Docker au service du DevSecOps

Date : 14 avril 2023
Speakers : Carmen Piciorus (Expert Sécurité à La Poste)
Format : Conférence (45mn)

Carmen est Présidente de l'association SignalSpam
Cette Experte en Sécurité énumère les technos utilisées à La Poste : Spring Boot, PHP Sypohny, Flutter, GitLab, Kafka, Elastic, Kubernetes et OpenShift, Consul, Terraform et Vault HashiCorp.

Vaincre la peur ?

Lors de son voyage en Floride, en se baladant en forêt, Carmen a pu croiser des lamentins, des ratons laveurs, des serpents et même des alligators. Fil conducteur de sa présentation, ce chemin dans la forêt lui ont fait penser aux attaques cyber qui peuvent être imprévisibles. Déployer des applications dans le Cloud (privé et publique) est dangereux et il faut bien se préparer. Docker nous y aide.

Docker et les développeurs

- Catalogue d'images internes et recommandées. L'équipe technique transverse a créé des images de base, choix des outils, scan de sécurité (Anchore). Ces images sont publiées dans un catalogue. Les images sont versionnées et taggées. Elles sont mises à jour automatiquement tous les jours.
- Le développeur utilise ces images dans son Dockerfile
- Le pipeline CI/CD GitLab scanne l'image Docker avant de pousser l'image dans Nexus puis OpenShift
- Son équipe cherche à diminuer la taille de l'image finale (jusqu'à 50% de réduction).
- Bonnes pratiques d'écriture du Dockerfile afin de mettre le code qui change dans le dernier layer afin d'utiliser au maximum le cache
- Qui écrit le Dockerfile ? Le dév avec en support l'équipe DevOps et la sécu
- Le build multi-stage permet de réduire la taille de l'image finale. Exemple d'une application node buildée dont les ressources statiques sont copiées sur l'image Apache
- Utilisation de docker-compose afin de démarrer plusieurs services en même temps : base de données, backend, frontend. Utilisation du filesystem pour ne pas perdre ses données.
- Builldpacks : transforment le code en image Docker. Supporte Java, Node, GO et Ruby. Layer des images : stack, runtime, dependency et app. Image non utilisée en production car non maitrisée. Utilisé sur des projets legacy.

Docker et l'équipe Sécurité

- La SSI a peur des menaces venant de l'extérieur, notamment sur les veilles applications
- Outils de détection de vulnérabilité. Scan statique du code souce avec owasp dependency check et le SAST : secret detection registry.gitlab.com/security-products.secret-definition :
- Cherche l'usage de certaines licences interdites

- Docker est utilisé par la Sécurité pour récupérer les images des outils GitLab **Secret Detection** et **OWASP Dependency**. Ce dernier cherche les dépendances dans target. Note de 7.
- Scan dynamique des applications à l'aide de l'OWASP Zed Attack Proxy (ZAP) : utilisation de l'image owasp/zap2docker-bare:latest. Des requêtes HTTP sont envoyées pour vérifier les cookies ReadOnly ou les failles XSS
- Outils de détection des vulnérabilités : scan du catalogue d'images et des images du projet. La plupart des vulnérabilités sont trouvées dans la couche applicative.
- Fuite de données : pas de secrets dans les configmap. Création de rôle de k8s pour limiter l'accès au cluster à certaines personnes. Certaines personnes ont accès au secret, d'autres au configmap en fonction de leur rôle. Utilisation de la commande *docker image inspect*. <image> :<tag> pour vérifier la partie env dans laquelle certains développeurs auraient pu placer des données sensibles. Exemple de fuite de credentials git sur Docker Hub.
- Les résultats du docker inspect sur l'image tournant sur OpenShift de prod sont centralisés dans Elastic. Recroisement des données : dette technique de l'application, métriques Sonar … -> fiche de santé de l'application. L'outil Anchore sort les SBOM permettant de corriger les vulnérabilités critiques.
- Difficulté de savoir quelle image Docker est utilisée en production. Utilisation de oc describe pod => nom des images avec le digest sha256 calculé sur les layers. Conseil de garder le digest de l'image Docker utilisée et pas seulement le tag.
- Bonnes pratiques : ne pas utiliser l'utilisateur root pour faire tourner une appli dans le Dockerfile

Docker et les Ops

- Indisponibilités, dépendances, résilience : peur des attaques pouvant amener à l'indisponibilité totale. Nécessité de savoir remonter tout un environnement.
- Beaucoup d'inter-dépendances entre applications : utilisation de Kafka pour l'asynchrone.
- L'ops utilise Docker sans le savoir : il manipule kubectl et pas de docker run

Prochaines étapes à la Poste

- Usage des buildpacks pour le dév et équipes transverses
- L'auto-devops : pipeline CI
- Base de connaissance ELK : état de santé des applications
- Cyber-résilience