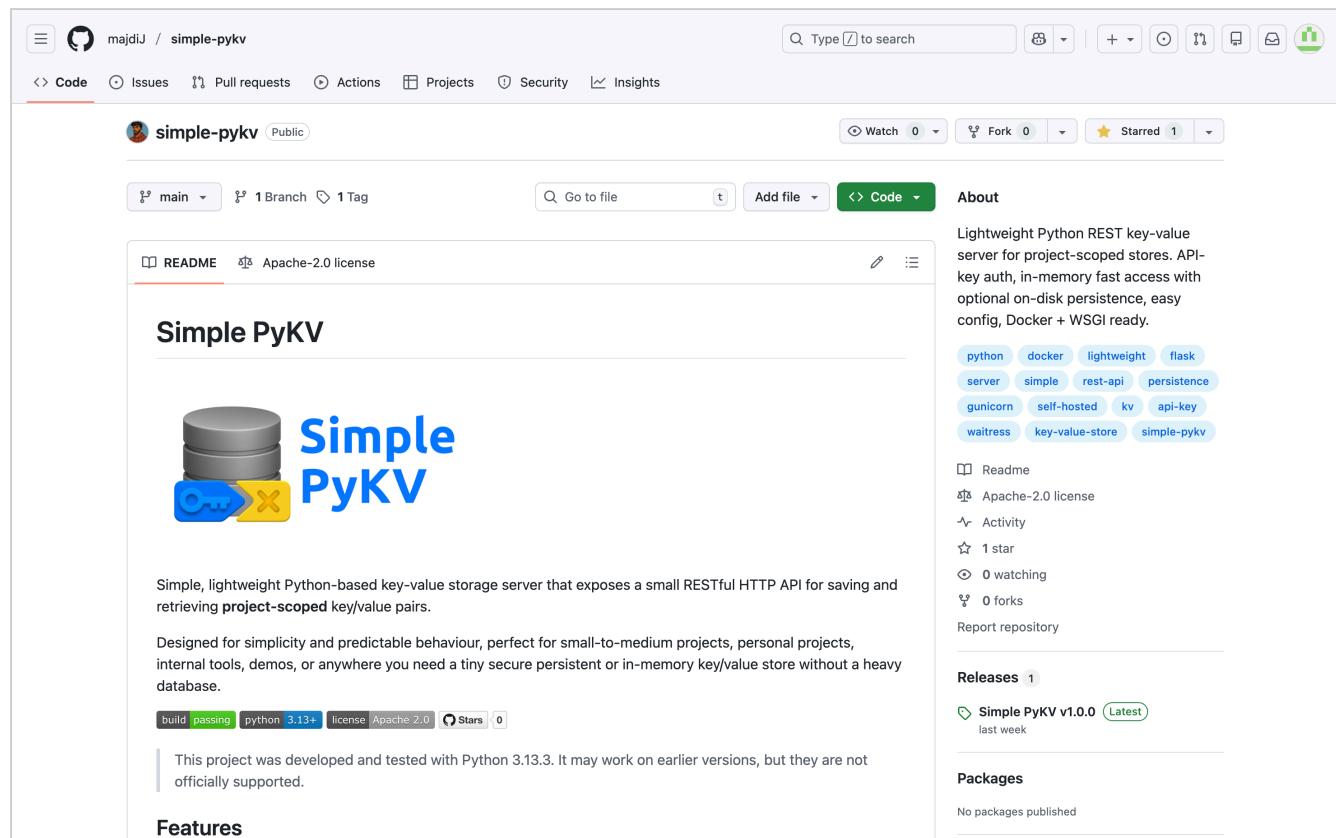


Building Simple-PyKV: A Lightweight Python Key-Value Store server

Adventures in creating an API backend for a simple key-value store using Python

By [Majdi Jaigirdar](https://majdij.com/articles/building-simple-pykv) | Published on 22 Dec 2025

<https://majdij.com/articles/building-simple-pykv>



The screenshot shows the GitHub repository page for 'simple-pykv'. The repository is public and has 0 stars, 0 forks, and 1 branch. The README file is visible, showing a logo of a database with a blue and yellow arrow pointing to it, labeled 'Simple PyKV'. The description states: 'Simple, lightweight Python-based key-value storage server that exposes a small RESTful HTTP API for saving and retrieving project-scoped key/value pairs.' It is designed for simplicity and predictable behaviour, perfect for small-to-medium projects, personal projects, internal tools, demos, or anywhere you need a tiny secure persistent or in-memory key/value store without a heavy database. The repository uses Python 3.13+, Apache 2.0 license, and is Docker ready. It includes support for gunicorn, self-hosted, kv, api-key, waitress, key-value-store, and simple-pykv. The repository has 1 star, 0 forks, and 0 watching. There is one release, Simple PyKV v1.0.0, which was released last week. No packages have been published.

In the computing world, there is no shortage of tools, services, and libraries to help developers solve problems. There's a common belief for nearly every problem a dev may face, there's likely a tool that they can use to help solve it. Often though the tools that are available can do the trick, but they don't quite fit the bill.

While I was working on my many different personal projects, prototypes, internal tools, and demos I continually ran into the same familiar issue of needing some sort of system where I could centrally store data that I could CRUD (Create, Read, Update, Delete) across different devices and programs. And the tools that were available like common databases felt just a bit too much for these lightweight needs, and I needed a more straightforward and efficient solution.

This recurring pattern of issues made me identify a gap in the toolbox. A need for a lightweight KV storage server that is easy to set up, use, and manage and allowing for CRUD operations on key-value pairs. Thus, *Simple-PyKV* was made.

In this article, I'll walk through the building of Simple-PyKV.

Table of Contents

1. [The main Point: Why not just use X?](#)
2. [Core Philosophy: Design, simplicity and Control](#)
3. [Security by Design: Isolation and Authentication](#)
4. [Technical Foundation: Python and REST](#)
5. [Project Scope and Utility](#)
6. [What's Next for Simple PyKV](#)
7. [See the Code](#)

The main Point: Why not just use X?

As mentioned, with any issues developers face there tends to be some sort of tool to help solve it. So why not just use an existing solution?

My dislike with existing solutions was that they were just too complex and/or overkill for my simple needs. Using full databases (PostgreSQL, MySQL) requires complex setup, schema management, high memory usage, and connection pooling, too much infrastructure for simple KV pair system. Even with NoSQL/caches (Redis, Memcached), while they're incredibly fast, useful, and scalable, setting them up requires a lot of overhead, APIs can be complex, and can be challenging to manage for small projects.

I also wanted something with easy format of data persistence in a human-readable way without special software, which I feel many database systems couldn't provide. I considered file storage/JSON files that would be solely saved on that machine, but that meant no easy way to access the data remotely and could cause data corruption with concurrent access.

Another reason was simply for the opportunity to learn. Building this project of simple key-value store server from scratch is a great way to deepen my understanding of RESTful APIs, data storage, backend development, and programming in Python using OOP principles.

Simple-PyKV is a middle ground, a self-contained, easy to deploy service that handles REST API requests, thread-safe storage, and project separation, letting a developer focus purely on storing and retrieving data without the overhead of managing a full database system.

Core Philosophy: Design, simplicity and Control

Currently studying Computer Science at university, and doing Object Oriented Programming (OOP) module focusing mainly on Java but also touching on Python. Learning OOP concepts and principles such as encapsulation, modularity, and separation of concerns and applying them in my Java code has influenced how I approached designing Simple-PyKV.

Though when Simple-PyKV was started, I only knew OOP solely in Java, I realised how useful these concepts are across programming languages, including Python. So I learnt how to make use of classes, methods, and attributes in Python to create a clean and maintainable codebase. Later through the module, we started learning about OOP in Python, which helped me solidify my understanding and use correct Pythonic conventions (which can be seen by me refactoring my code to be more Pythonic, evidenced by my [commit history](#) and [issues](#) on GitHub).

The code design of Simple-PyKV focuses on simplicity and clarity. Each class, method and attribute has a well defined role — Easy to understand and extend. This modularity allows for easier debugging, testing, and future enhancements.

Every project gets its own distinct, isolated namespace. This is crucial for handling multiple projects, multi-environment apps where data for 'App A' should never mix with 'App B'. Each project has its own authentication and storage settings. This isolation ensures data integrity and security across different applications using the same Simple-PyKV server.

Settings are managed by the `config.json` file. When the program starts, it reads this file to load all necessary configurations data like projects, authentication keys, and storage options. This approach makes the server highly configurable without needing to change code and being predictable, easily deployable across different environments and easily adaptable for different needs. [View configuration documentation here.](#)

Projects can choose to be in-memory only (for fast caching/temp data) or on-disk persistent (for configuration or state that must survive a server restart). Both are managed seamlessly by the config file or API calls.

Security by Design: Isolation and Authentication

For any program handling data, security is important. Especially for any service that exposes an API over the network, stores potentially sensitive data, or is used in multi-tenant environments - all of which apply to Simple-PyKV. Security must be baked into the design from the start.

By default the server generates strong secure API keys and by default these keys are never stored in plain text but rather hashed using secure hashing algorithms.

There are two levels of API keys in Simple-PyKV: System/Global API Key and Project API Keys. System/Global API Keys are used for managing the server itself - creating/deleting projects, viewing server status, etc. While Project API Keys are scoped to individual projects, allowing CRUD operations on keys/values within that specific project only.

These two levels of API keys follows the security principle of least privilege — important security concept, ensuring that users or systems only have the minimum access necessary to perform their tasks and prevent unauthorized access.

The `project_discoverable` and `keys_and_values_discoverable` settings are significant security measures. They allow developers to hide the very existence of a project or its keys from an attacker, even if they hold a valid API key, enforcing an extra layer of access control. This follows the security principle of "security through obscurity," which can be effective in reducing the attack surface.

Technical Foundation: Python and REST

There exists a wide range of programming languages and frameworks that can be used to build API servers and KV stores. Python, flask, and RESTful principles were the right choice for Simple-PyKV for several reasons: simplicity, readability, and extensive standard library. The aim of this project was to create a *simple*, accessible key-value store server, and Python's syntax and ecosystem align well with my goals.

The application uses Flask for the web framework. Crucially, for secure deployment, especially when connected to the internet, using a proper WSGI server like Gunicorn or Waitress over Flask's built-in server is important, this ensures security, performance, and reliability in production environments.

Though Simple-PyKV is meant for small amount of requests, there still are considerations for concurrency and thread-safety. Requests may try to read, write, update, or delete the same keys at the same time. Without proper safety design, this can lead to data loss/corruption. To handle this, I implemented python thread locking mechanisms around all parts where data is modified, ensuring that only one thread can modify a KV / system at a time. This provides data integrity even with concurrent requests.

Simple-PyKV API routes are designed in a way which makes their behavior and responses predictable. Following the standard conventions of RESTful APIs, using appropriate HTTP methods such as GET, POST, PUT, DELETE for respective operations. Making it simple and intuitive for developers to integrate with their projects. View detailed API documentation [here in routes.md](#).

Project Scope and Utility

Simple-PyKV isn't designed to replace full-fledged databases or key-value stores like Redis or DynamoDB. Instead, it's meant to fill a niche for lightweight, easy-to-deploy key-value storage for small projects, prototypes, or internal tools where simplicity and ease of use are more important than advanced features or scalability.

Feature	Description
Lightweight	Minimal dependencies and designed for small-to-medium or personal projects.
Python-based	Written in Python: easy to understand and extend to fit personal needs.
RESTful API	Exposes a small predictable HTTP API routes for server management and key/value operations.

Feature	Description
Project-scoped stores	Isolated namespaces per project (multi-project support).
API key authentication	Simple API-key auth (system-level and per-project keys).
Multiple auth headers	Supports different ways to take API keys (<code>Authorization</code> , <code>X-API-Key</code> , <code>Api-Key</code>) in requests.
CRUD operations	Simple create, read, update, and delete for keys and values.
In-memory & optional on-disk persistence	Use fast in-memory access with ability to persist data to disk for durability (optional per project).
Small config	Single <code>config.json</code> configuration file with appropriate and secure defaults.
Metadata support	Each key stores metadata (timestamps, size, type) with ability to access the metadata + value, or solely the value.
Key/value & project discovery controls	Security controls to allow all projects to be discoverable or restrict visibility as needed along with project key/value discovery options.
Thread-safe waitress/gunicorn/docker support	Designed for safe concurrent API access.
Simple logging / CUI	Recommended for production with Gunicorn/Waitress; includes Dockerfile and docker-compose guide.
Routes docs	Terminal logging / console UI available for basic monitoring.
	Detailed API routes documented in <code>routes.md</code> file with examples, explanations, support and code samples.

What's Next for Simple PyKV

Simple-PyKV is a small challenge project I built during my otherwise busy university and life schedule, so I don't have immediate plans for major new features. Simple-PyKV is out of beta and as of December 2025, its latest version is 1.0.0 has been released. Which means it's stable and ready for production use.

However, I do want to continue improving its stability, performance, and documentation over time and welcome contributions. Some potential areas for future exploration include:

- Advanced Data Types

Currently, it handles JSON and raw text, but adding explicit support for more complex structured data types might simplify client-side parsing.

- Replication and Backup

Implementing optional data replication or backup features to enhance data durability and recovery options via simple API calls or routine snapshots.

- Further data validation and constraints

Adding support for data validation rules or constraints (e.g., max size, allowed patterns) for keys and values to prevent invalid data storage and for config file options.

- More Logging Control

Fine-tuning the CUI/logging system to give developers more control over verbosity and log output formats.

- Enhanced Testing

As of now, there is no testing implemented. Adding a comprehensive testing code would improve reliability and facilitate future changes, especially as new features are added by contributors.

Simple PyKV is a direct result of my own need for a lean, secure, and easily manageable storage solution. It's a testament to choosing the right tool for the job, even if that tool needs to be built from scratch. Showing the power of programming to solve real problems in a practical way.

See the Code

- [GitHub Repository: Simple-PyKV](#)
- [README and Documentation](#)
- [Detailed API Reference](#)
- [License: Apache 2.0](#)

Simple-PyKV is licensed under the Apache License 2.0. See the LICENSE file for details.

Click above link for the full license text.

Note: Trademarks and logos are not included in the license.

By [Majdi Jaigirdar](#) | Published on 22 Dec 2025