Literate Computing for Reproducible Infrastructure による研究・教育環境の構築と運用

長久 勝[1)], 政谷 好伸[1)], 谷沢 智史[2)], 中川 晋吾[3)] 合田 憲人[1)]

1) 国立情報学研究所 クラウド基盤研究開発センター
2) 株式会社ボイスリサーチ
3) 有限会社カラビナシステムズ

mnagaku@nii.ac.jp

Construction and operation of research and educational environment by “Literate Computing for Reproducible Infrastructure”

Masaru Nagaku[1)], Yoshinobu Masatani[1)], Satoshi Yazawa[2)], Shingo Nakagawa[3)], Kento Aida[1)]

1) Center for Cloud Research and Development, National Institute of Informatics
2) Voice Research Inc.
3) Carabiner Systems, Inc.

概要

「Literate Computing for Reproducible Infrastructure（以下、LC4RI）」の方法論に基づき、「Jupyter Notebook」を使った、システムの構築と運用を実践している。実践の中で、システム構築や運用に関わるドキュメントの不備や自動化の弊害など、いくつかの既知の問題に対処できることが分かった。また、実践のために「Jupyter Notebook」の機能拡張も行っている。本稿では、LC4RI による既知の問題の解決方法、および、システムの構築と運用に LC4RI を活用するために実施した機能拡張について報告する。筆者らが機能拡張を行った「Jupyter Notebook」は公開されており、誰でも自由に利用が可能である。

1 はじめに

DevOps の概念やプラクティスの浸透により、web アプリケーションを中心として、開発成果物を簡単に繰り返し何度でもデプロイすることが一般的に可能となった。しかし、そのアプリケーションを載せる基盤の構築や運用においては、簡単に繰り返し何度でも再構築することは、未だに一般的に実現されているとは言い難い。

また、システムの構築や運用を、最終的に管理者、即ち人間が責任を持つワークフローであると考えた時、属人性や忘却が問題となる。この問題を、自動化によって解決しようとするアプローチもあるが、効果は限定的である。

筆者らは、これらの問題に対処するための方法論として「Literate Computing for Reproducible Infrastructure（以下、LC4RI）」を提案しており、所内に構築した研究・教育向けクラウド基盤の構築や運用において、実践を行っている[1][2]。本稿では、筆者らの実践を通して得られた LC4RI による既知の問題の解決方法、および、システムの構築と運用に LC4RI を活用するために実施した機能拡張について報告する。

2 構築と運用における既知の問題

2.1 構築における既知の問題

システムの構築において、よくある問題として、ドキュメントの不備を挙げることができる。例えば、ベンダーに外注され納品されたシステムには、日々の運用やトラブル発生時の対処のため、併せてシステムの構築方法や操作方法に関するドキュメントが納品されるのが一般的である。このドキュメントにおいて、しばしば、実行するコマンドの記載が漏れている、コマンドに指定する引数が誤っている、などの不具合が混入していることがある。

これは詳しく見ると、ドキュメントを作成した構築作業担当者にとって実行することが当たり前すぎるために書き漏らした、構築作業を試験的に実施した際に作成されたドキュメントの雛型に対して実環境の情報を上書きして最終版とする過程で上書き作業に漏れがあった、スケジュールの都

合から構築後に急いでドキュメントを作成したために誤りを混入してしまった、などの作業ミスによるものである。

また、納品時の検収において、日々の運用についてのドキュメントは、実際に試して不具合を見つけることができるが、トラブル発生時の対処（例えば、障害時の系切り替え手順や、バックアップからの復元手順）については、実際に試すことができないものもあり、不具合を見つけられない場合も多い。

研究・教育環境に特有の問題としては、さまざまな環境を、それぞれ特定の期間だけ使いたいというものがある。それぞれを別々のシステムとして構築し維持し続けることはコストパフォーマンスが悪いため、クラウド基盤上に、それぞれを必要な期間だけ構築し維持したいが、受講生の増減や、教材のアップデートなど、毎回の構築で変化する部分も多く、簡単ではない。

2.2 運用における既知の問題

システムの運用において、人為的なミスを防ぐ目的で、自動化を採用することが、近年、増えている。

しかし、現代のシステムは、互いに依存する複数のミドルウェアから構成されている場合が多く、個々の構成要素のセキュリティパッチや機能修正に伴って、当初に設定された自動化機構が、運用予定期間の途中で動かなくなる可能性がある。

また、自動化された部分について、運用している人間が機械に依存してしまい、関心が薄れることが多い。この状態になってしまうと、自動化機構が動かなくなった場合に、人間がリカバリを行うことが難しくなってしまう[3]。

自動化機構は、ツールと、そこで実行されるスクリプトで実現されることが多いが、システム固有の事情や、システムの改変の経緯などの情報を、同じドキュメントに書いておくことが難しい。スクリプトのコメント文として書くには量が多く、Wiki など分離したドキュメントにしてしまうと管理が煩雑になる。

3 LC4RI の提案

本稿で使う意味での「Literate Computing」は、「IPython」の開発者である Fernando Perez が、2013 年以降、ブログ[4]や論文[5]で言及しているものを指す。これは、ドキュメントとコードを同じテキストで一元管理する「Literate Programming」[6]の考え方を、IPython のブラウザベースの Notebook スタイルインタフェースである「Jupyter Notebook」（図１）[7]でコンピュータの操作に適用しようというものである。元々は、オープンサイエンスの文脈から提唱されている。

筆者らは、この「Literate Computing」を、システムの構築や運用に用いる実践として、Notebook を、そのまま実行可能な手順書として使っている。「for Reproducible Infrastructure」とは、構築手順書として正しく動く Notebook があれば、簡単に繰り返し何度でも同じシステムを再構築できるということである。また、ドキュメントとコードだけではなく、実際の動作結果も一体として管理・比較参照可能とすることで、システム固有の事情や、システムの改変の経緯などの情報を留め、運用者間で個々の環境や作業における関心を共有可能とすることで、人間中心の柔軟な自動化を実現するということでもある。

1 枚の Notebook は、それぞれ複数のドキュメント部分とコード部分から構成される。選択状態のコード部分を実行するボタンがあり、そのまま Notebook 上で処理を実行できる。実行結果はコード部分の直下に表示される。

図１：Jupyter Notebook

3.1 構築における既知の問題への対応

LC4RI の対象システムは、構築物の納品よりも、構築や運用のためのドキュメントの納品の方が重要となる点で、これまでの一般的なシステム構築とは異なる。

構築手順書である Notebook を実行して、正しくシステムが構築できなければ、意味がないため、検収する対象は構築物ではなく構築手順書となり、構築手順書によってシステムを正しく構築できることを確認することによって検収となる。

簡単に繰り返し何度でもシステムを再構築で

きるため、構築され試験動作中のシステム上で障害を発生させる試験も、容易に行える。障害対応手順を記した Notebook が正しく動かなかった場合も、原因を調べ、Notebook を修正し、再構築した環境で再試験するなど、整理された状態で反復作業が行える。

さまざまな環境を、特定の期間だけ、毎回少しずつ異なる状態で提供したい場合、それぞれの環境構築のための Notebook を用意することで、対応が可能である。年に数回しか行わない作業の場合、以前対応した人間でも作業内容を忘れていることが多いが、ドキュメント部分に記載された技術情報や併せて保存された実行履歴を手掛かりに、以前の対応を思い出して、作業にあたることができる。

3.2 運用における既知の問題への対応

「Jupyter Notebook」には、コード部分を順番にすべて実行する仕組みがあり、外部のスケジューラと連携することで、自動化に対応することが可能である。

何らかの要因で Notebook 全体の処理が失敗した時も、Notebook のドキュメント部分に記載された技術情報や併せて保存された実行履歴を手掛かりに原因を調べることができるため、人間が行うリカバリ作業が支援される。

システム固有の事情や、システム改変の経緯なども、Notebook のドキュメント部分に記入し、コード部分と合わせてメンテナンスが可能であり、関連する知識を集中的に管理できる。

システム運用における自動化の採用では、自動化できるかどうかを判断基準にした事例が多く、効果を考えて優先順位は付けるものの、基本的にすべて自動化したいというモチベーションで行われることが多い。Notebook ベースの場合は、自動化に対応できるだけでなく、スキルトランスファーのために一時的に人間が Notebook を読みながら作業するなどの対応も容易であり、人間中心の柔軟な自動化を実現できる。

4 LC4RI のための拡張

「Jupyter Notebook」は、もともと、対話的に計算機実験やデータ分析をするためのツールとして広まったもので、構築や運用のためのツールとして作られたものではないため、LC4RI のユースケースから見た場合、不便を感じる部分がある。筆者らは、それを補うために、複数のプラグインを開発し、「Jupyter Notebook」を拡張して用いている。それらのうちの 5 つを紹介する。

4.1 実行状況に応じた着色

Notebook のコード部分を自動で順次実行する場合など、どこまで処理が進んで、仮に失敗した場合もどこで止まったのか、素早く把握したいニーズがある。

これに応えるため、実行状況に合わせて、各コード部分を動的に着色することで、視認性を向上させた。着色がないと、Notebook を上から順に確認しないと状況が分からないが、この機能により、一目で把握できる。

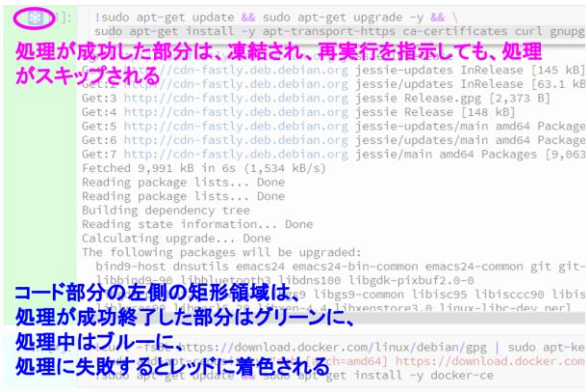

図２：実行状況に応じた着色と、実行後の凍結

4.2 実行後の凍結

構築や運用の作業においては、同じ操作を繰り返し行ってはならないもの、冪等性が保証されない操作が、多数存在する。例えば、設定ファイルに特定の文字列を 1 行追加する操作があった場合、誤って複数回、同じ操作を行うと、システムが設定内容を正しく解釈せず、起動に失敗することや、期待していない挙動で振る舞うことがある。

この問題に対処するために、正常に実行が終わったコード部分を「凍結」状態にし、その状態を解除しない限り、再実行を指示しても、実行がスキップされる機能を追加した。システムの再構築を担保できていて、元に戻すことが可能であっても、誤った操作でシステムが停止することを防げるわけではない。この機能を追加することで、操作のミスを低減させることを目指している。

4.3 実行結果の保存機能

Notebook には、ドキュメント部分とコード部分に加え、コード部分の実行結果を表示する部分がある。構築や運用の作業においては、コード部分

の実行結果がこうなるべき、そうでなければ問題が起きていると判断するべき、という場面が多くあるため、「コード部分の実行結果がこうなるべき」という情報は重要である。しかし、従来の Notebook では、その情報をドキュメント部に記載するしか方法がなかった。

そこで、実行結果を簡単に保存する機能を付与し、以前の実行結果と、今の実行結果の比較を支援するようにした。以前の実行結果を、一種のお手本として参照できるようになったことで、作業を進める上で、作業者に安心感が与えられる。また、専用のインタフェースにより、作業者が実行結果の比較を行うよう促すことで、作業中の見落としを防ぐ効果も期待できる。

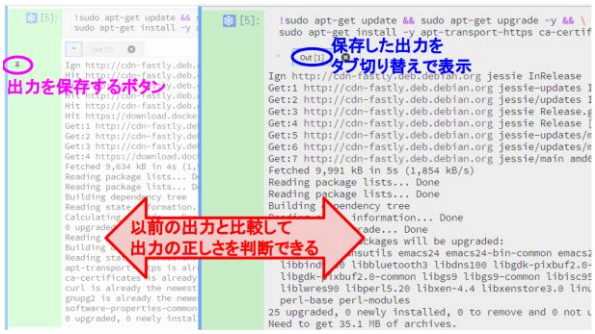

図４：以前の実行結果と比較

4.4 まとめ実行機能

多くの記述を含む Notebook は縦長になりがちで、視認性が悪くなり、全体を把握するのが難しくなる。この問題に対処するために、見出しによって章立てた単位で記載内容を畳み込み表示するプラグインとして「Collapsible Headings」がある。しかし、畳み込まれた場所に含まれるコード部分を実行するには、畳み込み表示を解除する必要があった。

そこで、畳み込み表示の際に、畳み込まれた場所に含まれるコード部分の状態をアイコンで表示し、更に、畳み込まれた場所に含まれるコード部分のすべてをまとめて順次実行できる機能を追加した。これにより、コード部分の実行状態も含め、視認性が向上した。作業の際には、ドキュメントを詳しく追いながら行いたい場合は、畳み込み表示を解除し、定型的なセクションは畳み込み表示の状態でまとめ実行するなど、柔軟に作業方法を選択することができる。

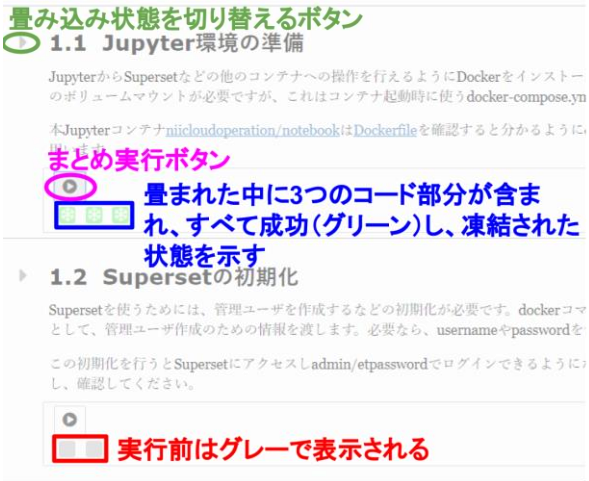

図５：まとめ実行機能

4.5 lineage の導入

運用に関する Notebook は、運用過程の中で、改変されたり、派生したり、増えていくことになる。運用の改善や、環境の変化などは、Notebook の改変や派生の履歴の中に、その痕跡を残すこととなる。筆者らは、Notebook の改変や派生の履歴を分析することで、運用に関するケーススタディが行えるのではないかと考えた。

そこで、Notebook と、その構成要素であるドキュメント部分やコード部分に、一意な ID「lineage」を付与して、Notebook の複製時に引き継がれる機能を追加した。これにより、lineage 群の類似度から、Notebook の系譜を特定できるようになった。

lineage の系統図を作成してみたところ、同じ系統においても、特定の時期に大きく変化が見られ、時代区分を確認することが可能となった。Notebook の系統と、変化が起きた時期を運用チームで確認すると、運用を見直した（つまり、Notebook も見直されている）時期など、実際の運用現場の変化と関連が確認できた。

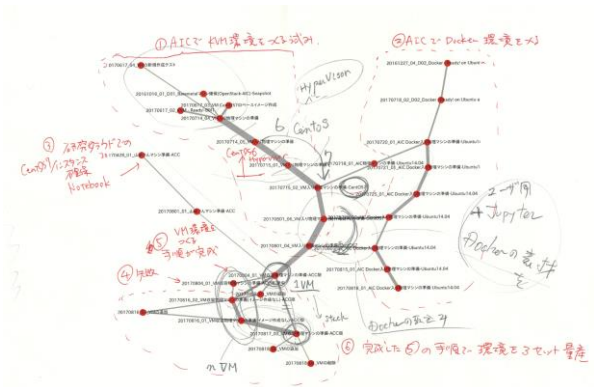

図６：lineage の系統図と運用チームの記憶

5 成果物の公開

LC4RI の考え方に基づき、筆者らが実務に使っている拡張された「Jupyter Notebook」は、それを構成するプラグイン群のソースコードを github で公開している。

NII クラウド運用チームの github リポジトリ
https://github.com/NII-cloud-operation

また、筆者らが実務で使用する際には、Docker 環境に、拡張された「Jupyter Notebook」をデプロイして使っている。このコンテナイメージも Docker Hub で公開している。

NII クラウド運用チームの
「Jupyter Notebook」コンテナイメージ
https://hub.docker.com/r/niicloudoperation/notebook/

この上で、アプリケーション環境を構築し運用するための Notebook 群も公開している。

Hadoop
https://github.com/NII-cloud-operation/Literate-computing-Hadoop

Elasticsearch
https://github.com/NII-cloud-operation/Literate-computing-Elasticsearch

6 さいごに

2014 年度に、新規システムの運用ドキュメントを Notebook で納品してもらうところから取り組みを始め、現在は、独自の構築・運用環境を整えるところまで進んだ。

実践の中で、効果を実感してはいるが、今後、客観的な効果測定にも取り組みたい。

また、成果を積極的に展開し、運用の現場の助けになれればと考えている。例えば、Notebook の書き方のパターンによるガイドなど、ノウハウの形式知化を進める取り組みが必要だと考えている。

参考文献

[1] 政谷 他、インフラ・コード化の実践における IPython notebook の適用、信学技報、115 巻、72 号、27-32、2015 年

[2] Y.Masatani 、 Collaboration and automated operation as literate computing for reproducible infrastructure、JupyterCon、2017 年

[3] 稲垣敏之、人と機械の共生のデザイン:「人間中心の自動化」を探る、森北出版、2012 年.

[4] F.Perez、"Literate computing" and computational reproducibility: IPython in the age of data-driven journalism 、 http://blog.fperez.org/2013/04/literate-computing-and-computational.html、2013 年.

[5] F.Perez 他、IPython: components for interactive and parallel computing across disciplines.、AGU Fall Meeting Abstracts.、2013 年.

[6] D.Knuth、Literate programming.、The Computer Journal、27 巻、2 号、97-111、1984 年.

[7] https://jupyter.org/