undefined Logo

Managing Session Data with Flask-Session & Redis

Todd Flask 6 Min Read

XI. Managing Session Data with Flask-Session & Redis
X. Handle User Accounts & Authentication in Flask with Flask-Login
IX. Connect Flask to a Database with Flask-SQLAlchemy
VIII. Compiling and Serving Frontend Assets in Flask
VII. Organizing Flask Apps with Blueprints
VI. Demystifying Flask's Application Factory
V. Configuring Your Flask App
IV. The Art of Routing in Flask
III. Handling Forms in Flask with Flask-WTF
II. Rendering Pages in Flask Using Jinja
I. Creating Your First Flask Application

When building we build applications that handle users, a lot of functionality depends on storing session variables for users. Consider a typical checkout cart: it's quite often that an

abandoned cart on any e-commerce website will retain its contents long after a user abandons. Carts sometimes even have their contents persist across devices! To build such functionality, we cannot rely on Flask's default method of storing session variables, which happens via locally stored browser cookies. Instead, we can use a cloud key/value store such as Redis, and leverage a plugin called Flask-Session.

Flask-Session is a Flask plugin which enables the simple integration of a server-side cache leveraging methods such as **Redis**, **Memcached**, **MongoDB**, **relational databases**, and so forth. Of these choices, Redis is an exceptionally appealing option.

Redis is NoSQL datastore written in C intended to temporarily hold data in memory for users as they blaze mindlessly through your site. Redis was designed for this very purpose, is extremely quick, and is free to use when spinning up a free instance on Redis Labs.

Becoming Familiar with Flask-Session

To understand Flask-Session's offering, a good place to start is by peaking at the settings that Flask-Session accepts:

SESSION_TYPE	Specifies which type of session interface to use. Built-in session types: • **null**: NullSessionInterface (default) • **redis**: RedisSessionInterface • **memcached**: MemcachedSessionInterface • **filesystem**: FileSystemSessionInterface • **mongodb**: MongoDBSessionInterface • **sqlalchemy**: SqlAlchemySessionInterface
SESSION_PERMANENT	Whether use permanent session or not, default to be True
SESSION_USE_SIGNER	Whether sign the session cookie sid or not, if set to True, you have to set **flask.Flask.secret_key**, default to be False
SESSION_KEY_PREFIX	A prefix that is added before all session keys. This makes it possible to use the same backend storage server for different apps, default “session:”
SESSION_REDIS	A redis.Redis instance, default connect to 127.0.0.1:6379
SESSION_MEMCACHED	A memcache.Client instance, default connect to 127.0.0.1:11211

SESSION_FILE_DIR	The directory where session files are stored. Default to use *flask_session* directory under current working directory.
SESSION_FILE_THRESHOLD	The maximum number of items the session stores before it starts deleting some, default 500
SESSION_FILE_MODE	The file mode wanted for the session files, default 0600
SESSION_MONGODB	A pymongo.MongoClient instance, default connect to 127.0.0.1:27017
SESSION_MONGODB_DB	The MongoDB database you want to use, default "flask_session"
SESSION_MONGODB_COLLECT	The MongoDB collection you want to use, default "sessions"
SESSION_SQLALCHEMY	A flask.ext.sqlalchemy.SQLAlchemy instance whose database connection URI is configured using the SQLALCHEMY_DATABASE_URI parameter
SESSION_SQLALCHEMY_TABLE	The name of the SQL table you want to use, default "sessions"

Even at first glance, it's pretty easy to understand what our options are.

Installation

To get started, we need to install 2 libraries: `Flask-Session` and `Redis`:

```
$ pip3 install flask-session redis
```

Configuration

Next, we need to configure our app. In our **config.py** file, we need to import the Redis library with `import redis` (we'll get to that in a minute). Next, we need to set the following variables in **config.py**:

- SECRET_KEY: Flask-Session won't work without a secret key; it's important to set this to a random string of characters (as always, make sure this is secure).
- SESSION_TYPE: Will be set to `SESSION_TYPE=redis` for our purposes.
- SESSION_REDIS: The URI of our cloud-hosted Redis instance. Redis URIs are structured a bit uniquely: `redis://:[password]@[host_url]:[port]`.

The full configuration of a Redis instance using a URI looks like this:

```
SESSION_REDIS = redis.from_url(environ.get('SESSION_REDIS'))
```

If you're having trouble with your config, feel free to borrow mine (this pulls values from .env):

config.py

```
"""App configuration."""
from os import environ
import redis


class Config:
    """Set Flask configuration vars from .env file."""

    # General Config
    SECRET_KEY = environ.get('SECRET_KEY')
    FLASK_APP = environ.get('FLASK_APP')
    FLASK_ENV = environ.get('FLASK_ENV')

    # Flask-Session
    SESSION_TYPE = environ.get('SESSION_TYPE')
    SESSION_REDIS = redis.from_url(environ.get('SESSION_REDIS'))
```

Initializing a Flask-Session Application

We already know much about the Flask application factory and how to initialize other Flask plugins such as Flask-SQLAlchemy and Flask-Login. Flask-Session is initialized in the same way: we set a global variable first, and then initialize the plugin with `sess.init_app(app)`. This is an example of an **__init__.py** file initializing Flask-Session, Flask-SQLAlchemy, and Flask-Login (this builds on the source code we used to implement Flask-Login):

__init__.py

```
"""Initialize application."""
from flask import Flask
from flask_sqlalchemy import SQLAlchemy
from flask_login import LoginManager
from flask_session import Session


db = SQLAlchemy()
login_manager = LoginManager()
sess = Session()
```

```
def create_app():
    """Construct the core application."""
    app = Flask(__name__, instance_relative_config=False)

    # Application Configuration
    app.config.from_object('config.Config')

    # Initialize Plugins
    db.init_app(app)
    login_manager.init_app(app)
    sess.init_app(app)

    with app.app_context():
        # Import parts of our application
        from . import routes
        from . import auth
        app.register_blueprint(routes.main_bp)
        app.register_blueprint(auth.auth_bp)

        # Create Database Models
        db.create_all()

        return app
```

With Flask-Session initialized, we're ready to see how this works!

Let's Mess With Some Variables

The cool thing about Flask-Session is that extends Flask native `session` object. Once we've configured our Flask app to use Flask-Session (as we already have), we can work with Flask session variables in the very same way as we would have if we were still using cookies. To get started, we simply import `session` from `flask` with this line: `from flask import session`. Let's review the basics of managing values in a Flask session:

Setting a Value

Setting a variable on a session looks a lot like setting values for any old Python dictionary object:

```
session['key'] = 'value'
```

Retrieving a Value

With a value saved to our session, we can retrieve and reuse it with `.get()`:

```
session_var_value = session.get('key')
```

Removing a Value

If the value we've saved has been used and is no longer needed, we can remove variables from our session using `.pop()`:

```
session.pop('key', None)
```

Demonstrating Sessions

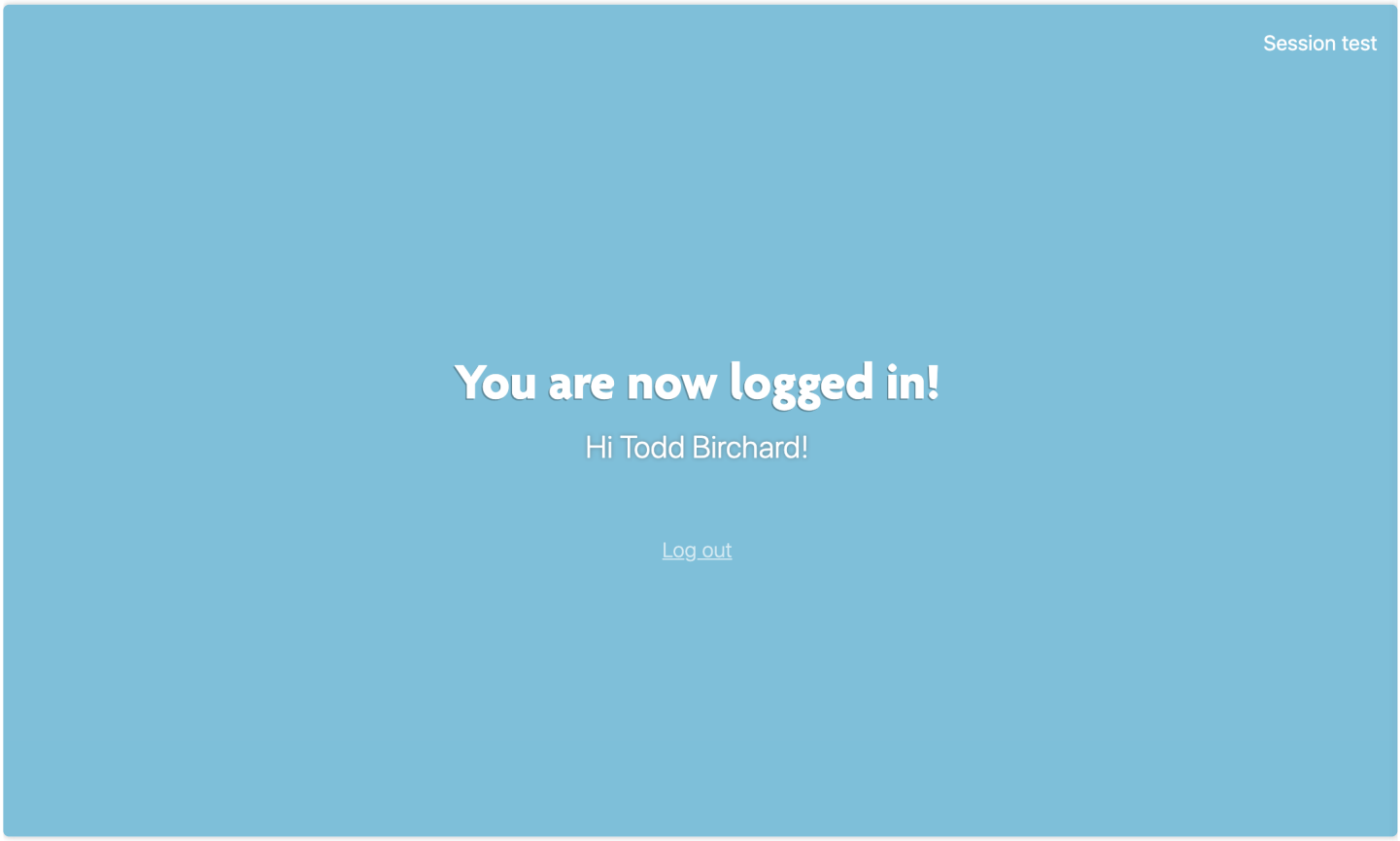
To see sessions working in action, we're going to create a couple of routes that create and display values saved to a session. First, we'll set a session variable in the route for our application's homepage. Next, we'll create a route for the specific purpose of displaying this variable:

routes.py

```
"""Routes for logged-in application."""
from flask import Blueprint, render_template, session
from flask_login import current_user
from flask import current_app as app
from .assets import compile_auth_assets
from flask_login import login_required


# Blueprint Configuration
main_bp = Blueprint('main_bp', __name__,
                    template_folder='templates',
                    static_folder='static')
compile_auth_assets(app)


@main_bp.route('/', methods=['GET'])
@login_required
def dashboard():
    """Serve logged in Dashboard."""
    session['redis_test'] = 'This is a session variable.'
    return render_template('dashboard.html',
                           title='Flask-Session Tutorial.',
                           template='dashboard-template',
                           current_user=current_user,
                           body="You are now logged in!")


@main_bp.route('/session', methods=['GET'])
```

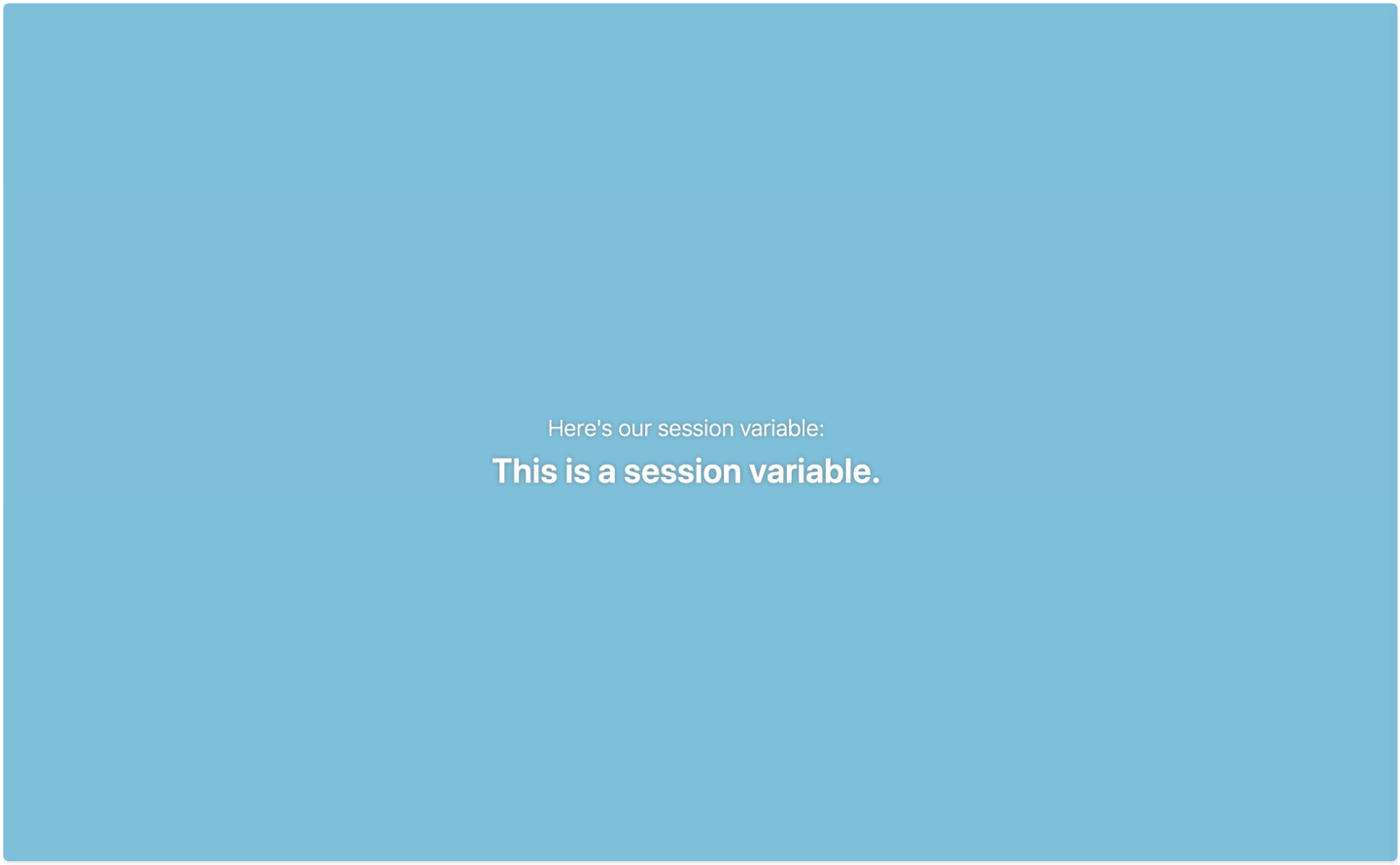
```
@login_required
def session_view():
    """Route which displays session variable value."""
    return render_template('session.html',
                           title='Flask-Session Tutorial.',
                           template='dashboard-template',
                           session_variable=str(session['redis_test']))
```

At first glance, it doesn't seem like anything has happened on our app's home page:



The logged-in homepage of our Flask app.

Behind the scenes, we've set a session variable named `redis_test`. When we navigate to http://127.0.0.1:5000/session, we can see this being displayed:



Displaying a Flask session variable's value.

Not only did we create a value that can persist across views, but since we've stored this value in a cloud Redis instance, it should persist across devices as well. I connected to my Redis instance using a GUI, and this is what came back:

key	value	type	ttl
session:b9bb3763-b4be-4e3c-8d5f-8775dc6b44fe	}q	STRING	2621377
session:c3efad4a-c6b1-4fa1-9789-f82300997d78	}q	STRING	2621387
session:eyJfZmxhc2hlcyI6W3siIHQiOlsibWVzc2FnZSIsII...	}q	STRING	2678232

A view of what's in our Redis instance.

Ahh, the keys and the values we stored are encrypted! This is why it was so important to set our secret key earlier. Nevertheless, we can see our session variables are successfully decoupled. Instead of depending on the user's browser or our app's local server, user session variables are nice and comfy in the cloud.

Get Your Hands Dirty

The best way of learning is doing, of course. For anybody who's interested, I've uploaded the source code for this tutorial to Github here:

hackersandslackers/flask-session-tutorial

:floppy_disk: :bow: Example Flask project for implementing Flask-Session with Redis. - hackersandslackers/flask-session-tutorial

hackersandslackers • GitHub

This has been another episode of Building Flask Apps! Join us next time when we... well, I'm not sure yet. Join us anyway! Peace fam.

Flask Python Software Development NoSQL

Todd Birchard

118 Posts Website Twitter

Engineer with an ongoing identity crisis. Breaks everything before learning best practices. Completely normal and emotionally stable.