INTRODUCTION INTO DROPWIZARD USING MONGODB

Alexander Reelsen
alexander@reelsen.net
@spinscale

AGENDA

- No agenda, dive into code samples...

ABOUT ME - ALEXANDER REELSEN

- Studied information systems
- 10 years linux system engineering, converted to software engineering
- Web framework enthusiast, fed up with complex java environment for simple webapps
- Other interests: Scaling web architectures, Web 2.0 (nosql, search)
- Author of Play framework cookbook
- Working at Lusini GmbH, building a b2b ecommerce platform
- Streetball/Basketball

WHY DROPWIZARD?

- Library/Framework for creating JSON web services in short time
- Simple to understand & debug
- Developed with operations in mind (monitoring)
- Good documentation
- Jetty, Jersey, Jackson, Metrics, Guava, Hibernate validator

WHY MONGO-JACKSON-WRAPPER?

- Check out mongo-jackson-mapper - very simple
- Morphia is another good choice due to DAO support
- Very simple to put objects in MongoDB
- Setting up MongoDB is not detailed here.
  Download. Install. Run.

SETTING UP POM.XML

```xml
    <dependencies>
        <dependency>
            <groupId>com.yammer.dropwizard</groupId>
            <artifactId>dropwizard-core</artifactId>
            <version>0.3.1</version>
            <scope>compile</scope>
        </dependency>
        <dependency>
            <groupId>net.vz.mongodb.jackson</groupId>
            <artifactId>mongo-jackson-mapper</artifactId>
            <version>1.4.1</version>
        </dependency>
    </dependencies>

<!-- Dont forget the shade plugin part from the documentation in your pom.xml! -->
```

See also: Building FAT jars

DEFINE MODELS

- Product, which has variants
- Product is a container which contains physical products
- Product = Cool T-Shirt
- Variant 1: blue, XL
- Variant 2: red, M

PRODUCT ENTITY

```
package model;

import java.util.List;
import javax.validation.Valid;
import net.vz.mongodb.jackson.Id;
import net.vz.mongodb.jackson.ObjectId;

public class Product {

    @Id @ObjectId
    public String id;

    @Valid
    public ProductData data;

    @Valid
    public List variants;

}
```

PRODUCTDATA

```java
package model;

import java.util.Map;
import org.hibernate.validator.constraints.NotEmpty;
import com.google.common.collect.Maps;

public class ProductData {

    // Acts as an unique identifier
    @NotEmpty
    public String lucid;
    @NotEmpty
    public String sku;
    @NotEmpty
    public String name;

    public Double price;

    public Map attributes = Maps.newHashMap();
}
```

SPECIALTIES ON ENTITIES

- Getters and setters omitted only for readibility
- `@Id` and `@ObjectId` annotations are **not** from the mongodb driver
- `@NotEmpty` is from hibernate validator (JSR 303 : Bean validation)
- ProductData has no `@Id` anntotation as it is intended to be used embedded only

CONFIGURATION VIA YAML

- MongoDB connection parameters
- Put it into `src/main/resources/config.yaml`
- Think about default values!

```
mongohost: localhost
mongoport: 27017
mongodb: mydb
```

CONFIGURATION JAVA CLASS

```java
package configuration;

import javax.validation.constraints.Max;
import javax.validation.constraints.Min;

import org.codehaus.jackson.annotate.JsonProperty;
import org.hibernate.validator.constraints.NotEmpty;
import com.yammer.dropwizard.config.Configuration;

public class ProductConfiguration extends Configuration {

    @JsonProperty @NotEmpty
    public String mongohost = "localhost";

    @Min(1)
    @Max(65535)
    @JsonProperty
    public int mongoport = 27017;

    @JsonProperty @NotEmpty
    public String mongodb = "yourdb";

}
```

CREATING A SERVICE

```java
package service;
// ...imports omitted...

public class ProductService extends Service {

    public static void main(String[] args) throws Exception {
        new ProductService().run(new String[]{"server", "src/main/resources/config.yaml"});
    }

    private ProductService() {
        super("app");
    }

    @Override
    protected void initialize(ProductConfiguration configuration, Environment environment) throws Excep
tion {
        Mongo mongo = new Mongo(configuration.mongohost, configuration.mongoport);
        DB db = mongo.getDB(configuration.mongodb);
        JacksonDBCollection<Product, String> products =
            JacksonDBCollection.wrap(db.getCollection("products"), Product.class, String.class);

        MongoManaged mongoManaged = new MongoManaged(mongo);
        environment.manage(mongoManaged);

        environment.addHealthCheck(new MongoHealthCheck(mongo));

        environment.addResource(new CreateProductResource(products));
        environment.addResource(new ProductResource(products));
        environment.addResource(new RemoveProductVariantResource(products));
    }
}
```

CREATING A SERVICE

- `initialize()` is run on startup by dropwizard
- A new `Mongo` and `DB` object is created from the configuration
- Mongo collection is wrapped into a mongo jackson collection
- These wrapped objects are handed over to resources
- A health check is added
- The `main()` method makes it easy to start via IDE

MANAGED MONGO INSTANCE

- Managed instances are simple interfaces with a `start()` and `stop()` method
- Allows to manage resources on application start and stop
- `start()` was not implemented as the resources need an already initialized connection

MANAGED MONGO INSTANCE

```java
package service;

import com.mongodb.Mongo;
import com.yammer.dropwizard.lifecycle.Managed;

public class MongoManaged implements Managed {

    private Mongo m;

    public MongoManaged(Mongo m) {
        this.m = m;
    }

    public void start() throws Exception {
    }

    public void stop() throws Exception {
        m.close();
    }
}
```

MONGO HEALTH CHECK

- Health checks are a wonderful operations features
- Allows to check for health inside of your application
- This sample checks for an existing mongo db connection (from your application, not from your icinga)

MONGO HEALTH CHECK

```java
package health;

import com.mongodb.Mongo;
import com.yammer.metrics.core.HealthCheck;

public class MongoHealthCheck extends HealthCheck {

    private Mongo mongo;

    public MongoHealthCheck(Mongo mongo) {
        super("MongoHealthCheck");
        this.mongo = mongo;
    }

    @Override
    protected Result check() throws Exception {
        mongo.getDatabaseNames();
        return Result.healthy();
    }

}
```

HELPING CODE

- Request lifecycle: Get a request with an ID
- Request lifecycle: Search for id in persistence store
- Request lifecycle: Execute action on id
- Request lifecycle: Return 404 if entity with id not found

- Having helpers for this is nice
- Playframework has: `notFoundIfNull(Object)`

HELPERS

```java
package utils;

import javax.ws.rs.WebApplicationException;
import javax.ws.rs.core.Response.Status;

import net.vz.mongodb.jackson.DBCursor;

public class ResourceHelper {

    public static void notFoundIfNull(DBCursor<?> cursor) {
        if (!cursor.hasNext()) {
            throw new WebApplicationException(Status.NOT_FOUND);
        }
    }

    public static void notFoundIfNull(Object obj) {
        errorIfNull(obj, Status.NOT_FOUND);
    }

    public static void errorIfNull(Object obj, Status status) {
        if (obj == null) {
            throw new WebApplicationException(status);
        }
    }
}
```

RESOURCES

- Creating a product only works if it does not exist already
- Removing products is a bit hacky by checking if the count of removed products is one
- Removing product variants uses some google guava magic, might be possible via MongoDB as well

CREATE A PRODUCT

```java
package resources.product;
// ... imports omitted ...

@Path("/product/")
public class CreateProductResource {

    private JacksonDBCollection<Product, String> products;

    public CreateProductResource(JacksonDBCollection<Product, String> products) {
        this.products = products;
    }

    @PUT
    public Response createProduct(@Valid Product product) {
        DBCursor<Product> cursor = products.find().is("data.lucid", product.data.lucid);
        if (cursor.hasNext()) {
            return Response.status(Status.CONFLICT).build();
        }

        products.save(product);

        return Response.noContent().build();
    }
}
```

GETTING & DELETING A PRODUCT

```
package resources.product;
import static utils.ResourceHelper.*;
// ... other imports omitted ...

@Path("/product/{id}")
public class ProductResource {

    private JacksonDBCollection<Product, String> products;

    public ProductResource(JacksonDBCollection<Product, String> products) {
        this.products = products;
    }

    @GET
    public Product getProduct(@PathParam("id") String id) {
        DBCursor<Product> cursor = products.find().is("data.lucid", id);
        notFoundIfNull(cursor);

        return cursor.next();
    }

    @DELETE
    public Response removeProduct(@PathParam("id") String id) {
        DBObject query = new BasicDBObject("data.lucid", id);
        WriteResult<Product,String> result = products.remove(query);
        int affectedObjects = result.getWriteResult().getN();

        if (affectedObjects != 1) {
            return Response.serverError().build();
        }
        return Response.noContent().build();
    }
}
```

REMOVING PRODUCT VARIANTS

```java
@Path("/product/{id}/{variant}")
public class RemoveProductVariantResource {

    private JacksonDBCollection<Product, String> products;

    public RemoveProductVariantResource(JacksonDBCollection<Product, String> products) {
        this.products = products;
    }

    @DELETE
    public Response removeVariant(@PathParam("id") String id, @PathParam("variant") String variant) {
        DBCursor<Product> cursor = products.find().in("data.lucid", id);
        notFoundIfNull(cursor);

        Product product = cursor.next();

        int size = product.variants.size();
        product.variants = removeVariantByLucid(product, variant);
        if (product.variants.size() == size) {
            return Response.notModified().build();
        }

        products.save(product);

        return Response.noContent().build();
    }
```

REMOVING PRODUCT VARIANTS

```
    public List<ProductData> removeVariantByLucid(Product product, String lucid) {
        Iterable<ProductData> iterables = Iterables.filter(product.variants,
            not(new MatchesLucidPredicate(lucid)));
        return Lists.newArrayList(iterables);
    }

    private static final class MatchesLucidPredicate implements Predicate<ProductData> {
        private String lucid;

        public MatchesLucidPredicate(String lucid) {
            this.lucid = lucid;
        }
        public boolean apply(ProductData input) {
            return lucid.equals(input.lucid);
        }
    }
}
```

USING CURL FOR TESTING

```
    curl -X PUT localhost:8080/product/ -v --header "Content-Type: application/json" -d' {
    "data" : {
        "sku" : "sku123",
        "lucid": "lucid123",
        "name": "T-Shirt foo"
    },
    "variants" : [
        {
            "sku" : "sku1",
            "lucid": "lucid1",
            "name": "Great variant product",
            "price" : 20.0,
            "attributes" : { "color": "yellow", "material" : "wool", "size": "L" }
        },
        {
            "sku" : "sku2",
            "lucid": "lucid2",
            "name": "Great variant product 2",
            "price" : 30.0,
            "attributes" : { "color": "green", "material" : "wool", "size": "M" }
        }
    ]
}'
```

Returns 204

ISSUING OTHER CALLS

- Getting the product (and prettyprinting it via python)

```
curl -s localhost:8080/product/lucid123  --header "Content-Type: application/json" | python
-mjson.tool
```

- Deleting a variant

```
curl -X DELETE localhost:8080/product/lucid123/lucid2 -v
```

- Deleting a complete product

```
curl -X DELETE localhost:8080/product/lucid123 -v
```

CHECKING HEALTH

Go to http://localhost:8081/healthcheck

- Good case:

```
* MongoHealthCheck: OK
* deadlocks: OK
```

- Now stop MongoDB and retry

```
! MongoHealthCheck: ERROR
* deadlocks: OK
```

IMPROVEMENTS - CONFIGURATION

- Nested configuration

```
mongo:
  host: localhost
  port: 27017
  db: mydb
```

- Caching via `@CacheControl` annotation

```
@CacheControl(maxAge = 6, maxAgeUnit = TimeUnit.HOURS)
```

- Authentication is easy. OAuth and Basic Auth ootb

```
@GET
public SecretPlan getSecretPlan(@Auth User user) {
    return dao.findPlanForUser(user);
}
```

METRICS

- Annotate every resource method with `@Timed`, `@Metered` or `@ExceptionMetered`
- See the metrics page
- Thread dump available under /threads

MORE FEATURES

- Tasks: Single actions which can be triggered via HTTP
- Scala support
- Templating via FreeMarker
- Database integration via JDBI, a nice layer on top of JDBC
- Metrics, self written cool metrics libarary, can be checked via port 8081
- Implement own JSON de-/serializers
- Streaming is supported, implement `StreamingOutput`

OPEN QUESTIONS & WISHLIST

- Providing XML APIs, maybe via jackson databind and own `MessageBodyWriter` and `MessageBodyReader`
- Live code reloading (no state in the server makes this possible)
- I hate freemarker, it is just not web-dev friendly
- Using MongoDB completely with a managed instance, haven't figured that one out yet

THANKS FOR LISTENING!

QUESTIONS?

Slides available at
http://spinscale.github.com/

alexander@reelsen.net
@spinscale

DOCUMENTATION & CREDITS

- Dropwizard

- Presentation done with reveal.js
- Cool zooming done with zoom.js