(http://baeldung.com)

Uso de JWT con Spring Security OAuth

Última modificación: 17 de marzo de 2018

por Eugen Paraschiv (http://www.baeldung.com/author/eugen/) (http://www.baeldung.com/author/eugen/)

Seguridad (http://www.baeldung.com/category/security-2/)

Primavera (http://www.baeldung.com/category/spring/) +

OAuth (http://www.baeldung.com/tag/oauth/)

1. Información general

En este tutorial discutiremos cómo hacer que nuestra implementación Spring Spring OAuth2 haga uso de los tokens web JSON.

También continuamos construyendo sobre el artículo anterior (/spring-security-oauth2-refresh-token-angular-js) en esta serie de OAuth.

2. Configuración de Maven

Primero, necesitamos agregar la dependencia *spring-security-jwt* a nuestro *pom.xml* :

```
<dependency>
    <groupId>org.springframework.security</groupId>
    <artifactId>spring-security-jwt</artifactId>
</dependency>
```

Tenga en cuenta que debemos agregar la dependencia *spring-security-jwt* tanto al servidor de autorización como al servidor de recursos.

3. Servidor de autorización

A continuación, configuraremos nuestro servidor de autorización para usar *JwtTokenStore* , de la siguiente manera:

```
@Configuration
@EnableAuthorizationServer
public class OAuth2AuthorizationServerConfig extends AuthorizationServerCon
    @Override
    public void configure(AuthorizationServerEndpointsConfigurer endpoints)
        endpoints.tokenStore(tokenStore())
                 .accessTokenConverter(accessTokenConverter())
                 .authenticationManager(authenticationManager);
    }

    @Bean
    public TokenStore tokenStore() {
        return new JwtTokenStore(accessTokenConverter());
    }

    @Bean
    public JwtAccessTokenConverter accessTokenConverter() {
        JwtAccessTokenConverter converter = new JwtAccessTokenConverter();
        converter.setSigningKey("123");
        return converter;
    }

    @Bean
    @Primary
    public DefaultTokenServices tokenServices() {
        DefaultTokenServices defaultTokenServices = new DefaultTokenService
        defaultTokenServices.setTokenStore(tokenStore());
        defaultTokenServices.setSupportRefreshToken(true);
        return defaultTokenServices;
    }
}
```

Tenga en cuenta que usamos una **clave simétrica** en nuestro *JwtAccessTokenConverter* para firmar nuestros tokens, lo que significa que necesitaremos usar la misma clave exacta para el servidor de recursos también.

4. Servidor de recursos

Ahora, echemos un vistazo a nuestra configuración del servidor de recursos, que es muy similar a la configuración del servidor de autorización:

```
@Configuration
@EnableResourceServer
public class OAuth2ResourceServerConfig extends ResourceServerConfigurerAda
    @Override
    public void configure(ResourceServerSecurityConfigurer config) {
        config.tokenServices(tokenServices());
    }

    @Bean
    public TokenStore tokenStore() {
        return new JwtTokenStore(accessTokenConverter());
    }

    @Bean
    public JwtAccessTokenConverter accessTokenConverter() {
        JwtAccessTokenConverter converter = new JwtAccessTokenConverter();
        converter.setSigningKey("123");
        return converter;
    }

    @Bean
    @Primary
    public DefaultTokenServices tokenServices() {
        DefaultTokenServices defaultTokenServices = new DefaultTokenService
        defaultTokenServices.setTokenStore(tokenStore());
        return defaultTokenServices;
    }
}
```

Tenga en cuenta que estamos definiendo estos dos servidores como completamente independientes y de implementación independiente. Esa es la razón por la que tenemos que volver a declarar algunos de los mismos granos aquí, en la nueva configuración.

5. Reclamaciones personalizadas en el token

Vamos a configurar una infraestructura para poder agregar algunas **reclamaciones personalizadas en el token de acceso** . Los reclamos estándar proporcionados por el marco están bien, pero la mayoría de las veces necesitaremos información adicional en el token para utilizar en el lado del cliente.

Definiremos un *TokenEnhancer* para personalizar nuestro token de acceso con estos reclamos adicionales.

En el siguiente ejemplo, agregaremos una " *organización* " de campo adicional a nuestro token de acceso, con este *CustomTokenEnhancer* :

```
public class CustomTokenEnhancer implements TokenEnhancer {
    @Override
    public OAuth2AccessToken enhance(
     OAuth2AccessToken accessToken,
     OAuth2Authentication authentication) {
        Map<String, Object> additionalInfo = new HashMap<>();
        additionalInfo.put("organization", authentication.getName() + rando
        ((DefaultOAuth2AccessToken) accessToken).setAdditionalInformation(a
        return accessToken;
    }
}
```

Luego, lo conectaremos a nuestra configuración de **servidor de autorización** , de la siguiente manera:

```
@Override
public void configure(AuthorizationServerEndpointsConfigurer endpoints) thr
    TokenEnhancerChain tokenEnhancerChain = new TokenEnhancerChain();
    tokenEnhancerChain.setTokenEnhancers(
      Arrays.asList(tokenEnhancer(), accessTokenConverter()));

    endpoints.tokenStore(tokenStore())
             .tokenEnhancer(tokenEnhancerChain)
             .authenticationManager(authenticationManager);
}

@Bean
public TokenEnhancer tokenEnhancer() {
    return new CustomTokenEnhancer();
}
```

Con esta nueva configuración en funcionamiento, he aquí cómo se vería una token de token de token:

```
{
    "user_name": "john",
    "scope": [
        "foo",
        "read",
        "write"
    ],
    "organization": "johnIiCh",
    "exp": 1458126622,
    "authorities": [
        "ROLE_USER"
    ],
    "jti": "e0ad1ef3-a8a5-4eef-998d-00b26bc2c53f",
    "client_id": "fooClientIdPassword"
}
```

5.1. Use el token de acceso en el cliente JS

Finalmente, querremos utilizar la información del token en nuestra aplicación cliente AngualrJS. Usaremos la biblioteca angular-jwt (https://github.com/auth0/angular-jwt) para eso.

Entonces, lo que vamos a hacer es utilizar el reclamo de " *organización* " en nuestro *index.html* :

```
<p class="navbar-text navbar-right">{{organization}}</p>

<script type="text/javascript"
  src="https://cdn.rawgit.com/auth0/angular-jwt/master/dist/angular-jwt.js
</script>

<script>
var app = angular.module('myApp', ["ngResource","ngRoute", "ngCookies", "an

app.controller('mainCtrl', function($scope, $cookies, jwtHelper,...) {
    $scope.organiztion = "";

    function getOrganization(){
        var token = $cookies.get("access_token");
        var payload = jwtHelper.decodeToken(token);
        $scope.organization = payload.organization;
    }
    ...
});
```

6. Acceso a reclamos adicionales en el servidor de recursos

Pero, ¿cómo podemos acceder a esa información en el lado del servidor de recursos?

Lo que haremos aquí es: extraer los reclamos adicionales del token de acceso:

```
public Map<String, Object> getExtraInfo(OAuth2Authentication auth) {
    OAuth2AuthenticationDetails details
      = (OAuth2AuthenticationDetails) auth.getDetails();
    OAuth2AccessToken accessToken = tokenStore
      .readAccessToken(details.getTokenValue());
    return accessToken.getAdditionalInformation();
}
```

En la siguiente sección, analizaremos cómo agregar esa información adicional a nuestros detalles de *autenticación* utilizando un *AccessTokenConverter* personalizado .

6.1. Custom *AccessTokenConverter*

Vamos a crear *CustomAccessTokenConverter* y establecer los detalles de autenticación con reclamos token de acceso:

```
@Component
public class CustomAccessTokenConverter extends DefaultAccessTokenConverter

    @Override
    public OAuth2Authentication extractAuthentication(Map<String, ?> claims
        OAuth2Authentication authentication
         = super.extractAuthentication(claims);
        authentication.setDetails(claims);
        return authentication;
    }
}
```

Nota: *DefaultAccessTokenConverter* utilizado para establecer los detalles de Autenticación en Nulo.

6.2. Configurar *JwtTokenStore*

A continuación, configuraremos nuestro *JwtTokenStore* para usar nuestro *CustomAccessTokenConverter* :

```
@Configuration
@EnableResourceServer
public class OAuth2ResourceServerConfigJwt
 extends ResourceServerConfigurerAdapter {

    @Autowired
    private CustomAccessTokenConverter customAccessTokenConverter;

    @Bean
    public TokenStore tokenStore() {
        return new JwtTokenStore(accessTokenConverter());
    }

    @Bean
    public JwtAccessTokenConverter accessTokenConverter() {
        JwtAccessTokenConverter converter = new JwtAccessTokenConverter();
        converter.setAccessTokenConverter(customAccessTokenConverter);
    }
    // ...
}
```

6.3. Reclamaciones adicionales disponibles en el objeto de autenticación

Ahora que el servidor de autorización agregó algunas reclamaciones adicionales en el token, ahora podemos acceder en el lado del servidor de recursos, directamente en el objeto de autenticación:

```
public Map<String, Object> getExtraInfo(Authentication auth) {
    OAuth2AuthenticationDetails oauthDetails
      = (OAuth2AuthenticationDetails) auth.getDetails();
    return (Map<String, Object>) oauthDetails
      .getDecodedDetails();
}
```

6.4. Prueba de detalles de autenticación

Asegurémonos de que nuestro objeto Autenticación contenga esa información adicional:

```
@RunWith(SpringRunner.class)
@SpringBootTest(
  classes = ResourceServerApplication.class,
  webEnvironment = WebEnvironment.RANDOM_PORT)
public class AuthenticationClaimsIntegrationTest {

    @Autowired
    private JwtTokenStore tokenStore;

    @Test
    public void whenTokenDoesNotContainIssuer_thenSuccess() {
        String tokenValue = obtainAccessToken("fooClientIdPassword", "john"
        OAuth2Authentication auth = tokenStore.readAuthentication(tokenValu
        Map<String, Object> details = (Map<String, Object>) auth.getDetails

        assertTrue(details.containsKey("organization"));
    }

    private String obtainAccessToken(
      String clientId, String username, String password) {

        Map<String, String> params = new HashMap<>();
        params.put("grant_type", "password");
        params.put("client_id", clientId);
        params.put("username", username);
        params.put("password", password);
        Response response = RestAssured.given()
          .auth().preemptive().basic(clientId, "secret")
          .and().with().params(params).when()
          .post("http://localhost:8081/spring-security-oauth-server/oauth/t
        return response.jsonPath().getString("access_token");
    }
}
```

Nota: obtuvimos el token de acceso con reclamos adicionales del servidor de autorización, luego leemos el objeto de *autenticación* desde el mismo que contiene información extra "organización" en el objeto de detalles.

7. KeyPair asimétrico

En nuestra configuración anterior usamos claves simétricas para firmar nuestro token:

```
@Bean
public JwtAccessTokenConverter accessTokenConverter() {
    JwtAccessTokenConverter converter = new JwtAccessTokenConverter();
    converter.setSigningKey("123");
    return converter;
}
```

También podemos usar claves asimétricas (claves públicas y privadas) para realizar el proceso de firma.

7.1. Generar JKS Java KeyStore File

Primero generemos las claves, y más específicamente un archivo *.jks* , usando la herramienta de línea de comandos *keytool* :

```
keytool -genkeypair -alias mytest
                    -keyalg RSA
                    -keypass mypass
                    -keystore mytest.jks
                    -storepass mypass
```

El comando generará un archivo llamado *mytest.jks* que contiene nuestras claves, las públicas y las privadas.

También asegúrese de que *keypass* y *storepass* son los mismos.

7.2. Exportar clave pública

A continuación, necesitamos exportar nuestra clave pública desde JKS generado, podemos usar el siguiente comando para hacerlo:

```
keytool -list -rfc --keystore mytest.jks | openssl x509 -inform pem -pubkey
```

Una respuesta de muestra se verá así:

```
-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAgIK2Wt4x2EtDl41C7vfp
OsMquZMyOyteO2RsVeMLF/hXIeYvicKr0SQzVkodHEBCMiGXQDz5prijTq3RHPy2
/5WJBCYq7yHgTLvspMy6sivXN7NdYE7I5pXo/KHk4nz+Fa6P3L8+L90E/3qwf6j3
DKWnAgJFRY8AbSYXt1d5ELiIG1/gEqzC0fZmNhhfrBtxwWXrlpUDT0Kfvf0QVmPR
xxCLXT+tEe1seWGEqeOLL5vXRLqmzZcBe1RZ9kQQm43+a9Qn5icSRnDfTAesQ3Cr
lAWJKl2kcWU1HwJqw+dZRSZ1X4kEXNMyzPdPBbGmU6MHdhpywI7SKZT7mX4BDnUK
eQIDAQAB
-----END PUBLIC KEY-----
-----BEGIN CERTIFICATE-----
MIIDCzCCAfOgAwIBAgIEGtZIUzANBgkqhkiG9w0BAQsFADA2MQswCQYDVQQGEwJ1
czELMAkGA1UECBMCY2ExCzAJBgNVBAcTAmxhMQ0wCwYDVQQDEwR0ZXN0MB4XDTE2
MDMxNTA4MTAzMFoXDTE2MDYxMzA4MTAzMFowNjELMAkGA1UEBhMCdXMxCzAJBgNV
BAgTAmNhMQswCQYDVQQHEwJsYTENMAsGA1UEAxMEdGVzdDCCASIwDQYJKoZIhvcN
AQEBBQADggEPADCCAQoCggEBAICCtlreMdhLQ5eNQu736TrDKrmTMjsrXjtkbFXj
Cxf4VyHmL4nCq9EkM1ZKHRxAQjIhl0A8+aa4o06t0Rz8tv+ViQQmKu8h4Ey77KTM
urIr1zezXWBOyOaV6Pyh5OJ8/hWuj9y/Pi/dBP96sH+o9wylpwICRUWPAG0mF7dX
eRC4iBtf4BKswtH2ZjYYX6wbccFl65aVA09Cn739EFZj0ccQi10/rRHtbHlhhKnj
iy+b10S6ps2XAXtUWfZEEJuN/mvUJ+YnEkZw30wHrENwq5QFiSpdpHFlNR8CasPn
WUUmdV+JBFzTMsz3TwWxplOjB3YacsCO0imU+5l+AQ51CnkCAwEAAaMhMB8wHQYD
VR0OBBYEFOGefUBGquEX9Ujak34PyRskHk+WMA0GCSqGSIb3DQEBCwUAA4IBAQB3
1eLfNeq45yO1cXNl0C1IQLknP2WXg89AHEbKkUOA1ZKTOizNYJIHW5MYJU/zScu0
yBobhTDe5hDTsATMa9sN5CPOaLJwzpWV/ZC6WyhAWTfljzZC6d2rL3QYrSIRxmsp
/J1Vq9WkesQdShnEGy7GgRgJn4A8CKecHSzqyzXulQ7Zah6GoEUD+vjb+BheP4aN
hiYY1OuXD+HsdKeQqS+7eM5U7WW6dz2Q8mtFJ5qAxjY75T0pPrHwZMlJUhUZ+Q2V
FfweJEaoNB9w9McPe1cAiE+oeejZ0jq0el3/dJsx3rlVqZN+lMhRJJeVHFyeb3XF
lLFCUGhA7hxn2xf3x1JW
-----END CERTIFICATE-----
```

Solo tomamos nuestra clave pública y la copiamos en nuestro **servidor de recursos** *src / main / resources / public.txt* :

```
-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAgIK2Wt4x2EtDl41C7vfp
OsMquZMyOyteO2RsVeMLF/hXIeYvicKr0SQzVkodHEBCMiGXQDz5prijTq3RHPy2
/5WJBCYq7yHgTLvspMy6sivXN7NdYE7I5pXo/KHk4nz+Fa6P3L8+L90E/3qwf6j3
DKWnAgJFRY8AbSYXt1d5ELiIG1/gEqzC0fZmNhhfrBtxwWXrlpUDT0Kfvf0QVmPR
xxCLXT+tEe1seWGEqeOLL5vXRLqmzZcBe1RZ9kQQm43+a9Qn5icSRnDfTAesQ3Cr
lAWJKl2kcWU1HwJqw+dZRSZ1X4kEXNMyzPdPBbGmU6MHdhpywI7SKZT7mX4BDnUK
eQIDAQAB
-----END PUBLIC KEY-----
```

Alternativamente, podemos exportar solo la clave pública agregando el argumento *-noout* :

```
keytool -list -rfc --keystore mytest.jks | openssl x509 -inform pem -pubkey
```

7.3. Configuración Maven

A continuación, no queremos que el archivo JKS sea recogido por el proceso de filtrado maven, por lo que nos aseguraremos de excluirlo en el *pom.xml* :

```
<build>
    <resources>
        <resource>
            <directory>src/main/resources</directory>
            <filtering>true</filtering>
            <excludes>
                <exclude>*.jks</exclude>
            </excludes>
        </resource>
    </resources>
</build>
```

Si utilizamos Spring Boot, debemos asegurarnos de que nuestro archivo JKS se agregue a classpath de la aplicación a través del complemento Spring Boot Maven - *addResources* :

```
<build>
    <plugins>
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
            <configuration>
                <addResources>true</addResources>
            </configuration>
        </plugin>
    </plugins>
</build>
```

7.4. Servidor de Autorización

Ahora, configuraremos *JwtAccessTokenConverter* para usar nuestro KeyPair de *mytest.jks* - de la siguiente manera:

```
@Bean
public JwtAccessTokenConverter accessTokenConverter() {
    JwtAccessTokenConverter converter = new JwtAccessTokenConverter();
    KeyStoreKeyFactory keyStoreKeyFactory =
      new KeyStoreKeyFactory(new ClassPathResource("mytest.jks"), "mypass".t
    converter.setKeyPair(keyStoreKeyFactory.getKeyPair("mytest"));
    return converter;
}
```

7.5. Servidor de recursos

Finalmente, debemos configurar nuestro servidor de recursos para usar la clave pública, de la siguiente manera:

```
@Bean
public JwtAccessTokenConverter accessTokenConverter() {
    JwtAccessTokenConverter converter = new JwtAccessTokenConverter();
    Resource resource = new ClassPathResource("public.txt");
    String publicKey = null;
    try {
        publicKey = IOUtils.toString(resource.getInputStream());
    } catch (final IOException e) {
        throw new RuntimeException(e);
    }
    converter.setVerifierKey(publicKey);
    return converter;
}
```

8. Conclusión

En este artículo rápido, nos centramos en configurar nuestro proyecto Spring Security OAuth2 para usar tokens web JSON.

La **implementación completa** de este tutorial se puede encontrar en el proyecto github (https://github.com/eugenp/spring-security-oauth) : este es un proyecto basado en Eclipse, por lo que debería ser fácil importarlo y ejecutarlo tal como está.

Suscribir ▾ ▲ el más nuevo ▲ más antiguo ▲ el más votado

Huésped

Sandro Augusto Oliveira

Hola,
gracias por el buen tutorial!
He descargado el código de Github pero no pude iniciar Auth Server, tengo la siguiente excepción:
Causado por: java.io.FileNotFoundException: class path resource [mytest.jks] no se puede abrir porque no existe .

¿Sabes lo que podría ser?

¡Tks por adelantado!
BR
Sandro

+ 0 − hace 1 año

Huésped

Eugen Paraschiv (http://www.baeldung.com/)

Oye, Sandro: sí, definitivamente puedo replicar el problema, investigarlo.
Saludos,
Eugen.

+ 0 − hace 1 año

Huésped

Sandro Augusto Oliveira

Hi Eugen,
Ok :). I've also faced an error on UI Password starting:
Exception in thread "main" java.lang.NoSuchMethodError: org.springframework.boot.builder.SpringApplicationBuilder.showBanner(Z)Lorg/springframework/boot/builder/SpringApplicationBuilder;
BR
Sandro

+ 0 − 1 year ago

Guest

Eugen Paraschiv (http://www.baeldung.com/)

Both should be fixed – you can now have a look. Hope that helps.
Cheers,
Eugen.

+ -1 − 1 year ago

Guest

DL

src/main/resources

true

*.jks

if you remove this .. file not found exception is resolved .. it states that do no read any .jks file hence compilers skips the file.

+ 0 −

1 year ago

Guest

wisely liu

IOUtils.toString(resource.getInputStream()) gives error method toString in class Object cannot be applied to given types;
required: no arguments
found: InputStream
reason: actual and formal argument lists differ in length

what happened?

+ 0 −

1 year ago

Guest

Dennis

Hi, probably no longer needed, but when going over this tutorial I came across the same problem. I solved it by using an older implementation of commons-io:

commons-io
commons-io
2.4

For some reason, the IOUtils is not found in 2.5, and I was given IOUtils from tomcat embedded as alternative.

+ 0 −

1 year ago

Guest

dziadeusz

Thanks Eugen, you saved my day just as always. I recommend the Spring Security Master Class to everyone as well.

+ 0 −

1 year ago

Guest

dziadeusz

How about using these by the way?
encrypt.key-store.location=
encrypt.key-store.password=
encrypt.key-store.alias=
encrypt.key-store.secret=

+ 0 −

1 year ago

Guest

Eugen Paraschiv (http://www.baeldung.com/)

Spring Cloud? Sure, that's a good option to handle configuration. There are a couple of writeups about Spring Cloud here on the site, and a few more coming.
Glad you're enjoying the site btw.
Cheers,
Eugen.

+ 0 −

1 year ago

Guest

VERGiL

Is there any easy was to access jwt token custom claims from a spring rest controller (resource server)?

+ 0 −

1 year ago

Guest

Eugen Paraschiv (http://www.baeldung.com/)

Hey Vergil,
While I haven't tried to do that explicitly, you should be able to do it, yes.
Keep in mind that you can inject the raw request in the controller, so you can certainly get to the token and implicitly the claims. Might be interesting to explore in an article though.
Cheers,
Eugen.

+ 0 −

1 year ago

Guest

Shubham Gupta

Hola, Eugen. Seguí tu tutorial y quería implementar Token JWT sobre los que ya salían de SpringBoot + SpringSecurity + Spring OAuth2 (http://pastebin.com/hufRRHMZ). Aquí está la respuesta normal que solía obtener: (http://pastebin.com/ibG99BbU). Intenté seguir los pasos pero luego mi servidor de autenticación responde con 404. Lo que hice: eliminé @EnableAuthorization de SecurityConfig1 y luego agregué el segundo archivo de configuración aquí es el enlace a mi segundo archivo de configuración (http: // pastebin. com / pPyYd3ud) y eliminé de SecurityConfig y luego con la misma solicitud obtengo 404. Actualización 2: Hola Finalmente, pude resolverlo. Proporcioné @EnableAuthorization en SecurityConfig2 y agregué. ... Leer más »

+ 0 −

hace 1 año

Cargar más comentarios

CATEGORÍAS

PRIMAVERA (HTTP://WWW.BAELDUNG.COM/CATEGORY/SPRING/)
DESCANSO (HTTP://WWW.BAELDUNG.COM/CATEGORY/REST/)
JAVA (HTTP://WWW.BAELDUNG.COM/CATEGORY/JAVA/)
SEGURIDAD (HTTP://WWW.BAELDUNG.COM/CATEGORY/SECURITY-2/)
PERSISTENCIA (HTTP://WWW.BAELDUNG.COM/CATEGORY/PERSISTENCE/)
JACKSON (HTTP://WWW.BAELDUNG.COM/CATEGORY/JACKSON/)
HTTPCLIENT (HTTP://WWW.BAELDUNG.COM/CATEGORY/HTTP/)
KOTLIN (HTTP://WWW.BAELDUNG.COM/CATEGORY/KOTLIN/)

SERIE

TUTORIAL "VOLVER A LO BÁSICO" DE JAVA (HTTP://WWW.BAELDUNG.COM/JAVA-TUTORIAL)
JACKSON JSON TUTORIAL (HTTP://WWW.BAELDUNG.COM/JACKSON)
TUTORIAL DE HTTPCLIENT 4 (HTTP://WWW.BAELDUNG.COM/HTTPCLIENT-GUIDE)
REST CON SPRING TUTORIAL (HTTP://WWW.BAELDUNG.COM/REST-WITH-SPRING-SERIES/)
TUTORIAL DE SPRING PERSISTENCE (HTTP://WWW.BAELDUNG.COM/PERSISTENCE-WITH-SPRING-SERIES/)
SEGURIDAD CON SPRING (HTTP://WWW.BAELDUNG.COM/SECURITY-SPRING)

ACERCA DE

ACERCA DE BAELDUNG (HTTP://WWW.BAELDUNG.COM/ABOUT/)
LOS CURSOS (HTTP://COURSES.BAELDUNG.COM)
TRABAJO DE CONSULTORÍA (HTTP://WWW.BAELDUNG.COM/CONSULTING)
META BAELDUNG (HTTP://META.BAELDUNG.COM/)
EL ARCHIVO COMPLETO (HTTP://WWW.BAELDUNG.COM/FULL_ARCHIVE)
ESCRIBIR PARA BAELDUNG (HTTP://WWW.BAELDUNG.COM/CONTRIBUTION-GUIDELINES)
CONTACTO (HTTP://WWW.BAELDUNG.COM/CONTACT)
INFORMACIÓN DE LA COMPAÑÍA (HTTP://WWW.BAELDUNG.COM/BAELDUNG-COMPANY-INFO)
TÉRMINOS DE SERVICIO (HTTP://WWW.BAELDUNG.COM/TERMS-OF-SERVICE)
POLÍTICA DE PRIVACIDAD (HTTP://WWW.BAELDUNG.COM/PRIVACY-POLICY)

EDITORES (HTTP://WWW.BAELDUNG.COM/EDITORS)

KIT DE MEDIOS (PDF) (HTTPS://S3.AMAZONAWS.COM/BAELDUNG.COM/BAELDUNG+-+MEDIA+KIT.PDF)