# Advanced Topics

* [Custom Pipelines](#AdvancedTopics-CustomPipelines)
  + [Custom Pipeline Templates in Project Repository](#AdvancedTopics-CustomPipelineTemplatesi)
  + [Import Custom Pipelines from your own Repository](#AdvancedTopics-ImportCustomPipelinesfro)
  + [Integration in estaTektonPipeline.json](#AdvancedTopics-IntegrationinestaTektonP)
  + [Pipeline Parameters](#AdvancedTopics-PipelineParameters)
  + [Example](#AdvancedTopics-Example)
* [Custom Pipeline Tasks](#AdvancedTopics-CustomPipelineTasks)
* [Start-Pipeline Task (ready in future version 0.29.0)](#AdvancedTopics-Start-PipelineTask(ready)
* [Environment Variables in Pipeline Tasks](#AdvancedTopics-EnvironmentVariablesinPi)
  + [Expose env variables to Docker build context](#AdvancedTopics-ExposeenvvariablestoDock)
* [Java and Maven Versions](#AdvancedTopics-JavaandMavenVersions)
* [Using Buildpacks](#AdvancedTopics-UsingBuildpacks)
* [SonarCloud Test Coverage with JaCoCo](#AdvancedTopics-SonarCloudTestCoveragewi)
* [Using ssh for git operations in your custom task](#AdvancedTopics-Usingsshforgitoperations)
  + [Generate a new ssh key](#AdvancedTopics-Generateanewsshkey)
  + [Add Public Key to Bitbucket User esta-build-xyz](#AdvancedTopics-AddPublicKeytoBitbucketU)
  + [Add Git SSH Secret to the desired Tekton Namespace](#AdvancedTopics-AddGitSSHSecrettothedesi)
* [Using Testcontainers](#AdvancedTopics-UsingTestcontainers)
  + [Known issues and troubleshooting](#AdvancedTopics-Knownissuesandtroublesho)

## Custom Pipelines

Although the ESTA Tekton pipelines work out-of-the box with build stacks like Spring Boot, Angular and Go it's obvious that we do not cover all use-cases and workflows. One of the big advantages of using Tekton is the ability to create and run individual pipelines and thereby use existing tasks from ESTA Tekton or from the public [Tekton Catalog](https://hub.tekton.dev/). Create your completely custom build pipelines while still taking advantage of the ESTA Tekton Controller and its neat integration with Bitbucket.

### Custom Pipeline Templates in Project Repository

Templates for custom pipelines or PipelineRuns respectively are stored directly within the repository along with the estaTektonPipeline.json file which references them.

As a convention, the PipelineRun templates shall be saved in the "tekton" folder and should be "self-contained". That means they'll be started by the ESTA Tekton Controller via an oc process -f <file> command with parameters extracted from the repository data and Git event. They preferably contain an inline [pipeline definition](https://tekton.dev/docs/pipelines/pipelineruns/#specifying-the-target-pipeline) with pipelineSpec, as shown in [this example](https://code.sbb.ch/projects/KD_ESTA_TEST/repos/esta-tekton-custom-templates-in-repo/browse/tekton/esta-tekton-custom-templates-in-repo-run.yaml). This guarantees the execution of a custom pipeline with the right version from the according Git branch, tag or revision. A good point to start is to copy [this template](https://code.sbb.ch/projects/KD_ESTA/repos/esta-tekton-pipeline-templates/browse/pipelines/custom-tasks-pipeline-run.yaml) which is a bare minimum pipeline template.

### Import Custom Pipelines from your own Repository

Custom pipelines and tasks can also be managed in a separate Git repository and made available in your own build namespace. Pipelines are then referenced by name instead of their filename. This general import of tasks and pipelines from additional repositories also allows to overwrite default tasks and pipelines defined by the ESTA Tekton system.

A list of Git repositories to import pipelines from can be added to the [Tekton Build Namespace Config](file:///C:\display\CLEW\Dedicated+Build+Namespaces#DedicatedBuildNamespaces-SetupyourownBuildNamespace) via a CLEW-Ticket with an additional parameter in the ArgoCD application definition:

**esta-tekton-pipeline-config values.yaml addition**

esta-tekton-pipeline-controller:

additionalTemplateRepos:

- https://code.sbb.ch/scm/kd\_esta\_test/esta-tekton-pipeline-custom-templates.git?at=refs/heads/master

Use fully qualified HTTP urls to the Git repository and denote the desired branch/tag/revision with the "at" query parameter.

From each repository .yaml files from the folders tasks and pipelines are periodically imported by the ESTA Tekton Controller. See [this repository](https://code.sbb.ch/projects/KD_ESTA_TEST/repos/esta-tekton-pipeline-custom-templates/browse) for some examples.

### Integration in estaTektonPipeline.json

The use of custom pipelines instead of default ESTA Tekton pipelines is controlled with the [pipeline configuration](file:///C:\display\CLEW\ESTA+Tekton+Parameters+-+estaTektonPipeline.json#ESTATektonParametersestaTektonPipeline.json-Pipelineconfigurationstructure) in estaTektonPipeline.json. Use the custom property to provide a [custom pipeline specification](file:///C:\display\CLEW\ESTA+Tekton+Parameters+-+estaTektonPipeline.json#ESTATektonParametersestaTektonPipeline.json-Custompipelineconfigurationstructure).

A custom pipeline is referenced by either one of the following properties:

"pipelineTemplate": "<name or file of pipelinerun template>"  
"pipelineRef": "<name of pipeline>"

pipelineTemplate refers to an [OpenShift Template](https://docs.openshift.com/container-platform/4.7/openshift_images/using-templates.html), which contains a PipelineRun object and any additional resources like PVCs, ConfigMaps, etc. if the property value ends with .yaml the according file within the "tekton" folder of the repository is used. Otherwise the value is considered a template name already present in the build namespace.

If instead the pipelineRef property is used in the pipeline configuration it is interpreted as a reference to a Pipeline resource stored in the Openshift build namespace. It'll be started using a [default template](https://code.sbb.ch/projects/KD_ESTA/repos/esta-tekton-pipeline-templates/browse/pipelines/esta-tekton-pipeline-custom-run.yaml).

### Pipeline Parameters

The [custom pipeline specification](file:///C:\display\CLEW\ESTA+Tekton+Parameters+-+estaTektonPipeline.json#ESTATektonParametersestaTektonPipeline.json-Custompipelineconfigurationstructure) can define additional parameters which are passed to the pipelinerun template:

"params": {  
  "VERSION\_TAG": "snapshot.${DATE}",  
  "IMAGE\_LABELS": "buildtype=snapshot"  
}

The parameter values can contain interpolated strings which will be replaced with globally defined parameters (${PARAM\_NAME}) or with values from estaTektonPipeline.json (${/json/pointer}).

### Example

Here's an example of a pipeline configuration in estaTektonPipeline.json starting a custom pipeline:

**Custom Pipeline definition** Quelle erweitern

{

"name": "custom-snapshot",

"enabled": true,

"triggerType": [

"GITEVENT",

],

"branchNamePrefixes": [

"master"

]

"custom": {

"pipelineTemplate": "custom-build-pipeline-run-template.yaml",

"params": {

"IMAGE\_TAG": "snapshot.${DATE}",

"APP\_NAME": "${/stages/1/argoCD/argoCdAppName}"

}

}

}

Experimental

Please note that the custom pipelines feature is still experimental and subject to changes.

## Custom Pipeline Tasks

In order to comply with the many requirements for customizing the standardized pipelines that come with ESTA Tekton, additional tasks can be injected into the build pipelines via pipeline configuration of estaTektonPipeline.json. With this powerful feature one can extend the Tekton pipelines using building blocks from the [ESTA Tekton task catalog](https://code.sbb.ch/projects/KD_ESTA/repos/esta-tekton-pipeline-templates/browse/tasks) or even with custom Tekton tasks from [your own repository](#AdvancedTopics-ImportCustomPipelinesfro).

**Pipeline configuration with custom tasks**

{

...

"pipelines": [

{

"name": "release",

...

"tasks": [

{

"name": "s3-push",

"taskRef": "esta-s3-push",

"params": {

"BUCKET\_URL": "s3://test-bucket/static-www",

"SOURCE\_DIR": "./dist",

"AWS\_REGION": "us-east-1",

"AWS\_CREDENTIALS\_SECRET": "pipeline-env-aws-credentails"

},

"runAfter": "quality-gate-check",

"runBefore": "maven-deploy"

}

]

}

]

}

In this this example, the additional task "esta-s3-push" from the ESTA Tekton task catalog is inserted into the pipeline to run after the quality-gate-check and before the maven-deploy task. The parameters expected by the task are provided right in the pipeline config and can also inherit values from other parameters using [interpolated strings](#AdvancedTopics-PipelineParameters). See the [parameter documentation](file:///C:\display\CLEW\ESTA+Tekton+Parameters+-+estaTektonPipeline.json#ESTATektonParametersestaTektonPipeline.json-Customtaskconfigurationstructure) for detailed explanations of all properties.



**Conventions for custom Tekton tasks**

When creating your own custom Tekton tasks, the following conventions must be adhered to:

* Defines exactly one workspace named "source"
* Only accepts parameters of type "string"
* Docker images used must be referenced via docker.bin.sbb.ch

**Executing custom taks in the finally section**

Tekton pipelines have a finally section where tasks are guaranteed to be executed in parallel after all PipelineTasks under tasks have completed regardless of success or error. In order to place a task in the finally section use the executionMode property with the value "FINALLY". By default, custom tasks in finally are only executed if the pipeline terminated successfully. Use the runCondition property as described in the the [parameter documentation](file:///C:\display\CLEW\ESTA+Tekton+Parameters+-+estaTektonPipeline.json#ESTATektonParametersestaTektonPipeline.json-Customtaskconfigurationstructure) if your tasks should run always or on failure. The runBefore and runAfter properties are not allowed for finally tasks.

The use of custom tasks requires some basic knowledge of [Tekton](https://tekton.dev/docs/pipelines/) and the structure of the ESTA Tekton pipelines, namely the names of the existing tasks in order to inject the custom tasks before/after them and the pipeline workspace. A good point to start is to copy the generic [esta-custom-exec](https://code.sbb.ch/projects/KD_ESTA/repos/esta-tekton-pipeline-templates/browse/tasks/esta-custom-exec.yaml) task.

## Start-Pipeline Task (ready in future version 0.29.0)

If you want to start another pipeline in your repository or another repository managed by the same Esta-Tekton instance, you can inject a new task into your pipeline configuration in your estaTektonPipeline.json.

Be aware, that the to be triggered pipeline config must have the triggerType: USER to be able to be started.

"pipelines": [

{

"name": "release",

"build": {

"packageAndDeployHelmChart": true

},

"triggerType": ["GITEVENT"],

"versionTagEventPatterns": ["^(\\d+\\.)?(\\d+\\.)?(\\\*|\\d+)$"],

"tasks": [

{

"name": "esta-start-pipeline",

"taskRef": "esta-start-pipeline",

"params": {

"GIT\_PROJECT": "KD\_ESTA\_TEKTON\_IT",

"GIT\_REPOSITORY": "esta-tekton-go-image-test",

"GIT\_REVISION": "refs/heads/master",

"PIPELINE\_CONFIG\_NAME": "continuous"

},

"runAfter": "git-clone",

"runBefore": "helm-lint"

}

]

}

]

This will trigger a new Pipeline over the REST API of esta-tekton-pipeline-controller. Behind the scene is a JWT Token Secret generated which is valid as long the triggering pipeline is running. This Token will be passed to the REST call to start the pipeline. You need to set following Parameters:

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| GIT\_PROJECT | The Git Project for example KD\_ESTA |
| GIT\_REPOSITORY | The name of the repository where you want to trigger an esta-tekton build. |
| GIT\_REVISION | The revision, branch or tag you want to trigger the build on i.e:  refs/heads/master  refs/tags/1.2.3 |
| PIPELINE\_CONFIG\_NAME | (Optional) the name of the config you want to trigger. Otherwise all matching pipelines will be startet. |

## Environment Variables in Pipeline Tasks

Custom environment variables can be exposed to pipeline tasks using OpenShift Secrets stored in your Tekton build namespace. By default the secret with the name esta-tekton-pipeline-env is loaded into pipeline tasks but a different secret name can be specified using the envSecret property in the [pipeline config block](file:///C:\display\CLEW\ESTA+Tekton+Parameters+-+estaTektonPipeline.json#ESTATektonParametersestaTektonPipeline.json-Pipelineconfigurationstructure). Each entry of the secret is now available as env var with the key name.

**Pipeline configuration structure**

{

"pipelines": [

{

...

"envSecret": "pipeline-env-continuous"

}

]

}

Pipeline secrets can easily be managed via the [Tekton Control Panel](file:///C:\display\CLEW\Tekton+Control+Panel#TektonControlPanel-PipelineSecrets) even without direct access to OpenShift.

### Expose env variables to Docker build context

While env variables from the given secrets are exposed to the build tasks in Tekton, they're not available in the Docker build context. In order to use them in a Dockerfile, they need to be passed as build arg to the Docker command. This can be achieved by mapping them with the [ocBuildVars](file:///C:\display\CLEW\ESTA+Tekton+Parameters+-+estaTektonPipeline.json#ESTATektonParametersestaTektonPipeline.json-Dockerconfigurationstructure) property in the estaTektonPipeline.json:

**Docker configuration structure**

{

"docker": {

...

"ocBuildVars": [

"MY\_API\_KEY": "$MY\_API\_KEY"

]

}

...

}

In the Dockerfile, you can declare the build argument with ARG MY\_API\_KEY and then use it with ${MY\_API\_KEY}.  
The values from the pipeline secrets with the key MY\_API\_KEY is now available in the Docker build context.

## Java and Maven Versions

The Java (JDK) version used for Maven builds can be chosen individually for each project from within the estaTektonPipeline.json file. The different versions are made available through custom builder images we use in the Tekton pipeline and which are maintained by the ESTA team. You can select the desired Java, Kotlin and Maven versions in the [builder config block](file:///C:\display\CLEW\ESTA+Tekton+Parameters+-+estaTektonPipeline.json#ESTATektonParametersestaTektonPipeline.json-Builderconfigurationstructure). Here's an example to select Java 11 for your builds:

**Maven configuration structure**

{

"builder": {

"java": "11",

"maven": "3.5"

}

}

The selection of versions available in the ESTA builder images is limited (see [parameter documentation](file:///C:\display\CLEW\ESTA+Tekton+Parameters+-+estaTektonPipeline.json#ESTATektonParametersestaTektonPipeline.json-Builderconfigurationstructure)). In order to use other versions or JDKs you can create your own builder image and specify it with the "customBuilderImage" property.

## Using Buildpacks

Container images can also be built using [Buildpacks](https://buildpacks.io/) instead of a classic Dockerfile. In order to enable the use of Buildpacks in Tekton, simply add a "buildpack" block (top-level) to your estaTektonPipeline.json. The "docker" block defining the "artifactoryDockerRepo" property is also required to specify the target for the build container image.

**Buildpack example**

{

...

"buildpack": {

"envVars": [

"BP\_NEW\_RELIC\_ENABLED=true"

]

},

"docker": {

"artifactoryDockerRepo": "esta.docker"

},

"mvn": {

...

}

...

}

By default ESTA Tekton uses the [Paketo](https://paketo.io/docs/concepts/builders/) "base" builder image but also [Heroku](https://devcenter.heroku.com/articles/buildpacks) builder images are supported. Since Buildpacks analyze the source code and automatically add their magic to the final image, there's little to configure. Some buildpacks can be configured or enabled via environment variables. Check the documentation of the specific buildpacks you want to be used for your project.

See the [parameter documentation](file:///C:\display\CLEW\ESTA+Tekton+Parameters+-+estaTektonPipeline.json#ESTATektonParametersestaTektonPipeline.json-Buildpackconfigurationstructure) for a complete reference.

Experimental

Please note that the buildpacks support is only experimental and subject to be removed again from the standard ESTA Tekton pipelines.

## SonarCloud Test Coverage with JaCoCo

The test coverage report will be uploaded by the pipeline and is available on SonarQube. If the [sonarscan step](file:///C:\display\CLEW\ESTA+Tekton+Parameters+-+estaTektonPipeline.json#ESTATektonParametersestaTektonPipeline.json-Sonarconfigurationstructure) is not enabled, the coverage report will not be uploaded to SonarQube.

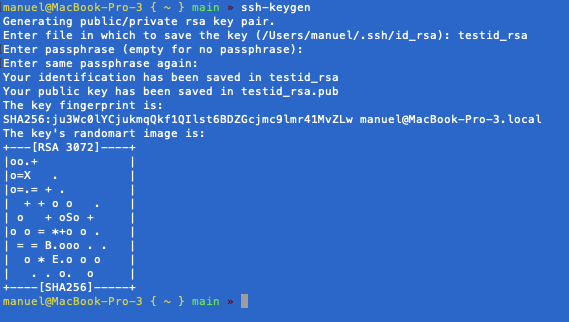
## Using ssh for git operations in your custom task

If you have a task or build job which relies on ssh for git operations, there is a manual workaround to support clones and git operations through ssh. To achieve that you need to do following things:

1. Generate a private/public keypair with ssh-keygen or your preferred tool
2. add the public key in your Bitbucket settings
3. create a new secret with known\_hosts and the private-key value
4. link the secret with the tekton serviceaccount

### Generate a new ssh key

You can generate a new key as showed in the screenshot with the tool ssh-keygen - **important,** don't set a passphrase otherwise Tekton won't be able to read the keyfile:



with the commands:

cat id\_rsa

cat id\_rsa.pub

you can display the content of the private and public key file.

### Add Public Key to Bitbucket User esta-build-xyz

Login on bitbucket with your esta-build-xyz user in a private browser tab: <https://code.sbb.ch/login?oauth_sso=false>

Copy the content of id\_rsa.pub into a new entry:

<https://code.sbb.ch/plugins/servlet/ssh/account/keys>

### Add Git SSH Secret to the desired Tekton Namespace

ESTA Tekton is already prepared for using SSH keys but still requires a manual step performed by a system admin. Please create a CLEW  support ticket in order to let us configure an SSH key in your Tekton namespace.

## Using Testcontainers

With the release 0.18.0 ESTA Tekton now supports to use [Testcontainers](https://testcontainers.com/) in build pipelines.

The build pipelines for Java/Maven, NPM and Go will start a container environment if the testcontainers feature is enabled in the pipeline config as shown in this example:

**estaTektonPipeline.yaml**

...

pipelines:

- name: continuous

...

build:

testcontainers:

enabled: true

Here's a [full example](https://code.sbb.ch/projects/KD_ESTA_TEKTON_IT/repos/esta-tekton-spring-boot-testcontainers/browse) project showing a simple Springboot application connecting to a Postgres database. Look into the [CustomerControllerTest](https://code.sbb.ch/projects/KD_ESTA_TEKTON_IT/repos/esta-tekton-spring-boot-testcontainers/browse/src/test/java/com/testcontainers/demo/CustomerControllerTest.java) class for the actual testcontainer integration.

The container environment to actually run the Testcontainers is provided by [Kubedock](https://github.com/joyrex2001/kubedock) which will essentially run the requested containers as pods in the Tekton namespace.

With the optional property kubedockParams in the testcontainers block in estaTektonPipeline.json you can define additional arguments passed to the kubedock server command in order to tweak the containers environment. See the [Kubedock documentation](https://github.com/joyrex2001/kubedock#containers) for details.

Experimental Feature

The support for Testcontainers is new and still experimental and comes with limitations.

Testcontainers depends on Docker, and raw Docker unfortunately is an issue in an Kubernetes managed environment like we have with OpenShift. There's [no official support](https://github.com/testcontainers/testcontainers-java/issues/1135) for it and the integration in Tekton is done with best effort but is not 100% compatible with a local Docker environment. Some containers may fail to run in the restricted OpenShift environment.

### Known issues and troubleshooting

**No parallel execution of builds with testcontainers**

Yes, the kubedock server is started with the [--lock](https://github.com/joyrex2001/kubedock#namespace-locking) option which will only allow to run one server in the Tekton namespace. This is to avoid collisions of multiple running testcontainers with network bindings.

**Failed to start container due to permission errors**

Not all testcontainer images run smoothly with the random UIDs assigned by OpenShift. If permission errors occur, analyze the image and set the user to run the container with using the label com.joyrex2001.kubedock.runas-user.

In Java this can be done like this:

postgres = new PostgreSQLContainer<>("postgres:15-alpine")

.withLabel("com.joyrex2001.kubedock.runas-user", "70");

**Debugging failing testcontainers**

Finding the cause when testcontainers don't come up is a bit tricky with Kubedock on OpenShift because the failed pods are immediately deleted when the test run fails. Here's how you can debug this:

1. Add a Thread.sleep(60000); to your test class before testcontainers are being stopped
2. Open the pods view of your Tekton namespace in the OpenShift Console and search for "kubedock" (see screenshot below)
3. Run the pipeline and watch the pods list until you see a kubedock pod
4. Click on the pod and look at the logs. They should reveal some information why the container couldn't start.  
     
   Unfortunately failed pods are deleted a few seconds later by Kubedock which makes analysis a bit hard.  
   For a more in-depth investigation, continue with the following steps:
5. Open the YAML view of the kubedock pod and copy it
6. After the pod vanishes again, remove the status part of the YAML and import it in OpenShift
7. Analyze the pod details and logs to see why the testcontainer fails

By default you're not allowed to start your own pods in a managed ESTA Tekton namespace. Talk to us to get this permission in order to debug testcontainers.

