5e année ISS 2018/19

Cloud Computing: Adaptability and Autonomic Management

Lab 3: Provisioning End-user Application in Cloud Platforms

S. Yangui (INSA/LAAS)

Accès au sujet : goo.gl/EPQfwA

Proxmox servers assignment : goo.gl/DDbdDt

Objectives

This lab aims at given students a good understanding of the service-based application lifecycle management in cloud computing context. The students will study the several provisioning process phases and investigate the way they should be implemented when provisioning: (i) sample HelloWorld code on existing and various cloud (PaaS-level) platforms (i.e. Google Cloud Platform, Cloud Foundry and Jelastic) and, (ii) the Resources Generator and Manager the students developed as part as Lab 2. The detailed objectives are detailed in what follows:

- Objective 1: Read and assimilate the several phases that make up cloud end-user applications provisioning process.
- Objective 2: Implement such a process using sample JEE application on Google Cloud Platform, Cloud Foundry and Jelastic. Various approaches and tools will be used for each one of the previously mentioned PaaS (e.g. through a plugin, a CLI and the Web).
- Objective 3: Learn how to build a more complex cloud application, and the application architecture concepts/principles you will need to succeed.
- Objective 4: Implement the lessons learned from Objective 3 to design and develop a simple MVC application with database access, on a PaaS.
- Objective 5: Design and develop your own embryonic PaaS using LXC containers on top of the Proxmox infrastructure and migrate your Resources Generator and Manager applications over there.

Provided Materials:

- Text addressing Objectives 1 and 3 that you need to read very carefully, several times if necessary.
- Sample JEE application artifacts (HelloWorld Web archives)

- Credentials for PaaS providers that you need to connect to when addressing Objective 2.

Email	Password
yy3155339A@gmail.com	5iss_A!?
gei5iss.a2@gmail.com	5GEIISSa2
yy3155339C@gmail.com	5iss_C!?
yy3155339D@gmail.com	5iss_D!?

Note: For Jelastic, you need to register and create your own credentials. It is easy, fast and free.

Required Releases:

- HelloWorld Servlet provisioned on Google Cloud Platform, Cloud Foundry and Jelastic. You have to provide the Lab's teacher with the link so he can test the application execution on the target PaaS. You imperatively need to deliver this within the lab hours.
- MVC application provisioned on a PaaS of your choice. You have to deliver: (i) the code published in Git and, (ii) the remote link to test its execution (to be mentioned in the shared Google sheet).
- Your PaaS Prototype running on top of Proxmox. You have to deliver: (i) the hosting VM/CTs and, (ii) the required credentials if there are any (VM/CTs, PaaS and so on).

General notes:

- Clarification: In this lab, "cloud applications" refers to the "end-user applications".
- Terminology 1: "Cloud" applications are also known as "cloud-based", "cloud-ready" or "cloud-enabled" applications. "Cloud-friendly" means the same although it is much less used.
- Terminology 2: Provisioning cloud applications is a continuous loop process that implements the whole applications lifecycle (i.e. develop, deploy and manage). This includes then the design and build activities previously mentioned. More generally, the different activities related to each one of these three phases will be discussed in detail in the rest of this document.

Task 1. Addressing Objective 1: Read and assimilate the several phases and steps that make up cloud end-user applications provisioning process

Provisioning cloud resources process consists of three main phases. Such a process implements the resources lifecycle and is strongly inspired from Service-oriented Computing paradigm (SOC) and the Telecommunications Information Networking Architecture (TINA) [1] [2]. In particular, provisioning cloud (end-user) applications in PaaS according to such process is briefly discussed in [3]. In what follows, in-depth details of the different phases:

1. Phase 1: Development

It involves (non-exhaustive list, not all sub-phases are mandatory, some of them might be concurrent):

- Modeling the application (e.g. designing the required service / component-based topology)
- Developing the sources (e.g. writing code and scripts, referring to remote components/modules published over repositories, importing required libs)
- Compiling the sources
- Testing and debugging
- Building the artifacts (a.k.a executables, archives, deployables)
- Packaging in appropriate wrappers if necessary (e.g. service containers, standalone frameworks)
- Modeling the management plan
- Settling the SLAs
- Specifying the deployment descriptor (a.k.a manifest)

2. Phase 2: Deployment

It involves:

- Allocating the required cloud resources
- Uploading the executables over these resources
- Activating the application according to the specified plans/SLAs (including the automatic binding and activation of the required services – such as storage, logging, monitoring and so on.

3. Phase 3: Management

It involves:

- Executing the application
- Performing the appropriate/required management operations that aim at optimizing its performance and reducing its operation cost. SLAs violations, from end-user point of view, would trigger the execution of a specific management operation. Management operations are specified through the management plans and might be implemented as workflows. Scaling up/down or migrating an application component are among the examples of management operations.

Task 2. Addressing Objective 2: Implement part of the provisioning process using sample JEE application on Google Cloud Platform, Cloud Foundry and Jelastic

As part of this task, you need to provision a HelloWorld Servlet on:

- Google Cloud Platform using Eclipse and Google Plugin
- Cloud Foundry using its CLI software and the Web console
- Jelastic using the Web console

Depending on the case study, you need to implement one or several steps for each phase of the application provisioning process.

Task 2.1 Provisioning HelloWorld Servlet on Google Cloud Platform

The goal of this subtask is to provision a HelloWorld Servlet on Google Cloud Platform. We will use Eclipse as IDE, as well as, the appropriate Google plugin.

For the development phase, you will need to:

- Install Google Plugin and Google SDK to settle your development environment
- Debug your code and “emulate” its deployment on the cloud (Google SDK allows this emulation on your own machine as if you were in the Cloud)

- Create a Google development project and link it to your local application

Note that for the development phase, there is no code to write. The sample code of the HelloWorld servlet is automatically generated by Google SDK.

For the deployment phase, you will push your application to the distant PaaS.

For the management phase, you will execute your application and get familiar with the Google management dashboard.

Work to do :

1. Install the Google Eclipse Plugin . Select **Help > Install New Software** in the bar menu and insert the following URL: https://dl.google.com/eclipse/plugin/4.6

Note: You may decide to use the novel tool launched by Google late 2018, i.e. Google Cloud Tools instead in order to proceed to the same deployment procedure.

Note: This plugin URL provided above is specific for Eclipse Kepler. For other distributions, you need to select the suitable plugin through the following link:

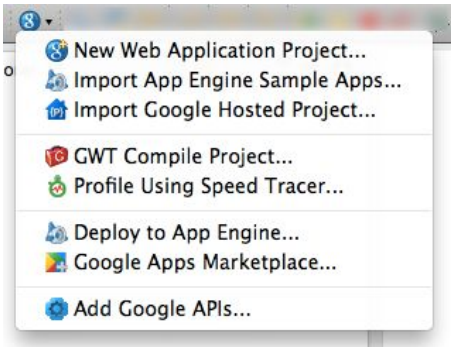
https://developers.google.com/eclipse/docs/install-eclipse-4.6

For the installation, check the followings plugins:

- Google Plugin for Eclipse
- SDK > Google App Engine Java SDK
- SDK > Google Web Toolkit SDK (optional).

Finally, you need to accept the review licenses and validate to start the installation. At the end, you need to validate the security warning and restart Eclipse.

If the installation is successful, a new Google icon with a specific menu will be added to your Eclipse.



Note: In order to support and develop GAE projects with Maven, please refer to: https://cloud.google.com/appengine/docs/java/tools/maven

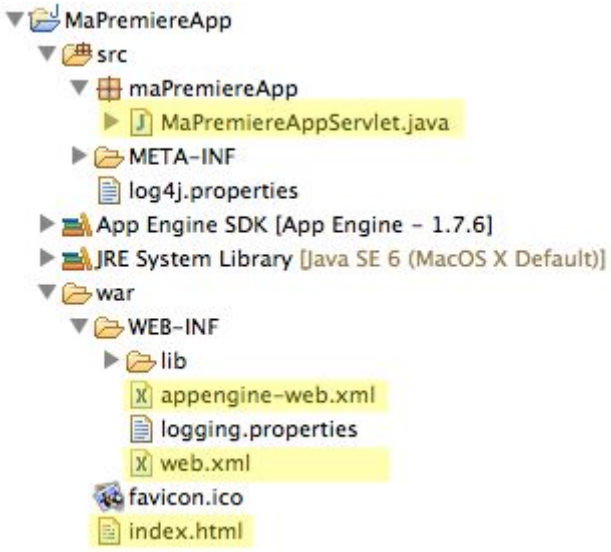
2. Create a new Google APP Engine application by clicking on the "**g**" Google button on Eclipse and select **New Web Application Project**.

The application creation wizard is displayed. You need to:

- Insert the project name (e.g. MyFirstApp)
- Insert the package name (e.g. myFirstApp)

- Uncheck “Use Google Web Toolkit” option.
- Check “Use Google App Engine” option

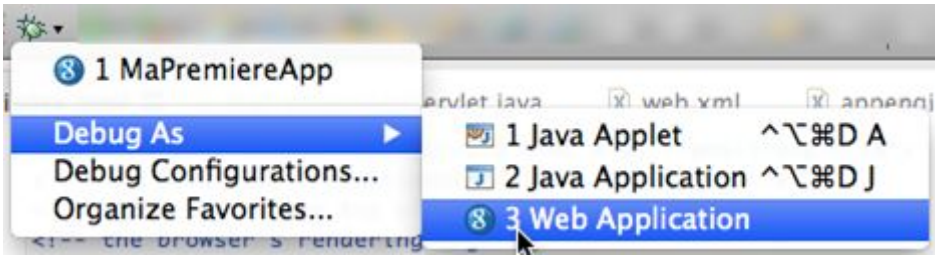
The created project architecture consists of a set of files and repositories. The content structure is very similar to a classical JEE project structure. The main files are highlighted in the following snapshot. Note that there is a specific XML file (i.e. appengine-web.xml) under \WEB-INF. Such a file allows to configure the use of Google resources/services (e.g. sessions management) by your application.



where :

- appengine-web.xml = google specification indicating the application identificator, version of the last code, ...
- web.xml = deployment descriptor of the application (application name, description, servlets JAVA classes, servlets mappings ⇔ “servlet-URL”)
- index.html = default welcome web page when the servlet is invoked.

3. Debug your application and test it locally (**Debug as" > "Web Application")**



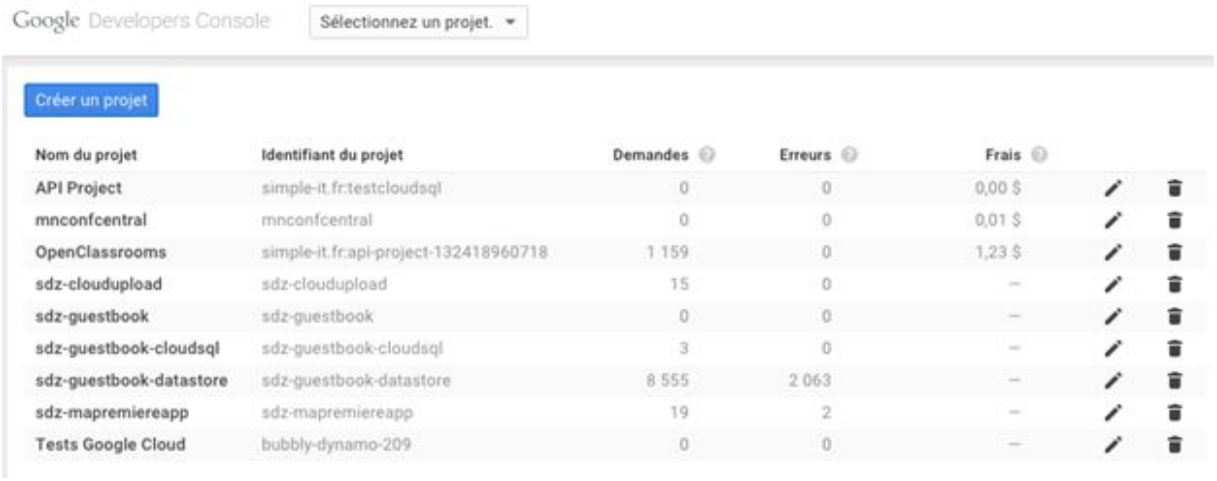
You should have **“*INFO: Dev App Server is now running*”** displayed then on your Eclipse console. This means that your local google mini-server is running.

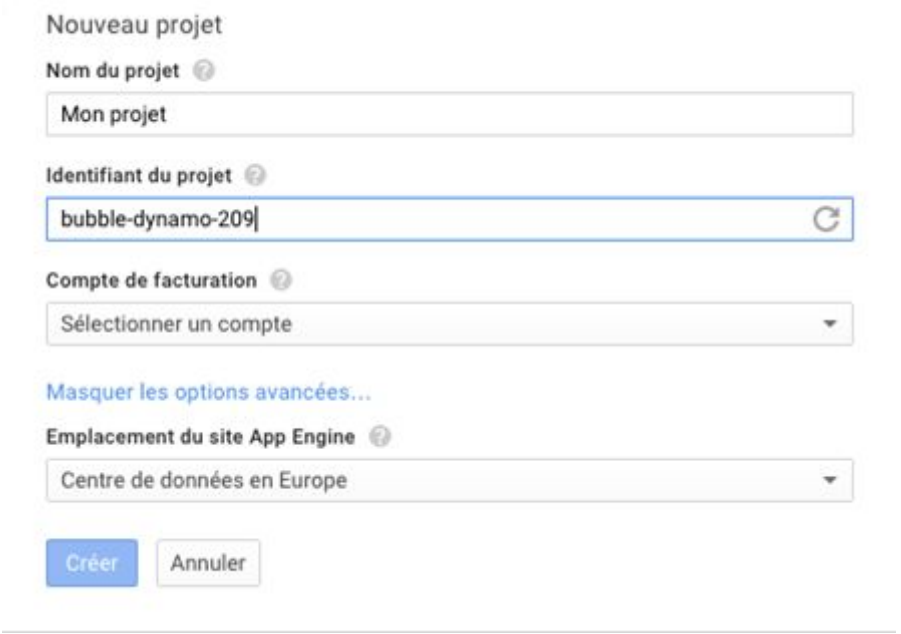
Use a browser and go to http://localhost:8888/ to test. If everything works well, you will be redirected to your index.html page.

Test the execution.

Note that it is possible to administer and monitor your application using the local administration tool provided by the plugin. The administration console is accessible at: http://localhost:8888/_ah/admin

4. Declare your application to Google. To that end, you need to visit http://console.developers.google.com, create a project and and get a unique ID for it.



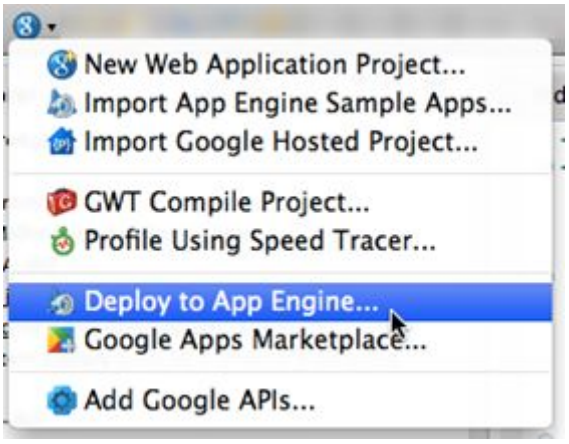


5. Configure your application and make it ready for deployment. Copy and paste this ID between the <*application*> elements in your **appengine-web.xml** file.

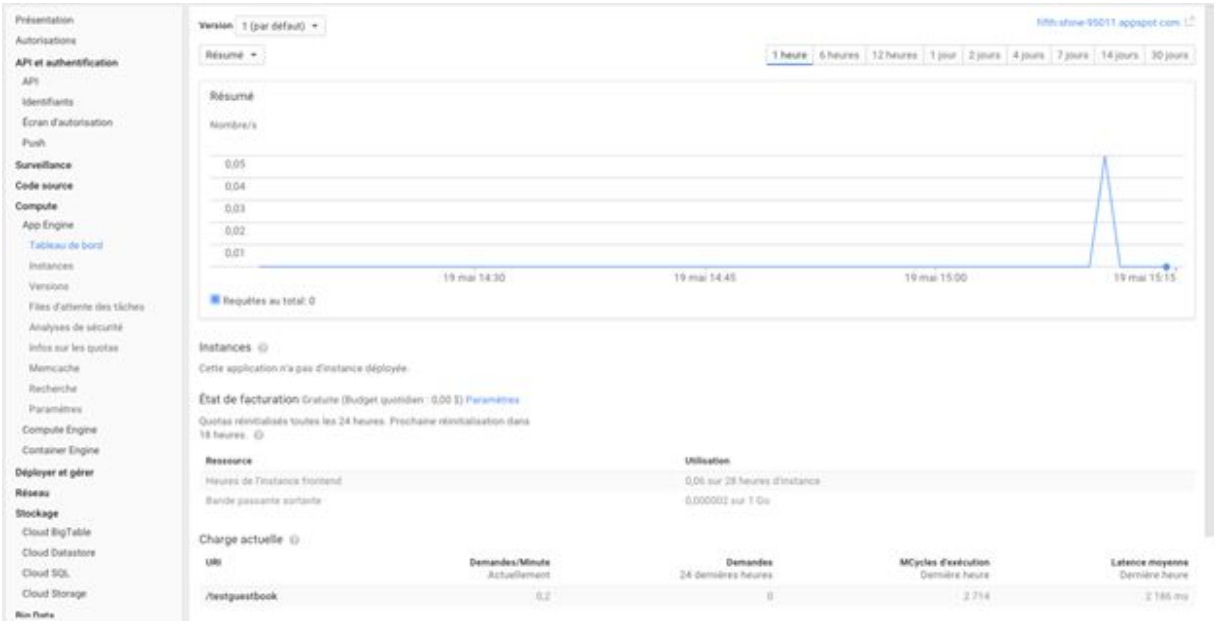
```
<?xml version="1.0" encoding="utf-8"?>

<appengine-web-app xmlns="http://appengine.google.com/ns/1.0">
<application>sdz-myfirstapp</application>
<version>1</version>
```

6. Deploy your application. Click on "**Deploy to App Engine**" in the google menu (Authentication to Google is required).



7. Manage your application. Once your App is deployed, you can manage it through the administration dashboard (Compute / App Engine / Dashbord).



Task 2.2 Provisioning HelloWorld Servlet on Cloud Foundry

The goal of this subtask is to provision a HelloWorld Sevlet on Cloud Foundry. To that end, we are going to use: (i) Pivotal Web Services (PWS) that provides Cloud Foundry as a Web service (deployed on top of AWS) and (ii) CF CLI.

For the development phase, we provide you with the application artifacts ready; however, you will need to:

- Install and configure the CF CLI as part of your development environment
- Settle the deployment descriptor (manifest.yml) provided with the artifact

For the deployment phase, you need to:

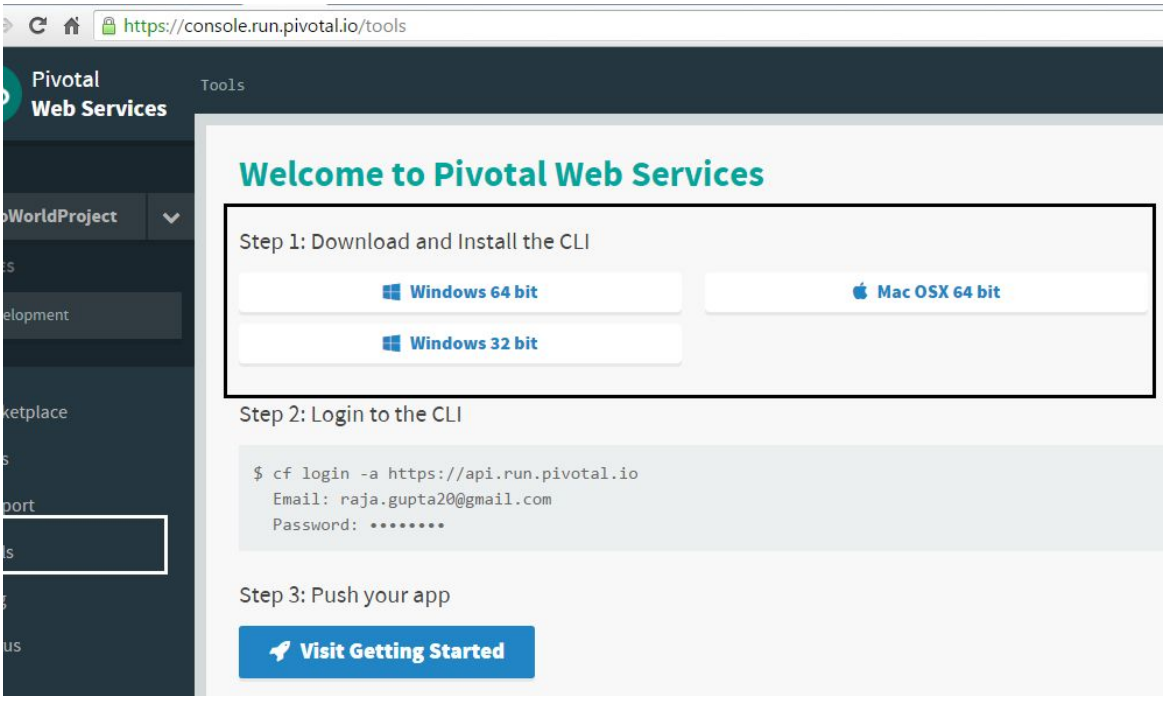
- Prepare the hosting environment on CF (creation of correspondent organization and space as target domain for your application)
- Push the artifacts there.

Note that for Cloud Foundry, the allocation of the PaaS resources is implicitly handled by the PaaS. Depending on the application type and implementation (i.e. JEE application in this case), the required PaaS resources (e.g. Apache Tomcat container and nginx router in this case) will be automatically allocated and configured by the PaaS rather than the developer.

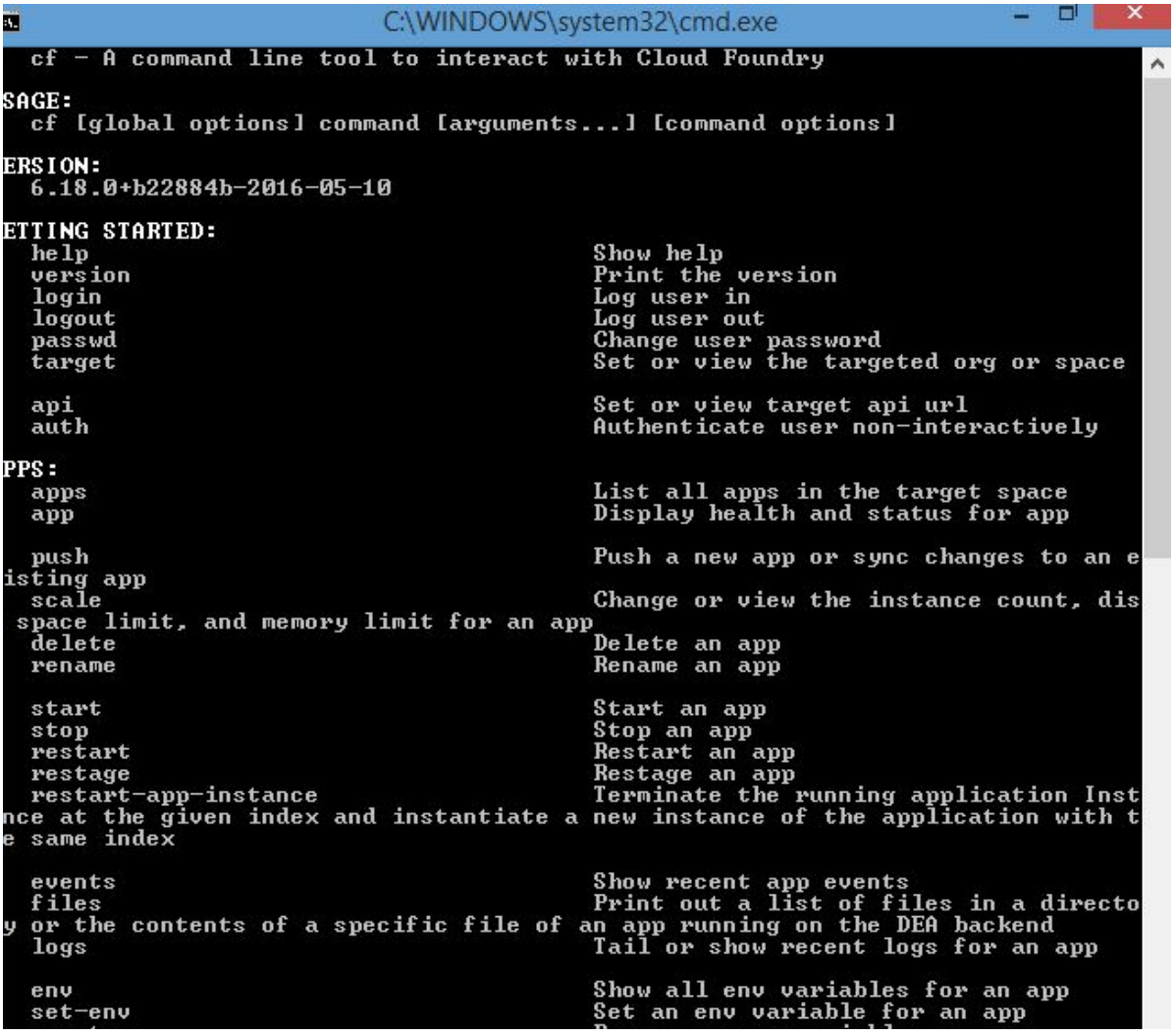
For the management phase, you will need to:

- Execute the application through the generated URL
- Scale up/down the application
- Bind it with a storage service
- Stop the application and terminate it

1. Connect to Pivotal Web Services , sign up and log in
2. Install the Cloud Foundry command line interface CF CLI (see http://docs.run.pivotal.io/cf-cli/install-go-cli.html).
   Tip: The simplest and the fastest way is to get the installer from the Web console (under Tools). If you are working on the INSA machines, you need to subvert the privileges restrictions when installing the client (See Appendix 1).



3. Open Command Prompt and run ***"cf help"*** to confirm that the tool is installed correctly. The example shows the beginning lines of output for this command. Read all the supported commands so you get familiar with the CLI capabilities. The comprehensive guidelines are available through the following link.
4. Enter command ***"cf api api.run.pivotal.io"*** to set the API endpoint. ***api.run.pivotal.io*** indicates the API of the public Cloud Foundry instances that we



are going to use and that is provided on top of Amazon infrastructure.
5. Enter command ***"cf login"*** to connect to the PaaS
6. Enter command ***"cf target [-o ORG] [-s SPACE]"*** to create and select a specific space under a given organization (e.g. WEB or INSA applications as ORG and DEV as space).
7. Download the HelloWorld artifacts and open the deployment descriptor (manifest.yml) to edit: (i) the application name (host attribute) and (ii) the number of the application's instances to be deployed (instances attribute).
8. Enter command ***"CD [HelloWorld_PATH]"*** to place your prompt under its folder.
9. Enter command ***"cf push"*** to upload the application artifact on cloud Foundry. The operation will not succeed and you will have errors. Based on the displayed information on your command line, can you explain this failure? For more details, you

might open and go through the logs using the appropriate commands or displaying them on the Web console.

```
2017-10-18T19:21:31.28+0200 [CELL/0] OUT Creating container
2017-10-18T19:21:31.80+0200 [CELL/0] OUT Successfully created container
2017-10-18T19:21:34.93+0200 [CELL/0] OUT Starting health monitoring of container
2017-10-18T19:21:35.05+0200 [APP/PROC/WEB/0] ERR Cannot calculate JVM memory configuration: There is insufficient mem
ry remaining for heap. Memory limit 256M is less than allocated memory 654457K (-XX:ReservedCodeCacheSize=240M, -XX:Max
irectMemorySize=10M, -XX:MaxMetaspaceSize=75931K, -XX:CompressedClassSpaceSize=15326K, -Xss1M * 300 threads)
2017-10-18T19:21:35.06+0200 [APP/PROC/WEB/0] OUT Exit status 1
2017-10-18T19:21:35.06+0200 [CELL/SSHD/0] OUT Exit status 0
2017-10-18T19:21:35.07+0200 [CELL/0] OUT Stopping instance a4bccbf9-6da5-46a0-5af3-2dae
```

10. Fix the error and retry to start the application once again.
11. Execute the application one more time (via a browser)
12. Using the Web console, check the applications status, used resources and monitored performances.
13. Scale up (respectively down) the application and try to investigate the impact of such operations on its behavior, performance, etc.
    Tip: Do not hesitate to execute it several times to that end.
14. Bind the application with a technical service from CF marketplace. There are few free services there that you can use such as basic SQL storage service.

Task 2.3 Provisioning HelloWorld Servlet on Jelastic

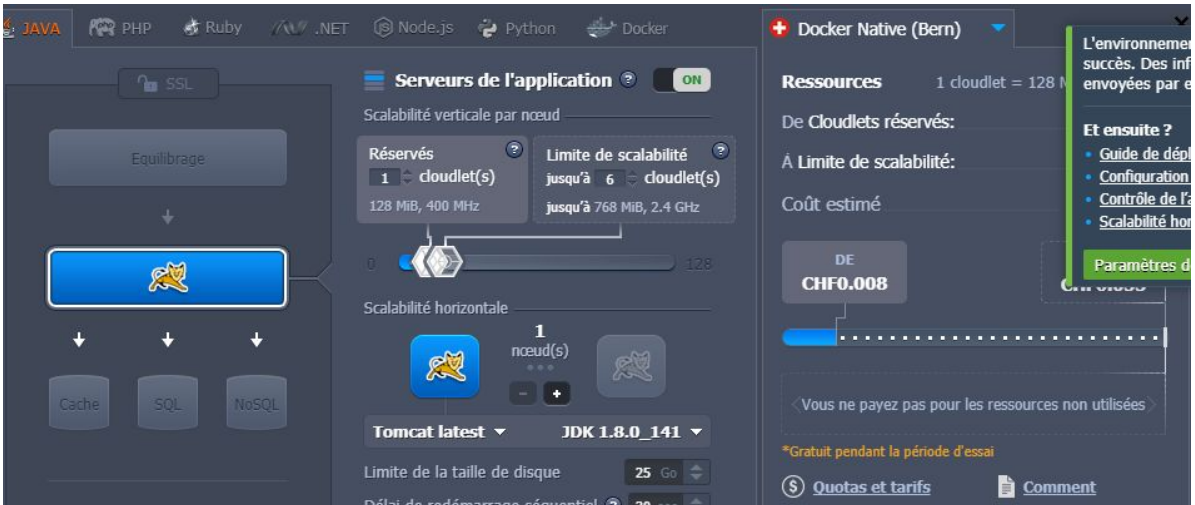
The goal of this subtask is to provision the same HelloWorld Servlet on Jelastic PaaS. To that end, we are going to use the public instance deployed over Hidora, the Swiss infrastructure provider accessible through the following link.

For the development phase, you will reuse the HelloWorld.WAR under (\target); however, you will need to settle the hosting environment of your application by yourself. Indeed, unlike CF, Jelastic requires manual and explicit settings of the required PaaS resources in terms routers, service containers, storage services and so on.

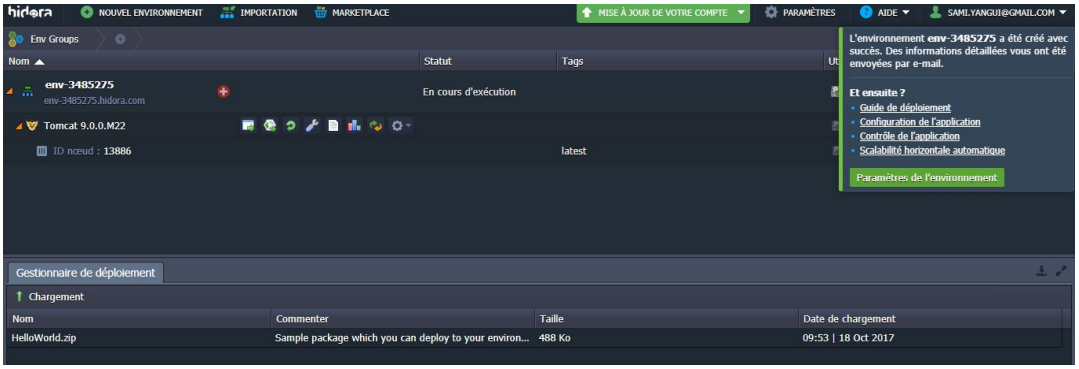
For the deployment phase, you will need to: (i) upload the application artifacts on Jelastic and, (ii) deploy it over the already created environment. Note that the concrete deployment in Jelastic is the binding between an application archives and a potential environment.

For the management phase, you will need to test the application execution.

1. Connect to Jelastic, select Hidora as IaaS provider and register to create a free trial account (Link)
2. Log in
3. Create the environment that you deem necessary to host and execute your HelloWorld Servlet



4. Upload your application artifacts



5. Deploy it
6. Execute it

Task 3 (Addressing Objective 3: Learn how to build a more complex cloud application, and the application architecture concepts/principles you will need to succeed)

Anyone who has built applications understands that applications that are specifically designed for the target platform will perform better and are more resilient and easier to manage. Designing applications for cloud platforms is no exception.

In the following, we discuss the key concepts to exactly understand how to go about designing and building optimal cloud applications. Note that simply migrating regular and legacy (complex) applications to the cloud as they are, although most of them are portable such as JEE applications, has led to poorly designed applications for cloud-based platforms, in the sense that they do not deliver the value on the cloud platform that enterprises expect.

Provisioning cloud applications relies on several architectural principles that mixes traditional software development concepts and reviews what is new with the cloud. It is a collection of best practices, concepts, and procedures for success:

- **Design the application as a collection of (micro) services:** Cloud applications are best deployed as a collection of cloud services, or APIs. You build up from the data to the services and then combine those services into composite services or complete composite applications (either orchestration or choreography). **This is service-oriented architecture, at its essence.** Designing tightly coupled applications that focus on the user interfaces or the control, rather than exposing the underlying business functions as services will not do the job. Separate the application components physically helps on tracking and auditing the many services that make up your application. Each service (component) to be executed on the proper machine instances, service containers, service/API managers and/or governance technology. Another advantage is the service reuse from other applications or more coarse-grained services. You can break up applications into several underlying services that have value when used by other applications.

- **Decouple the data:** If you tightly couple the data to the application, it will not find a good home in the cloud. Clouds are complex distributed systems that work best with application architectures that break out processing and data into separate components. For instance, you need to reconsider the example we developed as part as Lecture 3 where a given enterprise needs to execute compute-extensive software

at cloud and keeps critical data at home. However, you should be aware about the impact on the performance. Databases are reachable via Internet and that would inevitably cause latency. Specifically, database communications may determine how close your data sits to the services and applications that need to leverage it. To address that, you might consider using the huge variety of caching systems that cloud providers offer nowadays. These provide additional database performance by locally storing commonly accessed data, thereby reducing all database read requests back to the physical database.

- **Consider communications between the application components:** Chatty application components that constantly communicate with each other will lower the performance of the overall (composite) application, given that they are typically distributed over a network or the open Internet, where tolerance for high latency is desirable. You need to optimize communications between application components. For example, combine communications into a single stream of data or a group of messages, chatty application components that constantly interact with each other will lower the performance of the overall application.

- **Model, design and code for performance and scaling:** Extend considerations around how application components communicate to include overall performance as well. This includes understanding how (and when?) the application will scale up (respectively down) under an increasing (respectively decreasing) load. For instance, if 1000 or more end users log on and execute at the same time, how will (would) your application behave? Such a spike will trigger an increased traffic on the network, the increased load on the application servers, and the load placed on the back-end databases. This example might increase the load on the application servers by 80%, the load on the network by 10%, and the load on the database by 40% to 50%. Given that, adding 1000 more end users will likely saturate the application servers you have provisioned, and, consequently, you will need to spin up more application server instances. On the other hand, the network capacity might remain the same while the number of database instances may have to increase to handle the additional load. Armed with this model, you can figure out how best to scale up/down the application by automatically spinning up/down resource instances that are needed.Of course, nearly all the cloud providers offer auto-scaling capabilities for applications and resources nowadays; however, the most efficient path lies in understanding the application's workload profile and defining the path to scaling the application, as well as, putting mechanisms in place to ensure that it will, indeed, scale. To that end, you can benefit from the several technical services provided by the cloud providers such as monitoring and logging services. For instance, monitor overall application performance and create interfaces within the application to better enable performance monitoring will help to optimize your application scalability management. However, keep in mind that the application (un)provisions resources procedures should be innate to the application rather than the cloud platform.

- **Make security systemic within the application**: When hosting an application in the cloud, **security is NOT an afterthought**. Your application architecture should make security systemic to the application. Pick a security approach and technology prior to building your application that will be effective for the type of application you are running and that will address any compliance or other data-level security issues. You might reconsider the ambulatory healthcare application presented as part as Lecture 5 where applications handle and store personal vital data. It is clear that some security rules related to the patients' identity and privacy needs to be addressed

before going to the cloud. Generally speaking, cloud applications should leverage Identity and Access Management (IAM). Enterprises that develop mature IAM capabilities can reduce their security costs and, more importantly, become significantly more agile at configuring security for cloud applications. What is more, the use of IAM within cloud application provisioning will backfill into the enterprise, as these organizations modernize security approaches and technologies to align with the use of the cloud. In many cases, IAM will be provided as-a-Service to the developer/enterprise. This concept of cloud-delivered IAM quickly leads to the concept of centralized identity management. As you build more cloud applications using IAM, each application should become significantly more secure and more cost-effective. Your core objective is to design security into your application and take advantage of the native features of both the cloud and the IAM systems you use. On other hand, you need also to keep in mind that each application has its own requirements based upon the needs of the business. Security concept, requirement and even understanding often differs from one enterprise to another.

Lessons Learned (important): Building a cloud-ready application architecture requires that you pay attention to a few new things, but many of the traditional concepts are still important, such as sound design, testing, and learning from your mistakes. Most developers who provision applications on private or public cloud platforms will make some blunders, but as long as they recognize, correct, and learn from those mistakes, they will be well on their way to finding a more effective path to building applications in the cloud.

Understand that approaches such as service orientation should be given priority, even if it means longer initial application development lifecycles and bigger budgets. Even though you will probably pay more for application development in the cloud than you did for traditional application development, the investment in services pays huge dividends year in and year out. It is a smart investment!

Task 4 (Addressing Objective 4)

1. Design and develop a simple MVC application with database access according to the tips you learned from Task 3. You need to select and motivate the choice of the storage services you selected (SQL, No-SQL, Big Data support, etc.).
2. Provision the application on a PaaS of your choice.
3. Push the code on a Git Repo and provide us with the required releases.

Task 5 (Addressing Objective 5)

1. Design and make up your own PaaS over the Proxmox infrastructure. For validation purpose, you have two alternatives:
    a. Either you provision your Resources Generator and Manager applications (developed as part as Lab 2) over that PaaS.
    b. Or, you provision the MVC application you developed as part of Task 4 of this lab.

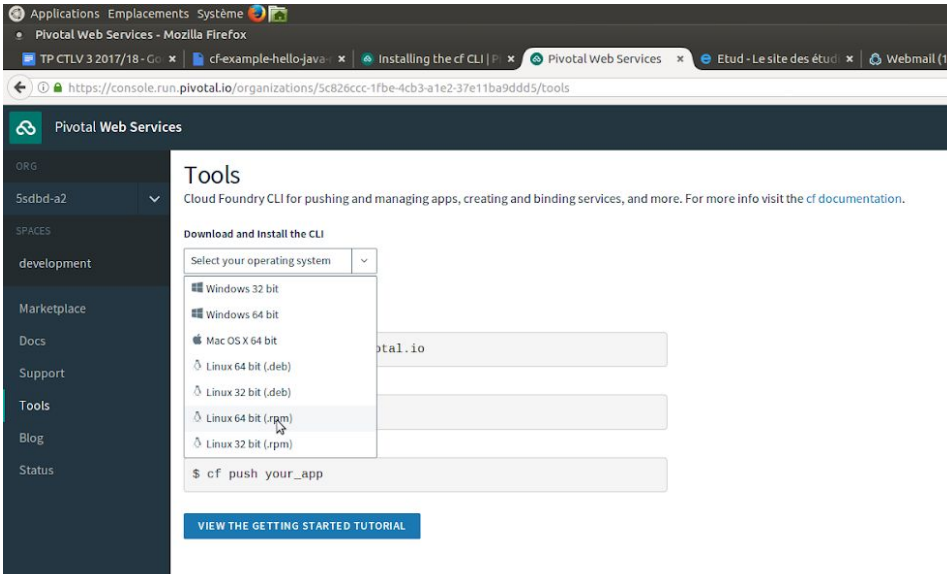
You might use a specific collection of LXC nodes to host and execute your applications while the rest as used as regular resources (to be managed). You might install and/or use remote services and/or APIs for storage.

References

1. H. Berndt, P. Graubmann, M. Wakano. “Service specification concepts in TINA-C”. In: Kugler HJ., Mullery A., Niebert N. (eds) Towards a Pan-European Telecommunication Service Infrastructure — IS&N '94. IS&N 1994. Lecture Notes in Computer Science, vol 851. Springer, Berlin, Heidelberg
2. M. Jacobs, P. Leydekkers. “Specification of Synchronization in Multimedia Conferencing Services Using the TINA Lifecycle Model,” Distrib. Sys. Eng., vol. 3, 1996, pp. 185–96.
3. S. Yangui, R.H. Glitho and C. Wette. “Approaches to end-user applications portability in the cloud: A survey”. IEEE Communications Magazine, vol. 54 No. 7, 2016, pp. 138-145.

Appendix 1

1. Download the 64bit.rpm client distribution from the CF Web console.



2. Extract the file and place your prompt under usr/bin to be able to execute cf.