



i_SUT

Malavika K.V

V1.0

22.06.2022



Project: Diagnostic Box Configuration
Rev: 1.0

Sl.no	TOPIC	Pg.No
1	i_SUT work flow on Node-red	3
2	RABBITMQ	6
3	Telegraf	

Project: Diagnostic Box Configuration
Rev: 1.0

What is i-SUT??

- System independent of iGate and Microgrid -To recreate and test the issue in Microgrid
- Building Virtual Plant model and logger with local ,cloud setup in node-red
- Checking API possibilities from different providers
- Building 1st Version Inverter model (sungrow 110cx)
- Exploring transfer protocol AMQP , RabbitMQ

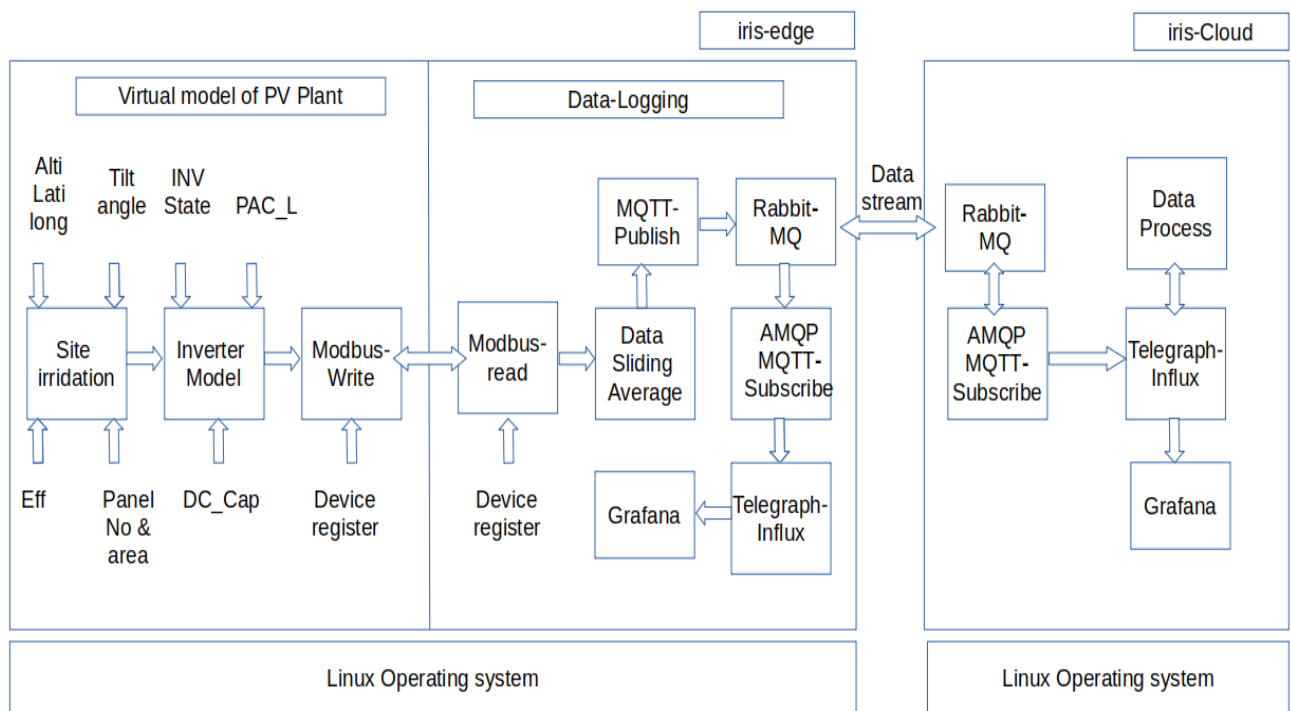
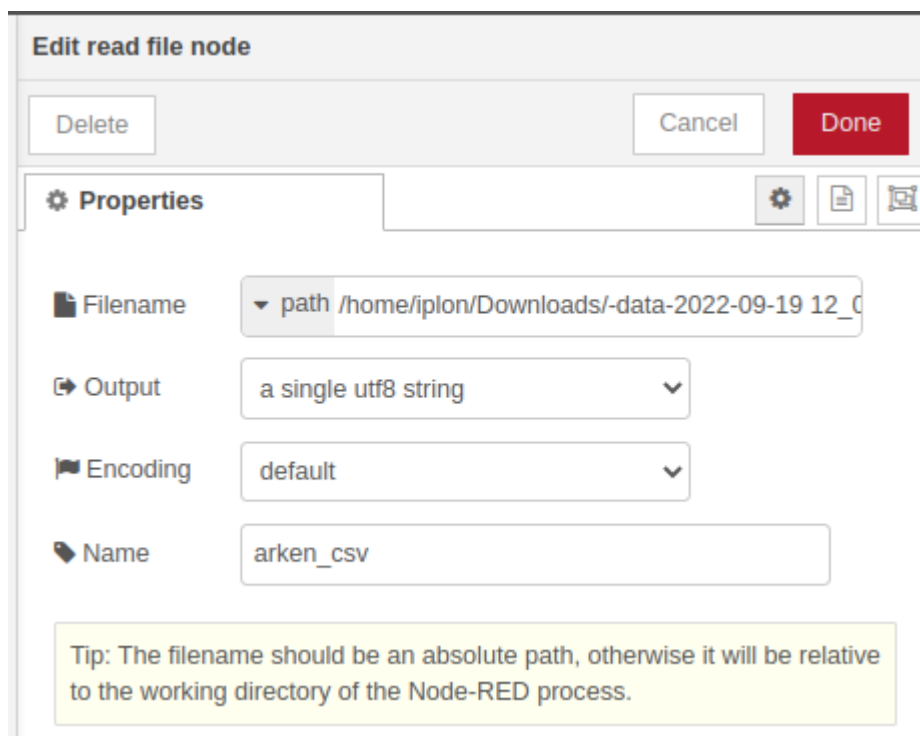


Fig – 1: Architecture of i_SUT

Project: Diagnostic Box Configuration
Rev: 1.0

	Standard	Standard
1	Time	Solar_Radiation (W/M2)
2	2022-09-18 00:00:00	0
3	2022-09-18 00:05:00	0
4	2022-09-18 00:10:00	0
5	2022-09-18 00:15:00	0
6	2022-09-18 00:20:00	0
7	2022-09-18 00:25:00	0
8	2022-09-18 00:30:00	0

➤ Read file node is used here to read this csv file need to fill the filename box with the path of particular csv filename



Edit read file node

Delete Cancel Done

Properties

Filename path /home/iplon/Downloads/-data-2022-09-19 12_0

Output a single utf8 string

Encoding default

Name arken_csv

Tip: The filename should be an absolute path, otherwise it will be relative to the working directory of the Node-RED process.

Fig – 4 read file node edit box

➤ A csv node used here to take out the csv file data into single msg per row format and also output as always sent column headers

Project:
Rev: 1.0

Diagnostic Box Configuration

Delete
Cancel
Done

Properties

Columns
comma-separated column names

Separator
comma

Name
Name

CSV to Object options

Input
Skip first
0
lines
☒ first row contains column names
☒ parse numerical values
☐ include empty strings
☒ include null values

Output
a message per row

Object to CSV options

Output
always send column headers

Newline
Linux (\n)

Fig – 4 csv file node edit box

- A delay node followed by csv node is used to limit each message per 10 second

Project: Diagnostic Box Configuration
Rev: 1.0

Fig – 5 delay node edit box

- So after this output will come as one message per 10 second so these single messages/10 s will go into function node as input inside the function node we wrote a javascript code for inverter simulation (inv_sungrow110CX_model in this flow)
- So by using the solar radiation from the reference csv file we are calculating the PAC,PF,SAC,UAC1,UAC2,UAC3,IAC1,IAC2,IAC3,QAC,UDC(1-12),PDC,IDC(1-12),INTERNAL_TEMPERATURE,FREQUENCY using corresponding equations

Project: Diagnostic Box Configuration
Rev: 1.0

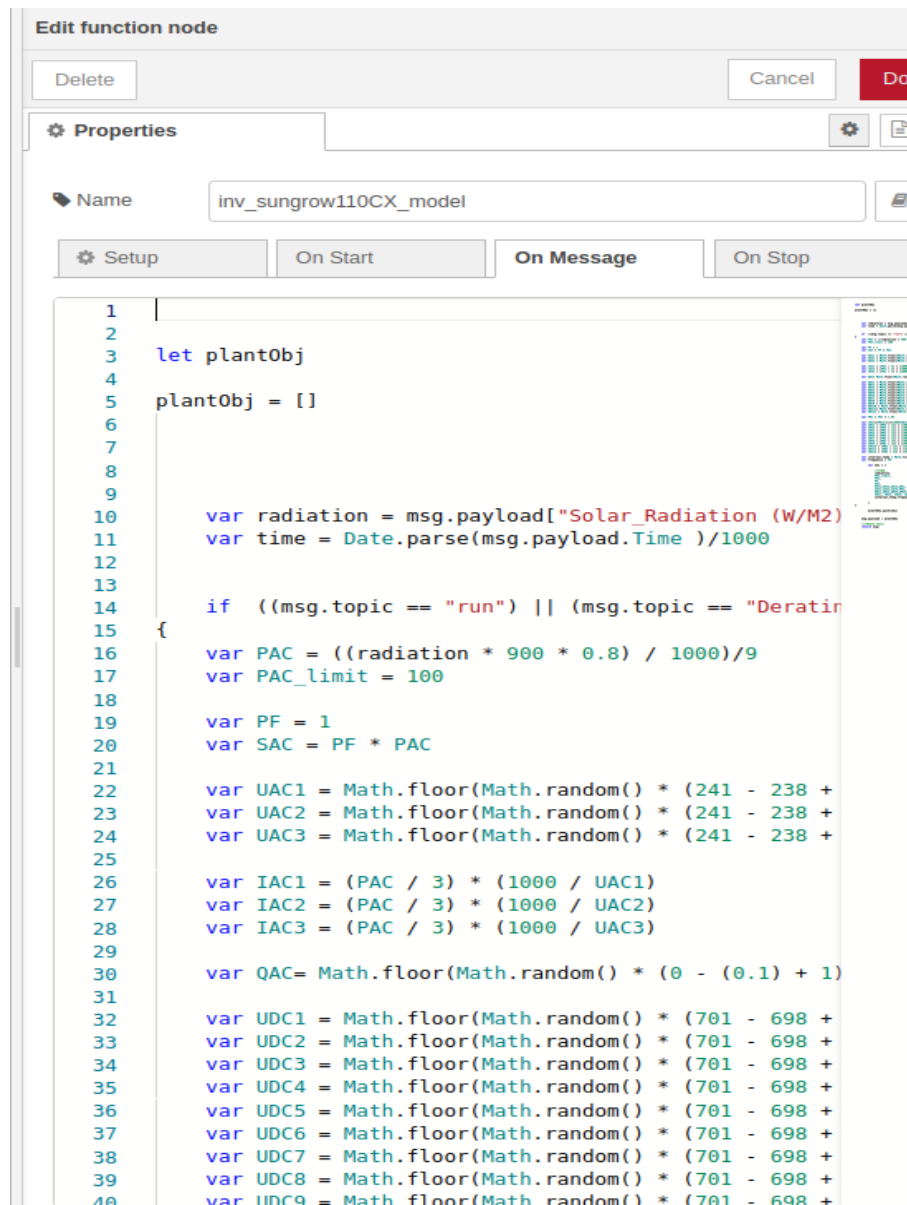


Fig – 6 function(inv_sungrow110CX_model)node edit box

➤ These output again giving into a function node which has code includes value, unit id, function code, quantity and address of the registers to where we are writing the data to write inverter simulation outputs to modbus server

Project: Diagnostic Box Configuration
Rev: 1.0

Edit function node

Delete Cancel Done

Properties

Name modbus write

Setup On Start On Message On Stop

```

1
2 msg.payload = {
3   'value': msg.payload[0],
4   'fc': 16,
5   'unitid': 10,
6   'address': 0,
7   'quantity': 39
8 }
9
10 return msg;
11

```

Fig – 7 function(modbus write)node edit box

- This function node outputs we are giving to modbus-flex-write node(have to install external node package node-red-contrib-modbus) where we have to give modbus server configuration details to where we are writing this inverter simulation outputs

Edit Modbus-Flex-Write node > Edit modbus-client node

Delete Cancel Update

Properties

Name Name

Type TCP

Host localhost

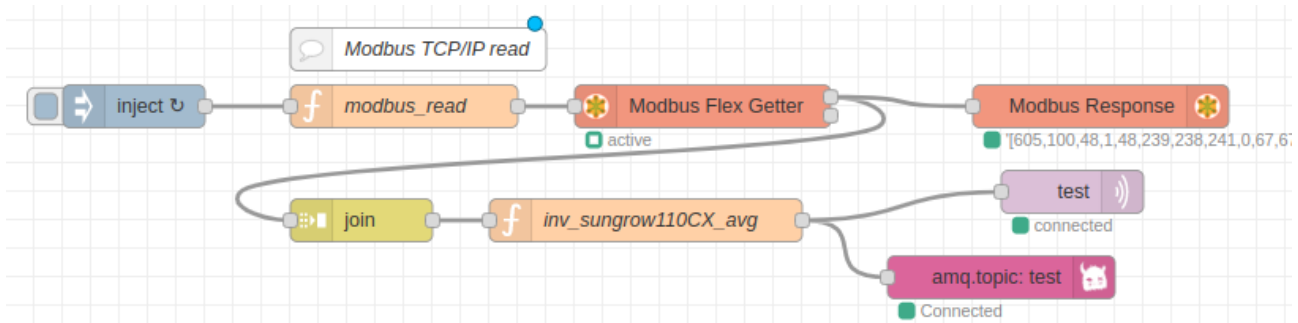
Port 10502

TCP Type DEFAULT

Project: Diagnostic Box Configuration
Rev: 1.0

- Modbus response node followed by modbus-flex-write node is used just to see the modbus responses sent to modbus server

Modbus read flow session:



- In above flow first we are setting triggering interval for the flow by using inject node for every 10s
- A function node which has code to read data from modbus server has connected after inject node code includes unit id,function code, quantity and address of the registers from where we are taking the data.

Edit function node

Delete

⚙ Properties

Name

modbus_read

⚙ Setup

On Start

On Mess

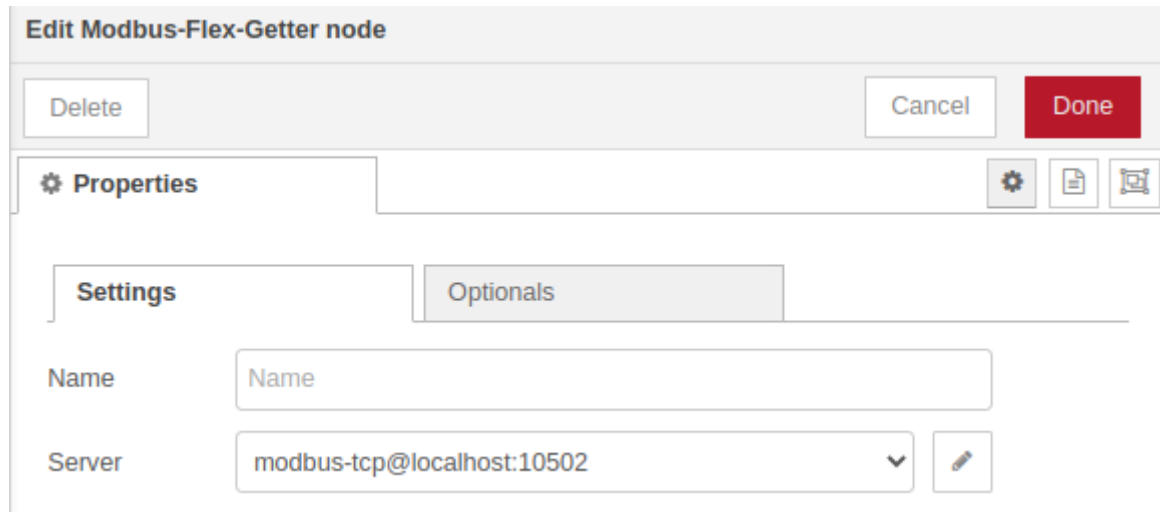
```

1 msg.payload = {
2   'unitid': 10,
3   'fc': 3,
4   'address': 0,
5   'quantity': 39,
6 };
7 return msg;
8
9

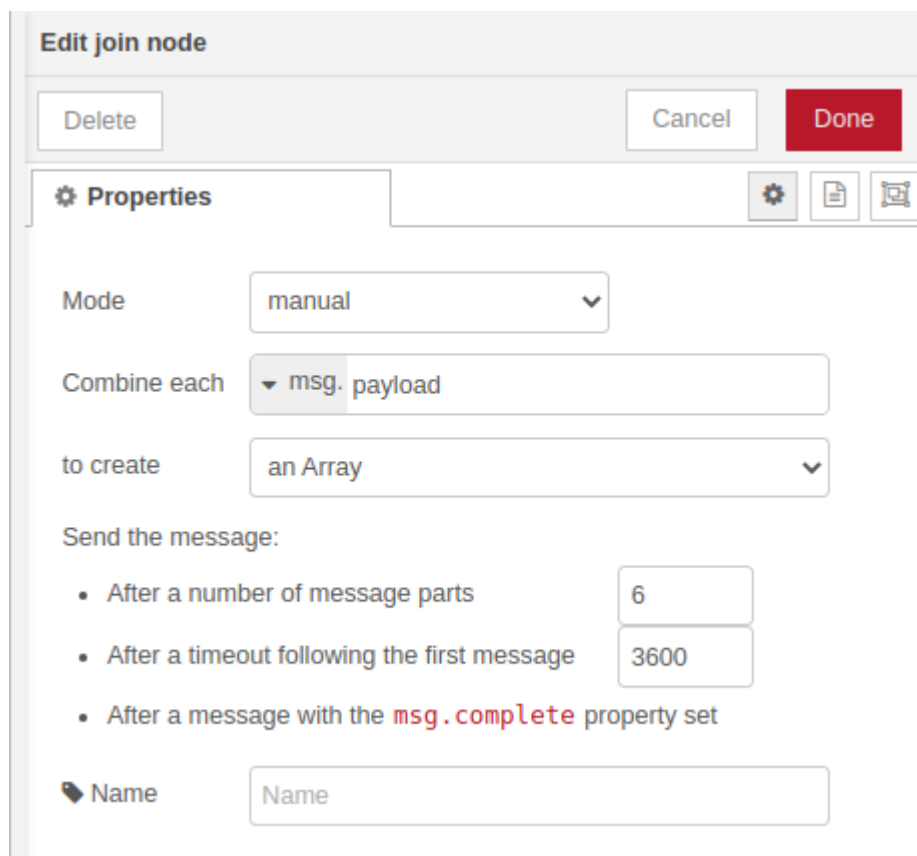
```

Project: Diagnostic Box Configuration
Rev: 1.0

- This function node outputs we are giving to modbus-flex-getter node(have to install external node package node-red-contrib-modbus) where we have to give modbus server configuration details from where we are taking this inverter simulation outputs



- Modbus response node followed by modbus-flex-getter node is used just to see the modbus responses sent to modbus server
- Join node used immediately after modbus-flex-getter node is to join every message coming in every 10s into one message per minute(Array of 6 messages per minute)



Project: Diagnostic Box Configuration
Rev: 1.0

- Forwarding joined messages into a function node which has a javascript logic code to calculate average of array of 6 messages came in a minute

```
01/11/2022, 14:51:33 node: Debug
1667294493147 : msg.payload : array[1]
▼ array[1]
▼ 0: object
  time: 1667294493147
  radiation: 0
  PAC_limit: 100
  PAC: 0
  PF: 1
  SAC: 0
  UAC1: 240.16666666666666
  UAC2: 238.66666666666666
  UAC3: 239.16666666666666
  QAC: 0
  IAC1: 0
  IAC2: 0
  IAC3: 0
  PDC: 0
  UDC1: 699.5
  UDC2: 699.33333333333334
  UDC3: 700
  UDC4: 699.66666666666666
  UDC5: 699.16666666666666
  UDC6: 699.5
  UDC7: 699.66666666666666
  UDC8: 699.5
  UDC9: 699.66666666666666
  UDC10: 700.16666666666666
  UDC11: 700.33333333333334
  UDC12: 699.33333333333334
  IDC1: 0
```

- output of function node connected to 2 nodes of 2 different message transfer protocol(mqtt,amqp)

1. mqtt out node: mqtt used to exchange messages from one machine to other which is connected to network .To publish messages to a machine we have to

Edit write file node

Delete

Cancel

Done

⚙️ Properties

⚙️

📄

🔗

📄 Filename

/home/iplon/Documents/modbus file.csv

⚙️ Action

append to file

▼

☐ Add newline (\n) to each payload?

☒ Create directory if it doesn't exist?

🚩 Encoding

default

▼

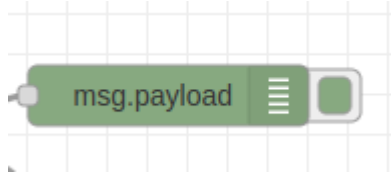
📁 Name

modbus file

Tip: The filename should be an absolute path, otherwise it will be relative to the working directory of the Node-RED process.

Project: **Diagnostic Box Configuration**
Rev: 1.0

- Use debug nodes to take out output from each nodes which we can see in debug window.



Project: Diagnostic Box Configuration
Rev: 1.0

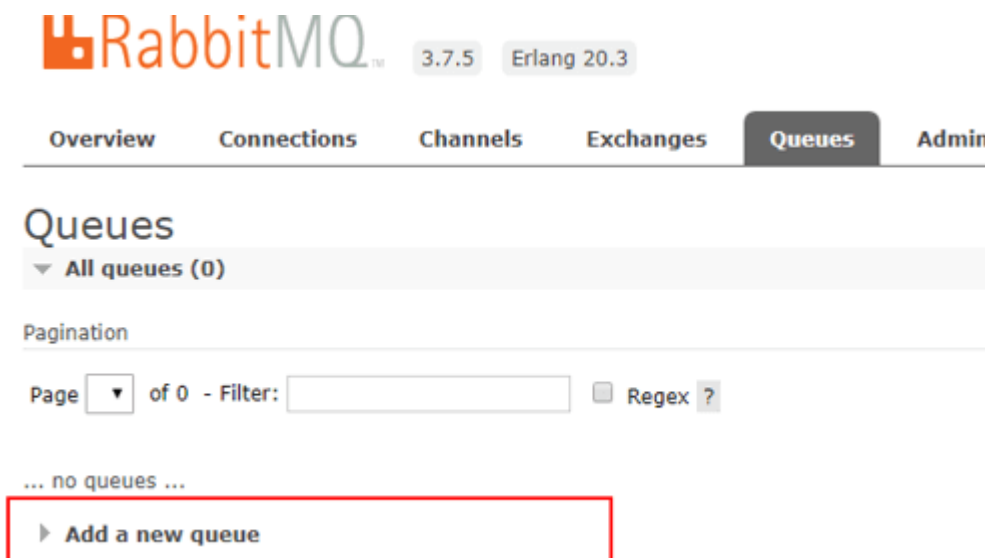
2. RABBITMQ

RabbitMQ is a **messaging broker - an intermediary for messaging**. It gives your applications a common platform to send and receive messages, and your messages a safe place to live until received.

installation guide for ubuntu: <https://linuxhint.com/install-rabbitmq-ubuntu/>
docker container installation and plugin enabling guide: <https://tewarid.github.io/2019/02/15/mqtt-with-rabbitmq-and-node-red.html>

after installation we can able to log into the management interface at <http://localhost:15672> using username/password guest/guest,

- Now we should create queue in rabbitmq navigate to **Queues** tab, you will see “**Add a new queue**” just click on that panel to expand like as shown below.
-



➤

After clicking on **Add a new queue** option, a new panel will open and that will contain a different properties to create a new queue like as shown below.

Project: Diagnostic Box Configuration
Rev: 1.0

Add a new queue

Type: Classic

Name:

Durability: Durable

Auto delete: ? No

Arguments: = String

Add
Auto expire ? | Message TTL ? | Overflow behaviour ?

Single active consumer ? | Dead letter exchange ? | Dead letter routing key ?

Max length ? | Max length bytes ?

Maximum priority ? | Lazy mode ? | Version ? | Master locator ?

Add queue

1. queue type there is 3 type of queues available in rabbitmq

1)classic

2)quorum

3)stream

2. Name

3. Durable (the queue will survive a broker restart)

Exclusive (used by only one connection and the queue will be deleted when that connection closes)

4. Auto-delete (queue that has had at least one consumer is deleted when last consumer unsubscribes)

5. Arguments (optional; used by plugins and broker-specific features such as message TTL, queue length limit, etc)

to know more about queues,arguments settings check

<https://www.rabbitmq.com/queues.html#basics>

<https://www.tutlane.com/tutorial/rabbitmq/rabbitmq-queues>

Project: Diagnostic Box Configuration
Rev: 1.0

Add a new queue

Type: Stream

Name:

Arguments: = String

Add Max length bytes Max time retention Max segment size in bytes Initial cluster size Leader locator

Add queue

2nd is name box is to give a naming to the queue

3rd **Arguments** (optional; used by plugins and broker-specific features such as message TTL, queue length limit, etc)

➤ After creating a queue, you can view queue which you have recently added, it is located just above the add queue panel like as shown below.

Queues

All queues (1)

Pagination

Page 1 of 1 - Filter: ☐ Regex

Overview			Messages			Message rates			+/-
Name	Features	State	Ready	Unacked	Total	incoming	deliver / get	ack	
demoqueue	D	idle	0	0	0				

After click on queue (**demoqueue**) name, the **Bindings** panel will expand and next it will ask for the exchange name, enter exchange ,routing key name which we have added in node-red amqp node setup and and click on **Bind** button.

Project: Diagnostic Box Configuration
Rev: 1.0

Overview **Connections** Channels

Exchanges **Queues** Admin

▼ Bindings

From	Routing key	Arguments	
(Default exchange binding)			

© tutlane.com

↓

This queue

Add binding to this queue

From exchange: *

Routing key:

Arguments: =

Bind

After click on **Bind** button, the defined exchange will be bind to our queue and that will be like as shown below.

▼ Bindings

From	Routing key	Arguments	
(Default exchange binding)			
<input type="text" value="demoexchange"/>	<input type="text" value="demokey"/>		Unbind

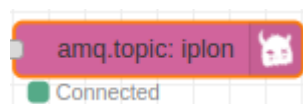
© tutlane.com

↓

This queue

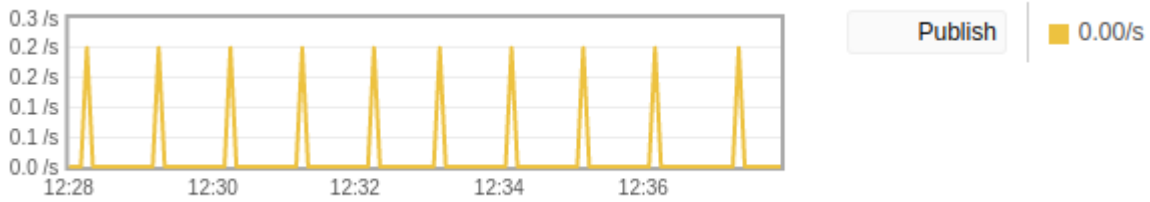
After binding, in case if you want to unbind it then you can click on unbind button to remove binding.

- Once node-red got connected to rabbitmq it will show connected symbol under amqp-out node like this



Project: Diagnostic Box Configuration
Rev: 1.0

so sent messages published and queue status we can see in rabbitmq like this

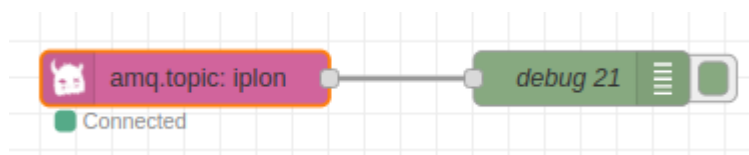


➤ incase there is no consumer is alive to consume that messages this will come to queue so qe can see how many messages are in queue in the graph like this



HOW TO ACKNOWLEDGE QUEUED MESSAGES :

➤ Use amqp-in node to consume published messages



Project:
Rev: 1.0

Diagnostic Box Configuration

Edit amqp-in node

Delete Cancel Done

Properties

Name: Leave blank to use exchange name

Broker: stream test

Prefetch: 3600

☐ noAck

Exchange Info

Type: Topic

Exchange Name: amq.topic

Routing Key: iplon

Queue Info

Queue Name: test

☐ Exclusive

☒ Durable

☐ Auto Delete

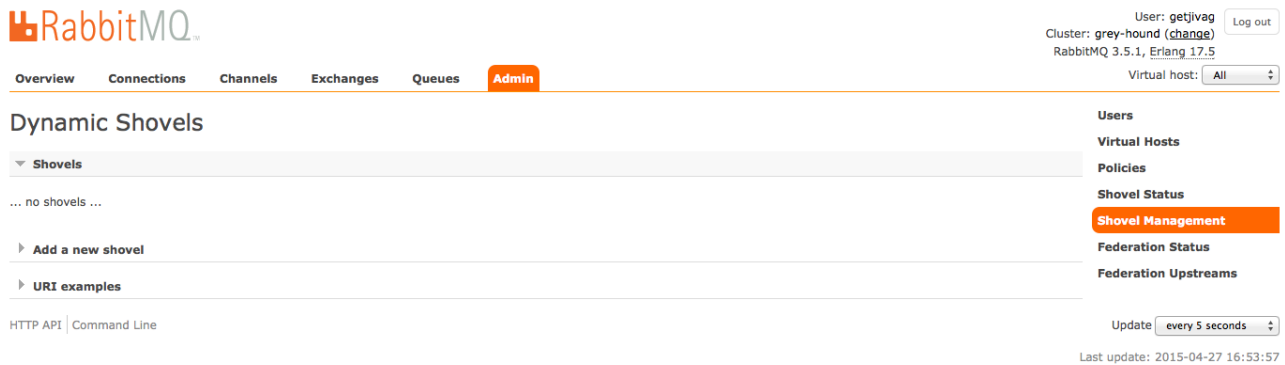
Disabled

- Add the broker configurations
- Fill all the box based on your rabbitmq setup you did and add the queue name from which queue we want to consume the data from tick the durable option only like the picture above
- Deploy the changes now we can see the published messages or queued messages in debug window.

Project: Diagnostic Box Configuration
Rev: 1.0

RABBITMQ SHOVEL MANAGEMET PLUGIN;

- Commant to enable shovel management plugin: **rabbitmq-plugins enable rabbitmq_shovel**
run this commant in terminal inside docker container
- now shovel management and shovel status is option is available in rabbitmq management
click on admin select shovel management



- Virtual host: Any vhost chosen
Name: The name of the shovel
Source: URI of the source instance. This should be the full CloudAMQP URI
Destination: URI of the destination instance.

▼ Add a new shovel

Virtual host:

Name:

Source: URI (?) Exchange: (Routing key: #)

Destination: URI (?) Exchange: (Routing key: #)

Prefetch count: (?)

Reconnect delay: (?) s

Add forwarding headers: (?)

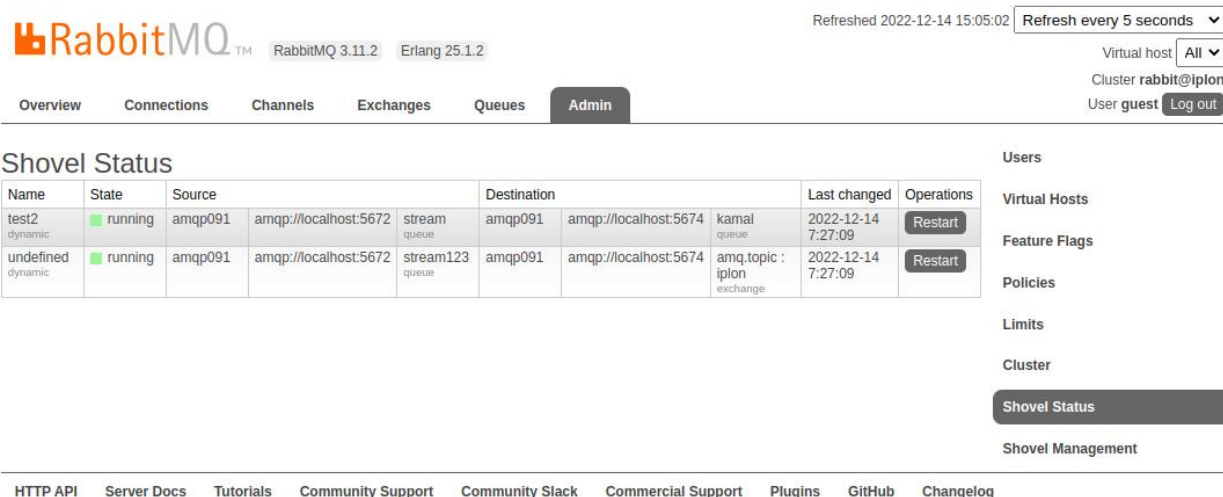
Acknowledgement mode: (?)

Auto-delete (?)

- After adding shovel we can check shovel status by clicking shovel status if it is running it will show like this

Project: Diagnostic Box Configuration

Rev: 1.0



The screenshot shows the RabbitMQ Admin interface. At the top, it displays 'RabbitMQ 3.11.2 Erlang 25.1.2' and a refresh button. The navigation bar includes 'Overview', 'Connections', 'Channels', 'Exchanges', 'Queues', and 'Admin'. The 'Shovel Status' section is active, showing a table with two rows of shovels. The first row is 'test2 dynamic' with state 'running', source 'amqp091', and destination 'amqp091'. The second row is 'undefined dynamic' with state 'running', source 'amqp091', and destination 'amqp091'. The table also shows 'Last changed' and 'Operations' (Restart) for each shovel. On the right sidebar, there are links for 'Users', 'Virtual Hosts', 'Feature Flags', 'Policies', 'Limits', 'Cluster', 'Shovel Status' (highlighted), and 'Shovel Management'.

Name	State	Source	Destination	Last changed	Operations
test2 dynamic	running	amqp091	amqp091	2022-12-14 7:27:09	Restart
undefined dynamic	running	amqp091	amqp091	2022-12-14 7:27:09	Restart

- we can now see the same queue in different rabbitmq
- for more information check this <https://www.rabbitmq.com/shovel.html>

RABBITMQ CLUSTERING

For clustering we need to setup different rabbitmq broker running in same network so, first we need to create a network in docker or local etc.

Create custom bridge network — mynet

```
# docker network create mynet
```

```
# docker network inspect mynet
```

now create rabbitmq brokers as per the given commands

Node 1

rabbit1 is hostname

erlang cookie is defined to facilitate cluster

Port 15672 is mapped to server to access GUI for RabbitMQ management.

```
# docker run -d --hostname rabbit1 --name myrabbit1 -p 15672:15672 -p 5672:5672 --network mynet -e RABBITMQ_ERLANG_COOKIE='rabbitcookie' rabbitmq:management
```

Node 2

Project: Diagnostic Box Configuration
Rev: 1.0

```
# docker run -d --hostname rabbit2 --name myrabbit2 -p 5673:5672 --link myrabbit1:rabbit1 --network mynet -e RABBITMQ_ERLANG_COOKIE='rabbitcookie' rabbitmq:management
```

Node 3

```
# docker run -d --hostname rabbit3 --name myrabbit3 -p 5674:5672 --link myrabbit1:rabbit1 --link myrabbit2:rabbit2 --network mynet -e RABBITMQ_ERLANG_COOKIE='rabbitcookie' rabbitmq:management
```

Now the nodes are up and running but cluster is still not configured. We have to connect to each running instance to configure and restart each node. We do following steps:

- connect to running instance
- stop the instance
- join cluster
- start the instance

Node 1

```
# docker exec -it myrabbit1 bashrabbitmqctl stop_apprabbitmqctl resetrabbitmqctl start_app
```

Node 2

```
# docker exec -it myrabbit2 bashrabbitmqctl stop_apprabbitmqctl resetrabbitmqctl join_cluster --ram rabbit@rabbit1rabbitmqctl start_app
```

Node 3

```
# docker exec -it myrabbit3 bashrabbitmqctl stop_apprabbitmqctl resetrabbitmqctl join_cluster --ram rabbit@rabbit1rabbitmqctl start_app
```

Now the nodes are up and running but cluster is still not configured. We have to connect to each running instance to configure and restart each node. We do following steps:

- ➔ connect to running instance
- ➔ stop the instance
- ➔ join cluster

Project: Diagnostic Box Configuration
Rev: 1.0

➔ start the instance

Node 1

```
# docker exec -it myrabbit1 bash
```

```
# rabbitmqctl stop_app
```

```
# rabbitmqctl reset
```

```
# rabbitmqctl start_app
```

Node 2

```
# docker exec -it myrabbit2 bash
```

```
# rabbitmqctl stop_app
```

```
# rabbitmqctl reset
```

```
# rabbitmqctl join_cluster --ram rabbit@rabbit1
```

```
# rabbitmqctl start_app
```

Node 3

```
# docker exec -it myrabbit2 bash
```

```
# rabbitmqctl stop_app
```

```
#rabbitmqctl reset
```

```
# rabbitmqctl join_cluster --ram rabbit@rabbit1
```

```
# rabbitmqctl start_app
```

connect to any one of the running instance and check the cluster status using following command

```
# rabbitmqctl cluster_status
```


Project: Diagnostic Box Configuration
Rev: 1.0

```
[ec2-user@ip-172-31-27-53 ~]$ docker exec -it myrabbit1 bash
root@rabbit1:/$ rabbitmqctl cluster_status
Cluster status of node rabbit@rabbit1 ...
Basic9

Cluster name: rabbit@rabbit1

Disk Nodes

rabbit@rabbit1
rabbit@rabbit2
rabbit@rabbit3

Running Nodes

rabbit@rabbit1
rabbit@rabbit2
rabbit@rabbit3

Versions

rabbit@rabbit1: RabbitMQ 3.8.2 on Erlang 22.2.6
rabbit@rabbit2: RabbitMQ on Erlang
rabbit@rabbit3: RabbitMQ on Erlang

Alarms

(none)

Network Partitions

(none)

Listeners

Node: rabbit@rabbit1, interface: [::], port: 25672, protocol: clustering, purpose: inter-node and CLI tool communication
Node: rabbit@rabbit1, interface: [::], port: 5672, protocol: amqp, purpose: AMQP 0-9-1 and AMQP 1.0
Node: rabbit@rabbit1, interface: [::], port: 15672, protocol: http, purpose: HTTP API

Feature flags

Flag: drop_unroutable_metric, state: enabled
Flag: empty_basic_get_metric, state: enabled
Flag: implicit_default_bindings, state: enabled
Flag: quorum_queue, state: enabled
Flag: virtual_host_metadata, state: enabled
root@rabbit1:/$
```

Now all rabbitmq brokers interconnected and created a cluster now
anychanges you are making in one node will reflect in another brokers

Project:
Rev: 1.0

Diagnostic Box Configuration

3. Telegraf

- **Influxdb installation** : follow the step <https://portal.influxdata.com/downloads/>
select the version and platform copy and paste the command in terminal
- **Telegraf installation** : follow the step <https://portal.influxdata.com/downloads/>
select the version and platform copy and paste the command in terminal

OR

- Use the tar file to install all the services like influxdb,telegraf,node-red,rabbitmq,grafana-

steps to use the file :

- \$ wget
https://github.com/malavikaiplon/i_sut/raw/2b41c8001fb77c9d6154c0a36514ffefab32caf2/docker-compose-grafana-influxdb.tar.gz

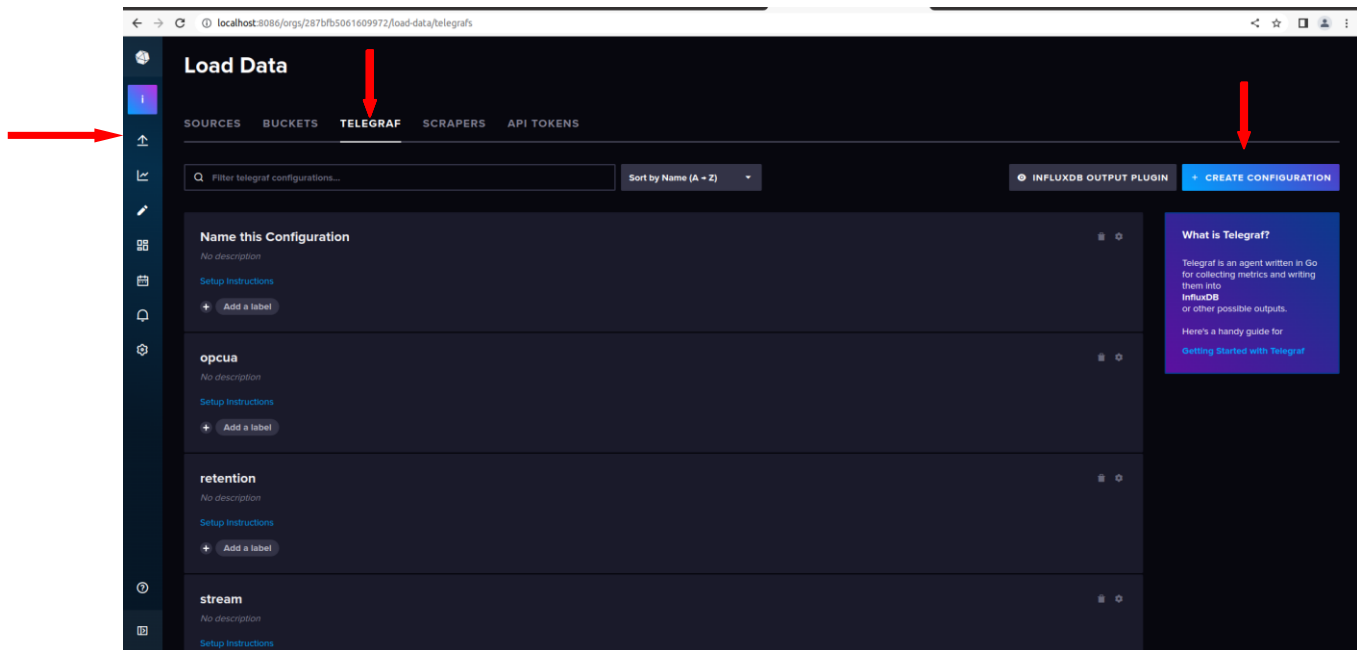
\$ tar -xvf docker-compose-grafana-influxdb.tar.gz

\$ cd docker-compose-grafana-influxdb/

\$ docker-compose up
- start influxdb docker container if it not started automatically go to the browser go to url <http://localhost:8086/>

Project: Diagnostic Box Configuration
Rev: 1.0

- click the uparrow icon > telegraf > create configuration



- Choose existing bucket or create new bucket and search for amq consumer click on it

• a new window will open there we can setup configuration for rabbitmq
give broker url in brokers = amqp://[localhost or ip of rabbitmq broker]:[port]
eg=amqp://localhost:5672

`brokers = ["amqp://localhost:5672/influxdb"]`

give username and password if there is any credential you set up other than default [guest,guest]

`username = "iplon"`
`password = "iplon321"`

give exchange = [type the same exchange from where we are consuming the message]

`exchange = "telegraf"`

Project: Diagnostic Box Configuration
Rev: 1.0

give queue=[type the same queue from where we are consuming the message]give

`queue = "test"`

set queue durability =durable

if you are taking input from a stream queue enable

`queue_passive =true`

`queue_passive = true`

`queue_durability = "durable"`

Create a Telegraf Configuration

Configuration Name

Your configuration name

Configuration Description

Your configuration description

```

1 # AMQP consumer plugin
2 [[inputs.amqp_consumer]]
3   ## Brokers to consume from. If multiple brokers are specified a random broker
4   ## will be selected anytime a connection is established. This can be
5   ## helpful for load balancing when not using a dedicated load balancer.
6   brokers = ["amqp://localhost:5672/influxdb"]
7
8   ## Authentication credentials for the PLAIN auth_method.
9   # username = ""
10  # password = ""
11
12  ## Name of the exchange to declare. If unset, no exchange will be declared.
13  exchange = "telegraf"
14
15  ## Exchange type; common types are "direct", "fanout", "topic", "header", "x-consistent-hash".
16  # exchange_type = "topic"
17
18  ## If true, exchange will be passively declared.
19  # exchange_passive = false
20
21  ## Exchange durability can be either "transient" or "durable".
22  # exchange_durability = "durable"
23
24  ## Additional exchange arguments.
25  # exchange_arguments = { }
26  # exchange_arguments = {"hash_property" = "timestamp"}
27
28  ## AMQP queue name.

```

Input plugin configuration will be appended to default **agent** settings and InfluxDB output upon saving

PREVIOUS

SAVE AND TEST

binding_key=[type the same routing key at the time of publishing message]

`binding_key = "test"`

data_format=json

`data_format = "json"`

Project: Diagnostic Box Configuration
Rev: 1.0

- click on save and test
- One window like this will open to configure api token click on generate new ap token click copy to click copy to clipboard paste this on terminal
- To start telegraf run the second command in terminal check if there is an error it will show in logs.
- After set up the telegraf configuration open the bucket you choose for this configuration. We can see the coming data on influxdb bucket .

Telegraf Setup Instructions

✕

1. Install the Latest Telegraf

You can install the latest Telegraf by visiting the [InfluxData Downloads](#) page. If you already have Telegraf installed on your system, make sure it's up to date. You will need version 1.9.2 or higher.

2. Configure your API Token

Your API token is required for pushing data into InfluxDB. You can copy the following command to your terminal window to set an environment variable with your API token.

```
export INFLUX_TOKEN=<INFLUX_TOKEN>
```

COPY TO CLIPBOARD🔄 GENERATE NEW API TOKENCLI

3. Start Telegraf

Finally, you can run the following command to start the Telegraf agent running on your machine.

```
telegraf --config http://localhost:8086/api/v2/telegrafs/0a77c4ba00224000
```

COPY TO CLIPBOARDCLI

