SDDCファシリティ記述向けDSLの設計に向けて

株式会社クルウィット
井澤 志充

2014/11/20 (Thu)
DataCenter とソフトウェア開発ワークショップ
@ 石川県ハイテク交流センター

今日のおはなし

- DCファシリティをコンピュータで管理/制御するために
- どうやってDCファシリティを記述していくか
  - DC構成要素のはなし
  - 要素間の関係の整理
  - 関係の記述について

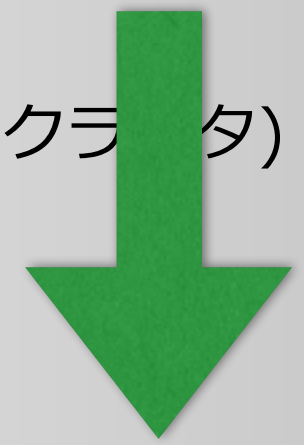
DCを構成する多彩な要素

- エリア/地域
- データセンター単位(e.g. コンテナ/ラッククラスタ)
- ラックの位置
- マウントされている機器
- 上記によって提供されているサービスなど

これらを計算機から管理/操作するためには
なんらかのデータセット化 + 操作インタフェースの定義
が必要

DCを構成する多彩な要素

- エリア/地域
- データセンター単位(e.g. コンテナ/ラッククラスタ)
- ラックの位置
- マウントされている機器
- 上記によって提供されているサービスなど



これらを計算機から管理/操作するためには
なんらかのデータセット化 + 操作インタフェースの定義
が必要

データセットの種類

- (比較的)Staticな情報
  - 機器のマウント情報
  - 機器間の結線情報
- Dynamicな情報
  - 通電状況
  - ネットワーク接続状況
  - ……

Staticな情報を記述するために(1)

- DCファシリティをデータセットとして記述するうえ「関係の定義」は必須
  - 包含関係
    - DC site ⊇ Rack cluster ⊇ Rack ⊇ ITs ⊇ etc.

- 人間が全体の接続関係を矛盾なく記述することは困難
  - そこで一対一対応の「関係」の記述を繰り返すモデル
  - ソフトウェアでそれらの関係から全体像を合成する

Staticな情報を記述するために(2)

- template (書く人に優しいもの)
  - ラックの構成や順序などは、どのラックもほぼ同じだったり似ていたりする。

- patch/stackable(部分書き換えをするもの)
  - 部分的にその他の情報と異なる部分に対する対処
    - 機器故障でそこだけ使わない等
  - 記述が(欠落|誤って)いるのか、あえて変えているのかの「意図」を表現しやすい

関係の種類

- 様々な「関係」について定義する必要がある
- 「つながっている」だけでは不十分
  - サービスの提供/利用の関係
    - Power supply
    - Network connectivity
  - グループ構成要素としての関係
    - 物理的
      - DC Region
      - mount position
    - 論理的
      - Customer resource

状態の種類

- 様々な「状態」について定義する必要がある
  - 依存関係の解決をした上での機器操作に必要
  - サービスの提供/利用の状態
    - Power supplyしている
    - Network connectivityを持っている
    - xxxサービスに依存している

これら「関係」「状態」を使ってこんなことをしたい

- たとえば IT サーバーを起動するときは…
  - 電源依存「関係」にあるUPSのうち少なくてもひとつは供給「状態」にある必要がある、とか。

- たとえば Rack まるまるダウンさせるときは…
  - Rack内部のサーバが提供しているサービスを(停止|マイグレーション)して、サーバを落とし、
  - ネットワークSWのVC関係を切断してから落とし、、(などなど)

関係定義に際しての注意点

- 人間がDCファシリティの関係を定義すると以下の問題が発生しやすい
  - 関係性に矛盾を入れ込んでしまう
  - 定義に「漏れ」が発生してしまう
- なぜならば、DC のファシリティは「量が多い」

解決するために

- 矛盾が発生しにくい仕組み＆チェックする仕組みが必要
  - ファシリティを記述するための DSL
  - 当該 DSL をチェックする Validator (not only Syntax check)
- 記述量を減らす仕組みが必要
  - Template & patch

Interface(アクション)

- Staticな情報はデータ構造として扱う
- データ構造に基づいた要素の操作はmethod(Interface)としてアクションを定義する
- 構造とアクションは分離して記述していけるようにする

DSL を作成するポイント

- 局所的な記述を可能にする
- 複数の「ファシリティ」と複数の「関係」を扱えるようにする
- 「ファシリティ」と「関係」の組み合わせを定義(限定)できるようにする

局所的な記述を可能にする

- ある2点のファシリティ間の関係を記述していく
  - (xml や yml のような) 全構造を一度に書き下す形と比べて、
  - 人間が記述しやすい書き方が可能になる
- Validator の実装が容易になる(かも)

複数の「ファシリティ」と複数の「関係」を扱えるようにする

- 例えば、あるITサーバーは…
  - あるUPS 2台から給電されている
  - あるスイッチからネットワークコネクティビティを供給されている
  - あるVMに計算リソースを供給している

関係に制約条件を設ける

- 「ファシリティ」と「関係」の組み合わせを定義(限定)できるようにする
  - ネットワークスイッチから給電してもらうことはできない
  - 東京 Region に所属しながら大阪の Rack にマウントすることはできない

Template & patch

- 膨大な記述の内訳は「ほぼ」同じ記述の繰り返しがほとんど
  - ファシリティとして同じような機器が大量に存在するため
- でも微妙に差異がある
  - だれと「関係」を持っているかが異なる、とか
  - 一部のみ特別な使い方をしている、とか
- Template を適用し、差分を patch で上書きしていく
  - Template に patch レイヤーを積み重ねるイメージ

DSL概念

```
# DSL としては以下の様な記述

nswitch "SW01" do
  template "xxxx"
  patch do
    ......
  end
end
```

```
# 解釈器
# ネットワークSWクラス定義
class Nswitch
  attr_accessor :port, .....
  def template(template_name)
    # templateを読み込んでクラス変数を定義していく
  end
  class Patch
    def initialize
      # 特定のクラス変数を書き換えていく
    end
  end
  class execute
  end
end

# ネットワークSW定義用メソッド
def nswtich(name, &block)
  sw = Nswitch.new(title)
  sw.instance_eval(&block)
  return sw
end
```

まとめ

- DC構成要素の記述対象として、要素間の関係の記述の必要性
- 関係を記述する上で、包含関係や単一関係の記述の必要性
- 管理者の記述負担を軽減する方法
  - テンプレーティングやパッチング
- DCファシリティ記述言語に向けて