



ARTICLE

Development of a Connected Smart Home Based on Containers

Ibrahim ALOUI¹ and Mounira TARHOUNI^{*2}

¹Department of Industrial computing, Higher Institute of Computer Science and Multimedia Gabes (ISIMG), Gabes University, Gabes, Tunisia, PROFESSIONAL MASTER IN EMBEDDED SYSTEMS & IOT

²Department of Multimedia and Web, Higher Institute of Computer Science and Multimedia Gabes (ISIMG), Gabes University, Gabes, Tunisia, PROFESSIONAL MASTER IN EMBEDDED SYSTEMS & IOT

*Corresponding author. Email: ibrahim.aloui@isimg.tn

Abstract

This project aims to develop a sophisticated smart home system using containerization technologies, notably Docker and Kubernetes, to streamline deployment, enhance scalability, and ensure seamless integration of IoT devices and applications. By encapsulating various components within lightweight, portable containers and utilizing Kubernetes orchestration, we aim to create a flexible and extensible platform for home automation.

Keywords: Smart Home, Containerization, Docker, Kubernetes, IoT, Flutter

Abbreviations: ISIMG: Institut Supérieur d'Informatique et de Multimédia de Gabès, MPSSI: Mastère Professionnel : Sécurité des Systèmes Informatiques, VAD: vitamin A deficiency

I Introduction

The development of smart home technology has significantly changed modern life by improving comfort, safety, and energy efficiency. Smart homes integrate various Internet of Things (IoT) devices such as smart thermostats, security cameras, and connected appliances to provide homeowners with remote control and automation capabilities. However, managing these devices presents deployment, scalability, and performance challenges.

Containerization provides a powerful solution to these challenges. Technologies such as Docker package applications and their dependencies into isolated, lightweight containers, ensuring consistent

performance across environments. Kubernetes is an orchestration tool that automates the deployment, scaling, and operation of these containers, thereby improving the reliability and scalability of the system.

By leveraging containerization and orchestration, smart home systems can achieve greater scalability, reliability, and ease of integration. This approach not only addresses the complexity of managing IoT devices, but also paves the way for a more efficient and cohesive smart home ecosystem. AI technologies such as integrated regression models can further improve functionality by providing predictive insights and optimizing operations.

II Related Work

In this section, we review existing research and projects related to smart home systems and containerization. These studies provide valuable insights into the application of containerization technology in the context of smart homes and offer various methods and strategies to improve the deployment, management, and scalability of IoT devices and applications.

- **SmartHome System Based on Containerization Technology** Zhong et al. 2017
This paper presents a SmartHome system based on containerization technology. It discusses the utilization of containers for managing smart home applications and devices, highlighting the benefits of this approach in terms of deployment and scalability.
- **Scalable Smart Home Infrastructure Using Container Orchestration**
This article explores the use of container orchestration platforms like Kubernetes to build scalable smart home infrastructures. It discusses the benefits of containerization in terms of resource optimization and seamless deployment of IoT applications.
- **Integration of Docker Containers in Smart Home Environments**
Investigates the integration of Docker containers in smart home environments to streamline application deployment and management. Discusses the advantages of containerization in improving system agility and reducing resource overhead.
- **Containerization of Home Automation Services for Improved Flexibility**
Proposes a containerization approach for home automation services to enhance flexibility and modularity. Examines how Docker containers can simplify the deployment and scaling of smart home applications.
- **Enhancing Security in Smart Homes Using Containerized Applications**
Explores the use of containerization technologies to enhance security in smart home environments. Discusses strategies for isolating and securing IoT applications using Docker containers.
- **Efficient Deployment of IoT Devices Using Kubernetes Orchestration**
Discusses the benefits of Kubernetes orchestration for deploying and managing IoT devices in smart homes. Examines how Kubernetes simplifies the deployment process and improves system reliability.
- **Containerization for Edge Computing in Smart Home Systems**
Investigates the use of containerization for edge computing in smart home systems. Discusses how Docker containers can be deployed at the edge to improve latency and efficiency in processing IoT data.
- **Managing Home Automation Workflows with Containerized Microservices**
Explores the use of containerized microservices to manage home automation workflows effectively. Discusses how Docker containers can encapsulate individual services, enabling easy scaling and maintenance.
- **Containerization for IoT Device Management in Smart Homes**
Presents a containerization approach for managing IoT devices in smart homes. Examines how Docker containers can simplify device management tasks and improve system reliability.

• Orchestrating Smart Home Applications with Kubernetes

Discusses the benefits of using Kubernetes for orchestrating smart home applications. Explores how Kubernetes automates deployment, scaling, and management tasks, leading to improved system performance and reliability.

III Proposed Approach

To overcome these challenges, this project proposes the development of a connected smart home system based on containerization technologies. By adopting Docker and Kubernetes, we aim to address the aforementioned issues by:

- Simplifying deployment and management through containerization, allowing for seamless integration of IoT devices and applications.
- Enhancing scalability and reliability through Kubernetes orchestration, enabling dynamic scaling and fault tolerance.
- Integrating AI capabilities into the smart home ecosystem using containerized AI models, thereby enhancing automation and intelligence.
- Providing a unified platform with a Django backend and Flutter mobile application for centralized control and monitoring of the smart home environment.

We will now present an overview of the entire system using a visual diagram.

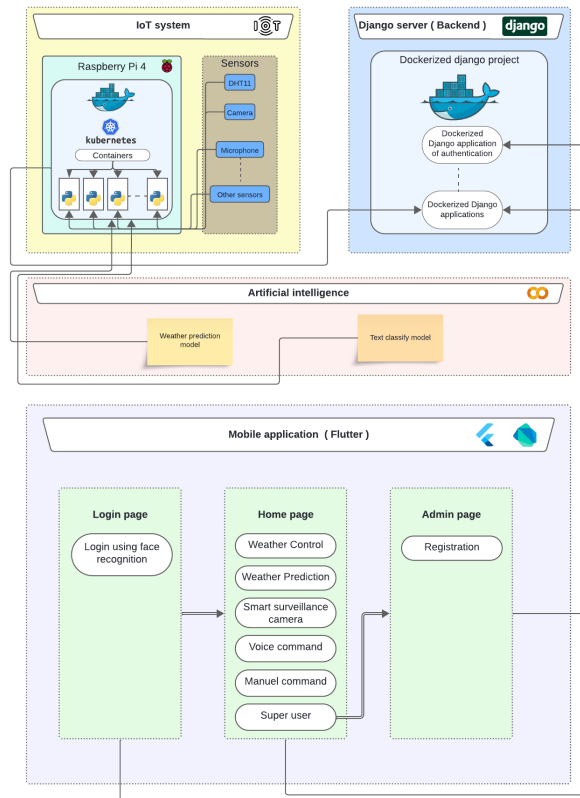


Figure 1. Entire system diagram

IV IoT System based on container

1 Architecture

2 containerization

V Ai regression based on container

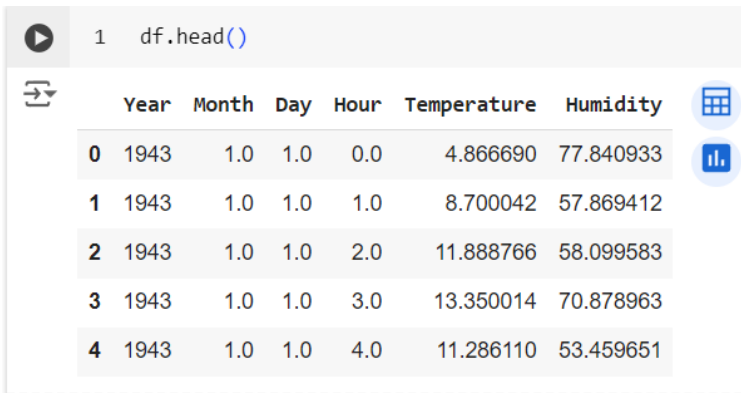
1 Ai regression

In this section, our objective is to develop an AI regression model capable of predicting temperature and humidity based on temporal data. This model will serve as a critical component of our smart home system, enabling real-time weather forecasting for enhanced user experience and environmental control.

1.1 Model Creation

To initiate the model creation process, we upload the dataset.csv to Google Colab, leveraging its powerful computing resources and collaborative features. Within the Colab environment, we employ three distinct regression algorithms: Gradient Boosting, Random Forest, and Decision Tree. These models are trained using the historical temperature and humidity data collected from IoT sensors within the smart home network.

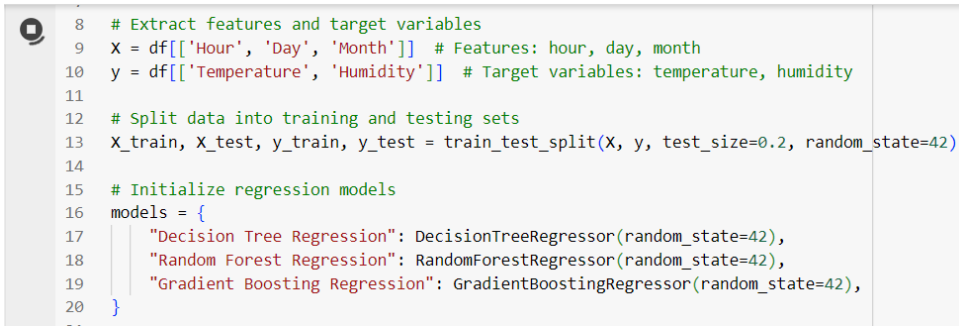
the following figure presents the structure of our dataset



	Year	Month	Day	Hour	Temperature	Humidity
0	1943	1.0	1.0	0.0	4.866690	77.840933
1	1943	1.0	1.0	1.0	8.700042	57.869412
2	1943	1.0	1.0	2.0	11.888766	58.099583
3	1943	1.0	1.0	3.0	13.350014	70.878963
4	1943	1.0	1.0	4.0	11.286110	53.459651

Figure 2. Dataset structure

after uploading our dataset , we are now ready for beginning the creation of our model : as we are mention we will use the the temporal data to predict weather data so we will put hour , Day and Month as features while Temperature and Humidity and then put the three models and just launch the train . The following figure represents the phase of train of our model.



```

8 # Extract features and target variables
9 x = df[['Hour', 'Day', 'Month']] # Features: hour, day, month
10 y = df[['Temperature', 'Humidity']] # Target variables: temperature, humidity
11
12 # Split data into training and testing sets
13 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
14
15 # Initialize regression models
16 models = {
17     "Decision Tree Regression": DecisionTreeRegressor(random_state=42),
18     "Random Forest Regression": RandomForestRegressor(random_state=42),
19     "Gradient Boosting Regression": GradientBoostingRegressor(random_state=42),
20 }

```

Figure 3. Train of model

2 containerization

3 Flutter Mobile Application Development

4 users case diagram

5 implementation

6 interfaces

VI orchestration

1 concept

2 implementation

3 Results

In this section, we will look at developing our mobile application using Flutter and Dart. These two powerful technologies will allow us to create powerful and feature-rich experiences for users.

To get started, it's important to set up our development environment and understand the structure of a Flutter project. This will lay the foundation for building and running our application smoothly.

3.1 Flutter project

First, let's take a look at the process of creating our Flutter project.

□ Installation and Setup:

At the beginning, we should download and install the Flutter SDK on our PC and add it to the path in the system as shown in the figure below :

□ Create a new Flutter project:

then we will use the command line `flutter create sh` to create a new Flutter project.

□ Develop the UI:

After that, we should use Flutter's declarative UI structure to create the UI for each page. Here is an example of the UI of the Login form code.

□ Implement the functionality:

At this level, we will implement the functionality for our application, such as the login process, home page features, and admin page. the following figure shows a snippet code of the functionality of the login button.

□ Running on a real device:

And the last step is to use the `flutter run` command to launch our app on a real device and test its performance.

Download then install Flutter

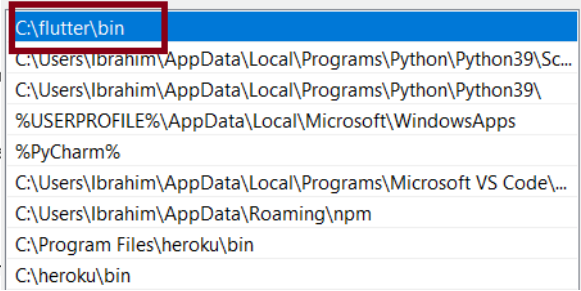
To install Flutter, download the Flutter SDK bundle from its archive on the Flutter website.

1. Download the following installation bundle to get the latest version of the SDK.



For other release channels, and older builds, check out the Flutter SDK archive.

(a) Downloading Flutter SDK



(b) Adding Flutter SDK to path in the system

Figure 4. Installation and Setup of Flutter

3.2 Flutter project

In this section, we present the design of a Flutter mobile application, focusing on three primary pages: the Login page, Home page, and Admin page. Each page serves a distinct purpose and is equipped with specific components to enhance user experience and functionality as shown in the following figure.

(a) Login page: The Login page is the initial interface where users authenticate their identity. The components on this page include:

- ① User Temporary Picture: Allows login using face recognition with a temporary picture.
- ② Username and Password Fields: Input fields where users can enter their username and password.
- ③ Login Button: A button for submitting the login credentials.
- ④ Verifying Indicator: A status indicator that appears when the login process is in progress.

(b) Home page: Once authenticated, users are directed to the Home page, which serves as the main dashboard. The components of this page include

- ⑤ User Greeting and Profile: Welcomes users with their name and profile picture, fostering a personalized environment.
- ⑥ Outside Weather Values: Offers real-time weather conditions outside, keeping users informed.
- ⑦ Inside Weather Values: Displays current indoor weather conditions, providing immediate environmental insights.
- ⑧ Weather Prediction Graph: Illustrates weather forecasts graphically, aiding users in planning ahead.
- ⑨ Smart Surveillance Camera: Enhances security monitoring by showcasing the status or live feed from a smart surveillance camera.
- ⑩ Manual Smart Home Controls: Allows users to manually control smart home devices, such as lights, thermostats, and locks.
- ⑪ Voice Command: Enables users to control smart home devices using vocal commands, providing a hands-free experience.
- ⑫ Superuser Button: Provides access for superusers to administrative functions, linking to the Admin Page.
- ⑬ Logout Button: Allows users to securely log out of their account.


```
Future<void> _login() async {
  if (_image == null ||
      _usernameController.text.isEmpty ||
      _passwordController.text.isEmpty) {
    _showErrorDialog('All fields are required');
    return;
  }
  setState(() {
    _isVerifying = true;
  });
  try {
    final imageBytes = await _image!.readAsBytes();
    final imageBase64 = base64Encode(imageBytes);
    final response = await http.post(
      Uri.parse('http://192.168.1.100:3000/api/login'),
      headers: {
        'Content-Type': 'application/json',
      },
      body: jsonEncode({
        'username': _usernameController.text,
        'password': _passwordController.text,
        'image': imageBase64,
      }));
  } catch (e) {
    _showErrorDialog(e.toString());
  }
}
```

Figure 6. Login form code snippet



Figure 7. Flutter mobile application