# Deno vs. Bash for Pipeline Tasks

Nach einem Prototypen für die Verwendung von Deno als Alternative zu Shell/Bash Script in Pipeline Tasks haben wir uns für die Migration entschieden:

[ESTA-5573](https://flow.sbb.ch/browse/ESTA-5573) - ESTA Tekton: Deno Scripting Closed

Bei der Umsetzung sind allerdings einige Zweifel aufgekommen, ob dies der richtige Weg ist und ob die Verwendung von Deno nicht doch eher den Code komplexer und schlechter lesbar macht.

## Warum Deno?

Developer Experience – unsere Entwickler sind fundierter in Typescript als in Bash

## Pros

* Loops sind einfacher
* if mit && oder || sind einfacher
* Einfachere JSON Verarbeitung
* Modularisierung (Funktionen/Lib)
* Testbarkeit

## Cons

* Deno Runtime muss in allen Images für Tekton Tasks installiert werden
* Shell Exec muss umständlich gewrapped werden
* Quoting von Shell Argumenten (Values müssen z.T. gesplittet werden)  
  runCommand("helm", ["package", chartDir, ...helmArgs.split(" ")]
* Keine Unterstützung für Pipes | und File-Redirects >  
  Alternative: <https://stackoverflow.com/a/62088422>
* source Command nur sehr hackish umgesetzt/emuliert
* Kein Streaming  
  cat <large-file> vs. console.log(await Deno.readTextFile(largeFilePath))
* Shell Commands/Utilities sind meinst einfacher  
  z.B. File Upload: curl -T <filename> <url>
* Asynchronität (await vergessen)
* Zielpublikum/Open Source: Build Manager sind meist näher bei Bash als bei Typescript
* Bashscript Beispiele für Bash Commands findet man sehr viele im Netz die man 1:1 lokal testen und rüberkopieren kann. Mit Deno muss man die zuerst noch umformen oder escapen.
* 1 Komplexitätslayer für Fehlerursachen mehr
* Modularisierung mit Funktionen geht auch mit Bash (Beispiel Tagging Pipeline)
* Bash Commands kann jedes Image vom Netz, Deno nicht.
* Die Mehrheit der Commands sind Bash, Pipes usw. Ein kleiner Teil sind Kontrollstrukturen.
* Typisierung nervt manchmal. Macht das Scripting umständlicher