The monitoring is working, now we want to focus to finish up the logging plus the tracing.
All relevant resources are located in https://gitlab.fhnw.ch/cloud/devops/templates/monitoring-logging-tracing and updated over there.

Task 1: Make Logging better

Loki is now collecting logs form all pods. Today, we want to adapt the logging by receiving json logs to have fields for indexing in place. This includes the adaption of the applications as well as the adaption of promtail, delivering the logs and grafana, showing the logs.

1. Go to eliza und roberta. Adapt both application to generate log as json. Be sure that the labels are the same for both applications. Refer to the movie in the second part of the lecture how to adapt.

2. Release and deploy your app and verify that only valid json is generated. Test locally beforehand.

3. In the monitoring-logging-tracing-repository, there is new commit containing the ability to parse json. Apply this commit to the promtail component of the loki stack.

4. Adapt the loki dashboard to search for e.g. loglevels (the search field is already present where you only need to adapt the query of the log overview at the botton. References how to do that are:
   https://grafana.com/blog/2020/04/08/loki-quick-tip-how-to-create-a-grafana-dashboard-for-searching-logs-using-loki-and-prometheus/
   https://grafana.com/docs/loki/latest/logql/log_queries/

5. If the dashboard works with filtering, store the json in a configmap and redeploy the dashboard.

Task 2: Tracing

For Tracing we rely on Jaeger combined with Opentelemetry-SDK-Bindings within Quarkus and Flask.

1. Install Jaeger related to the repository above in the tracing-folder. Read the Readme for detailed information.

2. Adapt Eliza so that it automatically publishes traces go jaeger:

   1. Read through https://quarkus.io/guides/opentelemetry.

3. Check the eliza-template repository under https://gitlab.fhnw.ch/cloud/devops/templates/chatbots/eliza

   1. Adapt your instance similar to the adaptions in the repository under the tag .0.2-monitoringtracinglogging. Read through the Readme.md and adapt your chatbots accordantly.

   2. Of course, within kubernetes, localhost won't work. You need to

    1. Inject a new configmap similar to connecting-worlds for eliza in the helm. Do not forget to reference it from the deployment-element of eliza.

    2. Provision that configmap as properties. Feel free to do it the easy way and only inject the necessary value from the values.yml directly to the value within the properties to be wrapped. If you deployed jaeger in the tracing namespace, this should only be the reference to the jaeger service:
       ```
       quarkus.opentelemetry.tracer.exporter.otlp.endpoint=http://jaeger-collector.tracing.svc.cluster.local:4317
       ```

4. Now, we have to focus on roberta to do the same thing. Check https://opentelemetry.io/docs/languages/python/automatic/example/ and enable roberta for generating traces as well. You can choose how you want to instrument your python app (automatic, programmatic or manual).

Tips for development:

- Build the application locally first including
  - native build of eliza and roberta and connecting worlds
  - jaeger based on the docker-compose in the template-repository
- you need to adapt the helm chart. Necessary adaptions are
  - a new configmap for injecting the application.properties to eliza. The adaption should contain the link to the jaeger-collector. This also needs an adaption to the deployment for eliza. Refer to the connecting-worlds-example to see how this can be done. The main difference is, that it might be easier to only inject parts of the application.properties (namely the value containing the url) instead of the entire file.
  - The URL for Roberta to the collector is the same and can be injected via sidecar or ENV-Variable to be consumed within the python sourcecode.