Retour à Proc#144 - Installation et mise à jour des serveurs et des applications	
Document technique	Rattaché à Proc#144. Non soumis à approbation (Proc#163) : tenu à jour par l'équipe de développement.

Document technique — API .NET (Stimulsoft.PDF.Forms + EF Core SQLite) sur Windows Server + IIS

Version enrichie suite à un déploiement réel (serveur app-logeas-ovh, site “Formulaire”) : chaque piège rencontré en production a été intégré à ce guide, avec sa cause exacte et sa correction. Suivre ce guide dans l'ordre, en particulier l'étape 2 (souvent sous-estimée), évite la quasi-totalité des problèmes rencontrés lors du premier déploiement.

Projet : ASP.NET Core Web API, .NET 8, Stimulsoft.PDF.Forms (formulaires PDF, en paire avec le package Angular `stimulsoft-forms` côté client) et Microsoft.EntityFrameworkCore.Sqlite. API déployée sur un domaine séparé de l'Angular (CORS nécessaire).

---

1. Objectif et architecture

Ce guide déploie une API ASP.NET Core (.NET 8) sur Windows Server via IIS, en tant que backend séparé pour une application Angular. IIS héberge l'API **in-process** via le module ASP.NET Core (ANCM) : pas de process séparé à surveiller, pas de reverse proxy manuel — IIS démarre/redémarre l'API automatiquement.

Schéma : Angular (autre domaine) → appel HTTPS avec en-tête CORS → IIS (port 443, certificat SSL, liaison par nom d'hôte/SNI) → module ANCM → API .NET (Stimulsoft.PDF.Forms + EF Core) → fichier SQLite sur le serveur, dans un dossier persistant distinct du dossier de déploiement.

Ce qui a posé problème lors du premier déploiement, dans l'ordre où ça s'est manifesté (pour comprendre pourquoi certaines étapes ci-dessous insistent autant sur la vérification) :

1. Le rôle IIS et le site étaient en place, mais le **.NET Hosting Bundle n'avait en réalité jamais été installé correctement** → erreur 500.19 (config invalide, code 0x8007000d) parce qu'IIS ne reconnaissait pas l'élément ``<aspNetCore>``.
2. Après installation du Hosting Bundle, IIS trouvait le module mais pas le runtime → erreur 500.31 (“Failed to load ASP.NET Core runtime”).
3. Une fois le runtime confirmé présent, le démarrage plantait quand même avec un code générique **0x8000ffff** → cause : le **Visual C++ Redistributable 2015-2022 (x64)** n'était pas installé (dépendance native d'ANCM, censée être installée par le Hosting Bundle mais pas toujours effective sans réparation + redémarrage complet du serveur).
4. Une fois ANCM capable de démarrer le process, l'app restait bloquée en **503 “Application Shutting Down”** en boucle → cause : droits NTFS insuffisants pour le compte du pool IIS sur le dossier contenant la base SQLite (l'app fonctionnait très bien lancée manuellement sous un compte Administrateur, ce qui a permis d'isoler le problème aux droits, pas au code).

5. Une fois les droits corrigés, des tests ``curl`` intermittents donnaient encore des erreurs de négociation TLS — cause : un problème de méthodologie de test (SNI), pas du serveur (voir étape 12).

2. Prérequis — à vérifier avant tout déploiement

- Accès administrateur au Windows Server.
- Le rôle « Serveur Web (IIS) » installé.
- Le SDK .NET 8 installé sur le poste/pipeline de build.
- Un nom de domaine dédié à l'API, distinct de celui de l'Angular (ex. ``api.mondomaine.fr``).
- Un certificat SSL couvrant ce domaine (souvent un wildcard déjà utilisé par d'autres sites du même serveur).
- Le fichier ou la clé de licence Stimulsoft.

> **Nouveau, issu de l'expérience réelle :** avant même de commencer, vérifiez que les deux dépendances suivantes sont déjà présentes sur le serveur cible si des applications .NET Core y tournent déjà — sinon, prévoyez leur installation dès l'étape 4 :
> - Le **.NET Hosting Bundle** (runtime + module IIS), PAS seulement le SDK ou le runtime “autonome”.
> - Le **Visual C++ Redistributable 2015-2022 (x64)** — dépendance native souvent oubliée, cause de l'erreur la plus difficile à diagnostiquer de tout ce guide (voir étape 4 et l'annexe B).

3. Étape 1 — Installer le rôle IIS

```
Install-WindowsFeature -Name Web-Server -IncludeManagementTools
```

4. Étape 2 — Installer le .NET Hosting Bundle ET le VC++ Redistributable, puis vérifier

C'est l'étape la plus critique de tout le guide. Ne passez pas à la suite sans avoir validé chaque commande de vérification ci-dessous — c'est exactement ce qui a manqué lors du premier déploiement, et qui a coûté la majorité du temps de dépannage.

4.1 Installer le Hosting Bundle

Téléchargez et exécutez le **« ASP.NET Core Runtime 8.0.x – Windows Hosting Bundle »** depuis https://dotnet.microsoft.com/download/dotnet/8.0 — bien le lien **“Hosting Bundle”**, pas “Runtime” seul (qui n'installe pas le module IIS).

4.2 Installer/réparer le Visual C++ Redistributable

Le Hosting Bundle est censé l'installer automatiquement, mais ce n'est pas toujours effectif. Installez-

le (ou réparez-le s'il est déjà présent) explicitement :

```
Invoke-WebRequest -Uri "https://aka.ms/vs/17/release/vc_redist.x64.exe" -
OutFile "$env:TEMP\vc_redist.x64.exe"
Start-Process "$env:TEMP\vc_redist.x64.exe" -ArgumentList "/install /quiet
/norestart" -Wait
```

4.3 Redémarrer complètement le serveur (pas seulement IIS)

```
Restart-Computer
```

Un simple ``iisreset`` ou ``net stop was`` / ``net start w3svc`` ne suffit pas toujours : le chargement de bibliothèques natives (hostfxr, VC++ runtime) peut nécessiter un redémarrage complet pour libérer des fichiers verrouillés ou finaliser l'installation.

4.4 Vérifier — ne pas sauter cette sous-étape

Après le redémarrage, dans une **nouvelle** session PowerShell (le PATH n'est chargé qu'à l'ouverture de session) :

```
# Le module IIS doit exister physiquement
Test-Path "C:\Program Files\IIS\Asp.Net Core Module\V2\aspnetcorev2.dll"
# doit renvoyer True

# Le runtime .NET doit être listé
dotnet --list-runtimes
# doit afficher une ligne "Microsoft.AspNetCore.App 8.x.x" ET une ligne
"Microsoft.NETCore.App 8.x.x"
```

Si l'une de ces deux vérifications échoue, ne continuez pas — reprenez l'installation avant d'aller plus loin. C'est ce contrôle, sauté lors du premier déploiement, qui aurait évité une bonne partie du dépannage.

5. Étape 3 — Configurer CORS dans le code

Si votre code lit les origines autorisées depuis la configuration (pattern recommandé, déjà en place dans ce projet via ``Cors:Origins``), il n'y a rien à changer dans le code — seulement dans ``appsettings.Production.json`` (étape 8). Sinon, dans ``Program.cs`` :

```
builder.Services.AddCors(options =>
{
    options.AddPolicy("logeas-web", policy =>
    {
policy.WithOrigins(builder.Configuration.GetSection("Cors:Origins").Get<stri
ng[]>() ?? [])
              .AllowAnyHeader()
```

```
                .AllowAnyMethod();
    });
});
// ...
app.UseCors("logeas-web");
```

> **Rappel important :** l'origine à autoriser est celle du site qui **appelle** l'API (l'Angular, ex. ``interface.logeas-web.fr``), jamais celle de l'API elle-même. CORS ne protège pas contre un appel direct (curl/Postman) — il ne concerne que les appels JavaScript faits depuis un navigateur, depuis une autre origine.

6. Étape 4 — Base SQLite : emplacement persistant et droits

Deuxième étape critique. Si le chemin de la base pointe (par défaut ou par erreur de configuration) vers un dossier à l'intérieur du dossier de déploiement, deux problèmes surviennent : la base est effacée à chaque mise à jour (``robocopy /MIR``), et le compte du pool IIS peut ne pas avoir les droits d'écriture nécessaires.

6.1 Choisir un emplacement en dehors du dossier de déploiement

Un dossier **frère** du dossier de déploiement (ex. ``C:\RemoteApp\Formulaire-Data`` à côté de ``C:\RemoteApp\Formulaire``) convient parfaitement : un ``robocopy /MIR`` ciblant uniquement ``Formulaire`` ne le touche pas. Un chemin totalement séparé (ex. ``D:\Data\...``) fonctionne tout aussi bien.

Si le code suit le pattern de ce projet (``Storage:DatabasePath`` lu depuis la configuration, avec ``Directory.CreateDirectory`` automatique), il suffit de renseigner ce chemin dans ``appsettings.Production.json`` (étape 8) — aucune modification de code nécessaire.

6.2 Donner les droits d'écriture au compte du pool IIS

Identifiez d'abord l'identité du pool (souvent ``ApplicationPoolIdentity`` par défaut, ce qui correspond au compte virtuel ``IIS AppPool\<NomDuPool>`` et qui est automatiquement membre du groupe local ``IIS_IUSRS``) :

```
Import-Module WebAdministration
Get-ItemProperty "IIS:\AppPools\<NomDuPool>" -Name processModel.identityType
```

Puis donnez les droits sur le dossier de données (le groupe ``IIS_IUSRS`` couvre le cas ``ApplicationPoolIdentity`` par défaut ; si l'identité est un compte de service spécifique, remplacez ``IIS_IUSRS`` par ce compte) :

```
icacls "C:\RemoteApp\Formulaire-Data" /grant "IIS_IUSRS:(OI)(CI)M"
```

> **Comment on a isolé ce problème en pratique :** l'application plantait sous IIS (503 en boucle)

> mais fonctionnait parfaitement lancée manuellement avec ``dotnet .\VotreApp.dll`` depuis une session Administrateur — la différence entre les deux étant précisément les droits du compte utilisé. Lancer l'app manuellement est le test le plus rapide pour distinguer un problème de code d'un problème de droits/environnement IIS (voir étape 11).

7. Étape 5 — Publier l'application

```
dotnet publish -c Release -o ./publish
```

Vérifiez que ``appsettings.Production.json`` et le fichier de licence Stimulsoft (si utilisé par fichier) sont bien inclus dans ce dossier.

8. Étape 6 — Copier les fichiers vers le serveur

```
robocopy publish C:\RemoteApp\Formulaire /MIR
```

9. Étape 7 — Créer le site dans IIS Manager

1. IIS Manager → « Sites » → Ajouter un site web.
2. Nom du site : ex. ``Formulaire``.
3. Chemin d'accès physique : ``C:\RemoteApp\Formulaire\``.
4. Liaison : https, port 443, nom d'hôte = ``formulaire.mondomaine.fr`` (liaison par SNI si le certificat est partagé avec d'autres sites du serveur — cas fréquent avec un certificat wildcard).
5. Pool d'applications dédié, en « No Managed Code ».
6. Démarrer le site.

10. Étape 8 — Configuration de production (appsettings.Production.json)

Créez ou complétez ``C:\RemoteApp\Formulaire\appsettings.Production.json`` avec, au minimum, les origines CORS et le chemin de la base :

```
{
  "Cors": {
    "Origins": [ "https://interface.mondomaine.fr" ]
  },
  "Storage": {
    "DatabasePath": "C:\\RemoteApp\\Formulaire-Data\\forms.db"
  }
}
```

11. Étape 9 — Vérification manuelle AVANT de tester via IIS

Ne testez pas directement via IIS en premier. Lancer l'app manuellement isole immédiatement les problèmes de code/configuration des problèmes d'environnement IIS — c'est le test le plus rapide et le plus fiable de tout ce guide, et celui qui a permis de trancher le plus vite lors du déploiement réel.

```
cd C:\RemoteApp\Formulaire
dotnet .\Logeas.Forms.Server.dll
```

Vous devez voir les migrations EF Core s'exécuter, puis :

```
info: Microsoft.Hosting.Lifetime[0]
      Application started. Press Ctrl+C to shut down.
```

Si ça plante ici (licence, connexion à la base, exception applicative), l'erreur s'affiche directement dans la console — corrigez avant de continuer. Si ça démarre correctement, **arrêtez le process (Ctrl+C)** avant de passer à l'étape suivante : le laisser tourner peut verrouiller le fichier SQLite ou entrer en conflit avec l'instance qu'IIS va lancer.

12. Étape 10 — Tester via IIS (méthode fiable)

Piège à éviter : ``curl -H “Host: monapi.fr” https://localhost/`` change l'en-tête HTTP mais **pas le SNI** envoyé pendant la négociation TLS. Si le site est en liaison par nom d'hôte (cas courant avec un certificat partagé entre plusieurs sites), ce test peut échouer de façon incohérente (parfois ça “marche” par repli de schannel sur un certificat par défaut, parfois non, notamment après un ``iisreset``).

La bonne commande, qui force le bon SNI tout en testant en local :

```
curl.exe -v -k --resolve formulaire.mondomaine.fr:443:127.0.0.1
https://formulaire.mondomaine.fr/
```

Un ``404`` sur ``/`` est normal et attendu pour une API sans route racine — c'est le signe que l'app tourne. Testez ensuite une vraie route de vos contrôleurs pour confirmer complètement.

Puis testez depuis l'extérieur (sans ``–resolve`` ni ``-k``, depuis n'importe quel poste) pour valider le DNS public et le certificat de bout en bout :

```
curl.exe -v https://formulaire.mondomaine.fr/
```

Enfin, testez depuis l'application Angular réelle, dans un navigateur, pour valider l'absence d'erreur CORS.

13. Étape 11 — Diagnostiquer une erreur 500/503 (méthode complète)

Si malgré tout l'application ne démarre pas sous IIS, suivez cette séquence dans l'ordre — c'est celle qui a permis de résoudre le déploiement réel, étape par étape, sans deviner :

1. **Le 500 générique masque le détail par défaut pour toute requête “distante”.** Testez en local avec le bon SNI (étape 10) pour obtenir la page d'erreur IIS détaillée plutôt qu'une page générique.
2. **Lisez le code d'erreur précis** dans la page détaillée (500.19, 500.30, 500.31, etc.) — chacun a une cause différente (voir annexe B).
3. **Observateur d'événements Windows → Journaux Windows → Application.** Deux sources à distinguer :
   - ``IIS AspNetCore Module V2`` : erreurs de démarrage du module lui-même (runtime introuvable, échec catastrophique de chargement).
   - ``.NET Runtime`` : exceptions applicatives non gérées, avec pile d'appels — c'est celle-ci qui donne la vraie cause si l'app plante après avoir démarré.
4. **Lancez l'app manuellement** (``dotnet .\VotreApp.dll`` depuis le dossier de déploiement) — voir étape 9. Si ça fonctionne manuellement mais pas sous IIS, la cause est presque toujours l'identité du pool (droits NTFS) ou une variable d'environnement absente sous IIS.
5. **Vérifiez l'état du pool** (``Get-WebAppPoolState -Name “<Pool>“``) — un pool ``Stopped`` peut résulter de la protection contre les échecs rapides (Rapid-Fail Protection) après plusieurs plantages successifs pendant les tests ; ``Start-WebAppPool`` le redémarre, mais ne résout pas la cause sous-jacente.
6. **Activez les logs stdout** (``stdoutLogEnabled=“true”`` dans le web.config) si aucune des étapes précédentes n'a suffi — mais vérifiez d'abord que le compte du pool a les droits d'écriture sur le dossier ``logs\`` (``icacls ... /grant “IIS_IUSRS:(OI)(CI)M”``), sinon aucun fichier n'apparaîtra malgré l'activation.

14. Étape 12 — Sécuriser l'API

- Swagger/OpenAPI désactivé ou protégé en production.
- CORS restreint à l'origine exacte de l'Angular, jamais ``AllowAnyOrigin`` en production sur un endpoint qui écrit des données.
- Droits NTFS restreints sur le dossier de déploiement et le dossier de données.
- Aucun secret (connection strings, licence) committé dans un dépôt Git public.
- Validation côté serveur de toutes les entrées, même déjà validées côté client.

15. Étape 13 — Procédure de mise à jour

1. ``dotnet publish -c Release -o ./publish``.
2. Sauvegarde de la base SQLite avant toute mise à jour de schéma (simple copie du fichier ``.db``, pool arrêté ou via l'outil ``backup`` de SQLite pour une copie à chaud cohérente).
3. Déposer ``app_offline.htm`` à la racine du site pour libérer les fichiers verrouillés.
4. ``robocopy publish C:\RemoteApp\Formulaire /MIR`` (la base, hors de ce dossier, n'est pas affectée).
5. Supprimer ``app_offline.htm``.

6. Revérifier avec la méthode de l'étape 10 (pas juste ``curl https://localhost/`` sans ``–resolve``).

16. Checklist finale (exhaustive)

Élément	Statut
Rôle IIS installé	☐
.NET 8 Hosting Bundle installé	☐
Vérifié : Test-Path aspnetcorev2.dll = True	☐
Vérifié : dotnet –list-runtimes affiche Microsoft.AspNetCore.App 8.x	☐
Visual C++ Redistributable x64 installé/réparé	☐
Redémarrage complet du serveur effectué après ces installations	☐
Politique CORS configurée avec l'origine exacte de l'Angular	☐
Dossier de données SQLite créé hors du dossier de déploiement	☐
Droits NTFS (IIS_IUSRS ou identité du pool) sur ce dossier	☐
appsettings.Production.json déployé (Cors:Origins, Storage:DatabasePath)	☐
license.key présent sur le serveur si chargement par fichier	☐
Application testée manuellement (dotnet .\App.dll) avant IIS	☐
Process manuel bien arrêté (Ctrl+C) avant test IIS	☐
Site créé dans IIS, pool en « No Managed Code »	☐
Certificat SSL/binding SNI configuré	☐
Pare-feu : ports 80/443 ouverts	☐
Test IIS via –resolve (pas -H Host seul) réussi (404 sur / = OK)	☐
Test depuis l'extérieur (DNS public) réussi	☐
Appel depuis l'Angular réel sans erreur CORS	☐
Une vraie route API testée avec succès	☐
stdoutLogEnabled repassé à false après diagnostic	☐
Sauvegarde de la base SQLite planifiée	☐

17. Annexe A — Table de décision : quel symptôme, quelle cause

Symptôme observé	Cause la plus probable	Où vérifier
500 générique, masque tout détail	Requête traitée comme “distante” par IIS	Retester en local avec –resolve (étape 10)
500.19, code 0x8007000d	Module ANCM non reconnu par IIS (Hosting Bundle absent/incomplet)	Test-Path aspnetcorev2.dll
500.31 “Failed to load ASP.NET Core runtime”	Runtime .NET absent ou bitness du pool incohérente	dotnet –list-runtimes + réglage “32 bits” du pool
Erreur générique 0x8000ffff au démarrage du module	VC++ Redistributable manquant ou non finalisé	Réinstaller vc_redist.x64.exe + redémarrage complet
503 “Application Shutting Down” en boucle, mais l'app tourne manuellement	Droits NTFS insuffisants pour l'identité du pool (souvent sur le dossier de données)	icacls sur le dossier de données, identité du pool

Symptôme observé	Cause la plus probable	Où vérifier
Connexion TLS reset par intermittence en testant avec curl -H Host	SNI non pris en compte par curl (-H Host ne suffit pas)	Utiliser –resolve à la place
Aucun fichier stdout créé malgré stdoutLogEnabled=true	Droits d'écriture manquants sur le dossier logs\	icacls sur le dossier logs
Erreur CORS dans la console navigateur	Origine absente ou incorrecte dans Cors:Origins	Vérifier appsettings.Production.json, protocole/domaine/port exacts
Les données disparaissent après une mise à jour	Base SQLite encore dans le dossier de déploiement écrasé par /MIR	Déplacer vers un dossier persistant hors déploiement

18. Annexe B — Dépannage rapide (référence courte)

Symptôme	Piste de résolution
Erreur 500.19	web.config invalide ou Hosting Bundle absent — vérifier Test-Path du module ANCM.
Erreur 500.31	Runtime absent ou bitness du pool incohérente — vérifier dotnet –list-runtimes et le réglage 32 bits.
Event ID 1010, code 0x8000ffff	VC++ Redistributable manquant — réinstaller puis redémarrer complètement le serveur.
503 en boucle, app OK en manuel	Droits NTFS du compte du pool sur le dossier de données.
curl : Recv failure / Connection was reset	Utiliser –resolve au lieu de -H Host pour un test SNI correct.
Erreur de licence Stimulsoft en prod uniquement	license.key absent du dossier publié, chemin incorrect.
Erreur 404.13 / 413 sur soumission de formulaire	Augmenter maxAllowedContentLength dans le web.config.
Erreur CORS dans la console navigateur	Origine incorrecte dans Cors:Origins, ou app.UseCors() appelé après app.MapControllers().

Document mis à jour le 18 août 2026, à partir d'un déploiement réel et de son dépannage complet.

From:
https://wiki-logeas.fr/certif/ - **dokuwiki-certif**

Permanent link:
https://wiki-logeas.fr/certif/doku.php?id=certif:procedure:miseenplaceserveur:serveurdotnet

Last update: **2026/09/30 14:39**