



Apache Karaf

The modulith runtime

APACHECON @HOME
Spt, 29th – Oct. 1st

2020

Who am I ?

JB Onofré <jbonofre@apache.org>

- Software engineer/Fellow at Talend
- ASF Member
- PMC member/commmitter for ~ 20

Apache projects (Karaf, Camel,
ActiveMQ, Felix, Aries, Beam,
Incubator, ...)



Apache Karaf runtime

- Lightweight and full features runtime
- Cloud/container ready
- Addressing bunch of use cases:
 - backend/frontend applications
 - IoT, messaging and integration
 - Micro services

Application Server or Micro Services ?

Application Server

Pro:

- Single middleware/infra
- Easy to test
- Easy to deploy

Cons:

- Limited scalability (development and runtime)
- Difficult update
- Memory footprint

Micro Service

Pro:

- High flexibility and granularity
- Rollup updates
- Highly scalable

Cons:

- Bunch of containers (cost ?), infra management
- Not easy to test and collaborated development
- Memory footprint (overall)

Karaf Modulith Runtime

- Not application server !
- Clever micro services (consolidated) grouped by unit
- Don't change dev model (development is the same, gives the opportunity for devops to optimize infra)

Multi purpose ?

- Apache Karaf is THE modolith runtime
- Unique runtime supporting several programming models/frameworks
- Any application module can be used by another application module in the same Karaf runtime thanks to bean/service registry
- Karaf provides turnkey and common features which can be used by application module: logging, monitoring (Decanter/Prometheus), ...

Web Application

- Karaf HTTP service supported patterns: registration, annotations, whiteboard, war, ...
- Can interact with other application modules (services, CDI, Spring Boot, ...)
- Support advanced runtime features (security, proxy, load balancing, ...)

Web Application - Annotation whiteboard

```
@WebServlet(name = "MyServlet", urlPatterns = "/my")
public class ExampleServlet extends HttpServlet {

    @Override
    public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        service(request, response);
    }

    @Override
    public void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        service(request, response);
    }

    @Override
    public void service(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException { ... }

}
```


Web Application - Class whiteboard

```
@Component(  
    property = { "alias=/servlet-example", "servlet-name=Example"}  
)  
public class ExampleServlet extends HttpServlet implements Servlet {  
  
    @Override  
    public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException,  
IOException {  
        ...  
    }  
  
}
```

Web Application - Registration

```
httpService.registerServlet("/servlet-example", new ExampleServlet(), null, null);
```

Web Application - WAR/WebBundle

```
$ bundle:install
```

```
webbundle:mvn:org.apache.karaf.examples/karaf-war-example-webapp/${project.version}/war?Web-ContextPath=example
```

Web Application - Proxy

```
# list available load balancing policy
```

```
karaf@root(> http:proxy-balancing-list
```

```
round-robin
```

```
random
```

```
# add a proxy to a local resource
```

```
karaf@root(> http:proxy-add /acna /system/console
```

```
# add a proxy to a remote resource
```

```
karaf@root(> http:proxy-add /maven https://repo1.maven.org/maven2/
```

```
# list the active proxies
```

```
karaf@root(> http:proxy-list
```

```
URL | ProxyTo | Balancing Policy
```

/acna	/system/console	
/maven	https://repo1.maven.org/maven2/	

JAXRS REST API

- Create/expose your APIs (Swagger/OpenAPI, ...)
- Karaf JAXRS service supported patterns: registration, annotations, whiteboard

JAXRS REST API - CXF Registration

```
@Path("/")
public class BookingServiceRest {

    @Path("/")
    @Produces("application/json")
    @GET
    public Collection<Booking> list() {
        ...
    }

    ...
}
```

```
@Component
public class RestService {
    private Server server;

    @Activate
    public void activate() throws Exception {
        JAXRSServerFactoryBean bean = new
        JAXRSServerFactoryBean();
        bean.setAddress("/booking");
        bean.setBus(BusFactory.getDefaultBus());
        bean.setProvider(new JacksonJsonProvider());
        bean.setServiceBean(new BookingServiceRest());
        server = bean.create();
    }

    @Deactivate
    public void deactivate() throws Exception {
        if (server != null) {
            server.destroy();
        }
    }
}
```

JAXRS REST API - Whiteboard

```
@Path("/booking")
@Component(service = BookingServiceRest.class, property = { "osgi.jaxrs.resource=true" })
public class BookingServiceRest {

    @Override
    @Path("/")
    @Produces("application/json")
    @GET
    public Collection<Booking> list() {
        return bookings.values();
    }

    ....
}
```

IoT and Integration - Apache Camel

- Apache Camel Karaf to run camel routes in Karaf
- Support Camel Spring, Blueprint, Java DSL

```
camelContext.start();
camelContext.addRoutes(new RouteBuilder() {
    @Override
    public void configure() throws
Exception {
        from("direct:foo")
            .id("foo")
            ...
    }
});
```


IoT and Integration - Apache Camel

- New coming feature ! karamel
- Wrapper/tooling to easily run/package Camel routes within Karaf

```
# run Camel routes
```

```
karamel run firstroute.java secondroute.xml thirddroute.jar
```

```
# create distribution archive packaging karaf, camel, routes (detecting required camel components, ...)
```

```
karamel package firstroute.java secondroute.xml thirddroute.jar
```

```
# create a docker image based on karaf one, packaging karaf, camel, routes
```

```
karamel docker firstroute.java secondroute.xml thirddroute.jar
```

```
# deploy and run on kubernetes packaging karaf, camel, routes
```

```
karamel deploy firstroute.java secondroute.xml thirddroute.jar
```

OSGi

- Karaf is internally powered on OSGi (you can use Karaf without knowing OSGi !)
- If you know OSGi, you can deploy your bundles
- Support any OSGi programming model: OSGi native, blueprint, SCR
- Shared service registry usable via all programming models supported by Karaf (OSGi, CDI, Spring Boot, ...)

OSGi - SCR

@Component

```
public class MyComponent {
```

```
    @Reference
```

```
    private BookingService bookingService;
```

```
    @Activate
```

```
    public void start() throws Exception {
```

```
        bookingService...
```

```
    }
```

```
}
```

CDI

- Decoupled from the CDI container (OWB, Weld, ...)
- Annotations to explicitly register/use service
- Several CDI modules in the same Karaf to use beans from one to others

CDI

```
@Service
@ApplicationScoped
public class MyServiceImpl implements
MyService {

    @Inject
    private InnerService innerService;

    @Override
    public String myMessage() {
        return "My " +
innerService.getMessage();
    }

}
```

```
@ApplicationScoped
public class SimpleClient {

    @Inject
    @Reference
    private MyServiceImpl
myService;

    ...

}
```

Spring Boot

- New coming feature !
- Karaf Spring Boot service allows to manage Spring Boot applications (folder or fat jar) and lifecycle
- No change in the spring boot module: no special plugin, no special MANIFEST, just the regular spring boot artifact.
- “Override” some Spring Boot beans to use Karaf services (logging, http, ...)
- Implicit bean registration in the Karaf service registry

Spring Boot - Packages

- Can be managed by running Karaf

instance

```
# install spring boot app module in karaf (several supported kind of sources)
```

```
karaf@root(>) spring-boot:install file:/path/to/first.jar
```

```
karaf@root(>) spring-boot:install file:/path/to/folder
```

```
karaf@root(>) spring-boot:install http://.../second.jar
```

```
karaf@root(>) spring-boot:install mvn:groupId/artifactId/version
```

```
# list the registered spring boot app module
```

```
karaf@root(>) spring-boot:list
```

Name	State
RestServiceApplication	false

```
# start a spring boot app module
```

```
karaf@root(>) spring-boot:start RestServiceApplication
```

```
:: Spring Boot :: (v2.3.3.RELEASE)
```

```
Spring Boot app RestServiceApplication started
```

```
karaf@root(>) log:display
```

```
08:13:07.887 INFO [pipe-spring-boot:start RestServiceApplication] Started
```

```
RestServiceApplication in 1.591 seconds (JVM running for 132.215)
```

```
# stop a spring boot app module
```

```
karaf@root(>) spring-boot:stop RestServiceApplication
```

- Can be packaged “all together”

```
# CLI
```

```
$ karaf-spring-boot package --name foo first.jar second.jar /path/to/folder
```

```
# create foo.tar.gz and zip, ready to run
```

```
$ karaf-spring-boot docker --name foo/1.0.0 first.jar second.jar
```

```
# create docker image, ready to run
```

```
$ docker run foo/1.0.0
```

```
# Maven plugin
```

```
$ mvn karaf:spring-boot-package
```

```
$ mvn karaf:spring-boot-docker
```

```
# Gradle plugin
```

```
$ gradle
```

Spring Boot - Stack

- Stack allows you to override or extend the spring boot module classloader
- Basically a stack is a folder containing jar files, corresponding classloader is added (parent first) to the spring boot app classloader
- Support several stacks to have a hierarchy of classloaders

Spring Boot - Stack

```
# adding stack
karaf@root(> spring-boot:stack-add
/path/to/stack1
# adding stack to spring boot app module
karaf@root(> spring-boot:install ... --stack
stack1
```

```
$ karaf-spring-boot package --name foo --stack
/path/to/stack1 --stack /path/to/stack2 ...
```

Karaf Packages

Dynamic (run and deploy)

```
$ ./build.sh ... --image-name my-karaf
Sending build context to Docker daemon
21.29MB
...
Successfully built d209a00ef33c
Successfully tagged my-karaf:latest
$ docker images|grep my-karaf
my-karaf
latest                d209a00ef33c        About
a minute ago         122MB
...
ssh -p 8101 karaf@host
karaf@root()> spring-boot:install
mvn:/.../spring-boot.jar
karaf@root()> spring-boot:start my-app
```

Static / Winegrower (build and run)

```
$ java -jar ...
$ mvn winegrower:run
$ docker run --name mykaraf -d my-dist
```

Karaf on Kubernetes

The screenshot shows the Amazon EKS console for a cluster named 'karaf'. The 'General configuration' section displays the following details:

- Kubernetes Version:** 1.13
- Platform Version:** eks.4
- Status:** ACTIVE
- API server endpoint:** <https://B4B30303C578D5F476D4BB2D0524CB52.yl4.eu-west-1.eks.amazonaws.com>
- Certificate authority:** LS0tLS1CRUdJTiBDRVJUSUZJQ0FUR50tLS0tck1JSUN5RENDQWJD20F3SUJBZ0lCQURBTklna3Foa2lHOXcwQkFRc0ZBREFTVjNld0VRWURWUWVFRXdwcmRX5mwKY201bGRHVnpNQjRYFRFNU1Ea3dOakE0TXpVME1sb1hEVEk1TURd016Q
- OpenID Connect provider URL:** <https://oid.eks.eu-west-1.amazonaws.com/id/B4B30303C578D5F476D4BB2D0524CB52>
- Cluster ARN:** arn:aws:eks:eu-west-1:295331841498:cluster/karaf
- Role ARN:** arn:aws:iam::295331841498:role/karaf-eks

The 'Networking' section shows:

- VPC:** [vpc-26caea40](#)
- Subnets:** [subnet-6a3de230](#), [subnet-48cda000](#), [subnet-848815e2](#)
- Security groups:** [sg-2292b25d](#)
- API server endpoint access:** Private access Disabled, Public access Enabled

```
$ kubectl expose deployment/karaf  
--type="NodePort" --port=8181  
service/karaf exposed
```

```
$ kubectl scale deployments/karaf  
--replicas=2
```

NAME	READY	UP-TO-DATE	AVAILABLE
karaf	2/2	2	2
AGE			4m34s

Q&A

<http://karaf.apache.org>

user@karaf.apache.org

dev@karaf.apache.org

#karaf on the-asf.slack.com

