Lab 1 - Partie 2

# Azure App Service : WebApp

Dans le lab précédent, nous avons vu comment créer une « function App » qui sert comme API pour simuler l’envoi de messages à notre IotHub.

Dans ce lab, je vous propose de déployer le site web de simulation des lampadaires (les fameux 5 lampadaires).

3 étapes nous attendent dans ce lab :

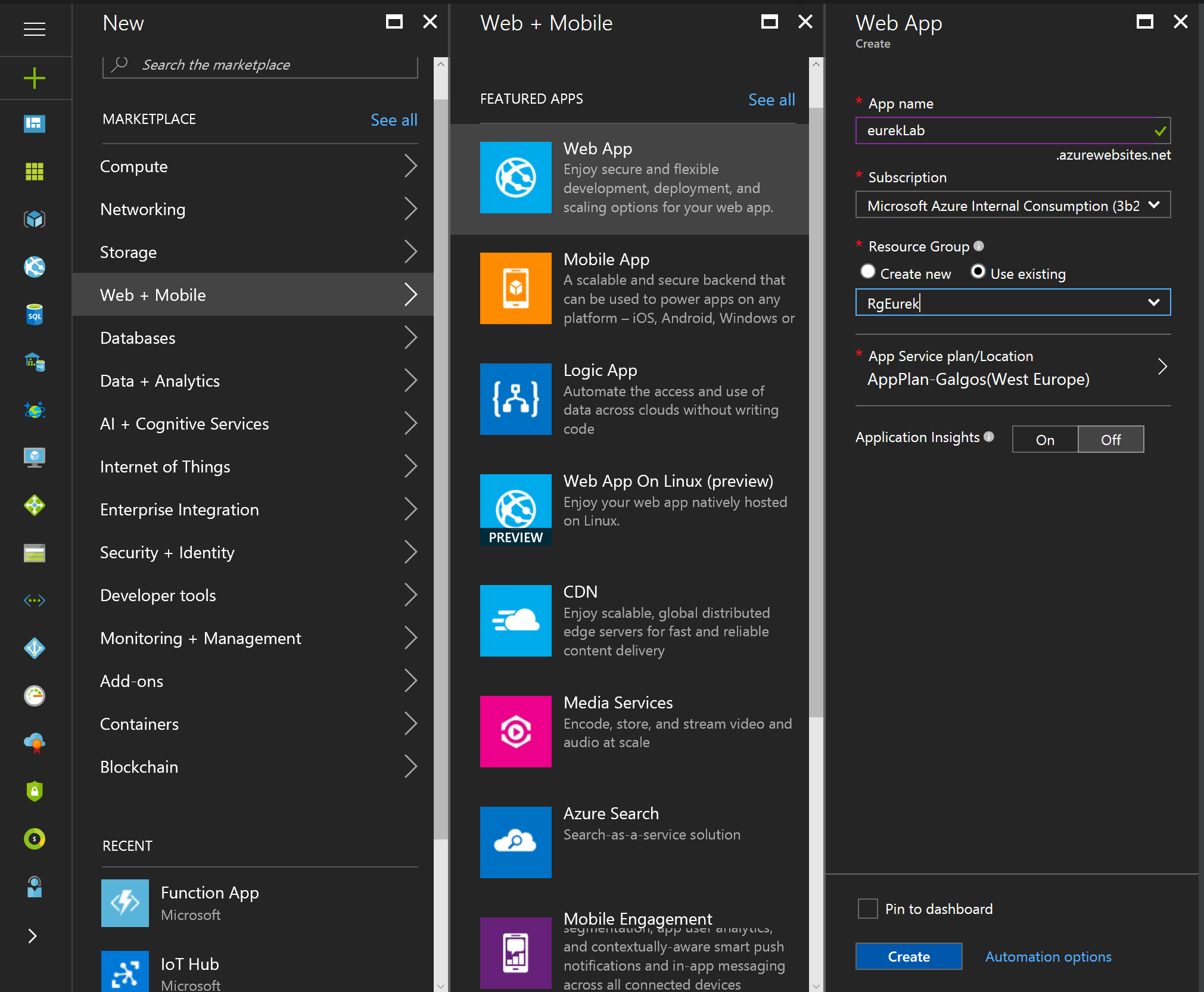
* Etape 1 : Créer une WebApp dans Azure
* Etape 2 : Modifier le code javascript de notre site web.
* Etape 3 : Déployer avec Visual Studio.

### Création de la WebApp dans Azure

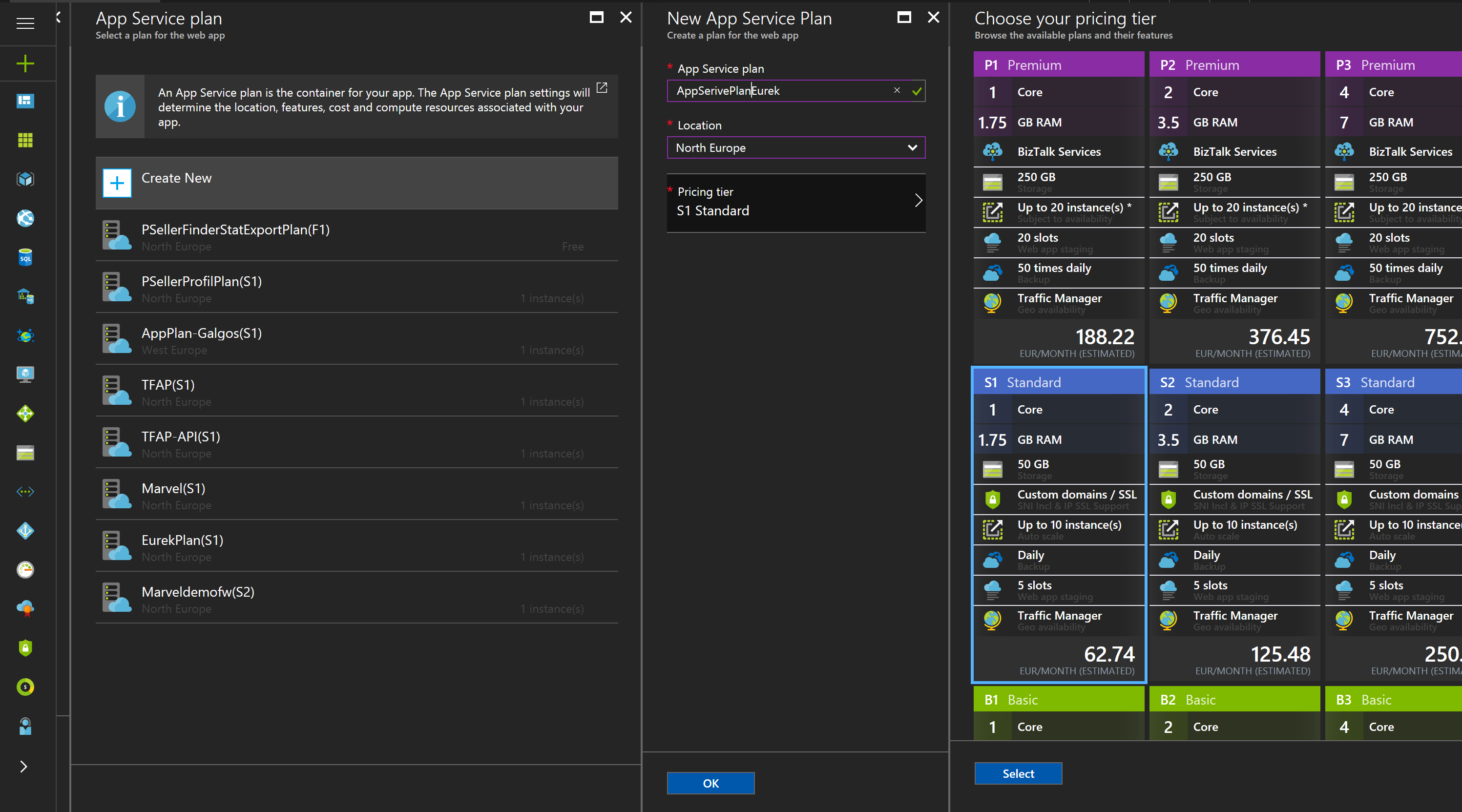
Dans le portail Azure

***WARNING : Tous les noms de services et key utilisés dans les Lab (Lab1 – xxx) doivent être modifié pour votre propre usage.***

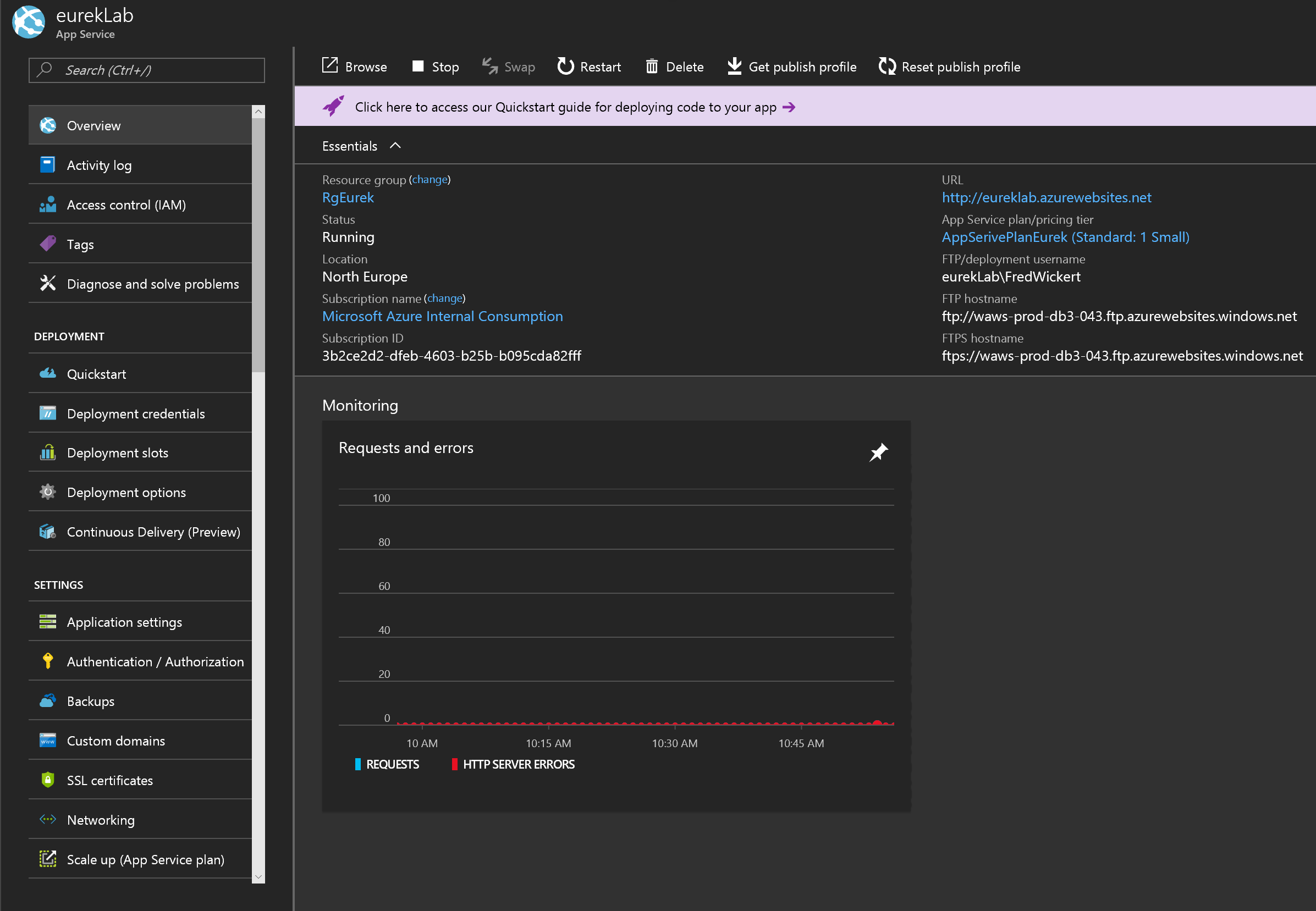
* Nom du service : eurekLab
* Ressource Group => « User existing » => RgEurek
* App Service Plan/Location => Voir image suivante



* Nom de l’App Service Plan : AppServicePlanEurek
* Location : NorthEurope
* Pricing Tier : S1 (vous pouvez choisir un autre plan, même gratuit pour ce site).



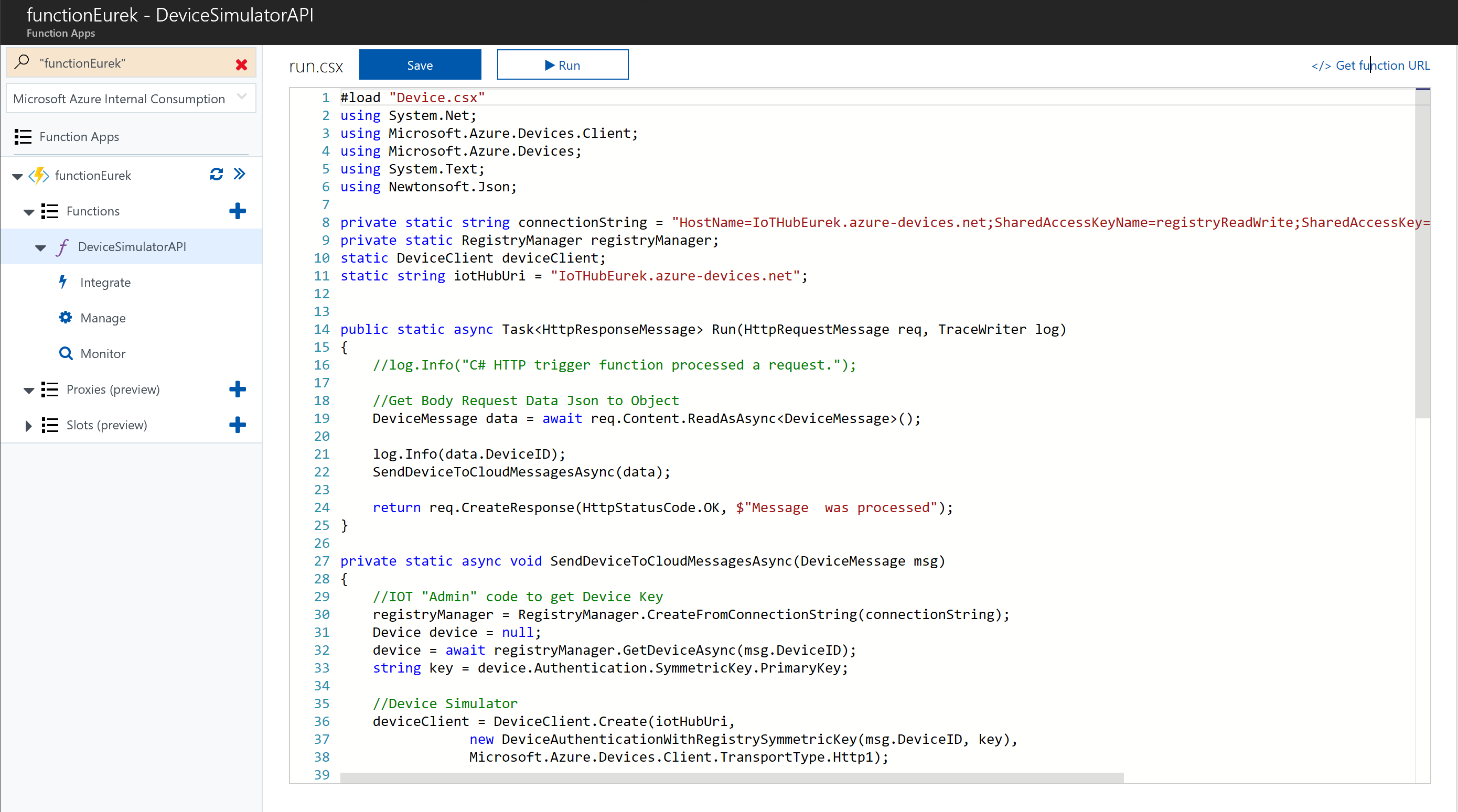
Attendre moins d’1 mn que le service soit provisionné.

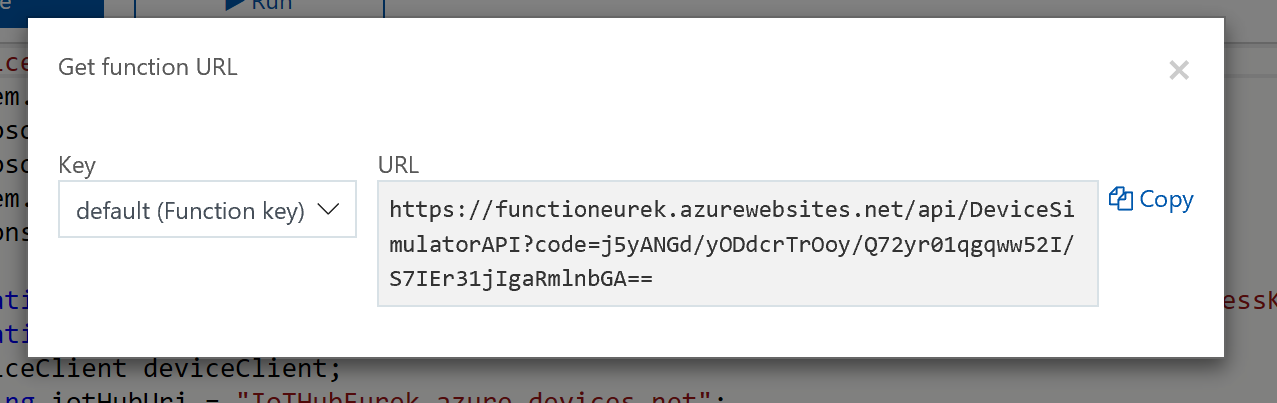


### Modification du code JavaScript

Nous allons dans un 1er temps récupérer le endpoint de notre API (notre function).

Dans le portail Azure, dans votre function.





Copiez cette url.

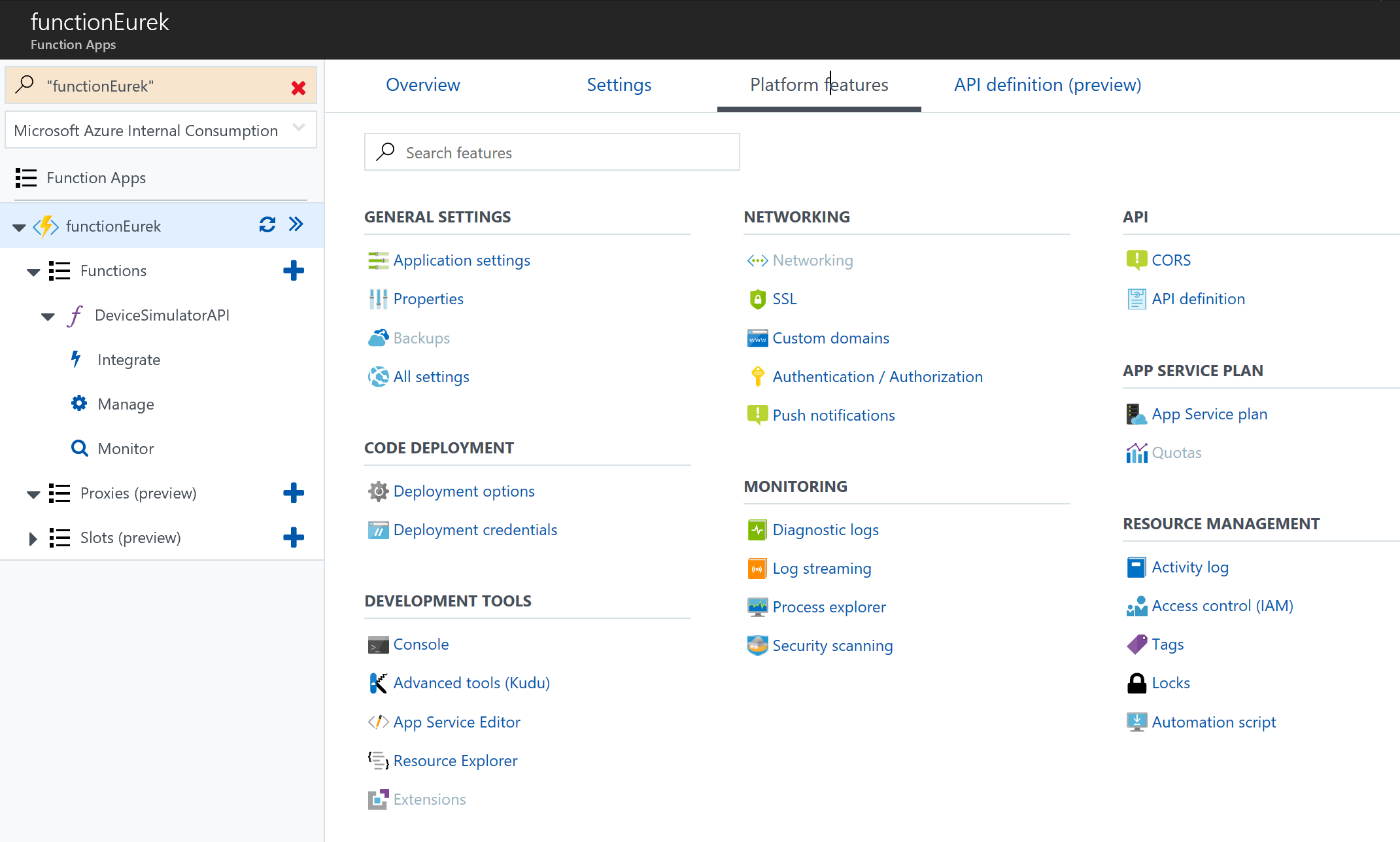
**https://functioneurek.azurewebsites.net/api/DeviceSimulatorAPI?code=j5yANGd/yODdcrTrOoy/Q72yr01qgqww52I/S7IEr31jIgaRmlnbGA==**

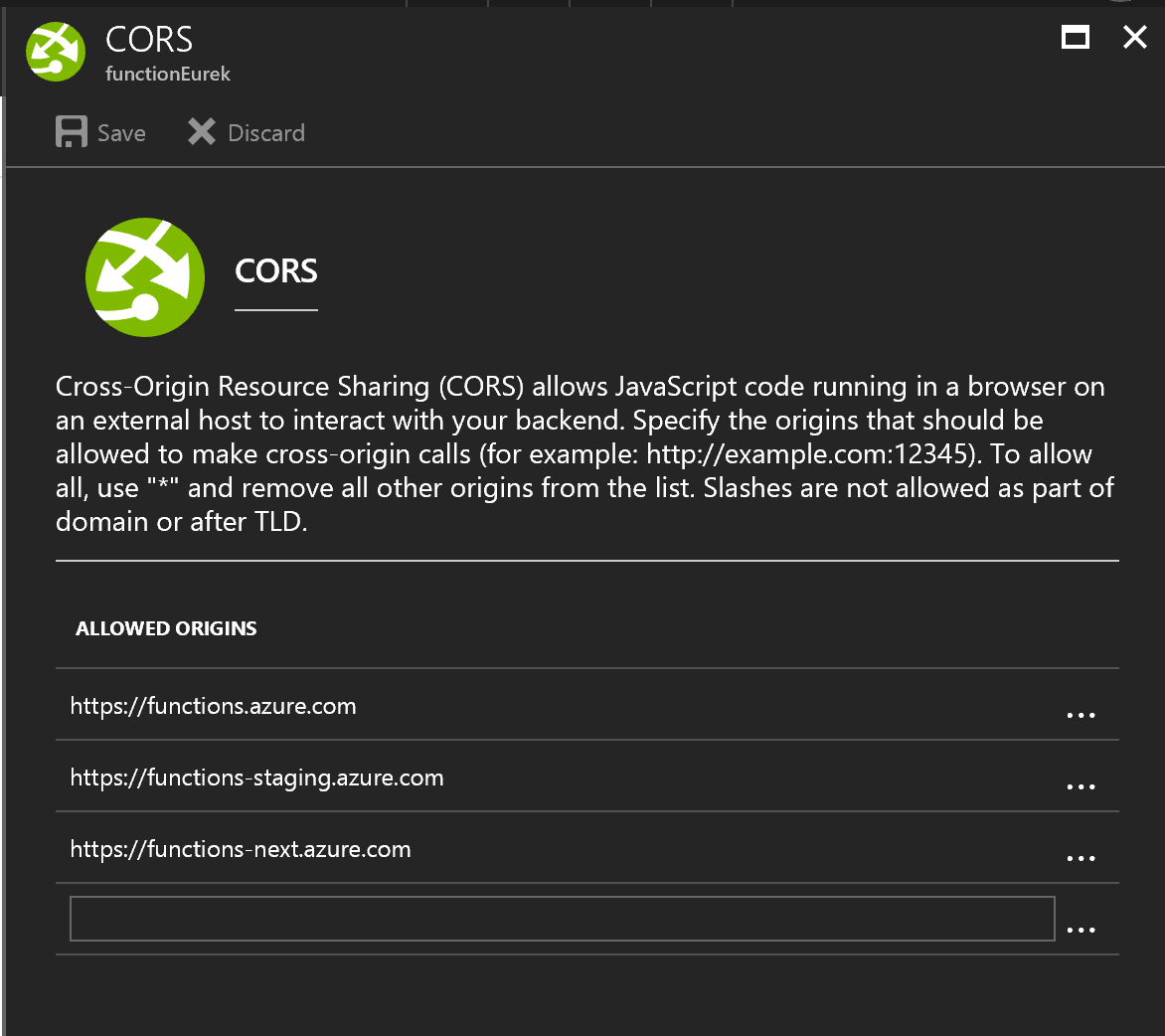
Restons quelques minutes sur notre API.

Nous sommes sur le point d’accéder à une API en faisant un appel, depuis notre site, via un autre domaine que celui utilisé par notre API.

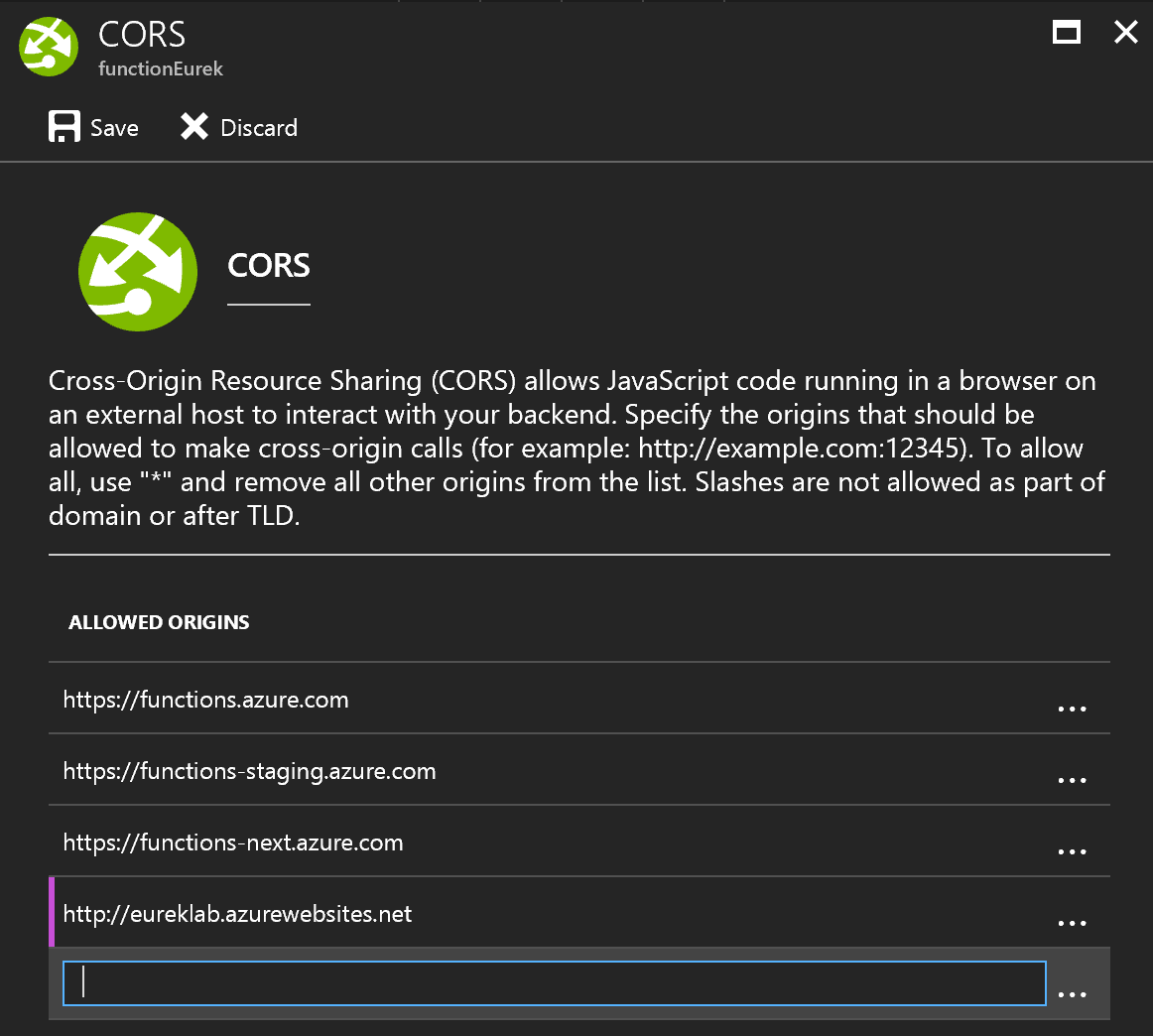
**Domaine API : functioneurek.azurewebsites.net  
Domaine du site : eureklab.azurewebsites.net**

Cela ne peut fonctionner, parce que par défaut notre API n’accepte que des appels de certains domaines.



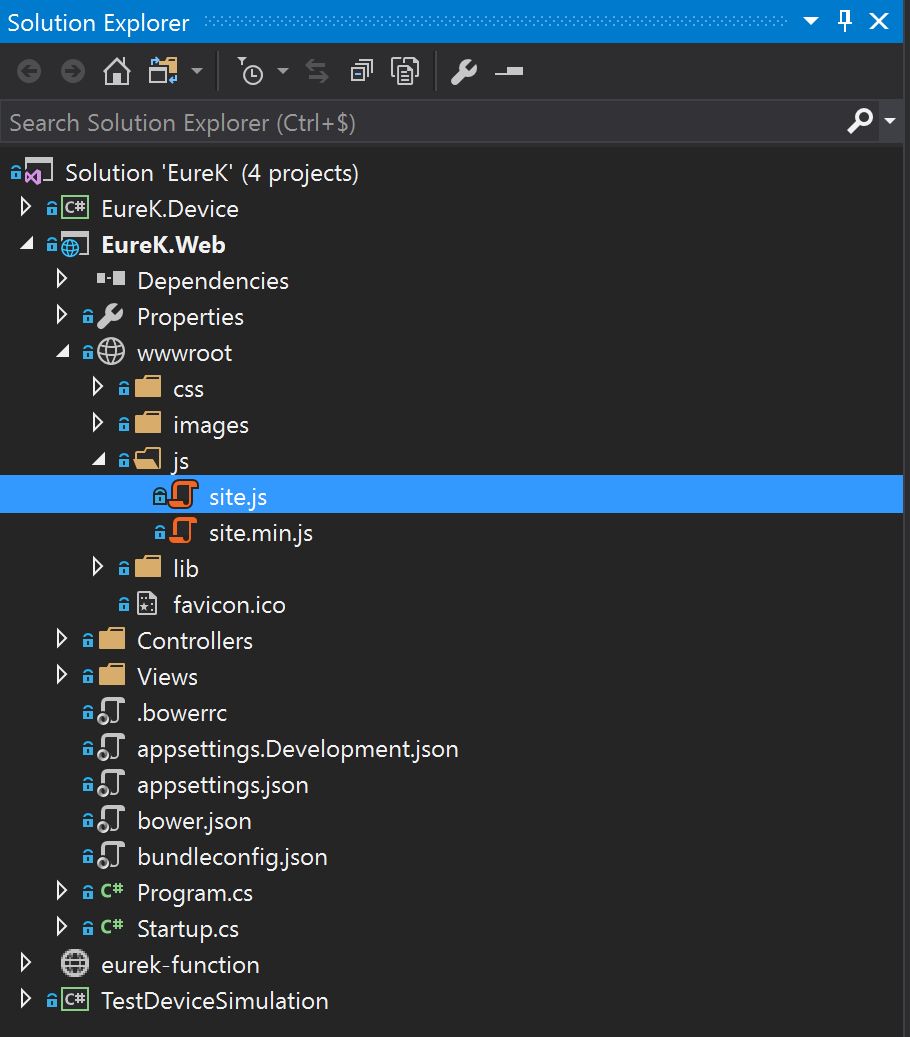


Rajoutons notre site web :



Parfait. Nous sommes prêt pour la suite.

Dans Visual Studio.



Tout en bas du code JavaScript du fichier site.js.

|  |
| --- |
| $.ajax({  url: "https://<Your function>.azurewebsites.net/api/DeviceSimulatorAPI?code=<Your Function Code>",  type:"POST",  contentType: "application/json",  dataType:"json",  data: msg  }); |

Modifier comme si dessous (coller l’url copiée) :

|  |
| --- |
| $.ajax({  url: "https://functioneurek.azurewebsites.net/api/DeviceSimulatorAPI?code=j5yANGd/yODdcrTrOoy/Q72yr01qgqww52I/S7IEr31jIgaRmlnbGA==",  type:"POST",  contentType: "application/json",  dataType:"json",  data: msg  }); |

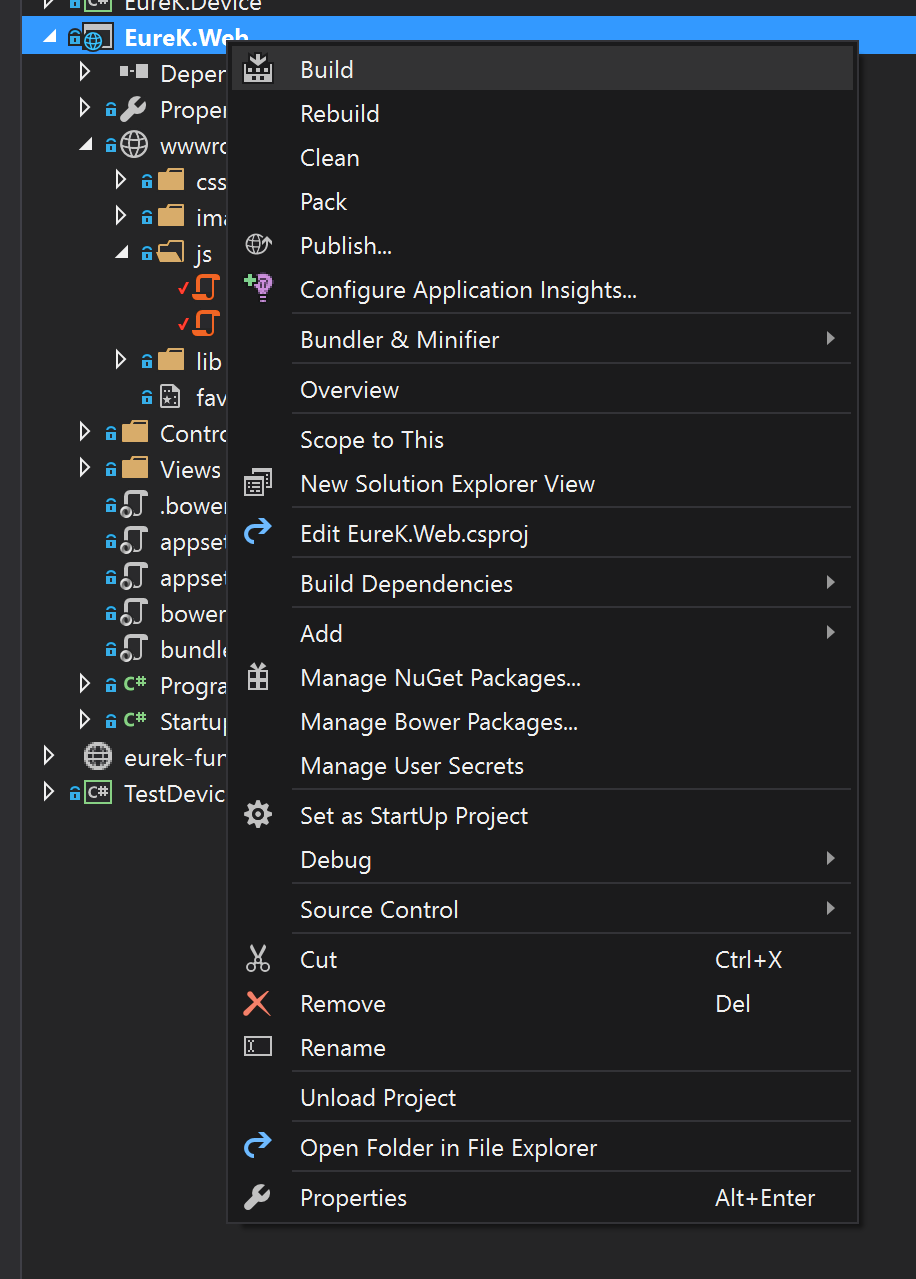
### Déploiement avec Visual Studio

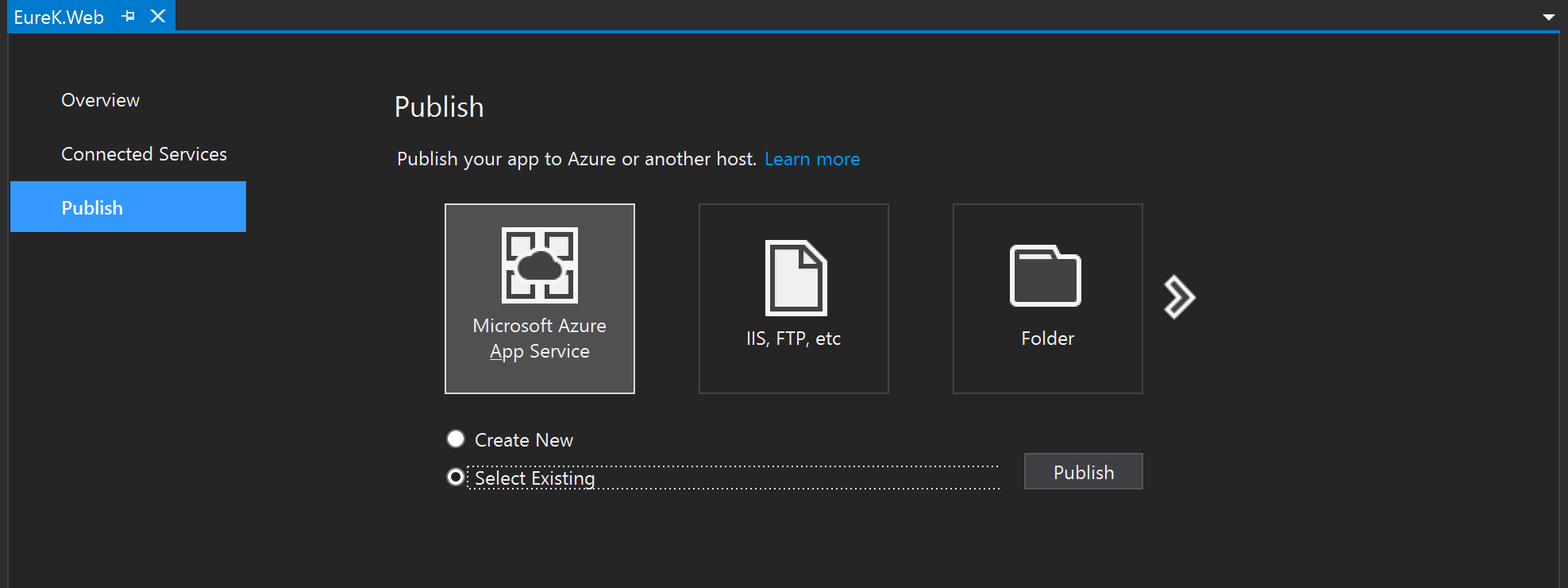
Pour déployer votre site web dans la WebApp il existe plusieurs possibilités : $

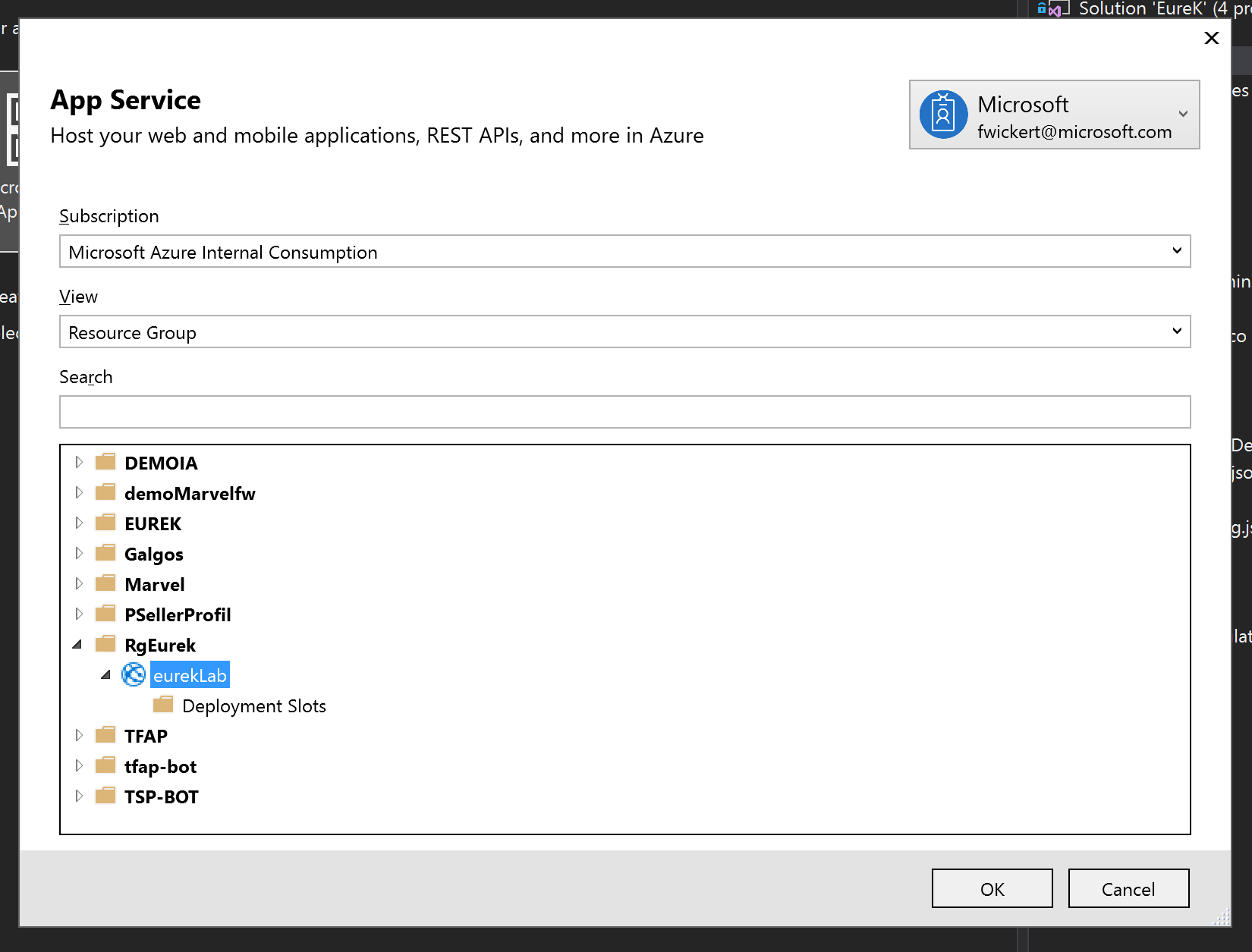
* FTP/FTPS
* WebDeploy
* Kudu : <https://blogs.msdn.microsoft.com/benjaminperkins/2014/03/24/using-kudu-with-windows-azure-web-sites>
* Déploiement Continue avec GitHub, VSTS, BitBucket, etc…
* Publish de Visual Studio,
* …

Pour simplifier, nous allons donc utiliser Visual Studio.

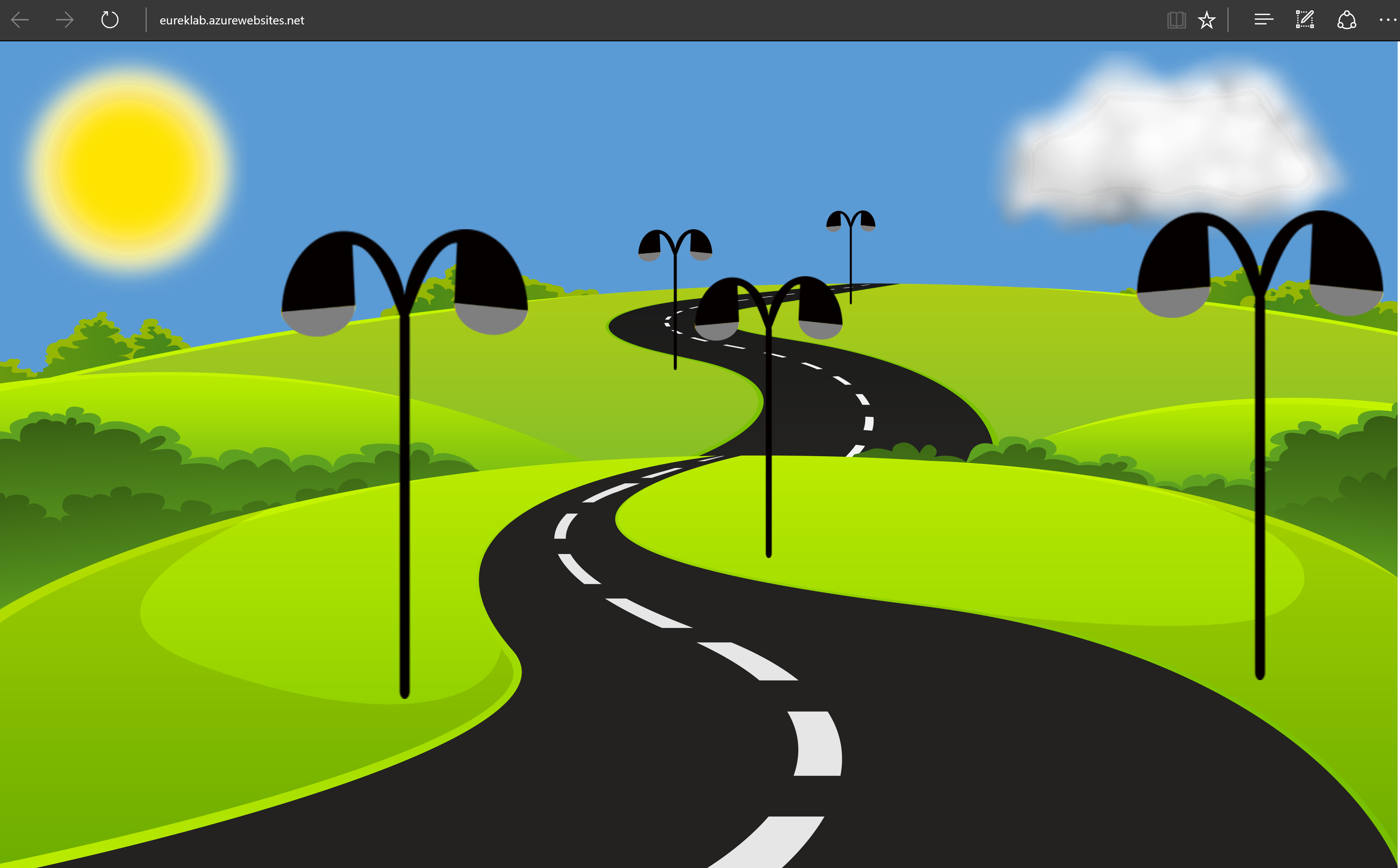
Vous pouvez faire un click-droit sur le projet « Eurek.Web » et choisir « Publish »







Au bout de quelques secondes, le site se lance tout seul dans votre navigateur.



### Test

Pour tester, si tout fonctionne correctement, nous allons utiliser ce site web et regarder si les messages arrivent bien sur l’IoTHub.

#### Commençons à faire envoyer des messages à notre IoTHub.

Sur le site web, cliquez sur le **soleil** (le nuage doit venir couvrir le soleil). Le lux change et les lampadaires envoient chacun un premier message.

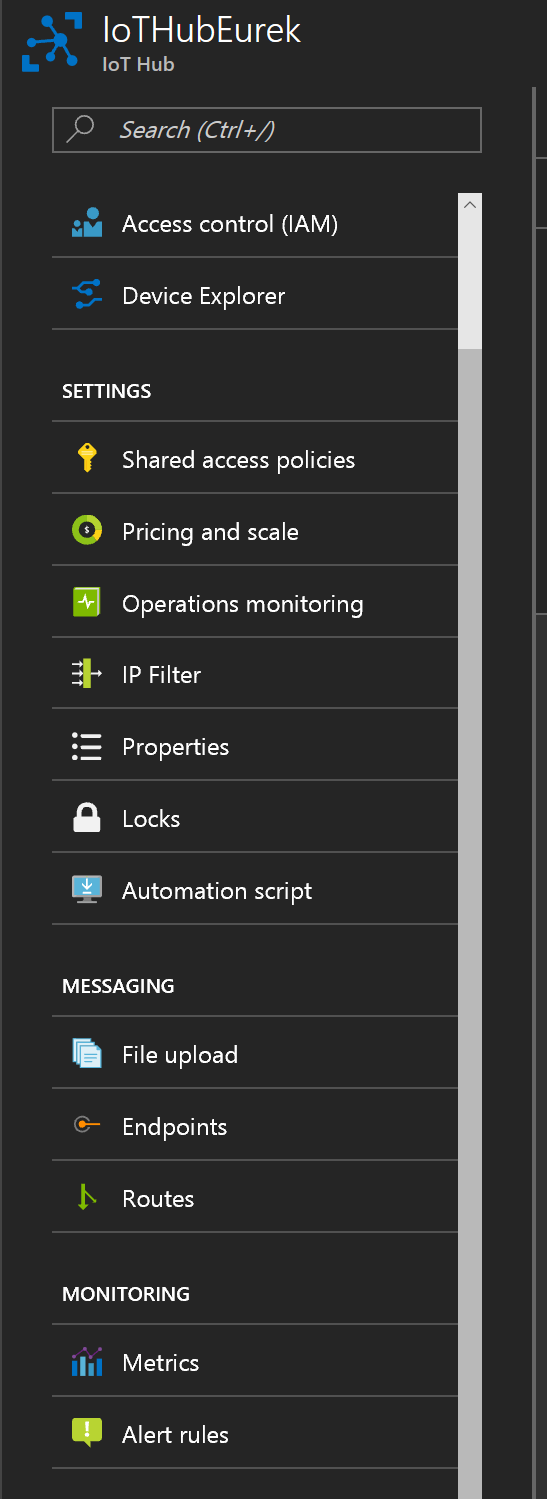
Cliquez ensuite sur le **nuage**, la nuit tombe est les lampadaires envoient à nouveau leurs états.

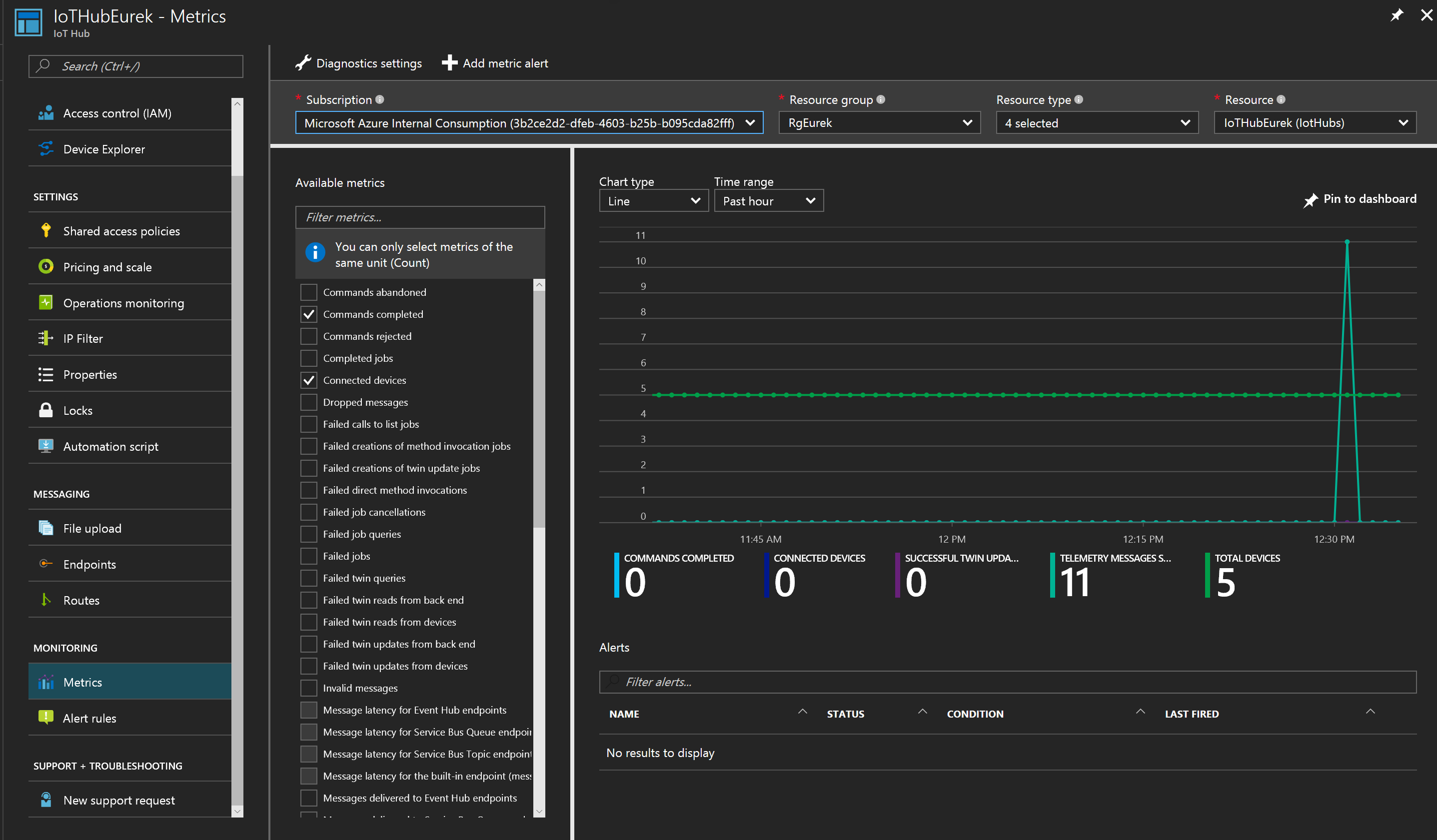
Cliquez ensuite sur la **lune**, la voiture vient percuter un lampadaire. Ce dernier est le seul à renvoyer un message à notre IoTHub.

En tout, la séquence complète envoie 11 messages à notre IoTHub.

Pour vérifier si les messages sont bien arrivés dans l’IoTHub.

Dans le portail Azure, dans votre IoTHub





5 devices connectés et 11 messages. Cela parait cohérent par rapport à notre scénario.