# Evolving from Standalone Smart Plug to AI-Powered House

## Transitioning to Open Source Intelligent Home Automation

Sven Ruediger
*AKAD University*
mail@sven-ruediger.com

***Abstract*—The modern home is increasingly becoming a hub of automated efficiency, necessitating the integration of diverse technologies. This paper introduces a FreeBSD-based [1] system that seamlessly integrates Home Assistant [2], Apache Kafka [3], and OpenSearch [4] for managing essential home functions such as lighting, shutters, and energy consumption. The system adeptly utilizes a range of data sources, including proprietary protocols like KNX and versatile ReST-APIs, to facilitate real-time analytics and event-triggered responses. This results in a self-learning home automation environment, uniquely prioritizing energy optimization while simultaneously enhancing comfort in contemporary homes.**

### Motivation

The motivation for this project has been the development of a modern smart home, anchored in open-source and on-premise solutions. The integration of photovoltaics and a state-of-the-art heat pump elevated this project from a hobby to a serious endeavor aimed at reducing both energy costs and CO2 footprint.

### The Role of FreeBSD

FreeBSD was selected primarily for its reliability, along with its low resource consumption and support for ARM architectures. These features make it ideal for a resource-efficient, portable home automation solution. Although currently running on high-power servers, all components are ARM64 compatible, enabling future migration to low-power SOCs. Additionally, FreeBSD's boot environments and ZFS snapshots provide a reliable fallback during updates, simplifying platform maintenance.

### Historical Overview and Challenges

- **Initial Attempts**: The project commenced nine years ago with the adoption of Home Assistant, selected for its Python-based architecture and support for Zigbee and Z-Wave. However, the initial reliance on Debian-based systems frequently led to administrative challenges.
- **Trials and Tribulations**: I encountered multiple issues, including failed microSD cards, OS-level breaking changes, and unresponsive Docker containers.
- **Switch to FreeBSD**: Approximately five years ago, a transition to FreeBSD was made, which addressed these challenges and provided a stable operating environment.
- **Advancements in Automation**: The project shifted from manual planning for automations to utilizing Elasticsearch/OpenSearch for intelligent data persistence along with ML tools for decision-making.
- **Incorporating Real-Time Data Processing**: The integration of Apache Kafka significantly enhanced the system's responsiveness to real-time events.

### The Hardware

The project benefited from the acquisition of an older house, which required comprehensive remodeling. This circumstance provided a unique opportunity to integrate advanced systems for sustainability and efficiency from the ground up. To achieve sustainability and efficiency, several advanced systems were integrated:

1. Photovoltaic System: The house is equipped with a 13KWp solar panel array, ensuring a good energy autonomy even in bad weather conditions. This setup significantly reduces energy costs and CO2 emissions.

2. Battery Storage: A 14KWh battery is included for storing excess solar energy. This helps in minimizing energy draw from the grid, enhancing both energy independence and efficiency along with emergency power functionality.

3. Switchable Sockets: Multiple power lines allow for flexible socket switching in each room. This feature enables remote control of non-smart devices, contributing to energy conservation.

4. Cabled Sensors and KNX System: All traditional 230V switches were replaced with KNX push buttons. Motion, window, and door sensors are connected via KNX and 0.8mm cables, amounting to over 2km of cabling. This system ensures reliability and does not require frequent battery changes. The backend technology of KNX offers room for future upgrades or replacements.

5. Heat Pump and Water Buffer: A Stiebel Eltron heat pump, connected to 1000l water buffers, efficiently converts peak solar energy into thermal energy. This setup optimizes the use of solar power for heating purposes.

6. Advanced Cabling: Each room features at least two 5xNYM cables (3 phases) and two duplex Cat7 connections. The Cat7 cabling is versatile, allowing for various data usages beyond just LAN connections.

7. Connectivity: The only wireless system is a cost-effective 433MHz weather station. The inverter connects via Ethernet, and the heat pump through a Modbus TCP gateway, ensuring robust and reliable communication.

This comprehensive hardware setup not only makes the house energy-efficient but also technologically advanced, setting a benchmark for smart home systems.

## The Platform

An PCEngines APU with OPNsense serves as the firewall, DNS, and DHCP server. The servers, running FreeBSD with ZFS, benefit from features like Boot Environments and ZFS for worry-free updates. All components are contained within jails, managed by Bastille, ensuring efficient deployment and maintenance. Home Assistant acts as the core for normalization and data visualization.

## I. Operations Perspective

In this setup, there is no preproduction system; all testing and development occur live. Jails are utilized to test new releases and seamlessly replace existing jails. The core components of the system are distributed across multiple jail hosts.

It is feasible to upgrade without downtime by employing jails with ZFS and reducing DNS TTLs. Prior to upgrading the host, the jail is sent to another host using ZFS-send. This setup uses Bastille, where jail configurations and data roots are separated, allowing IP addresses to remain fixed on the default and temporary hosts. The interim jail is then activated, and DNS entry switched to the temporary host. Following this, the original host can be upgraded and, once complete, the process is reversed to revert to the original configuration.

```
zfs snap bastille/jails/syslog/root@upg

zfs send bastille/jails/syslog/root@upg|\
ssh jailhost02\
zfs recv bastille/jails/syslog/root

ssh jailhost02 bastille start syslog

Change A record from IP
for syslog-ng@jailhost01
to IP for syslog@jailhost02

Wait for TTL expiration

jailhost01# bastille stop syslog

Perform upgrades on jailhost01

Revert the A record to point back
to syslog on jailhost01

ssh jailhost02 bastille stop syslog
```

For managing real-time events, messages, and ingest processes, this system can be integrated with Kafka, utilizing consumer groups to attach multiple processes/hosts.

Key lessons learned for enhancing uptime include:

- Employing `beadm` and `zfs snap` before performing upgrades.
- Setting resource limits for each jail.
- Creating rc scripts for commonly used scripts (Python's watchdog module is particularly useful), see Appendix 1.
- Avoiding temporary naming conventions.
- Shutting down databases before creating snapshots.
- Utilizing a whitelist of jails for auto-start (e.g., `bastille_list="homeassistant opensearch01 kafka01"`).
- Establishing secure emergency access to the system, with reverse tunnels (this will save your vacation or BSDcon).
- Investing in Out-of-Band Management (OOBM) licenses, such as iDRAC or iLO.
- Deploy a push services, messenger bot or at least email notification for alerts.

## Implementation and Energy Optimization

All components are hosted on-premises, with external APIs only for solar and weather forecasting. Apache Kafka manages real-time data processing, while OpenSearch handles long-term data storage and anomaly detection. Energy optimization is achieved through Home Assistant, which manages power from solar panels and adjusts the heat pump's power modes. Analyzing low-temperature floor heating curves with multiple external factors is complex and requires extensive trials. Another step in automating lights and power consumption of electronic devices involves several applications; for instance, during summer, the socket for the mobile AC is programmed to turn on as soon as the overproduction of electric energy reaches a predetermined level. The house has multiple Android tablets to see current power usage and control major switches in the house in a single dashboard. The tablets show simple state of energy like battery level and solar power, so high-energy (non smart) home appliances like dryers can be switched on manually to reduce extra electricity from the grid.

## Towards AI

A key to creating successful ML models is the quality of the data. One of the most important requirements is to have continous and consistant data sources. Therefore it is mandatory to get your tool close to high availability, rc scripts and watchdogs as described before, are absolutely neccesary for realiable data sources and storages. In addition the scripts themself need at least a basic error handling and threading to prevent data loss. Example code can be found in Appendix 2

As of now all stored data is manually loaded via a Jupyter notebook to (re)evaluate machine learning (ML) models. The K-Nearest Neighbors (KNN) algorithm proved effective in identifying relationships between various sensors (e.g. motion and door sensors) and lighting systems, yet this was just the beginning. The initial focus on simple anomaly detection was less effective due to numerous influencing factors. Integrating the heat pump revealed that creating a fully automated home requires a more in-depth investigation into ML models,

particularly in deep learning, with considerations for different seasons. OpenSearch serves as an ideal data store for ML, offering capabilities for real-time analytics. Additionally, Kafka facilitates the triggering of real-time events, such as motion detection, which do not require seasonal adjustments.

### Deep Learning: Utilizing RNN/LSTM

Recurrent Neural Networks (RNN) and Long Short-Term Memory (LSTM) networks, both suitable for time-series data, are being leveraged for predictive analytics in this project. These networks excel in processing sequential data, making them ideal for predicting future states based on sensor data. This capability is crucial for energy optimization and automation decisions in smart home environments. Currently, TensorFlow/Keras with a Tesla GPU is used on a separate host, with plans to deploy on more efficient hardware like the Google Coral TPU. Starting with Opensearch 2.6 an option to import machine learning models was introduced and opened up new chances of real time decision makeing and automated optimization of energy consumption.

### Conclusion

This project aims to show the effective integration of FreeBSD-based systems with advanced ML technologies. It underscores the importance of flexible and durable soft- and hardware, and advanced data processing techniques in realizing enhanced energy efficiency and automation in contemporary homes. The central role of FreeBSD in this endeavor showcases the capabilities and potential of open-source solutions in smart home automation.

## II. Future Work

Looking ahead, there are plans to deepen its integration with ML to further enhance the automation and efficiency of smart home systems. Key areas of focus include:

1) **Advanced Predictive Analytics**: Developing more sophisticated ML models to predict energy consumption patterns and optimize resource allocation. This could involve using deep learning algorithms to better understand daily usage patterns and automatically adjust home systems for maximum efficiency.
2) **Automated Machine Learning (AutoML)**: Implementing AutoML solutions to streamline the process of model selection, training, and optimization. This approach will not only save time but also enhance the accuracy of predictions by selecting the most suitable algorithms for this unique data sets.
3) **Real-Time Decision Making**: Integrating real-time data streams with ML models to make instantaneous decisions. For example, using sensor data to predict and respond to changes in weather conditions, thereby adjusting energy usage on the fly.
4) **User Behavior Analysis**: Utilizing ML to analyze user behavior and preferences to create a more personalized home automation experience. This could involve learning from user interactions with various home devices to automatically adjust settings according to individual preferences.
5) **Scalability and Portability**: Enhancing the scalability of the system to support a wider range of devices and architectures. This includes refining ML models to work efficiently on different platforms, ensuring that the benefits of smart home automation can be experienced on a broader scale.
6) **Enhancing Security with ML**: Exploring the use of ML algorithms to improve the security of the smart home network. This would involve the development of systems capable of detecting and responding to unusual network activity, thereby safeguarding against potential security breaches.

]

## References


[1] FreeBSD, https://www.freebsd.org.
[2] Home Assistant, https://github.com/home-assistant.
[3] Apache Kafka, https://github.com/apache/kafka.
[4] OpenSearch Project, https://github.com/opensearch-project/OpenSearch.
[5] KNX Association, https://www.knx.org/.
[6] Stiebel Eltron ISG, https://www.stiebel-eltron.de/isg.
[7] BastilleBSD, https://github.com/BastilleBSD/bastille.
[8] Fronius Solar API, https://www.fronius.com/.
[9] Homelab Panel, https://papers.freebsd.org/2020/bsdcan/.


## Appendix

### A. Scripts

*1) Fronius Service Script:*

*2) Fronius API:*

```
#!/bin/sh

# PROVIDE: fronius
# REQUIRE: DAEMON NETWORKING
# BEFORE: LOGIN
# KEYWORD: shutdown

# Run fronius_enable=YES
# to enable fronius service.


. /etc/rc.subr

name="fronius"
rcvar=fronius_enable

# daemon
pidfile="/var/run/${name}.pid"
python="/usr/local/bin/python3"
watchdog_script="/usr/local/bin/watchdog.py"
script_py="/usr/local/share/fronius2opensearch/main.py"
command=/usr/sbin/daemon
procname="daemon"
command_args=" -r -c -f -P ${pidfile} ${python} \
${script_py}"

load_rc_config $name
run_rc_command "$1"
```

Fig. 1. Sample RC Script

```
try:
        response = client.indices.create(index_name, body=index_body)
except:
        print("Error creating index. Already exists? Skipping...")
        pass

async def loop_indefinitely():
    while True:
        async with aiohttp.ClientSession() as session:
            try:
                response = await session.get(f"http://inverter01/solar_api/
                v1/GetPowerFlowRealtimeData.fcgi")
                data = await response.json()
                if type(data) != list:
                    data = [data]
            except:
                pass
            if data:
              for item in data:
                item["@timestamp"] = datetime.utcnow().strftime('%Y-%m-%dT%H:%M:%S')
                response = client.index(
                index = index_name,
                body = item)
                print(datetime.utcnow().strftime('%Y-%m-%dT%H:%M:%SZ'))
            print(data)
        await asyncio.sleep(5)

asyncio.set_event_loop(asyncio.new_event_loop())
loop = asyncio.get_event_loop()
loop.run_until_complete(loop_indefinitely())
```

Fig. 2. Sample API fetch