我補充細節:

1. 我說的是 Argo CD UI 的 sync 狀態

2. 用 kubectl apply 時，是用我從 kubectl get 出來另存的 YAML

3. 那些 metadata 欄位（如 creationTimestamp、uid 等）Git 裡不存在，是從 live 物件複製過來加進去的

感謝您的補充，這幾個細節非常關鍵，它們正好點出了整個問題的核心。讓我根據您提供的新資訊，重新為您做一次更精準的說明。

核心結論先行

您遇到的問題，根源在於一個經典的 GitOps 流程錯誤：您用一個從線上環境 **(kubectl get)** 導出的、帶有「狀態」的 **YAML** 檔案，去執行 **kubectl apply** 操作。

這個檔案代表的是「過去某一瞬間的實際狀態」，而不是您在 Git 中定義的「期望狀態」。ArgoCD 的工作就是維持「期望狀態」，所以當您用一個不同的狀態去覆蓋時，它自然會偵測到不一致。

第一問：為什麼 **kubectl apply** 後，造成 **OutOfSync**？**(**搭配您的新細節**)**

這個流程可以分解成以下幾步，您會立刻明白問題出在哪：

1. **ArgoCD** 的初始狀態 **(Synced)**
   - ArgoCD 讀取您 Git Repo 中的 YAML (我們稱之為 git.yaml)。這個 git.yaml 是乾淨的，裡面沒有 uid, resourceVersion 這些系統欄位。
   - ArgoCD 將 git.yaml 的內容同步到叢集，Kubernetes 創建了對應的資源 (Live Object)。
   - 此時，Live Object 被 Kubernetes 賦予了 uid, resourceVersion, creationTimestamp 等欄位。
   - ArgoCD 進行比對，因為它的比對規則會忽略這些由系統自動產生的欄位，所以它認為 Live Object 和 git.yaml 是一致的。**UI** 顯示 **Synced**。
2. 您的手動操作 **(kubectl get & kubectl apply)**
   - 您執行 kubectl get <resource> -o yaml > live-snapshot.yaml。
   - 這個 live-snapshot.yaml 檔案現在包含了所有 Kubernetes 自動產生的欄位（uid, resourceVersion, generation: 6 等等）。它實質上是線上環境的一個「快照」。
   - 您用這個 live-snapshot.yaml 檔案執行了 kubectl apply。
3. **OutOfSync** 的產生
   - 當您 apply 這個快照檔時，Kubernetes 接收了這個指令。即使資源的規格 (spec) 沒有變化，這個操作本身也會被視為一次「更新」。Kubernetes 會因此更新一些 metadata，例如 resourceVersion 會被跳到一個新號碼，generation 可能會因為某些 Admission Controller 的介入或細微的註解變化而遞增（例如從 6 變成 7）。
   - 現在，ArgoCD 再次進行它的例行比對：
     - 期望狀態 **(Git)**：依然是那個乾淨的 git.yaml。
     - 實際狀態 **(Live Object)**：是剛剛被您用 live-snapshot.yaml "碰觸" 過的物件。它現在的 generation 是 7，resourceVersion 也是最新的。

- ArgoCD 發現，Git 中定義的資源並沒有要求 generation 是 7，但線上的物件卻是 7。這個差異是 ArgoCD 無法忽略的，因為 generation 代表了 .spec 的變更歷史。
- 結果：ArgoCD 忠實地回報：「Git 裡的期望狀態和叢集裡的實際狀態有差異！」**UI** 顯示 **OutOfSync**。

第二問：為什麼 **kubectl edit** 後，可以解決 **Sync**？

這個操作的「療效」其實是一種假象，它只是恰好觸發了一次讓 ArgoCD 滿意的「重新同步」。

1. **kubectl edit** 的本質
   - 您執行 kubectl edit，打開了處於 OutOfSync 狀態的 Live Object。
   - 您手動刪除了 creationTimestamp, resourceVersion, uid, generation 這些欄位。
2. 儲存時發生的事
   - 當您儲存時，kubectl 會把您修改後的（殘缺的）YAML 送給 Kubernetes API Server。
   - API Server 看到這個請求，它的反應是：「好的，這是一次更新請求。喔，你把系統管理的欄位 (uid, resourceVersion 等) 刪掉了，這些我說了算，所以我會忽略你的刪除，並把它們重新填上正確的值。」
   - 重要的是，您在這個過程中沒有修改 **.spec** 部分。
3. 變為 **Synced** 的原因
   - 這個 edit 操作等於是觸發了一次不改變期望規格 **(spec)** 的更新。
   - ArgoCD 再次進行比對：
     - 期望狀態 **(Git)**：git.yaml（.spec 沒有變）。
     - 實際狀態 **(Live Object)**：剛剛被您 edit 過，Kubernetes 重新生成了系統欄位，但 .spec 和 Git 中定義的完全一致。generation 的值可能不再變化，因為 .spec 沒有被改動。
   - ArgoCD 的比對引擎一看，Live Object 的 .spec、labels 等關鍵部分和 git.yaml 完全匹配。那些它在意的差異（例如先前被 apply 變更的 generation）現在消失了或恢復正常了。而那些它本來就忽略的欄位 (resourceVersion 等) 也不影響比對結果。
   - 結果：ArgoCD 認為兩者沒有差異。**UI** 顯示 **Synced**。

簡單來說，您的 **edit** 操作就像是「搖一搖」這個資源，讓它重新整理了一下自身狀態，恰好這個新狀態符合了 **ArgoCD** 的比對規則。

第三問：到底能不能移除這些 **metadata** 欄位？

基於以上的解釋，答案更清晰了：

- 在 **Git** 的 **YAML (**期望狀態**)** 中：絕對不該有。您的 Git 檔案保持乾淨是完全正確的。

- 在從線上 **get** 下來的 **YAML** 中：它們必然存在，因為這是實際狀態的紀錄。但關鍵是，您不應該再用這個檔案去執行 **apply**。
- 在 **kubectl edit (**操作實際狀態**)** 時：您可以嘗試在編輯器中移除它們，但這是一個無效且沒有意義的操作。因為在您按下儲存的瞬間，Kubernetes 會把這些欄位重新加回來。您只是在做白工。

正確的 GitOps 工作流程

如果您需要修改由 ArgoCD 管理的資源，唯一正確的途徑是：

1. 修改您 **Git Repository** 中的那個乾淨的 YAML 檔案。
2. git commit 並 git push 您的變更。
3. 讓 ArgoCD 自動偵測到變更，並為您執行同步（Apply）操作。

這樣，期望狀態和實際狀態將永遠透過 ArgoCD 的控制保持一致，您就不會再遇到 OutOfSync 的困擾。