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

IntelliJ Terminal

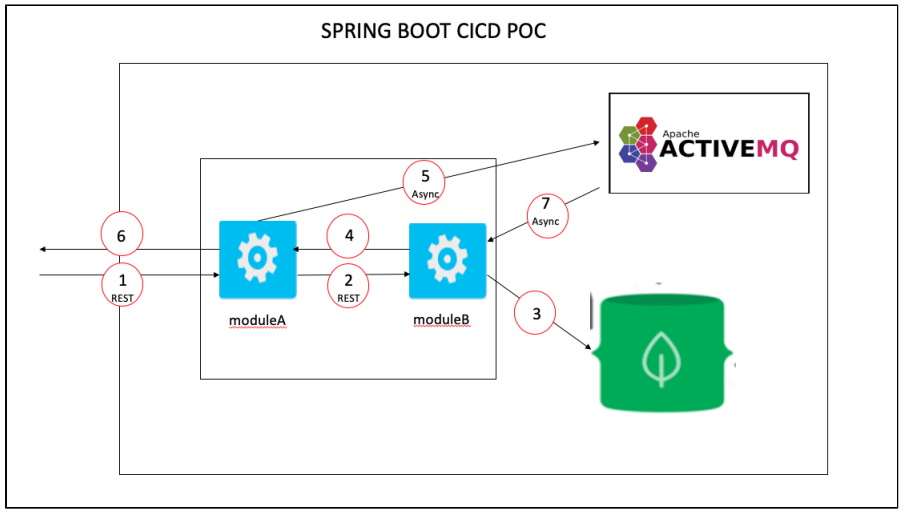
Engineering How-To Document

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

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.

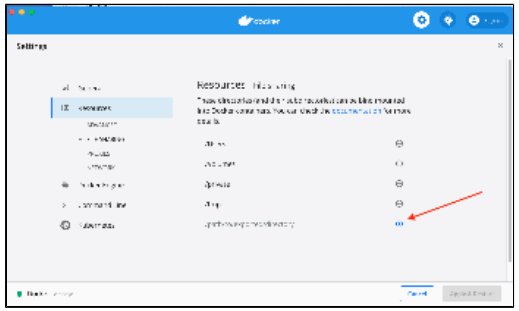
Create /usr/local/mongodb Directory

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

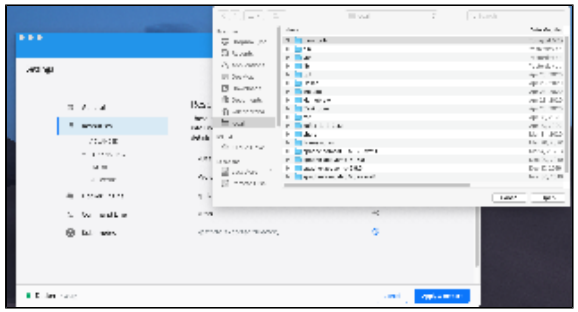
1. Cd to /usr/local.
2. Create mongodb and chnage the owner:

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

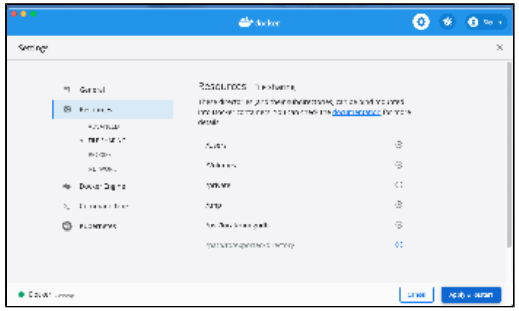
3. Open the Docker Desktop and go to Resources File Sharing and click the '+':



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

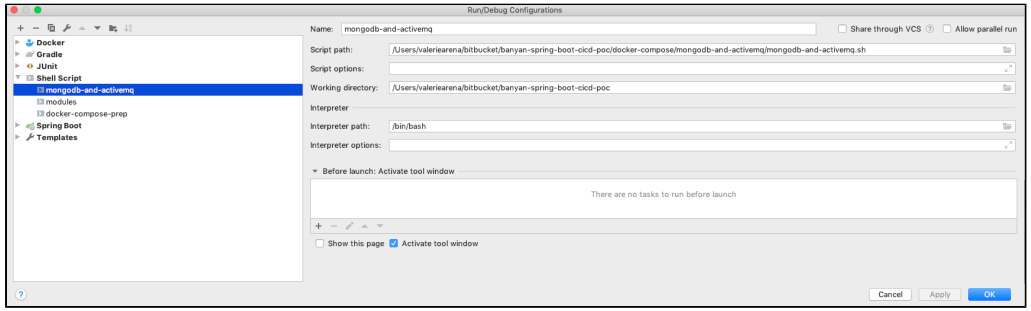
2. Import the project into IntelliJ.

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.

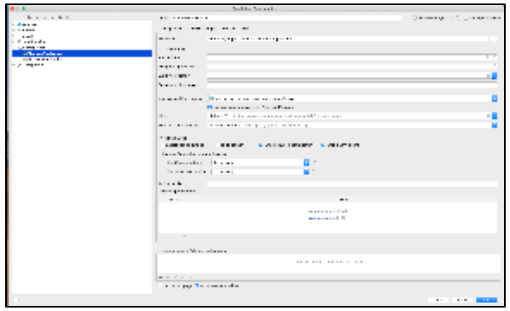
1. Go to Run Edit Configurations
2. Click + to Add New Configuration and select Shell Script.
3. Name the run config mongodb-and-activemq and select mongodb-and-activemq.sh from /docker-compose/modules/
4. Confirm that your run config looks like the following:

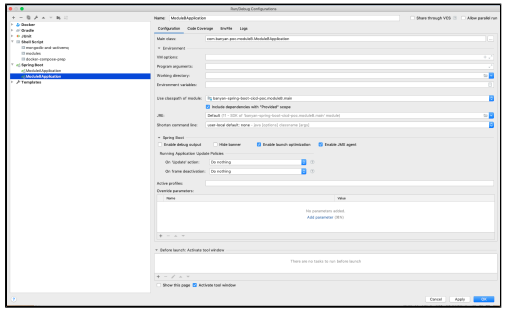


Create Spring Boot Run Configs

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:





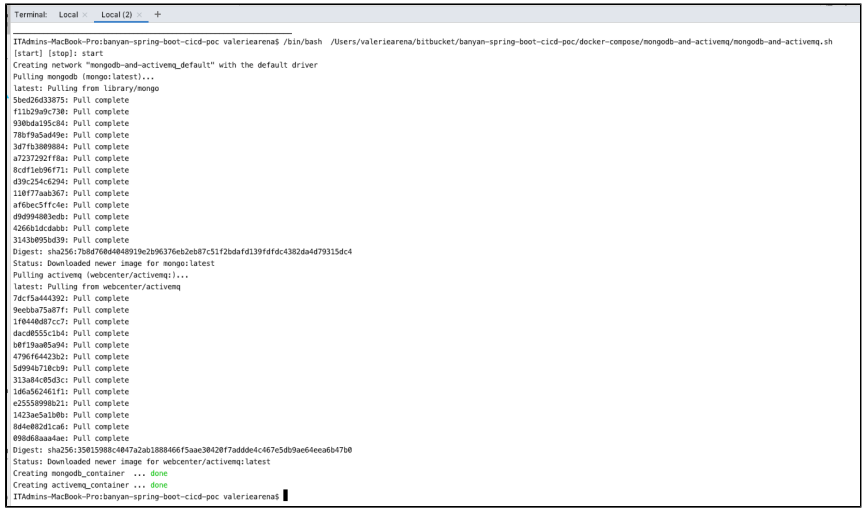
Start MongoDB and ActiveMQ

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

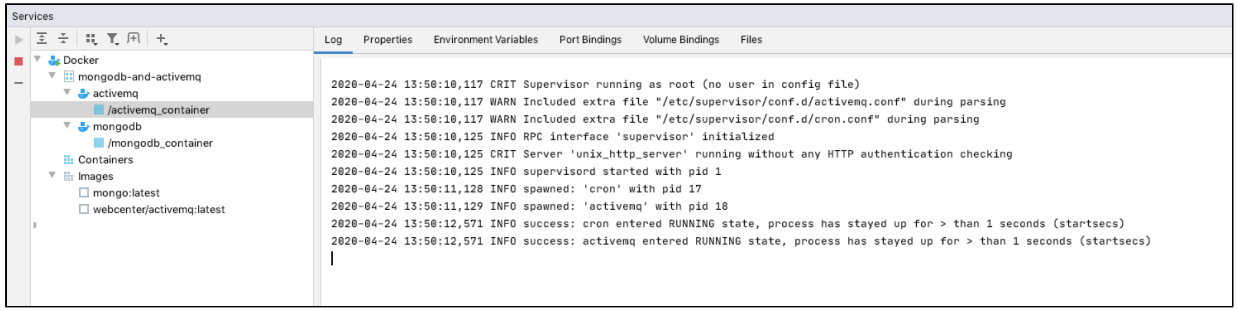
1. Select the modules run config and run it.
2. In the IntelliJ Terminal (View Tool Windows Terminal), enter 'start' in Terminal:

```
ITAdmins-MacBook-Pro:banyan-spring-boot-cicd-poc valeriearena$ /bin/bash  /Users/valeriearena/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:



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



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:

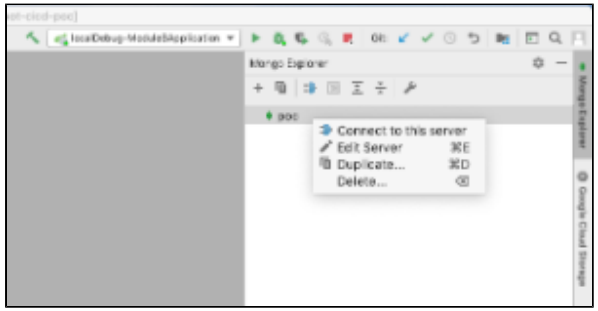
```
docker exec -it mongodb_container bash
mongo
use poc
db.version()
db.messages.count()
show collections
```

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

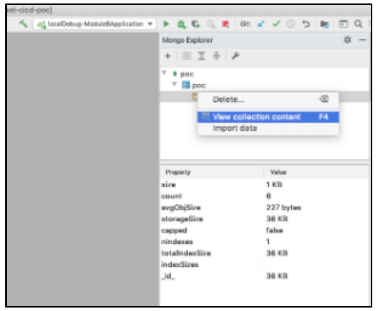
```
> exit
bye
root@b4ff42f3896a:/# exit
exit
ITAdmins-MacBook-Pro:banyan-spring-boot-cicd-poc valeriearena$
```

6. To connect to MongoDB via the IntelliJ Mongo plugin:

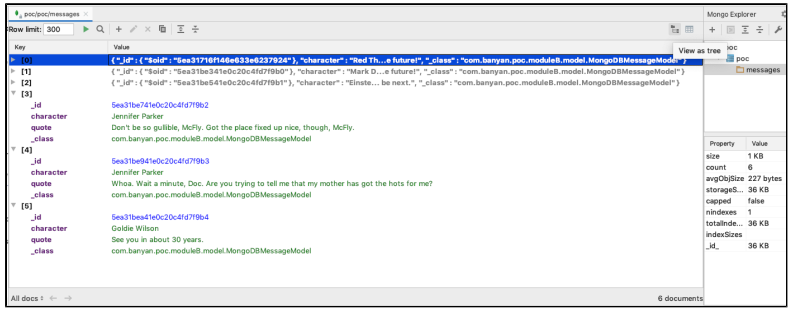
- Open Mongo Explorer and connect to the database:

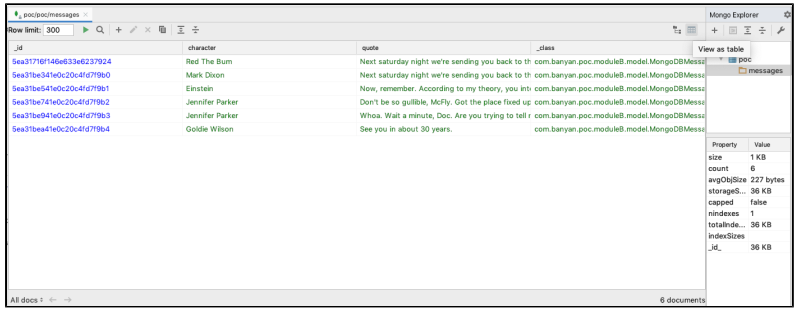


- View stats and documents:

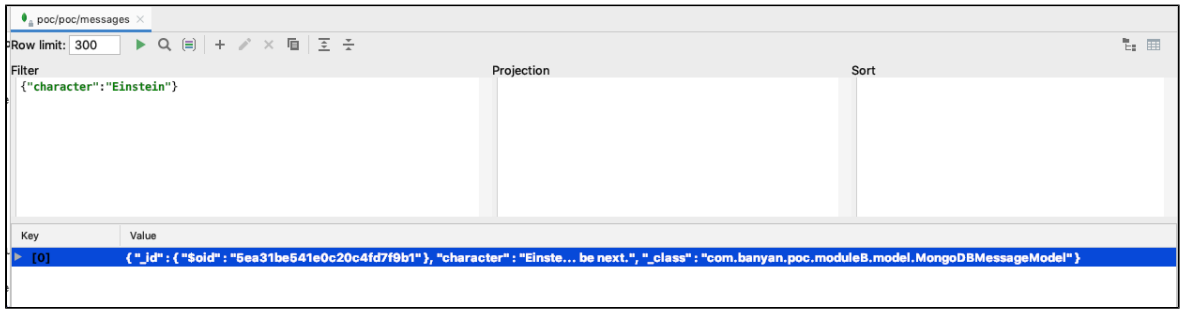


- View the document in a tree view or a table view:





- 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

1. Execute the Spring Boot debug config for moduleA.
2. 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.

1. 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.
2. In a third terminal, execute the cURL.

/name/store

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

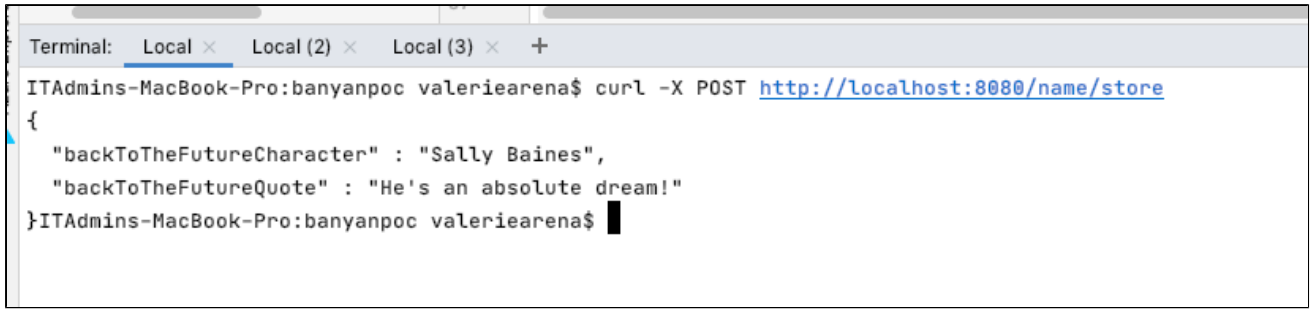
- moduleA makes a REST call to the /message endpoint in moduleB.
- moduleB creates a Back To the Future message and saves it as a MongoDB document.
- moduleB returns the Back To the Future message to moduleA.
- moduleA returns the Back To the Future message in the response.

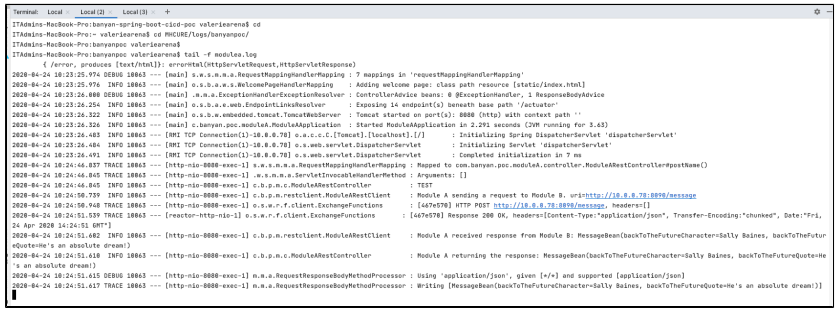
1. Execute the following cURL.

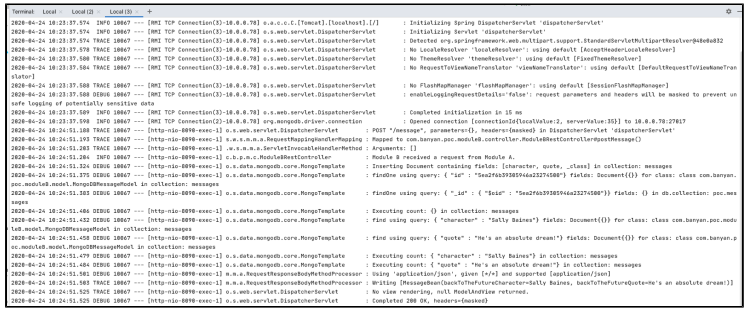
cURL

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

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







/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.

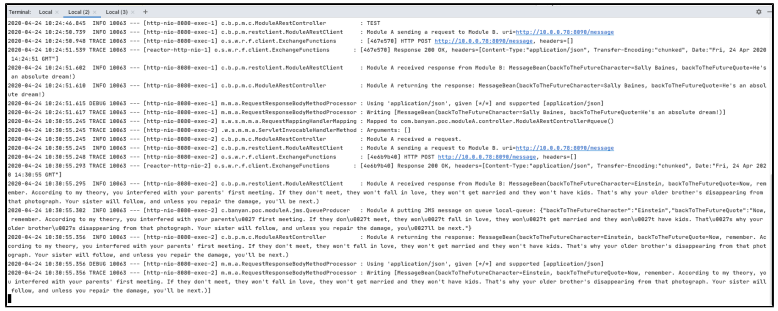
1. Execute the following cURL:

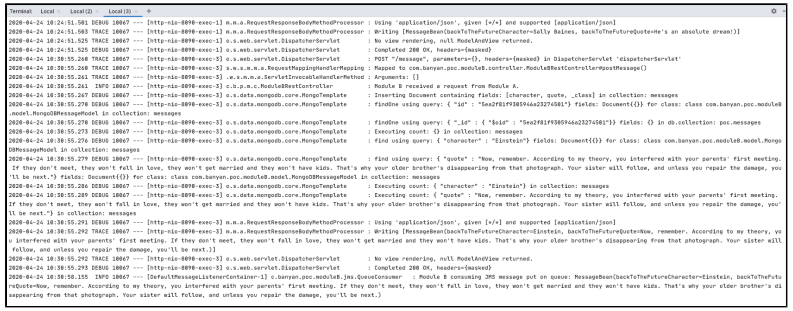
cURL

```
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 valeriearena$ 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 valeriearena$
```





/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.

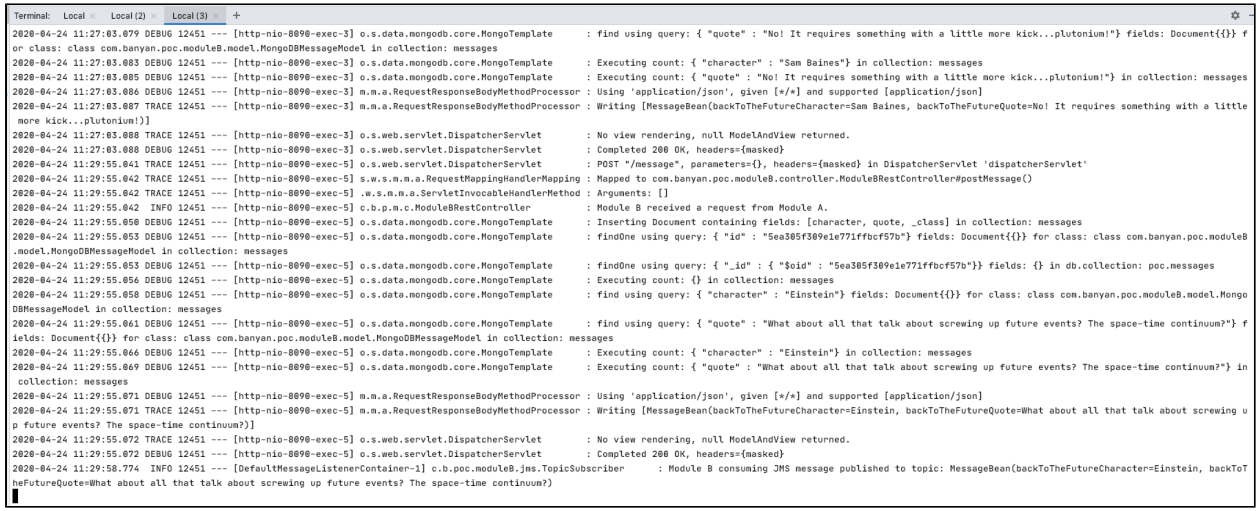
1. Execute the following cURL:

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

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

```
Terminal:  Local  Local (2)  Local (3)  +
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  Local (2)  Local (3)  +
ITAdmins-MacBook-Pro:~ valeriearena$ cd MHCURE/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.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.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.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.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       : [4481d08b] 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       : [4481d08b] 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 Local (2) 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{{}} for class: class com.banyan.poc.moduleB.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.RequestResponseBodyMethodProcessor : 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.RequestResponseBodyMethodProcessor : 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.moduleB.controller.ModuleBRestController#postMessage()
2020-04-24 11:29:55.042 TRACE 12451 --- [http-nio-8090-exec-5] .w.s.m.m.a.ServletInvocableHandlerMethod : Arguments: []
2020-04-24 11:29:55.042 INFO 12451 --- [http-nio-8090-exec-5] c.b.p.m.c.ModuleBRestController : Module B 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" : "5ea305f309e1e771ffbcf57b"} fields: Document{{}} for class: class com.banyan.poc.moduleB.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" : "5ea305f309e1e771ffbcf57b"}} fields: {} in db.collection: poc.messages
2020-04-24 11:29:55.056 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.moduleB.model.MongoDBMessageModel 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?"} fields: Document{{}} for class: class com.banyan.poc.moduleB.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.RequestResponseBodyMethodProcessor : 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.RequestResponseBodyMethodProcessor : Writing [MessageBean(backToTheFutureCharacter=Einstein, backToTheFutureQuote=What about all that talk about screwing up 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.moduleB.jms.TopicSubscriber : Module B consuming JMS message published to topic: MessageBean(backToTheFutureCharacter=Einstein, backToTheFutureQuote=What about all that talk about screwing up future events? The space-time continuum?)