HowToDoInJava

Dropwizard – BasicAuth Security Example

Last Updated: January 25, 2022 Dropwizard Dropwizard, Java Security

Using dropwizard, we have learned about creating REST APIs, writing client code and adding health-check filters. In this tutorial, we will learn to add **username/password based authentication** and **role based authorization** capabilities into REST APIs using **Basic Authentication**.

Table of Contents

Include Dropwizard Auth Module Maven Dependency
Add Custom Principal Object
Add Custom Authenticator
Add Custom Authorizer
Configure BasicCredentialAuthFilter
Secure REST APIs with @Auth Annotation
Test Dropwizard Basic Auth Code

Include Dropwizard Auth Module Maven Dependency

Authentication capabilities are added as separate module in dropwizard application.

```
<properties>
    <dropwizard.version>1.0.0</dropwizard.version>
</properties>
<dependency>
    <groupId>io.dropwizard</groupId>
    <artifactId>dropwizard-auth</artifactId>
    <version>${dropwizard.version}</version>
```

```
</dependency>
```

Add Custom Principal Object

In security, principal object represent the user for which credentials have been supplied with request. It implements **`java.security.Principal`** interface.

```
package com.howtodoinjava.rest.auth;

import java.security.Principal;
import java.util.Set;

public class User implements Principal {
    private final String name;

    private final Set<String> roles;

    public User(String name) {
        this.name = name;
        this.roles = null;
    }

    public User(String name, Set<String> roles) {
        this.name = name;
        this.roles = roles;
    }

    public String getName() {
        return name;
    }

    public int getId() {
        return (int) (Math.random() * 100);
    }

    public Set<String> getRoles() {
        return roles;
    }
}
```

Add Custom Authenticator

`Authenticator` class is responsible for verifying username/password credentials included in Basic Auth header. In enterprise application, you may fetch the user's password from database and if it matches then you set the user roles into principal object. In dropwizard, you will need to implements **`io.dropwizard.auth.Authenticator`** interface to put your application logic.

```
package com.howtodoinjava.rest.auth;

import io.dropwizard.auth.AuthenticationException;
import io.dropwizard.auth.Authenticator;
import io.dropwizard.auth.basic.BasicCredentials;

import java.util.Map;
import java.util.Optional;
import java.util.Set;

import com.google.common.collect.ImmutableMap;
import com.google.common.collect.ImmutableSet;

public class AppBasicAuthenticator implements Authenticator<BasicCredentials, User>
{
    private static final Map<String, Set<String>> VALID_USERS = ImmutableMap.of(
        "guest", ImmutableSet.of(),
        "user", ImmutableSet.of("USER"),
        "admin", ImmutableSet.of("ADMIN", "USER")
    );

    @Override
    public Optional<User> authenticate(BasicCredentials credentials) throws Authent
    {
        if (VALID_USERS.containsKey(credentials.getUsername()) && "password".equals
        {
            return Optional.of(new User(credentials.getUsername(), VALID_USERS.get(
        }
        return Optional.empty();
    }
}
```

Add Custom Authorizer

`Authorizer` class is responsible for matching the roles and decide whether user is allowed to perform certain action or not.

```
package com.howtodoinjava.rest.auth;

import io.dropwizard.auth.Authorizer;

public class AppAuthorizer implements Authorizer<User>
{
    @Override
    public boolean authorize(User user, String role) {
        return user.getRoles() != null && user.getRoles().contains(role);
    }
}
```

Configure BasicCredentialAuthFilter

Now let's register our custom classes into dropwizard security framework.

```
package com.howtodoinjava.rest;

import io.dropwizard.Application;
import io.dropwizard.Configuration;
import io.dropwizard.auth.AuthDynamicFeature;
import io.dropwizard.auth.AuthValueFactoryProvider;
import io.dropwizard.auth.basic.BasicCredentialAuthFilter;
import io.dropwizard.client.JerseyClientBuilder;
import io.dropwizard.setup.Bootstrap;
import io.dropwizard.setup.Environment;

import javax.ws.rs.client.Client;

import org.glassfish.jersey.server.filter.RolesAllowedDynamicFeature;

import com.howtodoinjava.rest.auth.AppAuthorizer;
import com.howtodoinjava.rest.auth.AppBasicAuthenticator;
import com.howtodoinjava.rest.auth.User;
import com.howtodoinjava.rest.controller.EmployeeRESTController;
```

```
import com.howtodoinjava.rest.controller.RESTClientController;
import com.howtodoinjava.rest.healthcheck.AppHealthCheck;
import com.howtodoinjava.rest.healthcheck.HealthCheckController;

public class App extends Application<Configuration> {

    @Override
    public void initialize(Bootstrap<Configuration> b) {
    }

    @Override
    public void run(Configuration c, Environment e) throws Exception {
        e.jersey().register(new EmployeeRESTController(e.getValidator()));

        final Client client = new JerseyClientBuilder(e).build("DemoRESTClient");
        e.jersey().register(new RESTClientController(client));

        // Application health check
        e.healthChecks().register("APIHealthCheck", new AppHealthCheck(client));

        // Run multiple health checks
        e.jersey().register(new HealthCheckController(e.healthChecks()));

        //****** Dropwizard security - custom classes ***********/
        e.jersey().register(new AuthDynamicFeature(new BasicCredentialAuthFilter.Bu
                                .setAuthenticator(new AppBasicAuthenticator())
                                .setAuthorizer(new AppAuthorizer())
                                .setRealm("BASIC-AUTH-REALM")
                                .buildAuthFilter()));
        e.jersey().register(RolesAllowedDynamicFeature.class);
        e.jersey().register(new AuthValueFactoryProvider.Binder<>(User.class));
    }

    public static void main(String[] args) throws Exception {
        new App().run(args);
    }
}
```

Secure REST APIs with @Auth Annotation

Adding **@Auth** annotation will trigger authentication filter on any API where it is put as parameter.

1) User must be authenticated. API is allowed to all users.

```
@PermitAll
@GET
public Response getEmployees(@Auth User user) {
    return Response.ok(EmployeeDB.getEmployees()).build();
}
```

2) User must be authenticated. API is allowed to all users with role “ADMIN” only.

```
@RolesAllowed({ "ADMIN" })
@GET
@Path("/{id}")
public Response getEmployeeById(@PathParam("id") Integer id, @Auth User user) {
    Employee employee = EmployeeDB.getEmployee(id);
    if (employee != null)
        return Response.ok(employee).build();
    else
        return Response.status(Status.NOT_FOUND).build();
}
```

In this way, you can add various authentication schemes in all your APIs as per your need.

Test Dropwizard Basic Auth Code

Let’s test out our secured APIs.

Call Any secured API

Basic Authentication Screen

http://localhost:8080/employees

Authenticated and allowed to all roles

http://localhost:8080/employees/1

Authenticated and allowed to ADMIN role only

Drop me your questions in comments section.

Happy Learning !!

Sourcecode Download

Was this post helpful?

Let us know if you liked the post. That's the only way we can improve.

Yes

No

Further Reading:

1. **Vaadin Spring Security with BasicAuth**

In this tutorial, we will learn to secure vaadin application behind basic authentication security provided by spring security module....

2. **Consume Dropwizard REST APIs with Jersey/HTTP Client**
   Dropwizard includes both Apache HttpClient and Jersey Client. Let's build REST client for consuming REST APIs across the network. ...

3. **Dropwizard Health Check Configuration Example**
   DropWizard health check is implemented by extending the HealthCheck class and returning Result.healthy() if everything is alright and Result.unhealthy() if...

Join 7000+ Awesome Developers

Get the latest updates from industry, awesome resources, blog updates and much more.

Email Address

Subscribe

* We do not spam !!

1 thought on “Dropwizard – BasicAuth Security Example”

Geraldo

May 7, 2020 at 10:32 am

Thanks, worked perfectly!

Reply

Leave a Comment

Name *

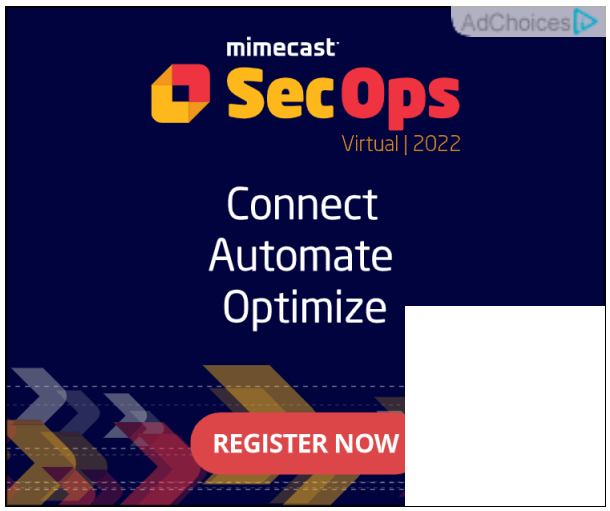
Email *

Website

Post Comment

Add me to your newsletter and keep me updated whenever you publish new blog posts

Search ...



HowToDoInJava

A blog about Java and related technologies, the best practices, algorithms, and interview questions.

Meta Links

- About Me
- Contact Us
- Privacy policy
- Advertise
- Guest Posts

Blogs

REST API Tutorial