Red Hat UKI DSA DevEx (Developer Experience)
OpenShift4
Guide to writing Labs

Precis

The latest version of the DevEx project has been engineered to allow anyone to both write/contribute labs and to generate workshops of any format that they need.

The approach allows for presenters to generate workshops based on any combination of the provided labs; this could be a workshop with a single lab, a workshop used as a demo of distinct technologies, a workshop focusing on specific technologies within the OpenShift space.

The source and releases of the DocBuilder are available at https://github.com/utherp0/workshop4

This guide explains how to write labs for the workshop. These labs can be submitted and contributed to the core set of labs within the project, or can be kept locally for customers or demos that are specific and contextual to customers.

The advantage of this approach is that you can tailor the workshops exactly to the need of the end attendee - if you need, for example, to present on Integration with OpenShift you can produce a workshop with just the Camel-K, Knative and other technologies.

Building the Workshop

Currently in order to build a workshop you need to have locally installed Maven (3.6+) and a JDK (8+).

You then need to clone the workshop repo (address above). When this has finished you will see there are two top level directories to be aware of - releases and labs.

Releases contains a prebuilt executable JAR file containing the DocBuilder. If you run this (using 'java -jar releases/xxxx.jar') it will list the parameters needed to execute a build.

These are:

1. Working directory. This is created by the DocBuilder
2. Manifest file. This defines the workshop; the metadata needed for branding the workshop and an ordered list of the labs required in the workshop documentation
3. The git clone directory root. This is the top-level directory where you cloned the repo. The DocBuilder uses this for the material to scaffold the build (labs, images)

4. The output directory. This needs to exist but be empty. The reason for this is that in the near future the DocBuilder will be provided as a Container Image which will have a running version of Apache in it along with the collateral to build the workshop. As part of the DocBuilder the output will be the existing http source directory so far the time being the DocBuilder assume the output directory exists.

There are examples of the manifests provided in the labs/manifests directory. These are provided for reference. A single run of the DocBuilder uses a single manifest.

The repo also contains a register, in the labs/manifests. This contains the short name for each lab, along with who is responsible for it, what version of OCP it was tested against and an English overview of the lab.

When you build a manifest from scratch it has the following format:

```
Facilitator: UKI DSA Team
Email: uther@redhat.com
Title: Domain Solution Architect Team
ClusterURL: example.manifest.com
DocumentTitle: Developer Introduction Workshop
----
introductiontooc
applicationbasics
deployments
applicationdeploymentstrategies
sdn
rbac
pvs
healthprobes
```

The first five lines are metadata for the workshop to be generated. Then, after the ---- divider, there is an ordered list of shortnames for the labs to include. Note that the DocBuilder automatically adds the Introduction and Prerequisites to every workshop. This is important to the structure of the labs and is discussed in the next section.

Once the build has completed the output directory will contain the generated HTML, PDF and an image directory (for hosting the HTML - it uses local referencing, i.e. “images/x”). You can copy these files to, for example, a HTTPD container for hosting or view them locally.

Writing a Lab

Each lab is an asciidoc document which is added to the complete source for a workshop and compiled. In the repo the registered labs are in the labs/src directory and it is suggested that when writing a lab you use one of the existing ones as a base to work from.

Each lab assumes that the prerequisites have been completed - it is worth putting a note at the start reminding attendees.

You can assume that the attendee has a Project call terminalX (where x is the user number) where command line commands can be entered.

The lab should create its own Project - the attendee should do this as the first action. All actions in the lab should take place only in Project(s) created in the lab. At the end of the lab the attendee should be instructed to remove the projects created as part of the lab.

There are no real style definitions for the labs but it is suggested that new labs follow a similar style as existing ones in order to maintain the flow and look of the workshop documentation. The loose style definitions are as follows:

1. All code extracts or code examples are wrapped in [source](return)----(return)*code*(return)----
2. Use TIP: and NOTE: to explain meanings external to the flow of the lab - i.e. an overview of the objects or a reference to some other technology
3. Screenshots are very much encouraged. Add them to the lab using image::(filename) and add the images to the images/ directory for the source (DocBuilder copies all images from labs/images when building the docs regardless of which labs are in the manifest for simplicity sake
4. Remember to close any asciidoc comments - if you open a comment with //// and don't close it the DocBuilder will carry the comment on and not render any labs after that point.

If you want to contribute a lab simply add the source and images into the repo and do a pull request. If the lab is suitable for wider distribution it will be merged and an entry will be added into the register.txt - this will allow people to write manifests that reference it.

Note that the DocBuilder verifies the existence of the labs during the build - if your manifest references a lab that doesn't exist the DocBuilder terminates the workshop build.