Antonio García-Domínguez
15th Transformation Tool Contest
STAF 2023, Leicester, UK

Asymmetric and Directed Bidirectional Transformation for Container Orchestrations

Motivation

DevOps and MDE

DevOps gaining importance

- Leite et al.: "continuous delivery of new software versions, while guaranteeing correctness and reliability"
- Stack Overflow 2022: 10% of respondents said they were DevOps specialists

MDE useful for DevOps

- Many tools operate from declarative descriptions in loose formats (e.g. YAML)
- MDE can help produce those - Piedade et al. reviewed notations for Docker Compose: they usually don't cover 100%

Impact of incomplete notations

Models are abstractions!

- Very often, the model will not cover 100% of what will go into the YAML
- For instance, Piedade found that DockStation did not cover networks
- Docker evolves quickly - we **will** miss things!

Model and YAML in sync?

- We get the first version of the YAML from the model
- DevOps specialists may tweak YAMLs in the field
- The model may also change over time
- We need to synchronise model and custom YAML

Classification as bidirectional tx

Asymmetric

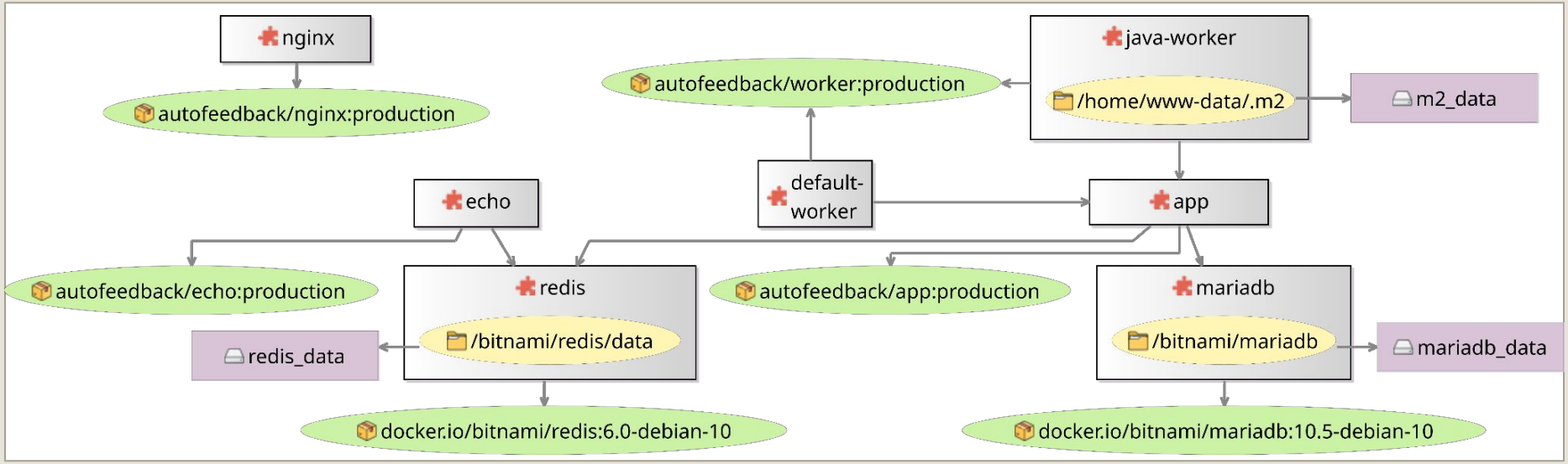
- YAML can contain all the information in the model, and more
- Different from another TTC case (Families to Persons), which was symmetric (neither side contains the other)

Directed

- Changes are only applied to one side at a time
- We have either:
  - edited the YAML and need to update the model, or
  - edited the model and need to update the YAML without losing extra info not in model

Transformation details

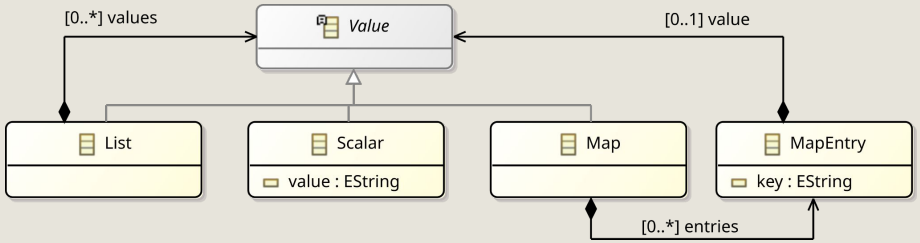
Source: Containers models



We have containers based on Docker images, with some storage volumes. Some containers require that others have started first.

Target: MiniYAML model

Metamodel



Simplification of YAML, with maps with string keys and scalar/list/map values. Scalars are always just strings.

Utilities

- Case includes Java programs to convert MiniYAML models to plain YAML files, and vice-versa
- Some advanced YAML features are not supported (e.g. anchors)

Sample generated MiniYAML model

- Version: always 2.4
- One service per container
- Containers can have "image", "replicas", "depends_on", and "volumes" settings
- Volumes just have names

Listing 1: Example YAML from Figure 3

```
version: '2.4'
services:
  mariadb:
    image: docker.io/bitnami/mariadb:10.5-debian-10
    volumes: ['mariadb_data:/bitnami/mariadb']
  redis:
    image: docker.io/bitnami/ redis:6.0-debian-10
    volumes: ['redis_data:/bitnami/redis/data']
  nginx: {image: 'autofeedback/nginx:production'}
  app:
    image: autofeedback/app:production
    depends_on: [mariadb, redis]
  java-worker:
    image: autofeedback/worker:production
    replicas: '2'
    volumes: ['m2_data:/home/www-data/.m2']
    depends_on: [app]
  default-worker:
    image: autofeedback/worker:production
    replicas: '2'
    depends_on: [app]
  echo:
    image: autofeedback/echo:production
    depends_on: [redis]
volumes: {mariadb_data: null, redis_data: null, m2_data:
```

Reference implementation in Epsilon

- First, created separate ETL scripts for each direction (Containers to MiniYAML, and MiniYAML to Containers)
- Next, created a merging transformation for the case where the YAML already exists, made up of three Epsilon scripts:
  - **Epsilon Comparison Language:** script matches elements from new and old YAML, using YAML paths (e.g. "services.mariadb.replicas")
  - **Epsilon Merging Language:** combines information from matched pairs (the new YAML takes priority in all the parts that are specified in the Containers model)
  - **Epsilon Transformation Language:** copies non-matching elements from the new and old YAML (e.g. the new YAML has a new container, or the old YAML specified a Docker Compose option which is not specified in the Containers model)

Comparison

Research questions

1. How **concisely** can we specify such a bx with current tools?

2. How well can such a bx **correctly** preserve customisations in the YAML which are outside of the bx, across various types of changes in the models?

3. How **scalably** would such a bx handle larger models, with more containers, more volumes, and more custom YAML elements outside of the transformation's control?

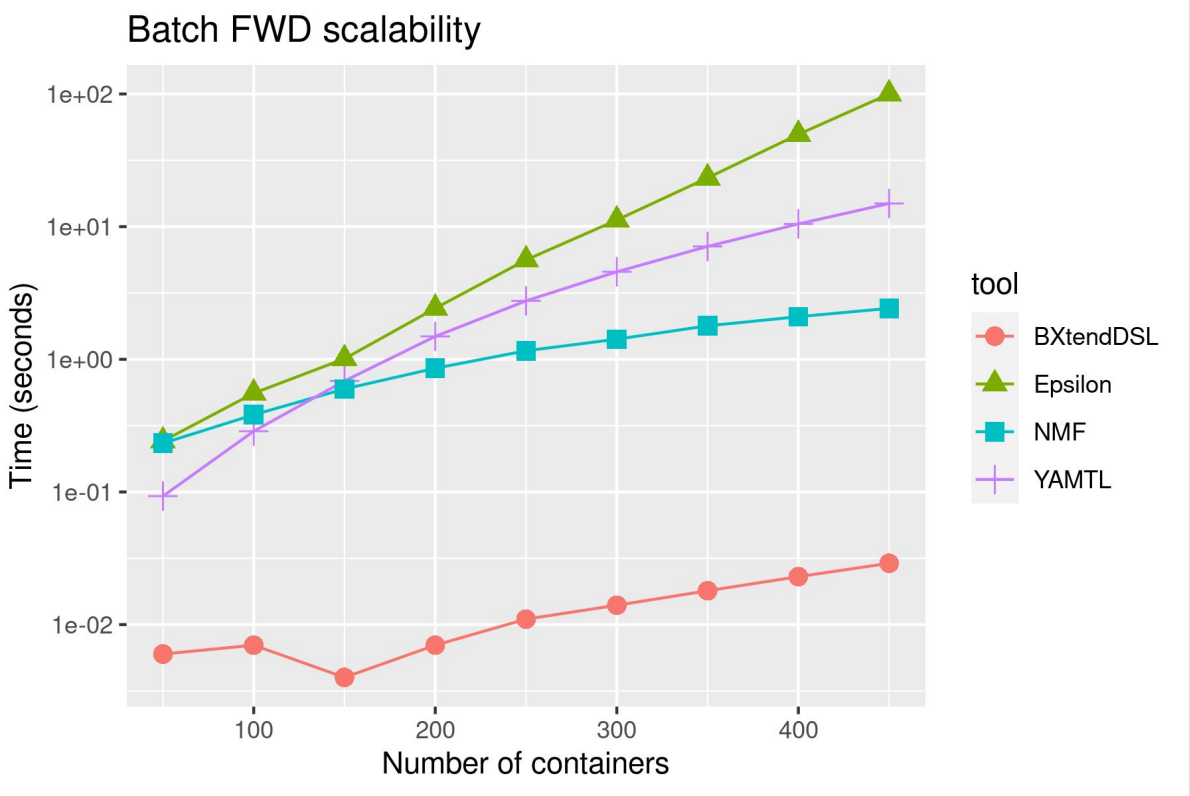
Correctness results

	Ignore YAML key order			Preserve YAML key order	
	Batch FWD	Batch BWD	Incr FWD	Batch FWD	Incr FWD
Reference (Epsilon)	✅	✅	❌ renCont	✅	❌ renCont, updReplicas
BXtendDSL	✅	✅	✅	✅	❌ updReplicas
NMF	✅	✅	✅	❌	❌
YAMTL	✅	✅	✅	✅	❌

Conciseness results (in words)

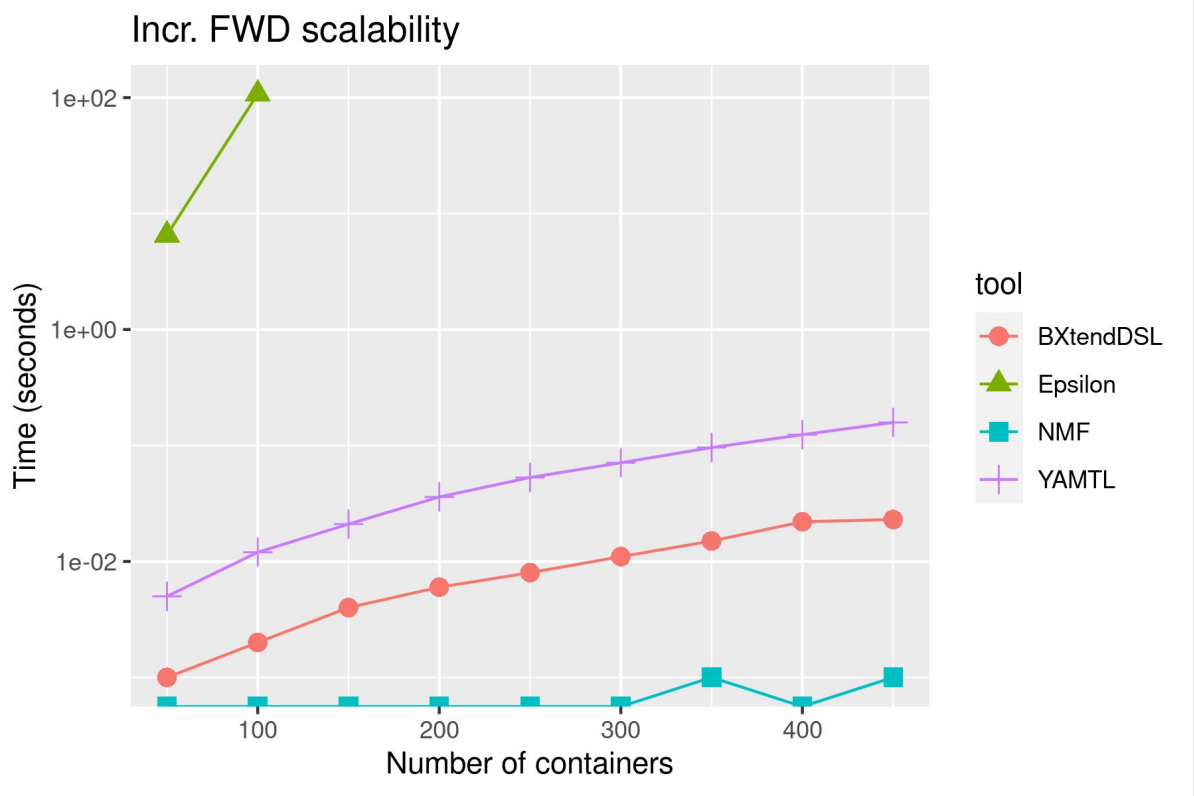
- Reference:
  - 677 (Java) + 419 (ETL) + 140 (EML) + 59 (EOL) + 70 (ECL) = 1365
- BXtendDSL:
  - 100 (BXtendDSL) + 788 (manually-written Xtend) = 888
- NMF:
  - 1383 (C#)
- YAMTL + EMF-Syncer:
  - 412 (Groovy)

Scalability: batch FWD



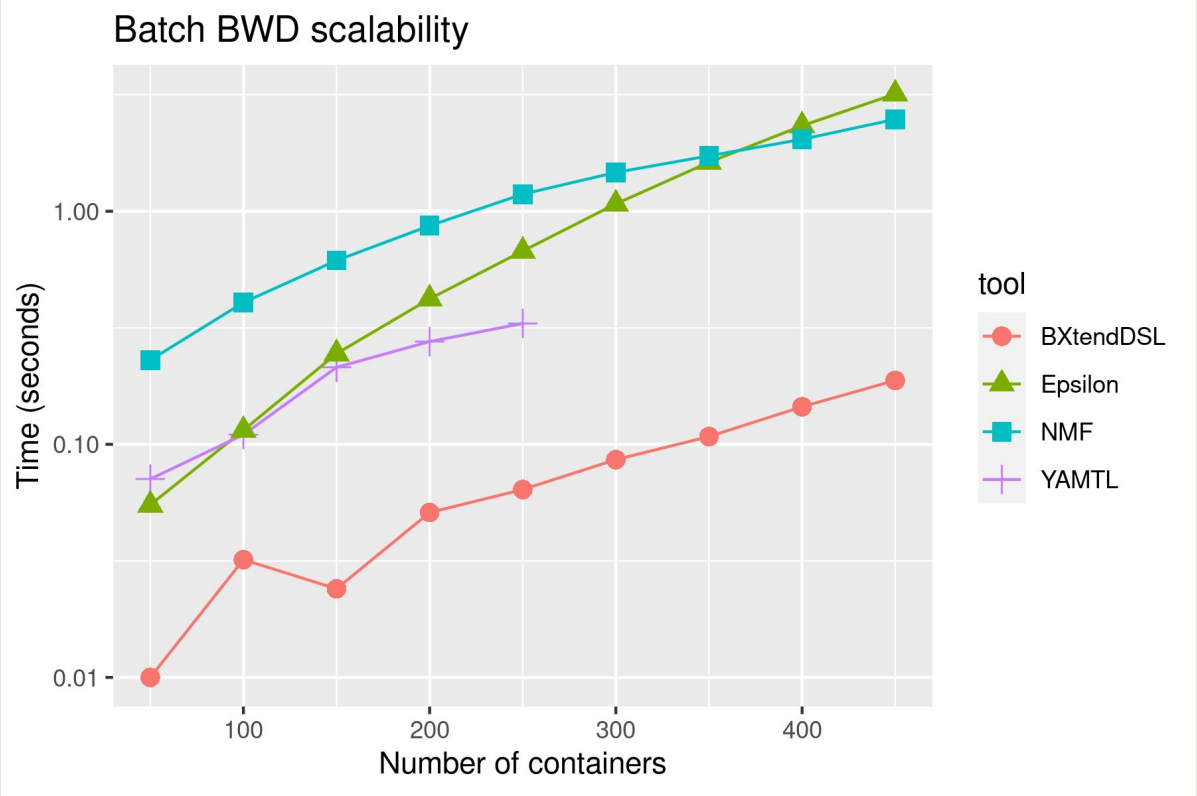
- BXtendDSL is fastest
- Then YAMTL and NMF
- Epsilon significantly slows down: need to do some profiling and fix bottlenecks!

Scalability: incremental FWD



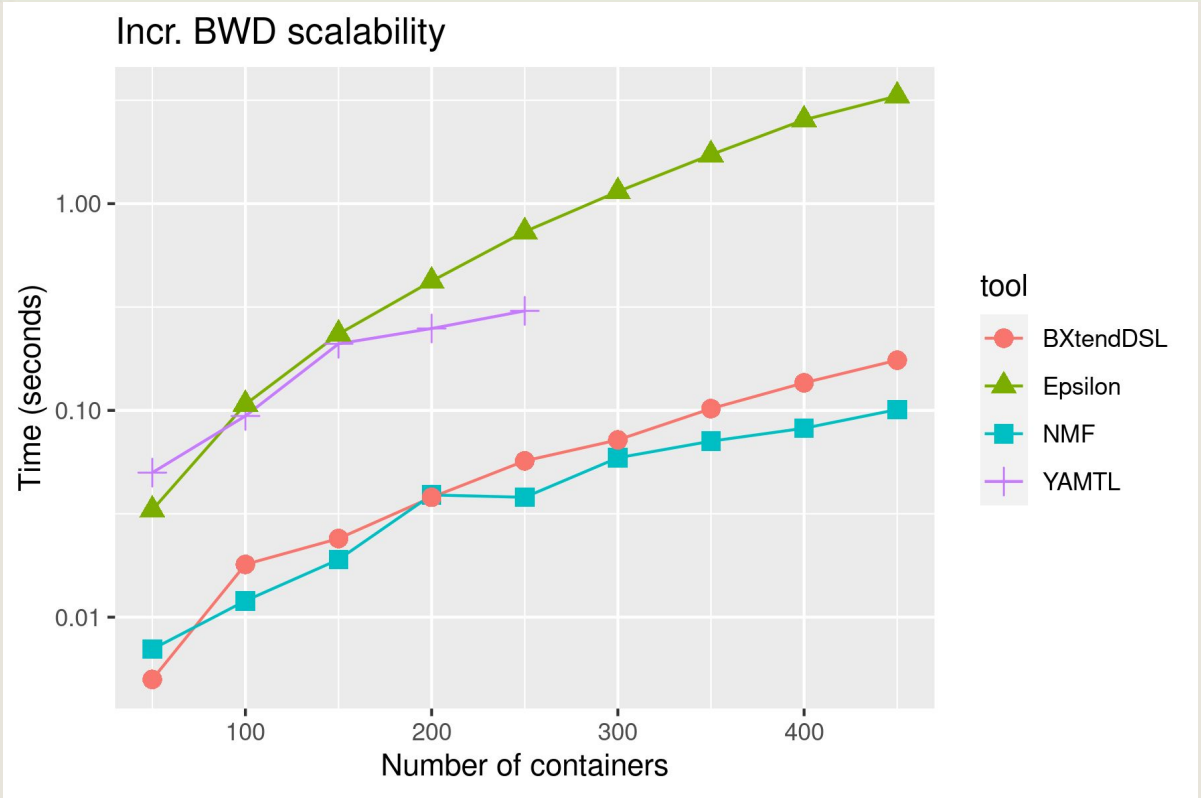
- NMF too fast to measure!
- BXtendDSL second, then YAMTL
- Epsilon really slowed down from 100 containers: most likely due to all-pairs matching in ECL (could be replaced with EPL)

Scalability: batch BWD



- NMF slower than Epsilon until 400 containers (C# process startup time?)
- BXtendDSL was fastest
- YAMTL crashed from 300 containers with a stack overflow exception
- Epsilon was slowest, but would still complete in a reasonable amount of time (~3.2s for 500 containers)

Scalability: incremental BWD



- NMF is now fastest when running incrementally (C# bootup time no longer matters)

Thank you!

a.garcia-dominguez@york.ac.uk