
docker-zipkin

Archived

This repository is archived as it has been folded into <https://github.com/openzipkin/zipkin/tree/master/docker>

Overview

This repository contains the Docker build definition and release process for Zipkin Server. It also contains test images for transport and storage backends such as Kafka or Cassandra.

Automatically built images are available on Quay.io under the OpenZipkin organization, and are mirrored to Docker Hub.

Regarding production usage

The only images OpenZipkin provides for production use are: * `openzipkin/zipkin`: The core server image that hosts the Zipkin UI, Api and Collector features. * `openzipkin/zipkin-slim`: The stripped server image that hosts the Zipkin UI and Api features, but only supports in-memory or Elasticsearch storage with HTTP or gRPC span collectors. * `openzipkin/zipkin-dependencies`: pre-aggregates data such that `http://your_host:9411/dependency` shows links between services.

If you are using these images and run into problems, please raise an issue or join [gitter](#).

The other images here, and `docker-compose`, are for development and exploration purposes. For example, they aim to help you integrate an entire zipkin system for testing purposes, without having to understand how everything works, and without having to download gigabytes of files.

For example, `openzipkin/zipkin-cassandra` was not designed for real usage. You'll notice it has no configuration available to run more than one node sensibly, neither does it handle file systems as one would "in real life". We expect production users to use canonical images for storage or transports like Kafka, and only those testing or learning zipkin to use the ones we have here.

Running

Zipkin has no dependencies, for example you can run an in-memory zipkin server like so: `docker run -d -p 9411:9411 openzipkin/zipkin-slim`

See the ui at (docker ip):9411

In the ui - click zipkin-server, then click "Find Traces".

Configuration

Configuration is via environment variables, defined by zipkin-server. Notably, you'll want to look at the `STORAGE_TYPE` environment variables, which include "cassandra", "mysql" and "elasticsearch".

Note: the `openzipkin/zipkin-slim` image only supports "elasticsearch" storage. To use other storage types, you must use the main image `openzipkin/zipkin`.

When in docker, the following environment variables also apply

- `JAVA_OPTS`: Use to set java arguments, such as heap size or trust store location.
- `STORAGE_PORT_9042_TCP_ADDR` – A Cassandra node listening on port 9042. This environment variable is typically set by linking a container running `zipkin-cassandra` as "storage" when you start the container.
- `STORAGE_PORT_3306_TCP_ADDR` – A MySQL node listening on port 3306. This environment variable is typically set by linking a container running `zipkin-mysql` as "storage" when you start the container.
- `STORAGE_PORT_9200_TCP_ADDR` – An Elasticsearch node listening on port 9200. This environment variable is typically set by linking a container running `zipkin-elasticsearch` as "storage" when you start the container. This is ignored when `ES_HOSTS` or `ES_AWS_DOMAIN` are set.
- `KAFKA_PORT_2181_TCP_ADDR` – A zookeeper node listening on port 2181. This environment variable is typically set by linking a container running `zipkin-kafka` as "kafka" when you start the container.

For example, to add debug logging, set `JAVA_OPTS` as shown in our docker-compose file:

```
1 - JAVA_OPTS=-Dlogging.level.zipkin=DEBUG -Dlogging.level.zipkin2=DEBUG
```

Runtime user

The `openzipkin/zipkin` and `openzipkin/zipkin-slim` images run under a nologin user named 'zipkin' with a home directory of '/zipkin'. As this is a distroless image, you won't find many utilities installed, but you can browse contents with a shell like below:

```
1 $ docker run -it --rm --entrypoint /busybox/sh openzipkin/zipkin
2 /zipkin $ ls
3 BOOT-INF META-INF org run.sh
```

docker-compose

This project is configured to run docker containers using docker-compose. Note that the default configuration requires docker-compose 1.6.0+ and docker-engine 1.10.0+.

To start the default docker-compose configuration, run:

```
1 $ docker-compose up
```

View the web UI at \$(docker ip):9411.

To see specific traces in the UI, select “zipkin-server” in the dropdown and then click the “Find Traces” button.

Slim

To start a smaller and faster distribution of zipkin, run:

```
1 $ docker-compose -f docker-compose-slim.yml up
```

This starts in-memory storage. The only other supported option for slim is Elasticsearch:

```
1 $ docker-compose -f docker-compose-slim.yml -f docker-compose-elasticsearch.yml up
```

MySQL

The default docker-compose configuration defined in `docker-compose.yml` is backed by MySQL. This configuration starts `zipkin`, `zipkin-mysql` and `zipkin-dependencies` (cron job) in their own containers.

Cassandra

The docker-compose configuration can be extended to use Cassandra instead of MySQL, using the `docker-compose-cassandra.yml` file. That file employs docker-compose overrides to swap out one storage container for another.

To start the Cassandra-backed configuration, run:

```
1 $ docker-compose -f docker-compose.yml -f docker-compose-cassandra.yml up
```

Elasticsearch

The docker-compose configuration can be extended to use Elasticsearch instead of MySQL, using the `docker-compose-elasticsearch.yml` file. That file employs docker-compose overrides to swap out one storage container for another.

To start the Elasticsearch-backed configuration, run:

```
1 $ docker-compose -f docker-compose.yml -f docker-compose-elasticsearch.yml up
```

Elasticsearch 5+ and Host setup The `zipkin-elasticsearch5` and `zipkin-elasticsearch6` images are more strict about virtual memory. You will need to adjust accordingly (especially if you notice elasticsearch crash!)

```
1 # If docker is running on your host machine, adjust the kernel setting directly
2 $ sudo sysctl -w vm.max_map_count=262144
3
4 # If using docker-machine/Docker Toolbox/Boot2Docker, remotely adjust the same
5 $ docker-machine ssh default "sudo sysctl -w vm.max_map_count=262144"
```

Kafka

The docker-compose configuration can be extended to host a test Kafka broker and activate the Kafka collector using the `docker-compose-kafka.yml` file. That file employs docker-compose overrides to add a Kafka+ZooKeeper container and relevant settings.

To start the MySQL+Kafka configuration, run:

```
1 $ docker-compose -f docker-compose.yml -f docker-compose-kafka.yml up
```

Then configure the Kafka sender using a `bootstrapServers` value of `host.docker.internal:9092` if your application is inside the same docker network or `localhost:19092` if not, but running on the same host.

In other words, if you are running a sample application on your laptop, you would use `localhost:19092` bootstrap server to send spans to the Kafka broker running in Docker.

UI

The docker-compose configuration can be extended to host the UI on port 80 using the `docker-compose-ui.yml` file. That file employs docker-compose overrides to add an NGINX container and relevant settings.

To start the NGINX configuration, run:

```
1 $ docker-compose -f docker-compose.yml -f docker-compose-ui.yml up
```

This container doubles as a skeleton for creating proxy configuration around Zipkin like authentication, dealing with CORS with zipkin-js apps, or terminating SSL.

Prometheus

Zipkin comes with a built-in Prometheus metric exporter. The main `docker-compose.yml` file starts Prometheus configured to scrape Zipkin, exposes it on port 9090. You can open `$DOCKER_HOST_IP:9090` and start exploring the metrics (which are available on the `/prometheus` endpoint of Zipkin).

`docker-compose.yml` also starts a Grafana container with authentication disabled, exposing it on port 3000. On startup it's configured with the Prometheus instance started by `docker-compose` as a data source, and imports the dashboard published at <https://grafana.com/dashboards/1598>. This means that, after running `docker-compose up`, you can open `$DOCKER_IP:3000/dashboard/db/zipkin-prometheus` and play around with the dashboard.

If you want to run the zipkin-ui standalone against a remote zipkin server, you need to set `ZIPKIN_BASE_URL` accordingly:

```
1 $ docker run -d -p 80:80 \  
2   -e ZIPKIN_BASE_URL=http://myfavoritezipkin:9411 \  
3   openzipkin/zipkin-ui
```

Legacy

Docker machine and Kafka If you are using Docker machine, adjust `KAFKA_ADVERTISED_HOST_NAME` in `docker-compose-kafka.yml` and the `bootstrapServers` configuration of the kafka sender to match your Docker host IP (ex. 192.168.99.100:19092).

Notes

If using a provided MySQL server or image, ensure schema and other parameters match the docs.