ar:kalix

Base Library for the Arrowhead Framework

By Emanuel Palm

What is it?

1. Java 11 libraries for Arrowhead system development.
2. A website, arkalix.se, with documentation and examples.
3. Maven artifacts, accessible via Maven Central.

What does it look like? (1/2)

```
var system = new ArSystem.Builder()
    .identity(identity)
    .trustStore(trustStore)
    .build();

system.provide(new HttpService()
    .name("kalix-example-service")
    .encodings(JSON)
    .accessPolicy(token())
    .basePath("/example")

    .get("/greeting", (request, response) -> {
        response.status(OK)
            .body("{\"text\":\"Hello, Arrowhead!\"}");
        return done();
     }));
```

What does it look like? (2/2)

```
var system = new ArSystem.Builder()
    .identity(identity)
    .trustStore(trustStore)
    .plugins(HttpJsonCoreIntegrator.viaServiceRegistryAt(hostnamePort))
    .build();

system.consume()
    .name("temperature")
    .using(HttpConsumer.factory())
    .flatMap(consumer -> consumer.send(new HttpConsumerRequest()
        .method(GET)
        .uri("/temperatures/a-32"))
    .flatMap(response -> response.bodyAsIfSuccess(TemperatureDto.class))
    .ifSuccess(temperature -> System.out.println(temperature.toCelsius()))
    .onFailure(Throwable::printStackTrace);
```

What's happening now?

1. The library is being used to implement the **Contract Proxy**, the **Plant Description Engine**, and, potentially, the **Cloud Uploader**.
2. More service integrators, bugfixes and other improvements are likely coming. An **Event Handler** integrator and a **Contract Proxy** integrator are being planned.
3. Fork the repository on github.com/emanuelpalm/arrowhead-kalix.

Who am I?

Emanuel Palm
PhD. Student
Luleå University of Technology
emanuel.palm@ltu.se

Design philosophy

1. Correctness

   Validate as much user input as possible, such as certificates and configuration data, and fail as soon as possible.

2. Convenience

   Never require users to explicitly specify details that can be correctly derived from context. Prioritize abstractions that minimize cognitive overhead.

3. Performance

   Optimize for short start-up time, high throughput and low latency. When possible, pay for performance at compile-time rather than at runtime.