CnfBox: OS自動インストール設定の対話型入力と統一記法

小山 智之[1,a)]　串田 高幸[1]

概要：同時に多数の Linux マシンをセットアップする状況で，OS インストールの自動化は必要不可欠である．しかし，それらを実現するために使うツールや手順は OS やツールごとに異なる．そのため，これらそれぞれを理解する負担がオペレータに生じる．本研究では，これら課題を設定支援ツール CnfBox で解決した．Linux ディストリビューションを対象に自動インストール手順と設定投入を共通化した．プログラム CnfBox を作成し，実験環境に作成した仮想マシンでの実験を行った．さらに既存技術との比較を行った．その結果，設定のコスト削減がみられた．

1. はじめに

クラウド技術の普及によりこれまで以上にマシンのライフサイクルが活発な状況にある．クラウド技術を使うことで，従来型のオンプレミスによる課題が解消されている．例えば，ハードウェアの管理やそれに伴うソフトウェアの管理があげられる．ハードウェアはその性質上，故障や耐用年数による定期的な交換が発生する．オンプレミスではこれに対応するため，エンジニアの雇用や保守契約の費用が発生していた．また，ハードウェアの交換に伴いソフトウェア設定の変更が発生する [1]．クラウド技術はこうした付加価値を生まない作業にかかるコストを削減する．

クラウド技術の構築・運用を行う立場では，複数台のマシンをセットアップする状況は依然として発生する．特に提供サービスの規模が拡大するたびにマシンの台数は線形に増加する．これらマシンは必ず同一の環境 (ハードウェアスペックや OS) とは限らず，マシンが増えるにつれて個々のマシンの性能や環境のばらつきが生じる．システムが大規模化するにつれ，設定のばらつきは拡大する [2]．それに伴い設定のコストは増加する．特に，これらマシンのセットアップを個別に手動で行う場合はよりコストが高まる．個々に異なるパラメータをオペレータが手動で設定する場合，設定パラメータの種類と設定パターンが増えるにつれて設定不備の発生する可能性が高まる．設定不備は不正アクセスを許し，セキュリティホールにもつながる [3]．コスト削減とセキュリティ対策の観点から，オペレーションの自動化は必要不可欠である．

こうした自動化を実現するツールには，ベンダーが提供するものがある．これらは，自作やサードパーティ製ツールとの拡張性の低さや自動化しにくい GUI，一部で手動作業が必要となる不完全さの問題を抱えている．オペレータは，システムのインストールやアップデートのたびにこうした課題と向き合う必要がある [4]．つまり，ベンダーの製品では十分に柔軟な自動化が実現できない状況にある．

大規模システムを運用するオペレータほど，自動化ツールを独自で開発し，修正を繰り返しながら運用を行う．こうしたツールは，システムごとに作成されるため，汎用性がない．また，こうしたツールは処理内容を把握した作成者のみ知る，属人化の温床となる恐れがある [5]．つまり，ツールは簡単なインストールや将来の変更やシステム独自のカスタマイズを行える簡単な仕組みである必要がある [6]．しかし，こうした条件を十分に満たし，十分に利用できるツールや仕組みはあまりない．

マシンのセットアップにおける作業はいくつかある．本研究は，その中でも OS のインストールに着目した．その理由は，著者自身が複数台の仮想マシンへの OS 自動インストールにおいて，設定フォーマットの違いやサーバ構築で課題に直面した為である．従来から OS インストールを自動化する取り組みは行われてきた．これら自動化の手法は，大別すると以下の 2 つになる [7]．

(1) 事前に用意したインストール済みイメージを複製
(2) 事前に用意した設定に基づく OS インストールの自動化

(1) の手法は，イメージを単純にコピーするだけであるため，インストールする OS を問わない．それゆえ，多くの OS の自動インストールで使用可能である．一方で，個々のマシンごとに柔軟な設定が難しい．個々のマシンごとに異なるパラメータを修正してイメージを作成するため，マ

[1] 東京工科大学コンピュータサイエンス学部
CDSL, TUT, Hachiouji, Tokyo 101–0062, Japan
[a)] C0117123

シンの種類が増えるにつれてイメージ数も増える．そのため，この手法は同一のスペックや設定のマシンを対象に OS 自動インストールをする場合には有効である．

(2) の手法は，マシンごとに柔軟に設定ができるため，パラメータが異なる複数のマシンへの自動インストールが (1) の手法に比べ容易に行える．一方で，OS やツールごとに設定記法や手順が異なるため，(2) に比べ汎用的に使用可能な手法ではない．

本研究では，(2) の手法が (1) に比べ汎用的な利用が可能であることから，(2) の手法における課題を設定支援ツール CnfBox の提案により解決する．

2 章では関連研究を取り上げる．関連研究の課題をふまえ 3 章では新たな提案を示す．4 章および 5 章では，提案をもとに実装を行う．実装をもとに 6 章では評価を行う．7 章および 8 章は議論と結論，今後の課題を示す．

2. 関連研究

OS のセットアップ（インストールや設定）を自動化する取り組みは，独自にツールを開発する手法や既存ツールをカスタマイズする手法が用いられてきた．

独自にツールを開発する手法は，既存の技術との互換性を考慮する必要がなく自由度が高まる．課題は初期開発時や継続的なメンテナンスにコストが発生することである．

Anderson らは数百台のマシンを対象に自動による Linux インストールと設定を行うシステム ”LCFG” を提案した [3]．このシステムでは，Solaris Jumpstart を Bootstap として使用している．これにより，OS ごとに異なる初期の作業手順の統一を試みた．また，設定をオブジェクト指向形式 (コード 1) で表現することで，ハードウェアごとに異なるパラメータの差分の吸収を提案した．

コード 1 Anderson らの設定フォーマット

```
auth.consolepermclass_cdrom ANDALSO(/dev/scd0)
```

高宮らは大規模クラスタを対象に高速なセットアップと管理を実現するツール ”Lucie” を提案した [8]．この取り組みでは，既存ツール (ISC DHCP, Kickstart) と独自ツール Lucie を使い，250 台のマシンへ OS インストールとソフトウェアパッケージの導入を行うことで測定を行った．Lucie では独自の設定フォーマット 2 の設定ファイルを NFS サーバへ配置し，それを Lucie クライアントが取得して自動的にインストールを行う．

既存ツールのカスタマイズによる手法は，新たな開発によるコストを抑えられ簡単に実装できる．課題は，既存ツールに依存するため自由度が低いことである．

Cristian らはシステム管理ツール Puppet と自動インストールツール Kickstart，DHCP サーバ ISC DHCP の利用により，Linux インストールと設定の自動化を提案した [9]．

コード 2 高宮らの設定フォーマット

```
define host { # ホスト cnode00 の定義
    host_name address mac_address use
    cnode00 192.168.10.1 00:50:56:40:40:b6
        hosttemplate
}
```

この手法では，DHCP サーバの設定を Puppet により生成し，OS の自動インストールを Kickstart でおこなった．さらに，ソフトウェアパッケージの導入を Puppet により実施した．

小久保らは既存ソフトウェア (Kickstart, ISC DHCP) と独自のプログラムにより自動インストールの高速化を提案した [10]．彼らは，設定する対象マシンごとの設定ファイルの生成をプログラムで並列に行った．これにより，従来のツールに比べ高速な設定を実施した．

これまで述べた関連研究の手法を表 1 に示す．

表 1 既存手法の比較

機能	Anderson[3]	高宮 [8]	Cristian[9]	小久保 [10]
OS 自動インストール	○	○	○	○
RedHat 系対応	○	○	○	○
Debian 系対応	×	×	×	×
設定記法	独自	独自	Puppet	不明

3. 提案

本研究で提案するアーキテクチャを図 1 に示す．図の緑色の箇所を新たに作成する．アーキテクチャは，オペレータの操作する CLI プログラム (CnfBox CLI) とデータを処理するサーバプログラム (CnfBox SRV) から構成される．また，CnfBox SRV は独自の統一設定ファイルにより設定を管理する．

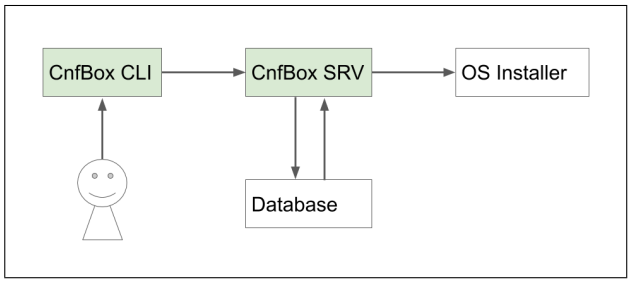

図 1 インストールの流れ

3.1 統一設定フォーマット

従来の手法では，設定の記述にオブジェクト指向記法や関数記法を採用している [3], [8]．こうした独自のフォーマットは，オペレータの新たな学習コストが発生する．ま

た，プログラムによりフォーマットを解釈する場合に，独自でパーサープログラムを作成する必要がある．さらに，Kickstart と preseed は異なる 2 つのツール間には設定フォーマットの互換性がない．そのため，同一の設定の記述には 2 倍の工数がかかる．それゆえ，一方のフォーマットを他方のフォーマットに変換するにも相互の設定パラメータへの理解が必要である．

こうしたツールごとの設定ファイル構文の差分を吸収するため，本提案では記述言語として Python を採用した．Python は豊富なライブラリや簡潔に記述できる構文，インスタンス変数への容易なアクセス（アクセサが標準機能にある）があるため採用した．また，Python プログラムを作成する経験があれば独自フォーマットに比べ容易に取り扱える．さらに，プログラミング言語 Python の標準機能を使うことで，変数（時刻や乱数）や基本構文（繰り返し，条件分岐），組み込み関数を利用できる．そのため，従来の設定フォーマットに比べより柔軟な表現が可能になる．

3.2 CnfBox CLI: 設定インターフェース

設定インターフェースを提供するクライアントプログラム CnfBox CLI を提案する．アーキテクチャを図 2 に示す．このプログラムはオペレータのための設定インターフェースを提供する．設定を送信するコマンドを実行することで，CnfBox SRV へオペレータの投入した設定を送信する．プログラムは 2 つの使い方がある．第一は，事前にテキストエディタで編集した設定ファイルを読み込ませる方法である．これは，従来の手法でも採用されている [3], [8], [9]．第二は，ターミナルによる対話型の設定方法である．ターミナルインターフェースに設定パラメータを入力することで設定を投入する．設定パラメータはオブジェクト指向によるリソース指定ができる．さらに，タブによる入力補完によりパラメータを覚えずとも設定ができる．

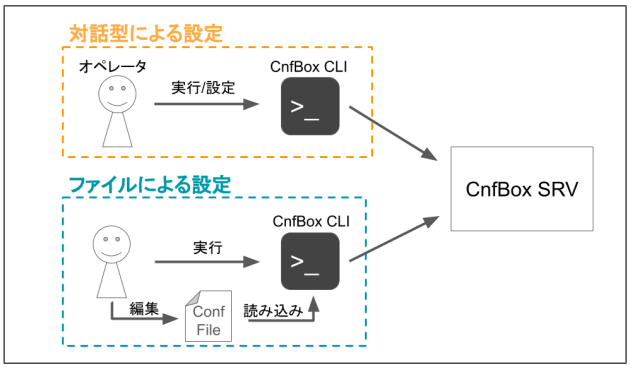

図 2　CnfBox CLI アーキテクチャ

3.3 CnfBox SRV: 変換サーバ

設定ファイルの生成，公開を行うサーバープログラム CnfBox SRV を提案する．このプログラムは次の 2 つの機能をもつ．アーキテクチャを図 3 に示す．

変換機能

CnfBox SRV は CnfBox CLI から設定（例：ネットワーク，OS，認証情報）を受け取る．受け取ったデータは，データベースサーバで自動生成される ID とともにデータベースで保存する．また，プログラムにより解析してデータ構造へ変換する．これを事前にプログラム内に定義した各設定ファイルのフォーマットと個々に照らし合わせて変換を行う．

配信機能

生成した設定ファイルをデータベースから取得し，内蔵する HTTP サーバで配信する．このサーバでは RESTful API を使用する．

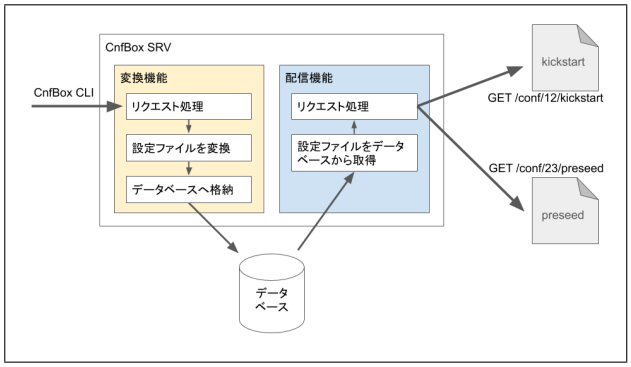

図 3　CnfBox SRV アーキテクチャ

複数フォーマットへの対応： 従来の手法 [3], [8], [9] は Kickstart に対応していた．Kickstart は RedHat 系ディストリビューション (例：CentOS, Fedora) のみに対応している．したがって，他の Linux ディストリビューション (例：Ubuntu, Debian) は従来の手法では対応できない．そのため，それらツールを使うことで OS の選択肢が限られる．本研究では，RedHat 系ディストリビューションだけでなくそれ以外のディストリビューションへ対応する．

フォーマットの維持： 既存技術である Kickstart と preseed との互換性を維持した．これにより，既存の Kickstart や preseed に対応するクライアントからのアクセスに対応できる．すなわち，既存インストーラへの変更を必要としない．複数の異なるフォーマットを統一フォーマットからプログラムにより変換し，設定を生成する．

アーキテクチャの変更： 本提案は，既に構築済みのアーキテクチャとの親和性が高い．すなわち，既存のアーキテクチャを変更せずに導入できる．LCFG[3] や Lucie[8] は，独自サーバを必要とする．このため，既に構築済みの環境へ新たな変更を加えなければならない．本提案は，既存の環境における変更を 1 パラメータに絞った．そのため，最小限の変更で導入を行える．なお，新たなサーバはパブリッククラウドへ配置するため既存環境へのサーバー追加も不要である．

サーバの運用コストを削減： 設定ファイルを配置する

サーバで，サーバレスアーキテクチャの利用を提案する．従来は，特定の目的で使用するサーバの維持および管理のコストが発生していた．設定ファイルを配置する単純なサーバを動作させるために，ミドルウェアや OS，ライブラリのメンテナンスが生じていた．サーバレスアーキテクチャの採用により，特定の目的に縛られたサーバが不要とした．

カスタマイズ性： CnfBox では，クライアントプログラムとサーバープログラムを分離している．そのため，インターフェースを揃えることで独自のクライアントプログラムを新たに作成できる．例えば，Web アプリケーションから Web インターフェースを利用した設定ファイルの生成や取得が行える．

4. 実装

アーキテクチャは，オペレータの操作する CLI プログラム (以下，CnfBox CLI) とデータを処理するサーバプログラム (以下，CnfBox SRV) から構成される．また，CnfBox SRV は独自の統一設定ファイル（以降 form.py とする）により設定を管理する．

4.1 CnfBox CLI

Python 3.7.6 により CnfBox SRV と通信する CLI クライアント CnfBox CLI を実装した．これには設定の投入，閲覧，サーバへの送信が実装されている．サーバとの通信には HTTP を使う．このプログラムは，Python プログラムとして使用できるだけでなく，Python インタプリタモードで起動することで対話形式で使用できる（図 4）．設定フォーマットの例をコード 3 に示す．

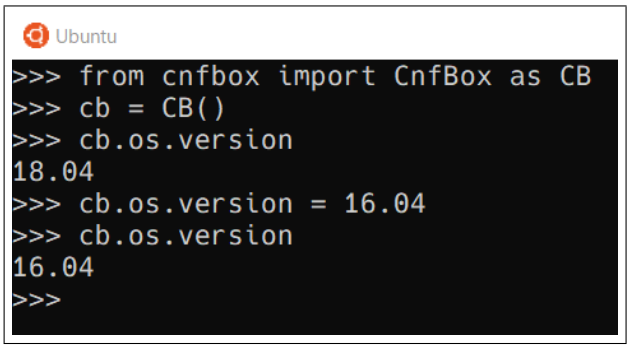

図 4　対話型ターミナルによる設定

4.2 CnfBox SRV

Python 3.7.6 により CnfBox CLI のリクエストを処理する API サーバ CnfBox SRV を実装した．これには CLI とのやり取り，構文のチェック，設定ファイルの公開が実装されている．クライアントとの通信には HTTP を，形式には JSON を採用した．Web アプリケーションライブラリには Flask(バージョン: 1.1.1)，データベースライブラリには PyMySQL(バージョン: 0.9.3) を使用した．実装した API エンドポイントを表 2 に示す．設定パラメータの登録機能と表示機能の 2 つを RESTful API により実装した．データベースは MySQL 5.7 を採用した．カラムはデータを識別する ID(INT 型, AUTO_INCREMENT 機能により生成) と設定パラメータ (JSON 型) の 2 つから構成される．

コード 3　form.py の例

```
from cnfbox import CnfBox

cb = CnfBox()

cb.os.name = 'centos'
cb.os.version = 8.0
cb.os.lang = 'us'

cb.user.name = ['alice', 'bob']
cb.network.device = 'ens160'
cb.network.iptype = 'static'
cb.network.ipaddr = '10.200.1.5'
cb.network.cidr = 16
cb.network.gateway = '10.200.1.254'
cb.network.dns = '10.200.1.3'
print(cb.show_json())
```

表 2　CnfBox SRV エンドポイント一覧

Method	Path	Role
GET	/conf/16	JSON 形式で設定を表示
GET	/conf/16/Kickstart	Kickstart 形式で設定を表示
GET	/conf/16/preseed	preseed 形式で設定を表示
POST	/conf	JSON 形式で設定を受け取り

4.3 インストール手順

本提案におけるインストール手順を図 5 に示す．以下では，図 5 の手順ごとに説明を行う．

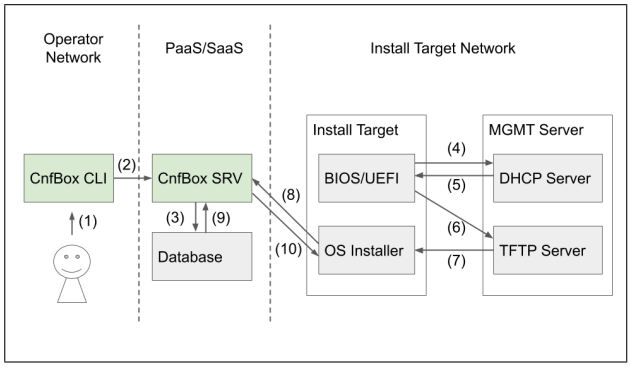

図 5　インストール手順

(1) オペレータは CnfBox CLI を利用してインストールのパラメータを投入する．
(2) CnfBox CLI は入力された設定を CnfBox SRV へ送信する．
(3) CnfBox SRV は受信した設定データから Kickstart,

preseeed の設定フォーマットを生成し，データベースへ保存する．

(4) インストール対象マシンの電源をいれ，ネットワークブートを選択．DHCP サーバへ IP 取得を試みる．

(5) DHCP サーバは IP 取得のリクエストに応答し，IP アドレスを払い出す．

(6) インストール対象マシンは，DHCP サーバーから提示された TFTP サーバへアクセスして OS インストーラを取得する．

(7) TFTP サーバはインストール対象マシンからのリクエストに対応する OS インストーラを返す．

(8) OS インストーラは CnfBox SRV へ設定ファイルの取得リクエストを送る．

(9) CnfBox SRV は受信したリクエストに対応する設定ファイルをデータベースから取得する．

(10) CnfBox SRV はデータベースから取得した設定を，OS インストーラへ返す．

(11) OS インストーラは取得した設定をもとに自動的にインストールを行う．

5. 評価

提案に基づく実装を，課題をもとに独自に作成した次の観点で比較した．また，生成された設定ファイルにより自動インストールが正常に行われるか実験を行った．

- 作成する設定ファイル数
- 記述する言語数
- 設定インターフェースの種類
- 設定ファイル配信サーバの運用

5.1 実験環境

研究室内部に設置された仮想化基盤上に仮想マシンによる実験環境を構築した．また，CnfBox SRV を Google Cloud Platform(以降，GCP とする) 上の Google App Engine と Google Cluod SQL を使い構築した．研究室環境と環境 (図 6) と GCP はインターネットを介してやり取りが可能である．実験では簡単のため，OS インストーラの起動と設定ファイルの指定を手動により行った．そのため，TFTP サーバを構築していない．

CnfBox SRV

GCP 上の Google App Engine(以降，GAE とする) へプログラムを設置する．GAE のランタイムは Python 3.7 を使用した．ファイアウォールルールにより，大学のネットワーク以外からの HTTP および HTTP によるアクセスを IP レンジに基づき遮断した．データベースサーバとの通信には Unix ドメインソケットを使用した．

データベースサーバ

GCP 上の Google Cloud SQL で MySQL 5.7 による

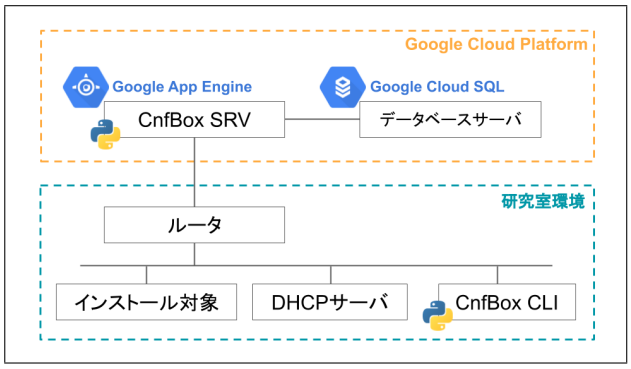

図 6 研究室環境と GCP 環境

データベースサーバ (vCPU: 1 コア, RAM: 614MB, HDD: 10GB) を構築した．CnfBox SRV と同様に大学以外からのアクセスを遮断した．GAE 実行ユーザに IAM ロール (Cloud SQL クライアント) を付与することで，データベースサーバへのアクセスを許可した．

CnfBox CLI

研究室環境に接続したラップトップ (CPU: Intel Core i7-8550U 1.8 GHz, RAM: 16GB, SSD: 512GB) で Python 3.7.6 の動作環境を構築して，CnfBox を実行する．

インストール対象

研究室環境の VMWare ESXi 上に VM(CPU: 1core, RAM: 2GB, SSD: 30GB) を作成した．OS はインストールされていない．

DHCP サーバ

研究室環境の VMWare ESXi 上に VM(CPU: 1core, RAM: 2GB, SSD: 30GB) を作成し，Ubuntu 18.04 上に ISC DHCP をインストールした．

5.2 実験方法

次の 3 ステップを 2 種類の OS(CentOS 7, Ubuntu 18.04) により実験し，正常に自動インストールが行えるか検証した．

(1) **CnfBox を GitHub からダウンロード：** GitHub リポジトリからソースコードをダウンロードする．

(2) **GCP のセットアップ：** GCP でプロジェクトを作成し，Cloud SQL のインスタンスを前述のスペックで作成する．

(3) **CnfBox SRV を GCP にデプロイ：** GAE に CnfBox SRV をデプロイする．GAE 実行ユーザの IAM ロールに Cloud SQL クライアントを追加する．ファイアウォールポリシーで大学以外からのアクセスを遮断する．

(4) **CnfBox CLI による設定の投入：** CnfBox CLI を Python 3.7.6 をインストールしたマシンで実行する．

(5) **CnfBox SRV へのアップロード：** CnfBox CLI から commit() メソッドを実行し，設定を CnfBox SRV へ

アップロードする．

(6) **VM の設定と起動：** 研究室環境の VM(インストール対象) に Ubuntu インストーラ ISO(ubuntu-18.04-live-server-amd64.iso) を DVD ドライブへ設定し，VM を起動する．

(7) **起動パラメータの設定：** ブートローダの起動オプションを Ubuntu 用コード 4 に修正して OS を起動する．実験では CnfBox SRV は https://cnfbox-dev.appspot.com/ に設置した．

コード 4　Ubuntu 起動オプション

```
append DEBCONF_DEBUG=5 auto=true locale=en_US.
    UTF-8 (略) interface=auto url=https://
    cnfbox-dev.appspot.com/conf/16/preseed vga
    =normal initrd=/install/initrd.gz quiet --
```

(8) **設定の配信：** OS インストーラは起動オプションに付与された設定の取得先である CnfBox SRV からの設定ファイルを取得する．

(9) **インストール状況の確認：** 自動インストールが正常に完了するか確認する．

(10) **パラメータを変えて実験：** 手順 (6) 以降を以下の変更を加えて再び実験する．

- **VM の設定と起動：** Ubuntu インストーラ ISO → CentOS インストーラ ISO(CentOS-7-x86_64-DVD-1908.iso)
- **ブートローダの起動オプション：** Ubuntu 用コード 4 → CentOS 用コード 5

コード 5　CentOS 起動オプション

```
linux text ks=http://cnfbox-dev.appspot.com/
    conf/16/Kickstart
```

5.3　実験結果

Ubuntu, CentOS ともに正常に自動インストールが行われ，コンソール経由でログイン可能だった．CnfBox SRV のアクセスログからも，正常に設定ファイルが取得できたことが分かる（コード 6）．

5.4　比較

CnfBox と同様の機能を提供する既存技術 (preseed + Kickstart) を比較した．作成する設定ファイル数および記述言語数が他に比べ，CnfBox の利用により削減された．これにより，オペレータの設定ファイル作成に費やす時間の削減につながった．従来はファイル記述型だけあった設定用インタフェースに対話型を加えることにより，設定ファイル作成におけるデバッグ時間の削減につながる．また，

コード 6　CnfBox SRV のアクセスログ

```
163.215.6.1 - - [13/Jan/2020:13:17:44 +0900] "
    GET /conf/16/preseed HTTP/1.1" 200 499 - "
    debian-installer" "cnfbox-dev.appspot.com"
     ms=2277 cpu_ms=1978 cpm_usd=5.5767e-8
    loading_request=1
163.215.6.1 - - [10/Jan/2020:17:42:27 +0900] "
    GET /conf/16/Kickstart HTTP/1.1" 200 604 -
     "urlgrabber/3.10.2" "cnfbox-dev.appspot.
    com" ms=79 cpu_ms=407 cpm_usd=6.7502e-8
    loading_request=0
```

補完によるユーザー体験が向上される．設定管理と変換を行う CnfBox SRV は PaaS へのデプロイが可能であることから，OS やミドルウェアの運用コストの削減につながる．

表 3　CnfBox SRV エンドポイント一覧

	CnfBox	preseed + Kickstart
作成する設定ファイル数	1	2
記述する言語数	1	2
設定用インターフェース	ファイル記述型/対話型	ファイル記述型
設定配信サーバ	PaaS/オンプレミス	オンプレミス

6.　議論

6.1　統一フォーマット

本研究では，複数存在する設定フォーマットの差異を吸収すべく Python ベースの統一フォーマットを提案した．設定フォーマットには，書きやすさ，学びやすさ，拡張しやすさを考慮した．当初は Ansible の採用した YAML 形式による記法を検討したが，拡張子やすさの観点で不十分であるため見送った．採用した Python による記法は，前述の 3 観点すべてを満たす十分なものだといえる．別のプログラムからの呼び出しや，データの受け取りが Python プログラム間であれば容易に行える．これは，従来の Kickstart や preseed, 他の先行研究でも実現されていない．そのため，本提案はプログラマブルな設定フォーマットとして優位性がある．

6.2　インタラクティブなインターフェース

従来のインターフェースはテキストエディタで編集した設定ファイルをコマンドによりテストする方法（例：ksvalidator）が存在した．この方法では，設定ファイルの作成にはファイル編集，ファイル保存，設定内容のバリデーションを繰り返し行う必要があった．CnfBox CLI は対話型による設定の投入方法を実現した．これにより，オペレータは設定パラメータ名を正確に覚えることが不要である．すなわち，設定パラメータへの理解が浅いオペレー

タでも設定が行える．本研究はインターフェースの改善により，設定の難易度の低下に貢献した．

6.3 設定フォーマットの変換

CnfBox では，統一フォーマットで記述された設定を既存ツールに対応する設定ファイルフォーマットに変換した．これにより，従来のツールへ変更が不要となる．これは，対応フォーマットが少ない範囲においては有効である．しかし，フォーマットが増えるにつれ CnfBox SRV での対応の手間が発生する．また，変換処理の追加時には CnfBox でのパラメータと変換後フォーマットとの対応付けが必要となる．これらを踏まえ，本提案で採用した設定フォーマットの変換は，フォーマットの差異を埋める汎用的で十分な手法とはいえない．

6.4 サーバー運用コスト

一般に，自動インストール設定ファイルは，HTTP サーバーに配置される．これを実現するために HTTP サーバの維持，管理が発生していた．本提案では，それらコストを低減させるべくサーバレスアーキテクチャを採用し，PaaS にデプロイした．その結果，ミドルウェアや OS カーネルの維持，管理コストを省くことを可能とした．特定の目的とするサーバを排除することにより維持，管理コストの削減を実現した．

7. 終わりに

CnfBox では，パスワードに代表される機密情報の管理が十分に検討されていない．IP レンジによるアクセス制限に加えて，暗号化した上で保存する方法の検討が必要である．これには HashiCorp の開発する Vault にあげられる KVS の採用を検討している．

RESTful API とデータベースの採用により，これまで意識されなかった設定のバージョニングを実現した．これは副産物であったが，設定ファイルの管理において意義のある結果だといえる．より，バージョン管理機能を強化することを検討したい．

参考文献

[1] Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I. and et al.: A View of Cloud Computing, *Commun. ACM*, Vol. 53, No. 4, p. 50–58 (online), DOI: 10.1145/1721654.1721672 (2010).

[2] Burgess, M. et al.: Cfengine: a site configuration engine, *in USENIX Computing systems, Vol*, Citeseer (1995).

[3] Anderson, P. and Scobie, A.: Large Scale Linux Configuration with LCFG, *Proceedings of the 4th Annual Linux Showcase & Conference - Volume 4*, ALS'00, Berkeley, CA, USA, USENIX Association, pp. 42–42 (online), available from ⟨http://dl.acm.org/citation.cfm?id=1268379.1268421⟩ (2000).

[4] Anderson, P.: Towards a high-level machine configuration system, *In Proceedings of the 8th Large Installations Systems Administration (LISA) Conference* (1994).

[5] 健太服部，真也溝江，正樹三角：Kompira：シンプルで軽量な IT 運用自動化プラットフォーム，第 54 回プログラミング・シンポジウム予稿集，Vol. 2013, pp. 99–106 (2013).

[6] Finley, B. E.: VA Systemimager, *Proceedings of the 4th Annual Linux Showcase & Conference - Volume 4*, ALS'00, USA, USENIX Association, p. 11 (2000).

[7] Katz, M. J., Papadopoulos, P. M. and Bruno, G.: Leveraging standard core technologies to programmatically build Linux cluster appliances, *Proceedings. IEEE International Conference on Cluster Computing*, pp. 47–53 (online), DOI: 10.1109/CLUSTR.2002.1137728 (2002).

[8] 安仁高宮，篤　真鍋，聡　松岡：Lucie：大規模クラスタに適した高速セットアップ・管理ツール，情報処理学会論文誌コンピューティングシステム（ACS），Vol. 44, No. SIG11(ACS3), pp. 79–88 (2003).

[9] Magherusan-Stanciu, C., Sebestyen-Pal, A., Cebuc, E., Sebestyen-Pal, G. and Dadarlat, V.: Grid System Installation, Management and Monitoring Application, *2011 10th International Symposium on Parallel and Distributed Computing*, pp. 25–32 (online), DOI: 10.1109/ISPDC.2011.14 (2011).

[10] 良輔小久保，和広松山，利彰三鴨，真司住元，浩一平井：超大規模 HPC システムのインストール高速化の提案，技術報告 18，富士通株式会社, 富士通株式会社, 富士通株式会社, 富士通株式会社, 富士通株式会社 (2018).