Configure Once, run everyhere!

Configuration with Apache Tamaya

trivadis makes IT easier.

Apache incubator

About Me

Anatole Tresch

Principal Consultant, Trivadis AG (Switzerland)

Star Spec Lead
Technical Architect, Lead Engineer
PPMC Member Apache Tamaya

@atsticks
anatole@apache.org
anatole.tresch@trivadis.com

trivadis
makes IT easier.

Agenda

- Motivation
- Requirements
- The API
- Configuration Backends
- Demo
- Extensions

Motivation

What is Configuration ?

Simple Key/value pairs?

Typed values?

trivadis
makes IT easier.

When is Configuration useful?

Use Cases?

How is it stored?

Remotely or locally?

Classpath, file or ...?

Which format?

All of the above (=multiple sources) ?

When to configure?

Develpment time ?

Build/deployment time?

Startup?

Dynamic, anytime?

Configuration Lifecycle ?

Static ?

Refreshing ?

Changes triggered ?

Do I need a runtime ?

Java SE?
Java EE?
OSGI?

trivadis
makes IT easier.

Requirements

Requirements

- Developer‘s Perspective
- Architectural/Design Requirements
- Operational Aspects
- Other Aspects

Developer‘s Requirements

- Easy to use.
- Developers want defaults.
- Developers don‘t care about the runtime (for configuration only).
- Developers are the ultimate source of truth
- Type Safety

Architectural/Design Requirements

Decouple code that consumes configuration from

- Backends Used
- Storage Format
- Distribution
- Lifecycle and versioning
- Security Aspects

Operational‘s Requirements

- Enable Transparency:
  - What configuration is available ?
  - What are the current values and which sources provided the value ?
  - Documentation
- Manageable:
  - Configuration changes without redeployment or restart.
  - Solution must integrate with existing environment

Other Aspects

- Support Access Constraints and Views
- No accidental logging of secrets
- Dynamic changes
- Configuration Validation

Accessing Configuration

The API

trivadis makes IT easier.

API Requirements

- Leverage existing functionality where useful
- Only one uniform API for access on all platforms!
- Defaults provided by developer during development (no interaction with operations or external dependencies)

Existing Mechanisms

- Environment Properties
- System Properties
- CLI arguments
- Properties, xml-Properties

Dependencies - API & Core

```
<dependency>
    <groupId>org.apache.tamaya</groupId>
    <artifactId>tamaya-api</artifactId>
    <version>0.2-SNAPSHOT</version>
</dependency>
<dependency>
    <groupId>org.apache.tamaya</groupId>
    <artifactId>tamaya-core</artifactId>
    <version>0.2-SNAPSHOT</version>
</dependency>
```

Programmatic API

```
Configuration config =
                ConfigurationProvider.getConfiguration();

// single property access
String name = config.getOrDefault("name", "John");
int ChildNum = config.get("childNum", int.class);

// Multi property access
Map<String,String> properties = config.getProperties();

// Templates (provided by extension)
MyConfig config = ConfigurationInjection.getConfigurationInjector()
                  .getConfig(MyConfig.class);
```

trivadis
makes IT easier.

Dependencies – Injection SE

```
<dependency>
    <groupId>org.apache.tamaya.ext</groupId>
    <artifactId>tamaya-injection-api</artifactId>
    <version>0.2-SNAPSHOT</version>
</dependency>
<dependency>
    <groupId>org.apache.tamaya.ext</groupId>
    <artifactId>tamaya-injection</artifactId>
    <version>0.2-SNAPSHOT</version>
</dependency>
```

```
@Config(value=„admin.server“, defaultValue=“127.0.0.1“)
private String server;

@Config(value=“admin.port“,
                 defaultValue=“8080“)
private int port;

@Config(value=“admin.connections“)
private int connections = 5;

@Config(„address“)
private Address address;
```

```
MyTenant t = new MyTenant();
ConfigurationInjection
    .getConfigurationInjector()
    .configure(t);
```

Apache incubator™

trivadis makes IT easier.

Configuration Backends

trivadis makes IT easier.

Configuration Backends

- Support existing mechanisms OOTB
- Provide a simple SPI for (multiple) property sources
- Define a mechanism to prioritize different property sources
- Allow different strategies to combine values
- Support Filtering
- Support Type Conversion

So what is a property source ?

PropertySource

```
public interface PropertySource {

   PropertyValue get(String key);
   Map<String,String> getProperties();
   boolean isScannable();
   String getName();
   int getOrdinal();
}
public final class PropertyValue{
   public String getKey();
   public String getValue();
   public String get(String key);
   public Map<String,String> getConfigEntries();
   ...
}
```

trivadis
makes IT easier.

Are there predefined property sources ?

Of course.

- System & Environment Properties
- (CLI Arguments)
- Files:

```
${configDir}/*.properties
```

- Classpath Resources:

```
/META-INF/javaconfiguration.properties
```

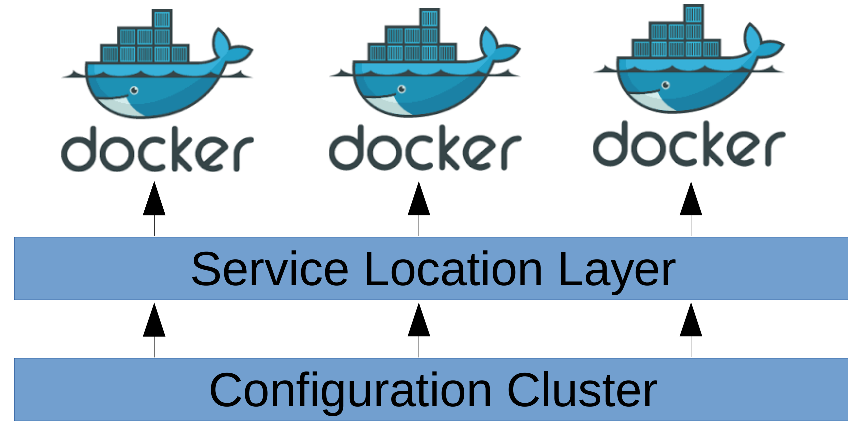
And how about remote configuration…?

Especially with Containers?

Remote configuration

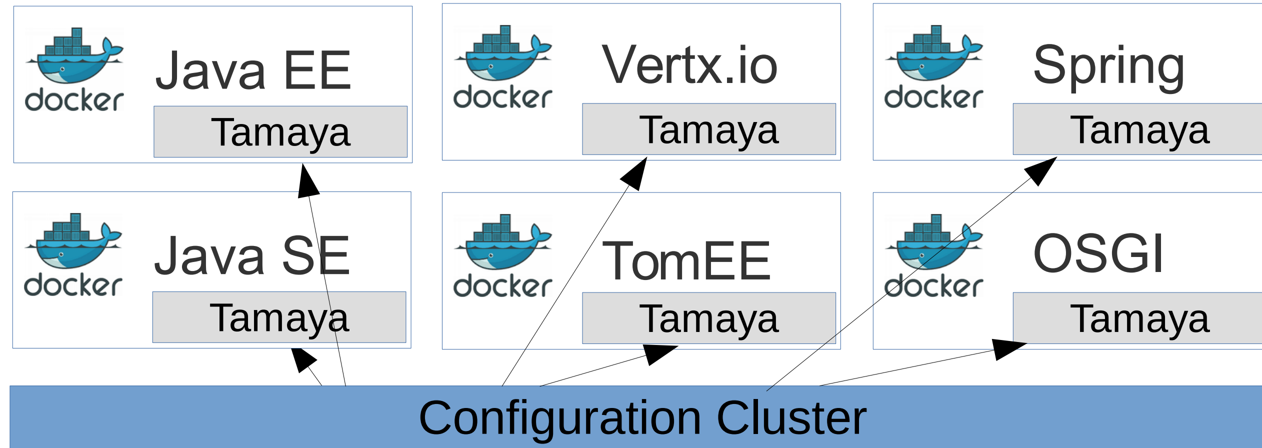
```
<dependency>
    <groupId>org.apache.tamaya.ext</groupId>
    <artifactId>tamaya-etcd</artifactId>
    <version>...</version>
</dependency>
```

- Configuration is read from remote source, e.g.
  - Etcd cluster
  - Consul cluster
  - Any Web URL
  - ...

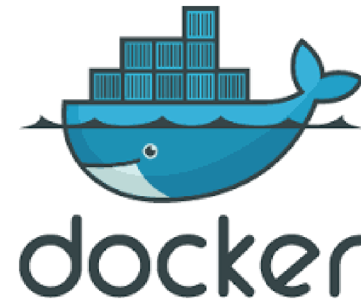


What kind of runtime I need ?

- In fact, it doesn‘t matter ! **Configure once, run everywhere !**



Excourse: Configuring Containers



- Configuration on deployment by **environment properties**:

```
docker run -e stage prod  -d -n MyApp user/image
```

or **Dockerfile/Docker Image**:

```
FROM java:8-jre
...
ENV stage prod
```

How to add custom configuration ?

Other files...

Or resources…

My self-written super fancy config database ?

Whatever I like ?

Use the SPI !

PropertySource

Property sources

- Mostly map to exact one file, resource or backend
- Have a unique name
- Must be thread safe
- Can be dynamic
- Provide an ordinal
- Can be scannable

PropertySource – Example

```
public class MyPropertySource extends BasePropertySource{
    private Map<String,String> props = new HashMap<>();
    public SimplePropertySource() throws IOException {
        URL url = getClass().getClassLoader().getResource(
                   “/META-INF/myFancyConfig.xml“);
        // read config properties into props
        ...
    }

    @Override
    public String getName() { return “/META-INF/myFancyConfig.xml“; };

    @Override
    public Map<String, String> getProperties() { return props; }
}
```

trivadis
makes IT easier.

PropertySource - Registration

- By default, register it using the `java.util.ServiceLoader`

```
→ /META-INF/services/org.apache.tamaya.spi.PropertySource

MyPropertySource
```

PropertySource Provider

Property source provider

- Allow *dynamic* registration of *multiple* property sources
  - E.g. all files found in a config directory
- Are evaluated once and then discarded
- Are also registered using the `ServiceLoader`.

PropertySourceProvider

```
public interface PropertySourceProvider{

    public Collection<PropertySource> getPropertySources();

}
```

trivadis
makes IT easier.

More SPI artifacts

Furthermore Tamaya uses

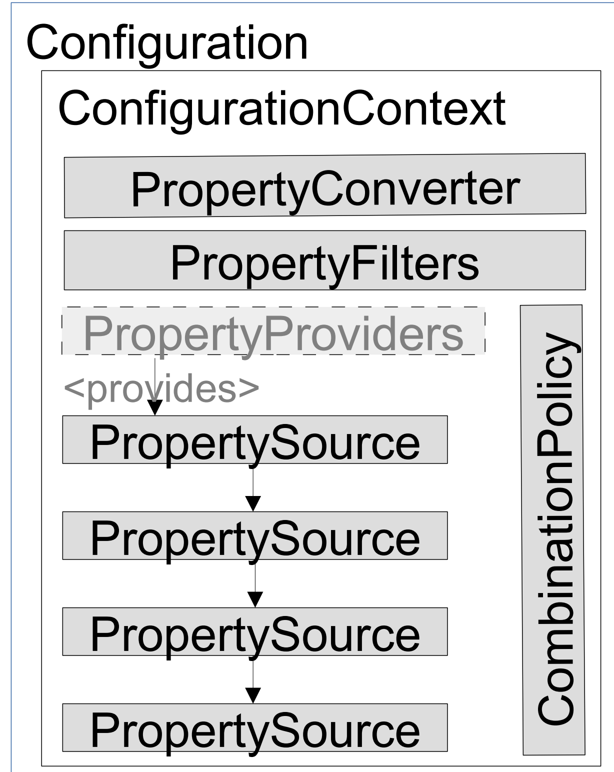
- Filters for filtering values evaluated (remove, map, change)
- Converters for converting String values to non-String types
- A `ValueCombinationPolicy`
  - determines how values evaluated are combined to a final value (defaults to overriding)

trivadis
makes IT easier.

And how these pieces all fit together ?

Apache Tamaya in 120 seconds...

1. **`Configuration`** = ordered list of **`PropertySources`**
2. Properties found are **combined** using a `CombinationPolicy`
3. Raw properties are **filtered** by `PropertyFilter`
4. For typed access `PropertyConverters` have to do work
5. **Extensions** add more features (discussed later)
6. **Component Lifecycle** is controlled by the `ServiceContextManager`



Apache incubator
trivadis makes IT easier.

So we have:

files, resources, sys- and env-properties

& an SPI to implement and register them ?

What else do we need ?

Easy Configuration:

„Meta“-Configuration !

Meta-Configuration

DRAFT !

- Configuration that configures configuration
- E.g. at `META-INF/tamaya-config.xml`
- Allows easy and quick setup of your configuration environment
- Allows dynamic enablement of property sources
- ...

```
<configuration>
    <context>
        <context-param name="stage">DEV</context-param>
    </context>
    <sources>
      <source type="env-properties" enabled="${stage=TEST || stage=PTA || stage=PROD}" ordinal="200"/>
      <source type="sys-properties" />
      <source type="file">
          <observe period="20000">true</observe>
          <location>./config.json</location>
      </source>
      <source type="resources" multiple="true">
          <multiple>true</multiple>
          <location>/META-INF/application-config.yml</location>
      </source>
      <source type="ch.mypack.MyClassSource">
          <locale>de</locale>
      </source>
      <source type="includes" enabled="${context.cstage==TEST}">
          <include>TEST.properties</include>
      </source>
    </sources>
</configuration>
```

DRAFT !

trivadis
makes IT easier.

Demo

trivadis makes IT easier.

Demo

- 1 Microservice
- Running on Java EE 7 (Wildfly)
- Multiple Configuration Sources:
  - Environment Properties
  - System Properties
  - Classpath
  - Files
  - Etcd Server

There is more!

Tamaya Extensions

trivadis makes IT easier.

```xml
<dependency>
    <groupId>org.apache.tamaya.ext</groupId>
    <artifactId>tamaya-resolver</artifactId>
    <version>...</version>
</dependency>
```

Property resolution...

java.home=/usr/lib/java

compiler=${ref:java.home}/bin/javac

```xml
<dependency>
    <groupId>org.apache.tamaya.ext</groupId>
    <artifactId>tamaya-resources</artifactId>
    <version>...</version>
</dependency>
```

Resource expressions…

```java
public class MyProvider extends AbstractPathPropertySourceProvider{
  public MyProvider(){
    super("classpath:/META-INF/config/**/*.properties");
  }
  @Override
  protected Collection<PropertySource> getPropertySources(URL url) {
      // TODO map resource to property sources
      return Collections.emptySet();
  }
}
```

And more: a topic on its own!

- *Tamaya-**spi-support**:* Some handy base classes to implement SPIs
- *Tamaya-**functions**:* Functional extension points (e.g. remapping, scoping)
- *Tamaya-**events**: Detect and publish ConfigChangeEvents*
- *Tamaya-**optional**: Minimal access layer with optional Tamaya support*
- *Tamaya-**filter**: Thread local filtering*
- *Tamaya-**inject-api**: Tamaya Configuration Injection Annotations*
- *Tamaya-**inject**:* Configuration Injection and Templates SE Implementation (lean, no CDI)
- *Format Extensions:* yaml, json, ini, … including formats-SPI
- Integrations with **CDI, Spring, OSGI*, Camel, etcd**
- *Tamaya-**mutable-config***: Writable ConfigChangeRequests*
- *Tamaya-**model****: Configuration Model and Auto Documentation
- *Tamaya-**collections****: Collection Support
- *Tamaya-**resolver**:* Expression resolution, placeholders, dynamic values
- *Tamaya-**resources**:* Ant styled resource resolution

…

Summary

trivadis makes IT easier.

Summarizing...

- A Complete thread- and type-safe Configuration API
- Compatible with all major runtimes
- Simple, but extendible design
- Extensible
- Small footprint
- Base for current Java EE 8 spec ?

trivadis
makes IT easier.

You like it ?

It is your turn !“

- *Use it*
- *Evangelize it*
- *Join the force!*

trivadis
makes IT easier.

Links

Project Page: http://tamaya.incubator.apache.org
Twitter: @tamayaconfig
Blog: http://javaeeconfig.blogspot.com
Presentation by Mike Keith on JavaOne 2013:
https://oracleus.activeevents.com/2013/connect/sessionDetail.ww?SESSION_ID=7755
Apache Deltaspike: http://deltaspike.apache.org
Java Config Builder: https://github.com/TNG/config-builder
Apache Commons Configuration: http://commons.apache.org/proper/commons-configuration/
Jfig: http://jfig.sourceforge.net/
Carbon Configuration: http://carbon.sourceforge.net/modules/core/docs/config/Usage.html
Comparison on Carbon and Others:
http://www.mail-archive.com/commons-dev@jakarta.apache.org/msg37597.html
Spring Framework: http://projects.spring.io/spring-framework/
Owner: http://owner.aeonbits.org/

Q&A

Thank you!

@atsticks

anatole@apache.org

trivadis
makes IT easier.