Haaga-Helia
University of Applied Sciences

Working as a backend software developer at F-Secure

Jim Komar

Bachelor's Thesis
Degree Programme in Business information technology
2019

Haaga-Helia
University of Applied Sciences

Abstract

22.11.2019

Author(s)
Jim Komar

Degree programme
Business information technology

Report/thesis title
Working as a backend software developer at F-Secure

70 / 1 pages

In this portfolio-like diary thesis I discuss my working tasks and methods as a junior software developer at the Finnish cyber security company F-Secure. The main purpose of the thesis to trace and analyse the development of my skills and my professional growth as a backend software developer according to the set list of goals in several development areas.

The diary covers a period of 10 working weeks, during which I describe my main work duties, tasks and responsibilities and perform a weekly research-based analysis of my learnings, reflecting on the new skills and my professional development in such main areas as Java, Python, Jenkins, Amazon Web Services, Splunk, audit logging, testing.

The result of this work is a detailed analysis of my professional achievements in each development area that I had set for myself, illustrated with specific examples from the diary. The conclusions show that I have reached all the goals that I have set for my development, and have grown a lot as a software developer.

The appendix contains a list of the abbreviations used in this work.

Keywords
Backend development, Java, Python, Jenkins, AWS, Splunk, audit logging, testing

Haaga-Helia
University of Applied Sciences

Table of contents

Table of illustrations

Haaga-Helia
University of Applied Sciences

1 Introduction

The observation period for this thesis is from the 1st of September 2019 to 29th of November 2019, thus making it roughly 13 weeks, including introduction and conclusion.

During this time, I will be working as a backend[1] developer at F-Secure. F-Secure is a leading Finnish cyber security and privacy company that provides related solutions and services to both businesses and consumers. I am a part of the team that works on backend services for one of the company's products. Backend development is sometimes also called "server-side development", which means that this area of web-development concentrates on operations between the server and the database, in other words, the processes that are happening "behind" the web-browser.

Due to the nature of the company's business, my non-disclosure agreement does not allow me to include many details about the services I work on, unedited pieces of code or screenshots, so I will mostly be talking about the kinds of tasks and generalized technical nuances of my everyday work.

My team consists of 3 developers, including myself. The other two are a mid-level developer and a senior developer/service owner, who is our boss and manager. Our team is just one of the many teams working on various products and services of F-Secure, from which we are fairly independent. As a team, we all are generally working on the same global tasks, dividing them into sub-tasks and distributing among ourselves according to our abilities, skills and knowledge.

When I started writing the thesis, I had already been working there for more than a year. The objective of this work is a detailed overview of my professional development and progress of my skills. As a result, I will have documented my professional growth and development during the period of writing the thesis. It will allow me to see my progress and to analyse the way I am developing and learning new skills.
The framework used daily in my working life includes:

[1] There is an ongoing debate about the usage of the term, with the options being "back end", "backend" and "back-end" (Pronschinske, 2013). For the consistency purposes, in this work the term will be referred to as "backend", for both adjectives and nouns.

1. Locally installed software: Java Development Kit for Java 8, IntelliJ IDEA (Java IDE), Python (2.7, 3.4, 3.7), PyCharm (Python IDE), Git, MySQL Workbench, Apache Tomcat, Amazon DynamoDB Local
2. Tools and services: Amazon Web Services, Jenkins, GlassFish Server
3. Technologies and frameworks: Spring Framework, Maven, Hibernate

The topics covered by this thesis are too diverse to have some main definite sources, so I am using a combination of various sources for topics including but not limited to Jenkins, Splunk, Python, Amazon Web Services, Java, Spring configuration, testing, logging. Documentation, manuals and developer guides are the most significant sources for theoretical and practical information, supported by expert blogs, various web sources and articles for learning the general information, best practices and practical advice for the studied technologies.

Stack Overflow, which is basically a professional online community for developers, is probably the main source of practical information for every developer because it allows to find an answer to almost any question related to code, errors, conventions, and advice for any related topic. It is an indispensable source of knowledge that often saves a lot of time and effort when writing and debugging the code. It is used a lot by me while performing my work duties during the period of writing this thesis. However, in my weekly analysis entries I rely mainly on the other sources.

Talking about necessary skills, the first and foremost skill needed in my work duties is a solid knowledge of Java, REST API, server-side web-programming and general understanding of web-services. Those were the key requirements of the job offer and are my primary focus in daily work. As I progressed in my internship, I acquired more skills that are now essential for my everyday work. Those skills include using Java-related frameworks, such as Spring and Maven; Python, both for software development and testing; writing and using unit tests, integration tests and system tests using such frameworks as JUnit and Mockito for Java, and PyTest and Nose tests for Python.

Testing in general is a big part of my duties because in our work environment testing is one of the primary points of focus, even though there are other teams fully focused on testing and test automation. Each new piece of code has to be covered by unit and sys-

tem tests, and the development pipeline depends on successful completion of several layers of tests, only allowing the changes to the application to be deployed if all the tests pass. Therefore, broken code will not be able to turn up in production environments.

Another primary skill is knowledge of Git, as our workflow is based on commits, pull requests and peer reviewing. It is necessary to understand the principles of Git, although detailed knowledge of the command-line language is not necessarily required as a lot of operations can be performed using a user interface. However, this knowledge proves quite useful, as some fine-tuning might be not always possible to do via the UI.
Knowledge of Amazon Web Services became my second most important skill after Java because most of the applications I work on are deployed to AWS. So, I had to learn to use, maintain, create and debug various Amazon services, the most important and heavily used being CloudFormation (scripting the AWS stacks), EC2 (cloud computing), S3 (storage service), DynamoDB (non-relational database), ELB (load balancer), IAM (permissions and access management), Route 53 (DNS services), Certificate Manager (security certificates), Lambda (serverless computing), CloudWatch (monitoring).

Apart from that, we use Jenkins automation server for deployment and testing of our applications. This implies the ability to utilize, create and maintain Jenkins jobs and Pipelines, which are written in Groovy programming language with often usage of shell scripting.

My work tasks always differ, so that I need to learn a lot of new skills in the process.

1.1 Key professional concepts

Amazon web services	Cloud computing platform
Back-end development	Server-side web-development
Debugging	Identifying and removing errors from the code
Java	Programming language
Jenkins	Automation server
Kanban	Agile software development methodology
Maven	Java framework for build automation
MySQL	Database management system
OPI	One of F-Secure internal services
Python	Programming language
RabbitMQ	Messaging software
SEBE	One of F-Secure internal services
SMI	One of F-Secure internal services
Sonar (SonarQube)	Code quality inspection software
Splunk	Monitoring software
Spring	Java application framework
Testing	Part of software development process

2 Framework

2.1 Analysis of your current work

In F-Secure, our working process is usually defined by the quarterly goals and tasks every team sets for themselves. My main work tasks for the next quarter are participation in the following activities:

1. Improve audit logging in one of the old backends (SEBE).

That means getting acquainted with a very big old project, which contains a lot of legacy code, and completely change the implementation of the logging. To do so I will first of all need to understand the logic of the project. These kinds of tasks usually take a lot of time, especially with projects as large as this one. The backend service contains 6 interdependent applications and over a hundred of endpoints. Then I will have to add the necessary annotations to the service's endpoints and write the related code. Finally, the whole thing has to be tested in Splunk to see if the logs now work as expected.

My current level of knowledge of audit logging framework is beginner. I have a general knowledge of what it is, and have done some copying and pasting in another project before, but I do not have a real understanding of how these things work and why the logging should be used. Therefore, I will need to learn how to use it by following the existing examples in other applications and doing my own research.

2. Configure Splunk for improved audit logging.

Splunk is a software that is used for log monitoring. My team is already using it for all the projects that we have but the planned audit logging improvements require some major configuration changes. I need to learn how to pick up the necessary logs, reroute them to specified indexes and drop those logs that we do not need to reduce traffic and used memory.

My current level of knowledge of Splunk is beginner. I have used it for monitoring and debugging our applications but never done any configuration work and know almost nothing about the way it works. I will need to read through the documentation and look into the examples of implementation of remotely similar tasks by other teams in the

company. In a good case, those will have to be slightly adjusted to our applications. In a bad case, we will have to figure it out on our own. After writing the code for the configuration changes, the audit logging improvements have to be tested in Splunk.

3. Set up a new AWS environment and deploy SEBE.

This task means creating a whole new separate environment for the SEBE backend service. That means looking into the existing Jenkins jobs and AWS stacks for other applications and doing the same for this service.

Since I have experience with preforming similar activities for another project I was working on during the previous year, I would take a liberty of calling myself a skilled performer for this task as I am fairly familiar with the general process, as well as smaller nuances. My prediction is that this will not take a lot of time because the general layout for the environment (AWS stacks, DNS routing, hosted zones, certificates and other time-consuming configurations) has been created / maintained by me or other team-members before. So, this task's focus is mostly on creating Jenkins jobs for the new environment and setting up the system tests to run against it.

4. Set up a local development environment for the SEBE project.

This is necessary to be able to test the changes I make and the code that I write without deploying it to the Jenkins Pipeline, but by running and testing it from my machine. I have never worked with this particular project before, so I do not have all the needed frameworks and software installed on my machine. I need to: a) install the old version of Python (2.7) and the requirements used for running the system tests (various Python libraries); b) set up a local instance of MySQL database.

I would say this particular task makes me a novice because it is very project-specific. I do not have a lot of experience with Python 2, and have never used a MySQL database. Moreover, I do not know the dependencies and potential problems of this large project and I have no experience in setting up this specific combination of tools and frameworks. With some assistance of my team-mates I should be able to succeed in setting up the environment but I suspect that I might face quite a few unexpected difficulties because configuring a new environment from scratch is always very different to

just showing someone the way it works. There usually are many details that nobody remembers anymore after several years of using the system.

5. Add a new endpoint for one of the internal protocols.

We need to add a backward compatibility for one of the protocols (OPI) used by the SEBE service. For that I have to add a new endpoint for the OPI1.1 and copy + adjust some of the functionality of OPI1.2. After writing the code I will have to write unit tests in Java and system tests in Python.

These are exactly the tasks I am most used to, as they contain only programming an testing and no configuration whatsoever, so I would call myself a skilful performer, with the only amendment that this service in unfamiliar to be, which can possibly slightly complicate the matter.

6. Smaller tasks that appear on the way or are passed on by other teams' requests.

For these it is difficult to predict the exact actions or necessary knowledge. There are two possible scenarios: either the task will be small and easy and match my skills, or I will have to learn something new as I work on it.

2.2 Development areas and goals

My current knowledge and skill-set corresponds to one of a junior developer. I have around 1,5 years of work experience at F-Secure, where I started as a trainee who knew nothing apart from the school knowledge gained at Haaga-Helia. At the point of writing this thesis I would consider myself, on one hand, experienced enough in the familiar area of tasks and quite skilful with Java, Python 3, AWS and Jenkins in general. I am confident even with large, detailed and complicated tasks that require using this "basic" skillset, and I am usually able to perform these kinds of tasks quickly, reliably and mostly on my own, with minimal corrections by my team.

On the other hand, I definitely lack the breadth of knowledge of an experienced developer, and there is a lot of technologies and software used by my team in daily work that I have never dealt with before. Therefore, one of the main development areas for me is to learn as many new tools, frameworks and things as possible in order to progress and increase

my professional level. Looking at the current tasks, the main ones would be the following: Splunk, Audit logging, MySQL configuration.

The kinds of tasks, tools and frameworks that are completely new for me might take quite a lot of time to see into. For such tasks I usually require quite a lot of guidance, especially at the very start. However, talking about short cycle development, I can say that I pick up and figure out new things rather fast and one of my strong features is the ability to conduct independent research and find and test out the relevant information. Still, my work needs supervision, because even if something that I have done is working as expected, I cannot know if it is compliant to all the conventions, security standards, performance optimization principles and simply common sense. So, while making my research I should not only focus on getting the task done but also on learning the best practices, and look for professional advice to make the result of my work operate in the most efficient way.

My general work methods should become more test-driven, as I sometimes forget to write the tests for my code until my team-mates remind me of that. The ways I work have already changed significantly as by the permanent responsibility to write the tests I am forced to write the code that is easier to test, but I still feel that there is a long way of growth ahead of me. I should try researching and trying out the test-driven development (TDD) process, which is a “test-first” approach to software development, and see if it fits my working style and makes my performing of the work duties more efficient.

By reflecting upon the areas described above, I have set a list of specific goals for myself for the observation period of this thesis. The list includes the goals related to both learning new skills and developing the existing ones.

1. Audit logging:
    - Understand why and how it should be used in software development;
    - Improve the current usage in projects we work on.
2. Splunk:
    - Get familiar with Splunk processes and general principles;
    - Be able to perform the necessary administrative operations;
    - Finish the task of logging improvements.
3. MySQL:
    - Set up a local database and use it for testing the applications;
    - Understand the configuration and be able to set it up for other projects.

4. Java:
    - Develop my skills further, learn new useful techniques and good practices;
    - Learn more about frameworks (Spring, Maven, Hibernate), administration.
5. Testing:
    - Learn to use Nose tests for Python system tests;
    - Try out TDD;
    - Improve my approach to testing in general.
6. General skills development:
    - Jenkins;
    - AWS;
    - Python;
    - Learn about new software/frameworks I have not used before.
7. Successfully completing my quarterly tasks.

2.3 Interest groups at work

My personal work can affect the following interest groups:

- My team-mates who work on the same task within the application/service;
- Other back-end teams who work on the services relating to/depending on ours;
- Client teams who develop the client applications that are using the backend services that we maintain;
- Production who put the services into use;
- (Optional) Company's partners ("operators") who distribute the product to end users;
- End users (customers who use the final product).

These groups are depicted in the graph below (Figure 1). The horizontal line designates the division of the groups to internal and external: the upmost two bars of the pyramid, which are operators and end users, are external groups; the rest are internal.

Figure 1. Interest groups possibly influenced by my work at F-Secure

Therefore, my work can potentially affect a large number of people within the company, as well as go beyond and result in leaving a tangible impact on the customer. In other words, the successful results of my work allow the end users to utilize the F-Secure product, whereas an interruption in the service caused by my mistake, if it arrives to Production, can make them unable to use the product in full.

This feels like a big responsibility and during my first months at F-Secure, I was heavily affected by its weight. However, the interest groups that should be central for my concern are my team-mates and other back-end teams, because these two groups are the ones that are directly connected to my work and therefore would be directly affected by most of my actions. Nevertheless, I am not forgetting about the eventual area of impact of my actions, and that makes me take my duties more responsibly.

2.4 Interaction skills at work

Since my team is small and I am a junior developer, my work does not involve a lot of communications. Our team works using the Kanban agile development approach, so every morning we have daily team meetings, during which we briefly discuss the results of the previous day's work and plans/tasks for the current day.

Interaction with other colleagues mostly happens via Microsoft Teams chats and channels (with related backend and client teams) or sometimes by email of JIRA ticket (to get help

from Production) but it can also happen face-to-face. From time to time I answer questions or give guidance, both by chat-box and in person, on the topics I am competent in.

I do not have any direct interaction with customer services. If there is some comment related to my work it would be descended from Production to my team and then to me via my boss/manager or the team-mate.

For me personally dealing with people from outside of my team in person can be sometimes challenging due to personal character traits. I prefer using MS Teams when talking to somebody I do not know personally. However, sometimes this option is inconvenient, especially if you need to show or explain something to the person, which is always easier to do face-to face.

3 Diary entries

Since most of the tasks require a lot of figuring out the application structure and logic, the diary entries will not be covering every day of my work life in all details, as many of them would only consist of one activity which is looking into the existing code and learning how it works specifically in this or that part of the application or the project. Another option would be searching for the information related to the task of the problem on the internet, looking through web-forums and discussion, reading articles or book chapters.

I will provide diary entries for 2-3 days of the week when there was some actual progress, or challenges, or learnings, or start of another task. The rest should be presumed as "re-searching" or "debugging", often fruitless for quite a long time, while nothing specific hap-pens. Furthermore, I do not always work full hours on all the weekdays, as I still have 1-2 classes to attend, although my average working time is 32-33 hours per week. The results and learnings of the whole week will be summarized and covered in weekly analysis en-tries, some of which could be shorter than the others due to a smaller number of tasks or less working time.

3.1 Observation week 1 (September 2 – 6)

Monday 2nd – Tuesday 3rd September 2019
Task: Delete old AMIs.

Currently all the old AMIs (Amazon Machine Images, virtual appliances for creating virtual machines in AWS) are stored in AWS for one of our applications. I need to create a Jen-kins job that would use an existing internal tool which searches for the unused (inactive) AMIs matching the name, picks them up, prints them to the log and removes most of them, leaving only a specified amount of the most recent ones, plus the ones that are cur-rently in use. There is also a dry-run option which allows to run the tool without deleting anything. This option is important because it allows to test what is going to be deleted be-fore actually deleting it. In our case, we have hundreds of old AMIs, so we would definitely first want to check if the tool works as expected.

After familiarizing myself with the tool, I wrote a Jenkins job that takes two parameters (number of AMIs to keep and dry-run Boolean) and then runs the command to execute the tool. The external tools on Jenkins are run with shell scripts that take the job's parameters,

so I wrote a script picking the parameters from Jenkins and running the tool in either a standard or a dry-run mode.

I did not find this task to be hard, as I already have experience in writing Jenkins jobs, and this one is a fairly short and simple one. There was a small comment by my boss about the shell script I wrote, concerning the good practices, which made me restructure the script a little, but other than that there were no complications.

Key learnings: the mechanism and usage of the internal AMI deletion tool; some shell and Jenkins good practices.

Wednesday 4th – Friday 6th September 2019
Task: Set up a new AWS environment and deploy SEBE.

I was working on this task together with my team-mate. First of all, we took a look at the existing environment. The good thing was that we already had all the underlying things (Jenkins jobs, AWS stacks, permissions, etc) configured and a version of SEBE was already in AWS. But it was a vast and complex AWS environment that included not only SEBE but also another service and some extra things. So, what we had to do was to create a new standalone SEBE environment to be able to deploy applications to this development/testing environment separately and independently of anything else.

For that, we performed the following steps:

1. Figure out which existing jobs are needed and in which order;
2. Try manually running those jobs in the right order to check if that works;
3. Write a new Jenkins Pipeline which runs those jobs automatically.

The first step involved looking through the large Jenkins Pipeline that contained all the job-steps for the existing joint environment which we are going to partially reproduce. It took some time because of its volume and complexity but there were no complications.

The second step involved some trial and error in part of finding the right order. The environment needs many interdependent elements, some of which have to be created before the others that depend on them, and given the big amount of jobs, those dependencies

are not always evident. After several tries, we covered all the necessary parts and got it right.

The last step was basically just writing down everything that we did to a Groovy Pipeline script and running a job creator, so that the new Pipeline appears on Jenkins.

This task was exactly corresponding my skills and experience, so I only had to spend some time figuring out how the SEBE service is currently deployed. This and testing the right order of the jobs to run took most of the time, while writing the actual code was the smallest and simplest step of the task, as it usually happens in such cases.

Key learnings: the mechanism and structure of the new SEBE environment.

Weekly analysis

This week, I had rather simple tasks which, being somewhat time-consuming, were not of a particular difficulty to me and mostly depended on the skills I already possessed. Nothing really required any clarification, as most of the work to be done was quite clear and straightforward. It depended on already existing setups and tools, so it was easy to figure out and perform the tasks. However, I still learned a few things that were new to me.

According to Jenkins official documentation, “Jenkins is a self-contained, open source automation server which can be used to automate all sorts of tasks related to building, testing, and delivering or deploying software” (Jenkins User Documentation). Our company uses continuous integration (CI), the meaning of which is nicely summarised in Guru99 blog: “In Continuous Integration after a code commit, the software is built and tested immediately. In a large project with many developers, commits are made many times during a day. With each commit code is built and tested. If the test is passed, build is tested for deployment. If deployment is a success, the code is pushed to production. This commit, build, test, and deploy is a continuous process and hence the name continuous integration/deployment” (Guru99.com). Support for integration with various tools, such as version control, Maven, AWS, etc, requires installation of the relative plugins.

Figure 2 below illustrates the general continuous integration process with use of Jenkins.

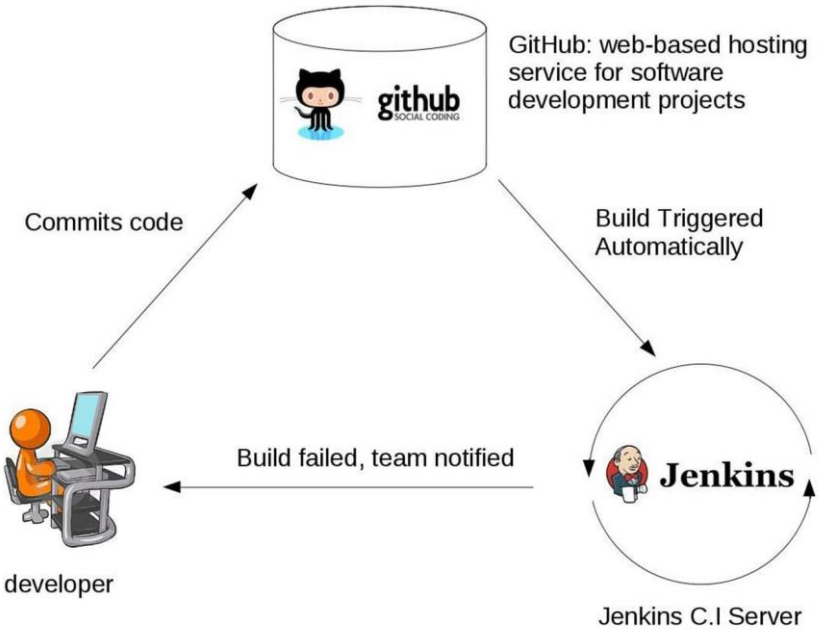

Figure 2. Continuous integration process with Jenkins

In our company, we use it as a main development pipeline for compiling the application, creating/updating the environment, testing the application, promoting to a higher tier environment. Out setup is slightly different than described above: after the code is deployed and tested in the CI environment, it is pushed to staging, which is a production-like environment and is used for much more testing to make sure that the application/service would work exactly as expected when deployed to production. Staging environments usually are almost exact replicas of production (Rouse, 2018). In our case, the difference is that in production we use multi-regional AWS setup and bigger instances, unlike in staging for cost-saving reasons. Each environment has its own Jenkins Pipeline, and the application and system tests are running only in CI.

The Jenkins Pipeline runs the entire development workflow as one code, while also separating the built steps. It comes as a plugin and is a very powerful tool for development of large multi-step projects. According to Andy Pemberton's "Top 10 Best Practices for Jenkins Pipeline", some of the important advice for writing and using Jenkins Pipelines include the following:

- Develop as code
- Organize work in stages
- Use parallel steps
- Use timeouts

These are the principles we are also using in the company. Most importantly, all the jobs must be written in code, not created manually via Jenkins UI. I have not thought about the reasons behind this but Pemberton states that this 1) would enforce good discipline and 2) allows to use a lot of additional capabilities and features, such as multi-branch, pull request detection and organization scanning for GitHub and BitBucket (Pemberton, 2016). The aim of the remaining 3 principles is performance optimisation. Organizing the work in stages allows better control for the process and simplifies debugging (Figure 3). Parallel steps option allows to run several processes simultaneously, speeding up the Pipeline run. In our setup, we use this to build AWS stacks which do not depend on each other, for example the database and the load balancer. As for the timing out, the article author suggests that their main usage is for approving inputs, but we do not use inputs in such way, so we set timeouts for operations that might potentially get stuck and block the Pipeline runs, such as AWS stacks operations or system tests runs.

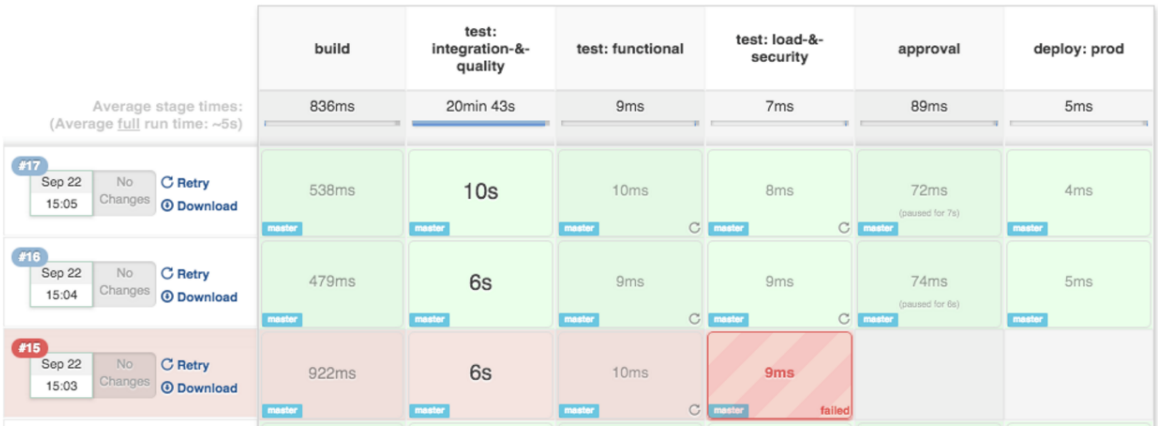

Figure 3. Jenkins Pipeline Stage View plugin example

As well as this, Jenkins can be used for building independent simple applications. We use it for compiling and running numerous separate tools, one of which is the AMI deletion tool that I was writing a job for.

When I first wrote the Jenkins job and the shell script for the AMI deletion, I added `DRY_RUN` as a string parameter the value of which defaulted to `"-r"`, which stands for the dry-run option in `ami_cleaner.py`. My shell script therefore looked like this: `'python3 ami_cleaner.py ${DRY_RUN} -k edited-ami-name -n ${TOP_N}; '`, where `TOP_N` stands for the number of the most recent AMIs to keep.

If `DRY_RUN` is empty, the command would run in a standard mode. If the parameter contained a necessary attribute, it would run in a dry mode. My boss reasonably commented

that although this would work, it was quite error-prone and I should better use a Boolean parameter + condition.

I originally intended to do so but I realized that Jenkins Job DSL (syntax for writing the jobs) does not support conditional statements inside the steps, so the condition would need to be added to the shell script part. This seemed more complex, so I settled for the option above. But after my boss's comment I agreed with him and made the related changes. Previous to this task, I did not know that a Jenkins job can contain a Boolean parameter because I have never encountered one in the scripts I was working with, so there was a small addition to my Jenkins skills as well as the shell scripting skills.

The final version of the shell script looks like this:

```
shell('''
if [ "${DRY_RUN}" = true ]; then
ami_cleaner_command="python3 ami_cleaner.py -r -k edited-ami-name
-n ${TOP_N}";
else
ami_cleaner_command="python3 ami_cleaner.py -k edited-ami-name -n
${TOP_N}";
fi
$ami_cleaner_command
''')
```

The code above first checks the Boolean value, then assigns the corresponding command string to the `ami_cleaner_command` variable, and finally, executes the command. This code is longer and more complex than the one-liner that I first wrote but I agree that it is much better as it prevents the possibility of running the command with some incorrect attribute instead of `-r`.

This week I have slightly broadened my knowledge of Jenkins by learning some of the good practices for maintaining and development. It is important to understand not only what you are doing but also why, and by performing and analysing the tasks, I have developed both my practical and theoretical skills. As for the tasks, the most important result is that now we have a running SEBE environment in AWS and I have knowledge about its structure and will be able to use it, which is important for my next tasks.

3.2 Observation week 2 (September 9 – 13)

Monday 9th – Wednesday 11th September 2019
Task: Audit logging improvement: change old `@Audit` to new `@AuditFlow` in SEBE.

This is one of my team's global tasks for the next quarter. The whole team is going to work on the logging improvements because we are getting ready to the service's migration and we need to know exactly which operations are used, by whom and how heavily. Current audit logging was found insufficient for these purposes and the mechanism is outdated. Old `@Audit` annotations and related logic have to be replaced with newer `@AuditFlow` which is used in the new services. It is an internal library written specifically to cover the company's needs.

SEBE service contains six big applications that depend on each other. We need to add the logging in such a way that it is not duplicated or merged, while covering all the operations that are possible to be performed by the service.

I changed the annotations and related code for four of the applications. My team-mate was working on the other two because they are bigger and contain more complex logic. It was a boring and monotonous work consisting of copying & pasting and changing many small details for each annotated method, adding logging parameters and names. As a result of three days' work, the old `@Audit` was replaced with new `@AuditFlow`.

Key learnings: `@AuditFlow` library; basic structure of SEBE applications; logging good practices.

Thursday 12th – Friday 13th September 2019
Task: Add annotation to the presentation endpoints.

The next step was to annotate all the endpoints in the applications. Currently in SEBE we have annotations on the logic layer methods (in service classes). We need to annotate the endpoints, which are in the presentation layer.

This task was almost the same as the previous one but this time I had to think myself how exactly it was supposed to be done. It was very time-consuming because everything had

to be done from scratch: audit profiles for each application, audit messages for each operation of each endpoint, audit names for every parameter used in every method. It was very hard to avoid mistakes of inattention caused by fatigue. This is where peer reviewing came in especially handy: another person has more chance to spot a small copy & paste error than you after spending hours on the same task. There were some typos and small mistakes in everyone's work and we fixed those together.

Key learnings: SEBE applications presentation layer structure; good practice: group audit messages by endpoint and order alphabetically.

Friday 13th September
Task: Remove the logging from logic layer and some of the endpoints.

After we tested the changes, we realized that now we had too many logs, some of them were duplicated and some unnecessary, so annotations from those need to be removed.

Logic layer services and presentation layer endpoints are interconnected. Endpoint calls the service with the parameters that it gets from the request. Therefore, these logs are being duplicated, since they basically record the same activities on two different layers. This is bad design, it makes searching the logs complicated and produces way too many logs, a big amount of which is unnecessary.

By the end of the week, all logging tasks were finished. Removing the annotations from logic was easier and faster than adding. However, it can get tricky when it comes to removing the duplicates because it is necessary to ensure that the corresponding endpoint is annotated, so I had to double check that every annotated method is covered by an annotated endpoint. Since SEBE is very large, it is not always easy to do because the project is multi-layered and uses many intermediary classes, so it is sometimes difficult to trace the origin of a method call. Thankfully, the IDE (IntelliJ) simplifies the process (project organization -> "find usages" command).

Key learnings: principles of good logging design; more knowledge about SEBE structure; useful IntelliJ feature.
Weekly analysis

The tasks at hand were simple in their nature but terribly monotonous and tiring. They did not require much skill or knowledge, but a lot of attentiveness and perseverance. I have never done this kind of task before, so it was a completely new experience for me and I must say I would have preferred a harder task that involves a lot of thinking and research. When you have to do such scrupulous work for such a long time, mistakes and errors are inevitable because you cannot help but get tired of looking at the same lines that you need to change slightly a couple hundred times. It is a very frustrating process and I felt rather annoyed by the need to do this.

If I could do it differently, I would have kept some other small tasks to switch activities from time to time. Some people might feel better when they have to perform this sort of activities but I learned that for me it is exhausting and leads to losing concentration which causes a lot of errors. I am much more productive if I can switch between different tasks when I get tired. This task may have improved my monotonous work skill but I am not sure that it is possible.

At least, the important piece of work has been done and I have learned a couple of useful things in the process. Before, I did not know much about audit logging and why it should be used, but now I understand the importance of it and the convenience of monitoring the application traffic with audit logging.

Audit logging serves the purposes of monitoring the behaviour of the application, both for 1) understanding what exactly is happening inside, which parts of the application are used, where the requests were sent from, and generally gathering the statistic information; and 2) debugging purposes, because, as rightly noted a software engineer Eugen Paraschiv in his article, “in real world production environment, you usually don’t have the luxury of debugging” (Paraschiv, 2017).

The same article, however, warns against logging too much or logging the wrong things. First of all, the security aspect. User credentials, financial data and other confidential information should never be logged in plain text, as the log files could be accessed by some unsecured system. The author suggests that the best way is not to log any sensitive data at all but in our case the information that we log might be needed for monitoring purposes and the nature of this data is not a major security threat. So, what I did to avoid compromising any private data was use a special converter class provided by the company’s au-

dit logging framework. It basically shortens the specified logged parameters and only records a part of them, so that it would be impossible to compromise any system or account, while it was still letting us have the data we need.

Excessive logging causes problems because, firstly, it becomes much more difficult to browse through and read the logs. Paraschiv confirms my concerns by stating that “too many log messages can also lead to difficulty in reading a log file and identifying the relevant information when a problem does occur”. Secondly, too many logs are thinning the Splunk limits, and when the limits are exceeded, the new logs are not let in until the old ones are cleared. And finally, it costs extra money to the company.

The standard architecture of most Java applications is a layered pattern, which usually consists of horizontal layers, each layer performing a specific role within the application (e.g., presentation logic or business logic) (Richards, 2015). There is no definite structure, so the number and types of layers may vary, and our applications usually use 3 layers: presentation (handling the communication and request/response logic), logic (business logic of the application) and persistence (accessing the database). The layers are vertically connected, so the endpoint on the presentation layer is the first to receive the request and it eventually directs it to logic with same or different parameters. This is important not only for following the application design conventions but also when it comes to audit logging.

We removed the audit logging annotations from the logic layer of our applications so that the logged data was not duplicated and only the necessary information was recorded. Moreover, annotating the endpoints makes much more sense than the inner services because we are most interested in the first interaction with our system. This way we can get more relevant information, as we record request, response and related information, not only the methods and parameters that were invoked, with any security tokens or password information being obfuscated. To sum up, the annotations changes that we made correspond the best practices of logging.

During this week, I greatly improved my understanding of the audit logging and found out how and why is should be used in web development, as well as some “dos” and “don’ts” of the process. I familiarized myself with our company Audit logging framework, which I will be using more in the future. Another useful learning was getting more familiar with the SEBE service. It will take a lot more time to become even intermediately acquainted with

the whole service but knowing more about some bits of it is definitely a benefit as most of my tasks for the nearest future involve working with SEBE applications.

3.3 Observation week 3 (September 16 – 20)

Monday 16th – Wednesday 18th September 2019
Task: Audit logging improvements: test if the new logging works and the old is dropped.

Now that both me and my team-mate have presumably finished with annotating the SEBE applications, we need to test the changes and carefully check if all the logs work.

There was a complication: even though we have configured the SEBE AWS environment previously, the tests do not run there because of some database connectivity problem. My team-mate is working on this issue. No test runs means no logs in the new development environment, therefore meanwhile I will have to check the logging manually by making a ssh connection to the current (old) testing environment and looking at the logs from command line.

My boss showed me how to do so. I needed to know the IP-address and the password of the GlassFish node and the path to the directory where the logs are stored. I could see that the log files with new names were created for most applications but now all of them. We had no way to check if the logs for the other applications were not there just because there were no calls yet since we merged the changes, or there was some error. This could be easily checked if we were able to run the tests in the new AWS environment but unfortunately that was not the case, so there was nothing more that could be done for now.

Task: Check the logs for any errors.

Then I needed to check the logs themselves and see if there were any problems inside. For that I decided to try using the WinSCP tool instead of command line because it is easier for me to look through and search the text files than the lines in the console window. I successfully connected to the node via WinSCP. I have not tried this at work but I used the tool at school for connecting to my remote folder on the school server and vaguely remembered how to use it.

Looking through the logs, I found some warnings similar to this one: `WARN AuditLog-Validator - - The key message not found from the profile class [edited].ConfigurationProfile.`

I informed the team of the finding and it turned out that the reason for this warning was that some Spring context setting was missing in the application code, so the logginh annotations were not fully operational. Basically, Spring could not resolve the audit profile that contained the information that was needed for recording the logs. It happened because the audit logging component is a separate library which has to be not only imported to `pom.xml` files (which was done) but also added to Spring context configuration. I had no idea what that was, so we sat together with my team-mate and looked through the applications, as she showed me how this should be configured. Then I followed the example and fixed the issue for other applications as well.

The task was not difficult and allowed me to learn many useful practical skills. I needed some assistance because this kind of work was never required from me before, so I had to be taught, which is absolutely fine in this case, since I am a junior developer and my team is always happy to show me new useful things.

Key learnings: how to ssh to our GlassFish nodes; using WinSCP for the same purpose; Spring context configuration.

Thursday 19th – Friday 20th September
Task: Splunk logs improvement:

1. Make sure that the old logs disappeared from Splunk and the new ones appeared;
2. Add settings for logs dropping and re-routing;
3. Test the changes.

The first step was simply to check if the logging in the applications was finally configured in the right way. It was fine after we fixed the Spring configuration last time, the new logs were appearing in Splunk as expected.

The second step is about making major changes to the current setup. First of all, I need to change the index to which the new logs are sent and which is used for searching them. The current index is for the whole SEBE and also contains application logs, server logs, access logs, etc. We want to move all the audit logs to the separate index `sebe_audit`.

And moreover, the audit logs need to be split into two parts: some of them stay and some either go to another separate index or are completely dropped. I need to add both functionalities for the same group of logs.

For a start, I have looked into other teams' configurations for the similar kinds of tasks and found out that I will need to make changes to `inputs.conf`, `indexes.conf` and `props.conf` files in the Splunk configurations, and add and configure `transforms.conf` for defining the rerouting schemas. It seemed fairly straightforward, so I added the necessary configurations after consulting with the Splunk documentation.

I struggled a little with the regular expressions needed to match the logs that are supposed to be rerouted or dropped. After some time of trying and testing I figured out the expressions to be used in `TRANSFORMS`. It was not hard on its own but I had little experience of dealing with regex syntax which is not always very easy to understand.

Then I faced another difficulty: the new indexes and source types just would not appear in Splunk. Not that our logs were configured wrong, but the source types themselves were not created, which meant some other problem than regex or configuration error. We looked into that together with the boss until we realised that we were trying to use an independent environment, for which Splunk forwarder settings are configured in the application project, not the common company's Splunk project.

Even though I was almost not familiar with Splunk configuration, the task did not seem especially difficult for me. I had good examples in front of me and needed to adjust them to our needs. That did not mean simple copying & pasting but it simplified the process.

The third step could not be done yet because my team-mate was still working on setting up the tests that we need to run in the environment, so it is postponed until that it done.

Key learnings: Splunk configuration in general; inputs, props and transforms; project-specific knowledge: AWS SEBE Splunk settings are defined in another place than the old environment SEBE Splunk settings; more experience with regular expressions.

Weekly analysis

During this week my skills developed a lot. I have used several new tools and software, learned some nuances of Java configuration, practiced regular expressions and deepened my knowledge of our internal services and tools. I feel this was a productive week and I am satisfied with my performance and learning.

First of all, learning to use new tools in daily work is always exciting and beneficial as it allows to perform a broader range of tasks in the future, so even the simple acquaintance with something new is a step to becoming a better developer. Now not only can I check the audit logs for SEBE, but also have full access to our inner network nodes which I did not have before, so I could not do some quick checks that I needed and had to ask my team-mates to do so for me. Learning this gave me more independence in my work.

Moreover, I started to be familiar with Splunk, which is one of the important tools that we use, so it will be easier for me to continue with the Splunk tasks in the future.

Splunk is a software for searching, monitoring, storing and analysing any machine-generated data, via a Web-style interface (Harris, 2010). It can be used for multiple purposes but in our company we use it for logging. In addition, it is possible to perform various operations and manipulations with the log data, and to do so, I got acquainted with some of the administrative functions of Splunk.

Even though I do not yet fully understand the architecture of Splunk, I now know some basics of logs configuration and how to use `inputs.conf`, `indexes.conf`, `props.conf` and `transforms.conf`.

According to Splunk documentation, `inputs.conf` is a file to configure settings for inputs, distributed inputs such as forwarders, and file system monitoring (Splunk Enterprise Admin Manual). As I learned in practice, it is used for defining the log sources, which are locations of the log files, and linking them together with the source type and the index name:

```
[monitor:///[edited]/[edited]/[edited]/*/*/logs/*audit*.log]
disabled = false
blacklist = (\.\d+\.log|jvm.log|server.log)
index = sebe_audit
sourcetype = sebews_audit
```

`indexes.conf` is a file that stores settings for the indexes, such as index name, the path to its location, time of keeping the logs in the database. New indexes are created by adding an entry to this file, such as:

```
[sebe_audit]
repFactor = auto
coldPath = $SPLUNK_DB/sebe_audit/colddb
homePath = $SPLUNK_DB/sebe_audit/db
thawedPath = $SPLUNK_DB/sebe_audit/thaweddb
frozenTimePeriodInSecs = 63244800
#reduce after 180 days
enableTsidxReduction = true
timePeriodInSecBeforeTsidxReduction = 15552000
```

`props.conf` contains properties settings, i.e. setting/value pairs for configuring Splunk software's processing properties. Here the source type is connected to the `TRANSFORMS` settings, for example:

```
[sebews_audit]
SHOULD_LINEMERGE = FALSE
LINE_BREAKER = ([\r\n]+)\d{4}\-\d{2}\-\d{2}
TRUNCATE = 250000
TRANSFORMS-drop = dropLicensingAuditLogs, dropPackagingAuditLogs,
dropTicketingAuditLogs, dropReportingAuditLogs, dropPushAuditLogs
#TRANSFORMS-spi = LicensingAuditLogsSPI, PackagingAuditLogsSPI,
TicketingAuditLogsSPI
```

`transforms.conf` is used for adding various transformation rules, such as, in our case, dropping the logs or rerouting them to another index, as well as field extractions. This rule drops all the logs matching the defined regex from Splunk completely:

```
[dropLicensingAuditLogs]
REGEX="appname":"licensing"
DEST_KEY=queue
FORMAT=nullQueue
```

My skills developed from zero knowledge about Splunk administration to such a level that I understand how to use some of the configuration files and how things work in the very general terms. I have learned to configure such administrative activities as adding new log sources, creating indexes, setting up logs' redirection, which were among my goals. It was quite a steep learning curve, and what could have been done differently is probably I would have read more documentation first before starting with trial and error and learning by doing. This is one of my qualities that could be both an advantage and a drawback: I tend to rush into practice, sometimes lacking enough theoretical skills, and return to the manuals and tutorials after my own attempts do not fully work out. On the other hand, reading this kind of the documentation is only helpful when you already have a general idea about what you need to do, which in this case was not possible for me until I started to try things out.

Another valuable skill development was learning about the Spring configuration. Although I am fairly good at Java, I do not know much about the administrative settings that are important for the application compilation, deployment and proper work. So, I familiarized myself deeper with Spring framework.

By reading the Spring documentation, I learned that annotations in Spring are actually a way of dependency injection (Tutorialspoint: Spring - Annotation Based Configuration). So, by using `@AuditFlow` and other related annotations I was in fact passing the F-Secure Audit library to the SEBE applications. What I did not know was that, according to Spring tutorial, annotation wiring is not turned on in the Spring container by default and needs to be enabled in Spring configuration file, as shown in Figure 4 below.

```
<?xml version = "1.0" encoding = "UTF-8"?>

<beans xmlns = "http://www.springframework.org/schema/beans"
   xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
   xmlns:context = "http://www.springframework.org/schema/context"
   xsi:schemaLocation = "http://www.springframework.org/schema/beans
   http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
   http://www.springframework.org/schema/context
   http://www.springframework.org/schema/context/spring-context-3.0.xsd">

   <context:annotation-config/>
   <!-- bean definitions go here -->

</beans>
```

Figure 4. Spring configuration: annotation context

By setting and configuring `<context:annotation-config/>` and defining the beans to be used, the application is instructed to recognize and use the annotations. However, there was another property that needed to be configured before the whole thing worked, which was AOP context.

According to the same tutorial, Aspect-oriented programming, or AOP, framework is one of the key components of Spring. In AOP the program logic is broken down into distinct parts called “concerns”. The functions that span multiple points of an application are called cross-cutting concerns and these cross-cutting concerns are conceptually separate from the application's business logic. (Tutorialspoint: AOP with Spring Framework). While in Object-oriented programming (OOP) the modular unit is a class, in AOP it is an “aspect”, which is a module having a set of APIs providing cross-cutting requirements. The common examples of aspects include auditing, logging, validation, caching, etc.

Therefore, the AOP settings also had to be added to the Spring configuration file: `xmlns:aop="http://www.springframework.org/schema/aop"` below `xmlns:context` configuration, and these lines `"http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-4.0.xsd http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-4.0.xsd"` to `xsi:schemaLocation` definition.

This kind of new knowledges helps me to grow professionally and become less of a “programmer” and more of a “developer”.

3.4 Observation week 4 (September 23 – 27)

Monday 23rd September 2019
Task: Add first use date to RabbitMQ messages.

The messages returned by RabbitMQ should contain the first use date of the subscription. The database already contains this date and the classes related to persistence layer have the related field. What needs to be done is to include this date into the message object. To do so, several actions are needed:

1. Make changes to the `Subscription` entity object definition (add field `firstUse`)

2. Make changes to the `MessageData` object definition (add field `firstUse`)
3. Change the method creating the message (include `firstUse` parameter)
4. Make sure that the date is sent in the proper format, which should be ISO 8601; convert if necessary
5. Test (write unit tests and system tests)

First of all, I had to look into the application structure to figure out the logic of the process. I was shown the starting point by my boss and then spent a quality time understanding how this part of the application works. Then I wrote the necessary pieces of code, covering steps 1-3.

This task was quite easy for me because it was some very basic Java programming. Looking at the existing code took much more time than writing the new code.

Key learnings: SEBE messaging logic; Java coding convention for the class elements.

Tuesday 24th – Thursday 26th September 2019
Task: First use date continues.

Step 4 is only possible to check via tests, so I started writing the tests for all the code that I have written. However, the testing part turned out to be problematic. While testing the new functionality was quite straightforward with the unit tests, the date format caused a problem. It was not possible to test it with unit tests or integration tests. So, I had to look into the Python system tests, which are completely separate from all the other tests, and find out which ones are relevant (and are receiving the actual, not mocked, message). As the application I am working on is very large, that took quite a lot of time.

The task is matching my skills, unit tests were completed quickly and easily. Researching Python tests took time due to the volume and complexity of the application.

Key learnings: the way how system tests are done for this application; Nose tests framework; now I can modify the existing tests and write my own ones in a consistent manner.

Friday 27th September 2019
Task: First use date continues.

In order to run the system tests, the application has to be compiled and the `.war` file uploaded to our test environment. But the compilation failed for me because some of the integration tests were failing. It turned out that those tests run against the database which I did not have installed and configured locally. So, I had to start setting up the local environment for SEBE, which was among my quarter tasks anyway.

Key learnings: SEBE project structure and integration tests logic; requirements for local setup.

Weekly analysis

A relatively simple task turned out to be unpredictably time-consuming because of all the environment-related complications. That is a usual problem when working with huge applications in complex environments. There is a lot of legacy code, as well, and I do not have all the necessary frameworks configured yet, so it takes time for me to figure out what to do when I need to work on some new small task.

My learnings this week were mostly contextual, that is to say, I was studying the code I had to work with. It is a crucial part of development process when you are not creating new content from scratch but working with some pre-written by somebody else or legacy code. SEBE is one of the most important services that our team maintains, so my learning about some small part of its structure is valuable for the future tasks, as I now know more about the service and will probably spend less time figuring out how things work elsewhere in the code, since most of the applications in the project share more or less similar logic.

Studying the existing code also allowed me to realize what I had to do first of all for setting up a local environment. I only needed one specific application to compile this time, so with the help of the integration tests failures I learned which database tables I need to have. Otherwise I would have had to spend a lot more time setting up and populating all the databases for all the applications in the project. Also, finding the SQL scripts that are used for the application deployment on the server facilitated the process because some of the administrative scripts (database creation, tables and keys configuration, roles and permissions settings, etc) could be run manually and I did not have to figure out how to do those

operations myself. Populating the tables with those scripts turned out to be a problem because of their length and complexity and dependencies but that problem was solved with the help of the team by using database dumps.

Moreover, studying someone else's code allows to see some good (or bad) practices by seeing and remembering something that can be later used (or avoided) in your own work, or learning about specific algorithms of useful methods. It is also beneficial to practice understanding someone else's code because that way you broaden your perception of the programming logic, comparing what is written to how you have/would have done the same task and analysing both your own and the other person's coding habits and learning from it either way.

I have also learned some basics about RabbitMQ. It is a messaging software that can be integrated into the application for automatic sending of the messages with configurable content. RabbitMQ is open-source and supports all the most popular programming languages (rabbitmq.com). I am yet to learn more about the details and mechanisms of its work but as for today, I have an understanding of its general principles and configuration using the Java client.

RabbitMQ is imported to the Java application from central Maven repository by adding a dependency to the project's `pom.xml` file. Then the message sender and receiver classes are created in the application, and the connection with the messaging server is configured. Sender and receiver can be in the same application or in completely independent ones. I have not yet needed to work with RabbitMQ itself, so I did not dive deep into its structure and principles, however now I have a starting point for further studying of this software.

Figure 5 and Figure 6 below show examples of basic sender and receiver classes. `ConnectionFactory` is setting the connection with the RabbitMQ server on the local machine at `localhost`, and `QUEUE_NAME` is the name of the destination queue, which is where the messages are stored in RabbitMQ.

```java
import com.rabbitmq.client.Channel;
import com.rabbitmq.client.Connection;
import com.rabbitmq.client.ConnectionFactory;

import java.nio.charset.StandardCharsets;

public class Send {

    private final static String QUEUE_NAME = "hello";

    public static void main(String[] argv) throws Exception {
        ConnectionFactory factory = new ConnectionFactory();
        factory.setHost("localhost");
        try (Connection connection = factory.newConnection();
             Channel channel = connection.createChannel()) {
            channel.queueDeclare(QUEUE_NAME, false, false, false, null);
            String message = "Hello World!";
            channel.basicPublish("", QUEUE_NAME, null, message.getBytes(StandardCharsets.UTF_8));
            System.out.println(" [x] Sent '" + message + "'");
        }
    }
}
```

Figure 5. RabbitMQ sender class example (rabbitmq-tutorials)

```java
import com.rabbitmq.client.Channel;
import com.rabbitmq.client.Connection;
import com.rabbitmq.client.ConnectionFactory;
import com.rabbitmq.client.DeliverCallback;

public class Recv {

    private final static String QUEUE_NAME = "hello";

    public static void main(String[] argv) throws Exception {
        ConnectionFactory factory = new ConnectionFactory();
        factory.setHost("localhost");
        Connection connection = factory.newConnection();
        Channel channel = connection.createChannel();

        channel.queueDeclare(QUEUE_NAME, false, false, false, null);
        System.out.println(" [*] Waiting for messages. To exit press CTRL+C");

        DeliverCallback deliverCallback = (consumerTag, delivery) -> {
            String message = new String(delivery.getBody(), "UTF-8");
            System.out.println(" [x] Received '" + message + "'");
        };
        channel.basicConsume(QUEUE_NAME, true, deliverCallback, consumerTag -> { });
    }
}
```

Figure 6. RabbitMQ receiver class example (rabbitmq-tutorials)

I have also acquired some less abstract knowledge, namely, Java conventions and practices related to the class elements order. I needed to order static and non-static variables and class methods. The conventions were clear on the former but somewhat ambiguous about the latter, so I analysed what I have read in different sources and after discussion with the team, structured our code like this:

1. Class variables (static): public, package-level, private
2. Instance variables (non-static): public, package-level, private
3. Constructors: public, private
4. Methods: public static, public, package-level static, package level, private static, private

According to the convention, class variables should come first, followed by the instance variables, constructors, and then the methods. Elements of the same level are then ordered by scope/accessibility, descending from public to private. The methods, however, do not have a strict hierarchy and “should” rather be ordered and grouped not by their scope or accessibility, but by functionality, in order to make the code more readable and understandable (Oracle, 1995-1999). In this particular case the class I was working with contained too many too different methods, and grouping them by functionality would not help to achieve the convention’s goal. The best readability would be reached by ordering the methods according to their scope, so I decided to order the methods in the similar manner as the other elements of the class.

Following the coding conventions is important because standardization and unification allow to make the code easier to read and understand for all the other developers, and consistency in general has a positive impact on the code quality, as it makes the code easier to review and debug. Nevertheless, most of them are only the guidelines which are designed to make life easier for the developer. If in some particular cases the aim of the conventions is better achieved by deviating from the standards, then, in my opinion, it is a reasonable thing to do.

To sum up, it can be said that this week my learnings were not so vertical but mostly horizontal. I learned more about the service I was working on, got acquainted with a new software (RabbitMQ) and its general principles and how it is used in the current project; and deepened my Java skills by researching some of the coding conventions. While these findings may not be strikingly revolutionary for my development, this knowledge expands my professional horizons, which is important for a developer, and is certainly crucial for successful completion of my current and subsequent work duties. Moreover, I have obtained enough information to be proceeding with the related task of setting up the local environment, which I am going to start next week.

3.5 Observation week 5 (September 30 – October 4)

Monday 30th September – Wednesday 3rd October 2019

Task: Set up SEBE locally and run the system tests against the GlassFish environment.

Steps:

1. Creating a virtual environment, installing the requirements for old Python version (2.7)
2. Setting up and connecting a local MySQL database with old MySQL version (5.7)
3. Migrating the necessary test data for the application compilation

The whole process was one problem sprouting after resolving the other. First of all, dealing with old Python dependencies proved to be very painful. The instructions from our old documentation did not work, so I had to experiment with installing different libraries of different versions, or look for substitute libraries that would be compatible with this and that, myself. After spending a lot of time, I figured out the set of libraries and versions that would work together.

MySQL setup also caused problems because first I tried to use the latest version but had to install the exact same version as on the server, otherwise not everything was compatible. And finally, the data migration was unexpectedly difficult. The application contains scripts for the database setup and population on the server but they are so complex and long that it was impossible to run them manually. After several failed attempts and discussion with the team, we came up with using the dump scripts from one of the working setups. So, my team-mate sent me the needed scripts and I was finally able to create and populate the database, followed by compiling the application and deploying it to GlassFish.

This task was rather difficult for me to complete because I mostly have experience with coding, not administration and frameworks setup. It is always very time-consuming to configure an environment, even for an experienced developer, especially if the software in use is not of the latest versions.

Key learnings: how to use local war in cloud GlassFish for local testing without merging the changes to server; how to setup MySQL and use Workbench.

Thurthday3rd – Friday 4th October 2019

Task: Fix the date format; test.

Then I was finally able to modify and run the tests, and it turned out that the date was not converted to the proper format, so I had to figure out how to do it. After trying several options, I used the `SimpleDateFormat` class and some already existing in our application methods which are used in other places for performing a similar task.

After that I had to spend a lot of time on the system tests again. There were two issues: the value of the seconds in the date in the database entry and in the message did not always match; there was also a 2-hour difference between the dates used in the test, even though the application makes sure the date is converted to UTC time zone.

After some time of debugging and monitoring I discussed my findings with the team and we came to conclusion that the seconds mismatch is irrelevant and should be ignored by checking the range of seconds instead of the exact value. The latter matter was caused by different settings of local time on my machine and on the RabbitMQ server. For fixing that I first tried to convert the date used in the test to the UTC time zone but it turned out that Python 2, in which those tests are written, does not have a simple option of recognizing time zones without importing any additional third-party modules that we did not want to have to install (Stack Overflow. How to convert local time string to UTC?). So, as a workaround for the testing purposes, I added the 2-hour difference to the time manually.

The task was, again, exactly of my skill level but there were many tricky details that complicated the process and took a lot of time to identify and solve.

Key learnings: the difference between time formats, time zones and how to use `SimpleDateFormat`; some particularities of Python 2; technical nuances such as time zone difference or a message sending gap.

Weekly analysis

This week felt more productive than the previous one because, despite spending too much time researching and configuring things, my work has had fruitful results and not only have I finally finished the task which was started last week (adding the first use date to messages and testing it), but also completed another important task (setting up SEBE locally) in process.

This week many things needed clarification, both “internal”, meaning SEBE and related frameworks, and “external”, meaning everything else related to software configuration, coding, testing. The “internal” part (which mostly included questions like how to connect this to that, or what do I need for this or that to work, or where can I get credentials for these operations) was clarified by my team, while for the other questions I have conducted my own research.

By solving the problems of Python 2 dependencies I have learned to adjust versions of different libraries in such a way that they contain the necessary modules and are not in conflict with each other. For our system tests for this project there is a script to download and install the necessary libraries. Some of the versions were specified, some were not. This caused clashes between the newer and older versions of different interdependent libraries, as a result of which the requirements could not be installed. I had to look through a lot of documentation in order to find out which versions would work together. It was a good way to practice working with documentation and finding the necessary information, and I feel that this helped me improve my analytical skills, and gave me practical knowledge, such as how to use https://pypi.org/.

I have also acquired some practical skills related to MySQL administration. MySQL is a relational database management system developed by Oracle. After having spent 1.5 years working with lightweight and flexible non-relational DynamoDB, switching back to the standard relational setup was slightly difficult and rather unpleasant for me. Moreover, at school in the Data Management and Databases course we were dealing with Microsoft SQL Server, which is different to MySQL. There is slight difference in the SQL statements syntax, as well as a completely different administrative interface. Although, I would probably not have remembered much about the SQL Server, anyway.

The Workbench configurations seem quite confusing when you start and quite straightforward once you have finished. For example, setting up the connection (Figure 7) and setting permissions (Figure 8).

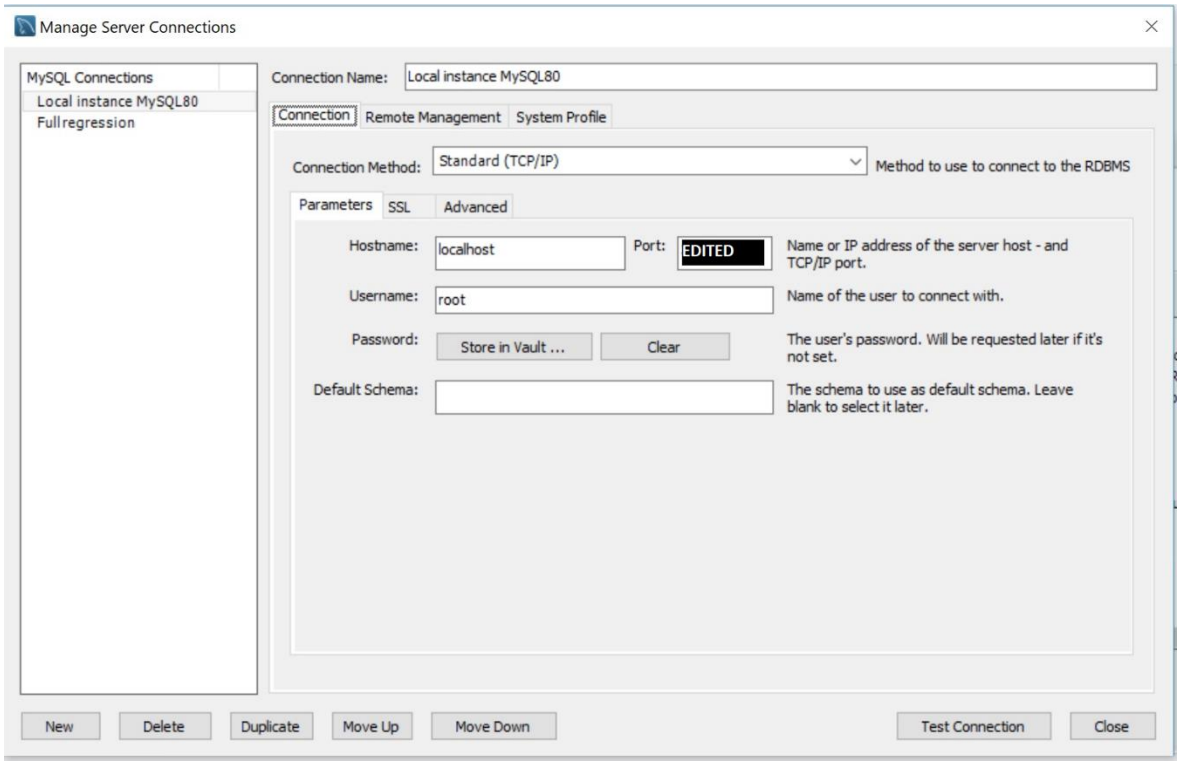

Figure 7. Connection settings in MySQL Workbench

To connect a local database instance to the application, the hostname and the port have to be the same as defined in the application. In our case, Hibernate framework is used in the application.

Hibernate is a tool for Java that provides a framework for mapping an object-oriented domain model to a relational database (Wikipedia). The configuration is set in the `hibernate.cfg.xml` file which should be placed in the application root. The settings have to include at least the following: the driver class, the URL and the port number, and user and the password (Tutorialspoint. Hibernate - Configuration).

```
<!— Database connection settings —>
<property name="connection.driver_class">com.mysql.jdbc.Driver</property>
<property name="connection.url">jdbc:mysql://localhost:[edited: port number]/licensing</property>
<property name="connection.username">licensing</property>
<property name="connection.password">[edited: password]</property>
<property name="hibernate.format_sql">true</property>
```

As for permissions, I have set them by running a script that I found in the code but I learned that it is also possible to do via Administration tab in Workbench (Figure 8).

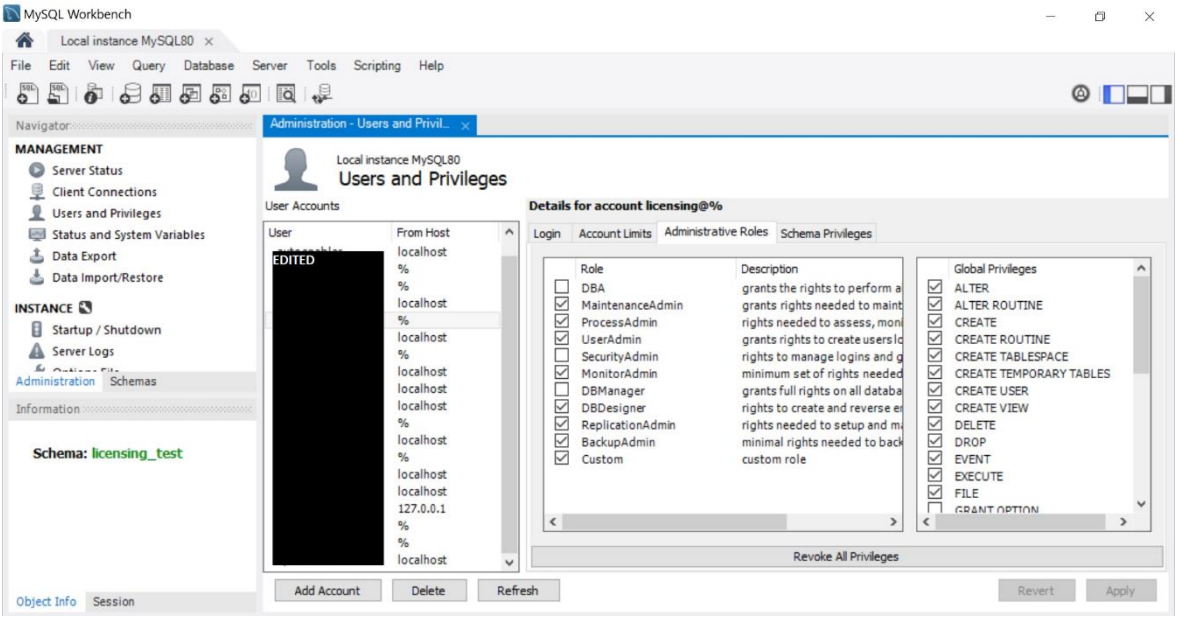

Figure 8. Users and Privileges settings in MySQL Workbench

Other learnings included such practical skills as deploying and testing the locally compiled `.war` file to the testing application server, running the system tests from local machine against it, configuring the remote debugger for IntelliJ which allowed to see exactly what was going on inside the application on the server. All of these skills are very useful for everyday development work and I will be utilizing them a lot in the future while performing other tasks.

I also discovered that, unlike Python 3, which I am used to be working with, Python 2 lacks the feature of time zone support, which forced me to modify the time manually by increasing it by two hours. We try to avoid adding new external libraries to old and massive projects in order not to break any dependency in other places, so this time I had to use a workaround which was not exactly the best way to solve this problem. I kept it in mind, but given the circumstances I describe above, I believe this was the optimal solution, especially considering the fact that it was not the application code, but the test.

I have also improved my Java skills by learning to use `SimpleDateFormat` class which is used for the purposes of parsing and formatting Java Date objects (Oracle, Class SimpleDateFormat). I find dealing with dates especially tricky in Java, and this somewhat

more convenient way of working with them makes it easier, or at least allows to achieve the goal. This is how I ended up using it:

```
SimpleDateFormat iso8601Format = new SimpleDateFormat(DateFor-
mats.DATETIME_FORMAT);

iso8601Format.setTimeZone(TimeZone.getTimeZone("UTC"));

String formatDate(Date date) {
        if (date != null) {
            return iso8601Format.format(date);
        } else { return null; }
    }

Subscription subscription = new Subscription(data.getSubscrip-
tionId(), formatDate(data.getFirstUse()));
```

Java 8 introduced the new Date Time API to replace the old and clumsy `java.util.Date`, `java.util.Calendar` and `java.util.Timezone`. The new simplified API is `java.time.*`, which is easy to use, flexible and convenient. It also solves the issues of the old API, such as poor inconsistent design and difficult time zone handling, and has other additional features and value types that make it very appealing (Oracle, Package java.time). It would be great and very convenient to use this for the format adjustments because it is modern and flexible, but unfortunately, it only works with newer versions of Java starting from jdk1.8, while SEBE is using an older jdk1.6. The necessity to develop and maintain the software using old and outdated frameworks is one of the most frustrating things I have encountered in my professional experience.

Summing it up, this week was full of new skills and knowledge for me. I have learned to use a new software (MySQL Workbench) and familiarized myself with GlassFish server and further deepened my understanding of the SEBE service. Moreover, I have obtained some new coding skills and learned about good practices in both Java and Python, as well as restrictions of their old versions and benefits of the newer ones. I have learned about the new framework (Hibernate) and the basics of its setup, so I will be able to further study it in the future. I have fully set up the local SEBE environment and tested and fixed the first use date format, therefore completing two of my tasks.

3.6 Observation week 6 (October 7 – 11)

Monday 6th October 2015
Task: Find out what caused the test failure and fix it.

It turned out that on the weekend, when my recent changes to both the application and the system tests were deployed to the server, the new system tests failed. Some of my recent changes must have provoked that.

First of all, I re-ran the tests locally once more to make sure that they pass, which they did. It could only mean that the error is server-specific. I looked into the log output and immediately found out the reason of failure: it was the 2-hour time difference which caused the assertion error. It turned out that time zones used locally and on Jenkins do not match. Therefore, I removed the code that modified the time. Now this particular test will be failing locally but passing on Jenkins, which is what matters.

Key learnings: time zone difference on local machine and on Jenkins.

Task: Change instance type of AWS EC2 from t3 to t3a.

Most of our applications run on AWS cloud instances (EC2), and AWS offer multiple instance types. We are currently using t3 instances. Production asked us to switch to a newer instance type because it is more cost-efficient.

This task is quick and easy and requires searching the project code and changing all the "t3" values to "t3a" and then deploying the application.

Key learnings: there is a more cost-efficient AWS EC2 option that we can use.

Tuesday 7th – Friday 10th October 2019
Task: Splunk optimisation: convert log event to JSON and extract fields to be able to search by them.

My team-mate fixed the DB issue in AWS and now we can run the SEBE tests to generate enough log data to test Splunk changes.

The setting `KV_MODE=json`, which is used for automatic field extraction from JSON documents, did not work for us because our logs are not in pure JSON format as they contain some additional lines before the JSON part. We need to get the JSON part manually and then extract the fields from it. After spending a quality time researching different possible methods to do so, I found the solution that worked for us (Splunk Answers).

Running the system tests to check if it works takes a long time because there is a very big number of tests and some of the logs that I need are produced only in the end of the tests run, so every time I need to test new changes, I have to wait quite a bit. In the end, it appeared that the field extraction still did not work.

At this point my Splunk skills were already not that much of a total beginner, since I have already done some similar configuration using `transforms.conf`. However, obviously my general knowledge of Splunk is still very limited, so I had to spend time to look for different options of solving the problem and figure out which one would work for us.

Key learnings: Splunk field extraction options and restrictions.

Task: Find out why new field extraction settings are not working.

I spend a lot of time debugging this and came to conclusion that the settings were correct but still something was wrong or missing.

We had to ask for advice in one of the company's MS Teams channels. It turned out that the configurations were put in the wrong place. We had them on indexer, which means the extractions would happen during the indexing time. But the extractions need to happen during the search time. This was not clear enough from the Splunk documentation. The solution was to put the configuration settings to the correct directory of the splunk-apps project ("search" rather than "indexer"). So, what I had to do was create new `transforms.conf` and `props.conf` files in another directory and move the code there. After moving the settings, it worked as expected.

This was completely new for me but after the colleague's clarifications both me and my boss understood how things were supposed to work and the solution was very easy.

Key learnings: Splunk structure, difference between forwarder, search, indexer; field extraction has to happen during search time, while index-related transformations during the indexing time.

Weekly analysis

This week I deepened my knowledge about Splunk a lot. I have already learned the basics of `TRANSFORMS` operations before but now I understand how they actually work. According to the documentation, Splunk search operation has several stages, each of them allows to perform different operations: indexers process the data, search distributes the search requests to the indexes, and forwarders send the data from remote server to the indexer (Splunk Documentation). The structure can be seen in Figure 9.

Obviously, indexing (meaning also dropping or rerouting the logs) happens during indexing time and therefore must be configured on indexer. While field extractions happen during the search time and have to be set up on the search head. I did not take this into account when I started writing the configurations and it was not clear for me from the `TRANSFORMS` documentation until I educated myself on the Splunk architecture.

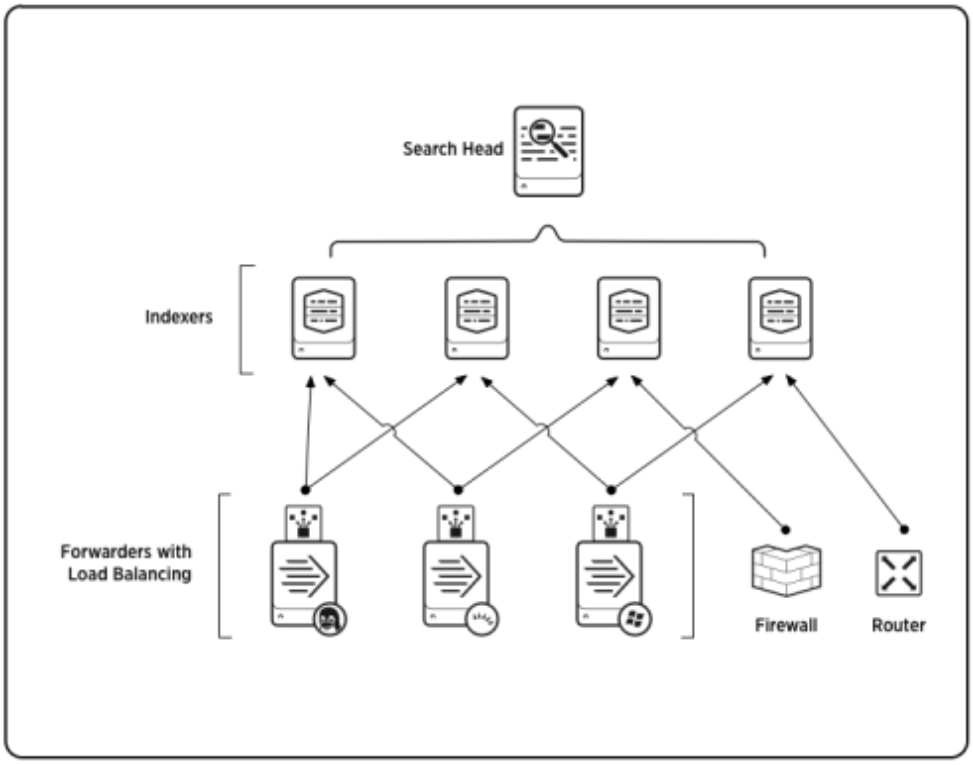

Figure 9. Splunk components structure

I have also learned what is field extraction and why/when it should be used. Basically, that means that Splunk parses the log events, finds the key-value pairs and adds the keys as new fields to its search options. For example, if the log event contains `"operatorId":"666"`, then `"operatorId"` can be extracted as a field and then directly used in the search command to list all log events that contain a certain value for this key.

This can be used if the automatic extraction does not work for any reason. One of such reasons might be mixed log event format. Another case would be if there are many key-value pairs in the log event but only few of them are needed. Then for better performance it is advised to extract the needed values manually (Splunk Answers).

In our case, we had to extract the JSON parts of the logs manually because of the mixed log event format. First, get the JSON part from the log event. Second, extract the fields in key-value format, which would have happened automatically with the `KV_mode=json` setting if our logs did not contain additional text lines.

```
# transforms.conf file
[extractJSON]
# Get the json payload from the logs based on { bracket
REGEX=(?P<json1>{.+)

[extractJSON-KV]
# Manually extract JSON key-value
REGEX=\"(\w+)\":[\s]*\"([^\,\}\"]+)
FORMAT=$1::$2
MV_ADD=true

# props.conf file
[sebews_audit]
REPORT-json = extractJSON, extractJSON-KV
```

I have deepened my knowledge of Splunk and progressed with my goals of familiarizing myself with general Splunk knowledge and learning to perform the administrative operations. Furthermore, the task of Splunk optimisation progressed a lot.

My understanding of AWS also developed as I researched the subject of EC2 types more. Amazon offers many options that vary in in size, CPU, memory, bandwidth, which affects the cost. They are designed to be suitable for different purposes. An AWS expert Aymen El Amri writes that variations of types T and M are the best choice for general mainstream applications because they offer the best balance of computing, network and memory capabilities. C-instances are best for computing, since they offer the highest performing processors. There are also types most suitable for in-memory databases (X1, R4, R3); accelerated computing for high-performance databases, large-scale machine learning and other high-workload server-side activities (P2, G3 & F1); storing and processing data with low-cost storage (I3, D1) (Amri, 2017).

For our instances, we used to use T4 and T2 in production and CI/staging, later switched to T3, and now will be using the new T3a instances which, according to AWS release, offer the same or better characteristics, while allowing up to 10% cost reduction (AWS, 2019). Now I understand the functional difference between the EC2 instance types and have an overall idea of what you need to pay attention to when choosing the right EC2.

Also, communication with another professional was a positive experience. Sometimes it makes more sense to ask for assistance than to try and figure everything out yourself. I am still learning in which circumstances which behaviour is beneficial, so this was a valuable lesson. My conclusion out of this situation is that asking for help is never a bad option. If you are stuck and tried everything you could, ask your team. If the team has no answer, ask outside the team. The chances that somebody in a large development company faced the similar problem or knows the solution, are very high.

3.7 Observation week 7 (October 21 – 25)

Monday 21st October 2019

Task: Splunk optimisation continues.

1. The log redirection regular expression can be improved
2. Find out why some logs are not redirected
3. Drop logs by operator, if possible

My boss mentioned that the regex that I wrote for log redirection in `transforms.conf` is correct but not optimal. He suggested changing it and I agreed that my way was unnecessarily overcomplicated.

Key learnings: regex good practices.

Task: Test Splunk and make sure the redirection works as expected.

Since the task is almost done, it is necessary to double check that everything runs as expected before deploying the changes to production. Testing suddenly showed that some of the logs were not redirected.

Tuesday 22nd – Thursday 24th October 2019
Task: Splunk optimisation continues: find out why redirection fails and fix it.

I spend a lot of time debugging and looking for patterns until I finally found out that the only thing that was different for the logs that failed to re-direct was their size: having the same application source and sometimes even the same activities, they were much longer than the other ones.

After a long time of debugging and googling for similar issues it turned out that the part of the log event that was matching the regex was never reached by Splunk indexers because by default it only searched into the first 4096 characters of the event, while some of our logs have more than 100,000 characters.

Then another long time was spent on figuring out how this could be fixed. Several options did not work for us until I found the `LOOKAHEAD` setting in Splunk documentation for `transforms.conf`. Adding a `LOOKAHEAD` setting with a custom value solved this problem.

This task took much longer than I expected because of some hidden nuances that were not and could not be obvious at the beginning of the task. No one from my team had any experience of performing this sort of activities, so I had to make all the research on my own. In the end I completed the tasks successfully and it did not take too long.

Key learnings: Splunk has some tricky default settings that can be easily missed if you do not know what exactly you are looking for; some of our logs are way too long.

Friday 25th October 2019

Task: Splunk optimisation continues: research the way of dropping some of the logs based on the value of a specific field (`operatorId`).

Currently we drop all the logs that match the certain setting. It would be good to have an option to switch off all the logs, apart from those that contain this specific key-value pair. I predict a problem here because the field extraction happens during the search time but the redirection/dropping during the indexing time, which happens earlier.

I first wrote: `REGEX="operatorId":"(222222|0)"`. But I was told that some logs might be in the format other than JSON, so we should make it more flexible just in case.

New: `REGEX="?operator.?Id"?(:|=| )?("?222222"?|"?0"?)`

`"?` here make the quotes optional, and `?(:|=| )` makes sure that either `:`, `=` or a whitespace character is allowed between key name and value and `.?Id` allows both "`operator`" and "`operatorId`" as the key name.

Unfortunately, that did not work, so we just used a regex similar to other log drops on the indexer by combining the "drop all" settings with "keep this specific operator". That did not work, either. After a short discussion, the boss said that this task was not important anyway and we should move on.

Now I feel confident enough to make predictions if something will or will not work, so my Splunk skills must have developed quite a bit.

Key learnings: more regex good practices; deeper Splunk knowledges.

Weekly analysis

This week felt both frustrating and productive. On one hand, I had to spend a lot of time researching and googling and trying things out, not knowing if anything would work. On the other hand, I realized that I already have enough knowledge and experience to make some judgements that would turn out to be true, and in the end, I have finally completed the main Splunk optimisation task.

Even though we ended up skipping the last sub-task task altogether, this was not a failure because I made a research, experimented and proved myself right. There probably is some other way of performing this task but it was not that important, so we decided to leave it untouched for the time being. The main point of this task was to allow Production to easily monitor the operations only for the specific operator. It can of course also be done by simply adding `"operatorId"={some value}` to the search command, so that Splunk shows only those logs that contain this key-value pair, which is now working quickly and reliably because of field extraction configuration. So, in this case the time and effort put into making it work was disproportionate to the benefit that the completion of the task would bring.

I deepened my quite basic knowledge of regular expressions. To be fair, I was not exactly knowing what I was doing. I still need to consult with the cheat list but I know more about the logic of some common operators, so next time when I have to work with regex it will be less blind iterating over all the available options, and more thinking and analysing.

Regex is a powerful technique that can increase performance greatly, if used correctly. Talking about the principles of the better regex, Liz Bennett suggests that usually, the more precise the expression is, the better, because it runs faster than the vaguer one, as it excludes the non-matching characters earlier. The best regex should include quantifiers, and specify which exactly characters or character sets should be matched or not matched, instead of using the dot. The author also mentions that a poorly-written regular expression can take 42 times as much time as a good and precise one (10,100 milliseconds vs. 240 milliseconds to process 1,000,000 lines) (Bennett, 2015), so this is something to be kept in mind.

Performance is quite an important issue because we have a lot of logs and we do a lot of calculations in Splunk, so every small operation's optimization counts. Looking at the code I wrote for the log rerouting (see below), I can reflect that the old expression was scanning all the text from the beginning of the file because of `^(.*)`, which is totally unnecessary and slows the process down, while the new one would skip all the non-matching text. Also, using grouping parentheses was completely unnecessary because we were not going to perform any further actions with the captured text.

old: `REGEX=^(.*)"appname":"packaging"(.*)"activity":".*EndpointSPI"(.*)`

new: `REGEX="appname":"packaging".*"activity":".*EndpointSPI"`

We were surprised to find out that some of our audit logs are that long (exceeding 100,000 characters). If we were not going to drop these longs anyway, we would have had to think of a way to reduce them somehow because having such big events in Splunk is not a good practice, as they may affect the search speed, as we do a lot of indexing. Moreover, we keep logs for quite a long time, so they pile up and eventually take up a lot of space, and our company's Splunk storage is always on the verge of being exceeded.

I have rarely thought about such things as optimization of performance, especially while dealing with big and difficult tasks because my aim usually was to make it work. But recently I have been doing more and more optimization tasks, so I started to try keeping it in mind. I think this speaks about my general growth as a developer because it is not really possible to think about performance when you do not have enough knowledge about how things can run and how they should. Of course, I do not always think about optimization but in the areas where I am comfortable and confident with my skills, these issues tend to turn up in my mind themselves. Neither Splunk or regex are these areas yet, so I only start considering such things after I research and read some related material, like this time. But I have entered the growth area and started to develop these new skills.

The main skill development this week was of course my knowledge of Splunk. Currently I am the only one in my team who has knowledge of the specifics and details of such administrative activities, as logs rerouting and dropping, and field extractions. I shared them with my team-mates, so their Splunk skills also improved because of my work. Most importantly, we have seemingly finished the task of Splunk optimisation.

3.8 Observation week 8 (October 28 – November 1)

Monday 28th October – Wednesday 30th October 2019
Task: More audit logging improvements.

After the Splunk task was finished, we did a big review of all the logging that we changed and it turned out that it was still incomplete and inconsistent. Some of the SEBE applications were not fully annotated; in some applications the annotations were not moved from logic to presentation layer; in one of them the logging did not work at all; in the other there were strange errors inside the logs.

We split the tasks: I started to work on the more straightforward task of cleaning up the applications where the annotations were in the wrong place or missing. It took time but there was not any problem during this task.

Task: Fix the logging in one of the applications.

I had a suspicion that the logging did not work because of the same lack of Spring context and/or AOP configuration, so I checked that first thing, and I was right. So, I added those settings and also made a couple of small changes to the annotations. However, after the application was deployed and the logs were checked in Splunk, it turned out that they did not show anything useful to us (because of the nature of the application), so it was decided that `@Audit` annotations should be removed from that application altogether, which I did. Sometimes things like this happen with old applications because no one is even sure anymore if they are in use and how they are used. Anyway, annotating it helped us in a way, now we know that we do not need to worry about the audit logs for this application.

Friday 1st November 2019
Task: Look through the Sonar report and fix the errors and vulnerabilities.

I used the Sonar tool, which is configured for SEBE, to find bugs and vulnerabilities in the recently committed code, if there were any. It showed a couple of small security issues and several major bugs. These bugs are not so severe that the application does not work properly, but rather they are not compliant to some coding guidelines. This time I cannot describe what exactly those were due to security reasons, but fixing them was not at all problematic and only required changing a few lines of code, as prompted by Sonar.

Key learnings: Sonar tool and using it for monitoring the applications.

Weekly analysis

Most of this week my tasks were very similar to what I have done before, so there was not that much to learn. We probably were not attentive enough with audit logging during the previous weeks, so we left some places unfinished, even though it looked complete at the time. But this is what the reviews are for. As we use Kanban methodology, every task has its own ticket sticker on the whiteboard with assigned JIRA issue number. JIRA is a project management tool that provides a virtual blackboard and allows to track the progress

and organize the workflow. When we move things to “done”, before the boss closes the issue in JIRA, we check everything one last time. Sometimes mistakes such as these come to light, and the ticket is back to the board until we resolve it.

One of the philosophical learnings this week was that sometimes you spend time and effort doing some task only to realise upon its completion that it was not needed in the first place. But there was no other way than to do it and see the result. It was a little frustrating for me that I had to undo the changes that I did, simply because they turned out to be irrelevant. On the other hand, the main reason why we were working on the logging in the first place was to identify which applications are in use and how exactly they are used, and that we did for this specific app as well, so my work was not fruitless.

As for the skills development, this week I learned about new software that we use, and found out how to use it. According to an article by Carlos. R. Hernández, Sonar, or SonarQube, is a web-based tool for analysing the code quality of Maven-based Java projects. It is an open-source platform which performs continuous inspection of the code and automatically detects potential bugs, security vulnerabilities, code smells, duplications, test coverage, compliance to coding standards, architecture and design, and provides metrics in an intuitive graphical UI (Hernández, 2017).

It is a very useful tool that can help to simplify the development workflow and improve the quality of the results. It usually runs inspections every time new code is added. In our company, Sonar is configured in such a way that the new inspection is triggered by a Jenkins job. Usually in this kind of setup, the Jenkins job would be triggered by a new application build or a new merge to the master branch. But in the CI environment we usually make a lot of commits to the same project every day, and no one is immediately checking Sonar reports because there are some more important things to do, and also because we would already know if we had some major problem with the code, as the deployment or the tests would fail. So, for our environments, the Sonar Jenkins job is not triggered by each new commit, but is set to automatically run every morning. It can also be run manually at any point in time. This way, we have consistent inspections without overdoing it.

During my first brief encounter with it, I found Sonar to be quite an interesting and easy to use tool. One of the most interesting features for me was the vulnerabilities analysis. The tool even detects the Maven dependencies with known vulnerabilities, which can be hard to discover on your own when the project has hundreds of dependencies, not all of which

are upgraded to the latest version. Other features, such as bugs, code smells or bad design detection are also quite helpful. For example, I learned that Date and Calendar objects should not be created as static. However, some of the reported bugs or code smells might be false positives, so the developer should not blindly trust the software. I believe, regular use of Sonar can teach me a lot about good practices and application design, and sometimes help with identifying major problems in the code that I write to fix it quickly.

3.9 Observation week 9 (November 4 – 8)

Monday 4th – Tuesday 5th November 2019
Task: Create an additional endpoint that would support OPI1.2 functionality for OPI1.1 calls:

1. Implement the code
2. Write unit tests
3. Write system tests

OPI is one of the protocols used in the company. It has 2 versions, the 1.2 is going to be taken out of use in the future. We were asked to add one of the functionalities of the older version to the other one.

Since the new endpoint has to be a mix of the existing two, I copied some code from both endpoints, combined it and modified it accordingly. The main difference was that the older protocol supported two more parameters, of which I had to take one. Then I made changes to the adapter class, which handles the request before sending in further to the logic layer of the application. I added a new method that would be called by the new endpoint.

OPI is a separate application that is downloaded to the current project as a Maven dependency. SEBE uses imported OPI's classes, such as special Request and Response containing the necessary fields. For my changes to work it is necessary to create new Request and Response classes in the OPI application. This was done by my boss because OPI is a SOAP service which is configured in a special way in XML with WSDL, not with Java. After deploying these changes, releasing a new version of OPI and re-importing the dependency to the current project, the changes will be automatically included and the new classes can be directly used in SEBE.

The code implementation was easy to do. This task is of the kind of everyday tasks that I am most accustomed to, as they are mostly Java programming, which is my main competence. I have completed it successfully and without assistance, even though there was a small complication when it came to importing a new version of OPI to the project, which I was able to resolve on my own. The next step is testing the code.

Key learnings: basics of WSDL configuration and SOAP calls; IntelliJ/Maven features for project and dependencies management.

Wednesday 6th November 2019
Task: Write unit tests for the new endpoint.

When I started to work on the unit tests, I discovered that the existing unit tests for the old endpoints were somewhat disordered. After discussing the matter with the team, it was agreed that I should refactor them. The problem was that the existing test class was a mixture of different tests which were testing several interdependent classes (not only the endpoint methods, but also the adapter and the related support process). I had to refactor the code by splitting the tests into several separate unit test classes, one application class per test class, and refactor the test methods a little, as well as create new small unit tests that checked the classes' basic functionality.

The task was simple in itself but refactoring big complex tests took some time. Moreover, these tests were written using older versions of JUnit and Mockito, which lacked some of the functional that I was used to, or behaved slightly differently than I expected, so I had to adapt to it.

Key learnings: unit testing good practices: only test one class at a time, avoid mocking too many dependencies; practice with older versions of Java testing frameworks.

Thursday 7th November 2019
Task: Write system tests for the new endpoint.

First of all, I had to find the existing tests that tested the old endpoints and figure out how they work to use them as examples. This naturally took some time. Apart from adding the new tests, I also needed to write some support methods for adapters used by the tests for imitating the client call, request creation and sending.

There was a lot of small things to do and to keep in mind but all in all I did not face any problem. Having examples in front of me was enough. I also learned about the structure of our system tests for this kind of operations, which was different from what I had worked with before, even though it was the same project.

Key learnings: more about SEBE system tests.

Friday 8th November 2019
Task: Change CostCenter tags.

I had two separate small tasks. First one is related to AWS. We use tags for all the AWS resources, and Production asked us to change the value of the CostCenter tag for all our team's projects. I searched for the specified value in all the git repositories and fetched the projects that needed changing. Then I carefully searched through every project and changed the values.

Task: Split full regression tests.

The second task was related to the fact that all the tests in the new SEBE AWS environment are run as a batch and it is not convenient to see which ones are currently running. Moreover, we have an issue of tests timing out and we would like to know why that happens, so we have decided to split the tests in separate commands.

For that I needed to modify the shell script used by the Jenkins job that runs the tests. We also wanted to temporarily restrict the tests to only some of the most important applications, so that they took less time to run through. I had to do some small research about Nose tests commands and then modified the script.

This day's tasks were quite small, easy-to-do and low-effort. I have completed them without any difficulties.

Key learnings: Nose tests commands and the way the tests are found.

Weekly analysis

This week was task-reach and quite fruitful. I have completed all my tasks without any significant difficulties or complications, and I have acquired some new skills even though the tasks were simple and did not involve a lot of research.

I have learned what is the OPI protocol which is used in our services and what are the main differences between its versions. This is a very project-specific knowledge but since my team's main and biggest global task is maintenance and development of several existing services, it is important to know how they work. Therefore, this was a valuable learning for me, even though it is small in comparison to all the things I am yet to learn even after 1.5 years in the company.

I will also keep in mind a basic but important thing that I realised while performing the task: if some code (in this case, an endpoint) is currently in use and you need to add some functionality to it, you should try to avoid changing it directly, so that it does not break. So, instead of changing the existing OPI1.1 endpoint, I added a second endpoint for the same URL path and put the changes there. So that the regular call will still go to the old endpoint but the call that contains different parameters will go to the new endpoint. Thus, we minimize the possibility of failures and simplify debugging and monitoring.

In addition, during the same task I learned some basics of SOAP messaging protocol. I have only worked with REST API before and only heard vague things about SOAP. Even though it seems to me that REST is both more convenient and more commonly used nowadays, it is still good to broaden your professional horizons and learn new things that might be of use, especially considering the fact that some of our old services use SOAP.

According to Nyma Malik, Simple Object Access Protocol, or SOAP, is a standard communication protocol system that permits processes using different operating systems like Linux and Windows to communicate via HTTP and its XML (Malik, 2017). For SOAP services, the input and output classes (Request, Response) are created in with WSDL, which is an XML format used to define the service's structure and containing the description of the whole service (W3C). After that these classes can be directly used in the application code and there is no need to create new Java classes for Request and Response objects and map them to request and response data.

An extract of WSDL code that defines the request:

```
<xsd:element name="editedNameRequest">
        <xsd:complexType>
            <xsd:choice minOccurs="1" maxOccurs="1">
                <xsd:element name="customer" type="Identity" />
                <xsd:element name="voucher" type="xsd:string" />
            </xsd:choice>
        </xsd:complexType>
</xsd:element>
```

The code above defines the element of name `editedNameRequest`, which is the name under which it will be accessed in Java code. The `xsd:choice` tag defines that the request may contain one of the following parameters: `customer` of type `Identity` or `voucher` of type `string`.

Although I am very familiar with unit testing as it is one of my main responsibilities, I am still learning something about it every time I need to work on the tests. At first it has mostly been some theoretical or practical skills but recently, having learned most of the needed “know-hows”, I have started moving into the good practices area.

A software engineer Brian McGlauflin writes that unit testing is “the practice of testing individual units or components of an application, in order to validate that each of those units is working properly”. Even though the “unit” is not strictly defined, in Java it is usually a single class. The aim of the unit test is to make sure each separate part of the system works as expected. Checking that several parts work together is the focus of integration tests (McGlauflin, 2019). Therefore, unit tests should be as basic as possible, which was not the case in the code I was working with. It was not something new for me but I have not given this much thought before performing this task. Now I will think more carefully about the way the testing should be done and will try to adhere to the good practices.

One of the good practices for unit testing is isolation. That means, as McGlauflin also adds, that the tests should be runnable in any order and on any machine, without affecting each other in any way. Another important thing is to only use a single use case for each unit test. This is a rule I am sometimes tempted to break, in cases when you need to test the exact same thing with multiple parameters. When I thought about it after reading about the good practices, I found out that the JUnit framework that I use for unit testing allows to use a new feature, parametrized tests. Basically, it simplifies the situation I described

above by allowing to execute a single test multiple times, feeding it a list of parameters (Deghani, 2019). Unfortunately, in the current project, as I already mentioned, tests run with an older version of JUnit, but I will definitely try this feature out in some newer project with JUnit 5.

While performing this task I have also researched the test-driven development strategy (TDD). It is a methodology in which the unit tests are written before any other code. According to the author of the Agile Manifesto Robert C. “Uncle Bob” Martin, the 3 main principles of TDD are:

1. You are not allowed to write any production code unless it is to make a failing unit test pass.
2. You are not allowed to write any more of a unit test than is sufficient to fail; and compilation failures are failures.
3. You are not allowed to write any more production code than is sufficient to pass the one failing unit test.

This is discussed by Matt Brooks in his article “Advanced test-driven development with Uncle Bob”. According to the author, such an approach is intended to force developers to write the code that is easier to test, and all the use cases are thought of in advance. Also, one of the results of such an approach is a close to 100% test coverage of the code, which allows a certain freedom of refactoring and being sure that it will not break the business logic (Brooks, 2017).

I personally tried to use this methodology but not for long. This approach takes some getting used to, and maybe when I start working on something completely new instead of changing the legacy code, I will give it another try. But in the course of this brief experiment, I started to think about test cases before writing any new code, which is certainly an improvement of my general approach to testing.

Last but not least, my knowledge of Nose tests broadened and now not only can I write the tests for Nose but also understand how the framework runs them. To split the test run, I had to change the command that looked like this:

```
nosetests -s -v -a \!sequential —processes=4 —process-timeout=1200
—with-xunitmp —xunitmp-file=../test-results/parallel_tests.xml —
```

```
exclude-dir=tests/upstream —exclude-dir=tests/reporting —exclude-
dir=tests/legacy —exclude-dir=tests/fuzz tests >> tests.log 2>&1
```

This basically means “run everything apart from the excluded directories”, while what we needed was to run the certain directories only. I made a research and learned that generally Nose looks into all the subdirectories and fetches the test files (Nose Documentation), so for the new commands we can drop the exclusions and just specify one directory we want:

```
nosetests -s -v -a \!sequential —processes=4 —process-timeout=1200
—with-xunitmp —xunitmp-file=../test-results/licensing_paral-
lel_tests.xml tests/licensing >> tests.log 2>&1
```

To sum up, this week I have improved my skills in testing by having learned more about Python testing and the Nose tests framework, and general testing principles for unit tests. I have also learned about and tried out a new approach to unit testing, which was an unusual experience that positively affected my way of thinking. As well as this, I got acquainted with a new technology (SOAP, WSDL), therefore broadening my general skillset, while finishing one of the quarterly goals.

3.10 Observation week 10 (November 11 – 15)

Monday 11th November 2019
Task: Change the Nose command.

The first task was the continuation of the last week’s last one. Now that we checked that my changes worked, I added the command for running all the other tests after the split ones. It was basically just adding two lines of code to the same script as before.

Task: Start looking into SMI for audit logging improvements.

We are going to do the same audit logging improvement work with another old service, which is called SMI. So, my task for a start was to set it up locally and start studying the service itself.

While setting it up, I encountered a problem. It was not related to MySQL anymore, as I have configured it for the new service easily, using my previously gained knowledge about configuration. The problem was the Java settings, because this project was using jdk1.6 for compilation, while my default Java is jdk1.8. I needed to either install jdk1.6 to my machine or instruct the service to compile with jdk1.8. I try to avoid installing the software unless it is crucial, not to clog my machine with things I only used once and forgot about. So, I preferred the latter way, though it was somewhat of a hack: I changed some settings in the `pom.xml` file, compiled the application and then undid the changes. After that the project compiles without a problem and no changes have been actually made. Since I only needed to make this change to be able to work locally, I believe this was a reasonable thing to do.

Key learnings: SMI structure; a Maven hack to trick the app into compiling with the right jdk.

Tuesday 12th November – Wednesday 13th November 2019
Task: Start annotating SMI.

This service is more difficult to annotate, even though it is smaller and less complex than SEBE. The problem is, it is a SOAP service, so we cannot annotate the endpoints, as they are written in XML. We have to think of the other places to add annotations, somewhere in the code as close to the endpoints as possible. Furthermore, SMI consists of several applications calling each other horizontally and in the end calling SEBE. In this case we want to know not only the external calls but also what is going on inside SMI, so we are going to add audit logging for these internal communications as well.

We discussed it as a team and decided on a starting point. Then I was doing the same monotonous job as previously with SEBE.

Key learnings: SMI; more about SOAP.

Friday 15th November 2019
Task: Create a new job for the operator deletion tool.

The old manually-created Jenkins job is incompatible with some of the administrative changes that were performed. My task is to rewrite the job to a Jenkins Pipeline.

As this job was created manually a long time ago, there is no code that I could use. Although, it would not make much sense to try to use it, anyway, since I need to convert it to a Pipeline which uses different syntax. So, I found a job definition and the tool settings for another Python tool that I worked on some time ago to use as an example. I created the configuration files for the tool and the Jenkinsfile for the job and started working on writing a job, which has some specifics I am not familiar with yet.

Weekly analysis

During my last reporting week, I unfortunately did not have any major tasks what would allow me to learn a lot of new things. Sometimes you get many new skills in a day, sometimes almost none in a couple of weeks, when the tasks are not challenging and correspond your skills too well. On one hand, I would prefer to have more challenge because I enjoy finding a solution to a problem, trying out new things and learning new skills. However, this can also be quite stressful, so these calm weeks with few tasks are a kind of relaxation.

This week I did not need a lot of help. First, we discussed all the tasks as a team, so everyone was at the same page and received some kind of instructions how to proceed. Although I asked for some assistance doing the Jenkins task because the configuration files require knowledge of some internal categories to describe such things as: who is the target user of this tool, to which inner structure it will be released, in which development state the tool is, what is the release version number, etc. My boss helped me with those since I had no possibility to know these things myself.

However, I learned a little more about Jenkins and Python. I have never configured a Python tool to be run on Jenkins before. One of my first tasks during this thesis was writing a Jenkins job for a Python tool. But that was very different, since that tool was not written and released by us, so it was already configured and we basically only needed to run it from Jenkins. The current case is a bit different because we are releasing our tool to the company's internal repository for anyone else to use. So, writing the Jenkins Pipeline syntax is the easiest part, the trickier one is to configure the tool to be used, correctly setting all the requirements, etc. I also needed to slightly refactor the tool's code, so that its entry point could be referenced in the setting file in the right way. This configuration file,

`setup.py`, is used for downloading the dependencies, creating packages, setting the application properties, and is run when installing the Python application with the command `pip install`.

According to Python Packaging User Guide, `setup.py` has two main functions:

1. It is the file where various aspects of your project are configured. The primary feature of `setup.py` is that it contains a global `setup()` function. The keyword arguments to this function are how specific details of your project are defined.
2. It is the command line interface for running various commands that relate to packaging tasks.

In our company, most of the Python tools are used with virtual environment. According to the documentation, a virtual environment is a semi-isolated Python environment that allows packages to be installed for use by a particular application, rather than being installed system wide (Python Documentation). I learned that installing the package inside the virtual environment allows you better control, since all the required dependencies are installed inside this environment, not on your machine or server, so you do not have to worry about overwriting existing packages, or about dependencies clashing, and it is easy to clean up afterwards. Also, it is very convenient to set the necessary Python version to the virtual environment and then use it as a project interpreter, which is quite useful in situations such as ours, when most of the projects are very version-dependent.

Summing it up, the main development this week was deepening my knowledge of Python. I have learned to configure Python projects and acquired some theoretical knowledge about the setup file and virtual environments. Before starting to work at F-Secure I did not know much Python, I only started a beginner online course a month or two before getting the job. So, I have learned everything I know about Python by practice, and by the time of starting this thesis I was already somewhat good at it, and every new Python task allows me to develop my skills and knowledge and grow in this direction.

4 Discussion and conclusions

Looking back at my work duties and skills at the beginning of the period of this thesis and comparing them to the current situation, I can say that by the end of the reporting period I have reached all the professional goals that I had set for myself. Let's now look at them in more detail.

1. Audit logging

I have learned that audit logging is a process of recording the important information of application's activities and parameters, which is used for monitoring its behaviour for the purposes of audit and/or debugging. My developer skills broadened because now I am able to not only write an application but also configure the logging for it and thus get valuable information about its state. I know how to use this information and what are the optimal ways of configuring the audit logging for different monitoring purposes. From what I would call an elementary level of knowledge in this area that I had when I started this task, I have professionally grown into a rather confident user and developer.

As a result of my work on this task, audit logging in SEBE was greatly improved: the old `@Audit` frameworks were replaced by the new one; the annotations were moved to the most important for us parts of the applications (endpoints); new logging was added to some of the applications. Therefore, we are now getting a better picture of the service, showing which processes are running in our backend, and this will be of great assistance when we start working on the data migration.

2. Splunk

Being a total beginner at Splunk when starting to work on the related tasks, I familiarized myself with general architecture and the processes of the Splunk platform; learned which specific operations happen on which of its components and how; understood the internal communications and dependencies. Now not only am I able to use Splunk for searching, monitoring, storing and analysing the logging data, but also to configure various settings related to the indexing, transforming of the logs, extracting the fields from logs events. By learning these skills, I was able to finish the task of Splunk reconfiguration for improvement of audit logging in SEBE.

As a result, our logs are now better structured; new indexes are created for simplifying the search operations; log events are more readable and easier to search; new transformation

configurations allow to easily drop or reroute certain groups of logs, if necessary. These improvements are relevant both for our development team and for Production, as they use Splunk for monitoring the production environments, so I personally have directly influenced one of the interest groups at the company in a positive way.

3. MySQL

This was another one of those areas where I had close to zero previous knowledge. In the process of setting up the local environment I have learned many valuable practical skills. First of all, how to connect a local instance of MySQL server to the application by setting the host, port number and other parameters using Hibernate framework configuration and MySQL Workbench. Furthermore, I familiarized myself with some of the essentials of the MySQL server administration, such as creating users and roles for the database, setting permissions and restricting access.

As a result, I have had a fully working environment for the SEBE project set up at my local machine, which I used for development and testing. The knowledge that I acquired during this process was sufficient for me to perform the same task again on my own when I needed to configure a new environment for the SMI project a few weeks later, so I reckon it can be said that I have consolidated this knowledge.

4. Java

Even though I had been already rather advanced in Java, and I have not had that many Java-related tasks during the reporting period, I can still say that I have grown as a Java developer, at least in my own eyes. One of the goals I had for my Java skills improvement was to learn more about the “administrative” side of Java, meaning different application settings, and the frameworks that we use with our applications. By the end of this thesis, I have learned about configuring the various aspects of the application with the use of Spring (AOP context and annotation context to be able to import and use custom annotations), Hibernate (for the database mapping) and Maven (for the dependencies management), thusly becoming more experienced in the configuration of Java projects. It is a new step for me from being simply a coder.

As well as this, I have developed my programming skills by getting to know some small things, such as conventions about the order of the elements in the class, new ways of dealing with date and time, and a couple of smaller tricks. I feel that in the end, all of these help me to better see the bigger picture as a Java developer.

5. Testing

One more of the already familiar areas for my development was testing. During the time I have been working at F-Secure, I have been forced to understand the value and the importance of testing, so I find the testing skills to be no less important for a developer than the programming ones. However, in practice I was often forgetting to write the tests and probably did not have a lot of general knowledge about testing. Therefore, I wanted to improve my approach to testing in general.

Talking about the specifics, I have learned some good practices of unit testing and used them to improve the existing badly designed tests, deepened my knowledge of testing frameworks by learning how to use older versions of JUnit and Mockito for Java, as well as completely new for me Python Nose tests, and discovered quite a useful feature of JUnit for running parametrized unit tests that I will be able to use in newer projects. As for more practical side, I have used my newly acquired knowledge to test and debug the application I was working on by writing and executing both unit and system tests.

As well as this, in the process, I tried out the test-driven development methodology, which is an approach in which a unit test is written before any other line of code for the same purpose. Even though at this time I was not convinced that I should change my professional habits to TDD, I feel that it was a useful experience because it made me think about the test cases while I am writing the new code, considering straight away the ways how it can be tested. I believe that continuing to work with the same mindset will eventually improve the quality of my code. Although it is hard to measure the results of “improving my approach to testing”, in my view, I have started moving in the right direction, and, therefore have reached this goal.

6. General skills development

Talking about the less specific or task-oriented development goals, I wished to improve my existing skills in Jenkins, AWS and Python in any way, and also to learn about and/or start to use some software or technologies that I have never worked with before.

Some of my skills developed less than others, but still there was at least a slight broadening of knowledge in each of those that I listed in my goals list in chapter 2.2. For AWS, I discovered the differences between the instance types and got some idea how to choose the right one for various purposes. For Jenkins, I learned some good practices for a) continuous integration in general, and b) using the Pipeline job; found out about the existence

of a Boolean parameter and used it in the job that I wrote; learned how some of the Jenkins good practices contribute to the performance optimization. The skill that I developed the most of these was Python. I had to deal a lot with it, performing very diverse tasks, such as writing the tests, managing the dependencies for an application, configuring the application for release, using virtual environment. One of the most vivid achievements for myself was getting familiar with Python 2 and generally learning to work with basically any version of Python. Also, some of the things from outside the goals list were getting new small but useful knowledge about shell scripting and regular expressions.

What I did not expect was the number of new tools and techniques I would get rather familiar with during just 10 weeks. These include Splunk (monitoring platform), MySQL (RDBMS), GlassFish (application server), SonarQube (code quality analysis tool), RabbitMQ (messaging software), SOAP (messaging protocol), WSDL (XML format). The level of knowledge acquired for these new learnings, of course, varies. But even the smallest one is still an important development because any new skill is a small step to becoming a better professional.

7. Successfully completing my quarter tasks

I also had this specific goal of completing all of my global quarterly tasks, which might not have had so much to do with my skills growth or professional development, but it was very important for me personally to see that I am actually achieving something and bring value to the company I work in. The fact that I managed to finish all the tasks in time is deeply satisfying for me because I sometimes get frustrated with myself when I am stuck for too long on something and feel like I am not progressing and even become disappointed with myself as a professional.

But one of the important things that I learned by writhing this thesis was that the work is never fruitless, even if it sometimes seems as such. If it takes more time than expected and there are no tangible results in the end, it does not mean that this time was wasted and nothing was achieved, because every experience is a learning, and every learning leads to some growth, even be it invisible at a time.

In some areas this time I only learned a little, the very basics. That only means that they are fields for my future development, the ones that I might have never considered before, or the ones that are somewhat familiar, or the ones that I feel confident in, because there is always only so much to learn. I would like to continue developing in all the areas that I

was analysing in this thesis but I might specify some of the less obvious ones. One of them is optimization. This is not something that you learn by doing one or two tasks but a rather long process that needs you to always keep it in mind. I will aim to look for the most optimal ways to make things work, to learn more good practices and make more research. Talking about something less abstract, my main areas of focus probably will be those configuration skills in Java and Python because I would like to be able not only to write the code, which is easy, but to maintain the applications, integrate them and understand the underlying processes.

Another personal discovery – well, not so much of a discovery, but more of a confirmation of what I had already known about myself, is that I perform best when I meet challenge. Consciously, I might not be wanting to exit my comfort zone, but in reality, I feel most satisfied with my work and myself if I succeed in a situation which needs me to solve a difficult problem, figure out a way of doing some unconventional task, find an answer to a tough question. I believe this is a good mindset for someone who wants to become a better professional because it is much harder to learn new skills and grow if you avoid challenge and prefer to do something you are good at over and over again. As even some of the tasks I described can show, I personally would not be able to make myself stick to one same old routine without trying to broaden my horizons.

Haaga-Helia
University of Applied Sciences

References

Amri, A. E. 15 September 2017. Choosing your AWS Instance. URL: https://medium.com/faun/choosing-your-aws-instance-61d8edcd3d4b. Accessed: 16 November 2019.

AWS. 24 April 2019. Amazon EC2 T3a Instances Are Now Generally Available. URL: https://aws.amazon.com/about-aws/whats-new/2019/04/amazon-ec2-t3a-instances-are-now-generally-available/. Accessed: 17 November 2019.

Bennett, L. 18 June 2015. Regexes: The Bad, the Better, and the Best. URL: https://www.loggly.com/blog/regexes-the-bad-better-best/. Accessed: 17 November 2019.

Brooks, M. 2 May 2017. Advanced test-driven development with Uncle Bob. URL: https://manifesto.co.uk/test-driven-development-uncle-bob/. Accessed: 17 November 2019.

Dehghani, A. 2019. Guide to JUnit 5 Parameterized Tests. URL: https://www.baeldung.com/parameterized-tests-junit-5. Accessed: 17 November 2019.

GitHub. RabbitMQ. Receive.java. URL: https://github.com/rabbitmq/rabbitmq-tutorials/blob/master/java/Recv.java. Accessed: 16 November 2019.

GitHub. RabbitMQ. Send.java. URL: https://github.com/rabbitmq/rabbitmq-tutorials/blob/master/java/Send.java. Accessed: 16 November 2019.

Guru99. What is Jenkins? Continuous Integration (CI) Tool. URL: https://www.guru99.com/jenkin-continuous-integration.html#2. Accessed: 17 November 2019.

Harris, D. 17 December, 2010. How Splunk Is Riding IT Search Toward an IPO — Tech News and Analysis. URL: https://gigaom.com/2010/12/17/how-splunk-is-riding-it-search-toward-an-ipo/. Accessed: 16 November 2019.

Hernández, C.R. 12 June 2017. How to improve your workflow with SonarQube. URL: https://engineering.bitnami.com/articles/how-to-improve-your-workflow-with-sonarqube.html. Accessed: 18 November 2019.

Jenkins User Documentation. URL: https://jenkins.io/doc/. Accessed: 17 November 2019.

McGlauflin, B. 25 June 2019. Unit Testing Best Practices: How to Get the Most Out of Your Test Automation. URL: https://dzone.com/articles/unit-testing-best-practices-how-to-get-the-most-ou. Accessed: 17 November 2019.

Malik, N. 7 September 2017. The Difference Between REST and SOAP APIs. URL: https://dzone.com/articles/difference-between-rest-and-soap-api. Accessed: 17 November 2019.

Nose Documentation. Finding and running tests. URL: https://nose.readthedocs.io/en/latest/finding_tests.html. Accessed: 17 November 2019.

Oracle. 1995-1999. Java Documentation. 3 – File Organization. URL: https://www.oracle.com/technetwork/java/javase/documentation/codeconventions-141855.html. Accessed: 16 November 2019.

Oracle. Package java.time. URL: https://docs.oracle.com/javase/8/docs/api/java/time/package-summary.html. Accessed: 16 November 2019.

Oracle. Class SimpleDateFormat. URL: https://docs.oracle.com/javase/7/docs/api/java/text/SimpleDateFormat.html. Accessed: 16 November 2019.

Paraschiv, E. 2017. 9 Logging Sins in Your Java Applications. URL: https://dzone.com/articles/9-logging-sins-in-your-java-applications. Accessed: 16 November 2019.

Pemberton, A. 2016. Top 10 Best Practices for Jenkins Pipeline Plugin. URL: https://www.cloudbees.com/blog/top-10-best-practices-jenkins-pipeline-plugin. Accessed: 17 November 2019.

Pronschinske. M. 1 October 2013. The Blogging Programmer's Style Guide: Front-End or Frontend? URL: https://dzone.com/articles/blogging-programmers-style. Accessed: 21 November 2019.

Python Documentation. Installing Python Modules. URL: https://docs.python.org/3/installing/index.html#installing-index. Accessed: 18 November 2019.

Python Packaging User Guide. Packaging and distributing projects. URL: https://packaging.python.org/guides/distributing-packages-using-setuptools/#setup-py. Accessed: 18 November 2019.

RabbitMQ. URL: https://www.rabbitmq.com/. Accessed: 16 November 2019.

RabbitMQ. Java Tutorial. URL: https://www.rabbitmq.com/tutorials/tutorial-one-java.html

Richards, M. 2015. Software Architecture Patterns. Chapter 1. URL: https://www.oreilly.com/library/view/software-architecture-patterns/9781491971437/ch01.html. Accessed: 16 November 2019.

Rouse, M. 2018. Staging environment. URL: https://searchsoftwarequality.techtarget.com/definition/staging-environment. Accessed: 17 November 2019.

Splunk Answers. 2016. Extract JSON data within the logs (JSON mixed with unstructured data). URL: https://answers.splunk.com/answers/117121/extract-json-data-within-the-logs-json-mixed-with.html. Accessed: 17 November 2019.

Splunk Answers. 2017. Is there any benefit to explicit field extraction vs letting Splunk do it on its own? URL: https://answers.splunk.com/answers/584013/is-there-any-benefit-to-explicit-field-extraction.html. Accessed: 17 November 2019.

Splunk Documentation. Components of a Splunk Enterprise deployment. URL: https://docs.splunk.com/Documentation/Splunk/8.0.0/Capacity/ComponentsofaSplunkEnterprisedeployment. Accessed: 17 November 2019.

Splunk Documentation. Indexes.conf. URL: https://docs.splunk.com/Documentation/Splunk/8.0.0/Admin/Indexesconf. Accessed: 16 November 2019.

Splunk Documentation. Inputs.conf. URL: https://docs.splunk.com/Documentation/Splunk/latest/Admin/Inputsconf. Accessed: 16 November 2019.

Splunk Documentation. Props.conf. URL: https://docs.splunk.com/Documentation/Splunk/8.0.0/Admin/Propsconf. Accessed: 16 November 2019.

Splunk Documentation. Transforms.conf. URL: https://docs.splunk.com/Documentation/Splunk/8.0.0/Admin/Transformsconf. Accessed: 16 November 2019.

Stack Overflow. How to convert local time string to UTC? URL: https://stackoverflow.com/questions/79797/how-to-convert-local-time-string-to-utc. Accessed: 17 November 2019.

Tutorialspoint. AOP with Spring Framework. URL: https://www.tutorialspoint.com/spring/aop_with_spring.htm. Accessed: 16 November 2019.

Tutorialspoint. Hibernate – Configuration. URL: https://www.tutorialspoint.com/hibernate/hibernate_configuration.htm. Accessed: 16 November 2019.

Tutorialspoint. Spring – Annotation Based Configuration. URL: https://www.tutorialspoint.com/spring/spring_annotation_based_configuration.htm. Accessed: 16 November 2019.

W3C. 15 March 2001. Web Services Description Language (WSDL) 1.1. URL: https://www.w3.org/TR/wsdl.html. Accessed: 17 November 2019.

Wikipedia. Hibernate (framework). URL: https://en.wikipedia.org/wiki/Hibernate_(framework). Accessed: 16 November 2019.

Appendices

Appendix 1. Abbreviations

AMI	Amazon Machine Images
AOP	Aspect-Oriented Programming
API	Application Programming Interface
AWS	Amazon Web Services
CI	Continuous Integration
CPU	Central Processing Unit
DB	Database
EC2	Elastic Compute Cloud
JDK (jdk)	Java Development Kit
JSON	JavaScript Object Notation
MS Teams	Microsoft Teams
OOP	Object-Oriented Programming
REST	Representational State Transfer
SOAP	Simple Object Access Protocol
TDD	Test-Driven Development
UI	User Interface
WSDL	Web Services Description Language
XML	Extensible Mark-up Language