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

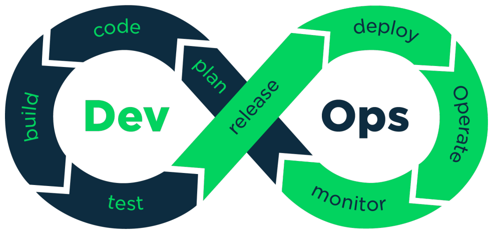
Rigi Research | University of Victoria

DRISO Desarrollo e Ingeniería de Software | Universidad ICESI

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

Hardware

- Networking devices
- Computing devices
- IoT devices
- Virtual devices

Software

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

- **Specification** occurs at design time — Dev
- Managing resources occurs at runtime — Ops
- Stakeholders expect documentation in **different levels of detail and abstraction**
- How do tools support linking design and runtime deployment concepts?

DESIGN & DEVELOPMENT → PROVISION → INSTALL → ACTIVATE → UPDATE → ADAPT

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

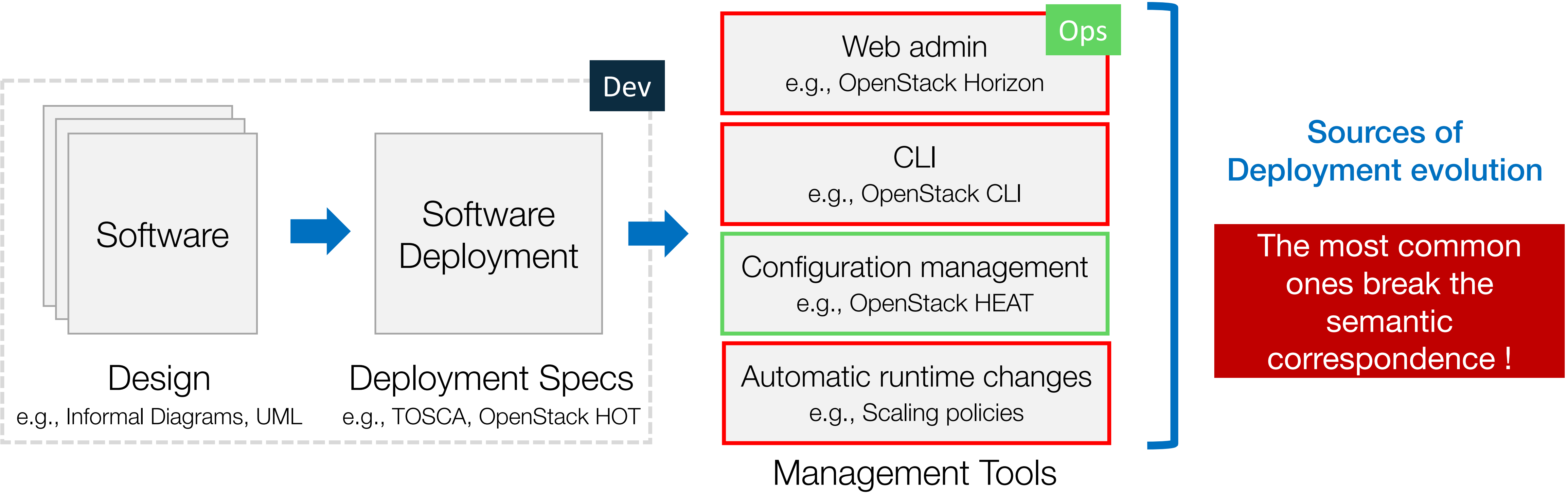
Tool support for the evolution of deployment specifications and configuration management at runtime

Miguel A. Jiménez, Norha M. Villegas, Gabriel Tamura, Hausi A. Müller: Deployment Specification challenges in the context of large scale systems. CASCON 2017: 220-226

CH4

Bidirectional Traceability

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



* Nugroho, Ariadi, and Michel RV Chaudron. "A survey of the practice of design--code correspondence amongst professional software engineers." Empirical Software Engineering and Measurement, 2007. ESEM 2007. First International Symposium on. IEEE, 2007.

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

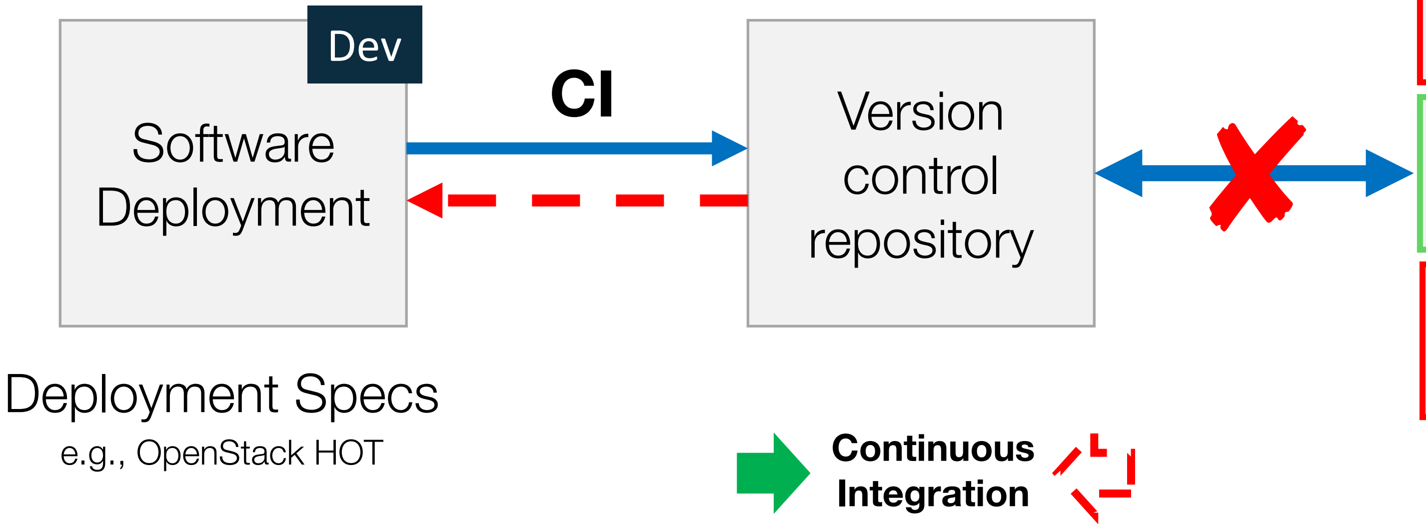
1. Developer specifies deployment using the most powerful VM (MPVM)
2. MPVM is not enough. Developer replicates the service
3. Infrastructure provider adds new machines, more powerful than MPVM
4. Developer never finds out and keeps using replicated MPVM
5. Waste of resources. Costs are higher

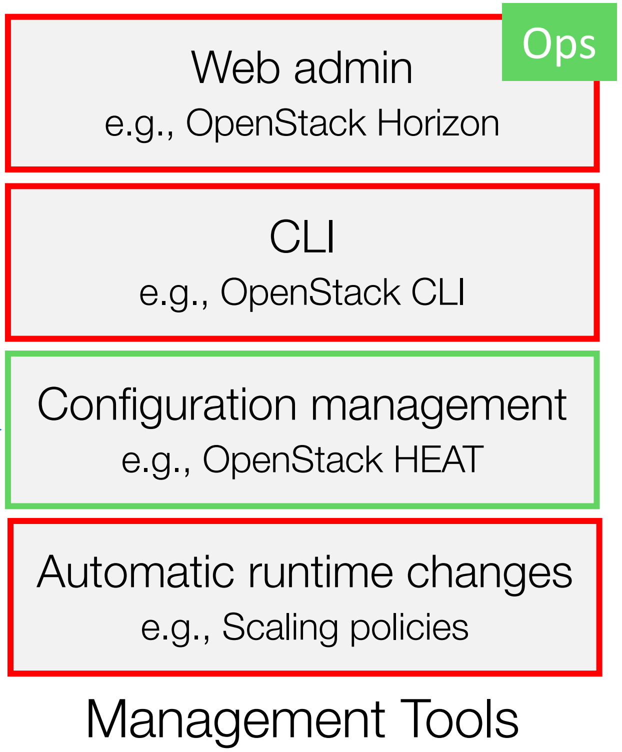
* Nugroho, Ariadi, and Michel RV Chaudron. "A survey of the practice of design--code correspondence amongst professional software engineers." Empirical Software Engineering and Measurement, 2007. ESEM 2007. First International Symposium on. IEEE, 2007.

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 ?**

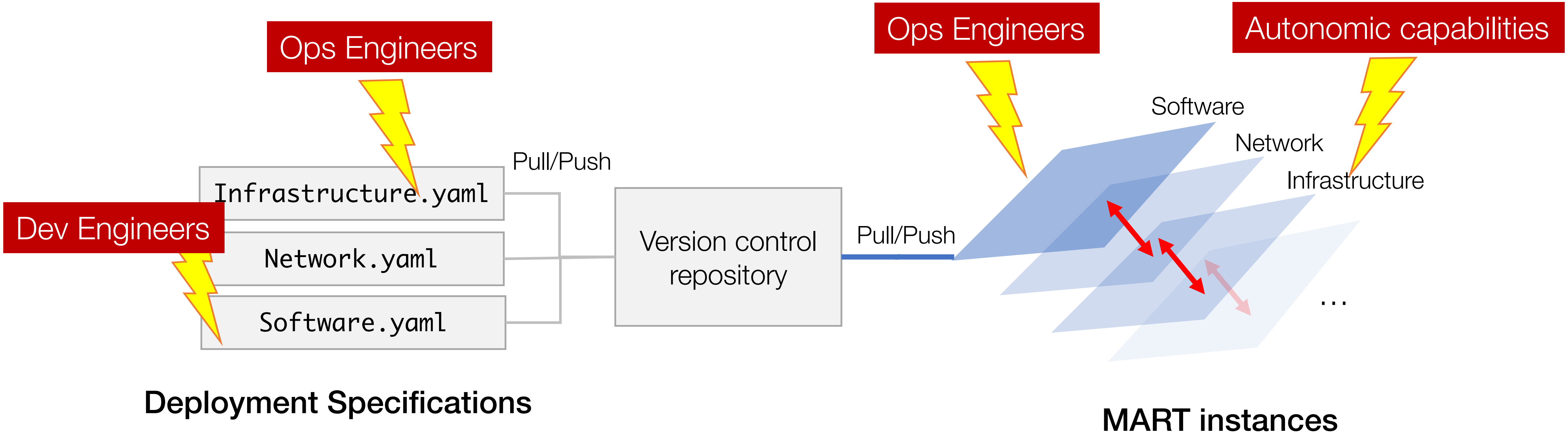


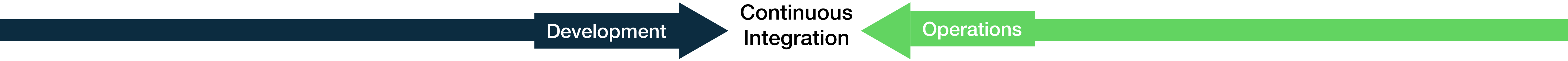


CI + Round-trip Engineering

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



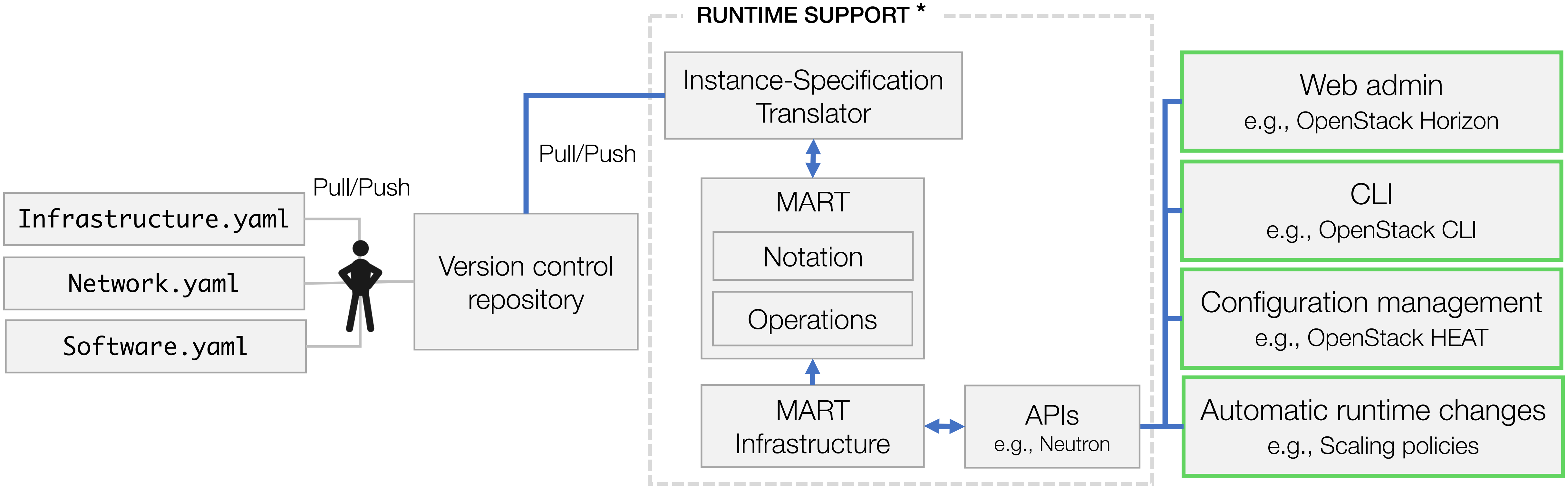


CI + Round-trip Engineering (cont'd)

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 !



* Castañeda, Lorena. "Runtime Modelling for Smart User-centric Cyber-Physical-Human Applications". PhD thesis, 2017

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)

What are the corresponding items for deployment 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**

MART

Quality assurance

CHALLENGE

Deploy MART &
Update system

Scenario 1 Revisited

Seamless collaboration of Dev & Ops roles !

infrastructure-v0.1.0.yaml

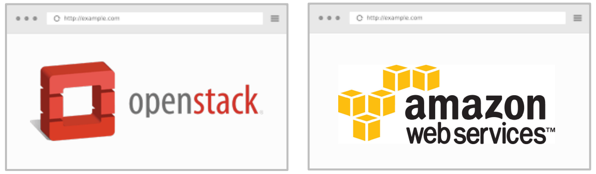
1. Developer specifies deployment

On push →

- MART is instantiated

```
$ deploy infrastructure-v0.1.0.yaml
```

2. Developer deploys the system



3. Ops engineer increases VM's properties

On action →

- MART is updated
- MART is translated into spec
- Specification is updated

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

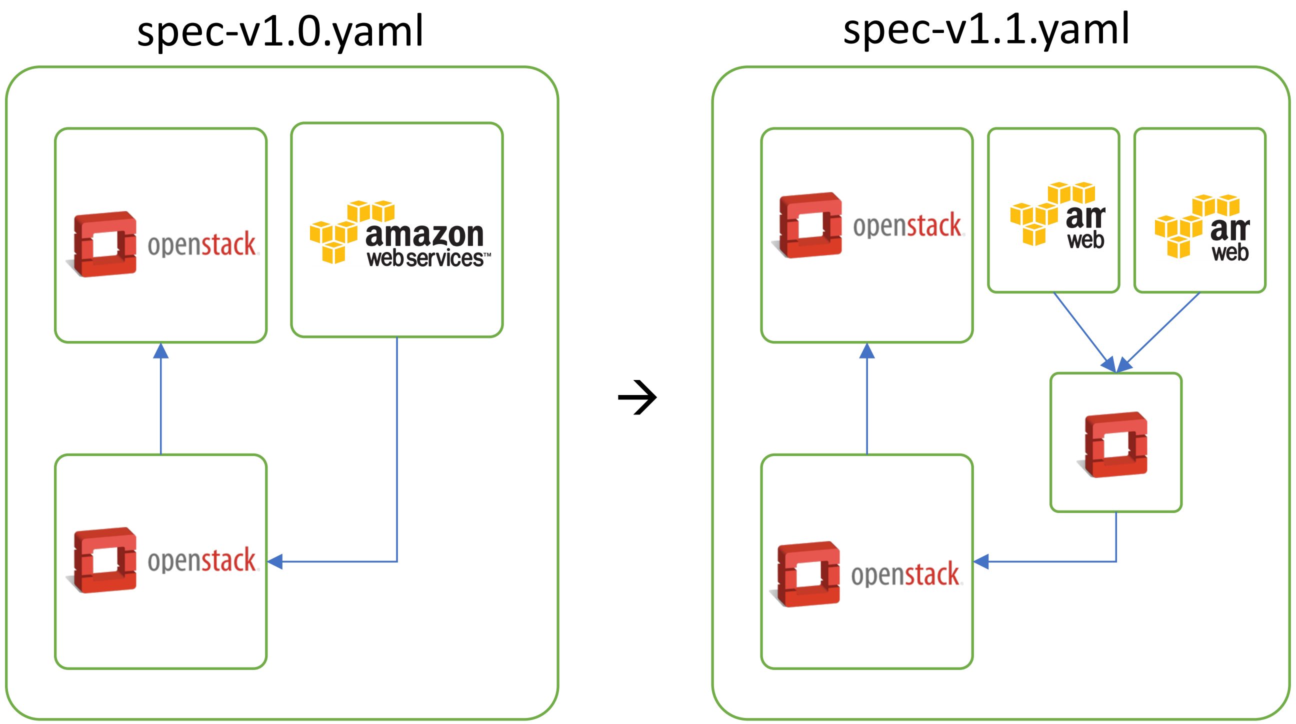
4. Developer modifies the spec. and re-deploys the system

On push →

- MART is updated from spec

Deployment Evolution (Future Work)

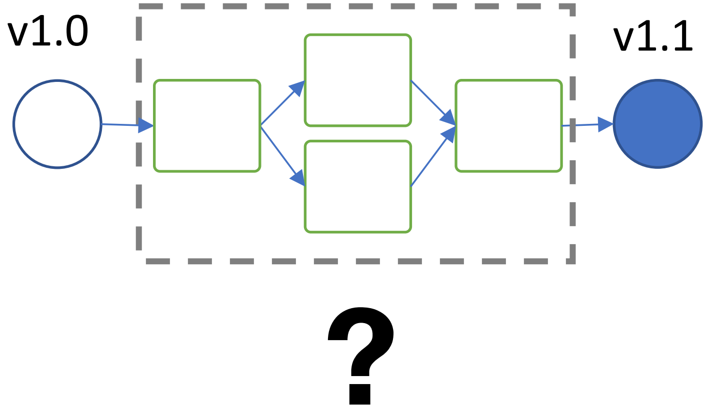
- Based on a current deployment spec. and the same spec. with some changes, find the execution workflow to realise those changes
- Deployment tools already offer some primitive way to update deployments



Deployment Specifications

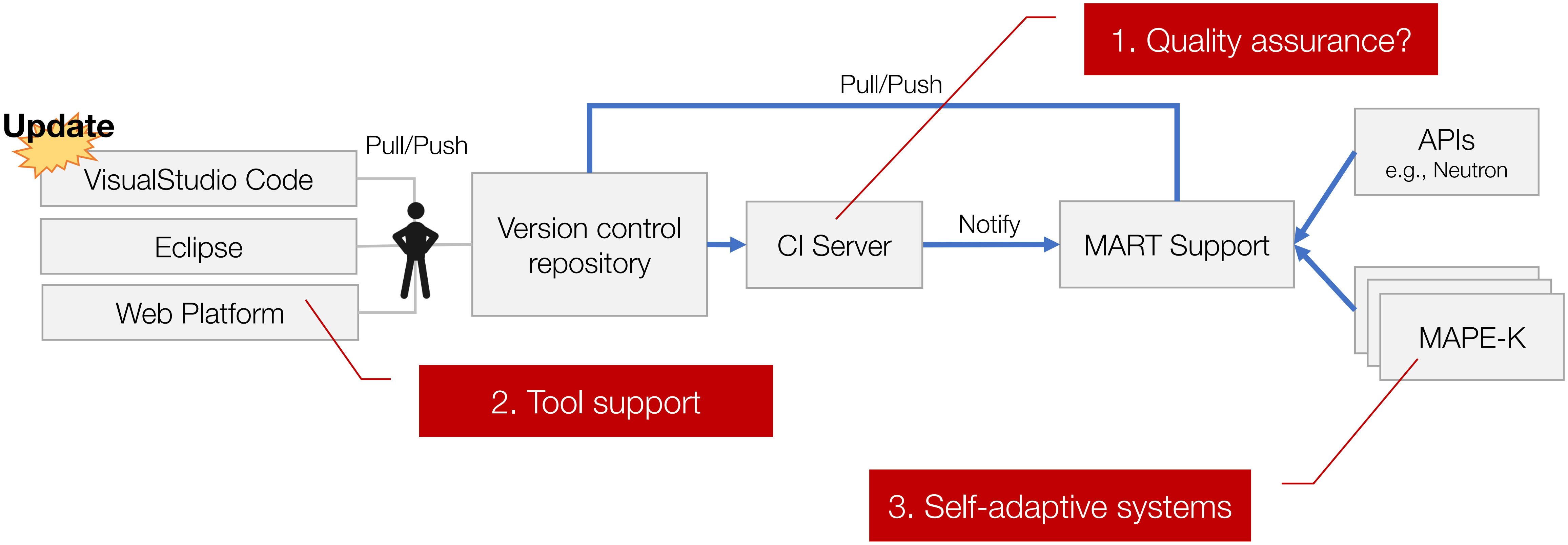
=

Automated Continuous Deployment

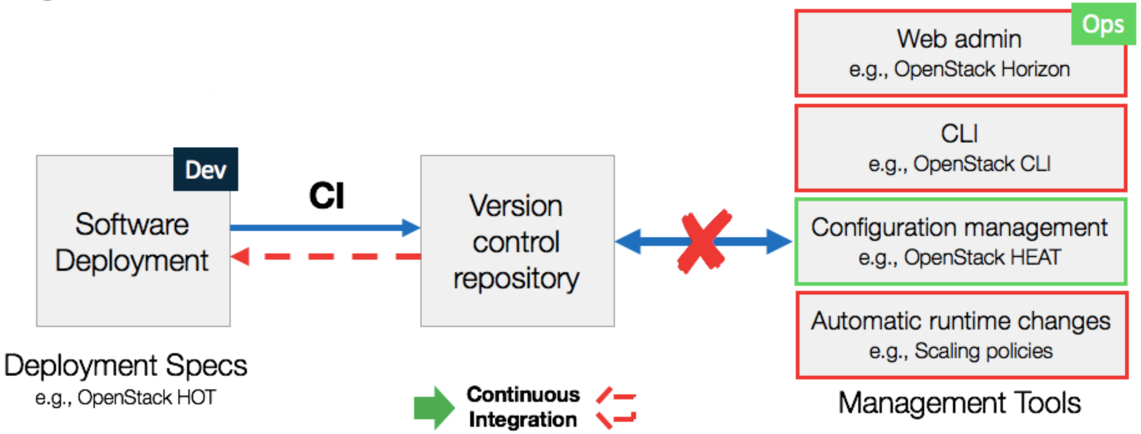


Deployment Workflow

Deployment Evolution (cont'd)

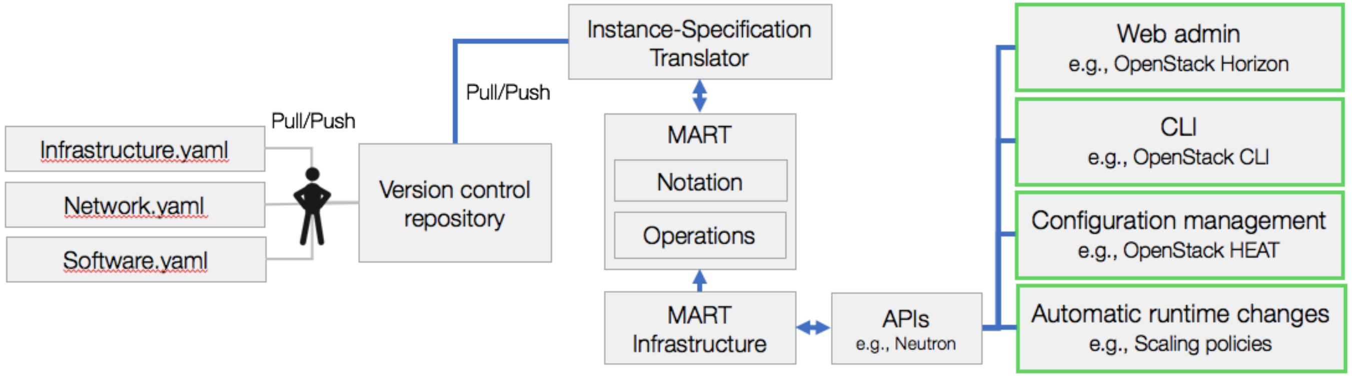


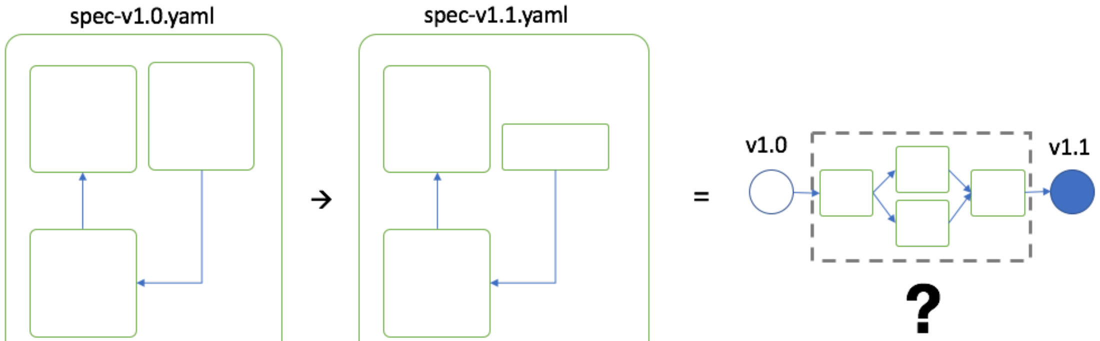
Conclusions



1. Problem:
Broken semantic correspondence

2. Solution:
Two-way Continuous Integration





3. Future work:
Quality assurance & Continuous deployment