

Contrat et recommandations pour les images d'application déployées sur la plateforme Atlas

Ce document est un sous-ensemble de la Doctrine de Sécurité de la plateforme, spécialisé sur la construction d'images applicatives qui seront hébergées sur la plateforme.

Il se décompose en trois catégories distinctes mais complémentaires :

1. Les exigences fortes, nécessaires à la sécurisation des plateformes hébergeant ces images afin de garantir une isolation des applicatifs, ainsi que les attentes concernant la conception des applications nécessaires pour les plateformes d'orchestration cloud-native. Ces éléments sont modélisés à partir des standards établis de l'industrie, et ne présentent pas de possibilité de débrayage.
2. Les futures exigences, en cours de construction, qui se saisissent des enjeux de sécurité nouvellement identifiés dans l'industrie, mais dont les solutions et l'outillage ne sont pas encore suffisamment convergents pour offrir un chemin clair de conformité. Pour ces attentes, il est attendu une sensibilisation et un travail d'étude en parallèle, dans l'attente d'une uniformisation de l'industrie qui amènera à une clarification de ces contraintes dans une version future.
3. Les recommandations, suggestions émises par la plateforme pour respecter les exigences mentionnées. Ces recommandations sont présentes à titre consultatif, et représentent ce que les auteurs de ce document pensent être des bonnes pratiques et chemins de simplifications. Ainsi, étant donné que l'impact causé par le non-respect de ces bonnes pratiques ne concernerait que la bonne santé de l'application et non la plateforme d'hébergement en elle-même, ce document n'émet pas de contrainte forte à leur sujet.

1. Exigences obligatoires sur les images (Contrat ferme)

1.1. Sécurité d'exécution

- Les containers **ne doivent jamais s'exécuter en tant que root**.
 - Il est obligatoire de définir explicitement un `USER` avec un UID non-root.
 - Aucune escalade de privilèges au runtime (ex: `su`, `sudo`).
- Conformité avec le profil **Restricted** du **Kubernetes Pod Security Standard (PSS)** :
 - Interdiction des containers privilégiés.
 - Interdiction d'accès aux namespaces hôte (network, PID, IPC).
 - `allowPrivilegeEscalation` doit être positionné à `false`.
 - Le profil Seccomp doit être `RuntimeDefault`.
 - Un filesystem root en lecture seule est recommandé et pourra devenir obligatoire.

Exemple (✅ Conforme)

```
FROM gcr.io/distroless/base
USER 1000
CMD ["myapp"]
```

Exemple (❌ Non conforme)

```
FROM ubuntu:20.04
CMD ["bash"]
```



Le respect de cette exigence est nécessaire afin de garantir qu'une image malicieuse n'a pas les capacités d'affecter d'autres applicatifs ou la plateforme dans son ensemble, et est donc le socle fondamental des contraintes non-négotiables.

Cette exigence est tirée directement du profil d'exécution "Restricted" de Kubernetes, qui représente le niveau le plus strict des profils standards, et qui sera appliqué automatiquement sur les images déployées sur les plateformes d'hébergement.

<https://kubernetes.io/docs/concepts/security/pod-security-standards/>

1.2. Comportement du filesystem

- Les applications **ne doivent jamais supposer que le filesystem root est modifiable**.
 - Utiliser **/tmp** ou des volumes `emptyDir` pour les données éphémères.
 - Les logs et données applicatives doivent être redirigés vers `stdout/stderr`
- Il est strictement interdit de stocker de l'état persistant dans l'image ou sur le disque local du container.

Exemple (✅ Conforme)

```
CMD ["myapp"]
```

Exemple (❌ Non conforme)

```
CMD ["myapp", "--log-file=/var/log/myapp.log"]
```



Les orchestrateurs de conteneurs déploient dynamiquement plusieurs instances de l'image, et peuvent à tout moment éteindre et relancer ces instances sur d'autres noeuds.

Pour cette raison, les applications cloud-native ne peuvent pas partir du principe que le stockage sous-jacent restera stable d'une requête à une autre.

<https://12factor.net/processes>

1.3. Réseau et services

- Les applications ne doivent jamais supposer des adresses IP ou des noms DNS statiques.
- Les applications **doivent écouter sur** `0.0.0.0` **et non** `localhost` .
- Les endpoints de santé (readiness et liveness probes) **doivent être définis et exposés**.
- Ces endpoints doivent être représentatifs de l'état réel de l'application et peu coûteux.



Similairement à la contrainte précédente, la nature dynamique de l'orchestration des conteneurs rend impossible de prédire l'adresse IP qui sera portée par le conteneur.

Les endpoints de santé sont nécessaires pour le bon fonctionnement de la fonctionnalité de load balancing de l'orchestrateur, qui se base sur les informations de ces endpoints.

<https://kubernetes.io/docs/concepts/configuration/liveness-readiness-startup-probes/>

1.4. Conformité au modèle 12-Factor App (Obligatoire)

Les applications doivent être conçues **conformément aux principes 12-Factor App**.

Spécifiquement :

- **La configuration doit être injectée via des variables d'environnement, ou alternativement par des fichiers injectés dans les conteneurs par la plateforme, les secrets doivent notamment privilégier le choix des fichiers injectés. Dans les deux cas, ces variables ou fichiers de configuration qui varient d'un environnement à un autre ne font pas partie de l'image, mais sont injectés dans le conteneur lors de son lancement.**
- **Les processus doivent être stateless et sans partage de contexte.**
- **Les logs doivent aller exclusivement sur stdout/stderr.**
- **Toute donnée persistante doit être externalisée vers des services dédiés.**
- **Les processus doivent pouvoir être arrêtés et redémarrés sans impact visible des utilisateurs (disposability).**

Exemple (✅ Conforme)

```
# In Dockerfile, config.yaml is not part of the image
CMD ["myapp", "--config-file=/etc/myapp/config.yaml"]

# When launching the container, e.g. via a pod manifest
apiVersion: v1
kind: Pod
metadata:
  name: example-myapp
spec:
  containers:
    - name: myapp
      image: myapp
      volumeMounts:
        - name: config
          mountPath: "/etc/myapp"
          readOnly: true
  volumes:
```

```
- name: config
  configMap:
    name: myapp-config
    items:
      - key: "config.yaml"
        path: "config.yaml"
```

Exemple (❌ Non conforme)

```
# In Dockerfile, env-specific config is baked into the image
ADD ./envs/prod/config.yaml /etc/myapp/config.yaml
CMD ["myapp", "--config-file=/etc/myapp/config.yaml"]
```



Cette exigence complète et synthétise les recommandations précédentes, qui forme des sous-ensembles des exigences des 12 factors.

Les 12 factors sont un document fondamental dans la conception des applications cloud-ready, initialement conçu par Heroku comme leur propre jeu d'exigences pour l'accueil d'applications sur leur plateforme.

Ce document est aujourd'hui une référence de l'industrie, et son respect est essentiel pour garantir non seulement le bon fonctionnement de l'application, mais également sa portabilité sur d'autres plateformes d'hébergement cloud-native.

<https://12factor.net/>

2. Exigences futures

Ces exigences se positionnent principalement autour du sujet de Supply Chain Security, inspirés notamment par le framework SLSA. Elles concernent l'augmentation des images produite en leur attachant des **attestations**, permettant de prouver cryptographiquement des éléments de sécurité.



La Supply Chain Security est le domaine de sécurité se concentrant sur l'intégrité du parcours ayant amené à la construction de l'image et sa mise à disposition à la plateforme.

Cet enjeu a été démocratisé à la suite de l'affaire SolarWinds en 2023, suite à quoi des travaux ont été initiés par l'industrie pour établir des standards de protection contre ce type d'attaque.

En particulier, le framework SLSA (<https://slsa.dev/>), encore en cours de conception, a pour objectif d'établir une checklist et des niveaux de maturités vis-à-vis de cet enjeu.

2.1. Build process et attestations de provenance

- Le processus de build doit produire des **attestations de provenance conformes à in-toto**.
- Les attestations doivent inclure :
 - L'identification explicite du pipeline CI ayant généré l'image.
 - Les résultats documentés des analyses de code statique et de détection d'injections.
- Les attestations doivent être incluses dans les métadonnées de l'image et seront vérifiées par la plateforme lors du déploiement.
- Le framework méthodologique SLSA doit servir de modèle pour la production de ces attestations

2.2. SBOM (Software Bill of Materials)

- Chaque image doit embarquer un **SBOM au format SPDX ou CycloneDX**.

- Le SBOM doit être intégré dans l'image lors du build.
- La plateforme analysera les SBOM et bloque les images contenant des vulnérabilités critiques.

2.3. Scans de sécurité pré-build (Responsabilité applicative)

- Les équipes applicatives doivent réaliser :
 - Une analyse statique de code de type SAST (SonarQube, Semgrep, Talisman, ...).
 - Une analyse dynamique de type DAST
- Les résultats doivent être documentés et inclus dans les attestations.

3. Recommandations (Bonnes pratiques non bloquantes)

3.1. Utilisation d'images minimales ou distroless

- Privilégier les images distroless, minimalistes ou basées sur `scratch`.
- Éviter les images généralistes type `ubuntu`, `debian` sauf cas strictement justifiés.

Exemple (Recommandé)

```
FROM gcr.io/distroless/java17
```

Exemple (À éviter)

```
FROM ubuntu:22.04
```




Nous recommandons la lecture des ressources écrites par Chainguard, qui est un acteur majeur dans l'écosystème distroless.

<https://www.chainguard.dev/unchained/minimal-container-images-towards-a-more-secure-future>

3.2. Tagging et versioning des images

- Ne jamais utiliser le tag `latest`.
- Utiliser des tags immuables et versionnés (`v1.2.3`, commit SHA, etc.).



La version pointée par `latest` pouvant évoluer sans prévenir, son utilisation amène à une perte de reproductibilité, une incapacité de rollback, et le risque d'introduction de bugs si `latest` est utilisé pour référencer une image d'un fournisseur tiers.

3.3. Outils de build automatisé sans Dockerfile

Nous recommandons fortement l'usage d'outils de build modernes permettant de produire des images sans écriture de Dockerfile personnalisé.

Exemples :

- **Cloud Native Buildpacks (CNCF)**
- **Spring Boot Build Image**
- **ko (Go)**
- **Jib (Java)**

Exemple (Buildpacks avec pack CLI)

```
pack build myapp --builder paketobuildpacks/builder:base
```

Exemple (Spring Boot)

```
./mvnw spring-boot:build-image
```

Exemple (ko pour Go)

```
KO_DOCKER_REPO=myregistry/myapp ko build ./cmd/myapp
```

Sauf cas spécifique ou contrainte technique avérée, il est préférable de s'abstenir d'écrire des Dockerfiles manuellement et de s'appuyer sur ces outils.

Si l'usage de Dockerfile est inévitable, privilégier les builds multi-stage pour garantir des images finales minimalistes et sans dépendances de build.



L'expérience montre que la conception d'un Dockerfile "correct" et conforme aux bonnes pratiques de l'industrie cache une complexité et une demande d'expertise plus intense que ce qu'il pourrait paraître, et représente une charge cognitive qui pourrait être abstraite.

https://www.youtube.com/watch?v=5uAe_PGWnxw















4. Attribution et responsabilité

En complément de la doctrine de sécurité Atlas, ce contrat rappelle que :

- La conformité aux exigences listées dans ce document est **une condition préalable et obligatoire pour tout déploiement sur la plateforme.**
- Les exigences relatives aux attestations de provenance, SBOM et scans pré-build **sont placées sous la responsabilité exclusive des utilisateurs de la plateforme et leurs prestataires.**
- Toute violation de ces exigences, ou toute compromission liée à des vulnérabilités introduites dans le code source ou les images, **sera systématiquement attribuée à l'équipe applicative.**

- La plateforme se réserve le droit de bloquer, supprimer, ou isoler toute image ou workload non conforme, sans préavis, en cas de détection d'une non-conformité critique.

5. Checklist de conformité des images applicatives

Exigence	Catégorie	Conforme ? (Oui/Non)	Référence
Image exécutée en tant qu'utilisateur non-root	 Exigence forte		1.1
Aucun container privilégié ou accès aux hôtes	 Exigence forte		1.1
<code>allowPrivilegeEscalation: false</code> activé	 Exigence forte		1.1
Profil Seccomp <code>RuntimeDefault</code>	 Exigence forte		1.1
Aucune écriture sur le root filesystem	 Exigence forte		1.2
Logs exclusivement via stdout/stderr	 Exigence forte		1.2, 1.4
Aucun état ou configuration embarqué dans l'image	 Exigence forte		1.2, 1.4
Application bind sur <code>0.0.0.0</code> (pas <code>localhost</code>)	 Exigence forte		1.3
Endpoints de santé exposés si applicables	 Exigence forte		1.3
Conformité 12-Factor App	 Exigence forte		1.4
Configuration via variables d'environnement	 Exigence forte		1.4
Process stateless et redémarrables	 Exigence forte		1.4
Build process automatisé, reproductible et attestations in-toto intégrées	 Exigence future		2.1
SBOM intégré dans l'image (format SPDX ou CycloneDX)	 Exigence future		2.2

Scans de sécurité pré-build réalisés et documentés	💡 Exigence future		2.3
Utilisation d'images minimales/distroless	✅ Recommandation		3.1
Utilisation de tags immutables	✅ Recommandation		3.2
Construction d'image sans Dockerfile	✅ Recommandation		3.3

Pour toute question sur les exigences supplémentaires de runtime, network policies, observabilité et audit, merci de vous référer à la [Doctrine de Sécurité Atlas](#).