Imubit home exercise - Yagel Ardan

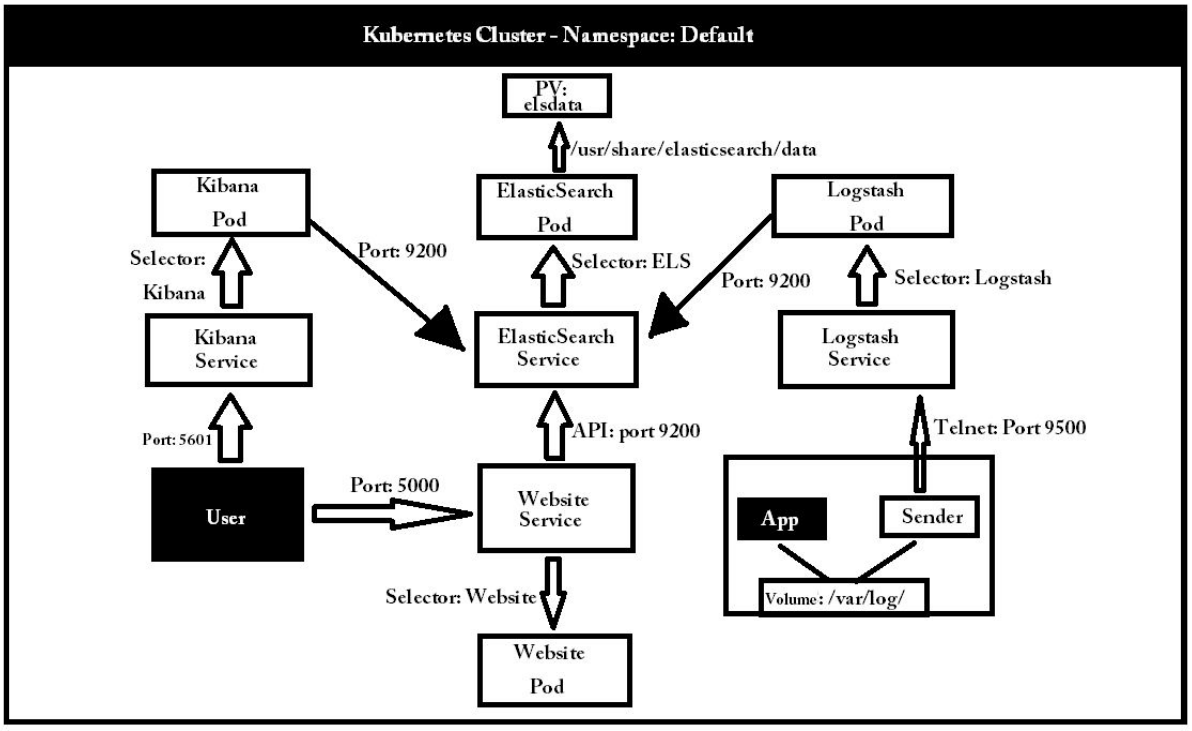
Intro:
As the challenge describes, there is separation between logs that arrives from docker containers, and logs that arrives from K8s.

My solution for K8s, is to have for each App image, a POD with two containers, one is the app itself (which only writes stdout/stderr logs), and the other is “reader” container, both share the /var/log/ volume. The POD’s yaml defines to the app container to write its stdout/stderr logs to /var/log/log.out, and the “reader” container reads this log file, and send it to LogStash using telnet. Logstash is connected to ElasticSearch service, and sends those apps logs to ElasticSearch.

ElasticSearch’s POD asks for storage (using Persistent Volume Claim) to save the logs on Persistent Volume (PV), so even if we delete the ElasticSearch’s POD, the logs wouldn’t be deleted. ElasticSearch saves the logs it gets from Logstash.

Then we want to show those apps logs to the DevOps engineer. As for the description of the exercise, Kibana is enough to show the logs of the apps to the engineer, but I still created my own PHP+APACHE website, that reads the logs from ElasticSearch (using API), and shows those logs, for each host.

<u>This is the architecture of the K8s cluster:</u>



My recommendation for app image outside the K8s cluster, is to put it inside the cluster (in POD), so it would be capable to save its logs on the K8s’s ELK.
But I still created a whole different solution for dockers, which you can read on the end of the document.

Imubit home exercise - Yagel Ardan

The solution for Kubernetes:

Notice to wait until step is over (check if its up and running), before you continue to the next step.

1. Take the repo from git:
   git clone https://github.com/yageldhn/k8s_dockers_logging.git

2. Run minikube, using (recommended on different tab):
   minikube --memory 4096 --cpus 2 start
   (for me it was enough resources, you can change it different values).

3. Enter to the repository:
   cd k8s_dockers_logging/

4. First let's create a PVC (a request for 2GB of storage, in the cluster), so I could have Persistence Volume, for ElasticSearch (so the data won't be deleted on restart of the pod):
   kubectl create -f k8s/deployment_elasticsearch_pvc.yaml

5. Create ElasticSearch Pod, and Service (So ElasticSearch could talk to Kibana and Logstash in the cluster):
   kubectl create -f k8s/deployment_elasticsearch.yaml
   kubectl create -f k8s/deployment_elasticsearch_service.yaml

   To check if the ElasticSearch Service & POD, are up and running:
   1. Run :
   kubectl proxy --address='0.0.0.0' --port=8001 --accept-hosts='.*'
   2. Enter:
   http://localhost:8001/api/v1/namespaces/default/services/elasticsearch-service:serving/proxy/
   And check if you see: **"tagline" : "You Know, for Search".**

   ***note:** In the ElasticSearch POD, I used my own ElasticSearch image, that uses JAVA 8. The problem is that the default ElasticSearch image comes with JAVA 9/10, which is not supported (references - https://www.elastic.co/guide/en/elasticsearch/reference/current/setup.html, https://discuss.elastic.co/t/docker-elastic-6-3-with-java-8/136028).

6. Create LogStash Pod, and Service:
   - Run the Pod by:
     kubectl create -f k8s/deployment_logstash.yaml

   - Run the Service by:
     kubectl create -f k8s/deployment_logstash_service.yaml
7. You can edit deployment_app_and_logger.yaml, and put any app image you want (which writes stdout/stderr in the yaml (instead of yageldhn/create-random-logs) - *notice to put the correct command to run the app in the args of the container*.

Imubit home exercise - Yagel Ardan

8. Run the Pod (which has two containers - app and “reader”, both share same /var/log directory) by:
   kubectl create -f k8s/deployment_app_and_logger.yaml

9. Create Website Pod, and Service:

   - Run the Pod by:
     kubectl create -f k8s/deployment_website.yaml

   - Run the Service by:
     kubectl create -f k8s/deployment_website_service.yaml

   To check if the Website Service & POD, are up and running:
   1. Run :
   kubectl proxy --address='0.0.0.0' --port=8001 --accept-hosts='.*'
   2. Enter:
   http://localhost:8001/api/v1/namespaces/default/services/website-service:serving/proxy/
   And check if you see Host of the app POD. Click it, and check if you see logs (on the box).

10. Create Kibana Pod, and Service:
    - Run the Pod by:
      kubectl create -f k8s/deployment_kibana.yaml

    - Run the Service by:
      kubectl create -f k8s/deployment_kibana_service.yaml

11. Now, in order to run the cluster to be access by web, run:
    kubectl proxy --address='0.0.0.0' --port=8001 --accept-hosts='.*'

12. Enter to Kibana and my PHP website:
    http://localhost:8001/api/v1/namespaces/default/services/kibana-service:serving/proxy/
    Go to “Discover” tab, Enter “*” in the index pattern, then select @timestamp field, to view the logs.

13. Enter to the PHP+Apache website, to view the logs:
    http://localhost:8001/api/v1/namespaces/default/services/website-service:serving/proxy/

Imubit home exercise - Yagel Ardan

The solution for Dockers (OPTIONAL):

As I mentioned above, I recommend to put the app image inside the K8s cluster (so it would share the same ELK stack). But if you still wants to save the logs on ELK stack, outside the K8s cluster, read the following instructions:

My solution for dockers is to have for each application container (which only writes stdout/stderr logs), another container (a “reader” container), which gets the logs from the app container, and send it to ELK (by sending the logs to LogStash, via TCP protocol).
Because on a host there could be many docker containers, and in K8s it could separate each two containers on each pod, the solution is a little different between dockers, and K8s.

In my solution for docker container, I have apps containers (that writes stdout/stderr logs), and for each app, there is need to run another container (“read_and_send_apps_logs:latest”), which gets the container ID of an app, gets the logs of the container (by sharing the docker.sock of the host, so it could run “docker logs <container id>”) and send it to logstash (using telnet).

All the below commands, needs to be run from home directory (.../k8s_dockers_logging), so all the paths will be correct:

1. We need to create local network, to imitate production network:
   sudo docker network create mynetwork

2. Pull ELK images:
   sudo docker pull elasticsearch:6.6.1
   sudo docker pull kibana:6.6.1
   sudo docker pull logstash:6.6.1

3. Create ElasticSearch with persistence data (volume the data outside the container):
   sudo docker run -dit --name elasticsearch -h elasticsearch --net mynetwork -p 9200:9200 -p 9300:9300 -v $(pwd)/elasticsearch/data/:/usr/share/elasticsearch/data/ -e "discovery.type=single-node" elasticsearch:6.6.1

4. Create Kibana container:
   sudo docker run -dit --name kibana -h kibana --net mynetwork -p 5601:5601 kibana:6.6.1

5. Let’s check if ElasticSearch and Kibana are up and running properly:
   - Enter to localhost:9200, and check if you see “You know, for seach”.
   - Enter to localhost:5601, and check if see the app without errors.
   - If you have any problem, check if you in the right directory, and wait a little for the apps to come up.

Imubit home exercise - Yagel Ardan

6. If the apps are up and running, lets run LogStash (we gonna run it, with GROK definition in LogStash.conf, in order to parse the logs. The input for LogStash is TCP):
   sudo docker run -dit --name logstash -h logstash --net mynetwork -p 5044:5044 -p 9500:9500 -v $(pwd)/logstash/pipeline/:/usr/share/logstash/pipeline -v $(pwd)/logstash/config/logstash.yml:/usr/share/logstash/config/logstash.yml logstash:6.6.1 logstash -f /usr/share/logstash/pipeline/logstash.conf

7. Let's run container that produce random stdout/stderr logs, at random intervals:
   sudo docker pull yageldhn/create-random-logs:latest
   sudo docker run -dit yageldhn/create-random-logs:latest
   - you can see the stdout/stderr logs by sudo docker logs -f <container id>.
8. Create container, that get the logs from "create-random-logs", and send it to logstash via TCP (using telnet):
   sudo docker pull yageldhn/read_and_send_apps_logs:latest

   Before running the following command, notice to put in **LOGS_CONTAINER_ID**, the container id of the app container(!):
   sudo docker run -dit -e LOGS_CONTAINER_ID=123456789 **-e LOGSTASH_PORT=9500 --net mynetwork -v /var/run/docker.sock:/var/run/docker.sock yageldhn/read_and_send_apps_logs:latest**

   Now you can enter to Kibana again and see the new logs in localhost:5601.

9. Although Kibana is enough to see the logs (from the details of the assignment), I still made PHP application that take via the API of ElasticSearch the logs, and order it by hosts:
   sudo docker pull yageldhn/website:latest

   sudo docker run -dit --name website --network mynetwork -p 5000:80 -p 80:9200 yageldhn/website:latest

   You can now enter to localhost:5000, and click on a host, and see the logs inside a box.