SWitCH – Computing systems and networks (SCOMRED)

2021/2022

Laboratory 06

Development environment and production environment.

Deploying contents into servers.

Apache/CGI based dynamic web contents. BASH based CGI applications.

Backend and web services testing - postman. Team project development.

1. Development environment and production environment

Under the point of view of software engineering, the software lifecycle is made of several stages, starting with planning, analysis, and design, then development together with testing, and finally deployment, operation, and maintenance. Of course, then it will often start again (it's a cycle).

Different stages in the software lifecycle will require different skills, tools, and possibly, different running environments for the application being developed. Three clearly distinct running environments can be identified for the application: **development environment**, **testing environment**, and **production environment**.

1.1. Development

Most appropriately, an Integrated Development Environment (IDE) should be used when developing applications (development stage), usually the IDE encompasses a running environment for the application, and thus it can be tested while being developed.

However, the availability of a running environment together with the development environment is not guaranteed. Generally speaking, it may not be available if the **production environment** (where the application was ultimately designed to be used) is substantially different from the **development environment**.

One example that immediately may strike our mind is developing an Android or IOS application in a Windows, Linux, or OS X workstation. For this specific scenario, the most often used solutions are either a device emulator or a real device directly attached to the workstation.

When developing a Web application backend, or distributed applications in general, it might as well not be possible to run and test the application in the development environment, simply because the required running environment is too complex to be established in the development environment. Take for instance a set of different applications that are supposed to run each on a different network node, using different web servers and database servers. Even if it was possible to setup all those services in the development environment it wouldn't be a good testing environment because all applications and services would be running on the same node. A testing environment is a better solution.

1.2. Testing environments

If a running environment for the application is not available in the development environment, then an external testing environment is required, from one example above, it may be an Android or IOS emulator.

Even if a running environment for the application is available in the development environment, when developing distributed applications with interdependencies, and applications depending on special network services, a complex testing environment is required. The ultimate solution is creating a testing

environment that mimics as far as possible the production environment, such a testing environment is often called sandbox.

Because it's a separate environment, testing over the sandbox will not disturb the production environment, and yet because they are very alike, once the applications are thoroughly tested, transferring them into production is pretty straight forward.

Operating systems virtualization is now becoming an alternative by creating an instant testing environment. By starting several containers, each running a different application or network service, a testing environment is created with the single purpose of running the applications once. Such testing environment only exists while the application is running, once the application ends running containers are stop and most often destroyed (ephemeral containers). One advantage of this kind of testing environment it that the environment is always the same when the application test starts.

The use of an instantly created containers-based testing environment, doesn't mean a full testing environment that better mimics the production environment, will not be required. However, in such case it will be then a clearly independent testing phase following the development.

1.3. Deployment

In several scenarios applications must be pushed into servers, this basically means uploading file. This is the case for production servers and as well for testing servers in a testing environment similar to the production environment.

To ensure a total independence from the IDE, from development libraries, and from development frameworks, production hosts should be used to run the final products. Such production hosts will simply not have the development related software packages installed.

Many popular IDEs provide deployment mechanisms, most often based on SFTP (SSH File Transfer Protocol, also known as Secure File Transfer Protocol). SFTP is an extension to the SSH protocol allowing file transfers in an FTP style, though it's not related to FTP (File Transfer Protocol) neither to FTPS (FTP over SSL/TLS).

Most SSH servers support SFTP, so, if you have SSH access to a host, it's likely you have SFTP access as well, by using the same credentials.

Compatibility between the development host and the production host is required. If it's an interpreted application, things are rather simple because the ASCII encoding is pretty much universal, thus text is text everywhere[1]. Nevertheless, the interpreters' versions must be compatible, usually backward compatibility is assured.

If it's a compiled application, binary compatibility is required. Java compiled programs, as we know, run on a special environment called Java Runtime Environment (JRE), it includes the Java Virtual Machine (JVM). For each operating system, a specific JRE implementation is required, yet the JRE is always the same, thus it behaves as an abstraction layer, meaning, the same compiled Java application will run over the JRE whatever the underlying operating system is.

For instance, a Java application may be developed and compiled in an MS-Windows system, then compiled files may be copied to a Linux system, and the application runs there with no changes. Even so, there are several Java versions, backwards binary compatibility is assured, e.g., an application developed and compiled in Java 8 will run on Java 11 runtime.

[1] Actually, as you know, text files in UNIX like systems (e.g., Linux) are slightly different from text files in Windows systems in the way text lines are ended. In DOS/Windows text files, each line ends with a Carriage Return (CR) followed by a Line Feed (LF). In UNIX text files, each line ends with Line Feed (LF). In Mac text files, prior to Mac OS X, lines end with Carriage Return (CR). Nowadays Mac OS uses the UNIX style (LF) line breaks.

2. The Hash Calculator – deployment, testing and frontend

In lecture six, a hash calculator backend application has been introduced, it's a BASH based CGI application.

```
#!/bin/bash
M_CONTENT_FILE=/tmp/.scomred-hash-calculate.$$.tmp
### I'm file /var/www/cgi-bin/hashCalculate with execute permission
error_response() {
 echo "Status: 400 Bad Request"
 echo "Content-type: text/plain"
 echo ""
 echo "ERROR: ${1}"
 rm -f $M_CONTENT_FILE
 exit
}
###
if [ -n "$CONTENT_LENGTH" ]; then cat > $M_CONTENT_FILE
else error_response "No content found on the request"; fi
if [ "$CONTENT_LENGTH" == "0" ]; then error_response "No content found on the request"; fi
###
if [ "$REQUEST_METHOD" != "POST" ]; then
       error_response "Invalid method. The only supported method is POST."
       fi
if [ -z "$QUERY_STRING" ]; then error_response "No query-string"; fi
ALG=${QUERY_STRING#algorithm=}
if [ "$ALG" == "$QUERY_STRING" ]; then error_response "Bad query-string: $QUERY_STRING"; fi
case "$ALG" in
        MD4|MD5|RIPEMD160|SHA1|SHA224|SHA256|SHA384|SHA512);;
        *) error_response "Invalid hash algorithm: $ALG";;
esac
HASHCODE="$(openssl dgst -$ALG $M_CONTENT_FILE)"
rm -f $M_CONTENT_FILE
echo "Content-type: text/plain"
echo ""
echo "${HASHCODE#*= }"
###
```

If the request has a content (body), as it's supposed to, then the CONTENT_LENGTH variable is defined (not empty), the content is available through the STDIN. The **cat** command is used to read all available data in STDIN and copy it to a temporary file. The $$ sequence in the temporary filename is expanded to the current process's PID, if two instances of this application are running at the same time in parallel each will have a unique PID, thus used filenames will be unique as well.

The only supported HTTP method is POST, if the received HTTP request is not a POST, then a compliant HTTP error message is sent by calling the error_response() function.

The HTTP request's URI is supposed to carry a query-string with the algorithm parameter, otherwise an error message is sent back.

If the hash algorithm is supported (actually supported by the openssl command), then the openssl dgst command is used to calculate the hash code using data in the temporary file. The result is sent as response's content after removing a prefix included in the openssl command output.

2.1. Deployment

To be treated by the web server as a CGI application some settings and conditions are required on the server side, students are expected to use the VS template number 13, **Apache and others on Ubuntu 20.04 LTS (Focal)**, as suggested in Lab 02.

CGI support is already enabled on this template, executable files (with execute permission) that are stored in folder **/var/www/cgi-bin/** are handled by the Apache web server as being CGI applications and not documents (stored in folder /var/www/html).

The **/var/www/html/** folder is the **document root**, meaning under URI point of view it's the root folder (**/**). This means the URI **/index.html** is in fact file **/var/www/html/index.html**. The entire directories tree and documents starting at folder **/var/www/html/** is available as URI starting at **/**.

Beyond this basic tree of documents starting from the **document root**, the web server may be configured to provide other resources that will be mapped into the basic tree as **aliases**. In this configuration folder **/var/www/cgi-bin/** is mapped to URI **/cgi-bin/**, thus the directories tree starting from **/var/www/cgi-bin/** is available as URI starting from **/cgi-bin/**.

Activities:

a) Deploy the Hash Calculator CGI application into the **/var/www/cgi-bin/** folder of your VS, you may name the file as **hashCalculate** (**/var/www/cgi-bin/hashCalculate**).

Don't forget granting it the execute permission to the file.

b) It should be now available as **http://vsX.dei.isep.ipp.pt/cgi-bin/hashCalculate**, you may test it in your web browser.

If everything went as expected, you will get the following plain text:

ERROR: No content found on the request

This is because this URI is a web service and it's not intended for direct user interaction.

c) Look at the application code to understand why and where this response message was generated.

2.2. Testing with Postman

Clearly, testing web services with a web browser and nothing else is not a solution.

As reported in lecture six, one of the most popular tools to test web services is Postman (https://www.getpostman.com/).

Activities:

a) Install on your workstation the Postman application, it's not mandatory to install the full desktop version, a Postman plugin for your web browser will do perfectly.

b) Refer to lecture six and execute yourself the sequence of tests presented there as an example.

c) Try other different tests over your web service. See if you can find a bug, e.g., some missing input data validation.

2.3. Developing a frontend for the backend

A web service like for instance our **hashCalculate** is not specifically designed to be used by a frontend, a web service is designed to be used by applications named as consumers the same way applications call local functions available on the local operating system.

Such web services consumer applications may or not be a frontend, for instance a backend application may as well be a web services consumer, for instance if at some point a hash code calculation is required our **hashCalculate** could be called. The fact is, by using AJAX a browser-based frontend application can become a web services consumer application.

Now that the backend (the **hashCalculate** web service) is tested, consumers may be developed.

Activity/challenge:

Develop a web browser-based frontend (HTML+JavaScript) to use the **hashCalculate** web service and make it directly available to users.

- The frontend UI can be as simple as one input text field to specify the hash algorithm name, a text area box to place data whose hash code will be calculated, and a calculate button for action.
- No significant input validations are required, those are already enforced on the backend, and under all points of view that's the right place to validate input data.