DevOps and R: Implementing an Entire R Development Life Cycle at Statistics Austria

To allow for blinded review:

Keywords: *R, DevOps, Statistical Software, development*

1 Introduction

In this contribution we present and outline the steps we have taken to facilitate the automated building and deployment of R packages [1] for internal usage at Statistics Austria. We discuss our environment that consists of an R server (Rstudio-Server Pro [2]), a git-server (Bitbucket [3]), a build-server (Jenkins [4]) and an artifact repository manager (JFrog Artifactory [5]).
In order to connect these servers, we developed a (complex) pipeline template and an internal utility package (useSTAT) which helps users to consistently set up R-based projects and linking those projects to a git repository and/or a continuous integration process.
We discuss the reasons for this setup and identify strengths and weaknesses of this approach. Furthermore, we show a typical use case and illustrate why we want to formalize our setup even more in the future to ensure consistent quality checks for all our internal R packages. [6]

2 Methods

2.1 DevOps

DevOps is a set of practices that combines software development (Dev) and IT operations (Ops). It aims to shorten the systems development life cycle and provide continuous delivery with high software quality. The main goal we want to achieve with our setup is reproducibility and consistency.

2.2 Automatic Building and Deployment

The main idea of our setup is that users that are developing R-packages internally should have an easy way to test, check and deploy their packages. In order to achieve this goal, we came up with the approach that is now discussed in the following sections. Figure 1 gives an overview on how the specific components of the process are interconnected.
After a Rstudio project has been linked with a git repository on the Bitbucket server (which can easily be done using the internal *useSTAT* package as shown in section 2.3), connecting the package to the build server is done via an http post-hook that has the same random value for both the git repository and the project on the build server. Setting up these hook can be tedious when done manually but is trivial when performed with package *useSTAT*. The purpose of this hook is that the git-server "notifies" the build server whenever an update to the repository has been applied. This allowed Jenkins (the build server) to start a new build whenever the repository has changed. We want to note however that the pipeline on the build server is not restricted to R packages. In fact, any project can be build, tested and deployed using custom scripts.

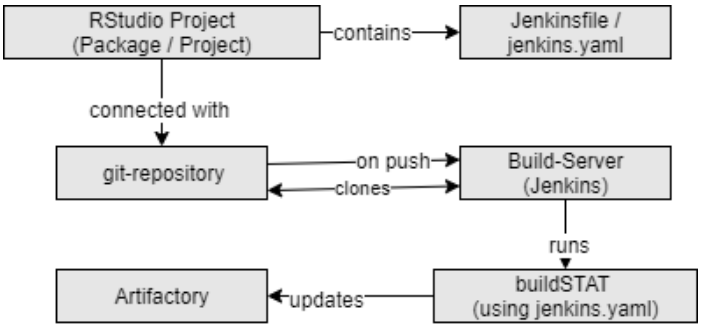

Figure 1: Overview of the DevOps process

In order to be able to build, check and test a project, two additional files are required:

- `Jenkinsfile`: allows to specify the pipeline that should be run as well as the docker image that should be used on the build server
- `jenkins.yaml`: contains options for the pipeline in simple `yaml`-format

Once an update has been pushed, the Jenkins build server clones not only the relevant repository that should be built but also a repository containing both the pipeline definitions as well as an internal R-package called *buildSTAT* that is used within the pipeline itself. The main purpose of *buildSTAT* is to make the process of modifying the pipeline itself as easy as possible.
The pipeline performs the following tasks:

- make sure that the docker-image used matches the image required in `Jenkinsfile`
- parses the `jenkins.yaml` file for possible errors
- it decides if an R-package or an R-project should be build and possible overwrites some options
- (optional) installation of (linux) system-packages, R-packages (from CRAN) and GitHub possibly specified in `jenkins.yaml`
- runs custom pre-build scripts possibly defined via the supplied options
- builds, checks and tests a package or runs custom scripts in case a project should be build taking a range of possible options into account
- (optional) runs post-build scripts including possible ssh-deployment of artifacts
- (optional) updates local CRAN repository by pushing correctly built R-package to the artifactory server

Since *buildSTAT* is always in the background, it is possible to define certain requirements before any package can be uploaded to the artifactory server. Such requirements may include version numbering or the automatic creation and deployment of a pkgdown [7] site for each package.
Using the pipline it is also possibly do deploy artifacts using the ssh-protocol to a webserver for easy access which makes it handy for example to keep R-based training materials. In our case, we use the pipeline to build our R training slides which are automatically built and deployed via the pipeline) ensuring that the material is always up-to-date without the need for manual work.

2.3 Userfriendly Tools for Development

In order to facilitate the usage, an internal R-package named *useSTAT* was written. The package design was modeled after the package *usethis* which allows users to set up GitHub repositories and travis jobs from the R command line via the REST API of GitHub and travis. Similary, *useSTAT* utilizes the REST API of both BitBucket and Jenkins to make sure a CI/CD workflow can be set up easily for new and existing projects. A new package can be set up with just a few function calls.

- `create_stat_package()` sets up a new package based on a template. The function can either be called directly or via a "new project wizard" which was developed as an rstudio project template.
- `use_statbucket()` creates a new BitBucket repository through the REST API and connects it to an existing project.
- `use_jenkins()` initializes a Jenkins project and connects it to the previously generated Bitbucket repository.

After this simple setup, every push to the Bitbucket repository will trigger Jenkins to run a package check and make sure the package developer is notified via mail in case this check causes any issues. As soon as the package is ready for production, a simple modification of `jenkins.yaml` can be used to include the package in the artifactory with the next push.

3 Conclusions

Setting up a toolchain for inhouse R developers should be as standardized and easy as possible, the *useSTAT* package can achieve just that. A defined setup facilitates easy hand-overs, good collaboration between developers and sustainability of the developed packages, product or code in general. Reproducibility is key to our work in official statistics and there version control and traceable deployment of R package is of utmost importance.

References

[1] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2020. URL https://www.R-project.org/.

[2] RStudio Server Pro. https://rstudio.com/products/rstudio-server-pro/. Accessed: 2020-10-06.

[3] Atlassian Bitbucket. https://bitbucket.org/product/enterprise. Accessed: 2020-10-06.

[4] Jenkins. https://www.jenkins.io/. Accessed: 2020-10-06.

[5] JFrog Artifactory. https://jfrog.com/artifactory/. Accessed: 2020-10-06.

[6] D.R. Rumsey. *Statistics For Dummies*. John Wiley & Sons, 2015.

[7] Hadley Wickham and Jay Hesselberth. *pkgdown: Make Static HTML Documentation for a Package*, 2020. URL https://CRAN.R-project.org/package=pkgdown. R package version 1.6.1.