

# JANOG34 : DCネットワークの新時代? —ホワイトボックススイッチの利用可否—

## **WhiteBox** で変わる運用(予測)

グリー株式会社 大山 裕泰

[ 2014/07/17 ]

# 自己紹介

大山 裕泰

- 所属：グリー株式会社  
インフラストラクチャ本部・開発部

## # やってきたこと #

- Sus v3 互換<sub>(予定)</sub>オリジナルカーネル “Onix” 開発
- OpenFlow Controller JAXON 開発

# 自己紹介

[Home](#)[Commur](#)

by [Murphy](#)  
[McCauley](#)

## Jaxon: Java Bindings for NOX

March 27, 2012 in [NOX](#), [ThirdParty](#)

A couple of weeks ago, there was a [post on nox-dev](#) announcing the [Jaxon](#) project which had just had [its source code](#) go up on Bitbucket. Jaxon uses Java Native Access to expose some of NOX Classic's API to Java, while being *almost* just an ordinary NOX component (that is, it only requires a minor change to NOX itself, which I think we could actually fix...). Of course, it comes with the prototypical sample application — an L2 learning switch equivalent to switch/pyswitch.

While Jaxon is a standalone project, it's primarily intended to be used as part of a bigger project — [Kumoi](#) — which is working towards using Scala to manage datacenters by representing datacenter entities (like virtual machines) as normal Scala objects. This should provide a consistent, integrated, programmatic interface with all the benefits of Scala (like type checking). Anyway, I don't know exactly what they are working on, but you can see exciting possibilities here for getting SDN into the mix too.

While Jaxon currently works with NOX Classic, it seems to Amin and me that the simplifications in the new NOX codebase may simplify the Jaxon effort as well. I've spoken to Hiroyasu Ohyama, the developer of Jaxon, and he seems optimistic about moving to the new codebase as soon as we finally release it. (Which I swear is any day now. Really.)

We know there's already at least one good [Java OpenFlow controller](#), so it's interesting to see NOX used here. I'll be curious to see how it progresses!

**WhiteBox** で変わる運用 (予測)

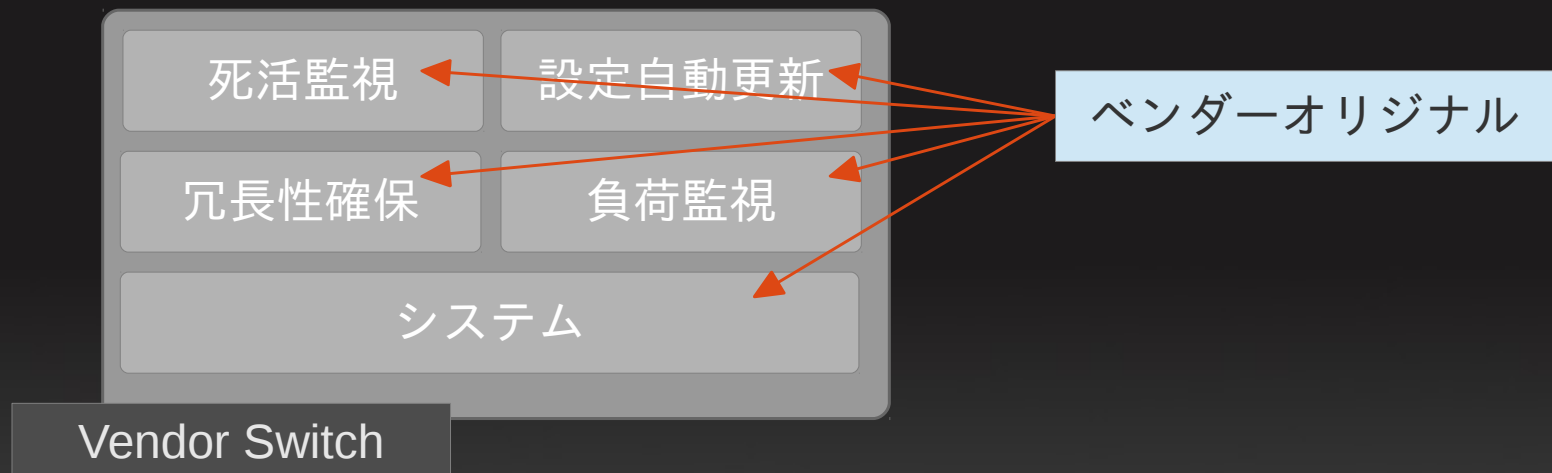
# WhiteBox で変わる運用予測

- 1) サーバシステムとの連携
- 2) デプロイの自動化による高速デリバリ
- 3) オープンシステムを利用した機能拡張

# 1) サーバシステムとの連携

## \* ベンダー製品の場合

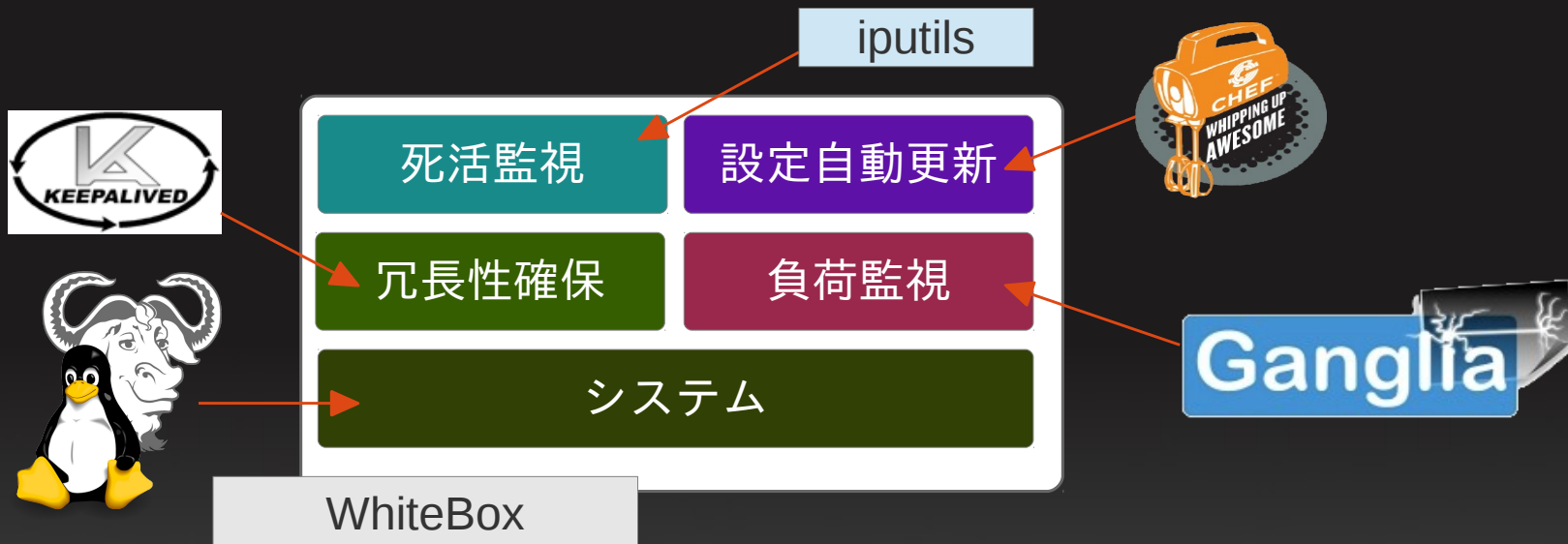
- 他のソフトウェアの実行・連携が難しい
- ネットワークの運用に長けた人材・ノウハウが要求される



# 1) サーバシステムとの連携

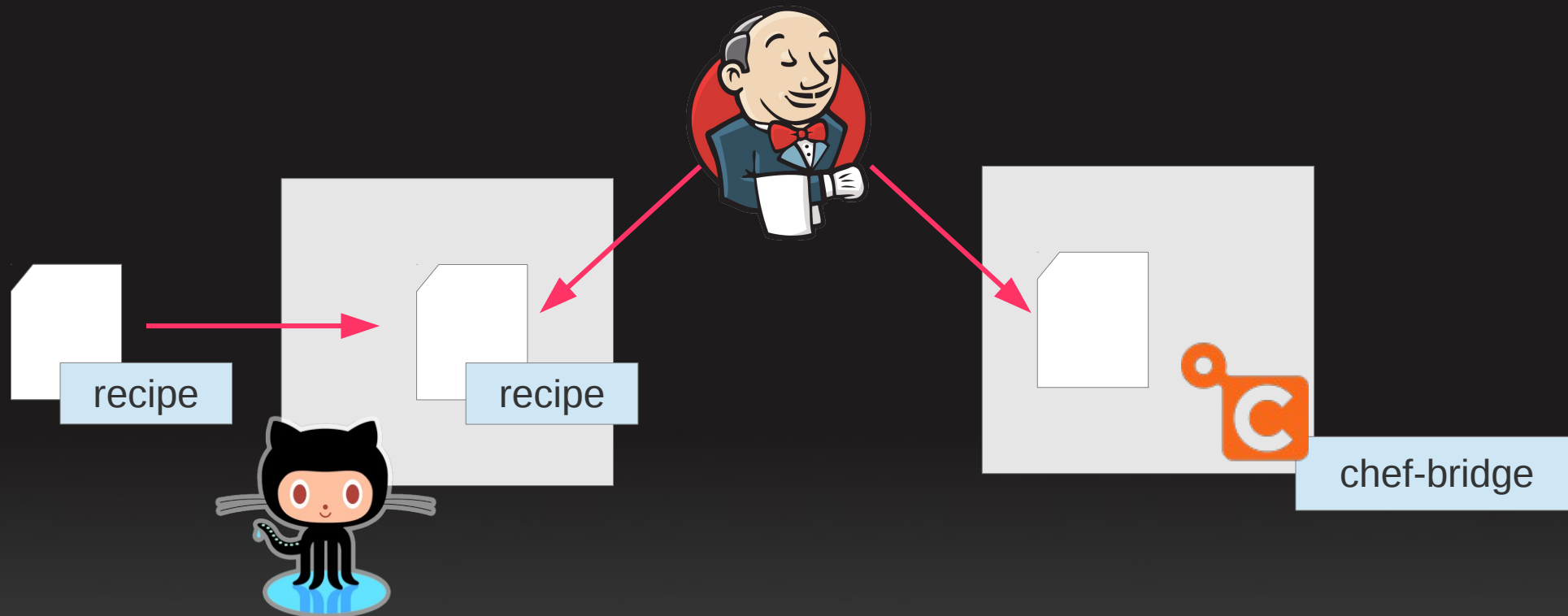
## \* WhiteBox の場合

- OSS を含む様々なソフトウェアを実行可能
- サーバ運用のソフトウェア, 人材, ノウハウを活用できる



## 2) デプロイの自動化

- \* Chef による WhiteBox のプロビジョニング
  - サーバに対するデプロイの仕組みを WB に応用

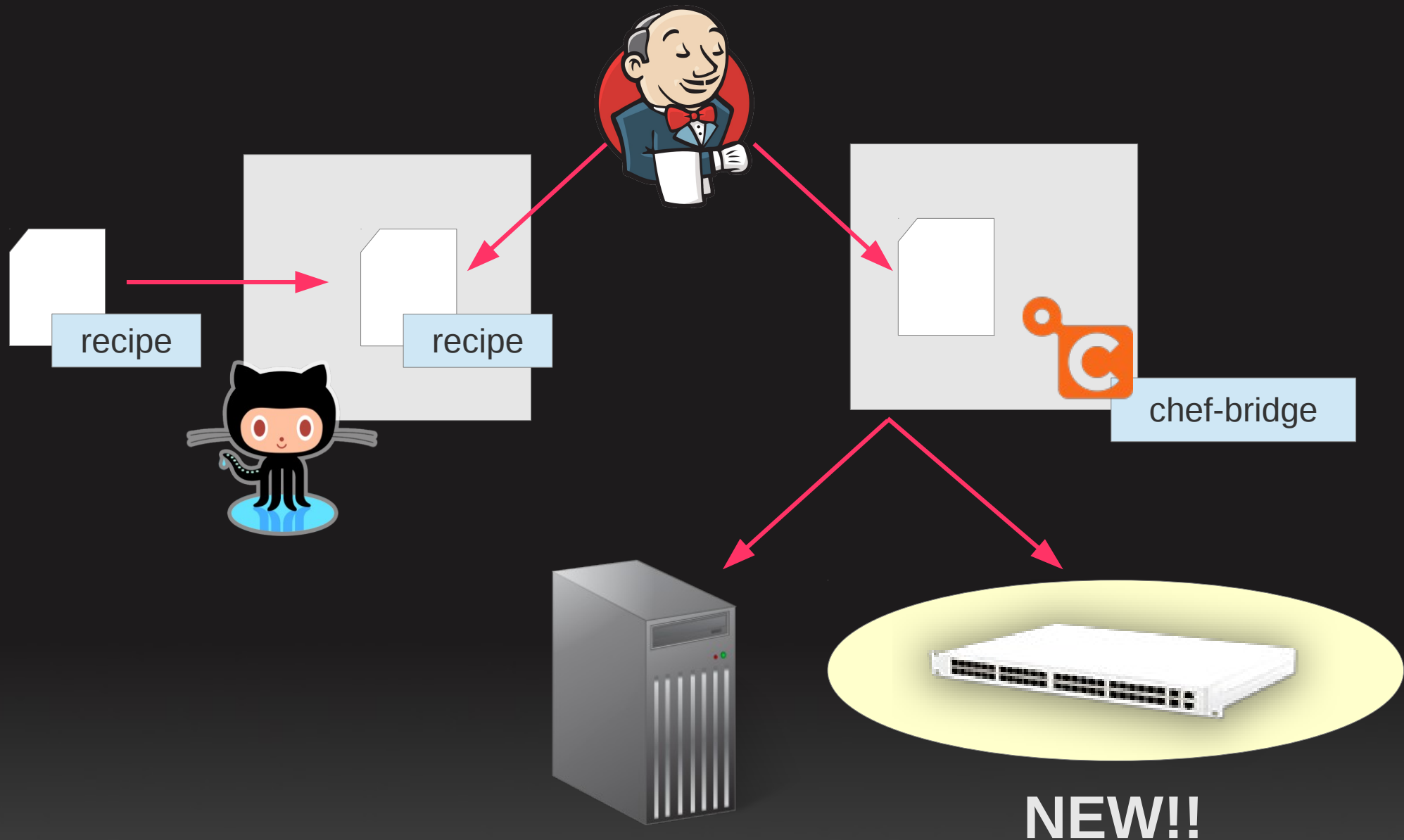


グリーのサーバデプロイの仕組み



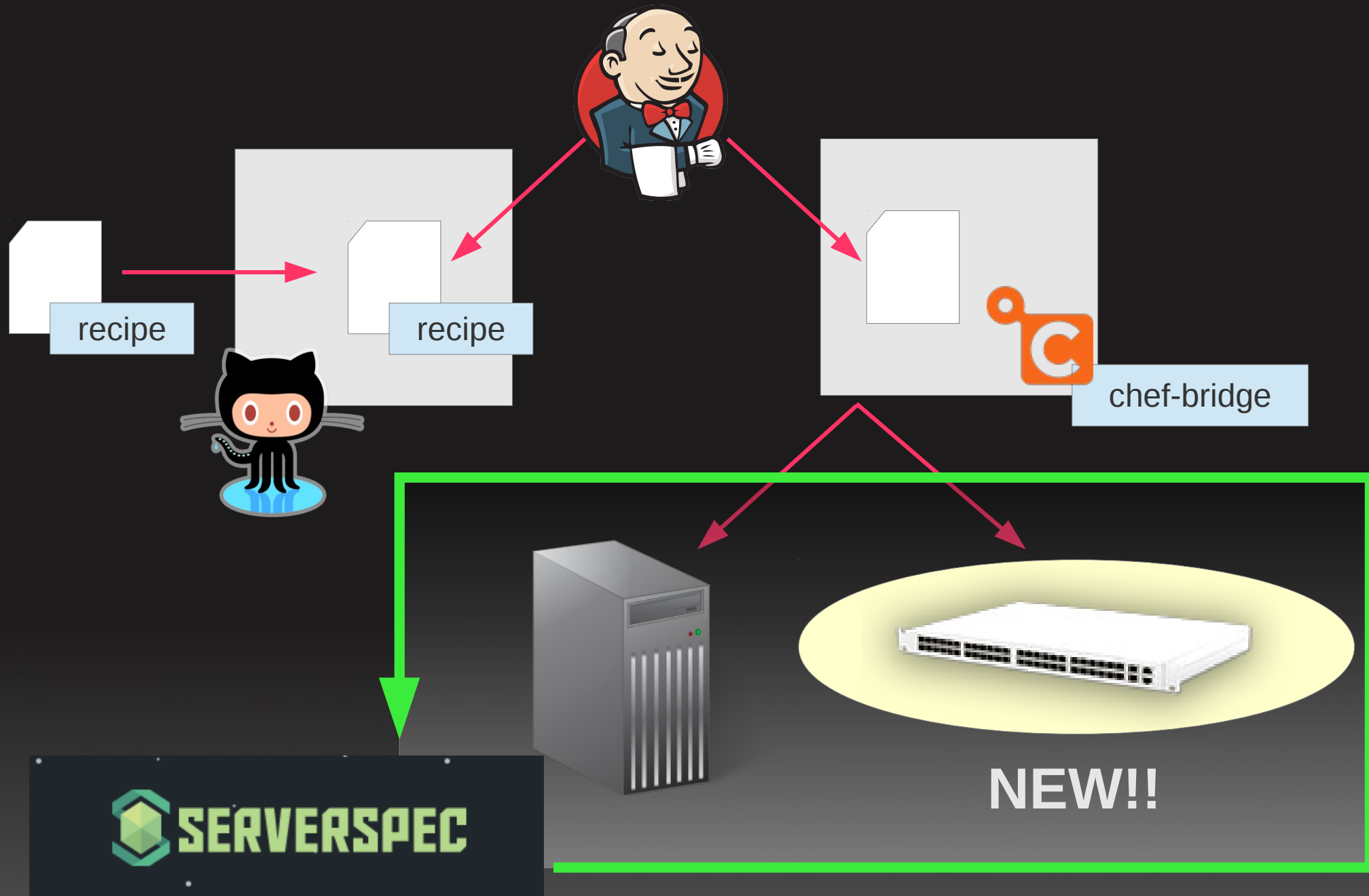
# \* Chef による WhiteBox のプロビジョニング

- サーバに対するデプロイの仕組みを WB に応用



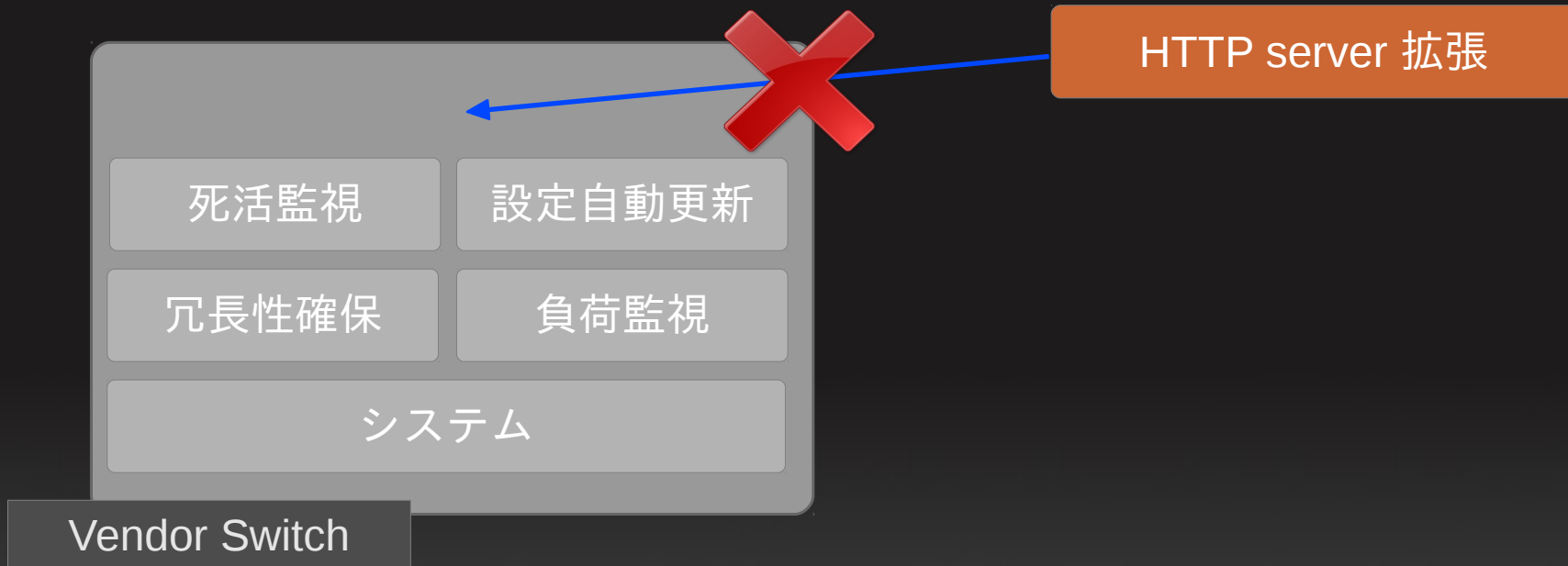
# \* Chef による WhiteBox のプロビジョニング

- サーバに対するデプロイの仕組みを WB に応用



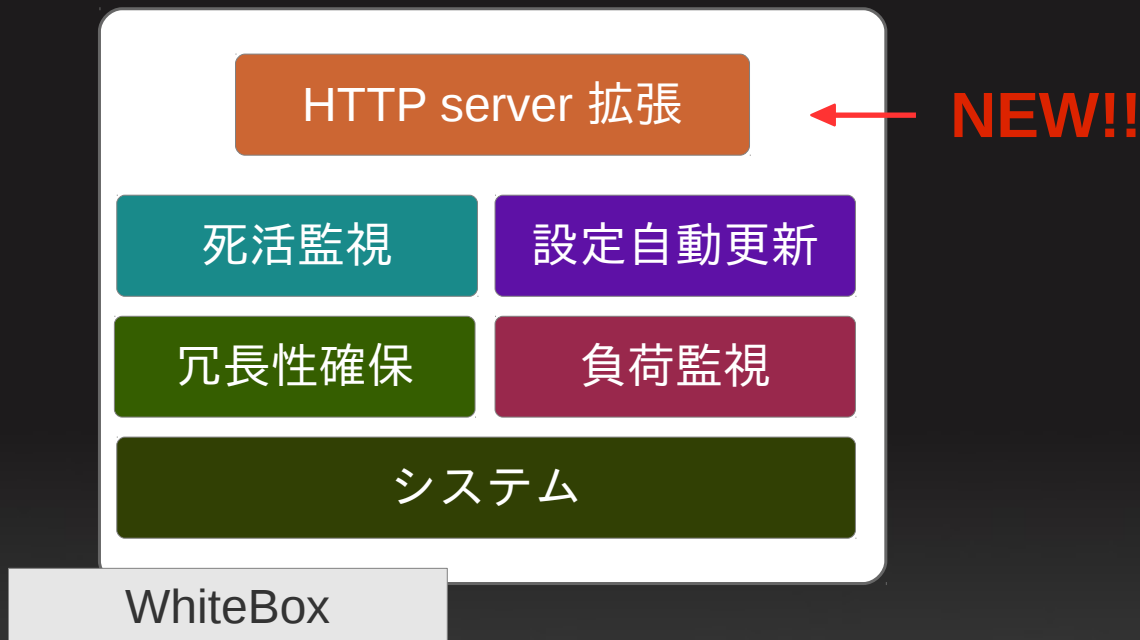
### 3) 機能拡張

- \* (ベンダー製の場合) 機能拡張はベンダーに依存
  - 何らかの追加機能を機器に持たせたい場合



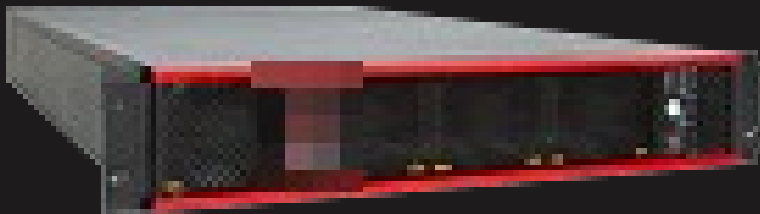
### 3) 機能拡張

- \* (WhiteBox製の場合) 自由に機能拡張が行える
  - 何らかの追加機能を機器に持たせたい場合



### 3) 機能拡張

- \* WB にストレージを載せた製品が登場
  - L2/L3 SW 兼サーバとしての利用が期待

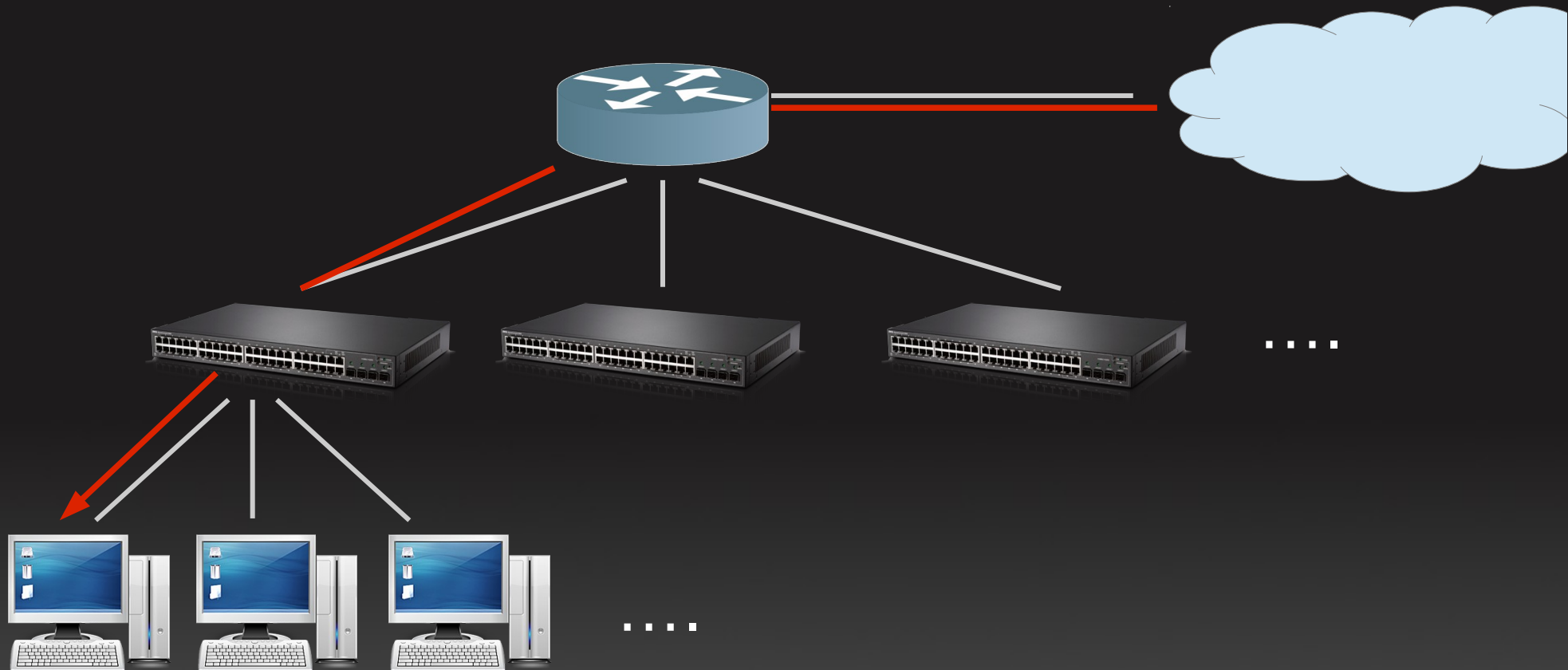


Network : 10GE x 48 + 40GE x 4  
CPU : Dual Xeon E5  
RAM : 256 GB  
Storage : 2x300G SSD  
          2X1T     HDD

### 3) 機能拡張

#### \* SW に HTTP Proxy を入れる

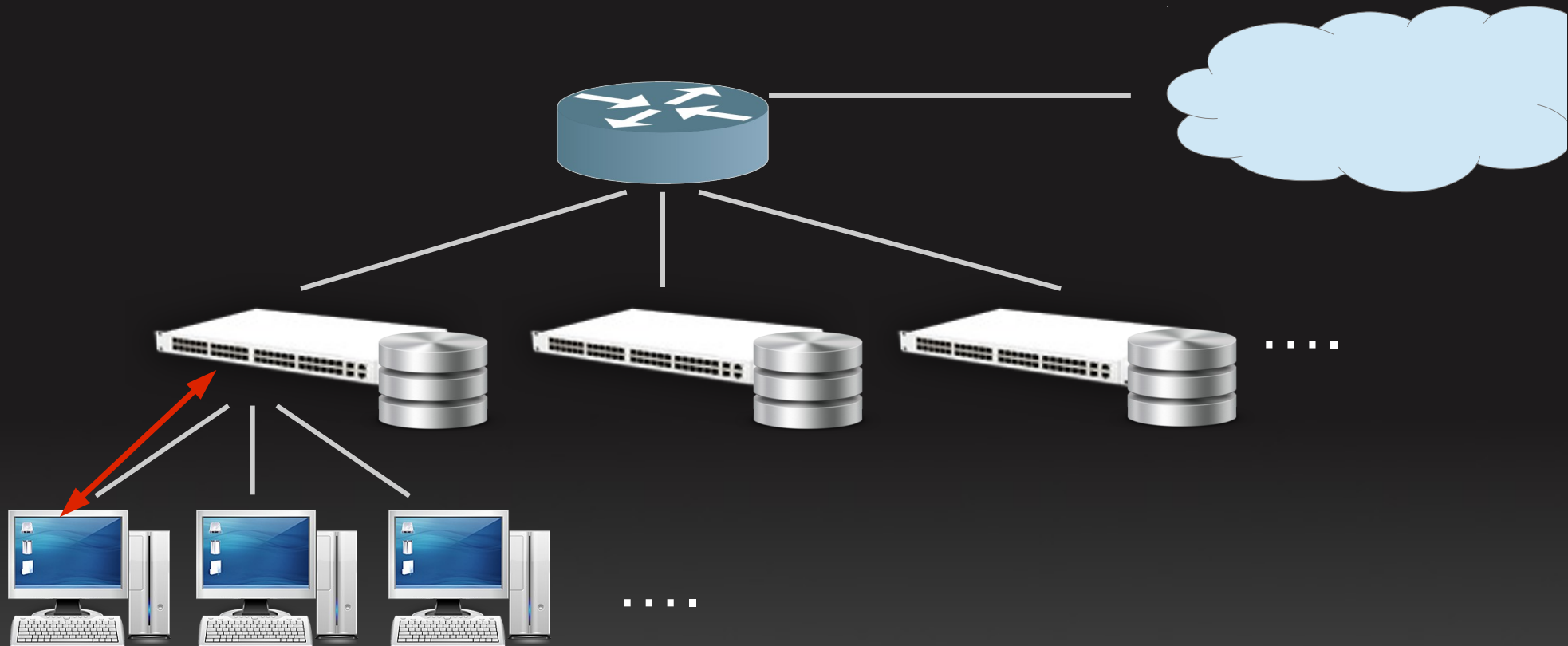
- 静的コンテンツを SW にキャッシュさせることで HTTP 通信を高速化できるかも... (@大規模オフィス環境)



### 3) 機能拡張

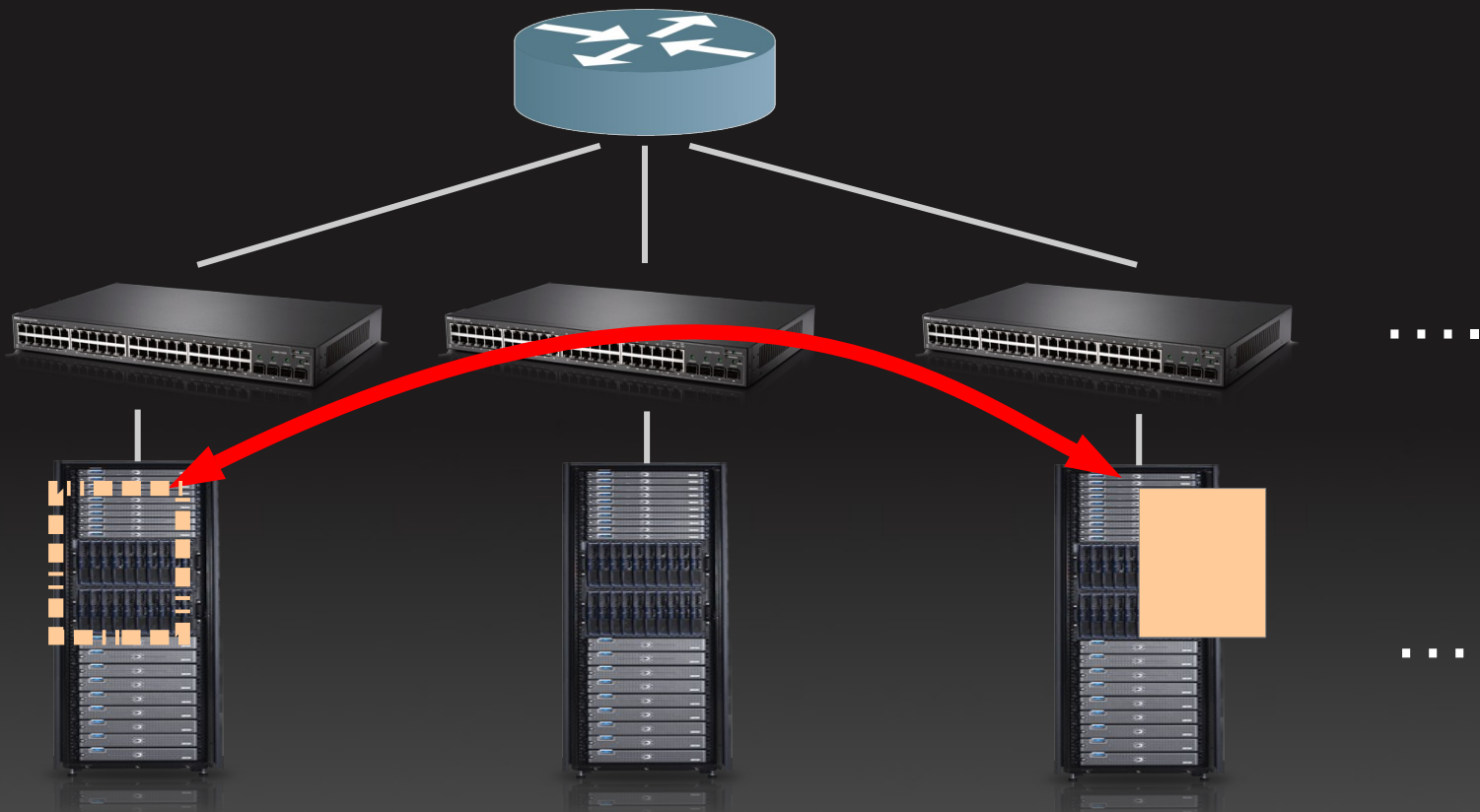
#### \* SW に HTTP Proxy を入れる

- 静的コンテンツを SW にキャッシュさせることで HTTP 通信を高速化できるかも... (@大規模オフィス環境)



### 3) 機能拡張

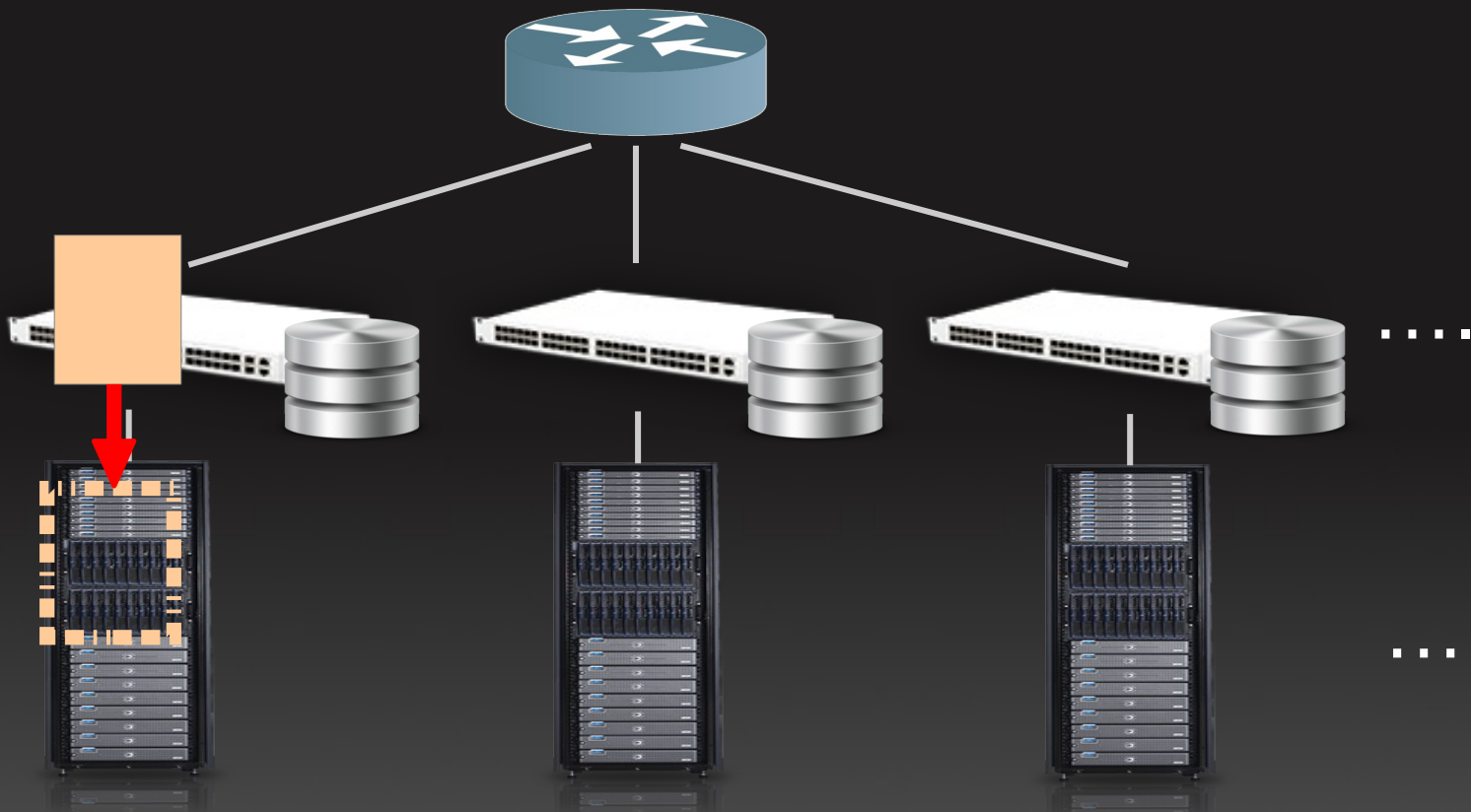
- \* SW に静的コンテンツ (apt, gem, OS image, etc) を配置
  - east-west トラフィックを抑制できるかも...





### 3) 機能拡張

- \* SW に静的コンテンツ (apt, gem, OS image, etc) を配置
  - east-west トラフィックを抑制できるかも...



# WhiteBox で変わる運用予測

## 1) サーバシステムとの連携

- サーバのソフトウェア、人材、ノウハウを活用

## 2) デプロイの自動化による高速デリバリ

- Scalable で Reliable なネットワークシステムが実現

## 3) オープンシステムを利用した機能拡張

- L7 まで含めた新たな使い方ができる

かもしれない!!

続いて  
デモ編

# Chef でデプロイしてみた

