Syndesis Developer Handbook

Version 1.1.3-9586-gfaceb1033, 2019-04-26

Syndesis Developer Handbook

Chapter 1. Introduction

This handbook contains all information required for installing, running and developing Syndesis from a developer's point of view.

Chapter 2. Buildtool "syndesis"

Syndesis uses a single tool for controlling various aspects of the build and developer related tasks. This script can be found in `$SYNDESIS_DIR/tools/bin` and is called `syndesis`. It requires bash and can be used on any Unix or macOS.

To have this script handy all the time it is recommended to either put this `bin/` directory into the path or add a symlink from `syndesis` into a directory which is already on your execution path.

```
# Navigate to syndesis project directory
cd $SYNDESIS_DIR

# Set path to include Syndesis' tool directory
$ export PATH=${PATH}:$(pwd)/tools/bin

# Alternatively, set a symbolic link to "syndesis"
$ ln -s $(pwd)/tools/bin/syndesis /usr/local/bin
```

The script can be used for various tasks which are selected by a so-called command which is the first argument to the script.

Just type `syndesis -h` to get an overview of the commands available:

Usage message

```
Usage: syndesis <command> [... options ...]

with the following commands

   build            Build Syndesis
   completion       Shell completion
   dev              Syndesis developer tools
   doc              Generate Syndesis Developer Handbook (SDH)
   install          Install Syndesis to a connected OpenShift cluster
   minishift        Initialize and manage a Minishift developer environment
   release          Perform a release
   system-test      Run system tests

"build" is the default command if no command is specified.
```

There are a handful of global options which can be used:

`--help`	`-h`	Print usage information. If a command is given print out commands specific options
`--rebase`	`-r`	Rebase your project directory before building to integrate upstream changes to your local repository. See below for details how this works.
`--verbose`		Set verbose mode, which is useful mostly only for debugging the script itself.

Rebase to upstream

To easily rebase on changes which have been merged upstream to master, you can use the option `--rebase` (short: `-r`). This command assumes that you have forked the Syndesis GitHub repositories and you have "origin" and "upstream" remotes like

```
$ git remote -v
origin    git@github.com:rhuss/syndesis.git (fetch)
origin    git@github.com:rhuss/syndesis.git (push)
upstream  https://github.com/syndesisio/syndesis (fetch)
upstream  https://github.com/syndesisio/syndesis (push)
```

With this in place, a `--rebase` performs the following steps:

- Refresh upstream remote: `git fetch upstream master`
- Try a `git rebase upstream/master` which rebases your current local working branch.
- If this fails because you have uncommitted work:
  - A `git stash` is performed
  - The rebase is retried and should succeed
  - `git stash pop` brings back your changes. Stashing can fail with conflicts which you would have to resolve on your own.

Development Modes

The Syndesis application consists of a set of Docker images OpenShift resources descriptors for installing Syndesis. For development, minishift is used, and most of the commands assume that you have minishift installed locally and executable directly from your path. Minishift can be downloaded and installed from https://github.com/minishift/minishift/releases

For development OpenShift S2I builds and image streams are used. This mode works also when running with a real OpenShift cluster and is not restricted to Minishift usage. The advantage is that a build automatically triggers a redeployment, so you don't have to kill any pods manually.

Commands

All other options are specific to each command. You get a list of those options with `syndesis <cmd> -h`. These options are described in detail in the next sections.

The following commands are available:

Command	Description
build	Used for building Syndesis and its various modules.
ui	Start the UI for local development *(not implemented yet)*
minishift	Start and install Syndesis on Minishift
install	Install Syndesis in a running cluster (other than Minishift)
system-test	Run System test against an OpenShift cluster
dev	Utility commands useful during development

Command	Description
doc	Generating and publish this documentation
release	Release Syndesis' Maven and NPM artefacts *(not implemented yet)*

If no command is given, `build` is the default. Remember a command must be the first argument, but there are additional possibilities to specify commands:

- You can use the form `--command` anywhere as an option, too. E.g. using `--minishift` is the same as specifying "minishift" as the first argument.
- A `--mode command` can be used, too (e.g. `--mode sytem-test`)

The next sections describe the commands in detail. To add a new command, just drop a script file into `$SYNDESIS_DIR/tools/bin/commands` directory with following structure:

```
#!/bin/bash

yourscriptname::description() {
    echo "Describe the command"
}

yourscriptname::usage() {
    cat <<EOT
    Describe the usage of the command
EOT
}

yourscriptname::run() {
    Do your stuff
}
```

2.1. syndesis build

The primary goal of `syndesis build` is to build Syndesis.

It is mainly a wrapper around Maven and Go but adds some convenience functionality for common developer workflows.

2.1.1. Usage

```
Build Syndesis builds

Usage: syndesis build [... options ...]

Options for build:
    -b  --backend                 Build only backend modules (core, extension,
integration, connectors, server, meta)
        --all-images              Build all modules with images: ui, server, meta,
```

```
s2i, operator, upgrade, camel-k
        --app-images               Build only application modules with Docker images
(ui, server, meta, s2i)
                                   and create images
        --infra-images             Build only infrastructure modules with Docker
images (operator, camel-k, upgrade) and create images
    -m  --module <m1>,<m2>, ..     Build modules
                                   Top level modules: ui, server, connector, s2i,
meta, integration, extension, common, operator, camel-k, upgrade
                                   Submodules: defined with [groupId]:artifactId (e.g.
:connector-sql)
    -d  --dependencies             Build also all project the specified module depends
on
        --skip-tests               Skip unit and system test execution
        --skip-checks              Disable all checks
    -f  --flash                    Skip checks and tests execution (fastest mode)
    -i  --image                    Build Docker via s2i images, too (for those modules
creating images)
        --docker                   Use a plain Docker build for creating images. Used
by CI for pushing to Docker Hub
        --multi-stage              Multi-stage Dockerfile support for operator build
(experimental)
    -p  --project <project>        Specifies the project / namespace to create images
    -g  --goals <g1>,<g2>, ..      Use custom Maven goals to execute for the build.
Default goal is install
    -e  --ensure                   If building the operator , run 'dep ensure' before
building. Otherwise ignored.
    -l  --local                    If building the operator, use a locally installed
go
                                   Otherwise run the Go build from a container (either
local or Minishift's Docker  daemon)
        --clean-cache              Used for the operator build to remove the local
dependency cache.
    -c  --clean                    Run clean builds (mvn clean)

        --resume <m1>              Resume build from maven module (mvn -rf m1)
        --batch-mode               Run mvn in batch mode
        --camel-snapshot <version> Run a build with a specific Camel snapshot. If no
argument is given the
                                   environment variable CAMEL_SNAPSHOT_VERSION is used
or an error thrown.
        --maven-mirror <url>       Use the maven mirror for the build, e.g. a maven
repo proxy in OpenShift. If no
                                   argument is given, the MAVEN_MIRROR_URL environment
variable is used or

'http://nexus.myproject:8081/nexus/content/groups/public', if none is set.
                                   See also https://docs.openshift.com/container-
platform/3.5/dev_guide/app_tutorials/maven_tutorial.html
        --man                      Open HTML documentation in the Syndesis Developer
Handbook
```

2.1.2. Modules

A plain `build` command without any options performs a plain `mvn install` for all Java and UI modules. Plus it also builds the infrastructure operator via Go (see syndesis-build-operator for details). The goal `install` is the default but you can customize the Maven goals with `--goals` (short: `-g`) option.

This compiles all Java and Javascript artifacts and also runs all tests and code checks for the application modules.

You restrict the build to certain modules, which are divided into two categories: *application modules* which are the modules building up Syndesis. And *infrastructure modules* which help in managing the application itself.

You can individually select specific modules by using the `--module` (short: `-m`) option with a comma-separated list of modules. Modules are either top level modules that combine a set of Maven submodules in a logical unit or individual submodules defined with `[groupId]:artifactId`.

The following top level modules are available:

Module	Description	`--all-images`	`--backend`	`--app-images`	`--infra-images`
server	Main backend providing a REST API for the user interface				
ui	The SPA user interface application				
meta	Meta data and verifier used for verifying connections and providing connector metadata				
connector	All connectors used by Syndesis out of the box				
integration	Support libraries for running integrations				
extension	Tools for developing and running Syndesis custom extensions				
common	Common modules containing common code				
s2i	S2I base image used for building the runtime images				
operator	Infrastructure operator for managing the application				

All option ending with `-images` will also build the corresponding Docker image.

In addition to using top level modules that combine to a set of submodules you can also specify a very specific submodule using its groupId and/or artifactId. For instance the module name for the SQL connector module would be `io.syndesis.connector:connector-sql` or simply `:connector-sql`. In case you skip the groupId the leading colon in the module name is very important because this identifies the module to be a submodule rather that a top level module.

When you build individual modules you can provide the option `--image` (short: `-i`) to create also the Docker image in the build, when the module is associated with a Docker image.

By default images are build via S2I against a running Minishift. This is the recommended way for developing as this automatically will trigger a redeployment after the build. However, for certain scenarios like when used in a CI system or when doing the release, the image creation can be done against are Docker daemon when the `--docker` is given. For this to work you must have access to Docker daemon, which you can verify with `docker ps`.

When the option `--dependencies` (short: `-d`) is given in addition to `--modules`, also all Maven modules which the specified modules depend on are build, too.

2.1.3. Tuning

By default, all checks like license or code quality checks are performed. Also, all unit and local integration tests are run. A full build eats up quite some time, but you should always run at full blast before submitting a pull request.

However, to speed up the turnaround, several speed-up options are available. The following table shows these options, and also how long a full clean build over all modules takes: (but without building images)

Option	Description	Time
none	Default mode with all checks and tests	
`--skip-tests`	Skip all unit and local integration tests	
`--skip-checks`	Skip sanity checks like for correct license headers and	
`--flash`	Fastest mode with skipping all checks and tests and with even some other aggressive optimizations	
`--maven -mirror <url>`	Use a maven mirror, e.g. a maven repo proxy in OpenShift with cached dependencies. See also [setting up a maven mirror](https://docs.openshift.com/container-platform/3.5/dev_guide/app_tutorials/maven_tutorial.html).	
`--goals <goal>`	Use custom Maven goals to execute for the build. Default goal is `install`.	

2.1.4. Infrastructure Operator

`syndesis build` can also build the infrastructure operator, which is a golang program.

You can build the operator by running `syndesis build -m operator` or as part of a module collection like `--all-images` or `--infra-images`

There are three modes, how the operator can be created:

- Running your go compiler locally
- Compiling in a local Docker daemon which allows volume mounts with the localhost
- Compiling in Minishift Docker daemon, which was made accessible via `eval $(minishift docker-env)`

See below for the details.

Load dependencies

In any case, before you compile first you should use the option `--ensure` (short: `-e`) to setup the dependency tree. This will download all source dependency and cache them locally. To get rid of this cache, use the option `--clean-cache`. This might be necessary when `go dep` 's cache gets into a weird state.

If you compile for the first time, then `--ensure` will be added automatically.

Compiling locally

This is the fastest way for compiling the operator. Use the option `--local` (short: `-l`) for selecting the local compile mode.

It is also the recommended way when you are working on the operator. You project setup needs to fit however: The main project directory must be reachable as `$GOPATH/src/github.com/syndesisio/syndesis`. You can either move your project directory to this location or work with a symlink:

```
cd ~/Development/syndesis
mkdir -p $GOPATH/src/github.com/syndesisio
cd ..
mv syndesis $GOPATH/src/github.com/syndesisio/
ln -s $GOPATH/src/github.com/syndesisio/ syndesis
```

By default this compiles into for your native architecture (amd64, darwin). When you use this mode with `--image` (short: `-i`) on macOS then go will be used as cross compiler so that the generated bimary can be used in a Linux image.

Compiling with a local Docker daemon

This is the default mode and is used also when doing the release. It use a builder image `syndesis/godev` which is created from the `tools/image` directory and fetched from Docker Hub.

For this mode to work your Docker daemon must support volume mounts to the system from where you are calling `syndesis`. This is the case on Linux for locally installed Docker daemon and for Mac with *Docker for Mac*. It is **not** the case for Minishift which runs in a disconnected VM. But see below how you still can use Minishift for building.

`dep ensure` and `go build` will be run from this `syndesis/godev` image, but with your local directory mounted into the container so that the fetched dependencies can be cached in the local directories `dep-cache` and `vendor` so that the can be reused for the next run. Also the binary will be stored in your local directory, but this will alway be a Linux (`amd64`) binary.

Compiling with Minishift

As Minishift is running in a remote VM you cannot bind a volume to your localhost. Therefor when running in Minishift mode the source code will be rsynced to a directory in the Minishift VM (`rsync` will be installed in the Minishift VM on the first run).

The build with `syndesis/godev` will then be started with a volume mount to the copied directory in the VM. After the build the generated binary is copied back with `rsync` to your local directory.

The Minishift mode is automatically detected and selected if `DOCKER_CERTS` contains a path to `.minishift`. You enable the Minishift Docker daemon for your local CLI with `eval $(minishift docker-env)`

Some simple benchmark reveals the following timings (in minutes) :

Context	Local	Local Docker (macOS)	Minishift Docker
Cold (no build cache)	0:40	10:09	1:36
Hot (with dep cache)	0:08	2:47	0:13

The fastest mode is obviously the local mode, followed surprisingly by Minishift. The local mode is probably slow because of how the macOS daemon mounts ts volumes (which can probably be optimized)

2.1.5. Examples

Some common usage examples for `syndesis build` are

```
# Build all images (app and infrastructure) with S2I
syndesis build --all-images

# Create all application images and re-deploy Syndesis in the
# Openshift cluster, but do it as fast as possible. Don't build
# any golang code
syndesis build --app-images --flash

# Create the infrastructure operator by running go locally
# and calling <code>dep ensure before</code>
syndesis build -m operator --local --ensure

# Use a Camel snapshot for a clean build, build all modules
syndesis build --clean --camel-snapshot 2.21.0-SNAPSHOT
```

2.2. syndesis ui

Commands useful for Syndesis UI development, a wrapper around typical `yarn` tasks

2.2.1. Usage

```
Syndesis UI tasks

Usage: syndesis ui [... options ...]

Options for ui:
    By default the 'ui' does nothing unless you pass one or more of the following
parameters:

    --install           Install UI package dependencies
    --lint              Run lint task
    --build             Build the UI code
    --run-tests         Run the tests task
    --watch-tests       Run the tests and watch for changes
    --serve             Serve the UI (defaults to staging, use --minishift or --local
to change this) and watch for changes
    --local             When serving, run against a local Syndesis server
    --minishift         Involve minishift as needed
-p  --project           Use the specified openshift project
    --nuke-everything   Use with --minishift, start or reset the minishift VM
```

2.3. syndesis minishift

With `syndesis minishift` you can adequately manage a minishift installation for hosting Syndesis. This command is especially useful for a simple and self-contained development workflow.

`syndesis minishift` requires that you have a current minishift in your path. You can download it directly from GitHub.

2.3.1. Usage

```
--install                 Install templates to a running Minishift.
-p  --project             Install into this project.
                          Delete this project if it already exists.
                          By default, install into the current project (without
deleting)
-c  --context             Install into this context. Default is 'minishift'
--reset                   Reset and initialize the minishift installation by
                          'minishift delete && minishift start'.
--full-reset              Full reset and initialie by
                          'minishift stop && rm -rf ~/.minishift/* && minishift start'
--memory <mem>            How much memory to use when doing a reset. Default: 4912
--cpus <nr cpus>          How many CPUs to use when doing a reset. Default: 2
--disk-size <size>        How many disk space to use when doing a reset. Default: 20GB
--vm-driver <driver>      Which virtual machine driver to use (depends on OS)
--show-logs               Show minishift logs during startup
--openshift-version <ver> Set OpenShift version to use when reseting (default: v3.9.0)
--tag <tag>               Syndesis version/tag to install. If not given, then the
latest
                          version from master is installed
--local                   Use the local resource files instead of fetching them from
GitHub
-o  --open                Open Syndesis in the browser
-y  --yes                 Assume 'yes' automatically when asking for deleting
                          a given project.
--memory-server <mem>     Memory limit to set for syndesis-server. Specify as "800Mi"
--memory-meta <mem>       Memory limit to set for syndesis-meta. Specify as "512Mi"
--test-support            Allow test support endpoint for syndesis-server
--man                     Open HTML documentation in the Syndesis Developer Handbook
```

2.3.2. Installing Syndesis

You can easily install Syndesis with the option `--install`. This option triggers the creation of all relevant OpenShift resources objects in the currently connected OpenShift project.

If you want to use a different project, then use `--project` (short: `-p`) to specify this project.

Any existing project will be deleted first when specified with `--project`. This option is also an easy and quick way to recreate a Syndesis installation.

2.3.3. Resetting Minishift

The quickest way to get a fresh Syndesis setup is to use `--project` which will install Syndesis into a clean, new project.

However, you can also recreate the whole Minishift installation with `--reset`. This will delete the Minishift VM (`minishift delete`) and create a new one (`minishift start`). It doesn't harm if the Minishift VM does not exist so that you can use `--reset` also on a fresh Minishift installation.

If you want to get a real clean installation use `--full-reset` which deletes the `~/.minishift` directory

which holds downloaded artefacts like the ISO image for the Minishift VM. Using `--full-reset` forces Minishift to re-download all those files.

There are several options which influence the re-creation of the VM:

Option	Description	Default
`--memory`	Memory to use for the Minishift VM.	4 GB
`--cpus`	Number of CPUs used for the Minishift VM.	2
`--disk-size`	Disk space used for Minishift.	20 GB
`--show-logs`	Whether to show OpenShift logs during startup.	false
`--vm-driver`	Which virtual machine driver to use. For OS X this can be 'virtualbox', 'xhyve' or 'vmwarefusion' (if insalled).	
`--openshift-version`	OpenShift version to use	3.7.1

2.3.4. Example

This short example performs the following actions:

- Stops and deletes a running Minishift VM (if existent)
- Removes `~/.minishift` (if existent)
- Install Syndesis in OpenShift modes (S2I builds & image streams) in project `syndesis`
- Open Syndesis UI in the default browser

```
# Complete fresh installation in project "syndesis"
syndesis minishift --full-reset --install --project syndesis

# Open Syndesis in default browser
syndesis minishift -o
```

2.4. syndesis system-test

The `system-test` command is for running a full blown system test of Syndesis by installing it in one of multiple projects which are managed in a pool.

2.4.1. Usage

```
Usage: syndesis system-test [... options ...]

Options for system-test:
    --project <project>       The test project to use
    --token <token>           Token for connecting to the server
    --server <url>            OpenShift server url to use for the tests.
                              If not given, use the currently connected server
    --pool <project>          If no project is given, use a pooling mechanism.
                              This pool has to be created before with --create-pool
    --test-id <id>            Id to identify the test run
    --create-pool <prefix>    Create project pool for system-tests with
                              all projects with the given prefix
    --list-pool               Show all locks for the pool
    --release-project <t-id>  Release project for given test id (or all if no test id
is given)
```

2.4.2. How it works

This section needs a more detailed explanation how the systems tests are working in detail.

2.5. syndesis dev

Dev commands are useful helpers for developing Syndesis

2.5.1. Usage

```
Usage: syndesis dev [... options ...]

Options for dev:
    --debug <name>            Setup a port forwarding to <name> pod (default: server)
```

This command enable port-forwarding of port 5005 from a specific pod (by default: "server") to port 5005 on the localhost. You then can point your Java IDE to port 5005 on localhost for connecting for remote debugging. As argument to `--debug` "server", "meta" and "atlasmap" can be used, which are our Java based services.

2.6. syndesis doc

This command is used to generated and manage this documentation which you are currently reading.

2.6.1. Usage

```
Usage: syndesis doc [... options ...]

Options for doc:
-d  --directory <dir>         Top-level dir holding doc source and output directory.
Default: "doc/sdh"
-i  --input <file>            Input file to use. Default: "index.adoc"
    --out <dir>               Directory to generate files (default: "output")
    --html                    Generate HTML pages
    --pdf <out>               Generate PDF and write it to <out> (default: "sdh.pdf")
    --epub <out>              Generate Epub and write it to <out> (default:
"sdh.epub")
    --gh-pages <msg>          Create everything into the gh-pages branch and commit
with <msg>
-o  --open                    Open HTML documentation in default browser after doc
generation
-l  --local                   Use locally installed commands instead of Docker image
```

`syndesis doc` uses by default a Docker container to create the documentation. Therefore a Docker daemon must be accessible, and it must allow bind volume mounts to the local `doc/sdh` directory. Bind mounts are possible for Linux Docker daemons and for "Docker for Mac". But it is not the case for the Minishift exposed Docker daemon, as this daemon is running isolated in a VM.

You can always run `asciidoctor` locally with the `--local` (short: `-l`) options. See the Asciidoctor manual for more information how to install Asciidoctor.

2.6.2. Output

Input and output has the same defaults but can be changed with the following options

`--input <file>`	`-i <file>`	Input document in Asciidoc format which is by default `$SYNDESIS_DIR/doc/sdh/index.adoc`
`--output <dir>`	`-o <dir>`	Output directory, default is `$SYNDESIS_DIR/doc/sdh/output`

By default `syndesis doc` creates the documentation in HTML format, but more formats are supported:

`--html`	Create HTML documentation in the output directory
`--pdf <name>`	Create PDF documentation. `<name>` is optional, but when given, then this specifies the file name generated in the output directory. By default `sdh.pdf` is used.
`--epub <name>`	Create the documentation in Epub format. `<name>` is optional, but when given, then this specifies the file name generated in the output directory. By default `sdh.epub` is used.

2.7. syndesis release

Use `syndesis release` for performing a release of Syndesis. A Syndesis release consists of:

- Maven artefacts of the backend and runtime services

- Docker images pushed to Docker Hub
- A set of OpenShift templates referencing these images

This chapter describes how you can efficiently perform a release and how to troubleshoot if something goes wrong. This documentation might also be interesting to you even when you do not perform a release on your own, as it might help you to understand how the various Syndesis artefacts fit together.

2.7.1. Usage

```
Perform a release

This command performs a release but does no automated rollback when something fails.
Please refer to the manual at https://docs.syndesis.io for more details and what needs
to be cleaned up in case of a failure.

Usage: syndesis release [... options ...]

Options for release:
-n  --dry-run                 Dry run, which performs the whole build but does no
tagging,
                              artefact upload or pushing Docker images
    --release-version <ver>   Version to release (e.g. "1.2.1"). This is a mandatory
argument.
    --snapshot-release        Snapshot release which can be created on a daily basis.
                              A timestamped version will be created automatically, and
no Maven artefacts
                              are pushed to maven central. No moving tag will be
moved, too.
    --dev-version <version>   Next development version. If not given, set to
                              "<major>.<minor>-SNAPSHOT" as calculated from
                              --release-version (e.g. "1.2-SNAPSHOT")
    --settings <file>         Path to a custom settings.xml to use for the release.
                              This file must contain all the credentials to be used
for Sonatype.
                              By default ~/.m2/settings.xml is used.
    --local-maven-repo <dir>  Local dir for holding the local Maven repo cache.
                              If not given, then a new
                              temporary directory will be used
                              (and removed after the release)
    --docker-user <user>      Docker user for Docker Hub
    --docker-password <pwd>   Docker password for Docker Hub
    --no-git-push             Don't push the release tag (and symbolic major.minor
tag)
    --git-remote              Name of the git remote to push to. If not given, its
trying to be pushed
                              to the git remote to which the currently checked out
branch is attached to.
                              Works only when on a branch, not when checked out
directly.

    --log <log-file>          Write full log to <log-file>, only print progress to
screen
    --skip-tests              Do not run tests
    --no-strict-checksums     Do not insist on strict checksum policy for downloaded
Maven artifacts
    --man                     Open HTML documentation in the Syndesis Developer
Handbook
```

A lot of output is produced during a release. If you are only interested to see the major steps only,

then you can use `--log` to redirect the output to a specific log file. This log file will contain all output (and if you add `--verbose` you see even more output), but on the console you will only see the current step that is actually performed.

Example

An example run for a dry run for `1.3.1` release on the current branch look like:

```
./tools/bin/syndesis release           \ ①
    --release-version 1.3.1            \ ②
    --local-maven-repo /tmp/clean-repo \ ③
    --log /tmp/build.log               \ ④
    --dry-run                            ⑤
```

① Always run `syndesis` from the repo and branch you want to release.

② The release version is mandatory and must be in the format `<major>.<minor>.<patch>`.

③ Use a clean local Maven repository to avoid side effects

④ Redirect the full output to `/tmp/build.log` but still print the main steps to the console.

⑤ Make only a dry run, without pushing any artefacts out nor checking in any changed files.

2.7.2. Preparations

To perform a release, certain preconditions need to be given.

First of all, you need to have access to the various systems to which release artefacts are uploaded:

- You need to be logged in to Docker Hub and your account needs to have write access to the syndesis Docker Hub organisation.
- You have `gpg` to have installed and set up a gpg-agent for being able to sign Maven artefacts during deployment in a non-interactive mode.
- You need to have access to the "syndesis" account on (oss.sonatype.org for being able to publish Maven artefacts. This credential needs to be added to either your `~/.m2/settings.xml` or you can use an settings file with the `--settings-xml` option. The credential needs to be added to the server with the id `oss-sonatype-staging`.

You have to perform the release from a locally checked out Syndesis repository, which can be either checkout from a branch like `1.2.x` for a patch release or directly from `master`. It is highly recommended to run the release directly from a freshly checked out directory:

```
# Go to a temporary directory
cd /tmp

# Clone repository afresh, but only use the last history entry (--depth=1)
git clone --depth=1 https://github.com/syndesisio/syndesis.git

# Jump into the directory
cd syndesis

# Switch to the target branch if needed (or stay on master)
git fetch --depth 1 origin 1.2.x:1.2.x
git checkout 1.2.x

# Call Syndesis from the checked out clone
./tools/bin/syndesis release --release-version 1.2.8 .....

# Push to origin after a successful release.
# This automatically done if --no-git-push is given
git push 1.2.8
git push -f 1.2

# Remove the temporary clone again
cd ..
rm -rf syndesis
```

Please note that you should always call `syndesis` out of the branch for which the release is for. If there is an issue due to bugs in the release script itself, please fix them on the branch with the usual developer process (i.e. opening a PR request). `syndesis release` must always work for the branch where this script is, too.

2.7.3. Release steps

A release consist of several different steps, which can be grouped into two groups:

- **Build steps** are performed to build the release and create the artefacts. Also during the build Maven artefacts are uploaded to the staging area for publishing to Maven central
- **Persist steps** are then used for releasing objects, pushing Docker images to Docker Hub, committing and tagging in Git (but only when the build steps have been performed successfully).

Build steps

- Check whether the current local Git clone is *clean*, i.e. that is does not have any modified files. The script will abort if this is the case.
- Update the versions of all `pom.xml` files below `app/` to the version given with `--release-version`. If no `--release-version` is given, then the script aborts.
- Run an `mvn clean install` to verify that the build is not broken and all tests succeed.

- Re-generate the OpenShift templates in `install` so that the image streams included in these templates refer to Docker images with the new version.
- Now run an `mvn -Prelease clean deploy` to deploy all artefacts to a new staging repository on oss.sonatype.org, the platform for release artefacts on Maven central. The staging repository on this Sonatype Nexus is validated and closed.
- If `--docker-user` and `--docker-password` is given, then a `docker login` is performed. Otherwise, it is assumed that the user is already logged in.
- The Docker images are created with `mvn -Prelease,image package` in the `server`, `meta`, `ui` and `s2i` modules.

If the option `--dry-run` (short: `-n`) is provided, the script drops the staging repository at Sonatype and stops. You should examine the generated files and before starting a real build, reset the repository (`git reset --hard`).

The builds are using a clean local Maven repository, which otherwise is usually taken from `~/.m2/repository`. This new local cache should ensure that we have a completely fresh build without interference from previous builds store in the local Maven cache in the home directory. You can provide such a directory with `--local-maven-repo` which will be taken directly (so it's good if you have to perform multiple runs like with `--dry-run`). If not provided, a new temporary directory is created and also *deleted* after the release run.

Persist steps

- Push Docker images to Docker Hub. In addition to the images that carry the full release version as the tag, also a tag for the *minor version* is attached and pushed. E.g. when the release version is `1.2.8`, then the minor version is `1.2`. If this minor version tag already exists on Docker Hub, its moved to the newly created version.
- The staging repository on Sonatype is released. It will take a bit, but the artefact should then be downloadable from Maven central soon after.
- Commit all modified local files to the local Git repo.
- Create a Git tag for the release version (e.g. `git tag 1.2.8`).

The next steps are for creating templates for the minor version:

- In `install` create new templates which contain image streams that reference images with the minor version (e.g. `syndesis/syndesis-server:1.3` for a release version of 1.3.8).
- Commit those generated templates
- Tag it with the minor version (e.g. `1.2`), overwriting an already existing minor version tag

Next, we are switching back to the next development version of the pom.xml files. This version can be given with `--dev-version`, but by default, it is calculated automatically as `<minor.version>-SNAPSHOT` (e.g. `1.2-SNAPSHOT`). This new version is then committed to the local git repository.

Finally, the tags just created on the local Git repo is pushed to the remote repository. You can omit this with the option `--no-git-push`. If to so, the last step can also be performed manually afterwards with:

```
git push 1.2.8
git push -f 1.2 ①
```

① Using `-f` as the minor tag needs to be moved.

Please be careful to **not** push the master branch upstream (i.e. do **not** a plain `git push`). We only want to have the tag with all the release preparation steps, not on the branch so that pull requests can be still be easily rebased with out conflict because of the temporary version changes.

2.7.4. Minor Version Templates

What is now the thing with this *minor version*? Why is the needed and how does it work?

Syndesis follows a semantic versioning approach. So, patch level releases (i.e. all releases which only change the last digit in 1.2.8) are fully compatible with all other patch level versions. In order to allow easy bug fix upgrades, we also create a tag which contains only the version parts up to the minor version (e.g. 1.2). These tags **always** points to the latest full version of its minor version. If, e.g. 1.2.8 is the latest 1.2.x version, then the tag 1.2 point to this 1.2.8 version. Corresponding to these Docker image variants, there exist two OpenShift templates variants:

- One set of templates directly references the Docker images which its full version, e.g. `syndesis/syndesis-ui:1.2.8`. Applying such a template will keep your application at precisely this patch-level. You would have to update your templates and recreate your applications if you want to upgrade.
- The other set of templates references images only via its minor version, e.g. `syndesis/syndesis-ui:1.2`. Using these templates has the advantage that application created from these templates automatically benefit from patch releases. The templates contain an image change trigger which will redeploy the application if the images change. So when we release the next patch level release, moving the minor version tag to this patch level release, then the application gets automatically redeployed, and it will pick up the new image.

These two sets of templates can be reached directly from GitHub as the git tags correspond to the Docker tags (i.e. a `1.2.8` tag and a `1.2` tag which will be moved forward).

2.7.5. Snapshot Release

With the option `--snapshot-release` a lightweight snapshot release for the images and templates can be created. The tag/version is calculated automatically by picking up the latest release number (e.g. 1.3.5), increasing the patch-level by one and adding a daily timestamp (e.g. 1.3.6-20180419). According to Semantic Versioning 2.0 this is considered to be a version larger than 1.3.5 but smaller than 1.3.6.

This tag can be referenced to in `syndesis install` and `syndesis minishift`.

In detail, a snapshot release differs from a normal release as it:

- … doesn't release artefacts on Maven central, but pushes Docker images and creates a Git tag for referencing the proper templates.

- … skips all checks and tests when building to maximise the likelihood that the release succeeds. The rationale here is to better have untested daily snapshot release than no snapshot release because of test failure (which in many cases are not because of errors, but of failure in the infrastructure)
- … force pushes the snapshot tag on GitHub so that multiple releases per day are allowed

Example

```
syndesis release \
     --snapshot-release \                    ①
     --local-maven-repo /tmp/clean-repo \    ②
     --git-remote origin \                   ③
     --docker-user "${DOCKER_USER}" \        ④
     --docker-password "${DOCKER_PASSWORD}"
```

① Enable snapshot release with a version in the format 1.3.5-20180419

② Point to an empty repository to avoid side effects when building

③ Push to the origin repository

④ Docker credentials required for pushing to Docker Hub

A daily Jenkins job with this configuration run on https://ci.fabric8.io for creating a daily snapshots.

2.7.6. Troubleshooting

When you run the `syndesis release` command and when it should not succeed, you might have to perform some cleanup steps yourself (there is now automatic rollback). However, care has been taken to move all persistent changes to the end of the release flow, so if something breaks early, you only need to clean up locally. If the process fails before the step === *Pushing Docker images* you only need to:

- Reset your local git repo with `git reset --hard`
- Potentially remove the create staging repository on `http://oss.sonatype.org/` (but it doesn't harm if it is not cleaned up immediately).

After pushing the Docker images, it should be improbable that things go wrong. But these things should take care of if this should be the case:

- Remove Docker Hub tags for the pushed images, which is best done on the Docker Hub Web UI
- Revert your local git commits to the point before the release. If you did this on a fresh checked out repo (as recommended), you just could delete the whole clone.

2.8. syndesis install

With `syndesis install` you can install Syndesis to an arbitrary OpenShift cluster. If you want to install to Minishift the syndesis minishift command is recommended as it supports some additional features specific to Minishift.

2.8.1. Usage

```
-s  --setup               Install CRDs clusterwide. Use --grant if you want a specific
user to be
                          able to install Syndesis. You have to run this option once
as
                          cluster admin.
-u  --grant <user>        Add permissions for the given user so that user can install
                          the operator in her projects. Must be run as cluster admin.
    --cluster             Add the permission for all projects in the cluster
                          (only when used together with --grant)
-p  --project             Install into this project. Delete this project
                          if it already exists. By default, install into the current
project
                          (without deleting)
    --operator-only       Only install the operator but no resource
    --route               Route to use. If not given, the route is trying to be
detected
                          from the currently connected cluster.
    --console             The URL to the OpenShift console
    --tag <tag>           Syndesis version/tag to install. If not given, then the
latest
                          version from master is installed
    --dev                 Prepare for development of Syndesis so that S2I builds of
                          Syndesis images are picked up properly (implies --watch)
    --force               Override an existing "Syndesis" if present
-r  --route <route>       Use the given route.
                          By default "syndesis.<minishift ip>.nip.io" is used.
-w  --watch               Wait until cluster is up
    --local               install from local Git repo when using. By default the
                          resource descriptor is downloaded from GitHub remotely.
-o  --open                Open Syndesis in browser when installation is ready (implies
--watch)
-y  --yes                 Assume 'yes' automatically when asking for deleting
                          a given project.
    --memory-server <mem> Memory limit to set for syndesis-server. Specify as "800Mi"
    --memory-meta <mem>   Memory limit to set for syndesis-meta. Specify as "512Mi"
    --test-support        Allow test support endpoint for syndesis-server

You have to run --setup --grant <user> as a cluster-admin before you can
install Fuse Online as a user.
```

The deployment happens to the currently connected OpenShift cluster. So it's mandatory that you have logged into the cluster with `oc login` before. You can check the status with `oc status`.

The installation process consists of two steps:

- An initial setup which has to be performed as *cluster admin* which is a one-off action which needs to be done only once.

- Installation of Syndesis into a specific project by the *app admin*, a regular user of OpenShiftm which can be performed as many times as required.

2.8.2. Initial Setup

In the initial setup, you have to register the custom resource definition (CRD) to allow to deploy `Syndesis` resources. This step has to be performed by the admin.

Also, if you want to allow an OpenShift user to install Syndesis on her own, then you have to grant specific permissions to her.

To perform this setup step, which needs to be performed only once per cluster, you have to run `syndesis` **while being connected as a cluster admin**. For `minishift` use `oc login -u system:admin` if you have the `admin-user` addon enabled.

```
syndesis install --setup
```

This will install only the CRD. In addition to grant a user *developer* the proper permission to create a `Syndesis` resource, you should add `--grant <user>`:

```
syndesis install --setup --grant <user>
```

This call adds permissions to read and write `Syndesis` resource objects for the current project. If you would instead want to allow the user managing `Syndesis` resources in the whole cluster, you should add a `--cluster`. This cluster-wide access is especially required when you plan to use the `--project` option to use a new project or recreate the existing one, as in this case the role association to this project gets lost.

2.8.3. Installing Syndesis

After the CRDs a registered, you can easily install Syndesis directly with

```
syndesis install
```

Depending on whether you have granted the current user access this step has to be done either as admin or as a regular user.

Example for Minishift

```
# Enable the admin user on Minishift
minishift addons enable admin-user

# Create a minishift instance
minishift start --memory 4192

# Switch to admin
oc login -u system:admin

# Register CRD and grant permissions to "developer"
syndesis --setup --grant developer --cluster

# Switch to account developer
oc login -u developer

# Install Syndesis
syndesis install
```

A route name can be given with `--route`. This step can be omitted as the operator can autodetect the route. If you provide the route manually, you need to check your OpenShift installation. Typically the route name is the name of your OpenShift project followed by the cluster's hostname. E.g. a route `--route proj186023.6a63.fuse-ignite.openshiftapps.com` is specific to the Fuse Ignite test cluster `6a63.fuse-ignite.openshiftapps.com` and for the project `proj186023`. However, you don't have to provide the route name.

If you want to have a link to the OpenShift console to read the Pod logs, you have to add the `--console` option with the full URL to the console. Unfortunately, this cannot be auto detected. If not given, no link appears.

By default, this commands installs Syndesis in the currently connected project, but you can specify an alternative project with `--project <project>`. If this project already exists, it gets deleted unconditionally before the deployment, so be careful when using this option. By default, you are asked whether you want to delete the project for recreation. You can switch off the security question with the option `--yes` (short: `-y`).

Don't use `syndesis install --project $(oc project -q) --yes`. You'll shoot yourself into the foot. Ask the author if you want to know more details.

If you want to wait until everything is running (including fetching of the Docker images), you can specify `--watch` (short: `-w`) which blocks the script until everything is set up.

You can also automatically open Syndesis in the browser after the installation with `--open` (short: `-o`)

2.8.4. Development mode

As with syndesis minishift you can also use this command to set up a development platform for

Syndesis. *Development platform* here means that you can create Docker images on your own with syndesis build and can use them with an automatic redeployment after the build.

You can switch on this mode with the option `--dev`. When the operator has deployed the application, the application imagestreams refer to Docker images pushed to Docker Hub. To change the imagestream references to the images built with `syndesis build --all-images`, these imagestreams needs to be patched after the initial images have been fetched from Docker Hub. If you use the `--dev` option, then this update is done automatically.

2.8.5. Selecting the version

With the option `--tag` you can select a specific version of Syndesis to install. By default, the currently checked out checked out branch is used.

Example

```
syndesis install --route syndesis.192.168.64.12.nip.io --tag 1.4
```

This example installs the latest Syndesis version of the 1.4 branch to the local cluster.

You can see a list of available tags with `git tag`. Tags prefixed with `fuse-ignite` are suited for the Fuse Online cluster as those templates do not contain images streams themselves but refer to the image streams installed on this cluster.

2.8.6. Quick installation

The following scripts are not yet updated and probably don't work as expected. Please stay tuned.

If you only want to install Syndesis without developing for, there is even an easier without checking out Syndesis into a local Git repository.

You can directly use the standalone installation script syndesis-install for installing Syndesis. Just download this script, save it as "syndesis-install" and then call it with

```
bash install-syndesis --route $(oc project -q).6a63.fuse-ignite.openshiftapps.com
--open
```

Or, if you feel fancy (and trust us), then you can directly install the latest version of Syndesis by deleting and recreating the current project with a single line:

```
bash <(curl -sL https://bit.ly/syndesis-install) -p $(oc project -q) -r $(oc project
-q).6a63.fuse-ignite.openshiftapps.com -o
```

All you need is to have `bash`, `curl` and `oc` installed and you need to be connected to an OpenShift cluster.

Use `install-syndesis --help` for a list of options (which is a subset of `syndesis install` described above)

Chapter 3. Syndesis Development

3.1. Maven Groups

Syndesis uses Maven as build tool. Maven groups are used to separate the various Syndesis parts.

In details Syndesis consists of the following groups:

Group	Maven	Docker Image	Description
common	`io.syndesis.common`		Syndesis shared common module
connector	`io.syndesis.connector`		Supported camel connectors
rest	`io.syndesis.rest`	`syndesis/syndesis-server`	REST backend for managing integrations. This is the main sever.
integration	`io.syndesis.integration`		Library used in the the integration runtimes
s2i	`io.syndesis.s2i`	`syndesis/syndesis-s2i`	S2I base image for building integrations
ui	`io.syndesis.ui`	`syndesis/syndesis-ui`	User interface SPA, talking to the REST backend
meta	`io.syndesis.meta`	`syndesis/syndesis-meta`	Service for connector meta-data and verification of connections
extension	`io.syndesis.extension`		Library and API for developing Syndesis extensions
test	`io.syndesis.test`		System tests for testing the whole applications

[syndesis groups] | *syndesis-groups.png*

Figure 1. Group dependencies

3.1.1. Naming Conventions

The following conventions are used for naming directories, modules and Java packages.

These conventions are mandatory and should be also checked for when doing pull request reviews.

- Each directory directly below `app/` is specific for a certain Maven group. E.g. the directory `app/extension` is reserved for all Maven modules belonging to the Maven group `io.syndesis.extension`. The directory name is reflected as the last name part.
- All names (groups, modules, package) are using the **singular** form. E.g. its a `io.syndesis.connector`, *not* `io.syndesis.connectors`.
- Each Maven module is prefixed with the last part of the group name. E.g. the directory `app/integration/api` holds a Maven module for the the Maven group `io.syndesis.integration`,

and the module's artefactId is `integration-api`.

- A module's directory name is directly reflected as the last part of the Maven module name. If the Maven module name consists of multiple parts (e.g. artifact `integration-project-generator`), then the corresponding directory is also a concatenated word (like in `integration/project-generator`). Multipart names should be the exception, though.
- There should be only one level deep modules, so each Maven group directory holds all Maven modules flat.
- Each module has a **single** top-level package, reflecting the Maven module name. E.g. for the Maven module `common-util` in group `io.syndesis.common` has a single top-level package `io.syndesis.common.util` This top-level package should reflect the artefact name, with dashes replaced by dots.

Not every module has been already transformed to this scheme. This will happen step-by-step. But for new groups and modules this scheme has to be followed.

3.2. Issue Labels

We use GitHub labels to categorize epics, issues and tasks. They are the foundation of our process, so please use labels for issues.

Labels are living entities. This document describes the current status and might be slightly outdated. Please send a PR to adopt this section if the label structure changes. Also feel free to discuss the label structure anytime. It's essential that labels describe our process, not that we have to adapt our process for these labels.

Labels are grouped. Each label consists of two parts: A **Group** and a **Name** which are separated by a slash (`/`). For example, the label `module/ui` is used to mark issue which is relevant to the Syndesis UI module.

The following label groups are available. There must be only at most one label from the "Exclusive" groups.

Group	Description	Excl.
cat/	Misc categories which can be added freely	
prio/	Priority of the issue. Only one `prio/` label must be added per issue. `prio/p0` is of highest priority, `prio/p2` the lowest one.	
ext/	Reference to external projects	
source/	Where did the issue originate from (stakeholders)? i.e. `source/qe` indicates that QE raised this issue.	
group/	Internal Syndesis modules	
source/	Which external group has created the issue	
notif/	Notification label which can be added and removed to ping certain subteams	

Group	Description	Excl.
pr/	Labels which are only relevant for pull request and which have also some semantics for the bot managing pull requests	
size/	Tee shirt size for issues. Sizing is a subjective assessment and should be done relative to other issues.	
status/	Status of an issue or PR.	

Each label group serves a particular purpose, and for each issue and PR, it should be considered whether a label from a group applies.

3.2.1. Groups

Labels from this group reference our application groups like "rest", "ui" or "connector". Each sub-team is responsible for one or more group, and every group has an 'owning' team. That does not mean that members of other teams are not allowed to work on such groups. Contrary, this is even encouraged. But its just there so that teams can filter on issues and PRs which are relevant to them.

An issue can carry many group labels. Especially Epics will carry more than such label as they touch more than one group (otherwise it wouldn't be an epic).

For Java code, a "group" roughly corresponds to a directory directly below `app/`

Group	Description
group/common	Syndesis shared common module
group/connector	Supported camel connectors
group/extension	Tools for developing Syndesis extensions
group/install	Installing Syndesis (templates, scripts)
group/integration	Library used in the the integration runtimes
group/meta	Service for connector meta-data and verification of connections
group/operator	Infrastructure operator related
group/s2i	S2I base image for building integrations
group/server	REST backend for managing integrations
group/ui	User interface SPA, talking to the REST backend
group/uxd	User experience (UX) designs

3.2.2. Categories

Labels from the `cat/` group are labels which can always be applied and which does not fit in another category. Currently we have these categories:

Category	Description
cat/bug	A bug which needs fixing.
cat/build	For issues which have relevance for the build system.
cat/design	A concrete UX design. Use this for PRs containing UX designs.
cat/discussion	This issues requires a discussion.
cat/feature	PR label for a new feature
cat/process	Development process related issues carry this label.
cat/question	For issues holding a question.
cat/research	Label used for issues which describe some research work
cat/starter	An issue which is easy to solve and can be used for ramping up new developers.
cat/techdebt	Label for issues identifying technical debt.
cat/techdoc	Technical developer information (likes this handbook ;-) related issues.
cat/user-story	A user story, which might not be an epic

3.2.3. Pull Requests

This category of labels is all about pull requests. All of them have a meaning for the pure-bot bot which watches a pull request and performs certain action. These actions also involve monitoring and creating labels.

The following labels are involved:

Notification	Description
pr/approved	This label will be automatically applied to a PR as soon as the PR has been approved at the end of a review. It is an indicator for pure-bot to automatically merge the pull request if it passes all required tests. You should not set this label manually for approving a PR but using the GitHub button to do so.
pr/needs-backport	This pull request needs a corresponding backport to the latest patch branch.
pr/review-requested	In our process it is not mandatory to have a PR review. However, if the author requests a review via the normal GitHub functionality, this label gets applied automatically. When this label is set on a pull-request, then the mandatory status check `pure-bot/review-requested` will only pass if at least a single pull request has been given, so prevents manual merging (without forcing).
status/wip	This is a PR request label which should be used for "Work-in-Progress" kind of PRs which has been submitted for early review. If this label is present on a PR, the PR is not merged, even when it is approved. A dedicated mandatory status check `pure-bot/wip` monitors this labels and prevents merging if this label is present.
status/1.4.x	This pull request is against the 1.4.x patch branch (analogous labels might appear over time)

3.2.4. Notification

Notification labels from the `notif/` group serve a particular purpose. They are used when one team

wants to notify another group that a specific issue might have them relevance to them.

Notification	Description
notif/doc	The issue needs some attention from the docs team. This might because a new feature has been introduced or, more important, an existing feature has changed for which a documentation already exists.
notif/pm	The issue needs input from product management.
notif/triage	Every new issue gets this label and is considered during a triage session for properly priorisation and categorisation. Remove this label after the triage has happened.
notif/uxd	This label should be used for issues which needs some attention from the UX team. This might because a new feature has been introduced or, more important, an existing feature has changed for which a UX design already exists.

It is important to note that these labels also be removed when the notification has been received.

For example, when a UI feature like an input form changes. Then the UI team attaches a `notif/uxd` label to the PR which introduces this change. The UX team, detects with a filter search on this label, that there is a new notification. It then decides, whether UX design needs to be updated or not. In any case, they are removing the `notif/uxd` label and add a `module/uxd` label if this PR indeed requires a UX design update. If no update is required, then the label is removed without replacement.

3.2.5. Source

Labels starting with `source/` indicate the origin of an issue. It should be applied to help in triaging and priotizing.

Notification	Description
source/qe	This issue has been raised by QE

3.2.6. External references

This label group should be used if an external system is referenced, which is not part of the Syndesis mono repo.

External Project	Description
ext/atlasmap	atlasmap data mapper
ext/qe	syndesis-qe suite
ext/docs	syndesis-documentation End user documentation

For the future, we plan to add more of these external repos into the Syndesis mono repo (like documentation or QE). If this happens, then labels should be converted to `module/` kind of labels.

3.2.7. Status

Status labels are unique since they may trigger some automatic actions.

The current status labels are:

Status	Description
status/blocked	The current issue is blocked by another issue. Refer to the issue itself to see what is blocking this issued. This label is purely informal.

3.3. Issue Management and Communication

With Github as the primary tool for logging and handling issues, it is important to become proficient in utilising its interface and taking advantage of the extra tools available.

3.3.1. Displaying Issues

Issues can be displayed natively in Github by clicking the 'Issues' button at the top of the main repository page. This provides a list of all the issues which can be filtered according to text, status etc.

However, additionally a more informative dashboard is available using Zenhub. Simply navigate to their website, scroll down and click 'Zenhub Browser Extension' / 'Add the extension'. This will install their addon into Firefox or Chrome. Once installed, a Zenhub tab will be displayed in the main repository page, which once clicked on, will display a dashboard of the issues divided into respective swimlanes according to their status.

3.3.2. Being Notified of Issues

Once assigned to the project, the registered email address should being receiving notifications from Github concerning any modifications to any issues. This can quickly fill an inbox so filtering these into their own folder is recommended (a new developer can expect to see 50-100 per day). Additional tools, like Octobox, are available for helping with managing these notifications so please ask other project members if you are struggling to handle them.

3.3.3. Administering Issues

Issues are the bedrock of the project and provide the window into understanding what each developer is currently working on. Therefore, it is important to log issues, assign issues and provide comments to issues. Github/Zenhub provide tools to connect issues and Pull Requests (PRs) and even if the PR does not require an issue (rarely!) then it should be possible to provide sufficient context in the PR that spells out its nature and relevance.

Once assigned, an issue is your's to log as much detail as required to portray any problems, difficulties or solutions. Should you need help then Github provides syntax using the @UserName syntax to prompt other's to comment on the issue. Given the plethora of notifications that developers receive on a daily basis, it is recommended to use this syntax if you require additional input, since this sends additional notifications specifically asking the developer concerned to

comment. In the event that you are unsure who to contact then it is not impolite to include a number of @UserNames in the issue and someone will respond and either reply directly or in turn notify a person who can. Everyone understands the importance of collaboration so speaking up is to be only encouraged.

3.3.4. Closing Issues

Development is about solving the issues and closing them down. Therefore, closing issues is a good thing. Do not be afraid to close an issue if the problem has been solved or indeed if the problem has gone away on its own (yes this does happen!). In the event. that a code change has occurred and a PR submitted then that PR will have been reviewed by another developer. There may be a back-and-forth in comments and requests for changes but finally the PR will be approved by the reviewer. Once that happens, the PR will be automatically merged and closed. If an issue is directly linked to the PR then that too will be closed as well. Should this not be required then the issue can of course be re-opened. Once the issue is closed then it is back to the dashboard to find a new one!

3.4. Local Development

If you'd like to get a local development environment up and running, for both the UI and REST API, this is how you'd do it.

Tips

- Build on branch, not master.
- Callback URL Example: https://syndesis.192.168.64.29.nip.io/api/v1/credentials/callback

---

3.4.1. Requirements

You can follow these steps if it's your first time setting up Syndesis, or if you want a fresh local installation to replace an existing one. Some environment-specific instructions may be available below as well.

1. Make sure you have installed node version >= 6.x.x and Yarn version >= 0.18.1.
2. Get a developer deployment of Syndesis running in a Minishift environment as described in the Syndesis Quickstart. Most are specific to your environment, so follow the sections below for a quick setup. The general instructions are:
    - Install a hypervisor for Minishift.
    - Install Minishift.
    - Install the OpenShift CLI.
    - Make sure it's in your `$PATH`

macOS

If you'll be using the Homebrew method, you'll obviously need to have Homebrew installed. Then, to install the hypervisor for Minishift and Minishift itself:

```
$ brew install docker-machine-driver-xhyve
$ brew cask install minishift
```

Finally, to install the OpenShift CLI, we recommend using Homebrew: `brew install openshift-cli`

Linux & Windows

- Install a hypervisor for Minishift. For macOS, we recommend using the Docker xhyve plugin here, which can be installed using Homebrew.
- Install Minishift. For macOS, we recommend you use the Homebrew method.

Please note that you need to have the `oc` binary available in your `PATH`. To do that, see here: https://docs.openshift.org/latest/cli_reference/get_started_cli.html

3.4.2. First-Time Setup

The goal here is to download the project to your laptop/PC, and to install Minishift, the VM that contains OpenShift.

```
$ git clone https://github.com/syndesisio/syndesis.git # or own fork
$ cd syndesis
$ syndesis minishift --full-reset
```

3.4.3. Running without Kubernetes/OpenShift

For some development tasks you can run Syndesis without running on a Kubernetes/OpenShift cluster. You won't be able to publish integrations, see activities or gather metrics, or any other functionality that requires a running cluster. This is best suited for quick turnaround development of backend APIs together with UI features. You'll need three components running the Syndesis backend, the PostgreSQL database it uses and the UI.

Start by running a PostgreSQL database in a Docker container:

```
$ docker run -d --rm -p 5432:5432 -e POSTGRES_USER=postgres -e
POSTGRES_PASSWORD=password -e POSTGRES_DB=syndesis postgres
```

the `--rm` option will remove the container once done and all the stored data in the database with it, if you wish to keep the data omit that option

Next run the Syndesis backend, this step requires that you have downloaded all the dependencies and built the backend:

```
$ syndesis build -f -c --backend # to download dependencies and build the backend
$ (cd app && ./mvnw -f server/runtime spring-boot:run)
```

The backend is started once you see the a line containing this in the output:

```
Started Application in 28.9 seconds (JVM running for 29.615)
```

After that UI can be started by running:

```
$ (cd app/ui && yarn start:local)
```

You can now access a running instance at https://localhost:4200.

3.4.4. Day-to-Day

This uses an existing Minishift instance.

NOTE: If you already followed the First-Time Setup above, you do not need to follow this. The Minishift VM will already have been started. Simply skip to the

Get the Latest Changes

```
$ git checkout master
$ git pull upstream master
$ git checkout <branch>
$ git rebase master
```

Start of the Day Make sure Minishift is running.

```
$ minishift status
```

Which should look like:

```
Minishift:  Running
Profile:    minishift
OpenShift:  Running (openshift v3.6.0+c4dd4cf)
DiskUsage:  11% of 17.9G
```

If it isn't, start it with:

```
$ minishift start
```

Login into and Set up OpenShift

This step is required regardless of whether it's a first-time install or not. It logs you in and points OpenShift to use Minishift resources.

```
$ oc login -u developer
$ eval $(minishift oc-env)
$ eval $(minishift docker-env)
```

The eval's set a number of environment variables, like change the `$PATH` and `$DOCKER_HOST`, so each time you do a Syndesis build it's good to make sure those are invoked.

End of the Day

```
$ minishift stop
```

Start the UI App & Open in the Browser

```
$ yarn start:minishift
$ open https://$(oc get routes syndesis --template "{{.spec.host}}")
```

Resetting the Database

This step is optional. This command expects Minishift to be running already. It's the `-i docker` that determines the workflow, for Roland it seems to work without that though.

It would clean the database if we increase the schema version, if we don't it remains the same.

```
$ syndesis build -m rest -f -i docker -k
```

Alternatively, you can use the REST API Endpoint: /api/v1/test-support/reset-db

Connecting to the Database

You can also port forward the DB's port using `oc port-forward` and then connect to the database using a tool like pgadmin to view the data. First get the DB pod's name either from `oc get pods` or from the OpenShift console. Then use the following command:

```
$ oc port-forward <db pod name> 5432:5432
```

Now start pgadmin and add a new DB server, use `localhost` for the `host` setting. For the username and password look on the DB pod's `Environment` page in the OpenShift console.

In pgadmin you can see the table by navigating into the tree under `Server Groups > Servers > syndesis > Databases > syndesis > Schemas > public > Tables > jsondb`. Right click, and then go to

`View Data > View All Rows`.

3.5. UI

After you've set up your initial Local Development environment, you're ready to contribute to the UI.

3.5.1. Install Dependencies

From the project root directory:

```
$ cd app/ui
$ yarn install
```

3.5.2. Start up the App

Using Minishift resources (recommended):

```
$ yarn start:minishift
```

The `yarn start:minishift` command works when it can properly detect your local development machine's IP address. A proxy server inside the minishift deployment will use that IP address to connect back to the development server being run by the yarn command. If detection of the IP is failing for you, then set the `SYNDESIS_DEV_LOCAL_IP` env variable to your local machine's IP address before running the yarn `yarn start:minishift` command.

3.5.3. Open in Your Browser

Open the Syndesis UI in your browser from the command line by running:

```
# on macOS
$ open https://$(oc get routes syndesis --template "{{.spec.host}}")
```

```
# on linux
$ xdg-open https://$(oc get routes syndesis --template "{{.spec.host}}")
```

```
# on windows
$ start https://$(oc get routes syndesis --template "{{.spec.host}}")
```

Another option is to run `minishift console`, go into **My Project** and click on the URL for the Syndesis app.

To verify that you're running against the development instance of the UI check the title of the

browser tab you've opened and ensure it says `DEVELOPMENT` in in somewhere.

Not using Minishift resources In the event that you have issues with Minishift. Don't be surprised if most things don't load and there isn't any data in the UI. Only use this if you're totally blocked and need to work on something minor/aesthetic in the UI.

```
$ yarn start
```

3.5.4. Running Tests

To run tests or lint there are two more commands you can run in separate terminals.

```
$ yarn test
```

3.5.5. Running Linter

If you don't, the CI will, and your PR build will likely fail.

```
$ yarn lint
```

You should be able to access the UI in your browser at http://localhost:4200

3.5.6. Code Documentation

We use Compodoc for documentation, or here to see how to format comments. Files get generated automatically in the `/documentation` directory. Read the documentation [here](https://compodoc.github.io/website/guides/getting-started.html) to see how to properly document features. You can automatically generate and run docs using Yarn:

```
$ yarn compodoc
```

Or manually with `compodoc -s`, or `compodoc` if you want it to simply generate the files in the default `/documentation` directory and run it with an HTTP server.

3.5.7. Technology Stack

Included in this stack are the following technologies:

- Language: TypeScript (JavaScript with @Types)
- Framework: Angular
- Testing: Cucumber.js (BDD Unit Test Framework), Karma (Unit Test Runner), Istanbul (Code Coverage)
- Linting: TsLint (Linting for TypeScript)

- Logging: WIP
- Code Analysis: Codelyzer (TsLint rules for static code analysis of Angular TypeScript projects) / WIP

3.6. Troubleshooting

When things go wrong, you want to try to identify the area that is causing problems (UI, REST API, etc). If it's the UI, look for errors in the browser console or the terminal to see if it's a dependency issue.

3.6.1. UI Dependency Issues

```
$ rm -rf node_modules
$ yarn install
```

3.6.2. VM Trouble

Not getting latest API changes

This is a known issue. This is the workaround for using the latest REST image from the Docker stream.

NOTE: This deletes your Minishift instance, installs OpenShift templates for the pods, and restarts Minishift.

Disclaimer: It's not 100% clear what `-i` docker for `Syndesis Minishift --install` does exactly, but there is no way to invoke those evals before you get a running VM, which is what `--full-reset` does. So as a rule of thumb, you can have a terminal with those evals and keep it open and do all of the Syndesis building from there.

```
$ syndesis minishift --full-reset --install -p syndesis -i docker
```

syndesis command not found If you get `'syndesis' command not found` then use the full path to the `syndesis` binary instead. This assumes you are in the root of the project directory.

```
$ ./tools/bin/syndesis minishift --full-reset --install -p syndesis -i docker
```

If OpenShift templates have been updated This should not be the first choice, since it changes the IP of the VM, and in general should not be necessary for just building and updating the version.

```
$ syndesis minishift --full-reset --install
```

VM Trouble

```
$ syndesis build
```

Other things you can try: - `rm -rf ~/.minishift` - Check the OpenShift console and look for logs. - Is it a xip or nip problem? http://downoruprightnow.com/status/nip.io

3.6.3. Still Having Trouble?

Ask on IRC #syndesis