Flex4Apps Documentation

Release latest

Jan 13, 2020

Contents:

CHAPTER 1

Introduction

1.1 Management summary

Goal of this page is to provide a manual and scripts to

- Get an reliable open source infrastructure for your business / environment
- Reproducibly running in less than 1 hour
- At a operating cost of less than 20 € per month

1.2 About the Flex4Apps

The convergence of cloud, communication and IoT infrastructure plus the trend towards virtual applications (e.g. migrating software to the cloud) create new challenges for application developers and infrastructure providers. The resulting systems are complex with dynamic resources hiding possible problems. This creates a requirement for flexible monitoring and optimization methods. The Flex4Apps project addresses the challenges of monitoring and optimizing large, distributed, cyber-physical systems. The goal of the project is to provide a solution to manage the high data volumes and complexity of system monitoring whilst disturbing the target system as little as possible.

The demonstrator is setup using the f4a.me domain.

1.3 High level architecture

Goal of the project is not to worry about "hardware" while operating an infrastructure. Currently servers must be rented / purchased. They need to be maintained and in case something breaks, the operation of the overlaying service / system is impacted.

Docker swarm is one solution to achieve this abstraction layer. For this approach three or more (virtual) servers are required. RAM and CPU power is key, SSDs help to improve the performance.

Even though docker runs on many platforms we decided to operate based on an Ubuntu 16.04 LTS. Windows is also possible but has some drawbacks for the infrastructure we build. Secondly we look at Kubernetes.

Docker only achieves the independence with regards to the stateless execution of services like webservers, mail servers and other services. To provide persistent storage a file server / cluster and a database server is needed, which turns out to be the most difficult challenge.

Alternatively, a similar Flex4Apps based system can be deployed using serverless technologies such as AWS Lambda or Azure Cloud Functions. Such a deployment is to be considered if the cost of renting servers is too high or elasticity needs demand a different approach. While it does depend on the use case, please note that the resulting system will be a little less portable than the Docker architectures.

CHAPTER 2

Best practices and state of the art

2.1 Best practices

2.1.1 Log file analysis

The importance of logs

In today's software industry, monolithic architectures are more and more replaced by smaller, distributed services. Paradigms such as cloud computing and microservices are central to this development.

While small programs providing one or a few well defined services are easy to understand in isolation, real-life systems often comprise a multitude of such programs, resulting in a high level of complexity and possibly unforeseen problems. The hardest part of fixing issues in such a system often consists of reconstructing the exact interactions and conditions under which the issue occurred. Logging often is the only way by which this information can be made available. But log files do not only provide crucial information for problem-solving in distributed systems; they additionally are a data source from which valuable business insights can be obtained.

Modern large-scale web services often produce vast amounts of logs, which makes manual inspection and analysis unfeasible. Automated analysis techniques such as machine learning can be applied to solve this problem. Cheap storage to save the logs and cheap computing power to analyse them are both available.

In this use case, tools to analyse unstructured log files are developed with the goal of creating an automated anomaly detection system. In this context, *anomaly* is defined as any irregular behaviour of the system on which the logs are produced, including software bugs and hardware failures. The real-life log files analysed within Flex4Apps are unstructured and produced by software running on complex telecommunication infrastructure elements.

Log Templater - A tool for log conversion

Central to our log analysis is *Log Templater*, a tool we developed to convert unstructured logs into structured (terminology is explained below) ones. We intend to make this tool open source. A link to the code will be published here.

How to format logs

The aim of the following guidelines on how to log is to facilitate analysis of the logging data, with a focus on automated analysis methods

Write structured log messages. Unstructured logs usually have a format similar to the following:

```
[timestamp] [severity] [process_name] log_message
```

The log message itself consists of a simple, unstructured string, for example

```
Failed login for user f4a (invalid password)
```

An equivalent structured message in JSON format is

```
{"message type": "failed login", "parameters": {"user": "f4a",
"reason": "invalid password"}}
```

For a human, an unstructured message is easy to interpret, but automatically extracting information from it is problematic. In the above unstructured example, a log parser would need to detect `Failed login` as the category or type of the message, `f4a` as the user and `invalid password` as the reason for the failed login. The information that `invalid password` is a reason for the failed login is not even explicitly present in the message. Treating a log message as a set of key-value pairs ensures that for every value present in the message, there exists a corresponding key or identifier, thus parsing information for subsequent analysis is made easy.

An additional advantage of structured logs is that adding information to existing log statements in the source code will not break existing analysis scripts, since these scripts will rely on key-value pairs and not on the exact order of words in the message.

A central component developed within the log analysis part of Flex4Apps is a log converter which is able to identify parameters in unstructured log messages and subsequently convert these messages to structured messages. Today, this feature is essential to perform analyses on a large variety of logs, since currently, unstructured log messages are still the norm. Documentation for this converter is available here.

Use a single structured format throughout all parts of the application. Otherwise, parsing logs from your application will only get more complicated.

Provide event IDs. Each source code statement producing a log entry should print a unique log ID. This makes filtering for specific messages or events easy.

Ensure consistency of key names. For entities which exist at different locations in the log-producing system, such as users, hosts, or session IDs, use the same key name in different message types. Do not use `user` for `failed login` and `username` for `requesting password change`, but keep the key consistent by choosing `user` in both cases. Consistent keys allow for easy tracking of entities throughout the system by filtering for all log messages containing entries such as `{"user": "f4a"}`. A central dictionary of existing key names which is accessible and modifiable by all developers may help to ensure consistency.

Make it easy to distinguish between events happening at different points in the system, but at the same time. This can be done by adding entries for hosts, processes, threads, etc. to log messages.

Provide as much context as possible. Treat each log message as a message about an event. All context necessary to understand the event should be present in the log message. Long log messages with many parameters are not an issue if they are well-structured, allowing for easy filtering of information relevant to the current analysis. Logging the class and function in which the event took place may be helpful.

At the time of development, most likely you will not be able to think of all possible questions someone might want to answer with the help of the log files written by your application. Consequently, providing as much information as possible may prove helpful in the future.

Severity levels have limited use in complex systems. An upstream event may not be critical for the component in which it takes place, but may cause problems in a component further downstream. The downstream component

may not be known to the developer of the upstream component, so choosing appropriate severity levels for all situations is not possible.

Use fine-grained timestamps with time zones. A granularity of at least milliseconds ensures the correct order of events happening in fast succession can be reconstructed. Time zone information is essential in distributed systems where individual components may be located on different continents.

2.1.2 Analytics in cloud solutions and software as a service (SaaS)

Since the SaaS and Cloud paradigms leverage the Internet as a distribution channel, SaaS offers software builders the possibility to offer their products and service on a global scale. Most SaaS business models are based on a "pay-as-you-go" principle, whereby customers pay a monthly fee based upon their usage of the product (i.e. number of users, storage, ...). It is fairly well understood which high level KPIs matter most for SaaS businesses; most notably acquisition cost (the cost to attract a new customers), lifetime value (the total revenue generated per customer over the life span of the relationship with the customer) and month over month growth. It is considered a best practice to deploy dashboards to monitor these high level KPIs. A growing number of SaaS products become available offering such dashboards as a service.

The role of product management in a SaaS business is to design "a system" by combining technologies into a total UX so that business goals are achieved. In this context, UX involves all touch points between company and customers (i.e. marketing material, product itself, pricing and price plans, support,) and includes both product design and interaction design.

Designing a successful user experience for any product is a wicked problem. I.e. there are only better and worse solutions, and only by experimentation, iteration, monitoring and validation one can determine if one is moving towards a better solution or away from it. This in itself is nothing new: in the software world, agile software development popularized the idea of incremental iterations and timely feedback. The Lean Startup movement applied a similar reasoning to not just product development, but to the entire process of business development and go-to-market, while more recently the growth hacking community applies a similar reasoning to marketing.

While the statement designing a successful user experience is wicked problem is true for most products (hardware or software, B2C or B2B), online businesses, and thus Cloud/SaaS businesses have a clear advantage: they can iterate and experiment very fast: indeed, deploying a new version of the product is often just a matter of minutes, thanks to advances in DevOps. SaaS providers have the possibility to monitor and measure the impact of a change instantly, via monitoring and analytics, and treat their entire operations (i.e. product, service, user base, etc.) as "a living lab". Some well-known examples of companies doing this are Netflix.com, Google.com, or the Huffington Post with their A/B testing of headlines.

In a SaaS context, and in general, for companies that want to use the "too cheap to meter" strategy to deal with the software paradox, the discipline of Product Management is dramatically changing and broadening. A SaaS product manager is not only responsible for defining the product's features and how to support these, she has to incorporate valorization and growth strategies as well. She will leverage the possibilities of instant change (DevOps) and instant feedback (metrics and analytics) and install a living lab throughout the entire SaaS operation.

In order to be successful at installing this living lab, software product managers for SaaS products will want to use analytics and usage data to make informed product management decisions, as well as install feedback loops that potentially expand to all stakeholders so that the information collected via the analytics can be leveraged throughout the entire SaaS organization.

Analytics best practices

When talking about data-driven product and growth management (often in an online SaaS content), it is more common to talk about analytics than to talk about logging, although conceptually, these are similar, in the sense that instead of logging information about how the system is behaving, one logs information on how the user is behaving on the system. So many of the best practices described above, apply for this use case as well. Specifically naming conventions: the

naming of the events is important, once a specific user event (e.g. user presses the submit button of the "create project" form) is given a name, one shouldn't go lightly on changing the name of this event, since that might skew reporting later on.

Use structured formats (e.g. JSON, protobuffers). For analytics event reporting, JSON is the most widely used format to serialize and transport the data. JSON is easy to read and write from within JavaScript, the most used front end language for digital services. Virtually every programming language has extensive support to parse and generate JSON documents. Moreover, many database systems (both SQL and NoSQL based) support JSON as a data type and allow querying within JSON data structures.

Consider front end and back end analytics. Many analytics are gathered on the client side, and then sent back to the analytics back end This is logical and easiest, since it is with the client side (web app, mobile, ...) that the user interacts. The downside of this approach is that, depending on what analytics technology is used, the user might block communication between client and analytics back end (e.g. though the use of ad blockers, ad blocking proxies, etc...). When you want to be absolutely certain to track certain events within your analytics system, consider logging these events from the server side (a user has no control over what happens on the server side).

Minimal structure of an analytics log message At a minimum, each analytics event message should have:

- A user ID: this is a unique identifier for the given user. This can be the same ID as the user is known in the actual system, like a guid or an email address.
- A time stamp when the event happened, preferable in UTC.
- The name of the event.
- Optionally, one can provide additional meta data that might be relevant for the given event. Meta data could be details on the users client (browser, OS, ...), metrics (e.g. for an event "watch_movie", the meta data could contain which move and the percentage of the video the user watched) or any other info that seems relevant.

Consider sending event data in batches. Depending on how much analytics you gather (e.g. you only record major events in the app vs. you record every single click a user does), one might consider batching the event data and send over multiple events at once, every x seconds, instead of sending the events over as soon as they happen. Especially for mobile apps, consider the offline case, whereby analytics are batched and cached when the client is used offline.

Use analytics to build user profiles. A user profile for this case indicates, for every single user, the activity that user expressed with the application. Based upon that data, one can derive the type of user (e.g. a project manager vs a project contributor, ...) , how engaged he/she is with the service (power user vs. novice trial user), latest activity, etc. In it's simplest form, this user profile is a timeline of all the events the user did on the system.

Build dashboards/queries to follow up on major KPIs. Gathering the analytics data is one thing, putting them to good use is another one. At the very least, build some dashboards that illustrate the major KPIs for the SaaS business. At least have a dashboard that illustrates the evolution of new accounts/unit of time, churn rates over time, breakdown of feature usage, funnel metrics and engagement metrics.

Feed back analytics data to all stakeholders. Dashboards are one way of feeding back the data and information to stakeholders, but there are other possibilities as well:

- Connect the user profiles with support systems, so that, whenever support questions come in, the support agent has the context of the given user (e.g. engagement level, the plan the user is one, ...) at her fingertips and thus can give tailored support (microcare)
- Feed back the analytics (or parts thereof) to the end users: e.g. as reporting on how all users within an organization are using the SAAS. These can be in the form of reports, dashboards ...
- Sales and account management: having a clear insight on the usage patterns of a given customer/account, the sales or account management organization can discuss potential opportunities to up sell/cross sell.

Business best practices

- Discuss and identify a feedback loop. Product managers and stakeholders can use the data to focus on important topics but will have to consciously look at the data and interpret it to form good decisions.
- Make sure to invest time in generating reports, spreadsheets or other forms of data that is usable by business users who actually want the data. If the form is not accessible enough it will not be used.
- Design for flexibility. As soon as data is available, new questions will arise which potentially need extra data to be gathered.
- Think about security and Data Protection. GDPR can have a significant impact if you collect data that might be identifiable.
- Think about the lifetime of the analytics. Storing everything forever might be a good idea but more often than not the value decreases rapidly. Make sure to pass this to the technical process.

Technical best practices

- **In cloud solutions, stateful deployments need the most attention.**

  Before starting Flex4Apps, project partners believed that managing fail-over scenarios for high-availability installations is the part which is most difficult to handle.

  During learning about cluster deployment, we realized various solutions exist for stateless services. Tasks become complicated if services need persistence (so-called stateful services).

  Storage has to be provided as an extra high-availability service, which makes it cost intensive. An alternative approach is to look for storage software which inherits support for multi-node cluster deployment. This can mean that a vendor lock-in can't be avoided.

- **S3 can be a good light cloud storage.**

  There is no vendor lock-in, because various implementations at commercial clouds exist next to various open source implementations. S3 can be deployed on small single nodes or on clusters with redundancy for high availability. However, S3 is only suited for pure storage, not for complex querying.

  If S3 is not good enough, there is no one-size-fits-all solution out there right now. Most secure and usable solutions are bound to a cloud provider. Check costs and functions of storage and for other needed functions of the cloud provider.

- Be aware of the cost factors of the chosen solution and see where the dominant factors lie as volumes increase.
- Compress close at the source (and try and match optimal parameters) because transmission costs can otherwise be very high.
- Filter data (preferably dynamically) at the source to keep storage and analytics persistency costs down.
- Unpredictable traffic surges might overwhelm your end-points. Also make sure to keep in mind where your clients are sending the data. Don't reinvent the wheel. Sending directly to AWS Kinesis Firehose will handle any load but requires AWS credentials.
- Define if and what *real-time* means for your use case. If the real time can work asynchronously and tolerate delays you can design the system to be very cost efficient.
- Don't try and find the ideal storage solution as there is none that is fully cross platform and portable across all cloud vendors unless you sacrifice features (dropping down to standard SQL instead of NoSQL, Big Data or analytic systems). Mix and match storage solutions as you see fit.

2.1.3 Home automation

Measured values from the residential area of tenants or house owners are to be automatically made available in third-party software. This increases the comfort of use. The user receives more information as a basis for decision-making on his supply contracts and consumption behaviour.

To implement this connection, the user employs a gateway that collects the measured values and transmits them via a defined interface to a platform derived from Flex4Apps components.

Best practices

Prefer industry standard over individual. Use-case home automation is characterized by a multitude of manufacturer-specific and self-contained solutions. In the course of the project it became clear that trust in such an individual standard can be disappointed.

Only through the consistent use of open interfaces and basic products can the dependency on individual providers and their business models be avoided. This enables solutions which work better in a very volatile environment and which can be quickly adapted to new circumstances.

Consider the overhead caused by addressing and encryption. Limited bandwidth can limit use case. Standard encryption procedures produce a little data overhead. This can be too much depending on the available throughput to the used media.

This became relevant in the attempt to set up in-house communication on an existing proprietary system solution. While the transmission of user data was just possible, encryption of this data was too much for the medium.

When estimating the required transmission capacity, it is therefore essential to consider the overhead caused by addressing and encryption.

Use MQTT as central transport layer. MQTT is widely used and accepted as a vendor-neutral transmission protocol. Consequently, employing MQTT enables the quick integration of new data sources from third parties.

2.2 State of the art

Within the Flex4Apps project the consortium looked at the state of the art at the time of project instantiation. If of interest, the resulting document “state of the art” can be requested at one of the service providers.

CHAPTER 3

Getting started with Flex4Apps

there are multiple ways to get started with Flex4Apps. Below you find some hints how to kickstart your application / use case. Otherwise you can always contact one of the project partners to setup your own Flex4Apps infrastructure for you.

3.1 . . . using Docker Swarm

you can start working with your own Docker Swarm quite quickly. Either execute the line below, or if you are skeptical about what the script does (and you usually should!):

- download the script manually and evaluate it for security risks
- then execute it if you feel comfortable or
- repeat the steps manually as they are documented in the script.

If the execution of the script fails - you might either have to specify your network interface or provide the public IP manually.

```
wget -O - https://raw.githubusercontent.com/Flex4Apps/flex4apps/master/src/docker/
↪setupBareMetal.sh | bash
```

3.2 . . . using Kubernetes

Quick start guide based on a Kubernetes setup is show at

3.2.1 Getting started with Kubernetes and Flex4Apps

This repository hosts example configuration files for the Getting started guide.

Goal

- show first setup to get in touch with toolbox
- use common building blocks as example
  - ElasticSearch
  - Kibana
  - Logstah
  - Mosquito
- use Kubernetes / Rancher as base ground

Documentation describe to deployment steps for a first setup. Some components has to be manually connected with each other depending on concrete use case.

The solution stores and analyses transport metadata from message broker mosquito to detect abnormal situations.

Requirements

System

- virtual machine or bare metal server
- 24 GB RAM, 200 GB space left on storage
- OS Debian 9 minimal

Skills

You should be familiar with kubernetes stateless and stateful sets. Helm charts will be used for deployment.

Architecture

Flex4Apps stack is divided in

- physical parts
- cloud part

The stack itself mainly covers the cloud part. From the physical side it receives data via API. In this example a MQTT message broker is used for connections from outside.

Inside the cloud part of the stack data distinguish in

- payload
- transport metadata

The processing of payload is application specific. Therefore the stack mainly covers the use and analysis of metadata. Transport metadata produced at system border - mosquitto message broker in this example. They will delivered to an logstash and persisted at an elastic search instance.

To analyse metadata for anomalies Kibana is used. This tool provides visualization and alerting.

Deployments

Set up Rancher / Kubernetes cluster (optional / unfertig)

Given a bare metal physical or virtual server the following steps needed to set up a first Flex4Apps stack.

Docker

Install docker according to Debian manual

Create file `/etc/docker/deamon.json` with following content

```
{
  "storage-driver": "overlay2",
  "iptables": false,
  "log-driver": "json-file",
  "log-opts": {
    "max-size": "25m", "max-file": "4"
  }
}
```

Iptables

Firewall rules has to be set manually.

```
sudo apt-get install iptables-persistent
```

If asked, save the existing rules for IPv4 and IPv6 and rename these files.

```
mv /etc/iptables/rules.v4 /etc/iptables/rules.v4.orginal
mv /etc/iptables/rules.v6 /etc/iptables/rules.v6.orginal
```

Helm / Tiller

running Rancher2 / Kubernetes Setup according to Quick-start-guide

Deploy core stack

Required:

- running Kubernetes Cluster
- installed Helm / Tiller

For this example it is assumed that context for the cluster is named 'f4a' on the local machine. Furthermore installation is demonstrated on a single node within a cluster. Different deployments styles need modifications at target hosts and some labels.

Preparation and adaption to individual environment

1. identify the host name (value of label kubernetes.io/hostname) of your target host. This value is called hostname later one.

2. set your DNS to this host an give a domain name, for example f4a.company.com. Keep attention to set also all subdomains to this main domain host. Add the following lines to your domain name server and adapt the IP. Note that there can be a difference between hostname and application domain name.

```
...
*.f4a                          IN CNAME  f4a
f4a                            IN A      192.168.100.1
...
```

3. Clone the Flex4Apps repository to local directory with

```
git clone https://github.com/Flex4Apps/flex4apps.git
```

4. in `/src/kuberentes/values.yaml` you have to adopt some values to your local environment; change ALL the default passwords and see comments in file
5. adapt `/src/kuberentes/templates/ssl.yaml` and set your :index:SSL certification data

Rollout

At Cluster all data will stored locally at `/data/{namespace}`. Namespace will be set at the next steps.

If everything is checked within config files, helm can be used to rollout the entire stack to your kubernetes cluster.

```
cd /src/kuberentes/
# check for syntax
helm upgrade --install --namespace --dry-run f4a  .
# do it for real
helm upgrade --install --namespace  f4a  .
```

After rollout some URLs are available:

- https://kibana.hostname.tld
- https://cerebro.hostname.tld
- https://hostname.tld/elasticsearch
- https://hostname.tld/grafana

ElasticSearch

ElasticSearch (ES) holds data produced by tracing interface of Flex4Apps mosquitto broker.

ElasticSearch can deployed in more than one node. In this example only one node is used.

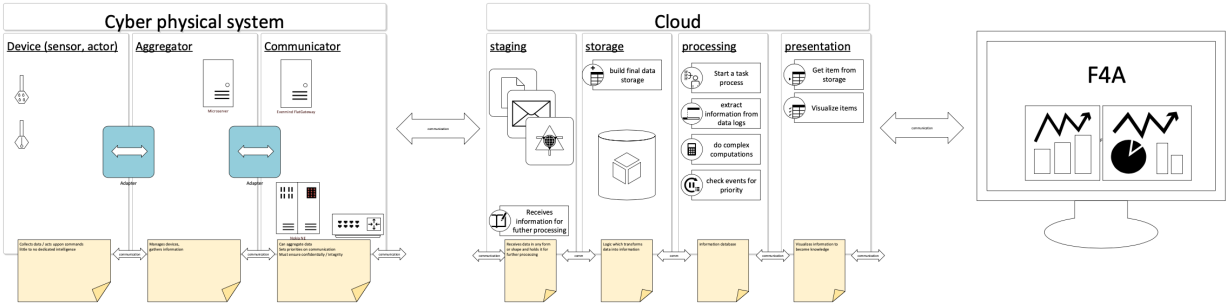
Deployment can be done by standard helm charts.

CHAPTER 4

Architecture

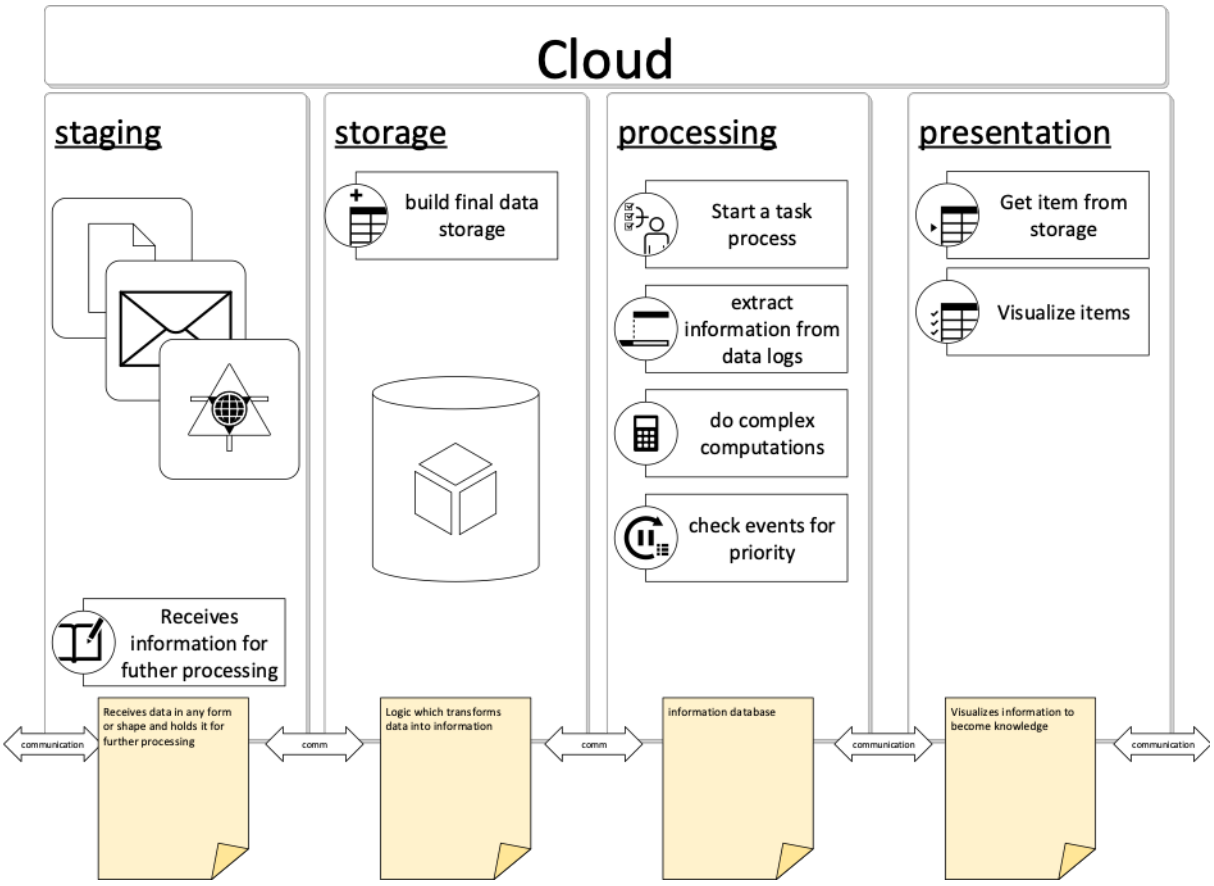
The Flex4Apps framework consists of several components that are described in the following section. For the sake of clarity, however, it is worth dividing it into two main components:

1. the Cyber-Physical-Systems - hardware, paired with software, which stands locally. This can range from a simple temperature sensor to a highly complex network communication device.
2. The Cloud System - a software solution that receives, processes, stores and displays data collected by Cyber-Physical Systems.



4.1 Cloud architecture

The Cloud architecture describes the software components of the Flex4Apps framework. The four main components are described below



4.1.1 Cloud staging

The Cloud Staging component takes care of the receipt and the cloud storage of data delivered by the CPS side. There are no restrictions on the Flex4Apps framework for file formats and communication protocols.

Interfaces The standards defined in the OSI model level 4 to 7 are used here. Data can thus be transmitted via HTTP / S, FTP / S, SMTP, SNMP, TCP, UDP.

Security The security of the cloud staging can be done at different levels. If the data has already been encrypted at the application level, an unsecured transmission level can be selected.

As a transition between Cloud and CPS side, a special emphasis is placed on security.

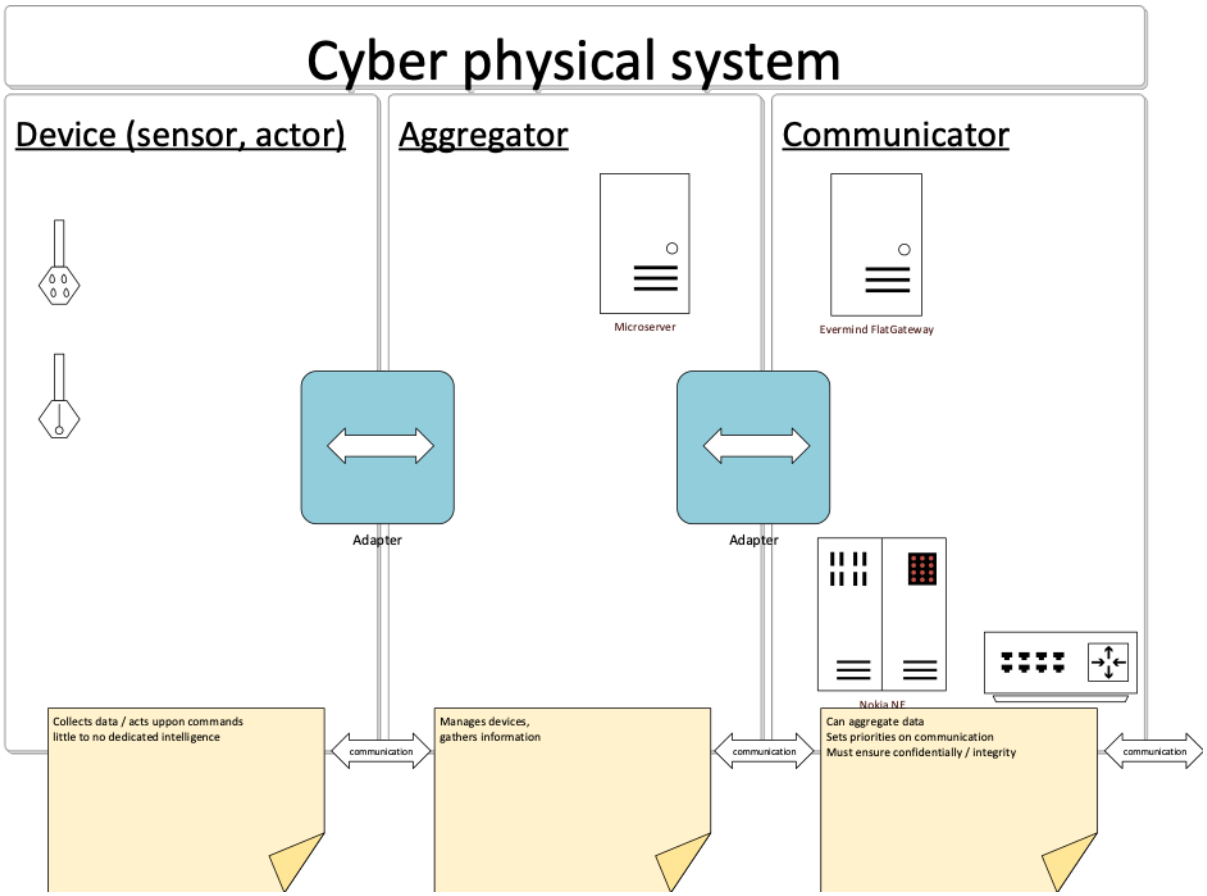
4.1.2 Cloud storage

The cloud storage component processes the data from the cloud staging area and transfers it to a database model according to standardized calculations. First clean-up activities are already carried out, which are characterized by regional standards. Here, above all, a uniform date, decimal & number groupings must be mentioned. The storage component also considers indexing requirements

4.1.3 Cloud processing

The cloud processing transforms the prepared data into quickly evaluable information. Based on a supplementary rule set, data are transformed, calculated, adjusted and processed.

4.2 Cyber-Physical Architecture



4.2.1 Device

The simplest "device" within the Flex4Apps framework is the device. It can collect data and receive / execute commands. A separate decision logic can be present but is not assumed. The task of the device is to ensure the communication to the next higher communication stage within the CPS constellation. The Flex4Apps framework also allows the device to communicate directly with the cloud.

Interfaces Are not considered within the framework of the Flex4Apps project. Provided communication with the higher-level cloud systems is ensured.

Security The communication is protected according to the environment, this can be controlled by physical access restrictions, secure transport channels as well as the use of encryption technologies.

4.2.2 Aggregator

The "Aggregator" is an administering instance for several "devices". It assumes the conditional control logic for connected devices.

Interfaces It provides a standardized interface (here REST) for controlling the assigned devices.

Security Communication to and from the aggregator runs according to defined security levels.

4.2.3 Communicator

The "Communicator" is the interface between the CPS world and the cloud system. The management of several assigned "aggregators" and "devices" takes place at the top level. A standardized interface (here REST) enables the detailed control of the connected system landscape.

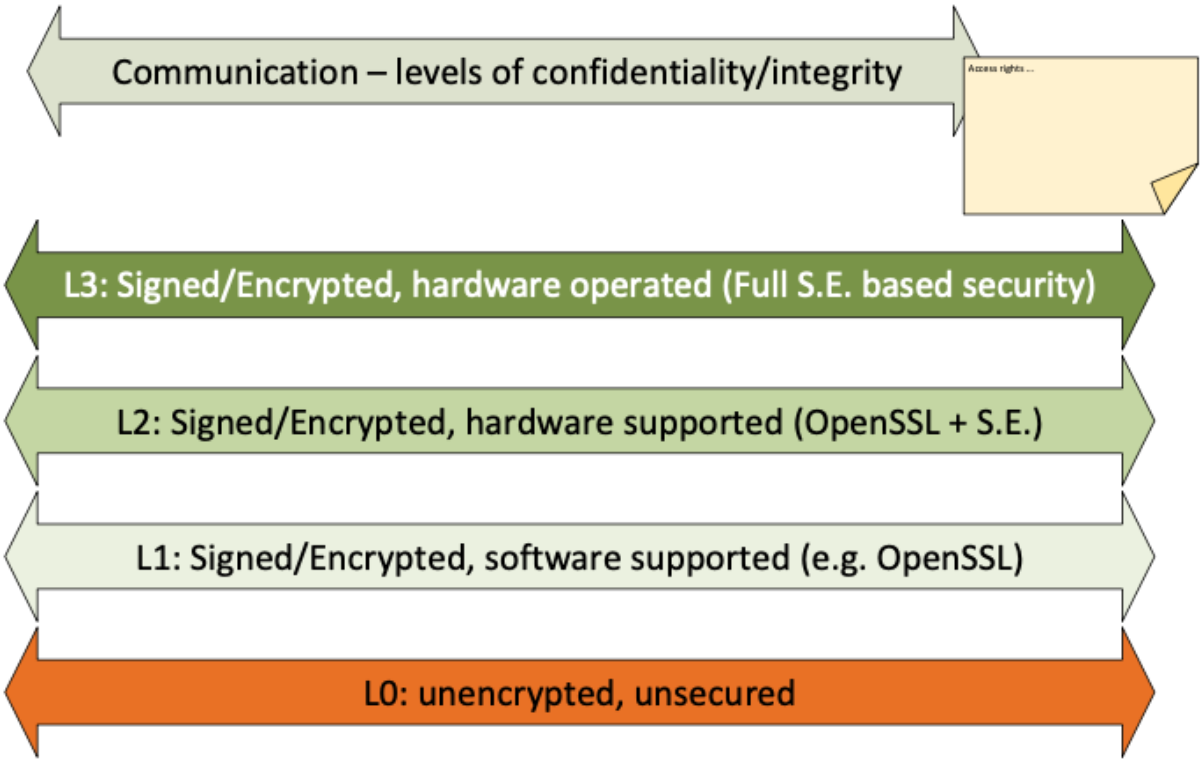
Interfaces Rest, Syslog – Other standards that can be received by Cloud Staging.

Security The communicator must be able to act as plug & trust communication bridge between the the CPS side and the cloud side

The Flex4Apps architecture allows the use of "custom business logic" to illustrate specific actions that are not solved by the Flex4Apps framework itself.

4.3 Security levels

Various security levels should be considered when implementing the Flex4Apps architecture.



CHAPTER 5

Docker

5.1 What is Docker

5.1.1 Step 1 - setting up the servers

Nowadays the servers are usually preinstalled or an installation process can be kicked off via web interface. For the F4A use case we chose Ubuntu 16.04 LTS (Long term support).

First we should ensure that the system is up-to-date and secure. This is done by kicking off the advanced packaging tool (apt). Within this process we can directly install the docker server component. All steps are done by issuing the following command:

```
apt-get update && apt-get upgrade -y && apt install -y docker.io
```

As docker is still being developed, certain functionality still changes. This tutorial has been created using the following docker version (you can find our yours by executing `docker version`):

```
root@nxp100:~# docker version
Client:
 Version:      1.13.1
 API version:  1.26
 Go version:   go1.6.2
 Git commit:   092cba3
 Built:        Thu Nov  2 20:40:23 2017
 OS/Arch:      linux/amd64

Server:
 Version:      1.13.1
 API version:  1.26 (minimum version 1.12)
 Go version:   go1.6.2
 Git commit:   092cba3
 Built:        Thu Nov  2 20:40:23 2017
 OS/Arch:      linux/amd64
```

(continues on next page)

(continued from previous page)

```
 Experimental: false
root@nxp100:~#
```

5.1.2 Step 2 - initiate a swarm

Setting up the docker swarm. A swarm is a group of computers:

```
docker swarm init --advertise-addr 89.144.27.100
```

getting the feedback:

```
Swarm initialized: current node (r3t3pu7rd74njml1afsf2uoev) is now a manager.

To add a worker to this swarm, run the following command:

    docker swarm join \
    --token <some secret token displayed here> \
    89.144.27.100:2377

To add a manager to this swarm, run 'docker swarm join-token manager' and follow the␣
↪instructions.
```

If you don't remember the token etc. - just run:

```
docker swarm join-token worker
```

5.1.3 Step 3 - preparing the domain

register any domain you like. Just make sure that the domain names are pointing to all the server IPs you have. With that load balancing / failover is possible:

```
f4a.me.           86400 IN  SOA nsa3.schlundtech.de. mail.tillwitt.de. 2017112808␣
↪43200 7200 1209600 86400
f4a.me.           86400 IN  NS  nsa3.schlundtech.de.
f4a.me.           86400 IN  NS  nsb3.schlundtech.de.
f4a.me.           86400 IN  NS  nsc3.schlundtech.de.
f4a.me.           86400 IN  NS  nsd3.schlundtech.de.
f4a.me.           600   IN  MX  10 mail.f4a.me.
*.f4a.me.         600   IN  A   89.144.24.15
*.f4a.me.         600   IN  A   89.144.27.100
*.f4a.me.         600   IN  A   89.144.27.101
*.f4a.me.         600   IN  A   89.144.27.102
*.f4a.me.         600   IN  A   89.144.27.103
nxp100.f4a.me.    600   IN  A   89.144.27.100
nxp101.f4a.me.    600   IN  A   89.144.27.101
nxp102.f4a.me.    600   IN  A   89.144.27.102
nxp103.f4a.me.    600   IN  A   89.144.27.103
nxp104.f4a.me.    600   IN  A   89.144.24.15
```

5.2 Docker commands

5.2.1 docker run vs docker compose

Advantages of docker run are that the command is easy to issue, just a copy & paste to the servers command line. Downside is, that the commands get quite long and adding line breaks introduces another possible fault. If you want to correct a running service you need to remove it first and then reissue it.

Advantages of using a docker-compose.yml is that they are usually easy to edit. Disadvantage is that you have to create them on the server first then issue the command to start them - so one additional step. But the biggest advantage is that they can be re-executed on existing services which will lead to a service update.

5.2.2 Examples

starting a generic web application with docker run

```
docker service create \
    --name demo \
    --label "traefik.port=80" \
    --network traefik-net \
    kitematic/hello-world-nginx
```

Thats all - and the service is running.

To create the same via `docker-compose.yml`

```
version: "3"

services:
  nginx:
    image: kitematic/hello-world-nginx
    networks:
      - traefik-net
    deploy:
      labels:
        - traefik.port=80
        - "traefik.frontend.rule=Host:demo.f4a.me"
networks:
  traefik-net:
   external: true
```

Then you need to issue the following command

```
docker stack deploy --compose-file docker-compose.yml demo
```

5.2.3 Conclusion

To quickly test a service - docker run is nice. But to maintain a production environment docker-compose files are strongly recommended.

5.3 Docker registry

Running your own registry

```
docker service create \
   --name backoffice \
   --network traefik-net \
   --label "traefik.port=5000" \
   --label 'traefik.frontend.auth.basic=flex4apps:$apr1$G9e4rgPu$jbn2AAk2F.OeGnRVFnIR/
↪1' \
   --mount type=bind,src=/swarm/volumes/registry,dst=/var/lib/registry \
   registry:2
```

5.3.1 Pushing to private registry

The local image needs to be tagged and then pushed

```
docker tag phabricator_image registry.f4a.me/phabricator
docker push registry.f4a.me/phabricator
```

Run that image

```
docker service create \
  --name demo \
  --label "traefik.port=80" \
  -e "GITURL=https://secret@gogs.tillwitt.de/NXP/homomorphic-encryption-demo.git" \
  flex4apps:GQfgCEsjkHC7LRf3Q9PkW4L6onDLtu@backoffice.f4a.me/homomorphic_img
```

5.3.2 Query the registry

Get the overview of all images:

```
https://registry.f4a.me/v2/_catalog
```

Get all tags of an image:

```
https://registry.f4a.me/v2/henc/tags/list
```

5.3.3 Private repository viewer

In the mean time there are good solutions to provide a secure, private and selfhosted docker registry like.

- https://goharbor.io/
- http://port.us.org/

5.4 Applications

5.4.1 What are applications

On the created system applications can be launched. Templates for applications are called images. Running applications are called containers.

5.4.2 Generic demonstrator

Simple demo

The following application is already working with the current setup.

To launch an easy demonstrator, lets instantiate a webserver and make it available at demo.f4a.net:

```
docker service create \
    --name demo \
    --label "traefik.port=80" \
    --network traefik-net \
    kitematic/hello-world-nginx
```

Generating a new user with password run:

```
htpasswd -nbm flex4apps password
```

or go to: http://www.htaccesstools.com/htpasswd-generator/

The output will be something like:

```
flex4apps:$apr1$XqnUcSgR$39wlPxxyyxPxXZjFb34wo.
```

Example for traefik label usage below. If single quotes are in the password they would need to be escaped.

To do that close the quoting before it, insert the escaped single quote, and re-open the quoting: `` `'first part'\''second part'` ``

How to start the demo service:

```
docker service create \
    --name demopw \
    --label "traefik.port=80" \
    --label 'traefik.frontend.auth.basic=myName:$apr1$a7R637Ua$TvXp8/lgky5MDLGLacI1e1
↪' \
    --network traefik-net \
    kitematic/hello-world-nginx
```

Docker compose

```
# PULL UPDATE & LAUNCH DOCKER
#     git pull && docker stack deploy --compose-file docker-compose.yml demo

# REMOVE STACK
#     docker stack rm demo

# ADD BASIC AUTH and ESCAPE FOR docker-compose usage
# htpasswd -bBn user password | sed 's/\$/\$\$/g' #escape for docker-compose usage

version: "3"

services:
  nginx:
    image: kitematic/hello-world-nginx
    networks:
```

(continues on next page)

(continued from previous page)

```
      - traefik-net
    environment:
      - test=noContent
    deploy:
      labels:
        - traefik.port=80
#         - "traefik.frontend.auth.basic=witt:$$2y$$05$
↪$kOFY7071ilbnpiJNDaIO9e1WeuhHnKtp9Adrevz4r8wJ3b3X1XuqW"
#         - "traefik.frontend.auth.basic=ich:$$2y$$05$$jTZv0re2cXmiGrzRxW./8Ofse.6g/
↪AEChvbMGdqYKIMqsr8xW/c"
#         - "traefik.frontend.auth.basic=user:$$2y$$05$$IRrTxLpG7ICzroI8Pb5P4.
↪p2rMXGqyeeZM857BJxTFzP5q9W4RYuS"
        - "traefik.frontend.rule=Host:demo.f4a.me"
networks:
  traefik-net:
   external: true
```

5.4.3 grafana

Grafana is an open source software for time series analytics

create the service like this:

```
docker service create \
  --name=grafana \
  --network traefik-net \
  --label "traefik.port=3000" \
  --mount type=bind,src=/swarm/volumes/grafana,dst=/var/lib/grafana \
  -e "GF_SECURITY_ADMIN_PASSWORD=someSecretPassword" \
  grafana/grafana
```

Below you find the docker-compose for grafana

```
# Flex4Apps documentation

# PULL UPDATE & LAUNCH DOCKER
#     git pull && docker stack deploy --compose-file docker-compose.yml grafana

# REMOVE STACK
#     docker stack rm grafana

# ADD BASIC AUTH and ESCAPE FOR docker-compose usage
# htpasswd -bBn user password | sed 's/\$/\$\$/g' #escape for docker-compose usage

version: "3.1"

services:
  server:
    image:   grafana/grafana
    volumes:
      -  /swarm/volumes/grafana:/var/lib/grafana
    networks:
      - traefik-net
    environment:
```

(continues on next page)

(continued from previous page)

```
      - "GF_SECURITY_ADMIN_PASSWORD=someSecretPassword"
    deploy:
      labels:
        - traefik.port=3000
        - "traefik.frontend.rule=Host:grafana.f4a.me"
      replicas: 1
#      update_config:
#        parallelism: 1
#        delay: 30s
#      placement:
#        constraints: [node.role == manager]

networks:
  traefik-net:
   external: true
```

5.4.4 Prometheus

Prometheus is an open-source software application used for event monitoring and alerting. It records real-time metrics in a time series database (allowing for high dimensionality) built using a HTTP pull model, with flexible queries and real-time alerting. The project is written in Go and licensed under the Apache 2 License, with source code available on GitHub, and is a graduated project of the Cloud Native Computing Foundation, along with Kubernetes and Envoy.

Below you find the reference code from the repository

```
# Flex4Apps documentation

# PULL UPDATE & LAUNCH DOCKER
#     git pull && docker stack deploy --with-registry-auth --compose-file docker-
↪compose.yml prom

# REMOVE STACK
#     docker stack rm prom

# ADD BASIC AUTH and ESCAPE FOR docker-compose usage
# htpasswd -bBn user password | sed 's/\$/\$\$/g' #escape for docker-compose usage

version: "3.1"

services:
  server:
    image: prom/prometheus
    volumes:
      -  /swarm/volumes/prometheus/data:/prometheus
      -  ./prometheus.yml:/etc/prometheus/prometheus.yml
    networks:
      - traefik-net
    deploy:
      labels:
        - traefik.port=9090
        - "traefik.frontend.auth.basic=witt:$$2y$$05$
↪$kOFY7071ilbnpiJNDaIO9e1WeuhHnKtp9Adrevz4r8wJ3b3X1XuqW"
        - "traefik.frontend.rule=Host:prometheus.f4a.me"
```

(continues on next page)

(continued from previous page)

```
      replicas: 1
      update_config:
        parallelism: 1
        delay: 30s
      placement:
        constraints: [node.role == manager]

  nodeexporter:
    image: prom/node-exporter:latest
    volumes:
      - /proc:/host/proc:ro \
      - /sys:/host/sys:ro \
      - /:/rootfs:ro \
    networks:
      - traefik-net
    ports:
      - 9100:9100
    deploy:
      mode: global
      labels:
        - com.group="prom-monitoring"

    command:
      - '--path.procfs=/host/proc'
      - '--path.sysfs=/host/sys'
      - --collector.filesystem.ignored-mount-points
      - "^/(sys|proc|dev|host|etc|rootfs/var/lib/docker/containers|rootfs/var/lib/
↪docker/overlay2|rootfs/run/docker/netns|rootfs/var/lib/docker/aufs)($$|/)"

networks:
  traefik-net:
   external: true
```

5.4.5 portainer

start portainer as a service we first need to create a data directory:

```
mkdir -p /docker/portainer
```

To start the container itself:

```
docker service create \
--name "portainer" \
--constraint 'node.role == manager' \
--network "traefik-net" --replicas "1" \
--mount type=bind,src=/var/run/docker.sock,dst=/var/run/docker.sock \
--mount type=bind,src=/docker/portainer,dst=/data \
--label "traefik.frontend.rule=Host:portainer.f4a.me" \
--label "traefik.backend=tool-portainer" \
--label "traefik.port=9000" \
--label "traefik.docker.network=traefik-net" \
--reserve-memory "20M" --limit-memory "40M" \
--restart-condition "any" --restart-max-attempts "55" \
```

(continues on next page)

(continued from previous page)

```
--update-delay "5s" --update-parallelism "1" \
portainer/portainer \
-H unix:///var/run/docker.sock
```

Below you find the reference code from the repository for portainer.

```
# PULL UPDATE & LAUNCH DOCKER
#     git pull && docker stack deploy --compose-file docker-compose.yml portainer

# REMOVE STACK
#     docker stack rm mariadb

# ADD BASIC AUTH and ESCAPE FOR docker-compose usage
# htpasswd -bBn user password | sed 's/\$/\$\$/g' #escape for docker-compose usage

version: "3"

services:
  server:
    image: portainer/portainer
    networks:
      traefik-net:
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock
      - /data/local/portainer:/data
    environment:
      - MYSQL_ROOT_PASSWORD=someSecretPassword
    deploy:
      labels:
        - traefik.port=9000
        - "traefik.frontend.rule=Host:portainer.f4a.me"
      placement:
        constraints: [node.role == manager]

networks:
  default:
  traefik-net:
   external: true
```

5.4.6 Phabricator

The database user needs to be able to create databases

1. change your password:

```
sed -i 's/<some secret>/yourPassword/g' Dockerfile
```

1. build the image with:

```
docker build -t phabricator_image .
```

2. tag and push it to the registry:

```
docker tag phabricator_image registry.f4a.me/phabricator
docker push registry.f4a.me/phabricator
```

3. deploy it from the registry by executing the docker-compose:

```
docker-compose up
```

Below you find the reference code from the repository for phabricator. This is not completely working at the end of the project

```
# PULL UPDATE & LAUNCH DOCKER
#     git pull && docker stack deploy --with-registry-auth --compose-file docker-
↪compose.yml phabricator

# REMOVE STACK
#     docker stack rm phabricator

# ADD BASIC AUTH and ESCAPE FOR docker-compose usage
# htpasswd -bBn user password | sed 's/\$/\$\$/g' #escape for docker-compose usage

version: "3.1"


services:
   web:
     image: registry.f4a.me/phabricator
     networks:
       - traefik-net
#     ports:
#       - "80:80"
     deploy:
      labels:
        - traefik.port=80
        - "traefik.frontend.rule=Host:phabricator.f4a.me"

#   db:
#     image: mysql:5.7
#     volumes:
#       - ./data:/var/lib/mysql
#     restart: always
#     environment:
#       MYSQL_ROOT_PASSWORD: root
#       MYSQL_DATABASE: phabricator
#       MYSQL_USER: phabricator
#       MYSQL_PASSWORD: phabricator

networks:
  traefik-net:
   external: true
```

5.4.7 MariaDB

MariaDB is free and open source relational database system. It was created as a :fork: from MySQL after Oracle started releasing new functionality not as open source anymore and due to the high support cost of MySQL.

As usual make sure that the path for data volume exists:

```
mkdir -p /swarm/volumes/mariadb
```

The initiate the docker service:

```
docker service create \
      --name mariadb \
      --publish 3306:3306 \
      --network traefik-net \
      --mount type=bind,src=/swarm/volumes/mariadb,dst=/var/lib/mysq \
      --label "traefik.port=3306" \
      -e MYSQL_ROOT_PASSWORD=someSecretPassword \
      mariadb:latest
```

Below you find the reference code from the repository for mariadb

```
# PULL UPDATE & LAUNCH DOCKER
#     git pull && docker stack deploy --compose-file docker-compose.yml mariadb

# REMOVE STACK
#     docker stack rm mariadb

# ADD BASIC AUTH and ESCAPE FOR docker-compose usage
# htpasswd -bBn user password | sed 's/\$/\$\$/g' #escape for docker-compose usage

version: "3"

services:
  server:
    image: mariadb:latest
    networks:
      traefik-net:
    volumes:
      - /data/shared/mariadb=/var/lib/mysq
    environment:
      - MYSQL_ROOT_PASSWORD=someSecretPassword
    deploy:
      labels:
        - traefik.port=3306
        - "traefik.frontend.rule=Host:mariadb.f4a.me"

  phpmyadmin:
    image: nazarpc/phpmyadmin
    networks:
      traefik-net:
    environment:
      - MYSQL_HOST=mariadb_server:3306
    deploy:
      labels:
        - traefik.port=80
        - "traefik.frontend.rule=Host:phpmyadmin.f4a.me"

networks:
  default:
  traefik-net:
   external: true
```

5.4.8 PhpMyAdmin

phpMyAdmin is a free and open source administration tool for MySQL and MariaDB. As a portable web application written primarily in PHP, it has become one of the most popular MySQL administration tools, especially for web hosting services.

The following command will start up PhpMyAdmin:

```
docker service create \
    --name phpmyadmin \
    --label "traefik.port=80" \
    --network traefik-net \
    -e ALLOW_ARBITRARY=1 \
    nazarpc/phpmyadmin
```

Below you find the reference code from the repository for phpmyadmin.

5.4.9 gogs

Gogs is a painless self-hosted Git service.

Pull image from Docker Hub.

very strange installation. First need to use –publish 3000:3000 and connect direct for install. Then remove instance and also remove published port. This is certainly something I need to review.

create the data volume for gogs:

```
mkdir -p /swarm/volumes/gogs
```

start the service:

```
docker service create \
    --name gogs \
    --mount type=bind,src=/swarm/volumes/gogs,dst=/data \
    --label "traefik.port=3000" \
    --network traefik-net \
    gogs/gogs
```

We now would suggest to rather look into Gitea

Gitea is a community managed lightweight code hosting solution written in Go. It published under the MIT license.

Below you find the docker-compose for gogs

Below you find the docker-compose for gitea

```
# PULL UPDATE & LAUNCH DOCKER
#     git pull && docker stack deploy --compose-file docker-compose.yml gitea

# REMOVE STACK
#     docker stack rm gitea

# ADD BASIC AUTH and ESCAPE FOR docker-compose usage
# htpasswd -bBn user password | sed 's/\$/\$\$/g' #escape for docker-compose usage

version: "3"
```

(continues on next page)

(continued from previous page)

```
services:
  server:
    image: gitea/gitea
    networks:
      - traefik-net
    volumes:
      - /swarm/volumes/gitea:/data
    deploy:
      labels:
        - traefik.port=3000
        - "traefik.frontend.rule=Host:gitea.f4a.me"
      replicas: 1
      update_config:
        parallelism: 1
        delay: 30s
networks:
  traefik-net:
   external: true
```

5.4.10 Drone

A continuous integration server which is open source, and tightly integrates with open source git platforms like gogs or services like github.

A good installation procedure is available here at http://docs.drone.io/install-for-gogs/. The corresponding commands for F4A are below:

```
docker run \
  --name drone \
  --label "traefik.port=8000" \
  --publish 8000:8000 \
  --publish 9000:9000 \
  -e DRONE_OPEN=true \
  -e DRONE_HOST=drone.f4a.me \
  -e DRONE_GOGS=true \
  -e DRONE_GOGS_URL=https://gogs.tillwitt.de \
  -e DRONE_SECRET=<some secret> \
  drone/drone:0.8

mkdir -p /swarm/volumes/drone

docker service create \
  --name drone \
  --label "traefik.port=8000" \
  --label "traefik.docker.network=traefik-net" \
  --network traefik-net \
  --mount type=bind,src=/swarm/volumes/drone,dst=/var/lib/drone/ \
  --publish 8000:8000 \
  --publish 9000:9000 \
  -e DRONE_OPEN=true \
  -e DRONE_HOST=drone.f4a.me \
  -e DRONE_GOGS=true \
```

(continues on next page)

(continued from previous page)

```
  -e DRONE_GOGS_URL=https://gogs.tillwitt.de \
  -e DRONE_SECRET=<some secret> \
  -e DRONE_ADMIN=witt \
  drone/drone:0.8

docker service create \
  --name drone_agent \
  --mount type=bind,src=/var/run/docker.sock,dst=/var/run/docker.sock \
  --network traefik-net \
  -e DRONE_SERVER=drone:9000 \
  -e DRONE_SECRET=<some secret> \
  drone/agent:0.8
```

Once setup, with in this case gogs, you can log into the web interface. After a short sync all repositories should be visible. Activate drone.io for the corresponding repository.

To tell drone.io what to execute you need to add a `.drone.yml` to your repository. Examples are below.

example:

```
image: dockerfile/nginx
script:
  - echo hello world

  publish:
    docker:
      registry: registry.f4a.me
      email: witt@f4a.me
      repo: registry.f4a.me/flex4apps/flex4apps/homomorphic-encryption
      file: homomorphic-encryption/Dockerfile
      context: homomorphic-encryption
      tag: latest
      secrets: [ docker_username, docker_password ]
```

5.4.11 elasticsearch and kibana

elasticsearch is a search engine based on the Lucene library. It provides a distributed, multitenant-capable full-text search engine with an HTTP web interface and schema-free JSON documents. Elasticsearch is developed in Java.

Inspired by https://sematext.com/blog/docker-elasticsearch-swarm/. Issue the following command:

```
docker service create \
   --name esc24 \
   --label "traefik.port=9200" \
   --label 'traefik.frontend.auth.basic=flex4apps:$apr1$G9e4rgPu$jbn2AAk2F.OeGnRVFnIR/
↪1' \
   --network traefik-net \
   --replicas 3 \
   --endpoint-mode dnsrr \
   --update-parallelism 1 \
   --update-delay 60s \
   --mount type=volume,source=esc24,target=/data \
 elasticsearch:2.4 \
   elasticsearch \
   -Des.discovery.zen.ping.multicast.enabled=false \
   -Des.discovery.zen.ping.unicast.hosts=esc24 \
```

(continues on next page)

(continued from previous page)

```
   -Des.gateway.expected_nodes=3 \
   -Des.discovery.zen.minimum_master_nodes=2 \
   -Des.gateway.recover_after_nodes=2 \
   -Des.network.bind=_eth0:ipv4_
```

Inspired by https://github.com/elastic/elasticsearch-docker/issues/91 and https://idle.run/elasticsearch-cluster

The host systems have to be prepared to run elasticsearch in a docker:

```
echo vm.max_map_count=262144 >> /etc/sysctl.conf && sysctl --system && sysctl vm.max_
↪map_count
```

The issue the following command to start three instances of elasticsearch:

```
docker service create \
  --replicas 3 \
  --name esc56 \
  --label "traefik.port=9200" \
  --label 'traefik.frontend.auth.basic=flex4apps:$apr1$G9e4rgPu$jbn2AAk2F.OeGnRVFnIR/1
↪' \
  --mount type=volume,source=esc56,target=/data \
  --network traefik-net \
  elasticsearch:5.6.4 bash -c 'ip addr && IP=$(ip addr | awk -F"[ /]*" "/inet .*\/24/
↪{print \$3}") && \
      echo publish_host=$IP && \
      exec /docker-entrypoint.sh -Enetwork.bind_host=0.0.0.0 -Enetwork.publish_host=
↪$IP -Ediscovery.zen.minimum_master_nodes=2 -Ediscovery.zen.ping.unicast.hosts=tasks.
↪esc56'
```

Below you find the reference code from the repository for the ELK stack version 2.4

```
# PULL UPDATE & LAUNCH DOCKER
#     git pull && docker stack deploy --compose-file docker-compose.yml drone

# REMOVE STACK
#     docker stack rm drone

# ADD BASIC AUTH and ESCAPE FOR docker-compose usage
# htpasswd -bBn user password | sed 's/\$/\$\$/g' #escape for docker-compose usage

version: "3"

services:
  server:
    image: drone/drone:0.8
    volumes:
      - /swarm/volumes/drone:/var/lib/drone/
    environment:
      - DRONE_OPEN=true
      - DRONE_HOST=drone.f4a.me
      - DRONE_GOGS=true
      - DRONE_GOGS_URL=https://gogs.tillwitt.de
      - DRONE_SECRET=<some secret password>
      - DRONE_ADMIN=witt
    deploy:
      labels:
        - traefik.port=8000
```

(continues on next page)

(continued from previous page)

```
        - "traefik.frontend.rule=Host:drone.f4a.me"

  agent:
    image: drone/drone:0.8
    command: agent
    depends_on: [ server ]
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock
    environment:
      - DRONE_SERVER=ws://server:8000/ws/broker
      - DRONE_SECRET=<some secret password>

networks:
  traefik-net:
   external: true
```

Below you find the reference code from the repository for the ELK stack version 5.6

```
# PULL UPDATE & LAUNCH DOCKER
#     git pull && docker stack deploy --compose-file docker-compose.yml elk56

# REMOVE STACK
#     docker stack rm drone

# ADD BASIC AUTH and ESCAPE FOR docker-compose usage
# htpasswd -bBn user password | sed 's/\$/\$\$/g' #escape for docker-compose usage

version: "3.0"

services:

  elasticsearch:
    environment:
      ES_JAVA_OPTS: -Xms2g -Xmx2g
    image: elasticsearch:5.6
    ulimits:
      memlock: -1
      nofile:
        hard: 65536
        soft: 65536
      nproc: 65538
    volumes:
      - /usr/share/elasticsearch/data
    networks:
      - traefik-net
    command: >
      bash -c 'ip addr && IP=$$(ip addr | awk -F"[ /]*" "/inet .*\/24/{print \$$3}") &
↪& \
          echo publish_host=$$IP && \
            exec /docker-entrypoint.sh -Enetwork.bind_host=0.0.0.0 -Enetwork.publish_
↪host=$$IP -Ediscovery.zen.minimum_master_nodes=3 -Ediscovery.zen.ping.unicast.
↪hosts=elasticsearch'
    deploy:
      mode: global
      labels:
        - traefik.port=9200
        - "traefik.frontend.rule=Host:esc56.f4a.me"
```

(continues on next page)

(continued from previous page)

```
        - "traefik.frontend.auth.basic=witt:$$2y$$05$
↪$kOFY7071ilbnpiJNDaIO9e1WeuhHnKtp9Adrevz4r8wJ3b3X1XuqW"
        - "traefik.frontend.auth.basic="
        - traefik.frontend.auth.basic=witt:$$2y$$05$
↪$kOFY7071ilbnpiJNDaIO9e1WeuhHnKtp9Adrevz4r8wJ3b3X1XuqW,sebastian.schmitz:$$apr1$
↪$cCDTHFrI$$m9QaIkyU8d86N4/r/jljJ/,flex4apps:$$2y$$05$$JvvVJToTDOfTzaNA/
↪oIflecuy5l3pct94/3PfCB/g/xq4qZ49baU.


  kibana:
    image: kibana:5.6
    environment:
      ELASTICSEARCH_URL: http://elasticsearch:9200
    networks:
      - traefik-net
    deploy:
      labels:
        - traefik.port=5601
        - "traefik.frontend.rule=Host:kb56.f4a.me"
        - traefik.frontend.auth.basic=witt:$$2y$$05$
↪$kOFY7071ilbnpiJNDaIO9e1WeuhHnKtp9Adrevz4r8wJ3b3X1XuqW,sebastian.schmitz:$$apr1$
↪$cCDTHFrI$$m9QaIkyU8d86N4/r/jljJ/,flex4apps:$$2y$$05$$JvvVJToTDOfTzaNA/
↪oIflecuy5l3pct94/3PfCB/g/xq4qZ49baU.



networks:
  traefik-net:
   external: true
```

Below you find the reference code from the repository for the ELK stack version 6.2

```
# PULL UPDATE & LAUNCH DOCKER
#     git pull && docker stack deploy --compose-file docker-compose.yml elk

# REMOVE STACK
#     docker stack rm drone

# ADD BASIC AUTH and ESCAPE FOR docker-compose usage
# htpasswd -bBn user password | sed 's/\$/\$\$/g' #escape for docker-compose usage

version: "3.0"

services:

  elasticsearch:
    image: docker.elastic.co/elasticsearch/elasticsearch:6.2.2
    environment:
      - cluster.name=docker-cluster
      - bootstrap.memory_lock=false
      - http.cors.enabled=true
      - http.cors.allow-origin=*
      - "ES_JAVA_OPTS=-Xms512m -Xmx512m"
#      - "discovery.zen.ping.unicast.hosts=elasticsearch"
    ulimits:
      memlock:
        soft: -1
```

(continues on next page)

(continued from previous page)

```
        hard: -1
    volumes:
      - /data/local/elasticsearch:/usr/share/elasticsearch/data
    networks:
      - traefik-net
#    command: >
#      bash -c 'yum install -y iproute && ip addr && IP=$$(ip addr | awk -F"[ /]*" "/
↪inet .*\/24/{print \$$3}") && \
#          echo publish_host=$$IP && \
#            exec bin/elasticsearch -Enetwork.bind_host=0.0.0.0 -Enetwork.publish_
↪host=$$IP -Ediscovery.zen.minimum_master_nodes=3 -Ediscovery.zen.ping.unicast.
↪hosts=elasticsearch'
    deploy:
      labels:
        - traefik.port=9200
        - "traefik.frontend.rule=Host:elastic.f4a.me"
        - "traefik.frontend.headers.customResponseHeaders='Access-Control-Allow-
↪Origin: *'"

        - traefik.frontend.auth.basic=witt:$$2y$$05$
↪$kOFY7071ilbnpiJNDaIO9e1WeuhHnKtp9Adrevz4r8wJ3b3X1XuqW,sebastian.schmitz:$$apr1$
↪$cCDTHFrI$$m9QaIkyU8d86N4/r/jljJ/,flex4apps:$$2y$$05$$JvvVJToTDOfTzaNA/
↪oIflecuy5l3pct94/3PfCB/g/xq4qZ49baU.,scai.ndv:$$apr1$$FDUP9rVB$$3.
↪oEyn1LEXDCUhcQ5LQza.,christian.pfeiffer:$$apr1$$7efxqZjC$$RKpeAwtFKXjQEA4MzS/K61,
↪f4a:$$2y$$05$$d01dYZVn8MZnIyu.EsSuB.cPDVBCmSSQOaEpt2P2EZTbAtG594pyG
      placement:
        constraints: [node.role == manager]



  kibana:
    image: docker.elastic.co/kibana/kibana:6.2.2
    environment:
      SERVER_NAME: elasticsearch
      ELASTICSEARCH_URL: http://elasticsearch:9200
    networks:
      - traefik-net
    deploy:
      labels:
        - traefik.port=5601
        - "traefik.frontend.rule=Host:kibana.f4a.me"
        - traefik.frontend.auth.basic=witt:$$2y$$05$
↪$kOFY7071ilbnpiJNDaIO9e1WeuhHnKtp9Adrevz4r8wJ3b3X1XuqW,sebastian.schmitz:$$apr1$
↪$cCDTHFrI$$m9QaIkyU8d86N4/r/jljJ/,flex4apps:$$2y$$05$$JvvVJToTDOfTzaNA/
↪oIflecuy5l3pct94/3PfCB/g/xq4qZ49baU.,scai.ndv:$$apr1$$FDUP9rVB$$3.
↪oEyn1LEXDCUhcQ5LQza.,christian.pfeiffer:$$apr1$$7efxqZjC$$RKpeAwtFKXjQEA4MzS/K61,
↪f4a:$$2y$$05$$d01dYZVn8MZnIyu.EsSuB.cPDVBCmSSQOaEpt2P2EZTbAtG594pyG
      placement:
        constraints: [node.role == manager]

  connector:
    image: ubuntu:17.10
    networks:
      - traefik-net
    tty: true
    stdin_open: true
    ports:
```

(continues on next page)

(continued from previous page)

```
      - 26:22
    deploy:
      labels:
        - traefik.port=80
        - "traefik.frontend.rule=Host:connector.f4a.me"
      placement:
        constraints: [node.role == manager]


networks:
  traefik-net:
   external: true
```

5.4.12 kibana

Kibana is an open source data visualization plugin for Elasticsearch. It provides visualization capabilities on top of the content indexed on an Elasticsearch cluster. Users can create bar, line and scatter plots, or pie charts and maps on top of large volumes of data.

Issue the following command:

```
docker service create \
   --name kb56 \
   --label "traefik.port=5601" \
   --label 'traefik.frontend.auth.basic=flex4apps:$apr1$G9e4rgPu$jbn2AAk2F.OeGnRVFnIR/
↪1' \
   --network traefik-net \
   -e "ELASTICSEARCH_URL=http://esc56:9200" \
   kibana:5.6
```

Below you find the reference code from the repository for the ELK stack version 2.4

```
# PULL UPDATE & LAUNCH DOCKER
#     git pull && docker stack deploy --compose-file docker-compose.yml drone

# REMOVE STACK
#     docker stack rm drone

# ADD BASIC AUTH and ESCAPE FOR docker-compose usage
# htpasswd -bBn user password | sed 's/\$/\$\$/g' #escape for docker-compose usage

version: "3"

services:
  server:
    image: drone/drone:0.8
    volumes:
      - /swarm/volumes/drone:/var/lib/drone/
    environment:
      - DRONE_OPEN=true
      - DRONE_HOST=drone.f4a.me
      - DRONE_GOGS=true
      - DRONE_GOGS_URL=https://gogs.tillwitt.de
```

(continues on next page)

(continued from previous page)

```
      - DRONE_SECRET=<some secret password>
      - DRONE_ADMIN=witt
    deploy:
      labels:
        - traefik.port=8000
        - "traefik.frontend.rule=Host:drone.f4a.me"

  agent:
    image: drone/drone:0.8
    command: agent
    depends_on: [ server ]
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock
    environment:
      - DRONE_SERVER=ws://server:8000/ws/broker
      - DRONE_SECRET=<some secret password>

networks:
  traefik-net:
   external: true
```

Below you find the reference code from the repository for the ELK stack version 5.6

```
# PULL UPDATE & LAUNCH DOCKER
#     git pull && docker stack deploy --compose-file docker-compose.yml elk56

# REMOVE STACK
#     docker stack rm drone

# ADD BASIC AUTH and ESCAPE FOR docker-compose usage
# htpasswd -bBn user password | sed 's/\$/\$\$/g' #escape for docker-compose usage

version: "3.0"

services:

  elasticsearch:
    environment:
      ES_JAVA_OPTS: -Xms2g -Xmx2g
    image: elasticsearch:5.6
    ulimits:
      memlock: -1
      nofile:
        hard: 65536
        soft: 65536
      nproc: 65538
    volumes:
      - /usr/share/elasticsearch/data
    networks:
      - traefik-net
    command: >
      bash -c 'ip addr && IP=$$(ip addr | awk -F"[ /]*" "/inet .*\/24/{print \$$3}") &
↪& \
          echo publish_host=$$IP && \
            exec /docker-entrypoint.sh -Enetwork.bind_host=0.0.0.0 -Enetwork.publish_
↪host=$$IP -Ediscovery.zen.minimum_master_nodes=3 -Ediscovery.zen.ping.unicast.
↪hosts=elasticsearch'
```

(continues on next page)

(continued from previous page)

```
    deploy:
      mode: global
      labels:
        - traefik.port=9200
        - "traefik.frontend.rule=Host:esc56.f4a.me"
        - "traefik.frontend.auth.basic=witt:$$2y$$05$
↪$kOFY7071ilbnpiJNDaIO9e1WeuhHnKtp9Adrevz4r8wJ3b3X1XuqW"
        - "traefik.frontend.auth.basic="
        - traefik.frontend.auth.basic=witt:$$2y$$05$
↪$kOFY7071ilbnpiJNDaIO9e1WeuhHnKtp9Adrevz4r8wJ3b3X1XuqW,sebastian.schmitz:$$apr1$
↪$cCDTHFrI$$m9QaIkyU8d86N4/r/jljJ/,flex4apps:$$2y$$05$$JvvVJToTDOfTzaNA/
↪oIflecuy5l3pct94/3PfCB/g/xq4qZ49baU.


  kibana:
    image: kibana:5.6
    environment:
      ELASTICSEARCH_URL: http://elasticsearch:9200
    networks:
      - traefik-net
    deploy:
      labels:
        - traefik.port=5601
        - "traefik.frontend.rule=Host:kb56.f4a.me"
        - traefik.frontend.auth.basic=witt:$$2y$$05$
↪$kOFY7071ilbnpiJNDaIO9e1WeuhHnKtp9Adrevz4r8wJ3b3X1XuqW,sebastian.schmitz:$$apr1$
↪$cCDTHFrI$$m9QaIkyU8d86N4/r/jljJ/,flex4apps:$$2y$$05$$JvvVJToTDOfTzaNA/
↪oIflecuy5l3pct94/3PfCB/g/xq4qZ49baU.



networks:
  traefik-net:
   external: true
```

Below you find the reference code from the repository for the ELK stack version 6.2

```
# PULL UPDATE & LAUNCH DOCKER
#     git pull && docker stack deploy --compose-file docker-compose.yml elk

# REMOVE STACK
#     docker stack rm drone

# ADD BASIC AUTH and ESCAPE FOR docker-compose usage
# htpasswd -bBn user password | sed 's/\$/\$\$/g' #escape for docker-compose usage

version: "3.0"

services:

  elasticsearch:
    image: docker.elastic.co/elasticsearch/elasticsearch:6.2.2
    environment:
      - cluster.name=docker-cluster
      - bootstrap.memory_lock=false
      - http.cors.enabled=true
      - http.cors.allow-origin=*
```

(continues on next page)

(continued from previous page)

```
      - "ES_JAVA_OPTS=-Xms512m -Xmx512m"
#       - "discovery.zen.ping.unicast.hosts=elasticsearch"
    ulimits:
      memlock:
        soft: -1
        hard: -1
    volumes:
      - /data/local/elasticsearch:/usr/share/elasticsearch/data
    networks:
      - traefik-net
#     command: >
#       bash -c 'yum install -y iproute && ip addr && IP=$$(ip addr | awk -F"[ /]*" "/
↪inet .*\/24/{print \$$3}") && \
#           echo publish_host=$$IP && \
#             exec bin/elasticsearch -Enetwork.bind_host=0.0.0.0 -Enetwork.publish_
↪host=$$IP -Ediscovery.zen.minimum_master_nodes=3 -Ediscovery.zen.ping.unicast.
↪hosts=elasticsearch'
    deploy:
      labels:
        - traefik.port=9200
        - "traefik.frontend.rule=Host:elastic.f4a.me"
        - "traefik.frontend.headers.customResponseHeaders='Access-Control-Allow-
↪Origin: *'"

        - traefik.frontend.auth.basic=witt:$$2y$$05$
↪$kOFY7071ilbnpiJNDaIO9e1WeuhHnKtp9Adrevz4r8wJ3b3X1XuqW,sebastian.schmitz:$$apr1$
↪$cCDTHFrI$$m9QaIkyU8d86N4/r/jljJ/,flex4apps:$$2y$$05$$JvvVJToTDOfTzaNA/
↪oIflecuy5l3pct94/3PfCB/g/xq4qZ49baU.,scai.ndv:$$apr1$$FDUP9rVB$$3.
↪oEyn1LEXDCUhcQ5LQza.,christian.pfeiffer:$$apr1$$7efxqZjC$$RKpeAwtFKXjQEA4MzS/K61,
↪f4a:$$2y$$05$$d01dYZVn8MZnIyu.EsSuB.cPDVBCmSSQOaEpt2P2EZTbAtG594pyG
      placement:
        constraints: [node.role == manager]



  kibana:
    image: docker.elastic.co/kibana/kibana:6.2.2
    environment:
      SERVER_NAME: elasticsearch
      ELASTICSEARCH_URL: http://elasticsearch:9200
    networks:
      - traefik-net
    deploy:
      labels:
        - traefik.port=5601
        - "traefik.frontend.rule=Host:kibana.f4a.me"
        - traefik.frontend.auth.basic=witt:$$2y$$05$
↪$kOFY7071ilbnpiJNDaIO9e1WeuhHnKtp9Adrevz4r8wJ3b3X1XuqW,sebastian.schmitz:$$apr1$
↪$cCDTHFrI$$m9QaIkyU8d86N4/r/jljJ/,flex4apps:$$2y$$05$$JvvVJToTDOfTzaNA/
↪oIflecuy5l3pct94/3PfCB/g/xq4qZ49baU.,scai.ndv:$$apr1$$FDUP9rVB$$3.
↪oEyn1LEXDCUhcQ5LQza.,christian.pfeiffer:$$apr1$$7efxqZjC$$RKpeAwtFKXjQEA4MzS/K61,
↪f4a:$$2y$$05$$d01dYZVn8MZnIyu.EsSuB.cPDVBCmSSQOaEpt2P2EZTbAtG594pyG
      placement:
        constraints: [node.role == manager]

  connector:
    image: ubuntu:17.10
```

(continues on next page)

(continued from previous page)

```
    networks:
      - traefik-net
    tty: true
    stdin_open: true
    ports:
      - 26:22
    deploy:
      labels:
        - traefik.port=80
        - "traefik.frontend.rule=Host:connector.f4a.me"
      placement:
        constraints: [node.role == manager]


networks:
  traefik-net:
   external: true
```

5.4.13 Registry

the Registry is a stateless, highly scalable server side application that stores and lets you distribute Docker images. The Registry is open-source, under the permissive Apache license.

Below you find the reference code from the repository

```
# PULL UPDATE & LAUNCH DOCKER
#     git pull && docker stack deploy --compose-file docker-compose.yml registry

# REMOVE STACK
#     docker stack rm registry

# ADD BASIC AUTH and ESCAPE FOR docker-compose usage
# htpasswd -bBn user password | sed 's/\$/\$\$/g' #escape for docker-compose usage

version: "3"

services:
  registry:
    image: registry:latest
    networks:
      - traefik-net
    environment:
      - "REGISTRY_HTTP_SECRET=myOwnSecret"
    volumes:
        - "/data/local/registry:/var/lib/registry"
    deploy:
      labels:
        - traefik.port=5000
        - "traefik.frontend.rule=Host:registry.f4a.me"
        - "traefik.frontend.auth.basic=witt:$$2y$$05$
↪$kOFY7071ilbnpiJNDaIO9e1WeuhHnKtp9Adrevz4r8wJ3b3X1XuqW"
      placement:
        constraints: [node.role == manager]
```

(continues on next page)

(continued from previous page)

```
networks:
  traefik-net:
   external: true
```

5.4.14 Sheperd

Docker swarm service for automatically updating your services whenever their base image is refreshed

Below you find the reference code from the repository

```
# PULL UPDATE & LAUNCH DOCKER
#     git pull && docker stack deploy --compose-file docker-compose.yml shepherd

# REMOVE STACK
#     docker stack rm shepherd

# ADD BASIC AUTH and ESCAPE FOR docker-compose usage
# htpasswd -bBn user password | sed 's/\$/\$\$/g' #escape for docker-compose usage

version: "3"

services:
  shepherd:
    image: mazzolino/shepherd
    networks:
      - traefik-net
    environment:
      - SLEEP_TIME="1m"
      - BLACKLIST_SERVICES="traefik" #space seperated names
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock
    deploy:
      placement:
        constraints: [node.role == manager]

networks:
  traefik-net:
   external: true
```

5.5 Security

5.5.1 Let's encrypt

Let's Encrypt (German "Lasst uns verschlüsseln") is a certification authority that went into operation at the end of 2015 and offers free X.509 certificates for Transport Layer Security (TLS). An automated process replaces the previously common complex manual procedures for the creation, validation, signing, setup and renewal of certificates for encrypted websites.

There are certain rate limits which apply and you might run into when experimenting with the swarm.

https://letsencrypt.org/docs/rate-limits/

To search for issued certificates you can look go to:

https://crt.sh/?q=%25f4a.me

5.5.2 Docker swarm networks

Overlay networking for Docker Engine swarm mode comes secure out of the box. The swarm nodes exchange overlay network information using a gossip protocol. By default the nodes encrypt and authenticate information they exchange via gossip using the AES algorithm in GCM mode. Manager nodes in the swarm rotate the key used to encrypt gossip data every 12 hours.*[0]

5.5.3 Docker secrets

in terms of Docker Swarm services, a secret is a blob of data, such as a password, SSH private key, SSL certificate, or another piece of data that should not be transmitted over a network or stored unencrypted in a Dockerfile or in your application's source code. In Docker 1.13 and higher, you can use Docker secrets to centrally manage this data and securely transmit it to only those containers that need access to it. Secrets are encrypted during transit and at rest in a Docker swarm. A given secret is only accessible to those services which have been granted explicit access to it, and only while those service tasks are running.

You can use secrets to manage any sensitive data which a container needs at runtime but you don't want to store in the image or in source control, such as:

Usernames and passwords SSH keys Other important data such as the name of a database or internal server Generic strings or binary content (up to 500 kb in size)†[0]

5.5.4 Docker notary

Notary is a tool for publishing and managing trusted collections of content. Publishers can digitally sign collections and consumers can verify integrity and origin of content. This ability is built on a straightforward key management and signing interface to create signed collections and configure trusted publishers.

With Notary anyone can provide trust over arbitrary collections of data. Using The Update Framework (TUF) as the underlying security framework, Notary takes care of the operations necessary to create, manage, and distribute the metadata necessary to ensure the integrity and freshness of your content.‡[0]

As the virtualisation platform Docker becomes more and more widely adopted, it becomes increasingly important to secure images and containers throughout their entire lifecycle. Docker Notary is a component introduced with Docker Engine 1.8 that aims at securing the "last mile" of this lifecycle (image distribution from the registry to the Docker client), by verifying the image publisher as well as image integrity. The verification is based on digital signatures created with multiple keys. Since an in-depth security analysis of key compromise scenarios in the context of Notary seems to be missing in scientific literature, an extensive attack model is developed in this work. Based on the model, it is argued that particularly devastating attacks can be prevented by storing some of the signing keys in a hardware vault, a Secure Element. It is described how an interface can be integrated into the Notary codebase that makes this possible. It is also shown that Notary's signing process needs to be extended to prevent an attacker from exploiting one particular Notary component, the Notary server, to create arbitrary signatures on his behalf.[*]_

[0] https://docs.docker.com/v17.09/engine/userguide/networking/overlay-security-model/
[0] https://docs.docker.com/engine/swarm/secrets/
[0] https://docs.docker.com/notary/getting_started/

CHAPTER 6

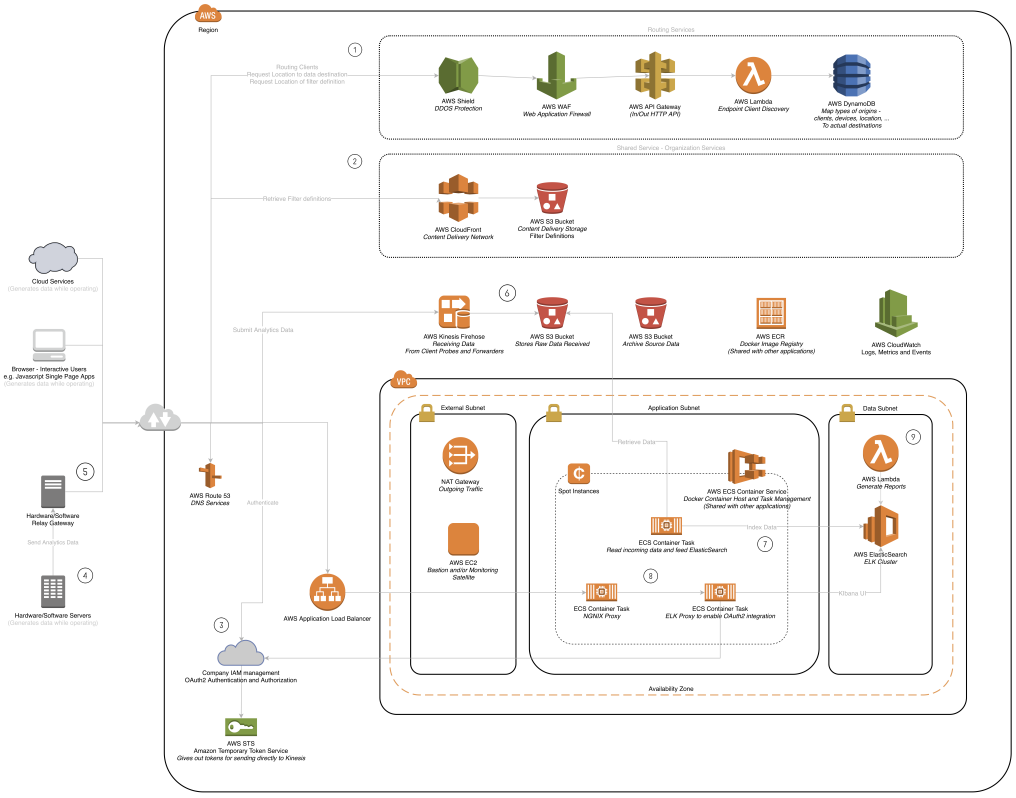
Serverless

As mentioned in the introduction, if your requirements and use cases suit a serverless approach then this is also possible, further removing yourself from concepts like virtual machines and containers. An example architecture deployment is shown here.

Flex4Apps Architecture for AWS

AWS Native Serverless Architecture

The convergence of cloud, communication and IoT infrastructure plus the trend towards virtual applications (e.g. migrating software to the cloud) create new challenges for application developers and infrastructure providers. The resulting systems are complex with dynamic resources hiding possible problems. This creates a requirement for flexible monitoring and optimisation methods. The Flex4Apps project addresses the challenges of monitoring and optimising large, distributed, cyber-physical systems. The goal of the project is to provide a solution to manage the high data volumes and complexity of system monitoring whilst disturbing the target system as little as possible.



1. Clients such as our servers installed at customer sites will call to the routing services endpoint to request the destination AWS Kinesis endpoint, allowing redirects to the appropriate global regions for data protection.
2. Clients will download a whitelist from the CDN explicitly listing which data should be collected and sent, allowing for on the fly changes if necessary. A cdn is well suited for many downloads of the same filter.
3. For integrated security an OAuth2 STS service will provide client code a valid temporary AWS token so they can start sending directly to Kinesis Firehose, this bypasses the need for serving endpoints which can potentially be hammered with data and keeps costs down.
4. Clients will buffer data and try and send the most optimal packets to kinesis to reduce bandwidth requirements and cost.
5. Aggregation Clients can also act as a gateway ensuring only a minimal number of servers need access to the internet and increase efficiency.
6. Firehose dumps the data into an S3 bucket as the volume and requests is hard to predict ensuring persistence.
7. An ECS docker container parses the incoming data and feeds this into ElasticSearch. This converts the hard to predict pull in a predictable push scenario. ElasticSearch can be easily overwhelmed and expensive to operate at scale, so to ensure you can run a minimal cluster, the docker container feeding it limits the items and insertion rate to keep running costs down.
8. Securing Kibana can be a challenge. One way to approach it would be through a specialised proxy integrating with the OAuth2 services. Kibana is exposed as a getting started UI.
9. To expose the data as analytical reports, a Lambda function could on a schedule produce reports and submit them to another system. They could be published to a website, an SQL database, an S3 bucket, a Data Warehouse, a big data analytics system or any BI tool. This further development is highly dependent on the use case and tools and is left as an exercise for the reader.

Flex4Apps Architecture Blueprint

This example leverages a serverless setup by using the Amazon AWS cloud ecosystem, but similar approaches can also be implemented using Microsoft Azure or Google Cloud.

A pdf version of this image can be found in the source repository on github.

6.1 Example serverless analytics collection stack

6.1.1 Infrastructure Compnonents

Components:

- S3 backup bucket (analytics-[environment]-backup)
- Dynamodb user profile table (analytics-[environment]-user-profile)
- AWS Elasticsearch stack (analytics-[environment]-elasticsearch)
- Elasticsearch domain policy
- Cloudwatch log group: (firehose_logs)
- Cloudwatch log stream: (firehose_s3_log_stream, firehose_es_log_stream)
- Kinesis firehose delivery stream: (analytics-[environment]-firehose-delivery)
- Elasticsearch IAM Policy
- Elasticsearch IAM user and key
- Firehose IAM Policy and role

This entire stack is deployable through a single sls deploy command.

```
#to deploy
cd analytics-services
yarn deploy
```

Components:

- Lambda Functions
- DynamoDB Migrations

6.1.2 Project overview

This repo has the following main folders:

- analytics-services
- demo-app

analytics-services

This folder contains the setup of the entire analytics backend. This

- an api gateway that listens for a POST on /event, that accepts json data in the form of .. code-block:: json *{user: "a userId", event: "an event", metaData: { json object containing meta data for the given event}}*
- a lambda function that is triggered by the above incoming POST request and that updates the user profile and forwards the incoming record to a kinesis firehose.

- firehose stream that forwards incoming events to an elasticsearch cluster and S3 backup bucket
- the elasticsearch domain
- the S3 backup bucket for the raw data
- all the necessary iam policies

To deploy this backend, one needs to first install the serverless toolkit (https://serverless.com/). This framework is an easy tool that automates packaging and deploying lambda functions really easy. This toolkit is written in nodejs and requires a nodejs runtime to be installed. Note that the serverless toolkit does not require the lambda functions themselves to be written in nodejs.

Once nodejs is installed, install the serverless toolkit.

```
npm install -g serverless
```

Next, setup a AIM user on AWS that has enough rights to create and update the resources needed. Detailed documentation can be found at https://serverless.com/framework/docs/providers/aws/guide/credentials/.

Than, in the analytics-services folder, run

```
yarn install
```

to install project dependancies. Note that for the analytics services, this npm install will install both the dependancies required for the services itself, as well as some dependancies that are introduced in the serverless.yml project description file.

Next you can deploy the project using

```
yarn deploy
# or
yarn sls deploy -v
```

This last command will setup the entire infrastructure. Under the hood, serverless uses AWS Cloudformation, a dedicated service by AWS to describe and deploy entire "stacks" of resources. AWS Cloudformation is a free service, one only incures costs for the resources that are deployed within the stack. The lambda functions in this folder are developed in nodejs.

All resources that are deployed with the previous command are part of the mystage stage. One can deploy multiple environments using different stages (e.g. –stage prod for production).

```
yarn remove
```

will teardown the entire infrastructure.

One thing to note: Without special precaution, the API gateway end points that AWS Gateway returns are rather cryptic. (e.g. endpoints: https://hvd9fb2p5f.execute-api.eu-west-1.amazonaws.com/devel/event). In order to have these endpoints in a more meaningful way (e.g. stats.mydomain.com), you will need to create an API Gateway domain. This can be done through serverless. (full documentation at https://serverless.com/blog/serverless-api-gateway-domain/)

This project includes the custom domain plugin that does this automatically. Since API Gateway only works with https, you'll need to setup an ssl certificate. For that, you will need to use AWS Certificate Manager (https://aws.amazon.com/certificate-manager/). You can use your own certificates or have AWS handle that for you. AWS Certificate manager is a free service.

Setup your custom domain in AWS Certificate Manager, be sure to do that in the US-East-1 region (it will not work otherwise). Once that is done, you can setup your custom API Gateway domain through:

```
yarn sls create_domain --stage mystage
```

This can take up to 40 minutes. If you use AWS Route 53 for your DNS, `sls create_domain` can update your DNS for you, you will need to edit the serverless.yml file for this. (For my demo, I'm not using Route 53)

Once your domain is up and running, simply redeploy using `sls deploy --stage mystage`

Deploy commands

```
cd analytics-services
yarn install
yarn sls deploy -v --stage mystage
```

To remove the stack: `yarn sls remove -v --stage mystage`

demo-app

This folder contains a small web application that can be used to send test events to the analytics stack. Be sure to edit app.js to post to the correct end point

6.1.3 Installation

- **Serverless**
  - `npm install -g serverless`
- Add credentials to the `.aws/credentials` file
- Do a `npm install` in the `analytics-services` folder

6.1.4 Accessing kibana

When the analytics service is deployed, a IAM user is created with as only privilage that that user can access the elastic search cluster. An access key and secret are generated and published as output. In order to access kibana, the easiest solution is to run a local proxy server. Check https://github.com/abutaha/aws-es-proxy for details.

What I did:

```
wget https://github.com/abutaha/aws-es-proxy/releases/download/v0.4/aws-es-proxy-0.4-
↪mac-amd64 -O aws-es-proxy
chmod +x aws-es-proxy
export AWS_PROFILE=esuser #be sure to add a esuser into your ~/.aws/credentials file,␣
↪copy over access key and secret
./aws-es-proxy -endpoint https://<search-endpoint>.<region>.es.amazonaws.com
```

you can now go to http://127.0.0.1:9200/_plugin/kibana and access your kibana dashboards.

Note that you can also setup a proper login for Kibana, be sure to check: https://aws.amazon.com/blogs/database/get-started-with-amazon-elasticsearch-service-use-amazon-cognito-for-kibana-access-control/

Unfortunately, at present it is not really possible to set this up in an automated fashion

CHAPTER 7

API and data flow

Based on the architectural structure of a Flex4Apps solution (https://f4a.readthedocs.io/en/latest/chapter04_architecture/_structure.html) there are different components connected to form a working system. Others are necessary to run individual components within the architecture.

7.1 APIs to exchange data between system components

Some of the components are designed already to closely work together while others have to be adapted. This section will give a survey on which APIs might be relevant to tailor Flex4Apps to a certain application.

The figure below shows some scenarios and API's to regard for this

Since APIs are version dependent we prefer to link to the API definitions rather than integrating them into this documentation.

7.1.1 API 1: XML RPC for HomeMatic

A Gateway (F4A-Box with optional and safe Genode-OS) collects sensor data from an aggregation device cyclically. For this it implements a vendor specific interface.

- https://github.com/hobbyquaker/XML-API/releases/tag/1.16
- https://www.eq-3.de/Downloads/eq3/download%20bereich/hm_web_ui_doku/HM_XmlRpc_API.pdf

7.1.2 API 2: Non public REST-API

If data is available online then no separate gateway is necessary. Some vendor specific code can be used to connect numerous data sources with Flex4Apps toolbox. This API is an example how to use various online data sources

- http://www.powerfox.energy/

- https://development.powerfox.energy/

7.1.3 API 3: Standard MQTT protocol

Using MQTT is very common for many cyber pyhsical (IoT) systems. There are two basic use-cases to be distinguished:

1. Devices (publishers) send their information directly to a managing instance (broker).
2. An aggregation device collects information according to API 1 or 2 and uses MQTT to forward it to a broker. Appropriate software can run on many hardware platforms.

For security reasons it is advisable to use a VPN tunnel to transfer MQTT packets. Alternatively (or additionally) MQTT devices shall use encryption to communicate with the broker.

- https://www.thethingsnetwork.org/docs/applications/mqtt/api.html

7.1.4 API 4: Store MQTT payload

In order to use aggregated information in consecutive processing operations Influx can be used.

- https://docs.influxdata.com/influxdb/v1.7/tools/api/

7.1.5 API 5: Present information with e.g. Grafana

Grafana accesses influx data via a standard data source driver. This allows inspection of data independent from a specific domain model. For third party applications there exist specific APIs to aggregate and visualize data series.

- https://docs.influxdata.com/influxdb/v1.7/tools/api/

7.1.6 API 6: Enhance MQTT data with additional meta information

A modified MOSQUITTO broker sends MQTT payload and further information about connectivity status, publisher- and subscriber-ID's and some more information to ELK stack. Logstash gathers these messages from a dedicated TCP connection.

- https://www.elastic.co/guide/en/logstash/current/plugins-inputs-tcp.html
- https://nmap.org/ncat/guide/index.html

7.1.7 API 7: Logstash pushes data to Elasticseach

Since Logstash is part of the Elastic-Stack (ELK-stack) there is no custom code or API adaptation needed.

7.1.8 API 8: Inspect Data with Kibana

Kibana is also part of ELK stack. So there is no API to adapt. However, a user can alternatively use the Elasticseach-API to query data directly if another application shall be used.

- https://www.elastic.co/guide/en/elasticsearch/reference/current/docs.html
- https://www.elastic.co/guide/en/elasticsearch/reference/current/docs-get.html

Additionally Elastic-Stack offers Machine learning and anomaly detection mechanism. To use it the following links are useful as an entry point:

- https://www.elastic.co/guide/en/x-pack/current/index.html

7.1.9 API 9: Alert anomaly events to external application

If X-Pack is used for anomaly detection it is possible to send alerts to different kind of receivers, e.g. e-Mail, Push-Service, HTTP-Post.

- https://www.elastic.co/guide/en/kibana/current/watcher-getting-started.html
- https://www.elastic.co/guide/en/x-pack/current/xpack-alerting.html

7.2 APIs to make components working

Beside the APIs mentioned before there are some more to make components up and running within the system. The following chapters describes some of them.

7.2.1 Docker Engine

Docker provides an API for interacting with the Docker daemon (called the Docker Engine API), as well as SDKs for Go and Python. The SDKs allow you to build and scale Docker apps and solutions quickly and easily. If Go or Python don't work for you, you can use the Docker Engine API directly.

The Docker Engine API is a RESTful API accessed by an HTTP client such as wget or curl, or the HTTP library which is part of most modern programming languages.

- https://docs.docker.com/engine/api/version-history/

7.2.2 Kubernetes

The Kubernetes API serves as the foundation for the declarative configuration schema for the system. The kubectl command-line tool can be used to create, update, delete, and get API objects.

- https://kubernetes.io/docs/concepts/overview/kubernetes-api/

7.2.3 Traefik reverse proxy

Traefik is a modern HTTP reverse proxy and load balancer that makes deploying microservices easy. Traefik integrates with your existing infrastructure components (Docker, Swarm mode, Kubernetes, Marathon, Consul, Etcd, Rancher, Amazon ECS, ...) and configures itself automatically and dynamically. Pointing Traefik at your orchestrator should be the only configuration step you need.

- https://docs.traefik.io/

7.2.4 Rancher

Rancher is an open source software platform that enables organizations to run and manage Docker and Kubernetes in production. With Rancher, organizations no longer have to build a container services platform from scratch using a distinct set of open source technologies. Rancher supplies the entire software stack needed to manage containers in production.

- https://rancher.com/docs/rancher/v1.6/en/api/v2-beta/

7.2.5 Security

Enabling the API will expose all configuration elements, including sensitive data.

It is not recommended in production, unless secured by authentication and authorizations.

A good sane default (but not exhaustive) set of recommendations would be to apply the following protection mechanism:

At application level: enabling HTTP Basic Authentication At transport level: NOT exposing publicly the API's port, keeping it restricted over internal networks (restricted networks as in https://en.wikipedia.org/wiki/Principle_of_least_privilege).

- https://docs.traefik.io/configuration/api/

7.2.6 MOSQUITTO

It is recommended to run MOSQUITTO on a server environment that allows to link easy with other components of the system. To get further information see:

- https://github.com/Flex4Apps/mosquitto

CHAPTER 8

Service and solution provider

Questions about Flex4Apps?

If you need additional information pls. get in contact with one of the following companies referring to the Flex4Apps ITEA project:

• DataStories	https://datastories.com
• Sirris	https://www.sirris.be
• Genode Labs GmbH	https://www.genode-labs.com
• HiConnect	http://hiconnect.de/index.php
• evermind GmbH	https://evermind.de
• Fraunhofer SCAI	https://www.scai.fraunhofer.de
• Fraunhofer IIS/EAS	https://www.eas.iis.fraunhofer.de

CHAPTER 9

Demonstrators

There are many demonstrators about this project:

9.1 Cloud provisioning

The video below demonstrates how quickly the docker based version of Flex4Apps can be deployed.

9.2 Cloud applications

based on the provisioned swarm, applications like Kibana and Elastic can be accessed as demonstrated in this video.

—

9.3 External references

based on the Flex4Apps project various companies deployed projects on their own also feeding back into the project. Below you find an example from a start up company within the DLT/ICO environment:

CHAPTER 10

References

10.1 related projects

SCRATCh - SeCuRe and Agile Connected Things (website)

About scratch: The development and operation of secure, large-scale IoT systems is difficult. Technological platforms providing the necessary building blocks to integrate devices and backbone logic exist, but do not address the major concerns of today's software-intensive systems: security, agility and a need for continuous deployment. SCRATCh proposes an integrative approach to IoT, security and DevOps practices through an architectural and process platform consisting of a hardware security foundation for device identity management and security metrics collection, DevOps IoT platform and DevSecOps process, promoting continuous secure operation.

The link: SCRATCh currently tests the F4A infrastructure to operate the project.

10.2 Github repos

Besides the github repository from which this documentation is generated, there are some other repositories which are useful within the scope of this project:

- Genode Operating System Genode and Genode-World
- MQTT Broker Mosquitto
- Grafana inside a AWS VPC on Fargate Grafana-VPC
- Example serverless analytics collection stack

Please also take a look at the Github organisation Flex4Apps as it allows easy access to all the resources above.

10.3 external articles & books

10.3.1 BOOK: Hyperscale and Microcare

Hyperscale and Microcare: The Digital Business Cookbook

A DIGITAL BUSSINESS IS DIFFERENT

Software is a Commodity Thanks to open source, the Cloud and API's, it is easier than ever before to build software. As a result, competition is tough, prices are under pressure. Stop developing software and start delivering service.

HyperScale & MicroCare To make the most out of a digital service, you'll want to service a large customer base with a limited team, yet offer each customer an experience that is as personal as possible

The true valuation of a digital service Why is there such a large gap between the book value of digital companies, and their valuation? Digital services build up new types of assets one does not (yet) find in their balance sheet. A strong online presence is one such asset, a loyal customer base is another

10.3.2 ARTICLE: High data protection and data security requirements

Article published in German: Hohe Anforderungen an Datenschutz und Datensicherheit

Abstract: The topic of data protection requires a multi-layered security architecture, which already takes into account security aspects in the The development process of products is integrated. NXP Semiconductors has formulated a series of recommendations and products which can protect against cyber attacks and ensure the security of personal data.

CHAPTER 11

Thanks

The Flex4Apps project saw the light during the brokerage days in Brussels in 2015. After successful submission of PO, FPP and local applications, the Flex4Apps project started in Belgium and Germany mid 2016 and ran for a total of 36 months.

Without the support of the project would not have been the same:

- BMBF - https://www.bmbf.de/foerderungen/bekanntmachung-1662.html
- DLR-Projektträger - https://www.softwaresysteme.pt-dlr.de/de/itea.php
- ITEA3 - https://itea3.org/
- VDI/VDE
- Vlaio - https://www.vlaio.be/nl

CHAPTER 12

other topics

Certain topics needed to be at hand during the project quite often. As they don't fint into a singular section we put them in here:

12.1 cheat sheet

12.1.1 Clean up of containers

remove all exited containers (should be run on each node):

```
docker rm $(docker ps -q -f status=exited)
```

attach bash to a running container:

```
sudo docker exec -i -t containername /bin/bash
```

Remove all containers:

```
docker rm $(docker ps -a -q)
```

Remove all images:

```
docker rmi $(docker images -q)
```

Whiping out everything:

```
docker system prune -a
```

12.1.2 Networking

Finding public ip:

```
wget -qO - https://api.ipify.org
```

12.2 Dos and don'ts

https://community.spiceworks.com/topic/1832873-a-list-of-don-ts-for-docker-containers

1) Don't store data in containers
2) Don't ship your application in two pieces
3) Don't create large images
4) Don't use a single layer image
5) Don't create images from running containers
6) Don't use only the "latest" tag
7) Don't run more than one process in a single container
8) Don't store credentials in the image. Use environment variables
9) Don't run processes as a root user
10) Don't rely on IP addresses

12.3 Updating this documentation

issue the following command to update this documentation:

```
docker run --name sphinxneeds --rm \
      -e "Project=Flex4apps" \
      -e "Author=Till Witt, Johannes Berg, Alex Nowak" \
      -e "Version=v0.1" \
      -v "$(pwd)/compose:/project/compose" \
      -v "$(pwd)/docs:/project/input" \
      -v "$(pwd)/output:/project/output" \
      -i -t tlwt/sphinxneeds-docker


docker run --name buildTheDocs --rm \
      -e "Project=Flex4apps" \
      -e "Author=Till Witt, Johannes Berg, Alex Nowak" \
      -e "Version=v0.1" \
      -v "$(pwd)/compose:/project/compose" \
      -v "$(pwd)/docs:/project/input" \
      -v "$(pwd)/output:/project/output" \
      -i -t sphinx_image
```

12.4 glossary

API Application programming interface

CPS cyber-physical systems

CPU Central processing unit

DNS The Domain Name System (DNS) is a hierarchical and decentralized naming system for computers, services, or other resources connected to the Internet or a private network.

DLT A distributed ledger (also called a shared ledger or distributed ledger technology or DLT) is a consensus of replicated, shared, and synchronized digital data geographically spread across multiple sites, countries, or institutions.

FTP The File Transfer Protocol (FTP) is a standard network protocol used for the transfer of computer files between a client and server on a computer network.

HTTP The Hypertext Transfer Protocol (HTTP) is an application protocol for distributed, collaborative, hypermedia information systems.

ICO An initial coin offering (ICO) or initial currency offering is a type of funding using cryptocurrencies. Mostly the process is done by crowdfunding but private ICO's are becoming more common.

LTS long term support

MQTT MQTT[2] (Message Queuing Telemetry Transport) is an ISO standard (ISO/IEC PRF 20922)[3] publish-subscribe-based messaging protocol.

RAM Random-access memory (RAM /ræm/) is a form of computer data storage that stores data and machine code currently being used.

REST Representational State Transfer (REST) is a software architectural style that defines a set of constraints to be used for creating Web services.

SDK A software development kit (SDK or devkit) is typically a set of software development tools that allows the creation of applications for a certain software package, software framework, hardware platform, computer system, video game console, operating system, or similar development platform.

SE A Secure Element (SE) is a chip of a smartphone, part of the SIM card or other form factors. It provides the necessary security and confidentiality for mobile applications such as mobile commerce, mobile payment, mobile banking or mobile security.

SMTP The Simple Mail Transfer Protocol (SMTP) is a communication protocol for electronic mail transmission.

SNMP Simple Network Management Protocol (SNMP) is an Internet Standard protocol for collecting and organizing information about managed devices on IP networks and for modifying that information to change device behavior.

SSD A solid-state drive (SSD) is a solid-state storage device that uses integrated circuit assemblies as memory to store data persistently. It is also sometimes called a solid-state device or a solid-state disk,[1] although SSDs do not have physical disks.

SSL Transport Layer Security (TLS), and its now-deprecated predecessor, Secure Sockets Layer (SSL),[1] are cryptographic protocols designed to provide communications security over a computer network.

TCP The Transmission Control Protocol (TCP) is one of the main protocols of the Internet protocol suite.

UDP In computer networking, the User Datagram Protocol (UDP) is one of the core members of the Internet protocol suite.

YAML/YML YAML ("YAML Ain't Markup Language") is a human-readable data-serialization language.

CHAPTER 13

Indices and tables

- genindex
- modindex
- search

Index