A Scalable Cloud-based Architecture to Deploy JupyterHub for Computational Social Science Research

Da Li, Robert Pyke, Runchao Jiang
Facebook, Inc.
Menlo Park, California, USA
{dali,robertpyke,runchaojiang}@fb.com

ABSTRACT

With the increasing popularity of computational approaches to conduct social science research, building a scalable and efficient computing platform has become a topic of interest for academia to empower research labs and institutes to analyze large-scale data. While social science researchers have been very excited about the advancement of emerging technologies in big data, deep learning, computer vision, network analysis, etc., they are also constrained by the available computing resources to analyze data. This paper describes a scalable solution to deploy JupyterHub for computational social science research on the cloud. We use a reference architecture on AWS to walk through the design principles and details. Our architecture has helped facilitate several collaborations between Facebook and academia. The case study (*Facebook Open Research and Transparency* platform) shows that our architecture, using technologies like containerization and serverless computing, can support thousands of users to analyze web-scale datasets.

CCS CONCEPTS

• **Computer systems organization → Cloud computing**.

KEYWORDS

Jupyter, Kubernetes, Cloud Computing, Scalability, Computational Social Science

ACM Reference Format:
Da Li, Robert Pyke, Runchao Jiang. 2021. A Scalable Cloud-based Architecture to Deploy JupyterHub for Computational Social Science Research. In *Practice and Experience in Advanced Research Computing (PEARC '21), July 18–22, 2021, Boston, MA, USA.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3437359.3465591

1 INTRODUCTION

Computational social science (CSS) usually refers to using computational approaches to study and explain social phenomena. CSS is a multi-disciplinary field that combines social science and computer science to deliver research outcomes in quantitative ways. CSS significantly extended the traditional empirical research approaches in social science to the extensive use of data analysis and computer simulation, which is considered revolutionary.

In the early years of CSS, researchers would use their personal computers to analyze collected data or conduct simulations. Nowadays, CSS research has become increasingly dependent on computing infrastructure availability, including data storage and computational resources. On the one hand, many advancements in computer science, including complex network analysis, neural networks, computer vision, text mining, etc., created excitement in CSS because of the potential to adopt similar tools and methodologies to analyze social science datasets. On the other hand, lack of programming expertise and data infrastructure resources have become barriers to conducting interdisciplinary research in computational social science.

Jupyter Notebook [10] allows a single user to write code and display inline results from a web-based interface. Once available, it quickly gained popularity among researchers. Later, the project *JupyterHub* [9] further extended the Jupyter Notebook to a multi-user platform, which makes it suitable for organizational use cases. An instance of JupyterHub offers both practical and legal/policy benefits because it allows a researcher to freely analyze data without needing a copy of it. This style of bringing "compute to the data" instead of the other way around is particularly beneficial when the data are very large (e.g., too slow and expensive to copy) and when there are legal or other limitations on its use (e.g., ensuring compliance via auditing and logging may be impossible without centralized infrastructure) [2]. This, along with the diverse themes and extensions available for the Jupyter ecosystem, make JupyterHub one of the best choices for scientific research communities.

However, JupyterHub only provides limited scalability for each of the notebook servers, thus is not capable of large-scale data processing in the notebook. To analyze large, especially web-scale datasets, users need to access a cluster and submit jobs from the notebook so that the cluster can process offloaded computation. This is difficult to set up for many small research labs and/or non-prestigious universities due to resource and budget limitations.

To address above-mentioned issues and democratize the data infrastructure of the social science research community, in this work, we propose a cloud-based architecture to host a scalable and cost-efficient JupyterHub deployment. Our contributions can be summarized as follows:

- We propose a scalable architecture to build the data infrastructure for computational social science research and exploit the state-of-the-art technologies, including containerization and serverless computing, to achieve elasticity and cost-efficiency.
- We present a reference design on AWS to reveal details and design principles. Our proposal is cloud-agnostic so that it can be extended to other cloud platforms.

- We demonstrate how to customize the software system in JupyterHub to provide flexibility for research without incurring maintenance overhead and security loopholes.
- Our proposal shows the potential to establish public standards that allow institutes to create data-sharing infrastructures for their own policy and legal purposes.

2 ARCHITECTURE

In this section, we present the proposed architecture in detail. We first give the design principles and an overview of the system. Then using a reference design, we walk through important components to demonstrate design and implementation trade-offs.

2.1 Design Principles and Overview

Based on our discussion in section 1, the proposed architecture follows a few design principles:

- **Security First**: For any platform with data, ensuring adequate security is always the first priority. In social science, researchers often need to access sensitive user data (e.g., personally identifiable information (PII) data, survey response) for analysis. So it's essential to take necessary measures to ensure the system can meet commensurate security standards.
- **Cloud-agnostic and Standard Components**: We want the architecture to be cloud-agnostic and consist of standard components. So with none or minimal modification, it can be deployed to common public cloud providers (Amazon Web Services (AWS), Google Cloud Platform (GCP), etc.) or on-premise data infrastructure.
- **High Scalability and Elasticity**: We aim to support terabyte to petabyte-scale datasets with the collaboration of tens to hundreds of researchers. We also want to be mindful of the costs and would like the system to be as elastic as possible. As a result, our design uses technologies like containerization and serverless computing.

Based on the above design principles, the architecture has three high-level components: a client-facing analytics environment, a proxy service (gateway), and a data processing infrastructure.

Client-facing Analytics Environment is where researchers can write, debug, and execute code to analyze the data. This requires the environment to offer a user-friendly interface with support for popular toolkits and programming languages.

Proxy Service serves as the middleman and the safeguard between the analytics environment and the backend storage and computing resources. The benefit of having the proxy service is to establish a security boundary between the client-facing analytics environment and other infrastructure services to secure the system and the data access.

Data Processing Infrastructure provides the storage and computing resources for the analytics environment. The data storage would include both SQL and NoSQL databases, as well as blob storage to store various structured, semi-structured, and unstructured data. The computing engine needs to be able to handle large-scale data with high efficiency.

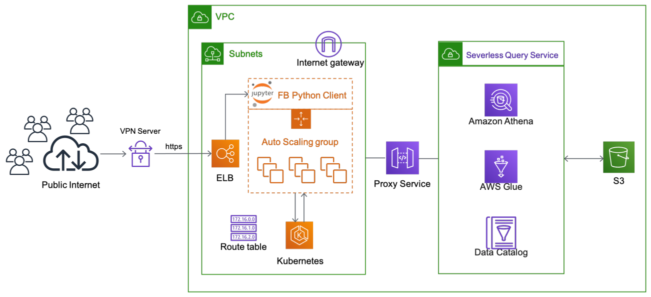

Figure 1: The reference design using AWS services.

2.2 Reference Design

Figure 1 shows a reference design of our proposed architecture on AWS. All the icons representing each service are following the standard of AWS terminologies.

The analytics environment is a JupyterHub instance deployed on Kubernetes. An *Auto Scaling Group* is configured to provide worker nodes to Kubernetes. JupyterHub is installed on *Amazon Elastic Kubernetes Service* (EKS) using Helm. With JupyterHub installed, logged-in users can create notebooks using supported kernels. Currently, Python, R, and Julia are supported in JupyterHub as the default kernels without further customization. But a complete list of kernels can be found on the official wiki [8]. The JupyterHub deployment is behind an *Elastic Load Balancer* (ELB) and accessible via HTTPS. In order to access the environment, users need to connect to a VPN service first.

The purpose of a Proxy Service is to provide a unified and secure call path to the backend services, including specific AWS resources (e.g., *Athena*, *Data Catalog*, *S3*) and other specific private APIs (e.g., login and authentication). Access to these AWS resources and private APIs is essential for users but granting access directly in the Jupyter notebook will incur significant security concerns. Thus, introducing the proxy service allows all the API calls to be invoked in a federated way, which is a significant security improvement. Also, the existence of the proxy service provides extra flexibility to handle various secrets without exposing them directly in the Jupyter notebook client environment.

Behind the Proxy Service are the AWS resources to process the data. The Jupyter Notebook is running on a Kubernetes pod, which offers computation resources to a user, but it's still a single node instance with limited CPU and memory available. In order to handle datasets at terabyte or even petabyte-scale, it's necessary to have a data warehouse-like backend to offer the storage and compute pool. AWS offers many solutions, including: *Athena* and *Redshift*. Athena is good for supporting the interactive query of web-scale datasets, while Redshift could give you a data warehouse-like capability.

2.3 JuputerHub on Kubernetes

JupyterHub is a multi-user platform to use notebooks to access computational resources. Its web-based UI makes it simple to provide access to users from web browsers on various devices. In our reference design, we deploy JuputerHub on Kubernetes. The Jupyter community has curated detailed instructions [1] for deployment in the cloud with Helm to manage Kubernetes applications.

Kubernetes is an open-source system for container orchestration and application management. It makes the deployment automation and horizontal/vertical scaling of containerized applications very easy. Kubernetes has become a popular system and supported by almost all major cloud providers. Our reference design utilizes the Amazon EKS, which is the AWS managed Kubernetes service. Amazon EKS utilizes an *Auto Scaling Group* (ASG), which contains a logical group of Amazon EC2 instances to perform automatic scaling and management. With ASG and Kubernetes Autoscaler, the JupyterHub on Kubernetes becomes a scalable and elastic solution to run with up to tens of thousands of users.

2.4 Authentication and Authorization

Authentication (also referred to as "AuthN") is the process of identifying a user/service. The result of AuthN is that the "Identity" is known. The identity consists of one or more principals (identifying pieces of information). JupyterHub Authenticator supports several OAuth-based login mechanisms, including Google and GitHub. We also implemented and open-sourced an authenticator based on Facebook Login flow.

Authorization (also referred to as "AuthZ") is the process of allowing a user/service access to a resource based on the "Identity". AuthN is knowing the identity; AuthZ is then giving that identity access based on its principals. By default, JupyterHub only supports two types of users: regular and admin. In our reference design, we implemented a more complicated permission system to federate access to different types of resources.

2.5 Proxy Service

The Proxy Service is implemented using *AWS API Gateway* with Lambda integration. The API Gateway defines: (1) The routing configuration for all calls. (2) The valid request models, call paths, and response models. (3) The integration type for each route (Lambda, Static-Content, EC2, S3, etc.) (4) The authorization configuration for each route. (5) The input/output transformations (optional) (6) The HTTP-status code mappings

The API Gateway can be configured with different hosting options: **REGIONAL** (public-facing, hosted in the specified region), **EDGE** (public-facing, hosted via a Cloudfront distribution (CDN)), or **PRIVATE** (hosted inside a VPC and only reachable either from the VPC, or from a peered-VPC). In our proposal, we recommend using a **PRIVATE** API Gateway. We propose hosting your Proxy Service API Gateway in the same VPC as the hosted JupyterHub environment or hosting it in a peered VPC. This ensures the API Gateway cannot be communicated to directly from the public internet, providing an additional layer of network security.

In addition, we propose using an authenticator in the API Gateway, which provides an additional auth check (beyond the Jupyter authenticator), before exposing users to backend resources. This API Gateway authenticator will allow you to use the user's identity information to make advanced decisions, e.g., you can enforce usage limits, enable auditing, provide per-user storage isolation, etc.

We model our Proxy Service APIs using OpenAPI [6]. API Gateway can generate an API Gateway endpoint using this model definition. While this a useful feature, the main benefit we see from using this standard definition is you can then generate clients using the open-source client generator package from the model you created. This lets you generate clients in your supported Jupyter Kernel programming languages, making it significantly easier to maintain multiple kernels, e.g., Python and R.

2.6 Serverless Computing

Serverless computing refers to the idea that the cloud provider would allocate machine resources on-demand, taking care of the servers on behalf of their customers. Serverless computing helps us get rid of server management, and the "pay-as-you-go" plan makes it cost-efficient for our erratic usage pattern. More importantly, serverless architectures are inherently scalable and elastic. In the reference design, we mainly exploit two serverless services from AWS: Lambda and Athena.

Lambda is an AWS serverless compute service to run code without provisioning an EC2 instance. We use Lambda extensively for our Proxy Service as AWS provides the native Lambda integration with API Gateway. Although Lambda has several limitations (e.g., execution time, code size), we find that all our use cases for the Proxy Service can fit nicely into this model.

Athena is a serverless interactive query service from Amazon. It is powered by Presto and works with a variety of data formats, including CSV, JSON, Apache Parquet, etc. Athena is easy to set up and use. Users only need to write standard SQL statements to query data from multiple sources once configured.

2.7 Auditability

Our proposal leverages the native logging and auditability support of the infrastructure components (e.g., S3 Access Logs and CloudTrail Events) to provide comprehensive auditability of user-level actions. For instance, user action related AWS API calls use user-specific IAM sessions. Same as all user executed Athena queries. Container Insights provides per-user pod metrics to track metrics that may be useful to audit, such as bytes sent and received. Besides, API Gateway provides per API call request and response logging via CloudWatch, as well as tracing support (X-Ray), allowing to record every request to the proxy service. As these calls are dependent on an authorizer, all API Gateway logs are attributable at user-level. These examples are not exhaustive, but demonstrate that by using a proxy service to facilitate access to data and leveraging the logging frameworks of the supporting services, we can provide comprehensive audit capabilities of all user actions.

3 SOFTWARE CUSTOMIZATION

While the previous section covers the architecture, this section will provide details on the software components of the proposed architecture. It includes the following two parts: (1) How to customize Jupyter Notebook; (2) How to customize the Docker Images.

3.1 Jupyter Notebook

The Jupyter Notebook comes with the default theme. But you can easily make it customizable by running "pip install jupyterthemes". This is the easiest way to enhance the coding environment and the presentation. Another way is to install Notebook extensions, which are modules that modify the user experience and interface.

For instance, Jupyter Notebook has built-in autocompletion by pressing the tab button. But there are extensions (e.g., Hinterland) that enable code autocompletion for every keystroke, which makes the experience very similar to an IDE. Besides these two methods, a more advanced approach involves front-end programming work by directly customizing the Jupyter Notebook source code and making your own distribution. This provides you the most flexibility in customizing the experience, but requires more upkeep.

3.2 Docker Images

In our system, we use a proprietary docker image repository provided by AWS for Kubernetes to pull images for the Jupyter notebook pods, so that we can customize the image based on our needs. In order to customize the docker image, a docker file will be used, which you can customize by adding instructions. You can start by importing the standard images so that you will already have a set of commonly used tools and then customize the software installation. For instance, if you need to install extra Python packages, you can add a line such as "RUN pip install nltk" in the docker file. Often, you want to install proprietary software developed by your own institute to perform analysis. You just need to pack them (e.g., build a python wheel) and install them by customizing the docker file. Once you update the docker file, you can compile it into a docker image and then push it to your image repository. With proper image tagging and the setting of corresponding helm config for JupyterHub, the new image will be able to be pulled by the pod automatically. This will allow you to customize as much as you can to meet all your needs.

4 CASE STUDY: FACEBOOK OPEN RESEARCH AND TRANSPARENCY

In 2018, Facebook began an initiative to support independent academic research on social media's role in elections and democracy. As part of the effort, we designed and built the *Facebook Open Research and Transparency (FORT)* platform to facilitate responsible research by providing flexible access to high quality and privacy protective data. Let's take two released datasets as examples.

URL Sharing [3]: This dataset, protected by differential privacy, consists of links that had been shared publicly on Facebook by at least 100 unique Facebook users. It included information about share counts, ratings by Facebook's third-party fact-checkers, user reporting on spam, hate speech, and false news associated with those links. The dataset includes more than 38 million unique links with aggregated information to help academic researchers analyze how many people saw these links on Facebook and how they interacted with that content with reactions. We've also aggregated these shares by age, gender, country, and month.

Ads Targeting [5]: This dataset contains targeting information for more than 1.65 million social issues, electoral, and political Facebook ads. It includes ads that ran during the three-month period prior to Election Day in the United States, from August 3 to November 3, 2020. We exclude ads with fewer than 100 impressions, which is one of several steps we have taken to protect users' privacy.

To help researchers analyze differentially private datasets, we install a proprietary software package, **S**tatistical **V**alid **Infer**ence (**SVInfer**), on the FORT platform. In general, a privacy protective dataset contains designed noise ingested to protect the individuals' privacy. If the researchers use the off-the-shelf statistical packages to analyze the dataset, they may reach biased conclusions caused by the noise, which makes the released dataset less useful. As a result, such analysis toolkits should be provided along with the privacy protective data releases. We initially implement *Linear Regression* in SVInfer based on [7] and expand to *Logistic Regression* and *Summary Statistics*. Our open-sourced [4] implementation includes both an in-memory and a scalable SQL version.

5 CONCLUSION AND FUTURE WORK

This paper proposes a scalable architecture for computational social science research and explores different design spaces. The case study shows that our architecture can support hundreds of users and execute over 1000 daily queries to analyze terabyte to petabyte-scale datasets released by Facebook.

In the future, we plan to extend our architecture to support additional technical and policy needs. Technically, we plan to support hybrid clouds (e.g., public-private) with GPU compute capability for machine learning workloads. We also want to introduce mechanisms to secure the system and offer fine-grained permission controls to facilitate collaboration among users. From a policy perspective, we plan to develop public standards for data-sharing infrastructure and the reproducibility of research.

ACKNOWLEDGMENTS

We would like to thank Andrew Gruen, Bennett Hillenbrand, and Chaya Nayak for reading earlier versions of this paper and contributing to many improvements. We thank the many Facebook engineers who built, reviewed, and scaled the original implementation of the architecture and software for providing us with the design insights.

REFERENCES

[1] 2017. *Zero to JupyterHub with Kubernetes.* Retrieved March 29, 2021 from https://zero-to-jupyterhub.readthedocs.io/en/stable/
[2] 2020. *NCI Cloud Resources.* Retrieved April 7, 2021 from https://datascience.cancer.gov/data-commons/cloud-resources
[3] 2020. *New privacy-protected Facebook data for independent research on social media's impact on democracy.* Retrieved March 30, 2021 from https://research.fb.com/blog/2020/02/new-privacy-protected-facebook-data-for-independent-research-on-social-medias-impact-on-democracy/
[4] 2020. *SVInfer.* Retrieved March 30, 2021 from https://github.com/facebookresearch/svinfer
[5] 2021. *Introducing new election-related ad data sets for researchers.* Retrieved March 30, 2021 from https://research.fb.com/blog/2021/02/introducing-new-election-related-ad-data-sets-for-researchers/
[6] 2021. *OpenAPI Initiative.* Retrieved March 30, 2021 from https://www.openapis.org/about
[7] Georgina Evans and Gary King. 2021. Statistically Valid Inferences from Differentially Private Data Releases, with Application to the Facebook URLs Dataset. *Political Analysis* (2021).
[8] Jupyter Kernels 2015. *Wiki: Jupyter Kernels.* Retrieved March 30, 2021 from https://github.com/jupyter/jupyter/wiki/Jupyter-kernels
[9] JupyterHub 2021. *JupyterHub: A multi-user version of the notebook designed for companies, classrooms and research labs.* Retrieved March 29, 2021 from https://jupyter.org/hub
[10] Thomas Kluyver, Benjamin Ragan-Kelley, Fernando Pérez, Brian E Granger, Matthias Bussonnier, Jonathan Frederic, Kyle Kelley, Jessica B Hamrick, Jason Grout, Sylvain Corlay, et al. 2016. *Jupyter Notebooks - a publishing format for reproducible computational workflows.* Vol. 2016.