

EXAMEN DEVOPS*Durée 1h30**Aucun document n'est autorisé***Consignes :**

- Lisez attentivement chaque question.
- Pour les questions « 1 seule réponse », cochez une seule option.
- Pour les questions « réponses multiples », cochez toutes les options correctes.

Q1 (1 seule réponse)

Dans un pipeline DevSecOps, quel énoncé est le plus correct à propos d'un Quality Gate ?

- A. C'est un contrôle qualité qui peut être ignoré en urgence.
- B. C'est un contrôle qualité/sécurité pouvant être bloquant selon la politique définie.
- C. C'est un scan de conteneur uniquement.
- D. Ni A ni B.

Q2 (1 seule réponse)

La distinction la plus exacte entre Continuous Delivery et Continuous Deployment est :

- A. Delivery construit des artefacts, Deployment non.
- B. Deployment inclut forcément une validation humaine en production.
- C. Delivery peut nécessiter un déclenchement/une approbation avant production ; Deployment déploie automatiquement si tout est conforme.
- D. Aucune des réponses.

Q3 (1 seule réponse)

Quel choix illustre le mieux « Build once, run anywhere » dans un contexte CI/CD ?

- A. Recompiler le binaire sur chaque environnement (DEV, STG, PROD).

- B. Construire un artefact immuable une fois, puis le promouvoir tel quel jusqu'en production.
- C. Construire en production pour éviter les différences d'environnement.
- D. Toutes les réponses.

Q4 (1 seule réponse)

Quel risque est le plus directement réduit par l'usage de conteneurs Docker dans une chaîne CI/CD ?

- A. Les différences d'exécution (« ça marche sur ma machine »).
- B. Les conflits de merge Git.
- C. Les erreurs de DNS public.
- D. Ni A ni B.

Q5 (réponses multiples)

Parmi les propositions suivantes, lesquelles décrivent correctement « Image » vs « Conteneur » ? (plusieurs bonnes réponses)

- A. Une image est un modèle immuable ; un conteneur est une instance en exécution.
- B. Un conteneur est toujours immuable mais une image est toujours modifiable.
- C. Un conteneur peut être supprimé/recréé à partir de la même image.
- D. Aucune des réponses.
- E. Toutes les réponses.

Q6 (réponses multiples)

Après un déploiement, quels signaux sont les plus pertinents pour décider « stop / rollback » ? (plusieurs bonnes réponses)

- A. Taux d'erreurs et codes 5xx.
- B. Latence p95/p99.
- C. Échec des probes readiness/liveness.
- D. Nombre de commits effectués aujourd'hui.
- E. Toutes les réponses.
- F. Ni A ni B.

Q7 (réponses multiples)

Après déploiement, quels signaux sont les plus fiables pour décider rollback/stop canary (dans les 10 premières minutes) ? (plusieurs bonnes réponses)

- A. Hausse du taux 5xx / erreurs applicatives.
- B. Dégradation de la latence p95/p99.
- C. Échecs readiness/liveness (probes) + redémarrages anormaux.
- D. Augmentation du nombre de commits dans la journée.
- E. « Tout va bien sur ma machine ».

Q8 (1 seule réponse)

Ton Quality Gate exige une couverture $\geq 80\%$ sur le « new code ». Le pipeline échoue. Quelle action est la plus pertinente ?

- A. Désactiver temporairement la Quality Gate.
- B. Ajouter/renforcer les tests unitaires et corriger la configuration de couverture (ex. JaCoCo) pour que SonarQube lise le rapport.
- C. Ajouter « || true » à l'étape Sonar.
- D. Baisser le seuil à 10%.

Q9 (1 seule réponse)

Quelle combinaison décrit le mieux le rôle des outils ?

- A. SonarQube = vulnérabilités OS ; Snyk = style de code.
- B. SonarQube = qualité + bugs + vulnérabilités dans le code ; Snyk = dépendances + conteneurs (SCA / container).
- C. SonarQube et Snyk font uniquement du déploiement.
- D. Aucune des réponses.

Q10 (réponses multiples)

Tu es DevOps Engineer. Après un déploiement, tu observes :

- montée des erreurs 5xx,
- latence qui explose,
- pods qui redémarrent en boucle.

Que fais-tu en priorité ? (plusieurs bonnes réponses)

SUPnum

- A. Stopper/pauser le rollout (si canary/rolling) et déclencher un rollback vers la version stable.
- B. Ignorer et attendre 1 heure : ça va se stabiliser.
- C. Vérifier logs applicatifs + métriques (CPU/Mem/GC/DB) + événements Kubernetes.
- D. Désactiver les probes readiness/liveness pour empêcher les redémarrages.
- E. Ouvrir un incident, notifier les parties prenantes, et sécuriser le périmètre (freeze des déploiements).

Q11 (1 seule réponse)

Un directeur te demande : « déploie quand même, même si la Quality Gate échoue ». Quelle est la réponse la plus professionnelle ?

- A. OK, je force et on corrigera après.
- B. Je force mais je cache le résultat du pipeline.
- C. Je refuse le bypass automatique ; je propose un processus d'exception formel (approbation, risque documenté, plan de rollback, patch immédiat).
- D. Je désactive SonarQube définitivement.

Q12 (1 seule réponse)

Le fait que la Quality Gate soit « bloquante » est principalement imposé par :

- A. Le code Java (annotations).
- B. La configuration du serveur Jenkins uniquement.
- C. La logique du pipeline (Jenkinsfile) + la Quality Gate définie dans SonarQube.
- D. Le Dockerfile.

Q13 (1 seule réponse)

Les 4 métriques DORA incluent :

- A. CPU, RAM, disque, réseau.
- B. Lead time, deployment frequency, change failure rate, MTTR.
- C. Bugs, features, commits, reviews.
- D. Nombre de serveurs, coût, disponibilité, incidents.

Q14 (1 seule réponse)

En Git, « Trunk-based Development » implique généralement :

- A. Des branches longues (plusieurs mois).
- B. Des branches courtes depuis main, main protégée, revue + pipeline.
- C. Pas de PR/MR.
- D. Push direct sur main obligatoire.

Q15 (1 seule réponse)

Le cours présente « Plan → Code → Build » comme :

- A. La phase d'exploitation (Ops).
- B. La partie « Développement Continu » (passerelle vers CI).
- C. La définition de Continuous Deployment.
- D. Une étape optionnelle qui ne doit pas être automatisée.

Q16 (réponses multiples)

Tu dois améliorer la qualité du code dès la phase Build. Quelles pratiques sont les plus alignées ? (plusieurs bonnes réponses)

- A. Code propre (lisible, maintenable).
- B. Conventions de style + linting/formatage.
- C. Tests unitaires écrits en parallèle du code.
- D. Ignorer les tests unitaires pour gagner du temps.
- E. Toutes les réponses.

Q17 (réponses multiples)

Dans une stratégie « Trunk-based Development », on retrouve : (plusieurs bonnes réponses)

- A. Branches courtes ($\leq 2-3$ jours) depuis main.
- B. Main protégée.
- C. Revue obligatoire avant fusion.
- D. Push direct sur main encouragé.
- E. Pipeline de validation automatique.

Q18 (réponses multiples)

Parmi les causes qui ont poussé à adopter DevOps, on retrouve : (plusieurs bonnes réponses)

- A. Les silos entre Dev et Ops.
- B. Les processus manuels créant des goulots d'étranglement.
- C. L'absence de framework front-end moderne.
- D. Les problèmes récurrents (erreurs, lacunes).
- E. Toutes les réponses.

Q19 (1 seule réponse)

Dans le cycle présenté, quelle séquence correspond le mieux à la partie « architecture applicative actuelle » (vue étapes) ?

- A. Plan → Code → Build → Test → Monitor.
- B. Plan → Code → Test → Build → Deploy.
- C. Code → Plan → Build → Deploy → Operate.
- D. Aucune des réponses.

Q20 (réponses multiples)

Commande : ``snyk test --severity-threshold=high || true``
Qu'est-ce que cela implique? (plusieurs bonnes réponses)

- A. Le stage ne cassera jamais, même si des vulnérabilités High/Critical sont trouvées.
- B. Cela viole le principe de « gates bloquants ».
- C. C'est recommandé en production pour éviter les interruptions.
- D. Cela masque le code retour de Snyk.

Q21 (réponses multiples)

Tu utilises ``waitForQualityGate()`` dans un pipeline. Quelles affirmations sont correctes ? (Plusieurs bonnes réponses)

- A. ``waitForQualityGate()`` nécessite généralement un webhook SonarQube → Jenkins pour éviter une attente inutile.
- B. ``waitForQualityGate()`` retourne un objet avec un champ ``status`` typiquement ``OK`` / ``ERROR``.
- C. ``waitForQualityGate()`` fonctionne même si l'analyse SonarQube n'a jamais été lancée.
- D. On peut bloquer le déploiement via ``if (qg.status != 'OK') { error(...) }``.