Stage 2: Product Development

The assessment for Cloud Computing is a team project. You will work in teams of four (unless otherwise advised), to develop a Platform-as-a-Service (PaaS) on top of an existing Infrastructure-as-a-Service (Iaas). The goal is to offer a platform, hosting a mix of micro-services and applications ("apps"). You will seek to form an ecosystem with other teams who play the role of Independent Software Vendors (ISVs).

In the next four weeks, your team will develop a full implementation of the ideas presented in your product pitch. This will be a cloud platform with two micro-services (the single sign-in service and the "peanut bank" payment service) and a set of apps that connect to your micro-services. *Your team will produce one app or micro-service per team-member*. Your team may also seek to attract apps from other ISV teams *that are consistent with your platform theme*.

The deliverable for this stage is a **group final report** describing your solution, and a **software system**, both due at the **end of week 10** (**one paper copy** of the report to the hand-in box; **one public URL** for the platform advertised **in the report** and **via the discussion forum**; **one backup copy** of the report and **one archived copy of the software** to MOLE, by 3pm Friday). It will demonstrate the complete working of the platform and of the apps on the platform, with examples. This assignment stage is weighted 65%.

In **week 11**, following this submission deadline, **stage 3 of the project will involve individual testing** of another team's platform and apps. This stage is weighted 10%. Details will be announced.

Sheffield Cloudbase Concept

[Elaborated from the Product Pitch]. The goal is to develop a Platform-as-a-Service aimed at Sheffield students, called "Sheffield Cloudbase", which is capable of hosting software apps developed initially by your team and eventually by other ISV teams. The platform will be hosted on the application server (Apache Tomcat) on your team's OpenNebula Virtual Machine, and it will be accessible through any web browser (e.g. Chrome, Safari, Explorer, or Firefox) on campus.

What is Sheffield Cloudbase? It is a virtual desktop offering useful apps to students. As an analogy, think of any tablet computer or mobile phone, with its attractive display of icons representing the various apps that can be launched. Some of these are standalone services; others may "consume" further apps on the same platform (typically, micro-services shared by many apps). The main guiding principle is that the apps you develop should be useful to Sheffield students, who are your target customers.

Each team will seek to provide: a PaaS with **two** micro-services (a single sign-in service; and a charging and payment service known as the "peanut bank"); and **two** or more software apps (which will follow the theme of your platform's business pitch). *Each team member will be individually responsible for one of the micro-services or apps.* You should come to group decisions about the

internal APIs to link apps and micro-services, and then should be able to deliver your individual app or micro-service in the later phase. It is permitted for the whole class to share and discuss general technical issues, or help to solve technical problems, via the MOLE discussion group forum (in the manner of *Stack Overflow*).

Platform Requirements

The Sheffield Cloudbase platform will observe some fairly standard design guidelines. Any Platform-as-a-Service is supposed to provide added value through the "micro-services" offered on the platform, such as **a single point of sign-in** (for security), or a **charging and payment** service (for metering app usage, charging app users and paying credits to the app developers). The following requirements should be observed:

- The platform should be designed to operate across all major browsers in a device-independent (tablet, PC, smartphone) fashion. This means you should use a responsive web design template, such as that offered by the w3-css stylesheet (see: http://www.w3schools.com/w3css/).
- Web page layouts should offer a standard look-and-feel, adopting HTML5 design guidelines, using semantic elements to describe document structure, such as header, footer, section, summary, details, etc. (see: http://www.w3schools.com/html/html5_semantic_elements.asp).
- The platform must provide business logic, through which apps may be mounted and executed on the platform, in the same display environment. For example, the business logic could be Java servlets and the display environment could be JSP page templates, or XHTML templates (XML-compliant HTML) that display as HTML (see later lectures).
- The platform designers must be able to supply a specification for app developers to follow that will allow apps to run on the platform. This should describe what API an app should have, in order to link with the platform's micro-services, and any other necessary meta-data, for example, a standard sized icon to display on the dashboard to represent the app.
- The platform designers must supply a way for apps to be uploaded and installed on their platform, so that they can be consumed by end-users, and possibly also configured to work with other apps on the same platform. Apps should typically be uploaded as packaged Java archives (JAR files) or Web archives (WAR files).
- The platform should supply a **single sign-in micro-service** that acts as the single point of authentication for all apps on the platform. This should grant authentication and authorisation tokens for the use of all other apps on the platform. Other apps can only be launched once the user has been authenticated and connected to the charging and payment micro-service so that his/her usage may be metered.
- The platform should supply a **charging and payment micro-service**, known as the "peanut bank", which keeps track of all user-accounts and their digital credit. Every user will have a default credit allocation when they first sign up and join the platform. The digital currency will be known as "the peanut" (what currency symbol?) *Users who consume services will be*

charged peanuts; and providers who offer services will be credited peanuts by the bank. ISV app providers should also be able to earn peanut credits on your platform.

- The charging and payment model should be fair and equitable for all providers and consumers, including the platform providers. The charging model should be *per-use of each app* (not one-time purchase; not per-monthly period). Compound services (built of other services) should charge *pro rata* more and if offered by several providers, each of these should receive a fair distribution of the *peanut* revenue.
- The platform should allow apps to read and write their necessary app data to disk, but may seek to control where the files for particular apps are kept. The Tomcat application server already offers a common directory structure for each app, under the platform root directory. Optionally, a platform may choose to impose security controls on the volume of data stored, or charge more peanuts for excessive use of storage.

Software Requirements

Each team will develop **two micro-services** and **two apps** (assuming four team members; larger teams will develop more apps). Each team member will *individually develop one service that is their own technical contribution* to the offerings on the team's platform. You should seek to ensure that the apps provided on each platform are sufficiently different. Your platform may also attract apps from other teams (acting as ISVs). The following requirements should be observed:

- The **single sign-in micro-service** must only be triggered when a user first attempts to launch any app or micro-service. This is to ensure that all apps communicate properly with the single sign-in micro service, to authenticate a user and bind him/her to an account in the peanut bank. Put another way, the user *should not be presented with a monolithic login-page* before being allowed to proceed to view the platform. The user should be able to view the platform and browse the different app offerings, before having to sign in.
- On activation, the **single sign-in micro-service** will offer a login form for existing users, with a button to navigate to an extended form to sign up a new user, with a new password. User accounts should be created using your own unique university email IDs (just the name-part, not the whole address) as the login ID, for example: *JBSmith197* or *AMJones62*. You may choose your own passwords. Once signed in, you should be redirected to whatever app or service you were about to use.
- Before you are signed in to the platform, you should be able to browse its unsecured features through the dashboard, e.g. describing the theme, or the apps available. Once you are signed into the platform, *the appearance of the dashboard should change to indicate that you are signed in, typically by showing extra top-level menu options that appear allowing you to view your peanut bank account, or sign out*.
- On first use of the single sign-in, a new user will be granted an account in the **charging and payment micro-service** (the "peanut bank"). They will receive a standard free allocation of 1000 peanuts, which they may "spend" when consuming services; or "earn" when providing services to other users. A signed-in user should be able to view their peanut bank account at

any time, and view a history of transactions against their account. A typical charge for using an app should be around 5 peanuts per use. You must define what "one use" means (one click; or one session). The peanut-bank only manages credits on one platform and is not expected to transfer credits across different platforms.

- Each **platform or ISV app** will connect with the platform's **single sign-in micro-service**, such that it cannot be operated independently until the app consumer (end-user) has signed in to the platform, and so identified him/herself to the app as a consumer to be charged for app usage. *Sign-in should happen once per session that the user is logged into the platform, and not per app usage* (see the lecture on sessions). If a user is already signed-in for the current session, they should be granted direct access to the apps.
- Each **platform or ISV app** will connect with the platform's **charging and payment micro-service**, such that *the app consumer's account may have peanuts debited, and the app provider's account may have peanuts credited*. A proportion of the charge for using an app should be paid to the platform (to reward micro-service providers). E.g. an app could charge 5 peanuts, credit the app provider with 3 peanuts, and credit the platform with 2 peanuts.
- While the micro-services do not earn revenue in the same way as apps, a proportion of the charge for using an app should be distributed evenly to the providers of these two micro-services. E.g. when the 2-peanut tax is levied for using the platform, 1 peanut should go to each of the micro-service providers, who will also have "peanut bank" accounts.
- Apps will offer an API (application programming interface) to other potential apps that might consume them. This could happen in two ways: if all apps run as separate Tomcat web-services, they will typically communicate via HTTP; but if apps are individual servlets running as a single Tomcat web-service, they may invoke each other directly through a Java interface with agreed standard names for methods, e.g. *start(), stop().* The former is more likely, given that this is the best way to bundle each app.
- Apps must be supplied in some standard format for uploading and deploying on a platform. There should be an agreed format for packaging up the app and all its meta-data. This will include the executable code, any template display screens (e.g. XML files), any visual icons used to represent the app on the platform. This will be a standard directory structure that is zipped (typically as a WAR: JEE Web archive) for uploading.
- Apps must be able to be unpacked and opened inside the platform's container for apps. This means that the platform should be able to unpack executable files, meta-data and icon images, put these in a suitable place, create a button for the app using the icon image, and create a link to launch the app. Standard Tomcat uploading may be able to deploy a WAR file ready for execution (but you will manage the appearance of the dashboard).

Ideas for Applications

Applications should be student-centric, oriented towards the needs of Sheffield students. Here you can be quite creative in coming up with apps that you think the average student would find useful on the platform. You can probably come up with better ideas than these suggestions:

- Lecture theatre app: how to find a particular lecture theatre anywhere on the Sheffield campus. How to get there if the normal access route is blocked by building work.
- Restaurant guide app: a dynamically-updated record of the best affordable places for students to eat in Sheffield. Restaurants could be ranked by their cost and quality, or with student recommendations.
- Assignment reminder app: a kind of calendar that reminds a student what they should be doing in the next few days, given that assignments may be due in a week or so. It should help to organise their time most effectively.

The main thing is to decide on a consistent "business proposition" for your platform. Is it themed around student academic support? Is it themed around student lifestyle and daily life? Is it themed around leisure and entertainment? Platforms with consistent themes will gain higher credit.

Cloud Ecosystems

This project may be completed to a good standard by meeting your team objectives. However, an extra stretch-goal will be to form software ecosystems, attracting custom apps from other ISV teams that "fit" with your platform's business proposition. You will need to have good communication between platform builders and app developers in the collaborating teams. For this, we have the MOLE discussion forum. You should use this to advertise your offering to the rest of the community. Here are important things to announce:

- what provide-interface an app will supply (what API it offers to its clients); and what require-interface the app needs (what API it expects platform micro-services to offer);
- what information bundle should be provided with an app (e.g. images); what size and style of image icons are needed; what configuration files (typically XML);
- what zipped directory structure should be used for bundling apps in a way that can be unbundled at the other end by the container.

The expectation is that certain clusters of platforms and app developers will emerge; and this kind of collaboration is explicitly desired as part of the project. This is what is referred to as a "cloud ecosystem". The ISVs aim to become the business partners of the platform provider.

Stage 2 Deliverable - Final Report

Your team should provide a **group report** summarising its achievement by the end of week 10. This should be 12-25 (min/max) sides long, excluding content index and references. It should contain the following information, which will be used in the marking scheme:

Business

What flavour of platform and apps you are offering (whether for entertainment, academic support, student life), what custom apps you were able to solicit (if any), and with which cloud ecosystem (possibly several ISV teams) your team is collaborating.

How well your efforts were synchronised within (and optionally, across collaborating) teams, what kinds of development standards, collaborative technology and group-working you set up, any disagreements you had and how you resolved them.

Design

What kind of web technology (e.g. JSP or XML templates for presentation?) and software stack (e.g. servlet architecture; XML or MySQL database?) your team eventually used to realise its products; whether this was the same, or different from your interim plan; and give reasons if you changed your mind about this.

What risk analysis you did, and what kind of fall-back options you had, and whether you used any of these, if the primary choice of technology did not work.

Platforms: what kind of platform architecture you are using (with a diagram). Use a **UML package diagram** (subsystem flavour) to show layered *software architecture*, and a **UML deployment diagram** to show client, server and container configuration in your *systems architecture*. Use a **UML class diagram** to illustrate the design of your *software components* and any plug-in *interface dependencies* (the latter may optionally be illustrated with a **UML component diagram** instead).

Apps: what kind of zipped directory structure you are using to store your apps and related meta-data, showing what kinds of file object you expect to find in standard locations. Describe what processes are needed to pack, unpack, install and invoke the apps.

Technical

You should clearly advertise the **URL of your team's platform** on OpenNebula so that project-markers can access your software system as described in your report.

Platforms: show off the major features of your platform, using suitable screenshots and evidence of any changes to back-end data stores while executing its main functions. You should aim to demonstrate the following platform features:

- Single point of sign-in for the platform-as-a-service (sign-in triggered by first use of any app; thereafter unrestricted access is granted to all apps for the user-session) and linkage to the user's "peanut bank" account;
- Creation of accounts with initial credit in the "peanut bank"; and suitable maintenance of accounts for both consumers and providers operating on the platform (show that credit is transferred from consumer to the relevant provider, or providers)
- Ability to upload, check and install new apps, showing whether this happens while the platform is live; or whether it requires a restart of the platform. Show whether you have a means of validating the app before it is deployed to consumers.

Apps and micro-services: give an account of each app/micro-service developed by each individual member of the team, describing how it fits with the business proposition for the platform, and show the following features:

- Examples of app screens using the suggested HTML5 and responsive-design stylesheets, for the main screens that consumers will see during normal usage.
- Evidence that the app executes and connects with the sign-in service and the “peanut bank” in order to deduct credit from the consumer and award credit to the app producer.
- Evidence that each of the micro-services executes and is consumed correctly by other applications, which cannot execute without connecting to them.
- If the app needs to connect to some back-end data storage, evidence that this data store is accessed and updated in suitable ways.

Security: show how you have addressed other standard concerns, such as handling loading issues (scaling, servlet pools, DB connection pools). Show how you have addressed security issues such as authentication (who you are) and authorisation (what you can do).

Finally: describe anything else of merit that you think should be reported.

Teamwork

Describe briefly which *development tasks* your team members actually carried out for the final stage. This should describe in a table what parts of the platform were built by which the team members; and how the different design or code artefacts were checked.

Describe in another table *which of the apps/micro-services* were developed by each team member. (In a standard team of four, there should be four apps; teams of a different size may have been given different per-person goals).

All team members must also sign an agreement describing what *share of effort* each of the team has put in (where 100% is an equal share; the total effort should add up to 100 * number of team members, typically four).