WISE-WARE System Architecture Document

Contents

Introduction

This System Architecture document is a definitive technical document describing the architecture of the WISE-WARE system. It targets software architects, system designers, and developers seeking an understanding of the system.

This content serves as a cornerstone for effective interaction with WISE-WARE, assisting decision-making throughout the system's lifecycle, from initial deployment to ongoing maintenance and potential future enhancements.

The subsequent sections are designed to progressively explore:

1. Important Concepts: This section establishes a firm foundation by outlining the core principles that underpin WISE-WARE’s function.
2. Hardware and Software: This section describes the hardware and software components on which WISE-WARE runs.
3. Deliverable Breakdown: The core deliverable's composition is provided, offering a detailed understanding of its contents upon download.
4. Container Inventory: This section discusses the various containerized sub-components that comprise WISE-WARE in-situ.

Important concepts

As a very high-level overview:

“WISE-WARE is a supervised system of containerised services”

If you are unfamiliar with the underlined concepts, brief explanations and links for further reading have been provided below; otherwise, you can skip through to the “Hardware and software” section.

Containerised services

To understand what “containerised services” means, you must first understand what “Docker” is, the principal tool used for making and distributing them.

Docker is a tool that helps developers package their applications into self-contained units called containers. These containers include everything the application needs to run, including its own operating system, its code, and settings. This means that a Docker container can run on any computer with Docker installed, the “host” machine.

Once a container has been designed, developed and tested, an “image” of that container is made. This image is what actually gets distributed, and once someone else has downloaded it they can create as many containers as they need, all identical to the original - to start with.

While the container is in use, it can perform tasks and it’s contents may be modified in the process. If something goes wrong and the container needs to shut down, it can easily be replaced with a new one using the original image.

In the ways described, Docker makes it much easier for developers to develop, ship, and utilise applications. For a more developer-oriented introduction to the technical concepts used, consider: https://docs.docker.com/get-started/overview/.

Supervised systems

Building modern applications often involves juggling multiple containerised services. While individual containers are useful for their specific tasks, managing them all manually can quickly become cumbersome.

Ensuring all your containers communicate smoothly and start in the right order can be time-consuming and hard to reproduce consistently. What if a container crashes, causing a domino effect on other containers dependent on it?

“Container supervision” is used to solve these problems. WISE-WARE uses a part of Docker called “Docker Compose”, which is used to define a collection of containers, their relationships and connections. It can be used to automate their startup procedure, restart them if they fail, and gracefully bring the system to a stop if needed.

Like how Docker allows single applications to be made reproducible, Docker Compose allows systems comprising many applications to be made reproducible. For a more developer-oriented introduction to using Docker Compose, consider: https://docs.docker.com/compose/. To learn more about the broader context of container orchestration, of which supervision is a part, consider: https://cloud.google.com/discover/what-is-container-orchestration.

Hardware and Software

This description focuses on the on-site solution using Raspberry Pi and Docker containers. WISE-WARE may be used as part of a separate distributed business intelligence solution with a different architecture. See “Business Intelligence with WISE-WARE” for information and examples for how this might be implemented.

Hardware

Host Machine

The primary hardware platform for WISE-WARE is the Raspberry Pi 4B (or later versions like the 5). This credit-card sized computer is a popular choice for its affordability and efficiency.

Host Machine Peripherals

Externally-installed peripherals for the host machine generally fall into 2 categories:

- Memory expansion: increases to the storage capacity of the host machine - such as an additional NVMe drive - to allow for more data to be collected without needing to be cleared, or for more services (whether in continuous or sporadic use) to be installed at a time
- Transceivers: devices that allow the host machine to communicate over networking protocols that the host machine does not have a module for - such as Zigbee or ZWave protocols - to allow for interoperability with a larger set of assistive technologies and control devices

Assistive Technologies

Assistive technologies are diverse in their form and function, but typically communicate with the host machine through wireless networking protocols like WiFi, BLE, ZWave or Thread. The introduction of a particular technology into a WISE-WARE installation may also necessitate the inclusion of a peripheral device, such as a transceiver (as described above).

Software

Operating System

WISE-WARE runs on a lightweight version of Linux called Raspberry Pi OS Lite (currently based on Debian 12). This minimal OS ensures smooth operation without unnecessary features.

Distribution

WISE-WARE is distributed in three parts:

1. Docker Compose File: This file defines and automates the deployment of all the software components needed for WISE-WARE to run.
2. Regenerator: This tool helps automate certain tasks to assist with (re)deployment.
3. Template files: These files contain placeholder keys that are populated by the Regenerator. They are not suitable for deployment themselves because of this, but contain the main settings that make WISE-WARE function once they have been populated.
4. Documentation: These guides provide instructions and information for using WISE-WARE.

Execution

On-site, WISE-WARE utilises Docker containers. Docker Compose manages the creation and execution of these containers based on the defined configurations.

Deliverable Breakdown

The core deliverable, as downloaded from https://github.com/CIREIEmergence/WISE-WARE-Release, has the following contents:

Item	Description
docker-install-debian.sh	A shell script which instals Docker according to the official instructions; is included as a convenience script, but there is no other special logic. Only needs to be run once, the first time a machine running a Debian-based distribution is used to host WISE-WARE.
wiseware-regenerate.sh	A shell script which clones and populates template files, as well as creating and assigning user and group ids to parts of the /containervolumes/ file structure. This exists to automate part of the deployment process, and should be re-run whenever the WISE-WARE host device is restarted, or otherwise may have changed IP address (context dependent). Can be easily extended to automate more parts of the (re)deployment or installation process, depending on how WISE-WARE has been otherwise extended or applied.
docker-compose.yml.template	Defines the set of containers which comprise stock WISE-WARE, their connections, and data volumes (see the Containers Inventory section). Not suitable for use itself as it contains placeholder keys, to be populated by wiseware-regenerate.sh, which are not valid syntax.
configuration.yaml.template	Mostly identical to the stock Home Assistant configuration file for the purposes of being a proof-of-concept. It’s additions are the inclusion of the Kafka integration (and the configuration of the IP address to which events are produced) and adding Node-RED to the Home Assistant sidebar for easier access. Not suitable for use itself as it contains placeholder keys, to be populated by wiseware-regenerate.sh, which are not valid syntax.
/containervolumes/	Contains a series of sub-folders, divided per-service; persistent data from each service is stored in it’s container volume, although some have multiple sub-folders within their volume. This allows WISE-WARE to be run, stopped and restarted without losing important data, such as automations, user settings, or historical event data. Are not in the downloaded files because they are created and configured as necessary, either by wiseware-regenerate.sh or during normal operation.

Container Inventory

The file paths given are either those on the host machine (beginning with ‘/’) or those in the container volumes directory (beginning with ‘containervolumes/’). Data volumes on the host machine are used to grant access to system resources required by the service, while those in the container volumes directory contain persistent data written to and read from during normal function.

For brevity, the host machine is referred to as “the PI”, although extensions of WISE-WARE could eventually use other host machines.

Some networking ports are exposed, so-called “External”, so that service is accessed through the PI’s IP address rather than the container’s. Some ports are not - being “Internal” - so are only accessible by other containers, through the container’s IP address. Ports will be distinguished by which type they are, in addition to elaboration on how that port allows them to function if it is important.

Service	Description	Networking	Data volumes
Portainer	Allows the user to log into a web-based container management GUI that is much more user friendly than Docker’s command line interface	Port 9000 - External (used to access the web-GUI easily)	/var/run/docker.sock - UNIX socket listened to by the Docker daemon, allows Portainer to execute docker commands
			containervolumes/portainer - session/ admin account data
Apache Zookeeper	Coordination service for distributed systems; is used in WISE-WARE as a prerequisite of Apache Kafka	Port 2181 - Internal (exclusively used by Kafka in stock WISE-WARE)	containervolumes/zookeeper/data - configuration data
		Port 22181 - External (mapped onto 2181 internally. Not used in stock WISE-WARE but can be used to facilitate horizontal scaling by integrating multiple WISE-WAREs)	containervolumes/zookeeper/log - log data
Apache Kafka	Distributed event streaming platform, which allows data to be stored durably. A single broker is used in stock WISE-WARE, but it is possible to integrate many brokers between multiple WISE-WAREs, or introduce a distributed business intelligence solution that aggregates many brokers	Port 29092 - Internal (not used in stock WISE-WARE)	containervolumes/kafka/broker/data - event data; stock WISE-WARE event data is state changes in Home Assistant
		Port 9092 - External (exclusively used by Home Assistant in stock WISE-WARE. Allows extensions to WISE-WARE to read and write event data)	containervolumes/kafka/secrets - used for security features, storing sensitive data in encrypted formats

Home Assistant	Smart home and IoT coordination, automation and management software, handles the communication with and integration of many smart devices from many brands. Also provides a highly customisable, user-friendly interface	Port 8123 - Host network mode (this container uses the network stack of the PI, rather than being a part of the isolated Docker network; this allows device discovery* and does not extend the attack surface more than exposing the port as normal. It also means that if this service needs to communicate with the other containers, it needs to use their external ports and the PI's IP address)	containervolumes/homeassistant/config - contains configuration data for Home Assistant, including the main config file, custom components, etc.
			/etc/localtime - allows synchronisation with the PI's local time; readonly
			/run/dbus - desktop bus, required for Bluetooth interaction; readonly
Node-RED	Visual programming tool to allow nontechnical users control over their smart home's automations and logic, without needing to know classic, textual programming languages. Integrated into the interface of Home Assistant	Port 1880 - External (used for integration with Home Assistant, which needs to network as an external service. Can be used to access the GUI in a web browser, but this is not needed due to the aforementioned integration)	containervolumes/nodered/data - configuration and flow (automations created in Node-RED) data

* Device discovery is a feature that allows Home Assistant to automatically detect nearby devices, download their integration code (if it is not already installed) and prompt the user to configure the device, without any need to manually search for IP addresses or similar