In this assignment, I will cover the usage of queueing, using Kafka, and caching technologies using Redis, to implement an event driven architecture for the Asset project.
We will review the implementation of Kafka producer at the organization service, on which a message will be sent to a Kafka consumer at the Asset service.
After publishing a message successfully, we will use the RedisRepository at Asset project to lookup for the asset that was passed through the Kafka stream.

Mvn clean package docker:build

```
 ---> d23bdf5b1b1b
ProgressMessage{id=null, status=null, stream=null, error=null, progress=null, progressDetail=null}
Successfully built d23bdf5b1b1b
Successfully tagged example:latest
[INFO] Built example
[INFO] ------------------------------------------------------------------------
[INFO] Reactor Summary:
[INFO]
[INFO] Config Server ...................................... SUCCESS [  9.449 s]
[INFO] Eagle Eye Licensing Service ........................ SUCCESS [  8.545 s]
[INFO] Eagle Eye Organization Service ..................... SUCCESS [  7.658 s]
[INFO] Eureka Server ...................................... SUCCESS [  5.511 s]
[INFO] Zuul Proxy Server .................................. SUCCESS [  5.548 s]
[INFO] assignment3_2-parent-pom 0.0.1-SNAPSHOT ............ SUCCESS [  0.243 s]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 37.545 s
[INFO] Finished at: 2019-05-15T23:14:27+03:00
[INFO] ------------------------------------------------------------------------
➜  Assignment3_2
```

Spring 3: Find 6: TODO Terminal CheckStyle Gerrit

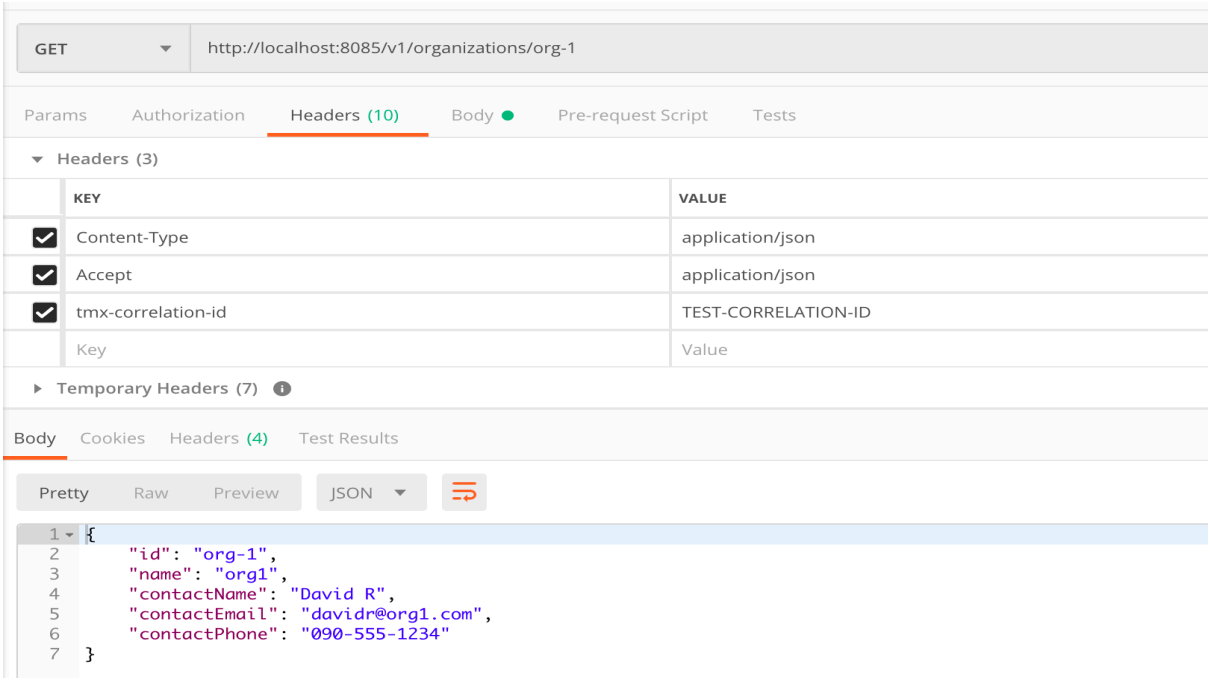
docker-compose -f docker/common/docker-compose.yml up

```
organizationservice_1  |       ssl.cipher.suites = null
organizationservice_1  |       ssl.truststore.type = JKS
organizationservice_1  |       security.protocol = PLAINTEXT
organizationservice_1  |       retries = 0
organizationservice_1  |       max.request.size = 1048576
organizationservice_1  |       value.serializer = class org.apache.kafka.common.serialization.ByteArraySerializer
organizationservice_1  |       ssl.truststore.location = null
organizationservice_1  |       ssl.keystore.password = null
organizationservice_1  |       ssl.keymanager.algorithm = SunX509
organizationservice_1  |       metrics.sample.window.ms = 30000
organizationservice_1  |       partitioner.class = class org.apache.kafka.clients.producer.internals.DefaultPartitioner
organizationservice_1  |       send.buffer.bytes = 131072
organizationservice_1  |       linger.ms = 0
organizationservice_1  |
organizationservice_1  | 2019-05-16 19:37:33.486  INFO 28 --- [nio-8085-exec-6] o.a.kafka.common.utils.AppInfoParser     : Kafka version : 0.9.0.1
organizationservice_1  | 2019-05-16 19:37:33.487  INFO 28 --- [nio-8085-exec-6] o.a.kafka.common.utils.AppInfoParser     : Kafka commitId : 23c69d
assetservice_1         | 2019-05-16 19:37:33.754 DEBUG 30 --- [afka-listener-1] r.t.a.e.h.OrganizationChangeHandler      : Received a message of t
assetservice_1         | 2019-05-16 19:37:33.755 DEBUG 30 --- [afka-listener-1] r.t.a.e.h.OrganizationChangeHandler      : Received a UPDATE event
organizationservice_1  | 2019-05-16 19:38:05.907 DEBUG 28 --- [nio-8085-exec-7] r.t.o.utils.UserContextFilter            : Entering the UserContex
organizationservice_1  | 2019-05-16 19:38:05.907 DEBUG 28 --- [nio-8085-exec-7] r.t.o.utils.UserContextFilter            : I am entering the organ
```

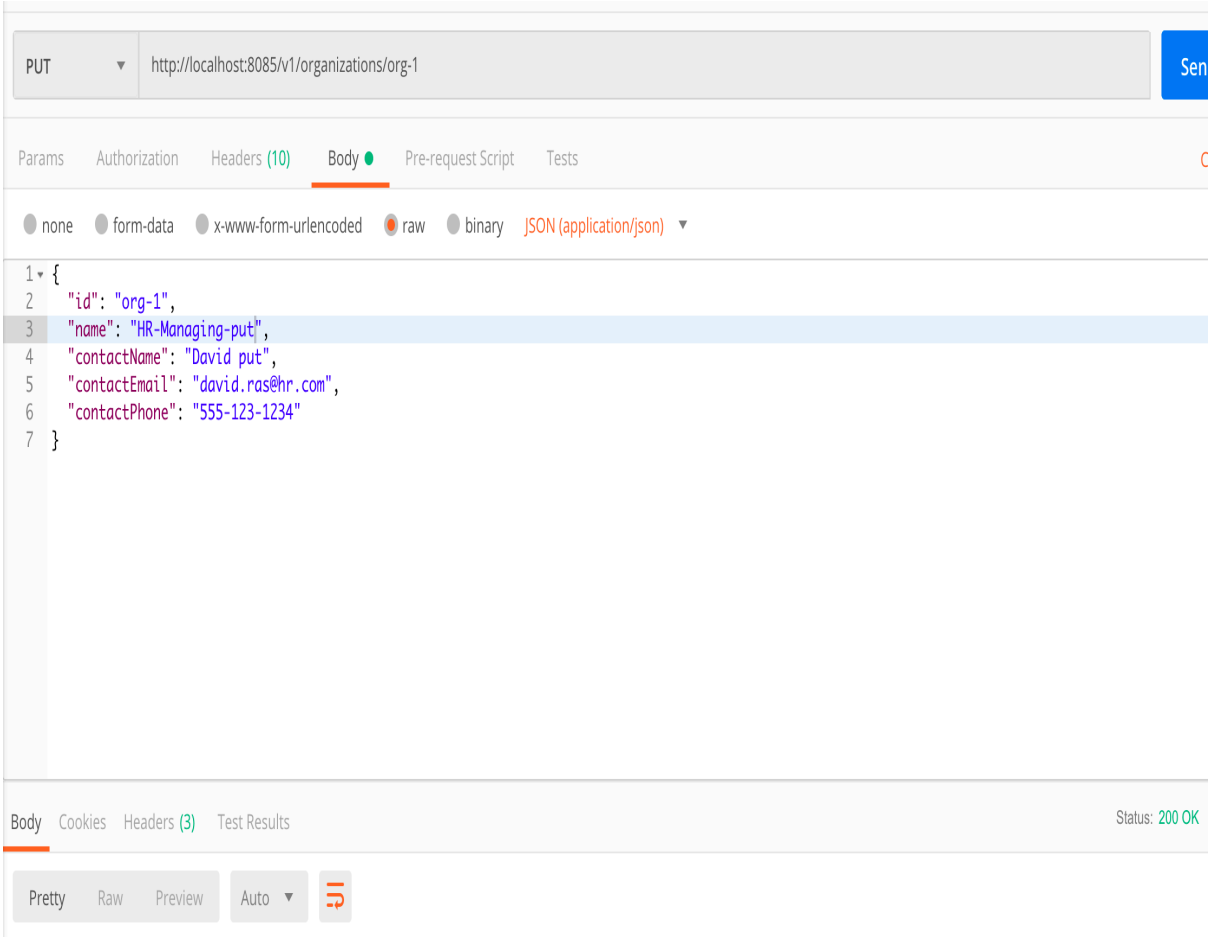
6: TODO Terminal CheckStyle Gerrit Spring

Dockerfile detection: You may setup Docker deployment run configuration for the following file(s): zuulsvr/src/main/docker/Dockerfile assetservice/src/main/docker/Dockerfile configsvr/src/main/dock

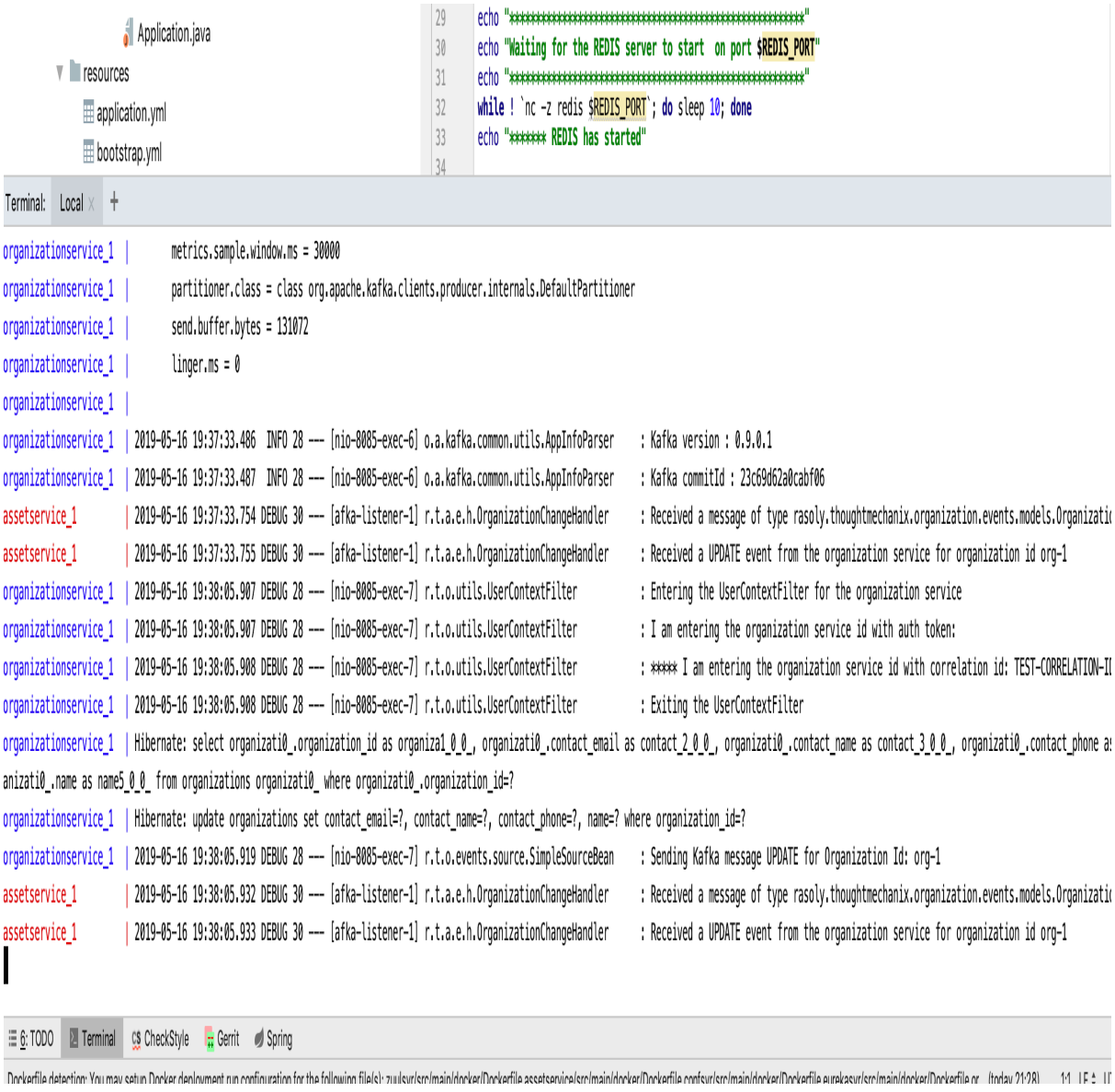
Getting an organization details.



Updating organization details using “Put”



The output shows that we've updated the organization entity



Asserting that both email, name and contact name was changed.