# Builder Image Config

Neu bauen wir grosse Builder Images mit mehreren preinstallierten Java, Node und Maven Versionen. Ursprünglich wollten wir mit sdkmanrc., nvmrc., pyenvrc. und goenvrc. arbeiten.   
Da diese aber alle etwas andere Strategien haben bei der Auswahl der installierten und verwendeten Versionen,  
wird der Update Prozess etwas schwierig und unkontrolliert. Bei gewissen rc files kann mit Wildcards gearbeitet werden. Wenn z.B. die Version 3.10.^ angegeben wird, so installiert pyenv automatisch die neuste 3.10.x Version.  
Bei der Selektion funktioniert das leider nicht. Dort muss man dann die explizite Version angeben. Bei sdkmanrc. gab es mal einen Feature Request zu einer Wildcardversionsunterstützung, implementiert wurde das allerdings nie.   
Wir wollen Wildwuchs verhindern und doch möglichst immer mit den neuesten Versionen Builden. Das heisst wir wollen dem User die Möglichkeit geben die entsprechende Major Versionen auszuwählen.

Im Builderimage gibt es ein Script, welches dann aus den Majorversionen die entsprechend korrekten preinstalled Versionen von Java, Maven, Kotlin, Python usw. selektiert. Somit ist der Code in den Tasks kurz und prägnant.

Wer andere Distributionen verwenden möchte, kann relativ einfach ein selbst maintained Builderimage mit Preinstalled Versions erstellen. z.B. über custom Build Arguments und weiteren releasejobs im estaTektonPipeline.json.

### Versionshandling im Builder Block estaTektonPipeline.json

Um das ganze Version Handling der Builders und dessen Konfiguration etwas zu streamlinen sollen die Konfigurationen für Java, JavaGraal, Kotlin, NPM, Go in einem eigenen Builder Block passieren.

**Ziel ist dabei auch eine Entkoppelung von der Selektion des SDK's** über das Helper Script wie sdkman, nvm, pyenv usw., da diese Technologien oder unsere Strategie ja wieder ändern können.

Generell werden nur die Major Versionen angegeben und die effektive Version wird dann durch ein Script im Image ermittelt und gesetzt.

Im estaTektonPipeline.json würden dann die entsprechenden Blöcke pro Technologie wie folgt aussehen.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Java** | **Java Graal** | **Kotlin** | **Python** | **Node** | **Custom Settings - 2. Ausbaustufe ermöglicht das Nachinstallieren von Versionen** |
| "builder": {  "java": "17",  "node": "16",  "maven": "3.8" }, | "builder": {  "javaGraal": "17",  "maven": "3.8" }, | "builder": {  "kotlin": "1.7" }, | "builder": {  "python": "3.10" }, | "builder": {  "node": "16" }, | "builder": {  "builderImage": "esta.docker.bin.sbb.ch/esta/myjavaBuilderImage:latest",  "builderImageArgs": ["java=17.0.3.1-librca", "maven=3.8.1"] }, |

### Default Versionhandling pro Namespace

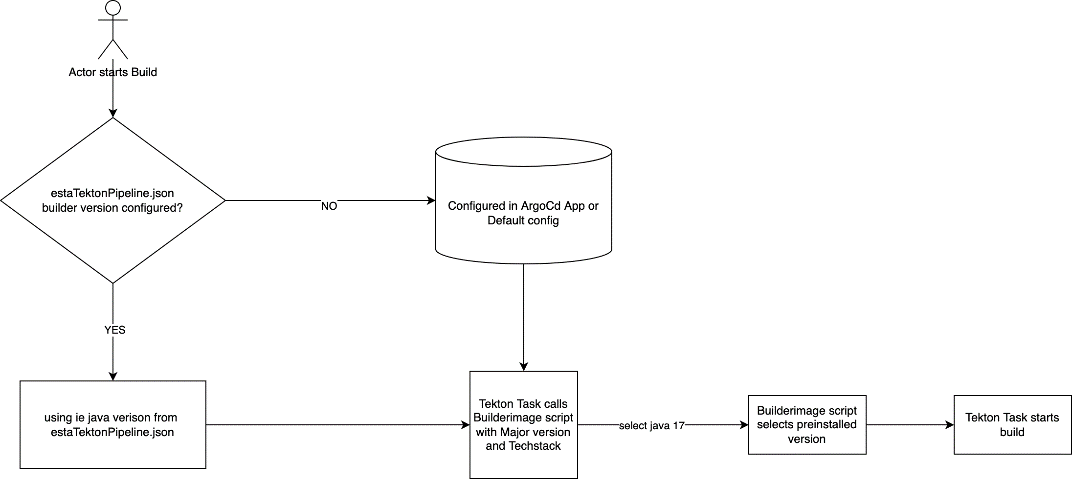
Pro Tekton Namespace sollen die default Versionen konfiguriert werden können, so dass nicht in jedem estaTektonPipeline.json die Versionsangabe gepflegt werden müssen. Diese werden dann falls nicht im estaTektonPipelien.json gesetzt als default verwendet werden.

Das wird über values.yaml Files gesteuert und kann dann im ArgoCd AppTemplate pro Tekton Namespace entsprechend konfiguriert werden.

javaDefaultVersion: '17'  
javaGraalDefaultVersion: '17'  
mavenDefaultVersion: '3.8'  
kotlinDefaultVersion: '1.7'  
nodeDefaultVersion: '16'  
pythonDefaultVersion: '3.10'  
goDefaultVersion: '1.16'

### Builderimages mit Version Selection Script

Die Builderimages für Mave, Java, Kotlin, NPM usw. wie auch für Python haben ein Script, welchem man die Major Version übergeben kann und es wird aufgrund deren automatisch die richtige vorinstallierte Version selektiert.



### Angepasste Builder Images über parameterisierbaren Build starten

Falls jemand auf Basis unserers Builderimages ein angepasstes Buildimage bauen möchte, können wir angepasste Builderimages die sich selber Updaten aber andere Versionen/Distributions verwenden die nicht Standard SBB sind. Da könnten wir 2 Wege gehen:

1. Ein parameterisierter Build steuerbar über das estaTektonPipeline.json bei dem man dann über BuildArgs die gewünschten Distributionen angibt. Die Buildargs werden dann von einem Script im Dockerfile entsprechend interpretiert und die entsprechenden SDK's installiert.
2. Zusätzliche Dockerfiles die auf dem Basis Dockerimage basieren und man dann manuell die entsprechenden Versionen setzt und installiert.

Beide Varianten haben ihre Vor- und Nachteile. Denke aber dass das eine gangbare Ausbaustufe ist.