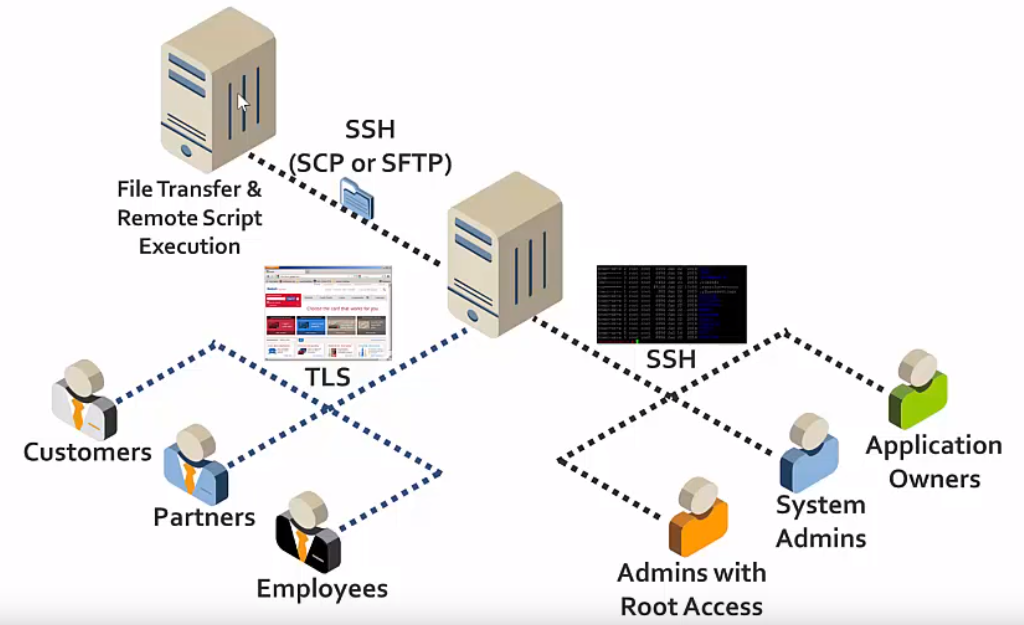
SSH/SSL Background for this exercise

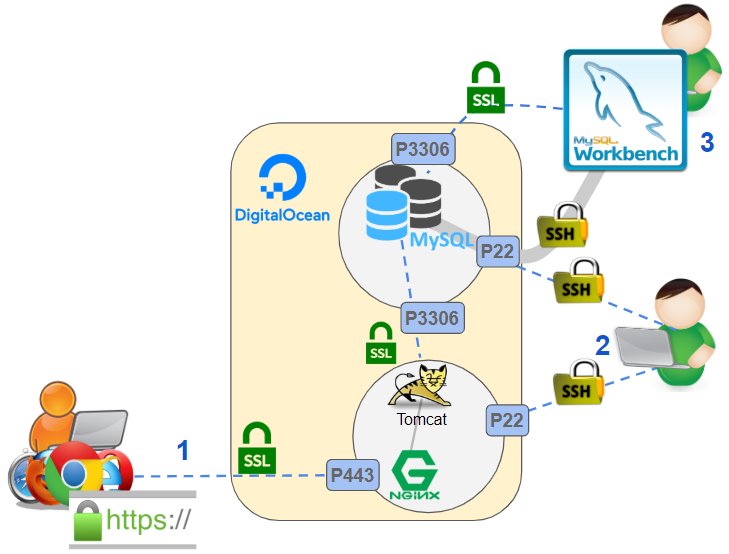
Since this is an education, focusing on how to **do “things”,** rather than plain theory, this will focus on understanding the concepts at a conceptual level,  and being apple to USE THEM.

We will do this in the reverse order, that is, first use it (this exercise), then understand it (week-13)  ;-)

The figure below is taken from this [video](https://www.youtube.com/watch?v=tCHldm7QTJo) which you should have watched before this exercise. It represents some of the common scenarios where SSH and TLS/SSL is used.



To make this a bit more clear, let's transform the figure into something you HOPEFULLY recognize.



1+ 2 represents the suggested third-semester setup (OK, last semester we added MySQL, Tomcat and Nginx on a Single Droplet), with DO-Droplets, used to host your Tomcat and MySQL servers. We delegated “encryption/decryption-tasks” to Nginx, which then forwarded (the un-encrypted) requests to Tomcat. The following scenarios were handled by this setup.

1. End-users could connect to your server, using TLS (https) without having to do anything else than entering the correct URL (http**s**)  in the browser (1 in the figure).
2. You could log-into your droplet(s) without using a password, using SSH (Secure Shell), once you had copied your public key to the droplet (2 in the figure).
3. Your Tomcat server could connect to MySQL either on the same Droplet or on a dedicated Droplet.

Securely connect to MySQL both from laptop and server to server

In this exercise, we will demonstrate different ways to securely connect to a MySQL server as sketched in the figure below.

For each of the steps, make sure you understand where “you are” relative to the figure below

Et billede, der indeholder tekst

Automatisk genereret beskrivelse

1. The usual way we connect to our droplet, but make sure you understand how many key pairs are involved, and why you always get the initial warning (where you have to spell YES) first time we connect.
2. Two different ways (there are more) to connect from Workbench, either with or without 3306 open in the firewall
3. Securely connect (using SSL) from a Java Project on your own laptop
4. Securely connect (using SSL) from another Droplet

Feel free to do this exercise in pairs or as a group, so you can split the Droplet costs.

If you have a functional setup with Tomcat you can use that, I do however recommend that you set up two dedicated droplets for this exercise, which if you use the guidelines below, should take less than 15 minutes.

Setting up the two required Droplets

1 )

Create  two new Ubuntu 18.x droplets. Name them as sketched below, to make it easy to locate the "right" one for any given task.

* **Security-Mysql**
* **Security-Tomcat**

Run **sudo apt update** on both droplets before anything else

2)

On **both** droplets, setup a non-root user the "usual" way as sketched below:

Login as root: ssh root@**your\_server\_ip**

Create a new user:  adduser **xxxx**

Grant administrative privileges to the new user: usermod -aG sudo **xxxx**

Copy  your local keys from your **root** account into the account for the new user (details [here](https://www.digitalocean.com/community/tutorials/initial-server-setup-with-ubuntu-18-04#if-the-root-account-uses-ssh-key-authentication), if you like):

rsync --archive --chown=**xxxx**:**xxxx** ~/.ssh /home/**xxxx**

Install MySQL 8.x on the first droplet (Security-Mysql)

1) On your local laptop, copy this script into the clipboard: <https://github.com/securitydatspring2019/spring2020-scripts/blob/master/setup-mysql.sh>

2) SSH into your Security-Mysql Droplet, using your non-root user and complete the following steps

Type: sudo nano setup-mysql.sh

Paste everything from the clipboard  into this file in nano .

MAKE SURE TO CHANGE the password for DB\_PW in the start of the script, and make sure to remember it, it is the root password for mysql.

Close the file again.

3) Now type:

sudo chmod +x setup-mysql.sh  (give executable rights to the file)

sudo ./setup-mysql.sh  (will execute the script)

4) Verify that you have a functional mysql server running: mysql -u root -p

Install Tomcat 9.x + Nginx on the second droplet (Security-Tomcat)

1)

On your local laptop, copy this script into the clipboard: <https://github.com/securitydatspring2019/spring2020-scripts/blob/master/setup-tomcat-nginx.sh>

2)

SSH into your Security-Tomcat Droplet, using your non-root user and complete the following steps

Type: sudo nano setup-tomcat-nginxl.sh

Paste everything from the clipboard  into this file in nano .

MAKE SURE TO CHANGE the two passwords in the start of the script. Close the file again.

3)

Now type:

sudo chmod +x  setup-tomcat-nginxl.sh  (give executable rights to the file)

sudo ./setup-tomcat-nginxl.sh  (will execute the script)

4) Verify that you have a functional Tomcat server running by executing this command:

curl localhost:8080

5) Verify that this is not accessible from outside by executing: sudo ufw status

6) Verify from a browser on your LAPTOP, that we can access the server seen i step 4, via Nginx:

http://IP\_FOR\_Security-Tomcat-Droplet

This gives you a non-secure connection, which is fine for this exercise, where focus is on letting this droplet connect to a MySQL server and not on the web-server itself.

If you insist is should not take more than 10 minutes to setup https using this section from the documentation provide last semester "Setting up your Droplet with SSL" since Nginx is already installed, BUT it's not required for this exercise

Now we are Ready to start

In the following, we will connect to this MySQL server in three different ways.

* From Workbench, via a SSH-tunnel, with the MySQL setup to allow connections ONLY on localhost, and the firewall to disallow port 3306 (B in the figure)
* From Workbench and a remote server using SSL (B in the figure)
* From a Droplet, different from the one hosting the MySQL server
* From a Simple Java Application, either running on your second droplet or locally on your laptop.

On the MySQL-server, log into a MySQL session using the root MySQL user: mysql -u root -p

Create the database we will use for all the following examples:

mysql> CREATE DATABASE example;

Create this database-user with Grants to the database you just created:

mysql> CREATE USER 'no\_ssl'@'127.0.0.1' identified by 'test';

mysql> GRANT ALL ON example.\* TO 'no\_ssl'@'127.0.0.1';

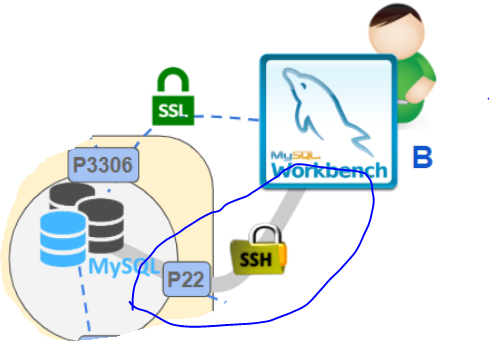
*Tilføj følgende, da ovenstående af en eller anden grund ikke virker alene. Alternativt opdater til version 19 af Workbench (Se Carolines tråd på Slack for baggrund):*

mysql> GRANT SELECT ON performance\_schema.\* TO 'no\_ssl'@'127.0.0.1';

mysql>FLUSH PRIVILEGES;

Why is it called no\_ssl

***Important:*** *If you ever plan to use this droplet for anything seriously make sure to remove this user and others created later. They are created with a default user/password to simplify the rest of this exercise*

Connecting to MySQL via an SSH Tunnel****

The purpose of this exercise is twofold.

It will provide a practical example of using an SSH-Tunnel and will demonstrate how we can access a MySQL server from a remote location, even if the server allows only connections from localhost, and the firewall disallows port 3306.

This example will connect with the user 'no\_ssl'@'127.0.0.1' created in the previous step.

On the server

To ensure that we can only connect to the database from localhost, open the file /etc/mysql/my.cnf with nano: sudo nano /etc/mysql/my.cnf

Add this to the end of the file:

[mysqld]

# Require clients to connect either using SSL

# or through a local socket file

# require\_secure\_transport = ON

bind-address = 127.0.0.1

Restart the server: systemctl restart mysql

Close port **3306** on the Firewall: sudo ufw deny mysql

In Workbench

Now it’s time to connect to the MySQL from your local computer.  Since the server only listens on localhost, and port 3306 is closed, the only way to do this is via an SSH-tunnel running on (default) port 22.

Do this as sketched below:

1) Select “Setup a New Connection” in Workbench  and add ***no\_ssl\_over\_tunnel*** as the “Connection Name” (we will add more connections, so now we know which user this connection belong to)

2) Select “Standard TCP/IP over SSH” as Connection Method

3) Fill in remaining details as outlined below:

* **SSH Hostname**: IP\_FOR\_YOUR\_MYSQL\_DROPLET:22
* **SSH Username**: SSH\_USER     (the normal non-root SSH user you have created on your droplet)
* **SSH Key File**:   (on Windows typically found on C:\Users\USERNAME\.ssh\id\_rsa)
* **MySQL Hostname:** 127.0.0.1 (We are using a tunnel, so this is what MUST be used when “we are **in**”
* **MySQL Server Port:** 3306 (We are using a tunnel, so once we “are **in**” the firewall has no effect)
* **Username:**  no\_ssl  (The db-user you created only for 127.0.0.1)

4) Press “Test Connection” and verify that you can connect over an SSH-Tunnel.

Connecting directly to MySQL with Mandatory SSL

As explained [here](https://www.ssh.com/ssh/tunneling/), tunnelling comes with its own set of risks, and when we need to connect applications (a web server for example) to the database this solution is cumbersome. In this part, we will connect directly to the MySQL-server on port 3306 using SSL, both from our local developer laptop, via Workbench, from a remote Droplet and from a Java Application. The following is a customized/shortened version of [this tutorial](https://www.digitalocean.com/community/tutorials/how-to-configure-ssl-tls-for-mysql-on-ubuntu-16-04).

Getting started

Change the database server to allow remote connections (using nano):

sudo nano /etc/mysql/my.cnf

Change the value for bind-address like this: bind-address = 0.0.0.0

Uncomment this line in the file:

require\_secure\_transport = ON

Restart the server:  sudo systemctl restart mysql

Allow port  3306 in the firewall: sudo ufw allow mysql

Check the Current SSL/TLS Status

Before you continue, check the current status of SSL/TLS on our MySQL-server instance:

Log into a MySQL session using the root MySQL user the usual way

mysql -u root -p

Check the status of our current connection to confirm that SSL is currently NOT in use for the connection:

mysql>  \s

Close the current MySQL session: mysql> exit;

Generate SSL/TLS Certificates and Keys

You can skip this step since it seems like MySQL 8.x creates the required certificates and keys during installation. It could be argued however, that it's always best to generate these things by yourself, by I suggest you skip the step highlighted below:

To enable SSL connections to MySQL, we need to generate the appropriate certificate and key files. A utility called mysql\_ssl\_rsa\_setup is provided with MySQL 5.7 and above to simplify this process.

The files will be created in MySQL's data directory, located at /var/lib/mysql. The MySQL process need to be able to read the generated files, so pass mysql as the user that should own the generated files:

sudo mysql\_ssl\_rsa\_setup --uid=mysql

Check the certificates and keys, either the ones already there, or those generated by you if you did the step above generated files by typing:

sudo find /var/lib/mysql -name '\*.pem' -ls

This should, among several other things, show the generated files with the user and group ownership.

These files are the key and certificate pairs for the:

* Certificate authority (starting with "ca")
* MySQL server process (starting with "server")
* MySQL clients (starting with "client").

Additionally, the private\_key.pem and public\_key.pem files are used by MySQL to securely transfer password when not using SSL.

Enable SSL Connections on the MySQL Server

Newer MySQL versions will look for the appropriate certificate files within the MySQL data directory when the server starts. Because of this, all you need to do to enable SSL is to restart the MySQL service:

sudo systemctl restart mysql

Log into a MySQL session again, using the root MySQL user. THIS TIME however, to ensure that the client makes a TCP/IP connection to the local server, use --host or -h to specify a hostname value of 127.0.0.1

mysql -u root -p **-h 127.0.0.1**

Take a look at the same information requested last time. Check the values of the SSL related variables:

mysql> SHOW VARIABLES LIKE '%ssl%';

Next, check the connection details again: mysql> \s

This time, the *specific SSL cipher is displayed, indicating that SSL is being used to secure our connection*.

Now create a new remote user, and add grants, which we will use to test the SSL connection.

mysql> CREATE USER 'everywhere'@'%' identified by 'test' REQUIRE SSL;

mysql> GRANT ALL ON example.\* TO 'everywhere'@'%';

mysql> FLUSH PRIVILEGES;

Exit back out to the shell: mysql> exit

Now. your MySQL server is configured with an SSL certificate signed by a locally generated certificate authority (CA). The server's certificate and key pair are enough to provide encryption for incoming connections which we will verify in the next section.

Testing Remote SSL Connections

1) From Workbench**Et billede, der indeholder tekst, kort

Automatisk genereret beskrivelse**

1) Select “Setup a New Connection” in Workbench and add ***ssl\_everywhere*** as the “Connection Name”.

2) Select “Standard (TCP/IP)” as Connection Method

3) Fill in remaining details as outlined below:

**In the Parameters tab:**

* **Hostname**: IP\_FOR\_YOUR\_MYSQL\_DROPLET
* **Port:** 3306 (Since we are connecting directly to the MySQL-server)
* **Username**: everywhere     (the user you just created for part)
* **Password**:  test   (if you did a raw copy of the CEATE USER statement)
* **Default Schema:** example (The database we have created for this demo)

**In the SSL tab**

* Use SSL: Require

4) Press Test Connection to verify that you can connect, using TCP.

2) From your Client Droplet (The one with Tomcat)**Et billede, der indeholder skilt

Automatisk genereret beskrivelse**

Getting started

For this part, you need Security-Tomcat Droplet if you already have a ready to use droplet with Nginx and Tomcat use this.  On this droplet you must install the MySQL package as outlined below:

sudo apt update

sudo apt install mysql-client

Now connect to make sure **you can connect** to the server successfully.

mysql -u everywhere -p -h **your\_mysql\_server\_IP**

Verify that the connection is secure: mysql> \s

Exit back out to the shell: mysql> exit

Attempt to connect insecurely, and **verify that this is not possible**:

mysql -u everywhere -p -h **your\_mysql\_server\_IP** --ssl-mode=disabled

Restricting clients allowed to connect

We can do even more to increase security for our database server

What we have done so far, is to allow access from everywhere, which is never a good idea

A quick way to restrict access to only selected IP's is via the firewall on the database server, as sketched below:

sudo ufw allow from **SECURITY\_TOMCAT\_IP** to any port 3306

Verify that you can still connect with the everywhere-user from the Security-tomcat droplet

If you still want to connect from your own laptop, you need to open up the firewall for your current IP in a similar way. Alternatively, as you have seen, you can just connect via the SSH-tunnel created in an earlier step.

 You could leave it  here, but to make this even more secure, you could remove the everywhere user and create a new user for ONLY the IP for your Security-tomcat droplet

3) From a Remote Java Application

Finally, create a simple (non-web)Java Maven Application and repeat the following steps which should all be familiar after last semester.

Complete the following steps:

* In NetBeans create a new Maven Java Application (not WEB)
* Add the  mysql-connector-java connector to the POM-file.
* Create a new Entity Class called Demo. DO NOT select “create persistence Unit”
* Add a single String-field to the class called info. Provide getters, setters and constructors as needed.
* Create a new folder **META\_INF** in  \src\main\resources
* In this folder, create a file persistence.xml and copy the content below into the file (change relevant values)

<?xml version="1.0" encoding="UTF-8"?>

<persistence version="2.1" xmlns="http://xmlns.jcp.org/xml/ns/persistence" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence http://xmlns.jcp.org/xml/ns/persistence/persistence\_2\_1.xsd">

  <persistence-unit name="pu" transaction-type="RESOURCE\_LOCAL">

    <provider>org.eclipse.persistence.jpa.PersistenceProvider</provider>

    <class>entity.Demo</class>

    <properties>

      <property name="javax.persistence.jdbc.url"

                value="jdbc:mysql://YOUR-DB-IP:3306/example?useSSL=true&amp;verifyServerCertificate=false"

      />

      <property name="javax.persistence.jdbc.user" value="everywhere"/>

      <property name="javax.persistence.jdbc.driver" value="com.mysql.jdbc.Driver"/>

      <property name="javax.persistence.jdbc.password" value="test"/>

      <property name="javax.persistence.schema-generation.database.action" value="create"/>

    </properties>

  </persistence-unit>

</persistence>

Finally create a new class Tester and copy the content below into the file

public class Tester {

  public static void main(String[] args) {

    EntityManagerFactory emf = Persistence.createEntityManagerFactory("pu");

    EntityManager em = emf.createEntityManager();

    try {

      em.getTransaction().begin();

      em.persist(new Demo("Info-1"));

      em.persist(new Demo("Info-2"));

      em.persist(new Demo("Info-3"));

      em.getTransaction().commit();

    }finally{

      em.close();

    }

  }

}

**Hint:** I had to add this entry to the POM-file to make it build:

 <dependency>

     <groupId>javax.annotation</groupId>

     <artifactId>javax.annotation-api</artifactId>

     <version>1.3.1</version>

</dependency>

Verify that you can connect using SSL, and that the Demo table has been created and populated with the three entries

Verify that you cannot connect without SSL, by changing the value of useSSL to false in the connection string.