**CA Technologies**

**ISO service**

|  |  |
| --- | --- |
| Document Version | 5.1 |
| Development Team | Upadhyay, Raghvendra  Suggula, Himabindu  Bose, Sandip |

New Central Service, which will act as a proxy between the callouts & ISO client backend

08

**Fall**

Table of Contents

[Present Design/ Issue Statement 3](#_Toc433376644)

[Approach 3](#_Toc433376645)

[Business Logic 3](#_Toc433376646)

[Extra Features 3](#_Toc433376647)

[Assumptions for Proposed Approach 3](#_Toc433376648)

[Schematic Representation 4](#_Toc433376649)

[Operational Details 5](#_Toc433376650)

[Server Requirements 5](#_Toc433376651)

[Configuration Details 5](#_Toc433376652)

[Sample JSON File 6](#_Toc433376653)

[User Interface 7](#_Toc433376654)

[Log Details 7](#_Toc433376655)

[ISOSysLog 8](#_Toc433376656)

[Individual Server Logs 9](#_Toc433376657)

[CVS/Build Details 14](#_Toc433376658)

[CVS Source Code location [Head Checkout] 14](#_Toc433376659)

[Build Information 14](#_Toc433376660)

[How to Build 14](#_Toc433376661)

[Changes to be done before a build after a code change 16](#_Toc433376662)

[Artifacts Generated 18](#_Toc433376663)

# Present Design/ Issue Statement

1. Centralized ISO server that processes the request response to a given Backend inside a synchronized block.
2. Issue is we have a single thread, which does write, followed by a read on the stream in a single call inside a synchronized block.
3. Given that it’s a synchronized block, processing is sequential.
4. Inside the synchronized block given that both the write & read are blocking calls; the time taken to complete execution (in case of a slow server) will have repercussions on the consecutive requests.

# Approach

## Business Logic

1. Separation of writes & read operation {thereby achieving parallel processing} for a given socket via two independent background threads; i.e. MessageSenderThread that keeps writing the request messages to the sockets stream; and MessageReceiverThread that keeps reading the responses from the sockets stream.
2. En queuing the Requests to be processed [ISOReqMessageStore] & responses received [ISORespMessageStore] from the backend in different queues.
3. ISOReqMessageStore to be populated as soon as the Request reaches the ISO Server. This will be done via the tomcat thread.
4. MessageSenderThread will keep looking up ISOReqMessageStore for messages and consequently write the message to the sockets stream.
5. MessageReceiverThread will populate ISORespMessageStore on receiving a message on the stream from the backend.
6. The tomcat thread 2(a) responsible for populating the ISOReqMessageStore will do a lookup on the ISORespMessageStore via a unique ID to identify the response to the corresponding request. The lookup will be done for a configurable amount of time; after which No Response received will be declared for the given request.
7. Each Message to be put in the queues will have the following data structure:

|  |  |
| --- | --- |
| Elements | Description |
| MessageContent | Entire message |
| MessageUID | Unique ID of the message {this will be used to match a request against a response}. |

## Extra Features

1. Provisioning reconnect (refresh/restart the socket connection for an individual server), stop, update and add new backend server without restarting tomcat server.
2. ISO service level & individual server level logging. TRACE supported at individual server level to enable print the request/responses in the Logs.

# Assumptions for Proposed Approach

1. Each message should have a unique ID. Will be handled at the ISO infrastructure level.

* STAN will be used as a unique ID for Network related messages (SIGN ON, ECHO, SIGOFF). *This would be a 6-digit number.*
* STAN<PAN> will be used as a unique ID for ISO 0200/0210 messages. *This would be a 22-digit number.*

1. Second byte of each individual message denotes the length of the message.
2. Caters only to ISO 8583:1987 Version messaging.

# Schematic Representation

Get Response

lookup(mUID1)

lookup(mUID2)

Client 1

Send Request

MessageSenderThread

MessageRecieverThread

Add(,mRes2,mUID2)

Add(m2,mUID2)

ISORespMessageStore

Add(m1,mUID1)

Client 2

BACKEND SERVER

read()

Add(,mRes1,mUID1)

Poll()

write()

ISOReqMessageStore

# Operational Details

## Server Requirements

|  |  |
| --- | --- |
| Supported Platform | RHEL 6.x (64 Bit) |
| JRE Version | Java 7 |
| Web Server | Apache tomcat 7.0.63 |

## Configuration Details

1. To be deployed in the form of a WAR.
2. Infrastructure related configuration & backend server configuration would be read from a JSON file (atmswitchconfig.json) present in the class path.

*How to set it to classpath:*

*1. Create a folder (say* ***caConfFile****) under tomcat home directory (<tomcat\_home>)*

*2. Place* atmswitchconfig.json *file under above directory.*

*3. Append* ***catalina.properties****’s common.loader entry with “****,${catalina.home}/caConfFile****”.*

|  |  |  |  |
| --- | --- | --- | --- |
| **Label** | **Value** | **Description** | **Mandatory/ Default Value if Entry NOT FOUND** |
| ***Global Level Settings*** | | |  |
| logFileLocation | /opt/arcot/logs/ISO | Location of the Directory where ISO infrastructure level logs is to be generated. | **Mandatory. Has to be filled in.**  **Please create the directory if it does not exist.** |
| ***Server Level Settings*** | | |  |
| host |  | Host IP of a given ISO backend server. | **Mandatory. Has to be filled in.** |
| port |  | Port of a given ISO backend server. | **Mandatory. Has to be filled in.** |
| connTimeOut |  | Connection Timeout in milliseconds for the given server. | 8000 |
| readTimeOut |  | Read Timeout in milliseconds for the given server. | 8000 |
| threadTimeOut |  | Time in millisecond for lookup on the ISORespMessageStore before declaring that Response was not received for a given request. | 5000 |
| retry |  | Number of tries to be made for a successful connection. | 3 |
| logLevel |  | Log level for a given server. | INFO |
| echoTimeInterval |  | Time interval in seconds between ECHO messages being replayed to a given server.  **THIS ENTRY NEEDS TO BE PRESENT ONLY FOR SERVERS WHICH SUPPORT ECHO.** | **0 🡺 No ECHO Supported.** |

### Sample JSON File

{

"serverConfig": [

{

"host": "localhost",

"port": 8083,

"connTimeOut": 8000,

"readTimeOut": 8000,

"threadTimeOut": 5000,

"retry": 2,

"logLevel": "INFO",

"echoTimeInterval": 1,

"uidFormat":"11,2"

},

{

"host": "localhost",

"port": 7500,

“logLevel” set to “TRACE”.

IMPLIES

Localhost\_7500 log will print request/response

"connTimeOut": 8000,

"readTimeOut": 8000,

"threadTimeOut": 5000,

"retry": 2,

"logLevel": "TRACE",

"echoTimeInterval": 1,

"uidFormat":"11,2"

},

{

"host": "localhost",

“echoTimeInterval” missing.

IMPLIES

This Server Does NOT support ECHO.

"port": 9530,

"connTimeOut": 8000,

"readTimeOut": 8000,

"threadTimeOut": 5000,

"retry": 2,

"logLevel": "INFO",

"uidFormat":"11,2"

}

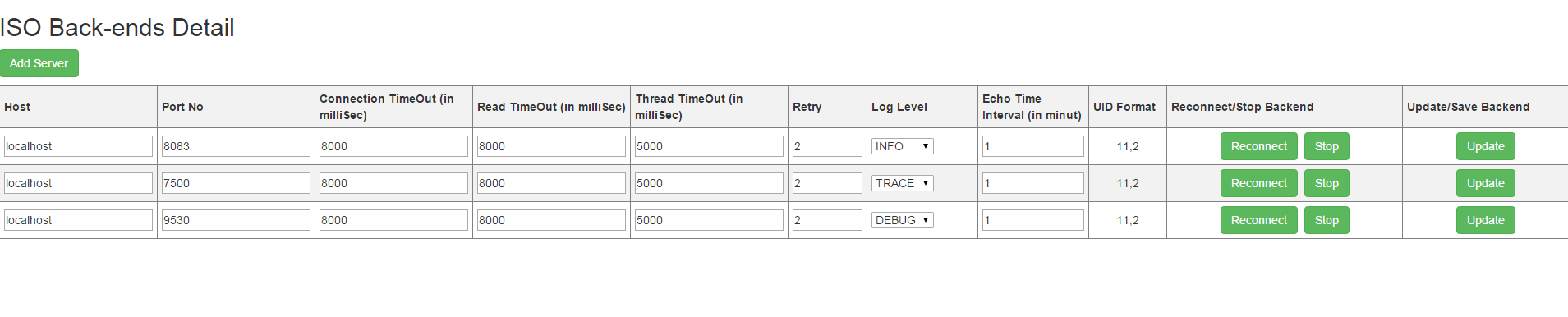
],

"logFileLocation": "/opt/arcot/logs/ISO"

}

### User Interface

Alternatively Infrastructure related configuration & backend server configuration could be managed from a User Interface.



A User interface is built to give following controls to the user when he logged in locally using localhost in the URL (for example: [http://localhost:8080/acyncMessageProcesor-1.0.0/ ISOServerConfiguration.htm](http://localhost:8080/acyncMessageProcesor-1.0.0/%20ISOServerConfiguration.htm) ; assuming acyncMessageProcesor-1.0.0 is the web application name) otherwise the whole page will be read only.

1. **Reconnect**: Provisioning reconnect (refresh/restart the socket connection for an individual server) to individual server without restarting tomcat server.
2. **Update**: An update can be made to a backend server to change its parameters or configurations. Reconnect operation must be done after update action to reflect changes.
3. **Stop**: This is force stop operation and this should be done only when we don’t want the particular server to run or to exclude a particular server from list of servers running on tomcat.
4. **Add Server**: Add server action should be done to add new backend server. After saving this server, reconnect operation needs to be done to establish connection to that server.

## Log Details

The following logs will be created for the ISO Service under the directory provided against “logFileLocation” in JSON configuration file. All the logs will be rolled over on a daily basis.

|  |  |  |
| --- | --- | --- |
| **Log Name** | **Description** | **LOG Level Supported** |
| ISOSysLog | ISO Service Level Log. This will log all the request/responses (will be denoted via there unique ID’s) across all the servers; along with the round robin time (time between request send out from the ISO service & response received from the backend server in Seconds). | INFO Only. Cannot be customized to have any other log level. |
| <host>\_<port> | Log for each ISO backend; <host> is the IP of the Backend Server & <port> is the port of the Backend Server. | INFO DEBUG  TRACE |

### ISOSysLog

#### Log description

1. **Initialization log lines** when web server containing ISO war is restarted. Assuming that only ISO backend server configured in the JSON file is for HOST=localhost, Port=7500.

* 2015-09-22 12:14:01,416 | INFO | [localhost-startStop-1:ISOSysLog] | Initializing ATM Switch Configuration....
* 2015-09-28 19:46:26,113 | INFO | [localhost-startStop-1:ISOSysLog] | **Build version 1.0.0-SNAPSHOT Build Date 20150928-1928**
* 2015-09-22 12:14:01,966 | INFO | [localhost-startStop-1:ISOSysLog] | [ (localhost\_7500) : localhost,7500,8000,8000,2,1] Loaded....

|  |  |
| --- | --- |
| **Contents** | **Description** |
| (localhost\_7500) : localhost,7500,8000,8000,2,1] Loaded.... | Connection between the ISO service & backend with host=localhost & port 7500 is established.  Also prints the properties set against this host port combination in JSON file. |

1. **Log line for a network message** for SERVER IP=localhost, Port=7500.

* 2015-09-22 10:53:46,340 | INFO | [Thread-5:ISOSysLog] | ( localhost\_7500 ,getResponse(, ,) ) Round Robin Time [ 0 ] Req [ 915258 ] Res [ 915258 ]

|  |  |
| --- | --- |
| **Contents** | **Description** |
| This is a network message. Identified via the 6-digit Unique ID. | |
| localhost\_7500 | For the server with Server IP = localhost, Port = 7500 |
| Round Robin Time [ 0 ] | Time taken for the request/response cycle between the ISO Service & backend server in Seconds |
| Req [ 915258 ] Res [ 915258 ] | Request/ response for Unique message ID = 915258 |

1. **Log line for an ISO 0200/0210 message** for SERVER IP=localhost, Port=7500.

* 2015-09-22 10:54:06,857 | INFO | [http-bio-8080-exec-1:ISOSysLog] | ( localhost\_7500 ,getResponse(, ,) ) Round Robin Time [ 0 ] Req [ 333210XXXXXXXXXXXXX3456 ] Res [ 333210XXXXXXXXXXXXX3456 ]

|  |  |
| --- | --- |
| **Contents** | **Description** |
| This is an ISO 0200/0210 message. Identified via the 22-digit Unique ID. | |
| localhost\_7500 | For the server with Server IP = localhost, Port = 7500 |
| Round Robin Time [ 0 ] | Time taken for the request/response cycle between the ISO Service & backend server in Seconds |
| Req [ 333210XXXXXXXXXXXXX3456 ] Res [ 333210XXXXXXXXXXXXX3456 ] | Request/ response for Unique message ID = 333210XXXXXXXXXXXXX3456.  NOTE: Unique ID given that it’s a ISO 0200/0210 message is STAN<PAN>.  Here expect the last 4 digits of the PAN all the other digits are masked. |

1. Log line wherein the response did not come back in a configured amount of time.

* 2015-09-22 11:00:57,501 | **WARN** | [localhost-startStop-2:ISOSysLog] | (localhost\_7500 ,getResponse(, ,) ) Round Robin Time [ 5 ] Req [ 379958 ] Res [ **NOTRCVD** ]

|  |  |
| --- | --- |
| **Contents** | **Description** |
| Line wherein the Response did not come back within the read timeout parameter set for the given server. | |
| localhost\_7500 | For the server with Server IP = localhost, Port = 7500. |
| Round Robin Time [ 5 ] | Assuming that the threadTimeOut against this host, port in JSON file is set to 5 seconds. |
| Req [ 379958 ] Res [ **NOTRCVD** ] | For Request with unique ID = 379958, we did not get the response back in 5 seconds; as a result of whih a NULL response is sent back. |

1. **Shutdown log lines** when webserver containing ISO war is shut down.

|  |  |
| --- | --- |
| **Contents** | **Description** |
| 2015-09-24 12:25:37,136 | WARN | [localhost-startStop-2:ISOSysLog] | ATM Switch Interface Shutting Down.... | Self Explainatory |
| 2015-09-24 12:25:37,137 | INFO | [localhost-startStop-2:ISOSysLog] | ( localhost\_7500 ,getResponse(, ,) ) Request message UID : 388046  2015-09-24 12:25:37,637 | INFO | [localhost-startStop-2:ISOSysLog] | ( localhost\_7500 ,getResponse(, ,) ) Round Robin Time [ 0 ] Req [ 388046 ] Res [ 388046 ] | Sign off (with message Unique ID as 388046) send & received for the given host & port. |
| 2015-09-24 12:25:37,638 | INFO | [localhost-startStop-2:ISOSysLog] | localhost\_7500 is - Closed. | Resources cleaned up against given host & port, and the socket is closed. |

### Individual Server Logs

1. Log name nomenclature followed is <host>\_<port>. Here <host>, <port> corresponds to the bank backend (or switch host & port).
2. Following log levels are supported: INFO, DEBUG, TRACE.

|  |  |
| --- | --- |
| **Log Level** | **Description** |
| INFO | Will log the length of data written on the socket stream along with the message Unique ID, length of data read from the socket stream along with the message Unique ID. |
| DEBUG | Will log the details (Store Size, along with the message being added or removed from the stores depicted via there Unique Id’s) pertaining to the Message Stores (both request & response). |
| TRACE | Will log the entire message along with its corresponding Unique ID. |

#### Log description

1. **Initialization log lines when web server containing ISO war is restarted**; and assuming we have an ISO server running with Host= localhost and Port= 7500 with ECHO enabled. Socket will be created, the background Sender & Receiver thread will be created, Sign On message will be sent out.

* 2015-09-22 19:07:14,894 | INFO | [localhost-startStop-1:localhost\_7500] | ( connect() ) Connection is established....
* 2015-09-22 19:07:14,896 | INFO | [Sender-Thread-3:localhost\_7500] | started MessageSenderThread....
* 2015-09-22 19:07:14,897 | INFO | [Receiver-Thread-4:localhost\_7500] | started MessageReceiverThread....
* 2015-09-22 19:07:14,930 | INFO | [localhost-startStop-1:localhost\_7500] | Attempting ( handleNetworkManagementMsg() ) SIGNON ( 275732 )
* 2015-09-22 19:07:14,934 | INFO | [Sender-Thread-3:localhost\_7500] | writeData( 57 , 275732 ) Data To be Sent
* 2015-09-22 19:07:14,959 | INFO | [Receiver-Thread-4:localhost\_7500] | receiveData( 57 ) Message Rcvd
* 2015-09-22 19:07:14,959 | INFO | [Receiver-Thread-4:localhost\_7500] | receiveDataContent( 57 , 275732 ) Data To be Sent
* 2015-09-22 19:07:15,435 | INFO | [localhost-startStop-1:localhost\_7500] | Successful ( handleNetworkManagementMsg() ) SIGNON ( 275732 )
* 2015-09-22 19:07:15,435 | INFO | [localhost-startStop-1:localhost\_7500] | ( startEchoHandler() ) Sign On Successful....

1. **Log line for a network message** **& ISO 0200/0210 messages**.

* **Log Level Set to INFO**. Assuming that the message is generated with STAN=204671 and for card number with last 4-digits as 3456= XXXXXXXXXXXXX3456

|  |  |
| --- | --- |
| **Log Line** | **Description** |
| 2015-09-22 19:07:47,517 | INFO | [Sender-Thread-3:localhost\_7500] | writeData( 203 , 204671XXXXXXXXXXXXX3456 ) Data To be Sent | Write data to the socket’s output stream. Length of the message is 203, and the unique ID for this message is 204671XXXXXXXXXXXXX3456. |
| 2015-09-22 19:07:47,519 | INFO | [Receiver-Thread-4:localhost\_7500] | receiveData( 187 ) Message Rcvd | Read data of length 187 from the socket’s input stream. |
| 2015-09-22 19:07:47,519 | INFO | [Receiver-Thread-4:localhost\_7500] | receiveDataContent( 187 , 204671XXXXXXXXXXXXX3456 ) Data To be Sent | Data read above contains message with Unique ID 204671XXXXXXXXXXXXX3456 |

* **Log Level Set to DEBUG**. Assuming that the message is generated with STAN=178889 and for card number with last 4-digits as 3456= XXXXXXXXXXXXX3456

|  |  |  |
| --- | --- | --- |
|  | **Log Line** | **Description** |
| 1 | 2015-09-22 19:27:35,180 | DEBUG | [**http-bio-8080-exec-1**:localhost\_7500] | ( reqStore, storeMessage( 178889XXXXXXXXXXXXX3456 ) ) Store Size [ 1 ] | Add message(with unique ID 178889XXXXXXXXXXXXX3456) to the ISOReqMessageStore. The **web server thread** handling the request will perform this action.  At this point of time the ISOReqMessageStore store just has 1 message. |
| 2 | 2015-09-22 19:27:35,180 | DEBUG | [Sender-Thread-3:localhost\_7500] | ( reqStore, retreiveMessage(178889XXXXXXXXXXXXX3456) ) Store Size [ 0 ] | Retrieve the message from the ISOReqMessageStore to be written to sockets output stream.  This MessageSenderThread background thread will do this action.  At this point of time the ISOReqMessageStore store just has 0 messages. |
| 3 | 2015-09-22 19:27:35,182 | DEBUG | [Receiver-Thread-4:localhost\_7500] | ( respStore, storeMessage(178889XXXXXXXXXXXXX3456) ) Store Size [ 1 ] | Response to message with unique ID 178889XXXXXXXXXXXXX3456 has been received by MessageReceiverThread which in turn stores it in ISORespMessageStore.  At this point of time the ISORespMessageStore has only 1 message. |
| 4 | 2015-09-22 19:27:35,685 | DEBUG | [**http-bio-8080-exec-1**:localhost\_7500] | ( respStore, retreiveMessage(178889XXXXXXXXXXXXX3456) ) Store Size [ 0 ] | The same **web server thread** as in section 1 of this table will lookup the ISORespMessageStore and retrieve the response & send it back to the JSP/callout. |

* **Log Level Set to TRACE**. Assuming that the message is generated with STAN=178889 and for card number with last 4-digits as 3456= XXXXXXXXXXXXX3456. *Everything remains the same as with DEBUG level except that now the messages will be printed in the logs.*

|  |  |  |
| --- | --- | --- |
|  | **Log Line** | **Description** |
| 1 | 2015-09-22 19:27:35,180 | TRACE | [**http-bio-8080-exec-1**:localhost\_7500] | ( reqStore, storeMessage( 178889XXXXXXXXXXXXX3456 ) ) Store Size [ 1 ] Message [ 0200623C000128E090001645445678901234563000000922192735178889192735092215100800000005214544567890123456=1510144293025517ARCOT100000000000000000ARCOT SYSTEMS INCORPORATED BANGALORE IND3561231211112312111 ] | Add message(with unique ID 178889XXXXXXXXXXXXX3456) to the ISOReqMessageStore. The **web server thread** handling the request will perform this action.  At this point of time the ISOReqMessageStore store just has 1 message. |
| 2 | 2015-09-22 19:27:35,180 | TRACE | [Sender-Thread-3:localhost\_7500] | ( reqStore, retreiveMessage(178889XXXXXXXXXXXXX3456) ) Store Size [ 0 ] Message [ 0200623C000128E090001645445678901234563000000922192735178889192735092215100800000005214544567890123456=1510144293025517ARCOT100000000000000000ARCOT SYSTEMS INCORPORATED BANGALORE IND3561231211112312111 ] | Retrieve the message from the ISOReqMessageStore to be written to sockets output stream.  This MessageSenderThread background thread will do this action.  At this point of time the ISOReqMessageStore store just has 0 messages. |
| 3 | 2015-09-22 19:27:35,182 | TRACE | [Receiver-Thread-4:localhost\_7500] | ( respStore, storeMessage(178889XXXXXXXXXXXXX3456) ) Store Size [ 1 ] Message [ 0210623C00012AE080001645445678901234563000000922192735178889192735092215100800000005214544567890123456=151014429302551700ARCOT100000000000000000ARCOT SYSTEMS INCORPORATED BANGALORE IND356 ] | Response to message with unique ID 178889XXXXXXXXXXXXX3456 has been received by MessageReceiverThread which in turn stores it in ISORespMessageStore.  At this point of time the ISORespMessageStore has only 1 message. |
| 4 | 2015-09-22 19:27:35,685 | TRACE | [**http-bio-8080-exec-1**:localhost\_7500] | ( respStore, retreiveMessage(178889XXXXXXXXXXXXX3456) ) Store Size [ 0 ] Message [ 0210623C00012AE080001645445678901234563000000922192735178889192735092215100800000005214544567890123456=151014429302551700ARCOT100000000000000000ARCOT SYSTEMS INCORPORATED BANGALORE IND356 ] | The same **web server thread** as in section 1 of this table will lookup the ISORespMessageStore and retrieve the response & send it back to the JSP/callout. |

1. **Shutdown log lines** when webserver containing ISO war is shut down. Only INFO mode is highlighted. The set of events will be as followed:
2. Sent out SIGN OFF
3. Stop the background threads
4. Close the streams related to the sockets & stop connection.

2015-09-24 12:51:05,943 | INFO | [localhost-startStop-2:localhost\_7500] | Attempting ( handleNetworkManagementMsg() ) SIGNOFF ( 576632 )

2015-09-24 12:51:05,944 | INFO | [Thread-3:localhost\_7500] | writeData( 57 , 576632 ) Data To be Sent

2015-09-24 12:51:05,945 | INFO | [Thread-4:localhost\_7500] | receiveData( 57 ) Message Rcvd

2015-09-24 12:51:06,449 | INFO | [localhost-startStop-2:localhost\_7500] | Successful ( handleNetworkManagementMsg() ) SIGNOFF ( 576632 )

2015-09-24 12:51:06,449 | INFO | [localhost-startStop-2:localhost\_7500] | ( cleanResource() ) Sign off Successful....

2015-09-24 12:51:06,449 | INFO | [localhost-startStop-2:localhost\_7500] | NetworkMessageThread is going to stop....

2015-09-24 12:51:06,449 | INFO | [localhost-startStop-2:localhost\_7500] | MessageSenderThread is going to stop....

2015-09-24 12:51:06,449 | INFO | [localhost-startStop-2:localhost\_7500] | MessageReceiverThread is going to stop....

2015-09-24 12:51:06,449 | WARN | [Thread-5:localhost\_7500] | ( NetworkMessageThread ) InterruptedException. The NetworkMessageThread was Interrupted....

# CVS/Build Details

## CVS Source Code location [Head Checkout]

profsvc/transfort/PSUtils/Java/AsyncMsgProcessor

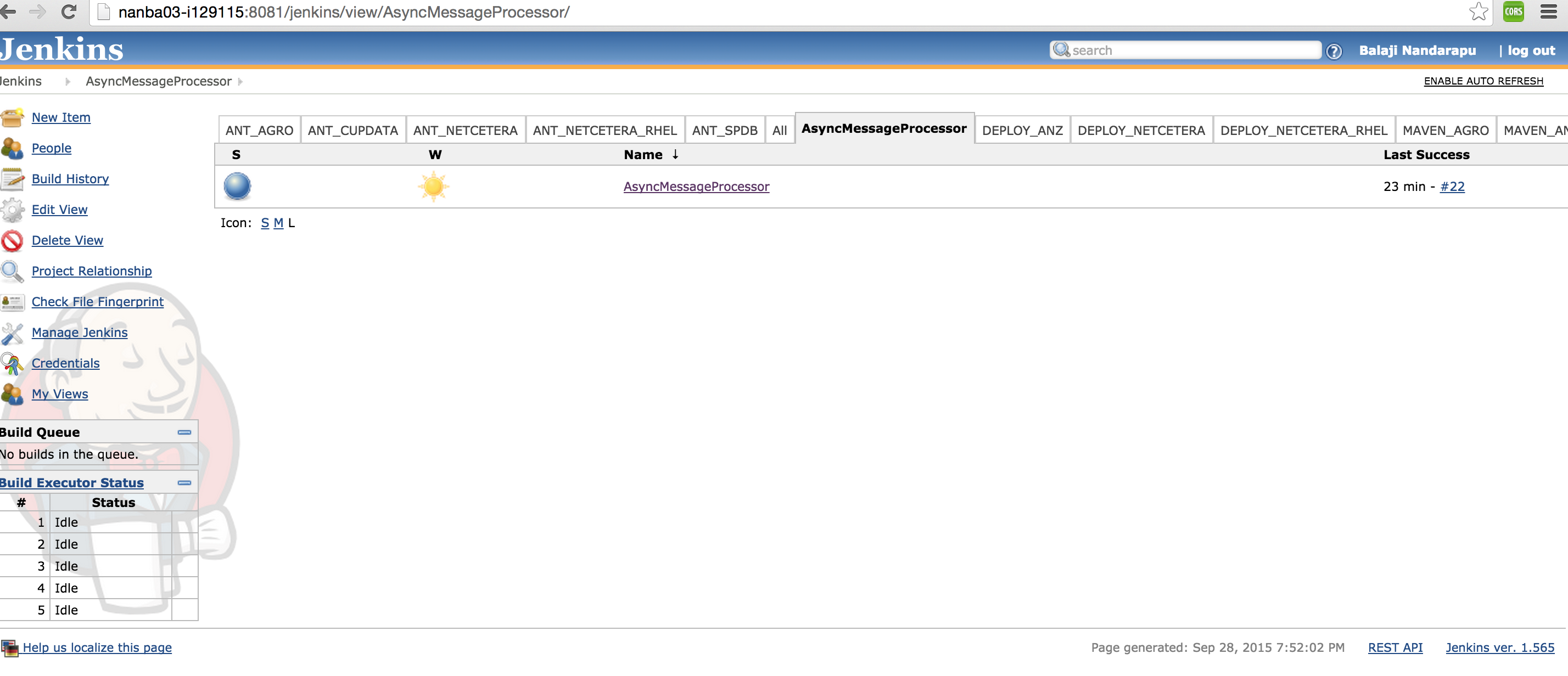
## Build Information

### How to Build

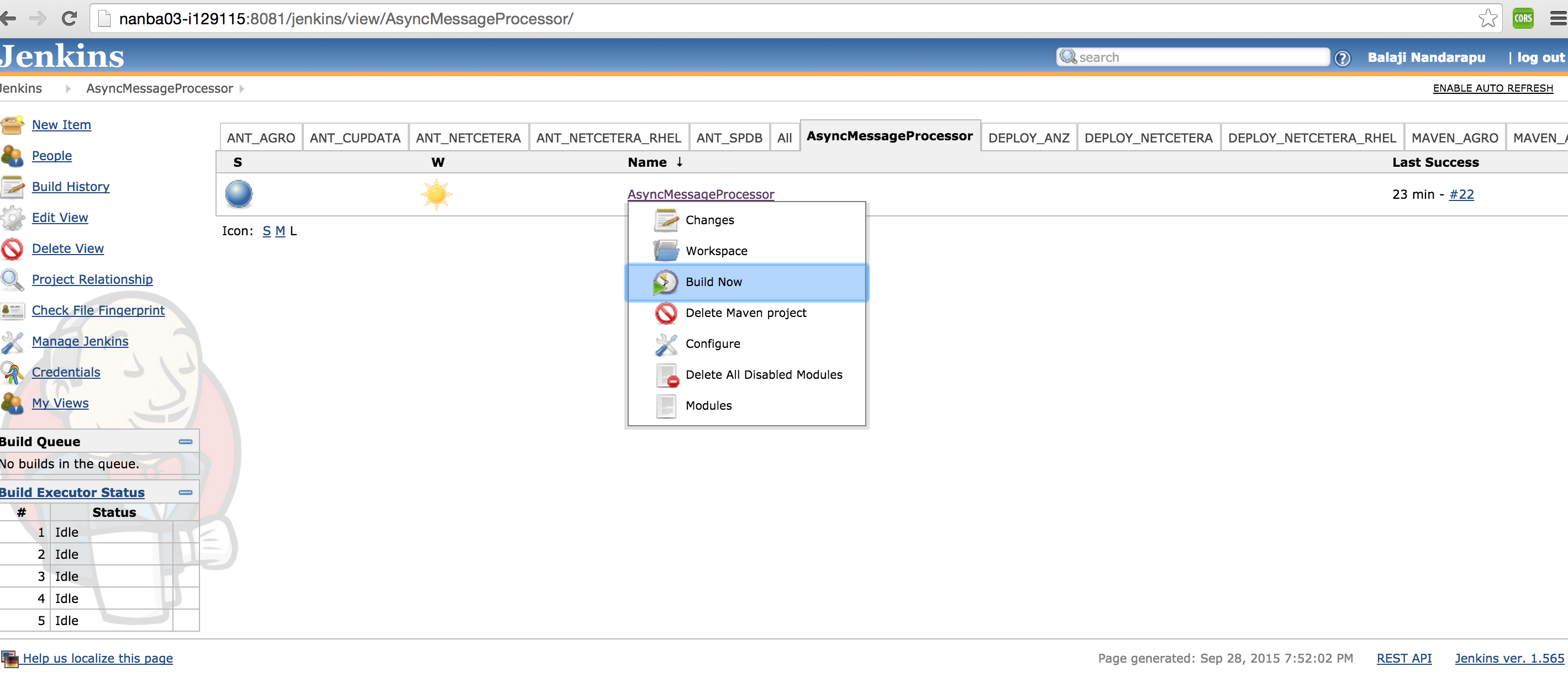
Assuming that the developer has already checked in the code.

1. Login to Jenkins & click on the provided View Name.

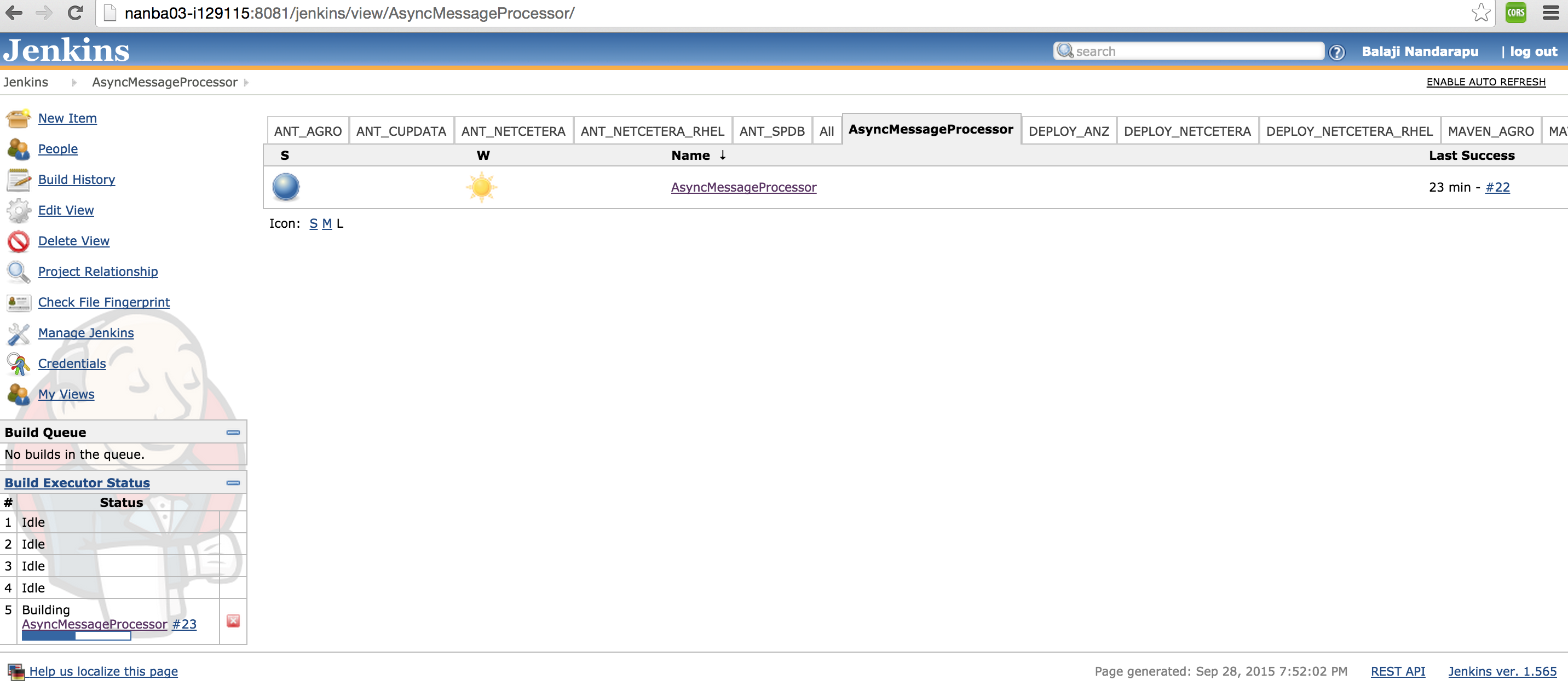
|  |  |
| --- | --- |
| **Information** | **Value** |
| Jenkin’s URL | http://nanba03-i129115:8081/jenkins/login |
| View Name | AsyncMessageProcessor |



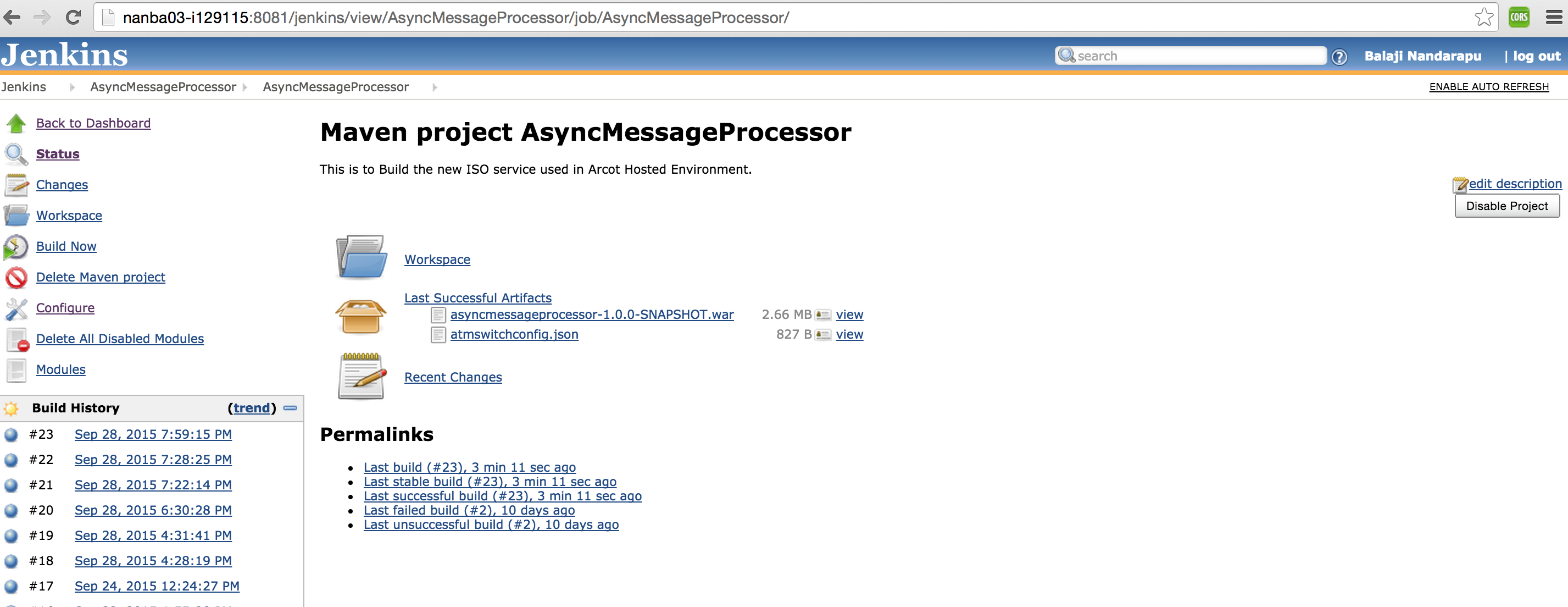
1. Click On Build Now to trigger the Build.



1. While the Build is in progress on the same page you will see the build progress

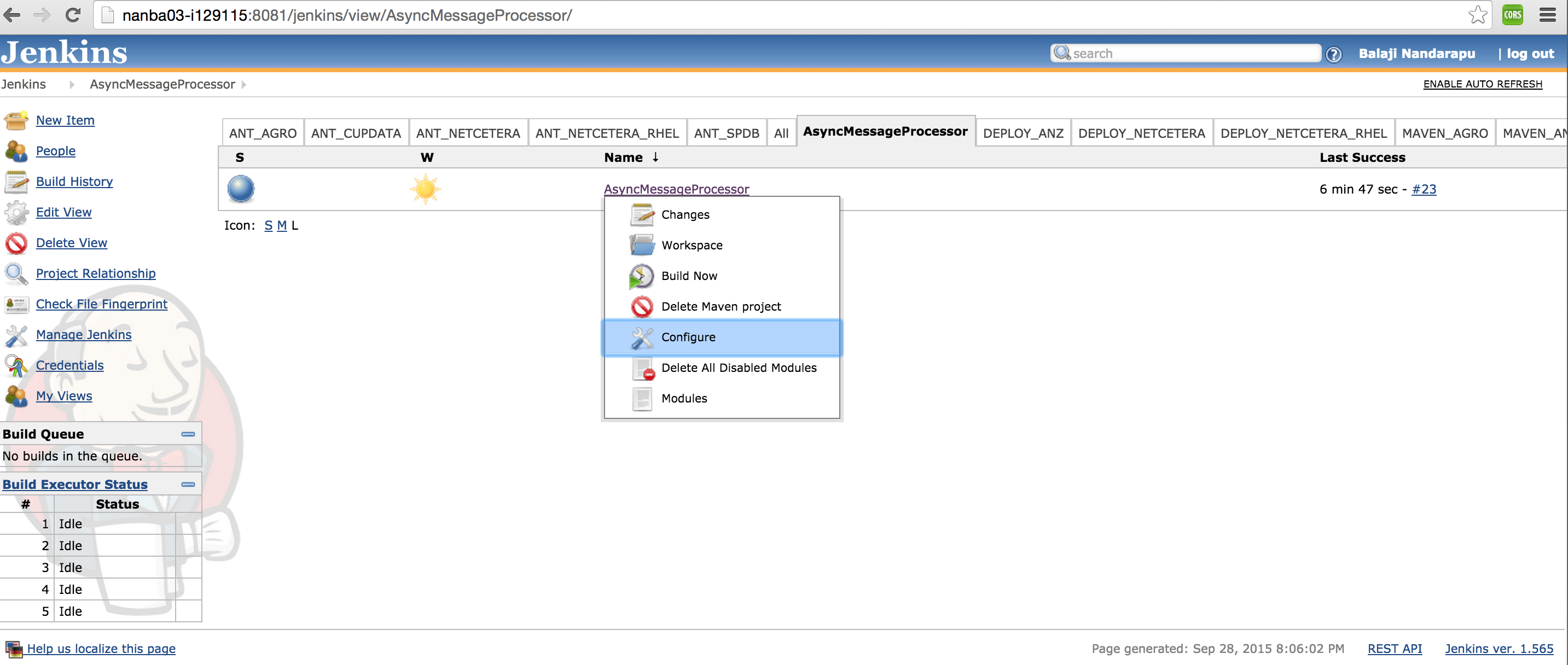


1. Get the artifacts by clicking on the AsyncMessageProcessor



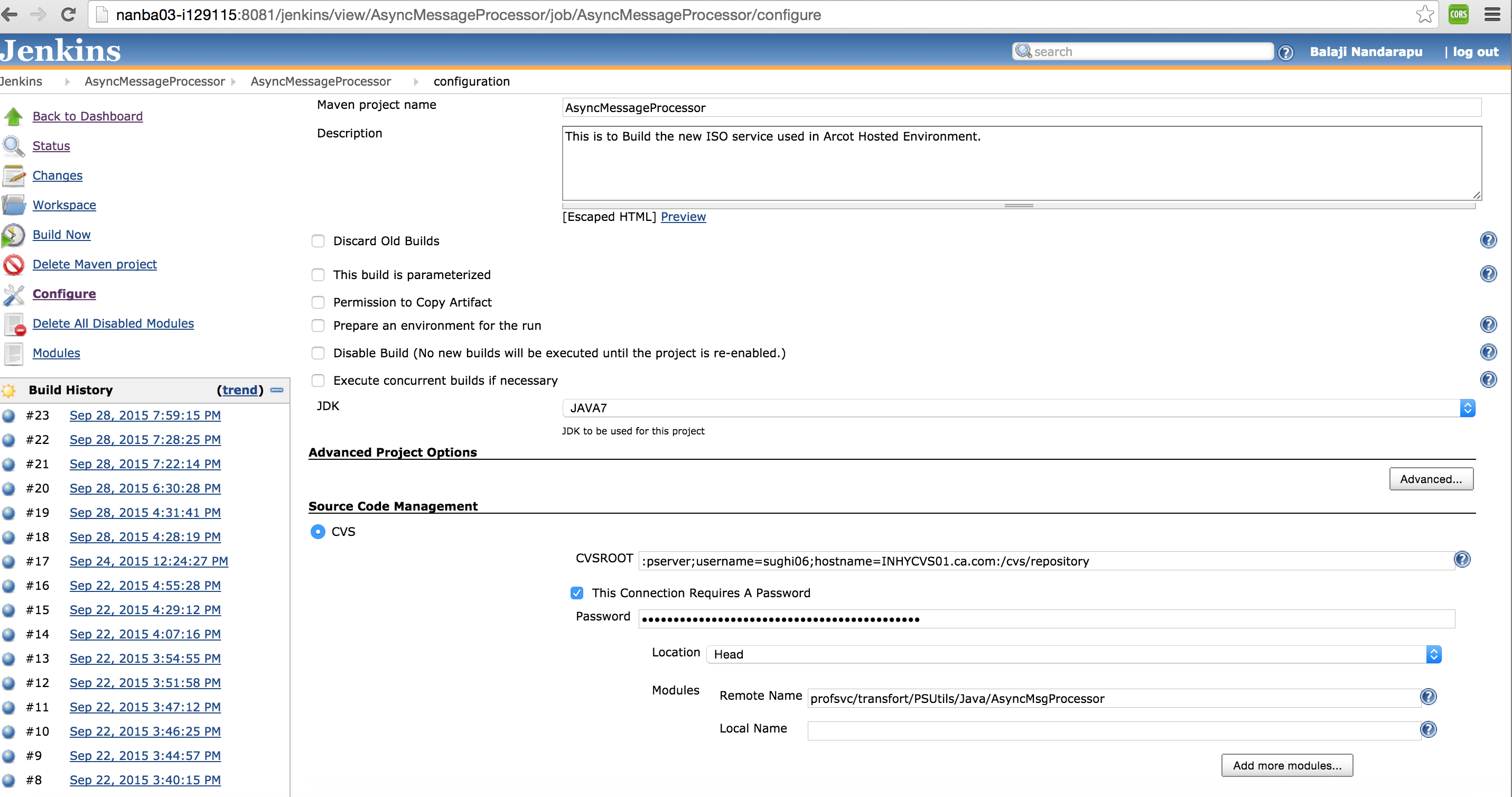
### Changes to be done before a build after a code change

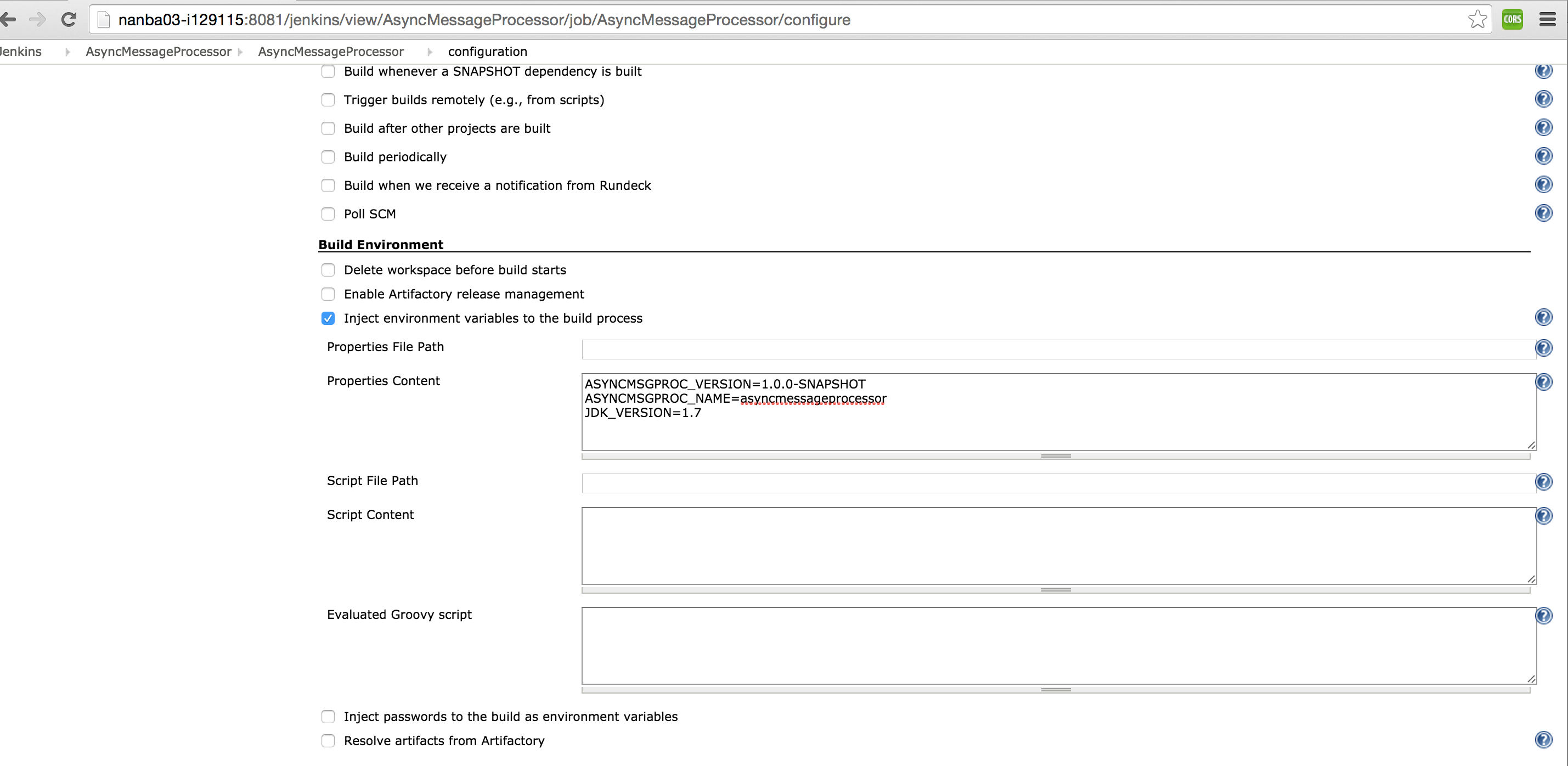
1. Click on Configure



1. Change the following

|  |  |
| --- | --- |
| **Sections** | **Description** |
| Source Code Management/CVS | Change the CVS credentials to the individual doing the build.  CVSROOT & password. |
| Build Environment/Properties Content | These are environment variables being read from the POM   |  |  | | --- | --- | | **Elements** | **Description** | | ASYNCMSGPROC\_VERSION | This version needs to be changed every time a code change is done.  NOTE: Use SNAPSHOT if you are building it for a preview Server. | | ASYNCMSGPROC\_NAME | This is the final name of the WAR file which needs to be deployed | | JDK\_VERSION | JAVA Version. | |





1. Save the changes in the **bottom of the page** & trigger the build.

### Artifacts Generated

|  |  |
| --- | --- |
| **Description** | **Files** |
| Final WAR File | <ASYNCMSGPROC\_NAME>-<ASYNCMSGPROC\_VERSION>.war |
| Configuration file which needs to be present in the classpath | atmswitchconfig.json |