Using Public and Free Platform-as-a-Service (PaaS) based Lightweight Projects for Software Architecture Education

Zheng Li
University of Concepción
Concepción, Chile
imlizheng@gmail.com

ABSTRACT

Background: Software architecture is a crucial while significantly challenging topic in the computer science curriculum. Although "learning by doing" is widely advocated to address this topic's abstract and fuzzy nature, educators are still facing various difficulties in practice, especially including students' vicious circle of inexperience and the mental model dilemma in experiential learning.
Aims: To help break the aforementioned vicious circle and mental model dilemma, this work aims to investigate our educational strategy of using lightweight projects with public and free PaaS resources (1) to help students accumulate architectural experience from the early stage and (2) to facilitate strengthening students' fundamental architecture knowledge.
Method: To collect more empirical evidence, we conducted action research on our educational strategy across three undergraduate-curriculum courses for two years. In particular, we employed the Technology Acceptance Model (TAM) to validate how well students received such an educational strategy.
Results: The students involved in the relevant coursework generally gave positive feedback with respect to learning theoretical concepts and a set of architectural patterns. Meanwhile, we also observed diverse learning curves for utilizing PaaS resources.
Conclusions: Although there still exist PaaS-related limits, our educational strategy can foster students' experiential learning and collaborative learning in fundamental architecture courses. We believe that there are also potentials to extend our work to broader and advanced courses of software architecture.

CCS CONCEPTS

• **Social and professional topics** → **Software engineering education**; • **Software and its engineering** → **Software architectures**; • **Applied computing** → *Collaborative learning*.

KEYWORDS

collaborative learning, experiential learning, lightweight project, Platform-as-a-Service (PaaS), software architecture education

ACM Reference Format:
Zheng Li. 2020. Using Public and Free Platform-as-a-Service (PaaS) based Lightweight Projects for Software Architecture Education. In *Software Engineering Education and Training (ICSE-SEET'20), May 23–29, 2020, Seoul, Republic of Korea.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3377814.3381704

1 INTRODUCTION

Software architecture is unanimously considered to be the starting point for designing new systems, for understanding legacy systems, and especially for addressing system quality issues [13]. More importantly, by macroscopically controlling enormous complexity in sophisticated systems, software architecture plays a fundamental role in satisfying both functional and nonfunctional requirements, as well as an essential role in achieving business goals [7, 22]. Correspondingly, from the educator's perspective, all students in the software engineering discipline should be equipped with both theoretical and practical knowledge of software architecture [30]; and from the learner's/practitioner's perspective, software architecture is one of the top core topics in the software engineering body of knowledge [24].

In particular, we adopt one of the modern definitions of software architecture [8] in this paper:

> *"The software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them."*

Unfortunately, software architecture has been voted as one of the highly important topics that are not extensively taught [24]. Such an educational gap could be a result from the challenges in teaching software architecture. The mainly discussed challenges include:

- **Abstract and Fuzzy Nature of the Topic**: Software architecture has been recognized abstract and fuzzy in nature, with various definitions, principles, concepts and notions. Trying to offer full-scope knowledge of software architecture will not only require long teaching sessions, but also introduce unexpected confusions to students' understanding [12, 13].
- **Vicious Circle of Inexperience**: Software architecture encompasses a set of decisions that have different impacts on a whole system's behaviors, and the decision making in software architecture generally requires a high level of experience [12]; while university students, including graduate students, usually have few chances to accumulate experience by dealing with complex high-level design tasks for real-world architectural problems [13, 26].

- **Mental Model Dilemma in Experiential Learning**: Given the abstract and fuzzy nature of this topic, it is difficult for students to learn software architecture purely from text books and lectures [13]. Thus, making hands dirty rather than sitting within classrooms will be particularly crucial for studying this topic [20, 23]. However, even without considering the aforementioned vicious circle of inexperience, architecture problems are usually "wicked problems" that barely have clear descriptions, definite formulations, and even ending criteria [11]. Since students are generally used to seeking clean solutions to well-defined problems, wicked and open-ended architecture practices can lead to further confusions for students [13, 26].

We believe that these challenges can and must be addressed gradually rather than all at once. For example, to relieve the Mental Model Dilemma in Experiential Learning, we should solid students' understanding of the abstract and fundamental architecture knowledge; to break the Vicious Circle of Inexperience, we can let students start from practicing small to medium software systems along with the theoretical study. After all, students need to be equipped with enough basic knowledge (concepts, principles, patterns, etc.) before being able to design suitable architectures for complex systems [26] and/or to make trade-off decisions in quality concerns of different stakeholders [11].

When it comes to the practice resources for student projects, mainly driven by the administration and financial concerns, we advocate employing the free quotas of Platform-as-a-Service (PaaS) from the public cloud market. In our two-year study, we have successfully fostered collaborative and experiential learning of various theoretical concepts and architectural patterns across three undergraduate courses, through lightweight student projects based on publicly available and free PaaS resources. In this paper we share the experience and lessons learned from our study, which makes a threefold major contribution:

- In theory, our work justifies the value and efficiency of small- to medium-size student projects in software architecture education. Although advanced courses like architectural design generally require more complex software systems [26], we have observed that lightweight systems can better connect students' programming skills to the fundamental architectural knowledge, and can then both increase students' sense of achievement and enhance their confidence in attending advanced courses.
- In practice, the student project details (together with example project reports) shared in this paper can directly be reused as teaching materials for the same or broader architectural topics of interest (e.g., system scalability or fault tolerance). In particular, considering that developing effective tasks and instrumentation is critical as well as challenging in software architecture education [12], we have briefly highlighted the project tasks in each of our action research iterations.
- For saving budgets, our work demonstrates that the free quotas of public PaaS are satisfying and supportive for students to learn software architecture collaboratively and experientially, at least at the bachelor-curriculum level. It is worth mentioning that, in addition to the generic benefits of cloud computing in education, employing those PaaS resources further helped us cut the monetary and administration costs down to zero in our study. Moreover, educators can expect richer available PaaS services as time goes by, because more and more public providers are using free offers to try to obtain future business opportunities [35].

The remainder of this paper is organized as follows. Section 2 reviews the related work on employing cloud computing to facilitate higher education. Section 3 specifies the research methodology that drives our study. Section 4 introduces our action research details about teaching five distributed architectural patterns. The lessons we have learned by far are summarized in Section 5, including both the benefits and the limits of our educational strategy. Conclusions and some future work are drawn in Section 6.

2 RELATED WORK

Our study essentially belongs to the broad research area of applying cloud computing to education. Employing cloud computing in education has been increasingly investigated since the early stage of this new computing paradigm [35]. Driven by the advantage of economies of scale from the utility's perspective, private educational clouds emerged mainly for computing resource-intensive courses (e.g., data mining [21]), infrastructural resource sharing/reusing [36], and/or better controllability and management. In practice, universities in many countries have successfully cut down their IT expenses and administration costs either by individually virtualizing their datacenters into elastic resource pools (e.g., Florida Atlantic University [28]) or by jointly building large-scale collaboration platforms (e.g., the Virtual Computing Lab among four Virginia colleges [37] and the NeCTAR Cloud across Australia's research community [25]). However, it is clear that developing private cloud still requires considerable investment especially at the beginning.

Along with the development of cloud computing, public cloud providers started offering education-dedicated products and services, such as Microsoft Education Cloud[1], IBM Cloud Academy[2], Amazon Education[3], etc. Nevertheless, these education-dedicated cloud resources could either incur unexpected costs [3] or require extra administration efforts. For example, according to our signed agreement with Amazon, we need to predefine requirements and submit students' basic information every semester for utilizing their educational resources.

In the meantime, educators also explored generic-purpose public cloud services to try to reduce the influences of the financial crisis, while most of the reports were focused on Software-as-a-Service (SaaS) [27, 35]. Various cloud SaaS apps like Microsoft Office 365 or Google Docs and Spreadsheets have been widely advocated and adopted both to support teachers in lecture preparations and to help students access learning materials [3].

Using PaaS in education is particularly discussed in [15], and the values are mainly recognized in programming courses. For example, benefiting from PaaS's ready-to-use development and runtime environment, students can emphasize improving their programming

[1]https://www.microsoft.com/en-gb/education/cloud.aspx
[2]https://www.research.ibm.com/university/cloudacademy/
[3]https://aws.amazon.com/education/awseducate/

skills, without spending extra time and efforts on understanding irrelevant information and configuring the underlying technology stack. Our work essentially extends the existing discussions, by arguing that PaaS can further help students study abstract concepts and foster higher-order thinking through the way of "learning by doing". Note that the previously discussed benefits of PaaS are also valid in our work in terms of easing the "doing", i.e. facilitating student project implementations.

In particular, unlike the existing claims about using large-scale and even real-world systems for comprehensive architectural education [12, 13], we argue that lightweight projects are useful and helpful for experientially learning fundamental knowledge of software architecture, and public PaaS's free quotas are satisfying enough for students to implement small to medium projects. In fact, by focusing on distributed software scenarios, we find it convenient and flexible to teach architectural concepts and patterns with the publicly available PaaS resources, as we do not have to go through any administration process or maintain a monolithic lab, no matter physically or virtually. To the best of our knowledge, our work is the first study on applying the free quotas of public PaaS to the software architecture education.

3 RESEARCH METHODOLOGY

Our work was originally driven by two research questions. Firstly, considering the aforementioned difficulties and challenges in involving industrial systems, we were wondering:

RQ1: Can students benefit from lightweight projects for collaborative and experiential learning of software architecture, at least with respect to the fundamental knowledge?

Secondly, since we decided to conduct an extensive study across multiple courses, to minimize the budgetary cost and administration overhead, we were wondering:

RQ2: Can public and free PaaS resources satisfy students' collaborative and experiential learning of software architecture, at least with respect to lightweight projects?

To pursue answers, we proposed an educational strategy of using public and free PaaS based lightweight projects for software architecture education, and performed an *action research* [34] based study on this strategy across three bachelor-curriculum courses during the past two years, such as cloud computing (a couple of lectures on cloud-native software engineering), software architecture fundamentals, and microservices architecture. The iterative process of our action research (cf. Figure 1) was mapped with a rough evolution path of software architecture, and each iteration (including Planning, Action, Observation and Evaluation) addressed one concrete architectural pattern, so as to incrementally cover a systematic knowledge body and eventually deliver relatively generic answers to those research questions.

Note that our action research iterations are based on student projects instead of semester/course cycles. Thus, we can not only perform multiple iterations but also repeat some iteration multiple rounds (we name them *shadow iterations*) within one semester, by sharing suitable student projects and relevant software architecture knowledge among different courses. It is also noteworthy that the shadow iterations have also provided us with useful insights and

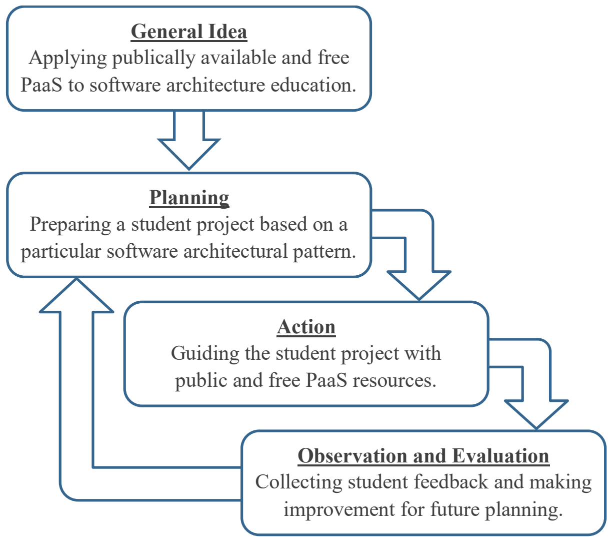
General Idea
Applying publically available and free PaaS to software architecture education.
Planning
Preparing a student project based on a particular software architectural pattern.
Action
Guiding the student project with public and free PaaS resources.
Observation and Evaluation
Collecting student feedback and making improvement for future planning.

Figure 1: The iteration process of our action research based study.

lessons (e.g., the same student project shared by different courses could still need different prerequisite materials).

In each iteration, the step Planning defines generic project tasks with respect to a specific architectural pattern. The step Action indicates the teaching activities of project assignment and supervision, rather than the student activities of project implementation. At the step Observation and Evaluation, we have employed the well-known Technology Acceptance Model (TAM) [10] to help reflect how well students received our educational strategy in terms of:

- **Perceived Usefulness**, indicating to what extent the students believe that they have advanced their understanding of an architectural pattern by implementing a lightweight PaaS-based software project.
- **Perceived Ease of Use**, indicating to what extent the students believe that they have implemented the lightweight PaaS-based software project without much difficulty or great effort.

As for the practice resources in student projects, in fact, at the time of writing we have identified more free and public PaaS services (e.g., Pivotal Web Services[4]). To stick to our existing work, however, this paper only includes the PaaS services that have been employed in multiple courses and semesters, such as:

- **Google AppEngine (GAE).**[5] Without necessarily upgrading to billing accounts, the free standard environment of GAE includes 28 basic (128MB memory limit) frontend instance-hours per day as well as 1 GB of data storage in total.
- **IBM Cloud Lite.**[6] From November 2017 on, IBM offers its never-expiring and free account type with 256 MB of instantaneous cloud Foundry runtime memory, plus usage-capped

[4]https://run.pivotal.io/
[5]https://cloud.google.com/appengine/
[6]https://www.ibm.com/cloud/lite-account/

access to selected services like Internet of Things Platform and Data Science Experience [5].

- **Dropbox**.[7] The basic account of Dropbox offers 2GB of data storage for free, and its well-documented APIs make Dropbox a programmable platform that can be integrated into web, Android, or iOS apps.

4 FIVE ITERATIONS IN OUR ACTION RESEARCH BASED STUDY

Since distributed computing is increasingly gaining popularity and significance in the software industry and education [14], we were particularly concerned with teaching and learning a set of distributed architectural patterns, including the Client-Server pattern, the Master-Worker pattern, the Broker pattern, the Model-View-Controller (MVC) pattern, and the Microservices-Architecture (MSA) pattern.

4.1 The First Iteration for the Client-Server Pattern

In our courses, the Client-Server pattern was taught as the simplest form of distributed (i.e. two-tier) software architecture. Given the natural-language description of an application, students were required to implement the application system locally, and then break the whole system into two parts with a client role and a server role respectively.

4.1.1 *Planning.*

To speed up the first iteration, we only planned individual-student projects with simple applications. We further constrained the implementation to be a single pair of client module and server module, without considering the scenario of accessing one server by multiple clients.

The generic tasks for students to practice the Client-Server pattern are defined as follows:

1) Implement a local-version system for the given application to get familiar with the business logic and the functional workflow.
2) Break the system into two functional modules, keep the client module locally, and deploy the server module to GAE or IBM Cloud Lite.
3) Make both the local-version system and the client-server system work properly with consistent logic. Note that code efficiency is not a concern, as long as two systems' functional workflows are comparable.
4) Set timers to both the local-version system and the client-server system to measure their wall-clock latency of performing the same job.
5) Conduct performance (latency) comparison between the local-version system and the client-server system, and analyze why their performances are different.

4.1.2 *Action.*

Benefiting from the simplicity, the Client-Server pattern was most widely practiced by our students in different shadow iterations with diverse applications. The generic implementation environment of the student projects can be illustrated as shown in Figure 2.

[7] https://www.dropbox.com/developers/documentation

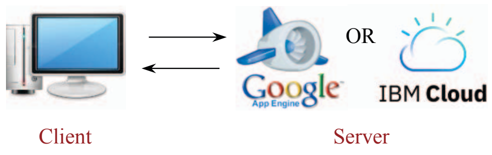

Figure 2: Project environment for students to practice the Client-Server pattern.

A traditional project assigned in class is to develop a student lookup system. The server side maintains the basic information of a group of students, while the client module issues queries to operate students' information via their IDs.

In many other student projects, the server part acts as a stateless functional component. The typical project applications are solving random sampling problems (e.g., Monte Carlo approximation of Pi), and the server components are then in charge of the sampling workloads. A particularly creative project is a video streaming system that has been used as a mobile micro-benchmarking tool, for checking App performance variation caused by different server locations and data sizes.[8]

4.1.3 *Observation and Evaluation.*

All the students succeeded in practicing the Client-Server pattern. It is worth mentioning that the relationship between software architecture and application performance in this case was well recognized/received by students after their practices.[9]

However, in different courses even the same project exhibited different degrees of difficulty. For the course of cloud computing, it is natural and convenient to supplement relevant knowledge of software architecture, and after that it is not difficult for students to work out their projects. On the contrary, it is hard to fit much cloud knowledge in the course of software architecture fundamentals, and thus students from this course usually have to take more efforts to accomplish the tasks. In particular, the aforementioned traditional project seems to be unexpectedly challenging for many students due to the dedicated cloud database techniques (e.g., cloud datastore [18]).

In addition, students' programming capacity also matters. It is clear that poor coding skills will negatively impact software development in general. In the worst case, some students could suffer from the lack of both the cloud knowledge and the programming experience at the same time. For example, we have received a student's feedback in such a situation:

> *"For me this Project was very interesting, and I learned much. (...) I don't study Informatics, but Industrial Engineering, therefore, it wasn't too easy for me. I hope I can keep up with the course in the following projects. (...) I didn't add much code, because I just followed the instructions and I installed (operated) a lot."*

[8] In this student project, the same video with different resolutions are used to vary data sizes, while different deployment regions of GAE [16] are used to vary server locations. For example, https://easymicro-2018.appspot.com/144 is the 144p video server deployed in São Paulo, and https://hidden-conquest-237802.appspot.com/720 is the 720p video server deployed in Tokyo.

[9] An example student report of the Client-Server project is shared online: http://bit.ly/3b324kx

4.1.4 Improvement.

Given the exposed issues in practicing the Client-Server pattern, we tried to both simplify the programming requirements and to ease the employment of PaaS services. For example, in the project of student lookup system, the data records are allowed to be hard-coded to avoid involving cloud databases at this iteration stage. Furthermore, to flatten the learning curve for utilizing PaaS resources, we have developed quickstarts of cloning functional modules to the PaaS platforms, as demonstrated in Appendix A.

4.2 The Second Iteration for the Master-Worker Pattern

The Master-Worker pattern was taught as a variant and an evolution of the Client-Server pattern, and also as a concrete Divide-and-Conquer (D&C) solution to complex computational jobs that involve similar or identical tasks. We particularly emphasized two software quality features (namely fault tolerance and scalability) that could largely benefit from the Master-Worker pattern.

4.2.1 Planning.

To enjoy more cloud resources (i.e. horizontal scaling), we intentionally constrained all the workers to be deployed in the cloud. Correspondingly, when explaining the evolution of the architectural patterns in this case, we clarified the naming conventions by changing the previous role of Client into Master and changing the role of Server into Worker.

The generic tasks for students to practice the Master-Worker pattern are defined as follows:

1) Prepare workers: Based on your previous client-server project, replicate more servers with the same functionality on GAE and/or IBM Cloud Lite. Note that those servers play the Worker role in this master-worker practice.
2) Prepare the master: Modify the client's functionality from your previous client-server project into a multi-thread program. Note that the client plays the Master role in this Master-Worker practice.
3) Let the master drive different amount of workers respectively to finish a particular fixed-size job, and set timers to measure their wall-clock latencies of performing the job.
4) Prepare the baseline: Modify the local-version application from your previous client-server project into a multi-thread application, and use timers to measure the wall-clock latency of performing the same job.
5) Observe the performance (latency) of different master-worker solutions against the baseline application, and use relevant knowledge to explain your observation.

4.2.2 Action.

In most cases, students could continue their client-server practices to learn the Master-Worker pattern as well as understanding the architectural evolution. Figure 3 shows the generic implementation environment in this iteration. Students were free to decide how many workers on which PaaS they would like to employ for their projects.

After explaining the D&C principle together with several examples, we also asked students to open minds and to freely identify D&C-friendly problems to solve. Not surprisingly, students' master-worker projects mostly dealt with Monte Carlo problems and parallel computing problems.

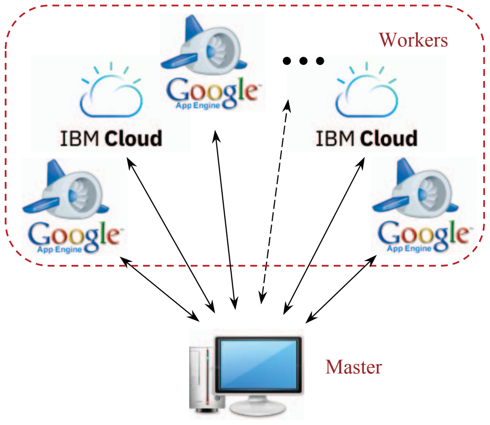

Figure 3: Project environment for students to practice the Master-Worker pattern.

4.2.3 Observation and Evaluation.

By playing with different number of workers, students generally reported that they had better understood: How the software-driven scaling matches the computing resource scaling [1], why scalability is an important quality feature of software systems, and how the Master-Worker pattern influences software performance even at the design time.[10]

In the meantime, many students noticed unexpected phenomena of the 500 Internal Server Error, especially when the workers were repeatedly assigned with small-size workloads:

> *"(...) So it was again the '500 Server Error'. And it showed up quite randomly when I was at home. So, I went to the university and tried it there but again it showed up quite randomly. Even in the LAN or when I changed my code. It's very frustrating that I couldn't solve this problem."*

Considering that the free quotas of public PaaS do not support automatic scaling, we believe there could be PaaS protection mechanisms that directly reject over-frequent requests in order to avoid suspicious denial-of-service attacks.

4.2.4 Improvement.

To bypass the possible PaaS-side constraints, we provided supplemental guidelines to the future students if they would like to issue multiple requests to the same worker. Firstly, we took this opportunity as a side benefit to help students further understand how the Master-Worker pattern helped tolerate random faults and still delivered acceptable results. Secondly, we suggested reducing the worker request frequency by introducing 10- to 30-second sleep time between two consecutive requests.

Note that we do not view the lack of autoscaling as a drawback of the free PaaS resources in this case, because students can learn the scalability-related knowledge more comprehensively by controlling different amounts of workers manually.

[10]An example student report of the Master-Worker project is shared online: http://bit.ly/37QIaXT

4.3 The Third Iteration for the Broker Pattern

The Broker pattern was taught as an architectural refactoring strategy to further decouple the distributed components based on the Client-Server pattern and the Master-Worker pattern. We distinguished between three broking mechanisms for students to practice, as specified below.

- Mechanism I: Broker as a shadow server to refactor the Client-Server pattern. This mechanism is meaningful and applicable particularly when there are multiple server candidates. The client asks the broker for passing on job requests that can be addressed by any candidate server. Once the job is assigned to and finished by a server, the broker transmits results back to the client. The client and the server do not see each other throughout the whole procedure.
- Mechanism II: Broker as a shadow master to refactor the Master-Worker pattern. In this case, the broker acts as a fully functional master, while it needs to wait for job requests from local masters. Once having a job, the broker exactly follows the Master-Worker pattern to cooperate with a set of workers, and eventually sends results of the whole job to the local master. Similarly, the local master and the involved workers do not necessarily know each other at all.
- Mechanism III: Caching results to further decouple the broker and the client/master. By caching job results at the broker side or a third-party storage spot, the local client/master can disconnect from the broker after sending out job requests, while fetching the results at any convenient time instead of waiting for the response from the broker.

4.3.1 *Planning.*

Recall that applications developed and deployed on IBM Cloud Lite will be removed in 10 days if no update. Therefore, we required students to use IBM Cloud Lite only to practice the three broker mechanisms, while emphasizing GAE-based workers for longer-term usage in this iteration.

The generic tasks for students to practice the Broker pattern are defined as follows:

1) Share and reuse the workers running on GAE from the previous projects.
2) Implement the brokering mechanism I, and use it to refactor your client-server project.
3) Implement the brokering mechanism II, and use it to refactor your master-worker project.
4) Implement the brokering mechanism III based on your implementation of the mechanism II, and use Drobox as the third-party storage spot for asynchronous access to the job results.
5) Qualitatively analyze the benefits and/or drawbacks of the three brokering mechanisms.

4.3.2 *Action.*

To satisfy the policy of IBM Cloud Lite (i.e. to increase the update frequency), each student was required to develop the three types of brokers as three modules with incremental functionalities of a single application. In particular, students were encouraged (and were also willing) to collaborate with each other to share and reuse their previous work, so as to reduce efforts on developing new workers. The project environment for implementing the three brokering mechanisms is roughly illustrated in Figure 4. For the purpose of conciseness, the environmental illustration merges the elements and data flows of the three brokering mechanisms together.

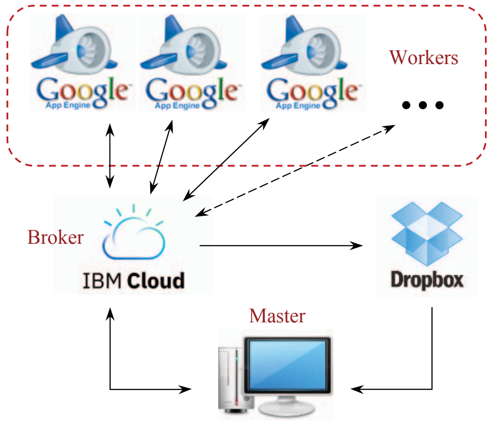

Figure 4: Project environment for students to practice the Broker pattern.

Note that the broker can also act as a simple proxy merely to redirect the client/master to suitable server/workers from its registry. However, we only gave explanations in the classes without asking students to practice, because the proxy mechanism would eventually lead the Broker pattern to the previously-taught patterns at runtime.

4.3.3 *Observation and Evaluation.*

According to the project reports, students have not only well received the Broker pattern, but have also realized its impacts on multiple software quality features, for example:

> *"There are a couple of advantages to using this approach (Mechanism III) ..., the first one is that one no longer needs to keep the local machine working constantly on the problem, it can just finish the computation at a later date, this is useful in case the computation is expensive..."* [11]

In addition, sharing and reusing the available workers have been proved efficient for accomplishing this project. However, it is also reported that many students encountered issues of non-standard worker responses (e.g., simple numbers vs. well-formatted JSON pairs). Consequently, they had to talk to each other and pay extra efforts on adapting or adjusting different worker response formats.

4.3.4 *Improvement.*

The non-standard worker response formats are essentially API compatibility issues. Therefore, we supplemented lectures on API (and especially RESTful API) as a prerequisite to the future iterations for advanced architectural patterns.

As for the shadow iterations of practicing the Broker pattern, we decided to leave the current situation as it is. Given our observations, we believe that identifying, discussing and solving the compatibility issues can further strengthen the students' collaboration and teamwork skills.

[11] An example student report of the Broker project is shared online: http://bit.ly/2mlyOk2

4.4 The Fourth Iteration for the MVC Pattern

We introduced the MVC pattern also as an evolution of the Client-Server pattern along two directions. The first direction is from the simple two-tier architecture to the advanced N-tier architecture, and different tiers address separate concerns. MVC is then a typical three-tier architectural pattern evolved from the two-tier Client-Server pattern. The second direction is the previous trend in thinning the client down. Although generic MVC is not always the case, we argue that cloud-native MVC follows this trend from the Client-Server pattern to the Browser-Server pattern.

4.4.1 Planning.

Given the long-lasting popularity of the MVC pattern in modern web applications, various web frameworks have been developed to facilitate MVC implementation [19], and some generic frameworks like Flask have also been demonstrated MVC-friendly. In particular, Flask is natively and widely supported by free PaaS Python runtimes, and its lightweight and flexibility can largely flatten the learning curve. Therefore, we selected PaaS-based Flask as an ideal platform for students to practice the MVC pattern. The corresponding tasks are defined as follows:

1) Clone the available open-source demo (similar to Appendix A) as your own project template.
2) Design a single-entity model and use it to replace the demo's model.
3) Get familiar with the PaaS-specific database solution. Develop trial code to try reading/writing data from/to the database.
4) Implement the controller functions including Create, Read, Update, and Delete (CRUD) for operating the model's data.
5) Modify the webpage templates into your application views for user interactions. Interface decoration is not needed.
6) Make your MVC-based application work properly and give a live presentation in class.

4.4.2 Action.

Following the aforementioned evolution path, we used the client-server student lookup system as an inspiration project for theoretical explanations. Students then mostly implemented MVC-based systems to manage their own information either on GAE or on IBM Cloud Lite, as illustrated in Figure 5. Although there were also projects modeling other (non-student) entities, the controller/view implementations were more or less the same.

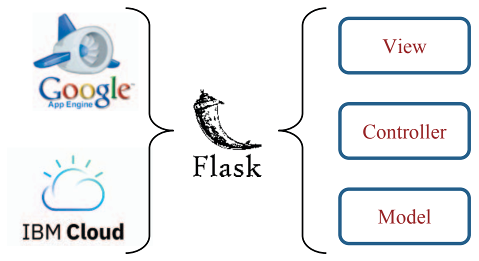

Figure 5: Using PaaS-native Flask framework for students to practice the MVC pattern.

4.4.3 Observation and Evaluation.

MVC seems to be a more challenging architectural pattern to practice by using PaaS resources. Students widely recognized that having an available open-source demo (Task 1) significantly facilitated starting their practices. However, since the generic demo did not cover any PaaS-specific database operation, many students reported difficulties in operating the model's data (Task 4). For example:

> *"The major difficulties were in the (GAE) Datastore, there is still some work to do for a better unserstanding of this plattform. But also we could imporove (improve) our understanding about the Flask framework and how APIs works."*

We believe that the difficulties were caused by students' knowledge gap between NoSQL and relational database. Students are generally equipped with the rational database knowledge, while the free PaaS databases employ NoSQL technologies.

In addition, we did not see clear performance difference between student teamwork and individual work in different shadow iterations, except for that several teams used the opportunity of trial coding (Task 3) to practice RESTful APIs.

4.4.4 Improvement.

Unlike our improvement strategy in the first iteration, here we tried to flatten the learning curve for using PaaS-specific databases by providing students with relevant tutorials (in the cloud computing course) or cheat sheets (in other courses).

In addition, considering the little teamwork value in practicing the MVC pattern at small scale, we planned to keep assigning student projects to individuals in the shadow iterations.

4.5 The Fifth Iteration for the MSA Pattern

To facilitate students' understanding, we explained the MSA pattern to be orthogonally paired with the MVC pattern (or layered pattern in generic) for breaking the architectural monolith. Since MSA shifts the hindering complexity from the inner monolithic application to the outer infrastructural architecture [6, 32], the MSA pattern is associated with many software quality features and knowledge points for students to study, such as maintainability, reliability, data consistency, concurrency, etc.

4.5.1 Planning.

When it comes to student projects for experiential learning or tutorial demonstration, unfortunately, there are few suitable candidates in the microservices domain [2]. Considering that the pet store program has widely been employed as an application-level "Hello World" demo to exemplify how to use particular languages, software frameworks or different sets of libraries to build web applications [31], we decided to let students collaboratively develop an MSA-based pet store for learning microservice knowledge and technologies. The corresponding tasks are then defined as follows:

1) Review the business logic of the pet store application.
2) Discuss the relevant bounded contexts by using the Domain-Driven Design (DDD) approach. (This is a prerequisite task to the rest of the tasks, which should reach consensus on four microservices, namely Customer, Pet, Shopping Cart, and Order).
3) According to the discussions, select one microservice to work on, and modify/migrate your previous MVC project into that

microservice with suitable APIs. Note that both the Shopping Cart model and the Order model would need two tables.

4) Build up teams and each team should comprise the four microservices. Every team member needs to individually implement an API gateway to realize a pet store application.
5) Considering possible compatibility issues among different microservices, everyone has the responsibility to answer the others' questions about your microservice, and also improve your microservice if needed.

4.5.2 *Action.*

Given the same concern about the 10-day limit of free IBM Cloud resources, students were asked to use only GAE to implement the four entity microservices for long-term usage/reuse, while both GAE and IBM Cloud Lite were acceptable for implementing the API gateway. Thus, the students' working environment can be illustrated as shown in Figure 6. It is noteworthy that the Flask-based implementations of individual microservices (i.e. MS *1*, MS *2*, ..., MS *N*) have all followed the MVC pattern.[12]

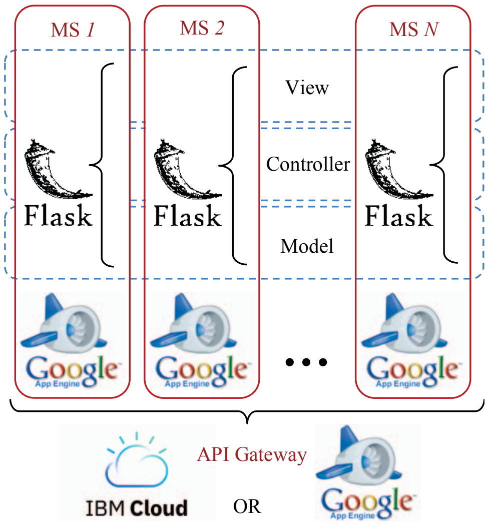

Figure 6: Project environment for students to practice the MSA pattern.

In particular, when collaborating with each other, students in turn played (and had to play) the roles of developer, domain expert, and user at different project stages.

4.5.3 *Observation and Evaluation.*

Compared to practicing the other architectural patterns, employing public and free PaaS seems particularly suitable to facilitate learning the MSA pattern collaboratively. Students have generally recognized the benefits of PaaS (or cloud) that make their microservices accessible and reusable by anyone, at any time, from anywhere. Correspondingly, most students obtained their first-hand experience not only in learning how MSA works,[13] but also in justifying the advantages of the MSA pattern:

> *"It is important to emphasize that the development was easier thanks to the fact that the microservices of the functions of the system were already implemented, allowing the parallel development of the different microservices and demonstrating the benefits of the architecture."*

On the other hand, we observed several incomplete student practices. The main challenge here still tends to be dealing with compatibility issues (e.g., the same entity attribute can be named differently in different microservices), although we have standardized API input/output with JSON formats. This challenge inevitably requires extensive discussions and negotiations among students. Unfortunately, such a collaboration overhead seems too high for some students. In fact, even students who successfully accomplished the planned tasks also highlighted this challenge as difficulties in their work (e.g., the discussion in http://bit.ly/31jRDVf).

4.5.4 *Improvement.*

Through deeper investigations, we found that the difficulties students met here had nothing to do with the employed PaaS services or previous API issues (cf. Section 4.3.3), but were mainly related to different naming conventions (e.g., "customer_name" vs. "name") of the input/output data. To reduce the extra effort on the naming issues, we decided, firstly, to reinforce our teaching about the API design; and secondly, to ask students to provide and share informative API usage demos when exposing their microservice APIs.

In addition, since the pet store development is a semester-long project, we further gave source-code level instructions to the slow-progress students/teams at suitable steps in the shadow iterations, so as to balance the whole class's learning pace.

5 DISCUSSIONS

Based on our observations on and evaluations of the individual iterations including shadow iterations, we respectively discuss the benefits and current limits of the proposed educational strategy.

5.1 Benefits of Our Educational Strategy

First of all, our whole work shows a promising way to address the challenge in learning abstract architecture knowledge [20]. As demonstrated in Section 4, the free and public PaaS resources are satisfying for students to build up lightweight software systems, while the small to medium student projects can support experiential learning in the software architecture domain. Although some studies argue that in-class applications cannot reveal the significance of software architecture [13], we believe that our educational strategy is particularly valuable in the fundamental courses. After all, students need to get familiar with the various architectural concepts and patterns before being able to make trade-offs among quality concerns for large and real-world systems.

[12] An example student report of implementing the microservice Pet is shared online: http://bit.ly/30RSt9Y

[13] An example student report of the MSA project is shared online: http://bit.ly/31jRDVf, and this student's API Gateway for finalizing the pet store application is implemented on GAE: http://charlie-pet-store.appspot.com/

Secondly, the public feature of the employed PaaS services largely facilitates and fosters student teamwork and collaborations in suitable projects. It has been identified that learning architecting is generally a teamwork-intensive process [13], not to mention that teamwork skills are particularly crucial for students to develop in distributed software scenarios [14]. By making software components functional and accessible publicly, students can accomplish more comprehensive investigations (e.g., relationships among the components and their impacts on software quality) not only by working together traditionally (e.g., in the lab), but also by having discussions and cooperations more flexibly (e.g., even at home). More importantly, following our educational strategy, students will have to collaborate on resource-intensive projects, in order to enjoy the collective free quotas from multiple individuals.

From the teachers' perspective, thirdly, the publicly accessible PaaS platforms make student projects easy to evaluate by allowing to check the project systems' (or components') runtime online. In contrast, before employing this educational strategy, verifying project reports usually required teachers and teaching assistants to observe students' live demos in the lab or even to compile their source codes locally.

Fourthly, utilizing the personal account-associated free quotas of PaaS can potentially reduce the plagiarism in student projects. Given the natural privacy concern about disclosing authentication and authorization information, students are barely willing to work on behalf of each other by sharing their cloud accounts and passwords, especially in the case of GAE, because the Google account is also linked to many other products like Gmail.

Last but not least, our educational strategy is well aligned with the wide trend of employing cloud computing to relieve the increasing budget shortage in education [28, 33]. Furthermore, compared with the education-dedicated cloud resources, the free and public PaaS does not require contacting any cloud provider or going through any paperwork for their usage, which even minimizes the traditional administration effort/cost [4, 9] into zero. In addition, without having any time/location constraint of usage, the employed PaaS services can assist students' life-long learning [35] by enabling students to access and improve their coursework outside of university and after graduation.

5.2 Limits of Our Educational Strategy

The limits of our educational strategy can be distinguished between the learning side and the teaching side.

From the standpoint of learning, recall that we employed TAM to evaluate our educational strategy in terms of Perceived Usefulness and Perceived Ease of Use (cf. Section 3). Although all the feedback we received on usefulness is positive, students have exhibited diverse learning curves for employing PaaS resources in their projects. To increase the ease of use of available PaaS services, as mentioned previously, we have tried to enrich teaching materials and especially practice guidelines (e.g., Appendix A) for different projects and even for different students. However, such improvement efforts in turn introduce new limits to the teaching side:

Firstly, when sharing the same student projects among different courses, teachers still need to pay attention to different course characteristics and accordingly adjust project preparations, guidelines and instructions. For example, in our case, quickstarts of using PaaS have been proved valuable and helpful to students' practice in pure software architecture courses, while these cheat sheets would oversimplify the complexity in the student projects for studying architectural patterns together with cloud computing.

Secondly, teachers have to check and possibly update the PaaS-specific teaching materials from semester to semester, even for the same student projects in the same course. It has been identified that less controllability is an inherent feature of using public cloud services. Cloud providers can always modify their service-level agreement along their business journey, without necessarily negotiating with the service users. For example, during our study, one of the previously available PaaS instances (i.e. GAE's Java 7 Runtime) was shut down and replaced with some others [17]. Therefore, we cannot expect the PaaS-specific teaching materials to be prepared only once and for all.

6 CONCLUSIONS AND FUTURE WORK

Driven by the widely discussed challenges and even dilemmas in software architecture education, we proposed an educational strategy of using public and free PaaS based lightweight projects to facilitate students' learning. After applying this strategy across three relevant courses for two years, we are able to answer "Yes" to both of the predefined research questions.

In detail, to respond RQ1, we conclude that lightweight projects can support undergraduate students to study software architecture via experiential learning and collaborative learning. Note that this conclusion does not violate the existing suggestions about employing real-world systems in advanced architecture courses, because their focus and ours are essentially on different learning contexts/levels. To respond RQ2, we conclude that public and free PaaS resources are satisfying for students to build lightweight projects for learning fundamental architecture knowledge. In particular, a unique feature of our educational strategy is to let students utilize their own PaaS resources in their "learning by doing" activities, which not only eliminates the budgetary cost and administration overhead, but also allows students to keep their coursework in the public cloud for self-studying at any time from anywhere.

Following the requirement of enriching empirical evidence for innovative teaching [4], we plan to keep extending our study along two directions in the future. On one hand, we have been trying to cover broader fundamental architecture knowledge as well as employing more available PaaS resources (e.g., the recently identified Pivotal Web Services). On the other hand, we will manage to introduce this educational strategy to suitable advanced courses at the postgraduate-curriculum level.

ACKNOWLEDGMENTS

This work was supported in part by the University of Concepción under Grant Teaching Innovation INICIA I19-027, and in part by Chilean National Commission for Scientific and Technological Research (CONICYT, Chile) under Grant FONDECYT Iniciación 11180905.

Following the Axiom 2 of service-dominant logic [29], the educational value is co-created by both teachers and students. This study cannot be accomplished without the author's talented students.

REFERENCES

[1] Martin L. Abbott and Michael T. Fisher. 2015. *The Art of Scalability: Scalable Web Architecture, Processes, and Organizations for the Modern Enterprise* (2nd. ed.). Addison-Wesley Professional, Crawfordsville, Indiana.

[2] Carlos M. Aderaldo, Nabor C. Mendonça, Claus Pahl, and Pooyan Jamshidi. 2017. Benchmark Requirements for Microservices Architecture Research. In *Proceedings of the 1st International Workshop on Establishing the Community-Wide Infrastructure for Architecture-Based Software Engineering (ECASE 2017)*. IEEE Press, Buenos Aires, Argentina, 8–13. https://doi.org/10.1109/ECASE.2017.4

[3] Abdulrahman Alharthi, Fara Yahya, Robert J Walters, and Gary B Wills. 2015. An overview of cloud services adoption challenges in higher education institutions. In *Proceedings of the Emerging Software as a Service and Analytics 2015 Workshop in conjunction with CLOSER 2015 (ESaaSA '15)*. SciTePress, Lisbon, Portugal, 102–109. https://doi.org/10.5220/0005529701020109

[4] Maria Teresa Baldassarre, Danilo Caivano, Giovanni Dimauro, Enrica Gentile, and Giuseppe Visaggio. 2018. Cloud Computing for Education: A Systematic Mapping Study. *IEEE Transactions on Education* 61, 3 (August 2018), 234–244. https://doi.org/10.1109/TE.2018.2796558

[5] Courtney Bittner. 2017. Introducing the IBM Cloud Lite account. Retrieved August 31, 2019 from https://www.ibm.com/blogs/bluemix/2017/11/introducing-ibm-cloud-lite-account-2/

[6] Justus Bogner and Alfred Zimmermann. 2016. Towards Integrating Microservices with Adaptable Enterprise Architecture. In *Proceedings of the IEEE 20th International Enterprise Distributed Object Computing Workshop (EDOCW 2016)*. IEEE Press, Vienna, Austria, 158–163. https://doi.org/10.1109/EDOCW.2016.7584392

[7] Grady Booch. 2007. The Economics of Architecture-First. *IEEE Software* 24, 5 (September-October 2007), 18–20. https://doi.org/10.1109/MS.2007.146

[8] Carnegie Mellon University. 2010. What Is Your Definition of Software Architecture. Retrieved February 02, 2020 from https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=513807

[9] Deka Ganesh Chandra and Dutta Borah Malaya. 2012. Role of cloud computing in education. In *Proceedings of the 2012 International Conference on Computing, Electronics and Electrical Technologies (ICCEET '12)*. IEEE Press, Kumaracoil, India, 832–836. https://doi.org/10.1109/ICCEET.2012.6203884

[10] Fred D. Davis. 1989. Perceived Usefulness, Perceived Ease of Use, and User Acceptance of Information Technology. *MIS Quarterly* 13, 3 (September 1989), 319–340. https://doi.org/10.2307/249008

[11] Remco C. de Boer, Rik Farenhorst, and Hans van Vliet. 2009. A Community of Learners Approach to Software Architecture Education. In *Proceedings of the 22nd Conference on Software Engineering Education and Training (CSEET '09)*. IEEE Computer Society, Hyderabad, Andhra Pradesh, India, 190–197. https://doi.org/10.1109/CSEET.2009.10

[12] Davide Falessi, Muhammad Ali Babar, Giovanni Cantone, and Philippe Kruchten. 2010. Applying empirical software engineering to software architecture: Challenges and lessons learned. *Empirical Software Engineering* 15, 3 (June 2010), 250–276. https://doi.org/10.1007/s10664-009-9121-0

[13] Matthias Galster and Samuil Angelov. 2016. What makes teaching software architecture difficult?. In *Proceedings of the 38th International Conference on Software Engineering Companion (ICSE '16)*. ACM Press, Austin, Texas, USA, 356–359. https://doi.org/10.1145/2889160.2889187

[14] Fáber D. Giraldo, Sergio F. Ochoa, Myriam Herrera, Andrés Neyem, José Luis Arciniegas, Clifton Clunie, Sergio Zapata, and Fulvio Lizano. 2011. Applying a distributed CSCL activity for teaching software architecture. In *Proceedings of the 2011 International Conference on Information Society (i-Society '11)*. IEEE, London, UK, 208–214. https://doi.org/document/5978540

[15] José A. González-Martínez, Miguel L. Bote-Lorenzo, Eduardo Gómez-Sánchez, and Rafael Cano-Parra. 2015. Cloud computing and education: A state-of-the-art survey. *Computers & Education* 80 (January 2015), 132–151. https://doi.org/10.1016/j.compedu.2014.08.017

[16] Google. 2019. App Engine Locations. Retrieved August 17, 2019 from https://cloud.google.com/appengine/docs/locations

[17] Google. 2019. Java 7 Runtime Shutdown. Retrieved September 21, 2019 from https://cloud.google.com/appengine/docs/deprecations/java7

[18] Google. 2019. Using Cloud Datastore. Retrieved August 31, 2019 from https://cloud.google.com/appengine/docs/standard/python/using-cloud-datastore

[19] Gulu99. 2019. MVC Tutorial for Beginners: What is, Architecture & Example. Retrieved September 17, 2019 from https://www.guru99.com/mvc-tutorial.html

[20] Christian Köppe and Leo Pruijt. 2015. Teaching Software Architecture Concepts with HUSACCT: Tool Demo. In *Proceedings of the 2015 ACM SIGPLAN International Conference on Systems, Programming, Languages and Applications: Software for Humanity (SPLASH-E '15)*. ACM Press, Pittsburgh, Pennsylvania, USA, 1–4.

[21] Arne Koschel, Felix Heine, Irina Astrova, and Igor Astrov. 2018. A Private Cloud for Data Mining Education. In *Proceedings of the 6th International Conference on Enterprise Systems (ES '18)*. IEEE Press, Limassol, Cyprus, 129–133. https://doi.org/10.1109/ES.2018.00027

[22] Philippe Kruchten, Henk Obbink, and Judith Stafford. 2006. The Past, Present, and Future for Software Architecture. *IEEE Software* 23, 2 (March-April 2006), 22–30. https://doi.org/10.1109/MS.2006.59

[23] Patricia Lago, Jia F. Cai, Remco C. de Boer, Philippe Kruchten, and Roberto Verdecchia. 2019. DecidArch: Playing Cards as Software Architects. In *Proceedings of the 52nd Hawaii International Conference on System Sciences (HICSS '19)*. AIS Electronic Library (AISeL), Maui, Hawaii, USA, 7815–7824. https://doi.org/10.24251/hicss.2019.940

[24] Timothy C. Lethbridge. 2000. What Knowledge Is Important to a Software Professional? *Computer* 33, 5 (May 2000), 44–50. https://doi.org/10.1109/2.841783

[25] Zheng Li, Rajiv Ranjan, Liam O'Brien, He Zhang, Muhammad Ali Babar, Albert Y. Zomaya, and Lizhe Wang. 2018. On the Communication Variability Analysis of the NeCTAR Research Cloud System. *IEEE Systems Journal* 12, 2 (June 2018), 1506–1517. https://doi.org/10.1109/JSYST.2015.2483759

[26] Tomi Mannisto, Juha Savolainen, and Varvana Myllarniemi. 2008. Teaching Software Architecture Design. In *Proceedings of the 7th Working IEEE/IFIP Conference on Software Architecture (WICSA '08)*. IEEE Computer Society, Vancouver, BC, Canada, 117–124. https://doi.org/10.1109/WICSA.2008.34

[27] Marinela Mircea and Anca Ioana Andreescu. 2011. Using Cloud Computing in Higher Education: A Strategy to Improve Agility in the Current Financial Crisis. *Communications of the IBIMA* 2011 (2011). https://doi.org/10.5171/2011.875547 article no. 875547.

[28] Kiran Bala Nayar and Vikas Kumar. 2018. Cost benefit analysis of cloud computing in education. *International Journal of Business Information Systems* 27, 2 (January 2018), 205–221. https://doi.org/10.1504/IJBIS.2018.089112

[29] Irene C. L. Ng and Stephen L. Vargo. 2018. Service-dominant (S-D) logic, service ecosystems and institutions: bridging theory and practice. *Journal of Service Management* 29, 4 (July 2018), 518–520. https://doi.org/10.1108/JOSM-07-2018-412

[30] Linda Northrop. 2006. Let's teach architecting high quality software. In *Proceedings of the 19th Conference on Software Engineering Education & Training (CSEET '06)*. IEEE Computer Society, Turtle Bay, HI, USA, 1–1. https://doi.org/10.1109/CSEET.2006.23

[31] Oracle. 2019. Java Pet Store. Retrieved August 7, 2019 from http://www.oracle.com/technetwork/java/petstore1-3-1-02-139690.html

[32] Marian Rusek, Grzegorz Dwornicki, and Arkadiusz Orłowski. 2017. A Decentralized System for Load Balancing of Containerized Microservices in the Cloud. In *ICSS 2016: Advances in Systems Science*. Advances in Intelligent Systems and Computing, Vol. 539. Springer, Cham, 142–152. https://doi.org/10.1007/978-3-319-48944-5_14

[33] Humphrey M. Sabi, Faith-Michael E. Uzoka, Kehbuma Langmia, and Felix N. Njeh. 2016. Conceptualizing a model for adoption of cloud computing in education. *International Journal of Information Management* 36, 2 (April 2016), 183–191. https://doi.org/10.1016/j.ijinfomgt.2015.11.010

[34] Mark K. Smith. 2001. Kurt Lewin: groups, experiential learning and action research. Retrieved August 7, 2019 from http://infed.org/mobi/kurt-lewin-groups-experiential-learning-and-action-research/

[35] Nabil Sultan. 2010. Cloud computing for education: A new dawn? *International Journal of Information Management* 30, 2 (April 2010), 109–116. https://doi.org/10.1016/j.ijinfomgt.2009.09.004

[36] Ping Xia, Liang Zhang, and Zhifeng Zeng. 2018. Research and Design of Regional Education Cloud System Based on University. In *Proceedings of the 8th International Conference on Management, Education and Information (MEICI '18)*. Atlantis Press, Shenyang, China, 434–438. https://doi.org/10.2991/meici-18.2018.85

[37] Steve Zurier. 2009. Virginia Universities Construct Virtual Computing Labs. Retrieved August 17, 2019 from https://edtechmagazine.com/higher/article/2009/05/virginia-universities-construct-virtual-computing-labs

A QUICKSTART FOR CLONING A GOOGLE APPENGINE APPLICATION FROM SCRATCH

We follow Google's convention to name the cloud-side software module to be an application in this quickstart. In our courses, the "AppEngine applications" are essentially functional components of student-developed software systems. This quickstart has been tested under Ubuntu 18.04 as the Google Cloud SDK environment.

A.1 Install Google Cloud SDK

1) Create an environment variable for the Cloud SDK distribution:
```
export CLOUD_SDK_REPO="cloud-sdk-$(lsb_release -c -s)"
```
2) Add the Cloud SDK distribution URI as a package source:
```
echo "deb http://packages.cloud.google.com/apt $CLOUD_SDK_REPO main"
| sudo tee -a /etc/apt/sources.list.d/google-cloud-sdk.list
```
3) Import the public key of Google Cloud Platform:
```
curl https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-key add -
```
4) Update the package list and install the Cloud SDK:
```
sudo apt-get update && sudo apt-get install google-cloud-sdk
```
5) Update the Cloud SDK components:
```
gcloud components update
```

A.2 Create a Google AppEngine Project and an Application Namespace

1) Connect your Cloud SDK to your Google account:
```
gcloud init
```
Then, you can follow the initialization instructions to login your Google account. Note that it is not necessary to become a billing account by registering any credit card information.
2) Google Cloud will have a default project created when you login for the first time. You can also choose to create a new project:
```
gcloud projects create [YOUR_PROJECT_NAME] --set-as-default
```
3) Verify the created cloud project:
```
gcloud projects describe [YOUR_PROJECT_NAME]
```
If the project has been created successfully, you should be able to see the project details including creation timestamp, status, name and ID.
4) Create an AppEngine application under a particular project:
```
gcloud app create --project=[YOUR_PROJECT_NAME]
```
Then, you need to select a geographical region where you want your application to be deployed. The created application for now can be viewed only as a namespace without any functionality.

A.3 Clone a Functional Application to Your AppEngine Application Namespace

1) Download a well-prepared application from the GitHub repository. For example, here is a worker for the Monte Carlo approximation of Pi for practicing the Master-Worker pattern:
```
git clone https://github.com/clandero/MonteCarloPi
```
2) Test the downloaded application locally by entering the application directory and running the command:
```
dev_appserver.py .
```
You should be able to test it through http://localhost:8080/NUMBER_OF_POINTS in any web browser. For example, if random sampling 10000 points, http://localhost:8080/10000 will return the number of points falling inside the circle.
3) Deploy the downloaded application to your AppEngine application namespace:
```
gcloud app deploy
```
If successful, you can check or run the cloned application for your own AppEngine project through
http://[YOUR_PROJECT_ID].appspot.com/NUMBER_OF_POINTS.