Configuration du Prototype d’implémentation des tests d’accessibilité dans Azure Devops

Skills Evolution Program, CentraleSupélec, Microsoft

Jérôme Cornier – jcornier@microsoft.com

Avril 2019

Résumé

Les solutions numériques se multiplient et parfois sont le seul média permettant d’accéder à certaines informations ce qui rend nécessaire la prise en compte de l’accessibilité de ces solutions. La législation a évolué dans ce sens, rendant obligatoire la conformité des sites Web publics aux standards d’accessibilité depuis Septembre 2018.

La conjugaison des exigences d’accessibilité plus importantes et du nombre croissant de solutions amène à réfléchir sur l’automatisation des tests d’accessibilité dans un contexte DevOps pour améliorer en continue la qualité des produits.

Ce document accompagne un mémoire portant sur l’implémentation des pratiques inclusives dans un continuous delivery pipeline. Il décrit la mise en œuvre d’un prototype Azure DevOps utilisant un agent spécifique d’automatisation des tests d’accessibilité basés sur l’API de Deque University.

Table des matières

[1 Introduction 5](#_Toc6564391)

[1.1 Prérequis 5](#_Toc6564392)

[2 Web App Classique - Préparation de l’environnement 6](#_Toc6564393)

[2.1 Architecture générale 6](#_Toc6564394)

[2.1.1 Architecture Azure de la Web App Windows Classique 6](#_Toc6564395)

[2.1.2 Architecture du pipeline pour la Web App Classique 7](#_Toc6564396)

[2.2 Création de l’application Web et publication dans GitHub 8](#_Toc6564397)

[2.2.1 Création de l’application dans Visual Studio 8](#_Toc6564398)

[2.2.2 Publication de l’application dans GitHub 10](#_Toc6564399)

[2.2.3 Ajout d’une page Web dans l’application 12](#_Toc6564400)

[2.2.4 Mise à jour dans GitHub 15](#_Toc6564401)

[2.3 Création de l’environnement Azure 16](#_Toc6564402)

[2.4 Création et configuration de l’organisation Azure DevOps 21](#_Toc6564403)

[2.4.1 Création d’une nouvelle organisation DevOps 22](#_Toc6564404)

[2.4.2 Création d’un nouveau Projet DevOps 22](#_Toc6564405)

[2.4.3 Création du pipeline « Builds » 23](#_Toc6564406)

[2.4.4 Création du pipeline « Release » 26](#_Toc6564407)

[3 Création de l’Agent Azure DevOps spécifique A11y 31](#_Toc6564408)

[3.1 Architecture Générale 31](#_Toc6564409)

[3.1.1 Architecture Azure 31](#_Toc6564410)

[3.1.2 Architecture de la machine virtuelle de l’Agent A11y 32](#_Toc6564411)

[3.2 Création de la VM Azure pour l’Agent DevOps 32](#_Toc6564412)

[3.3 Installation des dépendances logicielles 37](#_Toc6564413)

[3.3.1 Configuration Internet 38](#_Toc6564414)

[3.3.2 Installation de Firefox 38](#_Toc6564415)

[3.3.3 Installation du pilote Gecko 40](#_Toc6564416)

[3.4 Installation de l’agent DevOps 42](#_Toc6564417)

[4 Implémentation de l’Agent DevOps A11y dans le pipeline 47](#_Toc6564418)

[4.1 Implémentation de l’Agent dans le pipeline « Release » 47](#_Toc6564419)

[4.2 Déploiement de l’application Web « Production » 51](#_Toc6564420)

[4.3 Pipeline final 53](#_Toc6564421)

[5 Démonstration des tests d’accessibilité automatisés 54](#_Toc6564422)

[5.1 Première Release en échec avec la Web App initiale 54](#_Toc6564423)

[5.1.1 Erreur n°1, paramètre de langue du document manquant 55](#_Toc6564424)

[5.1.2 Erreur n°2, texte alternatif manquent sur une image 55](#_Toc6564425)

[5.2 Mise en conformité du code d’après le rapport de tests 56](#_Toc6564426)

[5.2.1 Correction du paramètre de langue 56](#_Toc6564427)

[5.2.2 Correction du texte alternatif de l’image 56](#_Toc6564428)

[5.2.3 Synchronisation des modifications avec GitHub 56](#_Toc6564429)

[5.3 Release réussie après correction 57](#_Toc6564430)

[5.4 Elévation du niveau d’accessibilité WCAG 2.0 A + AA 57](#_Toc6564431)

[5.4.1 Identification des défauts de conformité 59](#_Toc6564432)

[5.4.2 Remédiation pour le niveau A + AA 59](#_Toc6564433)

[5.4.3 Résultat avant / après 61](#_Toc6564434)

[6 Web App en Container – Préparation de l’environnement 63](#_Toc6564435)

[6.1 Architecture générale 63](#_Toc6564436)

[6.1.1 Architecture de la Web App Linux en Container 64](#_Toc6564437)

[6.1.2 Architecture du pipeline pour le container Docker 64](#_Toc6564438)

[6.2 Ajout de Docker à l’environnement de développement 65](#_Toc6564439)

[6.2.1 Installation de Docker sur le poste de développement 65](#_Toc6564440)

[6.2.2 Ajout du support ASP.NET et Docker dans Visual Studio 67](#_Toc6564441)

[6.3 Création de l’application Web en container et publication dans GitHub 68](#_Toc6564442)

[6.3.1 Création de l’application dans Visual Studio 68](#_Toc6564443)

[6.3.2 Modifier le fichier Dockerfile 70](#_Toc6564444)

[6.3.3 Test de l’application Docker en local 71](#_Toc6564445)

[6.3.4 Publication de l’application dans GitHub 72](#_Toc6564446)

[6.3.5 Ajout d’une page Web dans l’application 74](#_Toc6564447)

[6.3.6 Mise à jour dans GitHub 77](#_Toc6564448)

[6.4 Création de l’environnement Azure pour la Web App en Container 78](#_Toc6564449)

[6.4.1 Création du référentiel de containers 78](#_Toc6564450)

[6.4.2 Création des Web App for Containers 79](#_Toc6564451)

[6.5 Création du pipeline DevOps 83](#_Toc6564452)

[6.5.1 Création d’un nouveau Projet DevOps 83](#_Toc6564453)

[6.5.2 Création du pipeline « Builds » DevOps (YAML) 84](#_Toc6564454)

[6.5.3 Modification du fichier YAML 85](#_Toc6564455)

[6.5.4 Configuration des Web App for Containers Azure 87](#_Toc6564456)

[6.5.5 Création du pipeline « Release » du Container 89](#_Toc6564457)

[6.6 Implémentation des tests d’accessibilité automatisés 92](#_Toc6564458)

[6.7 Pipeline final 93](#_Toc6564459)

[6.7.1 CI 93](#_Toc6564460)

[6.7.2 CD 93](#_Toc6564461)

# Introduction

Ce document décrit les étapes pas à pas nécessaires pour la mise en œuvre d’une implémentation de tests d’accessibilité dans un pipeline DevOps pour une application Web.

L’objectif étant de se rapprocher des pratiques de développement actuelles l’implémentation est réalisée sur deux pipelines différents :

* Une Web App classique
* Une Web App en Container Docker

Les tests de conformité permettent aussi de montrer comment corriger le code ASP.NET de l’application pour passer les tests et publier le site Web en production.

## Prérequis

Les éléments suivants sont nécessaires pour la réalisation de ce prototype.

* PC Windows 10 ou Windows Serveur
* Visual Studio 2017 ou 2019
* Souscription Azure (<http://portal.azure.com>)
* Azure DevOps active (<http://dev.azure.com>)
* Profile GitHub existent (<https://github.com>)

# Web App Classique - Préparation de l’environnement

Le premier prototype vise à construire et déployer en continue une application Web en mode PaaS, c’est-à-dire publiée par un service d’application Cloud, dans ce cas App Service Azure.

## Architecture générale

Pour cette première application nous utiliserons la convention de noms suivante.

|  |  |
| --- | --- |
| **Nom** | **Composant** |
| Application Web |  |
| WebAppA11yClassic | Solution Visual Studio |
| WebAppA11yClassic | Référentiel GitHub |
| Ressources Azure |  |
| WebAppA11yClassicRG | Resource group |
| WebAppA11yClassicStaging | App Service « Staging » |
| WebAppA11yClassicStagingPlan | App Service Plan « Staging » |
| WebAppA11yClassic | App Service « Production » |
| WebAppA11yClassicPlan | App Service Plan « Production » |
| Azure DevOps |  |
| A11yDemo | Organisation DevOps |
| WebAppA11yClassic-ASP.NET-CI | Pipeline « Builds » |
| WebAppA11yClassic-ASP.NET-CD | Pipeline « Release » |

### Architecture Azure de la Web App Windows Classique

L’application est très simple, nous nous sommes basés sur une application Web ASP.NET Core proposée en standard par Visual Studio.

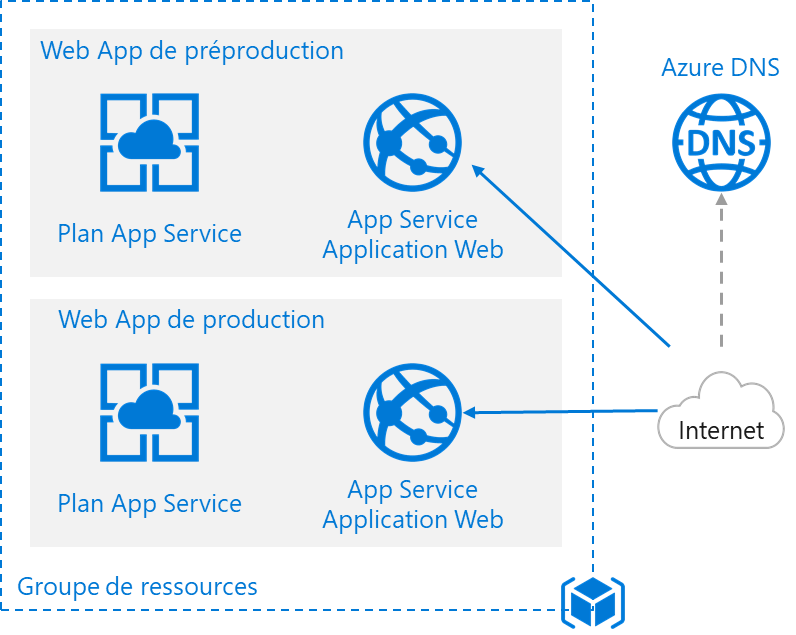


Figure : Architecture de la Web App

Elle est stockée dans un référentiel GitHub. Elle est constituée de 3 pages : Home, Privacy, Accessibility.

### Architecture du pipeline pour la Web App Classique

Ce pipeline doit s’activer automatiquement à chaque modification confirmée par un « commit » dans GitHub. Le pipeline débute par l’intégration continue (CI) et le déploiement continue (CD) pour construire et déposer l’application dans Azure.

La seconde partie est celle qui va contenir les tests d’accessibilité. Le pipeline de livraison « Release » publie l’application dans un premier Service Web App Azure de préproduction « Staging » puis exécute une série de tests d’accessibilité.

Si les tests sont validés alors le pipeline termine par la publication de l’application dans un second Service Web App de production.

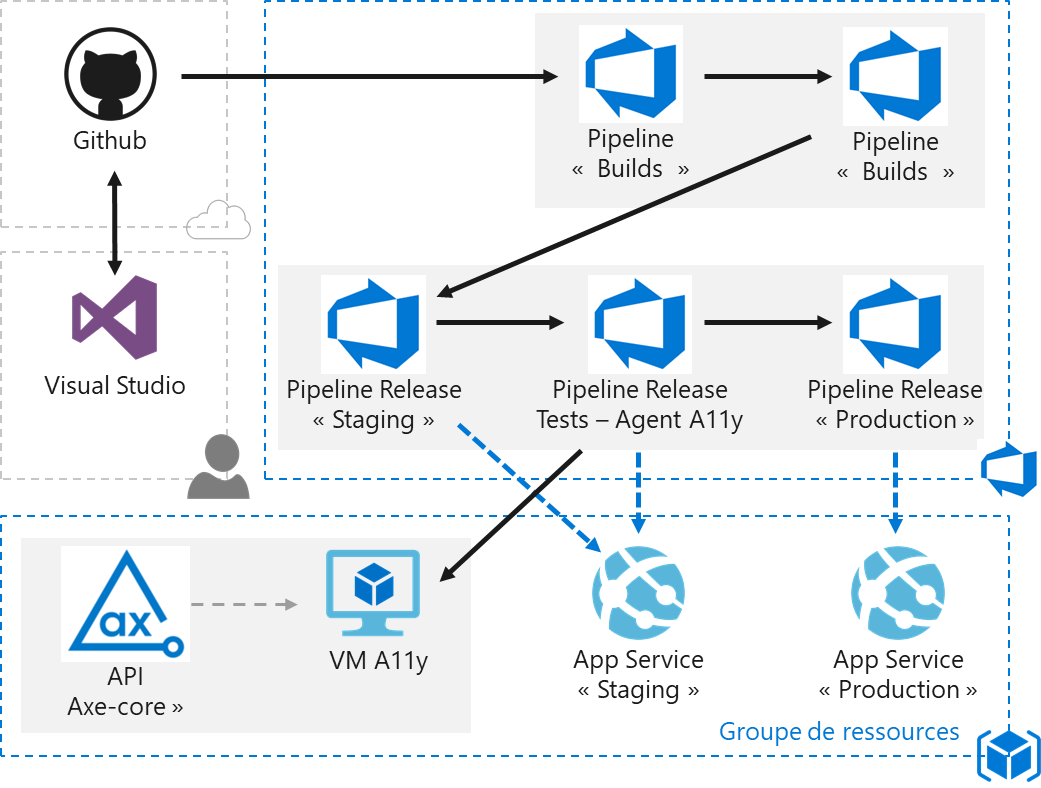


Figure : pipeline de livraison continue pour la Web App

Les modifications effectuées dans Visual Studio peuvent être poussées dans GitHub et inversement.

## Création de l’application Web et publication dans GitHub

Pour cette préparation nous procéderons en 4 étapes :

* Création de l’application dans Visual Studio
* Publication dans GitHub
* Création d’une nouvelle page dans le projet de l’application
* Contrôle de mise à jour dans GitHub

### Création de l’application dans Visual Studio

|  |  |
| --- | --- |
| **Illustration** | **Action** |
|  | Lancer Visual Studio  Créer un nouveau projet |
|  | Sélectionner .NET Core |
|  | Sélectionner Application web ASP.NET Core |
|  | Saisir le nom de l’application  Ne nom de la solution (le même)  Choisir le répertoire |
|  | Cocher Créer un répertoire pour la solution  Ne pas cocher Créer un dépôt Git (nous utiliserons GitHub)  Valider ces choix par OK |
|  | Vérifier que Application web est sélectionné  Décocher HTTPS  Ne pas cocher Docker  OK |
|  | Le projet apparaît dans Visual Studio / Explorateur de solutions |

### Publication de l’application dans GitHub

|  |  |
| --- | --- |
| **Illustration** | **Action** |
|  | Cliquer en bas à droite sur Ajouter au contrôle de code source  Cliquer sur Git |
|  | Cliquer sur Publier sur GitHub  ATTENTION Azure DevOps est un autre dépôt. Nous voulons sélectionner GitHub |
|  | Vérifier ou sélectionner le compte/profile GitHub  Saisir le nom de la Web App  Saisir une courte description (modifiable ultérieurement)  Cocher Référentiel privé  Publier |
|  | Le dépôt a été créé |
|  | Ouvrir <https://github.com>  Cliquer sur le nom de la Web App |
|  | Les fichiers sont présents |

### Ajout d’une page Web dans l’application

|  |  |
| --- | --- |
| **Illustration** | **Action** |
|  | Revenir dans Visual Studio, Explorateur de solutions  Déployer Pages et Shared |
|  | Cliquer bouton droit sur Privacy.cshtml  Copier  Cliquer bouton droit sur Pages  Coller  Le nouveau fichier Privacy-Copier apparaît |
|  | Renommer en Accessibility.cshtml |
|  | Dans Shared, cliquer sur \_Layout.cshtml  Le contenu s’ouvre dans la fenêtre centrale |
|  | Sélectionner et copier les 3 lignes correspondant à Privacy  (Ctrl-C, Ctrl-V, Ctrl-V) |
|  | Renommer Privacy en Accessibility sur la nouvelle ligne  Enregistrer (Ctrl-S) |
|  | Sélectionner Accessibility.cshtml |
|  | Remplacer Privacy par Accessibility  Enregistrer |
|  | Déplier Accessibility et sélectionner Accessibility.cshtml.cs |
|  | Renommer PrivacyModel en AccessibilityModel  Enregistrer |
|  | Cliquer bouton droit sur la Solution  Générer la solution |
| ========== Génération : 1 a réussi, 0 a échoué, 0 mis à jour, 0 a été ignoré ========== | La solution devrait se construire sans erreur |
|  | Cliquer sur IIS Express |
|  | La page Web s’ouvre  Tester la navigation et la page d’accessibilité |
|  | Stopper en cliquant sur la carré rouge |
|  | Cliquer bouton droit sur wwwroot  Ajouter  Nouveau dossier  Saisir le nom « images »  Valider |
|  | Enregistrer l’image ci-contre dans un fichier jpeg sous le nom  UN\_Logo\_Accessibility.jpg |
|  | Glisser et déposer le fichier image dans le dossier images |
|  | Cliquer sur Accessibility.cshtml |

Ajouter le logo dans le code qui doit ressembler à celui-ci-dessous. Il manque le alt=« xxx » pour l’image, c’est volontaire pour les tests d’accessibilité.

@page

@model AccessibilityModel

@{

ViewData["Title"] = "Declaration of Accessibility";

}

<div class="text-center">

<h1>@ViewData["Title"]</h1>

<p>This Demo Web Site is targeted to reach the Standard WCAG 2.0 A + AA</p>

<a href="https://www.un.org/fr/webaccessibility/logo.shtml">

<img src="~/images/UN\_Logo\_Accessibility.jpg" width="300" />

</a>

</div>

|  |  |
| --- | --- |
|  | Cliquer à nouveau sur IIS Express  Vérifier le contenu de la nouvelle page |

### Mise à jour dans GitHub

|  |  |
| --- | --- |
| **Illustration** | **Action** |
|  | Cliquer sur le stylo en bas à droite  (Autre solution : dans Team Explorer / Modifications) |
|  | Entrer une description correspondant aux modifications effectuées  Vérifier la liste des fichiers qui correspond à ceux modifiés précédemment  Cliquer sur Valider tout |
|  | Cliquer sur Synchroniser |
|  | Cliquer sur Pousser |
|  | La synchronisation a été effectuée |
|  | Ouvrir <https://github.com>  Naviguer dans Pages  Les nouveaux fichiers Accessibility doivent apparaître  Idem pour le logo dans wwwroot/images |
|  | Cliquer sur Commit  Le dernier push apparaît, après la création du projet et l’initialisation du dépôt |

## Création de l’environnement Azure

L’étape suivante consiste à préparer l’environnement Cloud pour recevoir l’application.

A partir du portail Azure (<http://portal.azure.com>) nous devons créer 2 Services d’application Web App avec leurs plans de services associés : la première Web App servira de préproduction « Staging » pour tester l’accessibilité et la seconde sera déployée en production si les tests sont réussis.

|  |  |
| --- | --- |
| **Illustration** | **Action** |
|  | Créer une nouvelle ressource App Services |
|  | Add  Saisir le nom de la Web App  Choisir la souscription (si plusieurs)  Créer en même temps le Resource Group en ajoutant « RG » à la fin du nom proposé |
|  | Cliquer sur « App Service plan/Location  Créer un nouveau plan |
|  | Saisir le nom suivi de « Plan »  Choisir l’emplacement (« West Europe » dans notre exemple)  Cliquer sur Pricing tier pour changer le Plan par défaut en F1 |
|  | Cliquer sur Dev / Test  Cliquer sur F1  Apply |
|  | Vérifier, puis OK |
|  | Cliquer sur Application Insights |
|  | Disable dans notre cas (pas utilisé dans le prototype actuel)  Apply |
|  | Vérifier, puis Créer |
|  | Patienter quelques secondes |
|  | Recommencer la création du second service pour le « Staging »  Add  Saisir le nom de la Web App avec « Staging » à la fin  ATTENTION : Use existing Resource Group et sélectionner le RG créé précédemment |
|  | Même opérations que précédemment pour le Service Plan et Application Insights  Create |
|  | Les deux App Services sont créés  Clique pour chacun d’eux sur « Pin to Dashboard » pour les ajouter à la page d’accueil |
|  | Les deux App Services sont actifs  Cliquer sur l’un ou l’autre pour vérifier |
|  | Cliquer sur l’URL qui apparaît |
|  | Une page Web s’ouvre confirmant le bon fonctionnement Azure |

## Création et configuration de l’organisation Azure DevOps

Le pipeline de livraison continue est ensuite créé en utilisant Azure DevOps (<http://dev.azure.com>).

L’opération nécessite plusieurs étapes :

* Création d’une organisation DevOps
* Création d’un projet
* Création d’un repository DevOps à partir de Github dans notre contexte
* Création du pipeline « Builds »
* Création du pipeline de livraison « Release »

Les étapes du pipeline sont les plus intéressantes pour ce prototypage, les précédentes sont juste des étapes de configuration basiques.

### Création d’une nouvelle organisation DevOps

|  |  |
| --- | --- |
| **Illustration** | **Action** |
|  | Nouvelle organisation |
|  | Continue |
|  | Saisir le nom de l’organisation et vérifier la localisation  Continue |

### Création d’un nouveau Projet DevOps

|  |  |
| --- | --- |
| **Illustration** | **Action** |
|  | Entrer le nom de la Web App  Entrer une description  Choisir Private  Create project |

### Création du pipeline « Builds »

« Builds » signifie Continuous Integration et Continuous Deployment. Cette partie du pipeline permet de construire et envoyer la solution dans Azure mais le Service d’Application n’est pas mis à jour.

Il existe deux manières de créer le pipeline « Builds » : soit en utilisant les briques de DevOps, soit en fournissant un fichier YAML (Yet Another Markup Language) qui correspond à un script d’installation.

Pour cette Web App nous avons choisi les briques DevOps qui illustrent mieux le fonctionnement. Nous l’avons simplifié en ôtant certains tests qui ne s’appliquent pas à notre application.

|  |  |
| --- | --- |
| **Illustration** | **Action** |
|  | Cliquer sur Pipelines |
|  | New pipeline |
|  | Cliquer sur Use the classic editor |
|  | Cliquer sur GitHub |
|  | Cpoier l’URL du dépôt GitHub  Coller dans le champ Connexion name  Cliquer sur Authorize using OAuth  Saisir les credentials si besoin |
|  | Sélectionner le dépôt de la Web App  Branch master par défaut  Continue |
|  | Sélectionner le template Azure Web App for ASP.NET  Apply  (le bouton Apply apparaît à droite de la ligne où se trouve la souris) |
|  | Mettre à jour le nom  Sélectionner la souscription Azure  Cliquer sur Authorize  Saisir les credentials dans la fenêtre qui s’affiche |
|  | Rafraîchir si besoin avec les boutons à droite des champs  Sélectionner la Web App créé précédemment |
|  | Cliquer sur VsTest – testAssemblies |
|  | Parcourir la liste des paramètres  Déplier Control Options  Décocher Enabled  Ce test n’est pas compatible avec GitHub |
|  | Même opération pour Publish symbols path |
|  | Save & queue |
|  | Cliquer sur le numéro de Build |
|  | Surveiller les tâches  Le pipeline complet dure plusieurs minutes |

### Création du pipeline « Release »

Le pipeline de livraison peut être paramétré pour démarrer dès que le pipeline « Builds » a terminé son déploiement, c’est ce que nous avons configuré.

L’étape suivante consiste à mettre en place le pipeline de déploiement de la Web App.

|  |  |
| --- | --- |
| **Illustration** | **Action** |
|  | Sélectionner Releases |
|  | New pipeline |
|  | Choisir Azure App Service deployment  Apply |
|  | Renommer en « Staging » |
|  | Cliquer sur Add an artifact |
|  | Choisir la Source (build pipeline) qui est le pipeline précédemment créé  Les autres champs sont automatiquement renseignés  Add |
|  | Cliquer sur l’icône de l’éclair dans le coin de la boite de l’Artifact |
|  | Cliquer sur l’interrupteur pour activer le déclenchement automatique du pipeline Release |
|  | Cliquer sur 1 job, 1 task |
|  | Sélectionner la souscription Azure  Sélectionner Web App on Windows  Sélectionner la Web App “Staging » |
|  | Positionner la souris sur New release pipeline  Cliquer sur le stylo |
|  | Changer le nom du pipeline |
|  | Save  OK |
|  | Cliquer sur All pipelines |
|  | Cliquer sur le nom du pipeline |
|  | Create release  Create |
|  | Cliquer sur le nom de la Release en cours |
|  | Patienter |
|  | Le déploiement devrait être réussi |
|  | Ouvrir le portail Azure  Sélectionner la Web App Staging |
|  | Cliquer sur son URL |
|  | La page du site publié dans Azure doit apparaître |

# Création de l’Agent Azure DevOps spécifique A11y

Nous avons retenu l’extension A11y Accessibility Testing développée par Drew Lewis [108] qui présente les avantages suivants :

* Cette extension s’appuie sur l’API libre axe-core développée et supportée par Deque University [109]
* L’API axe-core est sous licence MPL 2.0
* Elle a pour philosophie « zéro faux positifs »
* C’est une extension configurable pour un pipeline Azure DevOps
* Le principe est l’analyse du site en ligne, l’extension fonctionne quel que soit le code source et le type d’hébergement
* Standards supportés : WCAG Level A, WCAG Level AA, Section 508

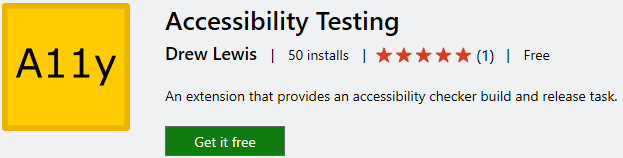


Figure 32 : A11y Extension Accessibility Testing, Drew Lewis

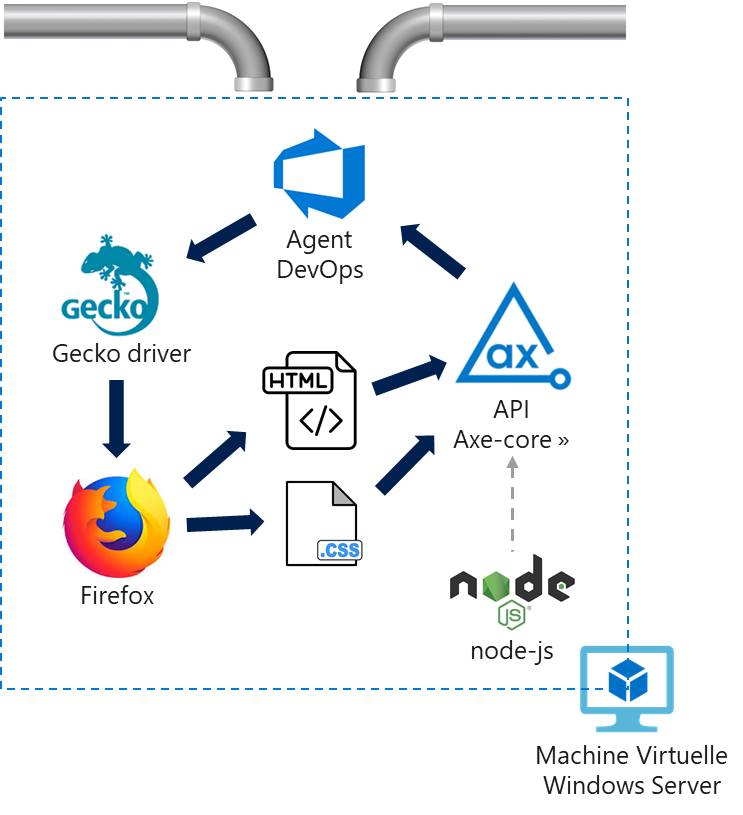
## Architecture Générale

### Architecture Azure

|  |  |
| --- | --- |
| **Nom** | **Composant** |
| Ressources Azure |  |
| VMAgentA11yRG | Resource group |
| VMAgentA11y | Virtual machine |
| VMAgentA11y\_DataDisk\_0 | Disk 0 – nom automatique |
| VMAgentA11y… | Disk 1 – nom automatique |
| Vmagenta11y… | Network interface – nom automatique |
| VMAgentA11y-ip | Public IP address – nom automatique |
| VMAgentA11y-nsg | Network security group – nom automatique |
| VMAgentA11yRG-vnet | Virtual network – nom automatique |

### Architecture de la machine virtuelle de l’Agent A11y

La solution est construite dans une Machine Virtuelle intégrée au pipeline CD.



## Création de la VM Azure pour l’Agent DevOps

Notre solution de tests d’accessibilité requiert plusieurs dépendances qui s’installent sur Windows. Nous avons donc choisi une machine virtuelle Azure préparée avec Windows Server 2016 Datacenter.

Le modèle de machine n’a pas besoin de puissance pour notre Web App, nous avons sélectionné le minimum auquel nous avons ajouté un disque SSD de 200 Go pour stocker les logiciels.

|  |  |
| --- | --- |
| **Illustration** | **Action** |
|  | Dans le portail Azure  <http://portal.azure.com>  Cliquer sur Resource groups  Add |
|  | Sélectionner la souscription et la localisation  Entrer le nom du resource group  Review + Create  Create |
|  | Attendre la confirmation |
|  | Cliquer sur Virtual machines  Add |
|  | Choisir la souscription  Entrer le nom de la machine  Choisir Windows Server 2016 Datacenter |
|  | Garder le DS1 v2 par défaut  Environ 42€ / mois |
|  | Saisir le nom de l’utilisateur Admin  Saisir 2 fois le mot de passe  Conserver ces credentials pour les prochaines étapes |
|  | Ajouter le port RDP 3389 |
|  | Next : Disks pour ajouter un disque persistent |
|  | Standard SSD est suffisant |
|  | Cliquer sur Create and attach a new disk |
|  | Standard SSD, 200Gb  OK |
|  | Review + create |
|  | Contrôler  Create |
|  | Patienter |
|  | Pin to dashboard |
|  | Vérifier les ressources crées |
|  | Go to resource |
|  | Cliquer sur Auto-shutdown |
|  | On  Choisir une heure d’arrêt automatique  Envoyer (ou pas) une notification  Saisir l’adresse email  Save |
|  | Cliquer sur Networking  Cliquer sur Public IP |
|  | Cliquer sur Configuration |
|  | Sélectionner Static  Save |
|  | Cliquer sur Dashboard |
|  | Cliquer sur le VMAgentA11y |
|  | Cliquer sur Connect |
|  | Cliquer sur Download RDP File  Enregistrer le fichier sur le Bureau |
|  | Double cliquer sur le fichier RDP  Cocher la case Ne pas me redemander…  Connexion |
|  | Cliquer sur Autres choix |
|  | Utiliser un autre compte |
|  | Entrer les credentials fournies précédemment lors de la création de la VM  OK |
|  | Cocher la case  Oui |
|  | Patienter le temps du chargement complet de la VM |

## Installation des dépendances logicielles

Les dépendances logicielles sont :

* L’Agent DevOps pour Windows qu’il faut associer au compte utilisateur pour DevOps [110]. Le lien permettant de télécharger l’agent est disponible dans les paramètres du projet dans DevOps ainsi que les commandes d’installation
* Navigateur Firefox pour Windows
* Gecko driver
* Node.js qui est inclus par défaut dans la VM Azure

### Configuration Internet

|  |  |
| --- | --- |
| **Illustration** | **Action** |
|  | Lancer Internet Explorer  Valider la recommandation  Fermer Internet Explorer |
|  | Ouvrir Server Manager  Cliquer sur Local Server |
|  | Cliquer sur On |
|  | Sélectionner Off  OK |

### Installation de Firefox

|  |  |
| --- | --- |
| **Illustration** | **Action** |
|  | Ouvrir Internet Explorer  Taper Firefox  Ouvrir la page Web de Firefox  Cliquer sur Download Now  Run |
|  | Patienter |
|  | Firefox d’ouvre  Cliquer sur le menu (3 barres en haut à droite)  Cliquer sur Options |
|  | Cliquer sur Privacy & Security |
|  | Cocher Deleet cookies … |
|  | Sélectionner Never remember history  Restart Firefox now |
|  | Cliquer sur Use Firefox as my default browser |
|  | La fenêtre des paramètres s’ouvre  Sélectionner Firefox  Fermer la fenêtre  Fermer Firefox et IE |

### Installation du pilote Gecko

|  |  |
| --- | --- |
| **Illustration** | **Action** |
|  | Lancer Firefox  Taper l’URL  <https://github.com/mozilla/geckodriver/releases> |
|  | Parcourir la page Web jusqu’en bas  Cliquer sur le fichier  geckodriver-v0.24.0-win64.zip  Save file  Fermer Firefox |
|  | Ouvrir l’Explorateur Windows  Créer un répertoire bin dans C:\ |
|  | Ouvrir le fichier zip téléchargé dans Downloads  Extraire (copier / coller) le fichier geckodriver dans le répertoire c:\bin |
|  | Windows+Q  Taper env  Cliquer sur Edit the system environnement variables |
|  | Environnement Variables |
|  | Sélectionner Path  Edit |
|  | New  Ajouter C:\bin  OK - OK |

## Installation de l’agent DevOps

Référence :

<https://docs.microsoft.com/en-us/azure/devops/pipelines/agents/v2-windows?view=azure-devops>

|  |  |
| --- | --- |
| **Illustration** | **Action** |
|  | Retourner sur la machine principale, dans DevOps, cliquer sur l’icône du profile  Cliquer sur Security |
|  | New Token |
|  | Choisir un nom  Choisir une période de validité  Full access  Create |
|  | Copier et conserver le Token généré |
|  | Ouvrir la machine virtuelle de l’agent  Lancer Firefox  Taper l’adresse dev.azure.com  Cliquer sur sign in  Entrer les credentials |
|  | Organization settings |
|  | Agent pools |
|  | Download agent |
|  | Windows  X64  Download  Save file |
|  | Ouvrir l’explorateur de documents  Sélectionner Downloads  Cliquer droit sur le nouveau fichier téléchargé vsts-agent-win-x64xxx  Extract All |
|  | Entrer le répertoire de destination  Extract |
|  | Contrôler le répertoire c:\agent |
|  | Lancer une fenêtre de commande PowerShell |
|  | Aller dans le répertoire c:\agent  Lancer le config.cmd  Entrer l’URL de l’organisation DevOps créée précédemment  Entrer pour l’option PAT (Token)  Copier / coller le Token généré précédemment |
|  | Aller dans DevOps  New agent pool |
|  | Saisir le nom de l’agent pour créer un pool dédié  OK |
|  | Retourner dans la fenêtre de commande et saisir le nom du nouveau pool  Entrer  Entrer à nouveau pour le nom de l’agent par défaut |
|  | Entrer pour le répertoire par défaut  Y Entrer pour tourner l’agent en tant que service  Entrer pour Network service par défaut |
|  | L’agent doit démarrer |
|  | Aller dans DevOps  Contrôler la présence de l’agent dans le pool qui porte son nom |

# Implémentation de l’Agent DevOps A11y dans le pipeline

L’agent personnalisé étant opérationnel l’implémentation des tests d’accessibilité peut être réalisée directement dans le pipeline de livraison.

Sachant que l’agent fonctionne en analysant un site Web en ligne il faut déployer une première Web App pour procéder aux tests. C’est pourquoi nous avons préparé 2 Services d’application dans Azure.

Nous avons ajouté au pipeline 2 étapes :

* Tests d’accessibilité sur la première Web App en ligne
* Si les tests sont passés avec succès, mise en ligne de la seconde Web App qui correspond au site Web de production

## Implémentation de l’Agent dans le pipeline « Release »

L’extension DevOps que nous utilisons prend en paramètre une URL. C’est cette page qui est analysée. Notre site Web possède plusieurs pages et nous voulons toutes les tester alors il faut ajouter autant de tâches de tests que de pages.

Le Référentiel d’accessibilité et le niveau sont paramétrables, nous avons choisi pour commencer le WCAG 2.0 niveau A.

Il est possible de choisir d’interrompre le pipeline en cas d’erreur ou de continuer. Nous avons configuré les tests pour s’enchaîner sur toutes les pages et recevoir l’ensemble des non-conformités en une seule fois.

|  |  |
| --- | --- |
| **Illustration** | **Action** |
|  | Aller dans DevOps  Ouvrir le projet WebAppA11yClassic |
|  | Cliquer sur le pipeline Releases |
|  | Cliquer sur Edit pour modifier le pipeline |
|  | Positionner la souris sur Staging  Cliquer sur Add (a stage) sous la boite |
|  | Cliquer sur Empty job |
|  | Entrer le nom de l’étape |
|  | Cliquer sur 1 job, 0 task |
|  | Cliquer sur + sur la ligne Agent job |
|  | Entrer Accessibility dans le champ de recherche |
|  | Accessibility Testing devrait apparaître  Positionner la souris dessus  Cliquer sur Install |
|  | Accessibility Checker est disponible  Positionner la souris dessus  Cliquer sur Add |
|  | Sélectionner Agent job |
|  | Sélectionner l’Agent pool VMAgentA11y  (c’est lui qui contient notre VM Custom) |
|  | Sélectionner Accessibility Checker à gauche dans la liste des tâches de l’Agent job |
|  | Mettre à jour l’URL à tester avec l’URL de la Web App Staging (copier / coller l’URL depuis Azure / Web App Staging)  Déplier Advanced  Contrôler wcag2a (garder ce paramètre pour l’instant) |
|  | Ajouter une seconde tâche Accessibility Checker pour la seconde page |
|  | Mettre à jour l’URL avec la page Privacy  Déplier Control Options  Changer le comportement : Even if a previous task has failed, unless the deployment was canceled |
|  | Ajouter une troisième tâche Accessibility Checker pour la troisième page |
|  | Mettre à jour l’URL avec la page Accessibility  Déplier Control Options  Changer le comportement : Even if a previous task has failed, unless the deployment was canceled |
|  | La liste des tâches doit contenir 3 fois Accessibility Checker, 1 tâche par page Web du site |
|  | Cliquer sur Save  OK |
|  | Cliquer sur Release  Create release  Create |
|  | Cliquer sur le nom de la Release |
|  | Cliquer sur Accessibility Testing  (Failed, c’est normal) |
|  | Le résultat devrait présenter des erreurs de type Impact Level :Serious ou Critical transmises par Accessibility Checker |

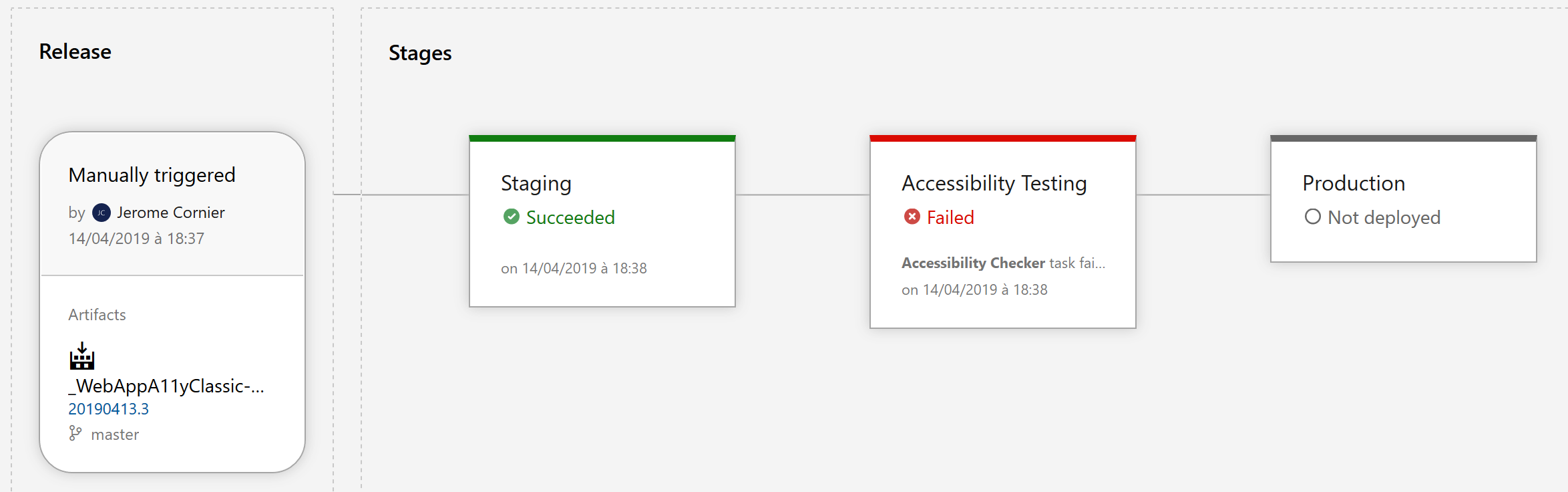
## Déploiement de l’application Web « Production »

La dernière étape consiste à déployer en production si les tests sont réussis. Pour cela nous allons ajouter une étape au pipeline Release.

|  |  |
| --- | --- |
| **Illustration** | **Action** |
|  | Cliquer sur le nom du pipeline Release |
|  | Cliquer sur Edit |
|  | Positionner la souris sur Accessibility Testing et cliquer sur Add |
|  | Positionner la souris sur Azure App Service deployment et cliquer sur Apply |
|  | Renommer l’étape en Production |
|  | Cliquer sur 1 job, 1 task |
|  | Sélectionner la souscription Azure  Conserver Web App on Windows  Sélectionner WebAppA11yClassic  (pas Staging, l’autre) |
|  | Cliquer sur Save  OK |
|  | Cliquer sur Release  Create release  Create |
|  | Cliquer sur le nom de la Release |

## Pipeline final

Voici à quoi ressemble le pipeline complet :



Pour cette Release certains tests ont échoué, la production n’a pas été mise en ligne. C’est précisément le comportement attendu en attendant les corrections pour la mise en conformité.

# Démonstration des tests d’accessibilité automatisés

La plateforme de démonstration étant opérationnelles nous pouvons procéder à des modifications du code source dans Visual Studio pour valider le pipeline de livraison continue dans son intégralité.

## Première Release en échec avec la Web App initiale

Nous l’avons évoqué précédemment, la solution initiale directement issue de Visual Studio comporte des erreurs de conformité.

|  |  |
| --- | --- |
| **Illustration** | **Action** |
|  | Cliquer sur Failed |
|  | Cliquer sur Accessibility Checker |
| 2019-04-14T16:38:37.0390880Z Rule:html-has-lang  2019-04-14T16:38:37.0390910Z Impact:Serious  2019-04-14T16:38:37.0390943Z Description:Ensures every HTML document has a lang attribute  2019-04-14T16:38:37.0390978Z More Info:https://dequeuniversity.com/rules/axe/3.1/html-has-lang?application=webdriverjs  2019-04-14T16:38:37.0457884Z ##[error]Impact Level:Serious; Rule:html-has-lang; <html> element must have a lang attribute  <html> | Dérouler jusqu’en bas le journal des erreurs  Dans ce cas il s’agit du paramètre de lang qui est absent du \_Layout.cshtml |

Le pipeline fourni le détail de l’erreur dans le journal de l’agent DevOps. La solution de tests s’appuyant sur l’API axe-core, le journal donne directement accès aux détails en ligne concernant les erreurs.

### Erreur n°1, paramètre de langue du document manquant

Pour la première erreur il s’agit du paramètre de langue du document qui est manquant.

<html lang="en">
  <!--document head and body-->
</html>

Figure : Correction proposée par Deque University

L’erreur est considérée « Sérieuse ». Le site Web de Deque University explique comment y remédier. En fin de page le lien vers la technique correspondante du W3C est disponible. Dans notre cas : [H57: Using language attributes on the html element](http://www.w3.org/TR/WCAG20-TECHS/H57.html).

### Erreur n°2, texte alternatif manquent sur une image

Cette erreur est volontaire pour nos tests. Nous avons ajouté une page « Accessibility » au projet dans laquelle nous avons inséré le logo de l’accessibilité fourni par les Nations Unies.

Voici le code initial de la page ASP.NET dans lequel il manque en effet le texte alternatif.

@page

@model AccessibilityModel

@{

ViewData["Title"] = "Declaration of Accessibility";

}

<div class="text-center">

<h1>@ViewData["Title"]</h1>

<p>This Demo Web Site is targeted to reach the Standard WCAG 2.0 A</p>

<a href="https://www.un.org/fr/webaccessibility/logo.shtml">

**<img src="~/images/UN\_Logo\_Accessibility.jpg" width="300" />**

</a>

</div>

## Mise en conformité du code d’après le rapport de tests

Nous pouvons corriger ces 2 erreurs, puis pousser les mises à jour dans GitHub ce qui a pour effet le déclenchement automatique du pipeline.

### Correction du paramètre de langue

Le paramètre à ajouter se situe dans la page partagée nommée \_Layout.cshtml.

<!DOCTYPE html>

**<html lang="fr">**

<body>

[... contenu de la page ...]

</body>

</html>

### Correction du texte alternatif de l’image

Même principe, nous pouvons ajouter le texte alternatif manquant.

<a href="https://www.un.org/fr/webaccessibility/logo.shtml">

<img src="~/images/UN\_Logo\_Accessibility.jpg" **alt="Accessibility Logo created by United Nations"** width="300" />

</a>

### Synchronisation des modifications avec GitHub

L’opération peut se faire directement depuis Visual Studio en synchronisant le projet local avec GitHub pour récupérer les modifications ayant pu être effectuées en parallèle par un autre développeur, puis en poussant nos modifications.

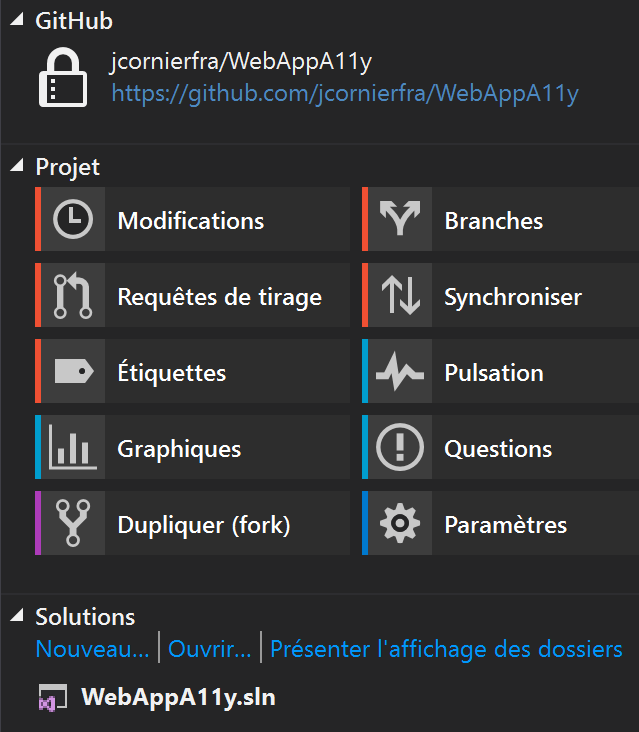


Figure : Capture d'écran, Visual Studio Team Explorer synchronisé avec GitHub

## Release réussie après correction

Après quelques minutes le pipeline nous donne le résultat de la dernière Release qui est maintenant conforme et déployé sur le Service d’application de production.

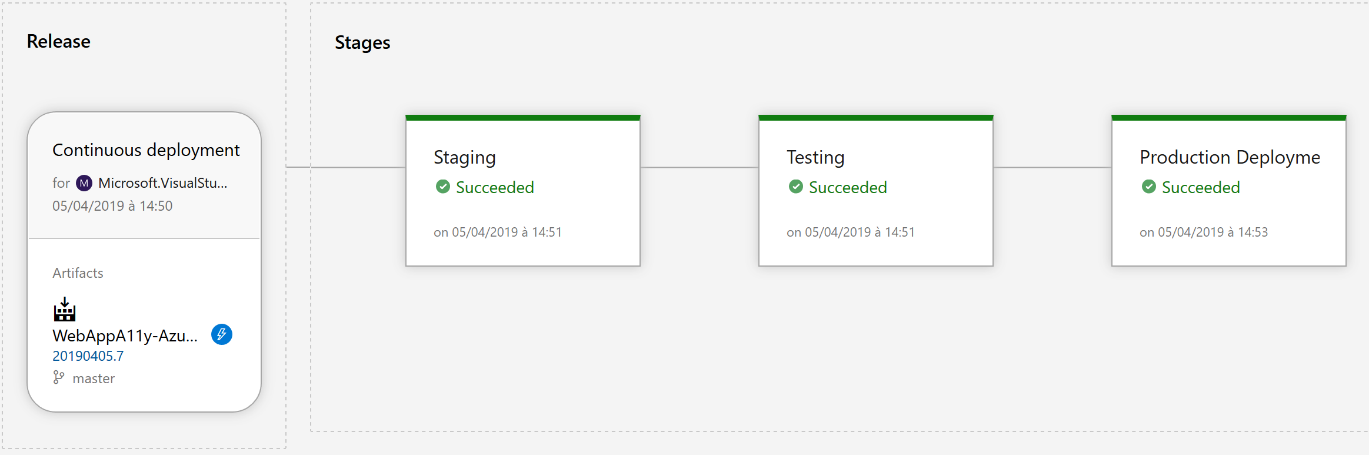


Figure : Pipeline affichant le succès de toutes les étapes

Nous vérifions sur le site Web publié, le texte associé est reconnu par le navigateur.



Figure : Capture d'écran du site Web avec texte associé à l'image

## Elévation du niveau d’accessibilité WCAG 2.0 A + AA

Les tests étant concluant nous allons augmenter le niveau des tests sur le site Web. Conformément à nos attentes l’élévation du niveau WCAG 2.0 A vers A et AA produit de nouvelles erreurs. Ce sont des erreurs de contraste, en effet le contraste est au niveau AA dans les référentiels d’accessibilité.

|  |  |
| --- | --- |
| **Illustration** | **Action** |
|  | Cliquer sur le nom du pipeline Release |
|  | Cliquer sur Edit |
|  | Cliquer sur 1 job, 3 Tasks dans l’étape Accessibility Testing |
|  | Pour chaque Accessibility Checker, ajouter le wcag2aa |
|  | Déplier Advanced  Cocher wcag2aa (garder wcag2a) |
|  | Cliquer sur Save  OK |
|  | Cliquer sur Release  Create release  Create |
|  | Cliquer sur le nom de la Release |

Le pipeline devrait maintenant présenter des erreurs de contraste.

### Identification des défauts de conformité

Pour faciliter l’identification de l’erreur dans la page nous pouvons utiliser le plugin WCAG 2.0 Contrast checker pour Firefox sur le poste de travail ou tout autre outil similaire.

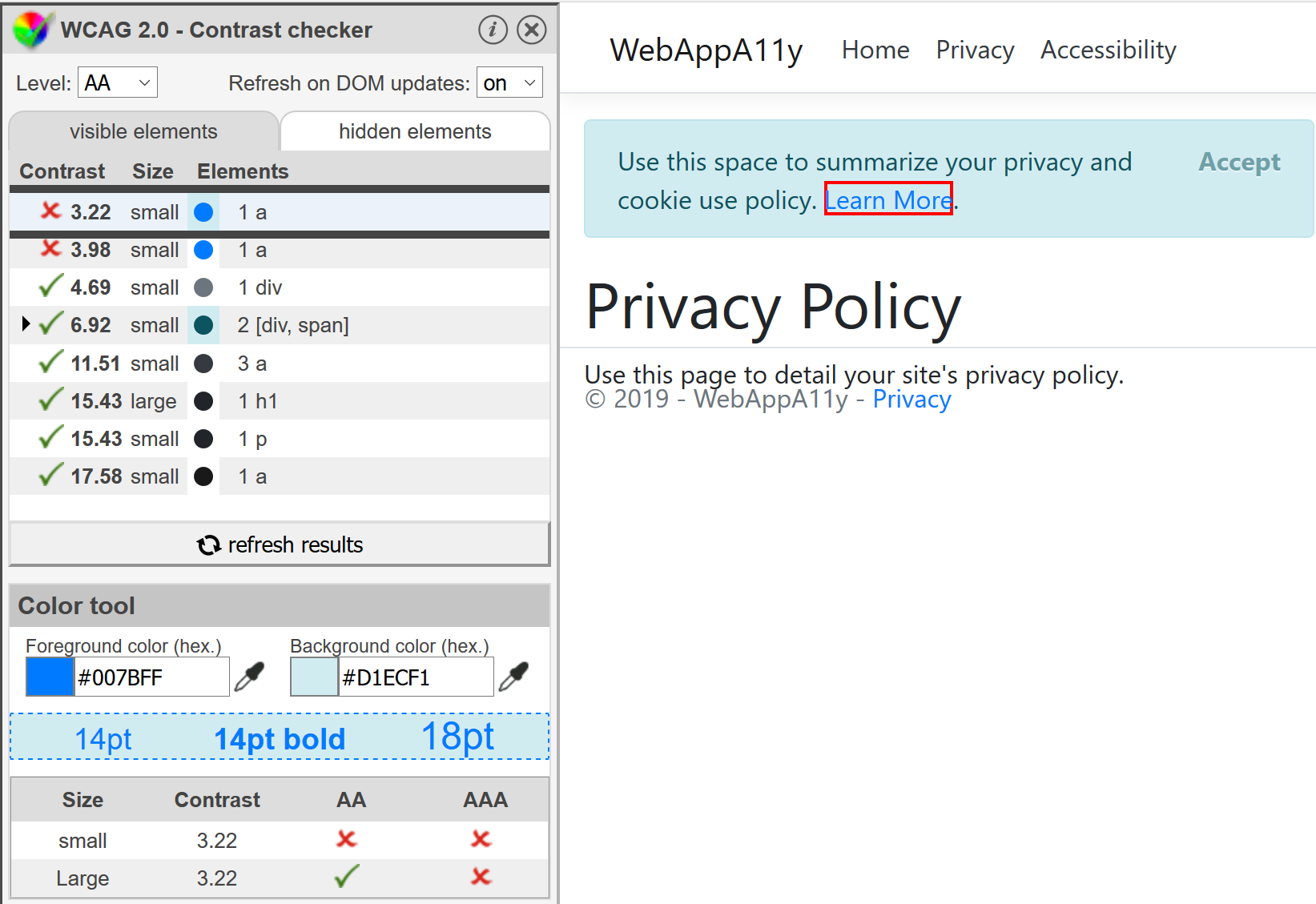


Figure : Extension Color Contrast pour Firefox

Les erreurs concernent les 2 liens « Learn More » et « Privacy » qui s’affichent en effet en bleu moyen sur bleu clair ou blanc avec un contraste de 3,22 et 3,98, ce qui n’est pas très contrasté et inférieur au critère du WCAG 2.0 qui exige un contraste minimum de 4,5 pour les textes de petite taille.

### Remédiation pour le niveau A + AA

Le site de Deque University vers lequel nous sommes orientés dans le journal d’erreur fourni aussi un outil d’analyse de contraste en ligne. Nous avons choisi de conserver la couleur de fond #D1ECF1 et le bleu du texte. Pour obtenir le contraste requis il faut foncer la couleur du texte, ce qui donne la valeur #0064CC.



Figure : Color Contrast Analyzer pour calculer la couleur nécessaire au contraste exigé

Nous apportons les modifications au code source, dans ce cas il faut ajuster les couleurs définies par la feuille CSS de la librairie Bootstrap dans le projet Visual Studio.

Notons que le choix de la nouvelle couleur répond au niveau AA mais pas au niveau AAA plus exigent. C’est le genre de choix qu’il est important de définir dès la conception ou dans la charte de Design pour éviter les corrections au moment du développement.

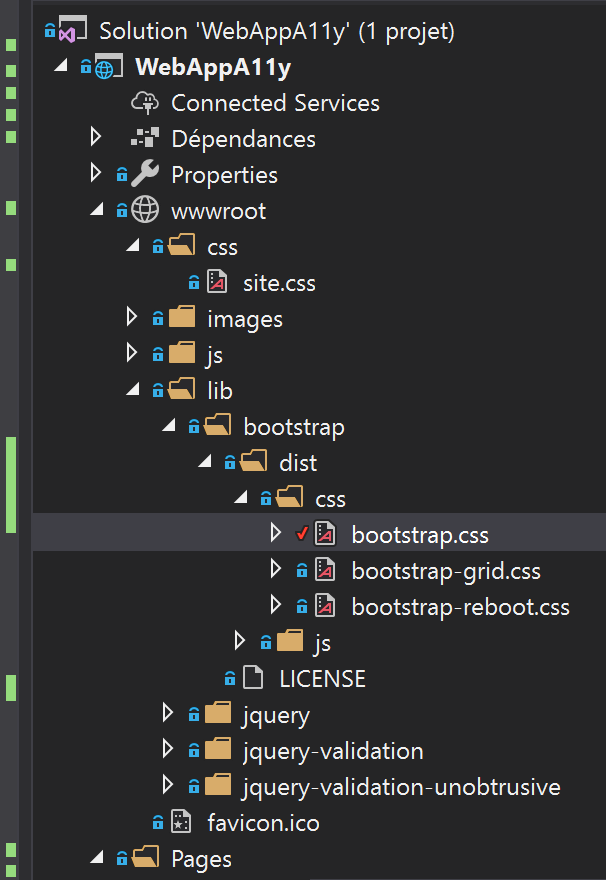


Figure : Positionnement de la feuille CSS dans le projet Visual Studio

Bootstrap est une librairie additionnelle Open Source sujette à des mises à jour. Il est déconseillé de la modifier. La bonne pratique dans ce cas est de procéder à un CSS override, c’est-à-dire l’ajout d’une nouvelle définition dans un autre fichier qui viendra se substituer à la définition originale. Il existe un fichier prévu pour cela dans le projet Visual Studio : site.css. Voici la modification apportée au code source.

...

/\* Bootstrap.css override to darken the blue color of links

-------------------------------------------------- \*/

a {

/\* color: #007bff; <- couleur bleue originale pour les liens \*/

**color: #0064cc**; /\* <- couleur bleue plus foncée conforme au WCAG 2.0 AA \*/

text-decoration: none;

background-color: transparent;

}

...

Les modifications sont poussées une nouvelle fois dans GitHub.

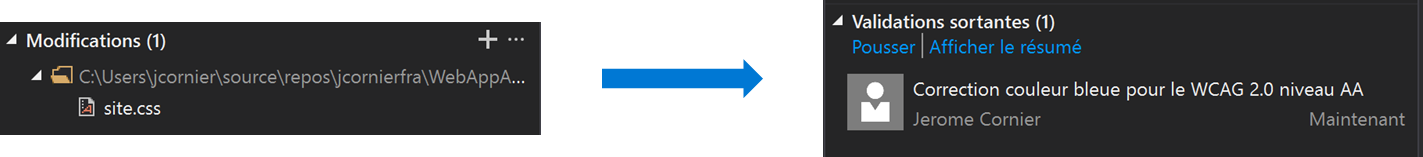


Figure : Push des modifications de VS vers Github

Le pipeline « Builds » est automatiquement lancé.

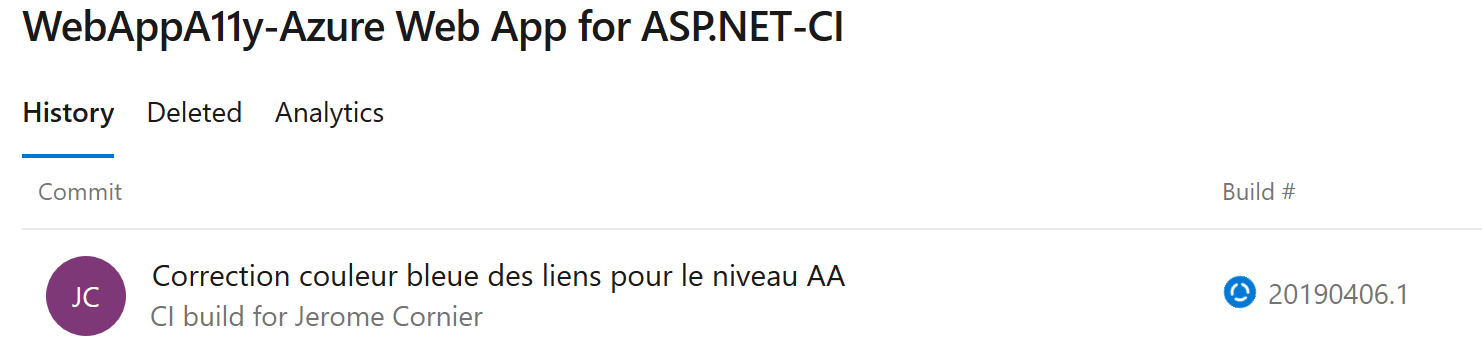
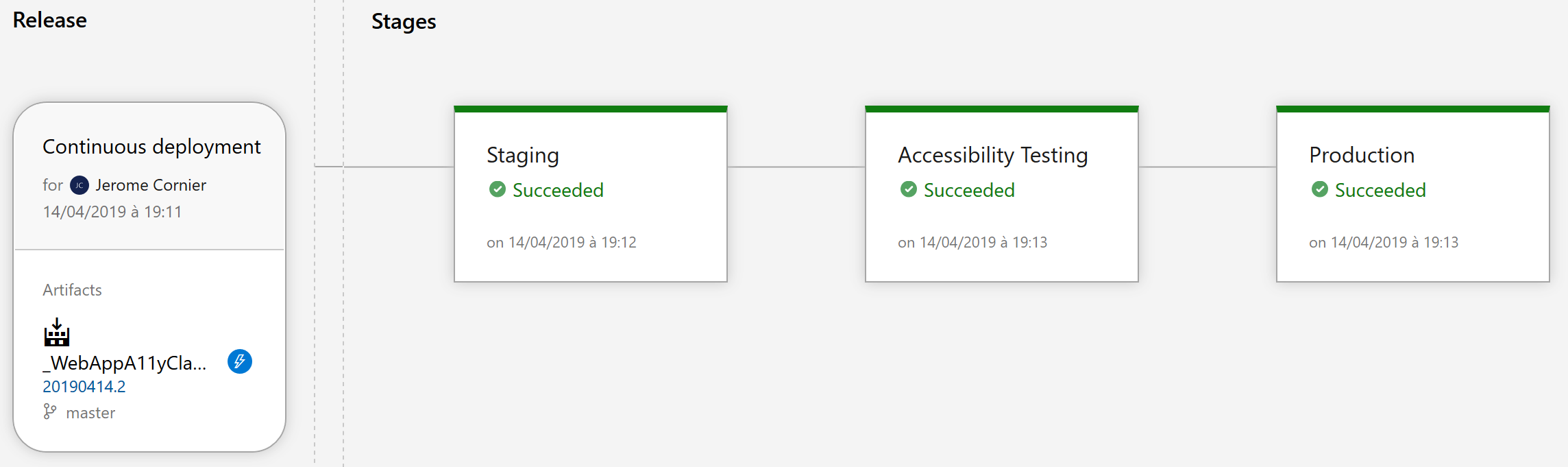


Figure : Pipeline « Builds » lancé automatiquement

Pipeline réussi.



### Résultat avant / après

Les tests aux niveaux WCAG 2.0 A et AA sont passés et le résultat n’est finalement pas très éloigné de l’original à ceci près qu’il est désormais conforme aux critères d’accessibilité.

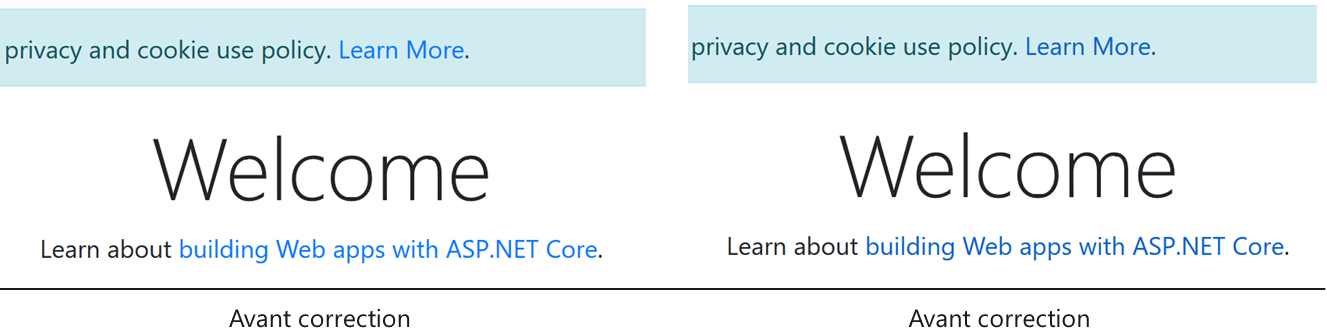


Figure : Captures d'écran avant et après pour illustrer la variation de couleur pour passer le niveau AA

# Web App en Container – Préparation de l’environnement

Le second prototype que nous avons construit met en œuvre des containers Docker pour héberger les deux sites Web de préproduction et de production.

## Architecture générale

Pour cette version de l’application en container Docker nous utiliserons la convention de noms suivante. Les principales différences par rapport à la version classique sont :

* Azure Container Repository pour stocker l’image du container
* Les App Services qui deviennent des Web App for Containers

|  |  |
| --- | --- |
| **Nom** | **Composant** |
| Application Web |  |
| WebAppA11yDocker | Solution Visual Studio |
| WebAppA11yDocker | Référentiel GitHub |
| Ressources Azure |  |
| WebAppA11yDockerRG | Resource group |
| WebAppA11yDockerACR | Azure Container Repository |
| WebAppA11yDockerStaging | Web App for Containers « Staging » |
| WebAppA11yDockerStagingPlan | App Service Plan « Staging » |
| WebAppA11yDocker | Web App for Containers « Production » |
| WebAppA11yDockerPlan | App Service Plan « Production » |
| Azure DevOps |  |
| A11yDemo | Organisation DevOps |
| WebAppA11yDocker-ASP.NET-CI | Pipeline « Builds » |
| WebAppA11yDocker-ASP.NET-CD | Pipeline « Release » |

### Architecture de la Web App Linux en Container

L’architecture diffère de l’application Web précédente car il faut prendre en compte :

* Le container Docker qui doit être stocké dans un Repository. Nous avons choisi Azure Container Registry qui suffit pour ce prototype simple
* Les services d’application qui doivent s’appuyer sur les containers, sachant que ceux-ci ont un système d’exploitation Linux

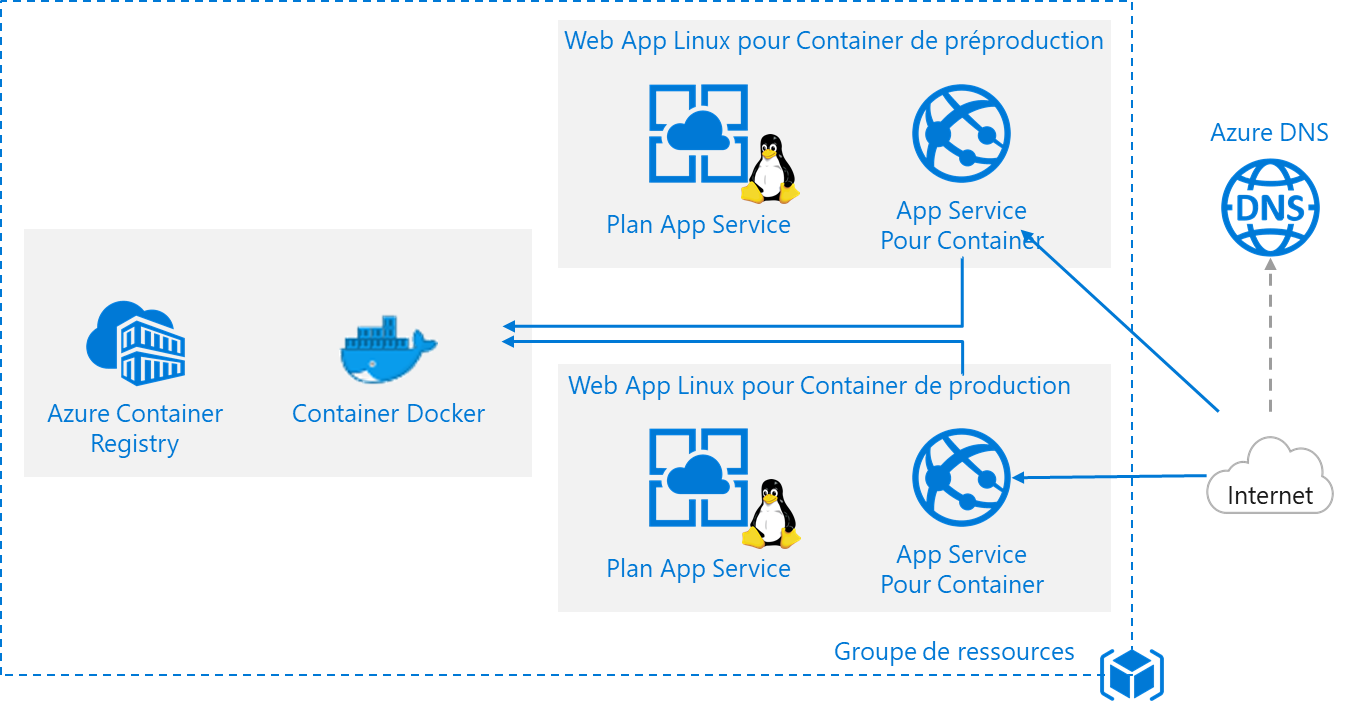


Figure 57 : Architecture Web App pour Container Docker

Nous aurions pu utiliser Azure Kubernetes Services au lieu de Azure Container Registry mais cela n’était pas justifié et surtout l’architecture AKS nécessite minimum 2 machines virtuelles puissantes qui portent le coût mensuel au-delà de notre souscription.

### Architecture du pipeline pour le container Docker

La présentation du pipeline pour la solution en container parait plus complexe car elle fait appel à des éléments d’architecture complémentaires mais en pratique elle se révèle plus rapide, avec moins d’étapes pour le pipeline « Builds ».

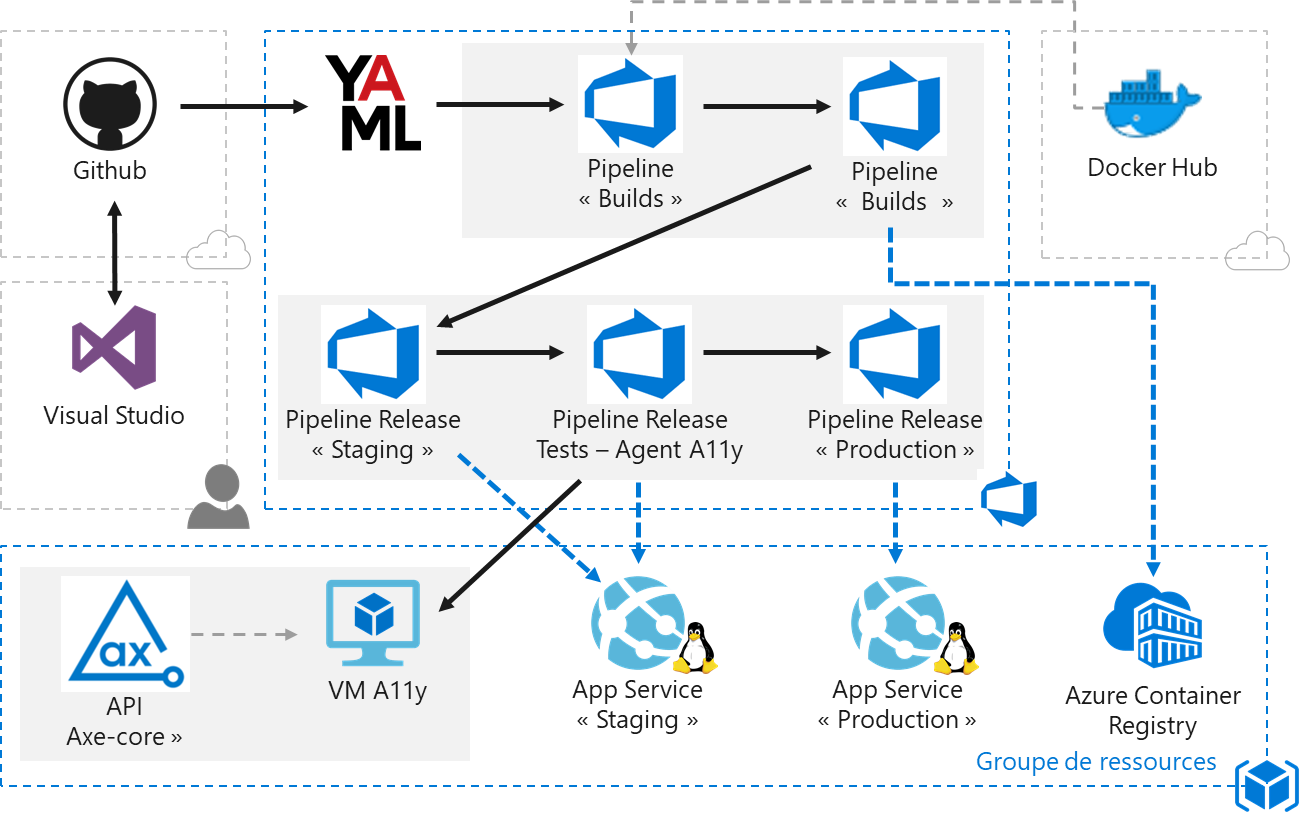


Figure : Pipeline DevOps pour une application Web en Container

Les principales différences avec la solution précédente sont :

* Un fichier YAML de script du pipeline « Builds » est nécessaire
* Le pipeline « Builds » doit tirer une image de base depuis Docker Hub pour la construire avec l’application Web
* Le déploiement n’est plus le chargement du site Web dans le Service d’application mais l’envoi du nouveau container dans Azure Container Registry

Les tests d’accessibilité automatisés quant à eux sont identiques puisqu’ils utilisent l’URL du site Web publié.

## Ajout de Docker à l’environnement de développement

Docker Desktop doit être installé sur le poste de travail. L’installation de cette application fournit le nécessaire pour créer des projets Visual Studio avec le support Docker et pour tester en local la solution.

Une extension des commandes en ligne permet de gérer les containers sur le poste local.

### Installation de Docker sur le poste de développement

|  |  |
| --- | --- |
| **Illustration** | **Action** |
|  | Ouvrir un navigateur  Taper l’adresse :  <https://www.docker.com/products/docker-desktop>  cliquer sur Download for Windows |
|  | Se connecter ou créer un compte si besoin  Cliquer sur Get Docker  Save |
|  | Aller dans le répertoire de téléchargement  Lancer l’exécutable d’installation de Docker |
|  | Décocher Use Windows containers |
|  | L’installation nécessite une déconnexion / reconnexion |
|  | Après la reconnexion Docker demande l’activation des fonctionnalités.  Cliquer sur OK |
|  | Cliquer droit sur l’icône Docker dans la barre de notification  Cliquer sur Settings |
|  | Sélectionner Shared Drives  Vérifier qu’un disque est partagé  Fermer la fenêtre des paramètres |
|  | Démarrer une invite de commandes  Taper docker version  La liste des composants et des versions doit apparaître  Fermer l’invite de commandes |

### Ajout du support ASP.NET et Docker dans Visual Studio

|  |  |
| --- | --- |
| **Illustration** | **Action** |
|  | Démarrer Visual Studio  Cliquer sur Outils, puis Obtenir les outils de fonctionnalités |
|  | Dans « Charges de travail » vérifier ou cocher que ASP.NET et Azure development sont cochés |
|  | Dans Composants individuels, vérifier ou cocher GitHub extension for Visual Studio |

## Création de l’application Web en container et publication dans GitHub

Pour cette préparation nous procéderons en 6 étapes :

* Création de l’application pour Docker dans Visual Studio
* Modifier le fichier Dockerfile
* Test avec Docker en local
* Publication dans GitHub
* Création d’une nouvelle page dans le projet de l’application
* Contrôle de mise à jour dans GitHub

### Création de l’application dans Visual Studio

Cette opération est similaire à celle déjà effectuée pour l’application Web classique. La principale différence est la prise en charge du container Docker dans le projet Visual Studio.

|  |  |
| --- | --- |
| **Illustration** | **Action** |
|  | Lancer Visual Studio  Créer un nouveau projet |
|  | Sélectionner .NET Core |
|  | Sélectionner Application web ASP.NET Core |
|  | Saisir le nom de l’application  Ne nom de la solution (le même)  Choisir le répertoire |
|  | Cocher Créer un répertoire pour la solution  Ne pas cocher Créer un dépôt Git (nous utiliserons GitHub)  Valider ces choix par OK |
|  | Vérifier que Application web est sélectionné  Décocher HTTPS  Cocher « Activer le support Docker »  SE : Linux  OK |
|  | Le projet apparaît dans Visual Studio / Explorateur de solutions  Le fichier Dockerfile apparaît, c’est une différence par rapport à l’application classique |

### Modifier le fichier Dockerfile

Le fichier Dockerfile créé par défaut ne fonctionne pas, il faut l’adapter à la structure de l’application. Voici le fichier que nous proposons pour prototype.

FROM microsoft/dotnet:2.2-aspnetcore-runtime AS base

WORKDIR /app

EXPOSE 80

FROM microsoft/dotnet:2.2-sdk AS build

WORKDIR /app

# Build solution

COPY . ./

RUN dotnet restore

RUN dotnet build -c Release -o out

FROM build AS publish

RUN dotnet publish -c Release -o out

# Build runtime image

FROM base AS final

WORKDIR /app

COPY --from=publish /app/out ./

ENTRYPOINT ["dotnet", "WebAppA11yDocker.dll"]

Ce script montre les opérations suivantes qui devront être réalisées pour construire le container avec l’application Web :

* Chargement de l’image aspnetcore-runtime en dotnet 2.2 qui contiendra l’application Web dans le répertoire /app et qui sera exposée sur le port http 80
* Chargement de l’image sdk en dotnet 2.2 pour construire et publier l’application Web
* Les sources sur projet sont copiées dans le répertoire /app de l’image sdk
* Dotnet restore permet de reconstruire le projet en chargeant le fichier de solution .sln
* L’application est construite puis publiée
* Les fichiers résultants sont ensuite copiés de l’image sdk vers l’image runtime
* L’application est devenue une librairie .dll qui sert de point d’entrée dans le container

Le principe est le même que ce soit sur le poste local ou dans le pipeline de livraison continue. Il peut exister quelques différences, par exemple lorsque plusieurs containers doivent exposer les sites sur des ports spécifiques, dans ce cas un fichier Dockerfile.override permet de modifier la création du container selon l’usage pour des tests locaux ou pour l’envoie dans le pipeline.

### Test de l’application Docker en local

|  |  |
| --- | --- |
| **Illustration** | **Action** |
|  | Cliquer droit sur la solution, puis Générer la solution pour vérifier la construction sans erreur |
| ========== Génération : 1 a réussi, 0 a échoué, 0 mis à jour, 0 a été ignoré ========== | Le résultat doit être réussi |
|  | Cliquer sur Docker |
|  | Après quelques instants le site web est ouvert dans un navigateur.  Localhost :xxxxx correspond à la machine locale qui expose le site sur un port étendu (le port 80 est généralement déjà utilisé) |
|  | Cliquer sur le carré rouge pour stopper le service |
|  | Ouvrir une invite de commandes (pas besoin du mode admin)  Taper docker container ps  Le container webappa11ydocker apparaît |
|  | Taper docker container stop <container ID> pour arrêter le service  Container ID peut être copier/coller depuis la liste précédente  Taper Exit pour fermer l’invite de commandes |

### Publication de l’application dans GitHub

Il s’agit de la même opération que pour l’application classique.

|  |  |
| --- | --- |
| **Illustration** | **Action** |
|  | Cliquer en bas à droite sur Ajouter au contrôle de code source  Cliquer sur Git |
|  | Cliquer sur Publier sur GitHub  ATTENTION Azure DevOps est un autre dépôt. Nous voulons sélectionner GitHub |
|  | Vérifier ou sélectionner le compte/profile GitHub  Saisir le nom de la Web App  Saisir une courte description (modifiable ultérieurement)  Cocher Référentiel privé  Publier |
|  | Le dépôt a été créé |
|  | Ouvrir <https://github.com>  Cliquer sur le nom de la Web App |
|  | Les fichiers sont présents et le fichier .dockerignore a été ajouté. |

### Ajout d’une page Web dans l’application

Les modifications que nous proposons sont les mêmes que pour l’application classique. Il n’y a aucune différence dans le code lui-même.

|  |  |
| --- | --- |
| **Illustration** | **Action** |
|  | Revenir dans Visual Studio, Explorateur de solutions  Déployer Pages et Shared |
|  | Cliquer bouton droit sur Privacy.cshtml  Copier  Cliquer bouton droit sur Pages  Coller  Le nouveau fichier Privacy-Copier apparaît |
|  | Renommer en Accessibility.cshtml |
|  | Dans Shared, cliquer sur \_Layout.cshtml  Le contenu s’ouvre dans la fenêtre centrale |
|  | Sélectionner et copier les 3 lignes correspondant à Privacy  (Ctrl-C, Ctrl-V, Ctrl-V) |
|  | Renommer Privacy en Accessibility sur la nouvelle ligne  Enregistrer (Ctrl-S) |
|  | Sélectionner Accessibility.cshtml |
|  | Remplacer Privacy par Accessibility  Enregistrer |
|  | Déplier Accessibility et sélectionner Accessibility.cshtml.cs |
|  | Renommer PrivacyModel en AccessibilityModel  Enregistrer |
|  | Cliquer bouton droit sur la Solution  Générer la solution |
| ========== Génération : 1 a réussi, 0 a échoué, 0 mis à jour, 0 a été ignoré ========== | La solution devrait se construire sans erreur |
|  | Cliquer bouton droit sur wwwroot  Ajouter  Nouveau dossier  Saisir le nom « images »  Valider |
|  | Enregistrer l’image ci-contre dans un fichier jpeg sous le nom  UN\_Logo\_Accessibility.jpg |
|  | Glisser et déposer le fichier image dans le dossier images |
|  | Cliquer sur Accessibility.cshtml |

Ajouter le logo dans le code qui doit ressembler à celui-ci-dessous. Il manque le alt=« xxx » pour l’image, c’est volontaire pour les tests d’accessibilité.

@page

@model AccessibilityModel

@{

ViewData["Title"] = "Declaration of Accessibility";

}

<div class="text-center">

<h1>@ViewData["Title"]</h1>

<p>This Demo Web Site is targeted to reach the Standard WCAG 2.0 A + AA</p>

<a href="https://www.un.org/fr/webaccessibility/logo.shtml">

<img src="~/images/UN\_Logo\_Accessibility.jpg" width="300" />

</a>

</div>

### Mise à jour dans GitHub

|  |  |
| --- | --- |
| **Illustration** | **Action** |
|  | Cliquer sur le stylo en bas à droite  (Autre solution : dans Team Explorer / Modifications) |
|  | Entrer une description correspondant aux modifications effectuées  Vérifier la liste des fichiers qui correspond à ceux modifiés précédemment  Cliquer sur Valider tout |
|  | Cliquer sur Synchroniser |
|  | Cliquer sur Pousser |
|  | La synchronisation a été effectuée |

## Création de l’environnement Azure pour la Web App en Container

L’étape suivante consiste à préparer l’environnement Cloud pour recevoir l’application.

A partir du portail Azure (<http://portal.azure.com>) nous devons créer :

* 1 référentiel de container, Azure Container Repository
* 2 Services d’application Web App pour Container avec leurs plans de services associés : la première Web App servira de préproduction « Staging » pour tester l’accessibilité et la seconde sera déployée en production si les tests sont réussis

### Création du référentiel de containers

|  |  |
| --- | --- |
| **Illustration** | **Action** |
|  | Dans le portail Azure, cliquer sur Create a resource |
|  | Taper Container Registry dans la barre de recherche, puis retour |
|  | Create |
|  | Nommer le registre, bom de la Web App suivi de ACR  Sélectionner la souscription Azure  Créer un nouveau groupe de ressources, nom de la Web App suivi de RG  Create |
|  | Pin to dashboard |

### Création des Web App for Containers

|  |  |
| --- | --- |
| **Illustration** | **Action** |
|  | Dans le portail Azure, cliquer sur Create a resource |
|  | Taper container dans la barre de recherche, puis retour |
|  | Cliquer sur Web App for Containers |
|  | Create |
|  | Saisir le nom de la Web App  Choisir la souscription (si plusieurs)  Utiliser le groupe de ressource créé précédemment  Sélectionner Linux  Cliquer sur App Service plan |
|  | Create new |
|  | Saisir le nom suivi de « Plan »  Choisir l’emplacement (« West Europe » dans notre exemple)  Cliquer sur Pricing tier pour changer le Plan par défaut en B1 |
|  | Cliquer sur Dev / Test  Cliquer sur B1  Apply |
|  | Vérifier, puis OK |
|  | Cliquer sur Configure container |
|  | Single Container, Quickstart  Apply  (nous reviendrons sur cette configuration ultérieurement) |
|  | Create  Pin to dashboard |
|  | Répéter l’opération pour la Web App « Staging » |
|  | Vérifier, Create  Pin to dashboard |
|  | Cliquer sur l’une des deux Web App for Container |
|  | Cliquer sur l’URL qui apparaît |
|  | Une page Web s’ouvre confirmant le bon fonctionnement de la Web App utilisant pour l’instant un container avec un serveur nginx |

## Création du pipeline DevOps

Le principe est le même que pour l’application classique. Les différences sont liées au modèle de container qui implique une intégration et un déploiement adaptés.

### Création d’un nouveau Projet DevOps

|  |  |
| --- | --- |
| **Illustration** | **Action** |
|  | Ouvrir <https://dev.azure.com> et cliquer sur l’organisation |
|  | Create project |
|  | Entrer le nom de la Web App  Entrer une description  Choisir Private  Create |

### Création du pipeline « Builds » DevOps (YAML)

|  |  |
| --- | --- |
| **Illustration** | **Action** |
|  | Cliquer sur Pipelines |
|  | New pipeline |
|  | Cliquer sur GitHub YAML |
|  | Cliquer sur le référentiel de la Web App  WebAbbA11yDocker |
|  | Saisir ou confirmer le mot de passe pour GitHub |
|  | Accéder au bas de la page  Sélectionner uniquement le nouveau référentiel  Approve and install |
|  | Choisir le compte à utiliser (en principe déjà connecté) |
|  | Cliquer sur ASP.NET Core |
|  | Un script YAML est proposé par défaut, nous devons l’adapter à notre application |

### Modification du fichier YAML

Remplacer le code par défaut par le code ci-dessous sans valider pour le moment !

# ASP.NET Core

# Build and test ASP.NET Core projects targeting .NET Core.

# Add steps that run tests, create a NuGet package, deploy, and more:

# https://docs.microsoft.com/azure/devops/pipelines/languages/dotnet-core

trigger:

- master

pool:

vmImage: 'Ubuntu 16.04'

variables:

imageName: 'webappa11ydocker:$(Build.BuildId)'

dockerId: ' webappa11ydockeracr'

dockerPassword: 'bR4vMqhsv9ii8iU/JCdLk3u8BmsKWksE'

steps:

- script: docker build -t $(dockerId).azurecr.io/$(imageName) .

workingDirectory: WebAppA11yDocker

displayName: 'Docker Build'

- script: |

docker login -u $(dockerId) -p $pswd $(dockerId).azurecr.io

docker push $(dockerId).azurecr.io/$(imageName)

env:

pswd: $(dockerPassword)

workingDirectory: /

displayName: 'Docker Push'

|  |  |
| --- | --- |
| Portail Azure |  |
|  | Ouvrir [https ://portal.azure.com](https://portal.azure.com)  Accéder au référentiel de container  (cliquer sur l’icône sur le dashboard ou aller dans Container registries) |
|  | Cliquer sur Access keys |
|  | Cliquer sur Enable  Copier le password |
| Azure DevOps |  |
|  | Retourner sur la page du pipeline DevOps  Remplacer le password par le nouveau |
|  | Save and run  Save and run |
|  | Les tâches de construction doivent réussir |

### Configuration des Web App for Containers Azure

|  |  |
| --- | --- |
| **Illustration** | **Action** |
|  | Ouvrir le portail Azure  <https://portal.azure.com>  Accéder au référentiel de containers |
|  | Cliquer sur repositories |
|  | L’image que nous avons préalablement construite et publiée devrait apparaître |
|  | Revenir sur le dashboard  Cliquer sur la Web App Docker Staging |
|  | Cliquer sur Container settings |
|  | Cliquer sur Azure Container Registry  Sélectionner le nom du référentiel  Sélectionner l’image  Sélectionner le tag avec le n° le plus élevé  Save |
|  | Cliquer sur overview |
|  | Cliquer sur l’URL |
|  | Une page Web s’ouvre  Patienter le temps du redémarrage du container  Le site web devrait apparaître |
|  | Réitérer les opérations pour la Web App for Container de production |

### Création du pipeline « Release » du Container

La création du pipeline « Release » est similaire à celle de l’application classique à l’exception du choix des tâches de déploiement du Service d’Application qui cette fois doit être pour container Linux.

|  |  |
| --- | --- |
| **Illustration** | **Action** |
|  | Sélectionner Releases |
|  | New pipeline |
|  | Choisir Azure App Service deployment  Apply |
|  | Renommer en « Staging » |
|  | Cliquer sur Add an artifact |
|  | Sélectionner le projet (Docker)  Sélectionner la source : le pipeline créé précédemment  Les autres champs sont automatiquement renseignés  Add |
|  | Cliquer sur l’icône de l’éclair dans le coin de la boite de l’Artifact |
|  | Cliquer sur l’interrupteur pour activer le déclenchement automatique du pipeline Release |
|  | Cliquer sur 1 job, 1 task |
|  | Les principales différences dans la version en Container se trouvent ici !  Sélectionner la souscription Azure  Sélectionner Web App for Containers (Linux)  Sélectionner la Web App “Staging »  Registry : saisir le nom complet du référentiel (webappa11ydockeracracr.azurecr.io)  Repository : saisir le nom de l’application (sans acr à la fin) |
|  | Positionner la souris sur New release pipeline  Cliquer sur le stylo |
|  | Changer le nom du pipeline |
|  | Save  OK |
|  | Cliquer sur All pipelines |
|  | Cliquer sur le nom du pipeline |
|  | Create release  Create |
|  | Cliquer sur le nom de la Release en cours |
|  | Patienter |
|  | Le déploiement devrait être réussi |

## Implémentation des tests d’accessibilité automatisés

L’implémentation des tests d’accessibilité est identique à celle mise en œuvre pour le prototype précédent, avec exactement les mêmes résultats en termes de conformité puisque l’application source est la même, présentée également sous forme de site Web à l’agent de test d’accessibilité, seule l’architecture en container diffère.

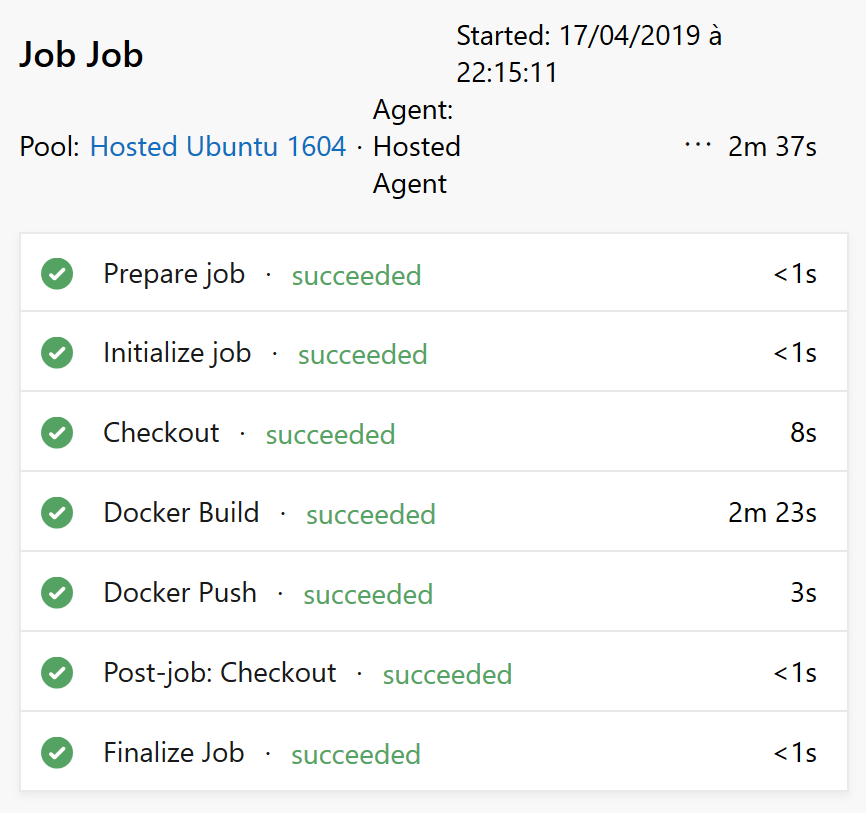
Les étapes de l’implémentation de l’Agent sont identiques.

La configuration de la Web App for Containers de production se configure comme la Web App Staging que nous venons de décrire.

## Pipeline final

Voici à quoi ressemble le pipeline CI/CD complet.

### CI



### CD

