ネットワーク運用
自動化の世界２

運用を支援する
システム / API の作り方

小島 慎太郎

codeout

http://about.me/codeout

ISP: 5年（ntt.net / AS2914）

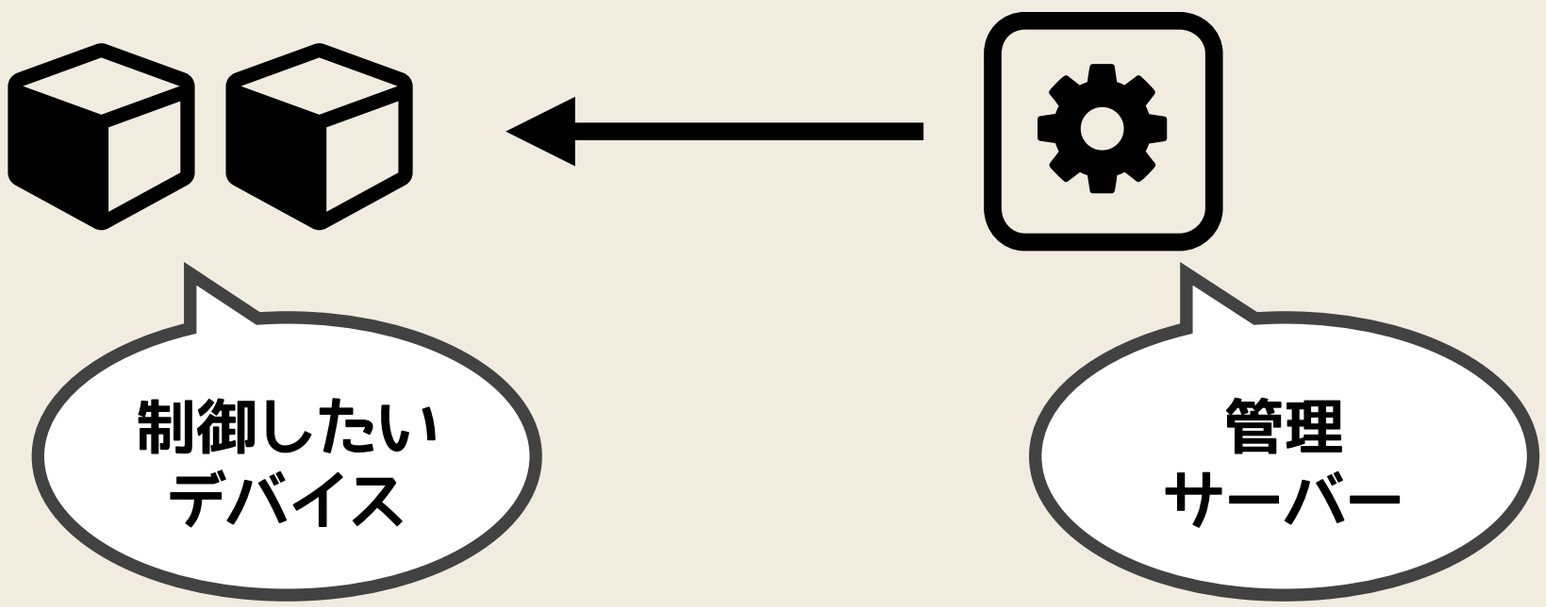
IX: 4年（JPNAP）

クラウド: 1年（BHEC）

ここまでの話

デバイスの制御

- 特に物理デバイス
  - CLI をparse するのはハードルが高い
  - コマンド実行のたびに show コマンドで確認
- NETCONF が使える



これからの話

ネットワーク運用には、
ほかにもいろいろ
コンポーネントがある

目的を達成するために、
それらをうまく連携させて、
自分たちの運用にあった形で、
どうやって自動化したらいい？

ネットワーク運用のコンポーネント

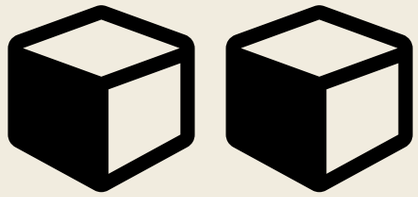
監視
サーバー

可視化/分析
サーバー

オペレーター

管理
データベース

制御したい
デバイス

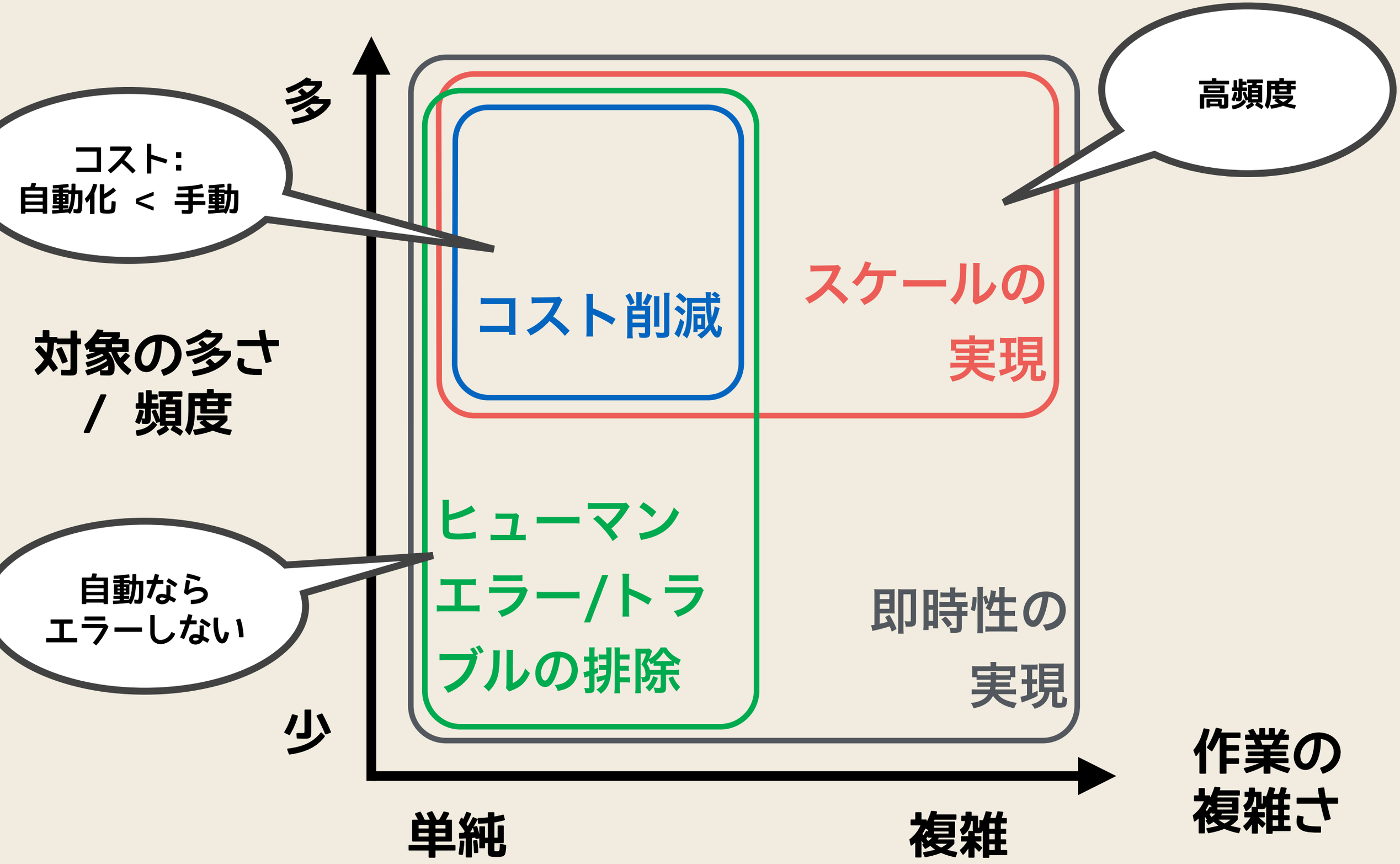


管理
サーバー

自動化の目的

- 効率化
    1. スケールの実現
    2. コスト削減
3. ヒューマンエラー/トラブルの排除
4. 即時性の実現

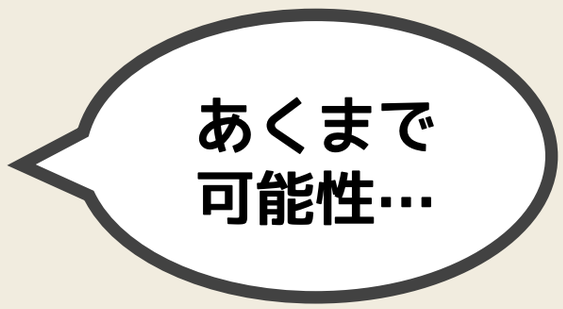
どこを自動化する？


多
少
対象の多さ
/ 頻度
単純
複雑
作業の
複雑さ
コスト:
自動化 < 手動
高頻度
自動なら
エラーしない
コスト削減
スケールの
実現
ヒューマン
エラー/トラ
ブルの排除
即時性の
実現

どこを自動化しない？

オペレーターの手 / 目を挟みたいプロセスはあるか？

- 人間の目でチェックしたい
  - 未知のエラーに気づく可能性
- ユーザーと一緒に作業したい
  - 作業内容が変わるかも
  - 自在にロールバック / 中断



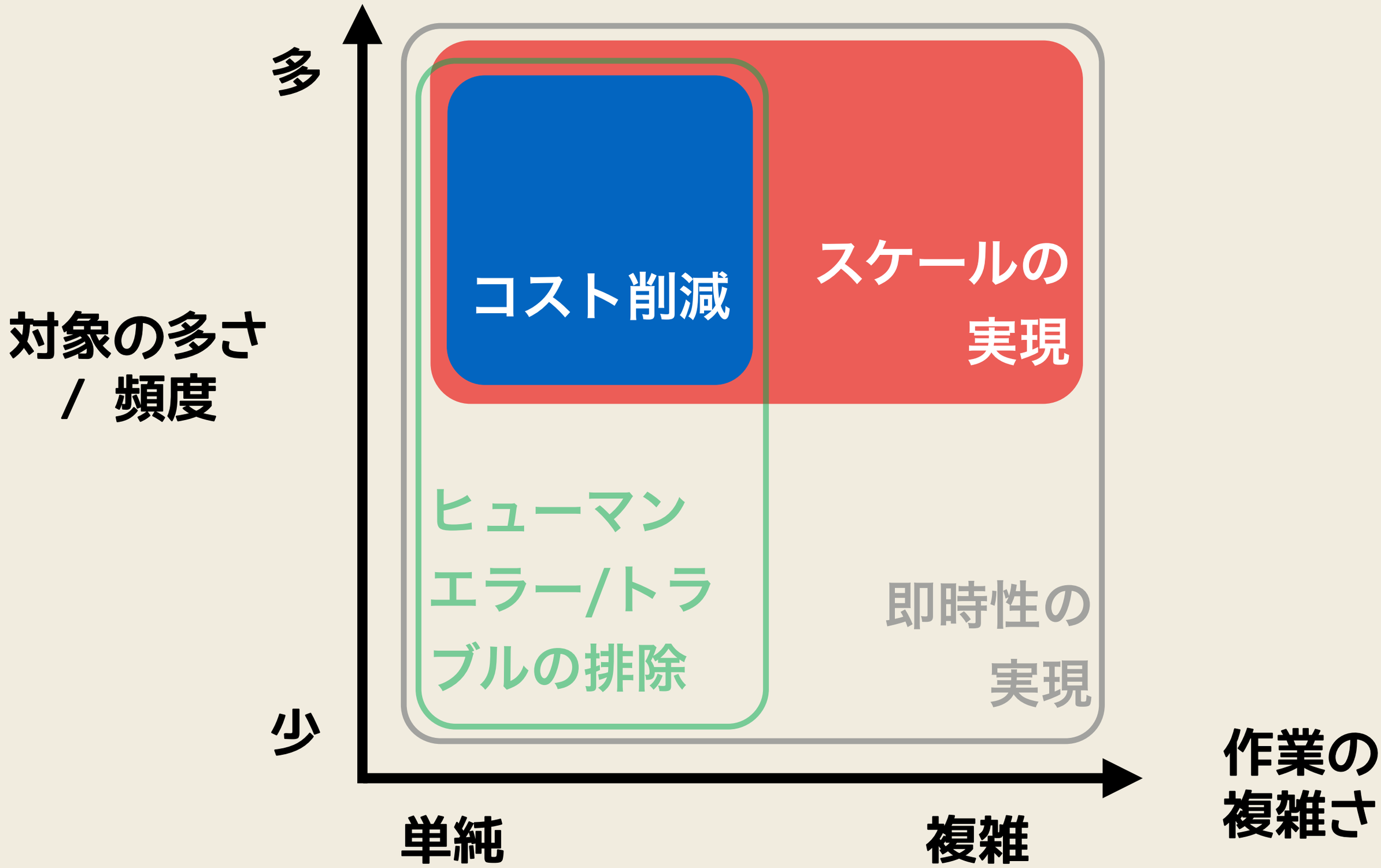
目的によって、
どこを自動化するか
どこを自動化しないか
が変わる

ntt.net (AS2914)

ntt.net (AS2914)

- グローバル Tier1 ISP
- 世界中に 数百台のルーター
- 低価格で提供することが重要

どこを自動化する？


多
対象の多さ
/ 頻度
少
単純
複雑
作業の
複雑さ
コスト削減
スケールの
実現
ヒューマン
エラー/トラ
ブルの排除
即時性の
実現

すべてを自動化

未使用のもの
含め、すべて監視

未使用のもの含め、
すべて取得

(2) 管理サーバーで
コマンド発行

(1) データ投入

(3) 設定投入

自動化を前提としたサービス

- 自動化できない機能はリリースしない
- ユーザー要望があっても、特別対応を避ける
    - 避けられなかったものは、例外として管理
    - 例外をテンプレートに差し込むしくみ
- 自動制御できないデバイスは使わない
- ワークフローを非同期にする

ProTip: rancid

- http://www.shrubbery.net/rancid/
- 付属の clogin, jlogin, flogin などが便利

```
$ clogin -x cmd.txt host1 host2
```

- expect スクリプトで、CLI の入出力をwrap
- サポートするデバイスが多い
  - エラー補足
  - プロンプト変化に追随

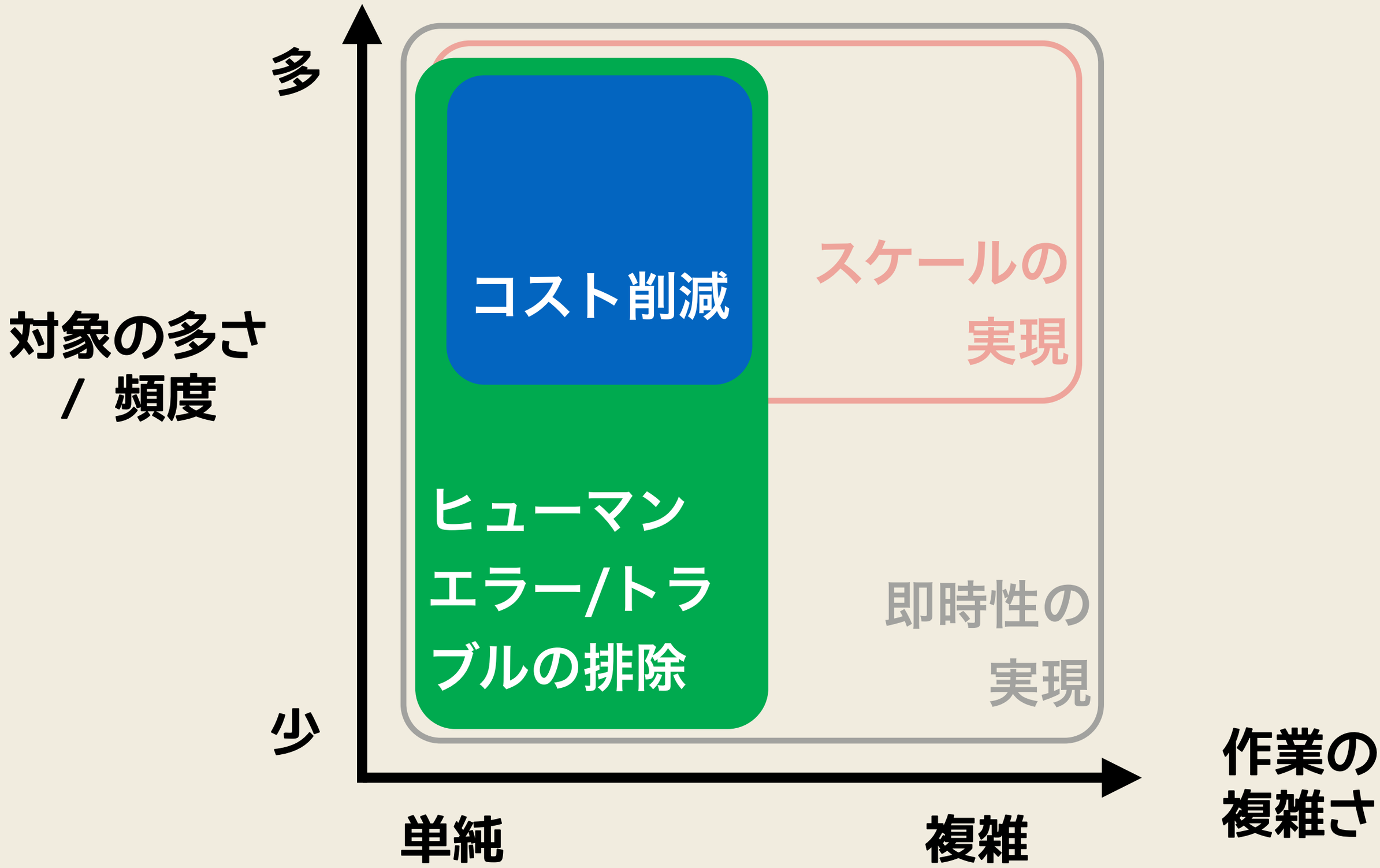
さて、一方

JPNAP では

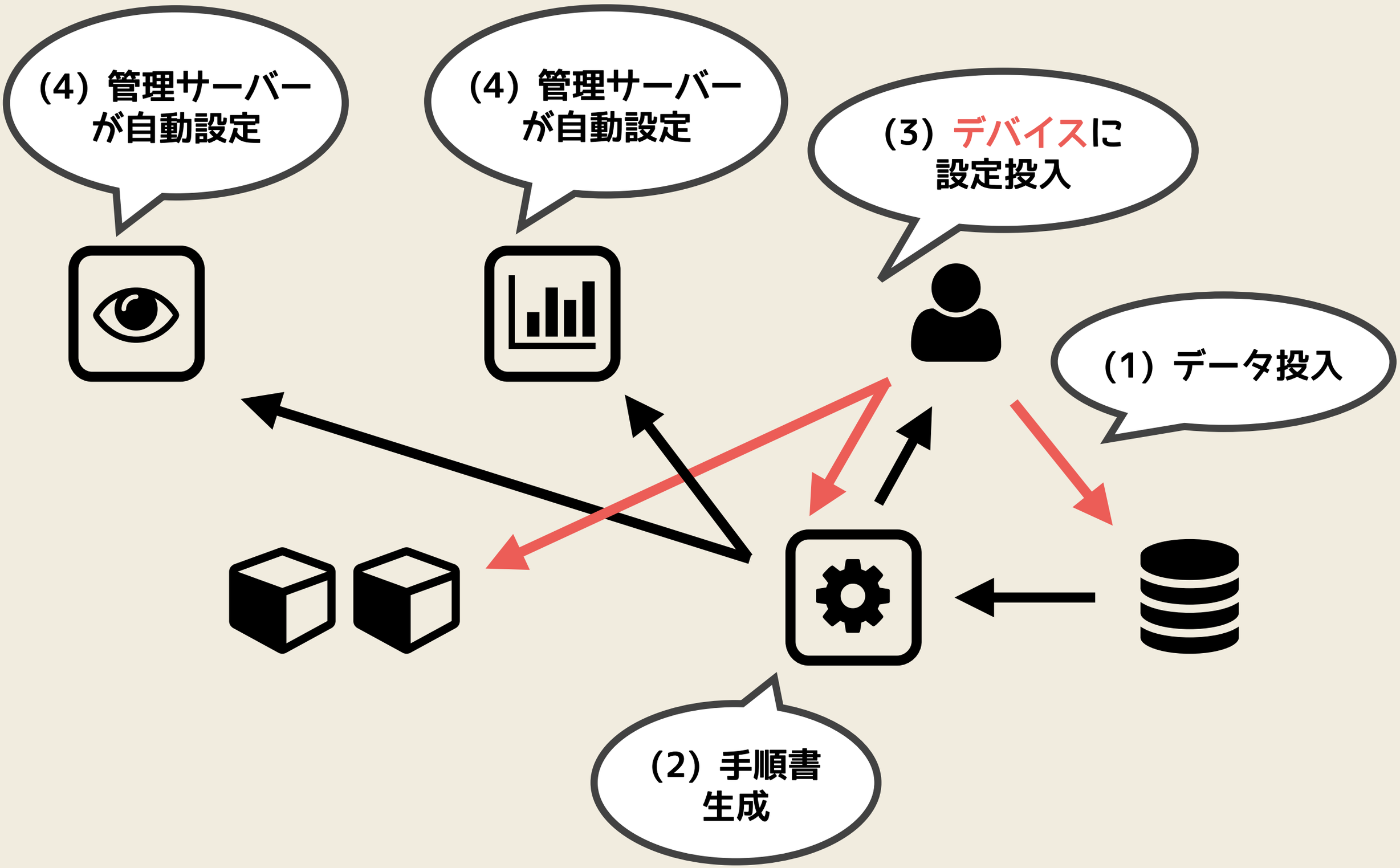
JPNAP

- 日本の IX
- サービスの安定提供が重要

どこを自動化する？


多
対象の多さ
/ 頻度
少
単純
複雑
作業の
複雑さ
コスト削減
スケールの
実現
ヒューマン
エラー/トラ
ブルの排除
即時性の
実現

デバイス制御以外を自動化



管理サーバー

ntt.net の
管理サーバーと
ぜんぜん違う

- 直接デバイスを制御する代わり、手順書を出力
  - エラー捕捉はオペレーターにまかせていい
  - わかりやすい自然言語にする必要あり
- 作業ステップを細かく分け、ステップ単位で自動化
  - オペレーター向けコマンド(CLI)

目的が違うと
まねても
うまくいかない

じゃあ、

どうすればいいの？

部品に注目する

- まねることから始めるのは重要
- でも、全体をまねてもうまくいかない

➡ 構成部品に注目して、自分たちのサービスにうまくハマるところだけを取り込む

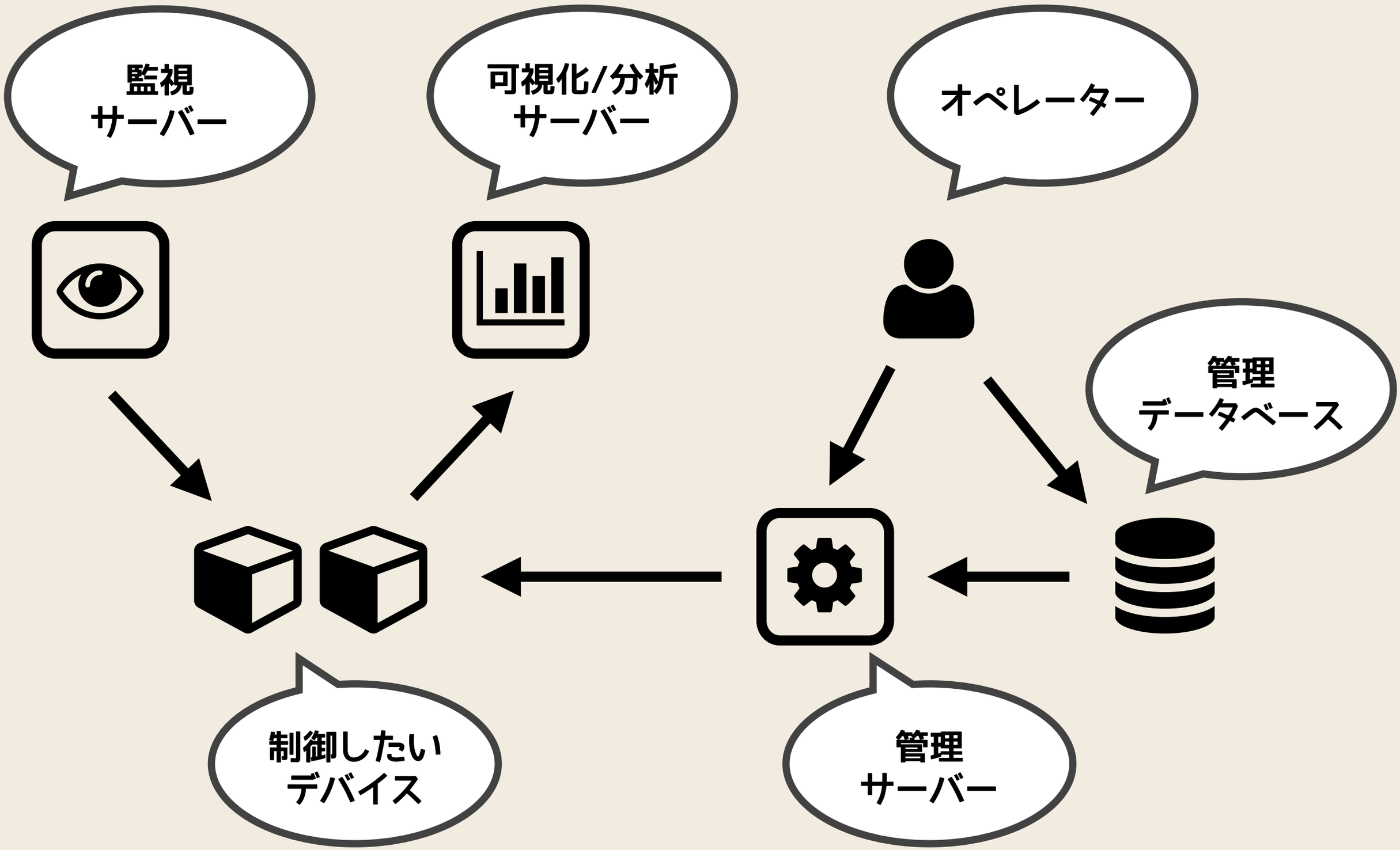
連携させる

- 部品間で通信する
- なるべくであれば、部品を交換可能に
  - いつか変更が必要になる
  - 必要になったとき、変更しやすい
    - 手を入れる範囲が小さい ➡ リスク低減
    - 部品単位でのテストができる
- インターフェイス (API) を決める

部品とAPIを

作ってみる

ネットワーク運用のコンポーネント



STEP #1: データベース

監視

可視化/分析

オペレーター

オペレーター用のGUIと API を持つ

制御したい
デバイス

管理
サーバー

自作する

- マルチベンダー環境で使える製品がない
  - ベンダー製のものはあるが、該当ベンダー専用
- サードパーティ製
  - 👍 マルチベンダー対応のものがある
  - 👎 API がない
  - 👎 拡張性が低い

Ruby on Rails + PostgreSQL

- IP Address / Prefix / MAC Address データ型がある！
- パフォーマンスを必要としない
- メンテナンス性を重視するべき
  - 慣れているか
  - 社内標準はあるか

プログラミング言語の選定

- まず、慣れている言語を選ぼう
- サーバー
  - LL に一日の長がある
    - 書ける人が多く、メンテナンスしやすい
    - 成熟したORM, Web フレームワーク
      - Go, Node より書きやすい
  - Ruby, Python, PHP, Perl, …
- クライアント
  - JavaScript

データベースの選定

- まず、慣れているデータベースを選ぼう
- RDBMS がいい
- データの整合性を維持したい
- パフォーマンスを必要としない
- PostgreSQL vs. MySQL
  - DigitalOcean の比較 が参考になる
  - バージョンによりふるまいが異なる場合があるので注意

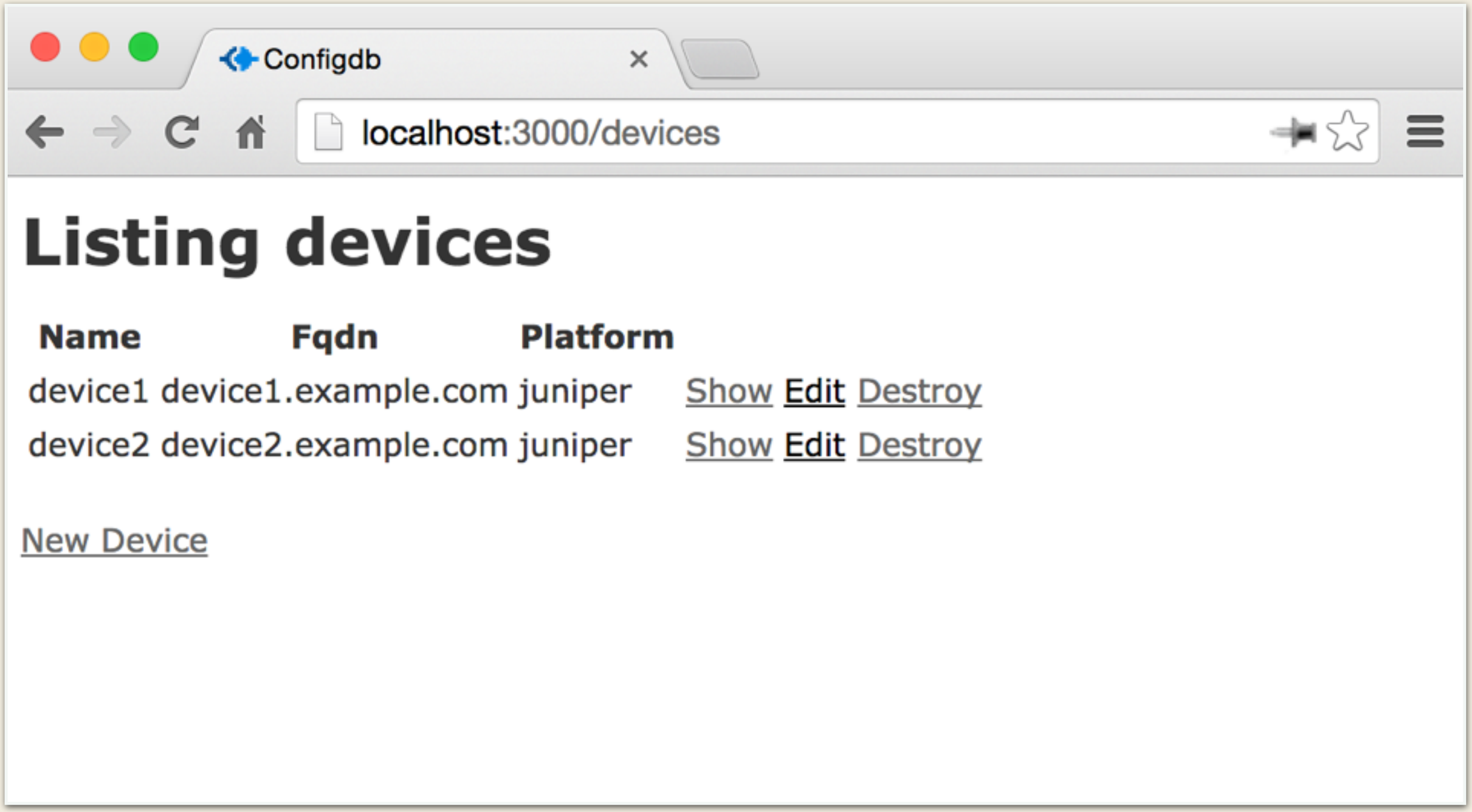
Device

https://github.com/codeout/iw2014_demo

```
$ gem install rails
$ rails new configdb --database=postgresql
$ cd configdb

$ rails g scaffold device name:text fqdn:text platform:text

$ rake db:create db:migrate
$ rails s

# tag v1.0
```

この時点での状態に
タグを打ちました

Device


Configdb
localhost:3000/devices
Listing devices
Name Fqdn Platform
device1 device1.example.com juniper Show Edit Destroy
device2 device2.example.com juniper Show Edit Destroy
New Device

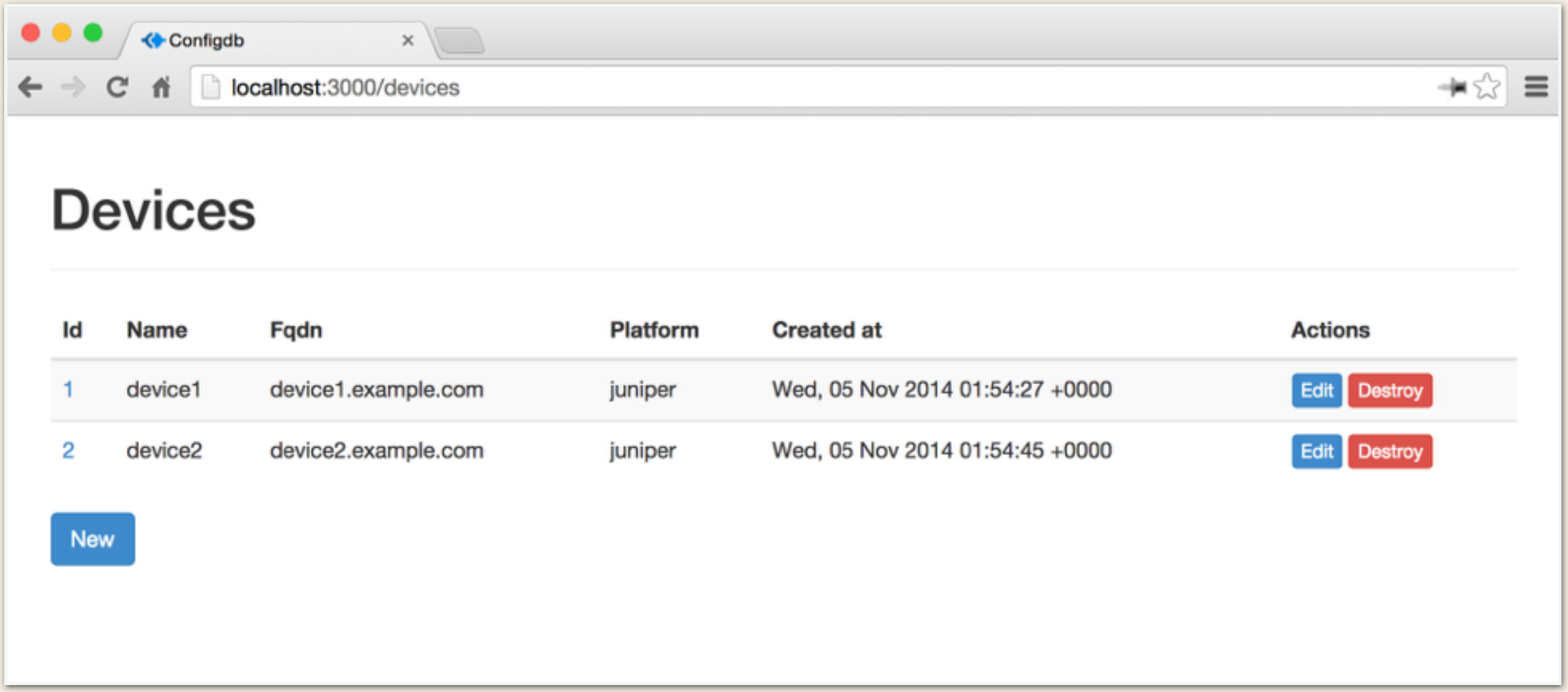
JSON REST API !

```
$ curl http://localhost:3000/devices.json | jq .

[
  {
    "id": 1,
    "name": "device1",
    "fqdn": "device1.example.com",
    "platform": "juniper",
    "url": "http://localhost:3000/devices/1.json"
  },
  {
    "id": 2,
    "name": "device2",
    "fqdn": "device2.example.com",
    "platform": "juniper",
    "url": "http://localhost:3000/devices/2.json"
  }
]
```

Device

```
$ vi Gemfile

+gem 'twitter-bootswatch-rails'
+gem 'twitter-bootswatch-rails-helpers'
+gem 'therubyracer'

$ bundle

$ rails g bootswatch:install flatly
$ rails g bootswatch:themed Devices

$ vi app/assets/javascripts/application.js

+//= require flatly/loader
+//= require flatly/bootswatch

$ vi app/assets/stylesheets/application.css

+ *= require flatly/loader
+ *= require flatly/bootswatch

$ rails s                                          # tag v1.1
```

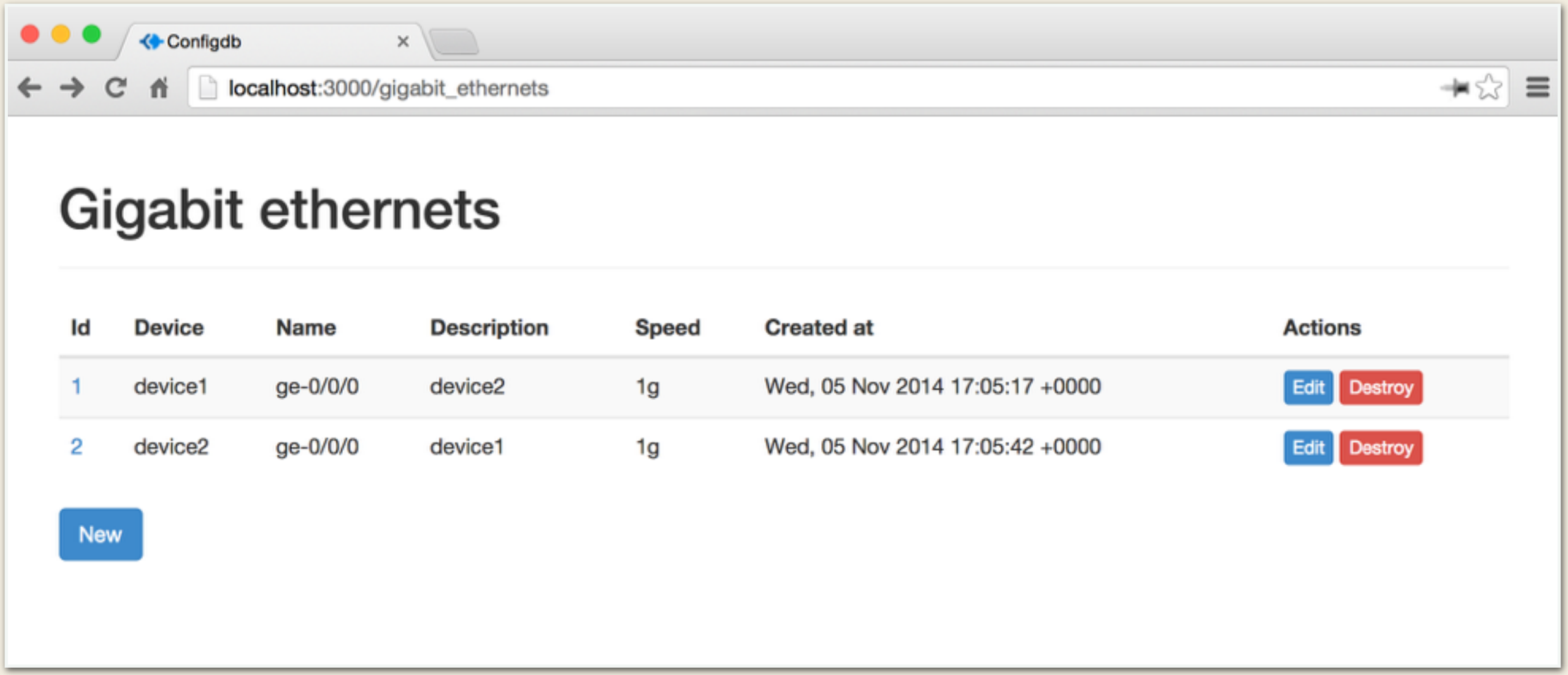
Device


Configdb
localhost:3000/devices
Devices
Id
Name
Fqdn
Platform
Created at
Actions
1
device1
device1.example.com
juniper
Wed, 05 Nov 2014 01:54:27 +0000
Edit
Destroy
2
device2
device2.example.com
juniper
Wed, 05 Nov 2014 01:54:45 +0000
Edit
Destroy
New

Interface

```
$ rails g scaffold gigabit_ethernet device:belongs_to
name:text description:text speed:text

$ rake db:migrate
$ rails g bootswatch:themed GigabitEthernets

# 表示部分 3行修正

$ rails s

# tag v1.2
```

Interface

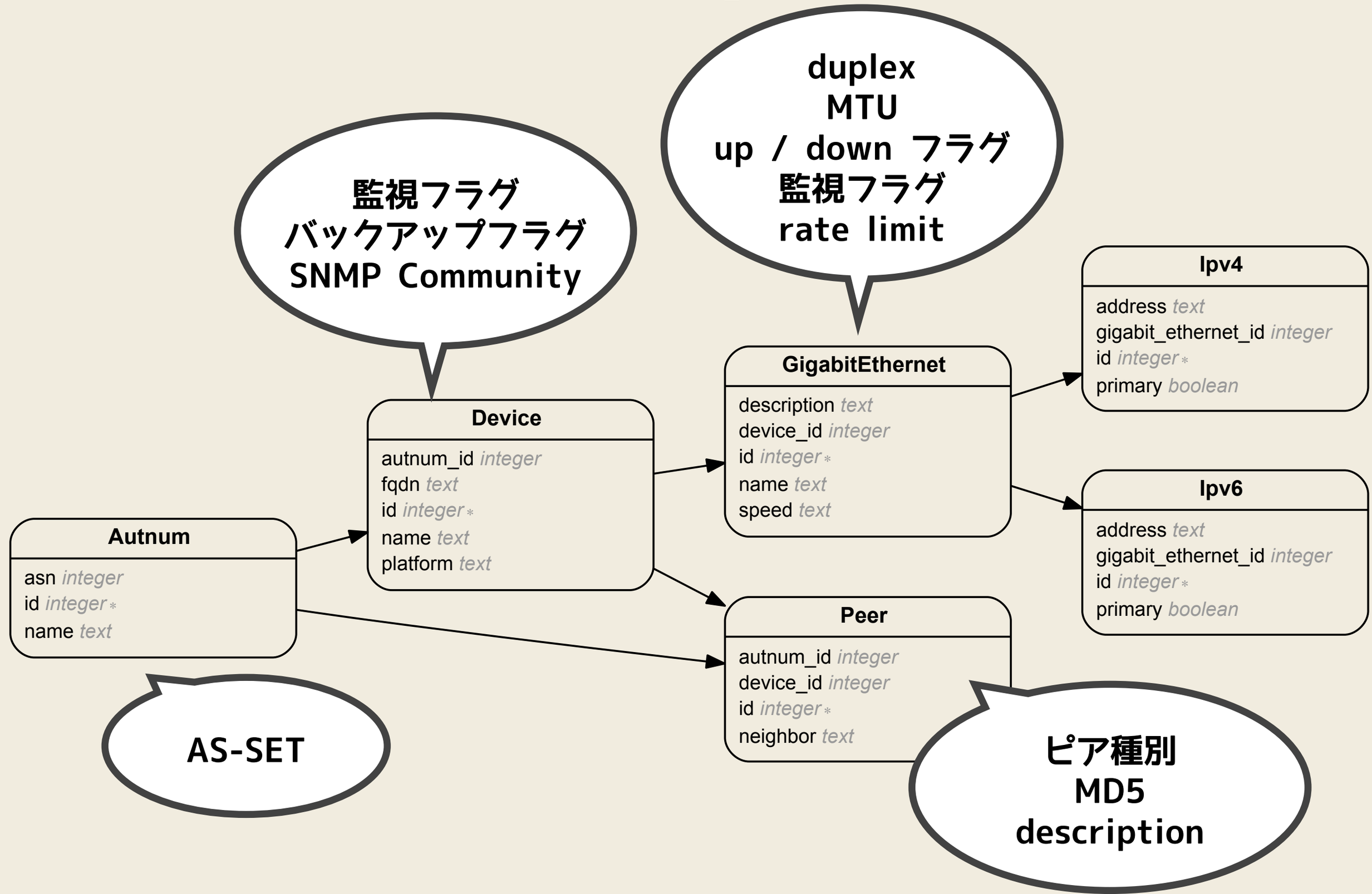

Configdb
localhost:3000/gigabit_ethernets
Gigabit ethernets
Id
Device
Name
Description
Speed
Created at
Actions
1
device1
ge-0/0/0
device2
1g
Wed, 05 Nov 2014 17:05:17 +0000
Edit
Destroy
2
device2
ge-0/0/0
device1
1g
Wed, 05 Nov 2014 17:05:42 +0000
Edit
Destroy
New

どんどん作る

- Ipv4
- Ipv6
- Autnum
- Peer

(tag v1.4)

ER図


監視フラグ
バックアップフラグ
SNMP Community
duplex
MTU
up / down フラグ
監視フラグ
rate limit
Autnum
asn integer
id integer*
name text
Device
autnum_id integer
fqdn text
id integer*
name text
platform text
GigabitEthernet
description text
device_id integer
id integer*
name text
speed text
Ipv4
address text
gigabit_ethernet_id integer
id integer*
primary boolean
Ipv6
address text
gigabit_ethernet_id integer
id integer*
primary boolean
Peer
autnum_id integer
device_id integer
id integer*
neighbor text
AS-SET
ピア種別
MD5
description

データモデル

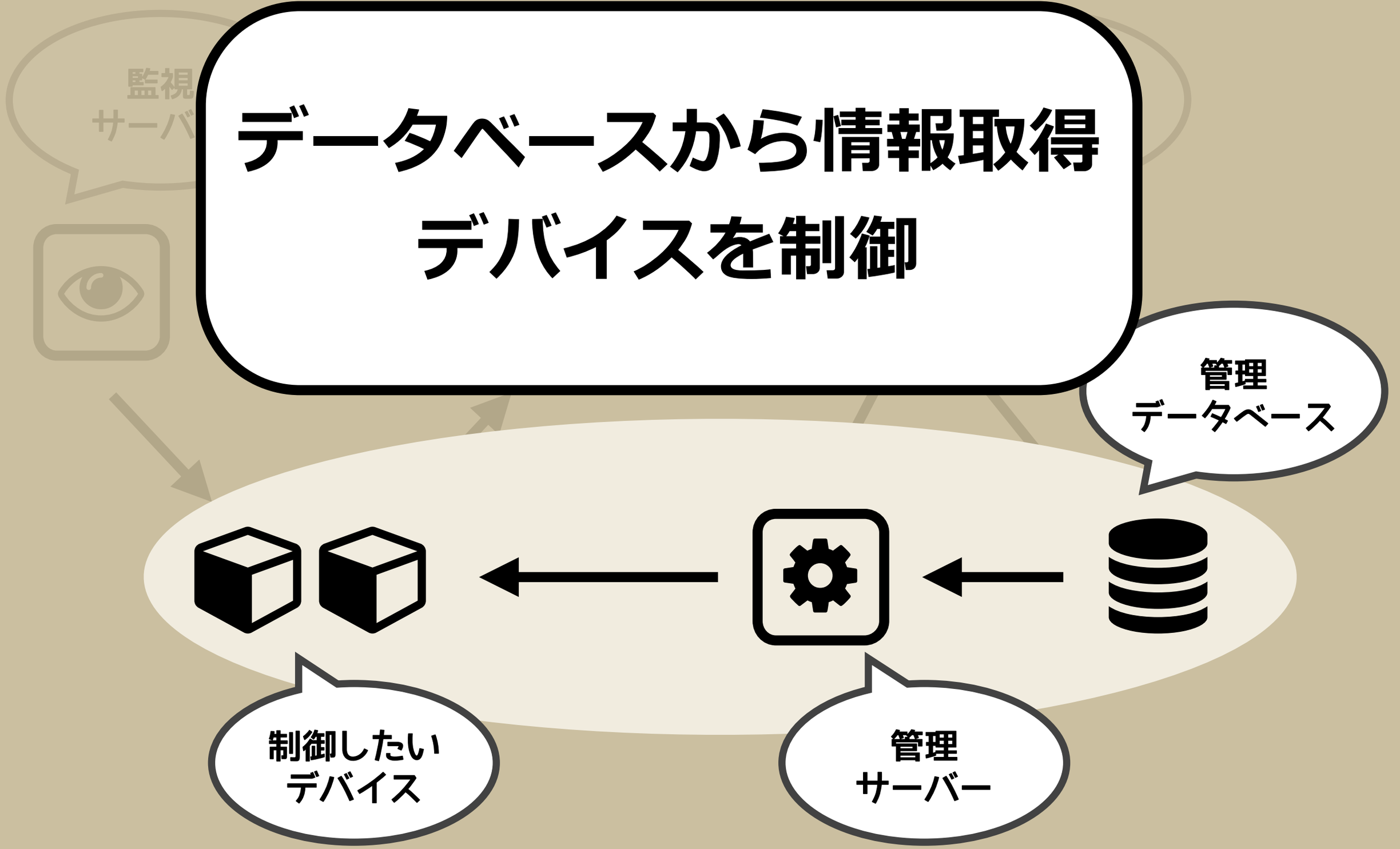
- いままでの形だと検討不足
- 標準化されているデータモデルがある
  - YANG Model
    - IETF NETCONF Data Modeling Language WG
      - “A YANG Data Model for Interface Management” (RFC 7223)
      - “A YANG Data Model for IP Management” (RFC 7277)
      - “A YANG Data Model for System Management” (RFC 7317)

- 属性名や構造など、YANG Model に合わせておく
  - やりすぎ注意！

メンテナンス性が
下がる

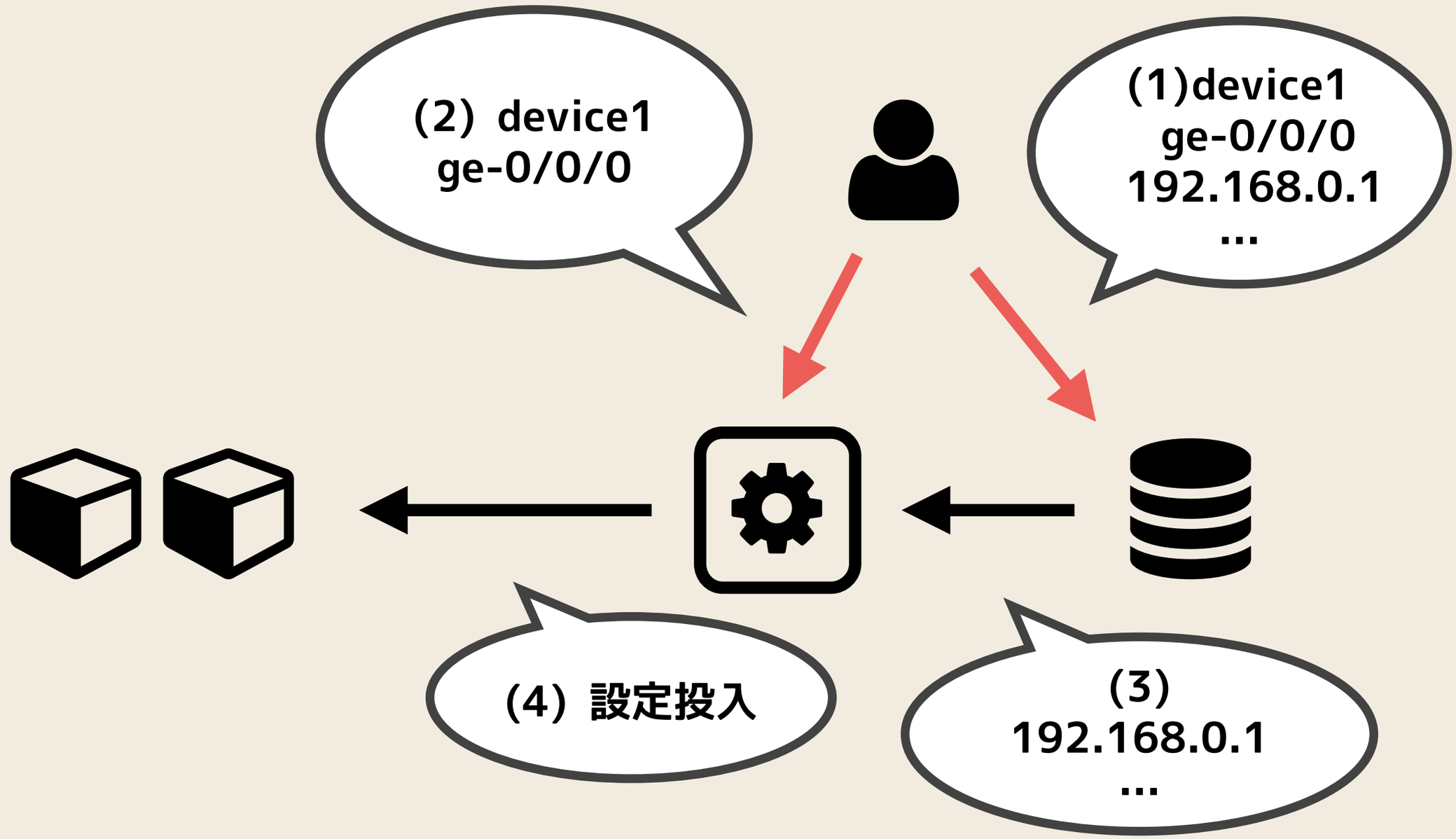
STEP #2: デバイス制御

データベースから情報取得
デバイスを制御



パラメーターを
データベースから取得

(2) で渡す情報は “キー” だけ



Python によるREST API Client

- rest_client.py

```
1 import json
2 from httplib import HTTPConnection
3
4 session = HTTPConnection('localhost:3000')
5 session.request('GET', '/devices.json')
6
7 response = json.load(session.getresponse())
8 print(json.dumps(response))
```

Python によるREST API Client

```
$ python rest_client.py | jq .

[
  {
    "id": 1,
    "name": "device1",
    "fqdn": "device1.example.com",
    "platform": "juniper",
    "url": "http://localhost:3000/devices/1.json"
  },
  {
    "id": 2,
    "name": "device2",
    "fqdn": "device2.example.com",
    "platform": "juniper",
    "url": "http://localhost:3000/devices/2.json"
  }
]
```

Server (Ruby) — Client (Python) だが、
ぜんぜん大丈夫

STEP #3: 監視

監視
サーバー

可視化/分析
サーバー

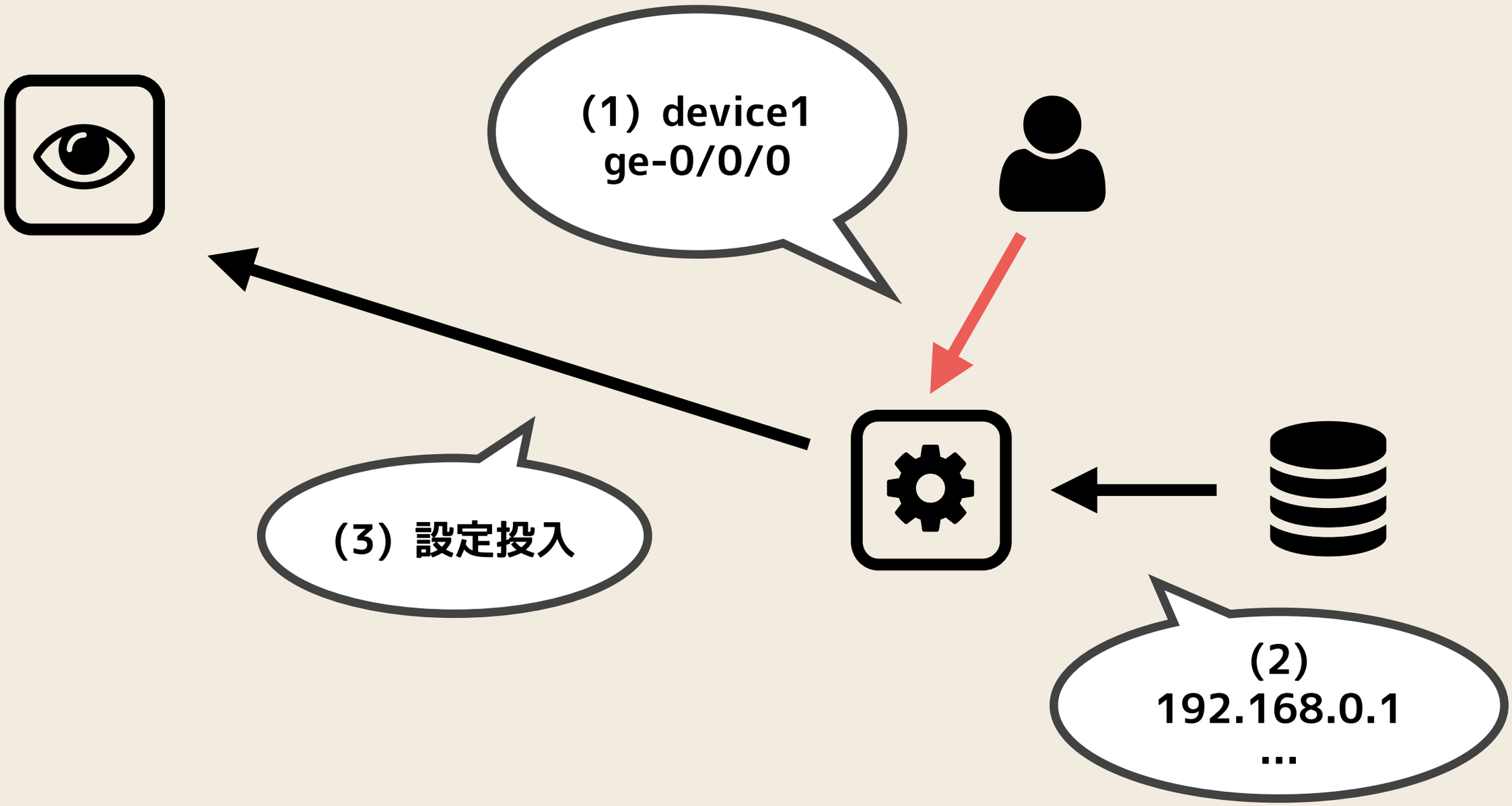
オペレーター

監視項目を自動設定する

制御したい
デバイス

管理
サーバー

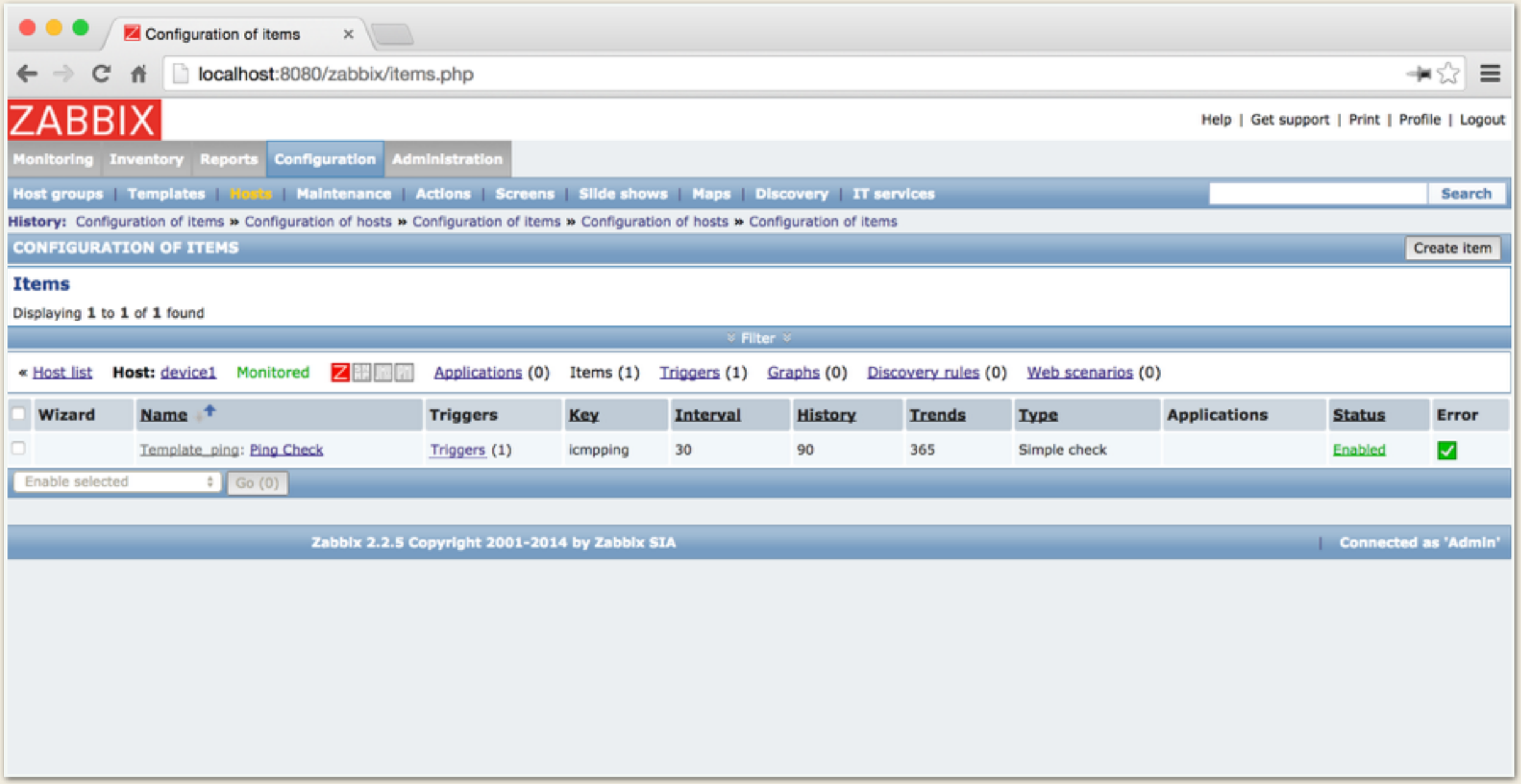
監視項目の自動投入



監視ツール

	設定用API	状態取得用API
Zabbix	○	○
Nagios	✖	○ (プラグイン)
Icinga	✖	○
Sensu	○	○

Zabbix


Configuration of items
localhost:8080/zabbix/items.php
ZABBIX
Help | Get support | Print | Profile | Logout
Monitoring
Inventory
Reports
Configuration
Administration
Host groups | Templates | Hosts | Maintenance | Actions | Screens | Slide shows | Maps | Discovery | IT services
Search
History: Configuration of items » Configuration of hosts » Configuration of items » Configuration of hosts » Configuration of items
CONFIGURATION OF ITEMS
Create item
Items
Displaying 1 to 1 of 1 found
Filter
« Host list Host: device1 Monitored Applications (0) Items (1) Triggers (1) Graphs (0) Discovery rules (0) Web scenarios (0)
Wizard
Name
Triggers
Key
Interval
History
Trends
Type
Applications
Status
Error
Template_ping: Ping Check
Triggers (1)
icmpping
30
90
365
Simple check
Enabled
Enable selected
Go (0)
Zabbix 2.2.5 Copyright 2001-2014 by Zabbix SIA
Connected as 'Admin'

Zabbix: 監視項目追加

JSON メッセージをHTTP POST する

```
{
  "params": {
    "interfaceid": "3",
    "hostid": "10105",
    "key_": "icmpping[192.168.0.1]",
    "name": "ping to ge-0/0/1",
    "delay": 30,
    "value_type": 3,
    "type": 0
  },
  "jsonrpc": "2.0",
  "method": "item.create",
  "auth": "580ef2d1e3d7db0f2962bca17de415d1",
  "id": 2
}
```

pyzabbix で実装

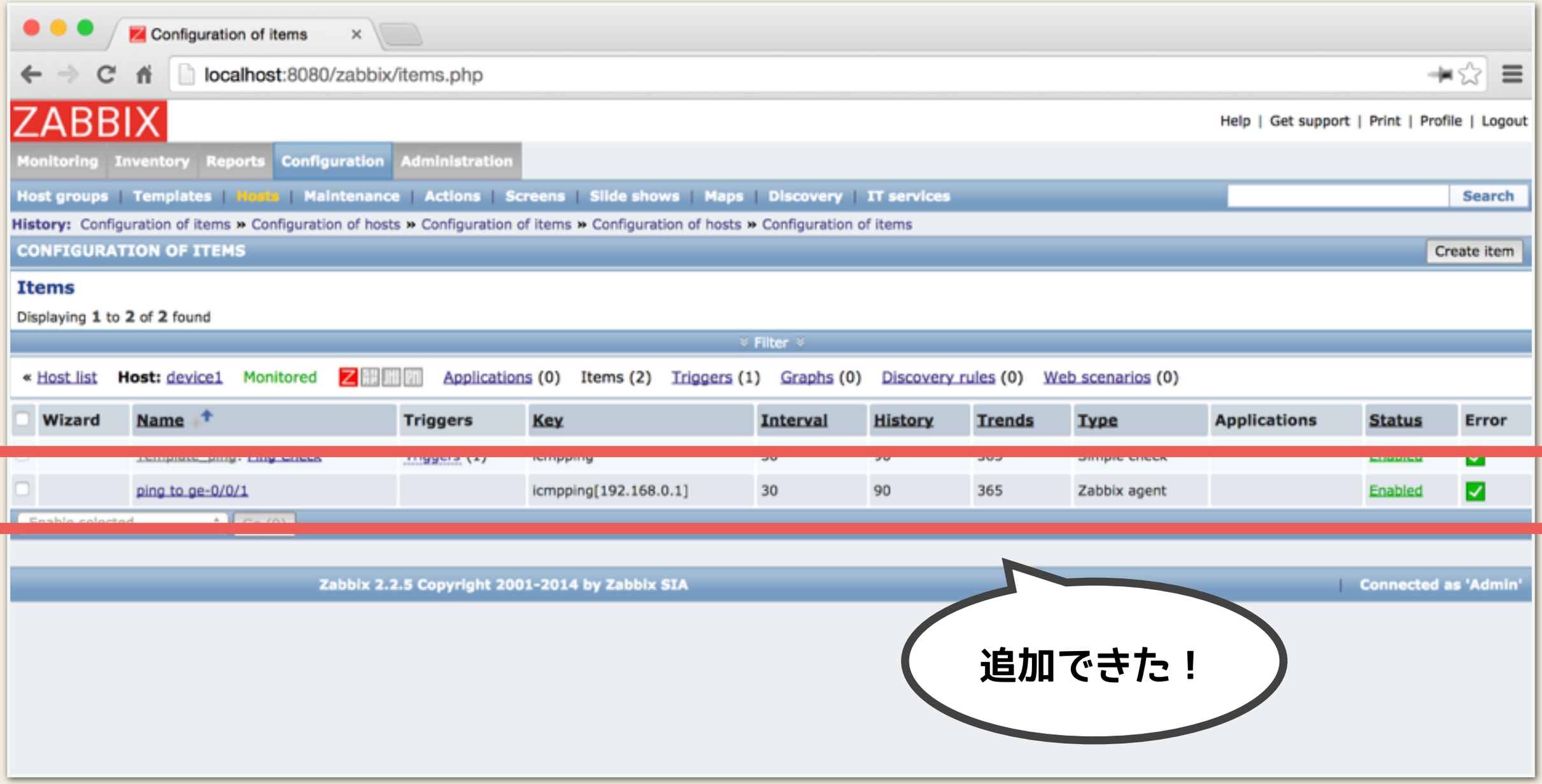
```
45 device = 'device1'
46 interface = 'ge-0/0/1'
47
48 device_id = Device(device).device_id()
49 interface_id = GigabitEthernet(device_id, interface).gigabit_ethernet_id()
50 addresses = Ipv4(interface_id).addresses()
51
52 api = ZabbixAPI('http://localhost:8080/zabbix')
53 api.login('admin', 'zabbix')
54
55 hosts = api.host.get(filter={'name': 'device1'}, selectInterfaces=['interfaceid'])
56
57 if hosts:
58   api.item.create(
59     hostid = hosts[0]['hostid'],
60     name = 'ping to %s' % interface,
61     key_ = 'icmpping[%s]' % addresses[0] ,
62     type = 0,         # Numeric (unsigned)
63     value_type = 3,  # Decimal
64     interfaceid = hosts[0]['interfaces'][0]['interfaceid'],
65     delay = 30
66   )
67
68 # tag v2.0
```

DB から
情報取得

JSON をPOST

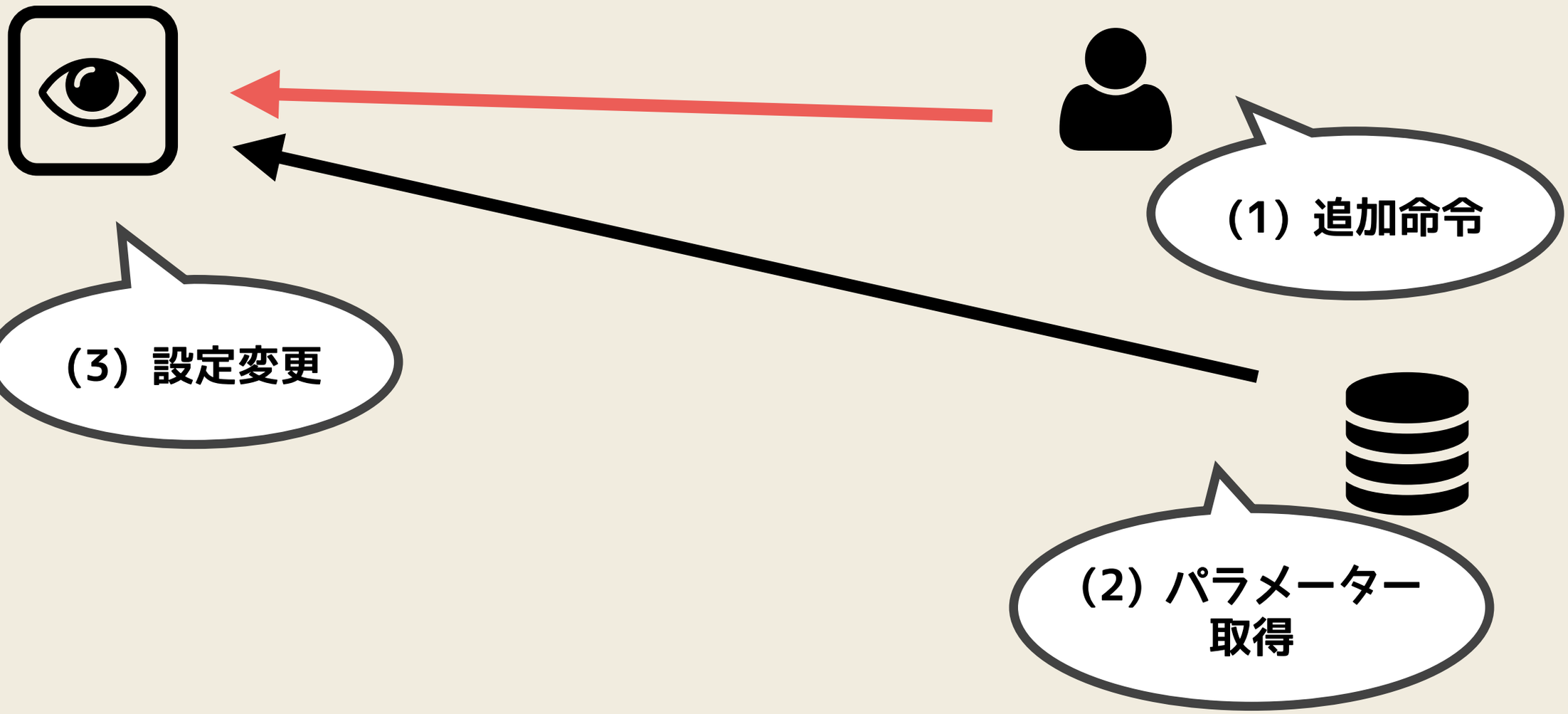
add_item.py

Zabbix: 監視項目追加



Nagios / Icinga:
監視項目追加

ローカルファイルを編集する必要がある



Nagios / Icinga:
パラメーター取得 + 設定作成

- config.rb

```
31 Device.all.each do |device|
32   interface = GigabitEthernet.find_by_device_id(device.id)
33   next unless interface
34
35   ipv4 = Ipv4.find_by_gigabit_ethernet_id(interface.id)
36   puts ERB.new(File.read('template.conf.erb')).result(binding)
37 end
```

- template.conf.erb

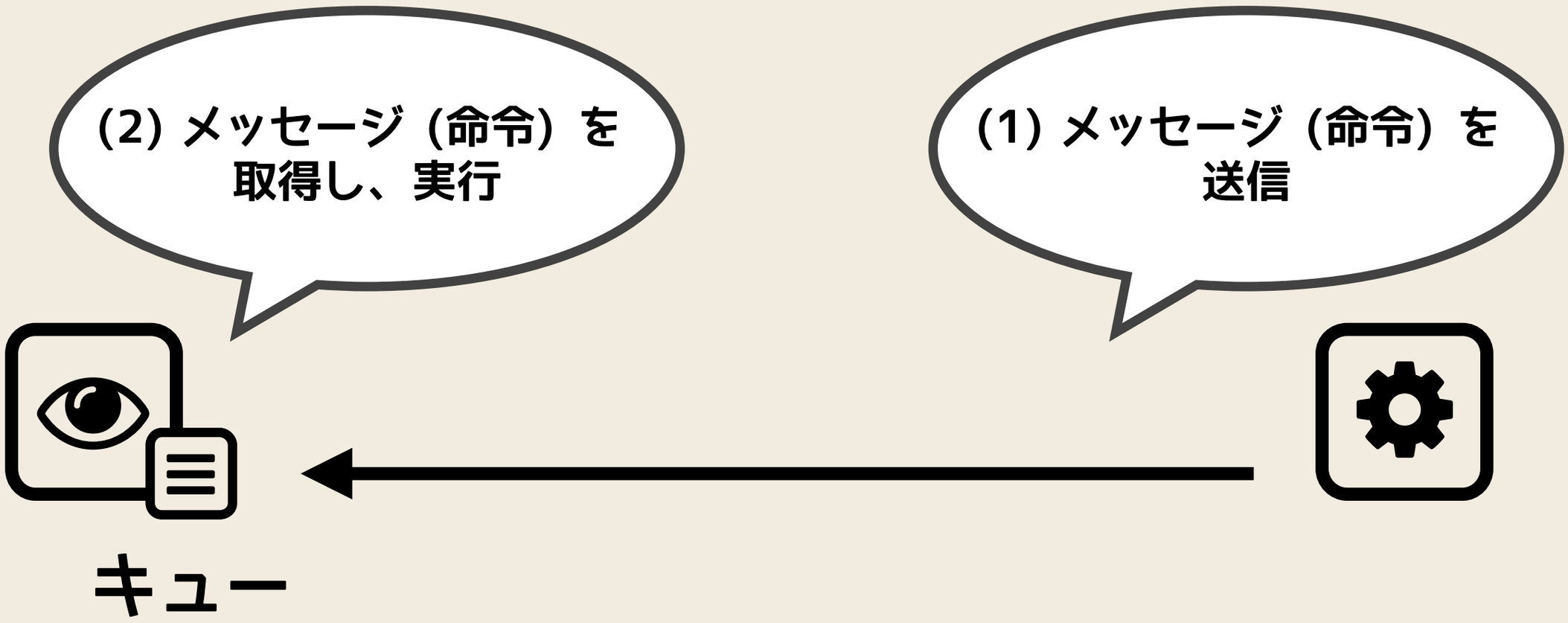
```
 1 object Host "<%= device.name %>" {
 2   import "generic-host"
 3
 4   address = "<%= ipv4.address %>"
 5 }
 6
 7 object Service "ping" {
 8   import "generic-service"
 9
10   host_name = "<%= device.name %>"
11   check_command = "ping4"
12 }                                                    # tag v3.0
```

ProTip: バージョン管理

- 設定ファイルなど、テキストデータを更新するたびにバージョン管理する
  - 設定の履歴を保存しておくとトラブル時に役立つ
  - 即時/安全にロールバックできる手段として

API 化したい:
メッセージキューを使う

- Sidekiq
- Resque
- RabbitMQ
- …



Nagios / Icinga: Sidekiq Server

- sidekiq_server.rb

```ruby
require 'sidekiq'

Sidekiq.configure_server do |config|
  config.redis = {namespace: 'icinga'}
end

class Icinga
  include Sidekiq::Worker

  def perform(command)
    case command
    when 'reload'
      # 設定更新 + Icinga リロード処理
    end
  end
end
```

メッセージ（命令）ごとの処理を記述

シリアルに処理

```
$ sidekiq -r ./sidekiq_server.rb -c 1
```

tag v4.0

Nagios / Icinga: Sidekiq Client

- sidekiq_client.rb

```
 1 require 'sidekiq'
 2
 3 Sidekiq.configure_client do |config|
 4   config.redis = {url: 'http://server.example.com:6379',
       namespace: 'icinga', size: 1}
 5 end
 6
 7 class Icinga
 8   include Sidekiq::Worker
 9 end
10
11 Icinga.perform_async(:reload)

# tag v4.0
```

サーバー側の
キューを指定

メッセージ (命令)
送信

STEP #4: ChatOps

監視
サーバー

可視化/分析
サーバー

オペレーター

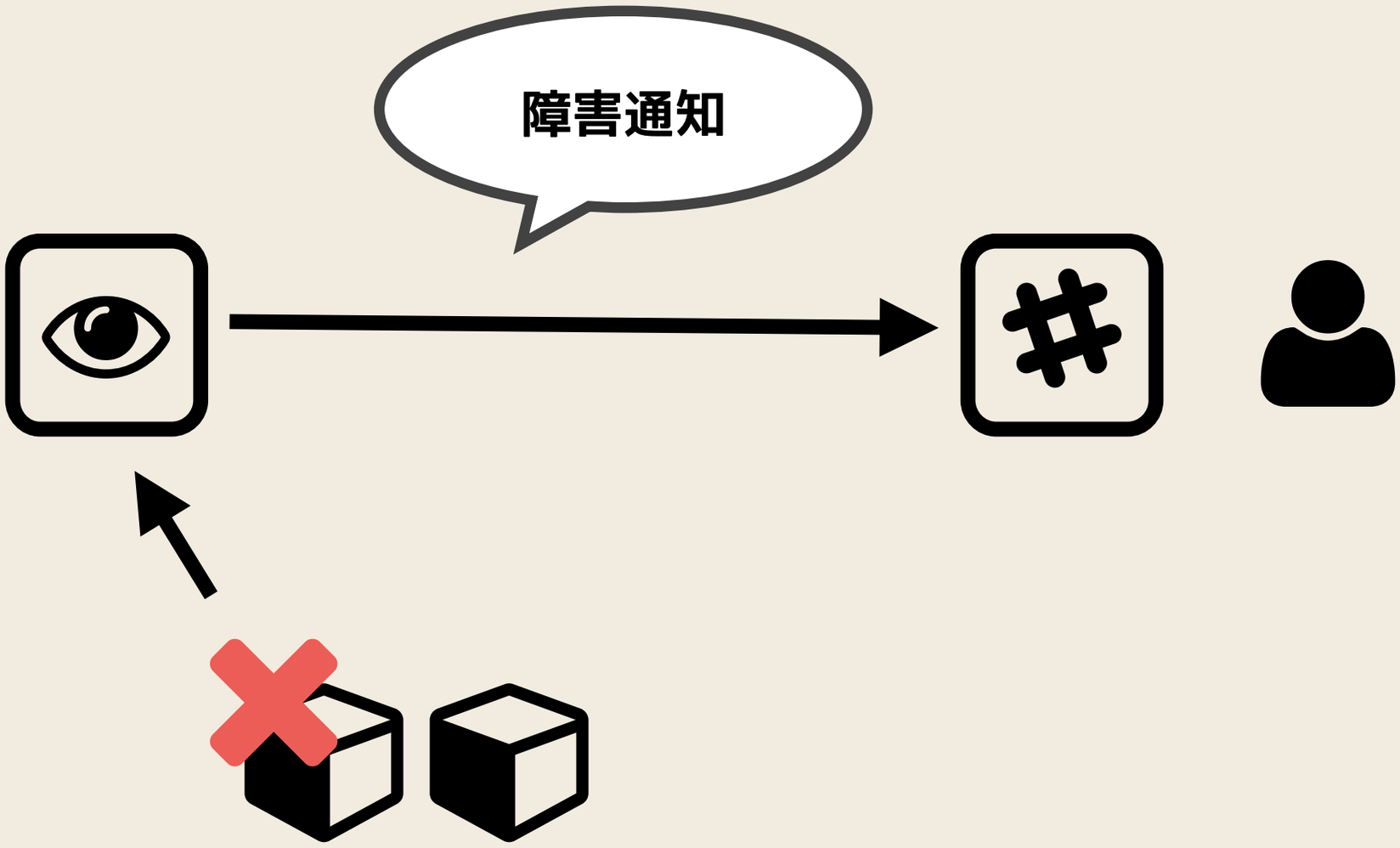
イベントをトリガに

決まった作業を実行し、情報を流す

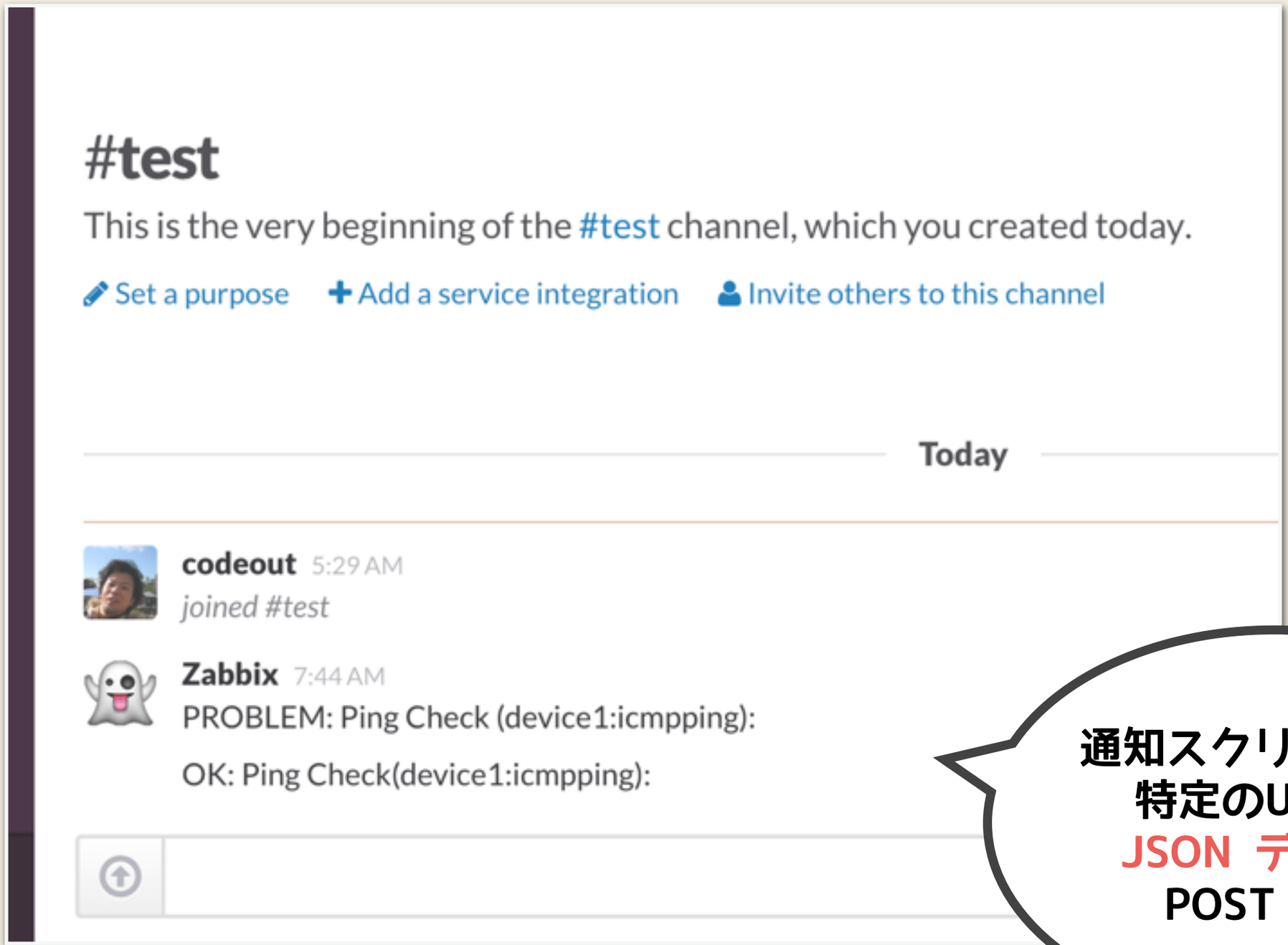
制御したい
デバイス

管理
サーバー

ChatOps:
イベントをチャットに流す

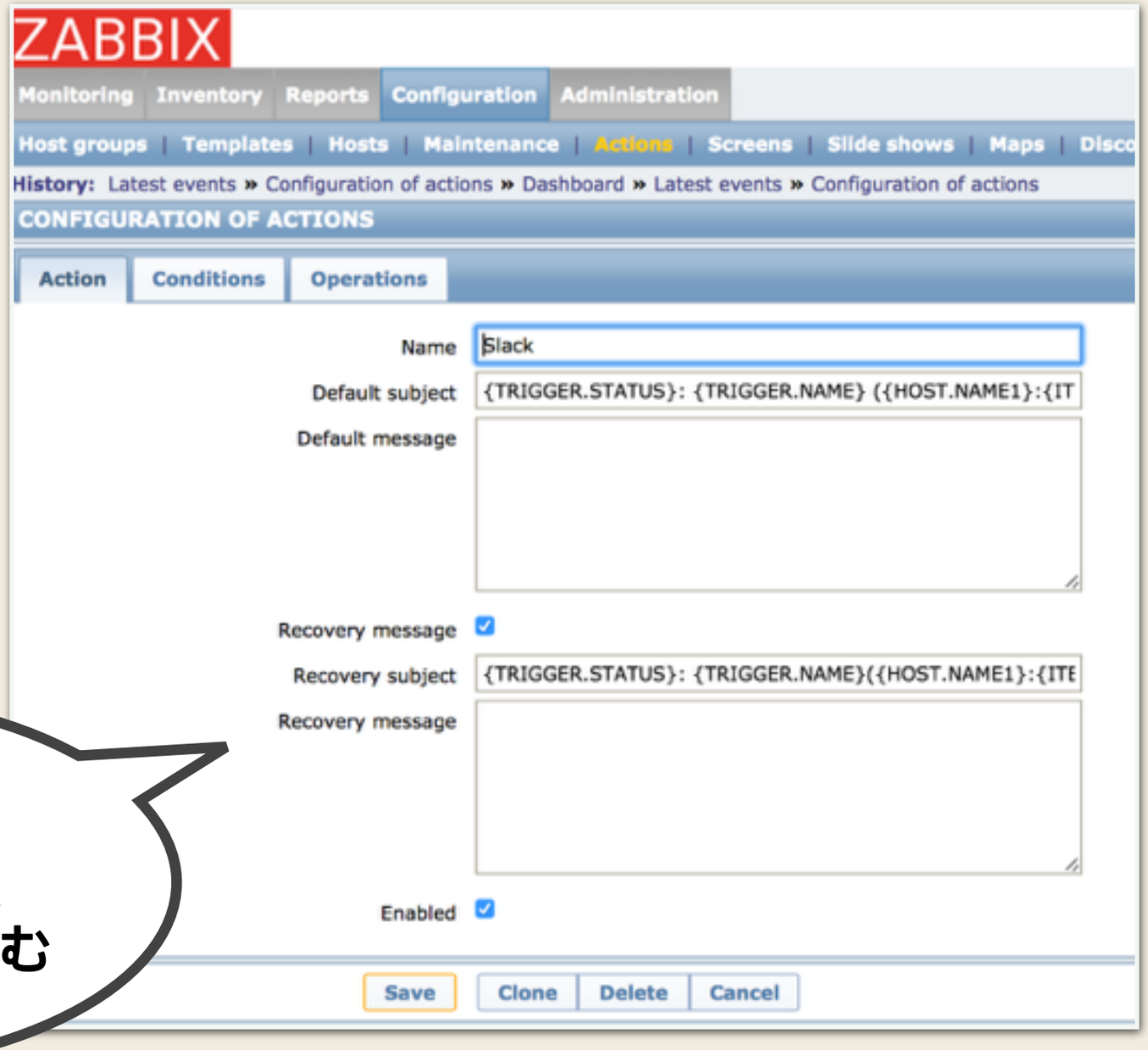


Slack


#test
This is the very beginning of the #test channel, which you created today.
Set a purpose
Add a service integration
Invite others to this channel
Today
codeout 5:29 AM
joined #test
Zabbix 7:44 AM
PROBLEM: Ping Check (device1:icmpping):
OK: Ping Check(device1:icmpping):
通知スクリプトから
特定のURL に
JSON データを
POST する

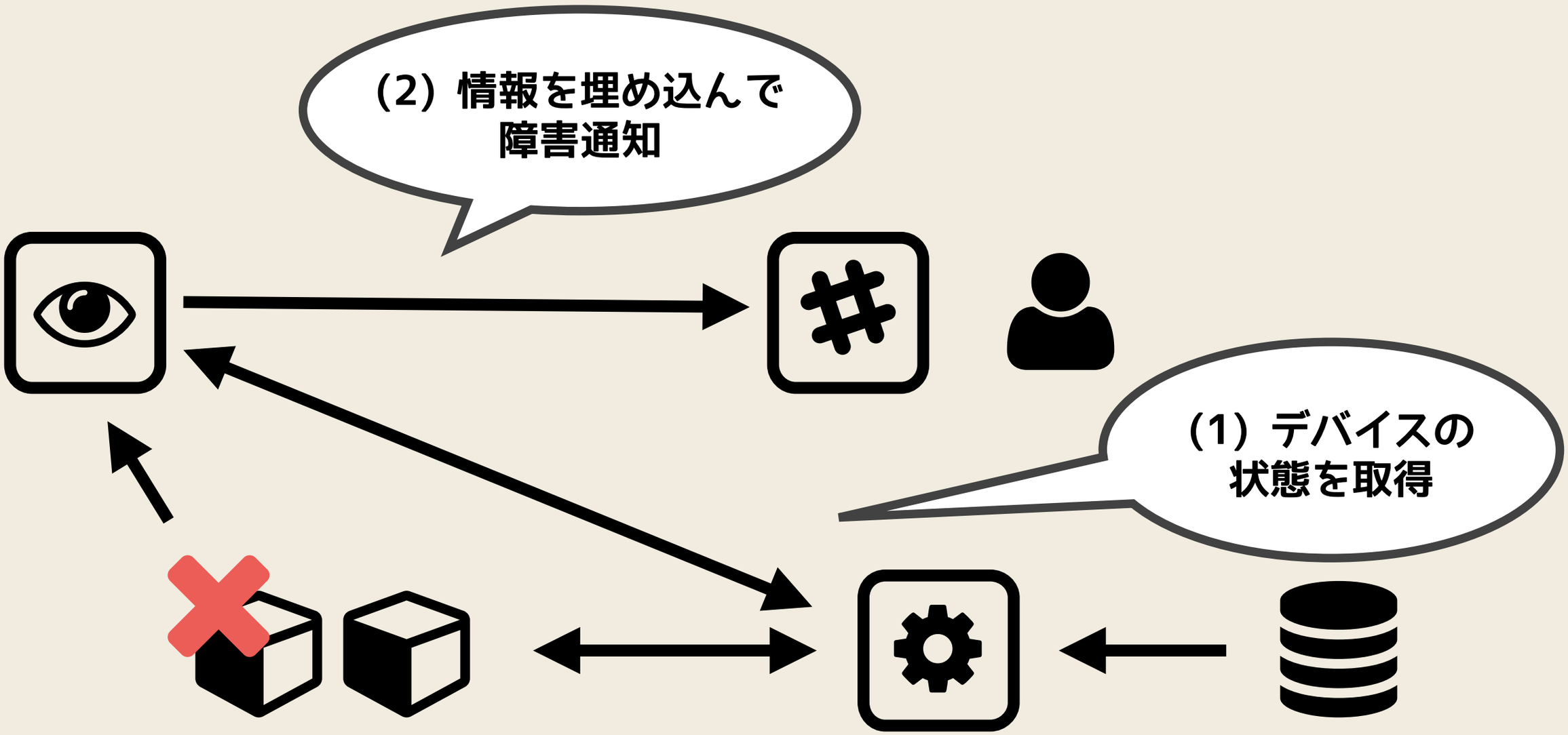
zabbix-slack-alertscript

https://github.com/ericoc/zabbix-slack-alertscript



チャットが
流れすぎないよう、
1行に情報を詰め込む

デバイスの状態を取得するAPI

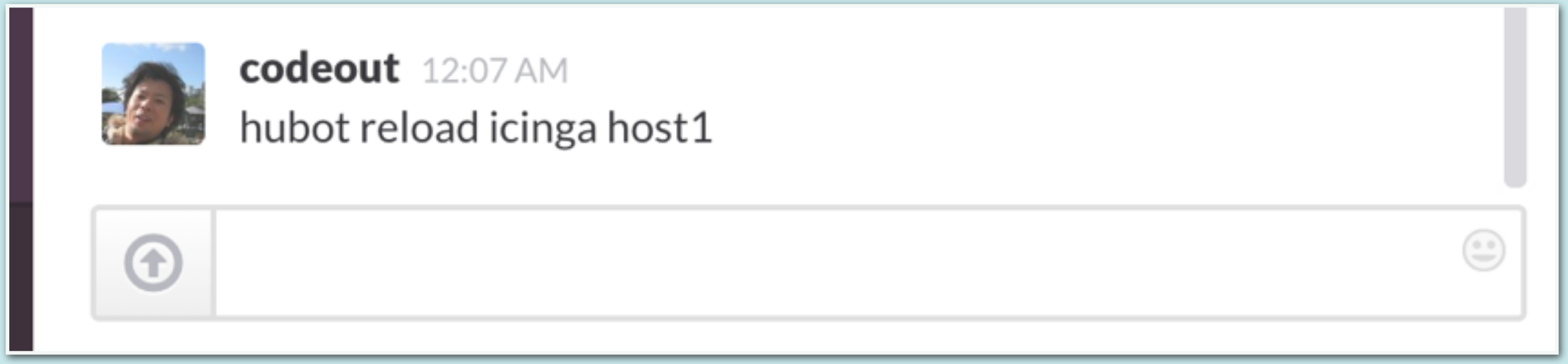


例: デバイスのバージョンを取得するAPI

```
$ curl http://localhost:3000/devices/3/status/platform.json | jq .
{
  "machine": "firefly-perimeter",
  "os-name": "JUNOS",
  "os-version": "12.1X46-D10"
}

# tag 5.0
```

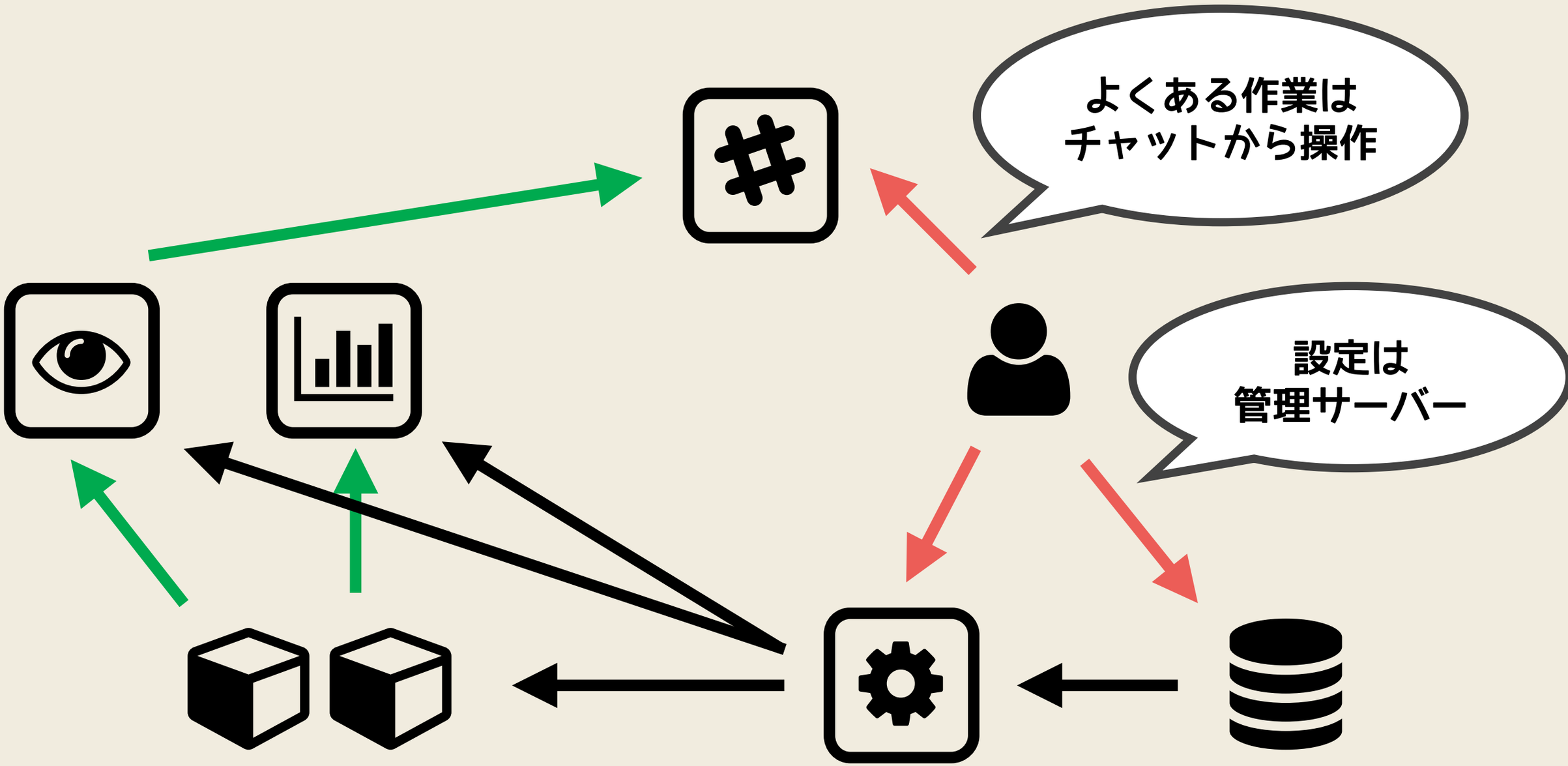
このようなAPIを作る

ProTip: チャットから操作



- チャットに、特定のメッセージ (命令) を書き込むことで運用システムを操作
  - hubot
  - チャットサービスに組み込まれている場合も
    - 例: Outgoing Webhooks (Slack)

いきなりスゴいのを目指したらダメ



目的に沿っていて、
うまく運用にハマりそう
なものから、少しずつ
連携させてみてください

連携し始めたら、
APIが変わってないか
テストしてください

運用支援システムを作るコツ

- 小さい部品を連携させる
- 部品は交換できるように
- API を決める
- テキストで手に入るものはバージョン管理
- 運用と噛み合っているか確かめながら、少しずつ
- API のテストを忘れずに

Questions ?