UNIVERSIDADE FEDERAL DE PERNAMBUCO

BSC COMPUTER SCIENCE

CENTRO DE INFORMÁTICA

Henrique Vicente de Oliveira Pinto

PICEL – HTTP MIDDLEWARE FOR IMAGE PROCESSING

UNDERGRADUATE WORK

RECIFE

2017

Henrique Vicente de Oliveira Pinto

PICEL – HTTP MIDDLEWARE FOR IMAGE PROCESSING

UNDERGRADUATE WORK

A B.Sc. Dissertation presented to the Center for Informatics of Federal University of Pernambuco in partial fulfillment of the requirements for the degree of Bachelor in Computer Science.

Supervisor: Vinicius Cardoso Garcia (vcg@cin.ufpe.br)

RECIFE, 2017

Henrique Vicente de Oliveira Pinto

PICEL – HTTP MIDDLEWARE FOR IMAGE PROCESSING

UNDERGRADUATE WORK

Trabalho apresentado ao Programa de Graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Bacharelado em Ciência da Computação.

Recife, 13 de Dezembro de 2017.

BANCA EXAMINADORA

_____

Prof. Vinicius Cardoso Garcia
(Orientador)

_____

Prof. Kiev Gama
(Avaliador)

Acknowledgements

Gostaria de agradecer ao meu orientador, Vinicius Cardoso Garcia, aos meus colegas de trabalho da Liferay, Inc. (principalmente ao meu chefe, Eduardo Lundgren e ao resto do meu time do projeto WeDeploy), a minha namorada Bruna Carolina Baudel de Santana (pelo carinho e várias leituras e excelentes sugestões para a melhoria deste trabalho e apoio durante o curso), a meus colegas da faculdade (que muito me ajudaram a estudar em momentos de dificuldades e se divertiram em momentos de alegria), e a minha família. Gostaria ainda de agradecer ao Prof. Kiev Gama, pela sua participação na banca examinadora.

Resumo

Arquivos de mídia digital como fotos e vídeo compõe uma parte significativa do tráfego de dados da Internet. Uma mera imagem (seja uma foto ou frame de vídeo) utiliza uma quantidade de espaço muito maior do que dados textuais. Servir estes arquivos eficientemente pode se tornar complicado e caro rapidamente. Você pode armazenar toda a Wikipédia anglo-saxônica em uma dúzia de GB, mas isto é espaço suficiente apenas para alguns minutos de vídeo 4K ou algumas centenas de fotos de uma câmera dSLR de uma viagem que você fez a um lugar bonito.

Este trabalho de conclusão de curso de graduação estuda e propõe uma solução envolvendo a integração de *picel*, um *middleware* HTTP (Hypertext Transfer Protocol) de processamento de imagem que pode ser implantado em grandes sites da web para simplificar e automatizar o processo de publicação. Uma implementação de uma versão beta do *picel* é demonstrada na seção de resultados para validar o conceito conjuntamente com código de exemplo de como usá-lo programaticamente.

Palavras-chave: Internet; web; middleware; micro-service; image; picel

Abstract

Media files such as photos and video compose a significant portion of Internet data traffic. A single picture (either a photo or a video) frame takes way much more space than a textual data. Serving such files with efficiency can get complicated and expensive fast. You can store the full English Wikipedia articles on a dozen GB, but this is only enough space for just a few minutes of 4K video or a few hundred dSLR photos of a trip you take to a beautiful place.

This undergraduate work studies and proposes a solution involving the integration of *picel*, an HTTP (Hypertext Transfer Protocol) image processing middleware that can be deployed on large scale websites to simplify and automate the publishing process.

An implementation of a beta version of *picel* is demonstrated in the results section to validate the concept along with example code of how to use it programmatically.

Key-words: Internet; web; middleware; micro-service; image; picel

Figure List

Table List

Table of Contents

1. Introduction

This chapter is an introduction to this undergraduate work. It consists of a context, a scenario, the motivation for developing the project, the intended objectives, and this work structure. This works is about the distribution of images on the web and focuses on the networking, infrastructure, and high-level aspects of the matter. It does not comprise in-depth details about low-level image manipulation algorithms or concepts. Lack of more publicly available data about image processing costs made it infeasible to theorize about costs savings quantitatively, though there is preliminary evidence supporting the hypothesis that this is the case [37, 85].

1.1 Context

In the early days of the web, a desktop computer was the most common client for accessing images [50]. Narrower bandwidth connections, lower processing power, fewer Internet users, and less powerful browsers meant that web pages were simpler and contained less visual content.

The first web browser was the WorldWideWeb (next renamed to Nexus), written by Sir. Tim Berners-Lee, with an initial release in December 1990. However, the browser that popularized the World Wide Web was NCSA Mosaic, also the first one to display images inline with text, instead of in a separate window [45].

Times have changed, and the average web page of the top 1000 websites passed 1600KB for the first time in July 2014 [50]. By 2015, images already comprised 62% of the top 1000 web pages total weight, according to Akamai, a leading *content delivery network* and *clouds services* provider [49]. The advent of smartphones with the ubiquitous digital camera is partly responsible for this increase in image and video traffic [46-48]. Not only professional editing software such as Adobe Photoshop is now more practical, but it is far way more comfortable to snap a photo or film something and share online than it used to be previously.

1.2 Problem Statement

During work on an e-commerce car website project [66], a necessity for a photo thumbnail system was identified by the author of this project for its search engine and product pages (with cropping functionality) [67]. During research, it was discovered many web applications [5, 42, 44] does image processing internally using bindings to image libraries such as GD [68] or ImageMagick (or did it at some point). However, some large websites uses dedicated image thumbnail software such as *thumbor* [5, 8], an open-source photo thumbnail service by Globo.com.

A decision to use *thumbor* was made. A few years later, during work on WeDeploy (cloud computing platform the author works on) [69] it was necessary to evaluate the Go programming language [70, 71]. For that, the author decided to build an alternative to *thumbor* with the challenge of responding the following questions:

1. Is it possible to define a more friendly API to get images?
2. Is it possible to integrate with existing systems easily?
3. Is it possible to minimize storage needs and contract footprints by delegating caching responsibilities?
4. Is it possible to distribute such software efficiently?

1.3 Motivation

Working with images might result in considerable costs and workload to process and store. Large sites such as Flickr [5] and Globo.com have published articles, presented talks, and publicly released code related to this subject [37, 60].

There are also a few companies that offer a broad range of image processing products or services for the web, such as *JPEGmini server*, by JPEGmini. However, most of these alternatives seem to require extensive setup or planning that might discourage both casual use and system implementation.

This lack of a variety of plug-and-play offers might explain in part why while most web applications software use external databases, many, such as MediaWiki (the software that powers Wikipedia) [51], do image processing internally instead of delegating this to a dedicated software.

Even when taking care to scale efficiently by handling image processing requests separately, this type of workload is not usually desired on most application servers, for it can be suboptimal, introduce errors on other areas, and impact system performance and reliability in unpredictable ways.

1.4 Objective

This undergraduate work studies the problem of distributing images on the web and presents a new photo thumbnail service with the objective of offering a simple and efficient open-source image processing middleware capable of doing simple image transformations such as cropping and resizing, and file format optimization. Though there is some software on the market for this area, they require certain effort to use that might not be found on certain circumstances (for example, due to the "Not Invented Here syndrome") [102].

Besides reducing software development costs, for small and large websites other benefits include reduced use of resources such as CPUs, memory, and storage space and potentially better compression of images and quality, which translates as bandwidth savings and performance, what dramatically impacts user experience and Search Engine Optimization [52]. Scalability and the possibility of easy integration [67, 75] of the system on a Content-Delivery-Network [100-101], a geographically distributed network of servers, are also welcome to reduce latency issues.

A secondary benefit of the broad adoption of a tool similar to this in the industry is the reuse of software, avoiding the expenses related to creating the same feature over and over,

given that most software for the web contains images. However, usually with its proprietary code for handling them [76].

1.5 Work structure

This work is composed of 5 chapters, including this introduction. In chapter 2, there is a analyze of large website infrastructure and comparison of photo thumbnail services on the market. In chapter 3, you can find the details of the implementation of *picel*, a new photo thumbnail service. In chapter 4, you can see a discussion of the results. In chapter 5, you can find a conclusion of this work and some proposals for future work. The repository with source code of this project is available at https://www.github.com/henvic/picel.

2 Concepts and background

2.1 Large-scale websites infrastructure

Large websites with high-traffic such as Google, Facebook, Wikipedia, and Netflix are known to have many internal services organized in layers, and only a few directly exposed to the public web [42]. This kind of pattern is known as multi-tier architecture (or n-tier) [39-42].

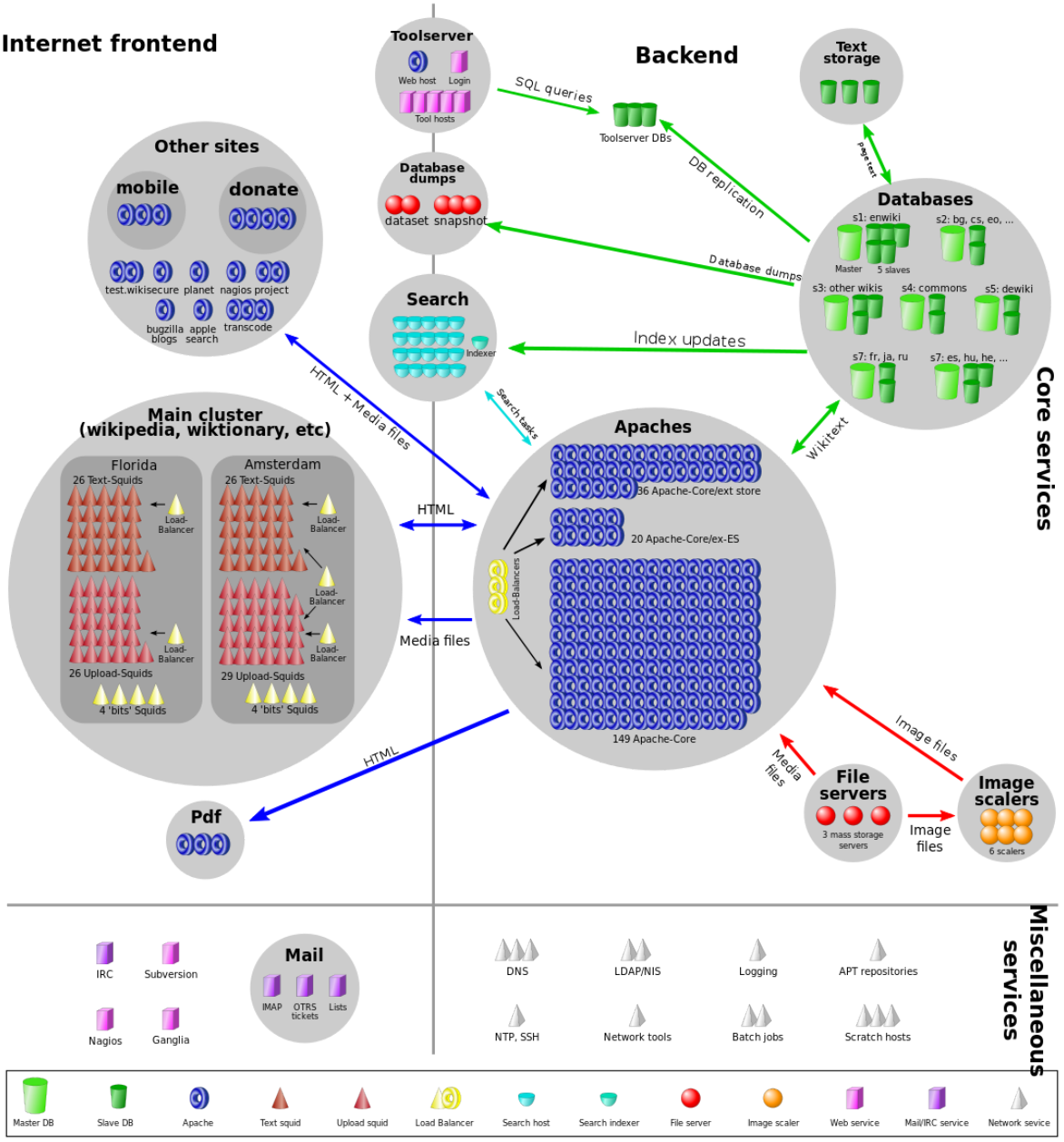

Figure 2.1. Wikimedia multi-tier topology (2010) [53]

Such tiers might be on a single monolithic application or, more commonly in large systems, on different application and servers, following the concept of micro-services architecture [54] [98].

As an example, Figure 2.1 shows the multi-tier topology of Wikimedia, the entity behind the Wikipedia, Wiktionary, and other smaller projects, and maintains clusters of servers in different locations worldwide. While MediaWiki (the software that powers it) is mostly a monolithic [53] it use some additional parts, such as ElasticSearch [73], to provide a higher experience for its users. Besides using different specialized software, it routes some of the requests to specialized servers (like for the image scalers, which shares the same code base of the main cluster) [74].

It might be the case even a traditional monolithic system benefits from using a shared-nothing architecture photo thumbnail middleware [67], what simplifies much of conventional image handling operations (what is usually not a first-class citizen on most application services). This reality has some parallels with database servers: most software that needs to store data on the web relies on external databases such as MySQL, PostgreSQL, etc. for a production environment, instead of embedded ones such as SQLite or even Lucene [42, 44].

2.2 Background

Image processing for the web involves, in many cases, resizing down a large image available on a storage server [8, 51]. Most of the processing work involves the high intensive use of processing power and memory to generate a new file, that is commonly saved to an ephemeral storage device for posterity [51]. Image operations involve algorithms that read and processes the images as large matrices of points with a variety of strategies for compression and image transformation [89]. Such complexities are reduced by the use of image libraries, as described in section 2.3.

2.3 Imaging libraries

Implementing image processing for complex image files such as JPEG and its peculiarities is hard. Therefore it is interesting to consider the use of open-source robust third-party libraries [17].

Three approaches for communication with such software are:

a. Calling external programs (e.g., file descriptor, regular files on a filesystem)
b. Calling embedded dependencies
c. Calling linked external dependencies

There are different issues with each of these different approaches. Calling external programs means that the portability of the software is affected because now the middleware requires more than just the operating system to work correctly and version compatibility issues such as "dependency hell" might arise [99]. Calling embedded dependencies removes this risk. However, compilation, portability, and distribution might still be affected due to availability

options of bindings for the language of choice for writing the middleware, and there is a risk of the system getting more fragile and less extensible/adaptable. Calling external dependencies has an even higher chance of compatibility issues [16].

ImageMagick

ImageMagick is a popular library for manipulating images created inside DuPont for solving the image manipulation needs of chemists and biologists, released in 1st August 1990 [18]. It is used for all image manipulations operations on all file formats, except to/from WebP. The photo-sharing website Flickr used to use it before moving to a private GPU-based solution [5].

WebP

WebP is a relatively new image format (initially released in 30 September 2010 and still not broadly supported like JPEG) developed by Google that has significant performance and containerization advantages over JPEG, GIF, and PNG [19]. *picel* uses its official implementation CLI tool to read and write .webp files. This library is only capable of converting files to the WebP file format (.webp) and produces better image quality results than the current implementation of ImageMagick as of the date of this work.

GD Graphics Library

GD is another popular graphics software library, initially released in the year 1994, and one of the most popular formats along ImageMagick [68].

High Efficiency Image File Format (HEIF)

HEIF is a new file format for individual images and image sequences, developed by the Moving Picture Experts Group (MPEG). It has compelling advantages over WebP, JPEG, PNG, and many others image formats designed for rasterized images [30, 64]. Including, less memory footprint and faster image resizing and cropping/zooming thanks to parallelism and how it subdivides the images internally. All this also allowed a new deblocking algorithm that can improve the quality of highly compressed images. During Apple's 2017 WWDC event it was announced they would embrace the new HEIFF by the MPEG working group [63] natively on newer iOS and macOS operating systems (already on the market) so this format might get popular on the web as well soon given Apple's extensive user base and incentive. It should be the standard option for web clients that announces its support capability by sending a HTTP *Accept* request header [30].

2.4 Photo thumbnail services on the market

thumbor

thumbor [8] is an open-source photo thumbnail service in Python by the Brazilian media and Internet company globo.com, which serves around 15 billion images with it monthly

(2017) [55], thumbor currently support different sources of images. By default, it uses a *File loader* approach, which requires the user to upload the image to the service [60].

It has a rich set of image manipulation features such as adding watermarks, inverting colors, and using face and feature detection algorithm for smart cropping. It uses an internal cache layer and has a built-in URL Tampering protection option to overcome the issue of malicious overload. It supports a plethora of file formats.

JPEGmini Server

JPEGmini is a paid proprietary software product for compressing the file size of JPEG photos efficiently. It has a server version [9] that JPEGmini advertises as a regular image processing server, but is more akin of a CLI tool which can be invoked programmatically using system calls to optimize JPEG files (only) in batch [61]. It is limited to file optimizations and doesn't have features such as image resizing or cropping. The output it generates is saved to a file.

Imgix

Imgix [14] is a paid real-time image processing and Content-Delivery-Network service used by large websites such as Kickstarter, reddit, The Guardian, Eventbrite, etc. It offers a great variety of options and image details extraction methods on its API, besides having a sandbox where you can try the operations available. It supports multiple sources and has caching features built-in. It supports a plethora of formats and is the most complete regarding documentation and capabilities by far.

2.5 Chapter summary

In this chapter, we saw that running large websites involves running a considerable amount of software together and that there is a variety of different vendors of imaging software for the web on the market, most using the same set of very robust image processing libraries. In the next chapter, we are going to see how the implementation for *picel* was crafted.

3 Implementation

picel is a photo thumbnail service middleware able to do common image operations such as cropping, resizing, and file format conversion, exposing this service through the HTTP protocol. It is designed to be deployed as an internal service that might be publicly exposed protected by a load balancer with a cache layer. It should be able to get and process JPEG, PNG, GIF, and WEBP files, and efficiently deliver them, acting similarly to a traditional Unix filter program [10]. RAW files are not accepted as input because of their "digital negative" scope [25], which means that they aren't ready for final distribution.

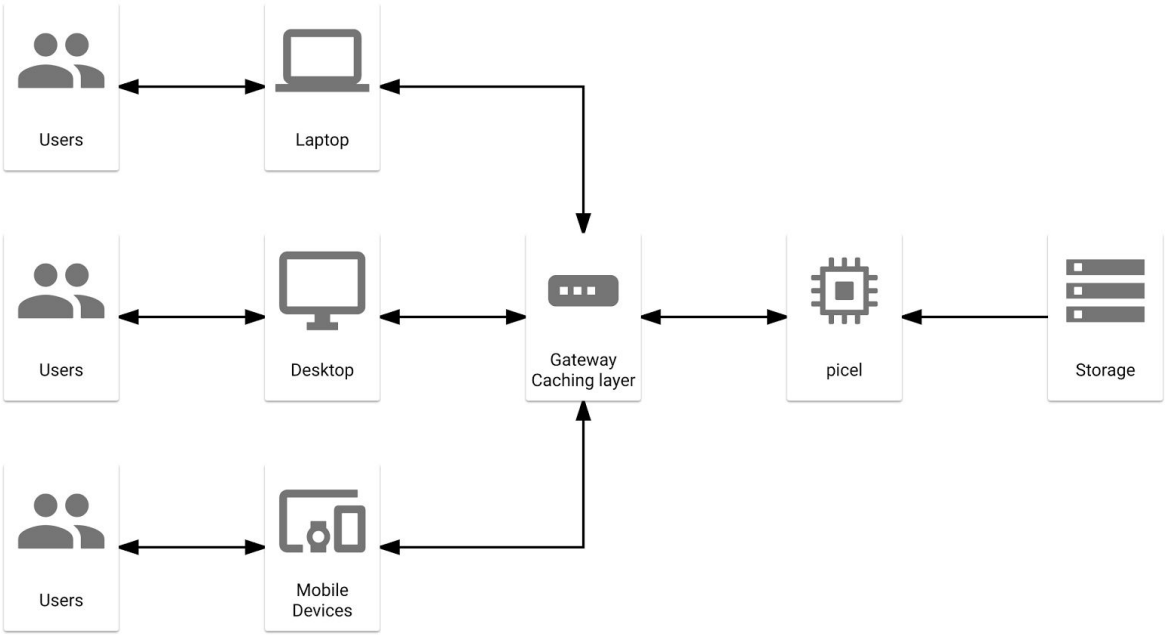

Figure 3.1. picel handling requests

picel handles incoming requests by requesting the image from a backend, processing it, and delivering the processed as the response to the user. However, besides serving files already in storage to final users, *picel* can validate a file as an image file to be stored and served. One approach for this is to save such file on a private backend server that can be accessed with *picel* and let it process the image. A decision to store the original file or not might be made. However, it is interesting always to serve users a modified version of the file to cope with the dangers of unrestricted file upload [62] and to do major image optimizations.

Following the UNIX philosophy [16], this middleware is focused on doing one thing only, and doing it well: serving as a special type of proxy/filter program that read images, transforms it, and sends the result as the response to the requester. The nginx server is a good fit for attending the recommendations of HTTP/2 and caching at the load balancer level explained in the subsections below [21].

3.1 Storage service

Similar software usually works with a variety of protocols such as NFS, FTP, HTTP, samba, etc. on the backend. *picel*, on the other hand, only consumes requests from an HTTP server. This decision was made for simplicity, given that HTTP is one of the natural and most common application protocols used on the web for serving files and has a low overhead [38]. Connections to the backend storage are ephemeral and happen at the time *picel* handles a request. An HTTP server must be set up to serve files available only on the filesystem.

3.2 Image processing

One of the advantages of the Go language is that it produces statistically-linked binaries by default [24]. That said, the option of invoking slave processes of external programs for doing the image processing was decided for the reasons explained in section 2.3. It is important to note that the cost of image processing itself usually far outweighs the cost of swapping processes [17].

One significant challenge is always to be sure to use a compatible version of an imaging library. This need can be met in two ways. Automatically, if using the published container for the project, which already contains all dependencies built-in (see section 3.10), or by running the available unit and integration tests. Nuances about how an image is processed, such as quality parameters, are mostly left the default, using the recommendations of the imaging libraries used. An important goal to meet is that it should be good enough to be used on a photography website.

3.3 API

picel has two modes: *single-site* and *multi-site* (default). On *single-site* mode, requests received by the middleware are always made to a given storage server. In the *multi-site* mode, an address must be passed encoded on the URL with the image file origin server.

A micro-format specification is defined for encoding/decoding picel URLs. For encoding, a new URL is composed by combining the original URL and the desired processing properties. For decoding, a parser extracts the parameters from a URL.

The format for the path part of a URL request to picel is: "[optional "s:"][optional host]*md*", where:

- "s:" is an optional modifier telling *picel* to use HTTPS when contacting a storage server
- Optional host is a host address for a storage server
- The *md (modified part)* is the original path without the last element appended with a, possibly, modified le (*last element*)

The "s:" modifier and optional host parts are only necessary when using *picel* with the multi-site mode (default).

The *le* part is a product of:

- Original path
- Resizing and cropping parameters might be added (crop factor:resize factor)
- File format: if ".extension" is used, output is coerced to the same input format, otherwise "_extension" should be used (e.g., ".jpg" coerces, "_jpg" does not)

The standard input format is JPEG. If a request is made to an endpoint without an extension, it automatically appends ".jpg". For performance, *picel* currently uses the WebP image format as the output format if the client sends an Accept request header unless an explicit output is requested on this line (for example, if it ends with ".jpg"). The separator "_" is used on the md part for adding parameters. If the original path contains a "_" it must be escaped by a leading "_". Example: "file__blackandwhite.jpg".

The query string parameter *?query* can be appended to a URL to check how *picel* resolves a given path. *picel* does not work with URLs that require query strings parameters.

Table 3.1 Examples of encoding/decoding.

picel URL	Original file address	Parameters
http://picel-server/animals/dog.jpg	http://<any>/animals/dog.jpg	Single mode: yes Output format: JPEG (coerced, in)
http://picel-server/animals/dog_800x.webp	http://<any>/animals/dog.webp	Single mode: yes Output format: webp (coerced, in) Width: 800px, Height: 56px
http://picel-server/127.0.0.1/animals/dog_jpg.gif	http://127.0.0.1/animals/dog.jpg	Single mode: no Output format: GIF (coerced, out)
http://picel-server/remote.local/animals/dog.webp	http://remote.local/animals/dog.webp	Single mode: no Output format: webp (coerced, in)
http://picel-server/s:example.com/animals/dog_800x.webp	https://example.com/animals/dog.webp	Single mode: yes Output format: webp (coerced, in) Width: 800px, Height: 56px
http://picel-server/s:example.com/animals/dog_800x_webp	https://example.com/animals/dog.webp	Single mode: yes Output format: better supported (with fallback to webp) Width: 800px, Height: 56px
http://picel-server/big__sur_500x.jpg	http://<any>/big_sur.jpg	Single mode: yes Output format: JPEG (coerced, in) Width: 500px, Height: 56px
http://picel-server/big__sur_0x0:600x300_112x56	http://<any>/big_sur.jpg	Single mode: yes Output format: better supported (with fallback to JPEG) Crop box points: (x, y, width, height) = (0, 0, 600, 300) Width: 112px, Height: 56px
http://picel-server/big__sur_0x0:600x600_100x	http://<any>/big_sur.jpg	Single mode: yes Output format: better supported (with fallback to JPEG) Crop box points: (x, y, width, height) = (0, 0, 600, 600) Width: 600px, Height: auto
http://picel-server/big__sur_50x40:600x600_x300	http://<any>/big_sur.jpg	Single mode: yes Output format: better supported (with fallback to JPEG) Crop box points: (x, y, width, height) = (50, 40, 600, 600) Width: auto, Height: 600px
http://picel-server/people_500x_gif	http://<any>/people.gif	Single mode: yes Output format: better supported

		(with fallback to GIF) Width: 500px, Height: auto
http://picel-server/people_500xx_	-	URI parsing error

3.4 Load balancer

In a networking perspective, load balancing improves the distribution of workloads across multiple servers by using any number of different metrics or heuristics, provides resilience/fail-tolerance by implementing redundancy and monitoring health-checks. Rate limiting to avoid abuse might be performed on the load balancer to prevent DoS attacks (on purpose or accidental overload due to faulty connections) returning 429 Too many Requests responses [15].

For better performance, support for the HTTP/2 binary protocol is recommended, like shows, the *HTTP2 Explained* study of Daniel Stenberg, author of the cURL library and Command-Line tool for transferring data using various protocols [20] for improved reliability, and lower latency. Using TLS (Transport Layer Security) to enable HTTPS at this level also removes overheads of the TLS handshake protocol negotiation and TLS record protocol from the application servers.

3.5 Caching

This implementation has no public caching mechanism. Instead, it is highly recommended that the load balancer has a caching layer, which should cacheable public requests (according to *Cache* policy response headers). Not doing so would jeopardize the performance of *picel* on a real-scenario use case significantly. Prefetching can also be used on the application level to warm the cache before use and is recommended as an upload hook. The duration span of a cached asset is better determined by the needs of the system and more study should be made regarding it [37], though in many cases, *forever* while the original asset exists is a good answer (especially when prefetching predefined parameters).

3.6 Handling requests

The following is the process for processing an image with picel:

1. User requests an image to the *picel* endpoint (directly or through a load balancer)
2. *picel* receives the request and unmarshal it
3. If the request is invalid, return error
4. Download image file from storage server
5. Spawn a *webp* or *imagick* process with the received image output parameters
6. Send a response to the client

The conversion parameters (including quality settings, preferences, etc.) are passed to the CLI imaging manipulation tools, along with the temporary file path for the original file, as program arguments. Image metadata (such as XMP and Exif) is always stripped, mostly due

to security reasons [65] (for example, avoiding exposing GPS coordinates, photographer personal identification, etc.), but also for generating smaller files.

The response sent to the client should be cached by the caching layer on the load balancer to avoid a second request from triggering the process from the beginning once again. To avoid concurrency issues making this happen during a window where the content is not cached yet, prefetching of a resource is to be used. Also, if concurrent requests occur, it might be identified, and the request to the original object and response shared for all requests.

3.7 Debugging and profiling

Profiling a Go program is relatively easy, safe to do on a production environment without hitting the performance of an entire system, and can be done remotely [22] through an HTTP web server on the application.

picel is already an HTTP server. Therefore it is only necessary to:

1. include the pprof profiler package (that works as *side-effects*)
2. protect the /debug/* routes to avoid exposing the profiling routes to the public web
3. register a few pprof HTTP endpoints [23] (this protection can be done on the load balancer level).

After these changes, profiling can be made with the built-in go tool pprof.

3.8 Software quality

This software needs to have a high Quality of Service if run as a critical infrastructure for large websites [55] or part of web systems where a high risk of failure is not tolerated [26].

3.8.1 Static code analysis

Go has many static code analysis tools. Doing static code analysis is an important step to decrease technical debt, improve source code quality, and reduce software defects [27]. The official *golint* tool is used for linting the source code against programmer errors. The official *go vet* tool is used to examine and report suspicious constructs. The *errcheck* tool is used for checking unchecked error returns. All code must completely adhere to these strict constraints (e.g., no code should be committed with one unchecked error or suspicious construct). Even false positives must be “fixed” for the sake of avoiding over-complication of quality control procedures.

Change Risk Anti-Patterns (CRAP) score, and cyclomatic complexity is also analyzed with the Codacy service [28]. The threshold for the cyclomatic complexity (that measures code complexity) is set to 10 and is analyzed with the Go tool *gocyclo*. The CRAP score for all code must be below 30 (the default value recommended by Codacy). The CRAP score is a correlation between the code test coverage and the cyclomatic complexity [86].

3.8.2 Unit and integration testing

A Continuous Integration (CI) system is used to run the unit and integration tests of *picel* [87]*, generating a test coverage report in the process* [88]. The unit tests are there for verifying if the encoder/decoder mechanism is working correctly and if the request handler is handling the connections correctly. For the integration step, the tests call the ImageMagick tool *identify* to compare images and determine if the generated image appears to be created successfully. This test usually generates the same binary files on a given machine. However, it is not deterministic if you compare results of different computers as it is related to what dependency version is running on it and other smaller factors.

Test assets (mostly image files; including for benchmark) are available on a secondary branch called *test_assets* [93] for the sake of not polluting the application/main branches with binary blobs on the versioning system, git [92]. All things on this branch should be inside a test_assets directory. Files generated by *picel* can be found inside the test_assets/compare directory.

3.8.3 Benchmark/Stress testing

ab is an Apache Benchmarking tool can be used for benchmarking *picel* to compare it with alternatives and to see how it handles multiple different files at once [29]. An implementation of it is still underway. For tests to be relevant, it is recommended that they are made on large machines similar in power to ones that would be used for serving images on a large scale website. It should be interesting to compare a distinct set of requests with different workload styles (i.e., image size, file sizes, etc.) multiple caching configurations, etc.

3.9 Avoiding premature optimization

There are at least two topics that were considered for addition on *picel*, but postponed indefinitely until enough data exists to support a final decision [12].

- Image recognition is a hot topic with services like Amazon Rekognition, Google Cloud Vision API [2], or Microsoft Azure's Computer Vision [3] offering image analysis solutions that work out of the box and extracts face detection, objects description, location info, and other data
- The use of a strategy for doing multiple resizing/cropping on the same image file

3.10 Distribution

There are two official supported forms of installing *picel*. The repository with source code and builds of this project is available at https://www.github.com/henvic/picel [6], and Docker images are available as "henvic/picel" in the Docker Registry. A JavaScript encoder library is available at https://www.github.com/henvic/picel-js [7].

Using the binary directly from the repository requires configuring your system with the ImageMagick and WebP dependencies, as explained at section 2.3. These dependencies should be installed with your operating system package installer.

3.11 License

This software is licensed under public domain (where applicable) or the permissive MIT License, as most Go software. This choice of licensing is to allow the indiscriminate use of the software [57], what is very welcome within the Go community, given the fact that Go is a statically-linked language and Go dependencies are necessarily pure source code (text). Therefore, the community is very wary of the risk of license proliferation of non-permissive licenses, what is known for taking away the focus on programming [58-59].

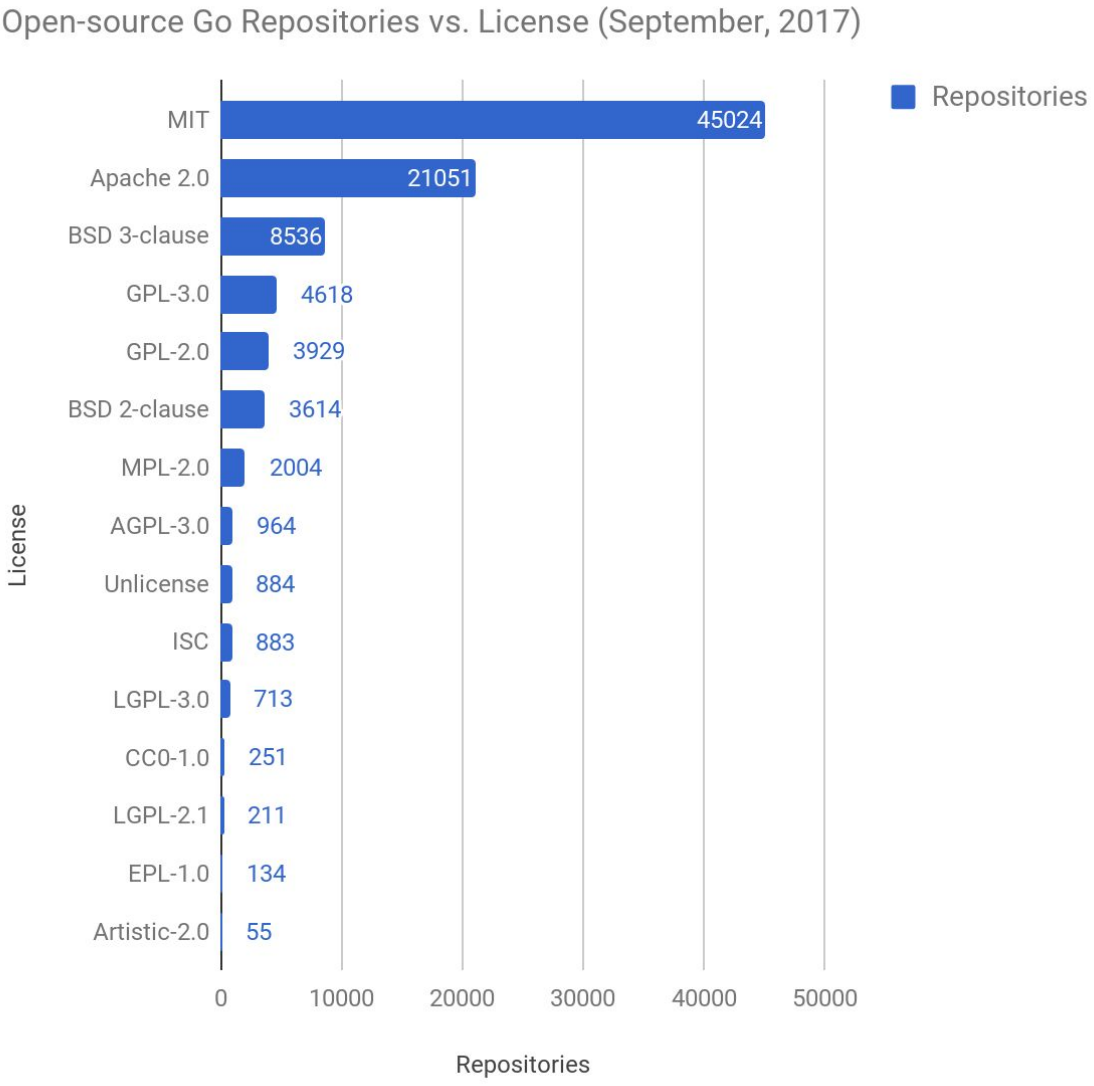

Figure 3.2 License data extracted from a query on Google's BigQuery [56]

Also interesting to know is that the market for cloud computing [11] components (both software and hardware) is very competitive and open-source solutions have a competitive advantage [77-80]. Not surprising Hashicorp [81] and Docker [82] are recent market successes [83, 84].

3.12 Chapter summary

In this chapter the implementation details of *picel* were presented, along with an explanation about its design principles, a description of the processes adopted to ensure that it achieves a minimum quality requirement, how to install it using Docker, and its permissive licensing choice. In the next chapter, we are going to do acceptance tests with it.

4. Results and Acceptance Testing

Even though *picel* URLs are easy to compose by hand, this is not a process you want to do manually. Therefore, an accompanying encoder library picel-js is available [33] to simplify the process of encoding requests on the browser. Although it is easier to start using picel on the beginning of a project, it is still easier to integrate with existing projects than the competing solutions presented here because it uses existing storage servers as the origin source for images. On this chapter, we are going to see how to run *picel* and an acceptance testing demonstration.

To run *picel* you need to either use its Docker image or get its binary from its repository page. Once you have the binary, you can execute it to list its available commands as shown below.

```
$ picel

Usage:
  picel [flags]
  picel [command]

Available Commands:
  serve            Serve images
  dependencies  Verify dependencies
  help             Help about any command

Flags:
  -c, --config string  .ini configuration file
  -h, --help           Show help message
  -v, --verbose        Verbose mode (show image processing output)
      --version        Print version information and quit

Use "picel [command] --help" for more information about a command.
```

The "picel dependencies" command checks the availability of all required dependencies. The "picel serve" command starts the picel server. It accepts the "--addr" flag, to configure on what port/net addresses *picel* should be run. The default port is 8123.

Running the command "picel serve --addr :8000", *picel* will try to listen on the port 8000 for all network interfaces available on the machine.

By default, picel runs the non-profiled *picel* binary for avoiding any safety issues, as explained in section 3.7. *picel* requires external dependencies, and this might make installation more complicated. However, it is also available on a Docker [82] container image on the public Docker Registry, containing all required dependencies necessary for the immediate use of the middleware:

You can install it using:
$ docker pull henvic/picel

Then, run it exposing it on its default port 8123:
$ docker run --publish 8123:8123 henvic/picel

You can use the --detach flag to run it as a daemon, instead:
$ docker run --detach --publish 8123:8123 henvic/picel

You can also run the *picel* with the profiler:
$ docker run --detach --publish 8123:8123 --env PICEL_PROFILER=true henvic/picel

Please be aware that ports below 1024 are privileged ports and if you try to run on it exposing *picel* on such low port on a UNIX-like system, you will need to have superuser/root privileges [90]. After picel is up and running, you can start testing it right away.

4.1 Example of image transformation

Important: Images presented here are not in scale and quality is degraded by the print.

The source for this cartoon is https://imgs.xkcd.com/comics/tar_2x.png

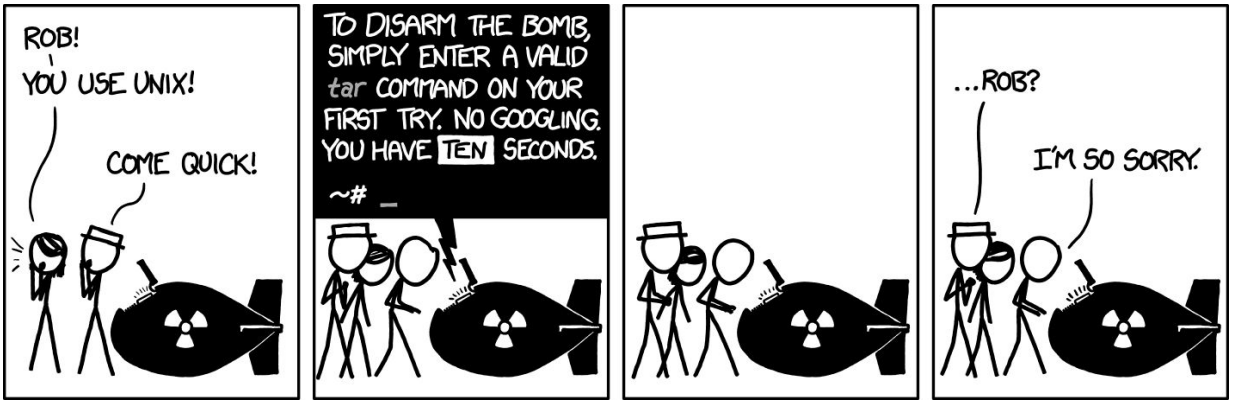

Figure 4.1 Original xkcd: tar #1168

JavaScript code for extracting the first panel:

```
picel.encode({
    prefix: "http://localhost:8123",
    backend: "https://imgs.xkcd.com",
    path: "comics/tar_2x.png",
    crop: {x: 0, y: 0, width: 346, height: 458},
    width: 300
});
```

This code would create the following URI:
http://localhost:8123/s:imgs.xkcd.com/comics/tar__2x_0x0:346x458_300x_png

Which is correctly parsed by the server as we can see if we visit
http://localhost:8123/s:imgs.xkcd.com/comics/tar__2x_0x0:346x458_300x_png?explain


```
{
   "message": "Success. Image path parsed and decoded correctly",
   "transform": {
      "image": {
         "id": "comics/tar_2x",
         "extension": "png",
         "source": "https://imgs.xkcd.com/comics/tar_2x.png"
      },
      "path": "comics/tar__2x_0x0:346x458_300x_png",
      "original": false,
      "width": 300,
      "height": 0,
      "crop": {
         "x": 0,
         "y": 0,
         "width": 346,
         "height": 458
      },
      "output": "jpg"
   },
   "errors": null
}
```


This corresponds to the following processed image:

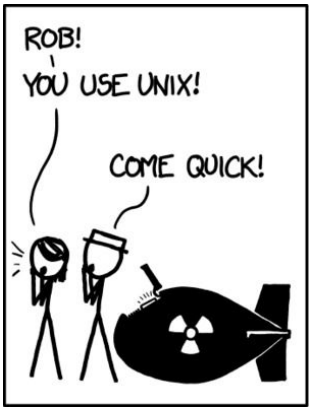

Figure 4.2 cropped xkcd: tar #1168 (13KB)

Please notice that ".png" was replaced as "_png", for the performance optimization objectives explained in section 3.3. Also, be aware that if you are using Docker and want to test local files, you must use a non-localhost/127.0.0.1 loopback address on the part of your URI, or else it is going to use the internal container loopback device, instead of your host machine.

4.2 Quality degradation and weight optimization

This test only comprehends a single image. More tests would be welcome, but as both ImageMagick and WebP are already very well-tested, this is not the focus of this work, and picel is using no image preferences arguments besides the quality parameter.

Image: "Hraunfossar & Barnafoss #2 Borgarfjörður, Iceland."
https://www.flickr.com/photos/henriquev/24016682888/

Source	Size (Weight)	Link
Original file	5472x3078 (15.5MB)	https://farm5.staticflickr.com/4453/24016682888_dfff44d47c_o.jpg
WebP generated by picel (using WebP)	5472x3078 (6.4MB)	https://gist.github.com/henvic/458ec4e4cbc60c3b045799d1cbc4abfa/raw/502d3a181a15c09f801d8f4a513a7d70960ffbf8/24016682888__dfff44d47c__o_jpg.webp
Resized by Flickr (with added sharpening)	800x450 (295KB)	https://farm5.staticflickr.com/4453/24016682888_68ce7cc7cb_c.jpg
JPEG generated by picel (using ImageMagick)	800x450 (251KB)	https://gist.githubusercontent.com/henvic/458ec4e4cbc60c3b045799d1cbc4abfa/raw/502d3a181a15c09f801d8f4a513a7d70960ffbf8/24016682888__dfff44d47c__o_800x.jpg
WebP generated by picel (using WebP)	800x450 (198KB)	https://gist.github.com/henvic/458ec4e4cbc60c3b045799d1cbc4abfa/raw/502d3a181a15c09f801d8f4a513a7d70960ffbf8/24016682888__dfff44d47c__o_800x_jpg.webp

Table 4.1 Comparing files of generated thumbnails

If you compare output, you may notice Flickr is using heavy sharpening on its thumbnails [96], while no sharpening treatment is being given on *picel*.

The main average error (MAE) for channel distortion after the conversion from the original file to WebP is almost none as we can verify with the ImageMagick compare tool [97]:

```
$ compare  -verbose -metric mae ../24016682888_dfff44d47c_o.jpg
24016682888__dfff44d47c__o_jpg.webp diff.png
            Channel distortion: MAE
              red: 552.222 (0.00842636)
              green: 401.081 (0.00612011)
              blue: 535.04 (0.00816419)
              all: 496.114 (0.00757022)
```

4.3 Stress tests

This test is a preliminary stress test and only serve to show that this server can achieve a high throughput, without comparing it to alternatives. A more serious analysis would require a dedicated machine not as readily available. For this stress test, the image used on 4.2 was used as well.

These tests were run on a macOS 10.13.1 running Docker 17.09.0-ce on a Late-2013 MacBook Pro Retina 15-inch 2.3 GHz Intel Core i7 with 16GB 1600 MHz DDR3. No graphics cards were used for the test [94]. The Docker preferences were set to use up to 8 CPU cores and 10GB of memory.

A copy of the Flickr original image was made to a local server and ab was used to make 200 requests with a concurrency level of 4 per time. It was found this machine is capable of handling about 2 requests per second for a 5472x3078 15MB JPEG file, resizing it to 800x450 251KB. Or 2.7 requests per second when resizing it to WEBP, with a final file of 198KB [95].

4.4 Chapter summary

Although there still need more validation for *picel* to be able to be labeled as a stable version that can be safely used in a production environment, the results appear to correspond with what is expected from a robust and stable middleware. Also, it seems to meet the expected requirements set during the planning and conceptual phases of this project. In the next chapter, we conclude this work.

5. Conclusion

The core expectations of the project were met mostly satisfyingly. Now, more tests need to be done to validate whether this solution is ready for prime time, and one possibility is to use strategies of feature toggles, A/B testing, and dark launches [34-35] on production environment until enough data is available for a proper definition.

Interestingly enough, building and deploying a middleware for processing image files at scale involves the area of networking expertise much more than of the actual processing of images, if a decision to delegate the processing to robust specialized software by major vendors is made.

This software would involve many other parts had it been decided that it would perform the image processing internally using available image manipulation libraries directly (or implementing one). While this would avoid the overhead of spawning new processes, it would add a lot of additional code and chances of defects by removing the isolation granted by this current state of separation of concerns.

This is why care was taken to whitelist what input files to accept before spawning an ImageMagick process to handle it. ImageMagick has some capacity to re-encoding video files, but that would add room for unexpected behaviors, and this is why most services do one or the other, not both. For example, while thumbor has support for photos. Amazon has the Elastic Transcoder [36] specialized for video, but not photos. If support for video is wanted, it is probably more worth to create it separately given that the requirements (latency, image size, request duration, resource utilization, caching worthiness, etc.) differ significantly, even though the processing material is similar.

5.1 Limitation

As expressed in section 3.9, this software has limitations regarding the lack of optimization for doing multiple operations on a copy of a single file and entirely lacks any Artificial Intelligence deciders or detectors.

5.2 Future work

Here is a not exhaustive list of possible future work for improving *picel* quality, support, and usefulness.

- Create guidelines on the documentation and boilerplate code to advance the use of it
- Create a comprehensive suite of benchmark/stress tests
- Use SHA1 hashes to store a cache of loaded files on the same place to minimize disk and networking I/O
- Improve the current accompanying encoding JS library by adding the identification of Internet connection quality/type and user device/screen size

- Publish more accompanying encoding/decoding libraries on other languages for server-side and non-web usage
- Improve logging functionality
- Consider sharpening resized down images
- Add support for rasterizing SVG (Scalable Vector Graphics)
- Replace the ImageMagick dependency with the more mature fork GraphicsMagick, that thumbor uses [8, 31-32]
- Add High Efficiency Image File Format (HEIFF) support and prioritize it when possible, for the reasons explained on Section 2.3.

References

[1] **Wikipedia:Size of Wikipedia.** Wikipedia, the free encyclopedia, 2017. Available in https://en.wikipedia.org/w/index.php?title=Wikipedia:Size_of_Wikipedia&oldicd=798288211. Last visited on 3rd December, 2017.

[2] **Google Cloud Vision API.** Google, 2016. Available in https://cloud.google.com/vision/. Last visited on 3rd December, 2017.

[3] **Microsoft Azure Vision API.** Microsoft, 2017. Available in https://azure.microsoft.com/en-us/services/cognitive-services/computer-vision/. Last visited on 3rd December, 2017.

[4] **Video file size calculator.** Toolstud.io, 2006. Available in https://toolstud.io/video/filesize.php. Last visited on 3rd December, 2017.

[5] Russell, A. **Real-time Resizing of Flickr Images Using GPUs.** Flickr, 2015. Available in http://code.flickr.net/2015/06/25/real-time-resizing-of-flickr-images-using-gpus/. Last visited on 3rd December, 2017.

[6] Pinto, H. ***picel* middleware repository.** GitHub, Inc., 2016. Available in https://github.com/henvic/picel. Last visited on 3rd December, 2017.

[7] Pinto, H. ***picel* encoder tool (JavaScript).** GitHub, Inc., 2016. Available in https://github.com/henvic/picel-js. Last visited on 3rd December, 2017.

[8] **Thumbor.** Globo.com, 2015. Available in https://github.com/thumbor/thumbor. Last visited on 3rd December, 2017.

[9] **JPEGmini Server**. Beamr Ltd., 2017. Available in http://www.jpegmini.com/server. Last visited on 3rd December, 2017.

[10] Raymond, E. S. **The art of Unix programming. Languages. Online version.** catb.org, 2003. Available in http://www.catb.org/esr/writings/taoup/html/ch14s04.html#perl. Last visited on 3rd December, 2017.

[11] Mell, P.; Grance, T. **The NIST definition of cloud computing**. NIST U.S. Department of Commerce, special publication, 2011.

[12] Knuth, D. E. **Structured Programming with go to Statements.** ACM Computing Surveys (CSUR), 6(4), pp. 261-301, 1974.

[13] Kreitz, G.; Niemela, F. **Spotify--large scale, low latency, P2P music-on-demand streaming In Peer-to-Peer Computing (P2P)**. IEEE Tenth International Conference, pp. 1-10, 2010.

[14] **Imgix.** Zebrafish Labs Inc., 2017. Available in https://www.imgix.com/. Last visited on 3rd December, 2017.

[15] **RFC 6585: Additional HTTP Status Codes: 429 Too Many Requests.** IEFT, 2012. Available in https://tools.ietf.org/html/rfc6585#section-4. Last visited on 3rd December, 2017.

[16] Raymond, E. S. **The art of Unix programming. Philosophy. Online version.** catb.org, 2003. Available in http://www.catb.org/esr/writings/taoup/html/philosophychapter.html. Last visited on 3rd December, 2017.

[17] Raymond, E. S. **The art of Unix programming. Taxonomy of Unix IPC Methods. Online version.** catb.org, 2003. Available in

http://www.catb.org/esr/writings/taoup/html/ch07s02.html#id2922002. Last visited on 3rd December, 2017.
[18] Cristy, J. **ImageMagick history.** ImageMagick Studio LLC, 2017. Available in http://www.imagemagick.org/script/history.php. Last visited on 3rd December, 2017.
[19] **WebP - A new image format for the Web.** Google Developers, 2016. Available in https://developers.google.com/speed/webp/. Last visited on 3rd December, 2017.
[20] Stenberg, D. **HTTP2 explained.** 2015. Available in https://daniel.haxx.se/http2/http2-v1.10.pdf. Last visited on 3rd December, 2017.
[21] **Nginx content caching.** NGINX Inc., 2017. Available in https://www.nginx.com/resources/admin-guide/content-caching/. Last visited on 3rd December, 2017.
[22] Cox, R.; Ma, S. **Profiling Go programs.** The Go Programming Language - The Go Blog, 2011. Available in https://blog.golang.org/profiling-go-programs. Last visited on 3rd December, 2017.
[23] Krylysov, A. **Profiling and optimizing Go web applications.** 2017. Available in artem.krylysov.com/blog/2017/03/13/profiling-and-optimizing-go-web-applications/. Last visited on 3rd December, 2017.
[24] **Why is my trivial program such a large binary?** The Go Programming Language - FAQ, 2017. Available in https://golang.org/doc/faq#Why_is_my_trivial_program_such_a_large_binary. Last visited on 3rd December, 2017.
[25] **Understanding Digital Raw Capture.** Adobe Systems Inc., 2004. Available in http://www.adobe.com/digitalimag/pdfs/understanding_digitalrawcapture.pdf. Last visited on 3rd December, 2017.
[26] Perrow, C. **Normal Accidents: Living with High-Risk Technologies**. Basic Books, 1984.
[27] Louridas, P. **Static code analysis.** IEEE Software. v. 23. 4. pp - 58-61, 2006. Available in http://ieeexplore.ieee.org/abstract/document/1657940/. Last visited on 3rd December, 2017.
[28] **Codacy code reviews & code analysis.** Codacy Automated Code Review, 2017. Available in https://www.codacy.com/. Last visited on 3rd December, 2017.
[29] **ab: Apache Benchmarking tool.** The Apache Software Foundation, 2017. Available in https://httpd.apache.org/docs/2.4/programs/ab.html. Last visited on 3rd December, 2017.
[30] WWDC. **High Efficiency Image File Format.** Apple Inc., session 513, 2017. Available in https://developer.apple.com/videos/play/wwdc2017/513/. Last visited on 3rd December, 2017.
[31] Friesenhahn, B. **Introducing GraphicsMagick Project.** 2003. Available in https://marc.info/?l=imagemagick-developer&m=104777007831767&w=2. Last visited on 3rd December, 2017.
[32] **GraphicsMagick Image Processing System**. GraphicsMagick Group, 2017. Available in http://www.graphicsmagick.org. Last visited on 3rd December, 2017.
[33] Pinto, H. ***picel* encoder tool (JavaScript).** GitHub, Inc., 2016. Available in https://github.com/henvic/picel-js. Last visited on 3rd December, 2017.

[34] Cook, T. **Hammering Usernames**. Facebook, 2009. Available in https://www.facebook.com/note.php?note_id=96390263919. Last visited on 3rd December, 2017.
[35] **Dark Launch.** DevOps Dictionary, 2015. Available in http://devopsdictionary.com/wiki/Dark_Launch. Last visited on 3rd December, 2017.
[36] **Amazon Elastic Transcoder.** Amazon Web Services, Inc., 2013. Available in https://aws.amazon.com/elastictranscoder/. Last visited on 3rd December, 2017.
[37] **A Year Without a Byte.** Flickr, 2017. Available in http://code.flickr.net/2017/01/05/a-year-without-a-byte/. Last visited on 3rd December, 2017.
[38] **The Zettabyte Era: Trends and Analysis.** Cisco, 2017. Available in https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/vni-hyperconnectivity-wp.html. Last visited on 3rd December, 2017.
[39] Tanenbaum, W. **Computer Networks (The Application Layer).** Pearson Education, Inc, 5th ed., 2011.
[40] Wang, A.; Tonse, S. **Announcing Archaius: Dynamic Properties in the Cloud.** Netflix, Inc., 2012. Available in https://medium.com/netflix-techblog/announcing-archaius-dynamic-properties-in-the-cloud-bc8c51faf675. Last visited on 3rd December, 2017.
[41] **Network Service Tiers: Custom Cloud Network.** Google (Alphabet, Inc), 2017. Available in https://cloud.google.com/network-tiers/. Last visited in 27th November, 2017.
[42] **Wikimedia servers.** Wikipedia, the free encyclopedia, 2017. Available in https://meta.wikimedia.org/wiki/Wikimedia_servers. Last visited in 27th November, 2017.
[43] Stringfellow, A. **What Is N-Tier Architecture?** DevOps Zone, 2017. Available in https://dzone.com/articles/what-is-n-tier-architecture. Last visited in 27th November, 2017.
[44] **Database server.** Drupal System Requirements, 2016. Available in https://www.drupal.org/docs/7/system-requirements. Last visited in 3rd December, 2017.
[45] Berners-Lee, T.. **Frequently asked questions - What were the first WWW browsers?** W3C Recommendation, 2017. Available in https://www.w3.org/People/Berners-Lee/FAQ.html#browser. Last visited in 3rd December, 2017.
[46] Kindberg, T.; Spasojevic, M.; Fleck, R.; Sellen, Al. T**he ubiquitous camera: An in-depth study of camera phone use.** IEEE Pervasive Computing, v 4, n 2, pp - 42-50, 2005.
[47] **Share of devices used to access the internet at home, in the United States, from 2010 to 2015.** Statistic, 2010. Available in https://www.statista.com/statistics/199055/devices-used-to-access-the-internet-at-home-in-the-united-states/. Last visited 25th March, 2017.
[48] **Interesting Stats.** HTTP Archive, 2016. Available in http://httparchive.org/interesting.php?a=All&l=Nov%201%202017. Last visited 1th November, 2017.
[49] Everts, T. **Page bloat: The average web page size is more than 2MB.** SOASTA, Akamai, 2015. Available in https://www.soasta.com/blog/page-bloat-average-web-page-2-mb/. Last visited 3rd December, 2017.

[50] **Average Web Page Breaks 1600K.** WebSite Optimization, 2014. Available in http://www.websiteoptimization.com/speed/tweak/average-web-page/. Last visited 3rd December, 2017.
[51] **Manual: Imagem Administration.** MediaWiki, 2017. Available in https://www.mediawiki.org/wiki/Manual:Image_administration. Last visited 3rd December, 2017.
[52] King, A. B. **Website Optimization: Speed, Search Engine & Conversion Rate Secrets.** O'Reilly Media, Inc., 2008.
[53] **File:Wikimedia-servers-2010-12-28.svg.** Wikimedia, 2010. Available in https://meta.wikimedia.org/wiki/File:Wikimedia-servers-2010-12-28.svg. Last visited 3rd December, 2017.
[54] Namiot, D.; Sneps-Sneppe, M. **On micro-services architecture.** International Journal of Open Information Technologies, v. 2.9, pp - 24-27, 2014.
[55] **Who is using thumbor.** Globo.com, 2017. Available in http://thumbor.readthedocs.io/en/latest/whos_using_it.html. Last visited 3rd December, 2017.
[56] Pinto, H. **Go repositories licenses on GitHub.** GitHub, Inc., 2017. Available in https://gist.github.com/henvic/5fa7abcf3518ed5f29a66eaa04e00ea2. Last visited 3rd December, 2017.
[57] Kinsella, S. **Do Business WITHOUT Intellectual Property.** Liberty.me, 2014. Available in http://www.stephankinsella.com/wp-content/uploads/publications/kinsella-do-business-without-ip-2014.pdf. Last visited 3rd December, 2017.
[58] Raymond, E. S; Raymond, C. O. **Licensing HOWTO.** catb.org, 2012. Available in http://www.catb.org/~esr/Licensing-HOWTO.html. Last visited 3rd December, 2017.
[59] Nimmer, T. R. **Open source license proliferation, a broader view.** 2005. Available in www.ipinfoblog.com/archives/licensing-law-issues-open-source-license-proliferation-a-broader-view.html. Last visited 3rd December, 2017.
[60] Schröder, P. **Thumbor image loader docs.** GitHub, Inc., 2015. Available in https://github.com/thumbor/thumbor/wiki/Image-loader. Last visited 3rd December, 2017.
[61] **JPEGmini Server**. Beamr Ltd., **2017**. Available in http://www.jpegmini.com/server. Last visited on 3rd December, 2017.
[62] **Unrestricted File Upload.** Wikipedia, the free encyclopedia, 2017. Available in https://www.owasp.org/index.php/Unrestricted_File_Upload. Last visited 3rd December, 2017.
[63] **High Efficiency Image File Format (HEIF)**. Moving Picture Experts Group (MPEG), 2017. Available in https://mpeg.chiariglione.org/standards/mpeg-h/image-file-format. Last visited 3rd December, 2017.
[64] **HEIF Comparison with other formats.** Nokia Technologies, 2017. Available in http://nokiatech.github.io/heif/comparison.html. Last visited 3rd December, 2017.
[65] Alvarez, P. **Using Extended File Information (EXIF) File Headers in Digital Evidence Analysis.** International Journal of Digital Evidence, v. 2, 3, 2004.
[66] Pinto, H. **Vehikel.** GitHub, Inc., 2013. Available in https://github.com/henvic/vehikel. Last visited 3rd December, 2017.
[67] Pinto, H. **Vehikel (video).** YouTube BR, 2014. Available in https://www.youtube.com/watch?v=dML0FQIUcTY. Last visited 3rd December, 2017.

[68] **GD Graphics Library.** Wikipedia, the free encyclopedia, 2017. Available in https://en.wikipedia.org/wiki/GD_Graphics_Library. Last visited 3rd December, 2017.
[69] **WeDeploy.** Liferay, Inc., 2015. Available in https://wedeploy.com. Last visited 3rd December, 2017.
[70] **Go.** The Go Programming Language, 2009. Available in https://www.golang.org/. Last visited 3rd December, 2017.
[71] Developers Google. **Google I/O 2010 - Go Programming (video).** YouTube, 2010. Available in https://www.youtube.com/watch?v=jgVhBThJdXc. Last visited 3rd December, 2017.
[72] Stonebraker, M. **The Case for Shared Nothing.** University of California, Berkley, Ca., 1997**.** Available in http://db.cs.berkeley.edu/papers/hpts85-nothing.pdf. Last visited 3rd December, 2017.
[73] Horohoe, C. **Wikimedia moving to Elasticsearch**. Wikimedia Foundation, 2014. Available in https://blog.wikimedia.org/2014/01/06/wikimedia-moving-to-elasticsearch/. Last visited 3rd December, 2017.
[74] **Help:Images.** MediaWiki, 2017. Available in https://www.mediawiki.org/wiki/Help:Images. Last visited 3rd December, 2017.
[75] Vakali, A.; Pallis, G. **Content delivery network status and trends.** IEEE Internet Computing, v 7, 6, 2003.
[76] Krueger, C. **Software reuse**. ACM Computing Surveys (CSUR), v 24, 2, pp - 131-183, 1992.
[77] Pike, R. **Go: Ten years and climbing.** 2017. Available in https://commandcenter.blogspot.com.br/2017/09/go-ten-years-and-climbing.html. Last visited 3rd December, 2017.
[78] **OpenStack project**. OpenStack, 2017. Available in https://www.openstack.org. Last visited 3rd December, 2017.
[79] **Open Compute Project.** Facebook, 2017. Available in http://www.opencompute.org. Last visited 3rd December, 2017.
[80] **RedHat OpenShift.** RedHat, 2017. Available in https://www.openshift.com. Last visited 3rd December, 2017.
[81] **Cloud Infrastructure Automation.** Hashicorp, 2017. Available in https://www.hashicorp.com. Last visited 3rd December, 2017.
[82] **Docker.** Docker Inc., 2017. Available in https://www.docker.com. Last visited 3rd December, 2017.
[83] Konrad, A. **Open-Source Darling Docker Cracks The Billion-Dollar Club With $95 Million Raise.** Forbes Media LLC, 2015. Available in https://www.forbes.com/sites/alexkonrad/2015/04/14/docker-raises-95-million-at-billion-valuation/. Last visited 3rd December, 2017.
[84] **HashiCorp profile.** Forbes Media LLC, 2017. Available in https://www.forbes.com/profile/hashicorp/. Last visited 3rd December, 2017.
[85] Heynemann, B. **Thumbor: Escalabilidade em processamento de imagens com reconhecimento facial para 20 milhões de brasileiros.** InfoQ, 2014. Available in https://www.infoq.com/br/presentations/thumbor-escalabilidade-processamento-de-imagens. Last visited 3rd December, 2017.

[86] Savoia, A. **This Code is CRAP.** Google Testing Blog, 2011. Available in https://testing.googleblog.com/2011/02/this-code-is-crap.html. Last visited 3rd December, 2017.
[87] Pinto, H. **picel.** Travis CI, 2017. Available in https://travis-ci.org/henvic/picel. Last visited 3rd December, 2017.
[88] Pinto, H. **picel code coverage report**. Coveralls, 2015. Available in https://coveralls.io/github/henvic/picel. Last visited 3rd December, 2017.
[89] Gonzalez, R.; Woods, R. **Digital Image Processing.** Pearson Education, Inc, 3rd ed., 2008.
[90] W3C. **Privileged ports.** W3C Recommendation, 1995. Available in https://www.w3.org/Daemon/User/Installation/PrivilegedPorts.html. Last visited 3rd December, 2017.
[91] **Tar.** xkcd, 2017. Available in https://xkcd.com/1168/. Last visited 3rd December, 2017.
[92] Kenlon, S. **How to manage binary blobs with Git.** Opensource.com, 2016. Available in https://opensource.com/life/16/8/how-manage-binary-blobs-git-part-7. Last visited 3rd December, 2017.
[93] Pinto, H. **picel test assets.** GitHub, Inc., 2013. Available in https://github.com/henvic/picel/tree/test_assets/test_assets. Last visited 3rd December, 2017.
[94] Pike, Q. **Device Passthrough (Most notably, GPU) (issue)**. GitHub, Inc., 2016. Available in https://github.com/mist64/xhyve/issues/108. Last visited 3rd December, 2017.
[95] Pinto, H. **Requests to resize image with picel.** GitHub, Inc., 2017. Available in https://gist.github.com/henvic/458ec4e4cbc60c3b045799d1cbc4abfa. Last visited 3rd December, 2017.
[96] **PLEASE fix over-sharpening!!!** Flickr, 2012. Available in https://www.flickr.com/help/forum/en-us/72157630651445278/. Last visited 3rd December, 2017.
[97] **Command-line Tools: Compare**. ImageMagick Studio LLC, 2017. Available in https://www.imagemagick.org/script/compare.php. Last visited 3rd December, 2017.
[98] Dragoni, Nicola & Giallorenzo, Saverio & Lluch-Lafuente, Alberto & Mazzara, Manuel & Montesi, Fabrizio & Mustafin, Ruslan & Safina, Larisa. (2016). **Microservices: yesterday, today, and tomorrow.** https://arxiv.org/pdf/1606.04036.pdf
[99] Donald, James. **Improved Portability of Shared Libraries.** (2003). Improved Portability of Shared Libraries. Princeton, 2003. https://web.archive.org/web/20070926130800/http://www.princeton.edu/~jdonald/research/shared_libraries/cs518_report.pdf
[100] Jaison Paul Mulerikkal, Ibrahim Khalil. **An Architecture for Distributed Content Delivery Network.** School of Computer Science, RMIT University, 2007. http://ieeexplore.ieee.org/abstract/document/4444113/

[101] Goerge Pallis, Athena Vakali. **Insight and Perspectives for Content Delivery Networks.** Communications of the ACM, v. 49, pp - 101-106, 2006. https://dl.acm.org/citation.cfm?id=1107462
[102] Baliga, Sandeep. Sjöström, Tomas. **Not Invented Here.** Kellogg School of Management at Northwestern University August 24, 1997. https://ksmb55a.kellogg.northwestern.edu/research/math/papers/1213.pdf