

Corrigé PSE 2018 : NuBo, mémoire et élasticité

Proposition de corrigé — non officielle. Examen professionnel 2018, épreuve écrite n° 2, Windows et UNIX. La proposition choisit Linux et Python 3 et traite les trois parties.

[Sujet officiel](#) · [Annales de ce concours ou examen](#)

Hypothèses : Linux et Python 3

La proposition choisit **UNIX/Linux et Python 3**. NuBo est un processus identifié par son PID ; les mesures portent sur la machine virtuelle qui l'exécute. Les compteurs de `/proc` utilisés ci-dessous existent dans le contexte de 2018. Les interfaces HTTP présentées sont une proposition de contrat, car le sujet n'en fournit pas les chemins.

Partie 1 — Mémoire et mesures

Deux modes d'allocation

Une application utilise notamment la **pile**, pour les cadres d'appel et certaines variables locales, et le **tas**, pour des allocations dynamiques dont la taille et la durée de vie peuvent dépasser un appel. L'espace virtuel du processus ne coïncide pas avec toute sa mémoire réellement résidente en RAM : une réservation virtuelle ne prouve pas que toutes les pages physiques sont déjà occupées.

On distingue aussi allocation initiale d'un grand bloc et allocations progressives selon les jeux de données. La seconde limite une réservation inutile mais exige de libérer les objets devenus inutiles. Des fichiers mappés et pages partagées compliquent l'attribution de chaque octet à un seul processus.

Quand la mémoire libre est épuisée

Le système récupère d'abord des pages réutilisables, notamment certains caches, et peut déplacer des pages vers le swap si celui-ci existe. **MemFree proche de zéro ne signifie donc pas nécessairement saturation**. En cas de pression persistante, les accès se ralentissent ; une allocation peut échouer et Linux peut déclencher une sélection de processus par l'OOM killer. Dans la VM sans swap, la marge de récupération est plus limitée : le programme doit traiter un échec d'allocation et le superviseur détecter un arrêt.

Les quatre routines demandées

Les valeurs de `meminfo` et `VmRSS` sont données en unités de 1024 octets, malgré leur étiquette historique « kB ». Les fonctions retournent ici des **octets**. `VmRSS` estime la mémoire résidente du processus ; ce n'est ni son espace virtuel total ni une mesure parfaite de sa contribution exclusive en présence de pages partagées.

```
from pathlib import Path
```

```

def champs_kib(chemin):
    valeurs = {}
    for ligne in Path(chemin).read_text().splitlines():
        nom, texte = ligne.split(":", 1)
        morceaux = texte.split()
        if len(morceaux) >= 2 and morceaux[1] == "kB":
            valeurs[nom] = int(morceaux[0]) * 1024
    return valeurs

def memoire_nubo(pid):
    return champs_kib("/proc/{}/status".format(pid))["VmRSS"]

def memoire_libre():
    return champs_kib("/proc/meminfo")["MemFree"]

def swap_utilise():
    m = champs_kib("/proc/meminfo")
    return m["SwapTotal"] - m["SwapFree"]

def swap_total():
    return champs_kib("/proc/meminfo")["SwapTotal"]

```

Un PID disparu ou une permission insuffisante produit une erreur à traiter ; il ne faut pas retourner arbitrairement zéro. Pour la surveillance du seuil, **MemAvailable** est plus pertinent que MemFree, car il estime ce qui peut encore être alloué sans recours au swap.

Désactiver le swap

L'absence de swap évite les latences de pagination vers le stockage et certains épisodes de thrashing ; elle peut rendre la performance de calcul plus prévisible. En contrepartie, une pointe mémoire peut provoquer plus vite un échec ou un arrêt de processus. Elle ne garantit ni vitesse ni sécurité par elle-même. Il faut dimensionner la VM, limiter la consommation et traiter les données en lots si nécessaire ; un redémarrage répété ne corrige pas une fuite mémoire.

Partie 2 — Élasticité et API

Horizontale et verticale

L'**élasticité horizontale** ajoute ou retire des instances NuBo ; elle suppose une distribution des tâches, l'absence de doubles traitements non maîtrisés et la collecte des résultats. L'**élasticité verticale** augmente ou réduit les ressources d'une instance, par exemple RAM ou CPU. Elle reste limitée par l'hôte et le logiciel et peut nécessiter une interruption. L'élasticité désigne une adaptation aux besoins, pas seulement la possibilité de démarrer beaucoup de machines.

Proposition de contrat HTTP

Les échanges utilisent **HTTPS**, des messages JSON et une authentification du nœud. Une requête POST `/jobs/claim` transmet son identité et ses mesures, puis réserve une tâche. POST convient car la réservation change l'état du serveur. La réponse contient un identifiant de tâche, un identifiant de réservation, une échéance et les données ou une URL de téléchargement autorisée. Un GET peut récupérer un jeu de données sans le réserver une seconde fois.

Le versement utilise PUT `/jobs/{id}/result`, associé à la réservation et à l'identité authentifiée : remplacer le résultat d'une même tâche selon ce contrat peut être rendu idempotent. Le serveur vérifie schéma,

taille, état et auteur. Une reprise après délai réseau réconcilie l'identifiant existant ; elle ne suppose pas que l'absence de réponse signifie absence de réservation.

Programme de collecte avec métadonnées

Le code emploie la bibliothèque requests, disponible en 2018. cle_requete est conservée pour la même demande en cas de reprise ; le serveur doit appliquer le contrat d'idempotence correspondant. Le certificat client identifie le nœud et le certificat d'autorité vérifie le serveur.

```
import socket
import requests

def mesure_memoire():
    m = champs_kib("/proc/meminfo")
    total = m["MemTotal"]
    libre = m["MemFree"]
    disponible = m["MemAvailable"]
    if total <= 0:
        raise ValueError("Mémoire totale invalide")
    return {
        "free_mib": libre / (1024 * 1024),
        "free_percent": 100 * libre / total,
        "available_mib": disponible / (1024 * 1024),
        "available_percent": 100 * disponible / total,
    }

def collecter(base_url, cle_requete, certificat, autorite):
    corps = {
        "node_id": socket.gethostname(),
        "memory": mesure_memoire(),
    }
    reponse = requests.post(
        base_url + "/jobs/claim",
        json=corps,
        headers={"Idempotency-Key": cle_requete},
        cert=certificat, verify=autorite, timeout=(5, 30),
    )
    reponse.raise_for_status()
    return reponse.json()
```

MemFree fournit la mémoire libre demandée par le sujet : sa valeur est transmise en Mio et en pourcentage de MemTotal. MemAvailable est conservé séparément pour l'orchestration ; il estime la mémoire utilisable sans pagination et ne remplace pas cette mesure exigée.

JSON exprime des champs nommés sans imposer l'interprétation d'un texte libre ; les unités figurent dans les noms. Les délais bornent une attente bloquée. Le nom de machine envoyé ne remplace pas l'authentification : le serveur le rapproche du certificat et de son registre de nœuds.

Partie 3 — Orchestration et sécurité

Seuil inférieur à 10 %

Un agent mesure périodiquement $100 \times \text{MemAvailable} / \text{MemTotal}$. Si la valeur est **strictement inférieure à 10 %**, il émet un événement horodaté avec l'identité du nœud et la mesure. Le contrôleur marque l'instance indisponible pour de nouvelles tâches, sécurise les résultats ou remet en file les tâches dont la réservation expire, puis orchestre l'arrêt et le démarrage de son remplacement. Il vérifie que le nouveau nœud est prêt avant de lui confier du travail.

L'événement porte un identifiant afin de ne pas lancer plusieurs remplacements pour la même alerte. Délais maximaux, nombre de remplacements et capacité minimale disponible évitent une boucle incontrôlée. Une confirmation de mesure ou une fenêtre courte peut filtrer le bruit si le cahier des charges le permet ; elle ne transforme pas silencieusement le seuil demandé en un autre seuil.

Agir de façon proactive

Une supervision collecte mémoire, pente de consommation, durée et taille des tâches, erreurs et capacité disponible. Des outils comme Prometheus ou Zabbix peuvent déclencher des alertes ; le contrôleur de VM exécute ensuite les actions autorisées. Une prévision simple du temps restant avant le seuil, confrontée au temps de démarrage d'une VM, permet d'anticiper un remplacement ou d'ajouter une instance.

On distingue surcharge ponctuelle, tâche anormalement grande et fuite récurrente. Adapter les lots ou corriger NuBo peut être plus efficace que multiplier les machines. Les décisions doivent rester bornées par un budget de ressources, une capacité maximale et un suivi des résultats.

Critères de sécurité

La **confidentialité** demande TLS, contrôle des accès et protection des données stockées ; l'**intégrité**, validation des messages et empreintes adaptées ; la **disponibilité**, redondance, limites de charge et reprise des tâches. L'**authenticité** relie chaque échange à une identité vérifiée ; la **traçabilité** conserve les opérations utiles avec des horloges cohérentes et une durée proportionnée.

Le serveur associe à la tâche l'identité authentifiée du nœud qui l'a réservée et exige la même identité, avec son jeton de réservation, pour le résultat. Un simple champ `node_id` fourni par le client n'apporte pas cette preuve. Jetons à durée limitée, protection contre le jeu, moindre privilège et secrets distincts réduisent l'impact d'un nœud compromis. Enfin, les résultats sont contrôlés selon les possibilités de calcul : être authentifié ne prouve pas qu'un résultat scientifique est exact.

Référence technique

[Documentation du noyau Linux : /proc, status et meminfo](#). Les chemins d'API sont une proposition pour l'application du sujet, non des adresses existantes.