GTACD

GiThub Action CicD

Introduction

Nous allons nous lancer dans une exploration passionnante au croisement du DevOps et de la cybersécurité. Notre objectif ? Comprendre en profondeur comment automatiser les processus de livraison logicielle, non pas juste pour la rapidité, mais avant tout pour renforcer la sécurité de nos infrastructures et de nos applications.

Pourquoi cette exploration pour des experts Cyber ?

Dans le monde actuel, le développement logiciel est indissociable de la sécurité. Les chaînes d'approvisionnement logicielles sont devenues des cibles privilégiées pour les attaquants. Automatiser la manière dont le code passe de l'idée au déploiement est crucial, mais cela crée aussi de nouvelles surfaces d'attaque si ce n'est pas fait correctement. Cette exploration vous donnera les outils et les réflexes pour identifier, atténuer et prévenir ces risques.

Ce que nous allons construire : Un pipeline sécurisé en 2 parties

Nous allons bâtir un pipeline CI/CD (Intégration Continue / Déploiement Continu) complet, en intégrant des mécanismes de sécurité à chaque étape clé

Partie 1 : GitHub Actions et la Cybersécurité du CI

Nous plongerons dans GitHub Actions, un outil puissant pour automatiser des workflows directement depuis vos dépôts de code.

Vous apprendrez à déclencher des vérifications de code automatiques qui serviront de première ligne de défense, détectant les anomalies avant même le déploiement.

Surtout, nous nous concentrerons sur la gestion sécurisée des informations sensibles (secrets) : comment les stocker, les utiliser sans les exposer dans les logs, et comprendre les vulnérabilités liées à leur mauvaise manipulation. C'est un aspect fondamental pour éviter les fuites de données critiques.

Partie 2 : Ansible, GitHub Actions et la Sécurité du Déploiement (CD)

Forts de vos connaissances en Ansible, nous intégrerons pour automatiser le déploiement sur des serveurs distants.

L'accent sera mis sur le durcissement des accès : vous configurez des clés SSH dédiées et appliquez les principes du moindre privilège pour l'utilisateur de déploiement. Minimiser les permissions est une règle d'or en cybersécurité pour limiter l'impact d'une compromission.

L'objectif ultime est de créer une chaîne automatisée et traçable : chaque modification de code sur GitHub déclenche un processus sécurisé qui inclura des tests et un déploiement fiable sur votre infrastructure. La traçabilité est essentielle pour l'audit et la réponse aux incidents.

Découverte et Prise en Main de GitHub Actions

Vous devez vous renseigner sur :

- Qu'est-ce que l'Intégration Continue, la Livraison Continue et le Déploiement Continu ?
- Pourquoi le CI/CD est-il devenu indispensable dans le développement logiciel ?
- Quels sont les risques et les défis de sécurité spécifiques à l'automatisation des pipelines (fuite de secrets, vulnérabilités introduites, accès non autorisés) ?
- Exploration des concepts clés : Workflows, Events, Jobs, Steps, Runners, Actions, Secrets.
- Anatomie d'un fichier de workflow (.yml).

Chaque groupe crée un nouveau dépôt GitHub public (Ne pas initialiser avec un README ni `.gitignore`.)

Clonez ce dépôt en local sur votre machine de travail

Job 1 : Création d'une Application Simple et d'un Script de Vérification

Développement d'une application web minimale :

Dans le répertoire racine de votre dépôt, créez un dossier nommé app/.

À l'intérieur de app/, créez un fichier index.html avec un contenu HTML basique (un titre, un paragraphe de bienvenue, et un commentaire pour une date de génération future).

Création d'un script de "test" simulé :

Dans le dossier app/, créez un fichier check_app.sh.

Ce script Bash doit simuler une vérification simple de l'application (par exemple, vérifier la présence du fichier index.html). Il doit afficher des messages indiquant le début et la fin des vérifications, et renvoyer un code de sortie de 0 en cas de succès et de 1 en cas d'échec.

Commitez vos modifications avec un message clair.

Poussez vos modifications sur la branche main de votre dépôt GitHub.

Job 2 : Premier Workflow GitHub Actions (CI)

Création du répertoire des workflows :

À la racine de votre dépôt, créez le dossier .github/workflows/.

Création du fichier de workflow CI :

Dans le dossier .github/workflows/, créez un fichier nommé ci_basic.yml.

Ce workflow doit respecter les spécifications suivantes :

Nom du workflow : Un nom explicite de votre choix (par exemple, "Basic CI Workflow").

Déclencheur : Le workflow doit se déclencher automatiquement lors de chaque push sur la branche main.

Job unique : Définissez un unique job appelé build-and-test.

Runner : Ce job doit s'exécuter sur un runner ubuntu-latest hébergé par GitHub.

Étapes (Steps) :

- Checkout repository : Utilisez l'action officielle actions/checkout@v4 pour cloner le contenu de votre dépôt sur le runner.
- Run application check script : Exécutez le script ./app/check_app.sh que vous avez créé.
- Display index.html content : Affichez le contenu du fichier app/index.html directement dans les logs du workflow.

Lancement et observation du workflow :

Committez le fichier ci_basic.yml et poussez-le sur la branche main.

Rendez-vous sur votre dépôt GitHub, puis cliquez sur l'onglet Actions.

Localisez votre workflow en cours d'exécution, cliquez dessus et explorez les logs de chaque étape pour vérifier que tout s'est déroulé comme prévu.

Job 3 : Gestion des Secrets et des Variables d'Environnement

Comprendre les Secrets GitHub :

Pourquoi est-il impératif d'utiliser les Secrets GitHub pour stocker les informations sensibles (clés API, mots de passe, clés privées SSH, etc.) plutôt que de les écrire en clair dans le code ou les fichiers de configuration du dépôt ?

Quels sont les risques de sécurité majeurs si un secret est exposé (par exemple, dans les logs publics, ou s'il est commité par erreur dans le dépôt) ?

Configuration des Secrets dans GitHub :

Dans l'interface web de votre dépôt GitHub, naviguez vers Settings -> Secrets and variables -> Actions.

Créez un secret nommé FAKE_API_TOKEN et donnez-lui une valeur fictive de votre choix.

Créez un second secret nommé ENV_TYPE avec la valeur DEV.

Intégration des secrets dans le workflow :

Modifiez le fichier .github/workflows/ci_basic.yml.

Ajoutez une nouvelle étape qui affichera la valeur de vos secrets et de variables d'environnement dans les logs du workflow. Assurez-vous d'inclure une variable d'environnement personnalisée (non-secrète) et une variable d'environnement prédéfinie par GitHub (par exemple, RUNNER_OS).

Observation : Commitez et poussez la modification. Exécutez le workflow et examinez attentivement les logs. Remarquez comment GitHub censure automatiquement la valeur des secrets dans les logs pour prévenir les fuites.

Simuler un échec contrôlé pour la gestion d'incidents :

Modifiez le script app/check_app.sh pour qu'il renvoie intentionnellement un code de sortie d'échec (exit 1).

Poussez cette modification. Observez le workflow échouer.

Analyse des logs : Apprenez à diagnostiquer les erreurs en lisant les logs du workflow. C'est une compétence clé en cyber pour l'analyse post-incident.

Important : Après cette expérience, remettez le script app/check_app.sh en état de succès (exit 0) pour les prochaines étapes.

Job 4 : Préparation du Serveur de Déploiement Sécurisé avec Simulation GitHub Actions

---

En raison des contraintes réseau (portail captif), nous utiliserons une approche de simulation qui valide tous les concepts de sécurité sans nécessiter d'accès externe à votre VM.

Configuration initiale de la VM cible :

Votre VM Debian doit être à jour avec SSH activé et Python installé. Vous devez configurer un utilisateur de déploiement à moindres privilèges.

Créez un nouvel utilisateur non-root sur votre VM, qui sera exclusivement utilisé pour les opérations de déploiement automatisées. Choisissez un nom d'utilisateur pertinent (par exemple, deployuser).

configurez les droits sudo pour cet utilisateur afin qu'il puisse exécuter les commandes nécessaires en tant que root sans demander de mot de passe.

Testez que l'utilisateur peut exécuter des commandes avec sudo sans demander de mot de passe.
Pourquoi est-il fondamental d'utiliser un utilisateur dédié avec le principe du moindre privilège pour les opérations automatisées de déploiement ? Quels sont les risques de sécurité majeurs à utiliser l'utilisateur root directement pour le déploiement ?

Génération et installation d'une clé SSH dédiée :
Sur votre machine de travail (locale), générez une nouvelle paire de clés SSH spécifiquement pour cet exercice. N'utilisez PAS vos clés SSH personnelles existantes

La clé privée ne doit pas être protégée par une phrase secrète (requis pour l'automatisation) ,utilisez une taille de clé d'au moins 2048 bits
Nommez les fichiers de manière explicite (ex: github_deploy_key)

Copiez la clé publique générée sur votre VM, dans le fichier ~/.ssh/authorized_keys de l'utilisateur deployuser. Assurez-vous que les permissions des fichiers et répertoires SSH sont correctement configurés.

Test de connexion :
Testez la connexion SSH en utilisant cette nouvelle clé et l'utilisateur deployuser pour confirmer que la connexion s'établit sans demander de mot de passe.

Pourquoi est-il crucial de ne pas utiliser ses clés SSH personnelles pour les déploiements automatisés ?

Quels sont les avantages d'avoir des clés dédiées par projet/environnement ?

Configuration des secrets GitHub

Dans votre dépôt GitHub, accédez à Settings → Secrets and variables → Actions.

Créez les secrets suivants :

SSH_PRIVATE_KEY : Collez le contenu complet de votre clé privée générée

SERVER_IP : L'adresse IP de votre VM (locale ou accessible depuis votre réseau)

SERVER_USER : Le nom de l'utilisateur de déploiement créé sur la VM

Pourquoi utilise-t-on les GitHub Secrets plutôt que de stocker les informations sensibles directement dans le code ?

Quels sont les risques d'exposition des credentials ?

Création du workflow de simulation GitHub Actions :

Créez un workflow GitHub Actions qui simule un déploiement sécurisé en validant tous les éléments de configuration sans nécessiter d'accès réseau externe à votre VM.

Le workflow doit inclure les étapes suivantes :

Validation de la clé SSH privée

Vérifier que la clé est au bon format , contrôler la taille de la clé (minimum 2048 bits),confirmer l'absence de passphrase

Validation de la configuration serveur

Vérifier que tous les secrets nécessaires sont définis .Valider le format des variables d'environnement

Simulation de connexion SSH :

Simuler les commandes qui seraient exécutées sur le serveur

Tester la configuration SSH (sans connexion réelle)

Afficher les commandes de déploiement qui seraient lancées

Contrôles de sécurité

Vérifier les bonnes pratiques de sécurité

Valider la robustesse de la configuration

Structure du workflow

Créez le fichier .github/workflows/secure-deployment-simulation.yml dans votre dépôt. Le workflow doit se déclencher sur les push et pull requests vers la branche principale.

Dans un environnement de production réel, quelles mesures supplémentaires pourriez-vous mettre en place pour sécuriser davantage le processus de déploiement automatisé ?

Tests et validation locale

Créez un script de test qui valide localement votre configuration avant de la pousser dans GitHub. Ce script doit tester la connectivité SSH, les permissions sudo, et simuler quelques opérations de déploiement basiques.

Votre script de test local doit vérifier :

La connexion SSH avec la nouvelle clé

L'exécution de commandes avec sudo

La création de répertoires d'application

La vérification des services système disponibles

Job 5 : Création du Playbook Ansible de Déploiement

Structure du répertoire Ansible :

À la racine de votre dépôt, créez un nouveau dossier nommé ansible/.

Création du Playbook de déploiement (ansible/deploy.yml) :

Dans le dossier ansible/, créez un fichier nommé deploy.yml.

Ce playbook Ansible sera chargé de déployer votre application (du dossier app/) sur le serveur. Il doit inclure les tâches suivantes :

Installer le serveur web Nginx.

Créer le répertoire de destination de l'application sur le serveur (par exemple, /var/www/html/tp-app).

Copier les fichiers de votre application (du dossier app/ de votre dépôt) vers ce répertoire de destination sur le serveur.

Configurer le serveur web pour servir votre application via un template Jinja2.

Redémarrer le service du serveur web pour appliquer les changements.

Création du template Nginx (ansible/nginx_vhost.conf.j2) :

Dans le dossier ansible/, créez un fichier nginx_vhost.conf.j2. Ce fichier est un template Jinja2 qui sera utilisé par Ansible pour configurer Nginx. Il devra contenir la configuration de base d'un hôte virtuel Nginx, en utilisant une variable Ansible pour l'adresse IP du serveur.

Test local du Playbook (fortement recommandé pour le débogage) :

Sur votre machine locale (là où vous avez cloné le dépôt), créez un fichier inventory.ini temporaire à la racine de votre dépôt, pour tester Ansible localement. Ce fichier doit pointer vers votre VM en utilisant l'utilisateur deployuser et la clé SSH dédiée.

Exécutez le playbook Ansible en ligne de commande.

Vérifiez que l'application est bien déployée et accessible via le navigateur sur l'IP de votre VM.

(Optionnel) : Si vous ne souhaitez pas versionner votre fichier inventory.ini local, ajoutez-le à votre .gitignore.

Job 6 : Intégration d'Ansible dans le Workflow GitHub Actions (CD)

Mise à jour du workflow CI/CD complet :

Dans votre dépôt, renommez le fichier .github/workflows/ci_basic.yml en .github/workflows/ci_cd_full.yml.

Modifiez-le pour ajouter la phase de déploiement. Votre workflow contiendra désormais deux jobs distincts :

Le job build-and-test (votre CI).

Un nouveau job nommé deploy, qui dépendra du succès du job build-and-test (needs: build-and-test).

Le job deploy devra inclure les étapes suivantes :

Checkout repository : Pour cloner le dépôt.

Set up SSH agent : Utilisez une action GitHub (webfactory/ssh-agent@v0.9.0) pour charger la clé SSH privée depuis le secret GitHub SSH_PRIVATE_KEY dans l'agent SSH du runner.

Create Ansible inventory file : Créez dynamiquement un fichier inventory.ini sur le runner, en utilisant les secrets GitHub SERVER_IP et SERVER_USER. Assurez-vous de définir les bonnes permissions sur ce fichier pour des raisons de sécurité.

Installez Ansible : Installez Ansible sur le runner (via pip).

Exécutez le playbook de déploiement Ansible. Passez les secrets (SERVER_IP, SERVER_USER) en tant que variables d'environnement à l'exécution d'Ansible, afin qu'ils puissent être utilisés dans votre template Jinja2.

Premier déclenchement du pipeline complet :
Committez le fichier .github/workflows/ci_cd_full.yml et le dossier ansible/ sur la branche main.

Suivez attentivement l'exécution des deux jobs (CI puis CD) dans l'onglet Actions de votre dépôt GitHub.

Une fois le déploiement terminé avec succès, vérifiez que votre application est bien accessible via un navigateur sur l'adresse IP de votre VM.

Job 7 : Test du Cycle CI/CD Complet et Rapide

Modification de l'application :

En local, modifiez le fichier app/index.html. Changez son contenu pour indiquer une nouvelle version de votre application (exemple, "V2.0 Déployé !").

Déclenchement du pipeline :

Committez cette modification à votre dépôt local.

Poussez la modification sur la branche main de votre dépôt GitHub.

Validation du déploiement :

Observez le workflow se déclencher automatiquement et s'exécuter.

Après le succès du déploiement, rafraîchissez la page dans votre navigateur web pour confirmer que la nouvelle version de votre application est désormais visible sur le serveur distant.

Comparez cette rapidité de déploiement avec une approche manuelle. Quels sont les avantages en termes de sécurité de cette automatisation (réduction des erreurs humaines, traçabilité des déploiements) ?

Job 8 : Bilan et Perspectives en Cyberdéfense

Récapitulatif des étapes clés et des outils utilisés.
Mise en lumière de l'automatisation complète de l'intégration et du déploiement.

Pour aller plus loin en sécurité CI/CD :

Scans de vulnérabilités (SAST/SCA) : Comment intégrer des outils d'analyse de code statique (SAST) et d'analyse de la composition logicielle (SCA) dans la phase de CI pour détecter les vulnérabilités dans le code source et les dépendances tierces.

Tests de sécurité dynamiques (DAST) : Possibilité d'intégrer des tests d'intrusion automatisés (DAST) après le déploiement.

Audits et logs : Comment les logs détaillés de GitHub Actions et d'Ansible peuvent être utilisés pour l'audit de sécurité et la détection d'anomalies ou de tentatives d'intrusion.

Gestion des environnements : Stratégies pour protéger les environnements de production (déploiements manuels approuvés, environnements isolés, contrôle d'accès strict).

Stratégies de Rollback : L'importance d'avoir une procédure de retour arrière rapide et fiable en cas de déploiement problématique.

Compétences visées

- Supervision et optimisation d'infrastructures
- Conception & évolution d'une infrastructure
- Déploiement & mise en production

Rendu

Le projet est à rendre sur votre github :

https://github.com/prenom-nom/gtacd

avec une documentation et copie d'écrans des différents scripts / logs

Base de connaissances

- Documentation principale GitHub Actions
- Syntaxe de workflow pour GitHub Actions
- Utilisation des secrets dans les workflows
- Événements déclencheurs de workflows
- Actions officielles pour les workflows