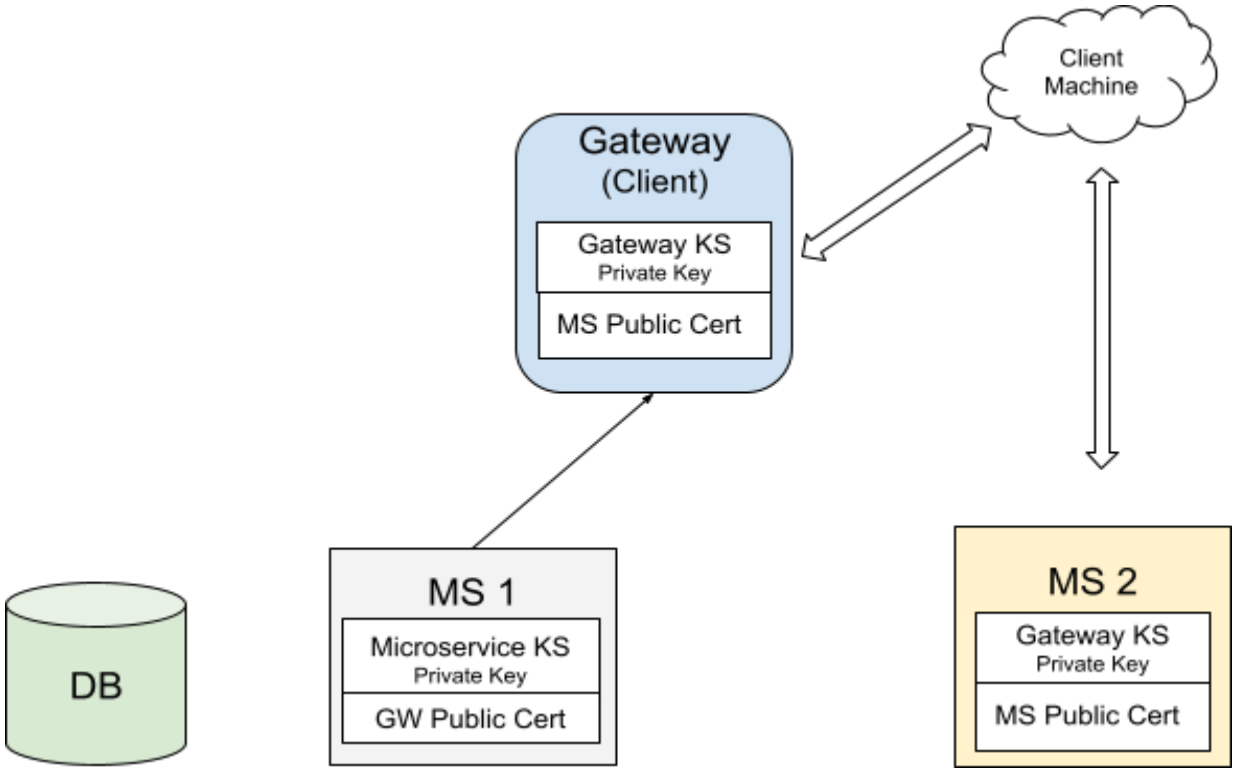
Spring Boot API Example

Per microservice principle, it is better to have a gateway application fronting all underlying microservices.

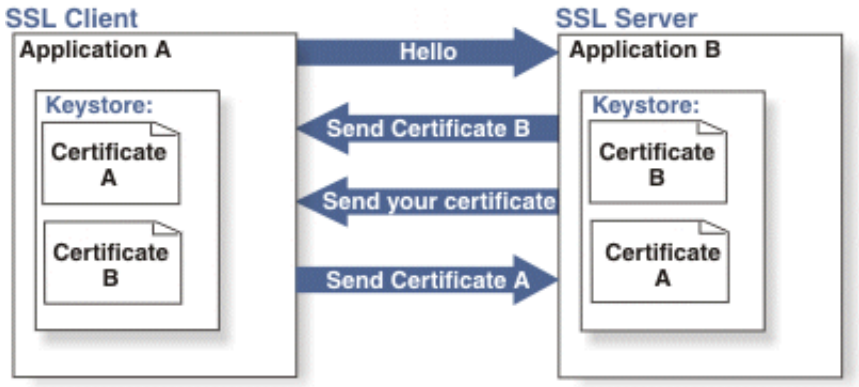


1-Way SSL

The server presents its certificate to the client and the client adds it to its list of trusted certificates. Then, the client can talk to the server.

2-Way SSL

2-way SSL is the same principle but both ways. i.e. both the client and the server have to establish trust between themselves using a trusted certificate.



Create A Self Signed Client Cert

```
keytool -genkeypair -alias my_api_gateway -keyalg RSA -keysize 2048 -storetype JKS -keystore
my_api_gateway.jks -validity 3650 -ext SAN=dns:localhost,ip:127.0.0.1
```

** SAN entries are required by Chrome and Safari.

Create Self Signed Server Cert:

```
keytool -genkeypair -alias my_api_services -keyalg RSA -keysize 2048 -storetype JKS
-keystore my_api_services.jks -validity 3650 -ext SAN=dns:localhost,ip:127.0.0.1
```

Create Public Cert file from Client Cert:

```
keytool -export -alias my_api_gateway -file my_api_gateway.crt -keystore my_api_gateway.jks
Enter keystore password:
Certificate stored in file <my_api_gateway.crt>
```

Create Public Cert file from Server Cert:

```
keytool -export -alias my_api_services -file my_api_services.crt -keystore my_api_services.jks
Enter keystore password:
Certificate stored in file <my_api_services.crt>
```

Import Client Public Cert File to Services jks File:

```
keytool -import -alias my_api_gateway -file my_api_gateway.crt -keystore my_api_services.jks
Enter keystore Password: my_api_services
```

Import Server Public Cert File to Gateway jks File:

```
keytool -import -alias my_api_services -file my_api_services.crt -keystore my_api_gateway.jks
Enter Keystore password: my_api_gateway
```

Configure SSL Server

1. Copy the server jks file (`my_api_services.jks`) to /src/main/resources/ folder of server application.
2. Add the entries shown below in application.yml (or application.properties)

```
spring:
 application:
   name: my_api_services
---
server:
 port: 9022
 ssl:
```

```
   enabled: true
   client-auth: need
   key-store: classpath:my_api_services.jks
   key-store-password: my_api_services
   Key-alias: my_api_services
   key-store-type: JKS
   key-store-provider: SUN
   trust-store: classpath:my_api_services.jks
   trust-store-password: my_api_services
   trust-store-type: JKS
```

3. Create a controller class with REST endpoint to serve the incoming request:

```
@RestController
@RequestMapping(value = "/my_api_services")
public class MyAPIController {
   @RequestMapping(value = "/data", method = RequestMethod.GET)
   public String getData() {
       System.out.println("Returning data from my_api_services data method");
       return "Hello from my_api_services-data method";
   }
}
```

Create SSL Client

1. Copy client jks file (`my_api_gateway.jks`) to src/main/resources/ folder of client application.
2. Add the following entries to application.yml

```
spring:
 application:
   name: my_api_gateway
---
server:
 port: 9011
 ssl:
   enabled: true
   client-auth: need
   key-store: classpath:my_api_gateway.jks
   key-store-password: my_api_gateway
   key-alias: my_api_gateway
   key-store-type: JKS
   key-store-provider: SUN
   trust-store: classpath:my_api_gateway.jks
   trust-store-password: my_api_gateway
   trust-store-type: JKS
---
endpoint:
 api_service: https://localhost:9022/my_api_services/data
```

3. Add the following dependencies to client pom file

```
       <dependency>
           <groupId>org.apache.httpcomponents</groupId>
```

```
            <artifactId>httpclient</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
```

4. Configure Spring Boot RestTemplate class to use the trust store with server certificate for 2 way https communication.

```
@Bean
public RestTemplate getRestTemplate() {
    RestTemplate restTemplate = new RestTemplate();

    KeyStore keyStore;
    HttpComponentsClientHttpRequestFactory requestFactory = null;

    try {
        keyStore = KeyStore.getInstance("jks");
        ClassPathResource classPathResource = new ClassPathResource("my_api_gateway.jks");
        InputStream inputStream = classPathResource.getInputStream();
        keyStore.load(inputStream, "my_api_gateway".toCharArray());

        SSLConnectionSocketFactory socketFactory = new SSLConnectionSocketFactory(
             new SSLContextBuilder()
                .loadTrustMaterial(null, new TrustSelfSignedStrategy())
                .loadKeyMaterial(keyStore, "my_api_gateway".toCharArray()).build(),
                NoopHostnameVerifier.INSTANCE);

        HttpClient httpClient = HttpClients.custom().setSSLSocketFactory(socketFactory)
                .setMaxConnTotal(Integer.valueOf(5))
                .setMaxConnPerRoute(Integer.valueOf(5))
                .build();

        requestFactory = new HttpComponentsClientHttpRequestFactory(httpClient);
        requestFactory.setReadTimeout(Integer.valueOf(10000));
        requestFactory.setConnectTimeout(Integer.valueOf(10000));

        restTemplate.setRequestFactory(requestFactory);
    } catch (Exception exception) {
        System.out.println("Exception Occured while creating restTemplate "+exception);
        exception.printStackTrace();
    }
    return restTemplate;
}
```

5. Create controller class with 2 methods:
    a. Get gateway's own data - `gwdata`
    b. Get microservice data - `msdata`

```
   @RequestMapping(value = "/gwdata", method = RequestMethod.GET)
   public String getData() {
       System.out.println("Returning data from my_api_gateway own data method");
       return "Hello from my_api_gateway_data method";
   }

   @RequestMapping(value = "/msdata", method = RequestMethod.GET)
   public String getMsData() {
```

```
        System.out.println("Got inside GATEWAY-ms-data method");
        try {
            String msEndpoint = env.getProperty("endpoint.api_service");
            System.out.println("API-MS Endpoint name : [" + msEndpoint + "]");

            return restTemplate.getForObject(new URI(msEndpoint), String.class);
        } catch (Exception ex) {
            ex.printStackTrace();
        }
        return "Exception occurred.. so, returning default data";
    }
```

Build & Execute

1. `mvn clean install`
2. `mvn spring-boot:run`

Verification

Browser

1. Convert JKS to PKCS12 for browser authentication

```
keytool -importkeystore -srckeystore my_api_services.jks -destkeystore
my_api_services.p12 -srcstoretype JKS -deststoretype PKCS12 -srcstorepass
my_api_services -deststorepass my_api_services -srcalias my_api_services -destalias
my_api_services -srckeypass my_api_services -destkeypass my_api_services -noprompt
```

2. Import .p12 file on MacBook
    a. Open keychain access
    b. Click on login under “keychains” and “Certificates” under Category
    c. Drag and drop the .p12 file here. It will prompt for the .p12 file password. Enter it (`my_api_services`) and add.
    d. Double click the cert you just uploaded and under “Trust” and select the “Always Trust” option. This will ask you for your login keychain password. Enter it and proceed.

3. Browser Test
    a. https://localhost:9011/my_gateway/ms1data
    b. https://localhost:9011/my_gateway/data

Postman

1. Convert .p12 to CRT file and KEY file

```
openssl pkcs12 -in my_api_services.p12 -clcerts -nokeys -out my_api_services.crt
openssl pkcs12 -in my_api_services.p12 -nodes -out my_api_services.key -nocerts
Enter Import Password: my_api_services
```

2. Launch Postman
3. Goto Postman > Preference (SETTINGS) > Certifications

4. Under “Client Certifications” section, click on “Add Certificate”