

Internet Week 2016

国立情報学研究所におけるクラウド運用の  
Jupyter NotebookとAnsibleによる  
機械化と  
その効能

国立情報学研究所 クラウド研究開発センター / 株式会社ボイスリサーチ  
谷沢 智史

# まずは自己紹介

谷沢智史

- 国立情報学研究所 / 株式会社ボイスリサーチ
- GitHub/Twitter: @yacchin1205

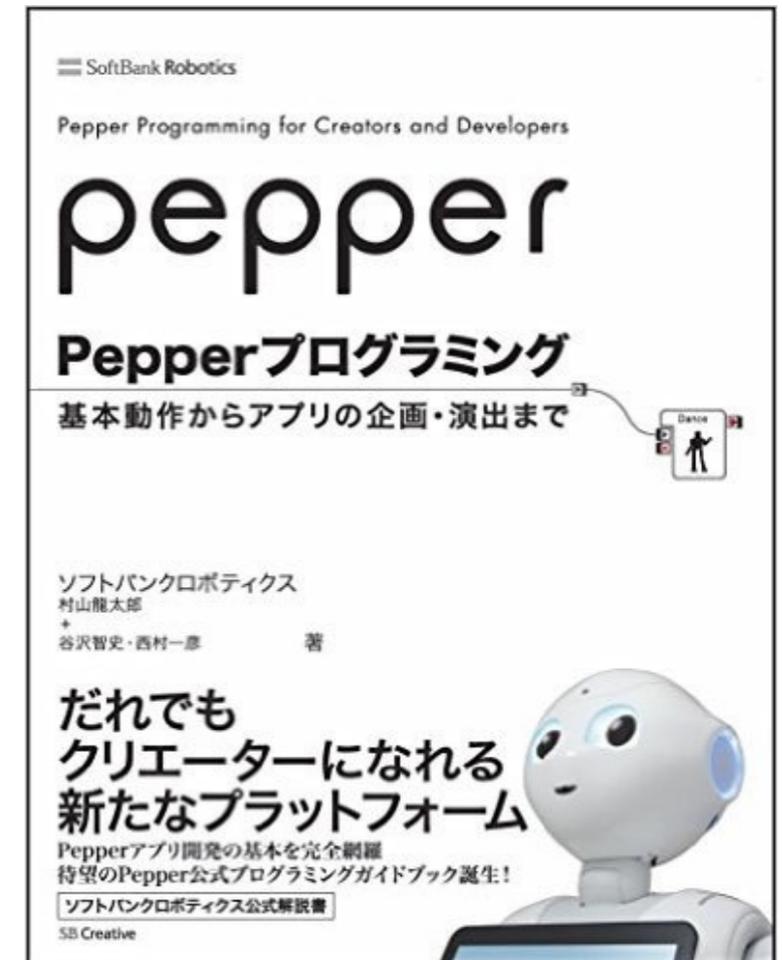
昔: オンラインゲームとか映像コンテンツとか

今: クラウド運用とかロボットとか

趣味: モニタリング(システムとか人体とか)

国立情報学研究所(NII)でプライベートクラウドの運用支援

東京大学で社会人博士とか (認知科学)



# 本日のアジェンダ

1. 国立情報学研究所におけるクラウド運用
2. クラウド運用の自動化→機械化
3. Literate Computing for Reproducible Infrastructureの紹介
4. Jupyter + Ansible環境のインストール・設定方法
5. Literate Computing for Reproducible Infrastructureの効能
6. Jupyter環境のカスタマイズ・拡張
7. まとめ

# 国立情報学研究所における クラウド運用

# 国立情報学研究所には・・・

## 所内研究者向けベアメタルクラウドサービス

- 研究クラウド / アカデミックインタークラウド

The image shows a screenshot of the OpenStack dashboard. On the left is a navigation sidebar with the OpenStack logo and the word 'openstack' in red. Below the logo is a 'DASHBOARD' button. The sidebar contains several menu items: 'プロジェクト 管理', '現在のプロジェクト cloud', 'Compute の管理' (with sub-items: '概要', 'インスタンス', 'イメージとスナップショット', 'アクセスとセキュリティ'), 'Manage Network' (with sub-item: 'ネットワーク'), and 'オブジェクトストア' (with sub-item: 'コンテナ').

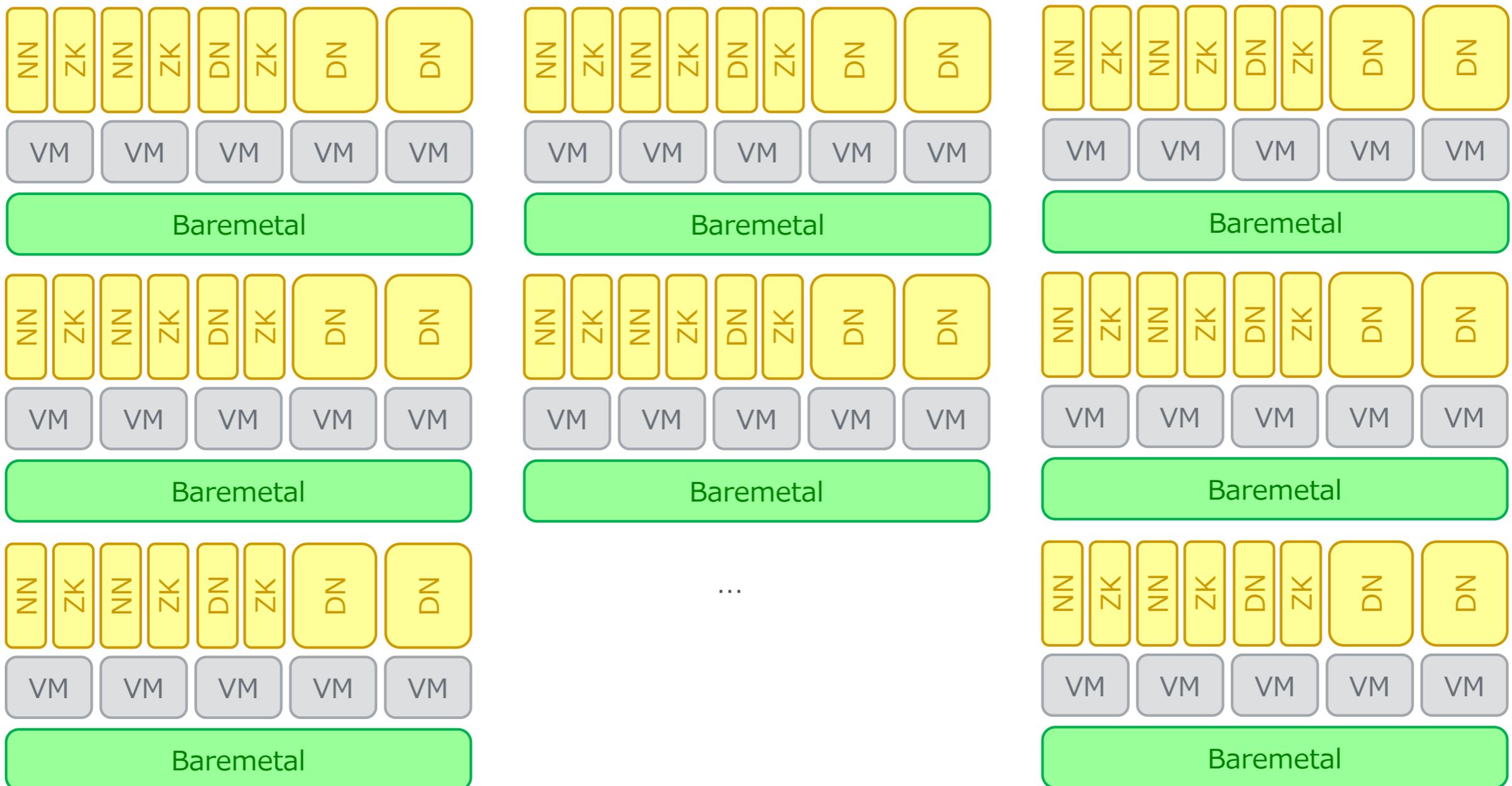
The main content area is titled '概要' (Summary) and shows a '次の役割でログイン中:' (Logged in with the following roles:) section. Below this, there are several horizontal bars representing different resources or services, each with a unique color-coded icon. The bars are arranged in a grid-like fashion, with some bars having multiple colored segments. The overall layout is clean and professional, typical of a cloud management interface.

# ベアメタルクラウドに至る経緯

- 仮想マシン貸し
  - edubase Cloud: Eucalyptusベース … (Hypervisor)150ノードくらい
- ベアメタル貸し
  - 研究クラウド: OpenStack (Diablo)ベース … (物理)70ノードくらい
  - アカデミックインタークラウド: OpenStack (Grizzly)ベース … (物理)70ノードくらい
  - 次期クラウド: OpenStack (Newton)ベース？
- OpenStackはすべてベアメタルクラウド
  - ベアメタルクラウド拡張 … Diablo, Grizzlyではdodaiと呼称
  - OpenFlowにより、ソフトウェアでベアメタルへの接続と利用者ネットワーク間の接続制御
  - NIIで実施していた勉強会の資料など
    - <http://www.slideshare.net/yacchin/openstackdodai-dodai-1>

# ベアメタル上でのサービス構築支援

- 主にHadoop族
  - 教材用途: VMベースにして自由に壊せるようにしたい / それを1週間だけほしいー
  - 分析用途: ログ集約にはFluentdがいいなあ / 認証つけて / etc...



# ベアメタルクラウド利用: BIGCHA

- <http://bigcha.net/>



BIGCHAとは?

2016開催概要

講義資料

お申込み



## BIGCHAとは?



幅広い分野のビッグデータを扱える!

Hadoopプログラミングの講義を受けて、新しい・面白いアプリを開発しよう!

# OpenStackとの闘い

- Diablo, Grizzly時代は特に・・・
- CLIツールやWeb UIの完成度は高くない
- APIがなければSQLを発行すればいいじゃない
- とにかくコードを読む
  - コード読解力(書いた人の気持ちを想像する能力含む)が必須

# OpenFlowとの闘い

- OpenFlowを積極的に採用・・・
- これまで(ネットワーク機器ベンダによる支配の時代)にないトラブル
  - 一部しか届かないパケット、分身するパケット、・・・
  - MACアドレスフィールドをちょっと間違える→広範囲、時間差で影響
- とにかくパケットキャプチャでパケットの行方を追う
  - 分量的に手作業は無理。コードを書きながら対応する能力が必須

# 効率的な運用に向けて・・・

- これらの死闘をこなせるスキルセットのエンジニア … 希少
- フォーカス: スキルトランスファーの難しさ
  - トラブルシューティング後、運用へのフィードバックが必要な部分
  - 要求される前提知識が多い … フルスタック、OpenStack  
コード、OpenFlow仕様
- **対象が多様 … すべてを自動化するのは難しい**
  - **エンジニアの闘いの記録をExplicitにする**
  - **過去の記録を「お手本」として運用を改善していく**

ところで

**自動化とは？**

# トヨタ生産方式

## 機械に人間の知恵を授ける

「自動化」ではない。ニンベンのついた「自働化」である。…「ニンベンのある自働化」の精神は、トヨタの社祖である豊田佐吉翁（1867～1930年）の自働織機の発明を源としている。佐吉翁の自働織機は、経糸が一本でも切れたり、横糸がなくなったりした場合、すぐに機械が止まる仕組みになっている。**すなわち、「機械に良し悪しの判断をさせる装置」をビルト・インしてあるのだ。**したがって、不良品が生産されることがない。

…この自動機にニンベンをつけることは、管理という意味も大きく変えるのである。すなわち**人は正常に機械が動いているときはいらずに、異常でストップしたときに初めてそこへ行けばよいからである。**

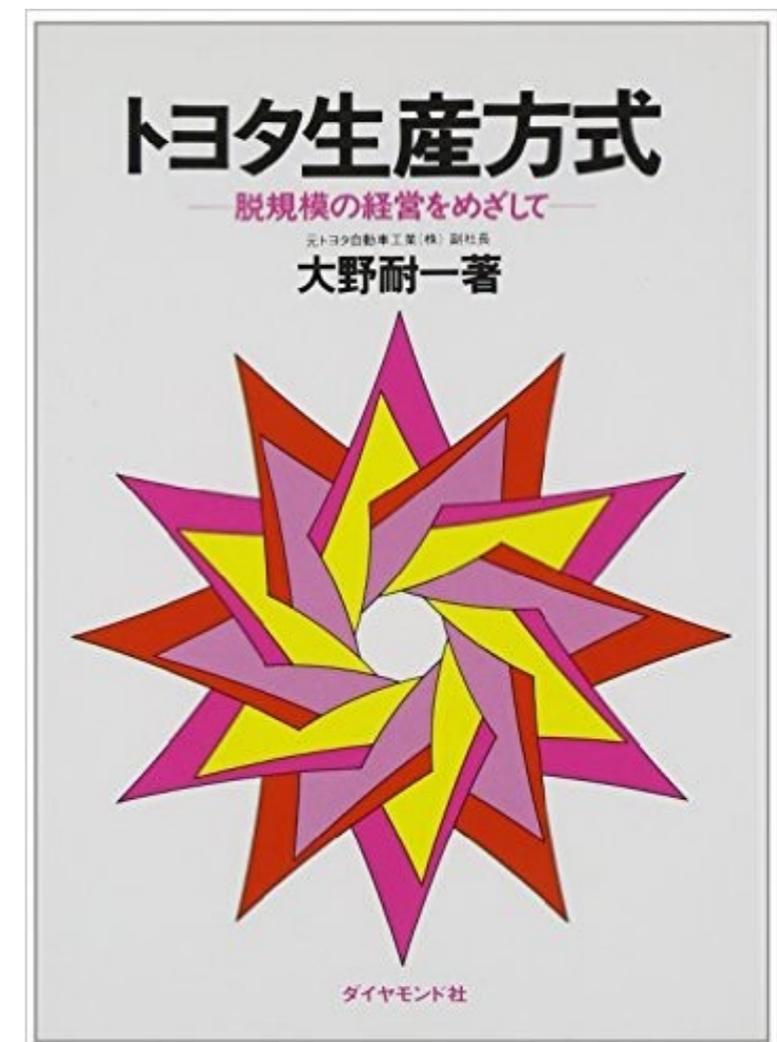
トヨタ生産方式 - 脱規模の経営をめざして - 元トヨタ自動車工業（株）副社長 大野耐一 著, 1978



# トヨタ生産方式

…これを別な面からみてみると人が常についていて異常のときに機械の代わりをすることは、いつまでたっても異常がなくなれないということである。…材料や機械に内在する問題が管理監督者の知らないところで繕われていては、いつまでたっても改善が進まないし、原価は安くない。**異常があれば機械をとめるということは問題を明らかにするという事**でもある。問題がはっきりすれば改善もすすむ。

トヨタ生産方式 - 脱規模の経営をめざして - 元トヨタ自動車工業（株）副社長 大野耐一 著, 1978



## 自動化

システムの制御に対して人間の介在を最小限にする/削減していく

## 自働化

異常の見える化 /

異常対応は人間の介在を前提

# インフラエンジニアの宿命 (さだめ)

## インフラは Moving Target

ソフトウェア・周辺環境は、猛スピードで変化する

- 既知のバグが修正されたり
- 新規の機能が追加されたり
- 不要な機能が削除されたり
- 既存の機能が改変されたり
- 新規のバグが混入されたり

未知の異常発生を前提としなければならない

→ **人間**が介在する自動化を志向

# あらためて、NIIでは

- 学術・研究機関 … **要求が多様！**
- 少量多品種
  - Hadoop族 … 組み合わせ
  - 仮想化・コンテナ化の要望 … さらに組み合わせ
- 自動化するには多様すぎる→**判断は人にさせよう**
  - 手順とその結果の見える化
  - 手順を機械的に再実行
  - 何が起きているのかを自覚させる

# 機械化

Hardiman, General Electric in 1965

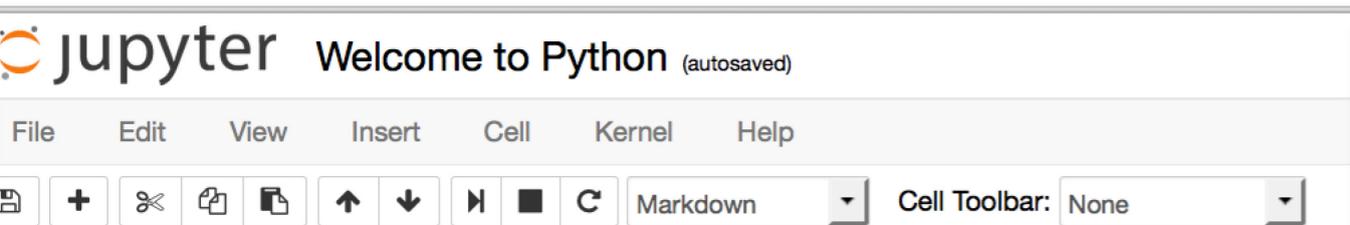
<https://www.engadget.com/2014/01/26/ge-man-amplifying-robots/>

**Literate  
Computing  
for  
Reproducible  
Infrastructure**

# Jupyter: Literate Computing



# Jupyter



Jupyter



## Welcome to the Temporary Notebook (tmpnb) service

This Notebook Server was **launched just for you**. It's a temporary way for you to try development version of the IPython/Jupyter notebook.

### WARNING

Don't rely on this server for anything you want to last - your server will be *deleted* minutes of inactivity.

Your server is hosted thanks to [Rackspace](#), on their on-demand bare metal servers, C

### Run some Python code!

## Data Scientist向けの道具

## 簡単にお試しできるJupyter Notebook環境

<https://tmpnb.org>

A full tutorial for using the notebook interface is available [here](#).

```
In [1]: %matplotlib inline

import pandas as pd
import numpy as np
import matplotlib

from matplotlib import pyplot as plt
import seaborn as sns

ts = pd.Series(np.random.randn(1000), index=pd.date_range('1/1/2012', periods=1000))
ts = ts.cumsum()

df = pd.DataFrame(np.random.randn(1000, 4), index=ts, columns=['A', 'B', 'C', 'D'])
df = df.cumsum()
plt.figure(); df.plot(); plt.legend(loc='best')
```

Out[1]: <matplotlib.legend.Legend at 0x7f38054c7cc0>  
<matplotlib.figure.Figure at 0x7f3838cf2b70>



# Literate Computing for Reproducible Infrastructure

## Jupyter + Ansible



運用に関する**すべての作業**における

- 実行コード
- 実行結果
- 説明

をNotebookとして、ひとまとめに

The screenshot shows a Jupyter Notebook interface with the following content:

### Spark HistoryServerの起動

```
In [47]: !ansible-playbook ansible/start_spark_historyserver.yml -i [ hosts ]
```

PLAY [hadoop\_spark\_history] \*\*\*\*\*

GATHERING FACTS \*\*\*\*\*

```
ok: [ ]
```

TASK: [check\_status\_spark\_history\_server] \*\*\*\*\*

```
ok: [ ]
```

TASK: [start\_spark\_history\_server] \*\*\*\*\*

```
changed: [ ]
```

PLAY RECAP \*\*\*\*\*

```
ok=3 changed=1 unreachable=0 failed=0
```

これでSparkの準備は完了。以下のURLからSparkの実行履歴が確認できる。

```
In [48]: sparkhistory_stdout = !ansible hadoop_spark_history -m ping -i [ hosts ]
sparkhistory_nodes = [l.split()[0] for l in sparkhistory_stdout if 'success' in l]
print("http://%s:18080/" % sparkhistory_nodes[0])
```

http:// :18080/

### インストール後確認

ディスクの使用状況は？

```
In [49]: !ansible all -a "df -H" -i [ hosts ]
```

filesystem	Size	Used	Avail	Use%	Mounted on
/dev/sda2	14G	3.2G	11G	24%	/
tmpfs	51G	0	51G	0%	/dev/shm
/dev/sda5	678G	105M	644G	1%	/mnt
/dev/sdb1	886G	1.1G	840G	1%	/var/log
/dev/sdc1	886G	115M	841G	1%	/hadoop

.....

filesystem	Size	Used	Avail	Use%	Mounted on
/dev/sda2	14G	5.9G	7.4G	45%	/
tmpfs	51G	0	51G	0%	/dev/shm
/dev/sda5	678G	105M	644G	1%	/mnt
/dev/sdb1	886G	951M	840G	1%	/var/log
/dev/sdc1	886G	114M	841G	1%	/hadoop

.....

filesystem	Size	Used	Avail	Use%	Mounted on
/dev/sda2	14G	3.2G	11G	24%	/
tmpfs	51G	0	51G	0%	/dev/shm
/dev/sda5	678G	105M	644G	1%	/mnt
/dev/sdb1	886G	1.1G	840G	1%	/var/log
/dev/sdc1	886G	115M	841G	1%	/hadoop

### 動作確認

#### HDFSの動作確認

hadoop client としての仮想マシンで、hdfs dfs -ls / のように実行してみる。

**Contents [-] o**

2. ZooKeeperの起動
3. ZooKeeperの動作確認
3. HDFS構築・起動
1. JournalNode
1. JournalNodeのインストール
2. JournalNodeの起動
2. NameNode
1. NameNodeのインストール
2. Primary側のNameNodeのフォーマット
3. Active側のNameNodeを起動
4. Backup側のNameNodeをPrimary側に同期させる
5. Standby側のNameNodeを起動させる
3. DataNode(SlaveNode)
1. DataNode/NodeManagerのインストール
2. DataNodeの起動
4. HDFSの初期設定
4. YARNのインストール・起動
1. ResourceManagerのインストール
2. NodeManagerのインストール
3. HDFSの準備
4. MapReduceHistoryServerのインストール
5. ResourceManagerを起動
6. NodeManagerを起動
7. MapReduceHistoryServerを起動
5. Pigのインストール
6. HBase
1. インストール
2. 起動
7. Tez
8. Hive
9. Swimlane
10. Sparkのインストール・起動
1. HDFSの準備
2. Sparkのインストール
3. Spark HistoryServerのインストール
4. Spark HistoryServerの起動
4. インストール後確認
5. 動作確認
1. HDFSの動作確認
2. MapReduceの動作確認
3. Pigの動作確認
4. Hiveの動作確認
5. 監視画面

# 1 About: Hadoop準備用Notebook

Hadoop環境の準備用のNotebookです。以下のソフトウェアをインストールします。

- HDFS
- YARN
- Pig
- HBase
- Hive

対象の物理マシンには、prerequisite Playbookが適用されているものとします。

## 2 インストール対象設定

この定義は表形式で管理します。 [hosts.csv](#) により定義します。

```
In [1]: target_cluster = 'Cluster2'
```

```
In [2]: %run ../common/loader.py

header, machines = read_machines("../resources/hosts.csv")

machines = [m for m in machines if m['Cluster'] == target_cluster]

print("Cluster(%s):" % target_cluster)
pd.DataFrame([get_row(header, m) for m in machines], columns=header)
```

Cluster(Cluster2):

Out [2]:

	Cluster	Type	Name	Internal IP	Service IP	VCPUs	Memory(MiB)	Ganglia	ZooKeeper	JournalNode
0	Cluster2	cn0107	cn01070601	10.15.2.81		12	98304	True	True	True
1	Cluster2	cn0107	cn01070602	10.15.2.82		12	98304	False	True	True
2	Cluster2	sn0202	sn02020801	10.15.4.10		16	65536	False	True	True
3	Cluster2	sn0202	sn02021601	10.15.4.6		16	65536	False	False	False
4	Cluster2	sn0203	sn02030801	10.15.4.22		16	65536	False	False	False

5 rows x 11 columns

- 1 About: Hadoop準備用Notebook
- 2 インストール対象設定
  - 2.1 Ansible用ホスト定義の生成
    - 2.1.1 インベントリファイル
    - 2.1.2 グループ変数定義(hadoop\_all)
  - 2.2 Notebook用変数の定義
  - 2.3 インストール対象マシンの確認
- 3 ソフトウェアのインストールと起動
  - 3.1 インストール(OS/cgroups)
  - 3.2 ZooKeeper構築
    - 3.2.1 ZooKeeperのインストール
    - 3.2.2 ZooKeeperの起動
    - 3.2.3 ZooKeeperの動作確認
  - 3.3 HDFS構築・起動
    - 3.3.1 JournalNode
      - 3.3.1.1 JournalNodeのインストール
      - 3.3.1.2 JournalNodeの起動
    - 3.3.2 NameNode
      - 3.3.2.1 NameNodeのインストール
      - 3.3.2.2 Primary側のNameNodeの起動
      - 3.3.2.3 Active側のNameNodeを起動
      - 3.3.2.4 Backup側のNameNodeを起動
      - 3.3.2.5 Standby側のNameNodeを起動
    - 3.3.3 DataNode(SlaveNode)
      - 3.3.3.1 DataNode/NodeManagerのインストール
      - 3.3.3.2 DataNodeの起動
    - 3.3.4 HDFSの初期設定
  - 3.4 YARNのインストール・起動
    - 3.4.1 ResourceManagerのインストール
    - 3.4.2 NodeManagerのインストール
    - 3.4.3 HDFSの準備
    - 3.4.4 MapReduceHistoryServerのインストール
    - 3.4.5 ResourceManagerを起動
    - 3.4.6 NodeManagerを起動
    - 3.4.7 MapReduceHistoryServerを起動
  - 3.5 Pigのインストール
  - 3.6 HBase
    - 3.6.1 インストール
    - 3.6.2 起動
  - 3.7 Tez
  - 3.8 Hive
  - 3.9 Swimplane
  - 3.10 Sparkのインストール・起動

## 2.2 DataNode - 現状

DeadとなっているDataNodeのログを確認する。

```
In [7]: !ansible -b -a 'ls -latr /var/log/hadoop-hdfs' -i {hosts} {target_node}
```

```
 | SUCCESS | rc=0 >>
total 4320984
-rw-r--r-- 1 hdfs hdfs 3024 Nov 29 2015 hadoop-hdfs-datanode-sn02031601.out.5
-rw-r--r-- 1 hdfs hdfs 3024 Dec 8 2015 hadoop-hdfs-datanode-sn02031601.out.4
-rw----- 1 root root 0 Jan 18 14:56 jsvc.out
-rw-r--r-- 1 root root 742 Jan 18 14:56 hadoop-hdfs-datanode-sn02031601.out.3
-rw-r--r-- 1 root root 742 Jan 18 16:39 hadoop-hdfs-datanode-sn02031601.out.2
-rw-r--r-- 1 root root 742 Mar 16 13:31 hadoop-hdfs-datanode-sn02031601.out.1
-rw-r--r-- 1 root root 742 May 17 07:07 hadoop-hdfs-datanode-sn02031601.out
-rw----- 1 root root 564 May 17 07:07 jsvc.err
-rw-r--r-- 1 hdfs hdfs 268435752 Jun 22 02:26 hadoop-hdfs-datanode-sn02031601.log.16
-rw-r--r-- 1 hdfs hdfs 268435629 Jun 23 05:17 hadoop-hdfs-datanode-sn02031601.log.15
-rw-r--r-- 1 hdfs hdfs 433 Jun 23 19:47 SecurityAuth-hdfs.audit
-rw-r--r-- 1 hdfs hdfs 268435634 Jun 24 00:49 hadoop-hdfs-datanode-sn02031601.log.14
-rw-r--r-- 1 hdfs hdfs 268435702 Jun 24 14:37 hadoop-hdfs-datanode-sn02031601.log.13
-rw-r--r-- 1 hdfs hdfs 268435534 Jun 25 02:13 hadoop-hdfs-datanode-sn02031601.log.12
-rw-r--r-- 1 hdfs hdfs 268435699 Jun 25 11:44 hadoop-hdfs-datanode-sn02031601.log.11
-rw-r--r-- 1 hdfs hdfs 268435630 Jun 25 22:24 hadoop-hdfs-datanode-sn02031601.log.10
drwxr-xr-x 16 root root 4096 Jun 26 03:06 ..
-rw-r--r-- 1 hdfs hdfs 268435489 Jun 26 09:45 hadoop-hdfs-datanode-sn02031601.log.9
```

*hadoop-hdfs-datanode-snXXXXXXXXX.log* を確認する。

```
In [8]: !ansible -b -m shell -a 'tail -n 300 /var/log/hadoop-hdfs/hadoop-hdfs-datanode-$(hostname).log' -i {hosts} {target_node}
```

```
because we rescanned it recently.
2016-06-30 17:55:12,060 INFO org.apache.hadoop.hdfs.server.datanode.VolumeScanner: VolumeScanner(/hadoop/data10/dfs/datadir, DS-cb8f181a-21b5-4872-8120-2be40d02a0b0): Not scheduling suspect block BP-1102141065-157.1.200.197-1444031914045:blk_1073949944_222674 for rescanning, because we rescanned it recently.
2016-06-30 17:55:12,063 INFO org.apache.hadoop.hdfs.server.datanode.VolumeScanner: VolumeScanner(/hadoop/data10/dfs/datadir, DS-cb8f181a-21b5-4872-8120-2be40d02a0b0): Not scheduling suspect block BP-1102141065-157.1.200.197-1444031914045:blk_1073949944_222674 for rescanning, because we rescanned it recently.
2016-06-30 17:55:12,068 INFO org.apache.hadoop.hdfs.server.datanode.VolumeScanner: VolumeScanner(/hadoop/data10/dfs/datadir, DS-cb8f181a-21b5-4872-8120-2be40d02a0b0): Not scheduling suspect block BP-1102141065-157.1.200.197-1444031914045:blk_1073949944_222674 for rescanning, because we rescanned it recently.
2016-06-30 17:55:12,070 INFO org.apache.hadoop.hdfs.server.datanode.VolumeScanner: VolumeScanner(/hadoop/data10/dfs/datadir, DS-cb8f181a-21b5-4872-8120-2be40d02a0b0): Not scheduling suspect block BP-1102141065-157.1.200.197-1444031914045:blk_1073949944_222674 for rescanning, because we rescanned it recently.
2016-06-30 17:55:12,071 INFO org.apache.hadoop.hdfs.server.datanode.VolumeScanner: VolumeScanner(/hadoop/data10/dfs/datadir, DS-cb8f181a-21b5-4872-8120-2be40d02a0b0): Not scheduling suspect block BP-1102141065-157.1.200.197-1444031914045:blk_1073949944_222674 for rescanning, because we rescanned it recently.
```

### Contents [-] o n t

- 1 About: DataNode sn02031601がDead
  - 1.1 Operation Note
  - 1.2 Target Cluster
- 2 現状の確認
  - 2.1 HDFSクラスタ - 現状
  - 2.2 DataNode - 現状
- 3 問題発生時の状況確認
  - 3.1 DataNode - 問題発生時
  - 3.2 HDFSクラスタ - 問題発生時
  - 3.3 現在のBlockの状態
- 4 ハードウェアの異常調査

# 文芸的 機械化

# “Literate Computing for Reproducible Infrastructure”

## 文芸的機械化

機械的に再現できる、人が読み解ける手順  
人が介在しつつも機械的に再現/再実行できる

- 計算機 = computation と 人間どうし = communication の分担
- 実際の実行結果が手順とともに, dataが live-code に埋め込まれていて再現できる
  - 手順や経緯が具体的に再現可能な形で表現・伝達

# 例: Hadoopの構築

## 1 About: Hadoop準備用Notebook

Hadoop環境の準備用のNotebookです。以下のソフトウェアをインストールします。

- HDFS
- YARN
- Pig
- HBase
- Hive

対象の物理マシンには、prerequisite Playbookが適用されているものとします。

## 2 インストール対象設定

この定義は表形式で管理します。 [hosts.csv](#) により定義します。

```
In [1]: target_cluster = 'Cluster2'
```

```
In [2]: %run ../common/loader.py

header, machines = read_machines("../resources/hosts.csv")

machines = [m for m in machines if m['Cluster'] == target_cluster]

print("Cluster(%s):" % target_cluster)
pd.DataFrame([get_row(header, m) for m in machines], columns=header)
```

Cluster(Cluster2):

Out [2]:

	Cluster	Type	Name	Internal IP	Service IP	VCPUs	Memory(MiB)	Ganglia	ZooKeeper	JournalNode	NameNode	DataNode	Res
0	Cluster2	cn0107	cn01070601	10.15.2.81		12	98304	True	True	True	True	True	True
1	Cluster2	cn0107	cn01070602	10.15.2.82		12	98304	False	True	True	True	True	True
2	Cluster2	sn0202	sn02020801	10.15.4.10		16	65536	False	True	True	True	True	True
3	Cluster2	sn0202	sn02021601	10.15.4.6		16	65536	False	False	False	False	False	False
4	Cluster2	sn0203	sn02030801	10.15.4.22		16	65536	False	False	False	False	False	False

### Contents [-] o n t

- 1 About: Hadoop準備用Notebook
- 2 インストール対象設定
  - 2.1 Ansible用ホスト定義の生成
    - 2.1.1 インベントリファイル
    - 2.1.2 グループ変数定義(hadoop\_all)
  - 2.2 Notebook用変数の定義
  - 2.3 インストール対象マシンの確認
- 3 ソフトウェアのインストールと起動
  - 3.1 インストール(OS/cgroups)
  - 3.2 ZooKeeper構築
    - 3.2.1 ZooKeeperのインストール
    - 3.2.2 ZooKeeperの起動
    - 3.2.3 ZooKeeperの動作確認
  - 3.3 HDFS構築・起動
    - 3.3.1 JournalNode
      - 3.3.1.1 JournalNodeのインストール
      - 3.3.1.2 JournalNodeの起動
    - 3.3.2 NameNode
      - 3.3.2.1 NameNodeのインストール
      - 3.3.2.2 Primary側のNameNodeのフォーマット
      - 3.3.2.3 Active側のNameNodeを起動
      - 3.3.2.4 Backup側のNameNodeをPrimary側に同期させる
      - 3.3.2.5 Standby側のNameNodeを起動させる
    - 3.3.3 DataNode(SlaveNode)
      - 3.3.3.1 DataNode/NodeManagerのインストール
      - 3.3.3.2 DataNodeの起動
    - 3.3.4 HDFSの初期設定
  - 3.4 YARNのインストール・起動
    - 3.4.1 ResourceManagerのインストール
    - 3.4.2 NodeManagerのインストール
    - 3.4.3 HDFSの準備
      - 3.4.4 MapReduceHistoryServerのインストール
      - 3.4.5 ResourceManagerを起動
      - 3.4.6 NodeManagerを起動
      - 3.4.7 MapReduceHistoryServerを起動
    - 3.5 Pigのインストール
    - 3.6 HBase
      - 3.6.1 インストール
      - 3.6.2 起動
    - 3.7 Tez
    - 3.8 Hive

# Automation ではなく.. Communication

人に判断をさせる

人のパフォーマンス: **Communication** が重要

- スキルセットが異なる当事者間においても、個々の作業の再現性を担保したい
- インフラの状態やそこに至るまでの経緯を理解し易くする
- その上で作業をカスタマイズ・再利用すると言ったプロセス自体も(Code として)見える化
- 伝達可能にする/蓄積・発展させやすくする

# 例: 収容設計情報から設定ファイルを作成

## 2 インストール対象設定

この定義は表形式で管理します。 [hosts.csv](#) により定義します。

```
In [1]: target_cluster = 'Cluster2'
```

```
In [2]: %run ../common/loader.py

header, machines = read_machines("../resources/hosts.csv")

machines = [m for m in machines if m['Cluster'] == target_cluster]

print("Cluster(%s):" % target_cluster)
pd.DataFrame([get_row(header, m) for m in machines], columns=header)
```

Cluster(Cluster2):

```
Out [2]:
```

Name	Internal IP	Service IP	VCPUs	Memory(MiB)	Ganglia	ZooKeeper	JournalNode	NameNode	DataNode	ResourceManager	Noc
cn01070601	10.15.2.81		12	98304	True	True	True	True	False	True	Fals
cn01070602	10.15.2.82		12	98304	False	True	True	True	False	True	Fals
sn02020801	10.15.4.10		16	65536	False	True	True	False	True	False	Tru
sn02021601	10.15.4.6		16	65536	False	False	False	False	True	False	Tru
sn02030801	10.15.4.22		16	65536	False	False	False	False	True	False	Tru

### 2.1 Ansible用ホスト定義の生成

AnsibleにHadoop環境構築をおこなわせるため、以下の2つのファイルを作成します。

- Inventory ... グループとホストの対応関係
- group\_vars ... 各Playbookに渡す変数

```
In [3]: import os
# 利用するインベントリファイル
hosts = "ansible/hosts-2"
```

# 例: 運用作業 … 故障時の対応

## 2 現状の確認

### 2.1 HDFSクラスタ - 現状

dfsadminが提供するレポートを確認する。

```
In [2]: !ansible --become --become-user hdfs -m shell -a 'kinit -k -t /etc/hadoop/conf/hdfs.keytab hdfs/${hostname};  
        hdfs dfsadmin -report' -i {hosts} hadoop_namenode_primary
```

```
██████████ | SUCCESS | rc=0 >>  
Configured Capacity: 177169429954560 (161.13 TB)  
Present Capacity: 168163497187247 (152.94 TB)  
DFS Remaining: 166313073862311 (151.26 TB)  
DFS Used: 1850423324936 (1.68 TB)  
DFS Used%: 1.10%  
Under replicated blocks: 0  
Blocks with corrupt replicas: 0  
Missing blocks: 0  
Missing blocks (with replication factor 1): 746
```

-----  
Live datanodes (6):

```
Name: 157. ██████████  
Hostname: ██████████  
Decommission Status : Normal  
Configured Capacity: 29528238325760 (26.86 TB)  
DFS Used: 341820078658 (318.34 GB)  
Non DFS Used: 1500922806880 (1.37 TB)  
DFS Remaining: 27685495440222 (25.18 TB)  
DFS Used%: 1.16%  
DFS Remaining%: 93.76%  
Configured Cache Capacity: 0 (0 B)  
Cache Used: 0 (0 B)  
Cache Remaining: 0 (0 B)  
Cache Used%: 100.00%  
Cache Remaining%: 0.00%  
Xceivers: 7  
Last contact: Thu Jun 30 18:42:56 JST 2016
```

```
Name: 157. ██████████
```

#### Contents [-] o n t

- 1 About: DataNode sn02031601がDead
  - 1.1 Operation Note
  - 1.2 Target Cluster
- 2 現状の確認
  - 2.1 HDFSクラスタ - 現状
  - 2.2 DataNode - 現状
- 3 問題発生時の状況確認
  - 3.1 DataNode - 問題発生時
  - 3.2 HDFSクラスタ - 問題発生時
  - 3.3 現在のBlockの状態
- 4 ハードウェアの異常調査

# 例: 動作確認もNotebookとして記述

## 3.2.2 ZooKeeperの起動

```
In [18]: !ansible-playbook ansible/start_zookeeper-server.yml -i { hosts }
```

```
PLAY [hadoop_zookeeper_server] *****

GATHERING FACTS *****
ok: [ ]
ok: [ ]
ok: [ ]

TASK: [start_zookeeper-server] *****
changed: [ ]
changed: [ ]
changed: [ ]

PLAY RECAP *****
                : ok=2    changed=1    unr eachable=0    failed=0
                : ok=2    changed=1    unr eachable=0    failed=0
                : ok=2    changed=1    unr eachable=0    failed=0
```

## 3.2.3 ZooKeeperの動作確認

```
In [19]: !pip install zk-shell
```

```
Requirement already satisfied (use --upgrade to upgrade): zk-shell in /usr/local/lib/python2.7/dist-packages
Requirement already satisfied (use --upgrade to upgrade): ansicolors in /usr/local/lib/python2.7/dist-packages (from zk-shell)
Requirement already satisfied (use --upgrade to upgrade): kazoo>=2.2.1 in /usr/local/lib/python2.7/dist-packages (from zk-shell)
Requirement already satisfied (use --upgrade to upgrade): tabulate in /usr/local/lib/python2.7/dist-packages (from zk-shell)
Requirement already satisfied (use --upgrade to upgrade): twitter.common.net in /usr/local/lib/python2.7/dist-packages (from zk-shell)
Requirement already satisfied (use --upgrade to upgrade): twitter.common.log=0.3.4 in /usr/local/lib/python2.7/dist-packages (from twitter.common.net->zk-shell)
Requirement already satisfied (use --upgrade to upgrade): twitter.common.quantity=0.3.4 in /usr/local/lib/python2.7/dist-packages (from twitter.common.net->zk-shell)
Requirement already satisfied (use --upgrade to upgrade): twitter.common.dirutil=0.3.4 in /usr/local/lib/python2.7/dist-packages (from twitter.common.log=0.3.4->twitter.common.net->zk-shell)
Requirement already satisfied (use --upgrade to upgrade): twitter.common.options=0.3.4 in /usr/local/lib/python2.7/dist-packages (from twitter.common.log=0.3.4->twitter.common.net->zk-shell)
Requirement already satisfied (use --upgrade to upgrade): twitter.common.lang=0.3.4 in /usr/local/lib/python2.7/dist-packages (from twitter.common.log=0.3.4->twitter.common.net->zk-shell)
```

# 例: 運用作業 ... 故障時の対応

## 2.2 DataNode - 現状

DeadとなっているDataNodeのログを確認する。

```
In [7]: !ansible -b -a 'ls -latr /var/log/hadoop-hdfs' -i {hosts} {target_node}
```

```
██████████ | SUCCESS | rc=0 >>
total 4320984
-rw-r--r-- 1 hdfs hdfs      3024 Nov 29  2015 hadoop-hdfs-datanode-sn02031601.out.5
-rw-r--r-- 1 hdfs hdfs      3024 Dec  8  2015 hadoop-hdfs-datanode-sn02031601.out.4
-rw----- 1 root root         0 Jan 18 14:56 jsvc.out
-rw-r--r-- 1 root root       742 Jan 18 14:56 hadoop-hdfs-datanode-sn02031601.out.3
-rw-r--r-- 1 root root       742 Jan 18 16:39 hadoop-hdfs-datanode-sn02031601.out.2
-rw-r--r-- 1 root root       742 Mar 16 13:31 hadoop-hdfs-datanode-sn02031601.out.1
-rw-r--r-- 1 root root       742 May 17 07:07 hadoop-hdfs-datanode-sn02031601.out
-rw----- 1 root root       564 May 17 07:07 jsvc.err
-rw-r--r-- 1 hdfs hdfs 268435752 Jun 22 02:26 hadoop-hdfs-datanode-sn02031601.log.16
-rw-r--r-- 1 hdfs hdfs 268435629 Jun 23 05:17 hadoop-hdfs-datanode-sn02031601.log.15
-rw-r--r-- 1 hdfs hdfs      433 Jun 23 19:47 SecurityAuth-hdfs.audit
-rw-r--r-- 1 hdfs hdfs 268435634 Jun 24 00:49 hadoop-hdfs-datanode-sn02031601.log.14
-rw-r--r-- 1 hdfs hdfs 268435702 Jun 24 14:37 hadoop-hdfs-datanode-sn02031601.log.13
-rw-r--r-- 1 hdfs hdfs 268435534 Jun 25 02:13 hadoop-hdfs-datanode-sn02031601.log.12
-rw-r--r-- 1 hdfs hdfs 268435699 Jun 25 11:44 hadoop-hdfs-datanode-sn02031601.log.11
-rw-r--r-- 1 hdfs hdfs 268435630 Jun 25 22:24 hadoop-hdfs-datanode-sn02031601.log.10
drwxr-xr-x 16 root root      4096 Jun 26 03:06 ..
-rw-r--r-- 1 hdfs hdfs 268435489 Jun 26 00:45 hadoop-hdfs-datanode-sn02031601.log.9
```

`hadoop-hdfs-datanode-snXXXXXXXXX.log`を確認する。

```
In [8]: !ansible -b -m shell -a 'tail -n 300 /var/log/hadoop-hdfs/hadoop-hdfs-datanode-$(hostname)
-i {hosts} {target_node}
```

```
2016-06-30 17:55:12,060 INFO org.apache.hadoop.hdfs.server.datanode.VolumeScanner: Volume
1b5-4872-8120-2be40d02a0b0): Not scheduling suspect block BP-1102141065-157.1.200.197-144
because we rescanned it recently.
2016-06-30 17:55:12,063 INFO org.apache.hadoop.hdfs.server.datanode.VolumeScanner: Volume
1b5-4872-8120-2be40d02a0b0): Not scheduling suspect block BP-1102141065-157.1.200.197-144
because we rescanned it recently.
2016-06-30 17:55:12,068 INFO org.apache.hadoop.hdfs.server.datanode.VolumeScanner: Volume
1b5-4872-8120-2be40d02a0b0): Not scheduling suspect block BP-1102141065-157.1.200.197-144
because we rescanned it recently.
2016-06-30 17:55:12,070 INFO org.apache.hadoop.hdfs.server.datanode.VolumeScanner: VolumeScanner(/hadoop/data10/dfs/datadir, DS-cb8f181a-2
1b5-4872-8120-2be40d02a0b0): Not scheduling suspect block BP-1102141065-157.1.200.197-1444031914045:blk_1073949944_222674 for rescanning
```

### Contents [-] o n t

- 1 About: DataNode sn02031601がDead
  - 1.1 Operation Note
  - 1.2 Target Cluster
- 2 現状の確認
  - 2.1 HDFSクラスタ - 現状
  - 2.2 DataNode - 現状
- 3 問題発生時の状況確認
  - 3.1 DataNode - 問題発生時
  - 3.2 HDFSクラスタ - 問題発生時
  - 3.3 現在のBlockの状態
- 4 ハードウェアの異常調査

### Contents [-] o n t

- 1 About: DataNode sn02031601がDead
  - 1.1 Operation Note
  - 1.2 Target Cluster
- 2 現状の確認
  - 2.1 HDFSクラスタ - 現状
  - 2.2 DataNode - 現状
- 3 問題発生時の状況確認
  - 3.1 DataNode - 問題発生時
  - 3.2 HDFSクラスタ - 問題発生時
  - 3.3 現在のBlockの状態
- 4 ハードウェアの異常調査

# 適用局面

日々の運用作業の証跡を記録 … Traceability

そこから手順を整理して再利用する … Reusability

マニュアルや教材を整備する … Reproducibility

これら複数局面での計算機利用を、

**「適切に分節化したものを同様に紡いで」**

記述・蓄積したい

# 作業前後の状態確認も明示的に記述

## インストール後確認

ディスクの使用状況は？

```
In [49]: !ansible all -a "df -H" -i { hosts }
```

```
██████████ | success | rc=0 >>  
Filesystem      Size  Used Avail Use% Mounted on  
/dev/sda2       14G   3.2G   11G   24% /  
tmpfs           51G    0    51G    0% /dev/shm  
/dev/sda5       678G  105M   644G    1% /mnt  
/dev/sdb1       886G   1.1G   840G    1% /var/log  
/dev/sdc1       886G  115M   841G    1% /hadoop
```

```
██████████ | success | rc=0 >>  
Filesystem      Size  Used Avail Use% Mounted on  
/dev/sda2       14G   5.9G   7.4G   45% /  
tmpfs           51G    0    51G    0% /dev/shm  
/dev/sda5       678G  105M   644G    1% /mnt  
/dev/sdb1       886G   951M   840G    1% /var/log  
/dev/sdc1       886G  114M   841G    1% /hadoop
```

まずは判断は人にさせる

徐々にコードが自動的に確認できることを増やしていく

# 構築・運用作業をAnsible Playbookとして抽象化

## NameNode

### NameNodeのインストール

```
In [16]: !ansible-playbook ansible/install_namenode.yml -i { hosts }
```

```
TASK: [java7 | install_oraclejdk] *****  
skipping: [ ]  
skipping: [ ]
```

```
TASK: [java7 | copy_bash_profile] *****  
ok: [ ]  
ok: [ ]
```

```
TASK: [java7 | copy_sudoers_conf_of_JAVA_HOME] *****  
ok: [ ]  
ok: [ ]
```

```
TASK: [namenode | install_namenode_packages] *****  
changed: [ ] => (item=hadoop-hdfs-namenode,hadoop-hdfs-zkfc)  
changed: [ ] => (item=hadoop-hdfs-namenode,hadoop-hdfs-zkfc)
```

```
TASK: [namenode | create_namenode_data_dir] *****  
changed: [ ] => (item=/hadoop/data/dfs/namedir)  
changed: [ ] => (item=/hadoop/data/dfs/namedir)
```

どうやればいいのか？

**Jupyter + Ansible**

**環境の構築**

# Jupyter+Ansibleのインストール

- ローカルマシンにDockerをインストールしテナ導入
- Macにインストール
- Windowsにインストール

# Dockerを使ったインストール

- <http://qiita.com/yacchin1205/items/928c2d53c307f18b587f>

The screenshot shows the Qiita article page for 'Jupyter+Ansibleを使ったインフラ運用のための下準備'. The article is by user 'yacchin1205' and was posted on December 10, 2016. It has 1560 views and 22 likes. The article is part of the 'jupyter notebook Advent Calendar 2016' series, specifically the 9th day. The page includes a search bar, a navigation menu, and social sharing buttons.

Advent Calendar、遅刻してしまったわけですが・・・

今年も、去年から引き続き国立情報学研究所さんにてJupyterとAnsibleを使ってインフラ運用をやらせていただきました。

本来、Jupyterはデータ分析のための道具として生まれたわけですが、文章+コードの形式で、Notebookとして実行内容を記述、実行して残しておくというやり方は、データ分析以外の場面でも役に立つなあ、としみじみ感じています。そんなわけで、Jupyter+Ansible+インフラ運用なネタを。

## この記事の内容

NIIクラウド運用チームにおけるインフラ運用のコンセプトについては

Jupyter Notebookを用いた文芸的インフラ運用のススメ

にて詳しく説明されているのでそちらを参照していただくとして、インフラ運用の"お手本(の一例)"Notebookとして、以下のようなNotebookを公開してみています。

- [Literate Computing for Reproducible Infrastructure - Basic Practice](#)
- [Literate Computing for Reproducible Infrastructure - Hadoop Practice](#)

イベントなどでお話ししますとみなさん割と興味を持っていただけるわけですが、意外とインフラとJupyterが連携するところまでの環境準備がめんどくさい感じで「軽く試してみよう」とはいかないのも事実です。

Social sharing buttons for the article, including Tweet (49), Pocket (59), G+1 (1), and Like 7.

User profile for yacchin1205, showing 889 contributions.

### 人気の投稿

- Dockerで5分くらいでGitLabを試す
- Java+Seleniumな自動テストプロジェクトをJenkinsさんをお願いする
- Java+SeleniumなWebアプリケーションの自動テストプロジェクト構築
- Pepper用Tweetボックスをつくる
- MQTTでIRKitやPhilips Hueの状況確認・操作を集約



# Dockerを使ったインストール

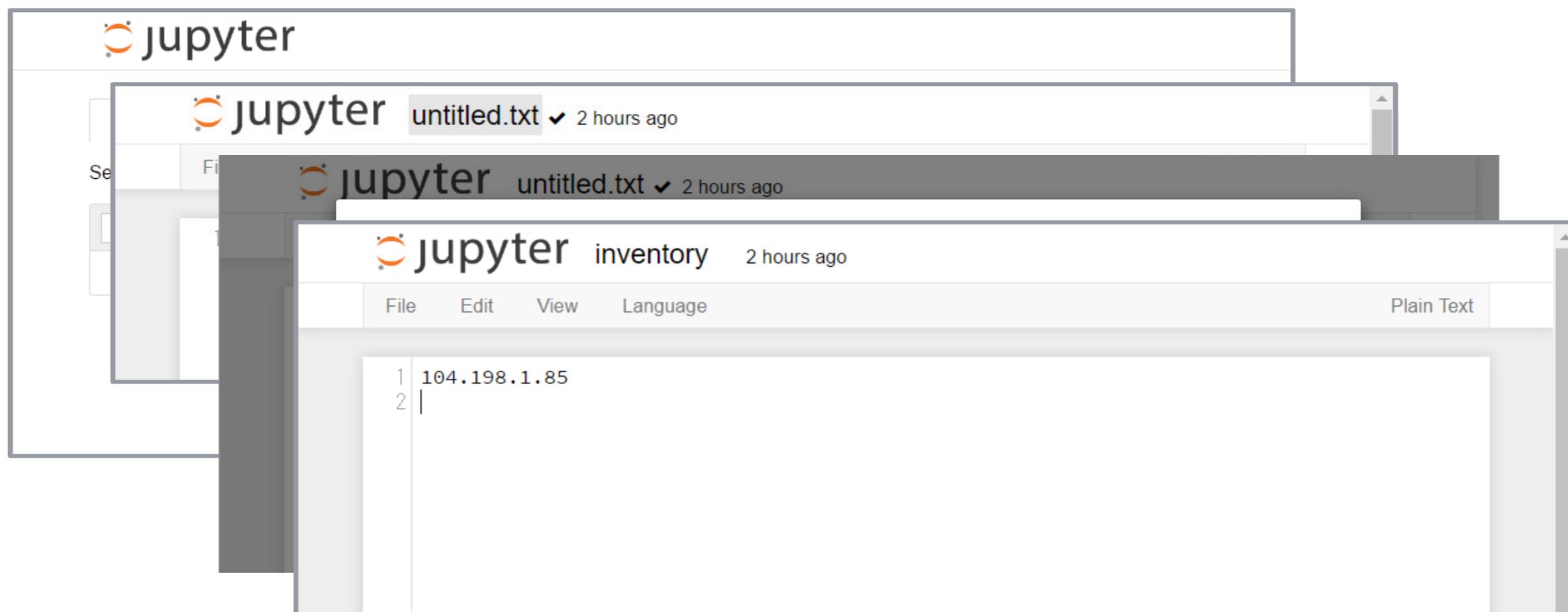
- Jupyter公式イメージに引きずられないほうがいいだろう・・・
- <https://github.com/yacchin1205/jupyter-for-infrastructure>
- インストール方法
  1. ご自分の環境にDockerをインストール
  2. ↑のリポジトリをcloneもしくはダウンロード
  3. `docker-compose up`
  4. ご自分の環境のブラウザで `http://localhost:8888`

# Dockerコンテナの使い方

- Inventoryの作成
- キーペアの作成
- ホスト情報の登録
- Ansibleによる操作

# Inventoryの作成

- (Notebookのディレクトリ)/inventory をデフォルトインベントリにしています – 一時的に確保したGCEインスタンスの例



# キーペアの作成

- 一例として...

**1 About: キーペアの作成**

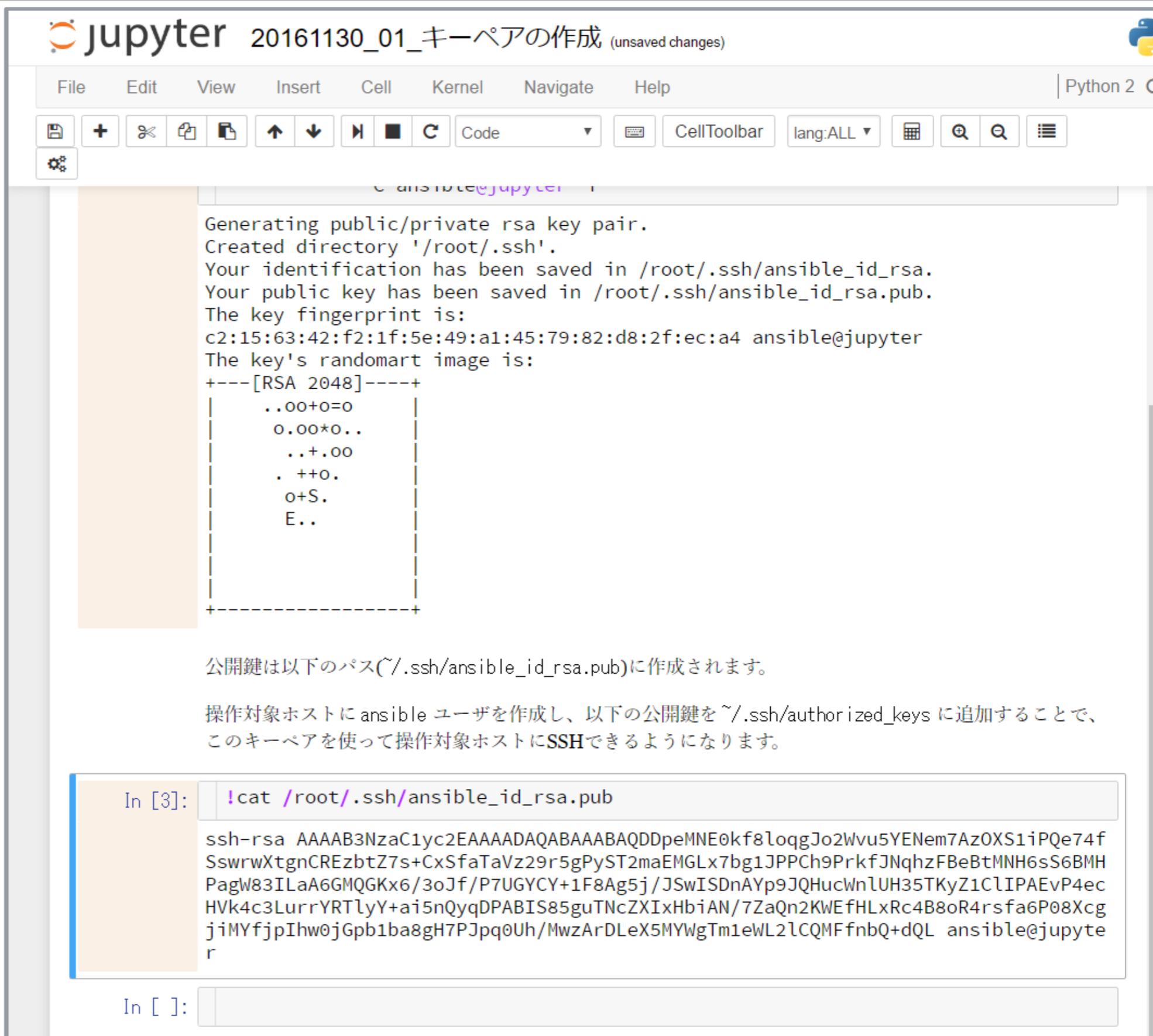
ansibleを使ったSSHアクセスのため、キーペアを作成します。作成したキーペアのうち公開鍵を操作対象のサーバに登録します。

```
In [1]: !ssh-keygen -t rsa -b 2048 -f /root/.ssh/ansible_id_rsa \
-C ansible@jupyter -P ''
```

Generating public/private rsa key pair.  
Created directory '/root/.ssh'.  
Your identification has been saved in /root/.ssh/ansible\_id\_rsa.  
Your public key has been saved in /root/.ssh/ansible\_id\_rsa.pub.  
The key fingerprint is:  
c2:15:63:42:f2:1f:5e:49:a1:45:79:82:d8:2f:ec:a4 ansible@jupyter  
The key's randomart image is:  
+---[RSA 2048]-----+  
 ..00+0=0  
 0.00\*0..  
 ..+.00  
 . ++0.  
 o+S.  
 E..

# キーペアの作成

- 一例として...



The screenshot shows a Jupyter Notebook interface with the title "20161130\_01\_キーペアの作成 (unsaved changes)". The notebook contains a code cell that has been executed, displaying the output of a command to generate an RSA key pair. The output text is as follows:

```
Generating public/private rsa key pair.  
Created directory '/root/.ssh'.  
Your identification has been saved in /root/.ssh/ansible_id_rsa.  
Your public key has been saved in /root/.ssh/ansible_id_rsa.pub.  
The key fingerprint is:  
c2:15:63:42:f2:1f:5e:49:a1:45:79:82:d8:2f:ec:a4 ansible@jupyter  
The key's randomart image is:  
+---[RSA 2048]-----+  
| ..00+0=0 |  
| 0.00*0.. |  
| ..+.00 |  
| . ++0. |  
| o+S. |  
| E.. |  
+-----+
```

公開鍵は以下のパス(~/.ssh/ansible\_id\_rsa.pub)に作成されます。

操作対象ホストに ansible ユーザを作成し、以下の公開鍵を ~/.ssh/authorized\_keys に追加することで、このキーペアを使って操作対象ホストにSSHできるようになります。

```
In [3]: !cat /root/.ssh/ansible_id_rsa.pub  
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQDDpeMNE0kf8loqgJo2Wvu5YENem7Az0XS1iPQe74f  
SswrwXtgnCREzbtZ7s+CxSfaTaVz29r5gPyST2maEMGLx7bg1JPPCh9PrkfJNqhzFBeBtMNH6sS6BMH  
PagW83ILaA6GMQGKx6/3oJf/P7UGYCY+1F8Ag5j/JSwISDnAYp9JQHucWnlUH35TKyZ1ClIPAEvP4ec  
HVk4c3LurrYRTlyY+ai5nQyqDPABIS85guTncZXIxHbiAN/7ZaQn2KWEfHLxRc4B8oR4rsfa6P08Xcg  
jiMYfjpIhw0jGpb1ba8gH7PJpq0Uh/MwzArDLEx5MYWgTm1eWL2lCQMFfnbQ+dQL ansible@jupyter  
r
```

```
In [ ]:
```

# ホスト情報の登録

- コマンドにおける対話操作

The screenshot shows a JupyterLab interface with a terminal window titled "20161130\_01\_キーペアの作成 (unsaved changes)". The terminal displays the following content:

```
SSH-Tsa AAAAB3NzaC1yc2EAAAADAQABAAQDDpenn...
SswrwXtgnCREzht7Zs+CxSfaTaVz29r5qPvST2maEMGL...
+----[RSA 2048]-----+

1.1 疎通確認

# 104.198.1.85 SSH-2.0-OpenSSH_6.6.1
no hostkey alg

In [6]: !ssh-keyscan -H 104.198.1.85 >> /root/.ssh/known_hosts

# 104.198.1.85 SSH-2.0-OpenSSH_6.6.1
# 104.198.1.85 SSH-2.0-OpenSSH_6.6.1
# 104.198.1.85 SSH-2.0-OpenSSH_6.6.1
no hostkey alg

pingに答えてくれるか?

In [7]: !ansible -m ping all

104.198.1.85 | SUCCESS => {
  "changed": false,
  "ping": "pong"
}

In [ ]:

YAB0wOVbebAs+qpAK2eLPewwIX9nXphzmJgq/vjb+OcofH1d1hLQGadpWe/WFXamZXxUU4ikzkxwjB
Jnw10MQBZ45jZiUkyX/4nEScfhLCYHnX6BMc1HhMp3kK6drrN2KHDVlHPKUco6vw7+nYRIoToMGuYTz
Ut05VMWhMyJKZVjWop+Cp7j1xjLCkG2yQQIBLd8oGm0j8P
# 104.198.1.85 SSH-2.0-OpenSSH_6.6.1
no hostkey alg

In [ ]:
```

On the right side of the terminal, there is a partial view of an SSH connection attempt:

```
NE0kf8loqgJo2Wvu5YENem7AzOXs1iP
Lx7bg1JPPCh9PrkfJNqhzFBeBtMNH6s
ISDnAYp9JQHucWnlUH35TKyZ1ClIPAE
iAN/7ZaQn2KWEfHLxRc4B8oR4rsfa6P
gTm1eWL2lCQMFfnbQ+dQL ansible@j

.1.85 (104.198.1.85)' can't be establish
:ed:7e:ca:3e:e9:4a:28:19:1f:71:91:2d:d1.
connecting (yes/no)?
```

# Ansibleによる操作

- ansible, ansible-playbookコマンドをJupyterから実行



**1.2 モジュールの実行**  
操作対象ホスト上でモジュールを実行する例。

```
In [5]: !ansible -b -m yum -a 'name=redhat-lsb-core' all
104.198.1.85 | SUCCESS | rc=0 >>
redhat-lsb-core.x86_64 0:4.1-27.el7.centos.1
  \n\nDependency Installed:\n  at.x86_64 0:3.1.13-20.el7
  bc.x86_64 0:1.06.95-13.el7
  cups-client.x86_64 1:1.6.3-22.el7
  cups-libs.x86_64 1:1.6.3-22.el7
  ed.x86_64 0:1.9-4.el7
  m4.x86_64 0:1.4.8-10.el7
  mailx.x86_64 0:12.5-12.el7_0.1
  patch.x86_64 0:2.7.1-8.el7
  smisc.x86_64 0:22.20-9.el7
  \n  redhat-lsb-submod-security.x86_64 0:4.1-27.el7.centos.1
  spax.x86_64 0:1.5.2-13.el7
  time.x86_64 0:1.7-45.el7
  \n\nComplete!\n"

In [6]: !ansible -a 'lsb_release -a' all
104.198.1.85 | SUCCESS | rc=0 >>
LSB Version:      :core-4.1-amd64:core-4.1-noarch
Distributor ID:  CentOS
Description:     CentOS Linux release 7.2.1511 (Core)
Release:         7.2.1511
Codename:        Core
```

# Macにインストール

- AnsibleはPython 2.7がよい
  - Ansible 2.2 … Tech Preview of Python 3 support
- 手っ取り早くAnacondaを使う例
  1. Anaconda2 (Python 2.7環境) をダウンロード・インストール
  2. `$ conda install jupyter`
  3. `$ conda install pip`
  4. `$ pip install ansible`
  5. `$ jupyter notebook`

# Windowsにインストール

<http://qiita.com/yacchin1205/items/8e5f6d312686a80f8af8>

Qiita

キーワードを入力

Hot Markdownによる情報共有サ...

投稿する

ストック一覧

0



## Jupyter+AnsibleをBash on Ubuntu on Windowsにインストール

4  
いいね

0  
コメント



Ansible 1099

Jupyter 213

BashOnUbuntuOnWindows 74

yacchin1205 2016年11月13日に投稿 | 投稿を編集 157views

ストック

今お仕事させていただいている国立情報学研究所では Literate Computing for Reproducible Infrastructureと銘打って、仮想環境の整備をするNotebookとかHadoopクラスタを構築するNotebookとかを公開しているわけですが、最初のとっかかりとしてJupyterとAnsibleがインストールされた環境を用意するところがちょっと考えどころになります。

Macを使っておられる方はいずれもpipでさっくり入るわけですが、Windowsだと色々悩みどころが出てきます。

まあ、インフラ系ツールを積極的に使うんならLinuxとかMacとか使えよという話はあるわけですが、今回はその悩みどころを我らがニューカマーBash on Ubuntu on Windows(以下BoW)で何とかしてやることにします。

Tweet 0 G+ 0

Like 4 Pocket 2



yacchin1205

863 Contribution

### 人気の投稿

- Dockerで5分くらいでGitLabを試す
- Java+Seleniumな自動テストプロジェクトをJenkinsさんをお願いする
- Java+SeleniumなWebアプリケーションの自動テストプロジェクト構築

# Ansibleの設定

- Ansibleをインストール
- キーペア生成
- 操作対象マシンに公開鍵登録 / sudoできるように
  - マシンのセキュリティポリシーに依存する部分
- 現在は、操作対象ごとにNotebook環境を用意
  - 運用者がアクセス可能な操作対象に応じて、Notebook環境のアクセス権を調整
- デフォルトインベントリの準備

# Notebook例

# “お手本”Notebook - Basics

基本的なNotebook例を公開していています。

<https://github.com/NII-cloud-operation/Literate-computing-Basics>

 T04_KVMのインストール-Report.ipynb	Remove ToC metadata	7 days ago
 T05_VMの一覧・到達性-Report.ipynb	Remove ToC metadata	7 days ago

 README.md

## About: Literate Computing for Reproducible Infrastructure README

Literate Computing for Reproducible Infrastructure: インフラ運用をJupyter + Ansibleでおこなう際のお手本Notebookです。

なお、これらのNotebookはNIIクラウドチーム内で行っている作業の考え方を示すためのもので、環境によってはそのままでは動作しないものもあります。



### 関連資料

- [Jupyter notebook を用いた文芸的インフラ運用のススメ - SlideShare](#)
- [Literate Automation \(文芸的自動化\) についての考察 - めもめも](#)

### お手本Notebook

お手本NotebookはこのNotebookと同じディレクトリにあります。Notebookは目的に応じて以下のような命名規則に則って

# “お手本”Notebook - Hadoop

Hadoop構築・運用のNotebook例を公開していています。

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

 T12b_Hadoop - Simple HANA job for Test.ipynb	Add notebooks for Hadoop	2 months ago
 T12c_Hadoop - Simple HBase query for Test.ipynb	Add notebooks for Hadoop	2 months ago
 T12d_Hadoop - Simple Spark script for Test.ipynb	Add notebooks for Hadoop	2 months ago
 T13b_Hadoop - Simple Hivemall query for Test.ipynb	Add notebooks for Hadoop	2 months ago
 hosts.csv	Add notebooks for Hadoop	2 months ago

 README.md

## About: Notebooks for Hadoop Clusters README

Literate Computing for Reproducible Infrastructure: インフラ運用をJupyter + Ansibleでおこなう際のお手本Notebookです。  
(Hadoop版)

このリポジトリでは、HDP(Hortonworks Data Platform, <https://jp.hortonworks.com/products/data-center/hdp/>)を利用してHadoopクラスタを構築し、運用するためのNotebook例を紹介しています。

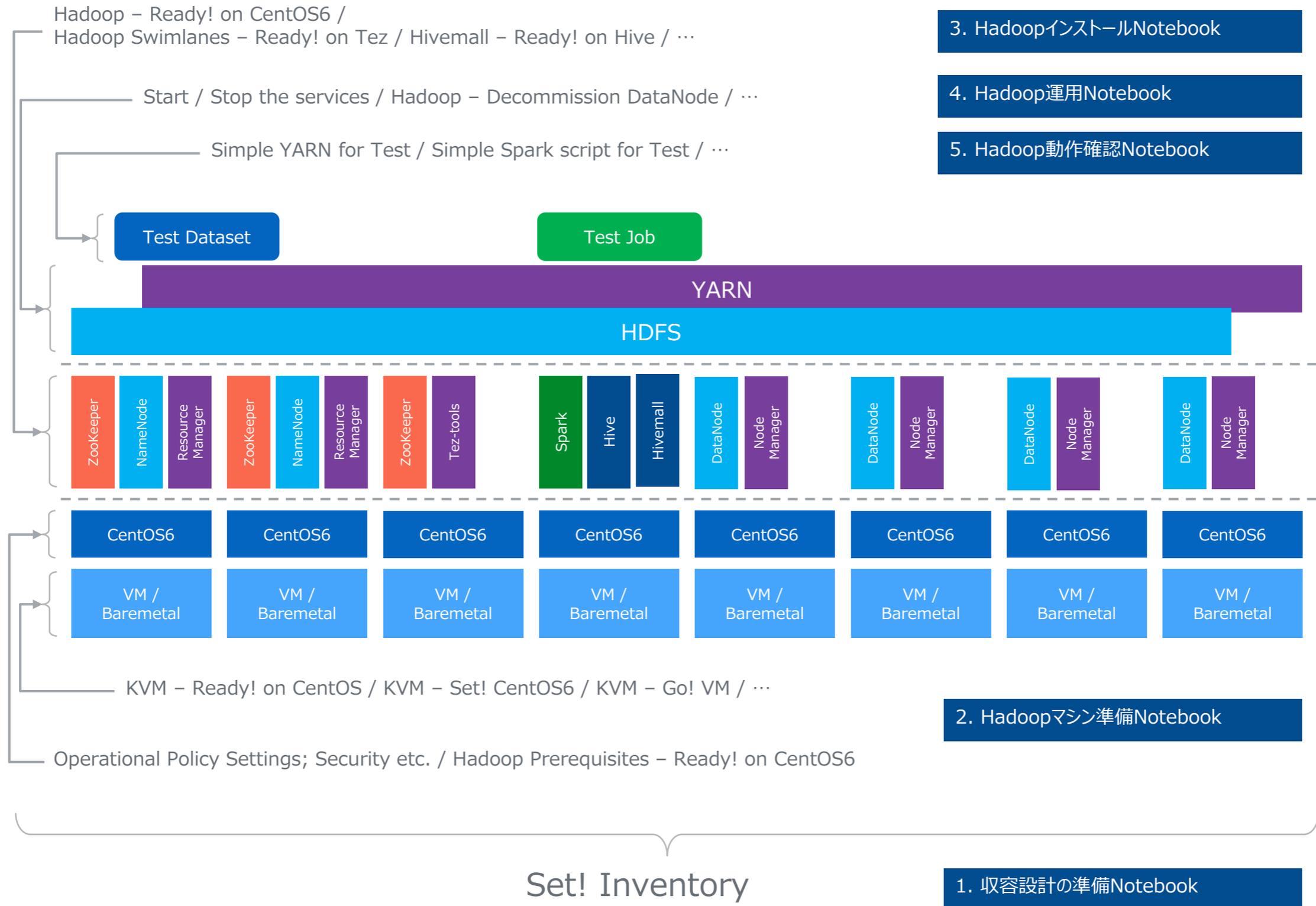
なお、これらのNotebookはNIIクラウドチーム内で行っている作業の考え方を示すためのもので、環境によってはそのままでは動作しないものもあります。



関連資料

# “お手本” Notebook - Hadoop

## • Hadoop Notebookの構成



# 今後...

- Elastic Stack用Notebookもあわせて開発中
- 実際にDatasetをElasticsearchに流しながら、
  - クエリの例
  - パフォーマンスチューニングのための勘所  
などをNotebookで解説(予定)！

效能

# 効能1: 「やり方」の伝達が簡単になる

- 昔の自分から今の自分への伝達
- 自分から同僚への伝達
- 理解する努力は必要・・・とっかかりを明示するだけ、とっかかりに対する理解を深める努力を不要にするものではない
  - 重要: 伝達された「やり方」は(Notebookに記述されている前提条件さえ満たされていれば)実行可能である(実行できたという証跡をともなっている)
  - わかりずらいことは、Notebookに追記することで改善することができる

## 効能2: 手順の漸進的な自動化

- いきなり自動化コードを書かなくてよい
  - インフラ自動化コード・・・テストが超難しい: テスト用インフラセットを準備する必要がある
- まずは人間の判断を介在させながらの手作業、判断の事例をNotebook中にまずは自然言語で蓄積する
- 判断事例が十分なものになってきたら、自然言語をコードで置き換えていく。手順自体をみなおす
- 例) HadoopのHDD故障対応
  - 対応を同じNotebookで繰り返すことで、手順自体が洗練されていく

# 罨の事例

# 同じ手順を繰り返すのが楽になるので

- 繰り返し作業の罨
- 1回目は試行錯誤的
  - 確認した項目すべてがNotebookに残っていない場合も
- 複数回Notebookを実施したことで安心感が出る
  - 「念のため」言ってNotebookに記述されている以外のことを確認している場合もある…

ある日

設定変更が反映されていない

との指摘

- <https://github.com/ansible/ansible/issues/13485>



Personal

Open source

Business

Explore

Pricing

Blog

Support

This repository

Search

Sign in

Sign up

ansible / ansible

Watch

1,405

★ Star

17,936

Fork

5,427

<> Code

🔔 Issues 1,069

🔗 Pull requests 402

📈 Pulse

📊 Graphs

## Handlers don't execute handlers in included #13485

New issue

🔒 Closed

galiev opened this issue on Dec 9, 2015 · 30 comments



galiev commented on Dec 9, 2015

When I include a file in a handler, the tasks in the included file are not executed in ansible 2.0.

Playbook redis.yml:

```
---
- hosts: redis
  gather_facts: true
  vars:
    os_firewall_use_firewalld: false
    port_open: "{{ redis_port|default(6379) }}"
  tasks:
    - include: roles/firewall/tasks/firewall_open_port.yml
  handlers:
    - include: roles/firewall/handlers/main.yml
```

**Not only the handler is not triggered, it does not throw any error if it does not exist.**

Labels

bug\_report

Milestone

stable-2.0

Assignees

No one assigned

19 participants



# 不幸中の幸い

- Ansible 2.0にアップデート後、ansible-playbookコマンドにバグが入った・・・
- コマンドの実行結果が残っている … 見落としていたことも証跡に記録されていた
- 正確なトラブル原因把握→迅速な対応が可能に

In [9]:

```
!ansible-playbook {temp_dir}/site.yml
```

```
PLAY [test-docker] *****

TASK [setup] *****
ok: [ ]

TASK [ : Gather OS Specific Variables] *****
ok: [ ] => (item=/tmp/tmpo4iOKc/roles/ /vars/Ubuntu.yml)

TASK [ : include] *****
included: /tmp/tmpo4iOKc/roles/ /tasks/ .yml for :

TASK [ : Prepare /etc/ ] *****
ok: [ ]

TASK [ : Add # to /etc/ ] *****
ok: [ ]

TASK [ : include] *****
included: /tmp/tmpo4iOKc/roles/ /tasks/ .yml for :

TASK [ : Add user] *****
changed:

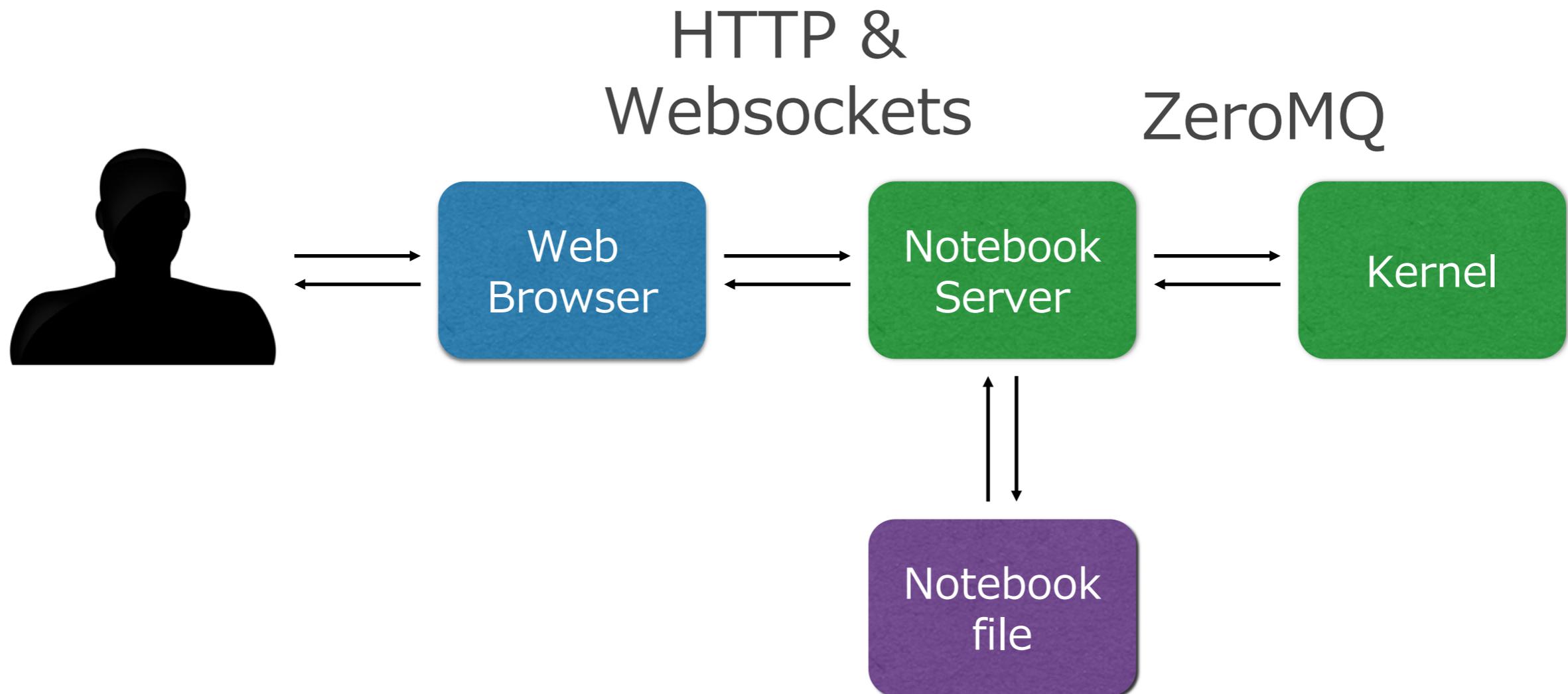
TASK [ : Prepare ] *****
changed:

TASK [ : Prepare /operators] *****
```

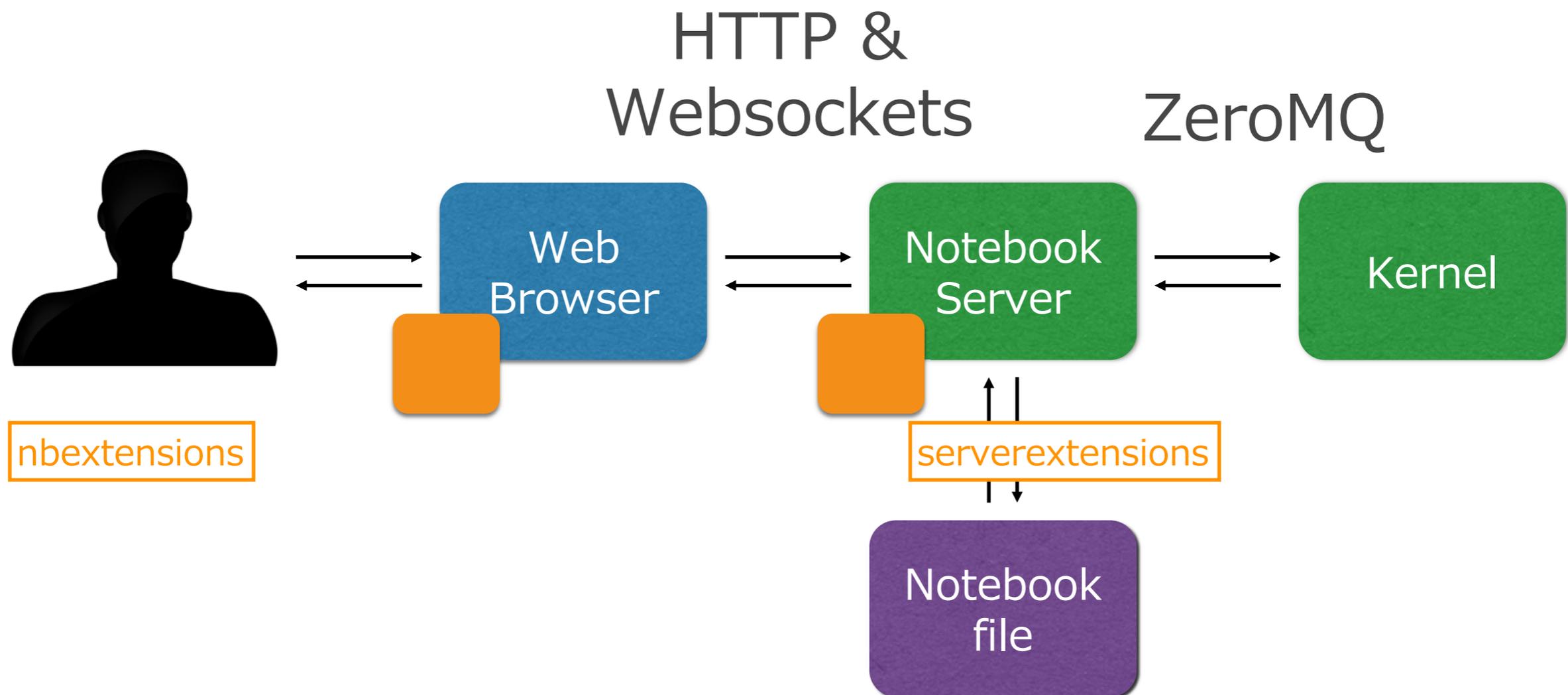
実行されたと思っていた  
項目がない・・・！

# Jupyter環境の拡張

# Jupyter Notebooks Architecture



# notebook extension system



# Extensionの種類

- nbextensions
  - ブラウザで動作するJavaScript拡張
  - UI, Notebookの動作を変更できる
    - ツールバーへのボタン追加
    - 新しいキーボードショートカット など
- Server extension
  - 4.2からの新機能
  - Notebook serverの動作を変更できる
    - 新しいHTTPエンドポイントを追加できる
    - ノートブックの処理にフックを追加可能

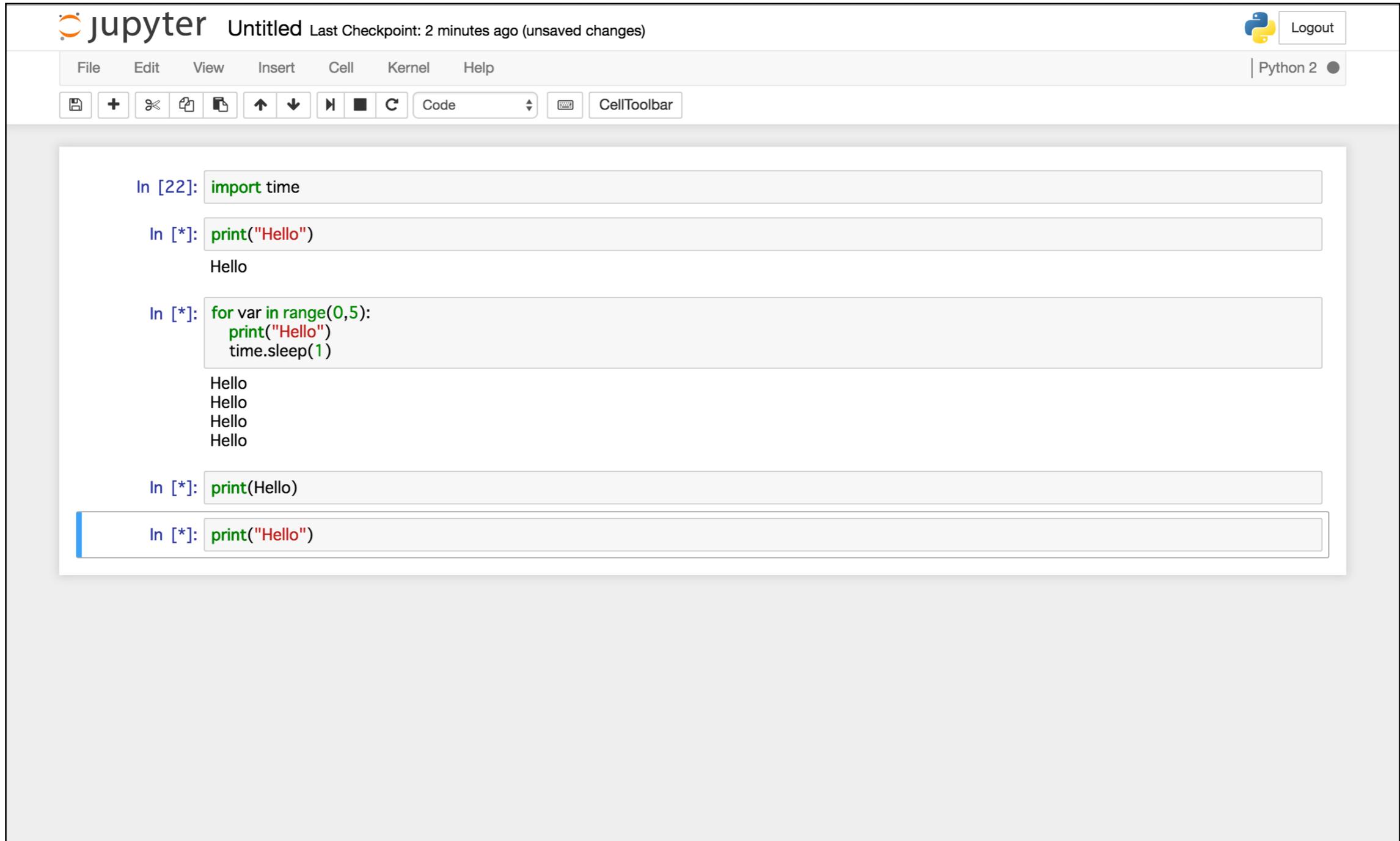
# ということで、nbextensionを作りました

- Jupyter\_code\_cell\_status
  - [https://github.com/NII-cloud-operation/Jupyter-code\\_cell\\_status](https://github.com/NII-cloud-operation/Jupyter-code_cell_status)
- Jupyter-multi\_outputs
  - [https://github.com/NII-cloud-operation/Jupyter-multi\\_outputs](https://github.com/NII-cloud-operation/Jupyter-multi_outputs)

**Jupyter-code\_cell\_status**

# Jupyter\_code\_cell\_status - 開発背景

- セルの実行状況がわかりづらい

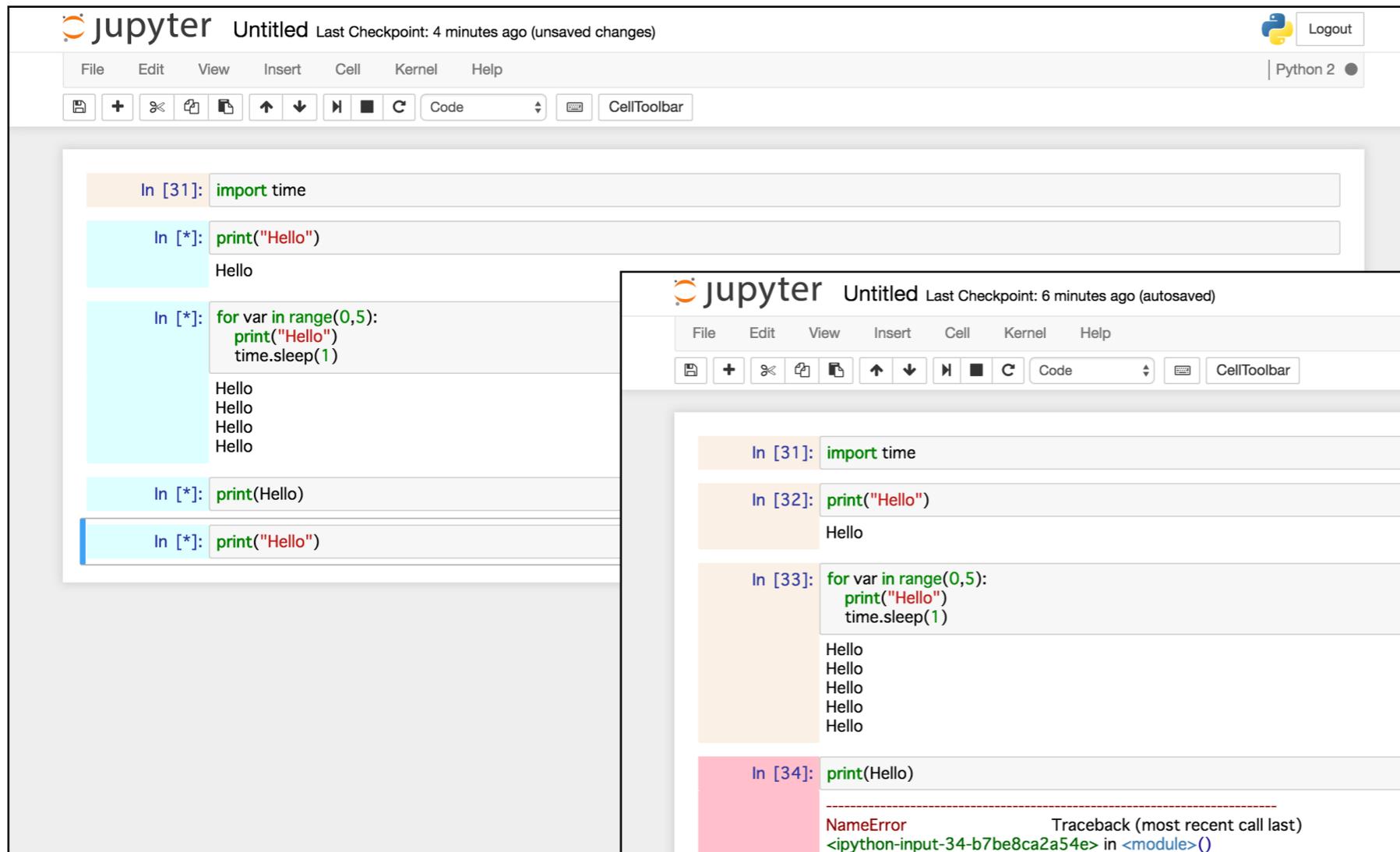


The screenshot shows a Jupyter Notebook interface with the following elements:

- Header:** "jupyter Untitled Last Checkpoint: 2 minutes ago (unsaved changes)" and a "Logout" button.
- Menu:** File, Edit, View, Insert, Cell, Kernel, Help.
- Language:** Python 2.
- Toolbar:** Includes icons for save, add, undo, redo, up/down arrows, play, stop, refresh, and a "Code" dropdown menu.
- Code Cells:**
  - Cell 1: `In [22]: import time`
  - Cell 2: `In [*]: print("Hello")` with output "Hello".
  - Cell 3: `In [*]: for var in range(0,5): print("Hello") time.sleep(1)` with output "Hello", "Hello", "Hello", "Hello".
  - Cell 4: `In [*]: print(Hello)`
  - Cell 5: `In [*]: print("Hello")` (highlighted with a blue bar).

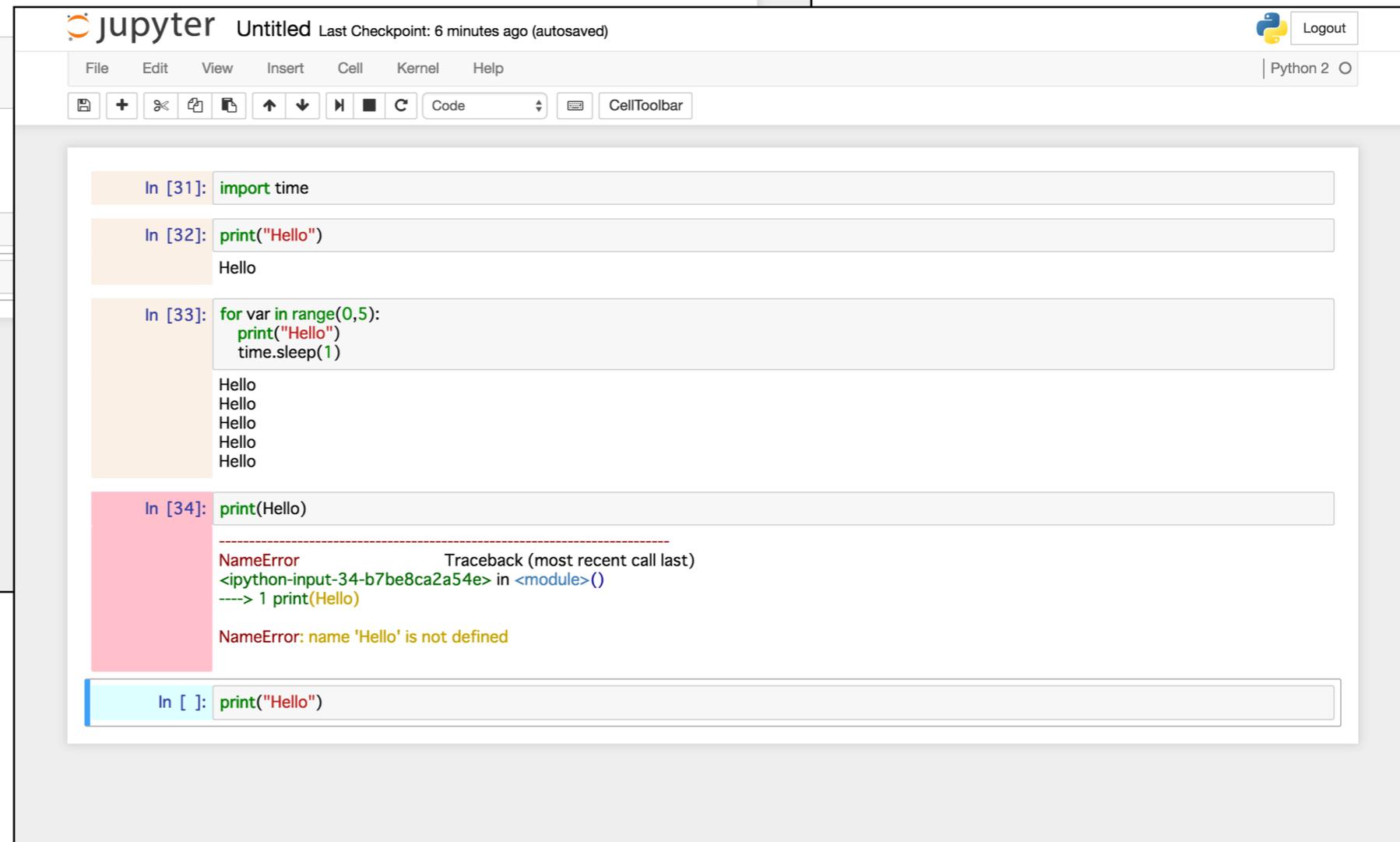
# Jupyter\_code\_cell\_status - 特徴

- セルの実行状態を色で可視化する



The screenshot shows a Jupyter Notebook interface with the following code cells and their execution status colors:

- Cell 1: `In [31]: import time` (Orange background, indicating it has been executed).
- Cell 2: `In [*]: print("Hello")` (Cyan background, indicating it is currently executing).
- Cell 3: `In [*]: for var in range(0,5): print("Hello") time.sleep(1)` (Cyan background, indicating it is currently executing).
- Cell 4: `In [*]: print(Hello)` (Cyan background, indicating it is currently executing).
- Cell 5: `In [*]: print("Hello")` (Cyan background, indicating it is currently executing).



The screenshot shows a Jupyter Notebook interface with the following code cells and their execution status colors:

- Cell 1: `In [31]: import time` (Orange background, indicating it has been executed).
- Cell 2: `In [32]: print("Hello")` (Orange background, indicating it has been executed).
- Cell 3: `In [33]: for var in range(0,5): print("Hello") time.sleep(1)` (Orange background, indicating it has been executed).
- Cell 4: `In [34]: print(Hello)` (Pink background, indicating it has failed with an error). The error message is: `NameError: name 'Hello' is not defined`.
- Cell 5: `In [ ]: print("Hello")` (Cyan background, indicating it is currently executing).

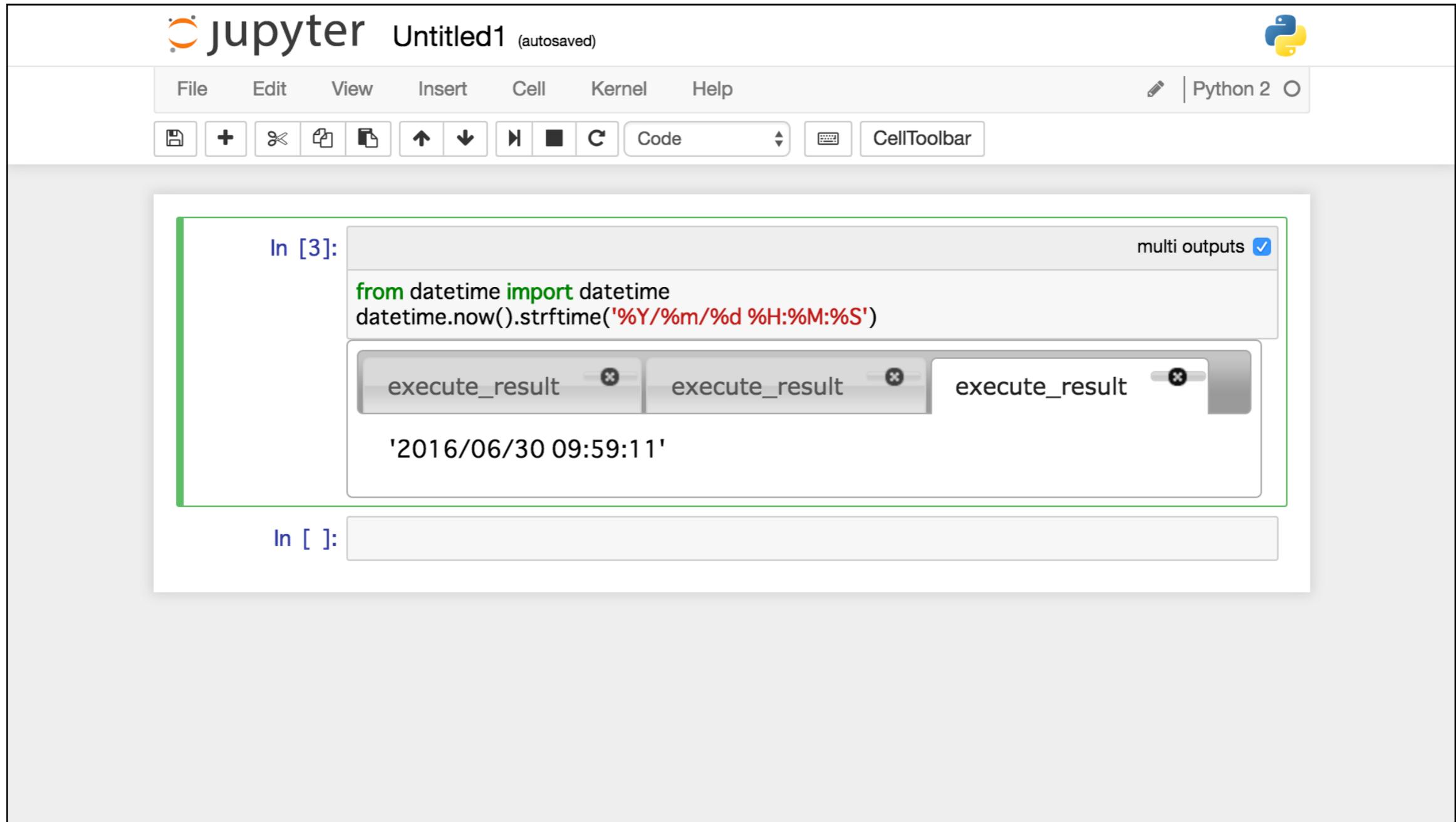
**Jupyter-multi\_outputs**

# Jupyter-multi\_outputs - 開発背景

- 実行履歴を残したい
  - 運用の証跡として
  - “Trial and Error”で変更するときの違いを比較したい
- 実行結果を比較したい
  - 前回の実行結果：今回の実行結果
  - お手本：自分のコード

# Jupyter-multi\_outputs - 特徴(1)

- 実行結果をタブとして出力



The screenshot shows the Jupyter Notebook interface for a file named 'Untitled1 (autosaved)'. The top menu bar includes 'File', 'Edit', 'View', 'Insert', 'Cell', 'Kernel', and 'Help'. The right side of the menu bar shows 'Python 2' and a refresh icon. Below the menu bar is a toolbar with icons for saving, adding, deleting, copying, pasting, undo, redo, and running code. The main area contains a code cell labeled 'In [3]:' with the following code:

```
from datetime import datetime
datetime.now().strftime('%Y/%m/%d %H:%M:%S')
```

The code cell has a 'multi outputs' checkbox checked in the top right corner. The output of the code is displayed as three separate tabs, each labeled 'execute\_result'. The first tab contains the string '2016/06/30 09:59:11'. Below the code cell is an empty code cell labeled 'In [ ]:'.

# Jupyter-multi\_outputs - 特徴(2)

- 保存・再表示が可能

The image illustrates the multi-output feature in Jupyter Notebook through three sequential screenshots:

- File Menu:** The first screenshot shows the Jupyter Notebook interface with the 'File' menu open. The 'Save and Checkpoint' option is highlighted, indicating the user's intention to save the current state of the notebook.
- Files View:** The second screenshot shows the 'Files' view of the Jupyter environment. The file 'Untitled1.ipynb' is selected, demonstrating that the notebook has been saved and is ready for reopening.
- Code Cell Execution:** The third screenshot shows a code cell with the following code:

```
In [3]: from datetime import datetime
datetime.now().strftime('%Y/%m/%d %H:%M:%S')
```

The 'multi outputs' checkbox is checked. The execution result is displayed as two separate 'execute\_result' blocks, each containing the output string: '2016/06/30 09:59:11'. This demonstrates how a single code cell can produce multiple outputs when the 'multi outputs' feature is enabled.

# Jupyter-multi\_outputs - 特徴(3)

- .ipynbフォーマットをそのまま使用
- Extensionがない環境でも開くことは可能  
(タブにはなりません)

セル

タブ

タブ

```
{
  "cells": [
    {
      "cell_type": "code",
      "execution_count": 2,
      "metadata": {
        "collapsed": false,
        "multi_outputs": true
      },
      "outputs": [
        {
          "data": {
            "text/plain": [
              "'2016/06/29 04:53:34'"
            ]
          },
          "execution_count": 1,
          "metadata": {},
          "output_type": "execute_result"
        },
        {
          "data": {
            "text/plain": [
              "'2016/06/29 04:53:36'"
            ]
          },
          "execution_count": 2,
          "metadata": {},
          "output_type": "execute_result"
        }
      ],
      "source": [
        "from datetime import datetime\n",
        "datetime.now().strftime('%Y/%m/%d %H:%M:%S')"
      ]
    }
  ],
  "metadata": {
    ~~~ After that it is omitted ~~~
  }
}
```

Jupyter notebookでの運用は大変なこともあります、

# LiterateComputing

for Reproducible Infrastructure

試してみませんか？

PullRequestもお待ちしています