Design of the technological architecture for PUMPIT project

Tomáš Bruckner
Department of Information Technologies, Faculty of Informatics and Statistics, University of Economics, Prague, Czech Republic
bruckner@vse.cz

DOI: 10.20470/jsi.v10i2.370

Abstract: A specific technological architecture of infrastructure and specific software elements is designed and evaluated for project of Pricelist of Usual Manday Prices of IT services. The principles for the architecture are based on requirements of vendor lock-in prevention, maximum ease of deployment, and maximal use of free opensource solutions. The resulted architecture is designed as Debian Linux host environment running Docker containers with isolated applications with own scripts in node.js, a community software as Elasticsearch, Kibana, Ethercalc, Nginx etc. The architecture was designed and evaluated iteratively during five runs of the project and is prepared for next phases.

Key words: Usual Price, Pricelist, Information Technology, Technological architecture, Docker, Stateless, Infrastructure as a Code

1. Introduction
This research is a part of a project, which goal is to prepare a Pricelist of Usual Manday Prices of IT related services (hereinafter the PUMPIT project), based on actually negotiated prices in contract published in public registry of contracts. The project is research initiated by 13 public institutions in Czech Republic, who plan to use the results as a material for preliminary price assessment of planned IT project during preparations of public tenders. The project outcome is the pricelist itself, built periodically on moving data, as well as methodology, processes, quality assurance and infrastructure suitable for the pricelist, and finally application of the mentioned above in first five performances of the pricelist construction (Bruckner 2018). This paper addresses the technology infrastructure part of the PUMPIT project.

2. The problem identification
The main goal from the point of view of the PUMPIT project is to design suitable infrastructure for the project. The problem is: which infrastructure is the suitable one, or, more generally, which principles should meet architecture of the infrastructure for this one or for similar projects. In this paper the identified principles are taken as the initial assumptions.

The goal of the paper is to design architecture of infrastructure, which meets the given principles, to demonstrate the properties for specific elements of the architecture and evaluate the architecture on first prototype of the project infrastructure by using the infrastructure for initial run of the PUMPIT project including the creation of the outcome: the pricelist.

The research method is stated in accordance with design science methodology of Peffers (2007), the parts of this paper address specific steps of the method, which are: problem identification and motivation, definition of the objectives for a solution, design and development, demonstration, evaluation and communication (where the communication step is taken as this paper itself).

3. Definition of required infrastructure principles
The presumed principles of the infrastructure come out from the nature of the PUMPIT project to which the infrastructure is applied:

First, considering that the PUMPIT project customer is the public sector, it is suitable to eliminate possible vendor locking and enable simple change of suppliers using the infrastructure, the method and the processes to make the pricelist in future iterations. Thus, better price could be achieved by vendor competition. It is presumed, that several vendors will use this architecture to make future versions of the pricelist, so they should be able to use the architecture without additional license acquirement issues or costs. Related to minimal cost requirement this leads to principle of using software with free license, focusing wholly on long term community supported open-source software projects.
Next requirement is to achieve ease of installation and configuration and simple reuse of once configured environment. The infrastructure environment should be easily renewed even when the production would be stopped or lost, expecting iterations in order of half a year and as well expecting possible changes in subject who uses the infrastructure in future iterations. These requirements lead to following principles:

- **Non-persistent applications.** Run of applications should be configured in a manner that downtime, system crash or system shutdown could not cause data loss. The data should be stored persistently outside configuration files and scripts and outside runtime environment of the applications, and therefore meet the minimum stateless service principle (Erl 2008).

- **All applications should be run in separated environment,** so that they are unable to influence other applications. This is important in relation with the open source principle, as the behavior of application is hardly predictable. The crash of one application should not mean crash of the whole environment.

- **Standard/community solutions.** Before configuration and installation is searched for pre-configured, ready-made and tested suitable solution made by communities.

- **All configurations should be documented by configuration files,** so that the deployment of the runtime environment was carried maximally automated. The configuration files should be handled the same way as the software/script files, e.g. should be versioned, as a DevOps principle Infrastructure as a Code (Morris 2015).

Those was the infrastructure principles as objectives for the solution, inspired by (Hoffmann 2016). Other requirements are determined by the method and the process defined by the PUMPIT project, here also taken as presumptions:

The architecture should cover the whole process designed by the PUMPIT project. The process consists of five steps: The Data acquisition, the Contract analysis, and the Price analysis including creation of the pricelist. The PUMPIT project is performed by small group of actors in two areas: (1) the IT professionals and (2) the IT services and contracts professionals. The run of the project is controlled by the IT professionals, and then the run of the software could be treated as system administration from command line interface. The parts of the project which involve the IT services and contracts professionals should be treated in the manner of common users of computer or mobile applications. These subsume the searching and displaying the contract documents and in co-operation validating and editing automatically extracted manday prices from the contracts.

Although the designed architecture is expected to be a simple and prototype solution, it should meet all properties stated by this chapter. For this paper the infrastructure principles are taken as crucial, because their fulfilment could be replicated to other similar projects and the results are generalizable.

### 4. Design and development of the solution

The initial prototype of the PUMPIT project was planned for three months, which included creation of all outcomes of the project as a prototype. Such short time enforced simultaneous creation of the pricelist, the method, the process and the infrastructure. Therefore, the technologies for the project was iteratively searched, examined and tested, simultaneously with, or short time before their usage. The usability of the technology was instantly examined, and the technology elements was changed or adjusted according their pragmatic usability for the project outcome. The initial technology architecture was designed in the beginning as the expected solution, and it was modified by exchanging its elements every time the single used technology was changed. When choosing technologies and building the architecture was considered the twelve-factors app principle (Wiggins 2017).

Some aspects of the development are contextually described in next chapters.

### 5. Demonstration of resulting architecture and chosen technologies

In this part is demonstrated the resulted technological architecture and specific technologies designed for the project related to the required principles.

#### 5.1. Solution architecture

The Figure 1 describes overall diagram of all used technologies. It is ordered from left to right by the process from acquiring the data resources, across the processing to the output. From bottom to up it is ordered from infrastructure technologies to application technologies.
In next chapters the specific technologies and its chosen principles are described:

### 5.2. Hardware and connectivity

For prototype solution was chosen standard server on Intel x86-64 platform, running virtualized (vmware) as a part of server farm of university of Economics Prague.

**Specification:**
- Dedicated internal memory is 12GB RAM (during development twice increased from 4GB to 8GM and finally to 12GB), 2GB swapfile, direct attached disk memory 1TB, 8 core Intel(R) Xeon(R) CPU E5-2420 0 @ 1.90GHz
- Server is connected to university local area network and over it to the Internet. Port 22 (SSH) is connected to the Internet and ports 80 (HTTP) and 443 (HTTPS) are open to the local network.
- Internet connection speed is 10 Gb/s
- Server has dedicated IPv4 address and a hostname.

### 5.3. Operating system Debian

As operating system there was chosen Linux Debian version Jessie. Debian is a free software based on social contract (Debian 2004).

Users are project team members, who connects to server via SSH with possible password and private key authentication form LAN and private key only from Internet.

The operating system is used as a platform for run of Docker containers and for persistent data storage. It is used in most simple configuration. First configuration is to install git versioning system and to connect to the files repository to download configuration files. Then, simple setup forced by Docker platform and Elasticsearch container (see below) are configured in specific bash deploy script, which is meant for primary configuration of the newly installed server and for Docker platform installation. Th Debian Jessie server environment is further referred as the host.

### 5.4. Virtualization of containers Docker and Docker-compose

For safe run of more separated applications interconnected by a virtual network was chosen platform Docker Community Edition (Docker 2019), distributed as an open source under Apache 2.0 license (Apache 2004).

Every application used in the project is run inside Docker container, so that the applications can’t mutually collide on the operating system level.
Docker-compose software (Docker Compose 2019) is used for cumulative start of infrastructure elements, applications and for virtual local network setup used for container mutual communication. The configuration is set in a docker-compose configuration file, so the full deployment can be started by running one command in Linux command-line interface.

The docker-compose configuration file as well as the configuration files for specific container are stored in one common file structure versioned by git system, so all configuration versions are managed. For container interconnection a virtual network is configured. Due difficulties with routing through proxy the docker-compose.yml configures fixed IP addresses for containers.

5.5. Reverse proxy Nginx

For access management of team members to specific applications was chosen Nginx software configured as reverse proxy (Nginx 2019). Nginx is open source software distributed under BSD-like license http://nginx.org/LICENSE.

The reverse proxy accepts requests for connection on port 80 and distribute it to specific containers. Nginx itself runs inside container, which uses official community nginx:stable-alpine, available from (Docker Hub 2019) and is reconfigured by nginx configuration files injected into the container during build.

Container Nginx is the only one which opens port to host (Debian) and thus to local university network. All other communication between architecture elements is performed via docker virtual network, due to limitation of unauthorized access. It is advisable due complex open source software like Elasticsearch may enable hacker attack when carelessly opened to the internet.

5.6. Node.js, data acquisition

Data acquisition from public registry is realized by own software scripts written on node.js (Node JS 2019) platform, and particularly in C#.

For main data acquisition, i.e. contract download, reading/ocr, indexing into Elasticsearch (see below) is used container based on community node:boron-slim (Docker Hub 2019), into which is during build installed software for ocr and text extraction from different types of files (xpdf, antiword, unrtf, tesseract-ocr). Engine node.js run in the container is parametrically optimized for low memory load and frequent garbage collecting due unpredictable memory load by text extraction software. Another auxiliary scripts that we wrote are run in their own containers.

For extracting plain text from several formats of contracts, a package of node.js text extraction tools Textract is used (Textract 2019).

5.7. Contract database and text analysis in Elasticsearch

For contract data (both fulltext and metadata) is used software Elasticsearch (Elasticsearch 2019) distributed under Apache license. Its Docker container comes from standard official container docker.elastic.co/elasticsearch/elasticsearch:5.3.0 from (Elastic Container 2019).

Its data folder is mapped to persistent data/elasticdata on host. Elasticsearch from some reasons obeys Docker standards and saves data under linux user UID:GID 1000:1000, and thus it is necessary to ensure the persistent folder elasticdata on host is accessible for read and write by this user.

In Elasticsearch container (as well as Kibana) is installed security software X-pack security by default (X-Pack 2019). Because this software is against principles above, in this configuration is switched off. The container listens in docker virtual network on hostname elasticsearch and port 9200.

In Elasticsearch is enabled CORS (Cross Origin Resource Access), which allows direct access into database management over Head plugin on specific address (configured in Nginx proxy).

5.8. Text analysis visualization – Kibana

Software Kibana (Kibana 2019) is used for visualization of data stored in Elasticsearch, for analysis of the content of the contracts and for determination of the way how price data will be extracted. Kibana is distributed under Apache license. Docker container with Kibana comes from standard official container docker.elastic.co/kibana/kibana:5.3.0 (Docker Hub 2019) and it is configured by configuration file due to switching off the X-pack security software. The container listens in virtual network on address kibana and port 5601, and this port is using reverse proxy open to local network on its own URL on port 80.
5.9. Direct access to Elasticsearch – Head
To display status of elasticsearch cluster and index and for direct input of some data to Elasticsearch, special container with software Head (Head 2019) is used (known as plugin Head). Community container mobz/elasticsearch-head:5-alpine is used (Docker Hub 2019). It is accessible over reverse proxy on specific URL.

5.10. Prices data storage and frontend for validation – EtherCalc
For price data storage and editing was chosen EtherCalc software (Tang 2019). EtherCalc is a spreadsheet developed and running on node.js technology. It stores data in Redis database. The spreadsheet is accessed by web browser and provide possibility of simultaneous co-operation of multiple users, and as well enables program access by API.
Engine EtherCals runs in its container, and for data storage uses another container with running Redis engine listening on 6379 port. Both containers, same as all mentioned above, are tolerant to fault or restarts, data are persistently stored outside containers on host in its folder specified by docker-compose configuration.
Docker container Redis is standard community container redis:latest (Docker Hub 2019), for EtherCalc there is made special build based on official container node.js node:boron-slim (Docker Hub 2019), due to ensure working behind reverse proxy (using configuration parameter basepath) and also CORS (cross origin resource sharing) is configured. Both containers are configured to Docker virtual network to be accessible by scripts. Ethercalc container opens port 8080 to virtual network and is accessible from world through reverse proxy on its URL. Similarly, as other used technologies both containers are started by docker-compose technology.
As a template for price tables there is prepared a structured file in SocialCalc format, which is applied to created tables by script.

5.11. RStudio
Statistical analysis is made by RStudio statistical software (RStudio 2019). RStudio is open source project under AGPL license, which enables run of R statistic software environment and running R scripts. Even automated, in prototype the analysis is run outside the server environment on PC on local RStudio installation. In Future the architecture is prepared for using the server version of the software, the R Server. All necessary libraries and dependencies for run of the script are automatically downloaded during first run of the script. The used libraries are: ggplot for graph plot, knitr for table plot, dplyr for statistical calculations, stringr for text manipulations and jsonlite for working with json data format.

5.12. Deployment of the infrastructure
To deploy the infrastructure to production environment it is necessary only to run one BASH deployment script on the host machine. It is advisable first setup rights for specific Linux users, and add them to the Linux group of the project, and linux group docker to allow runnig of the docker containers.
Also, the containers are configured to run after deployment, and the script are configured to start directly work. In case the environment should be ready some time before starting the project (or the follow up project), the configuration should be changed to delay the script operations.
In this prototype setup there is only one server without scaling, because the analyzed data and number of users are considered as not big. Nevertheless, the architecture is scaleable, even in case of the elasticsearch is scalable "out of the box", it is enough to run next machines with the same configuration (host environment and with elasticsearch container running) and add its network address to configure elasticsearch cluster.

6. Evaluation
The usefulness of the architecture and all the chosen technologies of the infrastructure was evaluated during the run of the PUMPIT project. The project successfully fulfilled its goals and the infrastructure was considered as applicable. The infrastructure is used in project continuation and is supposed to be used also in next phases. Use of specific technologies is subject of continuous innovation during following project phases.
Although the requirements were fulfilled, several issues occurred during the design and evaluation. The presumption of use of unconfigured host with applications running only in containers seems to be unachievable, because some available containers need some (even minor) configuration on host. The
Elasticsearch uses specific user UID:GID to store persistent data, so this user must be configured on host. Also, it necessary to adjust memory mapping in virtual address space on host, or the dockerized Elasticsearch crash on low memory. Some problems also occurred with setting interfaces between containers by environmental variables and by using default virtual network addresses. These were in the prototype solved temporarily by fixed IP addresses in the virtual network. Those issues somehow limit the benefits of the architecture, as they make more difficult to use commodity hosts and brings some unwanted integration dependencies in configuration.

Also, some opensource software for document reading and OCR in some specific cases behave unpredictably. This software was controlled by node.js script in a separate container. Sometimes the software during OCR captured huge memory inadequate to the size of the read document. This caused the whole host swapping RAM to disk and froze the system for few minutes. When assigned more host memory (from 4 to 8, and finally to 12 GB), the problem was solved for some cases. In some cases, the software completely froze the container, so the script was not able programatically solve the situation. For these cases was written a BASH script run on host, which monitored the outputs of the OCR container and in case of long inactivity deletes the container and creates new using docker-compose. In this particular situation the designed architecture appeared as very useful, because the problems of specific opensource software were isolated inside the container (otherwise the whole system would be affected), and moreover, the configuration of the stateless application in container and persistent data on host allowed not to solve the root cause of the problem by difficult refactoring of used community software, but instead of it use simple and pragmatic solution by renewing the affected container from the scratch.

7. Conclusion

The goals of the design were achieved. The solution architecture was designed and evaluated during the project and is prepared to be used with ease in further following phases of the project. All the configuration files are handled same as code files and are part of the project outcome. All used software is distributed under several license types which all allow free commercial use. Thus, the architecture and configurations are prepared to be used by several subjects, whoever will in future work on the further project phases. The deployment of the configuration is automated, it is tolerant to faults and restarts. All applications are run in separate environment, so the architecture is resistant to critical faults of single specific software. Also, the virtual network ensures that the environment is not open to the internet and thus relatively safe. The architecture meets all the defined principles.

8. References

Apache License ver. 2.0. 2004. Available at https://www.apache.org/licenses/LICENSE-2.0 [Accessed 1 April 2019]
Hoffmann, K., 2016: Beyond the Twelve-Factor App. O’Reily
Morris, K., 2015: Infrastructure as a Code. O’Reily


