`¶

Static files are automatically served by the development server. In production, you must define a `STATIC_ROOT` directory where `collectstatic` will copy them.

See Managing static files (e.g. images, JavaScript, CSS) for more information.

`MEDIA_ROOT` and `MEDIA_URL`¶

Media files are uploaded by your users. They're untrusted! Make sure your web server never attempt to interpret them. For instance, if a user uploads a `.php` file , the web server shouldn't execute it.

Now is a good time to check your backup strategy for these files.

HTTPS¶

Any website which allows users to log in should enforce site-wide HTTPS to avoid transmitting access tokens in clear. In Django, access tokens include the login/password, the session cookie, and password reset tokens. (You can't do much to protect password reset tokens if you're sending them by email.)

Protecting sensitive areas such as the user account or the admin isn't sufficient, because the same session cookie is used for HTTP and HTTPS. Your web server must redirect all HTTP traffic to HTTPS, and only transmit HTTPS requests to Django.

Once you've set up HTTPS, enable the following settings.

`CSRF_COOKIE_SECURE`¶

Set this to `True` to avoid transmitting the CSRF cookie over HTTP accidentally.

`SESSION_COOKIE_SECURE`¶

Set this to `True` to avoid transmitting the session cookie over HTTP accidentally.

Performance optimizations¶

Setting `DEBUG = False` disables several features that are only useful in development. In addition, you can tune the following settings.

`CONN_MAX_AGE`¶

Enabling persistent database connections can result in a nice speed-up when connecting to the database accounts for a significant part of the request processing time.

This helps a lot on virtualized hosts with limited network performance.

TEMPLATES¶

Enabling the cached template loader often improves performance drastically, as it avoids compiling each template every time it needs to be rendered. See the template loaders docs for more information.

Error reporting¶

By the time you push your code to production, it's hopefully robust, but you can't rule out unexpected errors. Thankfully, Django can capture errors and notify you accordingly.

LOGGING¶

Review your logging configuration before putting your website in production, and check that it works as expected as soon as you have received some traffic.

See Logging for details on logging.

ADMINS and MANAGERS¶

ADMINS will be notified of 500 errors by email.

MANAGERS will be notified of 404 errors. **IGNORABLE_404_URLS** can help filter out spurious reports.

See Error reporting for details on error reporting by email.

Error reporting by email doesn't scale very well

Consider using an error monitoring system such as Sentry before your inbox is flooded by reports. Sentry can also aggregate logs.

Customize the default error views¶

Django includes default views and templates for several HTTP error codes. You may want to override the default templates by creating the following templates in your root template directory: **404.html**, **500.html**, **403.html**, and **400.html**. The default views should suffice for 99% of Web applications, but if you desire to customize them, see these instructions which also contain details about the default templates:

- The 404 (page not found) view
- The 500 (server error) view
- The 403 (HTTP Forbidden) view
- The 400 (bad request) view

Python Options¶

It's strongly recommended that you invoke the Python process running your Django application using the -R option or with the **PYTHONHASHSEED** environment variable set to **random**.

These options help protect your site from denial-of-service (DoS) attacks triggered by carefully crafted inputs. Such an attack can drastically increase CPU usage by causing worst-case performance when creating **`dict`** instances. See oCERT advisory #2011-003 for more information.