CREDIT SUISSE

JavaOne 2014

An Draft for Java Configuration

Configuration in SE and EE

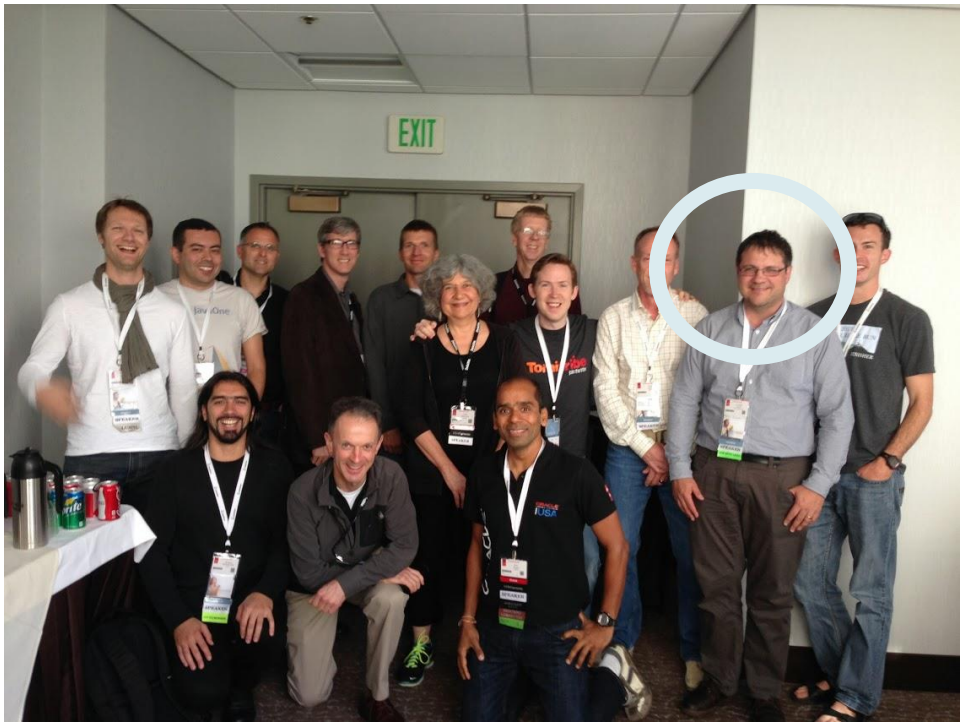
Anatole Tresch, Credit Suisse

September 2014

Bio

Anatole Tresch

- Consultant, Coach
- Credit Suisse
- Technical Coordinator & Architect
- Specification Lead JSR 354
- Driving Configuration



- Twitter/Google+: @atsticks
- atsticks@java.net
- anatole.tresch@credit-suisse.com
- JUG Switzerland (Europe)
- Zurich Hackergarten

Agenda

- Defining the Problem
- Use Cases
- Existing Approaches
  - Java EE
  - Java SE
  - Outlook
- Configuration API for the Java Platform
- Summary

Defining the Problem

Defining the Problem

What is configuration, anyway?

- Many different interested stakeholders
- Many things to configure
- Divergent views
  - Setup for a given server environment
  - Define the parameters of a runtime (staging, localization etc.)
  - Deployment descriptors
  - Technology-specific components (beans, wirings etc.)
  - Resource-specific settings (data sources, message queues etc.)
  - Scripting facility
- Different granularities, varying levels of applicability
- Different formats
- …

Use Cases

Use Cases (1)

- **Staging / Environment Dependent Configuration**
  - Available to one or to all apps, simple to define and manage
  - Deployment Profiles (production, test, staging etc.)
- **Single application, but multiple deployments**
  - Don’t want to rip open application for each deployment to add custom resources
  - Enable Product / Component Customization for COTS solutions
- **Manage your Configuration**
  - Don’t require manual non-standard deployment tool to override or mess up the build for configuration assembly
  - Configuration Servers (files, shares, Maven Repo, …)

Use Cases (2)

- **Scoped configuration**
  - Manage Visibility and Accessibility of Configuration – e.g. global/ear/app/tenant scope
  - Encrypt/protect entries
- **Flexible Configuration Access from any context**
  - Service Location Based Singleton access (Java API), also needed for legacy integration
  - Injection, e.g. with CDI, @Inject

Use Cases (3)

- **Dynamic SaaS Tenant configuration / Multi Tenancy**
  - Adding/Removing/Updating Tenants w/o restart
  - SaaS application uses an API to look up a given tenant configuration
- **Cloud**
  - Wire a deployed application to multiple decoupled cloud services and vice versa
  - Life Update changes, e.g. due to dynamic provisioning and elasticity
  - Support Automatic Provisioning

Use Cases (4)

- **Support Single Layered Configuration**
  - Standalone applications
  - Tests
  - Bootstrapping of Containers
- **Support Multi Layered Configuration**
  - Java EE
  - Multi-Tenancy, SaaS
  - OSGI
  - JigSaw
  - …

Use Cases (5)

Advanced Use Cases

- **Versioned configuration**
  - Dependency on a specific configuration version
- **Clustering support**
  - Node and cluster-based domain configurations
- **Security Constraints**
- **Configuration Templates**
- **Tool Support**
- **Remote Updates**

Existing Approaches

Existing Approaches

Overview (likely uncomplete)

- **Java EE**
- **Spring**
  - XML and annotations to configure Spring beans and overall behavior (`PropertyResolver`)
- **Apache Deltaspike**
  - Configure CDI beans, injecting configuration
- **Oracle Metadata Service (MDS)**
  - Extensive general purpose support for many different kinds of artifacts
- **Chef and Puppet**
  - Server-based Ruby scripting support for network management
- **Apache Commons Configuration**
- **JFig, Java Config Builder, Carbon**
- **java.util.Preferences/Properties, java.util.ResourceBundle**
- **In-House Solutions**

Existing Approaches (Java EE)

JSR 339 JAX RS Runtime Delegate

Runtime Delegate = abstract factory class that provides various methods for the creation of objects that implement JAX-RS APIs :

1. A resource with the name of `META-INF/services/javax.ws.rs.ext.RuntimeDelegate`
2. `${java.home}/lib/jaxrs.properties` exists and contains an entry `javax.ws.rs.ext.RuntimeDelegate`
3. System property `javax.ws.rs.ext.RuntimeDelegate`

Existing Approaches (Java EE)

JSR 352 Batch

```
META-INF/batch.xml
META-INF/batch-jobs/…
```

Support for substitutions within config files (though not explicit EL support), mostly for inter-operability within a deployment, but also supporting system properties.

Existing Approaches (Java EE)

JSR 338 JPA 2.1

`META-INF/persistence.xml`

JPA provides a metamodel `javax.persistence.metamodel.MetaModel`), but this is read-only.

A customized `javax.persistence.spi.PersistenceProvider`, `PersistenceProviderResolver` can be registered using `java.util.ServiceLoader`.

Existing Approaches (Java EE)

JSR 342 Java EE 7

```
META-INF/application.xml
META-INF/MANIFEST.MF (Main-Class attribute)
META-INF/application-client.xml
```

Several configuration options, but no EL support, no dynamics (except alt-dd concept).

Existing Approaches on EE

JSR 345: EJB 3.2 Core

`META-INF/ejb-jar.xml`

Use configurable Resource Connection Factories:

```
<resource-ref>
  <description>…</description>
  <res-ref-name>jms/qConnFactory</res-ref-name>
  <res-type>javax.jms.QueueConnectionFactory</res-type>
  <res-auth>Container</res-auth>
  <res-sharing-scope>Unshareable</res-sharing-scope>
</resource-ref>
```

Existing Approaches on EE

JSR 346: CDI 1.2

```
META-INF/beans.xml
META-INF/services/javax.enterprise.inject.spi.Extension
```

CDI Extensions

- Allow to change the CDI Model during deployment
  - Add/remove beans
  - Add/remove interceptors
  - Add/remove decorators
  - Customize Injections, Events
  - …

Existing Approaches on EE

JSR 349: Bean Validation 1.1; JSR 344: JSF 2.2

Bean Validation

`META-INF/validation.xml`
`META-INF/services/javax.enterprise.inject.spi.Extension`

`ConstraintValidatorFactory`, `MessageInterpolator`, `ParameterNameProvider` or `TraversableResolver` can be configured.

Existing Approaches on EE

JSF

- `javax.faces.application.ApplicationConfigurationPopulator`

-> This class defines a `java.util.ServiceLoader` service which enables programmatic configuration of the JSF runtime (XML `Document`) using the existing Application Configuration Resources schema.

Existing Approaches on EE

Overview

- **Deployment Descriptors**
  - Standardized: `web.xml`, `persistence.xml`, `ejb-jar.xml`, `beans.xml`
  - Vendor-specific: mostly administrative resources
- `<alt-dd>` feature (see http://tomee.apache.org/alternate-descriptors.html)

- **JSF Stages** (Development,Production,SystemTest,UnitTest)
- **JNDI Tree**

-> configuration must be known at build time!
-> No support for dynamic placeholders, e.g. using EL expressions
-> Despite CDI, JSF and Bean Validation not much support for deployment time configuration.

Existing Approaches (Java SE)

Existing Approaches Java SE (1)

Spring PropertyResolver

application.properties

```
appl.name=My Web Application
appl.home=/Users/webapp/applicat
ion/
db.driver=org.hsqldb.jdbcDriver
db.name=v8max.db
db.url=jdbc:hsqldb:file://${appl
.home}/

database/${db.name}
db.user=SA
db.pass=
```

```
<bean id="applicationProperties"
class="org.springframework.beans.factory.config.
                                PropertyPlaceholderConfigurer">
    <property name="location" value="classpath:application.properties"/>
</bean>
<bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource">
    <property name="driverClassName" value="${db.driver}"/>
    <property name="url" value="${db.url}"/>
    <property name="username" value="${db.user}"/>
    <property name="password" value="${db.pass}"/>
</bean>
```

… and you can write your own `PropertySource`/ `PropertyPlaceHolder`!

Existing Approaches Java SE (2)

Apache Deltaspike Configuration

application.properties

```
appl.name=My Web Application
appl.home=/Users/webapp/application/
db.driver=org.hsqldb.jdbcDriver
db.name=v8max.db
db.url=jdbc:hsqldb:file://${appl.hom
e}/

database/${db.name}
db.user=SA
db.pass=
```

```
public class MyCustomPropertyFileConfig
implements PropertyFileConfig {
  @Override
  public String getPropertyFileName() {
    return "application.properties";
  }
}
```

```
@ApplicationScoped
public class SomeRandomService {
  @Inject @ConfigProperty(name = "endpoint.poll.interval")
  private Integer pollInterval;

  @Inject @ConfigProperty(name = "endpoint.poll.servername")
  private String pollUrl;
  ...
}
```

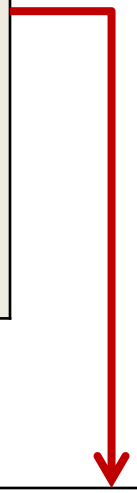
Existing Approaches Java SE (3)

Apache Commons Configuration

application.properties

```
appl.name=My Web Application
appl.home=/Users/webapp/applica
tion/
db.driver=org.hsqldb.jdbcDriver
db.name=v8max.db
db.url=jdbc:hsqldb:file://${app
l.home}/

database/${db.name}
db.user=SA
db.pass=
```



```
Configuration config = new PropertiesConfiguration("application.properties");
String dbUrl = config.getString("db.url"));
…
```

Existing Approaches Java SE (4)

ConfigBuilder (1)

```
@PropertiesFiles("config") // "config.properties", "config.<hostname>.properties", etc.
@PropertyLocations(directories = {"/home/user"}, contextClassLoader = true)
@PropertySuffixes(extraSuffixes = {"tngtech","myname"}, hostNames = true)
public class MyBean{

  public static class StringToPidFixTransformer
  implements TypeTransformer<String,PidFix> {
    @Override public PidFix transform(String input) { ... }
  }

  @DefaultValue("false") // values are automatically be converted to primitive types
  @CommandLineValue(shortOpt="t", longOpt="test", hasArg=false) // flag arg.
  private boolean runInTestMode;

  @DefaultValue("3")
  @CommandLineValue(shortOpt="rl", longOpt="runLevel", hasArg=true)
  private int runLevel;

  @EnvironmentVariableValue("PATH")
  @PropertyValue("path") // maps to the key "path" in the properties file
  private String path;
```

Existing Approaches Java SE (4)

ConfigBuilder (2)

```
  @SystemPropertyValue("user.name") // maps to field "user.name" in sys props
  @NotEmpty("username.notEmpty")    // JSR-303 validation
  private String userName;

  @TypeTransformers(StringToPidFixTransformer.class)
  @CommandLineValue(shortOpt="pc", longOpt="pidFixCollection",
                    hasArg=true)
  private Collection<PidFix> pidFixCollection;

  @TypeTransformers(StringToPidFixTransformer.class)
  @CommandLineValue(shortOpt="p", longOpt="pidFix", hasArg=true)
  private PidFix pidFix;

  @Validation
  private void validate() { ... }

  ...
}
```

Existing Approaches Java SE (5)

JFig

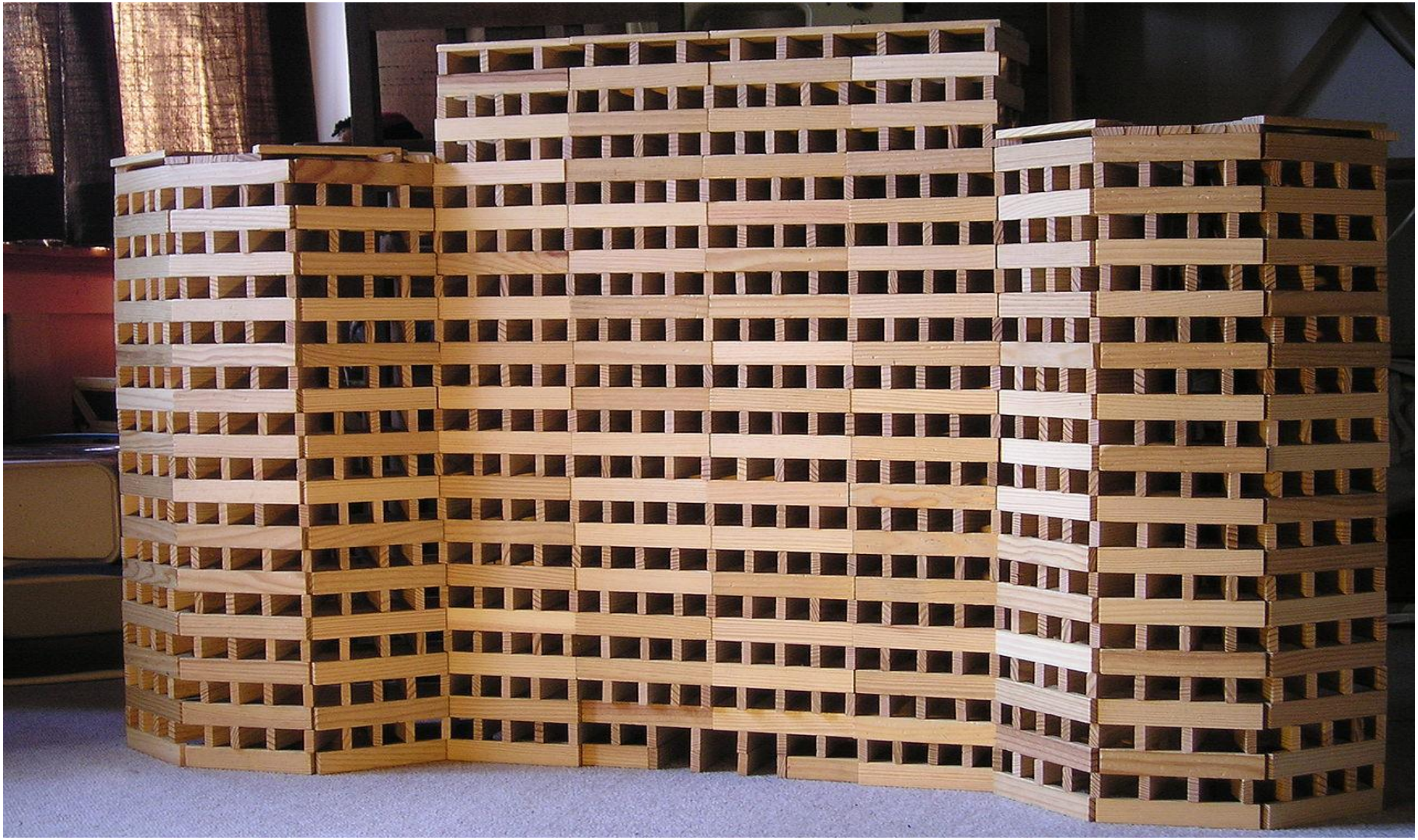
application.xml

```
<configuration>
  <section name="app1">
    <entry key="name" value="My Web Application"/>
    …
  </section>
  <section name="db">
    <entry key="driver" value="org.hsqldb.jdbcDriver" />
    <entry key="name" value="v8max.db" />
    <entry key="url"
value="db.url=jdbc:hsqldb:file://${appl.home}/
                      database/${db.name}" />
    <entry key="user" value="SA" />
    <entry key="pass" />
  </section>
</configuration>
```

```
java -Dconfig.filename=application.xml <yourClassName>

JFig.initialize();
String dbDriver = JFig.getInstance().getValue("db", "driver");
String dnBame = JFig.getInstance().getValue("db", "name");
…
```

Outlook



Common Aspects of Existing Approaches (1)

Java SE

- **Key/value based**
  - Basically all relevant solutions are based on simple key/value pairs (most of the based on Strings only)
- **Environment dependent (contextual)**
  - The Environment can be hierarchical, the stage is part of the env.
- **Composite**
  - Configuration can be composed from smaller composites
- **Multi-sourced**
  - Configuration can be provided by files, classpath resources, JNDI, systen/env properties, JNDI, preferences, databases, …
- **Multi-formatted**
  - Properties, xml-properties, json and dialects, other formats

-> Flexible and proven approaches

-> But no common API!

Common Aspects of Existing Approaches (2)

Java EE

- Mostly Xml based
- Multiple locations
- Mainly build time configuration only
- Only few SPIs enable deploy time configuration
- Administrative Resources are vendor specific
- No standard for EE application configuration

-> OK so far, but Cloud usage require more flexibility (deploy time configuration).
-> Some aspects can be added from outside during appliance assembly

Review & Proposal

How to go forward?

- Configuration is a classical Cross Cutting Concern
- Problem Domain of Configuration is well known
- Different Solutions already exist since long time
- Nevertheless almost no standard exists in Java!
- Java EE lacks of flexibility for several scenarios

-> Proposal: Standardize Configuration Acess in ME/SE

+ ME/SE targets all Java code

+ EE can benefit as well, where useful

-> … and, well, make EE more flexible

Configuration API for the Java Platform

Proposal for Java ME/SE

Configuration for the Java Platform

SE JSR Proposal

- **JSR based on SE 8, API also compatible with ME 8**
- **RI & TCK based on SE 8**
- **Defining a uniform model for Configuration Access**
  - String key/value based `PropertyProvider`
  - Higher level API with functional extension points
  - Can be implemented by several configuration solutions
  - To be discussed (may be not in scope for first release):
    - Layering, Composites and Remote Configuration
    - Configuration changes, react/listen for Changes
    - SPIs for multiple formats, type adapters
- **Optionally also define integration semantics in CDI?**

A Possible Configuration SE API (3)

PropertyProvider

```
public interface PropertyProvider {
  int size();
  default boolean isEmpty(){…}
  default boolean containsKey(String key){…}
  default Set<String> keySet(){…}
  default String get(String key);
  String getOrDefault(String key, String defValue){…}
  MetaInfo getMetaInfo();
  Map<String, String> toMap();
  default boolean isMutable(){ return false; }
  default MutablePropertyProvider toMutableProvider(){…}
  default void load(){}
}
```

```
PropertyProvider prov1 =
  PropertyProviders
    .of("classpath*:META-INF/config-*.xml");
PropertyProvider prov1 =
  PropertyProviders.ofSystemPropertiesProvider();
PropertyProvider prov = PropertyProviders.ofUnion(
  AggregatioPolicy.OVERRIDE, prov1, prov2
)
```

A Possible Configuration SE API (4)

Configuration

```
public interface Configuration extends PropertyProvider{

  default Boolean getBoolean(String key){… }
  default Boolean getBooleanOrDefault(String key,
                                      Boolean defaultValue){…}
  default Integer getInteger (String key){… }
  default Integer getIntegerOrDefault(String key,
                                        Integer defaultValue){…}
  …
  default <T> T getAdapted(String key, PropertyAdapter<T> adapter){…}
  default <T> T getAdaptedOrDefault(String key,
                                    PropertyAdapter<T> adapter,
                                    T defaultValue){…}

  Set<String> getAreas();
  Set<String> getTransitiveAreas();
  …
  default Configuration with(Config
  default <T> T query(ConfigQuery<T> query){…}
  void addConfigChangeListener(ConfigChangeListener l);
  void removeConfigChangeListener(ConfigChangeListener l);
}
```

```
Configuration config =
  ConfigurationManager
    .getConfiguration("myProductConfig");
Annotation[] annots = …;
Configuration config =
  ConfigurationManager.getConfiguration(annots);
```

A Possible Configuration SE API (2)

Modelling the Environment

```
public interface Environment
extends PropertyProvider{

  Environment getParentEnvironment();

}
```

```
Environmemt env =
  EnvironmentManager
    .getEnvironment();
Environmemt rootEnv =
  EnvironmentManager
    .getRootEnvironment();
```

A Possible Configuration SE API (5)

Example: Configuration Population

```
@ConfiguredBean(”com.mycomp.mysolution.tenantAdress”)
public final class MyTenant{

  @Configured(updatedPolicy=UpdatePolicy.NEVER)
  private String name;

  @Configured
  private long customerId;

  @Configured({”privateAddress”, ”businessAdress”})
  private String address;

  @ConfigChangeListener
  private void configChanged(ConfigChangeEvent evt){…}
}

MyTenant t = new MyTenant();
ConfigurationManager.configure(t);
```

An advanced Example

What can we do so far?

```
Map<String,String> cfgMap = new HashMap<>();
cfgMap.put("a", "Adrian"); // overrides Anatole

Configuration config = ConfigurationBuilder.of("myTestConfig")
  .addResources("classpath*:test.properties")
  .addPropertyProviders(
     AggregationPolicy.OVERRIDE,
     PropertyProviders.fromPaths("classpath:cfg/test.xml"),
     PropertyProviders.fromArgs(new String[]{"-arg1", "--fullarg",
                                             "fullValue", "-myflag"}),
     PropertyProviders.from(cfgMap)).build();


System.out.println(config);
System.out.println(config.getAreas(s -> s.startsWith("another")));

PropertyProvider filtered = PropertyProviders.filterSet(
   (f) -> {return f.equals("a") || f.equals("b") || f.equals("c");},
   config);
```

Summary

Summary

- **Make configuration a first class citizen in Java**
- **Benefits are spread across the whole platform**
- **Start modestly but design for the future**

- **But is the vision reasonable…?**

The Current State

What happened so far?

- **Java EE Configuration**
  - Would mainly enable Java EE to be configurable
  - I would suggest only deploy/startup time configuration
  - Currently **on hold** (deferred)

- **Configuration for the Java Platform (next slide)**
  - Would define the generic mechanism how configuration is modelled
  - Discussed as SE JSR, we need more support

-> Discuss today at 16:00 at Sutter room in the Hilton Hotel !

Links

- Java.net Project: http://java.net/projects/javamconfig
- GitHub Project (early stage): https://github.com/java-config
- Twitter: @javaconfig
- Presentation of 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/

Q & A

DEMO

&

???

The End

Thank you!

Appendix

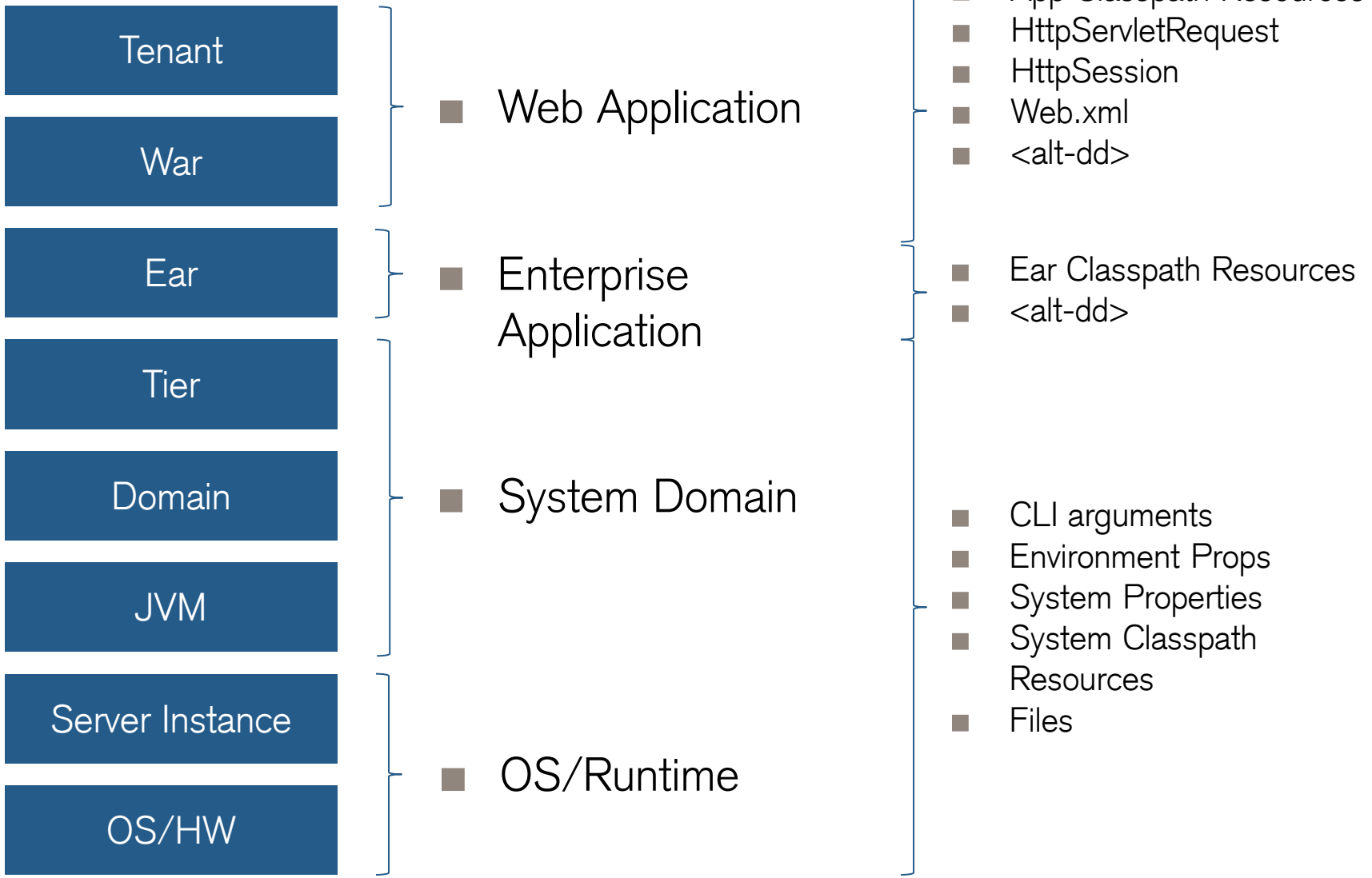
Layering and Composites (1)

Overview

- **Systems are built on design layers, e.g.**
  - Boot Runtime
  - Container Infrastructure
  - Components
  - API Adapters
  - …
- **Systems can be assembled using prebuilt modules or plugins**
- **Configuration typically comes with override mechanisms, hardcoding them makes configuration logic unmaintainable and untestable**

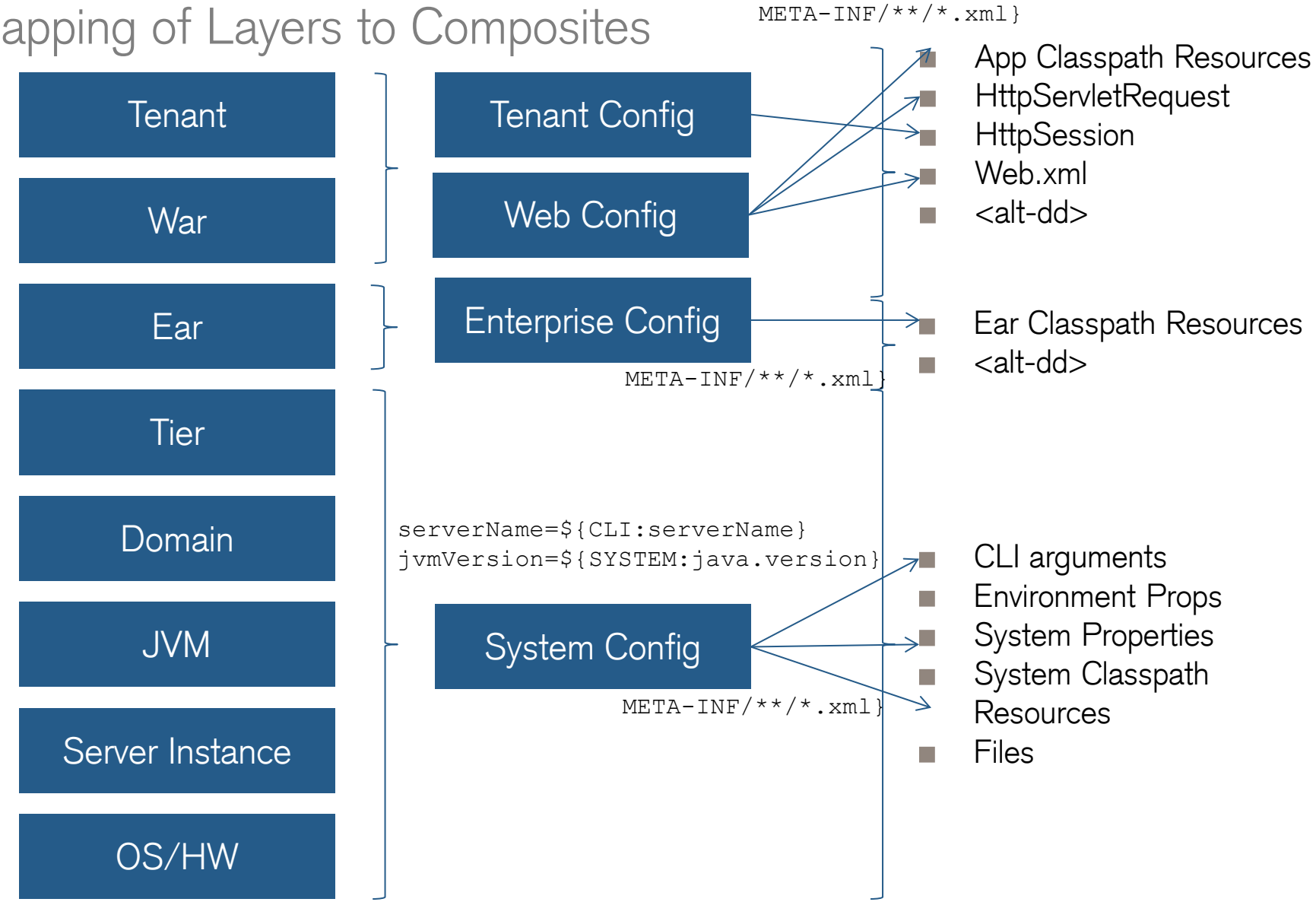
Layering and Composites (1)

Possible Layers in a Complex Setup



Layering and Composites (2)

Mapping of Layers to Composites



Dynamic Provisioning

Is it possible with Java EE ?

Initial Setup

- Create Java Appserver Node
- Start the Server

Dynamic Deployment

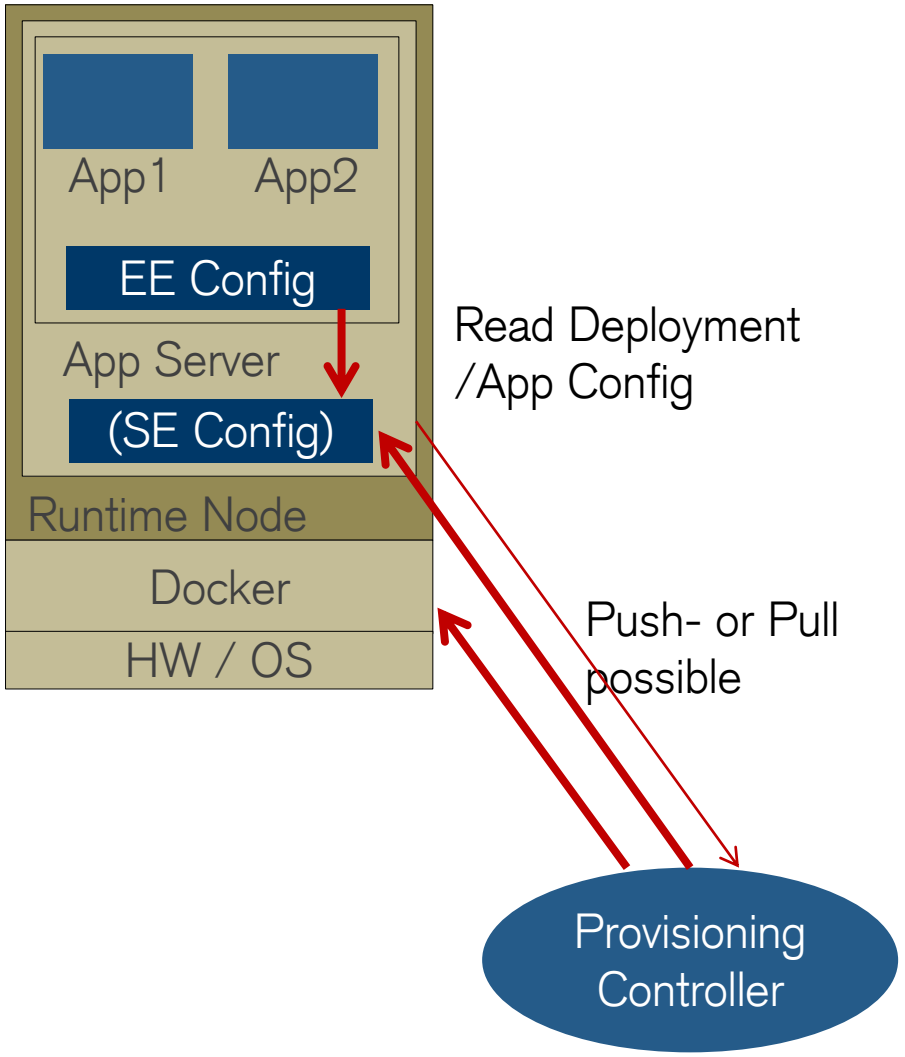
- Install the artifacts
- Deploy the artifacts
- Read deployment and app configuration (remote)
- Write status back to controller

Reconfiguration

- Restart the artifacts (deployment and app config will be reread), or
- Only update the app configuration

Undeployment

- Undeploy the artifacts



Combining SE and EE Configuration