System Integration F2020

Exercises

Henrik Kramselund Jereminsen
hlk@zencurity.com

March 8, 2020

Contents

Preface

This material is prepared for use in *System Integration F2020* and was prepared by Henrik Lund Kramshoej, Zencurity Aps. It describes the setup and applications for trainings and workshops where hands-on exercises are needed.

Further a presentation is used which is available as PDF from kramse@Github
Look for system-integration-exercisesin the repo security-courses.

These exercises are expected to be performed in a training setting with network connected systems. The exercises use a number of tools which can be copied and reused after training. A lot is described about setting up your workstation in the repo

https://github.com/kramse/kramse-labs

Prerequisites

This material expect that participants have a working knowledge of internet from a user perspective. Basic concepts such as web site addresses, IP-addresses and email should be known as well.

Have fun and learn

Exercise content

Most exercises follow the same procedure and has the following content:

- **Objective:** What is the exercise about, the objective
- **Purpose:** What is to be the expected outcome and goal of doing this exercise
- **Suggested method:** suggest a way to get started
- **Hints:** one or more hints and tips or even description how to do the actual exercises
- **Solution:** one possible solution is specified
- **Discussion:** Further things to note about the exercises, things to remember and discuss

Please note that the method and contents are similar to real life scenarios and does not detail every step of doing the exercises. Entering commands directly from a book only teaches typing, while the exercises are designed to help you become able to learn and actually research solutions.

Exercise 1

Date Formats 15 min

Objective:
See an example of time parsing, and realize how difficult time can be in system integration.

Purpose:
System integration often works with different representations of the same data. Time and dates are one aspect we often meet. Realize how complex it is.

Suggested method:
Visit the web pages of an existing tool, Logstash we will use througout the course and a standard for time and dates.

Write down todays date on a piece of paper, each one does their own.

Then lookup ISO 8601
`https://en.wikipedia.org/wiki/ISO_8601`

I recommend looking at a specific system, used for processing computer logs: Logstash
`https://www.elastic.co/guide/en/logstash/current/plugins-filters-date.html`

Hints:
When you receive a date there are so many formats, that you need to be very specific how to interpret it.

Parsing dates is a complex task, best left for existing frameworks and functions.

If you decide to parse dates using your own code, then centralize it - so you can update it when you find bugs.

Solution:
When you have a

Discussion:
Make sure to visit the web page:
`https://infiniteundo.com/post/25326999628/falsehoods-programmers-believe-about-time`

Did you realize how complex time and computers are?

Then consider this software bug:

”No, you’re not crazy. Open Office can’t print on Tuesdays.”
https://bugs.launchpad.net/ubuntu/+source/file/+bug/248619

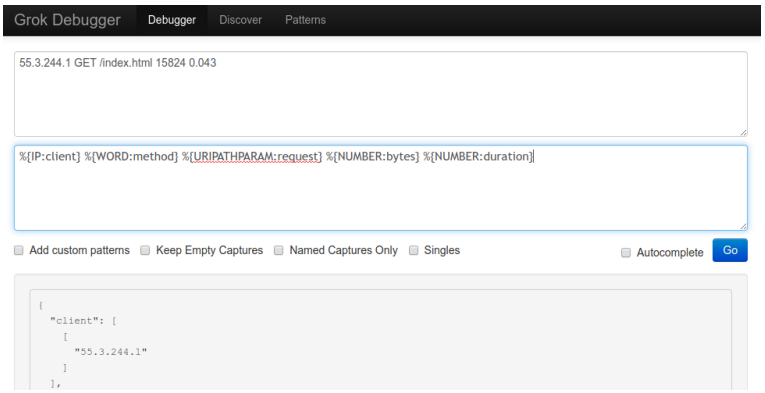
Linked from
https://www.reddit.com/r/linux/comments/9hdam/no_youre_not_crazy_open_office_cant_print_on/

Because a command `file` has an error in parsing data, files with PostScript data - print jobs with the text Tue - are interpreted as being Erlang files instead. This breaks the printing, on Tue(sdays).

We will go through this bug in detail together.

Exercise 2

Grok Debugger 15 min



Objective:
Try parsing dates using an existing system.

Purpose:
See how existing systems can support advanced parsing, without programming.

Suggested method:
Go to the web application Grok Debugger:
`https://grokdebug.herokuapp.com/`

Try entering data into the input field, and a parsing expression in the Pattern field.

Try the data from `https://www.elastic.co/guide/en/kibana/current/xpack-grokdebugger.html`

Hints:
The expression with greedy data is nice for matching a lot of text:
`%{GREEDYDATA:message}`

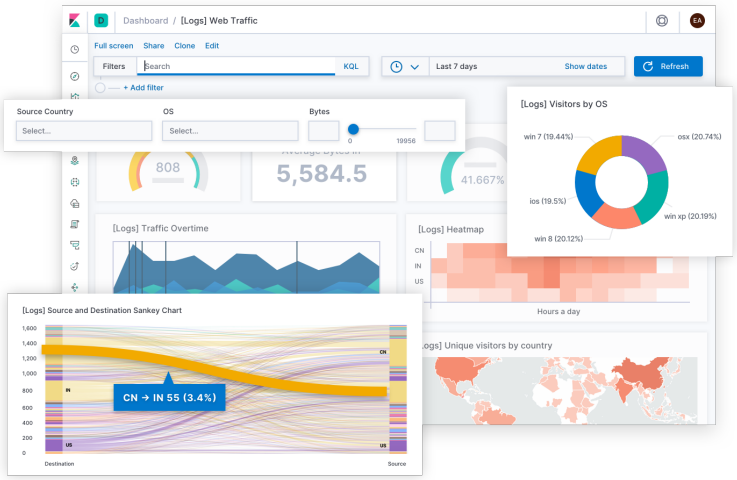
Try adding some text at the end of the input, and another part of the parsing with this.

Solution:
When you have parsed a line and seen it you are done.

Discussion:
The functionality Grok debugging is included in the tool Kibana from Elastic:
`https://www.elastic.co/guide/en/kibana/current/xpack-grokdebugger.html`

Exercise 3

Getting started with the Elastic Stack 15 min



Screenshot from `https://www.elastic.co/kibana`

Objective:
Get ready to start using Elasticsearch, read - but dont install.

Purpose:
We need some tools to demonstrate integration. Elasticsearch is a search engine and ocument store used in a lot of different systems, allowing cross application integration.

Suggested method:
Visit the web page for *Getting started with the Elastic Stack* :
`https://www.elastic.co/guide/en/elastic-stack-get-started/current/get-started-elastic-stack.html`

Read about the tools, and the steps needed for manual installation.

You dont need to install the tools currently, I recommend using Debian and Ansible for bringing up Elasticsearch. You are of course welcome to install, or try the Docker method.

Hints:
Elasticsearch is the name of the search engine and document store. Today Elastic Stack contains lots of different parts.

We will focus on these parts:

- Elasticsearch - the core engine
- Logstash - a tool for parsing logs and other data.
  https://www.elastic.co/logstash

  > "Logstash dynamically ingests, transforms, and ships your data regardless of format or complexity. Derive structure from unstructured data with grok, decipher geo coordinates from IP addresses, anonymize or exclude sensitive fields, and ease overall processing."

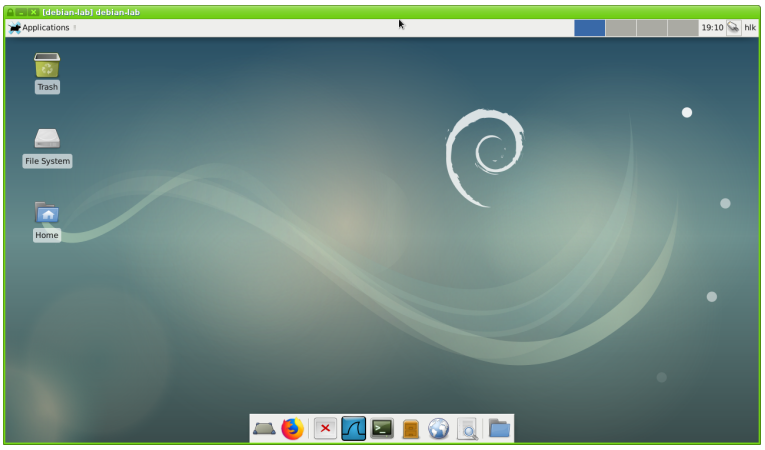
- Kibana - a web application for accessing and working with data in Elasticsearch
  https://www.elastic.co/kibana

Solution:
When you have browsed the page you are done.

Discussion:
You can read more about Elasticsearch at the wikipedia page:
https://en.wikipedia.org/wiki/Elasticsearch

Exercise 4

Check your Debian VM 10 min



Objective:
Make sure your virtual Debian machine is in working order. We need a Debian 10 Linux for running a few extra tools during the course.

This is a bonus exercise - only one Debian is needed per team.

Purpose:
If your VM is not installed and updated we will run into trouble later.

Suggested method:
Go to https://github.com/kramse/kramse-labs/ Read the instructions for the setup of a Debian VM.

Solution:
When you have a updated virtualisation software and Debian Linux, then we are good. Create a snapshot of the server, so you can return to this, in case it breaks later.

Discussion:
Linux is free and everywhere. The tools we will run in this course are made for Unix, so they run great on Linux.

Even Microsoft has made their cloud Linux friendly, and post articles about creating Linux applications:
`https://docs.microsoft.com/en-us/azure/security/develop/`

Exercise 5

Use Ansible to install Elastic Stack

Objective:
Run Elasticsearch

Purpose:
See an example tool used for many integration projects, Elasticsearch from the Elastic Stack

Suggested method:
We will run Elasticsearch, either using the method from:
`https://www.elastic.co/guide/en/elastic-stack-get-started/current/get-started-elastic-stack.html`

or by the method described below using Ansible - your choice.

Ansible used below is a configuration management tool `https://www.ansible.com/`

I try to test my playbooks using both Ubuntu and Debian Linux, but Debian is the main target for this training.

First make sure your system is updated, as root run:

```
apt-get update && apt-get -y upgrade && apt-get -y dist-upgrade
```

You should reboot if the kernel is upgraded :-)

Second make sure your system has ansible and my playbooks: (as root run)

```
apt -y install ansible git
git clone https://github.com/kramse/kramse-labs
```

We will run the playbooks locally, while a normal Ansible setup would use SSH to connect to the remote node.

Then it should be easy to run Ansible playbooks, like this: (again as root, most packet sniffing things will need root too later)

```
cd kramse-labs/suricatazeek
ansible-playbook -v 1-dependencies.yml 2-suricatazeek.yml 3-elasticstack.yml
```

Note: I keep these playbooks flat and simple, but you should investigate Ansible roles for real deployments.

If I update these, it might be necessary to update your copy of the playbooks. Run this while you are in the cloned repository:

```
git pull
```

Note: usually I would recommend running git clone as your personal user, and then use sudo command to run some commands as root. In a training environment it is OK if you want to run everything as root. Just beware.

Note: these instructions are originally from the course
Go to `https://github.com/kramse/kramse-labs/tree/master/suricatazeek`

Hints:
Ansible is great for automating stuff, so by running the playbooks we can get a whole lot of programs installed, files modified - avoiding the Vi editor ☺

Example playbook content

```
apt:
      name: " packages "
    vars:
      packages:
        - nmap
        - curl
        - iperf
        ...
```

Solution:
When you have a updated VM and Ansible running, then we are good.

Discussion:
Linux is free and everywhere. The tools we will run in this course are made for Unix, so they run great on Linux.

Exercise 6

Run Nginx as a load balancer

Objective:
Run Nginx in a load balancing configuration.

Purpose:
See an example load balancing tool used for many integration projects, Nginx

Suggested method:
Running Nginx as a load balancer does not require a lot of configuration.

First goal: Make Nginx listen on two ports by changing the default configuration.

- Start by installing Nginx in your Debian, see it works - open localhost port 80 in browser
  `apt install nginx`
- Copy the configuration file! Keep this backup,
  `cd /etc/nginx/;cp nginx.conf nginx.conf.orig`
- Add / copy the section for the port 80 server, see below
- Change *sites* to use port 81 and port 82

Creating a new site, based on the default site found on Nginx in Debian:

```
root@debian-lab:/etc/nginx# cd /etc/nginx/sites-enabled/
root@debian-lab:/etc/nginx/sites-enabled# cp default default2
root@debian-lab:/etc/nginx/sites-enabled# cd /var/www/
root@debian-lab:/var/www# cp -r html html2
```

Then edit files `default` to use port 81/tcp and `default2` to use port 82/tcp

- also make sure `default2` uses `root /var/www/html2`

Configuration changes made.

These are the changes you should make:

```
root@debian-lab:/etc/nginx/sites-enabled# diff default default2
22,23c22,23
<         listen 81 default_server;
<         listen [::]:81 default_server;
```

```
---
>         listen 82 default_server;
>         listen [::]:82 default_server;
41c41
<         root /var/www/html;
---
>         root /var/www/html2;
```

Config test and restart of Nginx can be done using stop and start commands:

```
root@debian-lab:/var/www# nginx -t
nginx: the configuration file /etc/nginx/nginx.conf syntax is ok
nginx: configuration file /etc/nginx/nginx.conf test is successful
root@debian-lab:/var/www# service nginx stop
root@debian-lab:/var/www# service nginx start
```

You can now visit `http://127.0.0.1:81` and `http://127.0.0.1:82` - which show the same text, but you can change the files in `/var/www/html` and `/var/www/html2`

NOTE: also verify that port 80 does *not work* anymore!

Adding the loadbalancer in nginx.conf.

We can now add the two *servers* running into a single loadbalancer with a little configuration:

Add this into `/etc/nginx/nginx.conf` - inside the section `http { ... }`

```
upstream myapp1 {
        server localhost:81;
        server localhost:82;
}

server {
        listen 80;

        location / {
                proxy_pass http://myapp1;
        }
}
```

And test using `http://127.0.0.1:80`

Hints:
Make changes to the two sets of HTML files

```
root@debian-lab:~# cd /var/www/
root@debian-lab:/var/www# diff html
```



```
html/  html2/
root@debian-lab:/var/www# diff html/index.nginx-debian.html html2/index.nginx-debian.html
4c4
< <title>Welcome to nginx!</title>
---
> <title>Welcome to nginx2!</title>
14c14
< <h1>Welcome to nginx!</h1>
---
> <h1>Welcome to nginx2!</h1>
```

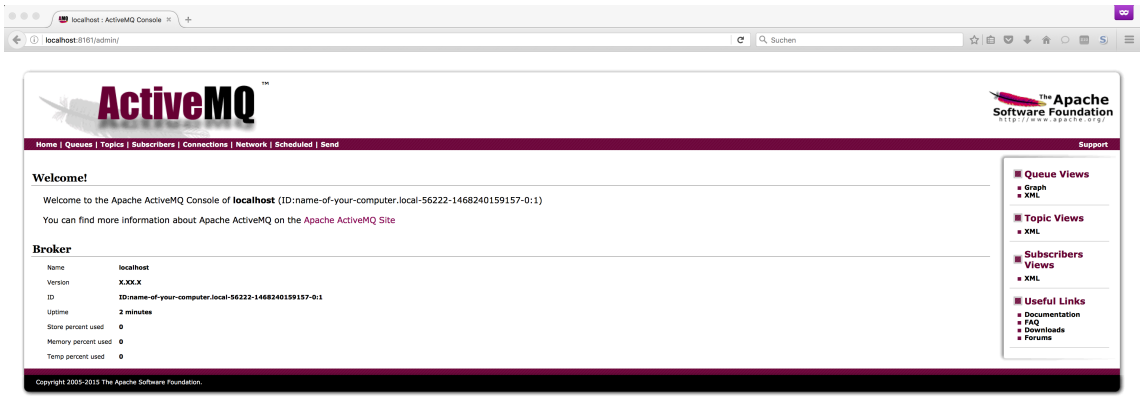
When reloading the page a few times it will switch between the two versions

Solution:
When you have Nginx running load balanced, then we are good.

Discussion:
Nginx is one of the most popular load balancers and web servers. Many sites use it for processing HTTPS/TLS before reaching application servers.

Exercise 7

Runing ActiveMQ



Objective:
Try running ActiveMQ manually - outside of Camel

Purpose:
System integration often works with JMS and ActiveMQ implements this. We can run ActiveMQ as part of Camel, but lets try running it alone.

Suggested method:
Visit the web page `https://activemq.apache.org/getting-started`, read and also follow download link.

I used `apache-activemq-5.15.11-bin.tar.gz`

Follow the instructions for getting ActiveMQ up and running on your Debian server.

Best course of actions is to use a root user and directory such as `/opt` - something like this:

```
cd /opt
tar zxvf /directory/where/you/downloaded/
mkdir data;chmod a+rwx data/
./bin/activemq console
```

Hints:
ActiveMQ is also available as a package in Debian - try searching with `apt search activemq` and can be installed with `apt install activemq`

Note: when installing the package maintainers version the configuration files are often found in the directories below `/etc` while running an unpacked tgz from the project they are close to the software. YMMV.

It is not recommended to use the Debian package for this exercise, but for production use it would be.

In this case there is a configuration file available, but not activated, try this:
`sudo ln -s /etc/activemq/instances-available/main /etc/activemq/instances-enabled/main`

When changing configuration files, if using the Debian package, you can get debug info: `/etc/init.d/activemq console main`

One problem I observed was the directory missing, which can be created using this:

```
# mkdir -p  /var/lib/activemq/data/
# chmod a+rwx  /var/lib/activemq/data/
```

afterwards when programs drop files, we can check the settings, ownership and tighten this.

Also this configuration does NOT start the web console!

Solution:
When you have a running ActiveMQ and can see the web administration you are done.

If you want to investigate further get the demos up and running:
`https://activemq.apache.org/web-samples`

Discussion:
What are the benefits of running ActiveMQ from Debian package vs running it from project binaries directly?

What version does your application need? How do you guarantee this?

Exercise 8

Runing RabbitMQ with Python

Objective:
Try running RabbitMQ with a few Python programs.

Purpose:
RabbitMQ is an alternative to ActiveMQ.

Suggested method:
Use the RabbitMQ tutorial to send a message. Use the tutorial:
`https://www.rabbitmq.com/tutorials/tutorial-one-python.html`

First you would install the server and suporting library - Pika:

```
apt install rabbitmq-server python-pika
```


Check status:

```
# service rabbitmq-server status
  rabbitmq-server.service - RabbitMQ Messaging Server
    Loaded: loaded (/lib/systemd/system/rabbitmq-server.service; enabled; vendor
    Active: active (running) since Sun 2020-03-08 13:57:38 CET; 36s ago
 Main PID: 1663 (beam.smp)
    Status: "Initialized"
     Tasks: 87 (limit: 2386)
    Memory: 77.9M
    CGroup: /system.slice/rabbitmq-server.service
            1659 /bin/sh /usr/sbin/rabbitmq-server
            1663 /usr/lib/erlang/erts-10.2.4/bin/beam.smp -W w -A 64 -MBas agef
            1898 erl_child_setup 65536
            1917 inet_gethost 4
            1918 inet_gethost 4

Mar 08 13:57:34 elastilab systemd[1]: Starting RabbitMQ Messaging Server...
Mar 08 13:57:38 elastilab systemd[1]: rabbitmq-server.service: Supervising proce
Mar 08 13:57:38 elastilab systemd[1]: Started RabbitMQ Messaging Server.
Mar 08 13:57:38 elastilab systemd[1]: rabbitmq-server.service: Supervising proce
root@elastilab:~#
```

Sender:

```
#!/usr/bin/env python
import pika
connection = pika.BlockingConnection(pika.ConnectionParameters(host='localhost'))
channel = connection.channel()
channel.queue_declare(queue='hello')
channel.basic_publish(exchange='', routing_key='hello', body='Hello World!')

print(" [x] Sent 'Hello World!'")

connection.close()
```

Receiver:

```
#!/usr/bin/env python
import pika
connection = pika.BlockingConnection(
    pika.ConnectionParameters(host='localhost'))
channel = connection.channel()
channel.queue_declare(queue='hello')
def callback(ch, method, properties, body):
    print(" [x] Received %r" % body)

#channel.basic_consume(queue='hello', on_message_callback=callback, auto_ack=True)
# With apt package python-pika 0.11.0-4, this works:
channel.basic_consume(callback, 'hello', no_ack=True)

print(' [*] Waiting for messages. To exit press CTRL+C')
channel.start_consuming()
```


Hints:
Running receiver would look like this:

```
kramse@elastilab:~/projects/rabbit$ ./recv.py
 [*] Waiting for messages. To exit press CTRL+C
 [x] Received 'Hello World!'
 [x] Received 'Hello World!'
 [x] Received 'Hello World!'
```

Solution:
When you have a running RabbitMQ with the Python programs working you are done.

Discussion:
RabbitMQ supports many libraries and languages, check out the list on: `https://www.rabbitmq.com/devtools.html`

What if you want to connect ActiveMQ and RabbitMQ?

Both suppoprt AMQP 1.0 - so this might be a way to support this.

