# Porkbun MCP Server Documentation

## 1. Introduction and Overview

The Porkbun MCP Server is a powerful, secure, and extensible server designed to provide a comprehensive interface to the Porkbun API. It is built using the Micro-service Controller Protocol (MCP) and offers a wide range of tools for managing domains, DNS records, SSL certificates, and more.

### 1.1 What is the Porkbun MCP Server?

The Porkbun MCP Server is a backend application that acts as a bridge between any MCP-compatible client and the Porkbun API. It simplifies the process of interacting with Porkbun’s services by providing a standardized set of tools and a secure environment for managing your domains and related resources. The server is designed for developers, system administrators, and anyone who needs to automate and manage their Porkbun account programmatically.

### 1.2 Key Features and Capabilities

* **Comprehensive API Coverage:** The server provides access to a wide range of Porkbun API features.
* **Secure Credential Management:** The server uses strong encryption (AES-256) to protect your Porkbun API keys.
* **Extensible Toolset:** The server’s functionality is exposed through a set of “tools” that can be easily listed and invoked by MCP clients.
* **Asynchronous and Performant:** Built on Python’s asyncio and aiohttp, the server is designed for high performance.
* **Containerized and Easy to Deploy:** The server is fully containerized using Docker, making it easy to deploy, manage, and scale.

### 1.3 Security Features

* **Encryption at Rest:** All sensitive data, including API credentials, is encrypted using Fernet (AES-256) before being stored.
* **Input Validation:** All user-provided input is rigorously validated to prevent common security vulnerabilities.
* **Secure by Default:** The server is designed to be secure by default, with features like read-only filesystems and non-root user execution in the Docker container.
* **Credential Management:** The server provides a secure way to manage your Porkbun API credentials.

## 2. Quick Start Guide

This section will guide you through the process of setting up and running the Porkbun MCP Server for the first time.

### 2.1 Prerequisites

* **Docker and Docker Compose:** The server is designed to be run in a Docker container.
* **Porkbun API Credentials:** You will need to have a Porkbun API key and secret API key.

### 2.2 Installation Steps

1. **Clone the Repository:** Clone the repository containing the Porkbun MCP Server to your local machine.
2. **Create an Environment File:** Create a .env file in the root of the project with the required environment variables.
3. **Build and Run the Container:** bash docker-compose up --build

### 2.3 Basic Configuration

The server’s configuration is managed through the config.py file and can be overridden using environment variables. The most important configuration options are PORKBUN\_API\_KEY, PORKBUN\_SECRET\_API\_KEY, and PORKBUN\_MCP\_ENCRYPTION\_KEY.

### 2.4 First Usage Example

Once the server is running, you can interact with it using any MCP-compatible client. Here’s an example of how to list the available tools using a simple Python script:

import asyncio  
from mcp.client import Client  
  
async def main():  
 async with Client() as client:  
 server = await client.connect("porkbun-mcp")  
 tools = await server.list\_tools()  
 for tool in tools:  
 print(f"Tool: {tool.name}")  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 asyncio.run(main())

## 3. API Reference Documentation

The Porkbun MCP Server provides comprehensive MCP functionality including:

* **26 Tools** for complete Porkbun API coverage
* **7 Resources** providing contextual documentation, examples, and validation rules
* **7 Prompts** offering guided workflows for common domain management tasks

### 3.1 Resources

The server provides contextual resources accessible via URI patterns:

| Resource URI | Description |
| --- | --- |
| porkbun://docs/api-overview | Complete Porkbun API overview and authentication guide |
| porkbun://docs/domain-management | Best practices for domain management and configuration |
| porkbun://docs/dns-records | Comprehensive DNS records reference and validation rules |
| porkbun://docs/security-practices | Security guidelines for API usage and domain protection |
| porkbun://docs/troubleshooting | Common issues, error codes, and troubleshooting solutions |
| porkbun://examples/dns-configurations | Real-world DNS configuration examples in JSON format |
| porkbun://schemas/validation-rules | Complete input validation rules and schemas |

### 3.2 Prompts

Pre-built prompt templates for guided domain management workflows:

| Prompt Name | Description | Arguments |
| --- | --- | --- |
| setup-new-domain | Complete new domain setup guide | domain, website\_ip, mail\_server, include\_www |
| migrate-domain | Step-by-step domain migration guide | domain, current\_registrar, has\_email |
| configure-dns-records | DNS configuration for common services | domain, service\_type, target |
| troubleshoot-dns | DNS troubleshooting and diagnostic guide | domain, issue\_description, record\_type |
| security-audit | Domain security audit and recommendations | domain, check\_dnssec, check\_ssl |
| bulk-domain-operation | Bulk operations on multiple domains | domains, operation, parameters |
| domain-portfolio-analysis | Portfolio analysis and optimization | include\_pricing, include\_expiry, include\_dns |

### 3.3 Tools

The server exposes its functionality through a comprehensive set of tools. Here is a list of the available tools, their parameters, and their output formats.

#### 3.3.1 General Tools

* porkbun\_ping: Pings the Porkbun API to check for connectivity and retrieves your IP address.
* porkbun\_get\_pricing: Retrieves the pricing for all TLDs supported by Porkbun.

#### 3.3.2 Domain Tools

* domain\_list\_all: Retrieves a list of all domains in your Porkbun account.
* domain\_get\_nsv: Retrieves the nameserver information for a domain.
* domain\_update\_nsv: Updates the nameservers for a domain.

### 3.3 DNS Tools

* dns\_retrieve\_records: Retrieves the DNS records for a specific domain.
* dns\_create\_record: Creates a new DNS record for a domain.
* dns\_edit\_record: Edits an existing DNS record.
* dns\_delete\_record: Deletes a DNS record.

### 3.4 SSL Tools

* ssl\_retrieve: Retrieves the SSL certificate bundle for a domain.

### 3.5 Error Handling

In the case of an error, the server will return a JSON object with a “status” field set to “ERROR” and an “error\_message” field containing a description of the error.

## 4. Security Best Practices

### 4.1 Credential Management

* **Use Environment Variables:** The recommended way to provide your Porkbun API credentials to the server is through environment variables.
* **Encryption Key:** Always set a strong, unique PORKBUN\_MCP\_ENCRYPTION\_KEY to encrypt your credentials at rest.

### 4.2 Encryption Details

The server uses the Fernet symmetric encryption algorithm, which is part of the Python cryptography library. Fernet guarantees that a message encrypted using it cannot be manipulated or read without the key.

### 4.3 Rate Limiting

The server includes a rate-limiting mechanism to prevent you from exceeding Porkbun’s API rate limits. By default, the server allows 10 requests every 10 seconds.

### 4.4 Input Validation

All input to the server’s tools is rigorously validated to prevent common security vulnerabilities. This includes validating domain names, IP addresses, and other data types.

### 4.5 Deployment Security

* Run the server as a non-root user.
* Use a reverse proxy (like Nginx) to handle SSL termination.
* Keep the server and its dependencies up to date with the latest security patches.

## 5. Deployment Guide

### 5.1 Docker Deployment

The recommended way to deploy the Porkbun MCP Server is using Docker. The repository includes a Dockerfile and docker-compose.yml file to make this process as easy as possible.

### 5.2 Environment Configuration

The server can be configured using environment variables. See the config.py file for a full list of available options.

### 5.3 Production Deployment

For production deployments, it is recommended to use the docker-compose.yml file provided in the repository. This will start the server in production mode, with a non-root user and a read-only filesystem.

### 5.4 Monitoring Setup

The docker-compose.yml file includes optional services for monitoring the server with Prometheus and Grafana.

### 5.5 Backup and Recovery

The deploy.sh script includes commands for backing up and restoring the server’s data volumes.

## 6. Troubleshooting and FAQ

### 6.1 Common Issues and Solutions

* **“Invalid API Key” Error:** Double-check that you have set the PORKBUN\_API\_KEY and PORKBUN\_SECRET\_API\_KEY environment variables correctly.
* **“Rate Limit Exceeded” Error:** You have exceeded Porkbun’s API rate limits.

### 6.2 Debugging Tips

* **Check the Logs:** The server logs all requests and errors to the console and to a log file.
* **Enable Debug Logging:** Set the LOG\_LEVEL environment variable to “DEBUG”.

### 6.3 Performance Optimization

* **Caching:** The server’s caching mechanism can significantly improve performance for repeated requests.
* **Asynchronous Requests:** Use an asynchronous client to take full advantage of the server’s performance.

## 7. Development Guide

### 7.1 Setting Up a Development Environment

To set up a development environment, you will need to have Python 3.11 and pip installed on your system. You will also need to install the dependencies listed in the requirements.txt file.

### 7.2 Running Tests

The repository includes a comprehensive test suite in the test\_porkbun\_mcp.py file. To run the tests, use the following command:

python3 -m unittest code/test\_porkbun\_mcp.py

### 7.3 Contributing Guidelines

1. **Fork the Repository**
2. **Create a Branch**
3. **Make Your Changes**
4. **Run the Tests**
5. **Submit a Pull Request**

## 8. Sources

* code/porkbun\_mcp\_server.py: The main server implementation.
* code/config.py: Configuration management.
* code/test\_porkbun\_mcp.py: Test suite.
* code/Dockerfile: Dockerfile for building the server image.
* code/docker-compose.yml: Docker Compose file for deployment.