

SSH Kernel: A Jupyter Extension Specifically for Remote Infrastructure Administration

UENO, Masaru
FUJITSU LABORATORIES LTD.
Kanagawa, Japan
ueno.masaru@fujitsu.com

IMAI, Yuji
FUJITSU LABORATORIES LTD.
Kanagawa, Japan
yuji.imai@fujitsu.com

Abstract—In the context of IT infrastructure operation, a methodology has been proposed that describe both operation instructions and commands in *executable runbooks* using Jupyter Notebook to achieve highly reproducible operation. However, to apply this methodology to remote infrastructure operation, operators are forced to rewrite too many commands in typical operation manuals. To address this issue, we developed a Jupyter extension to both solve this problem and improve the usability of executable runbooks. We then confirmed the effectiveness of this extension in a user study.

Index Terms—Service Management, Collaboration, Automation

I. INTRODUCTION

IT infrastructure operations can be categorized into routine operations and non-routine operations. Routine operations are generally automated with automation scripts. On the other hand, non-routine operations, such as troubleshooting and rarely performed operations, consist of step-by-step manual procedures. In other words, operators perform non-routine operations by repeating the following steps: (1) input and execute a command, (2) check output and system condition, and (3) input and execute the next command based on the condition.

This paper aims to provide a means that operators can perform non-routine operations effectively without depending on operators' skills or knowledge. To achieve this aim, some methodologies are considered necessary to record and share step-by-step manual procedures with ease and precision. For instance, recently a methodology that describes both the operation instructions and commands of step-by-step operations in an *executable runbook* using Jupyter Notebook [1] has been proposed. This methodology is called *Literate Computing for Reproducible Infrastructure* [2] [3] or the *GitLab Executable Runbook* [4]. Using this methodology, operators can execute commands and the outputs with ease and precision, as well as write any instructions related to the operation.

However, if network fabrics and servers, which are IT infrastructure components, are physically located far from the operators, it's hard to apply the methodology of executable runbooks to step-by-step manual procedures. This is because Jupyter Notebook is meant to establish and then destroy an SSH connection every time one executes a command remotely;

therefore, the last execution context is lost, and operators are forced to revert to the last context at the beginning of the next command every time. This makes executable runbooks less maintainable, i.e., writing and understanding runbook take more cost.

To address this issue, we developed a Jupyter extension specifically for remote command execution, which has the primary function of maintaining the execution context through a series of remote command executions.

We believe our work enables operators to utilize executable runbooks more efficiently and comfortably, thereby making knowledge sharing between operators smooth and achieving a reproducible infrastructure operation. Executable runbooks can also be used for retrospectives after troubleshooting, as well as for operational audits.

The remainder of the paper is organized as follows. Section 2 introduces executable runbooks, Section 3 discusses the challenges of applying executable runbooks in remote infrastructure operations, Section 4 describes the design and implementation of SSH Kernel, and Section 5 evaluates its performance.

II. EXECUTABLE RUNBOOKS

This section summarizes executable runbooks and its architecture.

A. Summary of Executable Runbook

Typical IT infrastructure operation are performed based on operation manuals, which provides commands to execute and instructions on doing so, methods to verify that the operation succeeded, etc. Since operation manuals address a wide range of contexts and knowledge about IT infrastructure, operators are taught to operate systems correctly. In this paper, we call these manuals *typical manuals*. Since typical manuals are intended to be read by operators, they must execute commands manually, which means they are often time-consuming and subject to mistakes.

On the other hand, the use of automation scripts has been increasing via tools such as Ansible, Chef, and others. Automation scripts can be executed automatically, so they reduce time required to execute commands as well as reducing the risk of human mistakes. However, as communication media between operators, since scripts are described only by

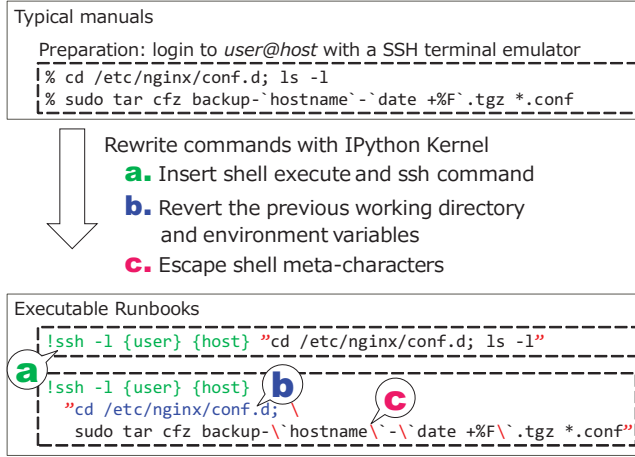


Fig. 5. An example to illustrate how to rewrite typical manuals into executable runbooks.

Also, in large systems, the costs of installing and maintaining kernel software on each server is unreasonable.

B. Problems in executing commands remotely with SSH

In option B above, since the target OS doesn't run Jupyter Notebook, operators must use two steps to create executable runbooks, which include remote commands: (1) input `! ssh user@host`, where `!` is a IPython Kernel function that executes shell command. (2) Input a remote command as a parameter of the `ssh` command. Fig. 5 shows a comparison of remote command notations for typical manuals and executable runbooks. Commands written in executable runbooks are more complicated, thereby less understandable and harder to write than typical manuals. The reasons for this are as follows:

- Since an SSH connection is established every time a remote command is executed, operators are forced to revert to the last environment variables and working directory every time.
- Operators must carefully escape shell meta-characters (e.g., newlines, quotes, \$) included in commands to execute remotely to prevent those characters from being interpreted by IPython Kernel.

IV. DESIGN AND IMPLEMENTATION OF SSH KERNEL

To resolve the issues described above, we defined the following requirements for executable runbooks:

- No need to install any software to the target OS
- No need to alter commands written in typical manuals
- No need to insert commands to restore the previous execution context every time

To satisfy these requirements, we developed a new kernel that interprets and runs the code of executable runbooks as remote commands while maintaining the context between codes.

Fig. 6 shows the architecture of the proposal method. Since this architecture follows the style of option B described in

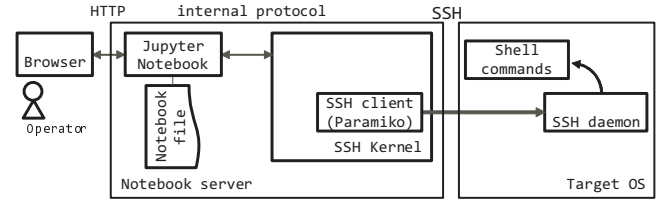


Fig. 6. Architecture overview

the previous section, operators needn't install a kernel into the target OS.

A. Overview of SSH Kernel

In this sections, we provide an overview of the SSH Kernel.

Fig. 7 shows an example of executable runbooks using SSH Kernel and Jupyter Notebook.

- (1) `%login` and `%logout` are special commands in the SSH Kernel that establish/destroy a connection to the target OS. Login options (e.g., hostname, IP address, login name, secret key) are read from the common configuration file of OpenSSH (i.e. `/.ssh/config`).
- (2) Commands written in code areas are executed as shell commands in the target OS. Since the commands are passed to the remote shell 'as is', there is no need to escape shell meta-characters. The working directory and environment variables in the previous remote execution remain in the next execution.
- (3) When a command is executed remotely, the SSH Kernel automatically logs not only the command outputs but also remote execution information.

In addition, SSH Kernel has a command-line completion feature activated by Tab key to improve usability and reduce mistyping shell commands.

B. Design and Implementation of SSH Kernel

This section describes some decision-related issues in the design and implementation of SSH Kernel.

1) Executing commands remotely in Jupyter Notebook:

To enable operators to log-in/log-out of the target OS using executable runbooks, we introduced two special functions `%login`, `%logout`. These functions enable operators login to the target OS and step-by-step remote command executions in a way similar to that of typical operations.

SSH Kernel is designed to establish a new connection at `%login` and to keep the connection until `%logout`. As such, the total response time required to execute remote commands is much improved from that of the IPython Kernel, with its need to establish/destroy a connection every time.

2) *Choice of libraries:* To improve portability, we chose Paramiko¹ as the SSH client library, which is a Python implementation of the SSHv2 protocol, instead of depending on OpenSSH²/libssh³. Also, to reuse exception handling and user interruption logic during remote command execution, we

¹<http://www.paramiko.org/>

²<https://www.openssh.com/>

³<https://www.libssh.org/>

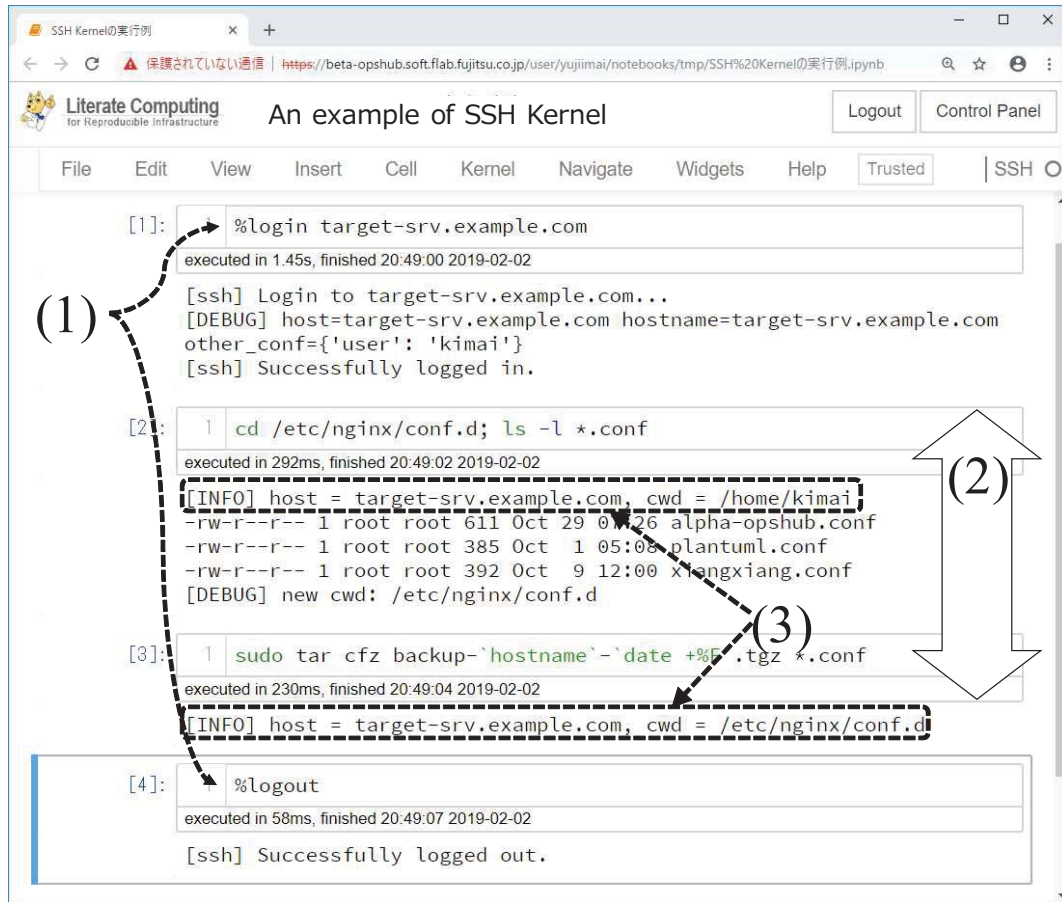


Fig. 7. An Example of executable runbooks using SSH Kernel and Jupyter Notebook

chose Plumbum⁴, which provides remote command execution functionality based on Paramiko.

To improve maintainability, we use Metakernel⁵ as a base kernel, which provides the typical functionality every kernel should have.

3) *Shared execution context between command executions:* We have two design options to share the execution context, i.e. the working directory and environment variables, through a series of remote command executions:

a) Keeping a shared context in a remote process. By this we mean, running a long running shell in the target OS to keep context, and performing a command execution by stdout/stderr string processing.

b) Managing execution context in local process. By this we mean, saving the remote context after a remote command execution and restoring it before the next remote command execution each time.

At a glance, (a) above looks simpler to implement than (b). However, option (a) incurs the risk of string mis-matching and hang ups at worst, because the end of a remote command execution is detected by string pattern matching. On the other

hand, although it is more complicated to implement, option (b) doesn't incur risk of string mismatching, because the end of command execution can be detected automatically when the SSH channel is closed. Thus, we chose option (b) to enable handling garbled characters in any command output.

To save the remote context, we implemented the following:

- Append a command to print the current environment variables to the command user input, then execute both remotely
- Capture and store the environment variables from the command output with pattern matching. The working directory is included in the environment variables in POSIX shells

To restore the remote context, we implemented the following:

- Run a command to change the last working directory
- Run a command to declare the last environment variables
- Run the command user input

A entire SSH Kernel implementation contains about 700 lines of Python code, not including tests. The source code is freely available. We have confirmed that the SSH Kernel works in the following environments:

- Python: Python 3.6, 3.7

⁴<https://plumbum.readthedocs.io/>

⁵<https://github.com/Calysto/metakernel>

TABLE I
THE NUMBER OF RESPONDENTS GROUPED BY JOB CATEGORY IN THE SURVEY

Job category	# of respondents
Systems engineer	2
Software engineer	1
R&D	4
Total	7

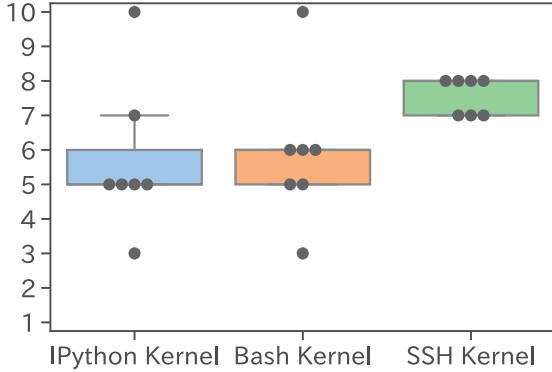


Fig. 8. A comparison of the relative usability scores between Jupyter kernels. We consider the score of SSH terminal emulators is 10.

- IPython 7.1
- Host OS: Ubuntu 16.04, 18.04, Windows 10 (Anaconda)
- Target OS: Ubuntu 16.04, 18.04, CentOS 6

V. EVALUATION

To evaluate the effectiveness of SSH Kernel, we conducted a user survey.

Table I shows the number of respondents grouped by job category. Even though the respondents' job categories and trial periods (from 2 weeks to 2 months) are different, we found that there are no significant differences between the answers.

A. Ease of writing and executing remote commands

The first question is to determine whether remote commands in the SSH Kernel and each existing kernel are easy or complicated to write and execute.

Fig. 8 shows the results. We asked the relative score when we consider the score of SSH terminal emulators is 10. As we can find from the figure, the relative ease of existing kernels ranges around 5 to 6, and that of the SSH Kernel ranges around 7 to 8. This implies that the SSH Kernel is easier to write and execute commands compared to existing kernels. However, it is more complicated than SSH terminal emulators.

B. Usability as compared with SSH terminal emulators

The second question is to determine how convenient the SSH Kernel is compared to SSH terminal emulators. As summarized in Table II, a large percentage of users consider

TABLE II
QUESTION: HOW OFTEN DO YOU FEEL INCONVENIENCE TO THE SSH KERNEL COMPARED TO SSH TERMINAL EMULATORS?

Option	# of answers
1. Feel inconvenience frequently	1
2. Feel inconvenience sometimes	5
3. Feel inconvenience rarely	1

the SSH Kernel to be inconvenient sometimes for the reasons described below (as provided by respondents in free form on the questionnaire).

First, most users complained that it doesn't support interactive commands like `vi`, `more`, and `sudo`. Without the interactive commands, they must rewrite these commands in non-interactive form. Second, some users reported that system administrators need to change their system security policy to use SSH Kernel. That is since some shell commands require root privilege to execute and SSH Kernel doesn't support interactive `sudo` command, system administrators have to make changes to either permit root login or execute `sudo` without inputting a password. In the Ubuntu OS, both of the changes are forbidden by default. Third, some users would like to use familiar keyboard shortcuts available in SSH terminal emulators (e.g., Ctrl+P for fetching the previous command in Bash).

On the other hand, some participants gave us positive feedback. One of our participants answered that SSH Kernel works intuitively in writing/reading executable runbooks when performing remote operations. Another participant responded that it is quite useful because there is no need to escape any shell metacharacters (e.g., newlines, wildcards), which are needed in other Jupyter kernels.

VI. CONCLUSION

In this paper, we addressed issues related to applying the executable runbooks methodology in the administration of remote infrastructure, and to do so, developed a new Jupyter extension, SSH Kernel. The results of our user study reveal that: (1) SSH Kernel improves usability in remote command execution compared to existing other kernels (2) there is room to improve the usability before replacing SSH terminal emulators with executable runbooks. The source code of SSH Kernel and the Python package is publicly available at <https://github.com/NII-cloud-operation/sshkernel>, <https://pypi.org/project/sshkernel>.

REFERENCES

- [1] T. Kluyver, B. Ragan-Kelley, F. Pérez, B. E. Granger, M. Bussonnier, J. Frederic, K. Kelley, J. B. Hamrick, J. Grout, S. Corlay *et al.*, "Jupyter notebooks-a publishing format for reproducible computational workflows." in *ELPUB*, 2016, pp. 87–90.
- [2] Y. Masatani, "Collaboration and automated operation as literate computing for reproducible infrastructure," in *JupyterCon*, 2017.
- [3] NII Cloud Operation Team. Literate computing for reproducible infrastructure. [Online]. Available: <https://literate-computing.github.io/>
- [4] GitLab. Runbooks. [Online]. Available: <https://docs.gitlab.com/ee/user/project/clusters/runbooks/>