Round-trip Software Engineering in DevOps:
Making the Infrastructure a Code Committer

Miguel Jiménez, Hausi A. Müller
{miguel, hausi}@uvic.ca

Norha M. Villegas, Gabriel Tamura
{nvillega, gtamura}@icesi.edu.co

March 5-6, 2018
Toulouse, France
Software Deployment

- Infrastructure to realize Deployment & Configuration (D&C)
- Industrial IoT and large CPS are a reality

### Network
- Networks, subnets, ports
- Security groups and access rules
- SDN/NFV
  - Networking devices
  - Computing devices
  - IoT devices
  - Virtual devices

### Hardware
- Data sources
- Data storage
- Data mining
- Data visualization
- Backend services

### Software
- Specification occurs at design time
- Managing resources occurs at runtime
- Stakeholders expect documentation in different levels of detail and abstraction
- How do tools support linking design and runtime deployment concepts?
Deployment Specification Challenges

**CH1**
Notations for specifying and visualising deployments from different perspectives and levels of abstraction

**CH2**
Deployment notations to support cross-cutting concerns

**CH3**
Notation and tool support for linking design and runtime deployment concepts

**CH4**
Tool support for the evolution of deployment specifications and configuration management at runtime
Bidirectional Traceability

Systematic approaches to maintain the correspondence between design and code are rarely used in practice*

Sources of Deployment evolution

- Web admin, e.g., OpenStack Horizon
- CLI, e.g., OpenStack CLI
- Configuration management, e.g., OpenStack HEAT
- Automatic runtime changes, e.g., Scaling policies

Bidirectional Traceability

Systematic approaches to maintain the correspondence between design and code are rarely used in practice*

**SCENARIO 1: Correspondence Mismatch**
1. Developer specifies deployment using OpenStack HOT
2. Developer deploys the system
3. Ops engineer increases VM's properties
4. Developer adds memory-intensive component
5. Developer cannot re-use deploy. spec as it is because of correspondence mismatch
6. Dev/Ops engineers manually re-deploy the system
7. Agility is broken

**SCENARIO 2: Informal Collaboration**
1. Developer specifies deployment using the most powerful VM (MPVM)
2. MPVM is not enough. Developer replicates the service
3. Infrastructure provider adds new machines, more powerful than MPVM
4. Developer never finds out and keeps using replicated MPVM
5. Waste of resources. Costs are higher

Continuous Integration

Where are all these changes logged?

How can they be traced back to their source?

How and when are stakeholders notified about these changes?

- **Infrastructure-as-Code**: Deployment specifications are *eventually* translated into code.
- Continuous integration is the solution! Isn’t it?

- Deployment Specs
  - e.g., OpenStack HOT

- Dev

- CI

- Version control repository

- Management Tools
  - Web admin
    - e.g., OpenStack Horizon
  - CLI
    - e.g., OpenStack CLI
  - Configuration management
    - e.g., OpenStack HEAT
  - Automatic runtime changes
    - e.g., Scaling policies

- Continuous Integration
**CI + Round-trip Engineering**

What if the infrastructure becomes a committer?

- Specifications can be managed through version control
- Each specification is represented by a model instance at runtime
- Specifications and model instances are kept in sync

Deployment Specifications

- **Infrastructure.yaml**
- **Network.yaml**
- **Software.yaml**

Version control repository

- Pull/Push

MART instances

- Ops Engineers
- Autonomic capabilities

Dev Engineers
CI + Round-trip Engineering (cont’d)

What if the infrastructure becomes a committer?

- Specifications can be managed through version control
- Each specification is represented by a model instance at runtime

Specifications are always up to date!

- Infrastructure.yaml
- Network.yaml
- Software.yaml

Version control repository

Pull/Push

Instance-Specification Translator

MART
- Notation
- Operations

MART Infrastructure

RUNTIME SUPPORT *

Web admin
e.g., OpenStack Horizon

CLI
e.g., OpenStack CLI

Configuration management
e.g., OpenStack HEAT

Automatic runtime changes
e.g., Scaling policies

Pull/Push

Contribution Model

1. The infrastructure as a **committer**
   
   **Pros**
   - No delay to reflect changes (instantaneous round-trip engineering)
   - Less merge conflicts
   
   **Cons**
   - Risk: unsupervised changes can break the system

2. The infrastructure as a **contributor** (fork + pull request)
   
   **Pros**
   - No risk
   
   **Cons**
   - Delay to reflect changes
   - Extra time spent reviewing changes
   - Merge conflicts are expected

**Pragmatic approach**: certain type of changes are directly committed, while others are requested
Conflict Resolution

1. Reliable Strategy (play safe)
   - One actor has priority over the other
   - Any change performed at **runtime** is discarded
   - Any change performed at **design** time is discarded

2. Best Effort Strategy
   - If the upstream changes aren’t related to local changes, try to merge

**Pragmatic approach**: the strategy to follow depends on the type of change to merge
CI Principles

Traditional CI approach (functional code)

✔ Maintain a code repository
✘ Automate the build
✘ Make the build self-testing
✔ Everyone commits to the baseline every day
✘ Every commit (to the baseline) should be built
✘ Keep the build fast
✘ Test in a clone of the production environment
✔ Make it easy to get the latest deliverables
✘ Everyone can see the results of the latest build
✘ Automate deployment

What are the corresponding items for deployment code?

- MART
- Quality assurance
- CHALLENGE
- Deploy MART & Update system
Scenario 1 Revisited

1. Developer specifies deployment

   $ deploy infrastructure-v0.1.0.yaml

2. Developer deploys the system

3. Ops engineer increases VM’s properties

   $ git pull & vim ...
   $ re-deploy infrastructure-v0.2.0.yaml

4. Developer modifies the spec. and re-deploys the system

Seamless collaboration of Dev & Ops roles!

- MART is instantiated
- MART is updated
- MART is translated into spec
- Specification is updated
- MART is updated from spec
Deployment Evolution (Future Work)

• Based on a current deployment spec. and the same spec. with some changes, find the execution workflow to realise those changes

• Deployment tools already offer some primitive way to update deployments

 Deployment Specifications

spec-v1.0.yaml

spec-v1.1.yaml

Automated Continuous Deployment

v1.0 = v1.1

Deployment Workflow
Deployment Evolution (cont’d)

1. Quality assurance?

2. Tool support

3. Self-adaptive systems

Update

Visual Studio Code
Eclipse
Web Platform

Pull/Push

Version control repository
CI Server

MART Support

APIs e.g., Neutron
MAPE-K

Notify

Pull/Push
Conclusions

1. Problem:
   Broken semantic correspondence

2. Solution:
   Two-way Continuous Integration

3. Future work:
   Quality assurance & Continuous deployment