Home Assistant

From NixOS Wiki
Jump to: navigation, search

Home Assistant (https://www.home-assistant.io/) is an open source home automation software that puts local control and privacy first. Powered by a worldwide community of tinkerers and DIY enthusiasts.

Installation methods

Upstream has defined several installation methods which they are willing to support. The NixOS module is not one of them. When you find a problem you can still report it upstream, if you are certain that the issue is relevant to upstream supported installation methods as well. If not, or if in doubt, please open an isssue on the nixpkgs issue tracker or visit the NixOS Home-Automation (https://matrix.to/#/#homeautomation:nixos.org?via=lossy.network&via=matrix.org&via=kack.it) (`#homeautomation:nixos.org`) Matrix room.

Upstream supported

Virtual machine

You could run your Home Assistant in a virtual machine, with your NixOS host providing the hypervisor.

- Install Home Assistant Operating System (home-assistant.io) (https://www.home-assistant.io/installation/linux#install-home-assistant-operating-system)
- NixOS: Headless Home Assistant VM (myme.no) (https://myme.no/posts/2021-11-25-nixos-home-assistant.html)

TODO: add example configuration

OCI container

You could run your Home Assistant in any kind of container runtime

- Install Home Assistant Container (home-assistant.io) (https://www.home-assistant.io/installation/linux#install-home-assistant-container)

Remember to allow port 8123

```
networking.firewall.allowedTCPPorts = [ 8123 ];
```

Example Configuration

Adding the following to your `configuration.nix` will bring up home-assistant listening on `:8123` on the host interface.

```
{
  virtualisation.oci-containers = {
    backend = "podman";
    containers.homeassistant = {
      volumes = [ "home-assistant:/config" ];
      environment.TZ = "Europe/Berlin";
      image = "ghcr.io/home-assistant/home-assistant:stable"; # Warning: if the tag does not change, the image wi
ll not be updated
      extraOptions = [
        "--network=host"
        "--device=/dev/ttyACM0:/dev/ttyACM0"  # Example, change this to match your own hardware
      ];
    };
  };
}
```

if there are issues with integration auto-discovery while using host networking you can open ports in a range.

Note: this snippet opens almost all available ports, this is probably not good for security!

```
networking.firewall.allowedTCPPortRanges =
  [{
     from = 100;
     to = 65535;
  }];
networking.firewall.allowedUDPPortRanges =
  [{
     from = 100;
     to = 65535;
  }];
```

to open these ports only on a specific interface. <interface name> Ex: Eth0 enp4s0

```
networking.firewall.interfaces.<interface name>.allowedTCPPortRanges =
  [{
     from = 100;
     to = 65535;
  }];
networking.firewall.interfaces.<interface name>.allowedUDPPortRanges =
  [{
     from = 100;
     to = 65535;
  }];
```

NixOS Module

You could run home-assistant using the NixOS module system at `services-home-assistant`. As of 2022-08-04 (2022.8.1) we have support for roughly 85.6% (885/1033) of components, which requires around 711 python dependencies.

Right now there are two major ways of running home-assistant using the module system:

- a) Using a fully declarative configuration
- b) Using a custom configuration, supplied by verbatim home-assistant configuration files

Using a custom configuration has the drawback, that we cannot automatically recognize and install component dependencies and it is not clear that we will continue to support these kinds of setups going forward. If you are using such a setup, please speak up in the Home-Automation room on Matrix (`#homeautomation:nixos.org`).

Declarative configuration

Set up your home-assistant by configuring the `services.home-assistant.config` attribute set as if it were your home-assistant YAML configuration. The module parses the root and platforms level to automatically discover integrations used and will add their dependencies to your home-assistant package.

The following is a minimal starting configuration, that has all the dependencies that are required to complete the initial configuration flow, that creates your first user.

```
{
  services.home-assistant = {
    enable = true;
    extraComponents = [
      # Components required to complete the onboarding
      "esphome"
      "met"
      "radio_browser"
    ];
    config = {
      # Includes dependencies for a basic setup
      # https://www.home-assistant.io/integrations/default_config/
      default_config = {};
    };
  };
}
```

If not using a reverse-proxy and you just want unencrypted access on a local network don't forget to update your firewall configuration to expose the port home-assistant is running on.

```
{
  networking.firewall.allowedTCPPorts = [ <other ports> 8123 ];
}
```

OpenSSL 1.1 is marked as insecure, refusing to evaluate

As of Home Assistant 2023.12.0 many components started depending on the `matter` integration. It unfortunately still relies on OpenSSL 1.1, which has gone end of life in 2023/09. For `home-assistant` deployments to work after this release you most likely need to allow this insecure dependency in our system configuration.

```
{
  nixpkgs.config.permittedInsecurePackages = [
    "openssl-1.1.1w"
  ];
}
```

First start

On your first start you may see multiple `ModuleNotFoundError` in Home Assistants journal log. These are dependencies required to set up devices Home Assistant already found on the network.

The appropriate component to load can be looked up in the `component-packages.nix` file, that gets auto-generated as part of the packaging process.

For example we can map the following error to

```
ModuleNotFoundError: No module named 'aioesphomeapi'
```

the `esphome` module quite easily.

```
{
  version = "2022.8.0";
  components = {
    [...]
    "esphome" = ps: with ps; [
      aioesphomeapi
      aiohttp-cors
      ifaddr
      zeroconf
    ];
    [...]
```

- https://github.com/NixOS/nixpkgs/blob/master/pkgs/servers/home-assistant/component-packages.nix (https://github.com/NixOS/nixpkgs/blob/master/pkgs/servers/home-assistant/component-packages.nix)

Using components without YAML configuration

When a component has no YAML configuration its dependencies can in theory be installed by mentioning the component name in `services.home-assistant.config.wled = {};` . This is deprecated, since Home Assistant will usually complain about the config having been migrated into the graphical user interface.

In recent versions of the home-assistant this use case has become more prominent and therefore received a more straightforward implementation, that also ensures that the component is still provided by Home Assistant.

```
{
  services.home-assistant.extraComponents = [
    "wled"
  ];
}
```

Making additional python packages available

We control the dependencies we pass into the Home Assistant python environment through module options that make the dependencies available, when their relative component was declaratively mentioned.

For other use cases like PostgreSQL support in the recorder component or the use of custom components, we provide an option to inject arbitrary dependencies from nixpkgs available python package set.

```
{
  services.home-assistant.extraPackages = python3Packages: with python3Packages; [
    # recorder postgresql support
    psycopg2

    # miele@home
    flatdict
    (callPackage ./pymiele.nix)
  ];
}
```

Using custom components

We provide a way to declaratively manage custom components through the NixOS module with the services.home-assistant.customComponents (https://search.nixos.org/options?channel=unstable&show=services.home-assistant.customComponents&from=0&size=50&sort=relevance&type=packages&query=home-assistant) option.

Custom components can be found under pkgs.home-assistant-custom-components (https://search.nixos.org/packages?channel=unstable&from=0&size=50&sort=relevance&type=packages&query=home-assistant-custom-components).

Using custom lovelace modules

We provide a way to declaratively manage custom lovelace modules through the NixOS module with the services.home-assistant.customLovelaceModules (https://search.nixos.org/options?channel=unstable&show=services.home-assistant.customLovelaceModules&from=0&size=50&sort=relevance&type=packages&query=home-assistant) option.

Custom components can be found under pkgs.home-assistant-custom-lovelace-modules (https://search.nixos.org/packages?channel=unstable&from=0&size=50&sort=relevance&type=packages&query=home-assistant-custom-lovelace-modules).

Reusing existing YAML configuration

The module also supports passing it an existing configuration, however that comes with certain drawbacks. For example we cannot automatically detect the components, that your configuration requires. In that scenario you will need to resolve dependencies manually using the packages `extraComponents` parameter. Also you will be unable to reuse configuration values between parts of your NixOS configuration. A barebones setup to get you started may look like this:

```
{
  services.home-assistant = {
    enable = true;
    # Pass the path to the directory where your configuration.yaml
    # resides, /var/lib/hass might be a good location.
    configDir = /var/lib/hass;
    # Override the package to handle dependency management manually
    package = (pkgs.home-assistant.override {
      # https://github.com/NixOS/nixpkgs/blob/master/pkgs/servers/home-assistant/component-packages.nix
      extraComponents = [
        "default_config"
        "esphome"
        "met"
      ];
      extraPackages = ps: with ps; [
        # Are you using a database server for your recorder?
        # https://www.home-assistant.io/integrations/recorder/
        #mysqlclient
        #psycopg2
      ];
    })
  };
}
```

You may find the following script helpful. It looks up missing dependencies from the `home-assistant.service` systemd unit journal: https://gist.github.com/AngryAnt/74c047a2b8438517c822ffdd9663aa57 (https://gist.github.com/AngryAnt/74c047a2b8438517c822ffdd9663aa57)

Running a recent version using an overlay

Home Assistant is a fast-paced open source project, that currently features one major release every month, and a handful of minor ones in between. Firmwares and API endpoints tend to change from time to time, so Home Assistant and its bindings need to keep up to keep things work. The version we provide at the branch off is just a snapshot in time, and does not receive any updates, because there would just be too many dependencies to backport. But with NixOS it is still possible to use the version in nixpkgs/unstable by creating an overlay and using the module from nixos-unstable.

```
let
  # Track NixOS unstable via nix-channel, or replace it with something like niv at your own discretion
  # nix-channel --add http://nixos.org/channels/nixos-unstable nixos-unstable
  unstable = import <nixos-unstable> {};
in
{
  nixpkgs.overlays = [
    (self: super: {
      inherit (unstable) home-assistant;
    })
  ];

  disabledModules = [
    "services/home-automation/home-assistant.nix"
  ];

  imports = [
    <nixos-unstable/nixos/modules/services/home-automation/home-assistant.nix>
  ];
}
```

Snippets

Reverse Proxying with nginx

If you run a public Home Assistant instance it is a good idea to enable SSL/TLS. The following configuration generates a certificate using letsencrypt:

```
  services.home-assistant.config.http = {
    server_host = "::1";
    trusted_proxies = [ "::1" ];
    use_x_forwarded_for = true;
  };

  services.nginx = {
    enable = true;
    recommendedProxySettings = true;
    virtualHosts."home.example.com" = {
      forceSSL = true;
      enableACME = true;
      extraConfig = ''
        proxy_buffering off;
      '';
      locations."/" = {
        proxyPass = "http://[::1]:8123";
        proxyWebsockets = true;
      };
    };
  };
```

Using PostgreSQL

Home Assistant supports PostgreSQL as a database backend for, among other things, its logger and history components. It's a lot more scalable and typically provides faster response times than the SQLite database, that is used by default.

Remember to make backups of your database, for Home Assistant is becoming more and more stateful and has moved away from a completely declarative YAML configuration for new and core components.

Also note that when overridding the package you may want to disable install checks as they tend to take a long time to complete.

```
services.home-assistant = {
  package = (pkgs.home-assistant.override {
    extraPackages = py: with py; [ psycopg2 ];
  }).overrideAttrs (oldAttrs: {
    doInstallCheck = false;
  });
  config.recorder.db_url = "postgresql://@/hass";
};

services.postgresql = {
  enable = true;
  ensureDatabases = [ "hass" ];
  ensureUsers = [{
    name = "hass";
    ensureDBOwnership = true;
  }];
};
```

Add custom lovelace modules

There are many useful and pretty lovelace components out there that you might want to integrate into your dashboards. Some of them are packaged up inside NUR repositories.

Usually you would install these into `/var/lib/hass/www` , but that comes with issues on NixOS, as their internal webserver does not follow symlinks. You could then think about exposing that directory over your webserver, but you'll notice that your webserver user has no permissions to enter the home assistant users state directory, which is due to proper hardening on the `home-assistant.service` .

```
let
  # https://nur.nix-community.org/repos/mweinelt/
  nur = import (builtins.fetchTarball "https://github.com/mweinelt/nur-packages/archive/master.tar.gz") {};

  mkLovelaceModule = name: {
    # https://www.home-assistant.io/lovelace/dashboards-and-views/#resources
    url = "/local/${name}.js?${nur.hassLovelaceModules."${name}".version}";
    type = "module";
  };
in {
  # Install lovelace components into temporary directory that can be
  # served by nginx.
  systemd.tmpfiles.rules = [
    "d /run/hass 0700 nginx nginx"
    "L+ /run/hass/mini-graph-card.js - - - - ${nur.hassLovelaceModules.mini-graph-card}/mini-graph-card-bundle.j
s"
    "L+ /run/hass/mini-media-player.js - - - - ${nur.hassLovelaceModules.mini-media-player}/mini-media-player-bun
dle.js"
    "L+ /run/hass/multiple-entity-row.js - - - - ${nur.hassLovelaceModules.multiple-entity-row}/multiple-entity-r
ow.js"
  ];

  # Instruct home-assistant to load these resources in the lovelace frontend
  services.home-assistant.config.lovelace = {
    resources = [
      (mkLovelaceModule "mini-graph-card") # https://github.com/kalkih/mini-graph-card
      (mkLovelaceModule "mini-media-player") # https://github.com/kalkih/mini-media-player
      (mkLovelaceModule "multiple-entity-row") # https://github.com/benct/lovelace-multiple-entity-row
    ];
  };

  services.nginx.virtualHosts."home.example.com" = {
    locations."/local/" = {
      alias = "/run/hass/";
    };
  };
}
```

Add custom components

In order to install a custom component, you have to place it in `/var/lib/hass/custom_components` . This can be achieved using systemd tmpfiles like so (for sonoff custom component):

```
  systemd.tmpfiles.rules = [
    "C /var/lib/hass/custom_components/sonoff - - - - ${sources.sonoff-lan}/custom_components/sonoff"
    "Z /var/lib/hass/custom_components 770 hass hass - -"
  ];
```

Combine declarative and UI defined automations

You can also declaratively define your automations while still being able to define them in Home Assistant UI. Automations defined in UI are stored in `/var/lib/hass/automations.yaml` . If you are not planning on using UI defined automations, then you can define them under `services.home-assistant.config.automation` , otherwise split them into `services.home-assistant.config."automation manual"` and `services.home-assistant.config."automation ui"` , like so:

```
  services.home-assistant.config =
  {
    "automation manual" = [
      {
        alias = "living room plug off";
        trigger = {
          platform = "time";
          at = "22:00";
        };
        action = {
          type = "turn_off";
          device_id = "someID"; #Inspect yaml of automation created in UI
          entity_id = "switch.living_room_plug";
          domain = "switch";
        };
      }
    ];
    "automation ui" = "!include automations.yaml";
 }
```

If you did not create any automations through the UI, Home Assistant will fail loading because the `automations.yaml` file does not exist yet and it will fail including it. To avoid that, add a systemd tmpfiles.d rule:

```
systemd.tmpfiles.rules = [
  "f ${config.services.home-assistant.configDir}/automations.yaml 0755 hass hass"
];
```

Combine declarative and UI defined scenes

Scenes are configured the same way automations(described above) are.

```
services.home-assistant.config."scene manual" = [];
services.home-assistant.config."scene ui" = "!include scenes.yaml";
```

Trust a private certificate authority

Home Assistant does not natively support adding a private CA to the certificate store (see this thread (https://community.home-assistant.io/t/add-private-cas-to-certificate-store/267452) for more details).

Home Assistant trusts certificates provided by the certifi python package, which nix overwrites with the cacert package. Using overrides you can append your root CA certificate to the certificates provided by certifi.

```
services.home-assistant.package = (pkgs.home-assistant.override {
  extraPackages = py: with py; [ ];
  packageOverrides = final: prev: {
    certifi = prev.certifi.override {
      cacert = pkgs.cacert.override {
        extraCertificateFiles = [ ./my_custom_root_ca.crt ];
      };
    };
}).overrideAttrs (oldAttrs: {
  doInstallCheck = false;
});
```

Example configurations

- Mic92's config (https://github.com/Mic92/dotfiles/tree/main/machines/eve/modules/home-assistant)

Misc

Run Home Assistant from GitHub repository

When developing Home Assistant for some test dependencies additional libraries are needed. A nix-shell expression for this is available here (https://github.com/nix-community/nix-environments).

Retrieved from "https://nixos.wiki/index.php?title=Home_Assistant&oldid=13466 (https://nixos.wiki/index.php?title=Home_Assistant&oldid=13466)"

Category (/wiki/Special:Categories): Applications (/wiki/Category:Applications)