Open Source Knowledge Enrichment

Installation Guide

Version 1.0

2 May 2019

Table of Contents

[1 Overview 5](#_Toc18408165)

[1.1 System Components 5](#_Toc18408166)

[1.1.1 External Components 5](#_Toc18408167)

[1.1.2 Internal Components 6](#_Toc18408168)

[1.1.3 System Diagram 6](#_Toc18408169)

[1.2 Minimum System Requirements 8](#_Toc18408170)

[1.3 System Sizing 9](#_Toc18408171)

[1.4 Document Conventions 9](#_Toc18408172)

[2 Administration Preparation 10](#_Toc18408173)

[3 System Preparation 11](#_Toc18408174)

[3.1 Firewall 11](#_Toc18408175)

[3.2 Editing Files on the Server 11](#_Toc18408176)

[3.3 Custom Users 11](#_Toc18408177)

[4 Database Server: PostgreSQL 12](#_Toc18408178)

[4.1 Provision Server 12](#_Toc18408179)

[4.2 PostgreSQL Install 12](#_Toc18408180)

[5 Elasticsearch Stack Machine 15](#_Toc18408181)

[5.1 Provision Server 15](#_Toc18408182)

[5.2 Elasticsearch and Kibana 15](#_Toc18408183)

[5.3 Cerebro 17](#_Toc18408184)

[6 Microservices Stack 19](#_Toc18408185)

[6.1 Provision Server 19](#_Toc18408186)

[6.2 Docker 19](#_Toc18408187)

[6.3 GitHub 20](#_Toc18408188)

[6.4 Microservices 20](#_Toc18408189)

[7 OSKE Stack / Server 22](#_Toc18408190)

[7.1 Provision Server 22](#_Toc18408191)

[7.2 PostgreSQL Client Install 23](#_Toc18408192)

[7.3 Custom Users 23](#_Toc18408193)

[7.4 Tomcat 23](#_Toc18408194)

[7.5 Voyant 25](#_Toc18408195)

[7.6 NGINX / HTTPS 26](#_Toc18408196)

[7.7 GitHub 36](#_Toc18408197)

[7.8 Software Builds 36](#_Toc18408198)

[7.9 Collector Daemon 37](#_Toc18408199)

[7.10 Collector Web Application 40](#_Toc18408200)

[8 Appendix A: AWS Notes 42](#_Toc18408201)

[9 Appendix B: Parameter Worksheet 43](#_Toc18408202)

[10 Appendix C: System Port Usage 45](#_Toc18408203)

[11 Appendix D: System User and Groups 47](#_Toc18408204)

[12 Appendix E: Change History 48](#_Toc18408205)

Table of Figures

[Figure 1 OSKE System Diagram 6](#_Toc18408206)

[Figure 2 Alternate System Diagram with Network Ports 7](#_Toc18408207)

[Figure 3 Google Console - Create Project 30](#_Toc18408208)

[Figure 4 Google Console - Access Credentials 30](#_Toc18408209)

[Figure 5 Google Console - Credentials 31](#_Toc18408210)

[Figure 6 Google Console - Choose Credential Type 31](#_Toc18408211)

[Figure 7 Google Console - Create Web Application ID 32](#_Toc18408212)

Table of Tables

[Table 1 External System Components 4](#_Toc18408213)

[Table 2 Internal System Components 5](#_Toc18408214)

[Table 3 Network Port Connectivity Requirements 7](#_Toc18408215)

[Table 4 Feasible AWS Instance Types 8](#_Toc18408216)

[Table 5 Useful Administrative Tools 9](#_Toc18408217)

[Table 6 Useful VI Commands 10](#_Toc18408218)

[Table 7 OSKE Install Parameters 43](#_Toc18408219)

[Table 8 System Port Usage 45](#_Toc18408220)

[Table 9 OSKE Users 46](#_Toc18408221)

[Table 10 OSKE Groups 46](#_Toc18408222)

[Table 11 Change History 47](#_Toc18408223)

# Overview

Open Source (OS) Knowledge Enrichment (OSKE/OpenKE[[1]](#footnote-1)) is a system to manage the complexities of capturing, formatting, manipulating, and understanding publicly available information. The internet-connected platform hosts techniques and analytics designed to be semi-automated and allow for configuration to scan a wide range of structured and unstructured data sources. The system helps capture the ”right” (relevant) information from the ”right” (reliable) sources, and the system provides capabilities to holistically analyze big data within an internet-connected environment and integrate data of value into the mission environment.

## System Components

OSKE utilizes many different components to provide a wide variety of capabilities to discover, capture, recall, and manage publicly available information. While these components may all run on a single server, consider spreading the components across multiple servers; this may be dictated by performance requirements. This install guide assumes four separate servers and removes dependency on a “big data” processing cluster such as Hortonworks Data Platform for Accumulo, HDFS, and Kafka. Refer to the *Administrator’s Guide* for details on sending data to HDFS.

### External Components

External components are either necessary infrastructure components (e.g., database server) or applications that have been developed by others. At a minimum, Elasticsearch and PostgreSQL must be utilized. Other components can be utilized by an organization’s needs.

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Required | Version | Purpose |
| Cerebro | No | 0.8.3 | Web administration tool for ElasticSearch  <https://github.com/lmenezes/cerebro> |
| Elasticsearch | Yes | 7.0.0 + | Stores crawled data from a variety of information sources. Requires at least 7.0.0 due to [Elasticsearch’s removal of types](https://www.elastic.co/guide/en/elasticsearch/reference/current/removal-of-types.html).  <https://www.elastic.co/products/elasticsearch> |
| Kibana | No | Match Elasticsearch | Provide visualization and monitoring capabilities of the system. Also provides a rest-based query page.  <https://www.elastic.co/products/kibana> |
| NGINX | No | 1.15.3 | Reverse proxy for access various web interfaces, provides SSL support, provides authentication with oauth2 and JWT support. |
| PostgreSQL | Yes | 9.4+,10+,11+ | Provides the management database for the application |
| Voyant | No | 2.2 + | Provides text analysis tools  <https://github.com/sgsinclair/VoyantServer> |

Table 1 External System Components

### Internal Components

Internal components are the actual OSKE system components. These components must be installed.

|  |  |
| --- | --- |
| Name | Description |
| Daemon | Java-based service that continually runs to scrap open-source content from the Internet. |
| Microservices | Variety of micro-services running as docker containers that provide additional ways to augment information for the system. |
| Web Application | End-user interface to explore new information, establish retrieval jobs, and search for information. |

Table 2 Internal System Components

### System Diagram

Figure 1 shows the interaction among the different components as well as how components can be organized together – both logically and physically.

Users make requests through the web application to establish jobs, discover new information, and perform other tasks. Both the daemon (through defined, scheduled jobs) and the web application (through domain discovery) access information on the internet and then process the data. The raw content is stored on the filesystem. The extracted information, which is formatted as a JSON object, is stored in Elasticsearch. Kibana can be used to provide analytics and visualize the stored JSON records. The microservices are called by the daemon, web application, and graph creation components. Exports of the JSON records can be sent/transferred to other systems (e.g., Spark / MapReduce clusters for analytics, but such functionality is outside the scope of the delivered system).

Components within the same “stack” boxes are generally executed on the same machine. For Elasticsearch, these components can be configured across a series of machines to form an Elasticsearch cluster if necessary. For the collector, at least one daemon must run on the same machine as the web application[[2]](#footnote-2). However, multiple daemon processes may be run on the same or different machines. The various microservices can run on multiple machines and with multiple instantiations if required by processing demands. The other components can be co-located or separated as each situation warrants.

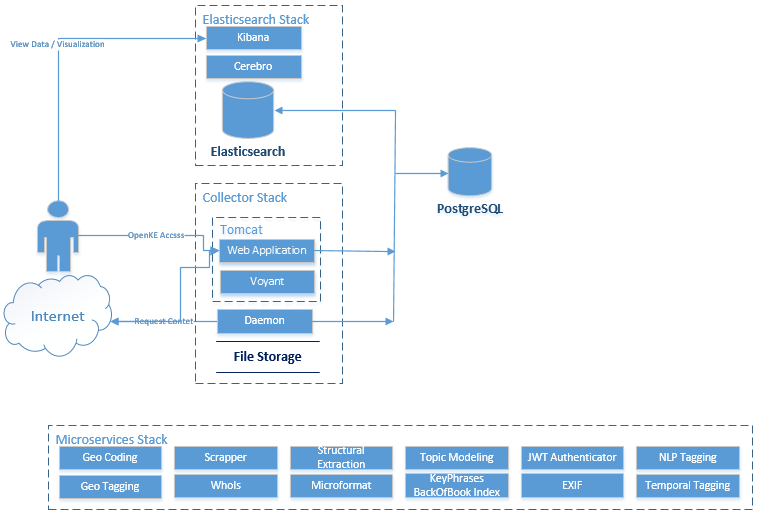


Figure 1 OSKE System Diagram

Figure 2 provides an alternate system diagram. Networking ports are recorded to assist with configuring any firewalls (including firewalld/iptables) and/or security groups (i.e., from AWS to protect EC2 instances).

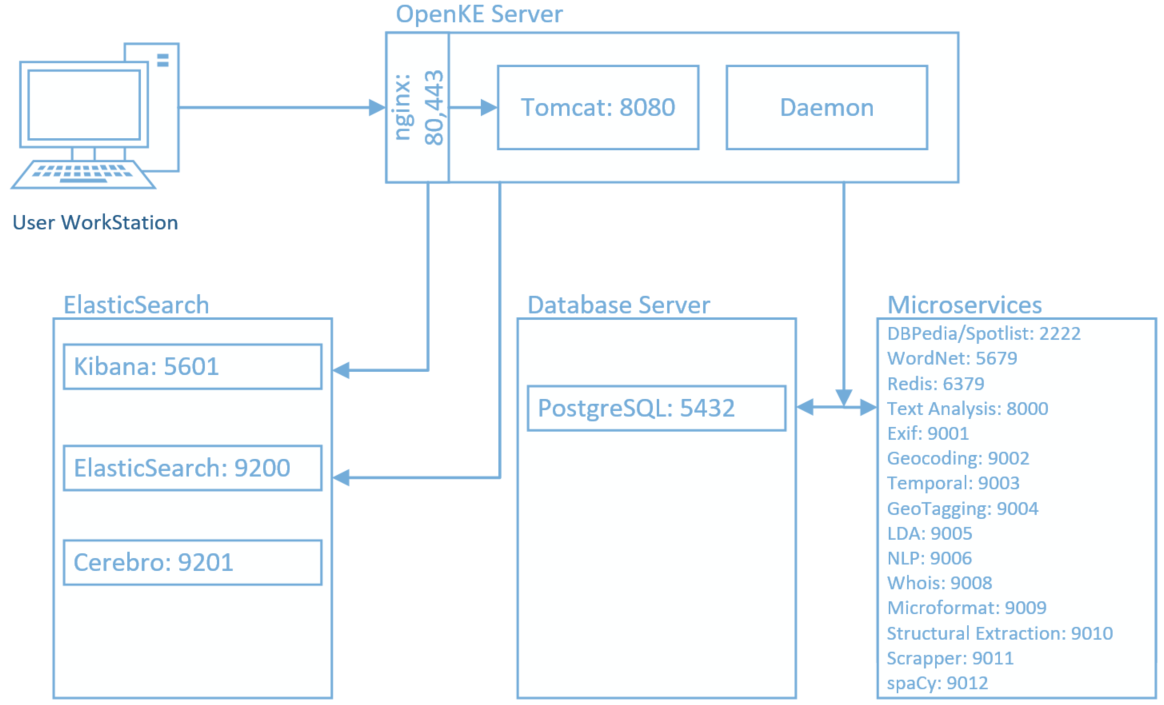


Figure 2 Alternate System Diagram with Network Ports

|  |  |
| --- | --- |
| Server | Network Accessibility |
| Database | OSKE and Docker Stack on port 5432 |
| OpenKE | “World” accessible on ports 80 and 443. Port 80 redirects to 443 |
| ElasticSearch | OpenKE and Docker Stack on port 9200 (ElasticSearch).  OpenKE for port 5601 (Kibana)  OpenKE for port 9201 (Cerebro) |
| Docker Stack | OpenKE on ports: 2222 ,5679 ,8000, 9001, 9002, 9003, 9004, 9005, 9006, 9008, 9009, 9010, 9011, 9012  (6379 is only used internally for the Text Analysis microserver) |

Table 3 Network Port Connectivity Requirements

## Minimum System Requirements

The full-suite of tools can execute on a single Linux server with 64gb of RAM, 100gb storage, and 8 CPU threads. Ideally, components should be spread apart. Exact specifications will depend upon the organization’s users, interacting usage, jobs, and collected data volume.

This document will describe the installation on 4 separate machines:

|  |  |
| --- | --- |
| Server / Purpose | Possible EC2 Instances |
| Database | M5 xlarge |
| OpenKE | M5 2xlarge |
| ElasticSearch | M5 4xlarge, R5 2xlarge |
| Docker Stack | M5 4xlarge, R5 2xlarge |

Table 4 Feasible AWS Instance Types

## System Sizing

System sizing will largely depend upon the volume of data collected and how fast (velocity) data collection needs to occur. Generally speaking, the OSKE components are memory constrained rather than processing based.

For the Tomcat service, allocate 2-8 gigabytes (gb).

For Elasticsearch, allocate at least 4gb and no more than 31gb. Additionally, do not allocate over 50% of the server’s memory for Elasticsearch. The Lucene search engine will dynamically use the remaining space through disk caches. <https://www.elastic.co/guide/en/elasticsearch/guide/current/heap-sizing.html>

## Document Conventions

Within this document, statements in a fixed-width font (e.g., Consolas) are commands to be executed or text that should be placed into files.

Names in bold and italicized are parameters to be replaced by installation specific values. The full list of these values is contained in Appendix B.

Unless specified otherwise, commands in this document must be executed as root. Either become root with “sudo su –” or prefix each command with “sudo”. This document assumes an installation on a CentOS 7.x or later platform. Appropriate adjustments will need to be made for other linux systems. For most install commands, the -y flag was excluded so the installation components can be reviewed.

Optional parameters are specified with square brackets. [ ]

<ctrl-d> means to press the ctrl and d keys at the same time.

Component versions change quite frequently both for bug and security fixes. Unless otherwise specified, use the latest versions whenever possible.

# Administration Preparation

To administer the OSKE system, the following software components should be installed:

|  |  |
| --- | --- |
| Software | Usage |
| pgadmin | GUI used to administrate the PostreSQL database instead of the pgsql command-line. |
| putty | Terminal window to ssh into a server. On MacOS, use the installed terminal program. |
| puttygen | Used to create SSH keypairs. On MacOS, utilize the ssh-keygen command line program |
| Atom / Notepad++ | Text editor – will be useful for making substitutions from the install directions in this guide. |

Table 5 Useful Administrative Tools

# System Preparation

This section covers the general setup for the servers/virtual machines.

General note: For many of the configuration files, only the applicable line changes are present.

System provisioning notes:

* All 4 hosts should be placed in the same availability zone
* Configure AWS security groups. Look at Figure #2 and Table #3 to define these rules.
* Provision the OSKE Server first, but save the installation for after the other 3 servers have been installed. By provisioning this server first, the security groups for the other servers can be configured correctly.

## Firewall

From a pure-convenience standpoint, it may just be easiest to disable internal firewalls among the various machines due to the number of ports used by the various components. (See Appendix C: System Port Usage for more details on the ports used). However, nearly all ports should not be freely available for public access – the primary exception is access to the web application. This document does step through setting up NGINX and SSL such that port 443 / https is used to access the web application. Further, the document describes using OAuth2 to authenticate the user at the NGINX layer, which prevents unauthenticated users from reaching any of the OSKE components.

## Editing Files on the Server

For this installation guide, vi (vim) is used to edit files on the server. Full documentation is available at <https://www.vim.org/>

The necessary editing changes should be able to be accomplished with these commands:

|  |  |
| --- | --- |
| Command | Usage |
| dd | delete the current line |
| i | enter “insert” mode to make text changes |
| <escape> | leave “insert” mode and enter “command” mode |
| :wq | write the file and quit vi (e.g., “save”) |
| :%s/searchValue/replaceValue/g | replace all values of “searchValue” with “replaceValue” |

Table 6 Useful VI Commands

## Custom Users

The OSKE system utilizes many different users and groups (see Appendix D: System User and Groups for the complete listing and use). Most of these accounts are auto-created through the installation process. Specific accounts are detailed for each of the four servers.

# Database Server: PostgreSQL

PostgreSQL is used as the management database – keeps track of the domains, configured jobs, executed crawls, and visited pages. Any version 9.4+ should work without issues. The project has utilized 9.4, 9.5, 9.6, 10.2, and 11.2. pgAdmin (<https://www.pgadmin.org/>) can be used as a client-side GUI to work with the database.

## Provision Server

1. Launch an AWS instance:

*EC2 Type*: M5 xlarge (16gb RAM should be sufficient)

*Image*: CentOS Linux 7 x86\_64 HVM EBS ENA 1805\_01 - ami-77ec9308

*Storage size*: 250gb

*Security Group*: The OSKE and microservices servers will need to contact this server through port 5432 on the private IP address space

1. Go back to the EC2 instance listing and add a Name (eg, “OSKE – DB”)
2. Access the server. Will utilize the “centos” user along with the appropriate private key.
3. Become the root user:

sudo su –

1. Update the server

yum update

yum install -y epel-release wget unzip

1. Install and enable ntp (network time synchronization)

yum install ntp

systemctl enable ntpd

systemctl start ntpd

## PostgreSQL Install

1. Install the official repo. Yum repositories are at <https://yum.postgresql.org/repopackages.php>

wget <https://download.postgresql.org/pub/repos/yum/10/redhat/rhel-7-x86_64/pgdg-centos10-10-2.noarch.rpm>

rpm -Uvh pgdg-centos10-10-2.noarch.rpm

1. Install PostgreSQL (-contrib is used for the pg-crypto library)

yum install postgresql10-server postgresql10-contrib

1. Initialize the database. UTF8 should be specified as the default encoding. By default, this places the database into /var/lib/pgsql/10/data. If a different location is utilized, modify the location directory after the “-D” flag.

sudo su – postgres

/usr/pgsql-10/bin/initdb -E UTF8 -D /var/lib/pgsql/10/data

/usr/pgsql-10/bin/pg\_ctl -D /var/lib/pgsql/10/data -l logfile start

/usr/pgsql-10/bin/pg\_ctl -D /var/lib/pgsql/10/data -l logfile stop

1. Allow for remote access and ensure local ip connections are authenticated (comment out the IPv4 local connections, IPv6 local connections, and then add the last line). The first line is used to authenticate the postgres user automatically into the database server while executing psql. Optionally, a password may be set for the postgres user and this method can be removed as well.

vi /var/lib/pgsql/10/data/pg\_hba.conf

# TYPE DATABASE USER ADDRESS METHOD

# "local" is for Unix domain socket connections only

local all all peer

# IPv4 local connections:

#host all all 127.0.0.1/32 ident

# IPv6 local connections:

#host all all ::1/128 ident

# Allow replication connections from localhost, by a user with the

# replication privilege.

#local replication postgres peer

#host replication postgres 127.0.0.1/32 ident

#host replication postgres ::1/128 ident

host all all 0.0.0.0/0 md5

1. Configure PostgreSQL such that it will listen in on an all defined addresses. Otherwise, the database server will only listen on the localhost (127.0.0.1).

echo "listen\_addresses = '\*'" >> /var/lib/pgsql/10/data/postgresql.conf

1. Switch back to the root user:

exit

1. Enable the service and start.

systemctl enable postgresql-10

systemctl start postgresql-10

1. Create the OSKE database user and database

sudo su - postgres

psql

CREATE USER ***POSTGRESQL\_OPENKE\_USER*** WITH PASSWORD '***POSTGRESQL\_OPENKE\_PASSWORD***';

CREATE DATABASE ***POSTGRESQL\_OPENKE\_DATABASE*** OWNER ***POSTGRESQL\_OPENKE\_USER***;

\connect ***POSTGRESQL\_OPENKE\_DATABASE***

CREATE EXTENSION pgcrypto;

<ctrl-d>

1. Install the table scripts form the Collector GitHub project (Collector/sql/database\_setup.sql). If the install is on the same server as the database server, substitute 127.0.0.1 for ***FULL\_DOMAIN\_NAME***.

psql -U ***POSTGRESQL\_OPENKE\_USER*** -h ***FULL\_DOMAIN\_NAME*** -d ***POSTGRESQL\_OPENKE\_DB*** \

-W f database\_setup.sql

Use “\d” to verify the tables created within psql. 24 tables should now exist.

The initial records (“system” domain and system administrator) for the OSKE system will be installed as part of the daemon installation.

# Elasticsearch Stack Machine

This server runs Elasticsearch and Kibana.

## Provision Server

1. Launch an AWS instance:

*EC2 Type*: M5 4xlarge (the R5 2xlarge is also acceptable)

*Image*: CentOS Linux 7 x86\_64 HVM EBS ENA 1805\_01 - ami-77ec9308

*Storage*: 1TB

*Security Group*: The OSKE server will need to contact Elasticsearch(9200) and Kibana(5601) on the private ip address space. The microservices server will contact this server as well on port 9200.

1. Go back to the EC2 instance listing and add a Name (eg, “OSKE – Elastic”)
2. Access the server. Will utilize the “centos” user along with the appropriate private key.
3. Become the root user: sudo su –
4. Update the server, install java and other utilities

yum update

yum install -y epel-release wget unzip

yum install -y java-1.8.0-openjdk-devel

1. Install and enable ntp (network time synchronization)

yum install -y ntp

systemctl enable ntpd

systemctl start ntpd

## Elasticsearch and Kibana

Elasticsearch is the primary application repository used to store JSON records, provide search capabilities, and an initial set of analytics. Additionally, the instrumentation events can put into a separate index within Elasticsearch.

Kibana is used as visualization platform.

Indexes are automatically created as needed by the Collector system as either the daemon or web application is started.

1. Install the Elasticsearch key and repo

rpm --import <https://artifacts.elastic.co/GPG-KEY-elasticsearch>

cat > /etc/yum.repos.d/elasticsearch.repo << EOF

[elasticsearch-7.x]

name=Elasticsearch repository for 7.x packages

baseurl=https://artifacts.elastic.co/packages/7.x/yum

gpgcheck=1

gpgkey=https://artifacts.elastic.co/GPG-KEY-elasticsearch

enabled=1

autorefresh=1

type=rpm-md

EOF

1. Install Elasticsearch and Kibana

yum install -y elasticsearch kibana

1. Fix networking for Elasticsearch. Listen on all IPs (supported in 5.0+) and allow CORS support. The “discovery.type” allows Elasticsearch to run as a single node. If Elasticsearch runs in a cluster, remove this line.

cat >> /etc/elasticsearch/elasticsearch.yml << EOF

network.host: 0.0.0.0

http.cors.enabled: true

http.cors.allow-origin: "\*"

discovery.type=single-node

EOF

1. Change the memory used by Elasticsearch. The below example changes from the default of 2gb to 24gb. Elasticsearch recommends that the minimum and maximum memory be set the change such that memory is completely allocated at the time Elasticsearch starts. Do not set Elasticsearch to use more than 50% of the available RAM for the server. The other 50% would then be utilized by Lucene and is dynamically used through cache support. Additionally, due to how Java utilizes pointer sizes, make the memory size less than 32 gigabytes.

vi /etc/elasticsearch/jvm.options

-Xms24g

-Xmx24g

1. Enable and start daemon processes for Elasticsearch

systemctl enable elasticsearch

systemctl start elasticsearch

1. Validate Elasticsearch has successfully started. (either command should be suffice)

curl localhost:9200

curl ***FULL\_DOMAIN\_NAME\_ELASTIC***:9200

1. Update Kibana configuration for networking and name support. If Elasticsearch is run on a different server than Kibana, change elasticsearch.url by changing localhost to ***FULL\_DOMAIN\_NAME\_ELASTIC***.

cat >> /etc/kibana/kibana.yml << EOF

server.host: 0.0.0.0

server.name: "OpenKE\_Kibana"

elasticsearch.hosts: <http://localhost:9200>

server.rewriteBasePath: false

EOF

1. Enable and start daemon processes for Kibana

systemctl enable kibana

systemctl start kibana

1. Verify that Kibana has started

curl ***FULL\_DOMAIN\_NAME\_ELASTIC***:5601

If problems do exist starting, most likely a permission issue exists. Review the contents of /var/log/messages

## Cerebro

Cerebro is a web-based user interface to help manage Elasticsearch. Originally is was built as an Elasticsearch plug-in and named “Kopf”. However, in Elasticsearch 5.x, user plugins were no longer supported by Elasticsearch and the developer transitioned the tool to execute as a separate web application. See <https://github.com/lmenezes/cerebro> for more details. This typically is installed on the same host/server as Elasticsearch. This install is optional.

1. Create a group and user for Cerebro. This user is only necessary if Cerebro will be utilized. Cerebro may also execute as an existing user. Adjust the systemd configuration to specify the user and group if another user and group is selected.

groupadd -g 899 cerebro

useradd -g 899 -u 899 -d /opt/cerebro cerebro

1. Download and extract the software. The latest releases are available at <https://github.com/lmenezes/cerebro/releases>

If necessary, adjust the versions as necessary in the commands below. The “current” directory is used to make upgrades slightly easier as only that softlink would need to be updated as a newer version is overlaid.

cd /opt/cerebro

wget <https://github.com/lmenezes/cerebro/releases/download/v0.8.3/cerebro-0.8.3.tgz>

tar xvf cerebro-0.8.3.tgz

ln -s cerebro-0.8.3 current

1. Create a start script

cat > /opt/cerebro/cerebroStart.sh << EOF

#!/bin/bash

/opt/cerebro/current/bin/cerebro -Dhttp.port=9201 -Dhttp.address=0.0.0.0 &

EOF

1. Allow the start script to be executed:

chmod 755 cerebroStart.sh

1. Remove the gzipped tar file

rm -f cerebro-0.8.3.tgz

1. Set the ownership of the files to cerebro

chown -R cerebro:cerebro /opt/cerebro

1. Create a service

cat > /etc/systemd/system/cerebro.service << EOF

[Unit]

Description=Cerebro - elasticsearch web admin tool

After=network.target

After=syslog.target

[Install]

WantedBy=multi-user.target

[Service]

Type=forking

User=cerebro

Group=cerebro

ExecStart=/opt/cerebro/cerebroStart.sh

EOF

1. Setup the application path such that the server doesn’t listen on the root path. This will allow us to use a reverse proxy to access the server.

vi /opt/cerebro/current/conf/application.conf

Change the application base path:

# Application base path

basePath = "/cerebro/"

1. Enable the server to start when the server starts and then start the service

systemctl enable cerebro

systemctl start cerebro

1. Validate that service is running –

curl ***FULL\_DOMAIN\_NAME\_ELASTIC***:9201

# Microservices Stack

This server runs Docker containers (i.e., microservices) to provide additional functionality to the OSKE system. Additionally, these containers can exist independently for use by other systems. These containers were initially developed to minimize dependency conflicts within the OSKE application.

## Provision Server

1. Launch an AWS instance: (search under community AMIs)

*EC2 Type*: M5 4xlarge (the R5 2xlarge is also acceptable)

*Image*: CentOS Linux 7 x86\_64 HVM EBS ENA 1805\_01 - ami-77ec9308

*Storage size*: 100gb

1. Go back to the EC2 instance listing and add a Name (eg, “OSKE – Docker”)
2. Access the server. Will utilize the “centos” user along with the appropriate private key.
3. Become the root user: sudo su –
4. Update the server, install utilities.

yum update

yum install -y epel-release wget unzip git

1. Install and enable ntp (network time synchronization)

yum install -y ntp

systemctl enable ntpd

systemctl start ntpd

## Docker

Docker hosts the microservice stack. Eventually, other components will also run within containers. While the CentOS EPEL repo does contain Docker, that version is not current. The installation instructions for docker for CentOS are at <https://docs.docker.com/install/linux/docker-ce/centos/>

1. Install required packages for docker

yum install -y yum-utils device-mapper-persistent-data lvm2

1. Add the docker-ce repo from docker.com

yum-config-manager --add-repo \

<https://download.docker.com/linux/centos/docker-ce.repo>

1. Install docker Community Edition

yum install docker-ce

1. Install docker compose (Note: there are just two commands to execute.)

curl -L "https://github.com/docker/compose/releases/download/1.24.0/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose

chmod +x /usr/local/bin/docker-compose

1. Enable and start the service

systemctl enable docker

systemctl start docker

## GitHub

To install the microservices, the GitHub repository must be accessible.

Download the OSKE repository via clone or download operation:

mkdir /root/git

cd /root/git

git clone git@github.com:LAS-NCSU/OpenSourceKnowledgeEnrichment.git

## Microservices

To install the microservices, there is a docker-compose.yml file that will perform the necessary build steps.

With two exceptions (DBPedia Spotlight - <https://github.com/dbpedia-spotlight/spotlight-docker> and WordNet - <https://github.com/jacopofar/wordnet-as-a-service>), the docker images are built from the OSKE code repository.

To build and start the Microservices:

cd /root/git/OpenSourceKnowledgeEnrichment/Microservice

vi docker-compose.yml

Modify these variables within the file:

* IDENTENIFYING\_EMAIL\_ADDRESS
* LOCATION\_IQ\_KEY (see Location IQ registration below)
* FULL\_DOMAIN\_NAME\_POSTGRESQL
* POSTGRESQL\_PORT
* POSTGRESQL\_OPENKE\_DB
* POSTGRESQL\_OPENKE\_USER
* POSTGRESQL\_OPENKE\_PASSWORD

docker-compose up -d

To stop the Microservices:

cd /root/git/OpenSourceKnowledgeEnrichment/Microservice

docker-compose up -d

To register for a Location IQ Key, visit <https://locationiq.com/register>

The “free” level should be sufficient. Location IQ can also be removed as provider for the geocoding service.

To view the run status of the docker services:: docker ps -a

To view all of the logs : docker-compose logs --follow

# OSKE Stack / Server

This server executes both the web application for user interaction as well as the daemon process that executes jobs to collect data from the Internet on a scheduled basis. The server will require larger amounts of storages as the raw data collected is also stored on the file system.

This setup installs the daemon into /opt/collector and the web application (via Tomcat) into /opt/tomcat

## Provision Server

1. Launch an AWS instance:

*EC2 Type*: M5 2xlarge

*Image*: CentOS Linux 7 x86\_64 HVM EBS ENA 1805\_01 - ami-77ec9308

*Storage size*: 100gb

1. Go back to the EC2 instance listing and add a Name (eg, “OSKE – Application”)
2. Access the server. Will utilize the “centos” user along with the appropriate private key.
3. Become the root user: sudo su –
4. Update the server, install java and other utilities. Development tools is needed to build nginx. Java may already be installed from the maven installation.

yum update -y

yum install -y epel-release wget unzip httpd git maven

yum install -y java-1.8.0-openjdk-devel

yum group install -y "Development Tools"

yum install -y pcre-devel zlib-devel openssl-devel libaio-devel

1. Install and enable ntp (network time synchronization)

yum install -y ntp

systemctl enable ntpd

systemctl start ntpd

1. Additional entropy for the server may be required. The web application may hang while starting if this is not installed – the code checks to ensure the JCE Unlimited Policy is in effect.

yum install -y haveged

systemctl enable haveged

systemctl start haveged

1. To make the deployment easier for users to access, register the server instance in the domain name system (DNS) (e.g., openke.domain.com). This also facilitates securing the service by enabling SSL (TLS) and acquiring a server SSL certificate. Consider attaching an elastic IP (AWS only) to the server so that the relevant DNS entries do not have to be continually changed when the server is stopped and started. This guide demonstrates using Let’s Encrypt to acquire a certificate. Let’s Encrypt does require the server to be accessible through the Internet for validation. If it is not directly accessible, DNS validation can be performed. <https://letsencrypt.org/how-it-works/>

## PostgreSQL Client Install

Install the client libraries for PostgreSQL. While this is not technically required for the install, it makes it easier to administer the database while on the OSKE server. These instructions also assume the client libraries are available during the installation of the Collector Daemon.

1. Install the official repo. Yum repositories are at <https://yum.postgresql.org/repopackages.php>

wget <https://download.postgresql.org/pub/repos/yum/10/redhat/rhel-7-x86_64/pgdg-centos10-10-2.noarch.rpm>

rpm -Uvh pgdg-centos10-10-2.noarch.rpm

1. Install PostgreSQL client software

yum install -y postgresql10

## Custom Users

The OSKE system utilizes a large number of users and groups (see Appendix D: System User and Groups for the complete listing and use). Many of these accounts are auto-created through the installation process.

While these instructions list specific IDs, these IDs may be changed as necessary. /etc/group defines the groups currently identified on the server, while users are contained in the /etc/passwd file.

1. Create group and user for the OSKE system:

groupadd -g 887 collector

useradd -g 887 -u 887 -d /opt/collector collector

1. Create the user for NGINX

useradd --no-create-home nginx

## Tomcat

OSKE’s web application runs within a Tomcat server. As it is deployed as a war file, the web application may be run on other servlet containers. As the Tomcat version within the standard Yum repositories was not current, Tomcat must be manually installed from a distribution from <http://tomcat.apache.org>. The project uses the 8.5.x baseline. The application should work fine under the 9.x baseline as well. Review which Tomcat version to install at <http://tomcat.apache.org/whichversion.html>

Due to file permission issues, the collector daemon and the web application as the same user. The issues arise from shared configuration files (which are locked down to user only access) as well as a shared directory used to upload and then process files.

1. Create the Tomcat directory.

mkdir /opt/tomcat

cd /opt/tomcat

Download the appropriate version of Tomcat. Due to apache mirrors, this link may be different. Downloads are available at <https://tomcat.apache.org/download-80.cgi>

wget http://mirror.cogentco.com/pub/apache/tomcat/tomcat-8/v8.5.40/bin/apache-tomcat-8.5.40.tar.gz

1. Extract the contents and remove the tar file

tar xvf apache-tomcat-8.5.40.tar.gz

rm apache-tomcat-8.5.40.tar.gz

1. Create a symbolic link that specifies the “current” installation

ln -s apache-tomcat-8.5.40 current

1. Create a systemd service file

cat > /etc/systemd/system/tomcat.service << EOF

[Unit]

Description=Apache Tomcat Servlet Container Engine

After=network.target

After=syslog.target

[Install]

WantedBy=multi-user.target

[Service]

Type=forking

User=collector

Group=collector

ExecStart= /opt/tomcat/current/bin/startup.sh

EOF

1. Create a setenv.sh file to set the logging and memory for Tomcat. The system requires a minimum of 512 mb for the JVM with a maximum of 8,192 mb (8gb) of memory.

cat >> /opt/tomcat/current/bin/setenv.sh << EOF

export CATALINA\_OPTS="-Xms512m -Xmx8192m -Dlogback.configurationFile=/opt/tomcat/current/conf/logback.xml"

EOF

1. Create the logging configuration file

cat > /opt/tomcat/current/conf/logback.xml << EOF

<configuration>

<appender name="STDOUT" class="ch.qos.logback.core.ConsoleAppender">

<encoder>

<pattern>%date{HH:mm:ss} %-5level [%thread] - [%logger{0}]- %msg%n</pattern>

</encoder>

</appender>

<logger name="com.github.tomakehurst" level="ERROR"/>

<logger name="org.eclipse.jetty" level="ERROR" />

<root level="INFO">

<appender-ref ref="STDOUT" />

<!--<appender-ref ref="FILE" />-->

</root>

</configuration>

EOF

1. Allow resource caching. (Used by Voyant). In /opt/tomcat/current/conf/context.xml add this line just before the </Context> tag:

<Resources cachingAllowed="true" cacheMaxSize="100000" />

1. Modify the ROOT application to redirect users to the collector web application:

cat > /opt/tomcat/current/webapps/ROOT/index.jsp << EOF

<% response.sendRedirect("/collector"); %

EOF

1. Set the ownership of Tomcat to the collector user

chown -R collector:collector /opt/tomcat

1. Enable and start the Tomcat service.

systemctl enable tomcat

systemctl start tomcat

From a best practices and security viewpoint, installed applications in the webapps directory should be removed. The examples application provides a sample application to view request headers that may prove useful in verifying the environment, but this application should not be kept long-term. Remove the pre-installed applications(docs, examples, host-manager, manager) by removing the corresponding directories in /opt/tomcat/current/webapps

## Voyant

Voyant is a web-based text analysis and visualization environment developed by Stéfan Sinclair and Geoffrey Rockwall. Visit <http://voyant-tools.org/> for documentation and a publicly available, online version to view.

While this process installs Voyant within tomcat by recursively copying the “\_app” directory into Tomcat’s webapp directory ***after*** the 1st step. Voyant can also be run as an independent application/service.

1. Get and install software

cd /opt/voyant

wget \

https://github.com/sgsinclair/VoyantServer/releases/download/2.4.0-M7/VoyantServer2\_4-M7.zip

unzip VoyantServer2\_4-M7.zip

rm VoyantServer2\_4-M7.zip

1. Install under Tomcat

systemctl stop tomcat

cp -r \_app /opt/tomcat/current/webapps/voyant

chown collector:collector /opt/tomcat/current/webapps/voyant

systemctl start tomcat

1. Validate the install

curl localhost:8080/voyant

## NGINX / HTTPS

While technically this step is optional, consider installing NGINX, configuring HTTPS, and turning-on the OAuth2 Google proxy for all but single-user development environments. While at one point, HTTPS and SSL certificates were both costly in terms of price and dollars, modern processors do not significantly slow down with HTTPS and Let’s Encrypt provides free SSL certificates. The privacy and security impacts of using HTTPS far outweigh any negatives. NGINX can also be configured to front Kibana and Cerebro.

NGINX needs to be installed from the source code as the version from standard yum repositories does not contain required modules.

1. Check whether or not nginx current exists. Stop, disable, and uninstall. OSKE requires a custom installation due to the integration with an oauth proxy.
2. Install the Certbot Let’s Encrypt Client along with dependencies to build NGINX

yum install -y certbot-nginx pcre-dev zlib-devel redhat-rpm-config \

openssl-devel gd-devel google-perftools-devel libxslt-devel \

perl-ExtUtils-Embed geoip-devel

1. Build and install NGINX. Use the latest versions. The configure options were determined by using the repository version with the “-V” flag.

mkdir /root/nginx

cd /root/nginx

wget [http://nginx.org/download/nginx-1.16.0.tar.gz](http://nginx.org/download/nginx-1.15.3.tar.gz)

tar xvf nginx-1.16.0.tar.gz

wget <https://github.com/openresty/headers-more-nginx-module/archive/v0.33.tar.gz>

tar xvf v0.33.tar.gz

wget <https://github.com/stnoonan/spnego-http-auth-nginx-module/archive/master.zip>

unzip master.zip

cd nginx-1.16.0

./configure --prefix=/usr/share/nginx --sbin-path=/usr/sbin/nginx \

--modules-path=/usr/lib64/nginx/modules --conf-path=/etc/nginx/nginx.conf \

--error-log-path=/var/log/nginx/error.log \

--http-log-path=/var/log/nginx/access.log \

--http-client-body-temp-path=/var/lib/nginx/tmp/client\_body \

--http-proxy-temp-path=/var/lib/nginx/tmp/proxy \

--http-fastcgi-temp-path=/var/lib/nginx/tmp/fastcgi \

--http-uwsgi-temp-path=/var/lib/nginx/tmp/uwsgi \

--http-scgi-temp-path=/var/lib/nginx/tmp/scgi --pid-path=/run/nginx.pid \

--lock-path=/run/lock/subsys/nginx --user=nginx --group=nginx \

--with-file-aio --with-http\_ssl\_module --with-http\_v2\_module \

--with-http\_realip\_module --with-http\_addition\_module \

--with-http\_xslt\_module=dynamic --with-http\_image\_filter\_module=dynamic \

--with-http\_geoip\_module=dynamic --with-http\_sub\_module --with-http\_dav\_module\

--with-http\_flv\_module --with-http\_mp4\_module --with-http\_gunzip\_module \

--with-http\_gzip\_static\_module --with-http\_random\_index\_module \

--with-http\_secure\_link\_module --with-http\_degradation\_module \

--with-http\_slice\_module --with-http\_stub\_status\_module \

--with-http\_perl\_module=dynamic --with-mail=dynamic --with-mail\_ssl\_module \

--with-pcre --with-pcre-jit --with-stream=dynamic --with-stream\_ssl\_module \

--with-google\_perftools\_module --with-debug \

--with-cc-opt='-O2 -g -pipe -Wall -Wp,-D\_FORTIFY\_SOURCE=2 -fexceptions -fstack-protector \

--param=ssp-buffer-size=4 -m64 -mtune=generic' \

--add-dynamic-module=/root/nginx/headers-more-nginx-module-0.33 \

--add-dynamic-module=/root/nginx/spnego-http-auth-nginx-module-master \

--with-http\_auth\_request\_module

make

make install

cp objs/ngx\_http\_headers\_more\_filter\_module.so /usr/lib64/nginx/modules/

chcon -t textrel\_shlib\_t '/usr/lib64/nginx/modules/ngx\_http\_headers\_more\_filter\_module.so'

chcon -t textrel\_shlib\_t '/usr/lib64/nginx/modules/ngx\_http\_auth\_spnego\_module.so'

1. Create the systemd service file

vi /etc/systemd/system/nginx.service

[Unit]

Description=The nginx HTTP and reverse proxy server

After=network.target remote-fs.target nss-lookup.target

[Service]

Type=forking

PIDFile=/run/nginx.pid

# Nginx will fail to start if /run/nginx.pid already exists but has the wrong

# SELinux context. This might happen when running `nginx -t` from the cmdline.

# https://bugzilla.redhat.com/show\_bug.cgi?id=1268621

ExecStartPre=/usr/bin/rm -f /run/nginx.pid

ExecStartPre=/usr/sbin/nginx -t

ExecStart=/usr/sbin/nginx

ExecReload=/bin/kill -s HUP $MAINPID

KillSignal=SIGQUIT

TimeoutStopSec=5

KillMode=process

PrivateTmp=true

[Install]

WantedBy=multi-user.target

1. Setup the /var/lib/nginx area

mkdir -p /var/lib/nginx/tmp

chown -R nginx:nginx /var/lib/nginx

1. Remove the entire “server” configuration from the /etc/nginx/nginx.conf file.
2. Enable the headers-more module in /etc/nginx/nginx.conf as the first line

load\_module /usr/lib64/nginx/modules/ngx\_http\_headers\_more\_filter\_module.so;

Add including the conf.d directory files as the last line in the http configuration

include /etc/nginx/conf.d/\*.conf;

1. Make the conf.d directory

mkdir /etc/nginx/conf.d

1. Validate the configuration. Fix any errors that exist.

nginx -t

1. Allow http to make connections (necessary for the kibana proxy)

setsebool httpd\_can\_network\_connect on -P

1. Enable and start nginx

systemctl enable nginx

systemctl start nginx

1. Create a configuration file for the server.

cat > /etc/nginx/conf.d/***FULL\_DOMAIN\_NAME***.conf << EOF

server {

listen 80;

listen [::]:80;

server\_name ***FULL\_DOMAIN\_NAME***;

}

EOF

1. If the server is not accessible from the internet, an SSL certificate will need to be installed manually. Some options:
   1. Manually install the certificate and use DNS validation to authorize the service. This process requires that the DNS records for the selected domain can be updated.

<https://serverfault.com/questions/750902/how-to-use-lets-encrypt-dns-challenge-validation>

<https://certbot.eff.org/docs/using.html#manual>

certbot -d openke.domain.com --manual --preferred-challenges dns certonly

* 1. Self-signed Certificate - <https://www.digitalocean.com/community/tutorials/how-to-create-a-self-signed-ssl-certificate-for-nginx-on-centos-7>

1. With internet access, use certbot to install the certificate

certbot --nginx -d ***FULL\_DOMAIN\_NAME***

Enter a contact email address. Review and agree to the terms of service. Choose whether or not to share the contact email address with the Electronic Frontier Foundation (EFF). Finally, chose to redirect HTTP traffic to HTTPS.

1. Make these modifications to server configuration file

upstream NGINX\_SERVICE\_NAME {

server 127.0.0.1:8080;

}

server {

server\_name ***FULL\_DOMAIN\_NAME***;

listen [::]:443 ssl ipv6only=on; # managed by Certbot

listen 443 ssl; # managed by Certbot

ssl\_certificate /etc/letsencrypt/live/***FULL\_DOMAIN\_NAME***/fullchain.pem;

ssl\_certificate\_key /etc/letsencrypt/live/***FULL\_DOMAIN\_NAME***/privkey.pem;

include /etc/letsencrypt/options-ssl-nginx.conf;

ssl\_dhparam /etc/letsencrypt/ssl-dhparams.pem;

#additional entries to add

client\_max\_body\_size 20M;

location / {

# force timeouts if one of backend is dead

proxy\_next\_upstream error timeout invalid\_header http\_500 http\_502 http\_503 http\_504;

proxy\_set\_header Accept-Encoding "";

proxy\_set\_header Host $host;

proxy\_set\_header X-Real-IP $remote\_addr;

proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;

proxy\_set\_header X-Forwarded-Proto $scheme;

add\_header Front-End-Https on;

# by default don’t perform redirections

proxy\_redirect off;

more\_clear\_input\_headers 'Variable-\*' 'Shib-\*' 'Remote-User' 'REMOTE\_USER' 'Auth-Type' 'AUTH\_TYPE';

more\_clear\_input\_headers 'displayName' 'mail' 'persistent-id';

proxy\_http\_version 1.1;

proxy\_set\_header Upgrade $http\_upgrade;

proxy\_set\_header Connection "upgrade";

proxy\_pass http://***NGINX\_SERVICE\_NAME***;

}

}

server {

if ($host = ***FULL\_DOMAIN\_NAME***) {

return 301 https://$host$request\_uri;

} # managed by Certbot

listen 80 ;

listen [::]:80 ;

server\_name ***FULL\_DOMAIN\_NAME***;

return 404; # managed by Certbot

}

1. To enable SSL access to Kibana through NGINX
   1. Add this location to enable kibana to be access via SSL:

location ~ ^/kibana/(.\*)$ {

rewrite /kibana/(.\*) /$1 break;

proxy\_pass http://PRIVATE\_IP\_ADDRESS\_ELASTICSEARCH:5601;

proxy\_http\_version 1.1;

proxy\_set\_header Upgrade $http\_upgrade;

proxy\_set\_header Connection 'upgrade';

proxy\_set\_header Host $host;

proxy\_cache\_bypass $http\_upgrade;

proxy\_buffer\_size 128k;

proxy\_buffers 4 256k;

proxy\_busy\_buffers\_size 256k;

}

* 1. If not yet performed, make the following modifications in /etc/kibana/kibana.yml

NOTE: This configuration should already exist in the file from the previous installation steps

echo 'server.basePath: "/kibana"' >> /etc/kibana/kibana.yml

echo 'server.rewriteBasePath: true' >> /etc/kibana/kibana.yml

* 1. Restart both kibana and nginx

systemctl restart kibana

systemctl restart nginx

1. To access Cerebro through NGINX, add this server location:

location ~ (^/cerebro/) {

proxy\_set\_header Host $host;

proxy\_set\_header X-Real-IP $remote\_addr;

proxy\_set\_header X-Forwarded-Proto https;

proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;

proxy\_pass http://***PRIVATE\_IP\_ADDRESS\_ELASTICSEARCH***:9201;

}

1. Now, configure NGINX to use an OAuth2 proxy to validate all requests and ensure a user is authenticated prior to any traffic being forward to the actual application servers. OSKE uses the oauth2\_proxy at <https://github.com/bitly/oauth2_proxy> for this.
   1. Install the ouath2\_proxy

mkdir /root/oauth\_proxy

wget [https://github.com/bitly/oauth2\_proxy/releases/download/v2.2/oauth2\_proxy-2.2.0.linux-amd64.go1.8.1.tar.gz \](https://github.com/bitly/oauth2_proxy/releases/download/v2.2/oauth2_proxy-2.2.0.linux-amd64.go1.8.1.tar.gz%20%5C)

-O /root/oauth\_proxy/oauth.tar.gz

tar xvf /root/oauth\_proxy/oauth.tar.gz -C /root/oauth\_proxy/

mv /root/oauth\_proxy/oauth2\_proxy-2.2.0.linux-amd64.go1.8.1/oauth2\_proxy /usr/local/bin

* 1. Create a systemd service: This –set-xauthrequest flag to run in pass headers back to NGINX

cat > /etc/systemd/system/oauth2proxy.service << EOF

[Unit]

Description=oauth2\_proxy daemon service

After=syslog.target network.target

[Service]

ExecStart=/usr/local/bin/oauth2\_proxy -config=/etc/oauth2\_proxy/config.cfg --set-xauthrequest

ExecReload=/bin/kill -HUP $MAINPID

KillMode=process

Restart=always

[Install]

WantedBy=multi-user.target

EOF

* 1. Register the project on Google and get the Client ID and Client Secret. See <https://github.com/bitly/oauth2_proxy/blob/master/README.md#google-auth-provider>

Visit <https://console.developers.google.com/cloud-resource-manager>

Click on Create Project

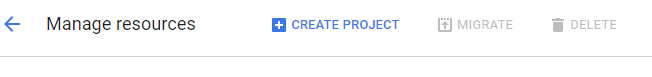


Figure 3 Google Console - Create Project

Select the project – may need to view under an organization

Select menu (3 horizontal lines), APIs & Services, and then credentials

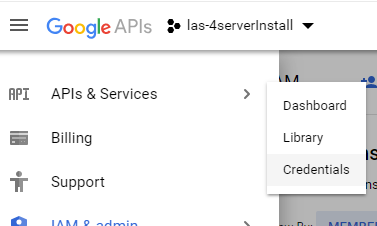


Figure 4 Google Console - Access Credentials

Then select OAUTH Consent Screen. Fill in just the product name and save

Then select credentials and create credentials:

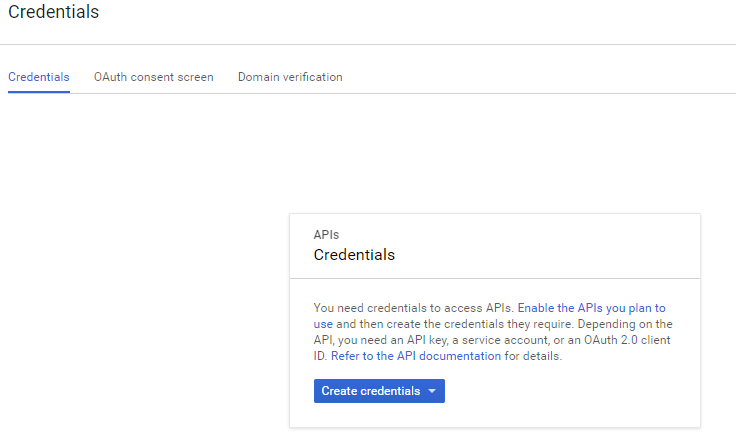


Figure 5 Google Console - Credentials

Create an OAUTH Client ID:

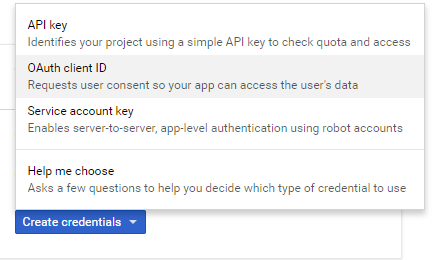


Figure 6 Google Console - Choose Credential Type

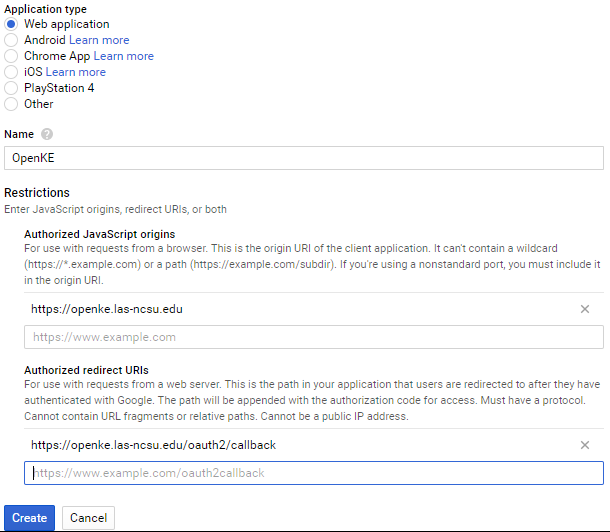


Figure 7 Google Console - Create Web Application ID

Click “Create”. Record the Client ID and Client Secret.

* 1. Generate a random value for the cookie secret

python -c 'import os,base64; print base64.b64encode(os.urandom(16))'

* 1. Create the configuration directory

mkdir /etc/oauth2\_proxy

* 1. Create a file to whitelist email addresses that should have access to the service:

touch /etc/oauth2\_proxy/authenticated\_emails.txt

* 1. Configure oauth2proxy

cat > /etc/oauth2\_proxy/config.cfg << EOF

## <addr>:<port> to listen on for HTTP/HTTPS clients

http\_address = "127.0.0.1:4180"

## the OAuth Redirect URL.

redirect\_url = "https://***FULL\_DOMAIN\_NAME***/oauth2/callback"

## the http url(s) of the upstream endpoint.

upstreams = [

"http://FULL\_DOMAIN\_NAME\_OPENKE/"

]

## The OAuth Client ID, Secret

client\_id = "***DEFINED GOOGLE CLIENT ID***"

client\_secret = "***DEFINED GOOGLE PROJECT SECRET***"

## If appropriate, set all email addresses from a given domain to be

## whitelisted. (e.g., replace DOMAIN\_NAME with "ncsu-las.org")

email\_domains=["DOMAIN\_NAME"]

## Authenticated Email Addresses File (one email per line in the file)

authenticated\_emails\_file = "/etc/oauth2\_proxy/authenticated\_emails.txt"

## Cookie Settings. Adjust the expiration time appropriately.

cookie\_name = "\_oauth2\_proxy"

cookie\_secret = "***SECRET\_DEFINED\_IN\_STEP\_D***"

cookie\_expire = "12h"

cookie\_secure = true

cookie\_httponly = true

EOF

* 1. Enable and start the service

systemctl enable oauth2proxy

systemctl start oauth2proxy

* 1. Verify the service is running. If not, look at the /var/log/messages for errors

systemctl status oauth2proxy

tail -10 /var/log/messages

* 1. Configuration for nginx server (editing the appropriate file in /etc/nginx/conf.d)

location /oauth2/ {

proxy\_pass http://127.0.0.1:4180;

proxy\_set\_header Host $host;

proxy\_set\_header X-Real-IP $remote\_addr;

proxy\_set\_header X-Scheme $scheme;

proxy\_set\_header X-Auth-Request-Redirect $request\_uri;

}

location = /oauth2/auth {

proxy\_pass http://127.0.0.1:4180;

proxy\_set\_header Host $host;

proxy\_set\_header X-Real-IP $remote\_addr;

proxy\_set\_header X-Scheme $scheme;

# nginx auth\_request includes headers but not body

proxy\_set\_header Content-Length "";

proxy\_pass\_request\_body off;

}

location / {

auth\_request /oauth2/auth;

error\_page 401 = /oauth2/sign\_in;

# force timeouts if one of backend is dead

proxy\_next\_upstream error timeout invalid\_header http\_500 http\_502 http\_503 http\_504;

more\_clear\_input\_headers 'Variable-\*' 'Shib-\*' 'Remote-User' 'REMOTE\_USER';

more\_clear\_input\_headers 'Auth-Type' 'AUTH\_TYPE';

more\_clear\_input\_headers 'displayName' 'mail' 'persistent-id';

proxy\_set\_header Accept-Encoding "";

proxy\_set\_header Host $host;

proxy\_set\_header X-Real-IP $remote\_addr;

proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;

proxy\_set\_header X-Forwarded-Proto $scheme;

add\_header Front-End-Https on;

# by default don’t perform the redirection

# proxy\_redirect off;

# pass information via X-User and X-Email headers to backend,

# requires running with --set-xauthrequest flag

#auth\_request\_set $user $upstream\_http\_x\_auth\_request\_user;

auth\_request\_set $email $upstream\_http\_x\_auth\_request\_email;

#proxy\_set\_header X-User $user;

proxy\_set\_header REMOTE\_USER $email;

# if --cookie-refresh is enabled add these lines for it to work with auth\_request

auth\_request\_set $auth\_cookie $upstream\_http\_set\_cookie;

add\_header Set-Cookie $auth\_cookie;

#auth\_request\_set $user $upstream\_http\_x\_session\_user;

#proxy\_set\_header X-Forwarded-User $user;

proxy\_http\_version 1.1;

proxy\_set\_header Upgrade $http\_upgrade;

proxy\_set\_header Connection "upgrade";

proxy\_pass http://***NGINX\_SERVICE\_NAME***;

}

Note: If Kibana and/or Cerebro is configured through NGINX, add first two lines from the “/” location into their respective location configuration.

location ~ (^/cerebro/) {

auth\_request /oauth2/auth;

error\_page 401 = /oauth2/sign\_in;

proxy\_set\_header Host $host;

proxy\_set\_header X-Real-IP $remote\_addr;

proxy\_set\_header X-Forwarded-Proto https;

proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;

# The rewrite is disabled, to actually have the login screen displayed.

#rewrite ^/cerebro/(.\*) /$1 break;

proxy\_pass http://***FULL\_DOMAIN\_NAME\_ELASTIC***:9201;

}

location ~ ^/kibana/(.\*)$ {

auth\_request /oauth2/auth;

error\_page 401 = /oauth2/sign\_in;

rewrite /kibana/(.\*) /$1 break;

proxy\_pass http://172.31.71.131:5601;

proxy\_http\_version 1.1;

proxy\_set\_header Upgrade $http\_upgrade;

proxy\_set\_header Connection 'upgrade';

proxy\_set\_header Host $host;

proxy\_cache\_bypass $http\_upgrade;

proxy\_buffer\_size 128k;

proxy\_buffers 4 256k;

proxy\_busy\_buffers\_size 256k;

}

## GitHub

To install the daemon, and web application, the GitHub repository must be accessible.

Once the repository can be accessed, download the OpenSourceKnowledgeEnrichmentrepository via clone or download operation:

mkdir /root/git

cd /root/git

git clone [git@github.com](mailto:git@github.com):LAS-NCSU/OpenSourceKnowledgeEnrichment.git

## Software Builds

Next, build the various components using maven. Dependencies will be built first.

1. Build crawler4j

cd /root/git/OpenSourceKnowledgeEnrichment/externalProjects/crawler4j

mvn install

1. Build boilerpipe

cd /root/git/OpenSourceKnowledgeEnrichment/externalProjects/boilerpipe/

mvn install

1. Build LAS-Common

cd /root/git/OpenSourceKnowledgeEnrichment/LAS-Common

mvn install

1. Build minIE

cd /root/git/OpenSourceKnowledgeEnrichment/externalProjects/minIE

mvn install

1. Build LAS-Common-NLP

cd /root/git/OpenSourceKnowledgeEnrichment/LAS-Common-NLP

mvn install

1. Build the Collector Daemon

cd /root/git/OpenSourceKnowledgeEnrichment/Collector

mvn dependency:copy-dependencies package install

1. Build the Collector Web Application

cd /root/git/OpenSourceKnowledgeEnrichment/CollectorWebApp

mvn package

## Collector Daemon

The daemon continuously runs to process defined jobs to collect information from the internet. The following actions are scheduled (by default every minute):

* 1. Check for any domain configuration changes or new domains
  2. Display current job status in the logs
  3. Stop any jobs that have inactive for over 5 minutes or that a user has marked to be stopped
  4. Start any jobs that are now ready to run
  5. Check for any “search alerts” to provide screen-based notifications to the user
  6. Perform other clean-up activities such as old, exported files

The daemon utilizes two locations on a server:

1. Data / log area (***OPENKE\_ DIRECTORY \_DATA\_LOG***)
2. Installed application and configuration files (***OPENKE\_DIRECTORY\_SOFTWARE****)*

Configuring OSKE is described in detail in the Administrator’s Guide.

To install the daemon, perform the following steps:

1. Create the data and log directory structure:

mkdir -p OPENKE\_DIRECTORY\_DATA\_LOG

mkdir OPENKE\_DIRECTORY\_DATA\_LOG/download

mkdir OPENKE\_DIRECTORY\_DATA\_LOG/externalExport

mkdir OPENKE\_DIRECTORY\_DATA\_LOG/logs

mkdir OPENKE\_DIRECTORY\_DATA\_LOG/upload\_directory

chown -R collector:collector ***OPENKE\_DIRECTORY\_DATA\_LOG***

1. Create the software directory structure: (Note: the sofware directory most likely already exists because of the collector user account being previously created.)

mkdir -p OPENKE\_DIRECTORY\_SOFTWARE/collector

mkdir OPENKE\_DIRECTORY\_SOFTWARE/collector/bin

mkdir OPENKE\_DIRECTORY\_SOFTWARE/collector/config

mkdir OPENKE\_DIRECTORY\_SOFTWARE/collector/lib

mkdir OPENKE\_DIRECTORY\_SOFTWARE/collector/bin

ln -s OPENKE\_ DIRECTORY \_DATA\_LOG/logs OPENKE\_DIRECTORY\_SOFTWARE/logs

chmod 700 OPENKE\_DIRECTORY\_SOFTWARE

1. Copy the configuration file from the OSKE git repository clone

cp /root/git/OpenSourceKnowledgeEnrichment/Collector/logging.properties \

***OPENKE\_DIRECTORY\_SOFTWARE***/collector/config

cp /root/git/OpenSourceKnowledgeEnrichment/Collector/logging.xml \

***OPENKE\_DIRECTORY\_SOFTWARE***/collector/config

cp /root/git/OpenSourceKnowledgeEnrichment/Collector/system\_properties.json.template \

***OPENKE\_DIRECTORY\_SOFTWARE***/collector/config/system\_properties.json

1. Create start script for collector

vi OPENKE\_DIRECTORY\_SOFTWARE/collector/bin/start.sh

#!/bin/bash

# Command starts the collector process

#

BASE\_DIR=OPENKE\_DIRECTORY\_SOFTWARE/collector

CONFIG\_DIR=$BASE\_DIR/config

COLLECTOR\_CLASSPATH=$BASE\_DIR'/lib/\*:'$BASE\_DIR/collector.jar

cd $CONFIG\_DIR

nohup java -Djava.util.logging.config.file=$CONFIG\_DIR/logging.properties -Dlogback.configurationFile=$CONFIG\_DIR/logging.xml -Xmx8g -classpath $COLLECTOR\_CLASSPATH edu.ncsu.las.collector.JobCollector 2>&1 | rotatelogs ***OPENKE\_DIRECTORY\_DATA***/logs/collect.log 86400 &

pid=$(jobs -p)

echo $pid > $BASE\_DIR/collector.pid

1. Change the ownership and permissions

chown collector:collector OPENKE\_DIRECTORY\_SOFTWARE/collector/bin/start.sh

chmod 744 OPENKE\_DIRECTORY\_SOFTWARE/collector/bin/start.sh

chown -R collector:collector ***OPENKE\_DIRECTORY\_SOFTWARE***/collector

chmod 600 ***OPENKE\_DIRECTORY\_SOFTWARE***/collector/config/\*

1. Edit the ***OPENKE\_DIRECTORY\_SOFTWARE***/collector/config/system.properties.json as appropriate based upon the configuration.
2. Put the initial entries into the user and domain tables. The two commands are in /root/git/OpenSourceKnowledgeEnrichment/Collector/sql/local\_setup.sql.template. Make sure all of the parameters are updated/defined appropriately.

cd OPENKE\_DIRECTORY\_SOFTWARE/collector/config

cp /root/git/OpenSourceKnowledgeEnrichment/Collector/sql/local\_setup.sql.template ***OPENKE\_DIRECTORY\_SOFTWARE***/collector/config/local\_setup.sql

vi local\_setup.sql

Edit all of the parameters that need to be changed and then execute the SQL commands in PostgreSQL.

psql -U POSTGRESQL\_OPENKE\_USER -h FULL\_POSTGRES\_DOMAIN\_NAME \

-d POSTGRESQL\_OPENKE\_DB -W -f local\_setup.sql

1. Copy application jar file and associated libraries from the build area: (note the name change on the collector jar)

cp /root/git/OpenSourceKnowledgeEnrichment/Collector/target/dependency/\* \

***OPENKE\_DIRECTORY\_SOFTWARE***/collector/lib/

cp /root/git/OpenSourceKnowledgeEnrichment/Collector/target/Collector-0.0.2.jar \

***OPENKE\_DIRECTORY\_SOFTWARE***/collector.jar

Future system updates require the rebuild of the software builds, copying of the lib directory, and then copying of the collector.jar file.

1. Set the ownership to collector

chown -R collector:collector /opt/collector/

1. Create a systemd service. OSKE can detect if it is in a catastrophic backtracking situation (<https://www.regular-expressions.info/catastrophic.html>) while using regular expressions to extract concepts from text. In this situation, it will kill itself if no jobs are actively running. We’ll use the “Restart=always” flag within system to start the service again. Note: if the daemon ran within the context of a Docker container, a restart policy would need to be applied to that container.

cat > /etc/systemd/system/collector.service <<EOF

[Unit]

Description=Collector DaemonCollector DaemonCollector DaemonCollector Daemon

After=network.target

After=syslog.target

[Install]

WantedBy=multi-user.target

[Service]

Type=forking

User=collector

Group=collector

Restart=always

RestartSec=5

ExecStart=***OPENKE\_DIRECTORY\_SOFTWARE***/collector/bin/start.sh

EOF

1. Start the daemon and view the log files to look for any issues. (\* assumes that no other log files exist. Otherwise tail the latest file.)

systemctl enable collector

systemctl start collector

Manual start instructions outside of a service:

***OPENKE\_DIRECTORY\_SOFTWARE***/collector/bin/start.sh

View log file:

tail -f ***OPENKE\_DIRECTORY\_DATA\_LOG***/logs/\*

## Collector Web Application

Tomcat hosts the web application.

Note: at one point, Tomcat ran as a separate user, but dealing with permission conflicts have just made it easier to run the tomcat application as collector.

1. Stop tomcat

systemctl stop tomcat

1. Create the application configuration file

vi /opt/tomcat/current/conf/Catalina/localhost/collector.xml

<?xml version='1.0' encoding='utf-8'?>

<!-- The contents of this file will be loaded for each web application -->

<Context>

<!-- Default set of monitored resources. If one of these changes, the -->

<!-- web application will be reloaded. -->

<WatchedResource>WEB-INF/web.xml</WatchedResource>

<WatchedResource>${catalina.base}/conf/web.xml</WatchedResource>

<!-- Uncomment this to disable session persistence across Tomcat restarts -->

<!--

<Manager pathname="" />

-->

<!-- Uncomment this to enable Comet connection tacking (provides events

on session expiration as well as webapp lifecycle) -->

<!--

<Valve className="org.apache.catalina.valves.CometConnectionManagerValve" />

-->

<Environment name="collectorConfigurationDirectory"

value="***OPENKE\_DIRECTORY\_SOFTWARE***/collector/config" type="java.lang.String" override="false"/>

</Context>

1. Copy the web application from the build directory

cp /root/git/OpenSourceKnowledgeEnrichment/CollectorWebApp/target/CollectorWebApp-0.1.0-SNAPSHOT.war \

/opt/tomcat/current/webapps/collector.war

1. Correct ownership

chown -R collector:collector /opt/tomcat

1. Start tomcat

systemctl start tomcat

# Appendix A: AWS Notes

AWS – need to check hostname and /etc/hosts when starting from a created image.

Creating EC2 instances:

<http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/get-set-up-for-amazon-ec2.html>

<http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/EC2_GetStarted.html>

OSKE utilizes the standard CentOS image to install.

aws --region us-east-1 ec2 describe-images --owners aws-marketplace --filters Name=product-code,Value

s=aw0evgkw8e5c1q413zgy5pjce

# Appendix B: Parameter Worksheet

This table contains the parameters used within this install guide. This table should be completed for the installation and then kept for future reference.

|  |  |  |
| --- | --- | --- |
| Parameter | Description | Sample |
| COLLECTOR\_ID | Used to identify instances of the collection daemon. Defined in system\_properties.json. | collect1 |
| COLLECTOR\_SECRET\_PHRASE | This phrase is combined with another string phrase defined internally within the system and then hashed using SHA256 to create an AES key to encrypt and decrypt passwords defined in system and job configurations. |  |
| EMAIL\_PASSWORD | Password used to authenticate an account to an email server to send emails. | password |
| EMAIL\_PORT | What port does the email server listen on? |  |
| EMAIL\_SERVER | What is the machine name for the email server? |  |
| EMAIL\_USER | What is the account name that can be used to send emails |  |
| FULL\_DOMAIN\_NAME | Fully-qualified domain name (FQDN) |  |
| FULL\_DOMAIN\_NAME\_ELASTIC | Server on which Elasticsearch runs. |  |
| FULL\_DOMAIN\_NAME\_MICROSERVICES | FQDN where the microservices execute. |  |
| FULL\_DOMAIN\_NAME\_NEO4J | Server on which Neo4J runs |  |
| FULL\_DOMAIN\_NAME\_OPENKE | Server on which the primary collector daemon and web application reside. |  |
| FULL\_DOMAIN\_NAME\_POSTGRES | Database server for postgres |  |
| FULL\_DOMAIN\_NAME\_VOYANT | Server used for Voyant. Typically installed on the same machine as OSKE. |  |
| KIBANA\_SERVER\_NAME |  | OpenKE-dev |
| LDAP\_BASEDN |  | ou=people,dc=ncsu,dc=edu |
| LDAP\_URL |  | ldap://ldap.ncsu.edu:389 |
| MICROSERVICE\_FULL |  |  |
| NGINX\_SERVICE\_NAME | Downstream service name that forwards HTTP requests once users have been authenticated. |  |
| OPENKE\_DIRECTORY\_DATA\_LOG | Location where the OSKE data and logs are stored | /hadoop/collector |
| OPENKE\_DIRECTORY\_SOFTWARE | Location where the OSKE software is stored | /opt/collector |
| POSTGRESQL\_DATA\_DIRECTORY | Data location for the database. | /var/lib/pgsql/10/data |
| POSTGRESQL\_OPENKE\_DATABASE |  | openke\_dev |
| POSTGRESQL\_OPENKE\_USER |  | openke |
| POSTGRESQL\_OPENKE\_PASSWORD |  | openKEdev! |
| POSTGRESQL\_SERVER | Server for the PostgreSQL database server. This is the same value as FULL\_DOMAIN\_NAME\_POSTGRESQL |  |
| POSTGRESQL\_PORT |  | 5432 |

Table 7 OSKE Install Parameters

# Appendix C: System Port Usage

The below table lists the ports used by OSKE and for what purpose.

|  |  |  |
| --- | --- | --- |
| Port | Application | Usage |
| 80 | NGINX | Redirects traffic to port 443 for SSL/TLS |
| 443 | NGINX | Primary connection point to access the collector web application externally |
| 2222 | DBPedia Spotlight Service | Allows text to be annotate with links to resources within DBPedia. <https://www.dbpedia-spotlight.org/> |
| 4180 | oauth2\_proxy | The OAuth2 proxy service listens on this port to authenticate users from NGINX. |
| 4835 | JWT Authorization Proxy | The JWT proxy service listens on this port to validate JWT tokens |
| 5432 | PostgreSQL | PostgreSQL database listener |
| 5601 | Kibana | Used to access Kibana. Traffic can be routed through NGINX for external access. |
| 5679 | Microservice: WordNet | Find synonyms for words for use during domain discovery sessions |
| 8000 | Collector: TextRank Service | Also does back of the book … |
| 8888 | Voyant |  |
| 9001 | Microservice: EXIF Extraction | Extracts photography-based metadata from an image |
| 9002 | Microservice: geocoding | Finds coordinates for a given location |
| 9003 | Microservice: time | Temporal analysis of document (Heideltime) |
| 9004 | Microservice: geotagging | Fixed location mentions in text. Leverages CLAVIN |
| 9005 | Microservice: topicmodel | Runs LDA topic modelling on a series of documents |
| 9006 | Microservice: nlp | Runs the Stanford CoreNLP on a document |
| 9008 | Microservice: whois | Produces whois data for a domain |
| 9009 | Microservice: microformat | Extracts microformat data (<http://microformats.org/>) from HTML |
| 9010 | Microservice: structural | Performs structural extraction of data from web pages based upon CSS selectors |
| 9011 | Microservice: scraper | Simple REST API to scrape a single web page using dynamic crawling, implemented using Puppeteer (<https://pptr.dev/>) |
| 9012 | Microservice: spaCy | Python-based NLP library for named entity recognition. |
| 9080 | Tomcat / Collector Web Application | Provides the collector web application. Voyant can also be installed here. |
| 9200 | Elasticsearch |  |
| 9201 | Cerebro | Management web user interface for Elasticsearch |
| 9300 | Elasticsearch native port |  |

Table 8 System Port Usage

# Appendix D: System User and Groups

|  |  |  |  |
| --- | --- | --- | --- |
| User | Group Membership | Created by | Usage: |
| apache |  | yum | Not directly used. An artifact of installing httpd to get the rotatelogs command |
| cerebro |  | manual | Runs the web user interfrace that provide a UI to elasticsearch |
| collector |  | manual | Runs the collector daemon and web application (under Tomcat) |
| elasticsearch |  | yum |  |
| kibana |  | yum |  |
| nginx |  | manual | Runs nginx that functions a reverse proxy service |
| ntp |  | yum |  |
| postgres |  | yum |  |

Table 9 OSKE Users

|  |  |  |  |
| --- | --- | --- | --- |
| Group | Created by |  |  |
| apache | yum |  |  |
| cerebro | manual |  |  |
| collector | manual |  |  |
| elasticsearch | yum |  |  |
| kibana | yum |  |  |
| ntp | yum |  |  |
| postgres | yum |  |  |

Table 10 OSKE Groups

# Appendix E: Change History

|  |  |
| --- | --- |
| Date |  |
| 20180314 | Added instructions to run the Collector as a systemd service  Added Change History Table |
| 20180325 | Microservices changes to support different processing path when generating “back of the book” indexes within domain discovery sessions. Created docker network and [Redis](https://redis.io/) image |
| 20180616 | Added instructions for installing the DBPedia Spotlight Docker image |
| 20180831 | Added additional parameters for NGINX / Kibana configuration –  https://discuss.elastic.co/t/courier-fetch-bad-gateway-6-4-0/146068/5 |
| 20180904 | Created version specific to installing on 4 machines in AWS w/out installing Hortonwork components |
| 20180908 | Changed sizing of OSKE to M5 2x large. 16gb should be sufficient  Also, the R5 2x large are acceptable alternatives for the Microservices and ElasticSearch instances. |
| 20190307 | Added wordnet service |
| 20190502 | Replaced individual docker installs with docker-compose process |
| 20190508 | Added scraper microservice. |
|  |  |
|  |  |
|  |  |
|  |  |

Table 11 Change History

1. The system is referred to both by OSKE and OpenKE throughout this documentation and the project. [↑](#footnote-ref-1)
2. This constraint is necessary to process content uploaded from a user through the web application by the daemon. [↑](#footnote-ref-2)