

**Réalisé par :**

**Sinda JAOUADI**

**COMpte rendu TP4 gitlab :**

**COMPLEX PIPELEINE**

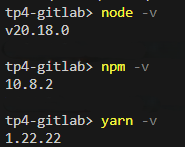
**Objectif :**

* Déployer un pipeline GitLab complexe en utilisant des jobs cachés (.hidden-job) et des extensions (extends).
* Valider le fonctionnement des jobs tout en réduisant les redondances dans la configuration.

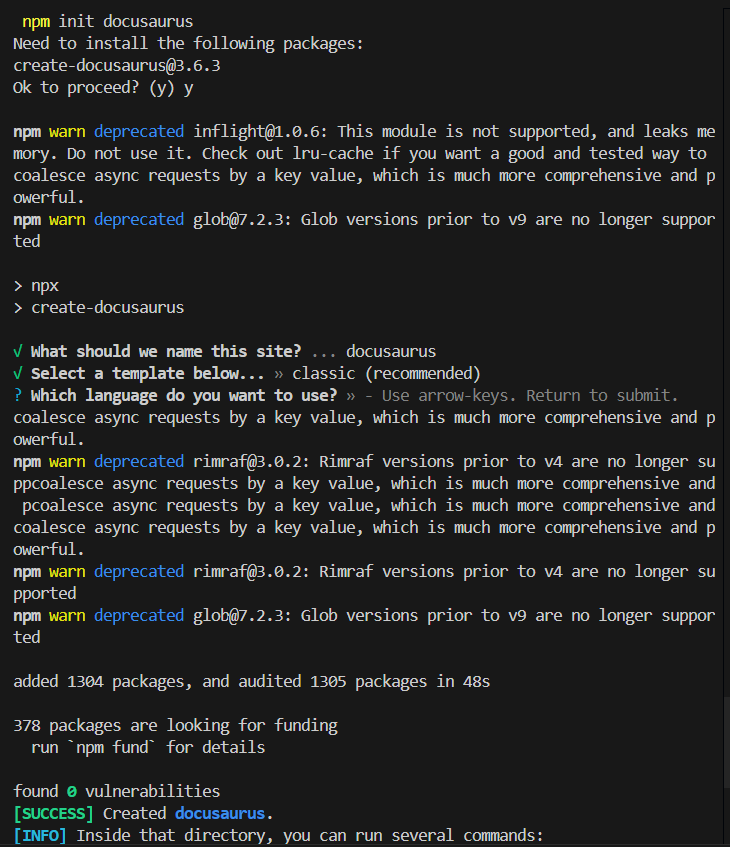
**Étapes Réalisées :**

1. **Créer le projet**

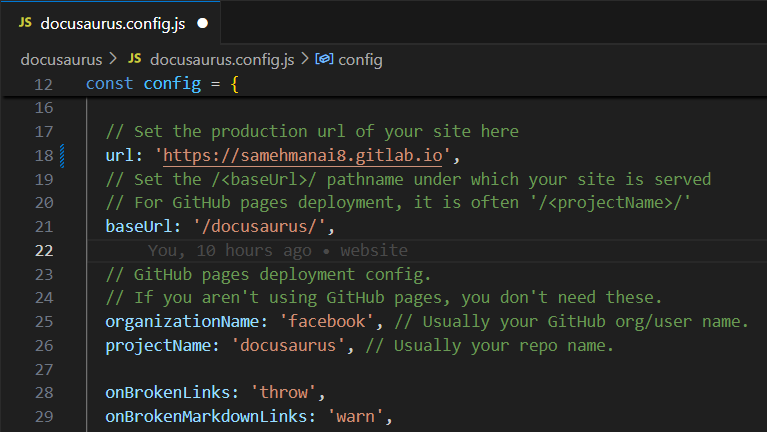
* Installer node. Js, npm et yarn



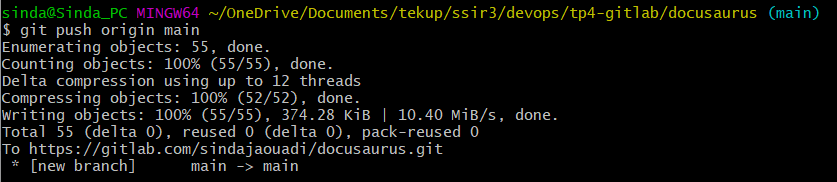
* Générer un site Docusaurus



* Mettre à jour le fichier de configuration Docusaurus

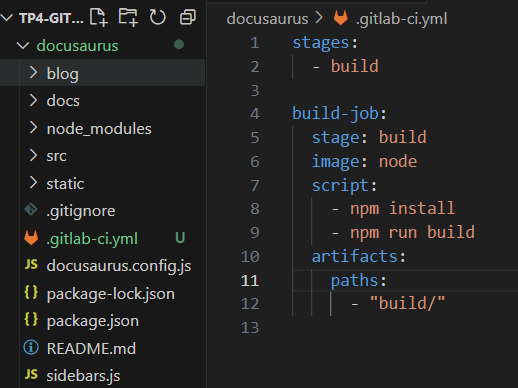
****

* Pusher le travail sur Gitlab



1. **Créer le pipeline build job**

Tout d’abord, on commence par créer le fichier .gitlab-ci.yml



Le pipeline décrit est défini par un job build-job avec les étapes suivantes :

* Image utilisée : node

Le pipeline s’exécute dans un environnement Docker basé sur l’image node. Cela garantit que les outils nécessaires (Node.js, npm) sont disponibles.

* Script exécuté :

npm install :

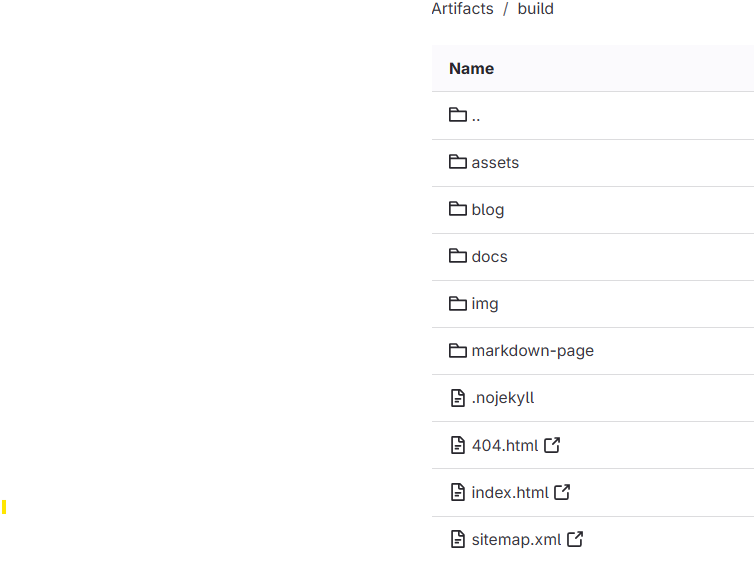
* + - Installe toutes les dépendances définies dans le fichier package.json du projet.
    - Prépare l'environnement pour la construction du site ou de l'application.

npm run build :

* + - Exécute le script de construction défini dans package.json (habituellement sous la clé "scripts": { "build": "..." }).
    - Génère une version optimisée de l'application, généralement un site ou une application prête à être déployée.
* Artifacts :

Les fichiers générés par la commande npm run build (souvent stockés dans un répertoire nommé build/) sont sauvegardés en tant qu’artefacts du pipeline. Ces artefacts peuvent être utilisés dans les étapes suivantes du pipeline ou téléchargés manuellement.

Après l’exécution du pipeline, voici le contenu du répertoire build/ :

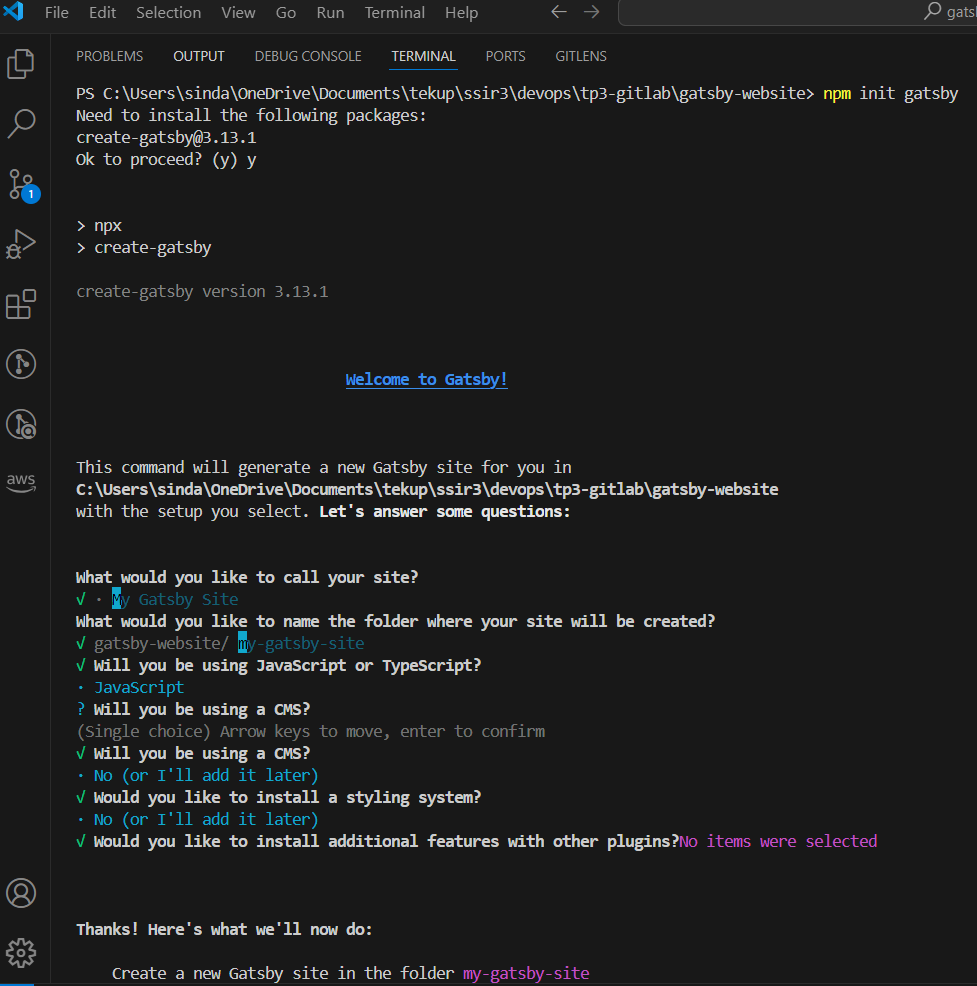


Créer un projet GitLab nommé gatsby-website

* Cloner le projet en local

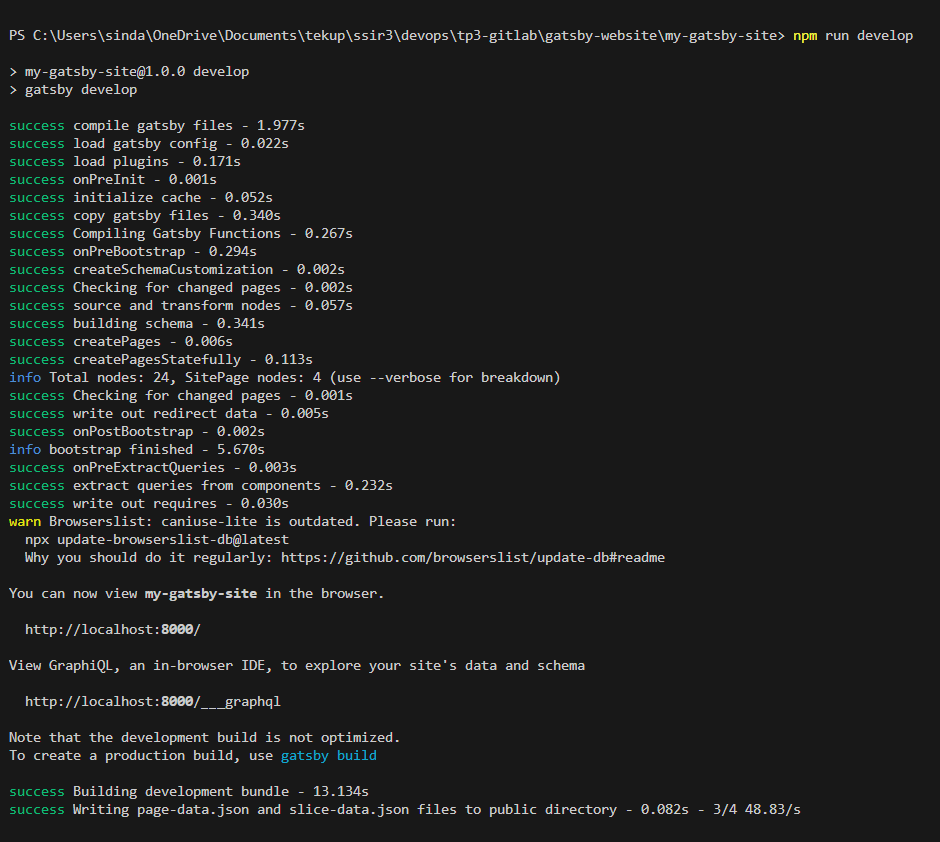
**Étape 2 : Création d’un site sur Gatsby**

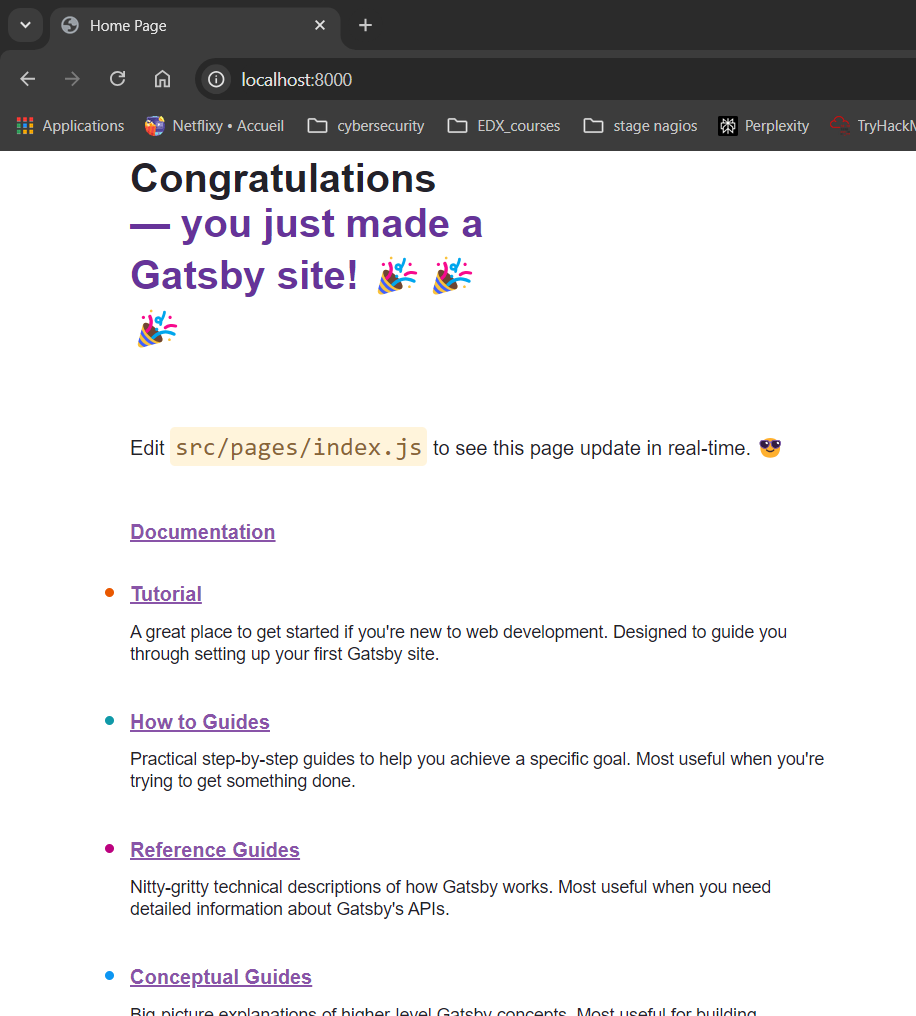
Créer un nouveau site : ***npm init gatsby***



Démarrer le serveur de développement local avec : ***npm run develop***

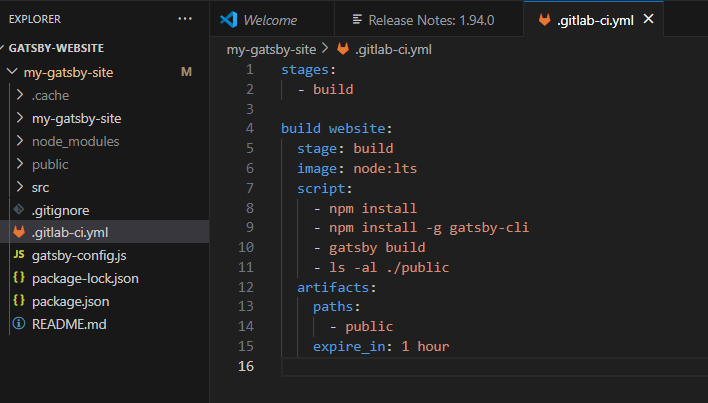
Gatsby démarrera un environnement accessible par défaut à [***http://localhost:8000***](http://localhost:8000)





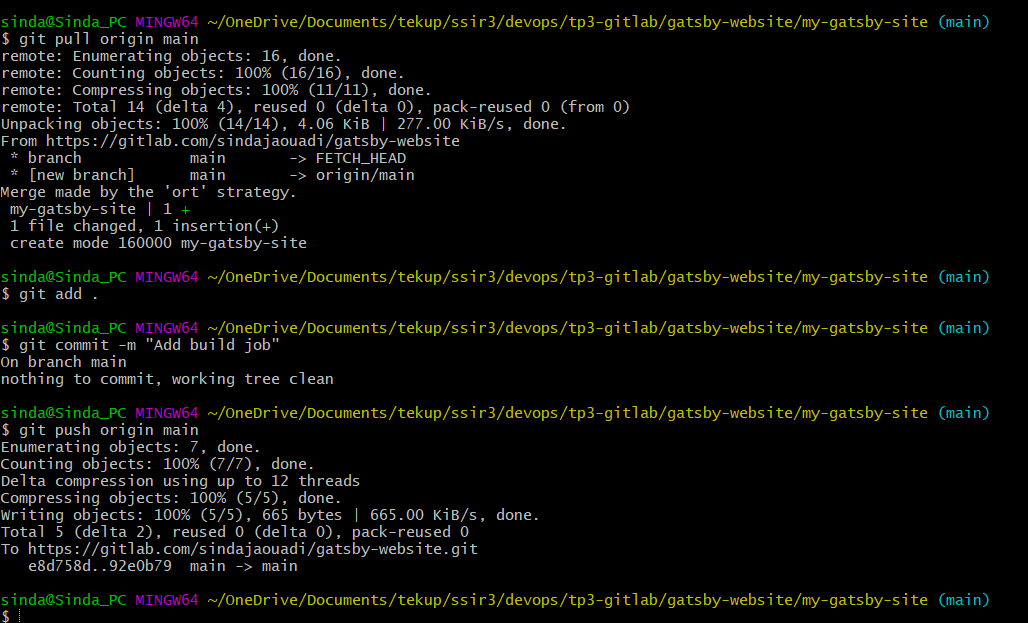
**Étape 3 : Configuration de la Construction du Site (Job Build)**

* À la racine du projet, créer un fichier .gitlab-ci.yml. Ce fichier définira les étapes du pipeline CI/CD.
* Modifier .gitlab-ci.yml pour inclure un job de construction



* **npm install** : Installe les dépendances du projet.
* **gatsby build** : Génère les fichiers statiques dans le dossier public.
* **Artefacts** : Les fichiers générés dans public sont sauvegardés pour être utilisés dans les étapes suivantes.

Sauvegarder les changements et pousse-les sur gitlab

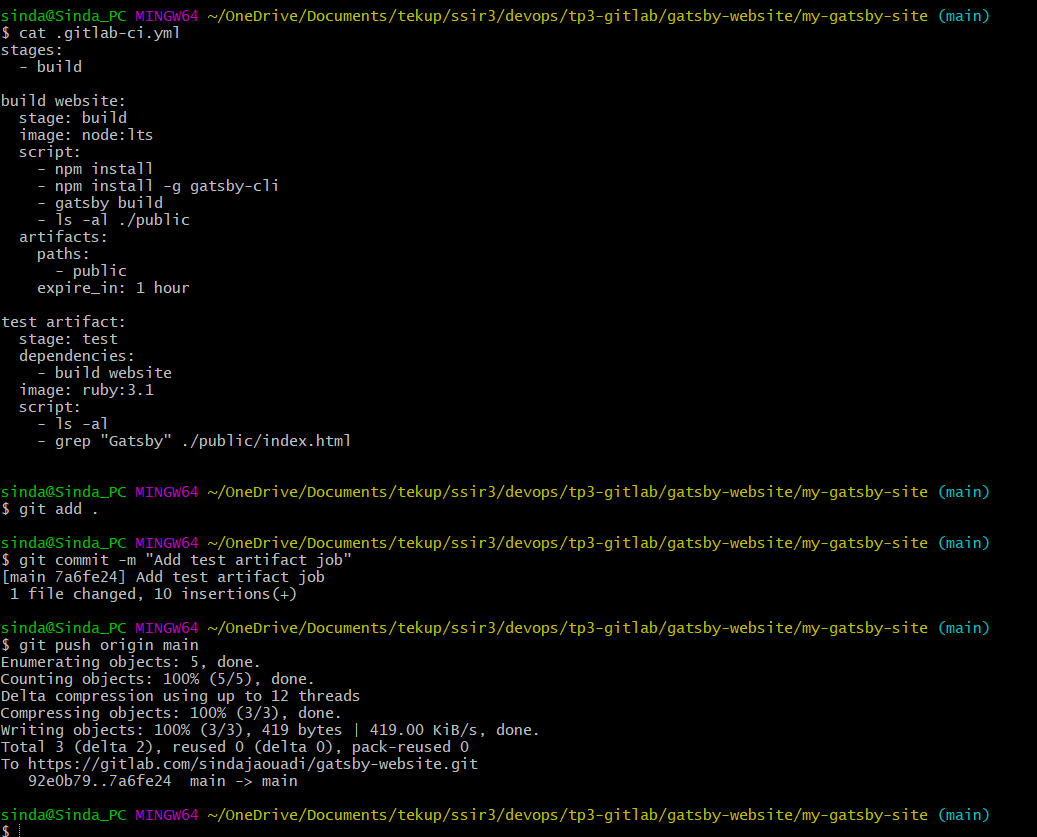
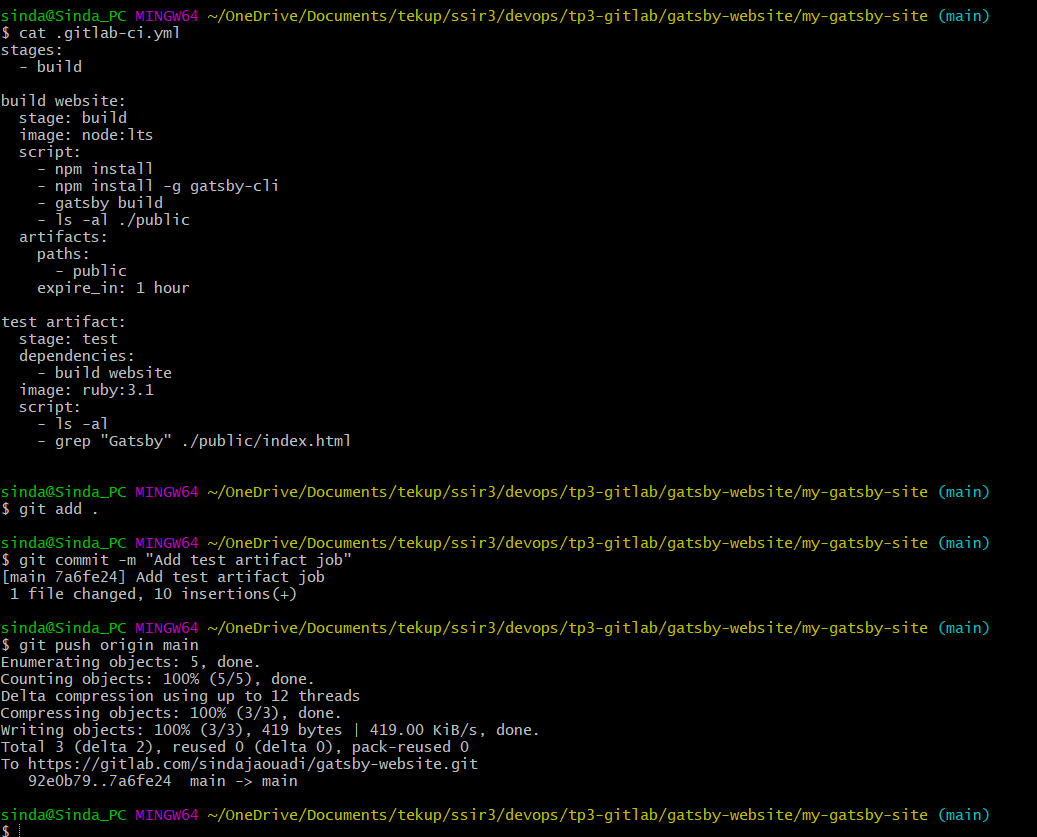


**Étape 4 : Vérification des Artefacts (Job Test Artifacts)**

Modifiez .gitlab-ci.yml pour ajouter un job qui teste les fichiers générés.

* **dependencies** : Télécharge les artefacts générés par le job build website.
* **grep** : Vérifie que le fichier index.html contient "Gatsby"

Après avoir modifié `.gitlab-ci.yml`, pousser à nouveau sur gitlab

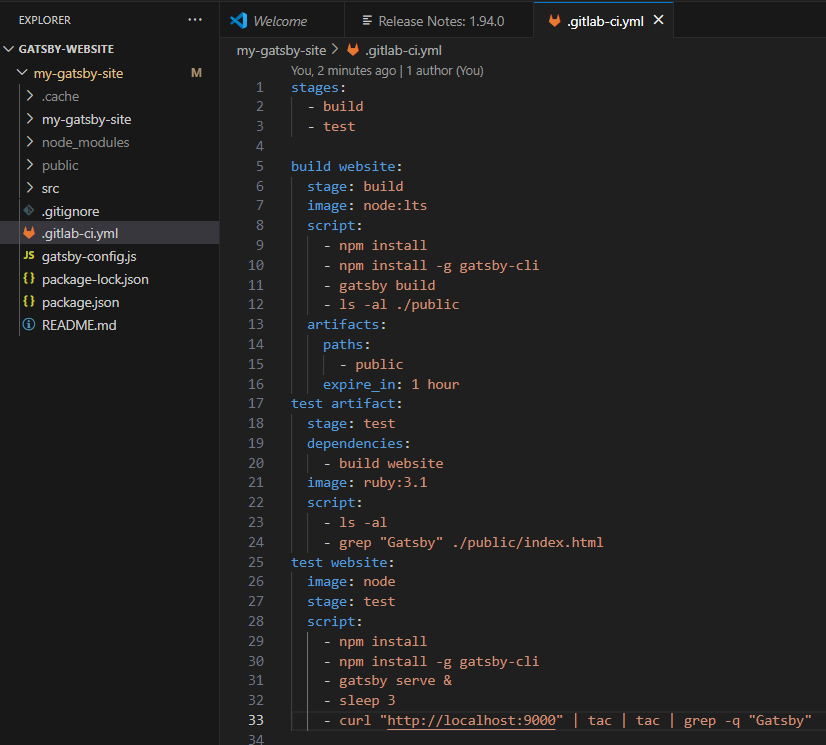


**Étape 6 : Test du Site Localement (Job Test website)**

Ajouter un job qui démarre le serveur Gatsby localement et vérifier que le site s’affiche correctement

* **gatsby serve &** : Lance le serveur en arrière-plan.
* **curl** : Vérifie que le contenu de la page contient "Gatsby"

Ajouter ces modifications et pousser-les vers GitLab



**Étape 6 : Déploiement du Site (Job Deploy)**

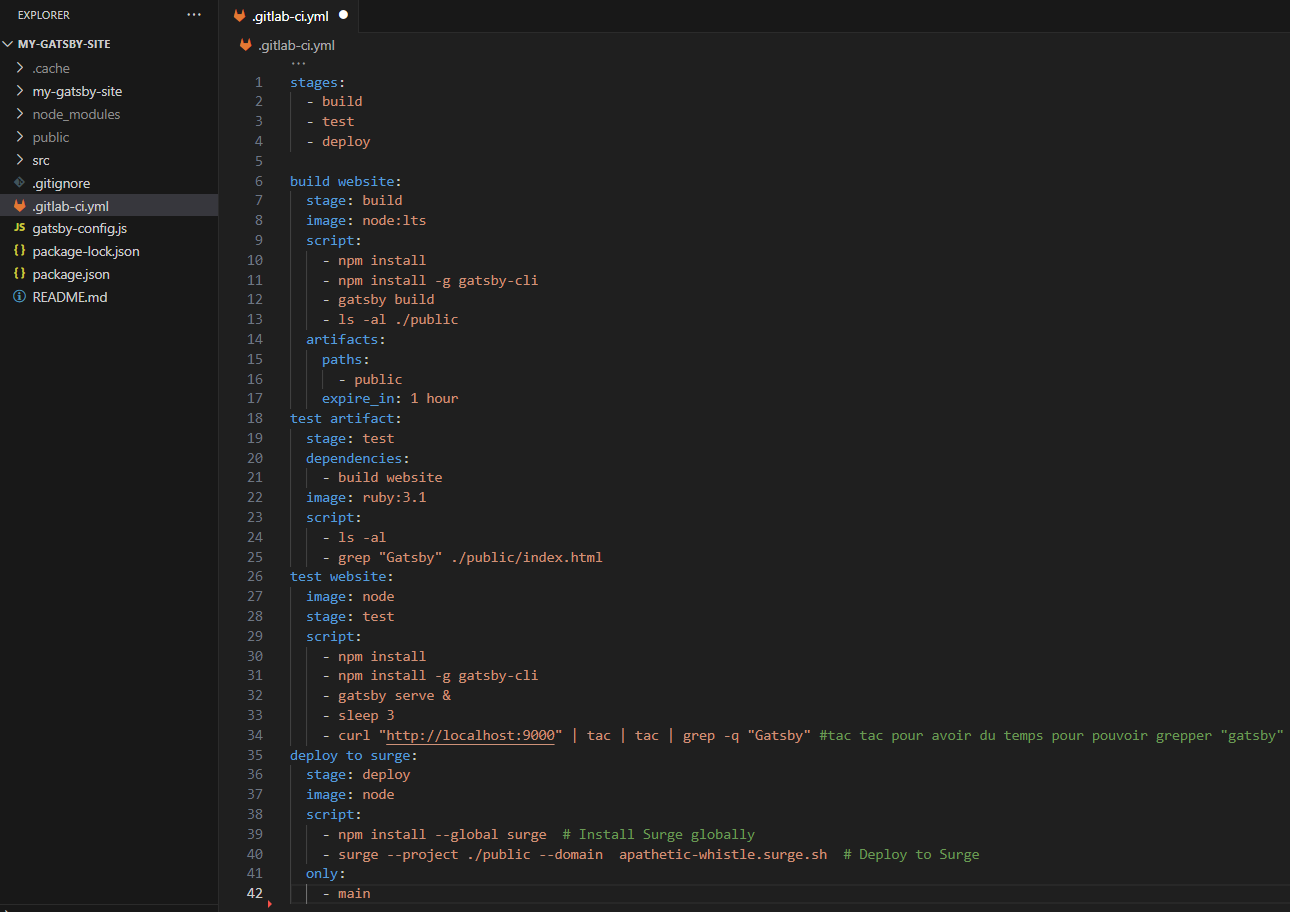
Installer surge et utiliser la commande ***surge***  pour avoir un nom de domaine.

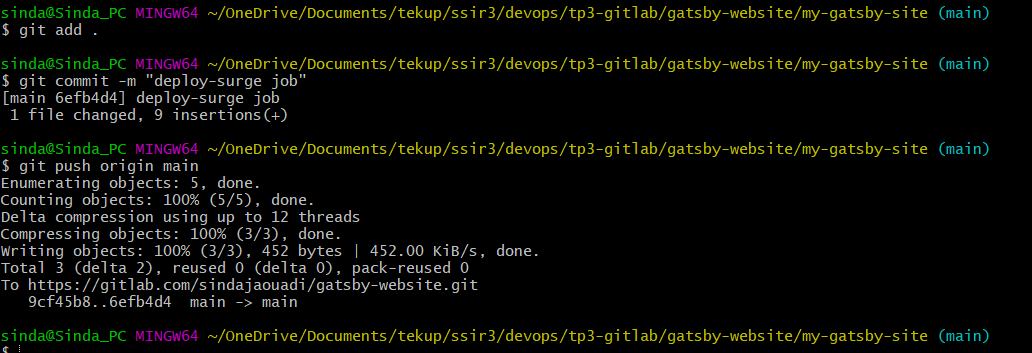
Ajoutez un job pour déployer le site sur Surge.

* **surge** : Commande pour déployer le site sur Surge avec le nom de domaine.

Poussez les modifications finales vers GitLab.



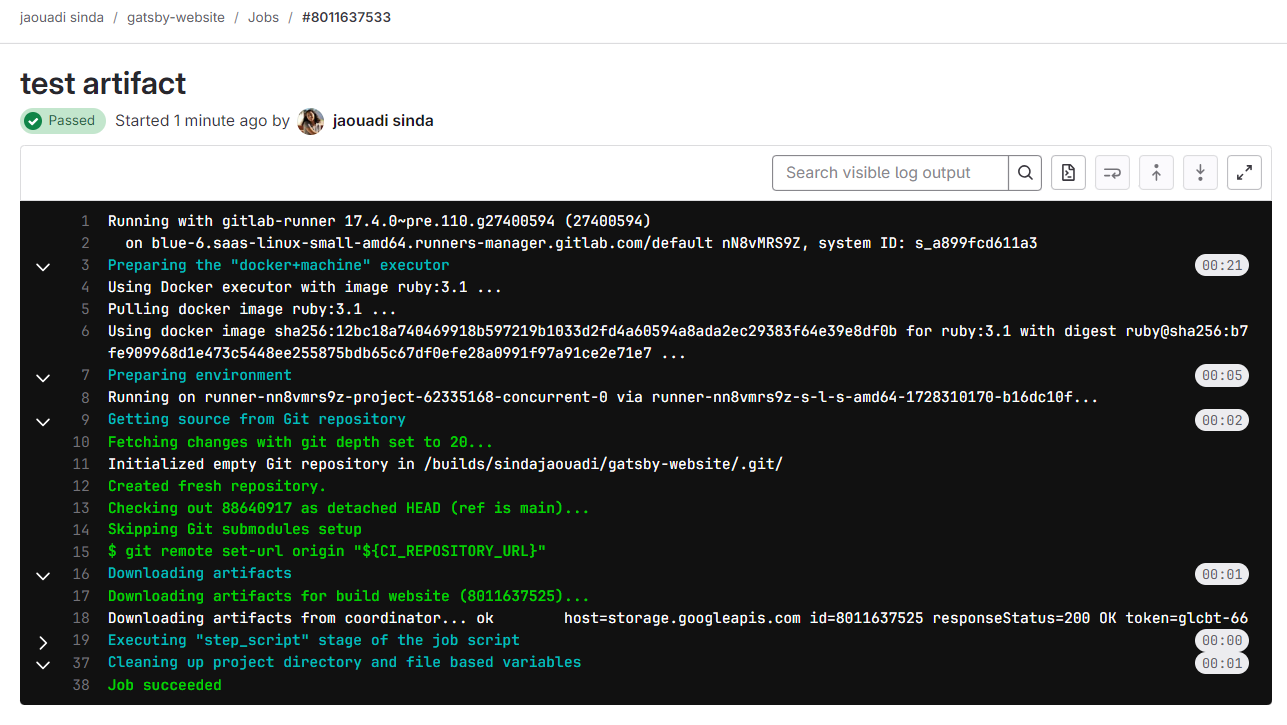
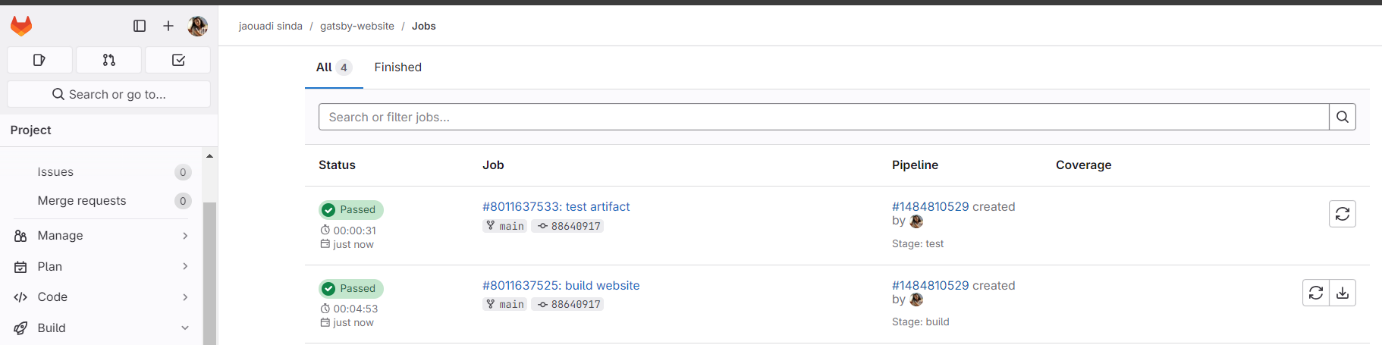


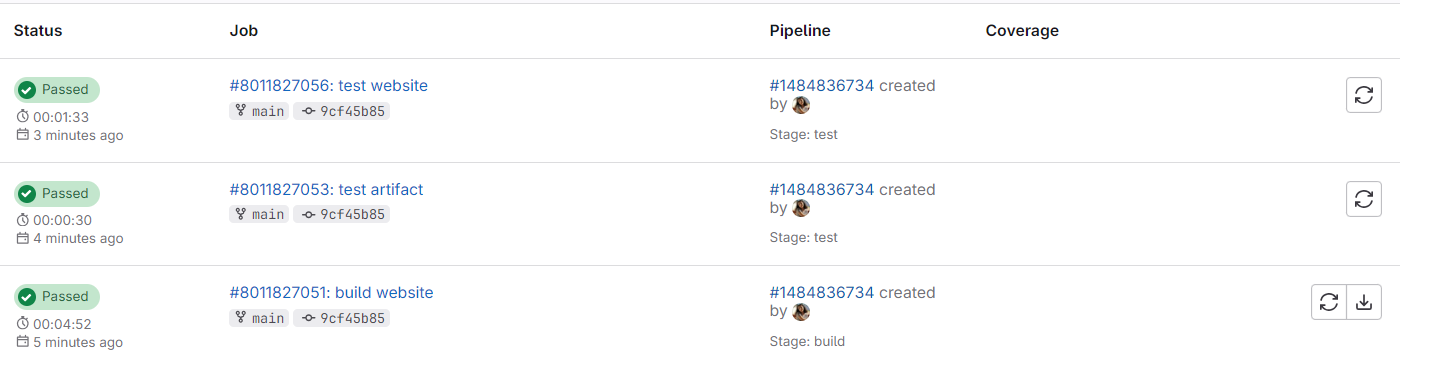


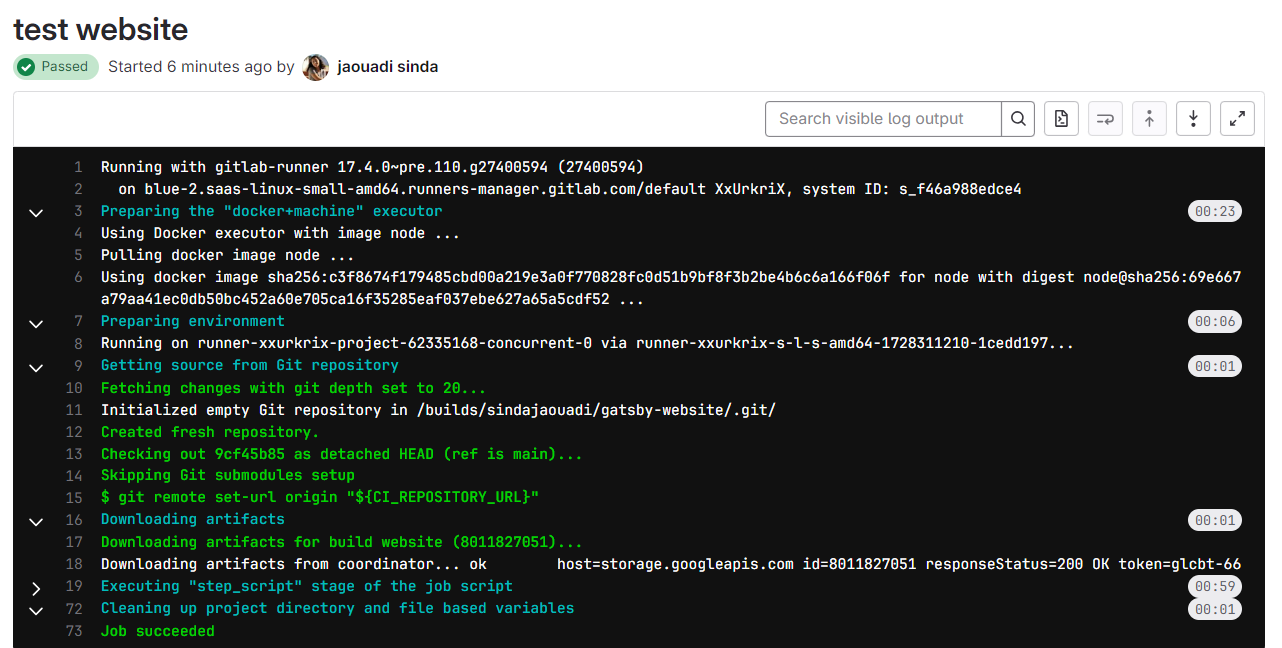
**Étape 7 : Vérification du Pipeline**

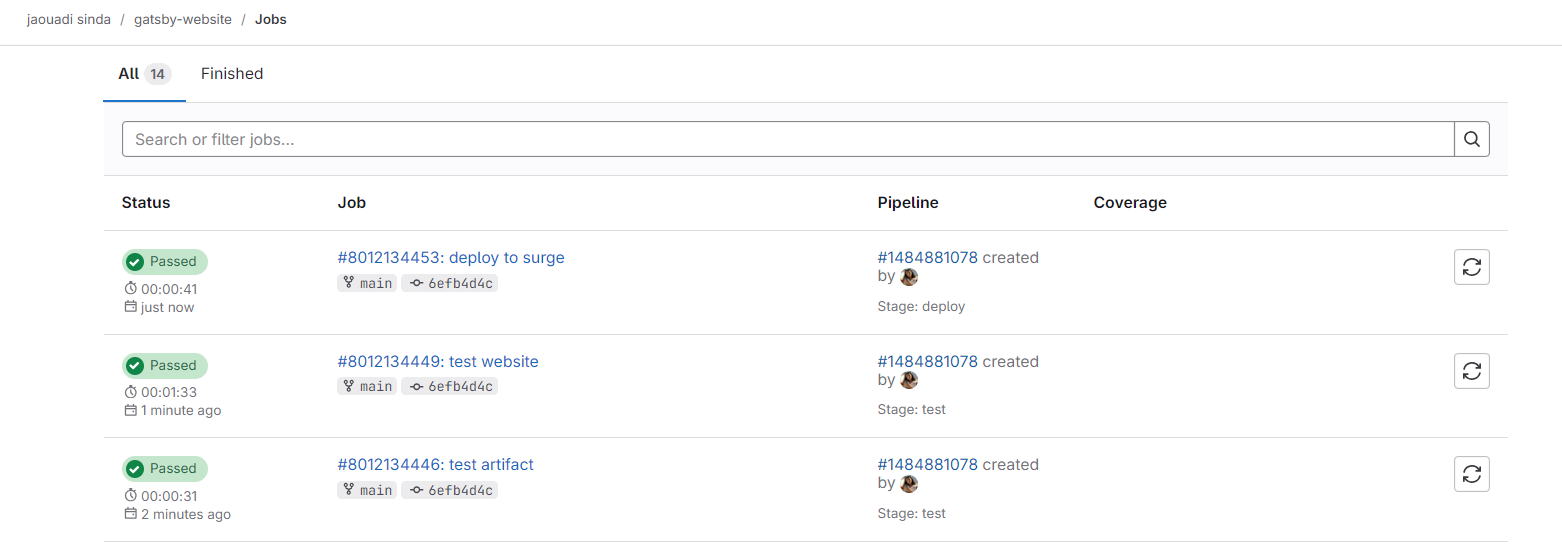
Sur GitLab :

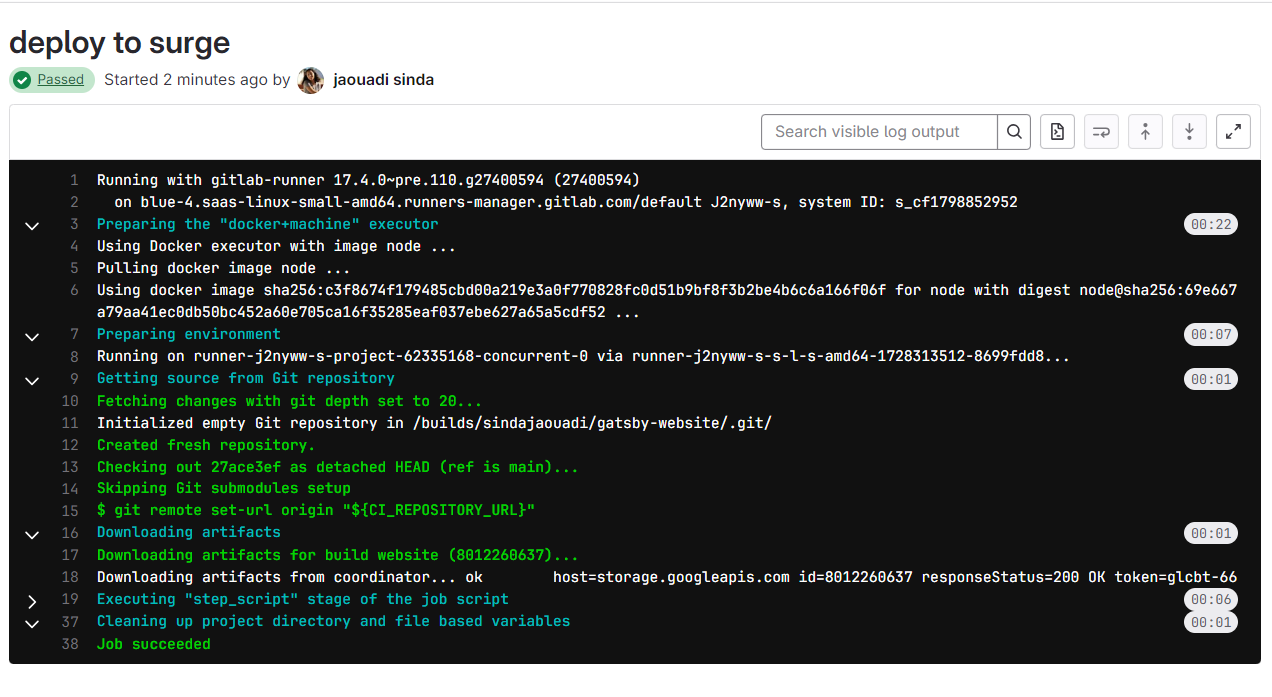
* Accédez à l’onglet CI/CD > Pipelines.
* Vérifiez que chaque étape (build, test, deploy) s’exécute correctement.











**Questions Guidées :**

1. Le job build est responsable de générer le site statique à partir des fichiers sources du projet Gatsby. Il utilise les commandes suivantes :

* **npm install** : Installe les dépendances définies dans le fichier package.json.
* **gatsby build** : Transforme les fichiers source (HTML, JavaScript, CSS) en une version optimisée et prête à être déployée.

Les fichiers statiques résultants (HTML, CSS, JS, images) sont placés dans le dossier ./public.

1. Les artefacts sont utilisés pour conserver les fichiers générés à l’étape build (par exemple, le dossier ./public) et les rendre accessibles aux autres étapes du pipeline comme les étapes de test ou de déploiement. Ils permettent de transférer efficacement les fichiers entre les jobs sans avoir à régénérer les résultats ou à copier manuellement les fichiers. Cela garantit la cohérence et optimise le temps d’exécution du pipeline.
2. **Vérifications dans test artifact** :

* La commande **ls -al** vérifie que le dossier ./public et ses fichiers existent.
* La commande **grep "Gatsby" ./public/index.html** s’assure que le fichier index.html contient bien un contenu attendu (ici, une référence à "Gatsby").

**Vérifications dans test website** :

* Le job démarre un serveur local avec **gatsby serve** et teste l’accessibilité du site via **curl**, en recherchant également le texte "Gatsby".

Ces tests garantissent que le site est bien construit et fonctionnel, et qu'il respecte les attentes en termes de contenu et de structure.

1. **Déploiement sur Surge** :

Le job deploy to surge installe l’outil Surge globalement via **npm install --global surge**.

Ensuite, la commande **surge --project ./public --domain apathetic-whistle.surge.sh** déploie les fichiers statiques du dossier ./public sur la plateforme Surge, rendant le site accessible à l’URL spécifiée.

**Comparaison avec d’autres solutions** :

* **Surge** : Idéal pour des sites statiques, facile à configurer, avec une URL personnalisée gratuite.
* **GitLab Pages** : Intégré directement à GitLab, mais nécessite un fichier gitlab-ci.yml plus complexe pour configurer les pages.
* **Netlify** : Offre un support avancé pour les sites statiques avec des fonctionnalités comme les builds automatiques, les formulaires, et les déploiements instantanés, mais peut nécessiter un compte pour une intégration complète.

**Conclusion** : Surge est simple et rapide pour des projets comme celui-ci, mais des alternatives comme Netlify ou GitLab Pages peuvent être plus adaptées pour des sites avec des besoins spécifiques ou une intégration profonde avec CI/CD.