Control anything SSH with Homeassistant's Hass.io

What is SSH and how to use it for controlling other machines?

SSH is a powerful command line-tool allowing for you to connect to a remote machine and issue any shell command remotely. Basically SSH is a way of **sending** a command to a remote machine without doing the typing in by sitting front of the computer yourself. This means you can do things like shutdowns, maintenance, updates or anything else you want with SSH (as long as you have the username and the password of that particular server). A good example would be shutting down remote server using this SSH command:

```
ssh donald@192.168.1.2 'sudo reboot'
```

This command will effectively restart a machine with IP 192.168.1.2 machine. Cool but how do you do that using Homeassistant? By using the command_line component. This is all fine and dandy but if you tried to connect to donald using this command you probably noticed that you needed a password for that. And now you have probably realized that you will not be able to put your password to the command_line component, since password request interactive. Luckily for you can do SSH commands without entering any passwords and yet still be safe at the same time. For this feat you will need a **public/private** key pair that will be used to connect to your remote machine without any passwords.

Can you even use SSH with Hass.io?

After my migration from Hassbian to Hass.io, I ran into an issue. The issue was that I could not send proper SSH commands to a computer running Hyperion. These issues with SSH were in fact caused by the concept of Docker isolation. This means that my Homeassistant's instance was completely separated from my host machine, thus not allowing me to run executables such as SSH and others. Luckily official Homeassistant Docker image has SSH client installed to the container so this mean that you can call SSH commands, but yet again there is a catch. Can you guess what is that catch? As mentioned before we NEED to authenticate ourselves using a password OR public/private key pair. So let us begin by making a public/private key pair.

Generating the SSH Keys

In reality, we need to generate ssh keys both on the **FROM** machine and the **TO** machine. Because this is the only way (in SSH) for BOTH machines can prove their identity to each other. In this tutorial a machine that we issue commands FROM will be called the MASTER & the machine and the machine executing commands SLAVE in our case MASTER is the machine running HASS.io instance.

Tutorial

In this tutorial we will make button in Homeassistant that when pressed will shut down our SLAVE server via SSH. Basically it will append a text file every time we press a button. This example will be a good starting point for controlling remote devices.

Prerequisites:

- An SSH connection to your Hass.io ResinOS host

In my case IP addresses were:

- MASTER IP **192.168.0.105** and SSH port **22222**
- SLAVE IP: **192.168.0.111**

Make SSH keys both on MASTER and the SLAVE

Generate SSH keys on HASS.io Homeassistant docker container

Set up MASTER:

Connect to the MASTER. This will not work "out of the box" so first follow official tutorial on how to connect to the HASS.io host running ResinOS.

```
ssh root@192.168.0.105 -p 22222
```

Now attach to the Homeassistant docker container. List available docker containers.

```
docker ps -a
```

Find one looking something like homeassistant/homeassistant3. Copy its CONTAINER ID looking something like b7dfc2f4d0c4. Then attach to your container,

```
docker exec -it b7dfc2f4d0c4 /bin/bash
```

now finally generate your SSH key, but this time in a different directory

```
mkdir /config/ssh
ssh-keygen -t rsa -f /config/ssh/id_rsa
```

lets checkout our two brand-new PUBLIC & PRIVATE keys

```
cd /config/ssh
ls -al
cat id_rsa.pub
```

if all went well you **public key** output should look something like this:

```
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQDyiprxeHAAieq2YtiXhFgSQIhZwvY6zsPAh
```

IMPORTANT: copy this value it is your PUBLIC key we will use later!

Set up SLAVE:

For testing purposes lets create a user called mister.slave connect to your SLAVE machine

```
ssh myUser@192.168.0.111
```

add a user called mister.slave

```
sudo useradd mister.slave sudo
sudo passwd mister.slave
```

This step may differ because Unix based distros use different commands for enabling sudo on your user in my case (Debian 9) I just had to write **sudo** when creating a new mister.slave user.

This will require a password write something memorable. We will delete this user later anyway. After creating a mister.slave switch to it:

```
sudo su mister.slave
```

Now that you became mister.slave user lets **finally** make our SSH key pair

```
ssh-keygen -t rsa
```

Okay we have our keys setup but what about passwordless connection from MASTER? Its easy we have to add the **public key** we copied before to our SLAVE machines' authorized_keys file.

```
echo "PASTE YOUR MASTER KEY INSTEAD OF THIS TEXT" >> ~/.ssh/authorized_ke
```

On some systems authorized_keys file must have specific Unix permissions set , and it will fail silently if you do not add these permissions so let's do that. Line below will only make the file readable and writable to our user.

```
chmod 600 ~/.ssh/authorized_keys
```

Okay our connection should be good to go. So what we just did is we created an SSH key pair on both MASTER and the SLAVE machines & we installed MASTERS **public key** onto SLAVE. Now the SLAVE trusts the MASTER machine and allows it to connect without a password.

You will probably be interested in running some commands with **sudo** without a password. For this we will need to add these lines to /etc/sudoers file.

WARNING: You should be very careful when editing /etc/sudoers file one bad character could lock you out from the system forever!

```
sudo visudo
```

Since visudo uses vim text editor it will require you to know some commands. Do not worry I will denote editor command like this: [[ ]]. Everything in the [[ ]] are editor commands that you will have to type **manually** on your keyboard.

Go to the bottom of the file

```
[[ shift + g ]] or simply [[ G ]]
```

enter insert mode

```
[[ i ]]
```

Paste this to the end of the file.

```
mister.slave    ALL=(ALL) NOPASSWD:    ALL
```

Exit text editor and save contents

```
[[ :wq ]]
```

If all went well you should be able to run sudo commands without password, lets test it out!

```
sudo whoami
```

Expected output should be root. So if console printed root and did not ask you for password congrats! You can now run all sudo commands without a having to enter your users password.

Test SSH connection from MASTER to SLAVE

let's connect to our MASTER machine again

```
ssh root@192.168.0.105 -p 22222
```

then lets issue a REBOOT command to our SLAVE

```
docker exec -it b7dfc2f4d0c4 /bin/bash
ssh -i /config/ssh/id_rsa -o StrictHostKeyChecking=no mister.slave@192.16
```

If all went well you congrats again you rebooted your `mister.slave` from Homeassistant manually!

`-i /config/ssh/id_rsa`: Defines in which directory our private SSH key can be found. `-o StrictHostKeyChecking=no`: Says to your SSH client to not prompt you with warning messages or yes/no questions when host has changed.

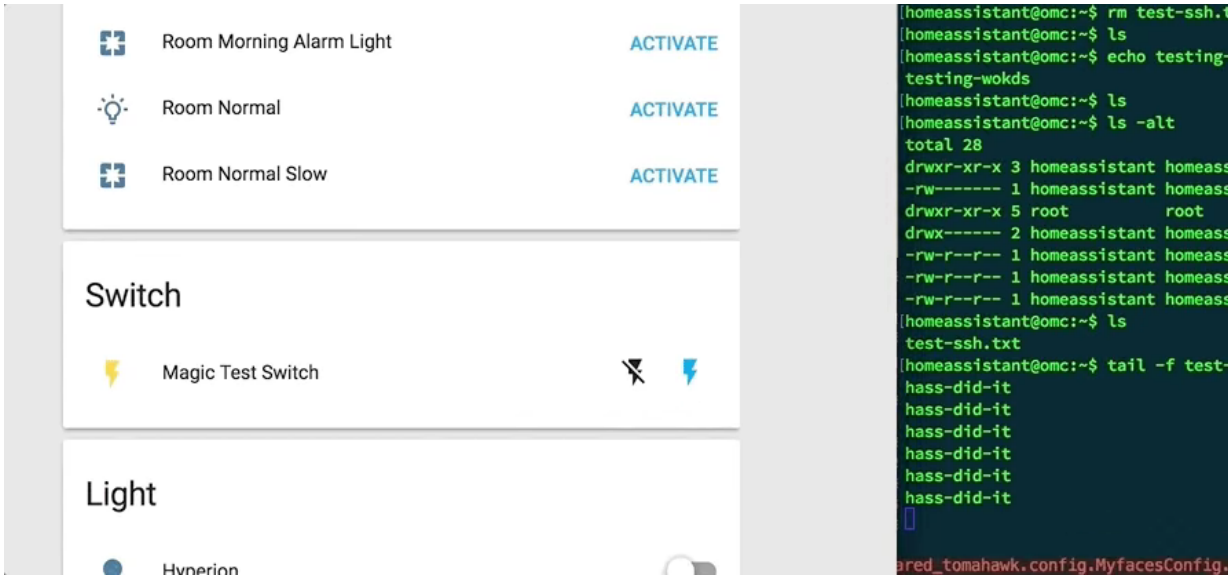
Keep in mind that `b7dfc2f4d0c4` is a unique ID of my docker container yours will definitely vary! if you are confused refer to steps we did in the beginning Set Up MASTER chapter. When your SLAVE has rebooted you can continue to Add Homeassistant Components.

Add Homeassistant Components

Add these lines to your `configuration.yaml` file and then restart Hass.io

```
switch:
  - platform: command_line
    switches:
      test_ssh:
        command_on: "ssh -i /config/ssh/id_rsa -o StrictHostKeyChecking=n
    friendly_name: Magic Test Switch
```

Now go to your Homeassistant dashboard and press your newly created button.



Conclusions

If you followed this tutorial thoroughly you should be good to seed to start for remote control via SSH. If you are a beginner this guide is a hard and if you managed to reboot something from Homeassistant you should pat your self on the back. Personally I used this method for turning on effects on my Hyperion daemon. Also, I am planning to make shutdown and reboot buttons on other server I have at home. One thing to mention though is that you will not have any console output when you call a remote command using `command_line` component, so monitoring things on a remote machine is not possible. You can check out my Hass.io config file for further inspiration here

return 33

Posted on: *04/25/21 17:13:36*, ***last edited on:*** *04/25/21 17:13:36*, ***written by:*** *GitHub*

Home RSS Atom