The TosKer open source environment

Antonio Brogi, Davide Neri, Luca Rinaldi, Jacopo Soldani
Department of Computer Science, University of Pisa, Italy

TOSCA TC meeting, 13th of December, 2018
Motivations

Cloud applications integrate **multiple interacting components**.

**Needs** (among others)

- **Select and configure an appropriate runtime environment** for each application component.

- **Orchestrate the management** of the application components on top of the selected environments.

Currently done manually and repeated whenever needed.
Motivations (cont.)

Consider the open-source\(^1\) application *Thinking* (for sharing thoughts).

![Diagram showing dependencies and connections between GUI, API, API RTE, and MongoDB]

- **GUI** dependsOn **API**
- **API RTE** hostedOn **API** connectsTo **MongoDB**

(API RTE)
- `node` v3
- `npm v6`
- `git`

(API RTE)
- `mvn` v3
- `git`

(MongoDB)
- `mongo`

manuel & driven by Thinking’s online docs

...what if Thinking’s reqs change?

manuel & must consider all of the above

...impact of changes in reqs?

---

\(^1\) [https://github.com/di-unipi-socc/thinking.](https://github.com/di-unipi-socc/thinking.)
Motivations (cont.)

Developers should only be required to describe

» the **components** forming an application,

» the **dependencies** occurring among them, and

» the **software support** needed by each component.

should then be **automated** by means of tools.
The TosKer environment

TosKer is an open-source solution for enhancing the current support for cloud applications, based on

- **TOSCA-based representation** for multi-component applications which can be used to specify only the application components and the software support they need
- Tools for **automatically completing/updating** TOSCA application specifications by discovering and including Docker-based runtime environments providing the software support needed by the application components
- Engine for **orchestrating** TOSCA applications on top of Docker-enabled hosts
The TosKer environment, concretely
The TosKer environment

- CSAR
- `.yaml`
- `.tosca`

- TosKERISER

- CSAR
- `.yaml`
- `.tosca`

- TosKER

- Docker Finder

- «incomplete» TOSCA specs

- Runnable TOSCA specs
We define three TOSCA node types¹:

» `tosker.nodes.Container` for Docker containers

¹ [https://di-unipi-socc.github.io/tosker-types/](https://di-unipi-socc.github.io/tosker-types/)
tosker.nodes.Container

- It can host software components.
- It can offer a connection endpoint.
- Configuration.
- List of supported software distributions.
- Base operating system distribution.
- It may require a volume to persist data.
Specifying applications only, with TOSCA

We define three TOSCA **node types**\(^1\):

» `tosker.nodes.Container` for Docker containers

» `tosker.nodes.Volume` for Docker volumes

» `tosker.nodes.Software` for the software components building an application

---

\(^{1}\) [https://di-unipi-socc.github.io/tosker-types/](https://di-unipi-socc.github.io/tosker-types/)
Containers can attach to it. It may require to be connected to another node. Its deployment may depend on the availability of another node. Capable of offering a connection endpoint.
Specifying applications only, with TOSCA

We define three TOSCA node types\(^1\):

- `tosker.nodes.Container` for Docker containers
- `tosker.nodes.Volume` for Docker volumes
- `tosker.nodes.Software` for the software components building an application

We exploit TOSCA normative relationship types to define interdependencies:

- `tosca.relationships.AttachesTo` for attaching a container to a volume
- `tosca.relationships.ConnectsTo` for connecting containers and/or components
- `tosca.relationships.HostedOn` for hosting components on containers/components
- `tosca.relationships.DependsOn` for deployment dependencies

\(^1\) [https://di-unipi-socc.github.io/tosker-types/](https://di-unipi-socc.github.io/tosker-types/)
Running example: Specifying *Thinking*'s components only
Specifying the software support needed by components

TOSCA natively permits constraining the nodes that can satisfy pending requirements.

The `node_filter` clause of a requirement permits indicating:

» the **node types** that can be used to satisfy such requirement, and

» constraints on (the **values of**) their **properties**.
Running example: Using `node_filter`

```
node_filter:
  type: tosker.nodes.Container
  properties:
    - supported_sw:
      - node: 6.x
      - npm: 3.x
      - git: x
    - ports:
      - 3000: 8080
    - os_distribution: ubuntu
```
The TosKer environment

- CSAR
- `yaml`
- `.tosca`

**TosKeriser**

- «incomplete» TOSCA specs

**Docker Finder**

- runnable TOSCA specs

**TosKer**
Using TosKeriser

$ toskerise FILE [COMPONENTS] [OPTIONS]

FILE
TOSCA YAML file or a CSAR to be completed

COMPONENTS
(optional) list of component to be considered (by default, all)

OPTIONS
- -interactive active interactive mode
- -force force the update of all containers
--constraints value constraints on image search
  (e.g. --constraints 'size<=100MB')
--policy=top_rated|size|most_used ordering of discovered images
Running example: Exploiting TosKeriser to complete *Thinking*

```
$ toskerise thinking.csar --policy size
```
Running example: Changing the components’ runtime

$ toskerise thinking.completed.csar -f --policy most_used
The TosKer environment

- CSAR
- .yaml
- .tosca
- TosKeriser
- TosKer

- «incomplete»
- TOSCA specs
- Docker Finder

- runnable
- TOSCA specs
DockerFinder is a tool for searching Docker images based on multiple attributes
- supported software distributions
- image size,
- stars and pulls,
- etc.

Three main steps:
1. **Download** and **analyse** Docker images.
2. **Generate** and **store** the descriptions in a local database.
3. **Users** submit multi-attribute queries.
The TosKer environment

CSAR

.yaml
.tosca

«incomplete»
TOSCA specs

TosKERISER

runnable
TOSCA specs

DOCKER
FINDER

CSAR

.yaml
.tosca

TosKER
$ tosker FILE [OPTIONS] COMMAND [ARGS]

FILE
  TOSCA YAML file or a CSAR to be deployed

OPTIONS
  -q|--quiet  hide command outputs
  --debug     print additional debugging log

COMMANDS
  exec        execute a management operation or a plan
  log         print the execution log of an operation
  ls          list all deployed applications
  prune       remove all files, containers, volumes, ...
Running example: A deployment plan for *Thinking*

```plaintext
#create DBVolume
dbvolume:Standard.create

#create and start MongoDB
mongodb:Standard.create
mongodb:Standard.start

#create and start GUICont
 gui_container:Standard.create
 gui_container:Standard.start

#create and start APICont
 api_container:Standard.create
 api_container:Standard.start

#create, configure and start ThinkingAPI
 api:Standard.create
 api:Standard.configure
 api:Standard.start

#create, configure and start ThinkingGUI
 gui:Standard.create
 gui:Standard.configure
 gui:Standard.start
```
Running example: Launching an instance of *Thinking*

```
$ toxker exec thinking.completed.csar --plan=thinking.up.plan
```

```
$ toxker exec thinking.csar --plan thinking.up.plan
(update memory: ok)
✓ Check deployment plan... Done
✓ Create network... Done
✓ Execute op "Standard.create" on "dbvolume"... Done
✓ Execute op "Standard.create" on "mongodb"... Done
✓ Execute op "Standard.start" on "mongodb"... Done
✓ Execute op "Standard.create" on "node"... Done
✓ Execute op "Standard.start" on "node"... Done
✓ Execute op "Standard.create" on "maven"... Done
✓ Execute op "Standard.start" on "maven"... Done
✓ Execute op "Standard.create" on "api"... Done
✓ Execute op "Standard.configure" on "api"... Done
✓ Execute op "Standard.start" on "api"... Done
✓ Execute op "Standard.create" on "gui"... Done
✓ Execute op "Standard.configure" on "gui"... Done
✓ Execute op "Standard.start" on "gui"... Done
```
Running example: Checking the status of *Thinking*

```
$ tosker ls
```

<table>
<thead>
<tr>
<th>Application</th>
<th>Component</th>
<th>Type</th>
<th>State</th>
<th>Full name</th>
</tr>
</thead>
<tbody>
<tr>
<td>thinking</td>
<td>dbvolume</td>
<td>Volume</td>
<td>created</td>
<td>thinking.dbvolume</td>
</tr>
<tr>
<td>thinking</td>
<td>mongodb</td>
<td>Container</td>
<td>running</td>
<td>thinking.mongodb</td>
</tr>
<tr>
<td>thinking</td>
<td>node</td>
<td>Container</td>
<td>running</td>
<td>thinking.node</td>
</tr>
<tr>
<td>thinking</td>
<td>maven</td>
<td>Container</td>
<td>running</td>
<td>thinking.maven</td>
</tr>
<tr>
<td>thinking</td>
<td>api</td>
<td>Software</td>
<td>running</td>
<td>thinking.api</td>
</tr>
<tr>
<td>thinking</td>
<td>gui</td>
<td>Software</td>
<td>running</td>
<td>thinking.gui</td>
</tr>
</tbody>
</table>
The TosKer environment

https://github.com/di-unipi-socc/TosKeriser

https://github.com/di-unipi-socc/TosKer

https://github.com/di-unipi-socc/DockerFinder