# **POC for creating documentation on GitHub using GitHub Pages**

**Goal**:

Find out whether it is feasible and meaningful for EST to create future documentation for projects/products that use open source software on GitHub using the GitHub Pages mechanism for generating HTML out of markdown “on the fly”.

To achieve this, we checked whether and how the following tasks can be achieved:

1. Fork the K8S documentation repository and create a landing page that can be used for CLC as well as for other Fujitsu open source projects
2. Strip down the K8S content for CLC purposes
3. Deliver the K8S dashboard documentation in Fujitsu-branding
4. Avoid the duplication of content
5. Rebase with the K8S repository – maybe automatically solve merge conflicts
6. Include documentation created with DitaWorks on the GitHub Pages

## Task 1: Fork the K8S documentation repository and create a Fujitsu-branded landing page

The repo can be forked and the mechanism that K8S uses to generate their documentation web site is feasible under the following circumstances:

A new organization had to be created because otherwise no left-hand navigation could be achieved, because K8S uses relative paths for their links. It was not possible to use a project-specific gh-pages in the dashboard repo of EST on GitHub. The landing page can be accessed (in anology to K8S) with the following link: <https://clcdocs.github.io>

The Fujitsu branding and the new landing page was achieved as follows:

* A new SASS stylesheet (branding.sass) was added and referenced in styles.sass.
* The index.html (the landing page) was modified, lots of K8S stuff was removed.
* The CNAME file was removed (K8S uses the CNAME file to set up a subdomain with their DNS provider). Therefore, the K8S domain name is <https://kubernetes.io>, whereas .github. is appended to “our” domain name.
* The header and footer (head-header.html and footer.html) that appear on every page were adapted. In the header, an additional button was defined so that there are now links to the K8S documentation, the CLC documentation, and a sample (Try CLC).
* The following images were changed/added:
  + favicon.png
  + features.png
  + fujitsu\_transparent.svg
  + nav\_logo.svg
  + nav\_logo2.svg

## Task 2: Strip down the K8S content

This can be achieved by editing the yaml files. However, links in the original K8S documentation must be checked everywhere. In addition, the content itself must be checked and, for example, references to the GCE must be removed.

Problem: Rebasing with the original K8S doc repo becomes impossible. The above steps would have to be repeated every time.

## Task 3: Deliver the K8S dashboard documentation in Fujitsu branding

The point here is that screen shots are included in the ui.md file. One solution to this could be to reuse the ui.md file and create a separate one (e.g. ui-fujitsu.md) and include the branded screenshots there.

However, to my mind, it does not make sense to duplicate the ui.md file. Since it is probably not possible to work with the original K8S sources and re-brand them, we should not spend the effort to re-brand the dashboard documentation, but focus on contributing to the K8S documentation.

## Task 4: Avoid the duplication of content

This is one of the challenging parts. It is related to Task 2 and Task 5, which imho is not possible with a reasonable amount of effort.

## Task 5: Rebase with the K8S repository

If we work with a fork of the K8S repo and strip down the K8S docu to what is supported by CLC, we will never be able to rebase with the original repo, because we will always end up in numerous merge conflicts.

## Task 6: Include documentation created with DitaWorks on the GitHub Pages

For PDF documentation generated out of our XML sources, this is no problem at all. For HTML content, the following process could be used:

1. Generate the HTML files in DitaWorks
2. Create a markdown page with the Table of Contents for the HTML files. Take the generated index.html file as the basis.
3. Leave the structure as is in the HTML files including images, references, etc.
4. Add the TOC markdown page to the .yaml files used for

Conclusion:

Feasible/easily reachable

* Set up a GitHub Pages repository for EST documentation or whole projects.
* A landing page for different projects with links to project-specific documentation
* Reuse the mechanisms implemented by K8S for generating the documentation
* Producing project-specific documentation for NEW projects in markdown format (for open-source projects only; CT-MG is excluded since a lot of conditional tags and shared text is used)

1. Create organization and user: gertisdemo
2. Fork the K8S repo and rename to gertisdemo
3. Add the branding and CLC doc and push to origin/master
4. Edit the guides.yaml file and remove K8S doc

Conclusion:

With reasonable effort, it is not possible to reuse / rebrand the K8S documentation repository. Therefore, I now continue with setting up a GitHub Pages site that could be used for any EST product-specific documentation. This repo is based on the K8S documentation repo; it reuses the mechanisms for generating TOCs etc.

You can have a look at

<https://gertisdemo.github.io>

The site can be customized as follows:

1. Add new content

I wish this page field:

Define in a javascript: js/scripts.js:

Redirects to <https://github.com/gertisdemo/gertisdemo.github.io/issues/new?title=I%20wish%20>” +

Before: Activate Issues in repository settings – **has no effect…**

**Setting up a GitHub Pages Site**

<https://help.github.com/articles/what-is-github-pages/>

<https://help.github.com/articles/creating-pages-with-the-automatic-generator/>

**Blogging with Jekyll**

Custom URLS

Custom domain

Project pages or product pages

GitHub Pages is a static site hosting service.

* GitHub Pages source repositories have a recommended limit of 1 GB.
* Published GitHub Pages sites have a recommended limit of 1 GB.
* GitHub Pages sites have a recommended bandwidth limit of 100 GB or 100,000 requests per month.

Jekyll is used as static site generator with GitHub Pages. Jekyll makes it easy to create site-wide headers and footers without having to copy them across every page. It also offers some other advanced templating features:

* Jekyll uses the Liquid templating language to process templates. It supports all of the standard Liquid tags and filters. Jekyll also add a few filters and tags of its own to make common tasks easier.

Liquid code can be categorized into objects, tags, and filters.

* Objects tell Liquid where to show content on a page. Objects and variable names are denoted by double curly braces:
  + Input: {{ page.title }}
  + Output: Introduction

In this case, Liquid is rendering the content of an object called page.title, and that object contains the text Introduction.

* Tags create the logic and control flow for templates. They are denoted by curly braces and percent signs. The markup used in tags does not produce any visible text. This means that you can assign variables and create conditions and loops without showing any of the Liquid logic on the page.
  + Input:

{% if user %}

Hello {{ user.name }}!

{% endif %}

* + Output:

Hello Adam!

Tags can be categorized into three types:

* + Control flow tags can change the information Liquid shows using programming logic:
    - Case/when

Creates a switch statement to compare a variable with different values. Case initializes the statement, when compares its values.

* + - If

Executes a block of code only if a certain condition is true.

* + - Unless

The opposite of if executes a block of code only if a certain condition is not met.

* + - Elseif/else

Adds more conditions within an if or unless block.

* + Iteration tags run blocks of code repeatedly
    - For – repeatedly executes a block of code
    - Break – causes the loop to stop iterating when it encounters the break tag.
    - Continue – causes the loop to skip the current iteration when it encounters the continue tag.
    - For (parameters)
      * Limit – limits the loop to the specified number of iterations
      * Offset – begins the loop at the specified index
      * Range – defines a range of numbers to loop through. The range can be defined by both literal and variable numbers.
      * Reversed – reverses the order of the loop
    - Cycle – loops through a group of strings and outputs them in the order that they were passed as parameters. Each time cycle is called, the next string that was passed as a parameter is output.
    - Cycle (parameters)
      * Cycle group for cases where you need multiple cycle blocks in one template.
    - Tablerow – generates an HTML table. Must be wrapped in opening <table> and closing <table> HTML tags.
    - Tablerow (parameters)
      * Cols – defines how many columns the tables should have
      * Limit – exits the tablerow after a specific index
      * Offset – starts the tablerow after a specific index
      * Range – defines a range of numbers to loop through. The range can be defined by both literal and variable numbers.
  + Variable tags create new Liquid variables.
    - Assign – creates a new variable. Wrap a variable in double quotes to save it as a string. {% assign foo = “bar” %} {{ foo }}
    - Capture – captures the string inside of the opening and closing tags and assigns it to a variable. Variables created through {% capture %} are strings.
    - Increment – creates a new number variable, and increases its value by one every time it is called. The initial value is 0.
    - Decrement
* Filters change the output of a Liquid object. They are used within an output and are separated by a hash | .
  + Input:

{{ “/my/fancy/url” | append: “.html” }}

* + Output:

/my/fancy/url.html