

(ED-0019) How-To: Setup IntelliJ for Banyan POC

IntelliJ Terminal

Engineering How-To Document

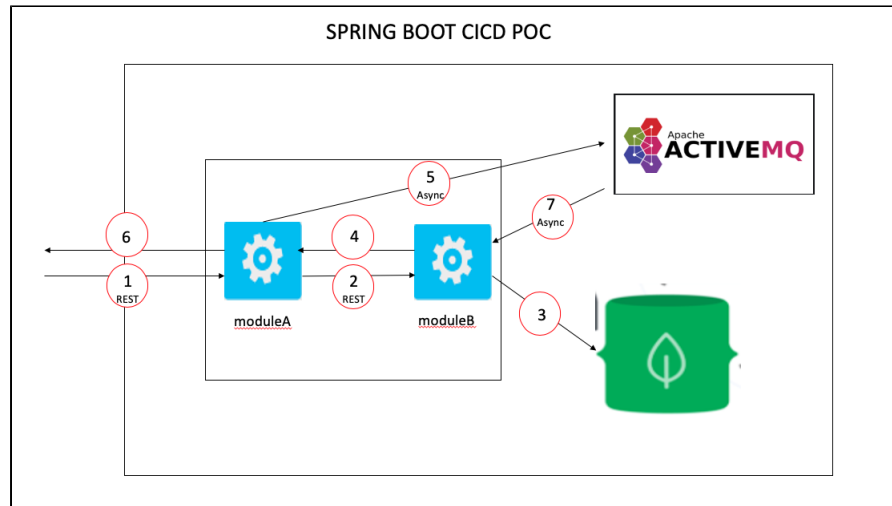
Versions:			Author:	Valerie Arena		Date:	24 Apr 2020
Product Version (i.e. MH-CURE)	Banyan		Reviewed by:			Reviewed date:	
	XXX		Last revised by:			Revised date:	

Table of Contents

- [Table of Contents](#)
- [Overview](#)
- [Step-by-step Guide](#)
 - [Install Docker](#)
 - [Install IntelliJ Mongo Plugin](#)
 - [Install IntelliJ Shell Script Plugin](#)
 - [Create /usr/local/mongodb Directory](#)
 - [Add Environment Variables](#)
 - [Clone the Repo and Import the Project](#)
 - [Create mongodb-and-activemq Run Config](#)
 - [Create Spring Boot Run Configs](#)
 - [Start MongoDB and ActiveMQ](#)
 - [Start moduleA and moduleB](#)
 - [Test cURLS](#)

Overview

The purpose of this document is to provide the steps for setting up the Banyan POC in IntelliJ. Once IntelliJ is set up, developers can deploy and debug the Banyan POC, which is made up of two Spring Boot applications. The following diagram is a highlevel diagram of the communication pathways between the modules and with MongoDB and ActiveMQ.



Step-by-step Guide



MongoDB and ActiveMQ are run as docker containers and do not need to be installed. The docker images are downloaded in the steps under Execute Modules Run Config.

Install Docker

Docker is used for the application containers that the Spring Boot applications run in.

1. Go to the following link and install the Docker Desktop on Mac. Installing the desktop will install both Docker and Docker Compose.

<https://docs.docker.com/docker-for-mac/install/>

2. Confirm that Docker is installed:

```
docker version
```

Install IntelliJ Docker Plugin

The Docker plugin is used to view your docker images and containers.

1. In IntelliJ go to Preferences Plugins and then select Marketplace.
2. Type in 'Docker' and then Install.
3. After restarting, got to Preferences Plugins and then select Installed.
4. Confirm that Docker is listed and is checked.

Install IntelliJ Mongo Plugin

The Mongo plugin is used to browse documents in the MongoDB database.

1. In IntelliJ go to Preferences Plugins and then select Marketplace.
2. Type in 'Mongo Plugin' and then Install.
3. After restarting, got to Preferences Plugins and then select Installed.
4. Confirm that Mongo Plugin is listed and checked.

Install IntelliJ Shell Script Plugin

The Shell Script plugin is used to execute shell scripts that automate the process of building/pulling docker images and deploying them.

1. In IntelliJ go to Preferences Plugins and then select Marketplace.
2. Type in 'Shell Script' and then Install.
3. After restarting, got to Preferences Plugins and then select Installed.
4. Confirm that Shell Script is listed and checked.

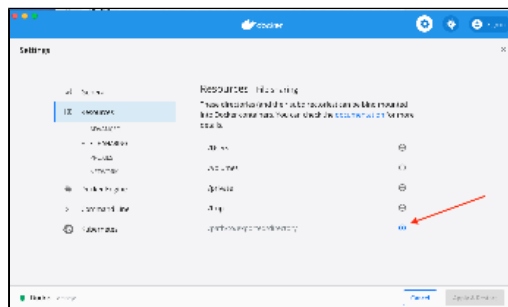
Create /usr/local/mongodb Directory

The mongodb directory will contain the database files. Saving the database files to /usr/local/mongodb ensures that database changes are not lost.

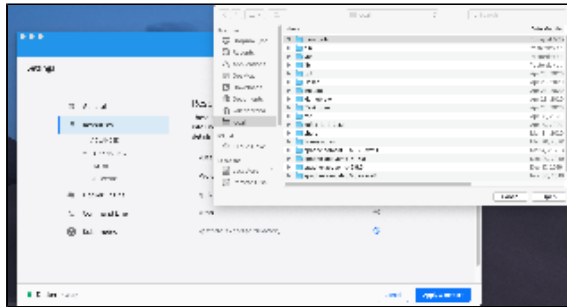
1. Cd to /usr/local.
2. Create mongodb and chnage the owner:

```
sudo mkdir mongodb  
sudo chown -R <user> mongodb/
```

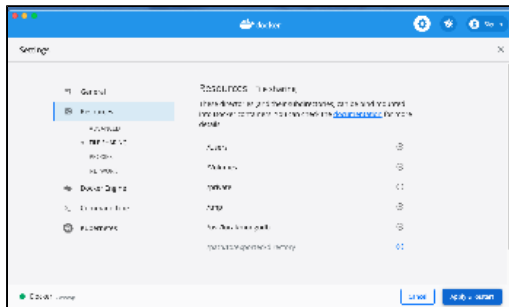
3. Open the Docker Desktop and go to Resources File Sharing and click the '+':



4. Select the /mongodb directory that you created:



5. Click apply and restart:



Add Environment Variables

The environment variables configure application properties. The environment variables are used in the property files listed under /resources of each module in the project.

! If IntelliJ is open, close and exit before doing the steps in this section

1. In Terminal, go to your home directory and open `.bash_profile`.
2. Add the following environment variables:

```
export INTELLIJ_PROJECT_PATH=[path to your IntelliJ project]
export SPRING_ACTIVE_PROFILE=local
export MONGO_DB_CONNECTION=mongodb://[IP of your mac]/poc
export MESSAGING_CONNECTION=tcp://[IP of your mac]:61616
export MODULEB_MESSAGE_ENDPOINT=http://[IP of your mac]:8090/message
export LOG_DIR=$HOME/MHCURE/logs/banyanpoc
```

3. Close and reopen Terminal. Type the following and confirm that you see the environment variables.

```
printenv
```

Clone the Repo and Import the Project

The modules of the POC are in the same project, which is a multi-module project.

1. Use SourceTree or Terminal to clone the following Bitbucket repo:

<ssh://git@devsvc.mobileheartbeat.com:7999/ban/banyan-spring-boot-cicd-poc.git>

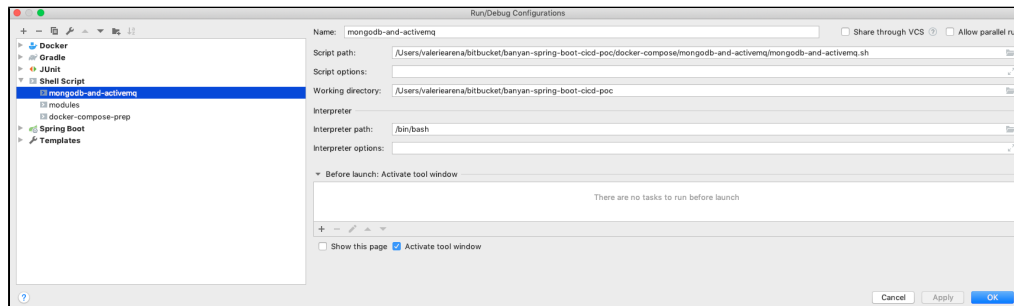
2. Import the project into IntelliJ.

Create mongodb-and-activemq Run Config

The mongodb-and-activemq run config, will do the following:

- The 'start' option will created and start the containers for MongoDB and ActiveMQ
- The 'stop' option will stop and remove the containers for MongoDB and ActiveMQ.

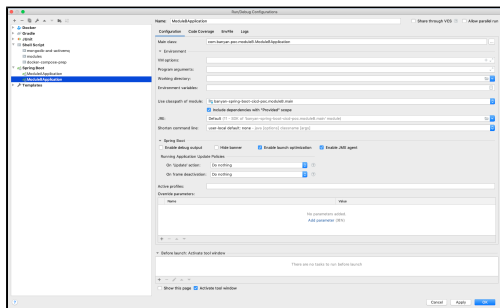
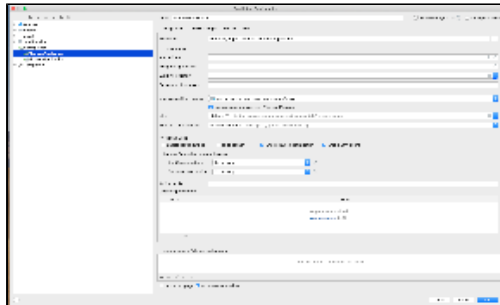
1. Go to Run Edit Configurations
2. Click + to Add New Configuration and select Shell Script.
3. Name the run config mongodb-and-activemq and select mongodb-and-activemq.sh from /docker-compose/modules/
4. Confirm that your run config looks like the following:



Create Spring Boot Run Configs

The Spring Boot run configs will start the Spring Boot apps.

1. Go to Run Edit Configurations.
2. Confirm that a Spring Boot run config exists for moduleA and moduleB.
3. If the run configs do not exist, create them. Confirm that they look like the following:



Start MongoDB and ActiveMQ



The docker images for MongoDB and ActiveMQ are downloaded in step #3.

1. Select the modules run config and run it.
2. In the IntelliJ Terminal (View Tool Windows Terminal), enter 'start' in Terminal:

```
ITAdmins-MacBook-Pro:banyan-spring-boot-cicd-poc valeriarena$ /bin/bash /Users/valeriarena/bitbucket/banyan-spring-boot-cicd-poc/docker-compose/mongodb-and-activemq/mongodb-and-activemq.sh
[start] [stop]: start
```

3. When starting for the very first time, the docker images for MongoDB and ActiveMQ will be downloaded. When the images have been downloaded and all the containers started, you should see the following:

```
Terminal: Local ▾ | Local (2) ▾ | +
ITAdmins-MacBook-Pro:banyan-spring-boot-cicd-poc valeriearena@ /bin/bash /Users/valeriearena/bitbucket/banyan-spring-boot-cicd-poc/docker-compose/mongodb-and-activemq/mongodb-and-activemq.sh
[start] [stop]: start
Creating network "mongodb-and-activemq_default" with the default driver
Pulling mongodb (mongo:latest)...
latest: Pulling from library/mongo
sha2d6d3073: Pull complete
f13ab9d734: Pull complete
9380da195c84: Pull complete
78bf9a5ad49e: Pull complete
3a7fb389f084: Pull complete
a7237201f1fa: Pull complete
8cdf1a69f7f1: Pull complete
d39c254c6294: Pull complete
13a7f7aa367: Pull complete
af6dc35f4c4e: Pull complete
d96994883ecb: Pull complete
d266b1d2dabb: Pull complete
3143b995d939: Pull complete
Digest: sha256:7a8d786a480919c2b67376cbe87c51f2bdaf139f1f6dc4382da4679335dc4
Status: Downloaded newer image for mongo:latest
Pulling activemq (webcenter/activemq)...
latest: Pulling from webcenter/activemq
7a1c9444202: Pull complete
9ee0ba75a87f: Pull complete
1f044b087cc7: Pull complete
da6805551b4: Pull complete
08f19a8a504: Pull complete
476f4422b21: Pull complete
5a994d718c09: Pull complete
313a84c85d3c: Pull complete
1a6a5a24e11: Pull complete
d2555998d21: Pull complete
1d23a53a104: Pull complete
864e802d1ca5: Pull complete
09d868aa4ae: Pull complete
Digest: sha256:39d15981c0d712ab1889466f5aac38a20f7da9d6c4c4675d9a9e4ee0b47b0
Status: Downloaded newer image for webcenter/activemq:latest
Creating mongodb_container ... done
Creating activemq_container ... done
```

4. In the Services window (View Tool Windows Terminal), connect to Docker. Confirm that you see the following:

The screenshot shows the Docker Desktop interface with the 'Log' tab selected for the 'activemq' container. The logs display the following messages:

```

2020-04-24 13:50:10,117 CRIT Supervisor running as root (no user in config file)
2020-04-24 13:50:10,117 WARN Included extra file "/etc/supervisor/conf.d/activemq.conf" during parsing
2020-04-24 13:50:10,117 WARN Included extra file "/etc/supervisor/conf.d/cron.conf" during parsing
2020-04-24 13:50:10,125 INFO RPC interface 'supervisor' initialized
2020-04-24 13:50:10,125 CRIT Server 'unix_http_server' running without any HTTP authentication checking
2020-04-24 13:50:10,125 INFO supervisord started with pid 1
2020-04-24 13:50:11,128 INFO spawned: 'cron' with pid 17
2020-04-24 13:50:11,129 INFO spawned: 'activemq' with pid 18
2020-04-24 13:50:12,571 INFO success: cron entered RUNNING state, process has stayed up for > than 1 seconds (startsecs)
2020-04-24 13:50:12,571 INFO success: activemq entered RUNNING state, process has stayed up for > than 1 seconds (startsecs)
  
```

5. Connect to ActiveMQ Web Console with admin/admin: <http://localhost:8161/admin>

6. To connect to MongoDB via IntelliJ Terminal:

- open a new IntelliJ Terminal and execute the following commands:

```

# Create the connection object
myConn=drv %>% connect(
  dbname="my_database",
  username="my_username",
  password="my_password",
  host="my_host",
  port="my_port"
)

# Send a query to the database
myQuery="SELECT * FROM my_table"

# Execute the query and retrieve the results
myResults=dbGetQuery(myConn, myQuery)

# Print the results
print(myResults)

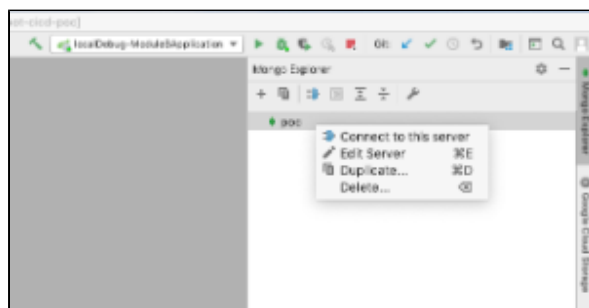
```

```
docker exec -it mongodb_container bash
mongo
use poc
db.version()
db.messages.count()
show collections
```

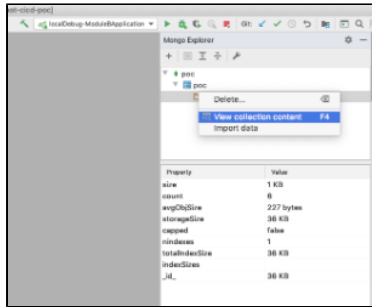
- to exit, type 'exit' and then 'exit' again:

```
> exit
bye
root@b4ff42f3896a:/# exit
exit
ITAdmins-MacBook-Pro:banyan-spring-boot-cicd-poc valeriearena$
```

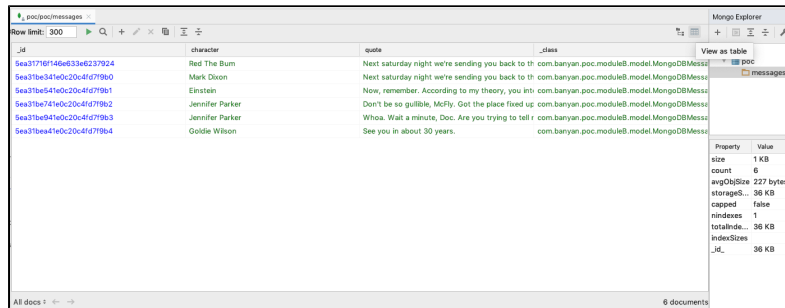
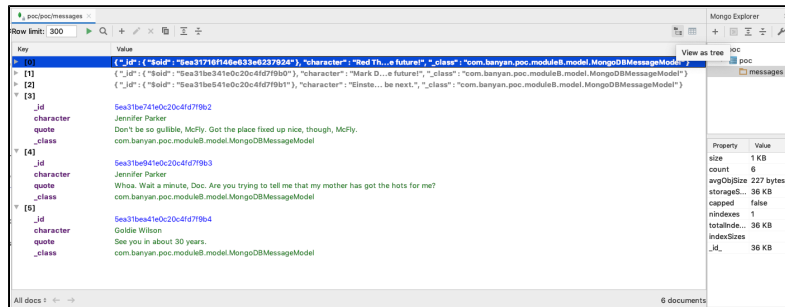
- Open Mongo Explorer and connect to the database:



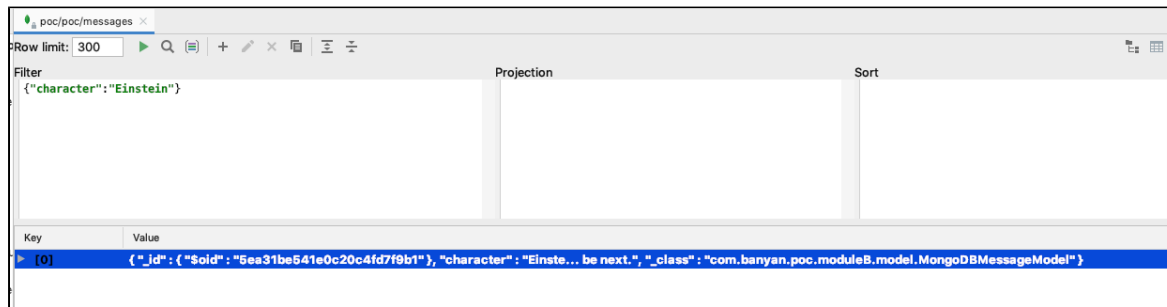
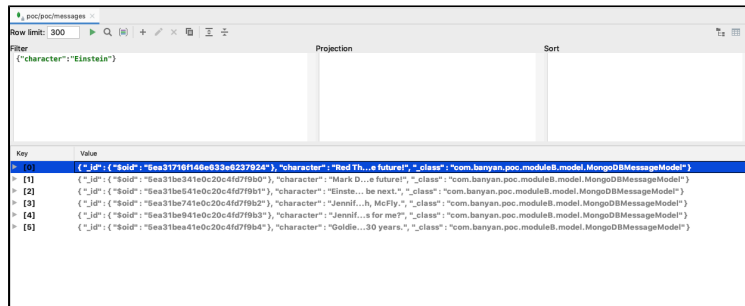
- View stats and documents:



- View the document in a tree view or a table view:



- If you want to run a query, click on the Magnifying glass icon:



- To learn more about the Mongo plugin and Mongo Explorer, go to <https://github.com/dboissier/mongo4idea> under Usage.

Start moduleA and moduleB

- Execute the Spring Boot debug config for moduleA.
- Execute the Spring Boot debug config for moduleA.

Test cURLS

The following cURLs make REST calls to three endpoints. Each endpoint executes a different communication pathway.

- Open two IntelliJ Terminals and tail the logs for moduleA and moduleB (in the directory that you configured for LOG_DIR environment variable) to confirm that you receive the expected responses.
- In a third terminal, execute the cURL.

/name/store

When a REST call is made to the /name/store endpoint, the following occurs:

- moduleA makes a REST call to the /message endpoint in moduleB.
- moduleB creates a Back To the Future message and saves it as a MongoDB document.
- moduleB returns the Back To the Future message to moduleA.
- moduleA returns the Back To the Future message in the response.

- Execute the following cURL.

cURL

```
curl -X POST http://localhost:8080/name/store
```

2. Confirm that the response and the logs are similar to the following:

```
Terminal: Local × Local (2) × Local (3) × +
ITAdmins-MacBook-Pro:banyanpoc valeriearena$ curl -X POST http://localhost:8080/name/store
{
  "backToTheFutureCharacter" : "Sally Baines",
  "backToTheFutureQuote" : "He's an absolute dream!"
}ITAdmins-MacBook-Pro:banyanpoc valeriearena$
```

```
Terminal: Local × Local (2) × Local (3) × +
ITAdmins-MacBook-Pro:banyanpoc valeriearena$ curl -X POST http://localhost:8080/name/store
{
  "backToTheFutureCharacter" : "Sally Baines",
  "backToTheFutureQuote" : "He's an absolute dream!"
}ITAdmins-MacBook-Pro:banyanpoc valeriearena$
```

```
Terminal: Local × Local (2) × Local (3) × +
ITAdmins-MacBook-Pro:banyanpoc valeriearena$ curl -X POST http://localhost:8080/name/store
{
  "backToTheFutureCharacter" : "Sally Baines",
  "backToTheFutureQuote" : "He's an absolute dream!"
}ITAdmins-MacBook-Pro:banyanpoc valeriearena$
```

/name/queue

When a REST call is made to the /name/queue endpoint, the following occurs:

- moduleA makes a REST call to the /message endpoint in moduleB.
- moduleB creates a Back To the Future message and saves it as a MongoDB document.
- moduleB returns the Back To the Future message to moduleA.
- moduleA creates a JMS message with the Back To the Future message and sends it to a JMS queue.
- moduleB JMS listener receives the JMS message and logs it.
- moduleA returns the Back To the Future message in the response.

1. Execute the following cURL:

cURL

```
curl -X POST http://localhost:8080/name/queue
```

2. Confirm that the response and the logs are similar to the following:

```
ITAdmins-MacBook-Pro:~banyanpoc valeriarena$ curl -X POST http://localhost:8080/name/queue
{
  "backToTheFutureCharacter" : "Einstein",
  "backToTheFutureQuote" : "Now, remember. According to my theory, you interfered with your parents' first meeting. If they don't meet, they won't fall in love, they won't get married and they won't have kids. That's why you r older brother's disappearing from that photograph. Your sister will follow, and unless you repair the damage, you'll be next."
}
ITAdmins-MacBook-Pro:~banyanpoc valeriarena$
```

[illegible][illegible]

/name/publish

When a REST call is made to the /name/publish endpoint, the following occurs:

- moduleA makes a REST call to the /message endpoint in moduleB.
- moduleB creates a Back To the Future message and saves it as a MongoDB document.
- moduleB returns the Back To the Future message to moduleA.
- moduleA creates a JMS message with the Back To the Future message and publishes it to a JMS topic.
- moduleB JMS listener receives the JMS message and logs it.
- moduleA returns the Back To the Future message in the response.

1. Execute the following cURL:

```
curl -X POST http://localhost:8080/name/publish
```

2. Confirm that the response and the logs are similar to the following:

```
Terminal: Local x Local (2) x Local (3) x +
ITAdmins-MacBook-Pro:banyan-spring-boot-cicd-poc valeriearena$ curl -X POST http://localhost:8080/name/publish
{
  "backToTheFutureCharacter" : "Einstein",
  "backToTheFutureQuote" : "What about all that talk about screwing up future events? The space-time continuum?"
}ITAdmins-MacBook-Pro:banyan-spring-boot-cicd-poc valeriearena$
```

```
Terminal: Local x Local (2) x Local (3) x +
ITAdmins-MacBook-Pro:~ valeriearena$ cd MHCIURE/Logs/banyanpoc/
ITAdmins-MacBook-Pro:banyanpoc valeriearena$ tail -f moduleA.log
2020-04-24 11:27:03.044 TRACE 12450 --- [http-nio-8080-exec-2] s.w.s.m.a.RequestMappingHandlerMapping : Mapped to com.banyan.poc.moduleA.controller.ModuleARestController#postName()
2020-04-24 11:27:03.046 TRACE 12450 --- [http-nio-8080-exec-2] .w.s.m.a.ServletInvocableHandlerMethod : Arguments: []
2020-04-24 11:27:03.046 INFO 12450 --- [http-nio-8080-exec-2] c.b.p.m.c.ModuleARestController : TEST
2020-04-24 11:27:03.046 INFO 12450 --- [http-nio-8080-exec-2] c.b.p.m.restclient.ModuleARestClient : Module A sending a request to Module B. uri=http://10.0.0.78:8090/message
2020-04-24 11:27:03.050 TRACE 12450 --- [http-nio-8080-exec-2] o.s.w.r.f.client.ExchangeFunctions : [1f4cd2f1] HTTP POST http://10.0.0.78:8090/message, headers=[]
2020-04-24 11:27:03.088 TRACE 12450 --- [reactor-http-nio-2] o.s.w.r.f.client.ExchangeFunctions : [1f4cd2f1] Response 200 OK, headers=[Content-Type:"application/json", Transfer-Encoding:"chunked", Date:"Fri, 24 Apr 2020 15:27:03 GMT"]
2020-04-24 11:27:03.090 INFO 12450 --- [http-nio-8080-exec-2] c.b.p.m.restclient.ModuleARestClient : Module A received response from Module B: MessageBean(backToTheFutureCharacter=Sam Baines, backToTheFutureQuote=No! It requires something with a little more kick...plutonium!)
2020-04-24 11:27:03.091 INFO 12450 --- [http-nio-8080-exec-2] c.b.p.m.c.ModuleARestController : Module A returning the response: MessageBean(backToTheFutureCharacter=Sam Baines, backToTheFutureQuote=No! It requires something with a little more kick...plutonium!)
2020-04-24 11:27:03.091 DEBUG 12450 --- [http-nio-8080-exec-2] m.m.a.RequestResponseBodyMethodProcessor : Using 'application/json', given [*/] and supported [application/json]
2020-04-24 11:27:03.091 TRACE 12450 --- [http-nio-8080-exec-2] m.m.a.RequestResponseBodyMethodProcessor : Writing [MessageBean(backToTheFutureCharacter=Sam Baines, backToTheFutureQuote=No! It requires something with a little more kick...plutonium!)]
2020-04-24 11:29:55.024 TRACE 12450 --- [http-nio-8080-exec-3] s.w.s.m.a.RequestMappingHandlerMapping : Mapped to com.banyan.poc.moduleA.controller.ModuleARestController#topic()
2020-04-24 11:29:55.025 TRACE 12450 --- [http-nio-8080-exec-3] .w.s.m.a.ServletInvocableHandlerMethod : Arguments: []
2020-04-24 11:29:55.025 INFO 12450 --- [http-nio-8080-exec-3] c.b.p.m.c.ModuleARestController : Module A received a request.
2020-04-24 11:29:55.025 INFO 12450 --- [http-nio-8080-exec-3] c.b.p.m.restclient.ModuleARestClient : Module A sending a request to Module B. uri=http://10.0.0.78:8090/message
2020-04-24 11:29:55.028 TRACE 12450 --- [http-nio-8080-exec-3] o.s.w.r.f.client.ExchangeFunctions : [4481d80b] HTTP POST http://10.0.0.78:8090/message, headers=[]
2020-04-24 11:29:55.072 TRACE 12450 --- [reactor-http-nio-3] o.s.w.r.f.client.ExchangeFunctions : [4481d80b] Response 200 OK, headers=[Content-Type:"application/json", Transfer-Encoding:"chunked", Date:"Fri, 24 Apr 2020 15:29:55 GMT"]
2020-04-24 11:29:55.074 INFO 12450 --- [http-nio-8080-exec-3] c.b.p.m.restclient.ModuleARestClient : Module A received response from Module B: MessageBean(backToTheFutureCharacter=Einstein, backToTheFutureQuote=What about all that talk about screwing up future events? The space-time continuum?)
2020-04-24 11:29:55.080 INFO 12450 --- [http-nio-8080-exec-3] c.banyan.poc.moduleA.jms.TopicProducer : Module A publishing JMS message to topic local-topic for Module B: {"backToTheFutureCharacter":"Einstein","backToTheFutureQuote":"What about all that talk about screwing up future events? The space-time continuum?"}
2020-04-24 11:29:55.134 INFO 12450 --- [http-nio-8080-exec-3] c.b.p.m.c.ModuleARestController : Module A returning the response: MessageBean(backToTheFutureCharacter=Einstein, backToTheFutureQuote=What about all that talk about screwing up future events? The space-time continuum?)
2020-04-24 11:29:55.135 DEBUG 12450 --- [http-nio-8080-exec-3] m.m.a.RequestResponseBodyMethodProcessor : Using 'application/json', given [*/] and supported [application/json]
2020-04-24 11:29:55.135 TRACE 12450 --- [http-nio-8080-exec-3] m.m.a.RequestResponseBodyMethodProcessor : Writing [MessageBean(backToTheFutureCharacter=Einstein, backToTheFutureQuote=What about all that talk about screwing up future events? The space-time continuum?)]
```

```
Terminal: Local x Local(2) x Local(3) +
2020-04-24 11:27:03.079 DEBUG 12451 --- [http-nio-8090-exec-3] o.s.data.mongodb.core.MongoTemplate : find using query: { "quote" : "No! It requires something with a little more kick...plutonium!"} fields: Document{{{}} f
or class: class com.banyan.poc.module8.model.MongoDBMessageModel in collection: messages
2020-04-24 11:27:03.083 DEBUG 12451 --- [http-nio-8090-exec-3] o.s.data.mongodb.core.MongoTemplate : Executing count: { "character" : "Sam Baines"} in collection: messages
2020-04-24 11:27:03.085 DEBUG 12451 --- [http-nio-8090-exec-3] o.s.data.mongodb.core.MongoTemplate : Executing count: { "quote" : "No! It requires something with a little more kick...plutonium!"} in collection: messages
2020-04-24 11:27:03.086 DEBUG 12451 --- [http-nio-8090-exec-3] m.m.a.RequestMappingHandlerMethodProcessor : Using 'application/json', given [*/] and supported [application/json]
2020-04-24 11:27:03.087 TRACE 12451 --- [http-nio-8090-exec-3] m.m.a.RequestMappingHandlerMethodProcessor : Writing [MessageBean(backToTheFutureCharacter=Sam Baines, backToTheFutureQuote=No! It requires something with a little
more kick...plutonium!)]
2020-04-24 11:27:03.088 TRACE 12451 --- [http-nio-8090-exec-3] o.s.web.servlet.DispatcherServlet : No view rendering, null ModelAndView returned.
2020-04-24 11:27:03.088 DEBUG 12451 --- [http-nio-8090-exec-3] o.s.web.servlet.DispatcherServlet : Completed 200 OK, headers={masked}
2020-04-24 11:29:55.041 TRACE 12451 --- [http-nio-8090-exec-5] o.s.web.servlet.DispatcherServlet : POST "/message", parameters={}, headers={masked} in DispatcherServlet 'dispatcherServlet'
2020-04-24 11:29:55.042 TRACE 12451 --- [http-nio-8090-exec-5] s.w.s.m.m.a.RequestMappingHandlerMapping : Mapped to com.banyan.poc.module8.controller.Module8RestController#postMessage()
2020-04-24 11:29:55.042 TRACE 12451 --- [http-nio-8090-exec-5] s.w.s.m.m.a.RequestMappingHandlerMethod : Arguments: []
2020-04-24 11:29:55.042 INFO 12451 --- [http-nio-8090-exec-5] c.b.p.m.c.Module8RestController : Module 8 received a request from Module A.
2020-04-24 11:29:55.050 DEBUG 12451 --- [http-nio-8090-exec-5] o.s.data.mongodb.core.MongoTemplate : Inserting Document containing fields: [character, quote, _class] in collection: messages
2020-04-24 11:29:55.053 DEBUG 12451 --- [http-nio-8090-exec-5] o.s.data.mongodb.core.MongoTemplate : findOne using query: { "id" : "5ea305f309e1e771ffb0cf57b"} fields: Document{{{}} for class: class com.banyan.poc.module8
.model.MongoDBMessageModel in collection: messages
2020-04-24 11:29:55.053 DEBUG 12451 --- [http-nio-8090-exec-5] o.s.data.mongodb.core.MongoTemplate : findOne using query: { "_id" : { "$oid" : "5ea305f309e1e771ffb0cf57b"} } fields: {} in db.collection: poc.messages
2020-04-24 11:29:55.054 DEBUG 12451 --- [http-nio-8090-exec-5] o.s.data.mongodb.core.MongoTemplate : Executing count: {} in collection: messages
2020-04-24 11:29:55.058 DEBUG 12451 --- [http-nio-8090-exec-5] o.s.data.mongodb.core.MongoTemplate : find using query: { "character" : "Einstein"} fields: Document{{{}} for class: class com.banyan.poc.module8.model.Mongo
DBMessageModel in collection: messages
2020-04-24 11:29:55.061 DEBUG 12451 --- [http-nio-8090-exec-5] o.s.data.mongodb.core.MongoTemplate : find using query: { "quote" : "What about all that talk about screwing up future events? The space-time continuum?"} f
ields: Document{{{}} for class: class com.banyan.poc.module8.model.MongoDBMessageModel in collection: messages
2020-04-24 11:29:55.066 DEBUG 12451 --- [http-nio-8090-exec-5] o.s.data.mongodb.core.MongoTemplate : Executing count: { "character" : "Einstein"} in collection: messages
2020-04-24 11:29:55.069 DEBUG 12451 --- [http-nio-8090-exec-5] o.s.data.mongodb.core.MongoTemplate : Executing count: { "quote" : "What about all that talk about screwing up future events? The space-time continuum?"} in
collection: messages
2020-04-24 11:29:55.071 DEBUG 12451 --- [http-nio-8090-exec-5] m.m.a.RequestMappingHandlerMethodProcessor : Using 'application/json', given [*/] and supported [application/json]
2020-04-24 11:29:55.071 TRACE 12451 --- [http-nio-8090-exec-5] m.m.a.RequestMappingHandlerMethodProcessor : Writing [MessageBean(backToTheFutureCharacter=Einstein, backToTheFutureQuote=What about all that talk about screwing u
p future events? The space-time continuum?)]
2020-04-24 11:29:55.072 TRACE 12451 --- [http-nio-8090-exec-5] o.s.web.servlet.DispatcherServlet : No view rendering, null ModelAndView returned.
2020-04-24 11:29:55.072 DEBUG 12451 --- [http-nio-8090-exec-5] o.s.web.servlet.DispatcherServlet : Completed 200 OK, headers={masked}
2020-04-24 11:29:58.774 INFO 12451 --- [DefaultMessageListenerContainer-1] c.b.poc.module8.jms.TopicSubscriber : Module 8 consuming JMS message published to topic: MessageBean(backToTheFutureCharacter=Einstein, backToT
heFutureQuote=What about all that talk about screwing up future events? The space-time continuum?)
```