Guide — Générer (VS Code) et déployer manuellement le serveur .NET sur Windows Server + IIS

*Ce guide couvre le flux de travail développeur : écrire/builder l'application dans VS Code, générer le paquet de publication, puis le déployer **à la main** (copie de fichiers, sans CI/CD ni Web Deploy automatisé). Pour la configuration serveur/IIS elle-même (Hosting Bundle, site IIS, CORS, base SQLite, dépannage), voir le guide dédié — celui-ci se concentre sur le cycle "je code → je publie → je copie sur le serveur".*

---

1. Objectif

Trois temps : développer et vérifier en local dans VS Code, générer un paquet de publication propre (``dotnet publish``), puis le transférer sur le serveur par copie manuelle (RDP ou partage réseau) sans écraser ce qui doit rester stable côté serveur (configuration de production, base de données).

Ce qui ne doit jamais être écrasé par un déploiement, à garder en tête tout du long :

- ``appsettings.Production.json`` (contient les vraies valeurs CORS, chemin de base, éventuellement la licence) — ne doit pas être committé avec des secrets, ni écrasé à chaque copie s'il diffère de celui du dépôt.
- Le dossier/fichier de la base SQLite — déjà hors du dossier de déploiement si vous avez suivi le guide serveur, donc naturellement épargné par une copie du dossier de publication.

2. Prérequis

- VS Code installé, avec l'extension **C# Dev Kit** (ou a minima l'extension **C#** d'OmniSharp) — Extensions (Ctrl+Shift+X) → rechercher « C# Dev Kit ».
- Le SDK .NET 8 installé sur le poste de développement :

```
dotnet --version
```

- Un accès au serveur de déploiement : soit par **Bureau à distance (RDP)**, soit par un **partage réseau** (lecteur mappé ou chemin UNC type ``\\app-logeas-ovh\c$\RemoteApp\``).
- Le serveur déjà configuré côté IIS (rôle, Hosting Bundle, VC++ Redistributable, site créé) — voir le guide serveur dédié si ce n'est pas encore fait.

3. Étape 1 — Ouvrir et vérifier le projet dans VS Code

- ``Fichier → Ouvrir le dossier...`` → sélectionner la racine du projet (celle contenant le ``.csproj``).
- VS Code propose généralement d'ajouter les fichiers ``.vscode/tasks.json`` et ``launch.json``

nécessaires au débogage — acceptez si proposé (ou voir étape 3.1 pour les créer manuellement).
- Ouvrez un terminal intégré : ``Terminal → Nouveau terminal`` (ou ``` Ctrl+` ```), il s'ouvre déjà positionné à la racine du projet.

3.1 (Optionnel) Créer une tâche de build/publish réutilisable

Pour éviter de retaper la commande à chaque fois, créez ``.vscode/tasks.json`` :

```
{
  "version": "2.0.0",
  "tasks": [
    {
      "label": "publish",
      "command": "dotnet",
      "type": "process",
      "args": [
        "publish",
        "${workspaceFolder}/Logeas.Forms.Server.csproj",
        "-c",
        "Release",
        "-o",
        "${workspaceFolder}/publish"
      ],
      "problemMatcher": "$msCompile"
    }
  ]
}
```

Ensuite, ``Terminal → Exécuter la tâche...`` → ``publish`` lance la publication sans retaper la commande.

4. Étape 2 — Vérifier en local avant de publier

Ne publiez jamais sans avoir vérifié que l'application compile et démarre proprement en local :

```
dotnet build
dotnet run
```

Vérifiez dans la console qu'il n'y a pas d'erreur de compilation, que les migrations EF Core s'appliquent sans erreur, et que le message ``Application started`` apparaît. Arrêtez ensuite avec ``Ctrl+C``.

> Si votre ``appsettings.json`` de développement pointe vers une base SQLite locale de test, c'est normal et attendu — c'est ``appsettings.Production.json``, présent uniquement sur le serveur (ou explicitement exclu du paquet, voir étape 6), qui redirigera vers la vraie base en production.

5. Étape 3 — Publier (générer le paquet de déploiement)

Dans le terminal intégré VS Code :

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

(ou ``Terminal → Exécuter la tâche... → publish`` si vous avez créé la tâche de l'étape 3.1).

Cette commande génère dans ``./publish`` : les DLL compilées, les dépendances, le ``web.config`` prêt pour IIS, et tout fichier marqué « Copy to Output Directory » dans le ``.csproj`` (typiquement ``appsettings.json``, et ``appsettings.Production.json`` **si** il est inclus dans le projet — voir avertissement étape 6).

6. Étape 4 — Vérifier le contenu du paquet avant de le copier

```
Get-ChildItem ./publish
```

Contrôlez la présence de :

- ``Logeas.Forms.Server.dll`` (ou le nom réel de votre assembly) et ``web.config``.
- ``appsettings.json`` (toujours présent, valeurs par défaut/dev).
- Le fichier de licence Stimulsoft, si votre code le charge par fichier.

> **Point d'attention sur ``appsettings.Production.json`` :** s'il est versionné dans votre dépôt Git avec les vraies valeurs de production (CORS, chemin de base), il sera republié et écrasera la version du serveur à chaque déploiement — ce qui est en fait acceptable **si** ce fichier ne contient aucun secret sensible et que ses valeurs sont censées être identiques à chaque déploiement (ex. l'origine CORS ne change pas souvent). Si en revanche il contient des valeurs qui ne doivent **jamais** être committées (chaîne de connexion avec mot de passe, clé de licence), gardez-le **hors du dépôt Git** (``.gitignore``) et **hors du dossier publié** — dans ce cas il doit être créé une seule fois manuellement sur le serveur (voir guide serveur, étape 8) et jamais recopié depuis le poste de dev. Adaptez l'étape 8 ci-dessous en fonction de votre choix.

7. Étape 5 — Copier vers le serveur

Deux méthodes possibles, au choix selon votre confort :

7.1 Par Bureau à distance (RDP)

1. Ouvrez une session RDP vers le serveur (``mstsc``).
2. Activez le partage du presse-papiers et/ou des lecteurs locaux dans les options de connexion RDP (``Options d'affichage → Ressources locales → Lecteurs`` avant de vous connecter), pour pouvoir glisser-déposer ou copier-coller le dossier ``publish`` depuis votre poste vers le serveur.

3. Collez le contenu dans ``C:\RemoteApp\Formulaire\`` (voir étape 8 pour la méthode qui préserve les fichiers à ne pas écraser).

7.2 Par partage réseau (recommandé, plus fiable pour les mises à jour répétées)

Si le serveur expose un partage administratif ou dédié accessible depuis le poste de dev :

```
# Depuis le poste de développement, dans le terminal VS Code
robocopy .\publish "\\app-logeas-ovh\c$\RemoteApp\Formulaire" /MIR /XF
appsettings.Production.json /XD logs
```

8. Étape 6 — Ce que la copie doit préserver

Que vous copiiez à la main (RDP) ou via ``robocopy`` (partage réseau), n'écrasez jamais :

- ``appsettings.Production.json`` **si** vous avez choisi de le maintenir uniquement côté serveur (voir avertissement étape 6) — avec ``robocopy``, l'option ``/XF appsettings.Production.json`` l'exclut automatiquement de la synchronisation, dans les deux sens.
- Le dossier ``logs\`` s'il existe déjà côté serveur (pas besoin de le republier, ``/XD logs`` l'exclut).
- Tout dossier de données (base SQLite) — normalement déjà hors de ``C:\RemoteApp\Formulaire\`` si vous avez suivi le guide serveur, donc naturellement non concerné par cette copie.

Si vous copiez à la main via RDP (pas de ``robocopy``), soyez vigilant à ne pas écraser ``appsettings.Production.json`` par un glisser-déposer global du dossier ``publish`` — copiez plutôt fichier par fichier, ou dossier par dossier en excluant ce fichier spécifique.

9. Étape 7 — Libérer les fichiers verrouillés avant la copie

Le process .NET de l'application a certains fichiers ouverts (DLL en cours d'exécution) — les écraser directement peut échouer ou nécessiter un redémarrage du pool après coup. Méthode propre :

1. Sur le serveur, déposez un fichier vide nommé ``app_offline.htm`` à la racine de ``C:\RemoteApp\Formulaire\`` (IIS/ANCM arrête proprement l'application en le détectant) :

```
New-Item -ItemType File -Path "C:\RemoteApp\Formulaire\app_offline.htm" -
Force
```

1. Effectuez la copie (étape 7).
2. Supprimez ``app_offline.htm`` :

```
Remove-Item "C:\RemoteApp\Formulaire\app_offline.htm"
```

```
L'application redémarre automatiquement à la requête suivante.
```

10. Étape 8 — Vérifier après déploiement

Reprenez la méthode de test fiable du guide serveur (le piège du SNI avec curl s'applique toujours) :

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

Un ``404`` sur ``/`` est normal (API sans route racine). Testez ensuite une vraie route, puis depuis l'application Angular réelle pour valider CORS. En cas de problème, reportez-vous à la table de décision du guide serveur (annexe A) plutôt que de repartir de zéro.

11. Checklist rapide de déploiement

Élément	Statut
Build local sans erreur (dotnet build)	☐
Démarrage local sans exception (dotnet run, puis Ctrl+C)	☐
Publication générée (dotnet publish -c Release -o ./publish)	☐
Contenu du dossier publish vérifié (DLL, web.config, licence)	☐
app_offline.htm déposé avant la copie	☐
Copie effectuée SANS écraser appsettings.Production.json ni le dossier logs/données	☐
app_offline.htm supprimé après la copie	☐
Test post-déploiement réussi (curl –resolve, route réelle, Angular)	☐

12. Annexe — Script de déploiement manuel prêt à copier-coller

Un script qu'on **lance soi-même à la main** (pas de CI/CD), juste pour fiabiliser la séquence et éviter les oublis (app_offline, exclusions) :

```
# À exécuter depuis le poste de dev, dans le terminal VS Code, à la racine
du projet
$serveur = "\\app-logeas-ovh\c$\RemoteApp\Formulaire"

dotnet publish -c Release -o ./publish

New-Item -ItemType File -Path "$serveur\app_offline.htm" -Force
Start-Sleep -Seconds 2

robocopy .\publish $serveur /MIR /XF appsettings.Production.json /XD logs

Remove-Item "$serveur\app_offline.htm"

Write-Host "Déploiement terminé. Pensez à tester avec curl --resolve."
```

Document préparé le 18 août 2026.