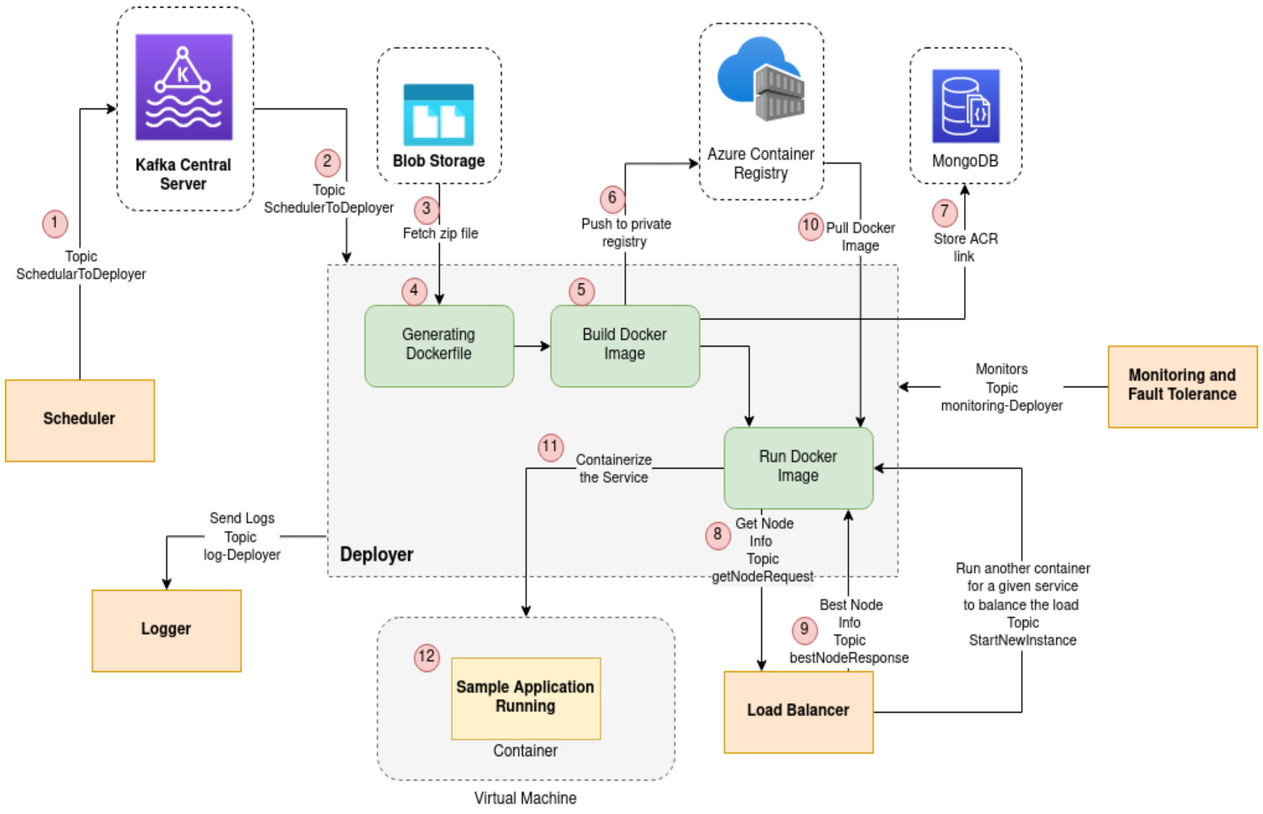
DEPLOYER SUB-SYSTEM

Introduction of deployer

- Deployer is the sub-system who is responsible for all the deployments of the app-developer application services.
- Deployer consumes requests from 2 sub-systems Scheduler and load-balancer.
- Whenever time arrives of the deployment of any application scheduler will request the deployer to deploy the application.
- Deployer will generate Dockerfile, build the image, store it to ACR.
- Now, the deployer asks for the best node from the load-balancer.
- Deployer will pull the image from ACR and run it on the node received from load-balancer.
- Deployer consumes the requests from load-balancer for scale-up and scale-down of the specific service on any given application.
- Scaling up means adding one more container for the given service, and scaling down means stopping and removing one of the containers for the specific service of any given application.
- Load-balancer does these requests when load increases/decreases on any container.

Deployer interaction with other sub-systems



Kafka Central Server
Blob Storage
Azure Container Registry
MongoDB
1
Topic SchedularToDeployer
2
Topic SchedulerToDeployer
3
Fetch zip file
4
5
6
Push to private registry
7
Store ACR link
10
Pull Docker Image
Generating Dockerfile
Build Docker Image
Run Docker Image
Monitors Topic monitoring-Deployer
Monitoring and Fault Tolerance
Scheduler
11
Containerize the Service
8
Get Node Info Topic getNodeRequest
Send Logs Topic log-Deployer
Deployer
Logger
9
Best Node Info Topic bestNodeResponse
Run another container for a given service to balance the load Topic StartNewInstance
12
Sample Application Running
Container
Virtual Machine
Load Balancer

DEPLOYER SUB-SYSTEM

Explanation of each step shown in above diagram

1. Scheduler produces the request for scheduling the application through the Kafka server.
2. Consuming the request of the scheduler from the Kafka server for deployment of some application. Request can be in 4 modes:
    a. **run:** Directly run the container of specified services because its image has already been formed and pushed to ACR.
    b. **build:** Just build the image for the specified services, push it to ACR and store path info in the data-base.
    c. **build and run:** First build the image, push to ACR and then run the image by pulling it from ACR.
    d. **stop:** Stop and remove the container for the specified services. Information of which service is running on which container and node will be found out from the service registry.
3. Fetch/Download the zip file from the blob-storage of specified service containing code, config.json and requirements.txt of requested application.
    a. We have used the concept of volume mounting here to execute the commands of docker. We need to set-up a docker environment inside the container where the deployer is running.
    b. We cannot do the setup of docker inside docker container, so what’s the solution?
    c. Here, to solve the above problem we have used volume mounting(Refer DOCKER file to get more info about volume mounting concept).
    d. Volume mounting is being done on source path (path of folder inside the container file system) “./services” and destination path (path of the host machine) “~/platform-deployer/services/”.
4. Unzip the file, generate a Dockerfile by reading the config.json file (using the concept of RPC) and adding it to the same directory where config.json resides.
5. Build docker image using the Dockerfile generated in above step.
    a. We cannot use docker commands inside the docker container as the deployer itself is running inside one container.
    b. We will take access to the terminal through the ssh, of our host machine and then go to the path where volume mounting is being done.
    c. Now, here execute the command of building a docker image.
6. Push the generated docker image to Azure Container Registry (ACR) for future use.

   a. After pushing the docker image to ACR, we will remove it from the host machine.
   b. We will also remove the down-loaded zip file and also the un-zipped folder from the container memory.
   c. It will automatically remove them from the host machine memory as well because of the volume mounting concept.

7. Store ACR image path and the port mentioned inside config.json file which is the port on which service server will be accepting the requests. This port will be used for port-mapping while running the image.
8. Produce the request to load-balancer through Kafka server for sending the info of best node/VM where we can deploy the application service.
9. Consume info(including shh login credentials) of the best node/VM produced by the load-balancer.
10. Pull the docker image stored in ACR.
    a. We will first make an ssh connection with the node given by load-balancer.
    b. Now, we will pull the docker image from ACR.
11. Run the pulled image on the node/VM given by load-balancer.
    a. Run the pulled docker image on the same VM/node given by load-balancer.
    b. Check that the image was successfully deployed on the container by checking the status of the container as ‘running’.
    c. Send the container info such as container name, container start time, port on which the out-side world can access the deployed service and service info like app id, app name , service name, etc to the node-manager.
    d. Node manager will store this info to the service-registry.
    e. Send the response back to the scheduler. It can be of 2 types
        i. Status = True: It means all the services required by the scheduler were deployed successfully.
        ii. Status = False: It means that all services have not been deployed properly they have some errors in code. Response will contain the list of services failed to run on the docker container.
    f. Ideally the response must also contain the IP and port where the app-server is being deployed, that we need to give to the app-developer. Using these IP and port the end-users will access the application of the app-developer. To achieve this we can tell the app-developer that they keep the name of one of the services as “app-server” which will be the server handling the request of end-users of the application.

DEPLOYER SUB-SYSTEM

Future Scope

- Store the request came in DB came from the scheduler or any other service to the deployer.
- These will be help-full in fault-tolerance. Whenever a deployer gets down and is restarted or a new instance starts it will first complete the pending request stored in the DB and serve the new request. All requests are being served parallely in the deployer.
- Flow of serving the request
  - Store the request in the DB, it will contain one token which will be unique identifying the request. These token will also be stored into the DB.
  - Let’s say 2 deployer instances are running Dep-1 and Dep-2.
  - Dep-1 accepted the request from the scheduler and stored the request in the DB.
  - Now, Dep-1 sent the request to the load-balancer for the best node info. In these requests we will send one unique access-token to the load-balancer which load-balancer will send back to the deployer with best node info.
  - We will store these access-token in the DB which will be mapped with one of the request stored in DB, so that when we receive the best node info from load-balancer we can bifurcate that these node is for which request.
  - Because It may be the case that request to load-balancer is done by Dep-1 and response is consumed by Dep-2.
  - After successfully serving the request we will remove the entry from DB.
  - In response to the scheduler also we will send the token sent by the scheduler.