ENGINEERING BLOG

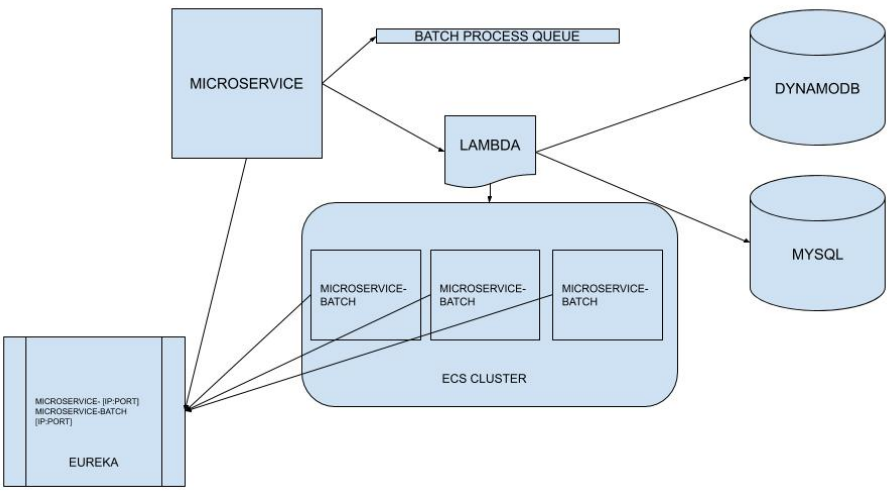
ASYNC BATCH JOB

admin July 18, 2019 Leave a comment Edit

This mechanism is useful to run CPU/Memory intensive operations in a separate instance of a microservice. Using this mechanism doesn't effect the performance of client facing instances.

Components involved :

1. AWS SQS
2. Lambda
3. DynamoDB
4. MySQL
5. AutoScalingGroup



ARCHITECTURE OVERVIEW

PROCESS OVERVIEW:

1. Microservices use AsyncBatchJobRequestHelper and invoke queueMessage which sends message to SQS queue and invokes a lambda function

```
public void queueMessage(String queueName, String
stringMessage, boolean isAsyncBatch) {

                // 1. Send message to SQS queue
                sqsRequestHelper.queueMessage(queueName,
stringMessage);
                APP_LOGGER.info("Queued message : " +
stringMessage + ", in queue :" + queueName);

                APP_LOGGER.info("IsAWSEnvironment : " +
isAWSEnvironment + ", isAsyncBatch : " + isAsyncBatch);
                // 2. Invoke lambda function that performs
auto-scaling based on queue size
                if (isAWSEnvironment && isAsyncBatch) {
                        APP_LOGGER.info("Invoking lambda
with name : " + asyncBatchJobLambdaName);
                        InvokeRequest invokeRequest = new
InvokeRequest().withFunctionName(asyncBatchJobLambdaName);

awsLambdaClient.invoke(invokeRequest);
                }

        }
```

2. Lambda function fetches metadata from DynamoDB and performs the following operations :

- Fetch approximate count of events in SQS queue

- Fetch current running instances count in the cluster
- Calculate and update required instance count in cluster

3. Each task that spins up as part of this service, shall register to Eureka as a BATCH task thus isolating itself from actual services’ requests

4. Each task that spins up as a batch task has an instance of AsyncNodeManager running which takes care of termination of the instance after the operation has been processed

```
@Transactional
        @Scheduled(fixedDelayString =
"${AsyncBatchJob.NodeManager.FixedDelay}",
initialDelayString =
"${AsyncBatchJob.NodeManager.InitialDelay}")
    private void manageNodeLife() {

        boolean isAWSEnvironment =
Arrays.asList(environment.getActiveProfiles()).contains("doc
ker");
        boolean isJobEnabled =
Boolean.valueOf(environment.getProperty(Constants.ASYNC_BATC
H_JOB_ENABLED));
        int currentNodeJobCount =
AsyncBatchJobListener.getCurrentNodeBatchJobCount();

        APP_LOGGER.info("Current node job count " +
currentNodeJobCount + ", at time " +
System.currentTimeMillis() + ", is this an AWS Environment :
" + isAWSEnvironment + ", is the job enabled : " +
isJobEnabled);

        // 1. Check number of jobs running in the task
        if (currentNodeJobCount == 0 && isAWSEnvironment &&
isJobEnabled) {

            String instanceId =
EC2MetadataUtils.getInstanceId();
            String asyncBatchJobInstanceTableName =
environment.getProperty(Constants.ASYNC_BATCH_JOB_INSTANCE_T
ABLE_NAME);

            AttributeValue attributeValue = new
AttributeValue(instanceId);
```

```
            // 2. Perform a dynamoDB request to check the
total job count of instance
            GetItemRequest getItemRequest = new
GetItemRequest()

.withTableName(asyncBatchJobInstanceTableName)
                    .addKeyEntry(Constants.INSTANCE_ID,
attributeValue);

            GetItemResult getItemResult =
dynamoDBAsyncClient.getItem(getItemRequest);

            int totalInstanceJobCount =
Integer.parseInt(getItemResult.getItem().get(Constants.CURRE
NT_JOB_COUNT).getN());
            String instanceState =
getItemResult.getItem().get(Constants.INSTANCE_STATE).getS()
;
            APP_LOGGER.info("Found " + totalInstanceJobCount
+ " jobs running on instance, with listeners status : " +
isListening.get() + ", and instanceState : " +
instanceState);

            if (totalInstanceJobCount == 0 &&
isListening.get() && instanceState.equals(Constants.ACTIVE)
&& AsyncBatchJobListener.getCurrentNodeBatchJobCount() == 0)
{

                // 3. Stop listening to messages
                if (isListening.get()) {
                        APP_LOGGER.info("Stop all jms
listeners");
                        jmsListenerEndpointRegistry.stop();
```

```
                        isListening.set(false);
                }

                // 4. Set scale in protection false for the
instance
                APP_LOGGER.info("Setting scale in protection
to false for " + instanceId);

AWSAutoScalingUtil.setScaleInProtection(amazonAutoScalingCli
ent,
environment.getProperty(Constants.AWS_EC2_AUTO_SCALE_GROUP_N
AME), Arrays.asList(instanceId), false);

                // 5. Invoke BatchScalingBySQS lambda, which
performs required scale down operation
                APP_LOGGER.info("Invoking lambda that
performs scale down");
                InvokeRequest invokeRequest = new
InvokeRequest().withFunctionName(asyncBatchJobLambdaName);

awsLambdaClient.invoke(invokeRequest);

                // 6. Update asyncBatchJobInstance table
                APP_LOGGER.info("Updating dynamodb state of
the instance");
                Map<String, AttributeValue> dynamoDBKey =
new HashMap();
                dynamoDBKey.put(Constants.INSTANCE_ID, new
AttributeValue().withS(instanceId));

                UpdateItemRequest updateItemRequest = new
UpdateItemRequest()

.withTableName(asyncBatchJobInstanceTableName)
```

```
                        .withKey(dynamoDBKey)

.addAttributeUpdatesEntry(Constants.INSTANCE_STATE, new
AttributeValueUpdate().withValue(new
AttributeValue().withS(Constants.INACTIVE)).withAction(Attri
buteAction.PUT))

.addAttributeUpdatesEntry(Constants.DELETED_ON, new
AttributeValueUpdate().withValue(new
AttributeValue().withS("" + new
Date())).withAction(AttributeAction.PUT))

.addAttributeUpdatesEntry(Constants.DELETE_INSTANCE_REQUEST_
COUNT, new AttributeValueUpdate().withValue(new
AttributeValue().withN("1")).withAction(AttributeAction.ADD)
)

.addAttributeUpdatesEntry(Constants.LAST_UPDATED_ON, new
AttributeValueUpdate().withValue(new
AttributeValue().withS("" + new
Date())).withAction(AttributeAction.PUT));

                UpdateItemResult updateItemResult =
dynamoDBAsyncClient.updateItem(updateItemRequest);

APP_LOGGER.info(updateItemResult.toString());

            } else {

                if
(instanceState.equals(Constants.INACTIVE)) {
                        // 7. Set scale in protection true
for the instance
                        APP_LOGGER.info("Setting scale in
```

```
protection to true for " + instanceId);

AWSAutoScalingUtil.setScaleInProtection(amazonAutoScalingCli
ent,
environment.getProperty(Constants.AWS_EC2_AUTO_SCALE_GROUP_N
AME), Arrays.asList(instanceId), true);

                        // 8. Update asyncBatchJobInstance
table
                        APP_LOGGER.info("Updating dynamodb
state of the instance");
                        Map<String, AttributeValue>
dynamoDBKey = new HashMap();

dynamoDBKey.put(Constants.INSTANCE_ID, new
AttributeValue().withS(instanceId));

                        UpdateItemRequest updateItemRequest
= new UpdateItemRequest()

.withTableName(asyncBatchJobInstanceTableName)
                                    .withKey(dynamoDBKey)

.addAttributeUpdatesEntry(Constants.INSTANCE_STATE, new
AttributeValueUpdate().withValue(new
AttributeValue().withS(Constants.ACTIVE)).withAction(Attribu
teAction.PUT))

.addAttributeUpdatesEntry(Constants.DELETED_ON, new
AttributeValueUpdate().withValue(new
AttributeValue().withS("ACTIVE")).withAction(AttributeAction
.PUT))

.addAttributeUpdatesEntry(Constants.DELETE_INSTANCE_REQUEST_
```

```
COUNT, new AttributeValueUpdate().withValue(new
AttributeValue().withN("0")).withAction(AttributeAction.PUT)
)

.addAttributeUpdatesEntry(Constants.LAST_UPDATED_ON, new
AttributeValueUpdate().withValue(new
AttributeValue().withS("" + new
Date())).withAction(AttributeAction.PUT));

                        UpdateItemResult updateItemResult =
dynamoDBAsyncClient.updateItem(updateItemRequest);

APP_LOGGER.info(updateItemResult.toString());
                }

                // 9. Start listening to messages from SQS
queues
                if (!isListening.get()) {
                        APP_LOGGER.info("Start all jms
listeners");
                        jmsListenerEndpointRegistry.start();
                        isListening.set(true);
                }

            }

        }


    }
```

SQS :

This queuing system is specifically chosen as we require the approximate count of the number of events in the queue. Since Kafka inherently doesn't provide such features, we finalised on using SQS

CODE BASE:

LAMBDA :
svn://192.168.10.7/iconcept/branches/coreInfra/Lambdas/NextServiceBatchLambda

function : ***NextServiceBatchLambda***

CLASSES :

They are part of NextServiceArchetype

- AsyncBatchJobRequestHelper
- AsyncBatchJobNodeManager
- AsyncBatchJobConfiguration
- AsyncBatchJob

DYNAMODB :

NextServiceBatchInstanceJobCount, NextServiceBatchGroupCount

STEPS TO BE FOLLOWED TO CREATE A NEW JOB

1. Create a cluster for the Batch Service
2. Specify "nlptype=batch" as environment variable in the task definition of service
3. Create SQS queue corresponding to the batch job
4. Add metadata corresponding to process in DynamoDB
5. Provide the parameters as part of properties file

```
# ---------------------------------- ASYNC BATCH RELATED
PROPERTIES ------------------------------

# --------------------- AWS CREDENTIALS FOR ASYNCBATCH JOB
USER -------------------------


AsyncBatchJob.Enabled = true
AsyncBatchJob.RequestHelper.Enabled=true
AWS.AsyncBatchJob.Access.ID = AKIAIKEVDUT6QEWVW4PA
AWS.AsyncBatchJob.Secret.Key =
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
AWS.AsyncBatchJob.Region = us-east-1
AsyncBatchJob.Listener.ConcurrencyCount=1
AsyncBatchJob.Deferred.Acknowledgement=false

AWS.AsyncBatchJob.EC2Autoscale.GroupName =
EC2ContainerService-NextTimeTable-Batch-t2-medium-
EcsInstanceAsg-1D7LEUISH6LW0

AsyncBatchJob.BatchGroupTableName =
NextServiceBatchGroupCount
AsyncBatchJob.InstanceTableName =
NextServiceBatchInstanceJobCount

AsyncBatchJob.QueueNames = async_time_table_generation_qa

AsyncBatchJob.BatchGroupIdFor.async_time_table_generation_qa
= time_table_generation
AsyncBatchJob.JobClassFor.async_time_table_generation_qa =
com.nexteducation.nextTimeTable.jobs.TimeTableGenerationBatc
hJob

AsyncBatchJob.Lambda.Name = QA_NextServiceBatchLambda
```

```
AsyncBatchJob.NodeManager.FixedDelay = 60000
AsyncBatchJob.NodeManager.InitialDelay = 120000

# ------------------------------------ MASTER ------------
------------------------

spring.datasourceMaster.hikari.maximum-pool-size = 20
spring.datasourceMaster.hikari.minimum-idle = 5

# ------------------------------------ REPLICA ------------
------------------------

spring.datasourceReplica.hikari.maximum-pool-size = 20
spring.datasourceReplica.hikari.minimum-idle = 5

# ------------------------------------ MSR --------------
------------------------

spring.msrDatasource.hikari.maximum-pool-size = 20
spring.msrDatasource.hikari.minimum-idle = 5


spring.jpa.hibernate.ddl-auto=none
spring.profiles.active=docker,batch
```

AUTHORS : Peddi Rohith

Leave a comment

Comment

Post Comment