On using Brane for data-processing pipelines

Roman Dahm[13598279], Kailhan Hokstam[13412485], Daan Vinken[12398233], and Yannick Brunink[11018747]

Universiteit van Amsterdam, The Netherlands

Abstract. This report presents an introductory experience of the Brane[1] framework, which aims to provide programmable orchestration of applications and networking. By discussing our experience of creating a modular data-processing pipeline for predicting if a computer will soon be hit with malware as part of the 2019 Microsoft malware prediction challenge [5] comprised of five Brane packages, we intend to inform the reader of the current possibilities and limitations of Brane. Please note that the time of writing, Brane is still under active development and not considered production ready.

Keywords: Brane · Containerization · Programmable Orchestration.

1 Introduction

The philosophy of the Brane framework is to allow domain scientists to freely compose data processing pipelines, without having to worry about the underlying implementation details or installation requirements. Domain scientists working with the proposed Brane pipeline are able to select a data-set, tune preprocessing parameters, and select the model and its parameters. To adhere to Brane's principle of modularity, we implemented and published five standalone packages `kaggle-brane`, `preprocessing`, `training`, `predicting` and `visualization` that together form a composeable pipeline to predict if a machine will soon be hit with malware as part of the 2019 Microsoft malware prediction challenge [5]. While the proposed pipeline only features a single model for demonstration purposes, the advantage of using a modular approach lies in the fact that the `training` and `predicting` packages can be easily swapped out for another predictive model.

This report is structured as follows: In the first section we briefly introduce each package in more detail. After, we discuss a possible Brane workflow by discussing our pipeline as a combination of the latter packages. Finally, we reflect on the experience of using Brane over the course of four weeks.

2 Brane data-processing pipeline

In this section, individual Brane packages that have been implemented will first be discussed separately and then with respect to their composition.

[1] https://docs.brane-framework.org/

2.1 Packages

To increase modularity and allow users of Brane to easily import individual packages, we chose to host the open source code for each package in their own public repository.

1. `kaggle-brane`[2]
   Kaggle[3] is an online community of data scientists and machine learning practitioners, hosting numerous datasets and challenges, that has become a vital accelerator for scientific progress and fosters interdisciplinary collaboration [2][1]. Our proposed pipeline too uses a dataset that was published on Kaggle as part of the 2019 Microsoft malware prediction challenge [5], which is why we decided to contribute a Brane package for interacting with the Kaggle API. At this point, only downloading challenge data is considered production ready, however, most of the Kaggle API, including finding datasets, submitting prediction challenge results or querying the leaderboard is implemented to offer users of Brane a straightforward integration.
2. `preprocessing` [4]
   The data processing pipeline is based on Vladislav Bogorod's solution to the Kaggle challenge [3]. The preprocessing package reads the CSV files that were downloaded from the Kaggle competition and into pandas[5] dataframes. It then transforms all features to categories, retains rows belonging to categories with more than 1000 occurrences, downsamples to create more balanced classes and then saves the resulting train and test set as files. Various techniques like using sparse matrices and transforming data in smaller groups are performed to reduce memory usage. The step that transforms all features to categories works for this specific dataset, but if a dataset with numeric features were to be used this transformation needs to be adapted, so that the number of categories does not become too large to reasonable handle.
3. `training` [6]
   The training package is still based on [3] and trains the `LightGBM` model using the preprocessed data generated by the `preprocessing` package. This package needs a path to files in a similar format as our the result of our `preprocessing` package. It stores the resulting `LightGBM` models as files. This package also does not train on the whole dataset at once to limit the amount of memory necessary.
4. `predicting` [7]
   The predicting package is still based on [3] and uses the `LightGBM` models saved from the `training` step to create a csv that is a valid Kaggle submission.

[2] https://github.com/romnn/kaggle-brane
[3] https://www.kaggle.com/
[4] https://github.com/yaaani85/wscbs2021-preprocessing
[5] https://pandas.pydata.org/
[6] https://github.com/yaaani85/wscbs2021-training
[7] https://github.com/yaaani85/wscbs2021-predicting

5. `visualization` [8]
   Prior to preprocessing the Malware prediction dataset, it can be of interest to explore the available data. Therefore a simple visualization package has been written and wrapped around the Matplotlib package. A *groupby* function has been implemented, to apply to columns of the dataset. Both a pie chart and barplot are available, to depict for instance the number of different platforms in the dataset. Additional features like dropping *not a number* (NaN) values and a threshold value for grouping have been added. An example of such a pie chart can be seen in figure 1.

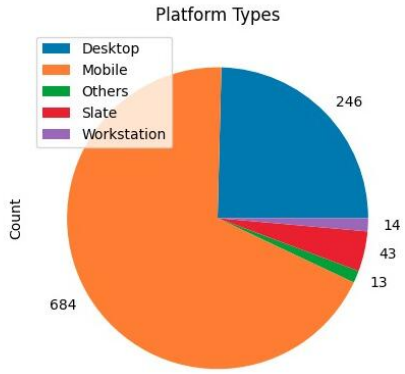

Fig. 1. Pie chart produced by the visualization package, representing the division of platform types.

2.2 Composed pipeline

The packages presented in 2.1 are embedded into the main project repository[9] in the form of git submodules in the *brane* directory. Instructions for how the full data processing pipeline looks like can be found in the *brane* directory.

2.3 Continuous Integration

When using Continuous Integration (CI), rapid feedback is provided through the use of automated builds and tests for every integration, which can lead to improved software quality [4]. To guarantee the public open source repositories maintain a functional state, we decided to use GitHub *Actions* to build and test every change to all package repositories. To encourage more Brane projects to make use of *Actions* to build and test their code, we also contributed

[8] https://github.com/daanvinken/wscbs2021-visualization
[9] https://github.com/hkstm/UvA.WSCBS2021

`setup-brane-action`[10], which can be used as a step in any *Actions* CI script to setup a Brane environment for testing. The action installs the Brane command line client and optionally starts a Brane instance and IDE. By default, prebuilt container images are used to speed up CI, however, it is also possible to force checking out the upstream source and rebuild. As of now, building and importing is tested for all packages, however, it must be noted that we did not find a convenient way to test functions of a package through BraneScript despite the susceptibility to mismatched `container.yml` specifications that should be tested more thoroughly for future work.

3 Discussion

Finally, we briefly discuss the experience of using Brane from both an end-user (domain scientist) and software engineering perspective.

From an end-user perspective, the possibility to use JupyterLab notebooks and an interactive REPL[11] is very comfortable and promising, however, we found that due to very rudimentary documentation on the features and syntax of the language it was difficult to start writing BraneScript code.

Furthermore, in the current early version of Brane, invalid statements such as importing unavailable packages did either not error or resulted in a crash, which is not user friendly for end-users with limited programming experience. We therefore believe that investing in more informative error messages would dramatically improve the overall end-user experience.

From an implementor's perspective, most members of the group also experienced problems and inconsistent behaviour as well, e.g. when running packages that output data work when invoked with `brane test`, but not in the Jupyterlab Notebook. Together with the fact that we were not able to find up-to-date examples of more sophisticated features such as instantiating objects in BraneScript, calling functions with nested types as input parameters, or deserializing function output with nested custom types, forced us to choose the most simple over a more elegant solution, which resulted in code duplication.

We also noticed the overhead of creating specifications for simple services to feel less natural compared to using an interface specification language such as ProtocolBuffers to generating service stubs and boilerplate code in any target language.

While working on setting up automated tests and code checks, we were also missing the ability to interpret entire BraneScript files without piping them into the `brane repl` to be able to write assertions about packages in BraneScript to be executed as part of the CI[12].

To conclude, we surely gained valuable experience understanding and debugging the implementation details of a reasonably complex distributed architecture such as Brane. Unfortunately, as was to be expected from software in the early

[10] https://github.com/romnn/setup-brane-action

[11] A read–eval–print loop (REPL) is an interactive language shell, where a program is executed line by line.

[12] Continuous Integration

development stage, we experienced numerous problems that hindered the development process, which is why it would be great to see the Brane framework mature to a more stable state in the future. To contribute beyond the feedback we provided as part of this report, we opened GitHub issues and pull requests for problems and fixes we encountered during the development.

Bibliography

[1] Adam-Bourdarios, C., Cowan, G., Germain, C., Guyon, I., Kégl, B., Rousseau, D.: The Higgs boson machine learning challenge. In: Cowan, G., Germain, C., Guyon, I., Kégl, B., Rousseau, D. (eds.) Proceedings of the NIPS 2014 Workshop on High-energy Physics and Machine Learning. Proceedings of Machine Learning Research, vol. 42, pp. 19–55. PMLR, Montreal, Canada (13 Dec 2015), http://proceedings.mlr.press/v42/cowa14.html

[2] Athanasopoulos, G., Hyndman, R.J.: The value of feedback in forecasting competitions. International Journal of Forecasting **27**(3), 845–849 (2011)

[3] Bogorod, V.: LightGBM: Baseline Model Using Sparse Matrix for microsoft malware prediction challenge. https://www.kaggle.com/bogorodvo/lightgbm-baseline-model-using-sparse-matrix (2019), accessed: 2021-06-04

[4] Duvall, P.M., Matyas, S., Glover, A.: Continuous integration: improving software quality and reducing risk. Pearson Education (2007)

[5] Microsoft: Microsoft malware prediction. https://www.kaggle.com/c/microsoft-malware-prediction (2019), accessed: 2021-06-04