Building Web Applications on Google Cloud Platform

Daniel Gareev
University of Luxembourg
Email: daniel.gareev.001@student.uni.lu

Alfredo Capozucca
University of Luxembourg
Email: alfredo.capozucca@uni.lu

Abstract

The primary objective of this project is to develop an introduction to the cloud computing concepts, learn how to work with Google Cloud Platform, and how to develop a simple web application and deploy it on App Engine. This project also focuses on some of the related software development concepts, such as project management and software engineering environments.

1. Introduction

The main goal of the project is to build a fundamental knowledge of cloud computing concepts, learn how to use Google Cloud Platform to build and deploy a simple web application on App Engine, as well as learn some of the relevant software development practices such as agile project management and software engineering environments.

Cloud computing is a rapidly growing technology that many organizations are adopting to enable agility, elasticity, and cost savings. It allows to innovate faster and focus on valuable IT resources on developing applications that differentiate the product rather than managing infrastructure and data centers.

2. Project description

2.1. Domains

2.1.1. Scientific

The scientific domain of the project mainly focuses around two following areas. First, we are exposed to the cloud computing domain, software development and software engineering environments.

Second, as we are guided by the need to keep the project on track, we will incorporate project management methodologies into this project. It is worth mentioning that the learning outcomes of this project is to get hands-on experience with various project management methods, in particular, agile methodologies.

2.1.2. Technical

The primary domain of this project is Google Cloud Platform (GCP) along with the Google App Engine and the development of a simple web application on Google Cloud Platform. It is also necessary to learn how each feature of the app will be implemented using technologies and services provided by GCP in an efficient, scalable and reliable way.

Additionally, the project extends to additional areas related to software development such as programming languages (i.e., Python), version control systems (i.e., Git), web frameworks (i.e., Flask for Python) and web languages (i.e., HTML, CSS).

2.2. Targeted Deliverables

2.2.1. Scientific deliverables

The main objective of this project is to understand the fundamental elements of cloud computing.

We first need to develop an introduction to the main concepts underlying cloud computing. What is cloud computing? What are the basic concepts and terminology? What are the benefits and constraints of using cloud computing? What are the essential characteristics of cloud computing (i.e., on-demand self-service, broad network access, etc.)? Then, we will look in-depth at different deployment models as well as service models of cloud computing (i.e., Infrastructure as a Service, Platform as a Service, and Software as a Service). We will look specifically into Platform as a Service, and Software as a Service as these are the main deployment models we will be exploring during this project.

We will also develop an understanding of project management methodologies and software engineering environment in the context of cloud computing.

2.2.2. Technical deliverables

The primary objective of the project is to use Google Cloud Platform and the features it provides in order to develop a simple web application that will be running on GCP's cloud computing services. We will develop an understanding of Google Cloud Platform along with the Google App Engine and its interaction with other services provided by GCP such as Datastore and Cloud Storage, among others, and how they interact within the GCP environment. For example, during the project, we should learn different authentication mechanisms proposed by GCP and alternatives for storing data (e.g., Cloud Storage, MongoDB, etc.), as well as how to choose the most appropriate implementation for our project.

To achieve this in practice, we need to analyze in-detail the underlying architecture of GCP and set up the development

environment: learn how to create a GCP project, work with the Google Cloud SDK, understand the deployment process and deal with Git version control during the building and deployment phases. It is also the requirement of this project to provide complete isolation of code and data, and operator permissions so that the project can be managed separately within the development, testing and production environments.

Moreover, Jira tool is going to be used for handling the tasks related to this project, as has been already noted above. The technical aspects covered by the project management deliverable is handling and task management with Jira tool.

Additionally, we must get acquainted with the skills in a range of technologies during the development of web application: we will use Python as the primary programming language and develop a knowledge of its Flask framework for web apps.

3. Background

3.1. Scientific background

3.1.1. Project Management

Project management is the practice of initiating, planning, executing and controlling the project's process in order to achieve specific goals and meet specific criteria at the specified time [4]. This information is generally described in project documentation, that is created at the beginning of the development process. The primary challenge of project management is to achieve all of the project goals within the given constraints [4].

In this project, we will specifically focus on the agile methodologies related to project management.

3.1.2. Version Control System

Version Control System (VCS) is a system that records changes to a file or set of files over time so that user can recall specific versions later. It is mainly designed to maintain software source code as the files being version controlled, though in reality, it can be used with any type of file [3].

In this project, we will be using VCS in order to facilitate the development process.

3.2. Technical background

3.2.1. Programming Language

Python is a powerful high-level programming language for general-purpose programming, comparable to Java, Ruby or Scheme [2]. While there are several reasons for choosing Python as a programming language for this project, one of the main factors was Python's selection of web frameworks that facilitates the development process. Python's syntax allows easier readability for rapid testing of algorithms as well as focus on problem-solving. Python is an open-source programming language and is supported by a range of resources and high-quality documentation which is also applicable to the Flask framework written in Python that we will use in this project. Besides prior experience with Python was also one of the important reasons for choosing this particular programming language.

3.2.2. Git

Git is an open source distributed version control system created to handle both small and large projects with speed and efficiency [3]. It is primarily used for source-code management in software development, but it can be used to keep track of changes in any set of files [3]. As a distributed revision-control system, it is focused on "speed, data integrity, and support for distributed, non-linear workflows" [3].

We will use Git extensively in this project as we will set up the integration between Git and GCP as will be discussed further.

4. A Scientific Deliverable 1: Cloud Computing Fundamentals

4.1. Requirements

The main objective of this deliverable focuses on building an understanding of the fundamental elements of cloud computing. This section thus provides an overview of introductory cloud computing topics that is followed by definitions of basic concepts and terminology. Moreover, a section in the appendix with the common notions of cloud computing is produced during the project.

4.2. Design

The main objective is to follow Introduction to Cloud Computing course provided by Udemy in order to build an understanding of the fundamental elements of cloud computing [1]. In this course, we are provided with the necessary resources and introductory cloud computing topics.

4.3. Production

Cloud computing refers to both the applications delivered as services over the Internet and the hardware and systems software in the data centers that provide those services. [5] It is important to introduce a general definition of cloud computing before getting into details. The definition that received wide acceptance was composed by the National Institute of Standards and Technology (NIST): *"Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. "* [6].

By the definition of NIST, the cloud model is composed of five essential characteristics, three service models, and

four deployment models [6]. We will primarily focus on the essential characteristics and three service models within the scope of this project.

There are three major types of cloud services: IaaS (Infrastructure as a Service), PaaS (Platform as a Service), and SaaS (Software as a Service).

SaaS allows people to use cloud-based applications running on a cloud infrastructure. The applications can be accessed through either a web browser or a program interface [6]. It is important to note that the consumer does not manage or control the underlying cloud infrastructure such as operating systems, storage, or network, with the possible exception of limited user-specific application configuration settings as shown in Figure 1 [6]. In fact, email services such as Gmail is an example of cloud-based SaaS services.

PaaS refers to cloud platforms that provide runtime environments for developing, testing, and managing applications. Consumer does not manage or control the underlying cloud infrastructure such as operating systems, storage, or network, but has control over the deployed applications, storage and possibly configuration settings for the application-hosting environment as shown in Figure 1 [6]. One of the examples of PaaS services is Google App Engine.

IaaS is a cloud service that provides basic computing infrastructure: servers, storage, and networking resources. IaaS allows a consumer to deploy onto the cloud infrastructure applications created using programming languages, libraries, and services supported by the provider [6]. It is important to note that the consumer does not manage or control the underlying cloud infrastructure but has control over operating systems, storage, and deployed applications as shown in Figure 1 [6]. Major IaaS providers include Google Compute Engine, Amazon Web Services and Microsoft Azure.

Cloud computing has five fundamental attributes, according to the definition of cloud computing proposed by the NIST [6]. First, customers get computing resources on-demand and self-service [6]. Cloud-computing customers use an automated interface and get the processing power, storage, and network they need, with no need for human intervention [6]. Second, they can access these resources over the network [6]. Third, the provider of those resources has a big pool of them, and allocates them to customers out of the pool [6]. That allows the provider to get economies of scale by buying in bulk. Customers don't have to know about the exact physical location of those resources. Fourth, the resources are elastic. Customers who need more resources can get more rapidly. When they need less, they can scale back. To sum up, we can see the essential attributes of cloud computing outlined in Figure 2.

4.4. Assessment

We have successfully presented the fundamental elements of cloud computing. The section in appendix establishes a set of basic terms that represent the fundamental concepts and aspects pertaining to the notion of cloud computing.

5. A Scientific Deliverable 2: Environments in the Context of Cloud Computing

5.1. Requirements

The main objective of this deliverable is to build an understanding of the concept of software engineering environments (SEE), develop an overview of SEE in the context of GCP, and implement multiple development environments with separate for our project.

5.2. Design

GCP projects form the basis for creating, enabling, and using all GCP services including managing APIs, adding and removing collaborators, and managing permissions for GCP resources [10].

It is worth mentioning that cloud-based software projects should employ multiple environments and it is essential that these environments should be completely isolated from one another, as they typically should have their own access permissions. These environments are typically created for developers, testers, and end-users and have specific respective names like *dev*, *test*, and *prod*. For example, the development team might have full access to the *dev* environment, but only limited access to the *prod* environment, with all code deployment driven only by automated scripts. Further, it is essential that the data in the different environments stay isolated.

One of the ways in which separation between environments can be achieved is explained in [11], that suggests creating a single GCP project for each of the environments and name them accordingly, such as *web-app-dev*, *web-app-test*, and *web-app-prod*. In our GCP project, we have followed the same approach, where for each environment we have created separated GCP project with specific storage, permissions, billing and other attributes. In this case, the projects provide complete isolation of code and data, and operator permissions can be managed separately [11].

5.3. Production

Any stakeholder involved in the software development life cycle of a software-related project needs means to engineer the solution (either a product or service) requested by the customer. It is thus worthwhile mentioning what is a software engineering environment as it provides a set of assets, features and functionalities aimed at being used in the development of a software-related project.

Environment refers to the collection of hardware and software tools a developer uses to build software systems [9]. By *Software Engineering Environment* (SEE) we define an environment that augments or automates the activities comprising the software development cycle [9].

The SEE thus provides an environment in which large software systems can be developed by integrating a set

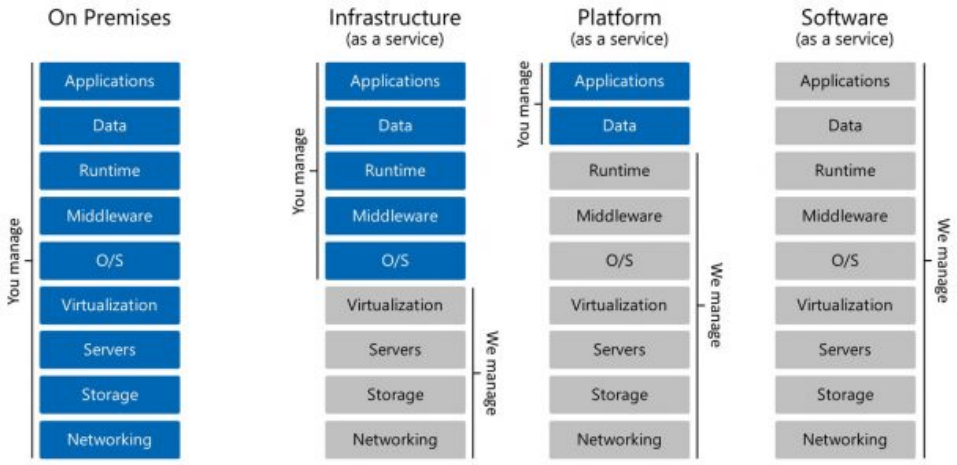


Fig. 1. Comparing cloud computing models Figure source: [8]

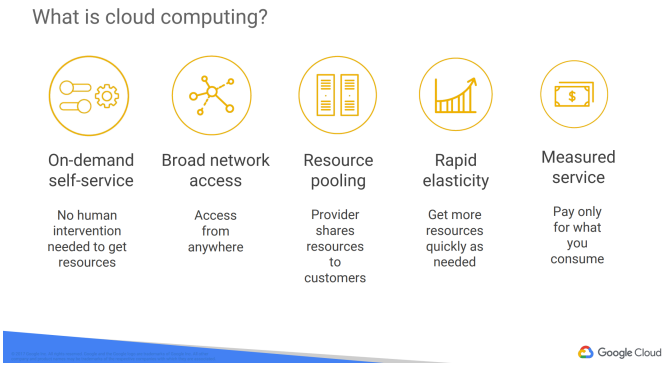


Fig. 2. Essential characteristics of cloud computing. Figure source: [7]

of tools and workflows aimed at supporting one or more development methods within a particular infrastructure that eases the communication and collaboration in a controlled way.

By using multiple environments application is made available to the specific user (i.e., developer, software tester or end-consumer), allowing them to access the application with specific permissions. The resources that each project contains remain separate across project boundaries.

It is essential to mention that in our case the environments employ separate repositories. For example, when there is a new update to the code, the application is automatically deployed on the specific allocated web page. The migration from the one environment to another implemented manually. For example, in order to go from the staging environment *test* to the production environment *prod*, a developer needs to first push the changes into the *test* environment, test the deployed application, and then merge the changes into *prod* environment. This is one of the possible solutions for working with separate environments in the context of GCP, but more sophisticated and automated solutions can be researched and implemented.

5.4. Assessment

In our case, we have introduced essential notions of SEE, as well as achieved the required level of environment separation within our cloud project. Although this might be not the most rigorous approach, it provides quite a high level of separation. There are pros and cons to using projects instead of services, and one must balance the tradeoffs depending on the situation. It is important for a microservices-based application to maintain code and data isolation between microservices, which we have achieved by using separate projects.

6. A Scientific Deliverable 3: Presentation of the Agile Project Management

6.1. Requirements

One of the main requirements of this deliverable is to scientifically present project management methodologies in the context of software development. In this project, we will specifically focus on the agile software development approach.

6.2. Design

In order to build an understanding of the agile software development approach, we will be referencing mainly scientific materials.

6.3. Production

Dealing with an increasingly complex organizational environment is a serious challenge for any software development project [12]. Traditional software development methodologies can be characterized as linear, sequential processes, and do not always reflect real-world development practices that are conducted in more volatile environments [12]. In fact, software developers have dealt with this complexity with iterative, often ad-hoc approaches [12].

It is thus essential that requirements for systems must change along with the development process. *"Even seemingly minor changes can produce unanticipated effects, as systems become more complex and their components more interdependent"* [12].

Agile Project Management (APM) is a structured and iterative approach to project management and product development [13]. This development methodology focuses on rapid iterative delivery, flexibility, and working code.

An adaptive APM-based framework includes several practices as outlined in [12]:

- The ability to manage and adapt to change
- A view of organizations as fluid, adaptive systems composed of intelligent people
- Recognition of the limits of external control in establishing order
- An overall humanistic problem-solving approach

There are several frameworks that help teams adhere to agile principles.

Kanban is one of the frameworks used to implement agile software development that represents a visual system for managing tasks as they move through a process [13]. Kanban visualizes both the process (the workflow) and the actual tasks passing through that process [13].

Another framework that is worthwhile to mention is Scrum framework that is used to develop, deliver, and sustain complex products [14]. The definition is outlined in Scrum Guideline, where Scrum is defined as *"a framework within which people can address complex adaptive problems, while productively and creatively delivering products of the highest possible value"* [14]. Scrum is founded based on empirical process control theory, or empiricism that asserts that knowledge comes from experience and making decisions based on what is known [14]. Scrum employs an iterative, incremental approach to optimize predictability and control risk [14].

6.4. Assessment

In this section, we have presented the most essential concepts related to project management. While we have not presented the most extensive guide on project methodologies, we successfully described the project-specific methodology that we used in our project.

7. A Technical Deliverable 1: Introduction to GCP

7.1. Requirements

In this section, we are required to describe the Google Cloud Platform fundamentals and its core infrastructure that includes computing and storage services. This will enable us to interact with Google Cloud Platform services as well as identify the purpose and value of Google Cloud Platform products and services.

7.2. Design

The main objective of this deliverable is to follow Google Cloud Platform Fundamentals: Core Infrastructure course on Coursera in order to build the initial knowledge of Google Cloud Platform [7]. This course introduces us to important concepts and terminology for working with Google Cloud Platform. The course offers us an extensive overview of some of the computing and storage services available in Google Cloud Platform, including Google App Engine, Google Compute Engine, Google Cloud Storage and Google Cloud SQL, that we will be using in this project [7]. In this course, we also learn about essential resource and policy management tools, such as the Google Cloud Resource Manager hierarchy and Google Cloud Identity and Access Management [7].

It is important to note that this knowledge will allow us to better understand GCP and develop an application on top of App Engine.

7.3. Production

Google Cloud Platform is a suite of cloud computing services that runs on the same infrastructure that Google uses for its end-user products, such as Google Search and YouTube [7]. GCP consists of a set of physical assets, such as computers and hard disk drives, and virtual resources, such as virtual machines (VMs), that are contained in Google's data centers [7]. Google Cloud Platform enables developers to build, test, and deploy applications on Google's highly secure, reliable, and scalable infrastructure [7]. Google Cloud Platform's products and services can be broadly categorized as Compute, Storage, Big Data, Machine Learning, Networking, and Operations/Tools [7]. This project considers mainly Compute and Storage services as we are using them to deploy our application.

7.3.1. An Overview of Google Cloud Platform

Any GCP resources that are allocated and used must belong to a project, which represents an organizing entity. A project is compromised of the settings, permissions, and other metadata that describe applications. Each GCP project has a project name provided by a user, a unique project ID provided either by user or GCP, and a project number provided by

GCP [7]. These identifiers are used in command lines and API calls. The following screenshot shows a project name, its ID, and number (refer to Fig. 3).

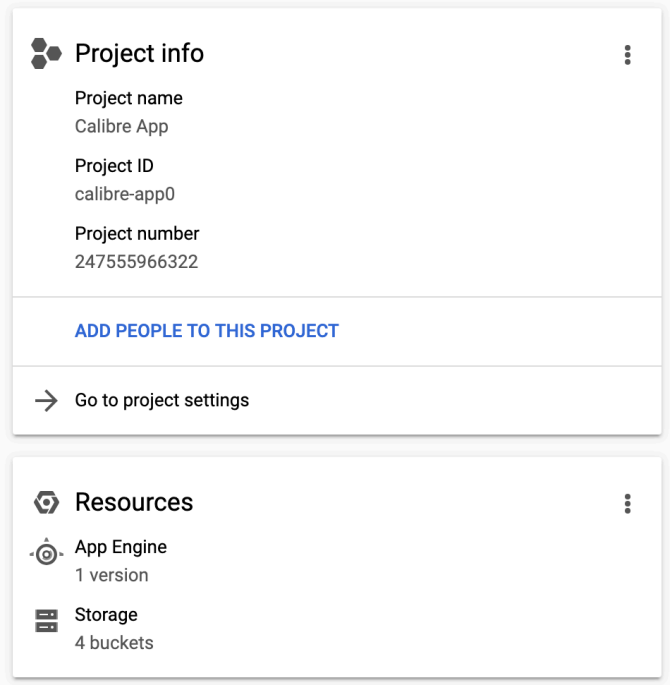


Fig. 3. Screenshot with a project name, its ID, and number

GCP provides three ways to interact with the services and resources.

- The *GCP Console* provides a web-based, graphical user interface that can be used to manage GCP projects and resources. In the console, a user can create projects, or choose an existing project, and use the resources in the context of that project (please refer to Figure 4).
- GCP is accessible using command-line interface. *Google Cloud SDK* is a set of tools that is used to manage resources and applications hosted on Google Cloud Platform [15]. These include the `gcloud`, `gsutil`, and `bq` command line tools [15]. The `gcloud` tool can be used to manage both development workflow and GCP resources (please refer to Figure 5).
- GCP also provides *Cloud Shell*, a browser-based, interactive shell environment for GCP that can be accessed from the GCP console (please refer to Figure 6).

7.3.2. An Overview of Google App Engine

Google App Engine is GCP's platform as a service (PaaS) for building scalable applications [7]. With App Engine, Google handles most of the management of the resources that makes deployment, maintenance, and scalability easier and allows to focus on the development by managing the application infrastructure [7]. It is especially suited for building scalable web applications. App Engine provides developers with built-in services and APIs such as NoSQL datastore, application logging, and a user authentication API [7].

7.3.2.1 Components of an Application

An App Engine app includes a single application resource that consists of one or more services [16]. Each service can be configured to use different runtimes and to operate with different performance settings. Within each service, it is possible to deploy versions of that service [16]. Each version runs within one or more instances, depending on how much traffic it is configured to handle [16].

App Engine app is created under GCP project when creating an application resource [16]. The App Engine application is a top-level container that includes the service, version, and instance resources [16]. The following diagram illustrates the hierarchy of an App Engine app running with multiple services (refer to Fig. 7).

Other GCP services, for example, Cloud Datastore, are shared across an App Engine app.

7.3.2.2 Services

Generally, App Engine services behave like microservices [16]. Therefore, the application can either run as a single service or it can be designed to be deployed as multiple services running as a set of microservices. [16]. Each App Engine app or service consists of the source code and the corresponding App Engine configuration files [16]. It is worth mentioning that the set of files that is deployed to a service represents a single version of that service and additional versions are created within that same service each time it is deployed.

7.3.2.3 Microservices Architecture on Google App Engine

Microservices refers to an architectural style for developing applications and allow a large application to be decomposed into independent constituent parts, with each part having its own realm of responsibility [17]. For example, to serve a single user or API request, a microservices-based application might call many internal microservices to compose its response [17]. With App Engine, code can be deployed to services independently, and different services can be written in different languages, such as Python, Java, Go, and PHP [17].

Furthermore, each service can have multiple *versions* deployed simultaneously. This structure allows testing a new version, A/B testing between different versions and roll-forward and rollback operations [17].

It is important to note while mostly isolated, services share some App Engine resources. For example, Cloud Datastore is all shared resources between services in an App Engine project [17]. These are shown in the following diagram (refer to Fig. 8).

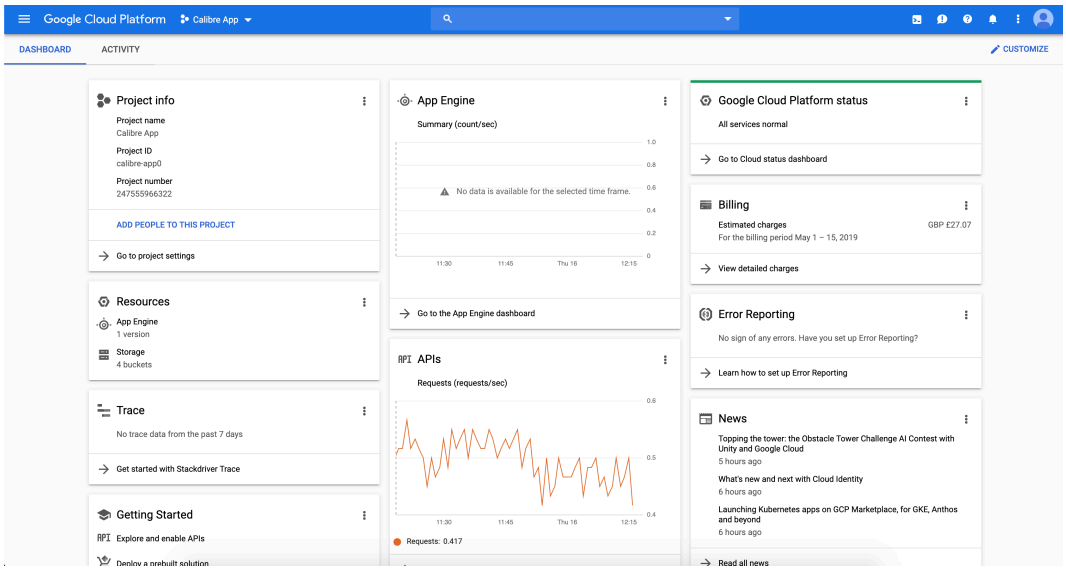


Fig. 4. Screenshot of the Google Cloud Platform Console

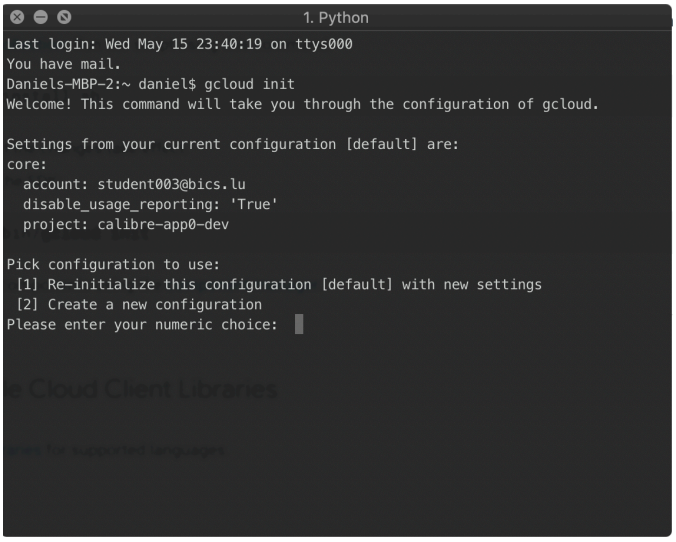


Fig. 5. Screenshot of the Google Cloud SDK

Fig. 6. Screenshot of the Cloud Shell

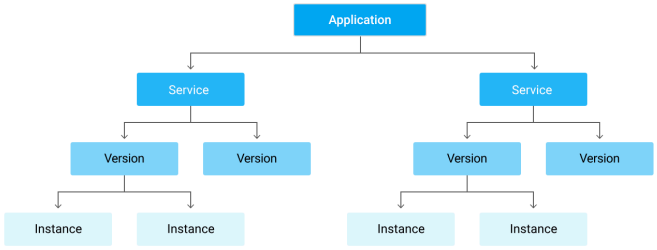


Fig. 7. Components of an application. Figure source: [16]

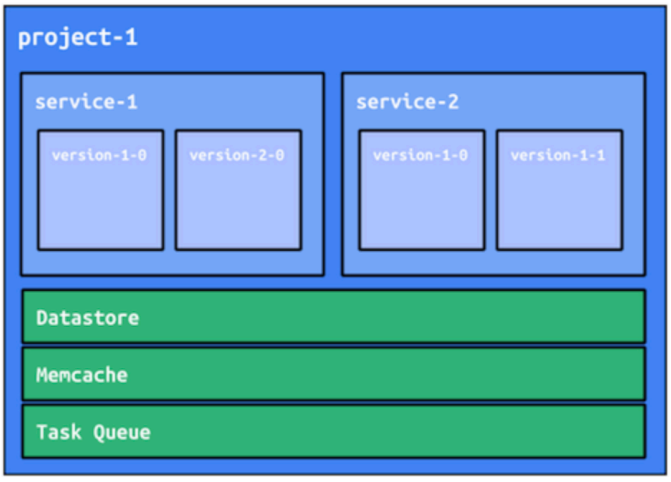


Fig. 8. App Engine Services as microservices. Figure source: [17]

7.3.2.4 Configuration Files

Each version of a service is defined in a *.yaml* file, which includes the name of the service and version [18]. Each service in an application must have its own *.yaml* file, which acts as a descriptor for its deployment. It is important to first create the *app.yaml* file for the default service before creating and deploying *.yaml* files for additional services within an application.

Generally, a directory for each service is created, which contains the service's YAML files and associated source code [18]. For simple projects, all the application's files can be placed in one directory as follows (refer to Fig. 9).

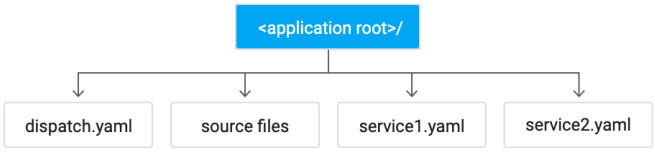


Fig. 9. Configuration Files. Figure source: [18]

7.3.2.5 Creating a GCP project for App Engine

Before running and deploying the application, we should install the Cloud SDK on the local machine and then set up a GCP project for App Engine.

In this project, we will be focusing on creating a project using Cloud SDK, but it is also possible to do so by using either GCP Console or Cloud Shell. We use the following command in order to create new project: `gcloud projects create [YOUR_PROJECT_NAME] --set-as-default` [23]. Then, we run the following command to select a region and create an App Engine application: `gcloud app create --project=[YOUR_PROJECT_NAME]` [23]. Then, we are required to install the following prerequisites: Git and gcloud component that includes the App Engine extension for Python by using the following command: `gcloud components install app-engine-python` [23]. Finally, we need to prepare our environment for Python development. It is recommended that we have the latest version of Python, `pip`, `virtualenv` and other related tools installed on our local machine [23].

Later in this paper, we will be focusing on how to deploy an app, make a change and view an application.

7.4. Assessment

All the functional and non-functional requirements have been met for this deliverable. Specifically, we have looked into Google Cloud Platform fundamentals and provided computing services. We have also interacted with Google Cloud Platform tools and services as well as developed an understanding of the value and purpose of GCP services.

8. A Technical Deliverable 2: A Python/Flask CRUD Web App for Viewing, Storing, Modifying and Retrieving Book Items

8.1. Requirements

The main objective is to develop a runnable implementation of the bookshelf 'toy' case study. This case study, written in Python and Flask framework, is seen as a simple CRUD application that includes authentication. The sample app is based on the Flask web application framework, but the concepts and technologies explored are applicable regardless of which framework one uses. The application should include in itself the range of the most appropriate GCP products such that they provide an efficient, scalable and reliable services for running the app.

The sample app stores a collection of book titles. Anyone with access to the app can modify the list. The following functional requirements should be achieved:

The functional requirements for end-user include:

- View the list of books.
- Add books to the list.
- Remove books from the list.
- Edit book details.
- Upload cover images for books.
- Sign in with a Google Account and view the books that they added to the list.

The functional requirements for developers include:

- Clone or download the sample app.
- Build the app and run it on the local machine.
- Deploy the app to App Engine.
- Walk through the sample code.
- Learn how the app stores structured data in Cloud Datastore.
- Learn how the app stores binary data in Cloud Storage.
- Learn how the app authenticates users using OAuth2.
- Learn how to deploy directly from Cloud Source Repositories
- Learn how to deploy an application stored in Cloud Source Repositories to App Engine when there is a new commit.

8.2. Design

First, to get a better understanding of the application's architecture to be developed and actual GCP services used, we will follow a simple web app tutorial provided by Google [22]. The bookshelf app is a sample web app written in Python that shows how to use various GCP products, including: App Engine flexible environment, Cloud Datastore, and Cloud Storage.

To be able to start programming on the project a set of different software tools was a prerequisite. First thing to do was the installation of the integrated development environment. For this project, we choose to work with Visual Studio Code as it provides a simple and intuitive interface as well as support for version control systems.

We will also follow a series of tutorials provided by Google showing how to deploy directly from Cloud Source Repositories as well as how to automate App Engine deployments with Cloud Build in order to automatically deploy an application stored in Cloud Source Repositories to App Engine when there is a new commit.

Then, we need to install Cloud SDK, which provides a set of tools that we can use to manage resources and applications

hosted on Google Cloud Platform. These include the gcloud command line tools, which we use mainly for authentication in GCP, managing GCP projects and deployment of app. Other prerequisites include installing Git, as well as preparing our environment for Python development.

8.3. Production

For setting up our GCP project, we used the GCP Console in which we have created a GCP project, an App Engine app, and then enabled billing for that project. Then, we have also enabled the Cloud Datastore Cloud Storage, and Logging APIs.

8.3.1. Application Architecture

The architecture of the application is decomposed into several parts:

- The structured data part deals with how app stores book information in a NoSQL database. The app's web page displays a form where the user enters the title, author, description, and publication date of a book using Flask web framework. The app stores this information in a database so it can be retrieved later for viewing or editing. For this part, we will be using Cloud Datastore as it provides zero-configuration, fully managed, highly scalable, non-relational database [22].
- The Cloud Storage part deals with how the sample app stores binary data in Cloud Storage. The app stores the cover images, that can be specified by the user for each book in a Cloud Storage bucket.
- The authorization part deals with how the sample app provides a sign-in flow for the user and provides users with personalized functionality. When a user is signed in, any books that the user enters are associated with the user and can't bee seen by other users. Signed-in users have access to their own books and books that are created by anonymous users.

The Fig. 10 shows the overview of the application's architecture in detail. The application serves content using `.html` files. The creation, retrieval, modifications or deletions are communicated through `crud.py` module, which in turn calls `model_datastore.py` and `storage.py` modules, that update the structured and unstructured data in Cloud Datastore and Cloud storage.

8.3.1.1 Configuration Settings

The `app.yaml` file specifies App Engine app's settings, including the Python application's runtime configuration (please refer to Listing 1). The runtime also looks for a `requirements.txt` file in application's source directory and uses `pip` to install any dependencies locally before starting application [24].

Our app also uses `config.py`, which contains all of the configuration values for the application.

8.3.1.2 Using Cloud Datastore

The app stores its persistent data, metadata for books, in Cloud Datastore. Cloud Datastore is a fully managed service that is automatically initialized and connected to the App Engine app [22]. To use Cloud Datastore, we simply edit the `config.py` file and set the following values: Set the value of `[PROJECT_ID]` to our project ID, and set the value of `[DATA_BACKEND]` to datastore.

8.3.1.3 Using Cloud Storage

To use Cloud Storage to store and serve unstructured binary data, we need to create need a high-level container, or bucket, for binary objects. Buckets are the basic containers that hold data in Cloud Storage [22]. In a terminal window, we need enter the following command: `gsutil mb gs://[YOUR_BUCKET_NAME]` and then set the bucket's default access control list (ACL) to public-read by using `gsutil defacl set public-read gs://[YOUR_BUCKET_NAME]` command in order to view uploaded images in the app. In order to use the created bucket, we need to replace the value of `[CLOUD_STORAGE_BUCKET]` in `config.py` file to our Cloud Storage bucket name.

The application uses Cloud Storage to store binary data, pictures in this case, while continuing to use a structured database, in our case, Cloud Datastore, or either Cloud SQL, or MongoDB, to store book information (please refer to Figure 12).

8.3.2. Implementation Details

In this section, we describe in detail how specific functional requirements were met.

8.3.2.1 Handling User Submissions with Forms

The HTML form to serve the web content in our app is created using Flask's template engine, Jinja2, which is a Python template engine [25] (please refer to Listing 2). The following Jinja2 template specifies that the form includes text input fields for *Title*, *Author*, *Date Published*, and *Description* (please refer to Figure 13).

1) When a user clicks Add Book, the `crud.py` `add()` function view displays the form (please refer to Listing 3).
2) When a user fills out this form and then clicks Save, the same function handles the form's HTTP POST action.
3) This action initiates the process of sending the submitted data to Cloud Datastore by passing the data to the `get_model().create` function.

The `model_datastore.py` file contains the code that performs CRUD functions for data stored in Cloud Datastore (please refer to Listing 4). For example, the `get_model().create` statement calls the create function in `model_datastore.py`, which sends the user's submitted data to the update function.

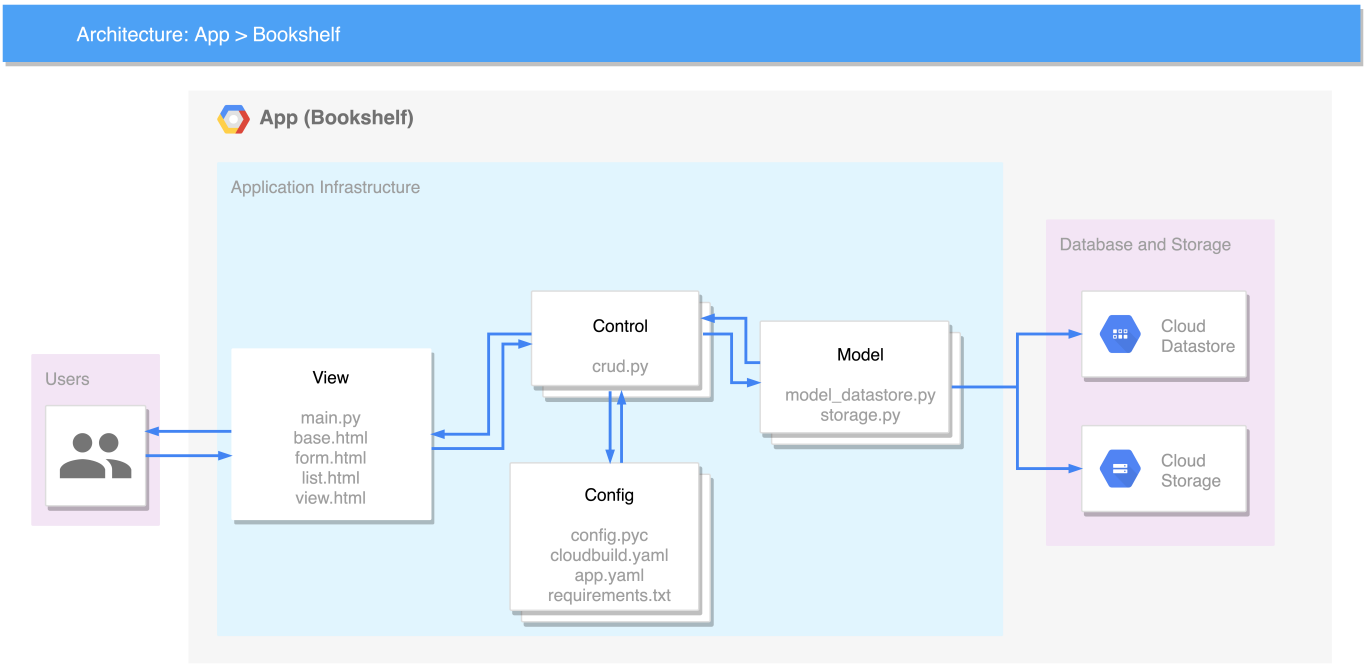


Fig. 10. Architecture of the Bookshelf application

```
runtime: python
env: flex
entrypoint: gunicorn -b :$PORT main:app

runtime_config:
  python_version: 3
```

Listing 1. app.yaml file

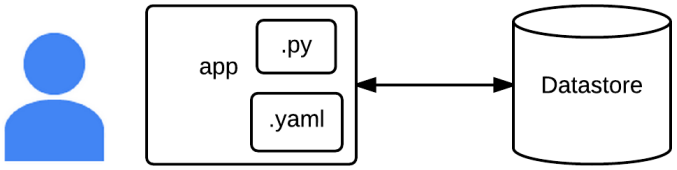


Fig. 11. App Structure. Figure source: [22]

After users have added books, clicking the Books link redirects to the `/books` page, which lists all of the books currently stored in Cloud Datastore. The `model_datastore.list` function is responsible for listing all of the books using data retrieved from Cloud Datastore (please refer to Listing 5).

8.3.2.2 Handling user uploads

The Flask framework has built-in functionality to parse file uploads and it makes the file object available in the files field of the request object (please refer to Listing 2).

The `upload_image_file` call dispatches to a helper function, also in `crud.py`, that calls the `storage.upload_file` function with the appropriate Cloud Storage bucket to handle uploading the files to Cloud Storage and allowed extensions defined in the config file (refer to Listing 6 and Listing 7).

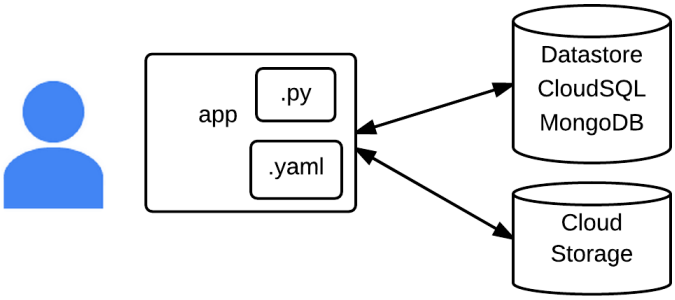


Fig. 12. App Structure. Figure source: [22]

The uploading to Cloud Storage is done using `upload_file` function as shown in Listing 8.

The `upload_file` function returns the public URL of the image that is returned by Cloud Storage and that is then saved to the database. Serving the image directly from Cloud Storage is therefore straightforward because the URL for the image is public as shown in Listing 9.

8.3.2.3 Authenticating Users

By using Google Identity Platform, we can access users information and ensure that their sign-in credentials are securely managed by Google [22]. In this app, we use OAuth 2.0 as it provides a sign-in flow for users and provides basic access to profile information about authenticated users [22].

First, we have created a web app client ID that lets the app authorize users and access Google APIs. To do so, we have created OAuth client ID credentials using GCP Console Credentials page as shown in Figure 14. Then, we need to use

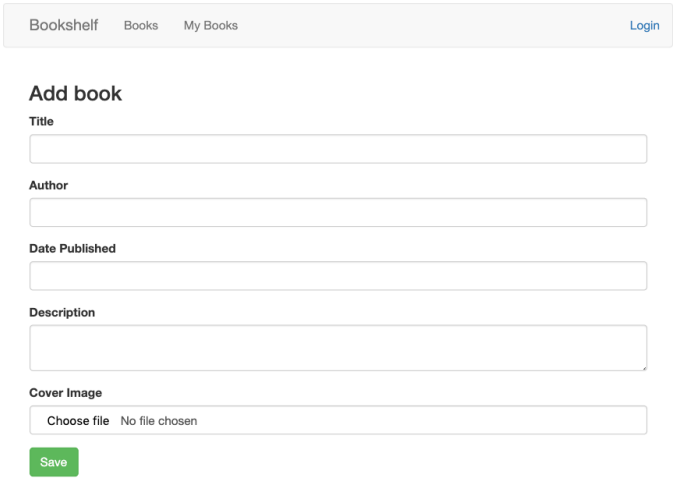
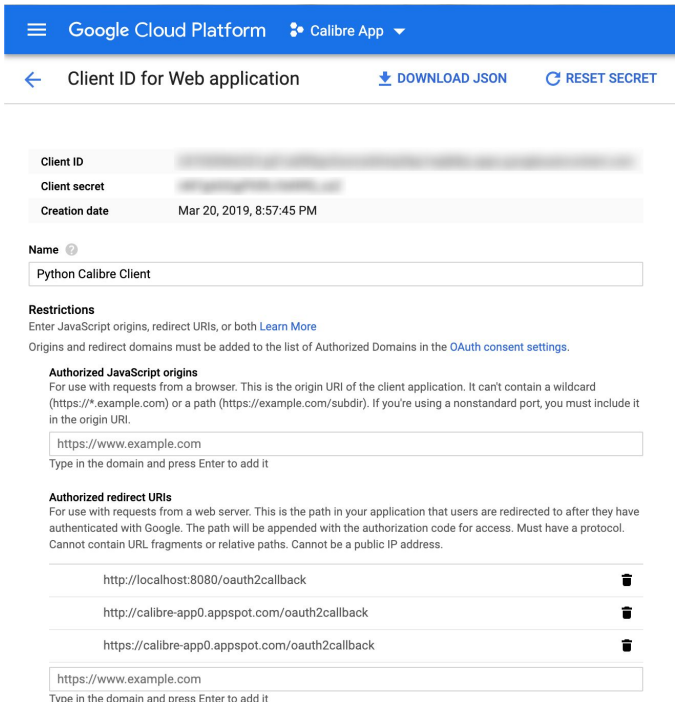


Fig. 13. The add/edit HTML form that lets users add and edit book submissions in the app

```
{% extends "base.html" %}

{% block content %}
<h3>{{action}} book</h3>

<form method="POST" enctype="multipart/form-data">

  <div class="form-group">
    <label for="title">Title</label>
    <input type="text" name="title" id="title" value
        ="{{book.title}}" class="form-control"/>
  </div>

  <div class="form-group">
    <label for="author">Author</label>
    <input type="text" name="author" id="author"
        value="{{book.author}}" class="form-control"
        />
  </div>

  <div class="form-group">
    <label for="publishedDate">Date Published</label
        >
    <input type="text" name="publishedDate" id="
        publishedDate" value="{{book.publishedDate}}
        " class="form-control"/>
  </div>

  <div class="form-group">
    <label for="description">Description</label>
    <textarea name="description" id="description"
        class="form-control">{{book.description}}</
        textarea>
  </div>

  <div class="form-group">
    <label for="image">Cover Image</label>
    <input type="file" name="image" id="image" class
        ="form-control"/>
  </div>

  <div class="form-group hidden">
    <label for="imageUrl">Cover Image URL</label>
    <input type="text" name="imageUrl" id="imageUrl"
         value="{{book.imageUrl}}" class="form-
        control"/>
  </div>

  <button type="submit" class="btn btn-success">Save
      </button>
</form>

{% endblock %}
```

Listing 2. form.html file

the given application client ID and client secret and modify the `config.py` file. In the OAuth2 configuration section, we need to set the values of `GOOGLE_OAUTH2_CLIENT_ID` and `GOOGLE_OAUTH2_CLIENT_SECRET` to the new ones.



```
@crud.route('/add', methods=['GET', 'POST'])
def add():
    if request.method == 'POST':
        data = request.form.to_dict(flat=True)

        book = get_model().create(data)

        return redirect(url_for('.view', id=book['id
            ']))

    return render_template("form.html", action="Add"
        , book={})
```

Listing 3. crud.py file

```
def update(data, id=None):
    ds = get_client()
    if id:
        key = ds.key('Book', int(id))
    else:
        key = ds.key('Book')

    entity = datastore.Entity(
        key=key,
        exclude_from_indexes=['description'])

    entity.update(data)
    ds.put(entity)
    return from_datastore(entity)


create = update
```

Listing 4. model_datastore.py file

The following diagram Figure 15 shows the application's components and how they interact with one another.

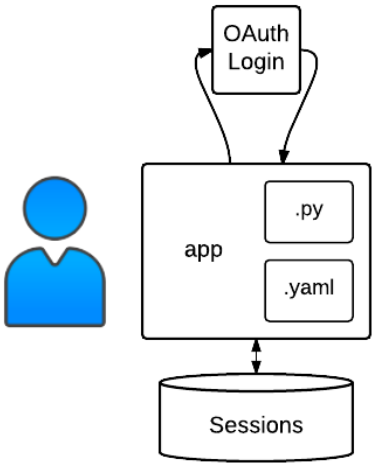


Fig. 15. App Structure. Figure source: [22]

Flask provides sessions backed by encrypted cookies, that provides the capability to store information about the current user in a session [22]. The web service flow for authenticating the user involves redirecting the user to Google's authorization service, processing the response by Google and then redirecting the user back to the app [22].

```
def list(limit=10, cursor=None):
    ds = get_client()

    query = ds.query(kind='Book', order=['title'])
    query_iterator = query.fetch(limit=limit,
        start_cursor=cursor)
    page = next(query_iterator.pages)

    entities = builtin_list(map(from_datastore, page
        ))
    next_cursor = (
        query_iterator.next_page_token.decode('utf-8
            ')
        if query_iterator.next_page_token else None)

    return entities, next_cursor
```

Listing 5. model_datastore.py file

```
image_url = upload_image_file(request.files.get('
    image'))
```

Listing 6. crud.py file

8.3.3. Testing Locally and Deploying the App

Before moving forward, it is essential to know how to test the application using the local development server, make a change to the code and deploy the app.

8.3.3.1 Local

1) First, we run the app on the local machine. To do so, we start the local development server on local machine with the following command: `python main.py` [22].
2) The local development server is now running and listening for requests on port 8080. In our browser, we need to enter the following address to view the app: `http://localhost:8080/` [22]. We should now be able to browse the app's web pages to add, edit, and delete books.
3) It is worth mentioning that we can leave the development server running while developing an application. The development server watches for changes in the source files and reloads them if necessary. We simply need to edit the source code and reload `http://localhost:8080/` to see the results.

8.3.3.2 GCP

1) In order to deploy the sample app to App Engine, we need to run the following command from within the root directory of the application where the app.yaml file is located: `gcloud app deploy` [22].
2) We should then visit `https://[YOUR_PROJECT_ID].appspot.com` in our browser, where `[YOUR_PROJECT_ID]` should be replaced with the GCP project ID [22]. We can also use `gcloud app browse` to launch a browser and view the app at this address [23].

```python
def upload_image_file(file):
    """
    Upload the user-uploaded file to Google Cloud
        Storage and retrieve its
    publicly-accessible URL.
    """
    if not file:
        return None

    public_url = storage.upload_file(
        file.read(),
        file.filename,
        file.content_type
    )

    current_app.logger.info(
        "Uploaded file %s as %s.", file.filename,
            public_url)

    return public_url
```

Listing 7. crud.py file

```python
def upload_file(file_stream, filename, content_type)
    :
    """
    Uploads a file to a given Cloud Storage bucket
        and returns the public url
    to the new object.
    """
    _check_extension(filename, current_app.config['
        ALLOWED_EXTENSIONS'])
    filename = _safe_filename(filename)

    client = _get_storage_client()
    bucket = client.bucket(current_app.config['
        CLOUD_STORAGE_BUCKET'])
    blob = bucket.blob(filename)

    blob.upload_from_string(
        file_stream,
        content_type=content_type)

    url = blob.public_url

    if isinstance(url, six.binary_type):
        url = url.decode('utf-8')

    return url
```

Listing 8. storage.py file

3) If we deploy our app again using the same command, we deploy its updated version. The new deployment creates a version of the app and promotes it to the default version. The earlier versions of the app remain, as do their associated VM instances, which can be seen in *Versions* screen in the GCP Console corresponding App Engine window.

8.3.4. Cloud Source Repositories

To make collaboration on this project easier and more secure, we manage the code on Cloud Source Repositories, a scalable, private Git repository [26]. Google Cloud Source Repositories provides Git version control to support collaborative development of an application or service [26].

Specifically, we have focused on the 'Quickstart for deploying from Cloud Source Repositories to Google App Engine' and 'Quickstart for creating a new repository' tutorials provided by Google [26] [27].

```html
<div class="media-left">
  {% if book.imageUrl %}
    <img class="book-image" src="{{book.imageUrl}}">
  {% else %}
    <img class="book-image" src="http://placekitten.
        com/g/128/192">
  {% endif %}
</div>
```

Listing 9. view.html file

We first create create a new GCP repository in either GCP Console or Cloud SDK. We will use Cloud SDK. We use `gcloud source repos create [YOUR_REPOSITORY_NAME]` command. Then, we clone the contents of the GCP repository into a local Git repository using `gcloud source repos clone [YOUR_REPOSITORY_NAME]` command. Let's assume we are editing `main.py` file. A simple workflow can be described as follows: First, we add the modified file using `git add main.py` so Git can commit it. Then, we commit the file using `git commit -m "Update main.py"`. Finally, we add the contents of the local Git repository to Cloud Source Repositories using push command `git push origin master`. Now, the repository can be accessed from within the Cloud Source Repositories using Source Browser [26] as shown in Figure 16.

We then set up a trigger that automatically deploys an application stored in Cloud Source Repositories to App Engine when there is a new commit using the Quickstart for automating App Engine deployments with Cloud Build tutorial provided by Google as outlined here [28].

8.4. Assessment

We have successfully built and deployed a Google App Engine app by using the resources and tutorials provided by Google. We have learned about the architecture of the application, as well as how various GCP services were used in specific parts of the application both during the development, building and deploying the application. By following this guideline, it is possible to replicate this application.

9. A Technical Deliverable 3: Jira: Agile Project Management Tool

9.1. Requirements

This deliverable aims to use Jira tracking system in order to facilitate the software development process in this project. It is also essential to learn the basics of Jira application as well as related concepts and terminology.

It is expected that by using the Jira tool we would produce specific statistical data about the project's progress as well as history logs related to the project. This may be particularly helpful in the future, as we could use this data to identify potential bottlenecks and determine points of improvements.

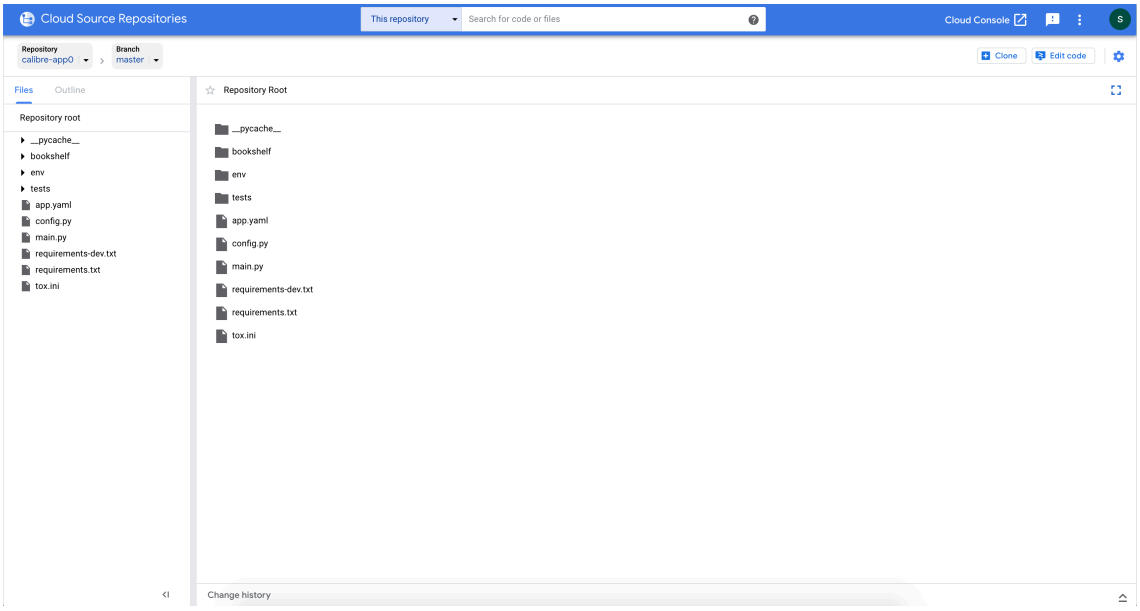


Fig. 16. Quickstart for creating a new repository. Source Browser

9.2. Design

In our project, we used Jira in order to track issues, bugs and other tasks related to the project. This allowed us to develop an understanding of related concepts and acquire hands-on experience with the tool.

Fig. 18. An example of the default workflow of Jira. Figure source: [21]

9.3. Production

Jira application is a workflow management system that is used including running projects, tracking assets, and other tasks that requires work moving through a workflow (please refer to Figure 17) [20]. In Jira, all project's tasks will be logged into a Kanban board, and each one will go through a number of workflows (processes), which allows to control its status and progress tasks in the system.

9.3.1. Workflows

Jira compromises of workflows, moving packets of work from A to B. Workflow dictates how an issue can be progressed in a project [21]. Workflows are often modeled on existing processes, and are made up of statuses (or steps) and transitions (movements between statuses) [21]. When an issue is created, it will automatically be assigned workflow and a status on that workflow [21]. An example of the default workflow that is used in Jira is shown below in Figure 18.

9.3.2. Projects

Projects are a way to group the issues, and apply a set of defaults that make sure all issues have the information that they need to be progressed and tracked through the workflow [21].

9.3.3. Issues

Issues are the work packets in Jira [21]. Each issue can be further defined by assigning the issue an issue type [21]. It may also include description, priority level, notifications, and user entry screens as shown in Figure 19. Issues then progress (or move) through Jira via an associated workflow.

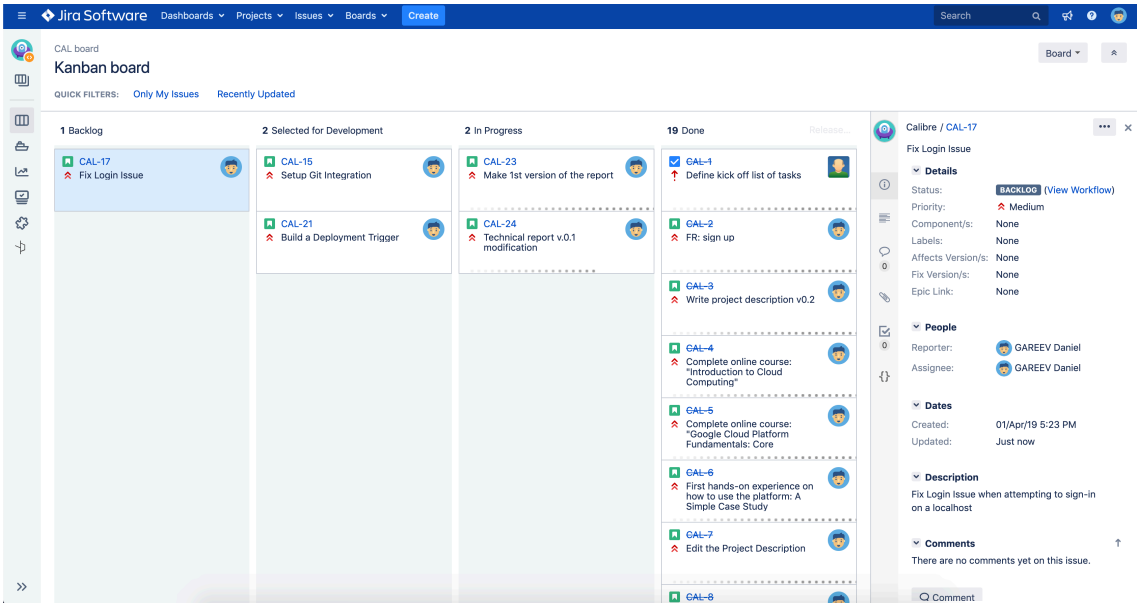


Fig. 17. Jira Kanban board

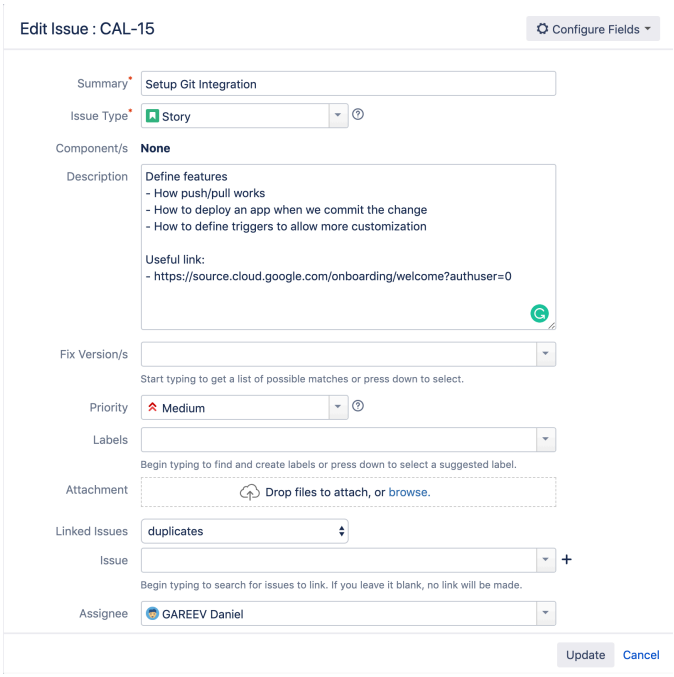


Fig. 19. Jira issue example

9.3.4. Report Log

The Figure 20 below shows the statuses of issues over time for our project. As a general trend, we can see that we have sustained the project on track and achieved stable and short cycle time for the product. This data can be used to determine future performance.

9.4. Assessment

In this project, we have successfully used Jira workflow management tool that helped us to manage tasks, resources and monitor the project's progress. We have also learned related concepts and terminology, that can later be used in similar tasks. It is evident from the provided report, that the tool allowed us to keep the project on track. We have therefore improved project tracking, tasks handling and communication between the team members.

10. Conclusion

In this project, we learned a lot about cloud computing, Google Cloud Platform and related concepts such as agile project management and software engineering environments.

Our project demonstrates that we have successfully build a basic understanding of the concepts related to the cloud computing, as well as developed and deployed the application using cloud computing services offered by Google.

While the results are satisfactory, additional improvements are possible, and further research and experiments are welcomed.

Overall, we are satisfied with our project and consider it successful.

References

[1] Introduction to Cloud Computing Retrieved from https://www.udemy.com/introduction-to-cloud-computing/.
[2] About Python https://www.python.org/about/
[3] Git. "Getting Started - About Version Control." git-scm.com/book/en/v1/Getting-Started-About-Version-Control.
[4] PMI (2010) A Guide to the Project Management Body of Knowledge Retrieved from http://www.cs.bilkent.edu.tr/ cagatay/cs413/PMBOK.pdf

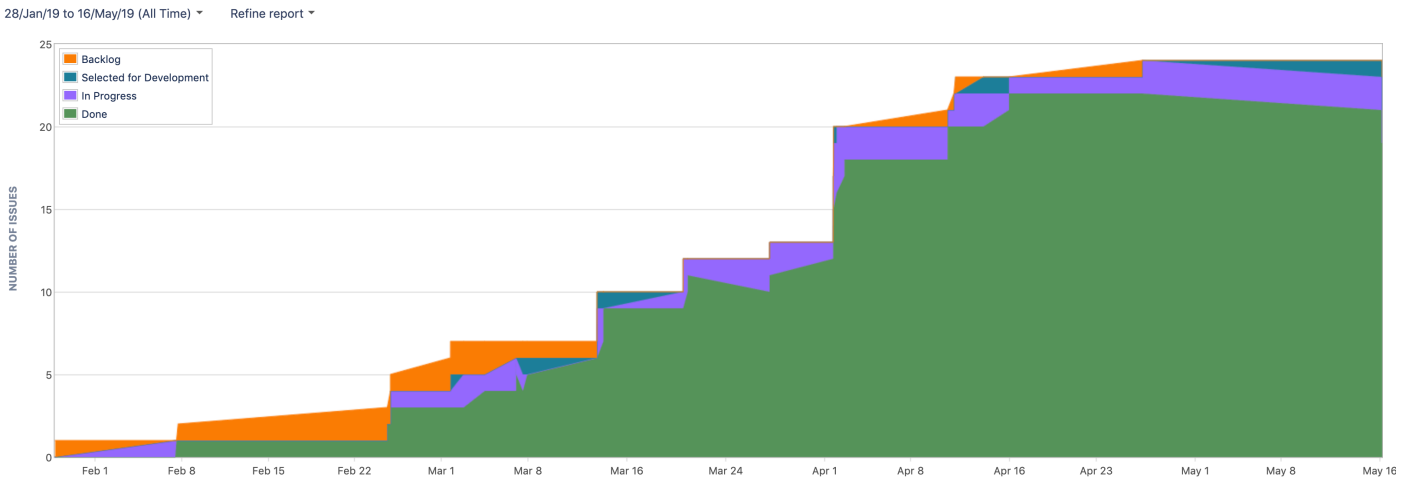


Fig. 20. Cumulative Flow Diagram

[5] JoSEP, Anthony D., et al. "A view of cloud computing." Communications of the ACM 53.4 (2010). Retrieved from http://www.academia.edu/download/34578652/a_view_of_cc.pdf

[6] Mell, Peter, and Tim Grance. "The NIST definition of cloud computing." (2011) Retrieved from https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf

[7] Google Cloud Platform Fundamentals: Core Infrastructure https://www.coursera.org/learn/gcp-fundamentals

[8] Turnkey Technologies. Microsoft Dynamics in the cloud - Evaluating IaaS, PaaS and SaaS. Retrieved from http://www.erpsoftwareblog.com/2016/09/hosting-microsoft-dynamics-cloud-evaluating-iaas-paas-saas/

[9] Dart, Susan A., et al. "Overview of software development environments." (1992).

[10] Creating and Managing Projects https://cloud.google.com/resource-manager/docs/creating-managing-projects

[11] Naming Developer Environments https://cloud.google.com/appengine/docs/standard/go/creating-separate-dev-environments

[12] Augustine, Sanjiv, et al. "Agile project management: steering from the edges." Communications of the ACM 48.12 (2005) Retrieved from https://people.eecs.ku.edu/ hossein/811/Papers/Agility/agile-manage-edges.pdf

[13] Kanban vs. Scrum https://www.atlassian.com/agile/kanban/kanban-vs-scrum

[14] The Scrum Guide https://www.scrumguides.org/scrum-guide.html

[15] Google Cloud SDK documentation https://cloud.google.com/sdk/docs/

[16] An Overview of App Engine https://cloud.google.com/appengine/docs/standard/python/an-overview-of-app-engine

[17] Microservices Architecture on Google App Engine https://cloud.google.com/appengine/docs/standard/python/microservices-on-app-engine

[18] Configuration Files https://cloud.google.com/appengine/docs/standard/python/configuration-files

[19] Getting started with JIRA Core. https://confluence.atlassian.com/jiracoreserver073/getting-started-with-jira-core-861255635.html

[20] Managing your workflows https://confluence.atlassian.com/adminjiraserver/managing-your-workflows-938847410.html

[21] What makes up JIRA Core? https://confluence.atlassian.com/jiracoreserver073/what-makes-up-jira-core-861255606.html

[22] Python Bookshelf app https://cloud.google.com/python/getting-started/tutorial-app

[23] Quickstart for Python App Engine Standard Environment https://cloud.google.com/appengine/docs/standard/python/quickstart

[24] The Python Runtime https://cloud.google.com/ appengine/docs/flexible/python/runtime

[25] Jinja2 - The Python Template Engine http://jinja.pocoo.org/

[26] Quickstart for deploying from Cloud Source Repositories to Google App Engine https://cloud.google.com/source-repositories/docs/quickstart-deploying-from-source-repositories-to-app-engine

[27] Quickstart for creating a new repository https://cloud.google.com/source-repositories/docs/quickstart

[28] Quickstart for automating App Engine deployments with Cloud Build https://cloud.google.com/source-repositories/docs/quickstart-triggering-builds-with-source-repositories

11. Appendix

11.1. Basic Cloud Computing Terms

11.1.1. Defenition of Cloud Computing

Cloud computing. Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction." [6]

11.1.2. Essential Characteristics

On-demand self-service. "A consumer can unilaterally provision computing capabilities, such as server time and network storage, as needed automatically without requiring human interaction with each service provider." [6]

Broad network access. "Capabilities are available over the network and accessed through standard mechanisms that promote use by heterogeneous thin or thick client platforms (e.g., mobile phones, tablets, laptops, and workstations)." [6]

Resource pooling. "The provider's computing resources are pooled to serve multiple consumers using a multi-tenant model, with different physical and virtual resources dynamically assigned and reassigned according to consumer demand. There is a sense of location independence in that the customer generally has no control or knowledge over the exact location of the provided resources but may be able to specify location at a higher level of abstraction (e.g., country, state, or datacenter). Examples of resources include storage, processing, memory, and network bandwidth." [6]

Rapid elasticity. "Capabilities can be elastically provisioned and released, in some cases automatically, to scale rapidly outward and inward commensurate with demand. To the consumer, the capabilities available for provisioning often appear to be unlimited and can be appropriated in any quantity at any time." [6]

Measured service. "Cloud systems automatically control and optimize resource use by leveraging a metering capability at some level of abstraction appropriate to the type of service (e.g., storage, processing, bandwidth, and active user accounts). Resource usage can be monitored, controlled, and reported, providing transparency for both the provider and consumer of the utilized service." [6]

11.1.3. Service Models

Software as a Service (SaaS). "The capability provided to the consumer is to use the provider's applications running on a cloud infrastructure. The applications are accessible from various client devices through either a thin client interface, such as a web browser (e.g., web-based email), or a program interface. The consumer does not manage or control the underlying cloud infrastructure including network, servers, operating systems, storage, or even individual application capabilities, with the possible exception of limited userspecific application configuration settings." [6]

Platform as a Service (PaaS). "The capability provided to the consumer is to deploy onto the cloud infrastructure consumer-created or acquired applications created using programming languages, libraries, services, and tools supported by the provider. The consumer does not manage or control the underlying cloud infrastructure including network, servers, operating systems, or storage, but has control over the deployed applications and possibly configuration settings for the application-hosting environment." [6]

Infrastructure as a Service (IaaS). "The capability provided to the consumer is to provision processing, storage, networks, and other fundamental computing resources where the consumer is able to deploy and run arbitrary software, which can include operating systems and applications. The consumer does not manage or control the underlying cloud infrastructure but has control over operating systems, storage, and deployed applications; and possibly limited control of select networking components (e.g., host firewalls)." [6]