

LinuC レベル2 Version10.0

技術解説無料セミナー

2020/06/27 開催

主題	2.04 システムの保守と運用管理
副題	2.04.6 システム構成ツール

本日の講師



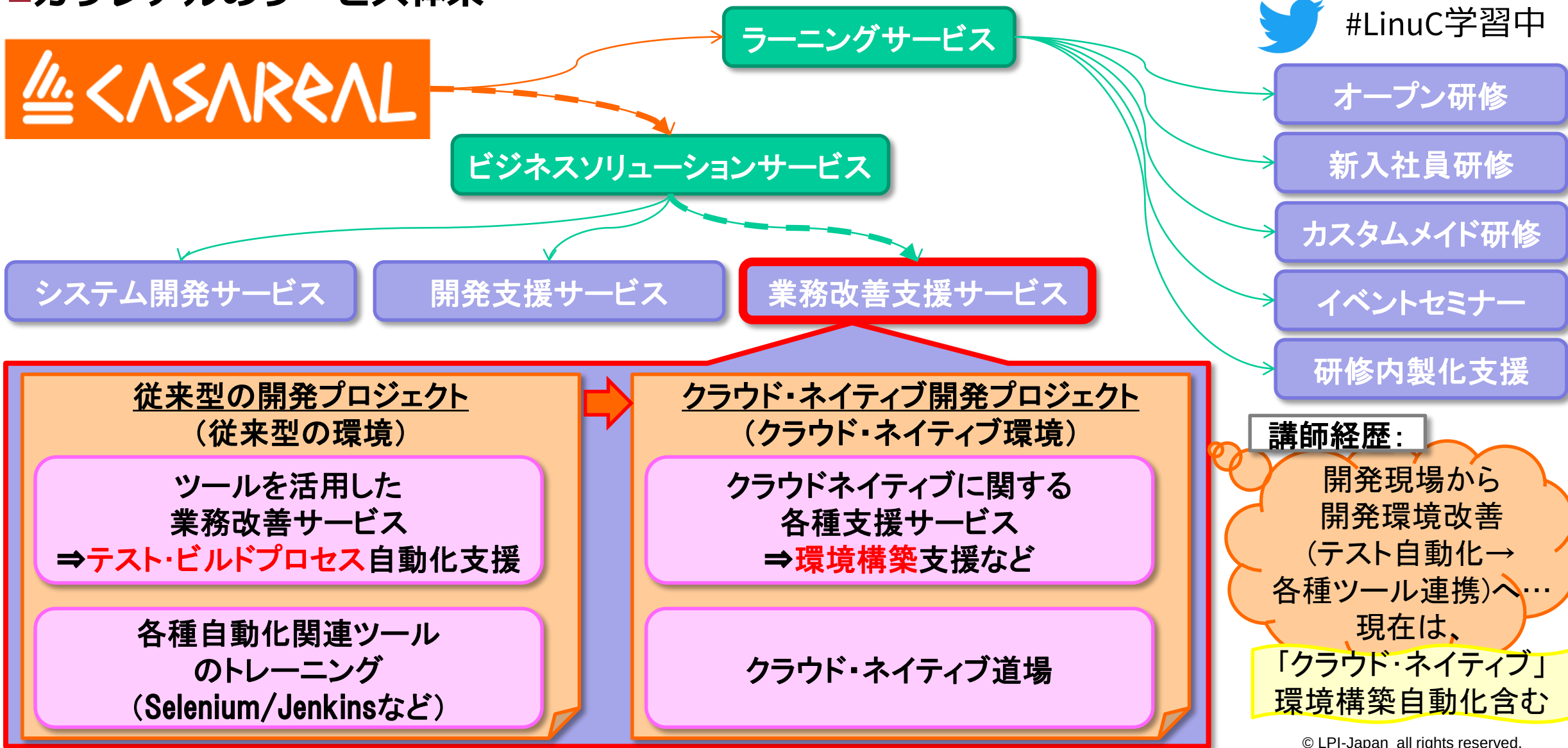
CASAREAL

株式会社 カサレアル (LPI-Japanアカデミック認定校)
植草 克友

LPI-JAPAN

© LPI-Japan / EDUCO all rights reserved.

■カサレアルのサービス体系





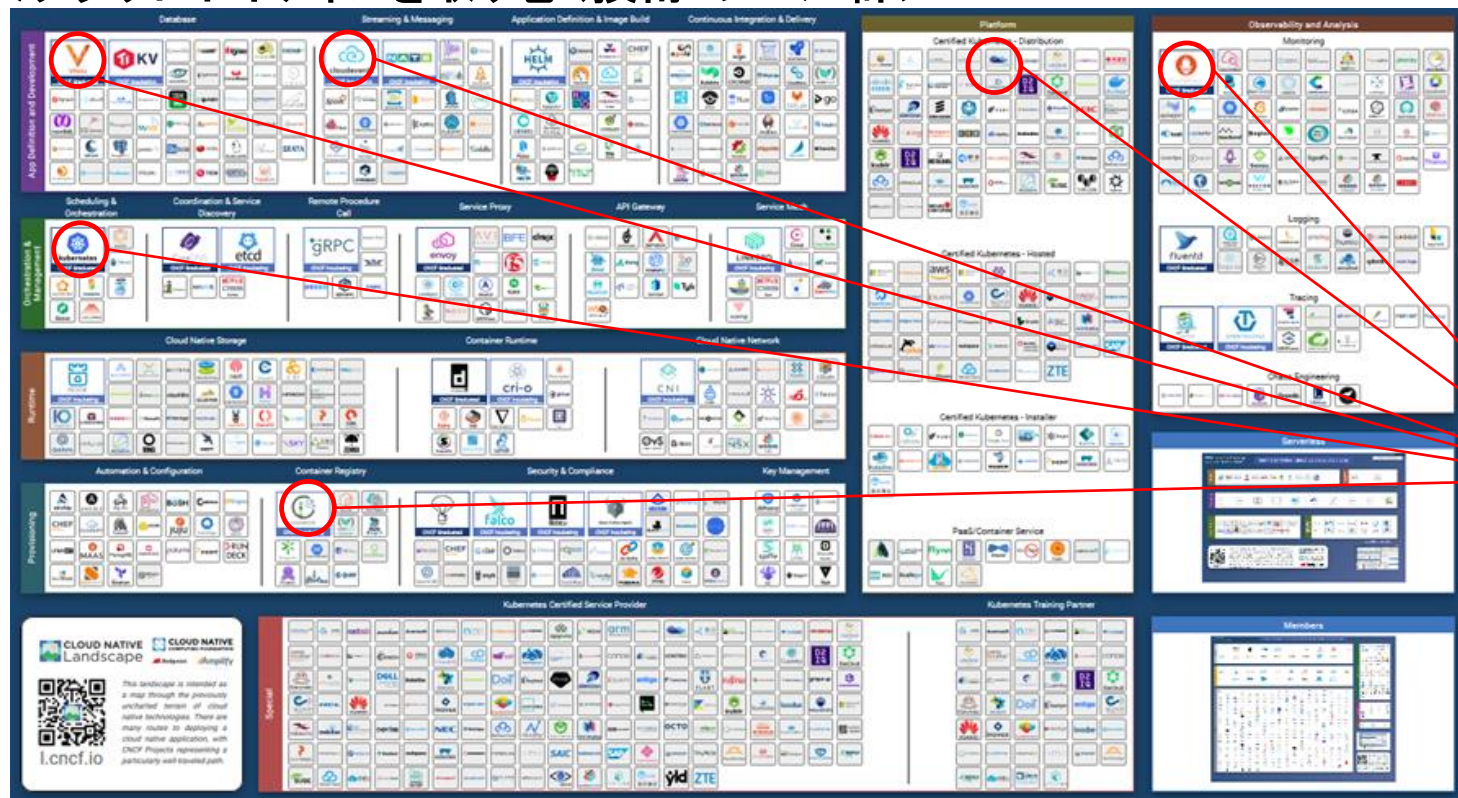
#LinuC学習中

■クラウド・ネイティブ道場

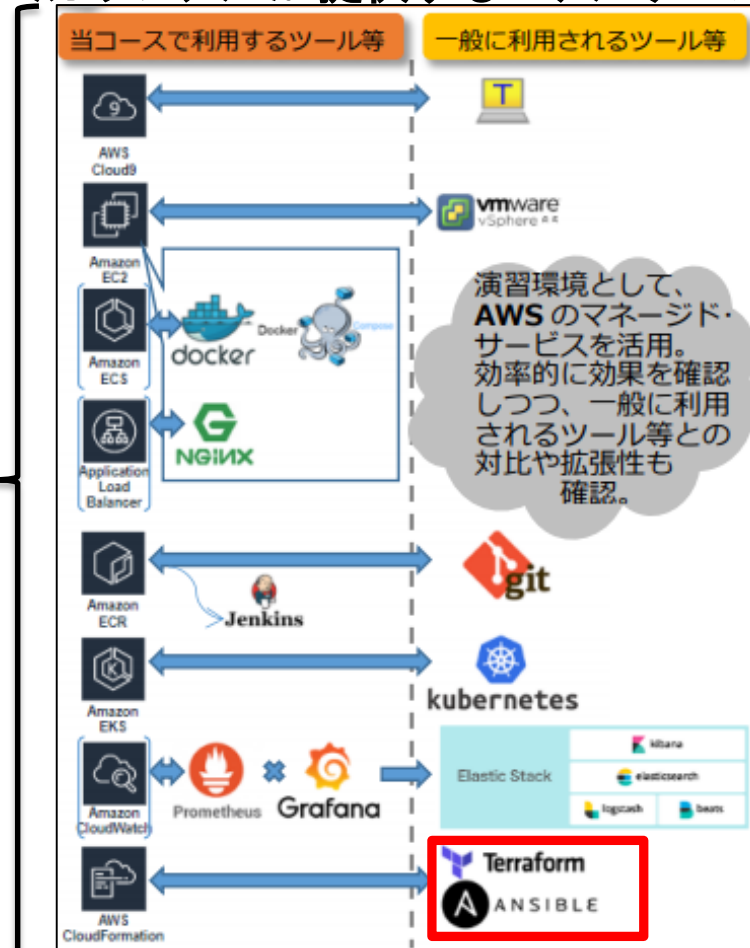
カサレアルが提供するモデルケースを通して、

クラウドネイティブの基礎的な考え方やスキルを身につけることができる

＜クラウドネイティブを取り巻く技術・ツール群＞



＜カサレアルが提供するモデルケース＞



1つ1つの技術領域・ツールの情報量が膨大！

・どこから手をつければ良いか？

・どの組み合わせが良いか？

↑「CNCF Landscape」
<https://landscape.cncf.io/>



■LinuCとは

クラウド時代の即戦力エンジニアであることを証明するLinux技術者認定資格

✓現場で「今」求められている新しい技術要素に対応

- ・ オンプレミス／仮想化を問わず様々な環境下でのサーバー構築
- ・ 他社とのコラボレーションの前提となるオープンソースへの理解
- ・ システムの多様化に対応できるアーキテクチャへの知見

✓全面的に見直した、今、身につけておくべき技術範囲を網羅

今となっては使わない技術やコマンドの削除、アップデート、新領域の取り込み

✓Linuxの範疇だけにとどまらない領域までカバー

セキュリティや監視など、ITエンジニアであれば必須の領域もカバー

■Version10.0と従来の出題範囲の比較

 #LinuC学習中

テーマ	Version 10.0	従来
LinuC-1	仮想技術 ・仮想マシン／コンテナの概念 ・クラウドセキュリティの基礎	← (Version10.0で新設)
	オープンソースの文化 ・オープンソースの定義や特徴 ・コミュニティやエコシステムへの貢献	← (Version10.0で新設)
	その他 → (Version10.0で削除)	アクセシビリティ、ディスククォータ、プリンタの管理、SQLデータ管理、他
LinuC-2	仮想化技術 ・仮想マシンの実行と管理(KVM) ・コンテナの仕組みとDockerの導入	← (Version10.0で新設)
	システムアーキテクチャ ・クラウドサービス上のシステム構成 ・高可用システム、スケーラビリティ、他	← (Version10.0で新設)
	その他 ・統合監視ツール(zabbix) ・自動化ツール(Ansible) → (Version10.0で削除)	← (Version10.0で出題範囲に含む) RAID、記憶装置へのアクセス方、FTPサーバーの保護、他



今回のテーマ

2.04 システムの保守と運用管理

↳ 2.04.6 システム構成ツール

↳ 自動化ツール(**Ansible**)

(Version10.0で出題範囲に含む)

「『Ansible』による環境構築の自動化」

I . 「Infrastructure
As Code」

II . Ansible とは…

III . Terraform との連携

概要

インベントリ、
プレイブックの説明

デモンストレーション

■「Infrastructure As Code」とは…

コードによるインフラストラクチャの構成と変更



#LinuC学習中

■背景(課題)

- ・インフラストラクチャの複雑化
 - 台帳管理（IT資産の構成、更新履歴など）
 - 実態と台帳の不整合（「冪等性」が確保されていない）
- ・適用対象の増加（ソフトウェアのアップデートなど）
 - 速やかに適用していく必要がある一方、適用後に正しく動作する事の確認も必要
 - 上記の適用および確認作業を、個別に手作業で実施することの限界

べきとうせい

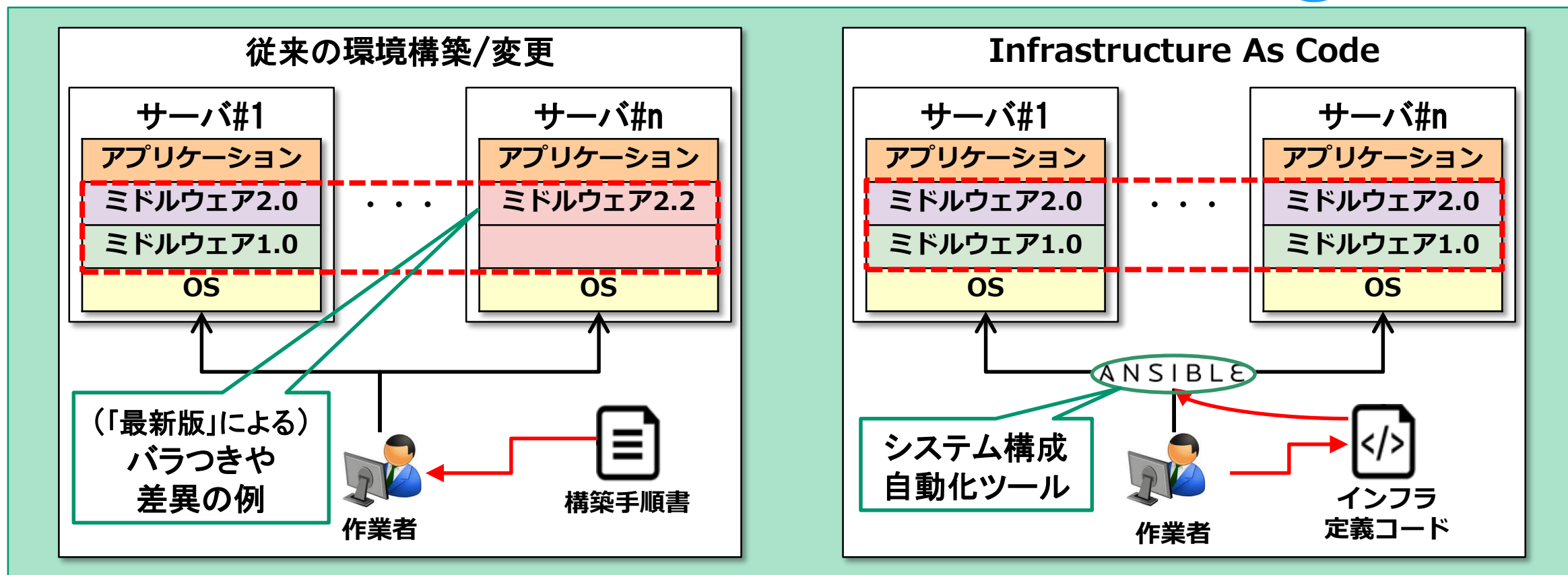
「冪等性」とは…

ある操作を何度繰り返し実行しても、結果は変わらないといった性質のこと

■対策のアプローチ

- ・台帳管理→構成や定義のコード化（※コードのバージョン管理が前提）
- ・人手作業→自動化

■従来の環境構築/変更と「Infrastructure As Code」の対比



• 「Infrastructure As Code」による環境構築/変更

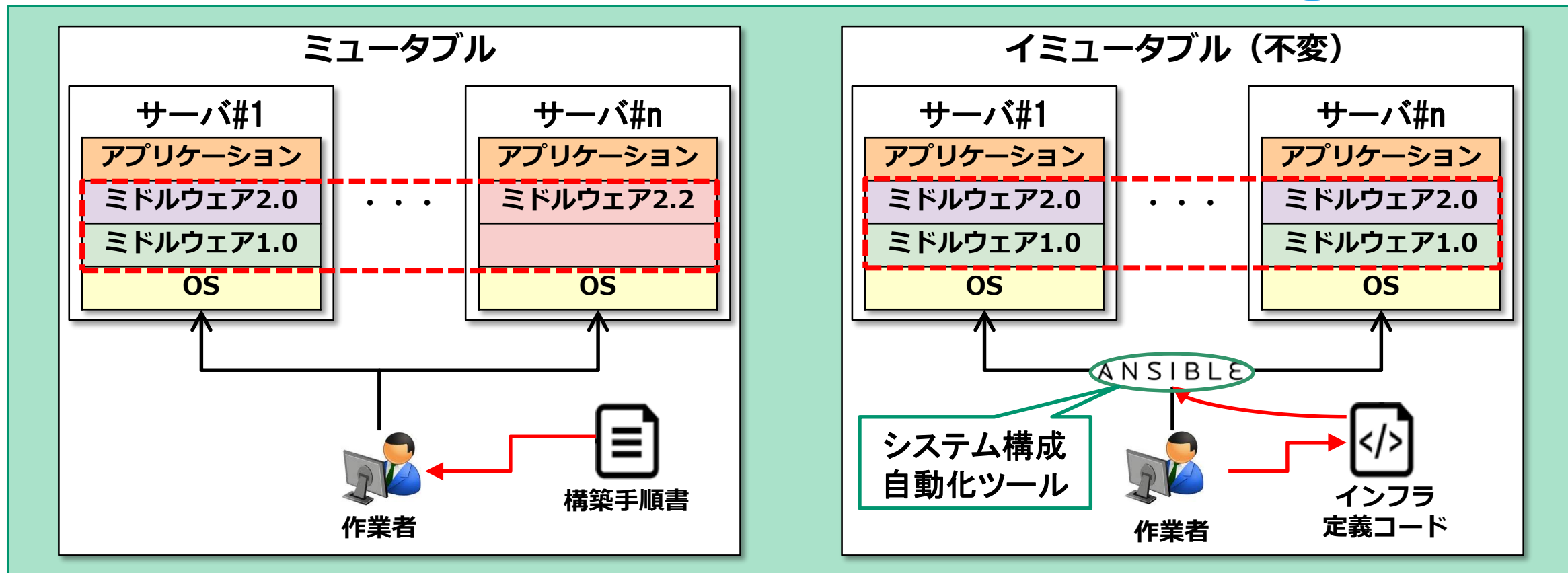
- コード化による実態(実際の構成や定義)と管理台帳間に生じる不整合の回避/低減
- 自動化によるヒューマンエラーの回避/低減

I. Infrastructure As Code

■ミュータブル・インフラストラクチャとイミュータブル・インフラストラクチャ



#LinuC学習中



- ミュータブル(無常)では、本番環境に更新やバージョンアップをかけ続ける。
- イミュータブル(不変)とは、本番環境に手を加えず、更新やバージョンアップの際は、破棄して新しい定義で再構築する。

■Ansible概要（GitHub Readmeより）



#LinuC学習中

Ansibleは、シンプルなIT自動化システムで、以下のような処理が可能

- 構成管理
- アプリケーション展開
- クラウドプロビジョニング
- アドホックタスク実行
- ネットワーク自動化
- マルチノードオーケストレーション

また、ロードバランサを使用したゼロダウンタイムのローリング更新など、複雑な変更を簡単に実現

■Ansible概要（GitHub Readmeより）



#LinuC学習中

設計の原則

- 簡潔なセットアッププロセスと最小限の学習曲線を提供すること
- マシンを並列かつ迅速に運用すること
- カスタムエージェントや追加のオープンポートを避け、既存のSSHデーモンを利用してエージェントレスにすること
- 機械と人間の両方に優しい言語でインフラストラクチャを表現できること
- セキュリティとコンテンツの監査/レビュー/書き換えの容易性を両立させること
- 再起動させることなく、新しいリモートマシンを簡単に追加できること
- Pythonに限定することなく、あらゆる動的言語でのモジュール開発を可能にすること
- root権限でなくても使用できること
- 今までにない最も使いやすいIT自動化システムにすること

■Ansible概要（まとめ）



#LinuC学習中

- Ansibleは構成管理ツール
パッケージ/ミドルウェアのインストールや設定、アプリケーションのデプロイ自動化が得意
- 最終的に実現したい構成に至る道筋を、1ステップずつ定義することでコード化(YAML)を実現
- SSH/winRMでサーバから接続にいくため、設定先へのエージェントのインストールが不要
- 豊富なモジュールが提供されているため、各種OSのサーバは勿論、ネットワーク機器やベンダー特有のソフトウェアの設定が可能
https://docs.ansible.com/ansible/latest/modules/list_of_all_modules.html
- 多くのクラウドサービスにも対応
- OSS（RedHat社がサポートを提供）

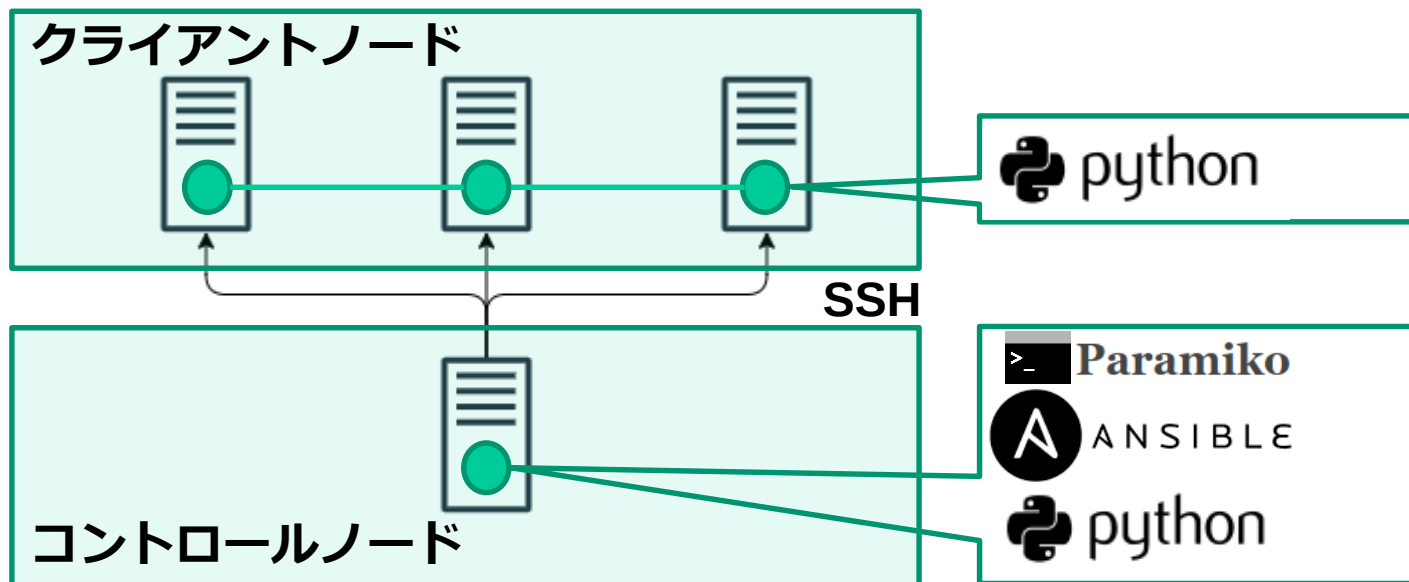
■Ansible概要（まとめ）



#LinuC学習中

Ansibleは構成管理ツール

- ・ パッケージ/ミドルウェアのインストールや設定
- ・ アプリケーションのデプロイ



■Ansible概要（インストール）



#LinuC学習中

●Ansibleの様々なインストール方法

- yumでのインストール
- pipでのインストール
- GitHub(ソースコード)からのインストール → rpmを作成してインストール

- AnsibleはPythonをベースに作られているため、
コントロールノード、クライアントノード双方にPythonのインストールが必要

https://docs.ansible.com/ansible/latest/installation_guide/intro_installation.html

- paramiko : PythonでSSH接続に、よく使用されるライブラリモジュール
<http://www.paramiko.org/>

■Ansible概要（インストール）



#LinuC学習中

デモ環境（AWS Cloud9）に、pipで「Ansible」をインストールした例

```
# pip install ansible

# ansible --version

# pip install paramiko
```

■デモ



#LinuxC学習中

Ansibleのインストール例として、

実際にpipで「Ansible」をインストールします。

■Ansible概要（初期設定ファイル）



#LinuC学習中

●設定するパラメータの例

パラメータ	デフォルト値	内容
forks	5	ターゲットノードへの並列処理を行うプロセス数を設定
og_path	-	Ansible実行コマンドログの配置場所
host_key_checking	True	ターゲットノードへSSH接続を行う際の、フィンガープリントチェックを行う

※実際には運用に合わせて適切な値に設定することが大切

https://docs.ansible.com/ansible/latest/reference_appendices/config.html

●設定ファイルの読み込み優先順位

1. 環境変数に設定されたファイルパス（例:ANSIBLE_CONFIG=/usr/local/ansible/.ansible.cfg）
2. カレントディレクトリに存在する設定ファイル（./ansible.cfg）
3. ホームディレクトリに存在する設定ファイル（\$HOME/.ansible.cfg）
4. /etc/ansible/ansible.cfg

■Ansible概要（初期設定ファイル）



#LinuxC学習中

例： `$HOME/.ansible.cfg`

```
[defaults]
```

```
forks = 15
```

```
log_path = $HOME/.ansible/ansible.log
```

```
host_key_checking = False
```

■Ansible概要 (ansible コマンド)



#LinuC学習中

ansible -h

```
usage: ansible [-h] [--version] [-v] [-b] [--become-method BECOME_METHOD]
               [--become-user BECOME_USER] [-K] [-i INVENTORY] [--list-hosts]
               [-I SUBSET] [-P POLL_INTERVAL] [-B SECONDS] [-o] [-t TREE] [-k]
               [--private-key PRIVATE_KEY_FILE] [-u REMOTE_USER]
               [-c CONNECTION] [-T TIMEOUT]
               [--ssh-common-args SSH_COMMON_ARGS]
               [--sftp-extra-args SFTP_EXTRA_ARGS]
               [--scp-extra-args SCP_EXTRA_ARGS]
               [--ssh-extra-args SSH_EXTRA_ARGS] [-C] [--syntax-check] [-D]
               [-e EXTRA_VARS] [--vault-id VAULT_IDS]
               [--ask-vault-pass | --vault-password-file VAULT_PASSWORD_FILES]
               [-f FORKS] [-M MODULE_PATH] [--playbook-dir BASEDIR]
               [-a MODULE_ARGS] [-m MODULE_NAME]
               pattern
```

Define and run a single task '**playbook**' against **a set of hosts**

■Ansible概要 (ansible コマンド)



#LinuC学習中

ansible localhost -m ping

[WARNING]: No inventory was parsed, only implicit localhost is available

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

localhostにpingモジュールで
ping を打つと
変更なしで
応答 (pong) が返ってきた
ことが分かる

■モジュール (1)



#LinuxC学習中

- 数千に及ぶ多彩なモジュール群から、代表的なモジュールを紹介

➤ **yum** : yumコマンドを利用してパッケージをインストールする

パラメータ	選択肢	説明
name	-	パッケージ名
state	present	インストール済みならそのまま
	latest	新しいのがあればupgrade
	absent	パッケージ削除

➤ **git** : git clone の処理を行う

パラメータ	選択肢	説明
repo		リポジトリ(ssh,https等が利用可能)
dest		clone先
depth		履歴取得数 最低1
update	yes no	yes(デフォルト値)の場合、pullも行う

■モジュール (2)



#LinuC学習中

➤ **get_url** : http経由でファイルをダウンロードする

パラメータ	選択肢	説明
url		DLしたいファイルのurl
dest		ファイルの保存先
others		その他fileモジュール同様のパラメータを指定可能

➤ **command** : shell コマンドを実行する (<, >, | を使用する場合は、shell の方を使う)

パラメータ	選択肢	説明
chdir		コマンドの実行前に、指定ディレクトリへcdする
creates		指定のファイル名が存在する場合、commandの 実行をスキップ する
removes		指定のファイル名が存在しない場合、commandの 実行をスキップ する

■モジュール (3)



#LinuC学習中

➤ **file** : ファイル、シンボリックリンク、ディレクトリの属性設定・削除を行う

パラメータ	選択肢	説明
path		ファイルのパス
mode		パーミッション設定
owner		所有者
group		グループ
recurse	yes/no	再帰 (state=directory時のみ有効)
src		state=link時のみ
state	file/link/directory/hard/absent	

➤ **copy** : ローカル端末のfilesディレクトリにあるsrcのファイルを、リモートのdestにコピーする

パラメータ	選択肢	説明
src		コピー元 (ローカル)
dest		コピー先 (リモート)
others		fileモジュールのパラメータ (owner,group,mode等)が指定可能

■Ansible概要（Ansibleの実行）

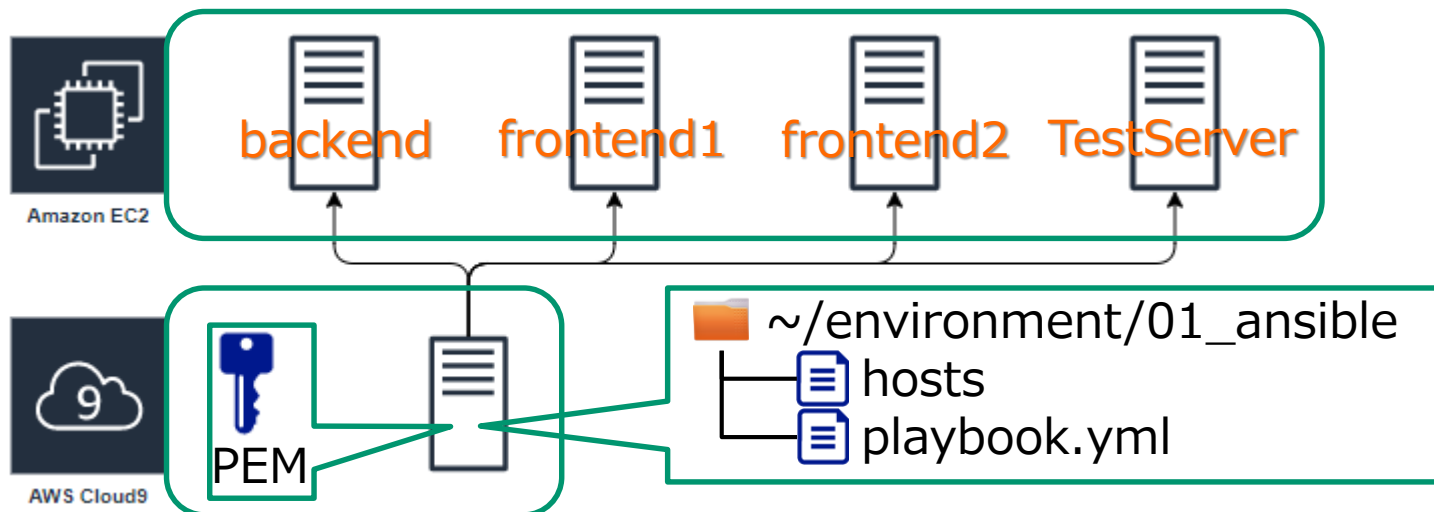


#LinuC学習中

●実行手順

STEP	概要	説明
1	ワークスペース作成	ワークスペースとして、ansible実行用のディレクトリを作成 → <code>~/environment/01_ansible</code>
2	インベントリファイル作成	上記ワークスペース内に、「hosts」ファイルを作成 → 次ページのように編集 → コマンドを叩いて動作を確認
3	プレイブック作成	上記ワークスペース内に、「playbook.yml」を作成 → 次の次ページのように編集 → コマンドを叩いて動作を確認

●デモの環境



■インベントリ



#LinuxC学習中

●「Hosts」ファイルの内容

```
[backend_servers]
backendのIPアドレス(54.250.30.209)

[frontend_servers]
frontend1のIPアドレス(3.112.243.155)
frontend2のIPアドレス(13.113.192.165)
```

※グループ化や変数化が可能

●確認

```
cd ~/environment/01_ansible

ansible -i hosts backend_servers
--private-key=~/.ssh/EC2KeyPair.pem
-u ec2-user -m ping
```

●結果

```
54.250.30.209 | SUCCESS => {
  "ansible_facts": {
    "discovered_interpreter_
      python": "/usr/bin/python"
  },
  "changed": false,
  "ping": "pong"
}
```

■プレイブック



#LinuC学習中

●「playbook.yml」ファイルの内容

```
---
- hosts: all
  tasks:
    - name: ping to servers
      ping:
        data: "pong"
```

●確認

```
cd ~/environment/01_ansible

ansible-playbook -i hosts playbook.yml
--private-key=~/.ssh/EC2KeyPair.pem
-u ec2-user -c paramiko
```

-vvvv : 詳細表示

実行結果	ステータス	結果内容
ok	成功	すでに定義された状態となっているため、処理を実行しなかった
changed	成功	タスクで指定したステータスと異なっていたため、変更を行った
skip	成功	タスクの実行条件に当てはまらなかったため、処理を行わなかった
unreachable	失敗	ターゲットノードに接続ができなかった
failed	失敗	何らかのエラーにより定義された状態にならなかった

■デモ



#LinuC学習中

Ansibleの基本的なコマンドの動き

```
ansible -h
```

```
ansible localhost -m ping
```

```
ansible -i hosts backend_servers --private-key=~/.ssh/EC2KeyPair.pem  
-u ec2-user -m ping
```

```
ansible -i hosts frontendend_servers --private-key=~/.ssh/EC2KeyPair.pem  
-u ec2-user -m ping
```

■サーバーのプロビジョニング（デモ：httpdのインストールとスタート）



#LinuC学習中

●「playbook.yml」ファイルの内容

```
---
- hosts: frontend_servers
  become: yes
  tasks:
    - name: Install httpd
      yum:
        name: httpd
        state: present
      notify: start httpd
  handlers:
    - name: start httpd
      service:
        name: httpd
        enabled: true
        state: started
```

●実行

```
cd ~/environment/01_ansible
```

```
ansible-playbook -i hosts playbook.yml
--private-key=~/.ssh/EC2KeyPair.pem
-u ec2-user -c paramiko
```

●実行前後での確認

実行前：このサイトにアクセスできません

実行後：「TestPage」が表示される

```
http://3.112.243.155/
```

```
http://13.113.192.165/
```

■デモ



#LinuC学習中

Ansibleによるプロビジョニングの動き

frontend の2台に、httpd がインストールされていない事を確認

→ [http://\(frondendのIPアドレス\)](http://(frondendのIPアドレス))で、http接続が拒否される

Ansibleで frontend の2台に、httpdのインストールとスタートを実施

```
ansible-playbook -i hosts playbook.yml --private-key=~/.ssh/EC2KeyPair.pem -u ec2-user -c paramiko
```

frontend の2台に、httpd がインストールされた事を確認

→ [http://\(frondendのIPアドレス\)](http://(frondendのIPアドレス))で、httpdのテストページが表示される

■Terraform概要



#LinuC学習中

**Terraformは、HashiCorp社が開発する
インフラ構築や設定をコード(テンプレートファイル)を使って自動化するツール**

以下のような複数のプロバイダー(クラウドサービスやツール)に対応

- AWS
- Azure
- CloudStack
- GoogleCloud
- Docker
- OpenStack ほか

**建築に例えるなら、上物(内装)を得意とする「Ansible」に対して、
「Terraform」は基礎工事を得意とする**

■Terraform概要（インストール）



#LinuC学習中

デモ環境（AWS Cloud9）に、pipで「Terraform」をインストールした例

1. Terraformのバージョンマネージャ「tfenv」をインストール

```
git clone https://github.com/tfutils/tfenv.git ~/.tfenv
echo 'export PATH="$HOME/.tfenv/bin:$PATH"' >> ~/.bash_profile
source ~/.bash_profile
tfenv --version
```

2. 現在インストール可能なTerraformのバージョン一覧を出力

```
tfenv list-remote
```

3. Terraformをインストール

```
tfenv install 0.12.26
tfenv use 0.12.26
terraform --version
```

■デモ



#LinuC学習中

Terraformのインストール

Terraformのバージョンマネージャ「tfenv」をインストール

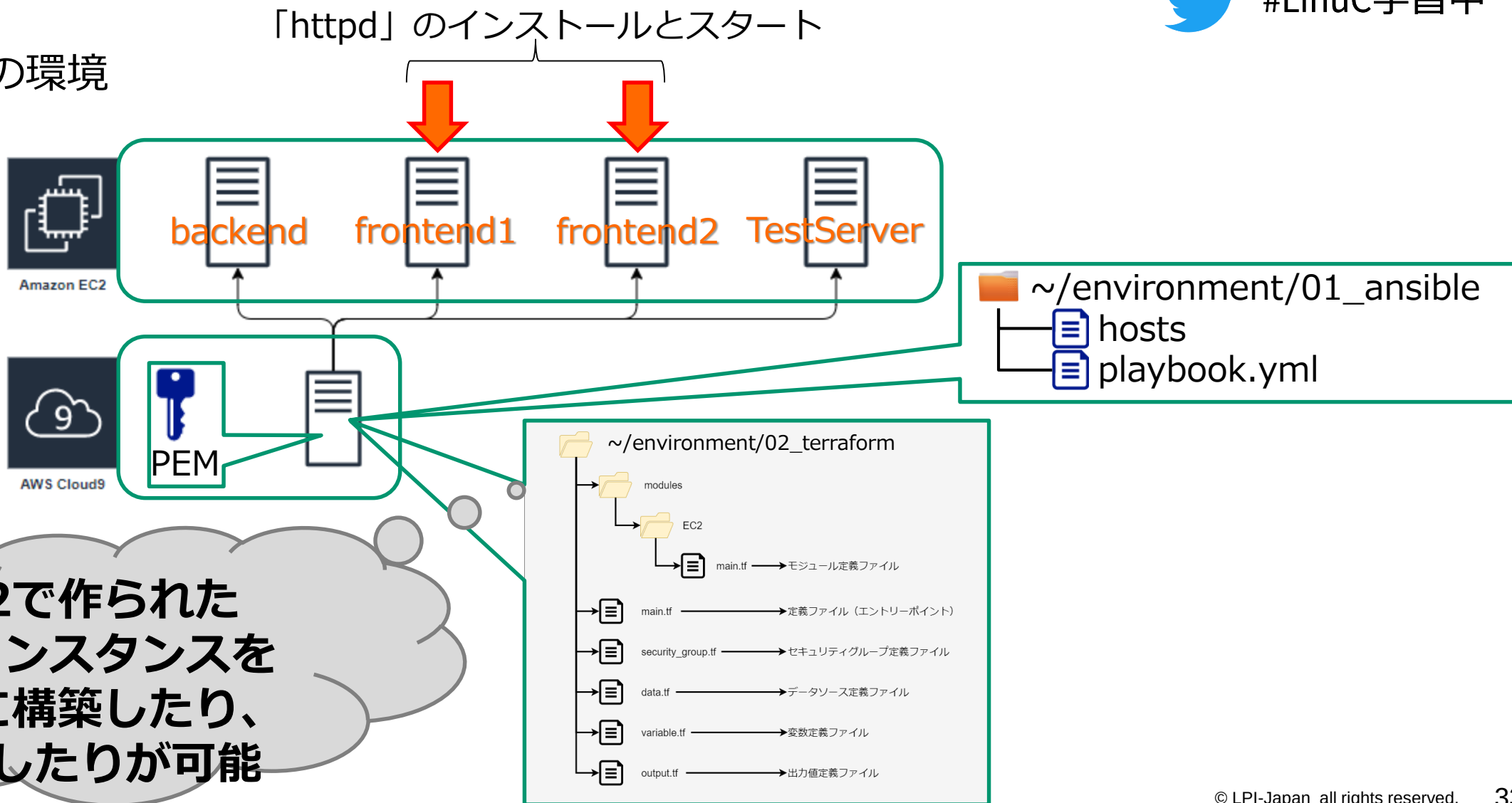
現在インストール可能なTerraformのバージョン一覧を出力

Terraformをインストール

■今回のデモ環境



●デモの環境



■ Terraform概要 (テンプレート)



#LinuC学習中

~/environment/02 terraform/main.tf

```
git clone https://github.com/tfutils/tfenv.git ~/.tfenv
echo 'export PATH="$HOME/.tfenv/bin:$PATH"' >> ~/.bash_profile
source ~/.bash_profile
tfenv --version### プロバイダー指定
provider "aws" {
  region = var.region
}
```

EC2インスタンス作成 (バックエンド)

```
module "Backend" {
  source      = "./modules/EC2"
  ami         = data.aws_ami.recent_amazon_linux_2.image_id
  key_name    = var.key_name
  security_group_id = aws_security_group.EC2_APP_SG.id
  name       = "Backend"
}
```

EC2インスタンス作成 (フロントエンド)

```
module "Frontend1" {
  source      = "./modules/EC2"
  ami         = data.aws_ami.recent_amazon_linux_2.image_id
  key_name    = var.key_name
  security_group_id = aws_security_group.EC2_APP_SG.id
  name       = "Frontend"
}
```

```
module "Frontend2" {
  source      = "./modules/EC2"
  ami         = data.aws_ami.recent_amazon_linux_2.image_id
  key_name    = var.key_name
  security_group_id = aws_security_group.EC2_APP_SG.id
  name       = "Frontend"
}
```

EC2インスタンス作成 (テストサーバー)

```
module "Test_Server" {
  source      = "./modules/EC2"
  ami         = data.aws_ami.recent_amazon_linux_2.image_id
  key_name    = var.key_name
  security_group_id = aws_security_group.EC2_APP_SG.id
  name       = "Test_Server"
}
```

■Terraform概要 (テンプレート)



#LinuC学習中

~/environment/02 terraform/security_group.tf

```
### セキュリティグループ作成
resource "aws_security_group" "EC2_APP_SG" {
  name      = "EC2_APP_SG"
  description = "EC2_APP_SG"

  ### インバウンドルール設定
  dynamic "ingress" {
    for_each = [22, 80, 3000]

    content {
      from_port = ingress.value
      to_port   = ingress.value
      protocol  = "tcp"
      cidr_blocks = ["0.0.0.0/0"]
    }
  }

  ### アウトバウンドルール設定
  egress {
    from_port = 0
    to_port   = 0
    protocol  = "-1"
    cidr_blocks = ["0.0.0.0/0"]
  }
}
```

data.tf

```
### EC2インスタンス作成に使用するAMI初期値
data "aws_ami" "recent_amazon_linux_2" {
  most_recent = true
  owners      = ["amazon"]

  filter {
    name = "name"
    values = ["amzn2-ami-hvm-2.0.?????????.?-x86_64-gp2"]
  }

  filter {
    name = "state"
    values = ["available"]
  }
}
```

■Terraform概要（テンプレート）



#LinuC学習中

~/environment/02 terraform/security_group.tf

```
### リージョン初期値指定
variable "region" {
  type = string
  description = "Enter Region"
  default = "ap-northeast-1"
}

### EC2キーペア名格納用変数
variable "key_name" {
  type = string
  description = "Enter SSH Security Key Name"
  default = "EC2KeyPair"
}
```

output.tf

```
output "Backend_public_ip" {
  value = module.Backend.public_ip
}

output "Frontend1_public_ip" {
  value = module.Frontend1.public_ip
}

output "Frontend2_public_ip" {
  value = module.Frontend2.public_ip
}

output "Test_Server_public_ip" {
  value = module.Test_Server.public_ip
}
```

■Terraform概要（テンプレート）



#LinuxC学習中

~/environment/02 terraform/modules/EC2/main.tf

```
### EC2キーペア名格納用変数
variable "ami" {}
variable "key_name" {}
variable "security_group_id" {}
variable "name" {}
```

```
### EC2インスタンス作成
resource "aws_instance" "EC2_NODE" {
  ami          = var.ami
  instance_type = "t2.micro"
  key_name     = var.key_name
  vpc_security_group_ids = [
    var.security_group_id
  ]
  associate_public_ip_address = "true"
  root_block_device {
    volume_type = "gp2"
    volume_size = "20"
  }
  tags = {
    Name = var.name
  }
}
```

```
### バックエンド作成後のパブリックIPアドレスを出力
output "public_ip" {
  value = aws_instance.EC2_NODE.public_ip
}
```

■基盤構築



#LinuC学習中

初期化

```
cd ~/environment/02_terraform
terraform init
```

テンプレートの作成

```
cd ~/environment/02_terraform
terraform plan
```

構築

```
cd ~/environment/02_terraform
terraform apply
```

インスタンスの作成 ▼ 接続 アクション ▼

Q タグや属性によるフィルター、またはキーワードによる検索

Name	インスタンス ID	インスタンスタイプ	アベイラビリティ	インスタンスのステータス	ステータスチェック	アラーム
aws-cloud9-AnsibleWorkShop-7...	i-023bc2f10390653cf	t3.small	ap-northeast-1d	running	2/2 のチェックに...	なし
Backend	i-09ca38056acac04a4	t2.micro	ap-northeast-1a	running	2/2 のチェックに...	なし
Frontend	i-00493731bbf768582	t2.micro	ap-northeast-1a	running	2/2 のチェックに...	なし
Frontend	i-0c4b1aa395962f06b	t2.micro	ap-northeast-1a	running	2/2 のチェックに...	なし
Test_Server	i-0997f0ae8024e17a1	t2.micro	ap-northeast-1a	running	2/2 のチェックに...	なし

実行確認が求められるので、
「yes」と入力

■基盤の破棄



#LinuC学習中

破棄

```
cd ~/environment/02_terraform
terraform destroy
```

実行確認が求められるので、
「yes」と入力

☒ New EC2 Experience
Tell us what you think

EC2 ダッシュボード New

イベント New

タグ

レポート

制限

▼ インスタンス

インスタンス

インスタンスタイプ

インスタンスの作成 ▼							
接続 アクション ▼							
🔍 タグや属性によるフィルター、またはキーワードによる検索							
☐	Name	インスタンス ID	インスタンスタイプ	アベイラビリティ	インスタンスの状態	ステータスチェック	アラート
☐	aws-cloud9-AnsibleWorkShop-7...	i-023bc2f10390653cf	t3.small	ap-northeast-1d	🟢 running	✅ 2/2 のチェックに...	なし
☐	Backend	i-09ca38056acac04a4	t2.micro	ap-northeast-1a	🔴 terminated		なし
☐	Frontend	i-00493731bbf768582	t2.micro	ap-northeast-1a	🔴 terminated		なし
☐	Frontend	i-0c4b1aa395962f06b	t2.micro	ap-northeast-1a	🔴 terminated		なし
☐	Test_Server	i-0997f0ae8024e17a1	t2.micro	ap-northeast-1a	🔴 terminated		なし

■デモ



#LinuC学習中

デモ環境の破棄

Ansibleのプロビジョニングを確認した環境（AWSのEC2インスタンス）を破棄
→次回、同じ環境をTerraformで再構築可能

これにより、何回でも繰り返し、同じ環境を構築することが可能

本スライドは、以上です。