Howto

How to create a simple maven project using Meecrowave ?

You should add the following dependencies do the dependencies section of your pom.xml (adjust version to current stable version)

```
<dependency>
    <groupId>org.apache.meecrowave</groupId>
    <artifactId>meecrowave-specs-api</artifactId>
    <version>${meecrowave.version}</version>
</dependency>
<dependency>
    <groupId>org.apache.meecrowave</groupId>
    <artifactId>meecrowave-core</artifactId>
    <version>${meecrowave.version}</version>
</dependency>

<!-- if you intend to have unit tests (you really should) -->
<dependency>
    <groupId>org.apache.meecrowave</groupId>
    <artifactId>meecrowave-junit</artifactId>
    <version>${meecrowave.version}</version>
    <scope>test</scope>
</dependency>
```

and the following plugin configuration to the build/plugins section of your pom.xml

```
<plugin>
    <!--
    For starting meecrowave via Maven. Just run
    $> mvn clean install meecrowave:run
    -->
    <groupId>org.apache.meecrowave</groupId>
    <artifactId>meecrowave-maven-plugin</artifactId>
    <version>${meecrowave.version}</version>
</plugin>
```

Then, you can start your app by running

```
mvn clean install meecrowave:run
```

How to add a REST Endpoint ?

You should declare your endpoint path and verd :

```
package org.mypackage;

import javax.enterprise.context.ApplicationScoped;
import javax.ws.rs.GET;
import javax.ws.rs.Path;

@Path("mypath")
@ApplicationScoped
public class MyEndpoint {

    /**
     * Ping / pong rest GET method, to check backend and replies to queries
     *
     * @return
     */
    @Path("/ping")
    @GET
    public String getPing() {
        return "pong";
    }
}
```

How to add a filter (simple case) ?

Use standard Servlet 4.0 @WebFilter annotation. A simple example :

```
package org.mypackage;

import java.io.IOException;
import javax.servlet.Filter;
import javax.servlet.FilterChain;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import javax.servlet.annotation.WebFilter;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**
 * A simple CORS filter
 *
 */
@WebFilter(asyncSupported = true, urlPatterns = {"/*"})
public class CORSFilter implements Filter {

    /**
     * A basic CORS filter, allowing everything
     */
    @Override
    public void doFilter(ServletRequest servletRequest, ServletResponse
servletResponse, FilterChain chain)
            throws IOException, ServletException {

        HttpServletRequest request = (HttpServletRequest) servletRequest;

        HttpServletResponse response = (HttpServletResponse) servletResponse;
        response.addHeader("Access-Control-Allow-Origin", "*");
        response.addHeader("Access-Control-Allow-Methods","GET, OPTIONS, HEAD, PUT,
POST, DELETE");
        response.addHeader("Access-Control-Allow-Headers","*");

        if (request.getMethod().equals("OPTIONS")) {
            // special case of return code for "OPTIONS" query
            response.setStatus(HttpServletResponse.SC_ACCEPTED);
            return;
        }

        // pass the request along the filter chain
        chain.doFilter(request, servletResponse);
    }
}
```

How to add a servlet ?

If your servlet requires no configuration that you would typically put in the web.xml file, you can use the @WebServlet annotation from the Servlet 3.0 specification.

If you need to configure the servlet, you should use a ServletContainerInitializer.

If you would have a declaration such as :

```
<servlet>
    <description>My Servlet</description>
    <servlet-name>MyServlet</servlet-name>
    <servlet-class>org.my.servlet.ImplementationClass</servlet-class>
    <init-param>
        <param-name>param-name</param-name>
        <param-value>My param value</param-value>
    </init-param>
    <load-on-startup>0</load-on-startup>
    <async-supported>true</async-supported>
</servlet>
<servlet-mapping>
    <servlet-name>MyServlet</servlet-name>
    <url-pattern>/my_mapping/*</url-pattern>
</servlet-mapping>
```

in your web.xml, you would have a SerlvetContainerInitializer such as :

```
package org.mypackage;

import java.util.Set;

import javax.servlet.ServletContainerInitializer;
import javax.servlet.ServletContext;
import javax.servlet.ServletRegistration;

import org.my.servlet.ImplementationClass;

public class MyServletContainerInitializer implements ServletContainerInitializer {
    @Override
    public void onStartup(final Set<Class<?>> c, final ServletContext context) {
        final ServletRegistration.Dynamic def = context.addServlet("My Servlet",
ImplementationClass.class);
        def.setInitParameter("param-name", "My param value");

        def.setLoadOnStartup(0);
        def.addMapping("/my_mapping/*");
        def.setAsyncSupported(true);
    }
}
```

Then, you should register this implementation of ServletContainerInitializer:

- in a SPI, in src/main/resources/META-INF/services/javax.servlet.ServletContainerInitializer:

```
org.mypackage.MyServletContainerInitializer
```

- or add it to Meecrowave configuration using a Meecrowave.ConfigurationCustomizer such as :

```
package org.mypackage;

import org.apache.meecrowave.Meecrowave;

public class ServletContainerInitializerCustomizer implements Meecrowave
.ConfigurationCustomizer {
    @Override
    public void accept(final Meecrowave.Builder builder) {
        builder.addServletContextInitializer(new MyServletContainerInitializer());
    }
}
```

Using this last option, the configuration will also be performed before unit tests are executed.

Your implementation of Meecrowave.ConfigurationCustomizer should be added to the configuration by appending its canonical name to the src/main/resources/META-

INF/org.apache.meecrowave.Meecrowave$ConfigurationCustomizer file.

How to add a valve ?

Simple cases should be handled using a meecrowave.properties file.

More complex cases can be handled using an implementation of Meecrowave.ConfigurationCustomizer.

In the following example, we instantiate a Tomcat RewriteValve and load the rewrite.config file we usually put in src/main/webapp/WEB-INF in a webapp packaged as a war, and that we would put in src/main/resources in a meecrowave app :

```
package org.mypackage;

import java.io.IOException;
import java.io.InputStream;
import lombok.extern.log4j.Log4j2;
import org.apache.catalina.LifecycleException;
import org.apache.catalina.valves.rewrite.RewriteValve;
import org.apache.meecrowave.Meecrowave;

/**
 * A bit of glue to set proxy / RewriteValve configuration at startup
 *
 */
@Log4j2
public class RewriteValveCustomizer implements Meecrowave.ConfigurationCustomizer {
    final String PROXY_CONFIG = "rewrite.config";
    @Override
    public void accept(final Meecrowave.Builder builder) {
        log.info("Loading proxy / rewrite configuration from {}", PROXY_CONFIG);
        log.info("This file should be in src/main/resources in project sources");
        try (InputStream stream = Thread.currentThread().getContextClassLoader()
.getResourceAsStream(PROXY_CONFIG)) {
            if (null == stream) {
                log.info("Rewrite configuration file {} not found", PROXY_CONFIG);
                return;
            }
            configuration = new BufferedReader(new InputStreamReader(stream)).lines()
.collect(Collectors.joining("\n"));
        } catch (IOException ex) {
            log.error("Error reading rewrite / proxy configuration file {}",
PROXY_CONFIG);
            return;
        }
        final RewriteValve proxy = new RewriteValve() {
            @Override
            protected synchronized void startInternal() throws LifecycleException {
```

```
                super.startInternal();
                try {
                    setConfiguration(configuration);
                } catch (final Exception e) {
                    throw new LifecycleException(e);
                }
            }
        };
        // at this time, we are still single threaded. So, this should be safe.
        builder.instanceCustomizer(tomcat -> tomcat.getHost().getPipeline().addValve
(proxy));
        log.info("Proxy / rewrite configuration valve configured and added to tomcat.
");
    }
}
```

Your implementation of Meecrowave.ConfigurationCustomizer should be added to the configuration by appending its canonical name to the src/main/resources/META-INF/org.apache.meecrowave.Meecrowave$ConfigurationCustomizer file.

A more complex example is available on Romain Manni-Bucau's blog.

How to add a web frontend ?

You should add a <webapp> element to the meecrowave plugin configuration. Example :

```
<plugin>
    <!--
        For starting meecrowave via Maven. Just run
        $> mvn clean install meecrowave:run
    -->
    <groupId>org.apache.meecrowave</groupId>
    <artifactId>meecrowave-maven-plugin</artifactId>
    <version>${meecrowave.version}</version>
    <configuration>
        <!-- include packaged app as webapp -->
        <webapp>src/main/webapp/dist</webapp>
    </configuration>
</plugin>
```

will add the content of the "dist" folder to your package and its files will be available on the application root.

Note that your frontend will be served when executing the app (on a mvn meecrowave:run or when running a packaged app). It will not be available during unit tests.

How to compile a Meecrowave application with GraalVM

You can use `native-image` directly but for this how to, we will use Apache Arthur which enables to do it through Apache Maven. The trick is to define the Tomcat and Meecrowave resources to use to convert the Java application in a native binary. For a simple application here is how it can be done.

we use Yupiik Logging in this sample to replace Log4j2 which is not GraalVM friendly, this JUL implementation enables runtime configuration even for Graalified binaries.

```
<plugin> <!-- mvn -Parthur arthur:native-image@runtime -e -->
  <groupId>org.apache.geronimo.arthur</groupId>
  <artifactId>arthur-maven-plugin</artifactId>
  <version>${arthur.version}</version> <!-- >= 1.0.3 or replace openwebbeans extension
by openwebbeans 2.0.22 dep + openwebbeans-knight with arthur v1.0.2 -->
  <executions>
    <execution>
      <id>graalify</id>
      <phase>package</phase>
      <goals>
        <goal>native-image</goal>
      </goals>
    </execution>
  </executions>
  <configuration>
    <graalVersion>21.0.0.2.r11</graalVersion> <!-- use this graal version (java 11
here) -->
    <main>org.apache.meecrowave.runner.Cli</main> <!-- set up meecrowave default main
- requires commons-cli -->
    <buildStaticImage>false</buildStaticImage> <!-- up to you but using arthur docker
goals it works fine and avoids some graalvm bugs -->
    <usePackagedArtifact>true</usePackagedArtifact> <!-- optional but enables a more
deterministic run generally -->
    <graalExtensions> <!-- enable CDI -->
      <graalExtension>openwebbeans</graalExtension>
    </graalExtensions>
    <reflections> <!-- enable cxf/owb/tomcat main reflection points  -->
      <reflection> <!-- used by meecrowave to test cxf presence -->
        <name>org.apache.cxf.BusFactory</name>
      </reflection>
      <reflection>
        <name>javax.ws.rs.core.UriInfo</name>
        <allPublicMethods>true</allPublicMethods>
      </reflection>
      <reflection>
        <name>javax.ws.rs.core.HttpHeaders</name>
        <allPublicMethods>true</allPublicMethods>
```

```
    </reflection>
    <reflection>
      <name>javax.ws.rs.core.Request</name>
      <allPublicMethods>true</allPublicMethods>
    </reflection>
    <reflection>
      <name>javax.ws.rs.core.SecurityContext</name>
      <allPublicMethods>true</allPublicMethods>
    </reflection>
    <reflection>
      <name>javax.ws.rs.ext.Providers</name>
      <allPublicMethods>true</allPublicMethods>
    </reflection>
    <reflection>
      <name>javax.ws.rs.ext.ContextResolver</name>
      <allPublicMethods>true</allPublicMethods>
    </reflection>
    <reflection>
      <name>javax.servlet.http.HttpServletRequest</name>
      <allPublicMethods>true</allPublicMethods>
    </reflection>
    <reflection>
      <name>javax.servlet.http.HttpServletResponse</name>
      <allPublicMethods>true</allPublicMethods>
    </reflection>
    <reflection>
      <name>javax.ws.rs.core.Application</name>
      <allPublicMethods>true</allPublicMethods>
    </reflection>
    <reflection> <!-- meecrowave registers it programmatically -->
      <name>org.apache.meecrowave.cxf.JAXRSFieldInjectionInterceptor</name>
      <allPublicConstructors>true</allPublicConstructors>
      <allPublicMethods>true</allPublicMethods>
    </reflection>
    <reflection> <!-- CXF SPI -->
      <name>org.apache.cxf.bus.managers.CXFBusLifeCycleManager</name>
      <allPublicConstructors>true</allPublicConstructors>
    </reflection>
    <reflection> <!-- CXF SPI -->
      <name>org.apache.cxf.bus.managers.ClientLifeCycleManagerImpl</name>
      <allPublicConstructors>true</allPublicConstructors>
    </reflection>
    <reflection> <!-- CXF SPI -->
      <name>org.apache.cxf.bus.managers.EndpointResolverRegistryImpl</name>
      <allPublicConstructors>true</allPublicConstructors>
    </reflection>
    <reflection> <!-- CXF SPI -->
      <name>org.apache.cxf.bus.managers.HeaderManagerImpl</name>
      <allPublicConstructors>true</allPublicConstructors>
    </reflection>
    <reflection> <!-- CXF SPI -->
```

```
    <name>org.apache.cxf.bus.managers.PhaseManagerImpl</name>
    <allPublicConstructors>true</allPublicConstructors>
  </reflection>
  <reflection> <!-- CXF SPI -->
    <name>org.apache.cxf.bus.managers.ServerLifeCycleManagerImpl</name>
    <allPublicConstructors>true</allPublicConstructors>
  </reflection>
  <reflection> <!-- CXF SPI -->
    <name>org.apache.cxf.bus.managers.ServerRegistryImpl</name>
    <allPublicConstructors>true</allPublicConstructors>
  </reflection>
  <reflection> <!-- CXF SPI -->
    <name>org.apache.cxf.bus.managers.WorkQueueManagerImpl</name>
    <allPublicConstructors>true</allPublicConstructors>
  </reflection>
  <reflection> <!-- CXF SPI -->
    <name>org.apache.cxf.bus.resource.ResourceManagerImpl</name>
    <allPublicConstructors>true</allPublicConstructors>
  </reflection>
  <reflection> <!-- CXF SPI -->
    <name>org.apache.cxf.catalog.OASISCatalogManager</name>
    <allPublicConstructors>true</allPublicConstructors>
  </reflection>
  <reflection> <!-- CXF SPI -->
    <name>org.apache.cxf.common.spi.ClassLoaderProxyService</name>
    <allPublicConstructors>true</allPublicConstructors>
  </reflection>
  <reflection> <!-- CXF SPI -->
    <name>org.apache.cxf.common.util.ASMHelperImpl</name>
    <allPublicConstructors>true</allPublicConstructors>
  </reflection>
  <reflection> <!-- CXF SPI -->
    <name>org.apache.cxf.service.factory.FactoryBeanListenerManager</name>
    <allPublicConstructors>true</allPublicConstructors>
  </reflection>
  <reflection> <!-- CXF SPI -->
    <name>org.apache.cxf.transport.http.HTTPTransportFactory</name>
    <allPublicConstructors>true</allPublicConstructors>
  </reflection>
  <reflection> <!-- CXF SPI -->
    <name>org.apache.cxf.catalog.OASISCatalogManager</name>
    <allPublicConstructors>true</allPublicConstructors>
  </reflection>
  <reflection> <!-- CXF SPI -->
    <name>org.apache.cxf.endpoint.ClientLifeCycleManager</name>
    <allPublicConstructors>true</allPublicConstructors>
  </reflection>
  <reflection> <!-- CXF SPI -->
    <name>org.apache.cxf.buslifecycle.BusLifeCycleManager</name>
    <allPublicConstructors>true</allPublicConstructors>
  </reflection>
```

```
    <reflection> <!-- CXF SPI -->
      <name>org.apache.cxf.phase.PhaseManager</name>
      <allPublicConstructors>true</allPublicConstructors>
    </reflection>
    <reflection> <!-- CXF SPI -->
      <name>org.apache.cxf.resource.ResourceManager</name>
      <allPublicConstructors>true</allPublicConstructors>
    </reflection>
    <reflection> <!-- CXF SPI -->
      <name>org.apache.cxf.headers.HeaderManager</name>
      <allPublicConstructors>true</allPublicConstructors>
    </reflection>
    <reflection> <!-- CXF SPI -->
      <name>org.apache.cxf.common.util.ASMHelper</name>
      <allPublicConstructors>true</allPublicConstructors>
    </reflection>
    <reflection> <!-- CXF SPI -->
      <name>org.apache.cxf.common.spi.ClassLoaderService</name>
      <allPublicConstructors>true</allPublicConstructors>
    </reflection>
    <reflection> <!-- CXF SPI -->
      <name>org.apache.cxf.endpoint.EndpointResolverRegistry</name>
      <allPublicConstructors>true</allPublicConstructors>
    </reflection>
    <reflection> <!-- CXF SPI -->
      <name>org.apache.cxf.endpoint.ServerLifeCycleManager</name>
      <allPublicConstructors>true</allPublicConstructors>
    </reflection>
    <reflection> <!-- CXF SPI -->
      <name>org.apache.cxf.workqueue.WorkQueueManager</name>
      <allPublicConstructors>true</allPublicConstructors>
    </reflection>
    <reflection> <!-- CXF SPI -->
      <name>org.apache.cxf.endpoint.ServerRegistry</name>
      <allPublicConstructors>true</allPublicConstructors>
    </reflection>
    <reflection>
      <name>org.apache.cxf.cdi.DefaultApplication</name>
      <allPublicConstructors>true</allPublicConstructors>
      <allPublicMethods>true</allPublicMethods>
    </reflection>
    <reflection>
      <name>org.apache.cxf.transport.http.Headers</name>
      <allPublicMethods>true</allPublicMethods>
    </reflection>
    <reflection>
      <name>org.apache.cxf.jaxrs.JAXRSBindingFactory</name>
      <allPublicConstructors>true</allPublicConstructors>
    </reflection>
    <reflection> <!-- used by cxf-cdi to test owb-web presence -->
      <name>org.apache.webbeans.web.lifecycle.WebContainerLifecycle</name>
```

```
        <allPublicConstructors>true</allPublicConstructors>
      </reflection>
      <reflection> <!-- instantiated by a SPI -->
        <name>org.apache.meecrowave.logging.tomcat.LogFacade</name>
        <allPublicConstructors>true</allPublicConstructors>
      </reflection>
      <reflection> <!-- for default binding since meecrowave uses a filter for jaxrs
-->
        <name>org.apache.catalina.servlets.DefaultServlet</name>
        <allPublicConstructors>true</allPublicMethods>
      </reflection>
      <reflection> <!-- tomcat does reflection on it -->
        <name>org.apache.catalina.loader.WebappClassLoader</name>
        <allPublicMethods>true</allPublicMethods>
      </reflection>
      <reflection> <!-- tomcat does reflection on it -->
        <name>org.apache.tomcat.util.descriptor.web.WebXml</name>
        <allPublicMethods>true</allPublicMethods>
      </reflection>
      <reflection> <!-- tomcat does reflection on it -->
        <name>org.apache.coyote.http11.Http11NioProtocol</name>
        <allPublicMethods>true</allPublicMethods>
      </reflection>
      <reflection> <!-- tomcat instantiates it by reflection -->
        <name>org.apache.catalina.authenticator.NonLoginAuthenticator</name>
        <allPublicConstructors>true</allPublicConstructors>
      </reflection>
      <reflection> <!-- should be all API a proxy can be created for -->
        <name>javax.servlet.ServletContext</name>
        <allPublicMethods>true</allPublicMethods>
      </reflection>
      <reflection> <!-- used by meecrowave integration -->
        <name>org.apache.cxf.jaxrs.provider.ProviderFactory</name>
        <methods>
          <method>
            <name>getReadersWriters</name>
          </method>
        </methods>
      </reflection>
      <reflection> <!-- used by meecrowave integration -->
        <name>
org.apache.johnzon.jaxrs.jsonb.jaxrs.JsonbJaxrsProvider$ProvidedInstance</name>
        <fields>
          <field>
            <name>instance</name>
          </field>
        </fields>
      </reflection>
      <reflection>
        <name>org.apache.johnzon.jaxrs.jsonb.jaxrs.JsonbJaxrsProvider</name>
        <fields>
```

```
          <field>
            <name>providers</name>
          </field>
        </fields>
      </reflection>
      <reflection> <!-- not needed with arthur 1.0.3 -->
        <name>org.apache.xbean.finder.AnnotationFinder</name>
        <fields>
          <field>
            <name>linking</name>
            <allowWrite>true</allowWrite>
          </field>
        </fields>
      </reflection>
    </reflections>
    <resources> <!-- register tomcat resources and optionally default meecrowave app
configuration -->
      <resource>
        <pattern>org\/apache\/catalina\/.*\.properties</pattern>
      </resource>
      <resource>
        <pattern>javax\/servlet\/(jsp\/)?resources\/.*\.(xsd|dtd)</pattern>
      </resource>
      <resource>
        <pattern>meecrowave\.properties</pattern>
      </resource>
      <resource>
        <pattern>META-INF/cxf/bus-extensions\.txt</pattern>
      </resource>
      <resource>
        <pattern>org/apache/cxf/version/version\.properties</pattern>
      </resource>
    </resources>
    <includeResourceBundles>
      <includeResourceBundle>org.apache.cxf.Messages</includeResourceBundle>
      <includeResourceBundle>
org.apache.cxf.interceptor.Messages</includeResourceBundle>
      <includeResourceBundle>
org.apache.cxf.bus.managers.Messages</includeResourceBundle>
      <includeResourceBundle>org.apache.cxf.jaxrs.Messages</includeResourceBundle>
      <includeResourceBundle>
org.apache.cxf.jaxrs.provider.Messages</includeResourceBundle>
      <includeResourceBundle>
org.apache.cxf.jaxrs.interceptor.Messages</includeResourceBundle>
      <includeResourceBundle>
org.apache.cxf.jaxrs.utils.Messages</includeResourceBundle>
      <includeResourceBundle>
org.apache.cxf.transport.servlet.Messages</includeResourceBundle>
      <includeResourceBundle>
org.apache.catalina.authenticator.LocalStrings</includeResourceBundle>
      <includeResourceBundle>
```

```
org.apache.catalina.connector.LocalStrings</includeResourceBundle>
      <includeResourceBundle>
org.apache.catalina.core.LocalStrings</includeResourceBundle>
      <includeResourceBundle>
org.apache.catalina.deploy.LocalStrings</includeResourceBundle>
      <includeResourceBundle>
org.apache.catalina.filters.LocalStrings</includeResourceBundle>
      <includeResourceBundle>
org.apache.catalina.loader.LocalStrings</includeResourceBundle>
      <includeResourceBundle>
org.apache.catalina.manager.host.LocalStrings</includeResourceBundle>
      <includeResourceBundle>
org.apache.catalina.manager.LocalStrings</includeResourceBundle>
      <includeResourceBundle>
org.apache.catalina.mapper.LocalStrings</includeResourceBundle>
      <includeResourceBundle>
org.apache.catalina.realm.LocalStrings</includeResourceBundle>
      <includeResourceBundle>
org.apache.catalina.security.LocalStrings</includeResourceBundle>
      <includeResourceBundle>
org.apache.catalina.servlets.LocalStrings</includeResourceBundle>
      <includeResourceBundle>
org.apache.catalina.session.LocalStrings</includeResourceBundle>
      <includeResourceBundle>
org.apache.catalina.startup.LocalStrings</includeResourceBundle>
      <includeResourceBundle>
org.apache.catalina.users.LocalStrings</includeResourceBundle>
      <includeResourceBundle>
org.apache.catalina.util.LocalStrings</includeResourceBundle>
      <includeResourceBundle>
org.apache.catalina.valves.LocalStrings</includeResourceBundle>
      <includeResourceBundle>
org.apache.catalina.valves.rewrite.LocalStrings</includeResourceBundle>
      <includeResourceBundle>
org.apache.catalina.webresources.LocalStrings</includeResourceBundle>
      <includeResourceBundle>
org.apache.coyote.http11.filters.LocalStrings</includeResourceBundle>
      <includeResourceBundle>
org.apache.coyote.http11.LocalStrings</includeResourceBundle>
      <includeResourceBundle>
org.apache.coyote.http11.upgrade.LocalStrings</includeResourceBundle>
      <includeResourceBundle>
org.apache.coyote.http2.LocalStrings</includeResourceBundle>
      <includeResourceBundle>org.apache.coyote.LocalStrings</includeResourceBundle>
      <includeResourceBundle>
org.apache.tomcat.util.buf.LocalStrings</includeResourceBundle>
      <includeResourceBundle>
org.apache.tomcat.util.codec.binary.LocalStrings</includeResourceBundle>
      <includeResourceBundle>
org.apache.tomcat.util.compat.LocalStrings</includeResourceBundle>
      <includeResourceBundle>
```

```
org.apache.tomcat.util.descriptor.LocalStrings</includeResourceBundle>
      <includeResourceBundle>
org.apache.tomcat.util.descriptor.tld.LocalStrings</includeResourceBundle>
      <includeResourceBundle>
org.apache.tomcat.util.descriptor.web.LocalStrings</includeResourceBundle>
      <includeResourceBundle>
org.apache.tomcat.util.digester.LocalStrings</includeResourceBundle>
      <includeResourceBundle>
org.apache.tomcat.util.http.LocalStrings</includeResourceBundle>
      <includeResourceBundle>
org.apache.tomcat.util.http.parser.LocalStrings</includeResourceBundle>
      <includeResourceBundle>
org.apache.tomcat.util.json.LocalStrings</includeResourceBundle>
      <includeResourceBundle>
org.apache.tomcat.util.LocalStrings</includeResourceBundle>
      <includeResourceBundle>
org.apache.tomcat.util.modeler.LocalStrings</includeResourceBundle>
      <includeResourceBundle>
org.apache.tomcat.util.net.jsse.LocalStrings</includeResourceBundle>
      <includeResourceBundle>
org.apache.tomcat.util.net.LocalStrings</includeResourceBundle>
      <includeResourceBundle>
org.apache.tomcat.util.net.openssl.ciphers.LocalStrings</includeResourceBundle>
      <includeResourceBundle>
org.apache.tomcat.util.net.openssl.LocalStrings</includeResourceBundle>
      <includeResourceBundle>
org.apache.tomcat.util.scan.LocalStrings</includeResourceBundle>
      <includeResourceBundle>
org.apache.tomcat.util.security.LocalStrings</includeResourceBundle>
      <includeResourceBundle>
org.apache.tomcat.util.threads.res.LocalStrings</includeResourceBundle>
      <includeResourceBundle>javax.servlet.LocalStrings</includeResourceBundle>
      <includeResourceBundle>javax.servlet.http.LocalStrings</includeResourceBundle>
    </includeResourceBundles>
    <extensionProperties>
      <extension.annotation.custom.annotations.properties>

javax.json.bind.annotation.JsonbProperty:allDeclaredConstructors=true|allDeclaredMetho
ds=true|allDeclaredFields=true,
        org.apache.meecrowave.runner.cli.CliOption:allDeclaredFields=true
      </extension.annotation.custom.annotations.properties>
      <extension.openwebbeans.extension.excludes>
        org.apache.cxf.Bus,org.apache.cxf.common.util.ClassUnwrapper,
        org.apache.cxf.interceptor.InterceptorProvider,
        io.yupiik.logging.jul,
        org.apache.openwebbeans.se
      </extension.openwebbeans.extension.excludes>
    </extensionProperties>
    <customOptions> <!-- force JUL usage since Log4j2 does not work well at all on
GraalVM -->
      <customOption>
```

```
-
Dopenwebbeans.logging.factory=org.apache.webbeans.logger.JULLoggerFactory</customOptio
n>
      <customOption>-
Djava.util.logging.manager=io.yupiik.logging.jul.YupiikLogManager</customOption>
    </customOptions>
  </configuration>
</plugin>
```

In terms of dependencies you can start with this for example:

```
<dependencies>
  <dependency>
    <groupId>org.apache.meecrowave</groupId>
    <artifactId>meecrowave-specs-api</artifactId>
    <version>${meecrowave.version}</version>
  </dependency>
  <dependency>
    <groupId>org.apache.meecrowave</groupId>
    <artifactId>meecrowave-core</artifactId>
    <version>${meecrowave.version}</version>
  </dependency>
  <dependency> <!-- we use this to have a reconfigurable JUL runtime in native binary
-->
    <groupId>io.yupiik.logging</groupId>
    <artifactId>yupiik-logging-jul</artifactId>
    <version>1.0.0</version>
  </dependency>
  <dependency> <!-- using openwebbeans arthur knight, graalvm will use the scanner
service from this module -->
    <groupId>org.apache.openwebbeans</groupId>
    <artifactId>openwebbeans-se</artifactId>
    <version>2.0.22</version>
  </dependency>
  <dependency>
    <groupId>commons-cli</groupId>
    <artifactId>commons-cli</artifactId>
    <version>1.4</version>
  </dependency>
</dependencies>
```

Last step is to disable log4j2 and tomcat scanning by default - indeed previous setup works if passed on the command line but since it is always the same settings it is saner to put them in a `meecrowave.properties` in the classpath:

```
tomcat-scanning = false
logging-global-setup = false
log4j2-jul-bridge = false
```

using a profile or a binary dedicated module you can keep the JVM mode using Log4j2 and the native mode using Yupiik Logging (just tweak dependencies and optionally use arthur exclude configuration).

an Arthur knight can be developed to replace all that configuration, it will auto-setup meecrowave/cxf/tomcat needed reflection, scan present tomcat and cxf bundles, auto register CXF SPI (bus-extensions.txt - optionally filtering them and the not loadable ones) classes for reflection, spec classes (`org.apache.cxf.jaxrs.utils.InjectionUtils.STANDARD_CONTEXT_CLASSES`), and likely inherit from openwebbeans extension CDI integration. It means that once done, using meecrowave can be as simple as `<graalExtension>meecrowave</graalExtension>`. If you think it is worth the effort, don't hesitate to do a pull request on Arthur or send a mail on dev@openwebbeans.apache.org.

With this setup you should get a `target/*.graal.bin` binary executable containing your application and meecrowave, just launch it to start your application and use it as a standard Meecrowave CLI!

until Apache OpenWebBeans 2.0.22, annotated mode can miss some beans, ensure to use 2.0.22 or more if you don't explicitly add a beans.xml.

you can need to adjust a few classes depending what you use of CXF. Previous setup will be sufficient for a module containing:

+

```
// test with:
// $ curl http://localhost:8080/hello?name=foo -H 'accept: application/json'
@Path("hello")
@ApplicationScoped
public class MyEndpoint {
    @GET
    @Produces(MediaType.APPLICATION_JSON)
    public Message get(@QueryParam("name") final String name) {
        return new Message("Hello " + name + "!");
    }

    @Data
    public static class Message {
        @JsonbProperty // mark at least one property to let arthur find it, see
extension.annotation.custom.annotations.properties
        private String message;
    }
}
```