# Parameterisierte Builds

Auf Jenkins existieren heute einige Pipelines mit Parameterisierbaren Builds. Solche Builds sollen auch in Esta-Tekton ermöglicht werden.

Wenn ein Build über das Tekton Ui gestartet wird, soll es die Möglichkeit geben diverse im estaTektonPipeline.json vordefinierte Buildparameter über das UI abzufüllen.

Grundsätzlich wäre eine Umsetzung nicht allzu kompliziert, der etwas aufwändigere Teil ist aber, im Ui die richtigen Felder zur nachfolgend richtigen Pipeline darzustellen.  
Gerade wenn eine Tagging Pipeline gestartet wird, soll anhand der Version (Version oder Preversion usw.) die richtige nachfolgende Pipeline evaluiert und auch etwaige Parameter richtig im UI dargestellt werden.

## Anforderungen

Die groben Anforderungen sollen hier gelistet werden.

* Liste der möglichen Parameter werden in dem estaTektonPipeline.json deklariert. :tick:
* Default Values können pro Pipeline überschrieben werden :question:
* Im UI werden dann entsprechend die richtigen Parameter der Buildpipeline dargestellt.
* Im Tagging UI muss die Buildpipeline anhand der Version predicted und entsprechend die default values abgefüllt werden. :question:
* Jedem Parameter kann einer der Typen zugewiesen werden: String, Multiline, Choice, MultiChoice, Boolean
  + Die UI Darstellung wäre wie folgt gemapped
    - String → Input Feld
    - Multiline → Multiline Feld
    - Choice → Radio Buttons (hier könnte auch Boolean darüber abgebildet werden)
    - Multi Choice → Checkboxen
    - Evtl. boolean Typ abbilden → Checkbox
* In einem ersten Wurf würde im Build-Assistent-UI einfach ein grosses Textfeld dargestellt, welches eine Liste aus Key=Values editierbar macht. So können die Builds dann parametrisierbar gemacht werden
* Fastbuildbutton würde bei parametrisierbaren builds die default Werte übergeben :question:
* Die Key=Values sollen auch per externem File abgefüllt werden können. Entweder automatisch per PipelineConfig oder im UI per Upload Button oder Load from URL Button. :question:
* Die Parameter werden als JSON in jedem TaskRun zur Verfügung stehen. :tick:
* Die Parameter sollen von Pipeline zu Pipeline weitergereicht werden. Dies soll durch die ganze Kette funktionieren: Tagging → Build → Staging. :tick:
* Die Parameter werden als JSON vom Controller aus den Parametern ausgelesen und an die nächste Pipeline übergeben. :tick:
* Die Parameter sollen in den PipelineDetails im UI dargestellt werden.  :tick:
* Rerun mit Parameter soll zu einem späteren Zeitpunkt ebenfalls funktionieren. Es wird dann ein UI mit den Parametern dargestellt. :question:

Die Anforderungen sind noch nicht abschliessend.

## EstaTektonPipeline.json

### Parameter Definitiion

Die Konfiguration der Parameter wird wie folgt aussehen:

"parameters": [

{

"name": "environment",

"type": "Multichoice",

"values": ["dev","test"],

"label": "Environment to stage",

"description": "Environment to stage to",

"options": ["dev", "test", "prod"]

},

{

"name": "notificationEmails",

"type": "Multichoice",

"values": ["manuel.wallrapp@sbb.ch","thomas.bruederli@sbb.ch"],

"options": ["manuel.wallrapp@sbb.ch","thomas.bruederli@sbb.ch","lukas.spirig@sbb.ch"]

"description": "The emailadresses which the notifications are sent to"

}

],

### Parameter Werte anhand eines Fileparse setzen

Hier gibt es ein Beispiel wie Parameterwerte anhand einer File Url (z.B. über Artifactory) gesetzt werden:

<https://code.sbb.ch/projects/KI_KIHUB/repos/kihub-pipelines/browse/src/main/pipelines/mmi-cus-dev/Jenkinsfile-PrepareP2Update#13-38>

Dieses Feature sollte bedacht werden, würde aber zu einem späteren Zeitpunkt nachgeliefert.

Ein Textfile welches z.B. in Artifactory abgelegt wäre hätte dann folgenden content:

successmail: [manuel.wallrapp@sbb.ch](mailto:manuel.wallrapp@sbb.ch)  
environment: dev

Dann kann pro Parameter ein Link auf das Textfile angegeben werden. Das heisst der Inhalt des Textfiles wird geholt und dann geschaut ob ein Parameter Wert vorhanden ist. Dieser wird dann geholt und entsprechend für den Build gesetzt.

### Parameter als Placeholder :question:

Es sollen auch ähnlich wie die predefined Variablen Parameter als Placeholder verwendet werden können. Dazu gibt man einfach den Parameter wie folg an:

"tasks": [{

"name": "post-build",

   "taskRef": "custom-post-build-job",

   "params": {

   "STAGE": "${parameters:environment}",

   "MAILTO": "${parameters:successemail}",

"runAfter": "maven-deploy"

}]

Diese Variablen werden dann beim prozessieren des estaTektonPipeline.json eingesetzt.

### estaTektonPipeline.json Beispiel

In estaTektonPipeline.json würde man global die Parameter definieren und sie dann pro Pipelinedefinition referenzieren. Das sieht dann in einem einfachen Json wie folg aus:

{

"$schema": "../../main/resources/estaTektonPipelineSchema.json",

"productName": "esta-tekton-pipeline-test-docker",

"helm": {

"chartName": "esta-tekton-pipeline-test",

"chartRepository": "esta.helm.local",

"chartRootFolder": "./mycharts"

},

"mvn": {

"parentPom": "pom.xml",

"deployableArtifactsPom": "pom.xml",

"artifactoryMavenRepo": "esta.mvn"

},

"parameters": [

{

"name": "environment",

"type": "Multichoice",

"values": ["dev","test"],

"label": "Environment to stage",

"description": "Environment to stage to",

"options": ["dev", "test", "prod"]

},

{

"name": "notificationEmails",

"type": "Multichoice",

"values": ["manuel.wallrapp@sbb.ch","thomas.bruederli@sbb.ch"],

"options": ["manuel.wallrapp@sbb.ch","thomas.bruederli@sbb.ch","lukas.spirig@sbb.ch"]

"description": "The emailadresses which the notifications are sent to"

}

],   "pipelines": [

{

"name": "snapshot",

"triggerType": [

"USER"

],

"build": {

"packageAndDeployHelmChart": true

},

"tasks": [

        {

         "name": "post-build",

         "taskRef": "custom-post-build-job",

          "params": {

           "STAGE": "${parameters:environment}",

           "MAILTO": "${parameters:successemail}"

         },

          "runAfter": "maven-deploy"

        }

     ]

}

]

}

## Parametertypen

Es werden folgende Parametertypen im estaTektonPipeline.json definierbar sein und dann entsprechend im Ui dargestellt:

* String - ein normales Inputfeld
* Multiline - ein multiline Inputfeld
* Choice - eine Radiobox mit Singleauswahl
* Multichoice - checkboxen mit Mehrfachauswahl
* Boolean - ist wie Multichoice aber nur mit einer Checkbox

Jeder Parameter stellt noch ein ? mit Hovereffekt dar, welches eine Kurzbeschreibung des Parameters aufpoppt.

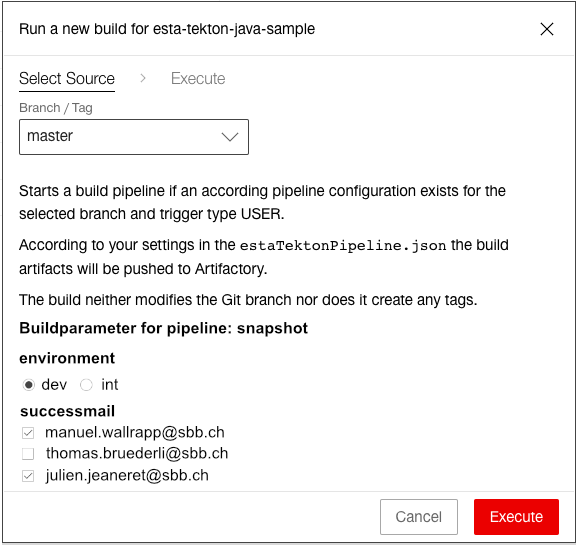
All diese Parameter können auch über das REST Api von Esta-Tekton angesprochen werden.

## Globale Default Values / Pipeline Default Values

Bei den Parametern sollen globale default Values gesetzt werden können. Die gelten dann überall wo dieser Parameter verwendet wird. Dieser default value kann dann aber pro Pipeline nochmals überschrieben werden. So können z.B. default Values für Cronjobs anders aussehen als Default Values für UI started jobs. Das überschreiben pro Pipeline ist noch nicht implementiert.

## Esta Tekton Ui Darstellung

Im Build Ui würden dann die Buildparameter beim Auswahl des branches dargestellt. Das heisst wenn der Branch oder Tag ausgewählt wird,   
ermittelt das Ui über einen Call auf den Controller die richtige Pipeline(s) und kriegt somit gleich die Buildparameter mit. Werden mehrere Builds gestartet, werden die Parameter nach Pipeline gruppiert in Tabs dargestellt.



Die Parameter werden dann bei der Execution des Builds mitgegeben.

CHALLENGE

Beim Starten einer Tagging Pipeline via UI wird das Filtering der anzuzeigenden Parameter aufgrund der selektierten/eingegebenen Version gemacht. Falls mehrere Pipelines mit Parameter Spezifikation matchen, würden die Parameter in sepraten Panes betitelt nach der Zielpipeline dargestellt. Dafür müsste man das PipelineMatching nach GitEvent ausführen und schauen welche Pipelines gestartet würden. Es müssen die Zielparameter aller Pipelines dargestellt werden, da

## Template Parameter

In den Templates wird es dann einen Parameter geben, der eine JSon Struktur mit den Parametern und dessen Werten beinhaltet.

Den Parameter würden wir mit USER\_INPUT\_PARAMETER benennen. Eine Convenience Funktion in den Tasks würde dann diese Parameter als Objekt oder Variablen im Typescript oder Bashscript zur Verfügung stellen.  
Auch soll es convenience Funktionen zur Verarbeitung der Parameter geben. Der Inhalt dieses Parameters würde wie folgt aussehen:

**Parameter Content**

{

"environment": "dev"

"successmail": ["manuel.wallrapp@sbb.ch", "julien.jeaneret@sbb.ch"]

}

In Typescript ist die Verarbeitung relativ einfach. In Bash Script würde die Verarbeitung mit JQ geschehen.

## Build Execution über REST

Die Buildexecution kann auch über REST geschehen wo auch die Parameter entsprechend gesetzt werden.

## Build Execution Ablauf und Parameter Weitergabe

Bei der Buildexecution werden die Parameter vom UI oder REST service an den Controller weitergereicht. Auch wird mit diesen Parametern ein Parameter Substitution ausgeführt. Das Gleiche wird auch für diese Variablen gemacht: [ESTA Tekton Parameters - estaTektonPipeline.json#Pre-definedPipelineParameters](file:///C:\display\CLEW\ESTA+Tekton+Parameters+-+estaTektonPipeline.json#ESTATektonParametersestaTektonPipeline.json-Pre-definedPipelineParameters)

Die Parameter sollen folglich in der ganzen Pipeline Kette verwendbar sein. Das heisst der Parameter USER\_INPUT\_PARAMETER wird am Ende der Pipeline ausgelesen und in das PipelineEvent Objekt geschrieben um dann mit diesem Event Objekt die neue Pipeline (Tagging → Build → Staging) zu starten. In die neue Pipeline wird dieser Parameter USER\_INPUT\_PARAMETER dann erneut injected.

### Auffinden von Parametern bei Bitbucket Events

Wenn die Taggingpipeline mit Parametern gestartet wird, so werden auch automatisch die Annotations auf der Pipeline gesetzt, welche die zu taggenden Versionen beschreiben.

Somit würde es bei einer Mavenpipeline 4 zusätzliche Annotations geben:

* gitProject (existiert bereits)
* gitRepo (existiert bereits)
* releaseVersion (effektiv released version)
* nextVersion (z.B. next snapshot version)

Anhand dieser Informationen kann dann die nachfolgende Pipeline den Pipelinerun auffinden und den Parametern USER\_INPUT\_PARAMETER auslesen.

### Injecten der Parameter in Pipeline und Tasks

Die User Defined Parameter werden folglich in alle Tasks, auch Custom Tasks über TemplateModifiers und TaskModifiers injected werden. Somit können diese Parameter in allen Tasks verwendet werden.

### Ablauf des Releaseprozesses und Parameterweitergabe

