

WISE-WARE configuration guides

Contents

| | |
|---|-------------------|
| Introduction..... | 3 |
| Regenerator script..... | 4 |
| Docker compose..... | 5 |
| Components..... | 5 |
| Configuration options..... | 5 |
| Adding a new service..... | 6 |
| Home Assistant..... | 7 |
| Accessing configuration files..... | 7 |
| Integrating a new service into Home Assistant's UI..... | 8 |
| Kafka..... | 8 |
| Listeners and advertised listeners..... | 8 |

Introduction

Regenerator script

The `wiseware-regenerate.sh` script is an optional but useful part of the WISE-WARE deployment process, as it automates several key tasks to streamline the setup. This script plays a vital role in ensuring a smooth and consistent deployment, and can easily be extended (either using existing functionality or adding new ones) as a given setup needs.

1. **Persistent Data Volume Handling:** A set of directories are created and assigned particular user and group ids. Because most directories are created as they are needed during normal operation, in stock WISE-WARE this is only used in cases where particular user and group ids are needed by a service container before it is created.
2. **Template Resolution:** A set of template files, containing placeholder keys, are populated with real values determined at runtime, such as by execution of shell commands to find and insert the current host IP address. This ensures that the generated files are properly configured for the current deployment, provided the preparatory steps are deterministic.
3. **Reporting and Tutorials:** A report message is printed that summarises the key steps a user should take to complete the deployment, such as how to start WISE-WARE.

Extensions to this script may include:

- New templates to that need regenerating, such as configuration files for new services introduced to the system
- Additional placeholder keys to populate, and the logic to determine their values at runtime
- Downloading files from remote repositories, such as authentication certificates
- Producing log files recording changes made during the script's execution, or helping to audit errors that occur

By understanding and potentially extending the functionality of the `wiseware-regenerate.sh` script, developers can tailor the WISE-WARE deployment process to their specific needs, making the platform more adaptable and easier to maintain in various environments.

To run the script, navigate to the directory containing the ``wiseware-regenerate.sh`` file and execute the following command:

```
sudo ./wiseware-regenerate.sh
```

Docker compose

WISE-WARE uses Docker Compose to automate deployment and supervision of its containerised services. “docker-compose.yml” has inline comments explaining itself, but this section will outline the important parts.

Components

The services used in stock WISE-WARE are:

- Portainer: A web-based user interface for managing Docker containers. Is used for easy management and monitoring of a WISE-WARE deployment through a GUI rather than a CLI.
- Zookeeper: A distributed coordination service and metadata broker. In WISE-WARE, is used to support Apache Kafka.
- Apache Kafka: A high-throughput messaging system for streaming data between services. In WISE-WARE, durably stores event data that is aggregated by Home Assistant, for later distribution or analysis (subject to user agreement, and use context).
- Home Assistant: An open-source home automation platform with an extensive library of community-sourced integrations, tutorials and support. Is used in WISE-WARE to integrate devices from many brands into a single data framework, and allows for automation and coordination without relying on cloud services (itself, some services like Amazon Alexa still require internet access for their own functionality).
- Node-RED: A visual programming tool often used to integrate IoT and web services. Is used in WISE-WARE to provide a low-code environment to lower the difficulty of use, for users and their delegates to automate parts of the smart home.

Configuration options

“docker-compose.yml” defines each service along with its configuration details. The following are the properties used in stock WISE-WARE, although there are others. Not all services use all of these properties:

- container_name: A unique name assigned to the container instance; can be distinct from the service name, and replaces any default generated name that would be given to the container on creation. Particularly useful for monitoring and management of containers in larger and more complex installations, or cases where there are multiple instances of a particular service.
- image: The Docker image to use for the service, streamed from a remote repository if not locally available.
- depends_on (optional): Defines services this service relies on and waits for them to be started before starting itself. Is used to control both the boot order of containers,

as well as automating the restarting of containers if a dependency fails. May be configured to wait for a service to start, to have finished, or be 'healthy' according to that service's health check (see below).

- **healthcheck (optional):** Defines a health check to ensure the service is running and healthy before other dependent services start. Some containers already have a health check defined, others need a custom one implemented in the compose file; in stock WISE-WARE, 2 examples using netcat have been included so Kafka will wait for Zookeeper to start up properly, and Kafka's dependents will wait for it in turn.
- **ports (optional):** Exposes ports from the container to the host machine, allowing external access to the service. Maps one of the host machine's ports to one of the service's ports, which do not need to be the same. This is not required if a container exclusively communicates with other containers in the same Docker network; services in a compose file are connected to the same network by default, unless configured otherwise.
- **environment (optional):** Sets environment variables for the containerised application. The variables available depend on the specific image, and there is not a universal set of standard variables.
- **volumes (optional):** Mounts directories from the host machine into the container for persistent storage of data or access to resources on the host machine; mounts of the latter type are typically readonly to prevent mutation of system resources. The locations in a container's filesystem that can be mounted to (and be utilised by the service) depends on the image, although many have similar structures due to being based on images of similar Linux distributions like Alpine and Ubuntu.
- **restart (optional):** Defines the restart policy for the container in case of failures if the container has been running for more than 10 seconds, eg. to always restart the container unless it was manually stopped. Relationships defined through the "depends_on" property will still cause containers to restart if one of their dependencies does.
- **privileged (optional):** Grants the container elevated privileges (use with caution).
- **network_mode (optional):** Defines how the container connects to networks; if unspecified, it is connected to the default network. If set to "host" mode, the container uses the host machine's network stack rather than connecting to a Docker network; if this container needs to connect to other containers, even those instantiated with the same compose file, it must do so through externally-exposed ports.

Adding a new service

Adding a new service to a Docker Compose file is well-introduced and explained here: [Use Docker Compose | Docker Docs](#)

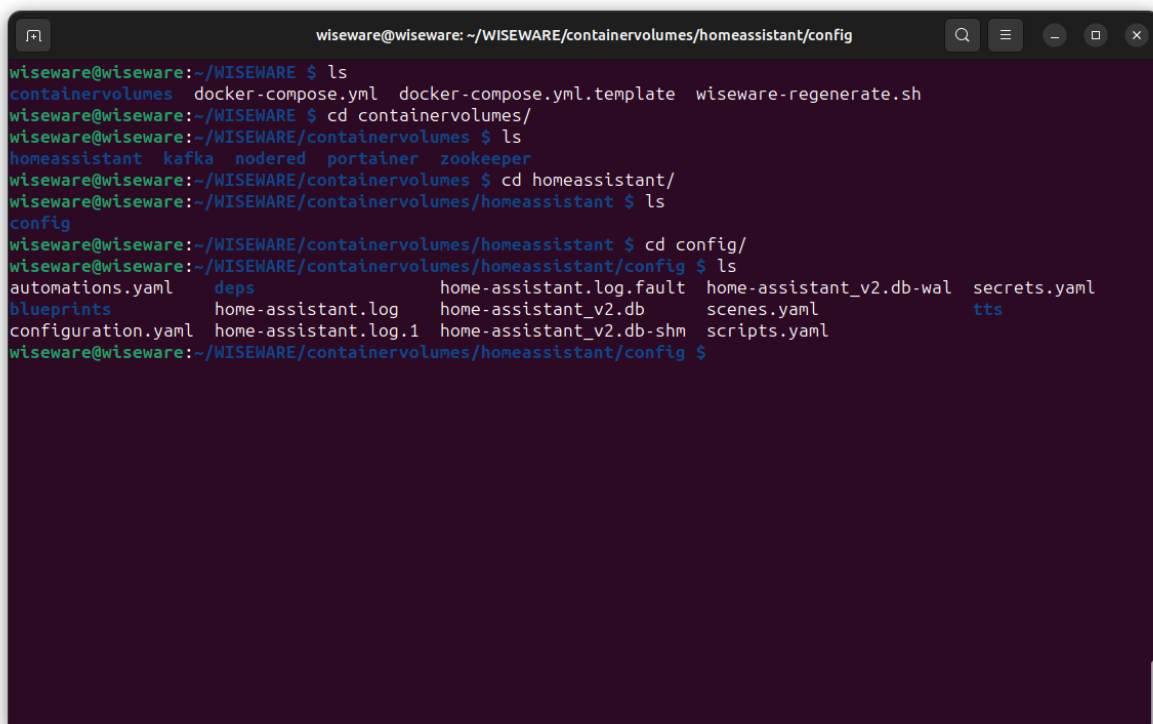
When adding a new containerised service, consider it's impact on WISE-WARE as a system:

- At what point in the boot order will the new service start, and will any other services now be dependent on it?
- If this new service is intended to be directly integrated with Home Assistant, how will they interface? There are several approaches: developing a custom integration; communicating using the REST API; adding the new service to the sidebar as an iframe, without directly integrating?
- Persistent container volumes should be confined to the “/containervolumes” folder of your WISE-WARE installation.
- Additions to “wiseware-regenerate.sh” or the compose template may be useful if setup is deterministic.
- Does it need any exposed ports, or would they be an unnecessary extension of the service's attack surface?

Home Assistant

Accessing configuration files

Home Assistant's configuration files are in /containervolumes/homeassistant/config:



```
wiseware@wiseware: ~/WISEWARE/containervolumes/homeassistant/config
wiseware@wiseware:~/WISEWARE $ ls
containervolumes  docker-compose.yml  docker-compose.yml.template  wiseware-regenerate.sh
wiseware@wiseware:~/WISEWARE $ cd containervolumes/
wiseware@wiseware:~/WISEWARE/containervolumes $ ls
homeassistant  kafka  nodered  portainer  zookeeper
wiseware@wiseware:~/WISEWARE/containervolumes $ cd homeassistant/
wiseware@wiseware:~/WISEWARE/containervolumes/homeassistant $ ls
config
wiseware@wiseware:~/WISEWARE/containervolumes/homeassistant $ cd config/
wiseware@wiseware:~/WISEWARE/containervolumes/homeassistant/config $ ls
automations.yaml  deps  home-assistant.log  home-assistant_v2.db-wal  secrets.yaml
blueprints  home-assistant.log  home-assistant_v2.db  scenes.yaml  tts
configuration.yaml  home-assistant.log.1  home-assistant_v2.db-shm  scripts.yaml
```

The main file, “configuration.yaml”, is the file first loaded by Home Assistant. It has a series of ‘include’ statements near the top, which allow for other configuration files to be used without having a single, messy file.

Integrating a new service into Home Assistant's UI

New services - whether hosted on the same machine or external to the WISE-WARE instance - can be integrated into the Home Assistant UI in several ways.

- [An iframe showing a web application](#) (used in stock WISE-WARE to integrate Node-RED)
- Communicating with the [Home Assistant REST API](#), or it's [WebSocket API](#) (most flexible integration option for existing systems)
- [Writing a Python Home Assistant integration](#) (useful for additions that can be highly coupled with Home Assistant)

Other options for integrating with WISE-WARE as a larger system are possible (for example, data processing applications using Kafka Streams), but for consolidating new functionality into the same, shared, user-friendly interface these are the main methods.

Kafka

Listeners and advertised listeners

Apache Kafka uses a system of “advertised listeners”, which essentially allow the configuration of which IP address a client will attempt to access after making it's first contact with the Kafka broker. Understanding and configuring these listeners correctly can be unintuitive without first studying this mechanism. See the linked article which serves as a good introduction: <https://www.confluent.io/en-gb/blog/kafka-listeners-explained/>.

Stock WISE-WARE has 2 advertised listeners configured; one which is for inter-container communications, and one for communication with external clients. The latter includes a placeholder value which is populated by `./wiseware-regenerate.sh` during setup.