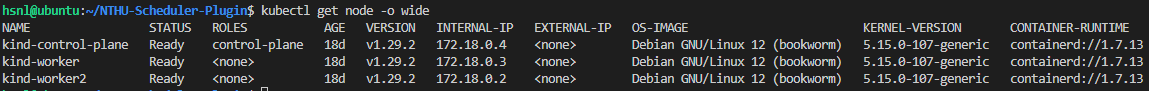
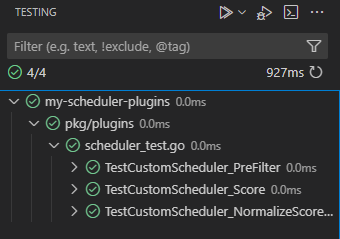
1. The screenshot of kubectl get nodes

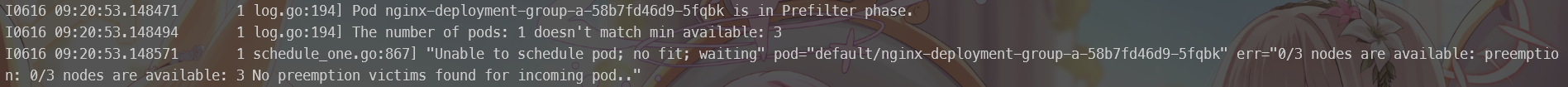


1. The screenshot of passing all the unit tests



1. Explain

* Prefilter
  + 2 個 deployment group，
    - Group1 replica 設為 3，minAvailable 設為 3
    - Group1 replica 設為 2，minAvailable 設為 3
  + Group1 剛 apply，第一個 replica pod 因為沒到 minAvailable，因此卡在 prefilter

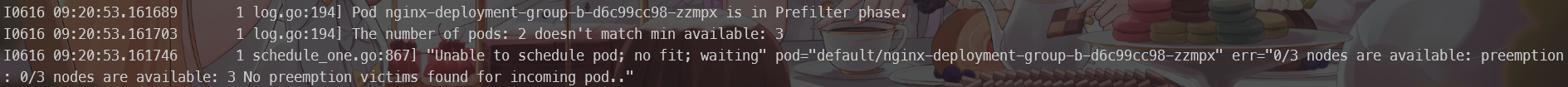


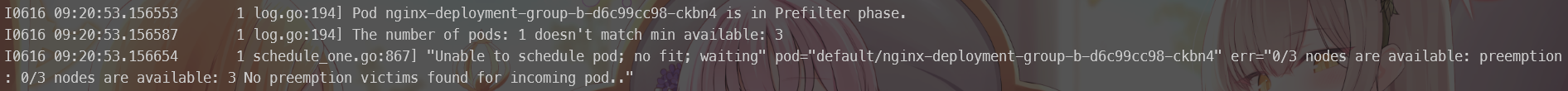
而其餘兩個應該是同時被創建，因此剛好符合 minAvailable，因此通過 prefilter



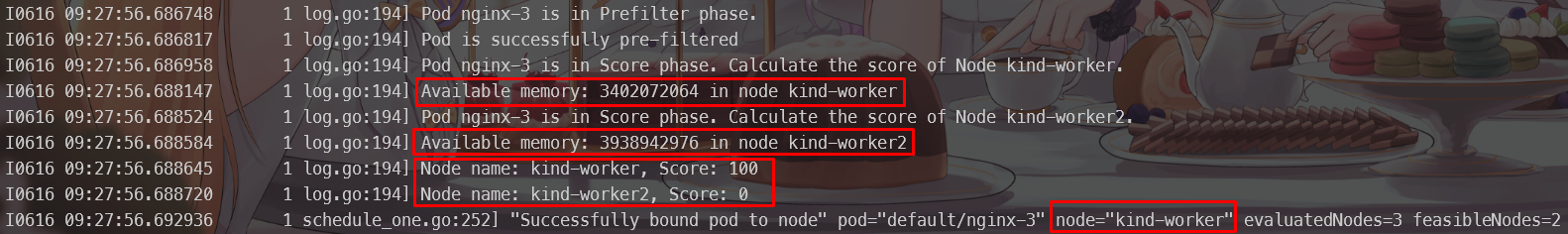


* + Group2 因為最多就 2 個 replica pod，而 minAvailable 設為 3，因此不管怎樣都不會過

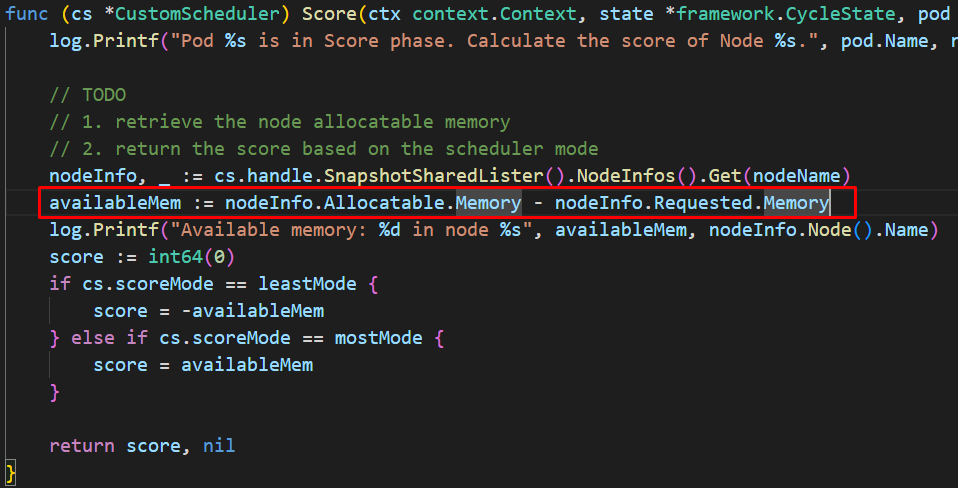




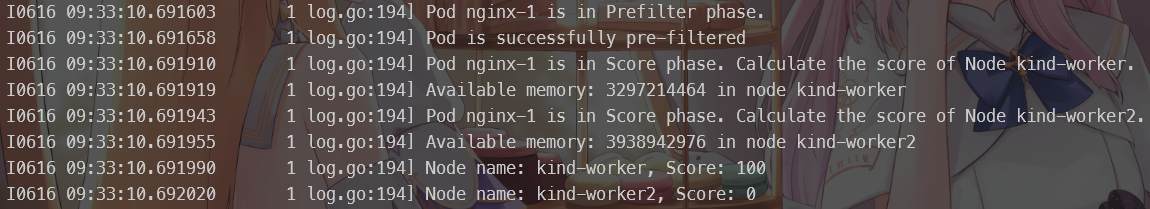
* Least mode
  + 3 個所屬Group A 的 pod，limits 和 requests 都設為 100Mi。kubectl 順序下去 create
  + minAvailable 設為 3，因此第一次 schedule 只有第三個 pod 進入 score，也正確選擇 memory 較少的 node



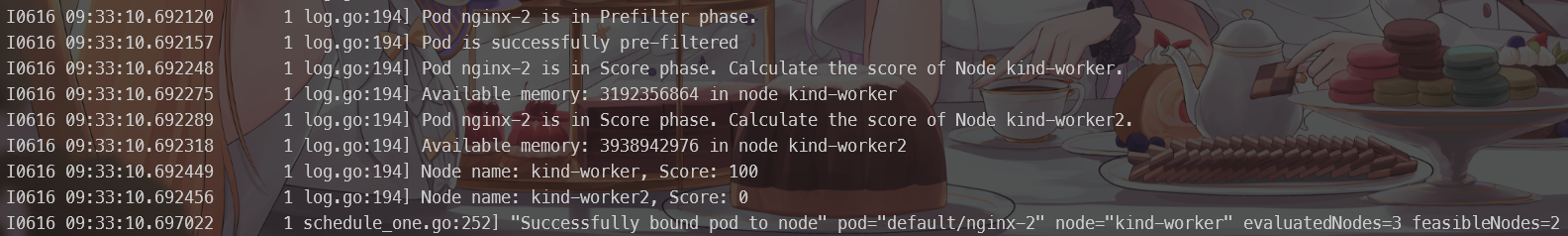
* + allocatable memory 我發現就算有 pod 在上面，數值都不會變。因此我有扣掉目前該 node 已被 requests 的 memory，以代表該 node 可能剩下的 memory



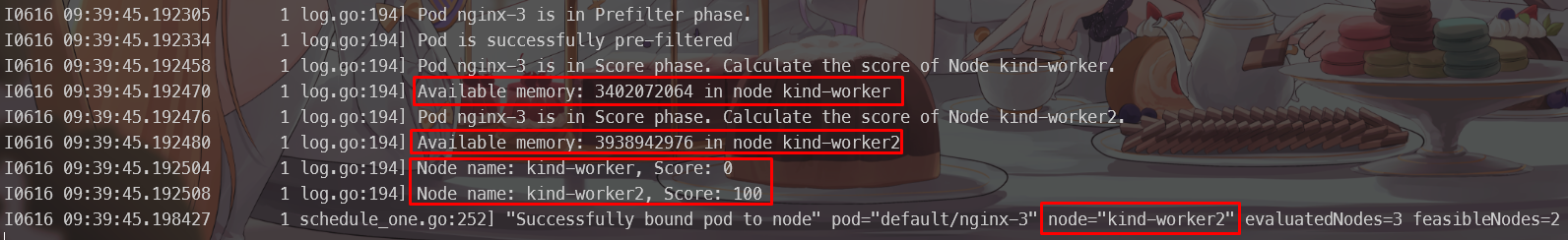
* + 由於 pod3 在 kind-worker node 上，因此 pod1 pod2 看到的數值就會更低，而其也正確地指派給 kind-worker



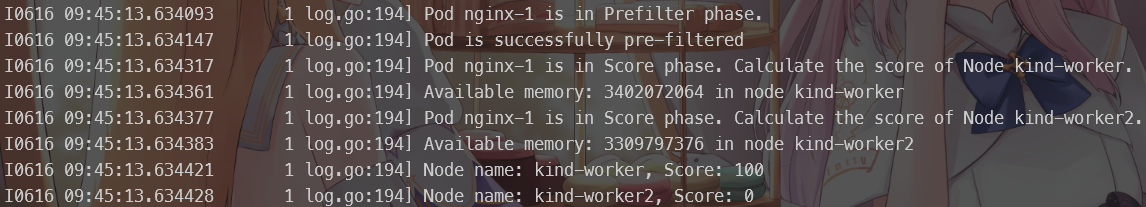




* Most mode
  + 3 個所屬Group A 的 pod，其limits 和 requests 為下所示。kubectl 順序下去 create
    - Pod1 (requests / limits): 100Mi / 100Mi
    - Pod2 (requests / limits): 100Mi / 100Mi
    - Pod3 (requests / limits): 600Mi / 600Mi
  + minAvailable 設為 3，因此第一次 schedule 只有第三個 pod 進入 score，也正確選擇 memory 較多的 node



* + 與前面提及一樣，allocatable memory 會扣掉該 node 已被 requested 的量
  + 經過 pod3 deploy，現在是 kind-worker memory 比較多，因此 pod1 選擇 kind-worker





* + 而經過 pod1 deploy，又變回 kind-worker2 memory 比較多，因此 pod2 選擇 kind-worker2

