Hauptkonfiguration - bffh.dhall

BFFH verwendet Dhall für die Struktur der Konfigurationsdateien BFFH verwendet RBAC für die Zugriffskontrolle.

Die Konfiguration von BFFH befindet sich in der Datei `bffh.dhall`. Die Datei kann auch umbenannt werden. Wichtig ist, dass sie dann überall korrekt referenziert wird (z.B. in Service Scripts).

Tipps & Tricks in Dhall

Kommentare

Zeilen, die mit `--` beginnen, werden automatisch als Kommentar gewertet

Imports, Umgebungsvariablen, URLs - Dhall-Konfigurationen verschlanken

Innerhalb einer Dhall können weitere Dhall-Dateien referenziert werden (Import Statements). Dadurch lassen sich Konfigurationen schlank und visuell gut in kleine Blöcke aufteilen. Außerdem lassen sich auch Umgebungsvariablen verwenden, sowie Inhalte aus URLs abrufen. Siehe https://github.com/dhall-lang/dhall-lang/blob/master/standard/imports.md

```
    -- Aus einer URL importierter Ausdruck
    let concatSep = http://prelude.dhall-lang.org/Prelude/Text/concatSep
sha256:fa909c0b2fd4f9edb46df7ff72ae105ad0bd0ae00baa7fe53b0e43863f9bd34a

in  { name = env:USER as Text -- Aus der Umgebung importierte Ausdrücke
    , age  = 23
    , publicKey = ~/.ssh/id_rsa.pub as Location  -- Pfad gelesen als Dhall-Ausdruck
    , hobbies = concatSep ", " [ "piano", "reading", "skiing" ]
    } : ./schema.dhall  -- Aus einer Datei importierter Ausdruck
```

Sie können Importe mit Integritätsprüfungen schützen, wenn Sie einen SHA-256-Hash anhängen (wie beim obigen `concatSep`-Import), und Sie können einen Wert auch als Rohtext importieren, indem Sie ihn `as Text` anhängen (wie beim obigen `env:USER`-Import),

oder als aufgelösten Pfad, indem Sie ihn `as Location` anhängen.

Importierte Ausdrücke können transitiv andere Ausdrücke importieren. Die oben importierte Datei `./schema.dhall` kann zum Beispiel auch andere Dateien importieren:

```
-- ./schema.dhall


{ name : ./schema/name.dhall
, age  : ./schema/age.dhall
, hobbies : ./schema/hobbies.dhall
}
```

... und wenn `./schema/hobbies.dhall` enthielte einen relativen Import wie z.B.:

```
-- ./schema/hobbies.dhall


List ./hobby.dhall
```

... dann würde sich der relative Import von `./hobby.dhall` tatsächlich auf `./schema/hobby.dhall` beziehen. Dies ist als "Importverkettung" bekannt: die Auflösung von Importen relativ zur Position des aktuellen Ausdrucks.

Die Konfiguration von bffh

Allgemeine Einstellungen

listens

Enthält die Adressen, auf die BFFH bei der Verbindung für die API hört. Standardport für BFFH ist `59661`

Beispiel:

```
listens =
[
   { address = "127.0.0.1", port = Some 59661 }
]
```

mqtt_url

Enthält die Adresse des MQTT-Servers, mit dem sich BFFH verbindet.

Die Adresse hat das Format `<protocol>://[user]:[password]@<server>:[port]`

- `protocol` wird benötigt und kann eins der folgenden Werte annehmen: `mqtt`,`tcp`,`mqtts`, `ssl`
- `user` und `password` sind optional
- `server` ist erforderlich und kann eine IP-Adresse oder ein Hostname sein
- `port` ist optional. Der Standardport ist `59661`

Beispiele:

```
mqtt_url = "tcp://localhost:1883"
mqtt_url = "mqtts://user:password@server.tld:port"
```

db_path

Enthält den Pfad für die interne Datenbank, die BFFH verwendet. BFFH wird zwei Dateien erstellen: `<db_path>` und `<db_path>-lock`. Es sollte sichergestellt werden, dass BFFH Schreibzugriff auf das entsprechende Verzeichnis hat.

Beispiel:

```
db_path = "/tmp/bffh"
```

Berechtigungen (Permissions)

Standardberechtigungen

BFFH verfügt über einige Standardberechtigungen, die der Verwaltung und den **Admin**-Rechten zugewiesen werden können.

`bffh.users.info` - Nutzerliste bekommen und Infos über diese Accounts erhalten
`bffh.users.manage` - Nutzerliste bekommen und Nutzer verwalten
`bffh.users.admin` - Globale Administration: Nutzer hinzufügen, löschen, ändern (z.B. Passwort-Reset)

Modellieren von Berechtigungen

Allgemeines Schema: `space.type.category.permission.model`

Administrator
`space.machines.printers.*`

Offene Berechtigung
`space.machines.printers.read.*`

BFFH verwendet eine pfadähnliche Zeichenkette als Erlaubnisformat, getrennt durch einen `.` Punkt. So besteht zum Beispiel `this.is.a.permission` aus den Teilen `this`, `is`, `a` und `permission`. Bei der Anforderung von Berechtigungen, z. B. in Maschinen, muss immer eine genaue Berechtigung angegeben werden, also z. B. `test.write`. Bei der Erteilung von Berechtigungen, z. B. in Rollen, können entweder eine genaue Berechtigung angegeben oder die beiden Platzhalter `*` und `+` verwendet werden. Diese Wildcards verhalten sich ähnlich wie Regex- oder Bash-Wildcards:

- `*` gewährt alle Berechtigungen in diesem Teilbaum. So wird `perms.read.*` für jedes von passen:
  - `perms.read`
  - `perms.read.machineA`
  - `perms.read.machineB`
  - `perms.read.machineC.manage`
- `+` gewährt alle Berechtigungen unter des Wertes. So wird `perms.read.+*` für jedes von passen:
  - `perms.read.machineA`
  - `perms.read.machineB`
  - `perms.read.machineC.manage`
  - **aber nicht** `perms.read`

Wildcards sind wahrscheinlich am nützlichsten, um Maschinen zu gruppieren, z.B. 3D-Drucker und eine Bandsäge:

1. Write (schreiben) Berechtigungen
   - `machines.printers.write.prusa.sl1`
   - `machines.printers.write.prusa.i3`
   - `machines.printers.write.anycubic`
   - `machines.bandsaws.write.bandsaw1`
2. Manage (verwalten) Berechtigungen
   - `machines.printers.manage.prusa.sl1`
   - `machines.printers.manage.prusa.i3`

- `machines.printers.manage.anycubic`
- `machines.bandsaws.manage.bandsaw1`

3. Admin Berechtigungen
   - `machines.printers`
     - Für alle Drucker
   - `machines.bandsaws`
     - Für alle Bandsägen

Dann erteilen wir den Rollen die entsprechenden Rechte:

- Nutze beliebige 3D-Drucker:
  - `machines.printers.write.+`
- Erlaube nur die Nutzung "billiger" Drucker:
  - `machines.printers.write.anycubic.*`
  - `machines.printers.write.prusa.i3`
- Erlaube das Verwalten der Drucker:
  - `machines.printers.+`
- Erlaubte das Administrieren aller Drucker:
  - `machines.printers.*`

Auf diese Weise klappt es trotzdem mit der Aufteilung, wenn später ein weitere Anycubic Drucker gekauft wird:

- `machines.printers.write.anycubic.i3`
- `machines.printers.write.anycubic.megax`

Konfiguration von Maschinen

`machines`

Enthält eine Liste der definierten Maschinen. Die Maschinen haben verschiedene Wahrnehmungsebenen, mit denen interagiert werden kann:

- `disclose` (offenlegen): Benutzer kann die Maschine in der Maschinenliste sehen
- `read` (lesen): Der Benutzer kann Informationen über die Maschine und ihren Zustand lesen
- `write` (schreiben): Der Benutzer kann die Maschine benutzen
- `manage` (verwalten): Der Benutzer kann als Manager mit dem System interagieren (Prüfen, Freigeben, Transferieren)

Jede Maschine muss eine ID haben, um in anderen Teilen dieser Konfiguration oder über die API auf die Maschine verweisen zu können. Und jede Maschine muss einen Namen haben.

Optionale Informationen

Um weitere Informationen über die Maschine bereitzustellen, können diesen Beschreibungen hinzugefügt oder ein externer Wiki-Link bereitstellt werden. Beide Attribute sind nur optional und müssen nicht gesetzt werden.

Beispiel:

```
machines =
{
   machine123 =
   {
      name = "Testmachine",
      description = Some "A test machine",
      wiki = "https://someurl"

      disclose = "lab.test.read",
      read = "lab.test.read",
      write = "lab.test.write",
      manage = "lab.test.admin"
   }
}
```

“machine123” is in this case the “Machine-ID”

Konfiguration von Rollen (roles)

Die Rollen werden in der Datei `bffh.dhall` konfiguriert. Wenn die Datei `roles.toml` im Verzeichnis vorhanden ist, kann sie gelöscht werden und kann nicht zur Verwaltung von Rollen verwendet werden.

`roles`

Enthält die Liste der definierten Rollen. Rollen haben eine Liste von Berechtigungen und können vererbt werden. Die Berechtigung kann ein Platzhalter in der Berechtigungsliste sein.

Beispiel:

```
roles =
{
   testrole =
   {
      permissions = [ "lab.test.*" ]
   },
   somerole =
   {
      parents = [ "testparent" ],
      permissions = [ "lab.some.admin" ]
   },
   testparent =
   {
      permissions =
      [
         "lab.some.write",
         "lab.some.read",
         "lab.some.disclose"
      ]
   }
}
```

Konfiguration von Aktoren (actors)

actors

Enthält eine Liste von Aktoren. Aktoren werden durch ein Modul und einen oder mehrere Parameter definiert. Aktuell **von Haus aus unterstützte Aktoren** (ohne zusätzliche Plugins) sind:

Dummy Actor

Für Testzwecke kann ein interner Dummy-Initiator genutzt werden:

Der „Dummy“-Initiator versucht alle paar Sekunden, einen Rechner als den angegebenen Benutzer zu verwenden und zurückzugeben. Er ist gut geeignet, um das System zu testen, führt aber zu Spam im Log und ist daher standardmäßig deaktiviert.

```
actors = {
  Dummy_123 =
    {
      module = "Dummy",
      params = {=}
    }
  }
```

Shelly Actor

Dieser Aktor verbindet BFFH über einen MQTT-Server mit einem Shelly Gerät.

Der `topic` Parameter des Shelly muss auf das Shelly-spezifische MQTT-Topic gesetzt werden.

Anleitung zum Auffinden des Shelly Topic

Beispiel:

```
actors =
{
   Shelly_123 =
   {
      module = "Shelly",
      params =
      {
         topic = "shellyplug-s-123456"
      }
   }
}
```

„Shelly_123“ ist in diesem Fall die "Actor-ID".

Process Actor

Dieser Aktor ermöglicht es, eigene Prozesse (z.B. Python, Bash, Perl, Java ...) mit dem BFFH Server zu verbinden. Im Beispiel heißt unser Aktor `Bash_123`.

`cmd` = Pfad der ausführbaren Datei
`args` = Argumente der ausführbaren Datei

Beispiel:

```
actors =
{
   Bash_123 =
   {
      module = "Process",
      params =
         {
            cmd = "./examples/actor.sh",
            args = "your ad could be here"
         }
   }
}
```

Konkret nutzbare Aktoren-Beispiele finden sich hier.

actor_connections

Verbindet den Aktor mit einer Maschine. Eine Maschine kann mehrere Aktoren haben. Verwenden Sie die "Machine-ID" (`machine`) und "Actor-ID" (`actor`).

Beispiel:

```
actor_connections =
[
   { machine = "Testmachine", actor = "Shelly_123" },
   { machine = "Another", actor = "Bash_123" },
   { machine = "Yetmore", actor = "Bash_234" }
]
```

Konfiguration von Initiatoren (initiators)

Initiatoren werden fast genauso konfiguriert wie Aktoren.

initiators

Enthält eine Liste von Initiatoren. Initiatoren werden durch ein Modul und einen oder mehrere Parameter definiert. Die Liste kann bzw. **muss** leer sein, wenn keine Initiatoren verwendet werden:

```
initiators = {=}
```

Sonst kann die Liste einen oder mehrere `Initiator` mit ihrem Name (Im Beispiel `Initiator_123`) enthalten:

Dummy Initiator

Für Testzwecke kann ein interner Dummy-Initiator genutzt werden:

Der „Dummy“-Initiator versucht alle paar Sekunden, einen Rechner als den angegebenen Benutzer zu verwenden und zurückzugeben. Er ist gut geeignet, um das System zu testen, führt aber zu Spam im Log und ist daher standardmäßig deaktiviert.

```
initiators = {
  Initiator_123 =
    {
      module = "Dummy",
      params =
      {
        uid = "Testuser"
      }
    }
  }
```

Process Initiator

Dieser Initiator ermöglicht es, eigene Prozesse (z.B. Python, Bash, Perl, Java ...) mit dem BFFH Server zu verbinden. Im Beispiel heißt unser Initiator `Bash_567`.

`cmd` = Pfad der ausführbaren Datei
`args` = Argumente der ausführbaren Datei

Beispiel:

```
initiators =
{
  Bash_567 =
  {
    module = "Process",
    params =
      {
```

```
            cmd = "./examples/init.py",
            args = "your ad could be here"
        }
    }
}
```

Konkret nutzbare Initiatoren-Beispiele finden sich hier.

init_connections

Verknüpfung von Maschinen mit Initiatoren. Ähnlich wie bei Aktoren können einer Maschine mehrere Initiatoren zugewiesen werden, aber ein Initiator kann nur einer Maschine zugewiesen werden. Verwenden Sie für die Zuweisung die "Machine-ID" (`machine`) und "Initiator-ID" (`initiator`).

```
init_connections = [
    { machine = "Testmachine", initiator = "Initiator_123" }
],
```

FabFire

spacename

Der Name des Spaces (die offene Werkstatt, das FabLab, der HackerSpace, etc.) wird im URN-Schema `urn:fabaccess:lab:{spacename}` verwendet. Wird er nicht definiert, wird der Wert "generic" vergeben. Diese Angaben benötigen wir für QR-Codes von Maschinen oder für DESFire Karten zur Nutzung von FabFire.

instanceurl

Wird für eine allgemeine Space Info genutzt und als URN im Code genutzt: `urn:fabaccess:lab:{spacename}\x00{instanceurl}`. Dieser Wert wird aktuell nicht verwendet, muss jedoch ausgefüllt werden, damit die Konfiguration `bffh.dhall` valide ist!

Konfiguration per Config Generator

Siehe Einfache Konfiguration mit dem FabAccess Config Generator

Konfiguration exportieren

```
BASE="/opt/fabinfra/bffh/target/release"
CFG="$DATA/config/bffh.dhall"
$BASE/bffhd --verbose --config $CFG --dump-users /tmp/users.toml --force
```

Siehe auch Cheat Sheet - Wichtigste Befehle (Übersicht)

Version #29
Erstellt: 23 Oktober 2024 22:53:35 von Mario Voigt (Stadtfabrikanten e.V.)
Zuletzt aktualisiert: 11 Dezember 2024 11:01:45 von Mario Voigt (Stadtfabrikanten e.V.)