The unbelievable Machine Company

NERD STUFF #3

Freifunk Access Point via AWS steuern:

WLAN für alle, nur nicht jederzeit

Ein technologisch-gastronomischer Use Case

von Christian Hufgard, The unbelievable Machine Company

Vor einiger Zeit habe ich bei einem befreundeten Gastronomen ein frei nutzbares WLAN auf Freifunk-Basis installiert. Zur Zufriedenheit aller. Denn neulich erzählte er mir, dass Menschen mittlerweile lange nach Geschäftsschluss auf seiner Restaurant-Terrasse verweilen würden. Deshalb würde er jetzt schon mal den Stecker ziehen.
Das ist allerdings recht uncool und außerdem unpraktisch. Es musste also eine andere Lösung her.

Theoretisch lässt sich das Client-WLAN bei einem Freifunk-Access-Point (AP) mit einem Konsolen-Befehl schnell abschalten:

```
uci set wireless.client_radio0.disabled=1 && wifi
```

Will man auch noch das 5 GHz-Netz abschalten, sieht das so aus:

```
uci set wireless.client_radio0.disabled=1 && uci set
wireless.client_radio1.disabled=1 && wifi
```

Als Cron-Job verpackt, wird das immer wieder gerne eingesetzt. Der Haken eines Cron-Jobs ist allerdings, dass er zu einem fest definierten Zeitpunkt schaltet. Und das ist bei flexiblen Öffnungszeiten im Gastrobereich eher doof. Also braucht es eine flexible Lösung, die auch ein Nicht-Techie einfach nutzen kann. Kosten soll sie natürlich nichts.

Ziel ist also eine Möglichkeit, von einem Handy aus per App das WLAN ab- und anschalten zu können. Der Zugriff auf das System muss über eine Anmeldung abgesichert sein. Um den Aufwand möglichst gering zu halten, soll es eine Progressiv Web Application (PWA) sein. Als Backend kommen AWS-Komponenten zum Einsatz. So weit wie möglich wird beides via Amplify kontrolliert.

Umsetzung: Lambda & Jumphost

Die Logik zur Zustandsabfrage und zum Switchen des Access-Point-Zustands wird ein Lambda übernehmen, das nach jedem Befehl den aktuellen Status des WLANs zurückgibt. Da es auch einen SSH-Client für Node.js gibt, habe ich mich für eine Lösung auf Node.js-Basis entschieden. Mit dem Wrapper node-ssh steht eine sehr einfache API zur Verfügung.

```
return ssh.connect({
  port: PORT,
  host: HOST,
  username: USER,
  privateKey: ssh_key
})
.then(function() {
  console.log(„Logged in to jumphost.“);
  return ssh.execCommand(command).then(function(result) {
    console.log(„Executed command at target host: „+command);
    const wifiEnabled = {‚wifiEnabled‘: result.stdout.indexOf(„‘0‘“) > 0};
    console.log(„Wifi state for target 1: „, wifiEnabled);
    return {
      statusCode: 200,
      body: JSON.stringify(wifiEnabled),
        „headers“: {
          „Access-Control-Allow-Origin“: „*“
        }
      };
    });
});
```

Wie immer gibt es aber einen Haken: Freifunk-Access-Points sind über eine öffentliche IPv6-Adresse erreichbar – aber Lambdas unterstützen nur IPv4. Also muss ein Jump-host her. EC2-Instanzen können IPv6, und da praktisch keine Ressourcen in Form von CPU oder RAM benötigt werden, reicht eine kostenfreie t2.micro-Instanz absolut aus.

Der SSH-Schlüssel für den Login wird im Parameter Store als verschlüsselter String hinterlegt, so dass er weder im Lambda-Quellcode noch als System-Variable hinterlegt werden muss.

Damit das Lambda auf unserem Jumphost keinen Unfug anstellen kann – Paranoia ftw! –, wird für diesen Schlüssel ausschließlich der Aufruf von ssh per authorized_keys erlaubt:

```
command="ssh -oStrictHostKeyChecking=no -t root@$SSH_ORIGINAL_COM-
MAND",no-port-forwarding,no-X11-forwarding,no-agent-forwarding ssh-rsa
AAAAB3NzaC1yc2E...8Ja5Ef24KID Lambda private key
```

(Aufmerksame Leser werden oStrictHostKeyChecking=no bemerkt haben – soviel zur Paranoia ;) Aber so erspart man sich, nach einem Neuanlegen des Hosts von Hand einen Login auf dem Zielhost auszuführen und den Key zu akzeptieren.

Leider unterstützt node-ssh kein agent-forwarding, weshalb auf dem Jumphost noch ein auf dem Zielhost bekannter privater SSH-Key hinterlegt werden muss. Irgendwas ist halt immer.

Alternativ könnte man auch rinet.d zur Portweiterleitung verwenden, wäre dann aber deutlich unflexibler, was das Einfügen neuer Access Points angeht.
Wie es der Zufall will, sollte dann auch noch ein zweiter Access Point ins Setup integriert werden – entsprechend wurde auch der JavaScript-Code angepasst.

Jetzt muss nur noch das abgesetzte Kommando so erweitert werden, dass es vom Jumphost zum Zielhost springt und dort den eigentlichen Befehl ausführt.

Der vollständige Code ist als TXT-Datei verfügbar.

Damit lässt sich über die Test-Funktion des Lambdas das WLAN ab- und auch wieder anschalten. Jetzt fehlen nur noch die App und ihre Anbindung ans Backend.

Auftritt Amplify.

Frontend

Amplify ist ein weiteres Tool aus der AWS-Welt, das manches einfacher macht. Aber es hat auch seine Grenzen, an die man leider sehr schnell stößt.

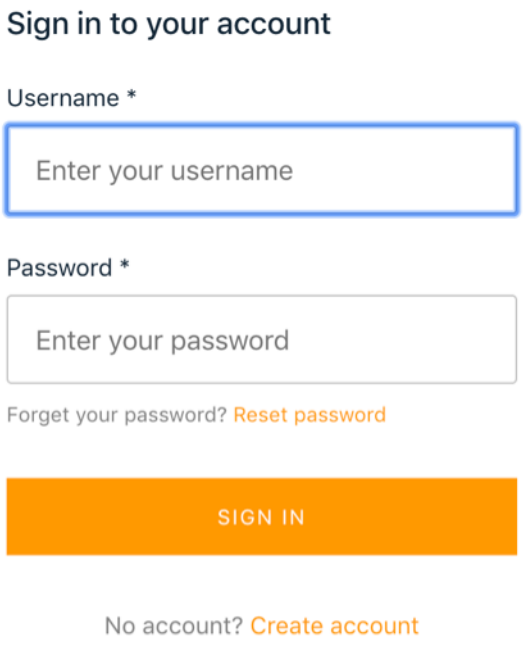
Mithilfe von Amplify wird für den Client ein React-Projekt mit Hosting und Authentication aufgesetzt. Damit das Ergebnis nachher als Progressive Web App (PWA) mit einem hübschen Icon auf dem Handy-Desktop abgelegt werden kann, wird im index.js die Registrierung vom serviceWorker aktiviert und im manifest.json ein Satz passender Icons abgelegt. Die Caching-Einstellungen bleiben einfach so wie sie sind. (Für eine ernsthafte App muss natürlich anders vorgegangen werden!)

Viel muss die App nicht machen, folglich hat sie auch nicht viel Code. Leider verlasse ich sehr schnell den Standard-Pfad von Amplify. Da ich keine 0815-Rest-CRUD-API baue, müssen sowohl die API als auch der Identity-Pool von Hand angelegt und konfiguriert werden.

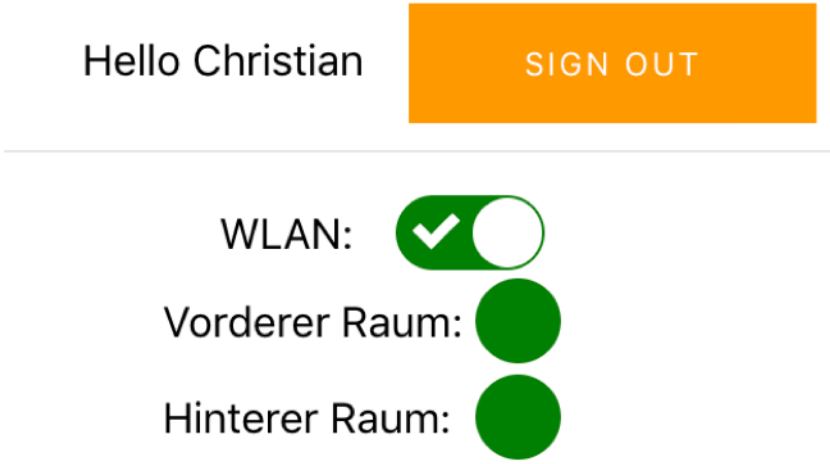
Der entsprechende Code ist als TXT-Datei verfügbar.

Das in App.css vorhandene Stylesheet wird minimal um einige Anweisungen erweitert. Auch dieser Code ist als TXT-Datei verfügbar.

Die Anmelde-UI ist eine fertige React-Komponente:



Die für den Anwender relevante Seite ist, wenig überraschend, auch recht überschaubar:



Leider ließ sich mit der verwendeten Amplify-Version Cloudfront nicht mehr nachträglich aktivieren, so dass auch hier Hand angelegt werden musste. Cloudfront wird benötigt, damit SSL-verschlüsselt auf die Webseite zugegriffen werden kann. Das wiederum ist notwendig, da PWAs zwingend das Laden von Ressourcen via HTTPS voraussetzen.

Fazit

Durch den Einsatz von AWS-Tools und -Komponenten konnte sehr schnell eine einfache Lösung umgesetzt werden, die als App auf dem Handy installiert werden kann. Da es sich um eine PWA handelt, ist das sowohl auf Android- als auch auf iOS-Geräten einfach möglich. Bis auf den Jumphost kommen nur verwaltete AWS-Komponenten zum Einsatz, so dass die Wartungsarbeiten auf ein Minimum reduziert sind. Sobald Lambdas mit der Außenwelt über IPv6 kommunizieren können, kann der Jumphost natürlich entfernt werden.

Als einziger Kostenpunkt fällt die EC2-Instanz an. Wer allerdings bereits einen Server mit SSH-Zugang irgendwo im Netz betreibt, kann auch diesen als Jumphost einsetzen.

Autor

Christian Hufgard ist Data Architect in Applications am *um Standort Frankfurt. In seiner Freizeit ist er 1. Vorsitzender eines Freifunk-Vereins. Sein bisher größtes Freifunk-Projekt bestand aus einer Installation auf dem „Hessentag“, dem größten und ältesten Landesfest Deutschlands. In der Spitze waren über 900 Nutzer in das Netzwerk eingeloggt.

Kontakt

The unbelievable Machine Company GmbH
Philipp Schlüter (Marketing & Sales)
Grolmanstr. 40
D-10623 Berlin
+49 (0) 30 88926560

whitepaper@unbelievable-machine.com
www.unbelievable-machine.com