

Guide

**Dépôt GitHub**

**Objectif : Créer un dépôt GitHub centralisé**

* **Versionner tous les fichiers critiques (.ini, .json, .yml, etc.) liés à Jitsi-Meet.**
* **Automatiser leurs transferts vers l'instance Infomaniak, même si instanciation manuelle locale.**
* **Garantir une synchronisation distante fiable et traçable (CI/CD ou push contrôlé).**

**Étapes nécessaires**

**1. Créer un dépôt GitHub structuré**

* **Arborescence suggérée :**
* **jitsi-meet-infra/**
* **├── etc/**
* **│ ├── jitsi/**
* **│ ├── prosody/**
* **│ └── nginx/**
* **├── opt/**
* **│ └── jitsi/**
* **├── scripts/**
* **│ ├── push\_to\_infomaniak.sh**
* **│ └── update\_configs.sh**
* **├── .github/**
* **│ └── workflows/**
* **│ └── deploy.yml (si CI/CD)**
* **└── README.md**

**2. Automatiser la synchronisation vers Infomaniak**

**Option A — SSH + rsync ou scp depuis un script Bash**

* **Écriture d’un push\_to\_infomaniak.sh :**

**Une image contenant texte, Police, capture d’écran

Le contenu généré par l’IA peut être incorrect.**

**Option B — CI/CD avec GitHub Actions**

* **Déploiement conditionné par main ou tag : .github/workflows/deploy.yml**

**Une image contenant texte, capture d’écran

Le contenu généré par l’IA peut être incorrect.**

**Requiert un accès SSH avec clé privée configurée via GitHub Secrets.**

**3. Faut-il utiliser n8n ?**

| **Besoin** | **n8n est-il nécessaire ?** |
| --- | --- |
| **Déployer des fichiers .ini, .yml, .json vers un serveur distant via SSH/SCP/rsync** | **❌ Non** |
| **Automatiser une suite de tâches conditionnelles, multi-services (ex : validation + appel API + trigger de backup)** | **✅ Oui** |
| **Intégrer des APIs externes (GitHub, Slack, mail, webhook) pour informer ou agir sur événements** | **✅ Oui** |
| **Avoir une interface graphique de pipeline sans script** | **✅ Oui** |

**Donc :**

**❌ Non, n8n n'est pas indispensable pour ton usage actuel (pousser des fichiers de conf).  
✅ Il devient utile si tu veux déclencher des actions complexes ou multi-étapes (ex : valider, notifier, exécuter, journaliser dans Notion).**

**Recommandation**

**Pour ton cas (Jitsi / Infomaniak) :**

* **Priorise rsync/scp** avec **clé SSH,** ou **GitHub Actions** si tu veux **pousser via git.**
* **Si besoin** futur de workflow visuel multi-actions **: ajoute n8n (ex : déclenchement si IP change, notification Telegram, backup GCP, etc.).**

**1. Arborescence GitHub :** [**https://github.com/001418-ENB/Tercium\_monitoring**](https://github.com/001418-ENB/Tercium_monitoring)

**Nom du dépôt proposé** **: jitsi-meet-infra**

**jitsi-meet-infra/**

**├── etc/**

**│ ├── jitsi/**

**│ ├── prosody/**

**│ └── nginx/**

**├── opt/**

**│ └── jitsi/**

**├── scripts/**

**│ ├── push\_to\_infomaniak.sh**

**│ └── update\_configs.sh**

**├── .github/**

**│ └── workflows/**

**│ └── deploy.yml**

**└── README.md**

Copier les fichiers réels dans les sous-dossiers **etc/** et **opt/,** puis les versionner avec **Git.**

**2. Script push\_to\_infomaniak.sh**

**Nom du fichier** : **push\_to\_infomaniak.sh  
Emplacement** : scripts/push\_to\_infomaniak.sh  
**Rôle** : Transférer manuellement tous les **fichiers critiques /etc/ et /opt/** vers le **serveur Infomaniak via rsync + SSH**

Une image contenant texte, capture d’écran, Police

Le contenu généré par l’IA peut être incorrect.

**Ne pas oublier pas : chmod +x scripts/push\_to\_infomaniak.sh**

**3. Fichier GitHub Actions : deploy.yml**

**Nom du fichier** : deploy.yml  
**Emplacement** : **.github/workflows/deploy.yml  
Rôle** : Déclencher un déploiement automatique sur push (branche main)  
**Prérequis** :

* Ajoute dans les **GitHub Secrets** :
  + **SSH\_PRIVATE\_KEY**
  + **REMOTE\_USER**
  + **REMOTE\_HOST**

Une image contenant texte, capture d’écran

Le contenu généré par l’IA peut être incorrect.

**Deux compléments, pour finaliser un dépôt GitHub propre et opérationnel :**

**4. Makefile**

**Nom du fichier : Makefile  
Emplacement : racine du projet (jitsi-meet-infra/)  
Rôle : Simplifier l'exécution de commandes avec des cibles explicites (make deploy, make push, make check…)**

**Une image contenant texte, capture d’écran

Le contenu généré par l’IA peut être incorrect.**

**✔️ make deploy exécute ton script Bash.  
✔️ make push lance rsync directement.  
✔️ make check teste si ton SSH est bien configuré.**

**5. README.md — Procédure Markdown**

**Nom du fichier : README.md  
Emplacement : racine du dépôt  
Rôle : Documenter la structure et les instructions d’utilisation (locale ou CI/CD)**

**jitsi-meet-infra/  
├── etc/ # Fichiers système de /etc/jitsi, /etc/prosody, etc.  
├── opt/ # Configuration et modules spécifiques de /opt/jitsi  
├── scripts/ # Scripts de push SSH et de mise à jour  
├── .github/workflows # Déploiement automatique via GitHub Actions  
├── Makefile # Commandes rapides (push, deploy, check)  
└── README.md**

**Une image contenant texte, capture d’écran, Police

Le contenu généré par l’IA peut être incorrect.**

**Configuration SSH (manuel ou GitHub Actions)**

* **Génère une paire de clés SSH : cd : ssh-keygen -t rsa -b 4096 -C "deploy@jitsi"**
* **Copie la clé publique vers l’instance Infomaniak : cd : ssh-copy-id -i ~/.ssh/id\_rsa.pub ubuntu@IP\_INFOMANIAK**
* **Dans GitHub → Settings → Secrets and variables → Actions :**
  + SSH\_PRIVATE\_KEY = contenu de ~/.ssh/id\_rsa
  + REMOTE\_USER = ubuntu
  + REMOTE\_HOST = your-instance.infomaniak.com

## Déploiement automatique via GitHub Actions

**À chaque push sur main, les fichiers de /etc et /opt sont transférés automatiquement.**

**Vérifie le statut ici : Actions > Deploy to Infomaniak.**

**Pour sécuriser les échanges entre ton terminal, GitHub, et ton serveur Infomaniak :**

**Résumé des rôles des clés SSH :**

| **Destination** | **Clé recommandée** | **Usage** |
| --- | --- | --- |
| **GitHub (Git push/pull)** | ✅ **Clé SSH spécifique GitHub** | Authentification au dépôt distant |
| **Serveur Infomaniak** | ✅ **Clé SSH machine/infra** | Déploiement automatisé via rsync, scp, etc. |
| **GitHub Actions → SSH** | ✅ **Clé dédiée CI/CD (privée dans Secrets)** | Déploiement distant automatisé |

**Cas actuels et conseils**

**1. ✅ Tu as déjà une clé SSH locale pour GitHub**

Si elle fonctionne (git push sans mot de passe), tu peux **la garder pour ton usage personnel**.

Vérifie avec :

**cd : ssh -T git@github.com**

**2. ✅ Clé SSH pour le serveur Infomaniak**

Clé locale que tu ajoutes via :

**cd : ssh-copy-id -i ~/.ssh/id\_rsa.pub ubuntu@IP\_INFOMANIAK**

**3. ✅ Clé SSH pour GitHub Actions (CI/CD)**

Tu dois **générer une nouvelle paire de clés dédiée au bot GitHub Actions** (et **ne pas** utiliser celle de ton terminal !)

**Étapes :**

# Sur ton terminal local (pas sur GitHub Actions)

**cd :** ssh-keygen -t rsa -b 4096 -f ~/.ssh/github\_action\_rsa -C "gh-action@jitsi-infra"

**Cela crée :**

* github\_action\_rsa (clé privée)
* github\_action\_rsa.pub (clé publique)

**Ensuite :**

* **Sur Infomaniak** :

Ajoute la clé **publique** (.pub) dans ~/.ssh/authorized\_keys de l'utilisateur ubuntu

* **Sur GitHub (repo > Settings > Secrets)** :
  + Ajoute la **clé privée** comme secret SSH\_PRIVATE\_KEY

**Résumé final des clés utilisées**

| **Contexte** | **Fichier (exemple)** | **Localisation** | **Usage** |
| --- | --- | --- | --- |
| **GitHub (terminal perso)** | **~/.ssh/id\_rsa\_github** | Local uniquement | **Git push/pull** |
| **Serveur Infomaniak (local)** | **~/.ssh/id\_rsa\_infomaniak** | Local + authorized\_keys | **scp, rsync, ssh** |
| **GitHub Actions → SSH** | **github\_action\_rsa (privée)** | **Privée dans GitHub**, publique sur le serveur | **CI/CD secure** |

**Procédure sécurisée – Gestion des clés SSH**

**Procédure complète de gestion des clés SSH pour sécuriser et automatiser tes échanges entre ton terminal local, GitHub, et le serveur Infomaniak :**

**Procédure sécurisée – Gestion des clés SSH**

**Objectif**

* **Assurer 3 canaux sécurisés :**
  1. **Git local ↔️ GitHub**
  2. **Terminal local ↔️ Serveur Infomaniak**
  3. **GitHub Actions ↔️ Serveur Infomaniak**

**1. Clé SSH personnelle pour GitHub**

* **Vérifier ou générer : cd : ls ~/.ssh/id\_rsa\_github**

**Si absent : cd : ssh-keygen -t rsa -b 4096 -f ~/.ssh/id\_rsa\_github -C "github-local@machine"**

* **Ajouter la clé publique à GitHub**

1. **Copie le contenu : cd : cat ~/.ssh/id\_rsa\_github.pub**
2. **Va sur GitHub → Settings → SSH and GPG keys  
   → New SSH key  
   → Colle la clé publique**

* **Configurer le ~/.ssh/config**

**Une image contenant texte, Police, capture d’écran

Le contenu généré par l’IA peut être incorrect.**

**2. Clé SSH pour ton accès local → Infomaniak**

* **Générer une clé dédiée cd : ssh-keygen -t rsa -b 4096 -f ~/.ssh/id\_rsa\_infomaniak -C "local-access@infomaniak"**
* **Copier la clé publique sur le serveur cd : ssh-copy-id -i ~/.ssh/id\_rsa\_infomaniak.pub ubuntu@IP\_INFOMANIAK**
* **Ajouter une entrée dans ~/.ssh/config**

**Une image contenant texte, Police, capture d’écran

Le contenu généré par l’IA peut être incorrect.**

* **Tester : cd :ssh infomaniak**

**3. Clé SSH dédiée pour GitHub Actions → Infomaniak**

* **Générer la clé sur ton terminal local**

**cd :ssh-keygen -t rsa -b 4096 -f ~/.ssh/github\_action\_rsa -C "gh-actions@jitsi-infra"**

* **Cela crée :**
  + **github\_action\_rsa (privée)**
  + **github\_action\_rsa.pub (publique)**
* **Ajouter la clé publique sur Infomaniak : cd :cat ~/.ssh/github\_action\_rsa.pub**
* **Puis connecte-toi sur le serveur : cd :ssh infomaniak**

**Et ajoute la clé dans : cd : nano ~/.ssh/authorized\_keys**

**Colle la ligne de clé publique et sauvegarde.**

**Ajouter la clé privée dans GitHub Secrets**

1. **Copie le contenu : cd : cat ~/.ssh/github\_action\_rsa**
2. **Va dans le repo GitHub → Settings → Secrets and variables → Actions  
   → Nouveau secret :**
   * **Nom : SSH\_PRIVATE\_KEY**
   * **Valeur : contenu de la clé privée (github\_action\_rsa)**

**Test final (facultatif)**

1. **Local : cd : ssh infomaniak**
2. **GitHub Actions :**Push un fichier sur la branche **main** pour déclencher **deploy.yml :**

**Une image contenant texte, Police, capture d’écran, conception

Le contenu généré par l’IA peut être incorrect.**

1. **Vérifie** dans **l’onglet Actions** du **dépôt GitHub** que le déploiement passe sans erreur**.**

**Sécurité finale :**

* **Donner** des noms clairs à chaque clé **(id\_rsa\_github, id\_rsa\_infomaniak, github\_action\_rsa)**
* **Ne jamais mélanger clef perso et clef CI/CD.**
* **Ajoute des règles de chmod :**

**Une image contenant texte, Police, capture d’écran, Graphique

Le contenu généré par l’IA peut être incorrect.**

**GitHub Actions : Secrets requis**

| **Secret GitHub** | **Contenu** |
| --- | --- |
| **SSH\_PRIVATE\_KEY** | **Clé privée github\_action\_rsa** |
| **REMOTE\_USER** | **ubuntu** |
| **REMOTE\_HOST** | **IP\_INFOMANIAK** |

**Bonnes pratiques :**

* **Ne jamais réutiliser une clé pour plusieurs usages.**
* **Ne pas versionner les clés privées.**
* **Ajouter une date d’expiration ou de rotation trimestrielle.**
* **Contrôler les accès à ~/.ssh/authorized\_keys sur le serveur.**

Synthèse complète et structurée de tout ce qui a été généré (ou à générer) pour créer une routine **GitHub + SSH** propre, sécurisée et automatisée pour ton projet **Jitsi-Meet** sur **Infomaniak** :

**SYNTHÈSE TECHNIQUE – Routines SSH + GitHub CI/CD**

**STRUCTURE DES FICHIERS À INTÉGRER**

**Répertoire principal : jitsi-meet-infra/**

| **Fichier / Dossier** | **Rôle** |
| --- | --- |
| **README.md** | **Présentation fonctionnelle et technique du dépôt** |
| **SECURITY.md** | **Documentation sur la gestion des clés SSH** |
| **Makefile** | **Automatisation des commandes locales (deploy, check, rollback, etc.)** |
| **.env** | **Centralisation des variables d’environnement (utilisateur/IP/ports)** |

**scripts/ – Scripts Bash d'automatisation**

| **Fichier** | **Rôle** |
| --- | --- |
| **push\_to\_infomaniak.sh** | **Transfert des dossiers /etc et /opt via rsync** |
| **post-deploy.sh** | **Vérification post-déploiement (services, ports, logs)** |
| **rollback.sh** | **Restauration à partir des backups (sous /backup)** |
| **automate\_keys.sh** | **Génération automatisée des clés SSH avec bons noms et permissions** |

**.github/workflows/**

| **Fichier** | **Rôle** |
| --- | --- |
| **deploy.yml** | **Déclenchement CI/CD vers Infomaniak à chaque push sur main** |

**État des fichiers dans ton dépôt GitHub jitsi-meet-infra :**

| **Nom** | **Fonction** | **Statut** | **Remarque** |
| --- | --- | --- | --- |
| **.env** | **Variables réelles (non versionné)** | **✅ OK** | Ignoré via **.gitignore** |
| **.env.template** | **Gabarit à versionner** | **✅ OK** | **À copier en .env** |
| **.gitignore** | **Exclusions de version** | **✅ OK** | Inclut **.env, clés, logs** |
| **automate\_keys.sh** | **Générateur de clés SSH** | **✅ OK** | **Script bien nommé** |
| **Deploy-jitsi-Configs.yml** | **CI/CD ou configuration Jitsi ?** | **✅À clarifier** | Renommer si utilisé par **GitHub Actions (ex : deploy.yml)** |
| **GitHub-Actions\_deploy.yml** | **CI/CD GitHub Actions** | **✅ OK** | **Nom alternatif à valider** |
| **Makefile** | **Routine principale** | **✅ OK** | Appelle **.env** |
| **Makefile.test** | **Variante ou test de CI** | **✅ OK** | **Bien séparé** |
| **post-deploy.sh** | **Vérification post-déploiement** | **✅ OK** | Appel via **make verify** |
| **push\_to\_infomaniak.sh** | **Script de push manuel** | **✅ OK** | Appelé **via make deploy** |
| **rollback.sh** | **Restauration depuis /backup/** | **✅ OK** | Appelé avec **BACKUP=** |
| **README.md** | **Description du dépôt** | **✅ OK** | **Très bon niveau** |
| **SECURITY.md** | **Documentation sur les clés SSH** | **✅ OK** | **Conforme à usage GitHub** |
| **Dépôt conseils et fichiers.docx** | **Support externe / interne** | **À déplacer** | Document Microsoft, non versionnable — à mettre en **/docs/ ou /support/ facultatif** |

**LOGIQUE GLOBALE DE LA ROUTINE AUTOMATISÉE**

**1. Clés SSH distinctes générées pour :**

**- GitHub (usage personnel)**

**- Serveur (accès manuel local)**

**- GitHub Actions (CI/CD)**

**2. Structure du dépôt versionnée avec tous les fichiers Jitsi :**

**- etc/jitsi**

**- opt/jitsi**

**- scripts/**

**- workflows/**

**3. Routine de push/déploiement :**

**make deploy**

**↳ push\_to\_infomaniak.sh**

**↳ rsync vers /etc et /opt**

**↳ logs + vérif post-deploy (optionnelle)**

**4. Sauvegarde automatique (optionnelle) sur le serveur :**

**/backup/etc/jitsi\_<date>**

**/backup/opt/jitsi\_<date>**

**5. Possibilité de rollback :**

**make rollback BACKUP=2025-07-28\_1500 rollback.sh utilise les backups horodatés**

**6. GitHub Actions fonctionne en parallèle :**

**- Avec clé dédiée CI/CD**

**- Déploiement automatique à chaque push sur main**

**Organisation finale :**

**jitsi-meet-infra/**

**├── .env.template**

**├── .env # non versionné**

**├── .gitignore**

**├── Makefile**

**├── Makefile.test**

**├── README.md**

**├── SECURITY.md**

**├── CONFIGURATION.md**

**├── docs/**

**│ └── Dépôt conseils et fichiers.docx**

**├── scripts/**

**│ ├── automate\_keys.sh**

**│ ├── post-deploy.sh**

**│ ├── push\_to\_infomaniak.sh**

**│ ├── rollback.sh**

**├── .github/**

**│ └── workflows/**

**│ ├── deploy.yml**

**└── deploy-alt.yml # (optionnel)**

L**iste compartimentée des commandes** terminal dans l'ordre exact pour **automatiser un dépôt jitsi-meet-infra**, depuis l'initialisation jusqu'au déploiement **CI/CD**.

**1. Initialisation du dépôt local**

**# Créer le dossier et se positionner**

**cd :** mkdir jitsi-meet-infra && cd jitsi-meet-infra

**# Initialiser un dépôt Git**

**cd :** git init

**# Ajouter un dépôt distant (exemple GitHub)**

**cd :** git remote add origin git@github.com:TonUser/jitsi-meet-infra.git

**2. Fichiers de configuration**

**# Créer le .gitignore**

**cd :** touch .gitignore

**# Créer le modèle d'environnement**

**cd :** touch .env.template

**# Créer la version active (non versionnée)**

**cd :** cp .env.template .env

**# Vérifier et éditer**

**cd :** nano .env

**3. Structure du projet**

**# Créer l’arborescence**

**cd :** mkdir -p etc opt scripts .github/workflows docs

# **Ajouter le README et documents**

**cd :** touch README.md SECURITY.md CONFIGURATION.md

**# Ajouter les scripts**

**cd :** touch scripts/automate\_keys.sh

**cd :** touch scripts/push\_to\_infomaniak.sh

**cd :** touch scripts/post-deploy.sh

**cd :** touch scripts/rollback.sh

**# Ajouter le Makefile**

**cd :** touch Makefile

**4. Génération des clés SSH**

**# Lancer le script automatisé**

**cd :** chmod +x scripts/automate\_keys.sh

**cd :**./scripts/automate\_keys.sh

**# Copier la clé CI vers Infomaniak**

**cd :** ssh ubuntu@IP\_INFOMANIAK "mkdir -p ~/.ssh && touch ~/.ssh/authorized\_keys"

**cd :** cat ~/.ssh/github\_action\_rsa.pub | ssh ubuntu@IP\_INFOMANIAK "cat >> ~/.ssh/authorized\_keys"

**5. Déploiement et vérifications**

**# Vérifier la connexion SSH**

**cd :** make check

**# Transfert vers Infomaniak**

**cd :** make deploy

**# Vérification post-déploiement**

**cd :** make verify

**6. Sauvegarde et rollback**

**# Sauvegarde horodatée**

**cd :** make backup

**# Rollback (exemple)**

**cd :** make rollback BACKUP=2025-07-28\_1500

**7. CI/CD GitHub Actions**

**# Créer le workflow**

**cd :** touch .github/workflows/deploy.yml

**# Ajouter les secrets dans GitHub :**

# SSH\_PRIVATE\_KEY

# REMOTE\_USER

# REMOTE\_HOST

**# Pousser le dépôt complet**

**cd :** git add .

**cd :** git commit -m "Initialisation complète"

**cd :** git push origin main

**Carte mentale – Projet jitsi-meet-infra :**

**jitsi-meet-infra/**

**│**

**├── Configuration**

**│ ├── .env.template → Modèle des variables**

**│ ├── .env → Fichier réel (non versionné)**

**│ ├── .gitignore → Exclut clés, logs, .env**

**│ └── CONFIGURATION.md → Explication complète du setup**

**│**

**├── Scripts**

**│ ├── automate\_keys.sh → Génère les clés SSH (GitHub, serveur, CI)**

**│ ├── push\_to\_infomaniak.sh → Transfert config via rsync**

**│ ├── post-deploy.sh → Vérifie services & ports après push**

**│ └── rollback.sh → Restaure à partir d’un backup horodaté**

**│**

**├── Automatisation**

**│ ├── Makefile → Commandes locales (push, deploy, verify, rollback)**

**│ ├── Makefile.test → Version alternative ou de test**

**│ └── .github/workflows/**

**│ └── deploy.yml → GitHub Actions : déploiement CI/CD sécurisé**

**│**

**├── Documentation**

**│ ├── README.md → Présentation du projet, commandes, contexte**

**│ ├── SECURITY.md → Gestion des clés SSH**

**│ └── docs/**

**│ └── Dépôt conseils et fichiers.docx → Notes internes ou support écrit**

**│**

**├── Données versionnées**

**│ ├── etc/ → Configurations Jitsi, Prosody, etc.**

**│ └── opt/ → Modules Jitsi, plugins ou scripts associés**

**│**

**└── Routines Make**

**├── make check → Vérifie la connexion SSH**

**├── make deploy → Exécute le script de push**

**├── make verify → Audit post-déploiement**

**├── make backup → Sauvegarde les dossiers sur le serveur**

**├── make rollback BACKUP=... → Restaure une version antérieure**

**└── make ci\_test → Simule un déploiement GitHub Actions**

**CLARIFICATIONS SUR .env, Makefile, DOCKER ET VERSIONNEMENT**

**1. .env : rôle et usage dans ce projet**

Tu **n’utilises pas .env pour Docker ici**, **mais pour :**

* Centraliser des **variables d’environnement** (utilisateur SSH, IP, ports, chemins…)
* Éviter de réécrire manuellement ces valeurs dans chaque script

**Exemple de .env (à la racine du dépôt)**

**# .env**

REMOTE\_USER=ubuntu

REMOTE\_HOST=1.2.3.4

SSH\_PORT=22

**# Chemins distants**

REMOTE\_ETC=/etc/jitsi

REMOTE\_OPT=/opt/jitsi

REMOTE\_BACKUP\_ETC=/backup/etc

REMOTE\_BACKUP\_OPT=/backup/opt

**Ce fichier sera lu dans le** **Makefile via une inclusion include .env**.

**2. Makefile : versionnable, structurable, différenciable**

Oui, **on peut versionner un Makefile**, et **tu peux en avoir plusieurs** si nécessaire :

**Stratégies de différenciation**

| **Type de Makefile** | **Fichier associé** | **Usage** |
| --- | --- | --- |
| **Standard** | **Makefile** | **Appelé par défaut via make** |
| **Variante déploiement test** | **Makefile.test** | **Appelé via make -f Makefile.test** |
| **Variante prod sécurisée** | **Makefile.prod** | **Appelé via make -f Makefile.prod** |

**STRATÉGIE RECOMMANDÉE**

Tu gardes **un seul Makefile principal** avec des cibles explicites :

**make deploy**

**make check**

**make rollback BACKUP=2025-07-28\_1500**

**make backup**

**make verify**

**make ci\_test**

Et tu relies **les variables dynamiques à .env** pour tout personnaliser sans modifier le code.

**DOCKER ?**

Si tu veux **plus tard** encapsuler le déploiement dans un conteneur (Jitsi ou CI/CD Dockerisé), tu pourras **réutiliser .env** aussi pour :

**docker-compose.yml:**

**env\_file:**

**- .env**

Mais **ici, ce n’est pas le cas** **.env** sert uniquement à injecter les paramètres dans les scripts **Bash** et le **Makefile**.

L**iste chronologique complète** des **commandes au terminal** pour créer et configurer un dépôt GitHub, selon ton contexte Jitsi/Infomaniak tel que détaillé dans le guide :

**1. Initialisation locale du dépôt**

Une image contenant texte, Police, capture d’écran

Le contenu généré par l’IA peut être incorrect.

Remplacer **TonUser** par ton identifiant GitHub.

**2. Préparation des fichiers et structure**

Une image contenant texte, capture d’écran, Police

Le contenu généré par l’IA peut être incorrect.

**3. Génération des clés SSH**

Une image contenant texte, Police, capture d’écran, Graphique

Le contenu généré par l’IA peut être incorrect.

**Ce script doit générer :**

* **Clé locale pour GitHub**
* **Clé pour accès manuel à Infomaniak**
* **Clé pour CI/CD (GitHub Actions)**

**4. Déploiement manuel vers Infomaniak**

Une image contenant texte, capture d’écran, Police

Le contenu généré par l’IA peut être incorrect.

**5. Sauvegarde / rollback (optionnel)**

Une image contenant texte, Police, capture d’écran

Le contenu généré par l’IA peut être incorrect.

**6. Mise en place CI/CD via GitHub Actions**



**Contenu** **deploy.yml** : déclenchement sur **push** → **déploiement via SSH**.

**Ensuite :**

Une image contenant texte, Police, capture d’écran

Le contenu généré par l’IA peut être incorrect.

**Ajoute dans** **GitHub > Settings > Secrets** :

| **Secret name** | **Valeur** |
| --- | --- |
| **SSH\_PRIVATE\_KEY** | **contenu de github\_action\_rsa** |
| **REMOTE\_USER** | **ubuntu** |
| **REMOTE\_HOST** | **IP ou domaine Infomaniak** |

**Il créer le dépôt via GitHub interface Web une seule fois** pour initialiser l’espace distant (nom du dépôt, visibilité). Ensuite, **tout peut être fait via le terminal**.

**Résumé final**

* **Obligatoire** : création initiale du dépôt via l’interface GitHub.
* **Le reste** (initialisation, push, CI/CD, déploiement, backup) est **100 % faisable en terminal**.
* Tu peux automatiser **entièrement** la chaîne via Makefile + GitHub Actions.

**Deux connexions bien distinctes**

| **Connexion** | **Sert à quoi ?** | **Exemple technique** |
| --- | --- | --- |
| **Git ↔ GitHub** | **Push/pull du code** | **git push, git clone, etc.** |
| **Local ↔ Instance Infomaniak (SSH)** | **Déployer/synchroniser le code** | **rsync, scp, ssh, GitHub Actions** |

**Réponse courte**

Tu **peux créer le dépôt GitHub, faire tous les commits et pushs depuis ton terminal local**, **même si tu n’as pas encore de connexion SSH vers l’instance Infomaniak**.

**Mais** :

On aura **besoin d’une connexion SSH fonctionnelle vers l’instance** **au moment du déploiement**, que ce soit :

* Via **push\_to\_infomaniak.sh**
* Ou via **GitHub Actions**

**Conséquences pratiques**

| **Action** | **SSH vers Infomaniak nécessaire ?** |
| --- | --- |
| **git init**, **git add**, **git commit** | ❌ |
| **git push origin main** | ❌ (nécessite juste accès GitHub) |
| **rsync, scp** (push manuel) | ✅ |
| **GitHub Actions** (push CI/CD automatisé) | ✅ (la clé publique CI doit être installée) |

**Recommandation de workflow**

1. **Prépare ton dépôt GitHub localement** sans attendre la connexion SSH.
2. **Quand tu seras prêt à déployer**, configure la clé SSH :
3. ssh-copy-id -i ~/.ssh/id\_rsa.pub ubuntu@IP\_INFOMANIAK
4. Puis teste :
5. ssh ubuntu@IP\_INFOMANIAK

**Un check-deploy.sh minimal pour tester :**

1. **Le rendre exécutable :**

**cd : chmod +x check-deploy.sh**

1. **Préparer un fichier .env à la racine avec : cd : REMOTE\_USER=ubuntu**

**REMOTE\_HOST=ip\_de\_ton\_instance**

**SSH\_PORT=22**

1. **Lancer le script :**

**cd : ./check-deploy.sh**

**Option Makefile**

**Ajoute une cible dans ton Makefile :**

**cd : check:**

**@bash ./check-deploy.sh**

**Puis**

**cd : make check**

**Si tu as plusieurs Makefiles dans un projet et que tu souhaites les distinguer tout en les maintenant actifs (sans désactivation), tu peux utiliser plusieurs extensions pour chaque fonction spécifique ou environnement.**

**Voici comment procéder pour distinguer les Makefiles tout en gardant une structure cohérente et fonctionnelle :**

**Utilisation de Makefiles avec des extensions :**

* **Renommer les fichiers Makefile avec des extensions spécifiques :**

**Par exemple :**

* + **Makefile : pour la configuration par défaut**
  + **Makefile.test : pour les tests**
  + **Makefile.prod : pour la production**
  + **Makefile.dev : pour l’environnement de développement**
  + **Makefile.deploy : pour les actions de déploiement**

**Exemples :**

* + **Makefile.test pour des cibles spécifiques aux tests**
* **Makefile.prod pour les cibles de production, etc.**
* **Exécuter un Makefile avec une extension spécifique :**

**Pour exécuter un Makefile avec une extension spécifique, utilise l'option -f :**

**Exemple pour utiliser Makefile.test :**

**cd : make -f Makefile.test check**

**Exemple pour utiliser Makefile.prod :**

**cd : make -f Makefile.prod deploy**

**Structure le projet, arborescence de répertoires :**

**.**

**├── Makefile # Makefile principal, pour la configuration par défaut**

**├── Makefile.test # Makefile pour l'environnement de test**

**├── Makefile.prod # Makefile pour la production**

**├── Makefile.dev # Makefile pour le développement local**

**├── Makefile.deploy # Makefile pour le déploiement automatisé**

**├── .github/**

**│ └── workflows/**

**│ └── deploy.yml**

**└── scripts/**

**├── push\_to\_infomaniak.sh**

**├── deploy.sh**

**└── test.sh**

**Conseils supplémentaires :**

1. **Variables communes :  
   Partager certaines variables entre plusieurs Makefiles, tu peux inclure un fichier commun :**

**Par exemple, dans chaque Makefile :**

**include common.mk**

**Dans**

**common.mk :**

**REMOTE\_USER=ubuntu**

**REMOTE\_HOST=example.com**

**SSH\_PORT=22**

1. **Faciliter l'exécution avec un Makefile global :  
   Si tu veux un Makefile global pour gérer les différentes cibles sans réécrire de commandes, tu peux faire appel à un Makefile principal qui inclut d’autres Makefiles selon l’environnement :**

**Exemple Makefile.main :**

**.DEFAULT\_GOAL := help**

**help:**

**@echo "Utiliser 'make -f Makefile.<extension>' pour un environnement spécifique."**

**include Makefile.test**

**include Makefile.prod**

**include Makefile.dev**

**include Makefile.deploy**

**Exemple d'usage des Makefiles**

**1. Exécuter des tests avec Makefile.test :**

**cd : make -f Makefile.test test**

**2. Déployer en production avec Makefile.prod :**

**cd : make -f Makefile.prod deploy**

**3. Déploiement avec Makefile.deploy :**

**cd : make -f Makefile.deploy deploy**