Using HashiCorp products at home

Eléonore Carpentier
(She/Her)

Cyber Security Engineer
at SolarisBank

Corentin Mors
(He/Him)

Software Engineer
at Dashlane

HashiCorp

Are you using IoTs at home?

There are everywhere.

Let's find them.


MONOPOLY
EDITION DE REIMS


197
réal
YI


RIP

Who's taking the picture?

Fair enough.


Canon
EOS
Please
Smile
Splendor

Using HashiCorp products at home

Eléonore Carpentier
(She/Her)

Cyber Security Engineer
at SolarisBank

Corentin Mors
(He/Him)

Software Engineer
at Dashlane

HashiCorp

Some holidays,
A hackathon,
3 lamps

Giving you some context

HashiCorp Holidays Hackathon

HashiCraft Holidays Hackstravaganza is a hackathon event for the community where we were invited to use our creativity to showcase one or more Hashicorp products in an unexpected way.

Our idea

We decided to make an uncommon use of Terraform and Vault by including them in our own home. To do this, we created a Terraform provider and a Vault extension for Home Assistant (an open-source platform to centrally manage IoTs) running on a Raspberry Pi.

What's Home Assistant?

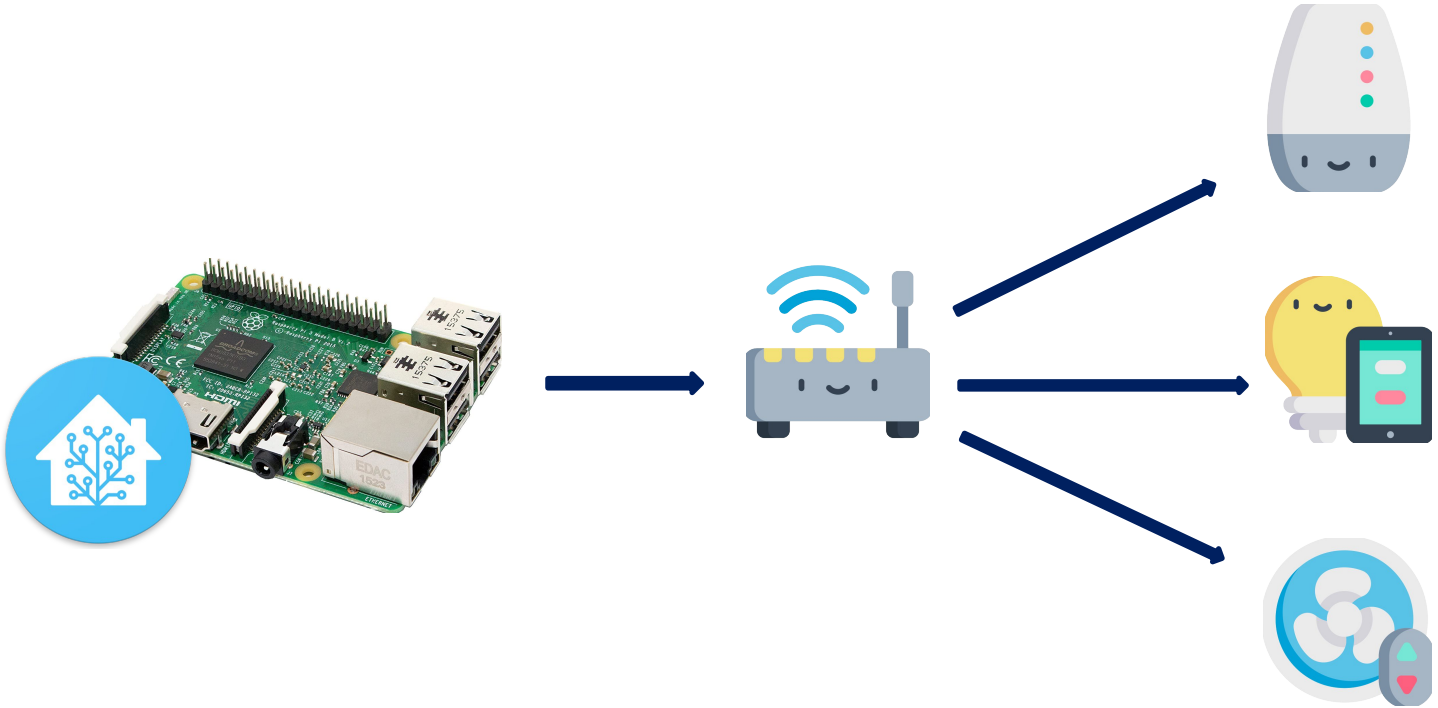
An aggregator for all your IoTs

You can easily connect all your devices to Home Assistant from your lamps, TV, speakers, vacuum cleaner...

Building fun

Home Assistant is open-source and open to any contribution you can add lot of extensions to fit your needs.

A small schema



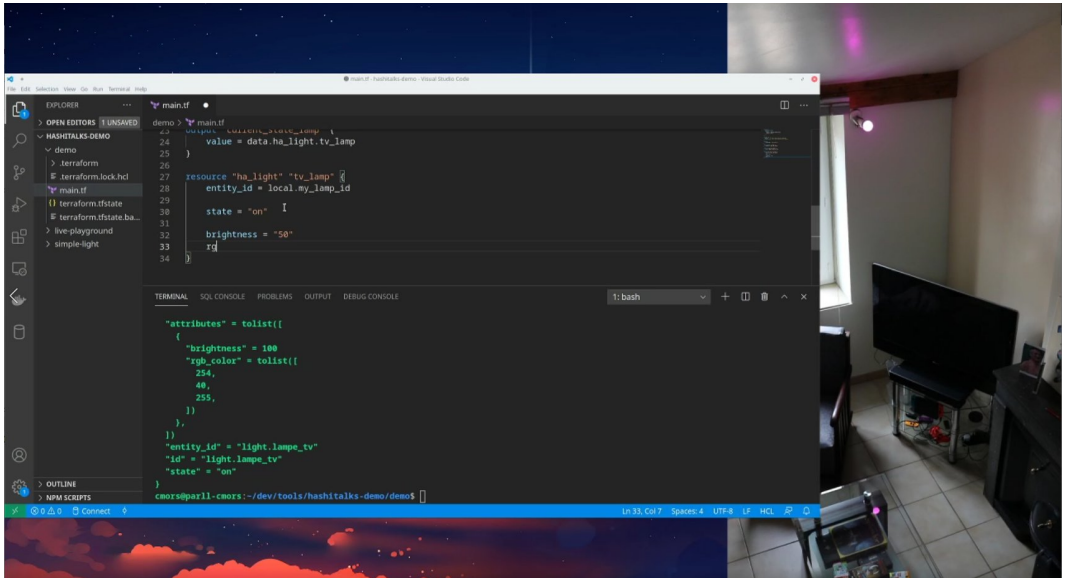
Terraform to control your home

Controling lamps with Terraform

It's demo time

A lamp and some TF code

In this demo, you'll have a glimpse of how to control your home with Terraform.



Let's take a look at the Terraform provider

client.go

```go
// Client is a strucutre containing all information about connection to host
type Client struct {
	HostURL    string
	HTTPClient *http.Client
	Token      string
}

func (c *Client) doRequest(req *http.Request) ([]byte, error) {
	req.Header.Set("Authorization", c.Token)

	res, err := c.HTTPClient.Do(req)
	if err != nil {
		return nil, err
	}
	defer res.Body.Close()

	body, err := ioutil.ReadAll(res.Body)
	if err != nil {
		return nil, err
	}

	if res.StatusCode != http.StatusOK {
		return nil, fmt.Errorf("status: %d, body: %s, reqUrl: %s, reqBody: %s", res.StatusCode, body, req.URL, req.Body)
	}

	return body, err
}
```

Let's take a look at the Terraform provider

client.go

```go
// Client is a strucutre containing all information about connection to host
type Client struct {
	HostURL    string
	HTTPClient *http.Client
	Token      string
}

func (c *Client) doRequest(req *http.Request) ([]byte, error) {
	req.Header.Set("Authorization", c.Token)

	res, err := c.HTTPClient.Do(req)
	if err != nil {
		return nil, err
	}
	defer res.Body.Close()

	body, err := ioutil.ReadAll(res.Body)
	if err != nil {
		return nil, err
	}

	if res.StatusCode != http.StatusOK {
		return nil, fmt.Errorf("status: %d, body: %s, reqUrl: %s, reqBody: %s", res.StatusCode, body, req.URL, req.Body)
	}

	return body, err
}
```

Let's take a look at the Terraform provider

client.go

```go
// Client is a strucutre containing all information about connection to host
type Client struct {
	HostURL    string
	HTTPClient *http.Client
	Token      string
}

func (c *Client) doRequest(req *http.Request) ([]byte, error) {
	req.Header.Set("Authorization", c.Token)

	res, err := c.HTTPClient.Do(req)
	if err != nil {
		return nil, err
	}
	defer res.Body.Close()

	body, err := ioutil.ReadAll(res.Body)
	if err != nil {
		return nil, err
	}

	if res.StatusCode != http.StatusOK {
		return nil, fmt.Errorf("status: %d, body: %s, reqUrl: %s, reqBody: %s", res.StatusCode, body, req.URL, req.Body)
	}

	return body, err
}
```

Let's take a look at the Terraform provider

client.go

```go
// Client is a strucutre containing all information about connection to host
type Client struct {
	HostURL    string
	HTTPClient *http.Client
	Token      string
}

func (c *Client) doRequest(req *http.Request) ([]byte, error) {
	req.Header.Set("Authorization", c.Token)

	res, err := c.HTTPClient.Do(req)
	if err != nil {
		return nil, err
	}
	defer res.Body.Close()

	body, err := ioutil.ReadAll(res.Body)
	if err != nil {
		return nil, err
	}

	if res.StatusCode != http.StatusOK {
		return nil, fmt.Errorf("status: %d, body: %s, reqUrl: %s, reqBody: %s", res.StatusCode, body, req.URL, req.Body)
	}

	return body, err
}
```

Let's take a look at the Terraform provider

schemas

```
func resourceLight() *schema.Resource {
        return &schema.Resource{
                CreateContext: resourceLightCreate,
                ReadContext:   resourceLightRead,
                UpdateContext: resourceLightUpdate,
                DeleteContext: resourceLightDelete,
                Schema: map[string]*schema.Schema{
                        "entity_id": {
                                Type:        schema.TypeString,
                                Required:    true,
                                Description: "ID of the resource in Home Assistant",
                        },
                        "state": {
                                Type:        schema.TypeString,
                                Required:    true,
                                Description: "State of the light on/off",
                        },
                        "brightness": {
                                Type:        schema.TypeInt,
                                Optional:    true,
                                Description: "Brightness of the light from 0 to 255",
                        },
                        "rgb_color": {
                                Type:     schema.TypeList,
                                Optional: true,
                                Elem: &schema.Schema{
                                        Type: schema.TypeInt,
                                },
                                Description: "Array of colors : red, green, blue",
                        },
```

Let's take a look at the Terraform provider

schemas

```
func resourceLight() *schema.Resource {
        return &schema.Resource{
                CreateContext: resourceLightCreate,
                ReadContext:   resourceLightRead,
                UpdateContext: resourceLightUpdate,
                DeleteContext: resourceLightDelete,
                Schema: map[string]*schema.Schema{
                        "entity_id": {
                                Type:        schema.TypeString,
                                Required:    true,
                                Description: "ID of the resource in Home Assistant",
                        },
                        "state": {
                                Type:        schema.TypeString,
                                Required:    true,
                                Description: "State of the light on/off",
                        },
                        "brightness": {
                                Type:        schema.TypeInt,
                                Optional:    true,
                                Description: "Brightness of the light from 0 to 255",
                        },
                        "rgb_color": {
                                Type:     schema.TypeList,
                                Optional: true,
                                Elem: &schema.Schema{
                                        Type: schema.TypeInt,
                                },
                                Description: "Array of colors : red, green, blue",
```

Let's take a look at the Terraform provider

schemas

```
func resourceLight() *schema.Resource {
        return &schema.Resource{
                CreateContext: resourceLightCreate,
                ReadContext:   resourceLightRead,
                UpdateContext: resourceLightUpdate,
                DeleteContext: resourceLightDelete,
                Schema: map[string]*schema.Schema{
                        "entity_id": {
                                Type:        schema.TypeString,
                                Required:    true,
                                Description: "ID of the resource in Home Assistant",
                        },
                        "state": {
                                Type:        schema.TypeString,
                                Required:    true,
                                Description: "State of the light on/off",
                        },
                        "brightness": {
                                Type:        schema.TypeInt,
                                Optional:    true,
                                Description: "Brightness of the light from 0 to 255",
                        },
                        "rgb_color": {
                                Type:     schema.TypeList,
                                Optional: true,
                                Elem: &schema.Schema{
                                        Type: schema.TypeInt,
                                },
                                Description: "Array of colors : red, green, blue",
```

Let's take a look at the Terraform provider

example of update

```
func resourceLightUpdate(ctx context.Context, d *schema.ResourceData, m interface{}) diag.Diagnostics {
        c := m.(*hac.Client)

        // Warning or errors can be collected in a slice type
        var diags diag.Diagnostics

        id := d.Get("entity_id").(string)
        state := d.Get("state").(string)
        brightness := d.Get("brightness").(int)
        rgbColor := d.Get("rgb_color").([]interface{})

        o, err := c.SetLightState(hac.LightParams{
                ID:         id,
                Brightness: brightness,
                RgbColor:   rgbColor,
        }, state)
        if err != nil {
                return diag.FromErr(err)
        }
```

Let's take a look at the Terraform provider

example of update

```
func resourceLightUpdate(ctx context.Context, d *schema.ResourceData, m interface{}) diag.Diagnostics {
        c := m.(*hac.Client)

        // Warning or errors can be collected in a slice type
        var diags diag.Diagnostics

        id := d.Get("entity_id").(string)
        state := d.Get("state").(string)
        brightness := d.Get("brightness").(int)
        rgbColor := d.Get("rgb_color").([]interface{})

        o, err := c.SetLightState(hac.LightParams{
                ID:         id,
                Brightness: brightness,
                RgbColor:   rgbColor,
        }, state)
        if err != nil {
                return diag.FromErr(err)
        }
```

Let's take a look at the Terraform provider

example of update

```
func resourceLightUpdate(ctx context.Context, d *schema.ResourceData, m interface{}) diag.Diagnostics {
        c := m.(*hac.Client)

        // Warning or errors can be collected in a slice type
        var diags diag.Diagnostics

        id := d.Get("entity_id").(string)
        state := d.Get("state").(string)
        brightness := d.Get("brightness").(int)
        rgbColor := d.Get("rgb_color").([]interface{})

        o, err := c.SetLightState(hac.LightParams{
                ID:         id,
                Brightness: brightness,
                RgbColor:   rgbColor,
        }, state)
        if err != nil {
                return diag.FromErr(err)
        }
```

All of them together: Extending HA Terraform plugin

Creating scenes

Automate your favorite home setup

with pre-configured scenes

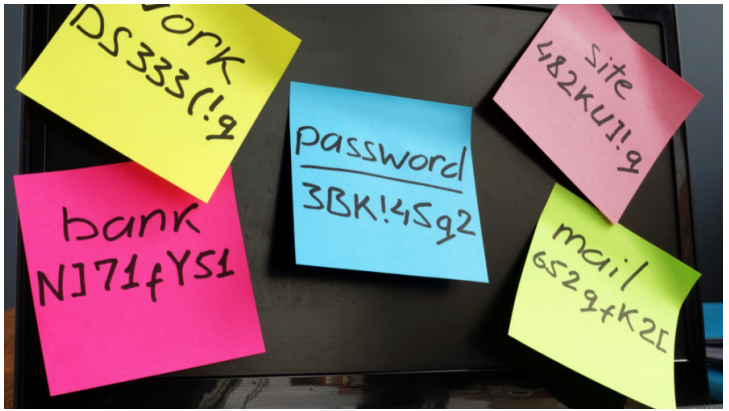
All of them together: Extending HA Terraform plugin

Adding more devices types

Extanding support for multiple IoT device to facilitate managing our home setup as code

Handling Terraform secrets with HA Vault

Where can we store home secrets?
In a post-it?
In a password manager?



How to access them programatically?

But to do so ... we need a home Vault

Vault at Home

A Vault addon for Home Assistant

Explanation

Vault is shipped as addon for Home Assistant running on a docker instance.

Vision

We aim to use Vault to store all secrets you need at Home.

github.com/tidalf/ha-addons

Configurable to fit your needs

Giving you explanation and demos

AWS KMS

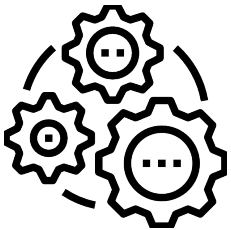
Unseal automatically utilizing an encryption key from AWS Key Management Service.

Keybase

Encrypt your unsealing keys with keybase to get an additional layer of security.

Pre-configuration

Ability to create a pre-configured admin user uppon start.



Various use cases

Having a Vault at Home is cool and safer

- Wifi password (one-time)
- SSH on your own machines
- Backend MySQL (rotate - one-time - give it your root user)
- PGP keys
- Cryptocurrencies wallet key
- Any IoT secret

What to remember?

Three things

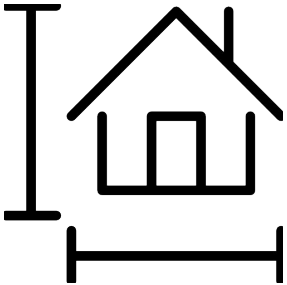
that you should keep in mind

Terraform is not that complex!

Go get into coding a Terraform provider even if you don't know Go lang.

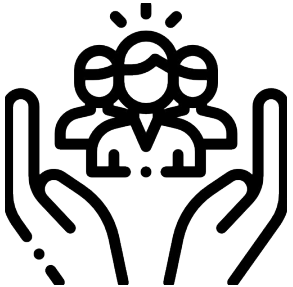
Can apply to any environment

No matter the size of your "company", it can be as small as your own house.



Large community

Lot of cool resources available on Internet such as awesome tutorials or cool and unexpected tools.



Thank You

Corentin : @mikescops on Twitter
Eléonore : @eleonore-carpentier on LinkedIn

Our projects

- Vault HA addon : github.com/tidalf/ha-addons
- Terraform HA provider : github.com/mikescops/terraform-provider-homeassistant
- Holidays Terraform module : github.com/shaeli/nowel