# Cloud Computing Spring 2022
Assignment 2: Sorting as a Service

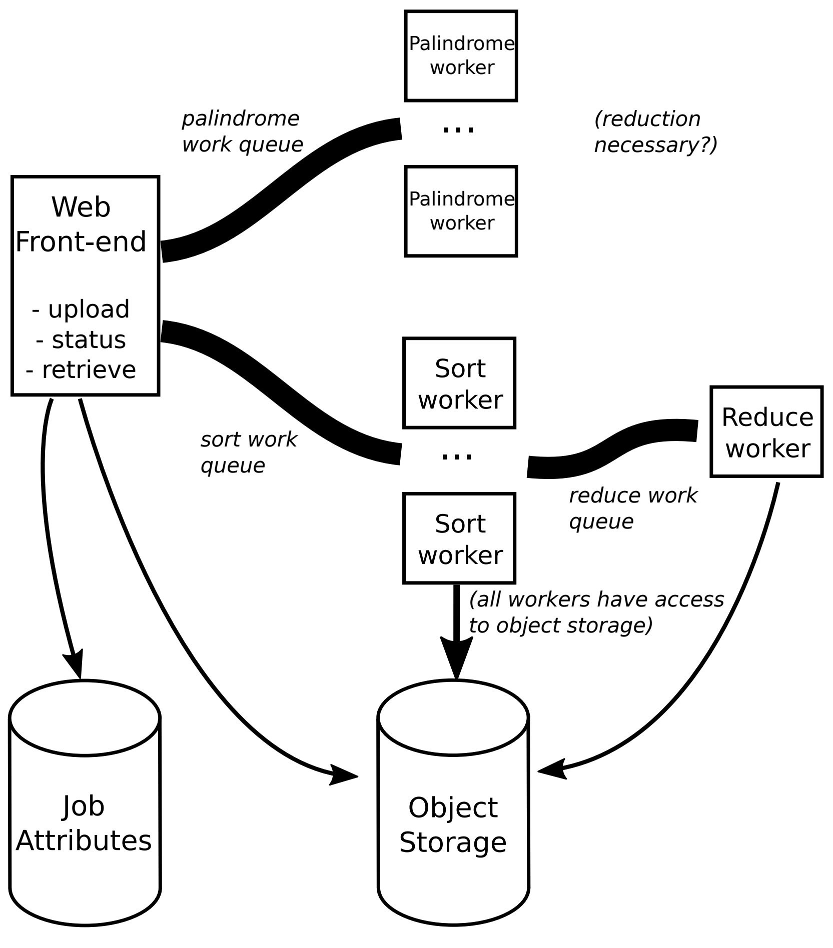
**Deadline:** Friday, June 24, 2022 ~~Tuesday, June 14, 2022~~

## 1 Aim and Scope

Whereas in the first assignment we worked on creating our own autoscaler on top of a rudimentary "IaaS environment", in this second assignment we will build a Cloud application on top of services provided by Google Cloud Platform. You will be working with typical components that are used in Cloud software such as Object storage, Kubernetes, Cloud Functions, and Pub/Sub messaging. The main objective of this assignment is to get an impression how Cloud applications are typically implemented and to gain experience using an actual Cloud platform.

The application that you will implement, named "Sorting as a Service", should present users with a web interface where a text file can be uploaded. The uploaded file is to be sorted **by lines** and the number of palindromes in the file, as well as the length of the longest palindrome found, file should be identified (e.g. "this dad is asa example 12321 enne period" contains 4 palindromes, the length of the longest is 5). The (web) front-end and back-ends are separated. All sorting and palindrome identification is to be performed by the back-end. The back-end consists of worker processes that process uploaded files in chunks. By doing so, multiple worker processes can work on multiple files as well as on parts of the same file at the same time. Additionally, because these worker processes will be stateless, the number of active worker processes can be easily scaled up and down. (In fact, we are using a MapReduce-like approach to sorting in this case).

The general architecture of the solution is as follows (all workers have access to the object store, but for the sake of clarity not all of these arrows have been drawn):



## 1.1 Components

Let us now discuss these components one by one:

1. Object storage will be used to store uploaded files, intermediate results and the final result. Within Google Cloud the object storage solution is called Google Cloud Storage. Language bindings are available for different programming languages. There is also a command line tool called gsutil that can be used to manipulate the object store.

2. Next to the actual files you also need to store jobs and job attributes. Pick a suitable database service to accomplish this. This could be Cloud SQL if you want a familiar interface. But you are also free to experiment with a NoSQL database such as Google Datastore.

3. A mechanism is needed to communicate between the different components. Upon file upload, the Web application should create messages that will instruct worker processes to sort all chunks of the file, and to identify palindromes in the file. When a worker process detects that it has just sorted the last remaining chunk, it should create a message to start the reduce phase (which will essentially perform a merge sort of all chunks). (Or if you implement a more sophisticated tree reduction, you could start reducing earlier in time).

   We suggest that you use Pub/Sub messaging as communication primitive. Pub/Sub messaging can be used from various Google services.

4. Back-end worker processes. Separate worker processes are required to perform sorting, reduction and palindrome identification. Worker processes will be listening for messages on Pub/Sub. When work is started, data to operate on will be fetched from object storage. We want to process a single file in parallel, so chunk the file and process the chunks in parallel. Develop the workers such that they don't unnecessarily download the entire object, but only the part that they need to operate on.

   Make sure to only acknowledge/remove messages from the work queue if the work has been completed. This ensures that in case the worker process crashes, the work item will be restarted by another worker process.

   The worker processes can be implemented in different ways. A first way is to implement the workers as stand-alone processes that will run in containers. A container is created for each worker process. Deploy your containers using Google Kubernetes Engine (GKE). Containers deployed on GKE have access to all Google services (see also the note in Section 4.2), including Cloud Storage and Pub/Sub. The second way is to deploy your worker processes as Cloud Functions that are, for example, triggered by messages posted on Pub/Sub.

   For the implementation of the worker processes, you are free to pick any programming language that you think is suitable. Given the data processing nature of the tasks, it might make sense to pick a compiled language instead of an interpreted one. Make sure that language bindings for the Google services are available for the programming language you would like to use.

5. The front-end Web application should have the following capabilities:

   (a) Ability to upload a text file to sort. It must be possible to upload large files (100+ MiB) without problems. For debugging, you may also want to provide a text box in which a (small) text file can be pasted. The file to sort is uploaded to the object store. Also a job descriptor is created in the job database in which the attributes of the job are maintained (number of chunks, how many chunks have been sorted, has the job completed, etc.).

   (b) After upload and submission, the user is given a job ID or token.

   (c) Using this token, it must be possible to obtain the status of the job (completed yes or no, number of chunks that have been processed (so % complete)).

(d) Once the job has been completed, the user can access the result using the token. The amount of identified palindromes and the length of the longest palindrome are shown to the user on the web page and additionally the user can download the resulting sorted file.

You may develop this web front-end in a programming language of choice. It is also up to you to decide how to deploy it. You can either deploy the web-frontend as a container in a Kubernetes cluster, in which case the web front-end port is exposed as a service using an external IP. You could run the web front-end in a VM. It is also possible to develop the web front-end using Google App Engine.

### 1.2 Parallel Processing

An important objective of the assignment is that a single file can be processed in parallel. To accomplish this an object needs to be split in parts or chunks. One option is to pick a chunk size such that workers can easily compute the file offset if they are given a chunk index. A line of a text file may cross the chunk boundary, however. Given that you need to sort line by line, you need to take this into account. So, the first chunk that needs to be sorted spans from the very first byte until the first new line found in the next chunk. The second chunk spans from the first newline found in the second chunk, until the first newline found in the third chunk, and so on. This does mean that a worker process needs to read a full chunk and (part of) the subsequent chunk from the object store.

### 1.3 Design Requirements

Another important objective of this assignment is to become familiar with different Cloud APIs and structures. Because of this, your application and architecture design must adhere to the following requirement: both Google Kubernetes Engine (to deploy a container) and Google Cloud Functions should be used. It is up to you what technology to use for which component. Document your motivation for the design in the report that will accompany your submission.

### 1.4 Experiment

Finally, we would like to perform a preliminary experiment to see whether scaling up the worker processes has a positive effect on the performance. Devise one or more test scenarios, such as processing a certain set of large files. In both GKE and Cloud Functions a maximum number of instances can be configured. Perform the experiment with different settings for this maximum number of instances. Another parameter that could be considered is the chunk size. Compare the obtained performance. Make sure to repeat each experiment at least three times. Given the time constraints, it is likely not possible to account for all effects of performance variability, so this is <u>not</u> required. As said, this is to be a preliminary experiment after all.

### 1.5 Report

Your submission must be accompanied by a report with the following contents. First, describe the setup and architecture of the solution that you have implemented. Motivate the technology choices and any implementation choices you have made. Also describe how to deploy your application. Finally, report on the experiment that you have performed.

## 2 Interfacing with Google Cloud Platform

In this assignment, you will use services available on Google Cloud Platform and also deploy your application within Google Cloud. You will be provided with a coupon code that gives you $50 of Google Cloud credit. Instructions on how to redeem this credit will be provided in e-mail. A

Google account is required. You can redeem the credit using an existing account, or create a new Google account. The credit remains valid for about a year. If you have credit remaining after this assignment, you can continue to explore Google Cloud or take advantage of several of the tutorials that are available.

In order to work with Google Cloud, the initial interface is the web-based dashboard or cloud console. The first thing to do is to create a project. When you access the cloud console for the first time, this is done automatically. Next, you will need to enable APIs that you would like to use in your project. This can be done on a case-by-case basis and sometimes happens automatically once you start using a certain service.

When you are going to write software that needs to interface with various Google Cloud services, it is useful to also have a command-line environment in which you can try things out. There are various ways this can be achieved:

- You can download the Google SDK and use the `gcloud` command and language bindings locally on your own workstation. It is not required to install this software locally however.
- The cloud console has a really nice feature called "Cloud Shell". You can enable this using the terminal icon in the toolbar. It will open a terminal in a web browser window. Within this terminal `gcloud` is available as well as a Python interpreter with Python bindings for the various Google services.
- If you don't like to use a terminal in a web browser, another option is to create a VM instance. This can be achieved using the Compute Engine service. You can create a small `f1.micro` instance. Pre-baked OS images are provided. The images provided by Google already include the Google SDK and interaction using the Google Cloud APIs works out of the box on these VM instances (authentication is being arranged transparently). You can add your SSH key to the VM instance to get ssh login access.

  Note that by default access to Cloud APIs from the VM instance is constrained. You can configure to which APIs a VM instance has access in the cloud console.

Also take advantage of the documentation that is available. The documentation provided by Google is of good quality and contains several examples.

## 3 Submission Details

Teams may be formed that consist of at most two persons. Assignments must be submitted through BrightSpace. For each team a single submission is expected. Ensure that all files that are submitted include names and student IDs. In case there are problems with the team work, contact the lecturer by e-mail.

**Deadline:** Friday, June 24, 2022 ~~Tuesday, June 14, 2022~~

As with all other course work, keep assignment solutions to yourself. Do not post the code on public Git or code snippet repositories where it can be found by other students. All code and reports that are submitted may be subjected to automated plagiarism checks. Cases of plagiarism will be reported to the Board of Examiners.

The following needs to be submitted:

| | |
|---|---|
| **Source code** | Source code for the developed front-end and back-end worker processes. For GKE, provide source code including Dockerfiles. For Cloud Functions, you can download the implemented functions as ZIP files from the Google Cloud Console. |
| **Deployment procedure** | Provide a brief description on how your application should be deployed. If applicable, include any custom configuration files that are required (e.g. if you had to write custom Kubernetes configuration files). |
| **Report** | Report in PDF format. See the subsection "Report" above for the expected content. |

The maximum grade that can be obtained is 10. Because there is quite some freedom in the implementation of the assignment, it is hard to provide you with a fixed assessment rubric. Submissions will be assessed on the following criteria:

- Completeness and functionality of the implemented application.
- Soundness of the application architecture.
- Sophistication of the implemented application.
- Quality of the content and layout of the report.
- Quality of the performed experiment.

# 4 Some Background Information

## 4.1 Worker processes

Some tips on the implementation of the worker processes. Where we say worker process this can refer both to a process running in a container or a Cloud Function.

- You want to have separate work queues (Pub/Sub topics) for sort tasks, reduce tasks and "find palindrome" tasks.

- There must be separate workers for the different tasks. The separate workers can of course share parts of the implementation.

- The front-end should create the sort tasks and "find palindrome" tasks. Upload the file to Cloud Storage. Determine how many chunks the file consists of; this depends on the chunk size you picked. Then create a message (task) for each chunk.

- Figure out a way to download just a chunk of an object (partial object), instead of every worker having to download the entire object.

- You need to sort line by line. A line may cross a chunk boundary (see also earlier in the assignment text).

- The sort workers need to store intermediate results. This can simply be done in a subdirectory of your Cloud Storage bucket.

- Subsequently, the reducer reads all of the objects that store part of the intermediate results and applies merge sort. The final result is stored in Cloud Storage again.

## 4.2 Docker and Kubernetes

To deploy containers for the workers or the web front-end using Google Kubernetes Engine, you need to containerize the different components. To create containers you need to write Docker files: `https://docs.docker.com/get-started/part2/`. You can create the containers on your local machine (or within a VM instance on Google Cloud). Subsequently, you can upload the containers to the Container Registry on Google Cloud. From there, the containers can be deployed using GKE.

Google has a tutorial on creating a simple container, uploading and deploying it using GKE: `https://cloud.google.com/kubernetes-engine/docs/tutorials/hello-app`
Note that in this example already an autoscaler is used: “horizontal pod autoscaler”. This might be of use in your assignment.

Another thing you will need is the ability to access Google APIs from containers running on Kubernetes. This requires some special steps to be undertaken. You will need to set up a “Workload Identity”. You need to closely follow the steps outlined here:
`https://cloud.google.com/kubernetes-engine/docs/how-to/workload-identity`

One part is unclear however. After the `kubectl annotate serviceaccount` command, you need to grant permissions to the necessary/applicable Google API services to this service account. To do so, you first need to add this new service account to IAM in the Google Cloud Console. In the IAM module, you need to click “Add”. For “New members”, start typing the name of the service account and it will autocomplete. Subsequently, select Roles (permissions) to grant, such as “Cloud Datastore user”, “Pub/Sub Publisher”, “Pub/Sub Subscriber”, etc. Don’t forget to add `namespace` and `serviceAccount` properties to the Kubernetes deployment files. You can also refer to Google’s tutorial on authenticating to the Cloud Platform:
`https://cloud.google.com/kubernetes-engine/docs/tutorials/authenticating-to-cloud-platform`

Finally: watch your credits! For each Kubernetes node a VM is launched (E2 instance by default). If you create a 6-node Kubernetes cluster, this is going to incur a cost of a couple of dollars per 24 hours. Suggestion: first create a single-node cluster and expand this at a later stage. Expanding the cluster can also be achieved using the Cloud console.