Ernesto Hernandez

Prototyping a Cloud VPS Management System with Django and the HCloud of Hetzner

Metropolia University of Applied Sciences

Master of Engineering

Information Technology

Master's Thesis

15th August 2025

PREFACE

This Networking and Services Master's Thesis is the second step of a journey that began when I was allowed to participate in the bachelor’s program in Information and Technology at Metropolia UAS. For this continuation of the journey, the master’s program has delivered all that it promised: lectures, practice, workshops, assignments, projects, and independent study.

I have met amazing Professors, and I am honoured for the opportunity to share space as a student of this master's degree program with:

1. Professor Ville Jääskeläinen: who introduced us to the Research and Methods skills for this thesis.
2. Professor Erik Pätynen: I have never met somebody so updated on the latest cybersecurity trends. I enjoyed every single one of his classes.
3. Professor Peter Hjort: My hyperfocus went to levels unknown to me thanks to his classes on Fundamentals of Data Science, Machine Learning, and Artificial Intelligence. It couldn’t be better.
4. Professor Saana Vallius: Incredible, refreshing classes in IoT.

Ernesto Hernandez

Abstract

Author:	Ernesto Hernandez
Title:	Prototyping a Cloud VPS Management System with Django and the HCloud of Hetzner
Number of Pages:	72 pages
Date:	15th August 2025
Degree:	Master of Engineering
Degree Programme:	Information Technology
Professional Major:	Networking and Services
Supervisors:	Ville Jääskeläinen, Lecturer and Program Coordinator

---

This thesis is about creating a prototype of an application using Python with the Django Framework to manage the Cloud VPS of Hetzner Datacenter. This is not an API development thesis, but rather will use the already made API of Hetzner to facilitate the management of their services through a Web Interface. The objective of the thesis is to demonstrate the use of current technologies to create an independent interface to interact with the services of the data centre in a clean and transparent way. This implementation will allow the use of the services in a white collar fashion, opening the possibilities for third parties to use this work to integrate the Hertzler products as resellers of their own or for management of the public cloud in the case of a private business. This work will demonstrate how to use JSON (JavaScript Object Notation) and the use of the standard HTTP methods like GET, POST, PUT, and DELETE for communicating between the Server (API) and the Client (the Application), with the use of Python, Django Framework and the Hcloud Library of Hetzner. The work will follow the best security practices of Django for real life implementation. The complete code is shared on:

https://github.com/ErnestoFinland/HetzManager

Keywords: API, Rest API, Django, Python, Prototype, HCloud, JSON, Curl

---

List of Abbreviations

2FA	Two-Factor Authentication
API	Application Programming Interface
AWS EC2	Amazon Elastic Compute Cloud
AWS	Amazon Web Services
BCRYPT	Blowfish crypt
CAPTCHA	Completely Automated Public Turing Test to Tell Computers and Humans Apart
CERN	Conseil Européen pour la Recherche Nucléaire
CI/CD	Continuous Integration and Continuous Delivery
CORS	Cross-Origin Resource Sharing
CPU	Central Process Unit.
CRUD	Create, Read, Update, and Delete
CSRF	Cross-Site Request Forgery
CSS	Cascading Style Sheets
DevOps	Development and Operations
DRF	Django REST Framework
EXT4	Fourth Extended File System
GCP	Google Cloud Platform
GDPR	General Data Protection Regulation
GUI	Graphical User Interface
HCLOUD	Hetzner Cloud
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IBM	International Business Machine
IFRAME	Inline Frame
IP	Internet Protocol
JS	JavaScript
JSON	JavaScript Object Notation
LAAS	Infrastructure As A Service.
MVC	Model View Control
MVT	Model View Template
MySQL	My Structured Query Language.
NIST	National Institute of Standards and Technology

ORM	Object-Relational Mapping
OWASP	Open Web Application Security Project
PAAS	Platform As A Service
PBKDF2	Password-Based Key Derivation Function 2
PIP	Python Install Package
PyPI	Python Package Index
RAM	Random Access Memory,
RBAC	Role-Based Access Control
REST API	Representational State Transfer Application Programming Interface
RESTful	Representational State Transfer
SAAS	Software As A Service.
SMTP	Simple Mail Transfer Protocol
SOC 2	System and Organization Controls 2
TLS	Transport Layer Security
UI	User Interface
UX	User Experience
VMs	Virtual Machines.
VPS	Virtual Private Server
WHMCS	Web Host Manager Complete Solution
XSS	Cross-Site Scripting

Table of Contents

1 Introduction

In the 1960s, there were already talks about the concept of users gaining large-scale computing power through:

> *“time-sharing, optimizing the infrastructure, platform, and applications, and increasing efficiency”.* ***(Wikipedia, 2025)***

Later in the 2000s, names like Amazon Web Services, Microsoft Azure, IBM Smart Cloud, Oracle Cloud, and others started making investments in this technology. Access to the services was initially through web applications, but the real power of escalation was through API (Application Programming Interface).

Nowadays, there is an extensive list of services and products offered by cloud providers where users can deploy Virtual Private Servers within minutes or seconds after placing the order, with the capability of scaling as necessary. The way all these products are provided is very convenient, but with it comes the complexity of provisioning, monitoring, and security.

Giants such AWS and Azure have extensive and expensive back-end interfaces, and this cost is passed on to the customers. There are, however, other organizations that offer a different infrastructure with more competitive prices, like Hetzner Online Gmbh.

Hetzner Online Gmbh is a data centre that provides the infrastructure and the technology without the burden of deployment from the major Cloud Providers. The products offered include VPS, Bare Metals, Colocations, Object Storage, and just a few more, and that is all. It is up to the customers what they do with the products, and the real difference is the price.

This thesis is about prototyping a management system using the API of Hetzner that allows customers to easily deploy and manage a VPS. This work can also be applied to companies that want to resell Hetzner products, especially for those

who want to have the initial infrastructure in a white-label design, where the name Hetzner may be hidden as they deploy resources transparently.

1.1 Problem Statement, Research Question, and Objectives

The problem statement resides in the context of the evolution of cloud computing, and it is presented in the next paragraph:

"Even with the growing need for affordable cloud services and VPS deployment, many tools are either closed-source, expensive, or impossible to personalize. With this, a gap is created for resellers looking for an easy customizable application that covers their needs."

Accordingly, this thesis also poses the following research question:

"How can a Django-based cloud management system be used to design a prototype that companies and individuals can provision, monitor, and control VPS resources?"

And with this research question, the objective of this thesis is simple and clear: "*to prototype a VPS management system using Hetzner's API, with resale/white-label capability*".

Within the question and the objective, two secondary objectives are also present:

i. Design: Develop an architecture that pairs Django's Model View Template with Hetzner's REST API.
ii. Prototyping: Build and validate a prototype, complete with a dashboard, automated provisioning, and features, and evaluate its performance under workloads.

1.2 Project Plan

The Project Plan was developed during the master's program. The content follows from left to right the elements described in it. Below is Figure 1 below lists all the topics:

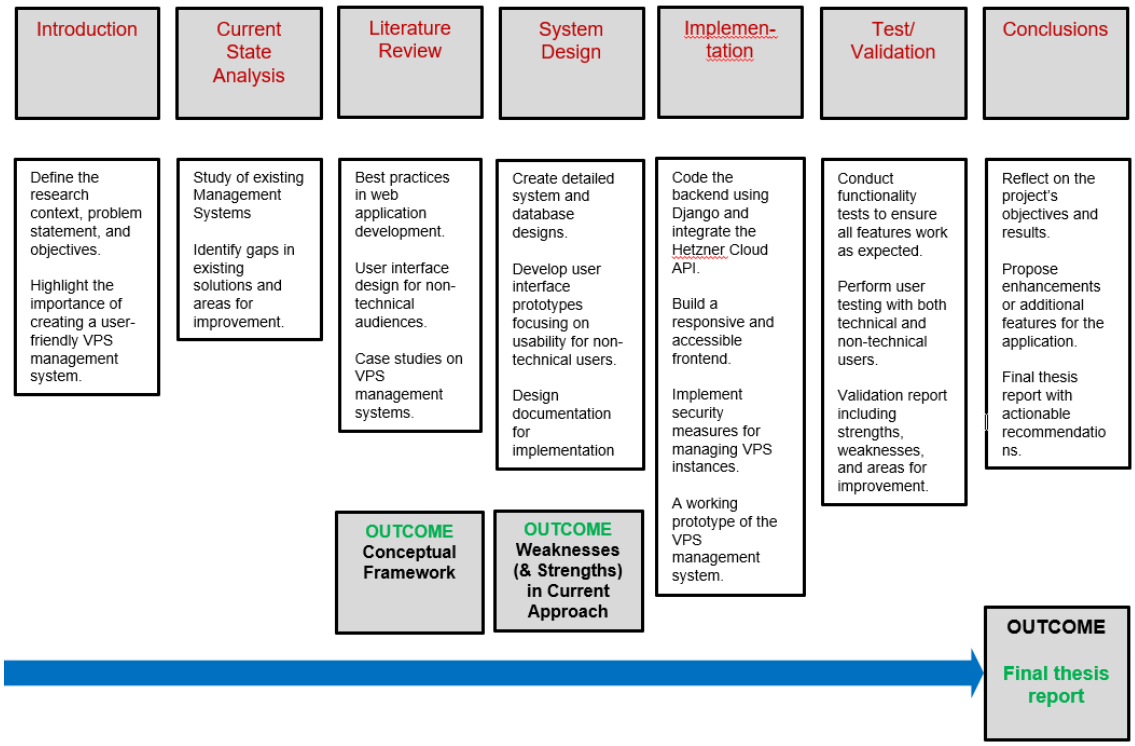


Figure 1. Project Plan.

1.3 Benefits of solving the problem

Clearly, there are advantages of solving this problem on three fronts:

i. Operational efficiency: Automated workflows can reduce the time it takes to create servers from minutes to seconds, enabling engineers to focus on higher-level tasks.
ii. Cost reduction: Central dashboards mitigate orphan or over-sized instances, reducing monthly bills.

iii. Security and compliance: standardized builds through templates reduce the complexity of your configurations and embedded authentication and audit logs to stick to the standards required to win clients.

The study contributes an open-source reference that other users can adapt to their own needs.

1.4 Methodological Approach

The thesis adopts a design-science methodology:

i. Requirements analysis and review of current scripts.
ii. Design of the models, views, templates, workflows, and security.
iii. Prototype development in Django, with Python, MySQL, Bootstrap, and JavaScript
iv. Evaluation integration, and Unit Test with Django.
v. Reflection of the results.

1.5 Scope and Delimitations

To keep the study focused and achievable within the timeframe, a set of boundaries is established. Below is Table 1, which describes what is included and excluded in this study, as well as the explanation.

Table 1. Scope and Delimitation of this thesis.

Included	**Excluded**	**Explanation**
Automated creation, start/stop, resize, snapshot, and deletion of standard Hetzner Cloud servers, Firewall, Images, Floating IP	Advanced networking (e.g., private VLANs, load balancers)	Core lifecycle operations cover ~85 % of current use cases; networking features would require separate, complex policy logic
User and Administration Access Control	Fine-grained per-resource permissions and user profile.	A simpler model is sufficient for the internal team of a small group
Basic monitoring (CPU, RAM, disk) pulled from Hetzner metrics	Deep application-level monitoring and alerting	Other tools like Prometheus and Grafana already fulfil this need
Encrypt passwords to protect user data and utilize CSRF tokens to prevent cross-site	Formal penetration testing and SOC 2 attestation, 2FA	Full compliance audit beyond thesis scope

request forgery attacks, and captcha.		

This thesis addresses a critical operational gap for companies by designing and deploying a Django-based control system for Hetzner VPS.

2 Current State Analysis

This section mentions some of the current VPS management systems that are being used by companies and organizations. These sections cover three among many others.

2.1 Study of Existing VPS Management Systems

To understand where this project fits within the current scenery, it’s important to first take a look at the existing tools available for managing virtual servers.

One popular example is the WHMCS Hetzner module, made by ModulesGarden. However, this tool has some constraints; it is proprietary, encrypted, and requires an annual license that may be expensive for solo developers, compact teams, startups, and freelancers. These factors make it less attractive for programmers or resellers who want more control or modification.

There are also broader server administration platforms like OpenStack, Virtualizor, and SolusVM. These provide a wide array of functions for hosting companies, often backing numerous cloud vendors. They can be pricey and bring unnecessary complexity for individuals that only need to control servers on a single scale, like Hetzner.

On the open-source side, instruments such as OpenNebula, Apache CloudStack, and Mist.io exist. These platforms are usually for corporate-level deployments, and they come with a steep learning curve. They require significant foundation and configuration, which makes them less practical for small-scale use or teams that want something small and centered.

While the market does offer a diversity of selections, most of them either demand significant financial investment, are too convoluted, or they don’t have the flexibility for customization. There’s an absence of instruments that are affordable, user-friendly, and intended specifically for smaller teams or resellers working with providers like Hetzner or similar datacenters.

2.2 Gaps and Areas for Improvement

After examining what's currently available, a few deficiencies emerge, gaps where existing tools fall short and where this thesis aims to contribute.

Firstly, customization and flexibility are often absent. Most commercial systems don't allow developers to tailor the software to their needs, particularly regarding branding or workflow modifications. An open-source solution would provide users the freedom to adapt the system to their use case.

Secondarily, vendor lock-in is prevalent. Most tools are built around a single provider alone and necessitate separate licenses per vendor. This renders it troublesome for users who want to experiment or switch between services without starting from scratch.

Thirdly, many of the platforms accessible nowadays are intended for users with a technical background, proffering intricate dashboards and sophisticated networking options. These characteristics are helpful in some instances but they can be overwhelming for non-technical users or small teams that simply want to generate, administer, and monitor virtual servers without complexity.

Lastly, usability and simplicity are regularly disregarded. The objective of this endeavor is to offer an alternative that is straightforward to employ while still being sufficiently powerful for real situations.

By concentrating on a clean, intuitive interface and core capabilities like server creation, resizing, and basic monitoring, this system aims to strike a balance. This undertaking is to prototype a system that's not just practical for Hetzner clients but also adaptable for any provider with a similar API structure, a practical starting point for resellers or developers looking for additional control and ease of use.

3 Literature Overview

The Literature Overview covers everything from the development of cloud technologies to current solutions for cloud management and APIs and JSON's role in cloud management to cloud computing and VPS management. The use of cURL for API rest calls and Hetzner’s cloud offerings are examined together with the Hcloud Python Library as well as the benefits of Python and Django in web development. The reliability and security in web frameworks such as Django for backend development is of utmost concern as noted in this section. Also, this section points out the ease of use for the end users and non-technical users.

3.1 Cloud Computing and VPS Management Overview

To understand the context of this endeavor, it is useful to revisit our definition of "cloud computing." As Sunilkumar Manvi and Gopal Shyam note, it referst to:

> *“a space over network infrastructure where computing resources such as computer hardware, storage, databases, networks, operating systems, and even entire software applications are available instantly, on-demand.”* **(*Sunilkumar Manvi, Gopal Shyam. 2021*)**

While these individual technologies, networks, repositories, and operating systems have been around for decades, cloud computing has transformed how they are provided. Rather than managing physical servers or software locally, users now access these services as adaptable, scalable resources via the internet.

This shift is particularly evident at the infrastructure level, where Infrastructure as a Service (IaaS) has become standard practice. IaaS allows providers to deliver virtualized assets, such as virtual machines (VMs), to customers instantly, as needed. These VMs are powered by hypervisors, which allocate CPU, memory, and network interfaces across physical hosts. The National Institute of Standards and Technology (NIST) defines a hypervisor as:

> *“manages the guest OSs on a host and controls the flow of instructions between the guest OSs and the physical hardware.”* ***(NIST, 2025)***

With this setup, users can generate virtual servers, also called Virtual Private Servers (VPS), that behave like physical machines but are far more malleable. These instances can dynamically scale resources, operate in isolated environments, and be deployed nearly instantly, making them ideal for many modern applications.

Table 2 below shows the most common infrastructures present nowadays as cloud services.

Table 2. Key infrastructure models and applications.

Model	**Features**	**Cases**	**Example**
Shared hosting	Single OS, shared environment, software-based isolation	Static websites, blogs	cPanel, Plesk, Webmin
VPS/Cloud	Virtualization, dedicated OS per user, elastic scaling	SaaS back ends, development	Hetzner Cloud, DigitalOcean
Bare-metal	Full hardware, no virtualization, single tenant	Databases, latency-sensitive apps	AWS EC2 Metal, Hetzner EX
Container PaaS	OS-level isolation, container images	CI/CD pipelines, microservices	Heroku, AWS ECS

The evolution over the past 20 years has altered corporate infrastructure strategies. Prior to widespread cloud adoption, maintaining an on-premise datacenter gives considerable financial and operational burdens. Hardware purchases, physical requirements, security, and constant maintenance created barriers to scaling applications and services. Even outsourcing to colocation facilities or leased servers barely reduced costs while complexity persisted.

Cloud computing emerged as alternative, virtualizing resources into elastic, pay-as-you-go services. IaaS freed organizations from owning physical machines, empowering dynamic provisioning of computer capacity.

Higher levels of abstraction like PaaS automated infrastructure management, concentrating attention on code deployment. The epitome, SaaS, removed all non-core responsibilities, delivering fully-managed applications through a browser. Microsoft 365 exemplifies this model, keeping Office software continuously updated and optimized behind the scenes.

This transition unleashed new flexibility. IaaS vendors enable instant scaling of virtual machines to meet fluctuating demand. Hypervisors seamlessly oversee operating systems, masking resource sharing across a distributed physical layer.

Customers control software and data while providers handle networking, storage, servers and virtualization platforms. PaaS pushes abstraction further by handling runtime environments, leaving users to build and run applications. SaaS takes the fullest advantage, outsourcing all non-product duties to experts maintaining hardware, networks and software platforms worldwide.

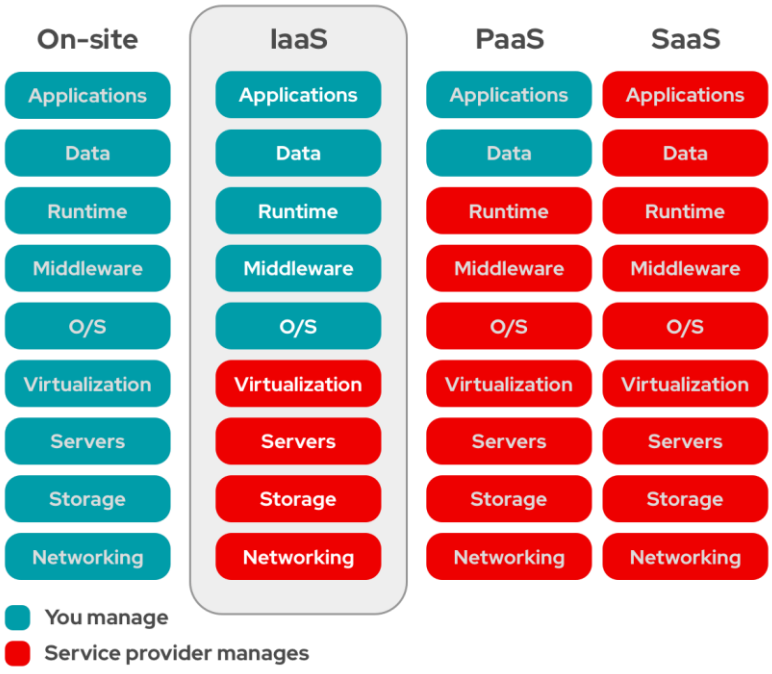


Figure 2. New model of cloud services. ***(Red Hat, 2025)***

3.2 Existing Cloud Management Solutions

Major cloud providers such as Amazon Web Services, Google Cloud Platform, and Microsoft Azure offer centralized management consoles that support a diverse assortment of features.

Picking Amazon Web Services for definition:

> *“Amazon Web Services (AWS) is a platform of web services that offers solutions for computing, storing, and networking, at different layers of abstraction. For example, you can attach volumes to a virtual machine—a low level of abstraction—or store and retrieve data via a REST API—a high level of abstraction”.* ***(Andreas Wittig, Michael Wittig, 2023)***

For example, the extensive AWS Management Console equips users with instruments to administer EC2 instances, access policies, surveillance dashboards, billing data, and auto-scaling groups in a facile manner.

The figure 3 shows the complex selection of services based only on VPS Compute.

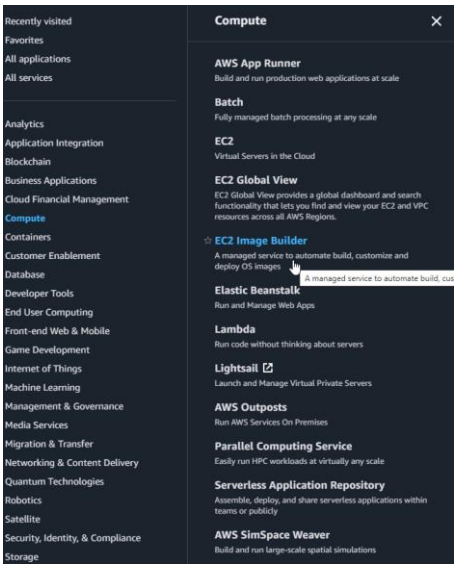


Figure 3. Amazon Web Services -Compute Services.

These interfaces skillfully fuse both graphical user interface access and exhaustive APIs, enabling automation and integration with external systems through programmatic means.

While these platforms without a doubt possess formidable powers, they usually come accompanied by steep costs, intricate service contracts, and uncertain pricing models that can confound even the smartest users.

In some scenarios, organizations struggle with issues such as constrained governance options or foggy data ownership policies, matters that have become increasingly significant in Europe since the stringent enforcement of the General Data Protection Regulation in 2018.

Consequently, some users have embarked on exploring smaller, more transparent providers like Hetzner, notably for projects where oversight over data location and cost predictability is a priority.

3.3 APIs and JSON in Cloud Management

By definition:

> “A web API is a remote interface for applications a web API enables one application (server, backend, or provider) to expose functions or operations that other applications (clients or consumers) can use, call, or consume remotely over a network.” ***(Arnaud, Lauret. 2025).***

At the core of most cloud automation lies the omnipresent Application Programming Interface. APIs allow software systems to interconnect by defining how requests must be made and how responses should be structured.

Simply put, you can see an API as a digital intermediary, it accepts queries, retrieves data, and returns it in a format recognizable to the client system. For example, if you're developing a climate app that displays the temperature in Hämeenlinna, you don’t need to construct a meteorological backend. Rather,

your application would utilize an API offered by a weather channel service to get the data and then place it within your interface. An actual API response example would be:

```
{
  "city": "Hämeenlinna",
  "temp": 22
}
```

Listing 1. Example of JSON Request from a Weather Channel.

Most APIs today employ a light format designated JSON (JavaScript Object Notation) for the intercommunication. JSON is facile to read, straightforward to generate, and language-agnostic, rendering it ideal for new web applications.

> *“The JavaScript Object Notation data format, or JSON for short, is derived from the literals of the JavaScript programming language. This makes JSON a subset of the JavaScript language. As a subset, JSON does not possess any additional features that the JavaScript language itself does not already possess. Although JSON is a subset of a programming language, it itself is not a programming language but, in fact, a data interchange format.”* ***(Ben Smith, 2015)***

In the case of Hetzner Cloud, their API is RESTful, employs HTTPS for secure transmission, and adheres to the standard CRUD operations (GET, POST, PUT, DELETE). This format makes it very accessible and easy to work for developers and simple to integrate into third-party applications.

As their documentation states:

> *”The Hetzner Cloud API operates over HTTPS and uses JSON as its data format.”* ***(Hetzner Official Documentation, 2025)***

3.4 cURL in API Testing

When developing or testing APIs, one commonly used tool is Curl.

> *“cURL, which stands for client URL, is a command line tool that developers use to transfer data to and from a server. At the most fundamental, cURL lets you talk to a server by specifying the location (in the form of a URL) and the data you want to send. cURL supports several different protocols, including HTTP and HTTPS, and runs on almost every platform. This makes cURL ideal for testing communication from almost any device (as long as it has a command line and network connectivity) from a local server to most edge deices.”* ***(Johanna Saladas. 2024)***

It’s a command-line utility found in most Unix-based systems and is used to send HTTP requests to web servers. Curl allows developers to quickly test endpoints and functions without building full applications, making it ideal for trying out new APIs or debugging troublesome ones. A versatile and lightweight program, curl remains invaluable for sending bespoke network requests during all phases of the development cycle. With curl, you can simulate requests such as:

```
curl -H "Authorization: Bearer $API_TOKEN" \
    "https://api.hetzner.cloud/v1/servers"
```

Listing 2. Curl Command representation.

The API will respond with structured JSON, which is something like:

```
{
  "servers": [
    {
      "id": 42,
      "name": "my-resource",
      "status": "running",
      "created": "2016-01-30T23:55:00+00:00",
      ...
    }
  ]
}
```

Listing 3. API respond example.

It's easy to test endpoints using Curl; it is possible to verify authentication headers and preview responses without needing the full frontend application. This capability makes it an essential tool for developers when developing cloud systems, with APIs like Hetzner's.

3.5 Hetzner Cloud and the Hcloud Python Library

While interacting with APIs through raw requests is viable, Hetzner streamlines the procedure by offering an official Python library called hcloud. This package wraps the API in convenient Python objects and approaches, simplifying managing resources from scripts or programs.

According to documentation (from the developer communities), hcloud allows Python developers to control, to mention some:

i. Server origination, deletion, rebooting, and resizing
ii. Floating IPs and firewall configurations
iii. OS images and snapshots
iv. Volume management
v. Resource protection settings

Figure 4 shows part of the customer console and its features. It is a very variate UI; all the present features can be controlled with the API using the Hcloud.

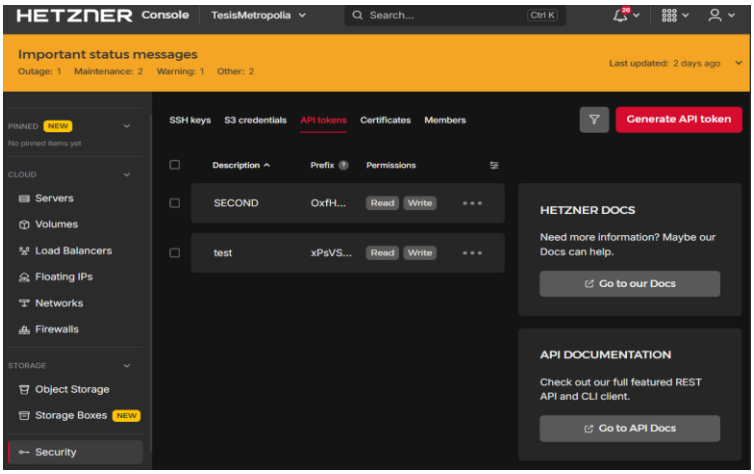


Figure 4. Hetzner Administration Console

This project makes full use of the Hcloud library, which can be installed via:

```
“pip install hcloud” (Official Hetzner Cloud python library. 2025)
```

Listing 4. PIP command to install the Hetzner Cloud.

With it, applications can perform almost everything accessible through Hetzner’s web interface, but from a clean, programmatic backend, making automation and customization significantly easier.

3.6 Python as the Development Language

Python has become one of the most widely adopted programming languages globally, and for good reasons. It offers a clean, readable syntax and supports various programming paradigms, including procedural, functional, and object-oriented.

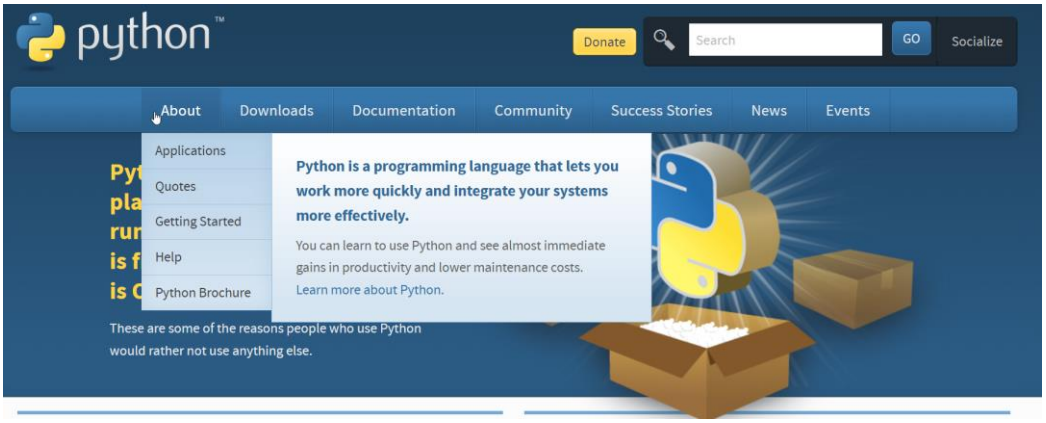


Figure 5. Python Official Web Site.

As Mark Lutz puts it in his book Learning Python:

> *“A general-purpose programming language that blends procedural, functional, and object-oriented paradigms and accelerates software development by reducing complexity.” (Mark Lutz, 2025)*

Python's success also lies in its massive ecosystem. Through the Python Package Index (PyPI), developers have access to thousands of libraries for everything from data science to web development. While Python code is interpreted rather than compiled, its performance is often very high thanks to underlying libraries written in C, which provide speed without sacrificing ease of use.

3.7 Django and the Web Application Framework

This project is built on Django, a high-level web framework for Python. As described in its official documentation:

> *“Django is a high-level Python web framework that encourages rapid development and clean, pragmatic design. It’s free and open source, and built by experienced developers so you can focus on your app rather than reinventing the wheel.” **(Django Project. 2025)***

Also is important to notice that it is true what it is written in the Djangoproject web site:

> *”With Django, you can take web applications from concept to launch in a matter of hours. Django takes care of much of the hassle of web development, so you can focus on writing your app without needing to reinvent the wheel. It’s free and open source”. **(Django Project. 2025).***

It is also true:

> *“Ridiculous fast, fully loaded, reassuringly secure, exceedingly scalable, incredibly versatile”. **(Django Project. 2025).***

Django was created in 2003 by developers at the Lawrence Journal-World newspaper and publicly released in 2005. It follows the Model-View-Template

(MVT) architecture, which separates logic, data, and presentation for better maintainability.

i. Model: Defines data structures using Python classes. Each model maps to a database table.
ii. View: Handles incoming HTTP requests and fetches or processes data to be displayed.
iii. Template: Uses HTML combined with template tags to generate dynamic content shown to users.

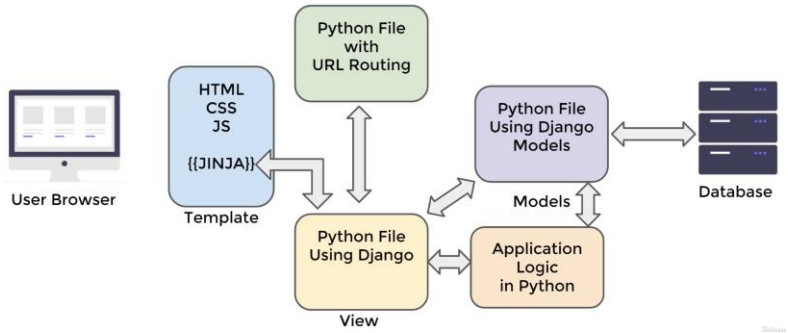


Figure 6. Model View Template layers.

Model View Template, From Django 4 and Python Full-Stack Developer Masterclass Created by: Jose Portilla, Pierian Training.

In this project, Django also integrates with the Django REST Framework (DRF). DRF provides features like serializers, view sets, token authentication, and pagination, making it ideal for building web APIs.

3.8 Security and Reliability with the Django Framework

Security is an essential component of any internet-based application, particularly when the system includes infrastructure provisioning, user admittance, and delicate server tasks. Django considers this factor carefully and has an array of intrinsic characteristics that aid designers build secure platforms out of the package.

One of Django's greatest benefits is that it applies protective defaults automatically, meaning that designers aren't needed to engineer each buffering from zero. It actively safeguards against many of the most widespread web weaknesses recognized by the *"OWASP Top Ten"* ***(OWASP, 2025)***

Vulnerabilities like:

i. SQL Interjection: Django utilizes a sturdy ORM (Item- relational mapping) system that mechanically escapes questions, which assists avert harmful inputs from tampering with the databank.
ii. Cross-Site Scripting (XSS): By default, Django templates auto-escape variables, which shields the application from user-submitted scripts being executed in the browser.
iii. Cross-Site Request Forgery (CSRF): Django includes CSRF protection in all POST forms. It uses a token-based validation system that ensures the request is coming from a trusted source.
iv. Clickjacking: The framework sets X-Frame-Options headers to prevent the site from being embedded in other websites, which helps prevent clickjacking attacks.
v. Cross-Origin Resource Sharing (CORS): Django can be easily extended with middleware to control how external resources interact with your backend.

Additionally, Django's user authentication and authorization system is crucial. It provides everything needed to manage users, passwords, roles, and permissions. Designers can define groups or individual access levels, restricting sensitive views to admin-only access or certain roles within an organization.

In addition, Django uses secure password hashing algorithms like PBKDF2, bcrypt, or Argon2. Passwords are never stored in plain text, and the system can be configured to enforce complexity rules or two-factor authentication if needed.

Django isn't just rapid and adaptable, it's also built to be safe. By configuring security as a default rather than an afterthought, Django offers developers a hard

foundation for building dependable, production-ready web systems without compromising on security.

Table 3. Common Web Vulnerabilities and Django's Built-in Protections.

Threat	Description	Django's Protection
SQL Injection	An attacker tries to inject SQL code into user input to manipulate the database.	Django ORM uses parameterized queries, preventing direct SQL manipulation.
Cross-Site Scripting	Injecting malicious scripts into pages viewed by other users.	Django templates auto-escape variables by default.
Cross-Site Request Forgery (CSRF)	Tricking users into performing actions without consent.	Django uses CSRF tokens in forms and rejects unauthenticated POST requests.
Clickjacking	Hiding malicious links behind clickable elements (e.g., iframe traps).	Django sets X-Frame-Options: DENY by default to prevent framing.
Password Theft	Storing or transmitting passwords insecurely.	Django hashes passwords using PBKDF2, bcrypt, or Argon2.
Insecure Sessions	Session hijacking or fixation attacks.	Django uses secure cookies, session rotation, and can enforce HTTPS-only sessions.

The framework automatically appends CSRF protection to any form that transmits data using the POST method. Here's what it looks like in a Django template:

```
<form method="post">
  {% csrf_token %}
  <input type="text" name="server_name">
  <input type="submit" value="Create Server">
</form>
```

Listing 5. CSRF Token in Django Forms

> *"The {% csrf_token %} template tag generates a hidden field that is rendered like this:*

```
<input type='hidden' name='csrfmiddlewaretoken'
value='26JjKo2lcEtYkGoV9z4XmJIEHLXN5LDR' />"

(Antonio Melé. 2024)
```

Listing 6. Example of a CSRF token rendered.

If the token is absent or invalid, Django will reject the form submission. This helps circumvent attackers from forging POST requests in the background via malicious websites.

Example 2: Password Hashing and Verification

When users create or update passwords in Django, they're never stored in plain text; they are stored using a hash algorithm. Here's a quick look at how Django handles password storage:

```
from django.contrib.auth.models import User

# Create a user with a secure hashed password
user = User.objects.create_user(username='admin', password='secure_password123')
```

Listing 7. Object user created with secure password.

Django applies the configured password hasher (e.g., PBKDF2 by default) and stores only the hashed result in the database.

To verify a password:

```
from django.contrib.auth import authenticate

user = authenticate(username='admin', password='secure_password123')
if user is not None:
    print("Authentication successful!")
else:
    print("Invalid credentials.")
```

Listing 8. Example of password verification code.

The examples above display that Django isn't just a friendly web framework; it's also engineered to prevent many of the vulnerabilities that are in existence in web applications.

By adopting secure defaults and with the help of tools like CSRF, hashing passwords, and automatic escape, Django allows developers to focus on building characteristics without needing to reinvent security foundations.

Particularly for cloud-based systems, the smallest oversight in security can have bad implications. A framework like Django offers peace of mind by supplying protection where most needed, at the intersection of infrastructure, data, and users.

3.9 The User Interface for Non-Technical Audiences

For non-technical audiences, the most important feature of a VPS management system is not just backend features and API integration, but also the user interface. Figure 7 shows the easy and simple dashboard created for the prototype.

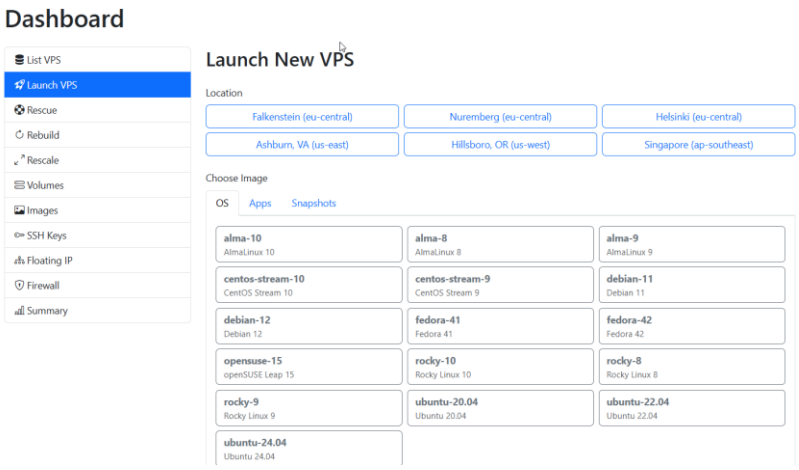


Figure 7. Dashboard of the Prototype.

For many small companies, customer assistance teams, or even freelancers and startup entrepreneurs, the interface frequently constitutes their sole means of interacting with the cloud platform.

If confusing, cluttered, or excessively technical, users will likely abandon it for more intuitive alternatives, regardless of reduced capabilities.

3.9.1 Usability as a Priority

> *"A core design principle for non-technical users is usability—how easily a person can learn and operate the system. According to Jakob Nielsen, a leading figure in usability research, a usable interface should be:*
>
> *i. Easy to learn*
> *ii. Efficient to use*
> *iii. Memorable*
> *iv. Minimally error-prone*
> *v. Pleasant to interact with"* ***(Nielsen, J. 1994)***

In the context of this thesis, that means reducing technical jargon, minimizing the number of steps required to create or manage a server, and using clear visual cues to guide users through tasks.

For instance, rather than showing full cloud configuration options, the interface can offer sensible defaults and tooltips that explain options in plain language. Figure 8 shows the style of the message the user will see when interaction with the prototype, in this case, the rescue mode shows "Boot your VPS into Hetzner's rescue System…"

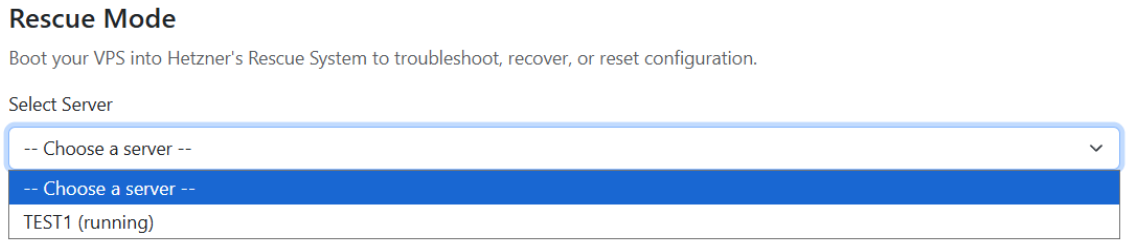


Figure 8. Configuration Option Example with tooltip about the Rescue Mode.

In this thesis context, that signifies simplifying technical language, minimizing the steps necessary to build or administer a server, and using clear prompts to assist users.

3.9.2 Design Patterns

Django's template framework, together with Bootstrap, permits building responsive, consistent, and clean interfaces. For example, using components like menus for choosing regions or features, modal windows for confirmation or showing passwords, icons and badges to indicate status, like running or stopped, and progress indicators for provisioning or rebooting.

Figure 9 shows the style of the design for choosing the location of the services.

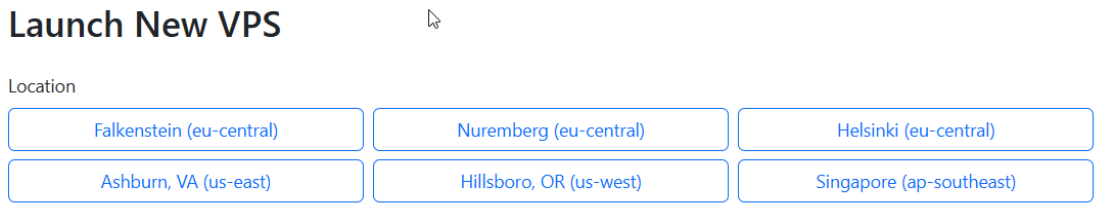


Figure 9. Selection of the Location of the Service when Deploying a VPS.

These components not just make the UI graphically attractive, but reduce the intellectual weight, which is crucial when the target audience isn't a specialist.

> A helpful idea here is the *"progressive disclosure"* technique, showing only the most basic information by default, and allowing more advanced options to be revealed on demand. This helps prevent users from being overwhelmed while still offering control***. (Cooper, A., Reimann, R., Cronin, D., Noessel. 2014.)***

3.9.3 Validation Through User Testing

One of the most solid approaches to ensure usability is to watch genuine people testing the software. Casual testing meetings with 3–5 people can uncover obstruction focuses that may not be evident even with improvement. These realizations are particularly significant for frameworks like VPS management systems, where a single error could prompt server deletion or data destruction.

Adding feedback like confirmation modals, undue, or read-only, before execution can fundamentally diminish the danger of client error. These little planning decisions provide certainty for non-specialized clients and soften the adoption.

In technical systems, effective interface design is often overlooked yet determines whether users feel empowered or intimidated. Prioritizing clarity, simplicity, and testing aims to deliver a genuinely usable system for more audiences without sacrificing functionality.

Whether a reseller, agent, or owner managing infrastructure, the goal remains the same: making advanced capabilities accessible without demanding advanced knowledge.

3.10 Case Studies on VPS Management Systems

To better comprehend the VPS management scenario, the examination of case studies shows the solutions for common challenges like automation, usability, pricing, and flexibility.

These examples provide valuable insights for what is successful, failure, and opportunities, especially for accessible alternatives developed targeting a broader audience.

3.10.1 Case Study: WHMCS + ModulesGarden Hetzner Integration

WHMCS, popular within hosting providers for billing and provisioning, where ModulesGarden develops a commercial plugin integrating WHMCS with Hetzner Cloud API to deploy, reboot, resize, and administer servers through the client area.

> *“Hetzner Cloud Servers For WHMCS is a module cleverly designed to automate the provisioning as well as all routine tasks in the management of Hetzner Cloud virtual machines. Having a free choice from a range of configurable options, your clients will easily*

order servers suited to all their personal preferences." ***(Modulesgarden, 2025)***

The Modules Garder module for WHMCS has a very similar configuration to the prototype, as shown in Figure 10.

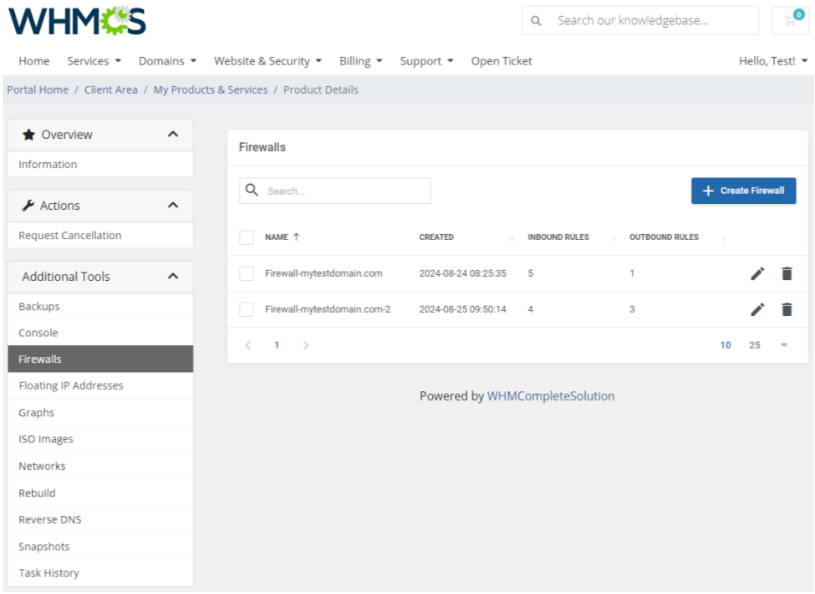


Figure 10. Modules Garden Module for Hetzner.

Strengths: Seamless integration with billing and support systems, multi-client access with roles and permissions, and time-saving automation for server lifecycle tasks.

Limitations: The solution is closed-source and protected with ionCube encryption, customization options are limited unless you pay for extended licensing, and The interface is heavily technical, with minimal UX optimization for non-admins.

This case highlights a familiar tradeoff: convenience through integration versus flexibility and transparency losses. It also spotlights gaps in non-technical usability addressed by the prototype.

3.10.2 Case Study: DigitalOcean Control Panel.

> *"DigitalOcean offers solutions that cater to the needs of those just learning to code, builders getting started on their big idea, and scalers, businesses that are growing rapidly and need a cloud solution that will scale alongside them. Putting simplicity first, we provide a range of cloud computing solutions, from virtual machines to managed Kubernetes, that are tailored to the needs of startups and small businesses.".* ***(Digital Ocean. 2025)***

DigitalOcean gains praise for its clean, intuitive UI where users launch servers (Droplets), attach storage and firewalls, and monitor traffic, all through a minimal, approachable dashboard, in under sixty seconds. Indeed, it looks clean, but at the same time is possible to see in the next Figure 11 how complex the selection of products and services for customers are.

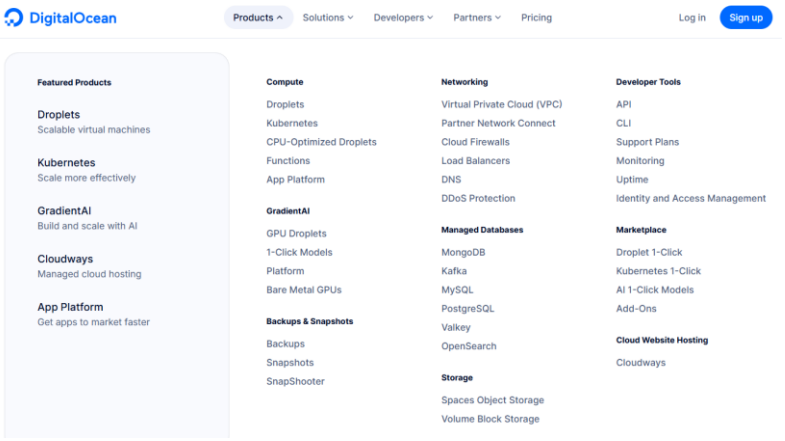


Figure 11. DigitalOcean menu product selections.

Strengths: Highly usable interface with excellent onboarding and documentation, API and UI are equally powerful and consistent and clear separation between beginner-friendly features and advanced options.

Limitations: Locked into DigitalOcean's infrastructure, no portability, advanced features require account upgrades, not open-source, and no white-label option.

While DigitalOcean offers a refined experience, it reinforces vendor lock-in. For users looking for custom branding or control over infrastructure sources (like resellers), it doesn't offer the openness needed.

3.10.3 Case Study: OpenNebula for Private VPS Hosting

> *"OpenNebula is a powerful, but easy-to-use, open source platform to build and manage Enterprise Clouds. OpenNebula provides unified management of IT infrastructure and applications, avoiding vendor lock-in and reducing complexity, resource consumption and operational costs."* ***(Opennebula, 2025)***

OpenNebula allows full virtualization, network automation, and storage management, and has been deployed to oversee thousands of virtual instances at places such as CERN and Telefonica. As Figure 12 shows, this is a very complex far beyond reach for starters, and a very high learning curve.

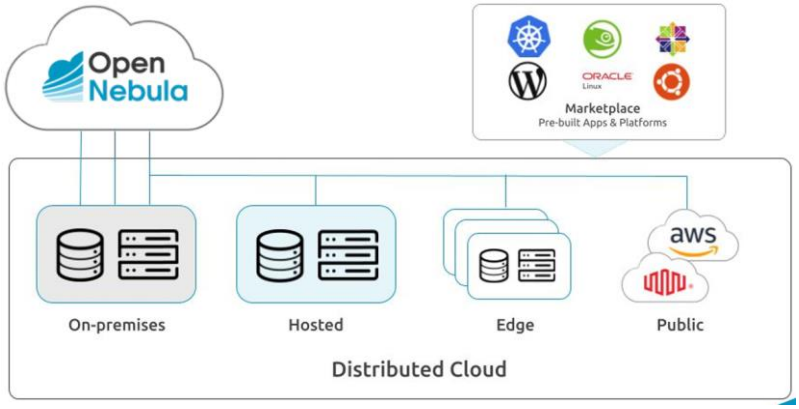


Figure 12. Cloud Management Cluster. ***(Opennebula, 2025)***

Strengths: Full control over infrastructure, highly customizable and extensible, and enterprise-grade scalability.

Limitations: Steep learning curve; requires significant DevOps expertise, overhead in setup and maintenance for smaller teams, and no built-in billing, branding, or lightweight UI by default.

This case demonstrates what's achievable with open-source cloud orchestration, but also underscores that not every organization necessitates that level of intricacy. For resellers or smaller businesses, a lighter tool constructed on Django and Hetzner’s API may be far more practical and suitable.

3.10.4 Lessons Learned

These case examinations highlight several key lessons that directly inform the direction of this thesis.

i. Simplicity is a competitive advantage, especially for users without deep infrastructure experience and know-how
ii. Customization and branding are often overlooked by large providers but are vital for resellers.
iii. Open-source access allows developers to innovate, adapt, and avoid vendor lock-in.

By drawing from the strengths of these platforms and avoiding their pitfalls and shortcomings, the system designed in this thesis aims to offer a middle ground: powerful enough to be useful, lightweight enough to remain manageable, and open enough to be reused across providers.

4 System Design

This section outlines the architectural foundation and design principles used to develop the VPS management system. It begins with a detailed look at the system and database design, focusing on the models and relationships required to support the functionalities such as user registration, server provisioning, and resource tracking.

Next, it presents user interface prototypes specifically tailored for non-technical users, highlighting usability and clarity as central goals.

Also, it unveils the documentation that supports consistent execution, extensibility, and maintenance. Collectively, these aspects guarantee that the system is both robust in function and accessible in practice.

4.1 Detailed system and Architecture

The system employs the Django framework, which adheres to the Model-View-Template (MVT) structure. This composition provides a separation of tasks, building a system easy to expand, maintain, and scale.

At the core of the application is a set of custom-made models that handle user data, authentication, server provisioning, and integration with the Hetzner Cloud API.

The application consists of the following major components:

i. Models define the information structure and impose business rules at the database level.
ii. Views include the logic for dealing with HTTP requests and interfacing with Hetzner’s API.
iii. Templates render dynamic content to the user, offering a reactive and intuitive interface.

The system utilizes MySQL as the relational database backend, with Django's ORM layer giving the abstraction over raw SQL.

4.1.1 Custom User Model

Central to the authentication system is a custom model entitled CustomUser, which extends Django's AbstractBaseUser and PermissionsMixin. This model replaces the default username with an email-based login and incorporates extra fields tailored for customer identity and billing.

Key fields include:

i. Personal information: first_name, last_name, email, phone_number
ii. Billing and legal info: company_name, street_address, vat_number, y_tunnus
iii. Security details: security_question, security_answer, date_of_birth
iv. Tokens: api_token, confirmation_token (used for email verification)

Each user is affiliated with a unique API token, which allows access to Hetzner's cloud interface from within the dashboard.

```
api_token = models.CharField(max_length=64, unique=True, blank=True, null=True)
```

Listing 9. API token field generated through the Model.

4.1.2 Relational Model Design and Database Schema

The database design is normalized and modular. While the CustomUser model stores user-specific data, other system components, like VPS instances, volumes, and snapshots, are fetched live from Hetzner via API rather than stored persistently. This decreases the need for complex foreign key relationships and ensures real-time accuracy.

Django's inherent Session, Group, and Permission models are also in use for access control and administrative grouping.

The schema stores each client's identity records, contact, invoicing, and privacy. This enables user enrolment, identity verification, API secret, and individual

access. Meanwhile, the framework customizes Django's classifications to satisfy the needs of VPS users.

4.1.3 Primary Database Table: custom_user

Table 4 shows the schema created by Django custom user Model. The table is generated automatically in the database after the migrations are executed. The migrations are the execution code for the Django Models.

Table 4. MySql Table custom _user.

Field Name	Type	Description
id	AutoField (PK)	Primary key.
first_name	CharField(30)	User’s first name.
last_name	CharField(30)	User’s last name.
email	EmailField (unique)	Used as the primary login credential.
phone_number	CharField(20)	User’s phone number.
company_name	CharField(100, blank)	Optional company name.
street_address	CharField(255)	User’s main address.
street_address2	CharField(255, blank)	Secondary address line.
city	CharField(100)	City name.
state	CharField(100)	State or province.
postcode	CharField(20)	Postal code.
country	CountryField	ISO code of user’s country.
vat_number	CharField(50, blank)	Optional VAT ID.
date_of_birth	DateField	Required for identity and verification.
nationality	CountryField	Country of citizenship.
y_tunnus	CharField(50, blank)	Optional Finnish business ID (Y-tunnus).
security_question	CharField(255)	Predefined or custom security question.
security_answer	CharField(255)	Encrypted or hashed answer to security question.
is_active	BooleanField	Enables/disables account access.
is_staff	BooleanField	Grants admin panel access (for staff users).
api_token	CharField(64, unique)	Token for authenticating with Hetzner Cloud API.
email_confirmed	BooleanField	Tracks whether the user verified their email.
confirmation_token	CharField(64, blank)	Temporary token for verifying email addresses.

4.1.4 Authentication Behavior

USERNAME_FIELD = 'email' (Replaces the default username with email for login)
REQUIRED_FIELDS = ['first_name', 'last_name']. (These are enforced during superuser creation or management commands).

4.1.5 Token Management

api_token: is generated by Hetzner and treated like secrets.token_hex(32) during the first save.

confirmation_token: is generated for email confirmation and can be refreshed using the generate_confirmation_token() method.

4.1.6 Related Django Tables (built-in)

The custom_user model inherits from Django's PermissionsMixin, enabling links to:
auth_group: roles/groups of users
auth_permission: fine-grained access rights
django_session: active login sessions

4.2 Model Design Considerations

The custom user model was chosen from the beginning because the default user model uses a login name and password, and the ideal for a login system is to use the email as a login name, plus a password. This is easy to achieve by overriding the classes.

Also, the possibility of extending features like integrating modules for billing, user levels, permissions, or audit logs, or other providers, requires the use of a custom user model.

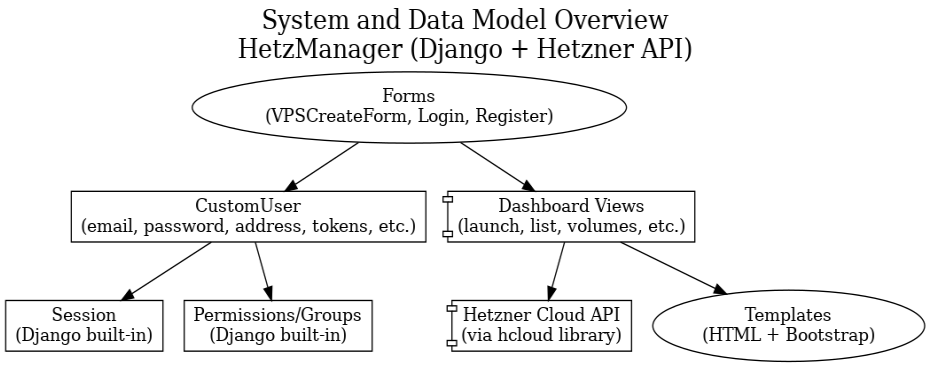


Figure 13. Hetzner and Django Interaction

4.3 Authentication and Validation

The detailed registration form (RegistrationForm) handles all validations, from password matching to CAPTCHA solving and nationality selection using django-countries. Designed with both security and usability in mind, its logic confirms passwords and enforces unique emails.

```
def clean(self):
    cleaned_data = super().clean()
    if cleaned_data.get("password") != cleaned_data.get(“confirm_password”):
        self.add_error('confirm_password', "Passwords do not match")
```

Listing 10. Validation code for user password.

This prepares users for interacting through APIs while maintaining integrity at the application level.

4.4 User Interface Prototypes Focusing on Usability

The visual design prioritizes clarity, accessibility, and ease of use, especially for those without experience. By borrowing familiar patterns and lightweight, responsive components, it provides a simple environment where both technical and non-technical users can deploy and manage servers with confidence.

Built on Django templating with Bootstrap 5, Font Awesome, and Bootstrap Icons for their simplicity, consistent components, and community support, rapid prototyping resulted in a modern and mobile-friendly design.

i. HTML/CSS Templates: All UI pages are extended with a single base..
ii. Icons and Feedback: Icons are used to improve navigation and reduce the need for textual instructions.
iii. Forms and Validation: All forms use clear labels, placeholders, and inline validation to help users and avoid mistakes.

The dashboard features a sidebar for navigating sections grouped by task and labeled intuitively with names and icons, such as:

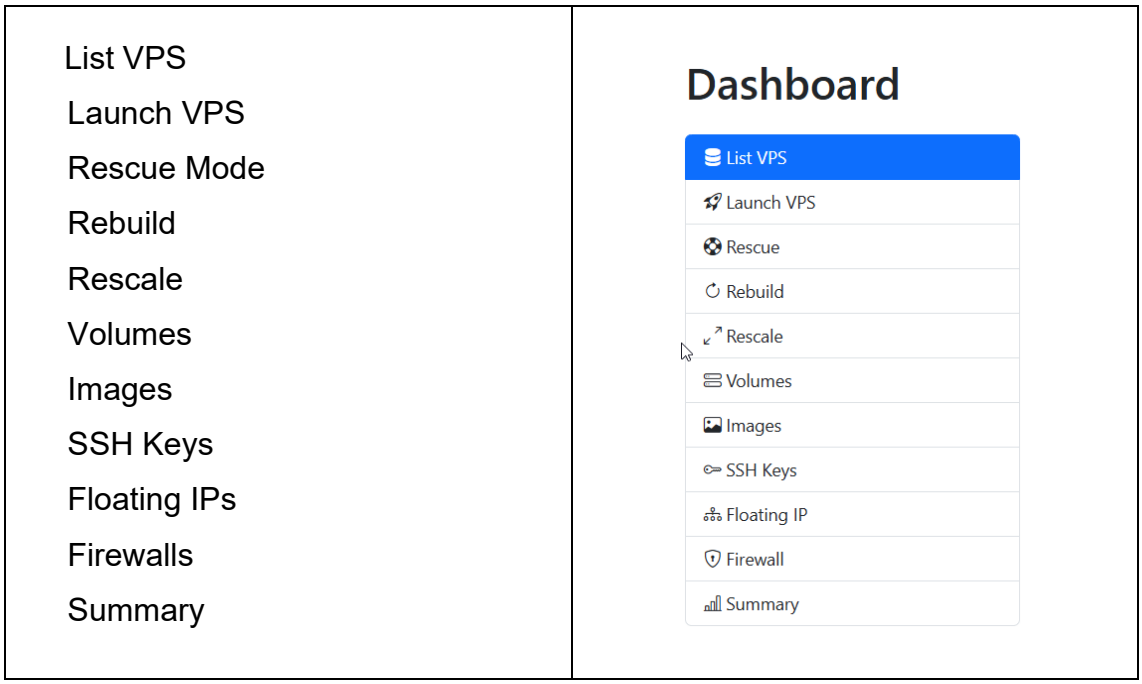


Figure 14. Menu list of the prototype.

Once the user is inside the Dashboard, it remains there. This allows changes to the views or options without losing the layout.

4.4.1 Prototyping for Non-Technical Users

To avoid complexity in system where it is not necessary, the UI uses the following principles for good usability:

i. Progressive Disclosure: Advanced features like protection flags and custom labels are hidden until necessary.
ii. Confirmation Modals: If you are about to take destructive action, like, for example, remove a server or volume, you must confirm your action using a modal dialog.
iii. Tooltip Guidance: Tooltips and context-aware hints to explain technical terms such as “Snapshot” or “Floating IP”.

Figure 15 shows an example of what a tooltip and context-aware is, as it hints and explains the difference between snapshots and backups.

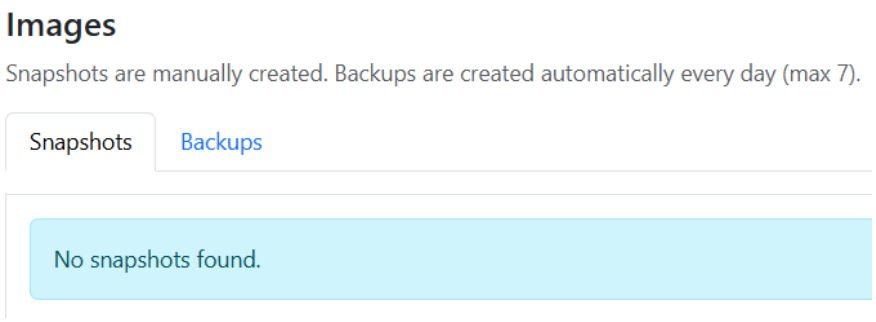


Figure 15. Images view. Snapshot and Backups.

The Launch VPS form (dashboard_launch. html) reduces this to a few fields: Server Name, Location (dropdown), Server Type (dropdown), Image (OS/Application), and SSH Key selection

Figure 16 demonstrates how simple it is to navigate between the options to create a new VPS. The whole process can be done in a few seconds. The VPS is built, deployed, and delivered in less than 10 seconds.

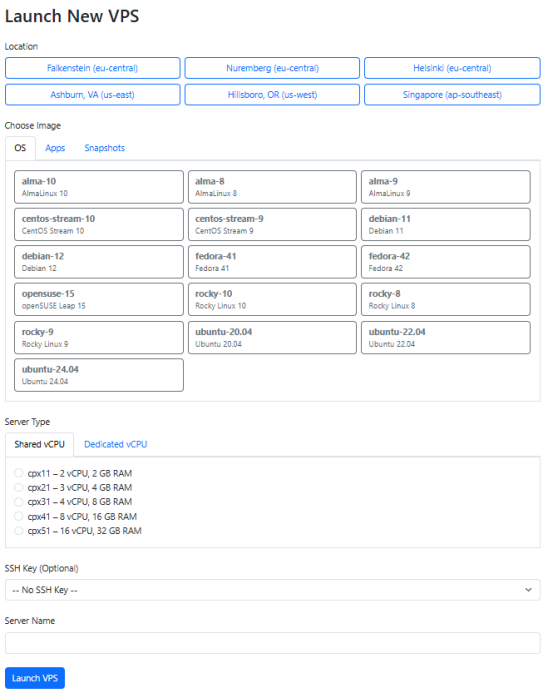


Figure 16. The Launch New VPS complete view.

Every dropdown is populated live from Hetzner's API, giving the user an always updated set of options, and hiding fields like networking or firewall rules that are not needed.

4.4.2 Responsive, Accessible Design and User-centred Considerations

The system was built with responsiveness. Users can access and interact with the system from smaller screens, tablets, or smartphones with the same set of functionalities. Additionally, form fields are large enough for touch interaction, statuses are indicated by color-coded labels like green for “running”, red for “stopped”, and text contrasts meet accessibility.

During the prototyping phase, users with non-technical expertise tested and provided some key insights:

i. Dropdown menus should avoid abbreviations, like fsn1 becomes Falkenstein in Europe Central.

ii. Users appreciated having separate OS images and applications in different tabs.
iii. Labeling buttons with verbs (“Launch VPS”, “Enable Protection”) improved clarity over generic terms (“Submit”, “Save”).

Figure 17 shows the selection of buttons and actions that can be applied to a VPS; all action is labelled and easy to understand.

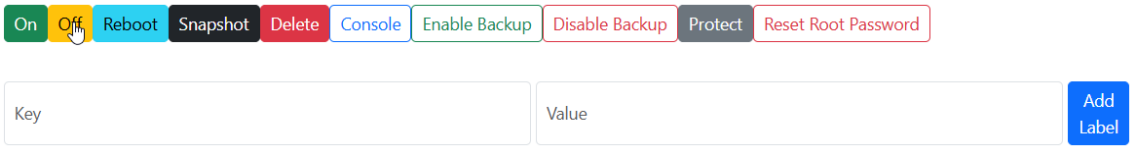


Figure 17. Action Buttons.

4.5 Design Documentation for Implementation

Good design documentation is essential for new users to be able to pick up an existing system and implement further needs or changes. Documentation in this project bridges the gap between the logical design, the code implementation, and the functional aspects of a production system.

The documentation covers three main areas: system architecture and configuration, sode-level documentation and docstrings, and inline comments.

4.5.1 System Architecture and Setup

For system documentation, the guide describes how to create a development and a production environment. This includes:

i. Environment configuration using “.env” or Django’s “settings.py” structure
ii. Dependencies listed in “requirements.txt”
iii. Optional use of “venv” or “pipenv” for isolated Python environments
iv. Reverse proxy and WSGI server setup like Apache + mod_passenger or Gunicorn + Nginx.

Each aspect is detailed with install instructions, anticipated file locations and security notes, including secret key handling and API token management.

```
export HETZNER_API_TOKEN=your-token-here
python manage.py runserver
```

Listing 11. Representation of the Token when communicating with the API.

4.5.2 Code Documentation and Readability

Models, views, and Forms also contain a brief Python docstring. This way, developers can immediately get an idea of what each class, function, and method does without having to go digging into the code to do that.

Examples include Models like CustomUser with descriptions of each field, Views such as dashboard_launch, annotated to explain how form data is converted into requests, and Forms like VPSCreateForm, detailing validation rules and dynamic choice handling.

Logging statements, using the Python logging module, also allow to log API calls, failed operations, and user activity, facilitating an auditable implementation.

4.5.3 API Integration Strategy

The integration with the Hetzner Cloud API has its own, dedicated section of the documentation, including: authentication: API tokens are persisted and checked per-user upon login or API request, respectively, request/response handling: An example of what those requests (and their responses) look like in curl and Python hcloud client for create a server or seeing the metrics, and Error handling: Typical API faults, server locked, or no image, are gracefully presented back to the user with error logs.

The endpoints, like servers, images, and metrics, are documented with their parameters and expected payload formats.

4.5.4 File Structure and Roles

The file structure of the application is described with comments and the descriptions of roles are:

```
/HetzManager/                          # Root Project
/HetzManager/hetzmanager.log           # Log file
/HetzManager/manage.py                 # Command line utiliry
/HetzManager/requirements.txt          # List of Modules
/HetzManager/HetzManager/              # Main Default Project Application
/HetzManager/HetzManager/settings.py   # Configuration and settings
/HetzManager/HetzManager/urls.py       # paths at project level
/HetzManager/identity_app/views_admin.py  # Custom admin
/HetzManager/identity_app/models.py    # Data schema
/HetzManager/identity_app/views.py     # Classes methods and functions
/HetzManager/identity_app/urls.py      # Application paths
```

This allows new developers to orient themselves quickly when working on specific parts of the system.

5 Working Prototype

HetzManager, the working prototype for the VPS management system, was developed to show that building a modular white-label control panel which integrates with Hetzner's Cloud API is in fact possible!.

Built with the Django web framework, the system is designed to provide a secure and easy-to-use front end combined with a robust backend automation system that enables users to start up, configure, and monitor cloud VPSs without having to log in directly to the Hetzner console.

The app wraps around basic VPS lifecycle functions, providing an interface to manage server setup, snapshots, firewall, storage, and usage statistics for resources, all through a Bootstrap layout.

The product was designed as an API-first, extensible, and user-friendly product for developers, as well as nontechnical customers looking for an easy-to-use server management experience.

5.1 Backend Implementation with Django and Hetzner API

The backend of the HetzManager system was designed and implemented using Django, which was chosen based on the very good security system implemented with it, its modularity, and the speed of development with it.

Django’s MVT allows for separating concerns effectively, leading to well-organized authentication, API requests, form processing, and error management.

The system is build using a custom user model (CustomUser) which is derived from Django’s AbstractBaseUser and supports login with a email address, custom personal information, security questions, VAT fields, and for API tokens. CAPTCHA-protected forms are present in registration and log-in to fight against bots.

Figure 18 shows the email verification modal for the customers after their first time registration.

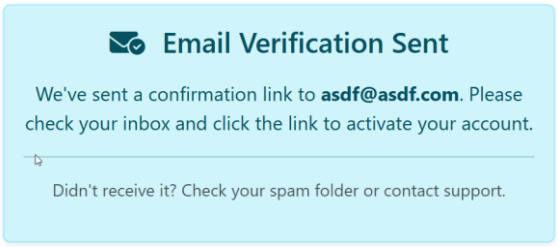


Figure 18. Email verification modal.

New users need to validate email with secure link token before logging in. This verification flow is done through token creation and base64 encoded user-ids, and a secure URL handled by the confirm_email views.

5.1.1 API Token Integration and Access Control

A Hetzner API Token (HETZNER_TOKEN) is stored securely per user account and is used to authenticate requests made against the Hetzner Cloud API. Token validity is verified during the sign-in process and retested during queries to the hcloud.

The user is redirected to a special "access denied" page in cases where the token has not been provided or has expired in an attempt to terminate any backend operation without explicitly triggering the authorization view.

Also available for external systems for token-based access is a lightweight /api/resource/ endpoint. It checks the Authorization header token and if valid, it will respond with a signed token confirmation.

5.1.2 Server Lifecycle Management and Modular Dashboard Architecture

The dashboard can execute all server lifecycle operations:

i. Launch: Users can define the server type, image (OS/app/snapshot), location, SSH key in a structured form. No SSH key: resets the root password and displays it securely in views.
ii. Actions: Users are able to to power on/off, reboot, delete, label, snapshot, restore, and reset the root password of server via the dashboard_actions endpoint interface dashboard_actions.
iii. Metrics: Traffic, CPU and disk I/O stats are collected from Hetzner's prometheus-based metrics endpoint and presented in the UI views.

All of these features are built on top of the hcloud python library that has full Python bindings for the Hetzner REST API, removing the need to write interaction logic with servers.

Backend views are following a modular route scheme (/dashboard/section/<section>/) to promote component decoupling and maintainability. Each section is separated into a dashboard_section view that can host its own VPS listing, for launching a VPS, for resource summary, or for SSH key management.

Other dedicated routes include:

/dashboard/rebuild/	Reinstall a server from an OS, app, or snapshot or image
/dashboard/volumes/	Manage block volumes like create, resize, attach and delete
/dashboard/firewall/	Create custom firewall groups with inbound and outbound rules
/dashboard/images/	View and delete snapshots and backups
/dashboard/rescue/	Enable or disable rescue mode and reboot the server
/dashboard/floatingip/	Manage floating IPs with region

Every action is signed, checked and logged on the server before communicating with Hetzner's API for security and traceability.

Figure 19 shows the logs created by Django for every execution the customer does inside their dashboard. This is accessible only to administrators of the site.

Different message types, like info, warnings, emergency, and basic alerts, can be displayed.

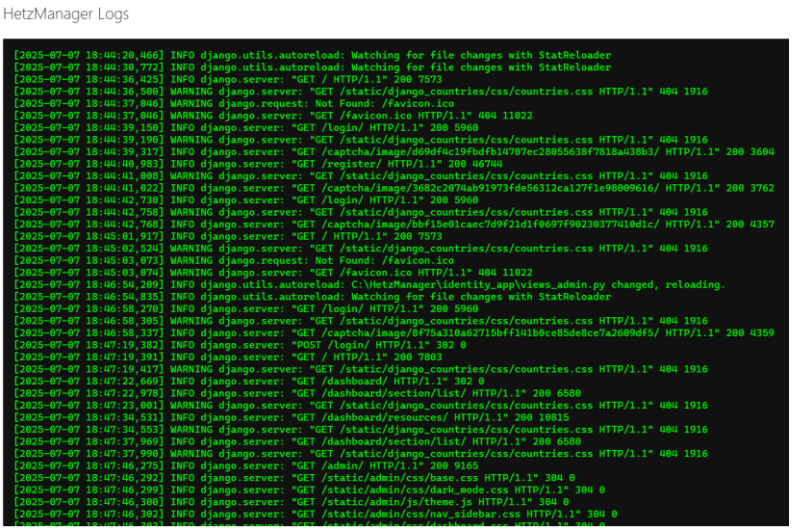


Figure 19. Log screen from the admin panel.

5.2 Frontend Implementation and Responsiveness

The backend of HetzManager itself is a standard Django application with Djngo templates and Bootstrap 5 to power the responsive and modern user interface. The main objective was to develop an easy-to-use dashboard for novice users with an easy way to manage VPS and infrastructure.

5.2.1 Template Structure Layout and Sidebar-Based Navigation

The site uses a common base template (base.html) which contains a responsive navbar, sidebar, and content areas. Everything is built using Bootstrap 5 grid, so there is no need to worry about having problems when using the site on desktop or mobile devices. Visual flair comes from contextual button style, modals, badges, and icons using Bootstrap Icons FontAwesome.

A vertical sidebar provides access to all major sections of the dashboard: List VPS, Launch VPS, Rescue / Rebuild / Rescale, Volumes, Snapshots & Backups (Images), SSH Keys, Floating Ips, Firewall Rules, and Resource Summary.

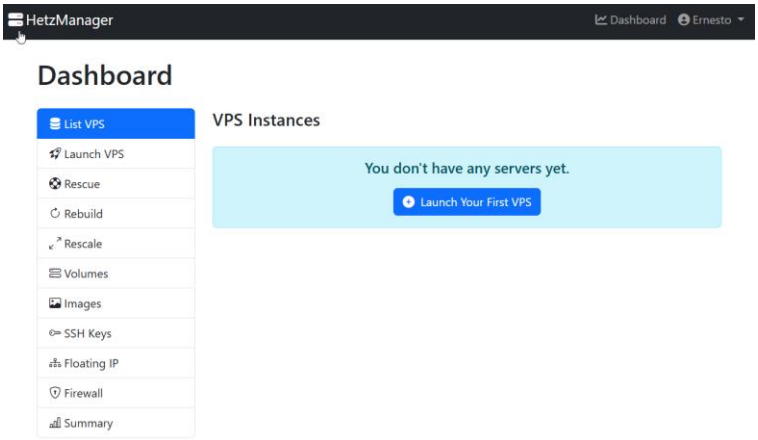


Figure 20. Sidebar-Based Navigation

Each section is highlighted using active logic for accessibility and user orientation on the dashboard.

5.2.2 Interactive and Contextual Actions

Interactive features include dynamic forms, accordion-style views for resource lists, and modal overlays for confirming root password reset and deleting the server. Server cards and volume tables include contextual action buttons such as power toggle, reboot, management, and snapshot restore, all grouped using buttons in the dashboard_actions.

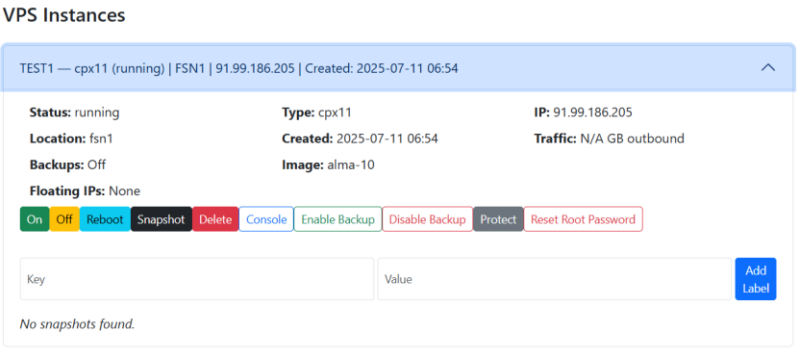


Figure 21. The accordion opens after selecting the VPS TEST1.

Users also have visual feedback using the Django message framework for following operation success/error alerts. I will style these alerts with Bootstrap's alert components for a more consistent and clear look.

Figure 22 shows the message to the user after enabling the rescue model.

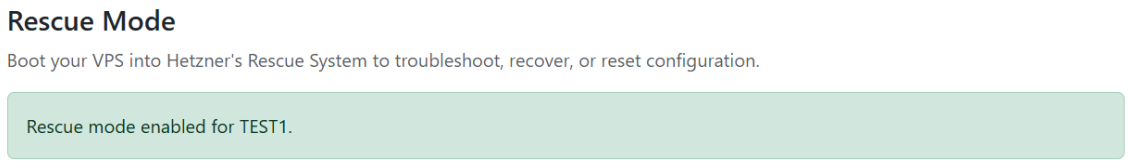


Figure 22. Feedback after selection Rescue mode.

5.2.3 Usability for Non-Technical Users

Design decisions emphasized accessibility and simplicity with structured forms with a placeholder that helps with the text and dropdowns, required field validation for both client and server, default values like the EXT4 file system for volumes, and English instructions, like for example, tooltips, section headers, and labels.

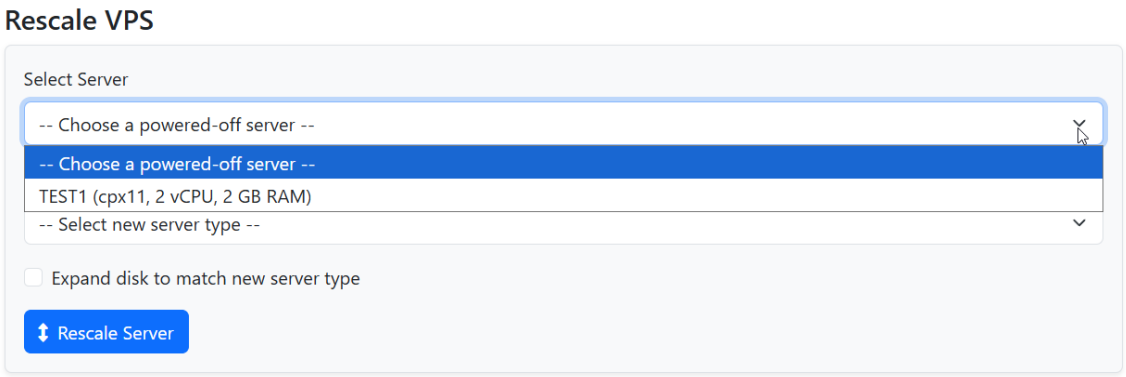


Figure 23. View of the Rescale VPS.

Each section is enclosed in its own view, and with this, it minimizes distractions and overhead so people can concentrate on a single task at a time, like starting a server, managing volumes, or editing firewall rules.

5.3 Accessibility, Cross-Device Testing, and Security Measures

A responsive nature was verified in standard screen sizes of smartphones, tablets, and PCs. Touch-friendly and scalable fonts were selected, and labels applied to key components, including the sidebar navigation, to provide support for assistive technologies.

Because HetzManager directly manipulates cloud infrastructure and handles sensitive user data, security also had to be one of the highest priorities while developing it.

Multiple layers of safeguards both at network and application level protect you from unauthorized access, misuse, and also from spamming attempts and other automated attacks.

A “Completely Automated Public Turing test to tell Computers and Humans Apart” (CAPTCHA) mechanism is a simple question designed to prevent bots from submitting buttons, since the information is posted only if the CAPTCHA has been answered.

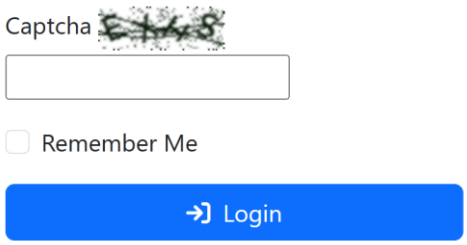


Figure 24. Captcha from the User Login View.

Figure 24

5.3.1 Email Confirmation Workflow

In order to prevent users from creating fake or unverified accounts, new users must verify their email, using a tokenized link sent to their profile email following their registration. This process involves:

i. Generating a secure 64-character confirmation token.
ii. Encoding the user ID in base64.
iii. Sending a custom email with a confirmation link (/confirm-email/<uidb64>/<token>/).
iv. Activating the account only after the user confirms via this link.

The confirmation status is stored in the email_confirmed field and is strictly checked before login is permitted on the views.

5.3.2 API Token Validation and Isolation

Every user comes with their own Hetzner API token, which is stored in the database in a safe manner and only used for making authenticated API calls. These tokens are also checked at login that they are still active and will resolve.

An API token access end point (/api/resource/) is available for clients to integrate, secured by a tight header authorization scheme. Invalid or missing tokens block access to the dashboard, separating backend logic from improper use and avoiding incomplete credential issues.

It may be exaggerated, but it does not hurt that the next code is at the beginning of every view that uses the dashboard, which means if there is no token or the token is malformed then it will redirect to a custom page for a 401-error message.

```
token = request.headers.get('Authorization')

if not token or not token.startswith("Token "):
    logger.warning("API access denied: missing or malformed token.")
    return JsonResponse( data: {'error': 'Missing or malformed token'}, status=401)

token_value = token.split(" ")[1]
```

Listing 12. Code for verification of existence of API Token.

5.3.3 Session, Access Control, and Admin Interface Hardening

To restrict views to authenticated users we rely on Django's built-in authentication and session middleware, which blocks unauthenticated users from accessing protected views.

The views like dashboard_section, dashboard_launch, dashboard_volumes etc all have @login_required blockage against people not signed in.

```
@login_required
def dashboard(request):
    """
```

Listing 13. Login Required present on every dashboard view.

For admin-only features like log display views_admin, the @staff_member_required decorator is applied to only allow access to staff, which is another manifestation of privilege separation.

The Django admin interface is isolated behind CAPTCHA-based login for all users, check that blocks non-staff users from accessing the /admin/, even if authenticated, and read-only API tokens and confirmation tokens are displayed in the admin for traceability.

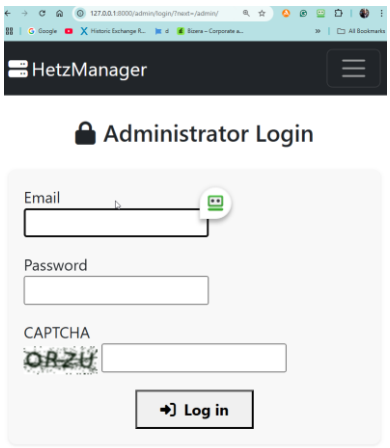


Figure 25. Custom Admin Login View.

These help ensure that administrative access is deliberate, traceable, and resistant to automated attacks to the custom_admin.

5.3.4 HTTPS and Deployment Recommendations

The development environment is of course running over HTTP as, however the system is intended to only be deployed behind HTTPS. All Session cookies, email passwords and all API Tokens are to be encrypted in transit with TLS.

The system also provides for SMTP-based e-mail delivery of messages for production use (setup via the EMAIL_BACKEND) in settings.py.

5.4 Working Prototype

The result of this development was a working prototype that implements all essential aspects of a VPS management system by means of the Hetzner Cloud API. This system has oversight for the entire life cycle of the virtual servers from instantiation, through management, resize, rebuilds and monitoring of VPS. It also offers utilities for controlling volumes, snapshots, firewalls, SSH keys, and floating IPs all in one place.

The prototype was validated with functional and explorative scenarios to verify that the main workflows worked as expected, including server provisioning, action execution, and communication with the API. The dashboard UI was tested on different devices and user roles and several API tokens were tested for checking secure token-based isolation.

The full implementation source code of this thesis is provided in annex. The particular codebase structure is modular and extensible, and can be easily extended for other cloud providers sharing an API structure similar to Amazon.

6 Testing and Validation

The success of the HetzManager prototype is marked by the HetzManager prototype should work "as expected", it should work with real API interactions, and it should meet the needs of technical and non-technical users.

6.1 Functionality Testing

Functionality was tested at various stages of development to ensure that all features worked as expected. Base operations like server start, stop, restart, rebuild, snapshot taking, volume managing, firewall rule creating, floating IP adding and SSH key managing were all tested against live Hetzner API endpoints.

The system was also tested at different configurations, such as server type, region, and image type, OS, apps, and snapshots. The interface was tested in both successful and unsuccessful states, including edge cases such as bad user input or a momentarily unavailable API response. No blocker or critical issues were found in this process, and all of the planned functionality was reportedly functioning properly.

For testing usability several users with different knowledge of technical skills tested HetzManager. Developers were able to answer with knowledge only a developer would know, while even non-technical users were asked questions such as how to start a VPS, attach a volume, do a rescue mode, or configure a firewall.

Feedback suggested that the dashboard was easy to use, and it is believed it was due to the sidebar navigation, clean design and good labeling. Although the UI design was labeled “basic,” users commented that it worked and was comprehensible.

Recommendations for improvement would be to provide more tooltips, more real-time feedback and inline validation but these are possibilities for the future.

6.2 User Testing

Developers and non-techies were asked to accomplish tasks like starting a VPS, connecting a volume, activating rescue mode, or setting up a firewall.

Feedback received showed that the dashboard was easy to navigate with the sidebar nav, clean design, and appropriate labeling. The UI is visually "basic," but users found it to be functional and easy to use.

Suggestions for more realism included adding further tooltips, more live updates, and in-line validation, all of these could be considered as improvements for the future.

6.3 Validation Report

The application has been stable during testing, and no showstopper bugs were found in the flow. The backend code is clean, easy to read, and documented through comments and Python docstrings for maintainability.

The application is implemented with modular Django views and forms, providing simple ways to add new features. The application utilizes email validation, captcha, access control, and robust API token matching the service.

Even though the UI is functional, it is still a very simple design. Improvement can be done in the future with other JavaScript libraries, which could positively affect the user experience. Simple Templates may be useful, but they don't use fancy JavaScript interaction, dynamic UI updates can be implemented with, for example, AJAX.

6.4 Automated Testing

The system includes a tests.py module using Django’s TestCase class and *“Python’s unittest.mock library*”. **(Python Documentation, 2025).**

The tests themselves concentrate on feature fundamentals such as:

i. Validations: Validates registration form for Password mismatch error etc.
ii. User Authentication: Check if a registered user with authenticated email and CAPTCHA can log in, and an unregistered one cannot.
iii. API Token Access: It verifies whether the users have the valid token to access the API resource endpoint or not, and the invalid token requesters are rejected.
iv. Dashboard access: Any non-authenticated user is redirected to the login, authenticated users can visit the dashboard.
v. Volume Management: Simulates the volume creation journey with a mock of API response to test that the expected data is correctly managed and validated.

These tests give people confidence in the reliability of the system and lay the basis for subsequent testing and CI/CD pipeline integration.

The HetzManager prototype was evaluated over multiple users, with different API tokens, and diverse configurations. All core assembly worked as designed, and the system was capable of its basic purpose. The codebase is production-ready and acts as a basis for extended development by others.

7 Conclusions and Recommendations

The creation of HetzManager shows that a modular, customizable, user-friendly and open source VPS management platform is possible and that it worth making it based on the Hetzner Cloud API.

The project intended to solve problems with available proprietary and restricted licence solutions by providing an open, Django-based free alternative able to act as a base for others to develop their own control panels.

The fundamental goals set up at the beginning of this thesis were:

i. Create a functional web based control panel for managing VPS in Django.
ii. Hook into the Hetzner Cloud API to manage servers, volumes, firewalls, images, and networks.
iii. An easy to understand and user friendly UI for technical and non-technical users.
iv. Security enforcement, including protection through CAPTCHA, email verification, and token-based API access.

All objectives were successfully achieved. The resulting prototype provides support for complete VPS lifecycle management, server actions, image processing, volume operations, firewall, floating IPs, as well as a summary of the resources.

The reliability of the system was tested by several users at various skill-levels, and system stability, security and practicality for real-life application were confirmed.

Based on the thesis project, it can be stated that a custom control panel for VPS infrastructure can indeed be implemented using Django and the Hetzner API and that the control panel can be useful for people with a need for customization of the code.

The next is a list of considerations for organizations or developers interested in cloud management systems:

i. To start with Hetzmanager and have a minimal prototype that has been tested and documented.
ii. Focusing on user experience and operational security.
iii. New cloud platforms and features can be added with minimal effort.

The complete source code of the project is provided on GitHub. Readers and contributors are free to extend this work for their specific use case. They can do whatever they want with it.

Proposed Enhancements: Although the prototype delivers its primary objectives, the following improvements may increase the value to users and ease of use:

i. UI Improvements: The current UI is minimal on purpose. Next versions will be based on a more advanced UI framework, like JavaScript or React, for a more interactive user experience.
ii. Role-Based Access Control (RBAC): Providing additional roles to users (e.g., admin, developer, billing-only) could improve defining access control.
iii. Multi-Cloud support: Abstracting the API client logic would enable to easily integrate others cloud provider like AWS, DigitalOcean, Vultr, OVH, and make the platform independent from the cloud.
iv. Audit Logs and Billing Dashboards: Logs of user activities and summaries of monthly usage and charges could also help operational and administration needs.
v. Two-Factor Authentication (2FA): To further secure your account you can include 2FA using the likes of django-otp.

It’s deliberate that these improvements are left open for future contributors and as such, the project’s aspiration to be a white-label, reusable foundation.

8 Summary

This thesis is about the development of HetzManager, a custom-built control panel designed to manage virtual servers using the Hetzner Cloud API. The idea came from a simple need: most VPS management tools today are expensive, closed-source, and tied to a single provider. The goal was to build something open, flexible, and easy to use, especially for people who aren’t experts in cloud infrastructure.

Using the Django framework, the system was made to handle everything a user might need: creating and managing servers, attaching storage, setting up firewalls, using snapshots and backups, and even enabling rescue mode.

A clean and responsive web interface was designed with Bootstrap to make the experience as intuitive as possible. Security was handled by Django toolsset, with CAPTCHA protection, email confirmation, and API token validation.

The prototype was tested by users with different levels of technical knowledge. Automated unit tests were also created to check things like login, form validation, and API access. The feedback was positive, the platform is functional, cleanly coded, and easy.

While the system is complete, there’s plenty of room for new features. Features like role-based access, two-factor authentication, and support for other cloud providers could be added later.

In the end, HetzManager shows that it's possible to take back control from closed platforms and create something simple, secure, and tailored to user needs, all without sacrificing flexibility or freedom.

References

Andreas Wittig, Michael Wittig. Amazon Web Services in Action, Third Edition. Manning Publications. May 2023. 552 pages

Antonio Melé. Django 5 By Example - Fifth Edition. Packt Publishing. April 2024. 813 pages.

Arnaud Lauret. The Design of Web APIs, Second Edition. Manning Publications. June 2025. 536 pages

Ben Smith. Beginning JSON. Apress. March 2015. 324 pages

Bootstrap. Front end toolkit. <https://getbootstrap.com/>. 2025

Cloud Services: <https://www.redhat.com/en/topics/cloud-computing/what-is-iaas> 2025.

Cooper, A., Reimann, R., Cronin, D., Noessel, C. About Face: The Essentials of Interaction Design. Wiley. 2014

DigitalOcean. <https://www.digitalocean.com/>. 2025.

Django Project. <https://www.djangoproject.com/>. 2025.

Hetzner Official Documentation. <https://docs.hetzner.cloud/reference/cloud> 2025

Hetznercloud/hcloud-python. Github Repository. <https://github.com/hetznercloud>. 2025.

Johanna Saladas. Official IBM Blog. What is cURL and how does it relate to APIs?. <https://developer.ibm.com/articles/what-is-curl-command/>. 27 February 2024.

Mark Lutz. Learning Python, 6th Edition. O'Reilly Media, Inc. February 2025. 1,172 pages.

ModulesGarden. <https://www.modulesgarden.com/products/whmcs/hetzner-cloud-servers> (2025)

Nielsen, J. Usability Engineering, Morgan Kaufmann. El Sevier. 1994.

NIST - National Institute of Standards and Technology. US Department of Commerce. <https://csrc.nist.gov/glossary/term/hypervisor>. 2025

Official Hetzner Cloud Python library. <https://github.com/hetznercloud/hcloud-python>. 2025

Open Worldwide Application Security Project. https://owasp.org/www-project-top-ten/. 2025

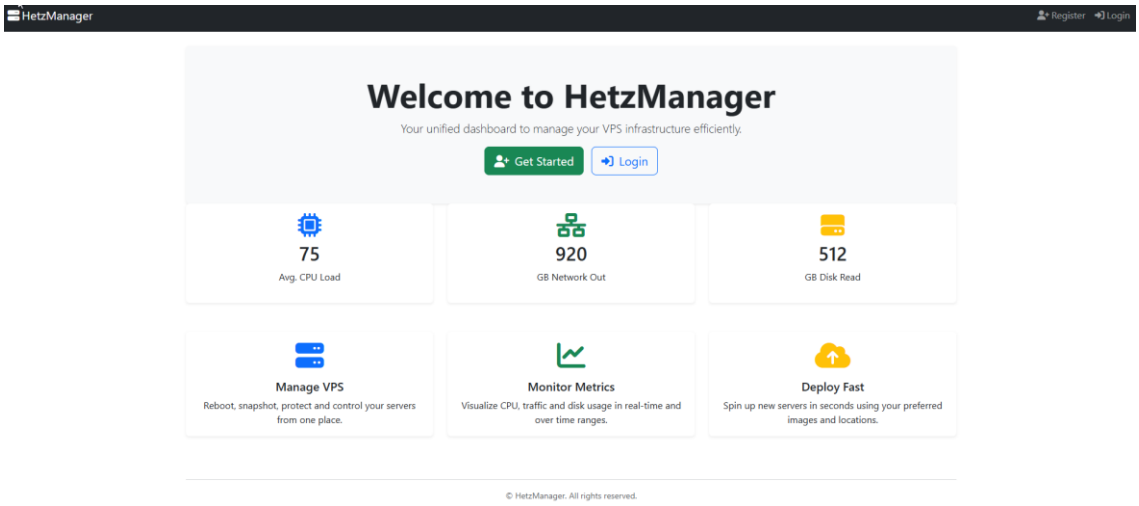
Python unittest.mock library. <https://docs.python.org/3/library/unittest.mock.html>. 2015

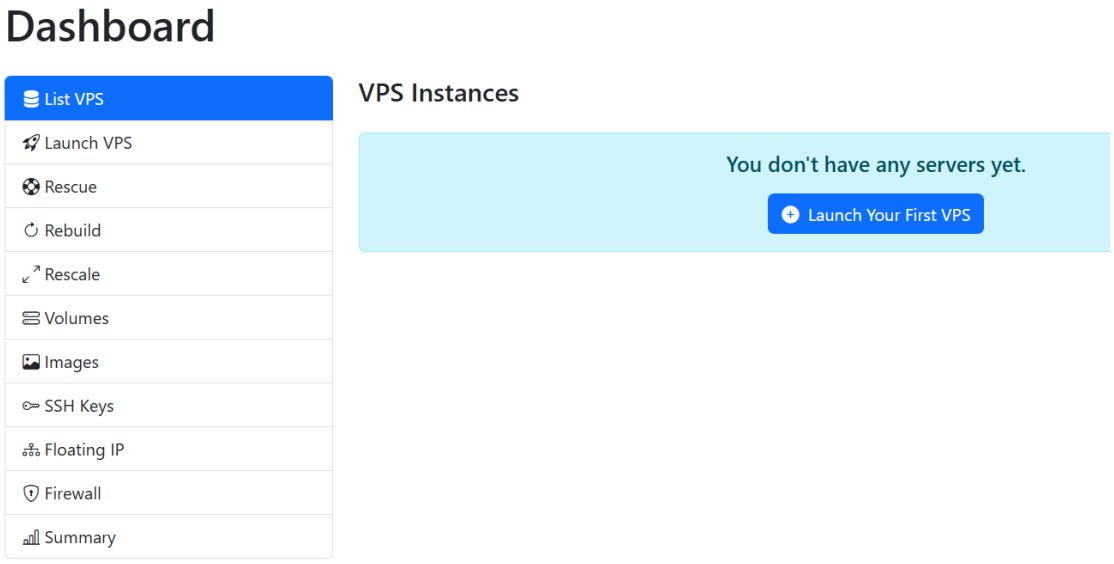
Sunilkumar Manvi, Gopal Shyam. Cloud Computing. CRC Press. March 2021 350 pages.

WHMCS. <https://www.whmcs.com/>. 2025.

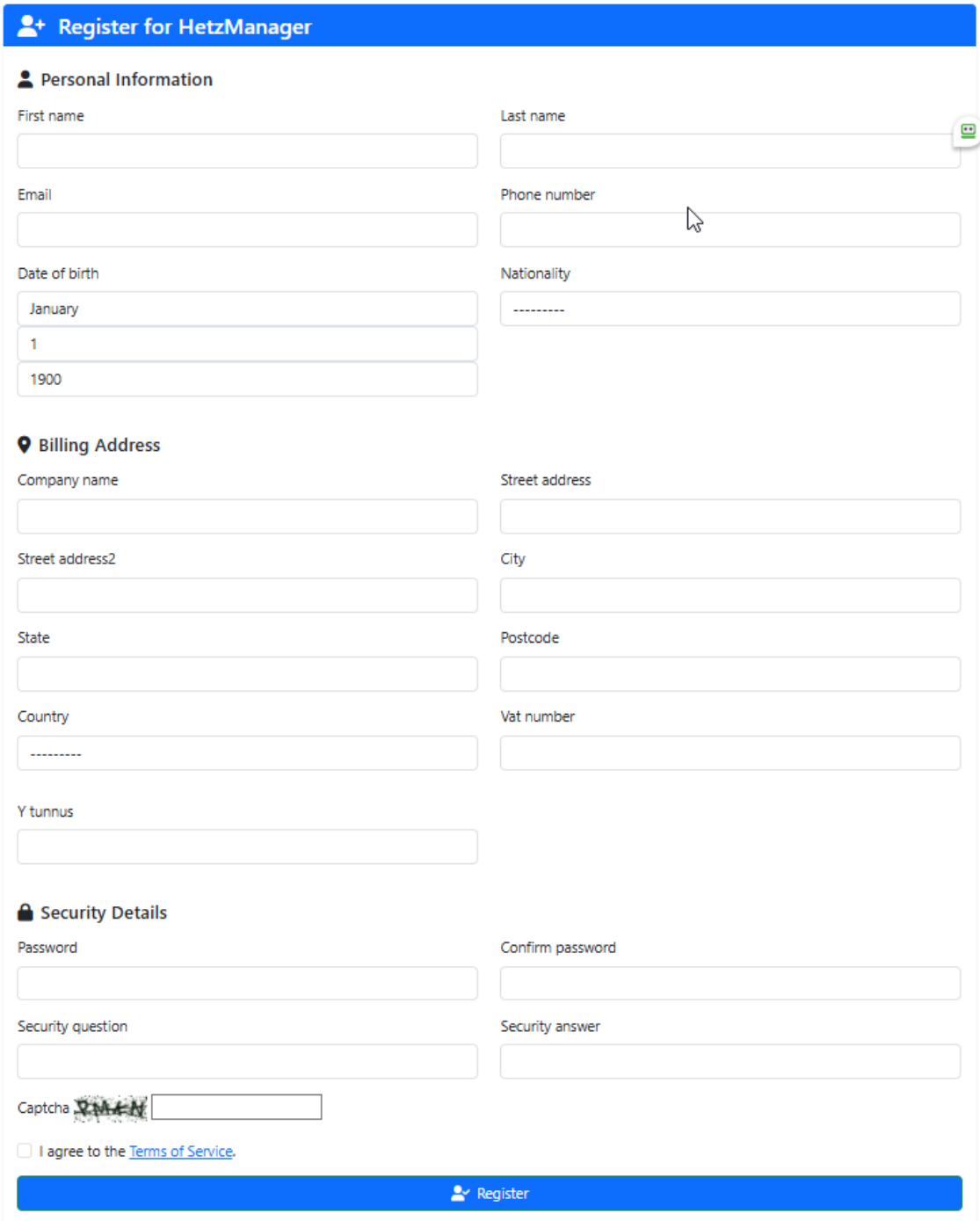
Wikipedia. Cloud Computing. <https: //en. Wikipedia .org /wiki/ Cloud_computing>. 2025.

Appendix 1: View of the Welcome Page and Dashboard

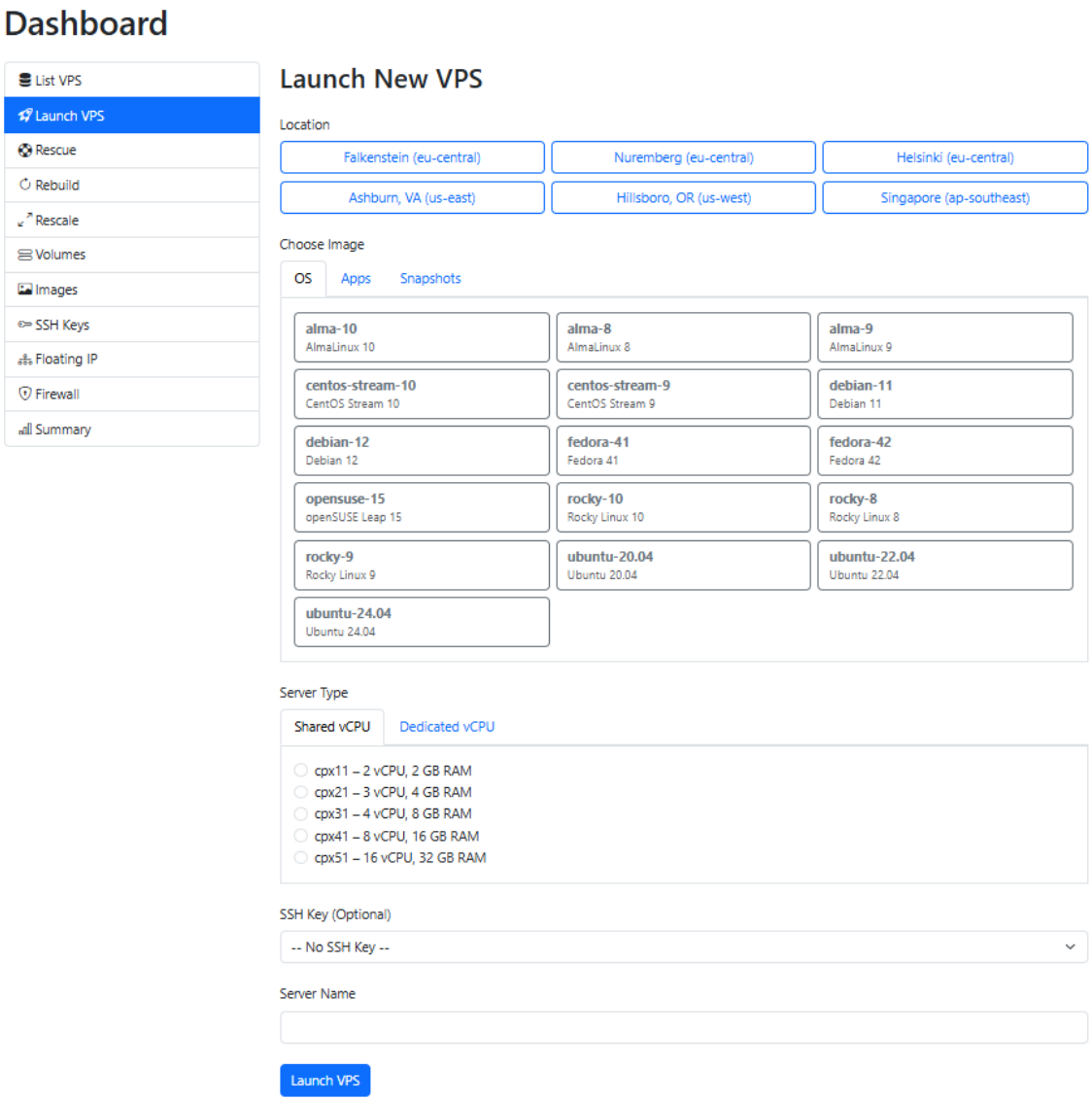




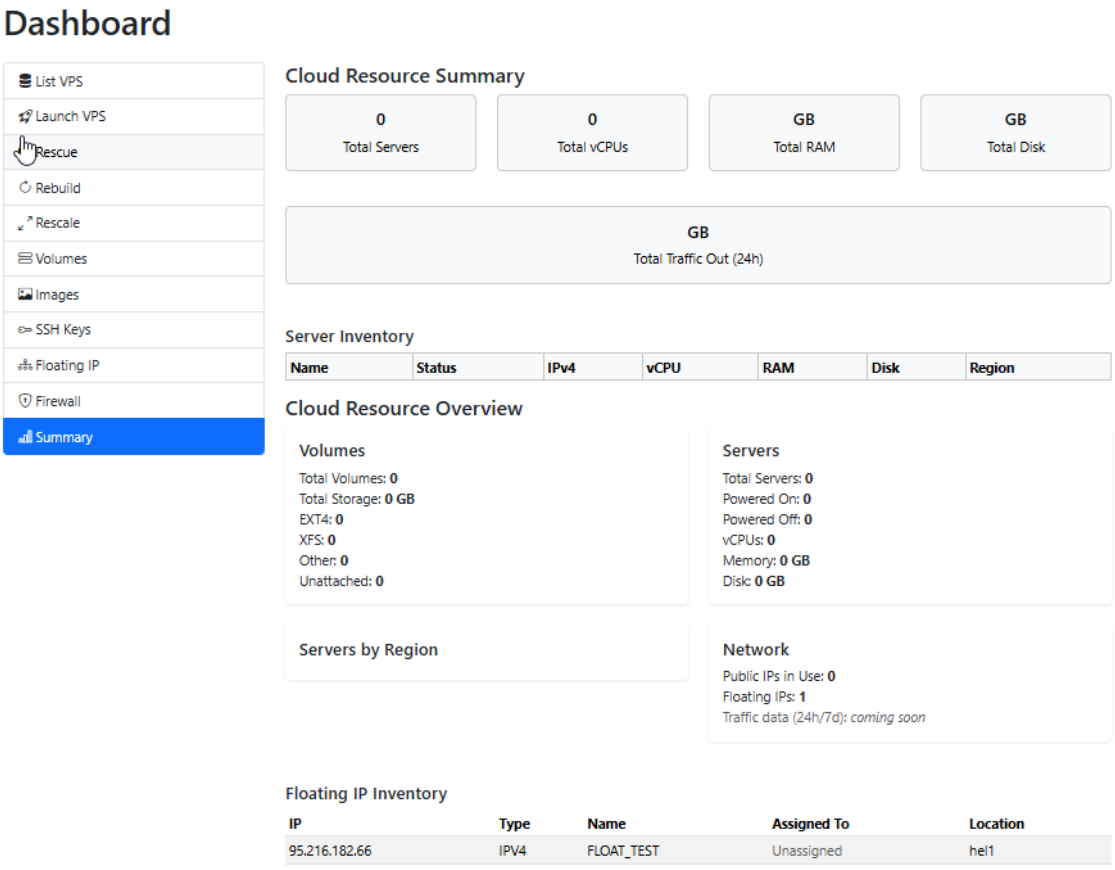
Appendix 2: View of the Register Form



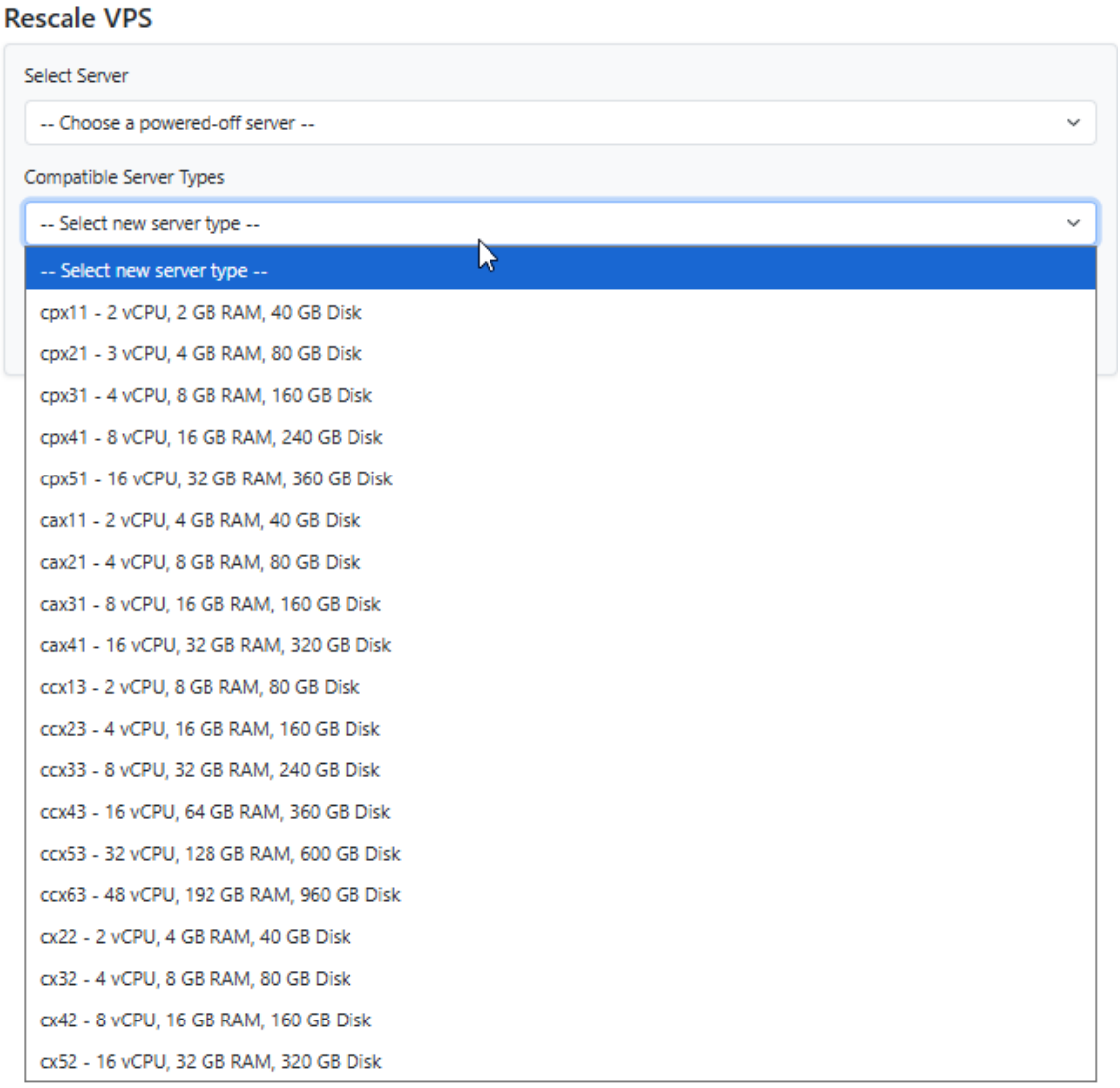
Appendix 3: View of the Launch New VPS



Appendix 4: View of the Summary



Appendix 5: View of the Rescale VPS list options



Appendix 6: View of the VPS Instance options