The Journal of Open Source Software

Caliban: Docker-based job manager for reproducible workflows

Sam Ritchie[1], **Ambrose Slone**[1], **and Vinay Ramasesh**[1]

1 Google, United States of America

DOI: 10.21105/joss.02403

Software

- Review
- Repository
- Archive

Editor: Patrick Diehl

Reviewers:

- @lukasheinrich
- @arokem

Submitted: 22 June 2020
Published: 15 September 2020

Summary

Caliban is a command line tool that helps researchers launch and track their numerical experiments in an isolated, reproducible computing environment. It was developed by machine learning researchers and engineers, and makes it easy to go from a simple prototype running on a workstation to thousands of experimental jobs running in a Cloud environment.

Motivation

Modern machine learning research typically requires a researcher to execute code in multiple computing environments. To investigate some property of a machine learning model, the researcher has to write a script that can accept a path to some dataset, train a model using some set of configurable parameters, and generate measurements or a serialized model for later analysis.

Writing and debugging model training code is fastest on a local workstation. Running the script to generate measurements almost always takes place on some much more powerful machine, typically in a Cloud environment. The imagenet dataset (Deng et al., 2009) is 144 GiB, for example, far too large to process on a stock laptop.

Moving between these environments almost always requires a nontrivial amount of effort on the part of the researcher, unrelated to the original research question. In Python, a common language for machine learning research, the researcher installs their script's dependencies in a system-wide package registry. The Cloud environment can have different dependencies available, or different versions of the same dependency; different versions of Python itself; or different software drivers, which elicit different behavior from the script that seemed to work locally.

This environment mismatch introduces friction into the research process. The only way to debug a script that succeeds locally and fails in a remote Cloud environment is to attempt to interpret the often-cryptic error logs.

Docker

One solution to this problem is Docker (Merkel, 2014). A researcher can package their code and dependencies inside of a Docker container and execute this container on different platforms, each with different hardware options available but with a consistent software environment.

Many Cloud services (Google's Cloud AI Platform, Amazon Sagemaker etc) allow users to submit and execute Docker containers that they've built locally.

Packaging research code inside of a Docker container has many benefits for reproducibility (Cito & Gall, 2016). But the process of building a Docker container is difficult, error-prone and orthogonal to the skill set of a machine learning researcher. The friction of debugging between local and Cloud environments is solved, but only by accepting a not-insignificant baseline level of toil into the local development experience.

Projects like MLFlow (Zaharia et al., 2018) attempt to streamline the container creation process, but still force the researcher to absorb much of Docker's mental model.

Caliban and Reproducible Research

Caliban is a command line tool that solves this problem by providing execution modes with opinionated, intuitive interfaces for each phase of machine learning research - interactive development, local execution, cloud execution and data analysis in a notebook environment.

With Caliban, the researcher executes all code using Caliban's various subcommands. This process is, for the researcher, identical to the process of executing code directly. To prepare a research environment, all they need to do is specify required packages in a `requirements.txt` file and Caliban will automatically make those dependencies available inside the container.

Behind the scenes, all software execution has moved inside of a Docker container. This makes it transparent to move that execution from a local environment to Cloud, enabling a research project to grow from a simple prototype running on a workstation to thousands of experimental jobs running on Cloud with little to no cognitive load.

This removal of friction allows a researcher the freedom to be creative in ways that their psychology would resist, given the typical pain caused by moves between environments.

In addition, Caliban makes it easy to launch multiple jobs with varying command-line arguments with a single command, using experiment configuration files.

Impact

Before we introduced Caliban in our lab, researchers reported spending multiple weeks learning how to run their experiments at scale. Caliban allows a new researcher to reach this level of proficiency in under an hour. Multiple papers currently in preparation contain research results generated from thousands of experiments executed using Caliban. Given the limited tenure of a typical machine learning internship and residency, the efficiency boost offered by Caliban can materially change the scope of project that a researcher would be willing to take on.

In addition, any research conducted with Caliban that the researcher open sources is trivially executable by any interested party. The Docker environment managed by Caliban guarantees that anyone with access to the shared source code repository will be able to run experiments in an environment identical to the environment used in the original research program.

Caliban's Execution Environments

Caliban provides a suite of execution engines that can execute the containers built by Caliban.

`caliban shell` generates a Docker image containing any dependencies declared in a `requirements.txt` and/or `setup.py` in a project's directory and opens an interactive shell. Any update to code in the project's folder will be reflected immediately inside the container environment. The `caliban shell` environment is identical to the environment available during Cloud execution, up to access to different hardware.

`caliban notebook` starts a Jupyter notebook or lab instance inside of a Docker image containing the project's dependencies. As with `caliban shell`, the environment available to the notebook is identical to the Cloud environment.

`caliban run` packages a project's code into a Docker container and executes it locally using `docker run`. If the local machine has access to a GPU, the instance will attach to it by default, with no need to configure drivers.

`caliban cloud` allows a researcher to submit a research script to Google's AI Platform. The code will run inside the same Docker container available with all other subcommands. Researchers can submit hundreds of jobs at once. Any machine type, GPU count, and GPU type combination specified will be validated client side, instead of requiring a round trip to the server.

`caliban build` builds the Docker image used in `caliban cloud` and `caliban run` without actually running the container or submitting any code.

`caliban cluster` allows a researcher to create and submit jobs to a Kubernetes cluster. This environment is superficially similar to the Cloud environment offered by Google's AI Platform and `caliban cloud`, but a different range of hardware and pricing is available to the researcher.

`caliban status` displays information about all jobs submitted by Caliban, and allows a researcher to cancel, inspect or resubmit large groups of experiments.

Related Work

Reproducible Environments Binder (Forde et al., 2018) is a project that allows a researcher to open up Jupyter notebooks hosted in a git repository in a software environment described by the repository's `requirements.txt` file. The motivation is similar to `caliban notebook`, with the added benefit of being able to interact with a notebook in the browser, without any burden of configuring a local machine. repo2docker offers the same ability to execute a repository of notebooks in its required environment. This environment can be customized and configured in similar ways to containers build by Caliban.

Scientific Cloud Computing A related series of approaches to lowering the friction of scientific computing on the Cloud expose primitives to the user that can execute both locally, or in parallel on many Cloud machines. A "pure" Python function is a function depends only on its inputs. If a scientific experiment is built out of pure functions, it becomes possible to write a framework that can execute that function in parallel on a large set of distinct inputs in a Cloud environment. Pywren (Jonas, Venkataraman, Stoica, & Recht, 2017) is a project that makes this possible on AWS Lambda. Cloudknot (Richie-Halford & Rokem, 2018) extends this model to functions requiring more computational resources. Cloudknot packages code into a Docker image, providing the same potential level of configurability as Caliban.

References

Cito, J., & Gall, H. C. (2016). Using docker containers to improve reproducibility in software engineering research. In *Proceedings of the 38th international conference on software engineering companion*, ICSE '16 (pp. 906–907). New York, NY, USA: Association for Computing Machinery. doi:10.1145/2889160.2891057

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., & Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. In *2009 ieee conference on computer vision and pattern recognition* (pp. 248–255). Ieee. doi:10.1109/cvpr.2009.5206848

Forde, J., Bussonnier, M., Fortin, F.-A., Granger, B. E., Head, T., Holdgraf, C., Ivanov, P., et al. (2018). Reproducing machine learning research on binder. In.

Jonas, E., Venkataraman, S., Stoica, I., & Recht, B. (2017). Occupy the cloud: Distributed computing for the 99%. *CoRR*, *abs/1702.04024*. Retrieved from http://arxiv.org/abs/1702.04024

Merkel, D. (2014). Docker: Lightweight linux containers for consistent development and deployment. *Linux journal*, *2014*(239), 2.

Richie-Halford, & Rokem. (2018). Cloudknot: A Python Library to Run your Existing Code on AWS Batch. In Fatih Akici, David Lippa, Dillon Niederhut, & M. Pacer (Eds.), *Proceedings of the 17th Python in Science Conference* (pp. 8–14). doi:10.25080/Majora-4af1f417-001

Zaharia, M., Chen, A., Davidson, A., Ghodsi, A., Hong, S. A., Konwinski, A., Murching, S., et al. (2018). Accelerating the machine learning lifecycle with mlflow.