Totem-Documentation Documentation

Release 1.0.1 alpha

Maximilian Schott

December 13, 2016

Installation Using The Installer Script

A simple mostly unconfigured bare installation of Holmes-Totem and/or Holmes-Storage can be achieved by executing the *universal-installer.sh*.

To get access to the installer script, clone the repository found at Holmes-Toolbox.

```
git clone https://github.com/HolmesProcessing/Holmes-Toolbox
```

A fully functional system needs several components:

- Cassandra / MongoDB
- RiakCS / Amazon S3 / LocalFS (Component of Holmes-Storage)
- RabbitMQ
- Holmes-Storage
- Holmes-Totem

Of these components the following can be installed with the installer script:

- Cassandra
- RabbitMQ
- Storage (optionally configured for LocalFS)
- Holmes-Totem

1.1 Holmes-Toolbox Universal Installer

1.1.1 Usage

```
./universal-installer.sh
    [--rabbitmq]
    [--cassandra]
    [--storage [path:...] [repo:...] [[config-url:...]|[config-create:...]] [no-initscript] [erase]
    [--totem [path:...] [repo:...] [branch:...] [no-services] [no-initscript] [erase]]
    [--resume]
```

1.1.2 Log-Output

The Universal Installer logs all output to a file called `universal-installer.log` which is located in the same directory as the installer script.

1.1.3 Option-Details

--rabbitmq	Install RabbitMQ (Task scheduler for Holmes-Totem)
--cassandra	Install Apache-Cassandra (Database)
--storage	Install Holmes-Storage (Storage backend for Holmes-Totem)

Available options for Holmes-Storage:

path:/install/path	Path to install Holmes-Storage in. *(default: /data/holmes-storage)*
repo:http://repo.url	Repository that Holmes-Storage is pulled from, without protocoll and without `.git` extension. *(default: github.com/HolmesProcessing/Holmes-Storage)*
config-url:http://config.url	Specify the location of the configuration file, will be loaded with curl. *(incompatible with config-create)*
config-create:TYPE	Specify the configuration to be created. *(semi-automatic, incompatible with config-url option)*
no-initscript	Do not install init scripts for upstart/systemd.
erase	Purge any previous installation in the specified installation location. **This does NOT purge existing database entries.**
skip-setup	Do not run database setup routines.

Available config-create `TYPE's` are:

local	Install Cassandra, configure Holmes-Storage to use Cassandra and LocalFS.
local-objstorage	Configure Holmes-Storage to use Cassandra and LocalFS.
local-cassandra	Install Cassandra, configure Holmes-Storage to use Cassandra and S3.
cluster	Configure Holmes-Storage to use Cassandra and S3.
local-mongodb	Install MongoDB, configure Holmes-Storage to use MongoDB.
cluster-mongodb	Configure Holmes-Storage to use MongoDB.

Default `TYPE` is cluster.

--totem Install Holmes-Totem

Available options for Holmes-Totem:

path:/install/path	Path to install Holmes-Totem to. *(default: /data/holmes-totem)*
repo:http://repo.url	Repository that Holmes-Totem is pulled from. *(default: https://github.com/HolmesProcessing/Holmes-Totem.git)*
branch:repository-branch	Branch to checkout after cloning the repository. (If not supplied, no checkout)
no-services	Do not install Totems default services. **Implied by ''no-initscript''**
no-initscript	Do not install init scripts for upstart/systemd
erase	Purge any previous installation in the specified installation location.

--resume Attempt to resume at the last known checkpoint.

Use with utmost care

This option should only be used with the exact same parameters as before.

Otherwise proper functionality cannot be guaranteed.

Further, installing Holmes-Storage may fail repeatedly if it fails at the

database setup. See the trouble shooting section for info on how to deal with this.

1.1.4 Important Notes

- **Unique Paths**

  If you install both, Holmes-Totem and Holmes-Storage, the given installation paths must be unique for each of them.

- **Totem requires a custom configuration**

  Please fork the original repository and create/adjust configuration as necessary.

  A repository with a working config is available at:

```
https://github.com/ms-xy/Holmes-Totem.git
```

1.1.5 Trouble Shooting

- **If Storage fails to start / setup due to timeouts connecting to Cassandra**

  In this case, try using the following repository during installation (`repo` option for `--storage`):

```
https://github.com/ms-xy/Holmes-Storage.git
```

This version of Holmes-Storage sets the request timeout to 10 seconds (default 600ms). If this does not help, your chosen server most likely offers too little resources for a reasonable Cassandra installation and you should use a different one.

Note: You can use the `--resume` flag to skip all previous steps and retry the database setup directly.

Warning: Delete any tables that got created or the database setup will fail with the error that the tables already exist. (Cassandra creates the tables, even if the connection timed out)

1.1.6 Examples

Here's a quick rundown of examples. For detailed information, please see the corresponding example section.

- A complete setup (RabbitMQ + Cassandra + Storage + Totem) on one server:

```
./universal-installer.sh \
    --rabbitmq \
    --cassandra \
    --storage \
        config-create:local \
    --totem \
        repo:"your-git-repo"
```

- Only Storage, in cluster mode (configure Cassandra + S3 external):

```
./universal-installer.sh --storage
```

- Only Totem, without its services as init job, erase an old installation if there is one in the same installation directory:

```
./universal-installer.sh \
    --totem \
        repo:"your-git-repo" \
        no-services \
        erase
```

- Install Storage and Totem in custom directories:

```
./universal-installer.sh \
    --totem \
        repo:"your-git-repo" \
        path:/custom/install/path/totem \
    --storage \
        path:/custom/install/path/storage
```

1.2 Local Installation

A full local installation needs to serve all 5 components.

However, it is not really useful to install a RiakCS object storage on a single node, thus it is sufficient to only install the following 4 components:

- Cassandra
- RabbitMQ
- Holmes-Storage (configured for LocalFS)
- Holmes-Totem

The following command automatically installs all the mentioned components and guides the user through a basic configuration for Holmes-Storage:

```
universal-installer.sh \
    --cassandra \
    --rabbitqm \
    --storage \
        config-create:local \
    --totem \
        repo:"some://custom.repo.url"
```

The `--cassandra` and `--rabbitmq` flags should be self-explanatory.

The `--storage` flag tells the installer to run the Holmes-Storage installer and orders it to create a config file for a local installation. Whilst this should work without a problem on any decent server, it may fail during the database setup routine if Cassandra is a bit slow (default timeout is 600ms per request). If you run into this issue, you have two options:

- Try using a version of Holmes-Storage modified specifically to address this issue.
- Tune the Cassandra installation to serve the requests faster.

For the first option there exists a repository that addresses users that face this issue. Instead of `--storage`, use: `--storage repo:"https://github.com/ms-xy/Holmes-Storage.git"`.

If you opt to do the manual tuning, the bottleneck during the setup might be the slow table creation commands.

Please note that Holmes-Totem **requires** custom configuration files. Fork the Holmes-Totem GitHub repository (can be found here) and create the necessary configuration files / edit existing configuration files. (A very basic working configuration can be found here)

Details about the configuration can be found in the Holmes-Totem Configuration section of this documentation.

For trouble shooting, please refer to the trouble shooting section.

1.3 Totem + Storage + Database Cluster

Any decent installation should consist of Holmes-Totem and Holmes-Storage backed up by all other required components installed externally. In other words, you want to install RabbitMQ, Cassandra/MongoDB, and RiakCS on dedicated servers.

This example assumes that you have already set up these boxes. For RabbitMQ you can use the installer. However, you need at least 3 nodes for Cassandra and 4 nodes for RiakCS. The configuration necessary to get such a cluster up and running cannot be done by the install script.

In order to install Totem and Storage and then configure Storage for cluster mode use:

```
universal-installer.sh \
    --storage \
    --totem \
        repo:"some://custom.repo.url"
```

The `--storage` flag tells the installer to run the Holmes-Storage installer. Since `create-config:cluster` is the default mode, you can omit this option.

Please note that Holmes-Totem **requires** custom configuration files. Fork the Holmes-Totem GitHub repository (can be found here) and create the necessary configuration files / edit existing configuration files. (A very basic working configuration can be found here)

Details about the configuration can be found in the Holmes-Totem Configuration section of this documentation.

For trouble shooting, please refer to the trouble shooting section.

1.4 Trouble Shooting

If your error is not listed here and there is no issue on github, yet, please file a new one on the Toolbox repository and describe your problem as well as a way to reproduce it. (Repositories can be found here: Documentation, Totem, Storage, and Toolbox)

- **The Cassandra database setup failed due to a connection timeout**

  Holmes-Storage uses a third party library to connect to Cassandra which sets the default connection timeout to 600ms. This is a pretty low value, which can easily cause problems when creating new tables (slow machines may need several seconds to complete such operation).

  To work around this issue, either modify the Holmes-Storage source code yourself, or use the version provided at `https://github.com/ms-xy/Holmes-Storage.git`. It sets the timeout to 10s.

  However, make sure that you delete any table that has been created before you rerun the installer. (Even if the connection times out, Cassandra still creates the respective table)

```
./universal-installer.sh --resume --storage \
    repo:"https://github.com/ms-xy/Holmes-Storage.git"
```

Note: The `--resume` flag will order the install script to skip all steps up to the one that errored out.

- **The services for Totem fail to start**

  Make sure that your services are all configured and that the configuration has no errors.

  By default all services only have a service.conf.example file or similar, which isn't read upon starting the service.

  Also the example configuration file does not necessarily work out of the box.

  (Example: Yara service misses its Yara rules)

CHAPTER 2

Installation

Note: If you are new to Holmes-Totem, you should check out the section about the automatic installation of Holmes-Totem.

2.1 Install Docker

Linux Kernels before 3.10 can not run Docker. Please verify by issuing `uname -r` in a terminal that your Kernel is suitable.

```
sudo apt-get update
sudo apt-get install curl

# install docker
curl -sSL https://get.docker.com/ | /bin/sh

# install docker-compose
sudo sh -c "curl -L https://github.com/docker/compose/releases/download/1.7.0/docker-compose-`uname -
sudo chmod +x /usr/local/bin/docker-compose
```

Done. Yes that's it, now you have docker compose and docker.

Warning: Installing docker-compose like this will overwrite any existing docker-compose installation.

2.2 Install Holmes-Processing

The sections below details the setup of all the possible components. Whilst Java 8 is not really a component, it is a requirement for 2 other component and as such has its own section.

(Cassandra requires Java 7 or 8, Holmes-Totem requires Java 8)

A complete system consists of:

- Holmes-Totem
- Holmes-Storage
- Apache Cassandra

- Riak CS
- RabbitMQ

2.2.1 RabbitMQ

Totem requires a AMPQ message broker, with the default one being RabbitMQ. Each Holmes-Totem instance queries this message broker for tasks and pushes results back to it. Holmes-Storage queries the queue and stores the results in the database.

To install RabbitMQ with base settings:

```
sudo apt-get update
sudo apt-get install apt-transport-https wget

echo 'deb http://www.rabbitmq.com/debian/ testing main' | sudo tee /etc/apt/sources.list.d/rabbitmq.l
wget -O- https://www.rabbitmq.com/rabbitmq-signing-key-public.asc | sudo apt-key add -

sudo apt-get update
sudo apt-get install rabbitmq-server

sudo service rabbitmq-server start
```

In case SELinux is installed, it may prevent port binding. Make sure the following ports are open for RabbitMQ to work properly:

```
4369 (epmd), 25672 (erlang-dist)
5672, 5671 (AMQP 0-9-1 with and without TLS)
15672 (management plugin)
61613, 61614 (if STOMP is enabled)
1883, 8883 (if MQTT is enabled)
```

2.2.2 Java 8

Java is a requirement for Cassandra as well as for Holmes-Totem. Holmes-Totem currently only works with Java 8 due to limitations of the frameworks that are used.

The easiest way to install it is to use the webupd8team repository:

```
sudo apt-get update
sudo apt-get install apt-transport-https

## Ubuntu before 12.10:
# sudo apt-get install python-software-properties
#
## All others:
sudo apt-get install software-properties-common

sudo add-apt-repository -y ppa:webupd8team/java
sudo apt-get update

sudo apt-get install -y oracle-java8-installer
sudo update-alternatives --config javac
```

2.2.3 Apache Cassandra

Cassandra is the default database for Holmes-Totem. It should be set up with 3 nodes at least for redundancy and load balancing.

Cassandra requires Java 7 or 8 (how to install Java 8 see above).

```
sudo apt-get update
sudo apt-get install python python3 libjna-java curl

## Ubuntu 16.04 or greater requires installing python-support:
# sudo curl -o /tmp/python-support_1.0.15_all.deb http://launchpadlibrarian.net/109052632/python-supp
# sudo dpkg -i /tmp/python-support_1.0.15_all.deb
# sudo rm /tmp/python-support_1.0.15_all.deb

echo "deb http://debian.datastax.com/community stable main" | sudo tee /etc/apt/sources.list.d/cassan
curl -L http://debian.datastax.com/debian/repo_key | sudo apt-key add -

sudo apt-get update
sudo apt-get install -y cassandra=3.0.5
```

2.2.4 Riak CS / S3 Object Storage

RiakCS is the default object storage database for Holmes-Totem. Its installation and configuration is complex. Please refer to the RiakCS website for help with installing it.

Alternatively you can use Amazon S3 for object storage.

2.2.5 Holmes-Storage

This is the component responsible for storing and retrieving objects and results. It is a wrapper around the database and object storage.

Before installing Holmes-Storage, you need to install The Go Programming Language.

```
sudo apt-get update
sudo apt-get install curl

curl -o /tmp/go.tar.gz -L "https://storage.googleapis.com/golang/go1.6.1.linux-amd64.tar.gz"
sudo tar -C /usr/local -xzf /tmp/go.tar.gz
rm /tmp/go.tar.gz

# The following environmental variables need to be set in order to run go:
export GOPATH="$HOME/go"
export PATH=$PATH:/usr/local/go/bin
export GOROOT=/usr/local/go
```

The next step is to download and build Holmes-Storage, which `go` happily does for you (note that there is no `https://` or `git://` in front of your URL):

```
go get -v -x -u "github.com/HolmesProcessing/Holmes-Storage"
```

After go finishes downloading and compiling Holmes-Storage, the executable can be found in `$HOME/go/bin/Holmes-Storage`.

Create a dedicated installation folder in `/data/holmes-storage`.

Copy the executable there.

Download the configuration example from the root of the Holmes-Storage repository (here) and adjust it to your needs.

2.2.6 Holmes-Totem

Docker

If you want to run Holmes-Totem's services, you need to install Docker. Please refer to the Docker section.

Then after the installation of Holmes-Totem, you can simply run `docker-compose up -d .` in the `config` directory of your Holmes-Totem installation.

Scala Build Tool

In order to compile Holmes-Totem you have to install `sbt` (Scala Build Tool). See the official website for more info.

```
echo "deb https://dl.bintray.com/sbt/debian /" | sudo tee /etc/apt/sources.list.d/sbt.list > /dev/nul
sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv 2EE0EA64E40A89B84B2DF73499E82A75642
sudo apt-get update
sudo apt-get install -y sbt
```

Clone and build Holmes-Totem

After installing `sbt` clone Holmes-Totem from the GitHub repository and build it from source.

```
git clone https://github.com/HolmesProcessing/Holmes-Totem.git
cd Holmes-Totem
sbt assembly
```

Which will produce a jar-file, which in turn you can start by issuing:

```
java -jar target/scala-2.11/totem-assembly-1.0.jar config/totem.conf
```

However, you'll need to configure it first. (See the Holmes-Totem section of this documentation)

2.2.7 Init Scripts

If you want your Holmes-Totem / Holmes-Storage to start on system start, you need to install service or unit files (depending on which init system you have).

Example files can be found in the Holmes-Toolbox .

Systemd unit files go into `/etc/systemd/system/`.

Upstart configuration files go into `/etc/init/`.

2.3 Install Fileserver

In case you are hosting files on a different machine than just Totem - which is not unlikely - you'll have to distribute your samples to your services somehow.

The easiest way to do that is by having a Fileserver running on each Totem host, that allows accessing the samples stored in the /tmp directory of the host.

The simplest Fileserver possible is Tornado.

```
import tornado.web

settings = {
    "static_path": "/tmp"),
    "cookie_secret": "YOUR_SECRET_VALUE",
    "login_url": "/login",
    "xsrf_cookies": True,
}

application = tornado.web.Application([
    (r"/*", tornado.web.StaticFileHandler,
     dict(path=settings['static_path'])),
], **settings)
```

Safe as `fileserver.py` and run it with `python2 fileserver.py`.

In case you prefer encapsulating it into a Docker container:

```
FROM ubuntu:14.04

RUN apt-get update && apt-get -y upgrade
RUN apt-get install -y python python-pip
RUN pip install tornado

RUN mkdir -p /fileserver
ADD fileserver.py /fileserver

RUN useradd -s /bin/bash fileserver
RUN chown -R fileserver /fileserver

USER fileserver
WORKDIR /fileserver

# serving as a HTTP file server on port 80 of the container
EXPOSE 80

CMD python2 /fileserver/fileserver.py
```

Build and run your Dockerfile:

```
docker build -t fileserver .
docker run -p 8080:80 -v /tmp:/tmp:ro fileserver:latest
```

This would publish the container port 80 to the host port 8080.

Docker

Some of the key components can be easily installed within a Docker container.

For a guide of how to install Docker, please see the Manual Installation section!

3.1 Totem

Here is a Dockerfile for running your Totem in Docker.

We do not provide any images for Totem, as you would most likely configure it different than we do.

Note: You will want to fork your own copy of the repository in order to change the settings.

Also note that Totems services are Docker containers on their own and should be ran outside of the Totem container.

```
FROM nimmis/java:oracle-8-jdk

# enable https for apt
RUN apt-get update
RUN apt-get install -y apt-transport-https

# install scala sbt
RUN echo "deb https://dl.bintray.com/sbt/debian /" | sudo tee /etc/apt/sources.list.d/sbt.list > /dev
RUN apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv 2EE0EA64E40A89B84B2DF73499E82A75642A
RUN apt-get update
RUN apt-get install -y sbt

# setup Holmes-Totem
RUN apt-get install -y git
RUN mkdir /data
WORKDIR /data
RUN git clone https://github.com/HolmesProcessing/Holmes-Totem
WORKDIR /data/Holmes-Totem
RUN sbt assembly

# start totem
CMD java -jar /data/Holmes-Totem/target/scala-2.11/totem-assembly-1.0.jar /data/Holmes-Totem/config/t
```

3.2 Storage

We provide a Docker image for Holmes-Storage, that you can find at
`https://hub.docker.com/r/msxy/holmes-storage/`.

This image is an **unconfigured** Holmes-Storage.

Before using the image, you have to first create a new Holmes-Storage configuration file. See the Holmes-Storage section within the manual installation section.

After you created a configuration file, move it to an empty folder and create a Dockerfile there:

```
FROM msxy/holmes-storage:latest
ADD config.json /data/holmes-storage/
EXPOSE 8016
```

The Dockerfile needs to expose the port configured in the configuration. Further the configuration should bind to the IP address of `0.0.0.0` not localhost.

Now build the image:

```
docker build -t holmes-storage .
```

After you have done this, you can start a Holmes-Storage instance by issuing:

```
docker run holmes-storage
```

If you want to use LocalFS as the object storage, it is highly recommend to mount part of your local filesystem into the docker container to avoid data inconsistency between your database and your object storage:

```
docker run -v "/host/path/:/data/holmes-storage/objstorage-local-fs:rw" holmes-storage
```

If any of the other components is running on your host machine bound to localhost or 127.0.0.1 (for example Cassandra) it is impossible to access them from inside the container whilst it is in bridge mode. In this case you need to share the hosts network stack with your container using the `--net=host` option:

```
docker run --net=host holmes-storage
```

3.3 RabbitMQ

For details see the RabbitMQ image on *hub.docker.com*.

To start and use it, issue the following command on your Docker host:

```
docker run -d --hostname my-rabbit --name some-rabbit rabbitmq:latest
```

3.4 Apache Cassandra

For details see the Apache Cassandra image on *hub.docker.com*.

- Running Cassandra on a single host:

```
        # start a new node
        docker run --name cassandra1 -d cassandra:3.5

        # connect another node to the newly created cluster
        docker run --name cassandra2 -d --link cassandra1:cassandra cassandra:3.5
```

- Running Cassandra on multiple hosts:

```
        # start a new node (substitute 10.42.42.42 by your servers IP)
        docker run --name cassandra1 -d -e CASSANDRA_BROADCAST_ADDRESS=10.42.42.42 -p 7000:7000 cass

        # connect another node to the newly created cluster (substitute 10.43.43.43 by your servers
        docker run --name cassandra2 -d -e CASSANDRA_BROADCAST_ADDRESS=10.43.43.43 -p 7000:7000 -e C
```

CHAPTER 4

Services for Totem

4.1 Communication Protocol

Totem communicates with its services via the http protocol. The request for the service result is of type GET with an empty body.

```
http://address:port/servicename/sampleid
```

The result returned by the service is an appropriate status code, paired with a valid JSON reply body on success or a http error message reflecting the error that occured. (plus optionally a JSON reply body)

```
{
  "key": "value",
  "key": {
    "subkey": "value",
    "subkey": "value"
  }
}
```

This means that your service must act like a webserver. If the service isn't written in a language that offers simple webserver capabilities (like golang), then using a wsgi compatible python wrapper is adviced, like Tornado.

4.2 Accessing Samples

If Totem and the service run on the same host, it is sufficient for the service to be able to access Totems sample storage location. (by default this is /tmp)

When running the service in Docker, the `-v` command line option does the trick to achieve that:

```
docker run -v /tmp:/tmp:ro imagename
```

This maps the hosts /tmp folder read-only to the Docker container providing the service.

However if the service can't access the sample directory, there needs to be a fileserver installed with Totem allowing the service access to the samples stored by Totem. Any fileserver should suffice.

4.3 Writing a Service

4.3.1 Overview

For a service to be accepted into the main repository, the proper additions to the following files need to be made:

- **Config Files**
  - `totem.conf`
  - `docker-compose.yml.example`
  - `compose_download_conf.sh`
- **Scala Files**
  - `driver.scala`

Additionally the following files need to be added:

- **Service Files**
  - `Dockerfile`
  - `LICENSE`
  - `README.md`
  - `acl.conf`
  - `service.conf.example`
  - `watchdog.scala`
  - `<SERVICE_NAME_CAMELCASE>REST.scala`
- **Any additional files your service needs**
  - Python e.g.: `service.py`
  - Go e.g.: `service.go`

Warning: Try to stick to the naming convention (uppercase/lowercase/camelcase were appropriate) to avoid confusion!
If you did not write a service yet, please consult the subcategories.

4.3.2 Details

Files to Edit

The following table declares variables used in the sections below.
These need to be replaced by their respective values.

Compare with other service implementations for suitable values.

SERVICE_NAME	The name of the service, e.g. My_Totem_Service
SERVICE_NAME_CAMELCASE	CamelCase version of your service's name, e.g. MyTotemService
SERVICE_NAME_UPPERCASE	Upper case version of the service's name, e.g. MY_TOTEM_SERVICE
SERVICE_NAME_LOWERCASE	Lower case version of your service's name, e.g. my_totem_service
CONFSTORAGE_SERVICE_NAME	String CONFSTORAGE_ followed by an upper case version of your service's name, e.g. CONFSTORAGE_MY_TOTEM_SERVICE
SERVICE_IP	The IP at which the service is reachable. Usually this should be 127.0.0.1.
SERVICE_PORT	The convention for ports is simple, file processing services are in the `77xx` range, no-file services are in the `79xx` range. Additionally by default there is a gap of 10 free ports per service, so if the first service starts at `7700` the second starts at `7710`.
SERVICE_CLASS_SUCCESS	The name of the service's success class, e.g. MyTotemServiceSuccess
SERVICE_CLASS_WORK	The name of the service's work class, e.g. MyTotemServiceWork

Config Files

The following files can be found in the `config/` folder within the Holmes-Totem repository.

totem.conf.example

- The entry in the totem.conf tells Totem your service exists, where to reach it, and where to store its results.

  The `uri` field supports multiple address entries for automatic load balancing.

```
totem {
    services {
        SERVICE_NAME_LOWERCASE {
            uri = ["http://SERVICE_IP:SERVICE_PORT/analyze/?obj="]
            resultRoutingKey = "SERVICE_NAME_LOWERCASE.result.static.totem"
        }
    }
}
```

docker-compose.yml.example

- Holmes-Totem relies on Docker to provide the services. As such all services need to provide an entry in the docker-compose file.

```
services:
  SERVICE_NAME_LOWERCASE:
    build:
      context: ../src/main/scala/org/holmesprocessing/totem/services/SERVICE_NAME_LOWERCASE
      args:
        conf: ${CONFSTORAGE_SERVICE_NAME}service.conf
    ports:
      - "SERVICE_PORT:8080"
    restart: unless-stopped
```

If the service processes files (i.e. it needs access to `/tmp/` on the host), the following option needs to be added additionally to build, ports, and restart:

```
    volumes:
      - /tmp:/tmp:ro
```

compose_download_conf.sh

- This file runs docker-compose with certain environmental variables set, that allow fetching service configuration files from a server. Add an `export` statement like this:

```
export CONFSTORAGE_SERVICE_NAME=${CONFSTORAGE}zipmeta/
```

> **Warning:** In the above example, `${CONFSTORAGE}` is the actual term and nothing needs to be replaced there.

Scala Files

The following files can be found in the `src/main/scala/org/holmesprocessing/totem/` folder within the Holmes-Totem repository

driver/driver.scala

- Import your services scala classes (see the respective section for information on these classes).

```
import org.holmesprocessing.totem.services.SERVICE_NAME_LOWERCASE.{
    SERVICE_CLASS_SUCCESS,
    SERVICE_CLASS_WORK
}
```

- Add a case to the method `GeneratePartial`

```
def GeneratePartial(work: String): String = {
  work match {
    case "SERVICE_NAME_UPPERCASE" => Random.shuffle(services.getOrElse("SERVICE_NAME_LOWERCASE",
  }
}
```

- Add a case to the method `enumerateWork`.

> **Warning:** If your service does not process files but rather the input string, use `uuid_filename` instead of `orig_filename` below.

```
def enumerateWork(key: Long, orig_filename: String, uuid_filename: String, workToDo: Map[String,
  val w = workToDo.map({
    case ("SERVICE_NAME_UPPERCASE", li: List[String]) => SERVICE_CLASS_WORK(key, orig_filename,
  }).collect({
    case x: TaskedWork => x
  })
  w.toList
}
```

- Add a case to the method `workRoutingKey`

```
def workRoutingKey(work: WorkResult): String = {
  work match {
    case x: SERVICE_CLASS_SUCCESS => conf.getString("totem.services.SERVICE_NAME_LOWERCASE.resul
        }
    }
```

Files to Add

The following table declares variables used in the sections below.

These need to be replaced by their respective values.

Compare with other service implementations for suitable values.

SERVICE_NAME	The name of the service, e.g. My_Totem_Service
SERVICE_NAME_CAMELCASE	CamelCase version of your service's name, e.g. MyTotemService
SERVICE_NAME_UPPERCASE	Upper case version of the service's name, e.g. MY_TOTEM_SERVICE
SERVICE_NAME_LOWERCASE	Lower case version of your service's name, e.g. my_totem_service
CONFSTORAGE_SERVICE_NAME	String CONFSTORAGE_ followed by an upper case version of your service's name, e.g. CONFSTORAGE_MY_TOTEM_SERVICE
SERVICE_IP	The IP at which the service is reachable. Usually this should be 127.0.0.1.
SERVICE_PORT	The convention for ports is simple, file processing services are in the `77xx` range, no-file services are in the `79xx` range. Additionally by default there is a gap of 10 free ports per service, so if the first service starts at `7700` the second starts at `7710`.
SERVICE_CLASS_SUCCESS	The name of the service's success class, e.g. MyTotemServiceSuccess
SERVICE_CLASS_WORK	The name of the service's work class, e.g. MyTotemServiceWork

Service Files

All files in this section belong into the folder `src/main/scala/org/holmesprocessing/totem/services/SERVICE_NA`

Dockerfile In order to make optimal use of Docker's caching ability, you must use the given Dockerfiles and extend them according to your services needs.

- Dockerfile for Python2:

```
FROM python:2-alpine

# add tornado
RUN pip install tornado

# create folder
RUN mkdir -p /service
WORKDIR /service

# add holmeslibrary
RUN apk add --no-cache \
        wget \
    && wget https://github.com/HolmesProcessing/Holmes-Totem-Service-Library/archive/v0.1.tar.gz
    && tar xf v0.1.tar.gz \
    && mv Holmes-Totem-Service* holmeslibrary \
    && rm -rf /var/cache/apk/* v0.1.tar.gz

###
# service specific options
###

# INSTALL SERVICE DEPENDENCIES
#
#   RUN pip install <stuff>
#   RUN apk add --no-cache <stuff>
#   RUN wget <url> && tar xf <stuff> ...
#   ...
#

# add the files to the container
COPY LICENSE /service
COPY README.md /service
COPY service.py /service

# ADD FURTHER SERVICE FILES
#
#   COPY specialLibrary/ /service/specialLibrary
#   COPY extraFile.py /service
#   ...
#

# add the configuration file (possibly from a storage uri)
ARG conf=service.conf
ADD $conf /service/service.conf

CMD ["python2", "service.py"]
```

- Dockerfile for Python3

```
FROM python:alpine

# add tornado
RUN pip3 install tornado

# create folder
RUN mkdir -p /service
WORKDIR /service

# add holmeslibrary
RUN apk add --no-cache \
        wget \
    && wget https://github.com/HolmesProcessing/Holmes-Totem-Service-Library/archive/v0.1.tar.gz
    && tar xf v0.1.tar.gz \
    && mv Holmes-Totem-Service* holmeslibrary \
    && rm -rf /var/cache/apk/* v0.1.tar.gz

###
# service specific options
###

# INSTALL SERVICE DEPENDENCIES
#
#   RUN pip install <stuff>
#   RUN apk add --no-cache <stuff>
#   RUN wget <url> && tar xf <stuff> ...
#   ...
#

# add the files to the container
COPY LICENSE /service
COPY README.md /service
COPY service.py /service

# ADD FURTHER SERVICE FILES
#
#   COPY specialLibrary/ /service/specialLibrary
#   COPY extraFile.py /service
#   ...
#

# add the configuration file (possibly from a storage uri)
ARG conf=service.conf
ADD $conf /service/service.conf

CMD ["python3", "service.py"]
```

- Dockerfile for Go 1.7:

```
FROM golang:alpine

# create folder
RUN mkdir -p /service
WORKDIR /service

# get go dependencies
RUN apk add --no-cache \
        git \
    && go get github.com/julienschmidt/httprouter \
    && rm -rf /var/cache/apk/*

###
# passivetotal specific options
###

# INSTALL SERVICE DEPENDENCIES
#
#   RUN go get <stuff>
#   RUN apk add --no-cache <stuff>
#   RUN wget <url> && tar xf <stuff> ...
#   ...
#

# create directory to hold sources for compilation
RUN mkdir -p src/SERVICE_NAME_LOWERCASE

# add files to the container
# sources files to to GOPATH instead of /service for compilation
COPY LICENSE /service
COPY README.md /service
COPY service.go /service
```

```
# ADD FURTHER SERVICE FILES
#
#   COPY specialLibrary/ /service/specialLibrary
#   COPY extraFile.go /service
#   ...
#

# add the configuration file (possibly from a storage uri)
ARG conf=service.conf
ADD $conf /service/service.conf

# build service
RUN go build -o service.run *.go

# clean up git
# clean up behind the service build
# clean up golang it is not necessary anymore
RUN apk del --purge \
        git \
    && rm -rf /var/cache/apk/* \
    && rm -rf $GOPATH \
    && rm -rf /usr/local/go

CMD ["./service.run"]
```

Warning: If you require a more complex namespacing in your service's code, check out the Passivetotal service's Dockerfile

LICENSE

- The license under which the service is distributed.

README.md

- An appropriate readme file for your service (also displayed if the service's info url is looked up)

  Example:

```
# Passivetotal service for Holmes-Totem

## Description

A simple service to check PassiveTotal for additional enrichment data.
If you do not have an API key, visit http://www.passivetotal.org to get one.

## Usage

Build and start the docker container using the included Dockerfile.

Upon building the Dockerfile downloads a list of TLDs from iana.org.
To update this list of TLDs, the image needs to be built again.

The service accepts domain names, ip addresses and emails as request objects.
These have to be supplied as a parameter after the request URL.
(If the analysisURL parameter is set to /passivetotal, then a request for the
domain www.passivetotal.org would look like this: /passivetotal/www.passivetotal.org)
```

```
    The service performs some checks to determine the type of the input object.
    If a passed domain name contains an invalid TLD, it is invalid and rejected.
    If a passed email address contains an invalid domain, it is rejected.
    If a passed IP is in a reserved range, it is rejected. (ietf rfcs 6890, 4291)

    Only if a request object is determined valid, it is sent to selected passivetotal
    api endpoints. The maximum of simultaneous requests is 9.
    If an error is encountered in any of the api queries, the request fails and returns
    an appropriate error code. Check the local logs for detailed information.
    If the query succeeds, a json struct containing all 9 api end points is returned.
    Those endpoints that were not queried are set to null.
```

acl.conf

- Currently empty

service.conf.example

- JSON file containing service settings like internal port (default must be 8080), but also service specific settings like maybe output limits or parsing limits.

  Example:

```
{
    "port": 8080,
    "max-output-lines": 10000
}
```

watchdog.scala

- Currently empty

Service Logic

- At least one executable file or script is required to run your service.

  This could be for example a Python script or a Go executable.

  Note: See the respective section for the contents of this one.

  TODO: create this documentation section

- All additional files required by your service also belong into the folder `src/main/scala/org/holmesprocessing/totem/services/SERVICE_NAME`.

4.4 Example Service: Hello World

Lets implement a service as simple as possible: Hello World.

4.4.1 totem.conf

```
zoo {
  version = "1.0.0"
  download_directory = "/tmp/"
  requeueKey = "requeue.static.totem"
  misbehaveKey = "misbehave.static.totem"

  rabbit_settings {
    host {
      server = "127.0.0.1"
      port = 5672
      username = "guest"
      password = "guest"
      vhost = "/"
    }
    exchange {
      name = "totem"
      type = "topic"
      durable = true
    }
    workqueue {
      name = "totem_input"
      routing_key = "work.static.totem"
      durable = true
      exclusive = false
      autodelete = false
    }
    resultsqueue {
      name = "totem_output"
      routing_key = "*.result.static.totem"
      durable = true
      exclusive = false
      autodelete = false
    }
  }

  enrichers {
    helloworld {
      uri = ["http://127.0.0.1:8888/helloworld/"]
      resultRoutingKey = "helloworld.result.static.totem"
    }
  }
}
```

4.4.2 driver.scala

```scala
package org.novetta.zoo.driver

import java.util.concurrent.{Executors, ExecutorService}

import akka.actor.{ActorRef, ActorSystem, Props}
import org.novetta.zoo.actors._
import org.novetta.zoo.services.helloworld{HelloWorldSuccess, HelloWorldWork}
import org.novetta.zoo.types._

import org.json4s._
import org.json4s.JsonDSL._
import org.json4s.jackson.JsonMethods._
import akka.routing.RoundRobinPool
import org.novetta.zoo.util.Instrumented

import java.io.File

import com.typesafe.config.{Config, ConfigFactory}

import scala.util.Random

object driver extends App with Instrumented {
  lazy val execServ: ExecutorService = Executors.newFixedThreadPool(4000)
  val conf: Config = if (args.length > 0) {
    println("we have args > 0, using args")
    ConfigFactory.parseFile(new File(args(0)))

  } else {
    ConfigFactory.parseFile(new File("/Users/totem/novetta/totem/config/totem.conf"))
  }
  val system = ActorSystem("totem")

  val hostConfig = HostSettings(
    conf.getString("zoo.rabbit_settings.host.server"),
    conf.getInt("zoo.rabbit_settings.host.port"),
    conf.getString("zoo.rabbit_settings.host.username"),
    conf.getString("zoo.rabbit_settings.host.password"),
    conf.getString("zoo.rabbit_settings.host.vhost")
  )

  val exchangeConfig = ExchangeSettings(
    conf.getString("zoo.rabbit_settings.exchange.name"),
    conf.getString("zoo.rabbit_settings.exchange.type"),
    conf.getBoolean("zoo.rabbit_settings.exchange.durable")
  )
  val workqueueConfig = QueueSettings(
    conf.getString("zoo.rabbit_settings.workqueue.name"),
    conf.getString("zoo.rabbit_settings.workqueue.routing_key"),
    conf.getBoolean("zoo.rabbit_settings.workqueue.durable"),
    conf.getBoolean("zoo.rabbit_settings.workqueue.exclusive"),
    conf.getBoolean("zoo.rabbit_settings.workqueue.autodelete")
  )
  val resultQueueConfig = QueueSettings(
    conf.getString("zoo.rabbit_settings.resultsqueue.name"),
    conf.getString("zoo.rabbit_settings.resultsqueue.routing_key"),
    conf.getBoolean("zoo.rabbit_settings.resultsqueue.durable"),
    conf.getBoolean("zoo.rabbit_settings.resultsqueue.exclusive"),
    conf.getBoolean("zoo.rabbit_settings.resultsqueue.autodelete")
  )

  class TotemicEncoding(conf: Config) extends ConfigTotemEncoding(conf) {
    def GeneratePartial(work: String): String = {
      work match {
        case "HELLOWORLD" => Random.shuffle(services.getOrElse("helloworld", List())).head
      }
    }

    def enumerateWork(key: Long, filename: String, workToDo: Map[String, List[String]]): List[TaskedW
```

```
      val w = workToDo.map({
        case ("HELLOWORLD", li: List[String]) =>
          HelloWorldWork(key, filename, 60, "HELLOWORLD", GeneratePartial("HELLOWORLD"), li)

        case (s: String, li: List[String]) =>
          UnsupportedWork(key, filename, 1, s, GeneratePartial(s), li)

        case _ => Unit
      }).collect({
        case x: TaskedWork => x
      })
      w.toList
    }

    def workRoutingKey(work: WorkResult): String = {
      work match {
        case x: HelloWorldSuccess => "helloworld.result.static.zoo"
      }
    }
  }

  val encoding = new TotemicEncoding(conf)

  val myGetter: ActorRef = system.actorOf(RabbitConsumerActor.props[ZooWork](hostConfig, exchangeCon
  val mySender: ActorRef = system.actorOf(Props(classOf[RabbitProducerActor], hostConfig, exchangeCon


  // Demo & Debug Zone
  val zoowork = ZooWork("http://localhost/rar.exe", "http://localhost/rar.exe", "winrar.exe", Map[Str

  val json = (
    ("primaryURI" -> zoowork.primaryURI) ~
      ("secondaryURI" -> zoowork.secondaryURI) ~
      ("filename" -> zoowork.filename) ~
      ("tasks" -> zoowork.tasks) ~
      ("attempts" -> zoowork.attempts)
    )

  private[this] val loading = metrics.timer("loading")

  val j = loading.time({
    compact(render(json))
  })

  mySender ! Send(RMQSendMessage(j.getBytes, workqueueConfig.routingKey))

  println("Totem is Running! \nVersion: " + conf.getString("zoo.version"))
}
```

4.4.3 HelloWorldREST.scala

```
package org.novetta.zoo.services.zipmeta

import dispatch.Defaults._
import dispatch.{url, _}
import org.json4s.JsonAST.{JString, JValue}
import org.novetta.zoo.types.{TaskedWork, WorkFailure, WorkResult, WorkSuccess}
import collection.mutable


case class HelloWorldWork(key: Long, filename: String, TimeoutMillis: Int, WorkType: String, Worker:
  def doWork()(implicit myHttp: dispatch.Http): Future[WorkResult] = {

    val uri = HelloWorldREST.constructURL(Worker, filename, Arguments)
    val requestResult = myHttp(url(uri) OK as.String)
      .either
      .map({

      case Right(content) =>
        HelloWorldSuccess(true, JString(content), Arguments)

      case Left(something) =>
        HelloWorldFailure(false, JString("Hello World failed ... :/ (maybe the service isn't up and r

    })
    requestResult
  }
}


case class HelloWorldSuccess(status: Boolean, data: JValue, Arguments: List[String], routingKey: Stri
case class HelloWorldFailure(status: Boolean, data: JValue, Arguments: List[String], routingKey: Stri


object HelloWorldREST {
  def constructURL(root: String, filename: String, arguments: List[String]): String = {
    arguments.foldLeft(new mutable.StringBuilder(root+filename))({
      (acc, e) => acc.append(e)}).toString()
  }
}
```

4.4.4 Dockerfile

```
# choose the operating system image to base of, refer to docker.com for available images
FROM ubuntu:14.04

# install what you need here
RUN apt-get update && apt-get -y upgrade
RUN apt-get install -y python python-pip
RUN pip install tornado

# create a folder to contain your service's files
RUN mkdir -p /service

# add all files relevant for running your service to your container
ADD helloworld.py /service

# create a new user with limited access
RUN useradd -s /bin/bash service
RUN chown -R service /service

# switch user and work directory
USER service
WORKDIR /service
```

```
# expose our container on some port
EXPOSE 8888

# define our command to run if we start our container
CMD python2 /service/helloworld.py
```

4.4.5 helloworld.py

```
import tornado
import tornado.web
import tornado.httpserver
import tornado.ioloop

import os
from os import path


class Service (tornado.web.RequestHandler):
    def get(self, filename):
        data = {
            "message": "Hello World!"
        }
        self.write(data)


class Info(tornado.web.RequestHandler):
    def get(self):
        description = """
            <p>Copyright 2015 Holmes Processing
            <p>Description: This is the HelloWorld Service for Totem.
        """
        self.write(description)


class Application(tornado.web.Application):
    def __init__(self):
        handlers = [
            (r'/', Info),
            (r'/yourservice/([a-zA-Z0-9\-]*)', Service),
        ]
        settings = dict(
            template_path=path.join(path.dirname(__file__), 'templates'),
            static_path=path.join(path.dirname(__file__), 'static'),
        )
        tornado.web.Application.__init__(self, handlers, **settings)
        self.engine = None


def main():
    server = tornado.httpserver.HTTPServer(Application())
    server.listen(8888)
    tornado.ioloop.IOLoop.instance().start()


if __name__ == '__main__':
    main()
```

Totem's Services are pretty simple to understand. They build upon JSON as a messaging format, use a RESTful API and are completely independent of Totem itself. The Scala you need to add one is, in most cases, as simple as it can be: copy and paste.

This chapter contains detailed information about how Services communicate with Totem, access samples, how to write a new Service and a Service example.