

# Table of Contents

Telegraf.....	2
Installation on RHEL.....	2
Agent Configurations.....	2
Global Tags.....	2
Agent.....	2
Input Plugin.....	3
Output Plugin.....	3
Measurement Filtering.....	3
Starting Telegraf Agent.....	4
Sample Telegraf Configuration.....	4
Jolokia.....	7
Download.....	7
Configuring Jolokia Agent.....	8
Configuring on Tomcat.....	8
Configuring on WebLogic.....	8
Configuring for Spring Boot.....	8
Configuring Multiple Jolokia Instances on Same VM.....	8
Configuring for cassandra.....	8
Configuring for Kafka.....	8
Kafka +Zookeeper Monitoring.....	9
Telegraf conf for kafka input metrics.....	9
Redis Monitoring.....	18
Telegraf conf for Redis input metrics.....	18
RabbitMQ Monitoring.....	18
Telegraf conf for Rabbitmq input metrics.....	19
Aerospike Monitoring.....	19
Telegraf conf for Aerospike input metrics.....	19
ElasticSearch Monitoring.....	19
Telegraf conf for elasticsearch input metrics.....	20
MongoDB Monitoring.....	20
Telegraf conf for MongoDB input metrics.....	20
MySQL Monitoring.....	20
Telegraf conf for Mysql input metrics.....	20
Prometheus Monitoring.....	21
Telegraf conf for prometheus input metrics.....	21
Cassandra Monitoring.....	21
Telegraf conf for cassandra input metrics.....	21
Port +Health check URL Monitoring.....	27
Purpose of Port/ping/url Monitoring:.....	27
Monitoring Architecture.....	27
Monitoring Via Heartbeat:.....	28
Heartbeat Insatllation:.....	28
Heartbeat configuration:.....	28
Steps for PORT,PING,URL Heartbeat Setup.....	28
Heartbeat output setup.....	30
Elasticsearch output:.....	31
Logstash Output:.....	31
Redis OutPut:.....	31
File Output:.....	31
Console Output:.....	32

Monitoring Via Shell Script:.....	32
grafana screenshot by grafana-image-renderer plugin.....	33

# System Metrics+Middleware Service Metrics to InfluxDB with Telegraf

## Telegraf

Telegraf is part of the TICK Monitoring Stack and is a plugin-driven server agent for collecting & reporting metrics.

Telegraf has integrations with over 100 input plugins to pull data from different type of applications & middleware.

It also has output plugins to send metrics to a variety of other datastores, services, and message queues, including InfluxDB, Graphite, OpenTSDB, Datadog, Librato, Kafka, MQTT, NSQ, and many others.

### Installation on RHEL

- `Wget https://dl.influxdata.com/telegraf/releases/telegraf-1.6.0-1.x86_64.rpm`
- `yum localinstall telegraf-1.6.0-1.x86_64.rpm`

## Agent Configurations

### Global Tags

Values specified under **[global\_tags]** will be tagged in all metrics that are published to the output plugin.

### Agent

All configuration related to agent are put under **[agent]** tag. Useful attributes under this tag are:

- **interval:** Default data collection interval for all inputs
- **round\_interval:** Rounds collection interval to 'interval' i.e. if interval "10s" then always collect on :00, :10, :20, etc.
- **metric\_batch\_size:** Telegraf will send metrics to output in batch of at most metric\_batch\_size metrics.
- **metric\_buffer\_limit:** Telegraf will cache metric\_buffer\_limit metrics for each output, and will flush this buffer on a successful write.

- **collection\_jitter:** Collection jitter is used to jitter the collection by a random amount. Each plugin will sleep for a random time within jitter before collecting.
- **flush\_interval:** Default data flushing interval for all outputs. You should not set this below interval. Maximum flush\_interval will be flush\_interval + flush\_jitter
- **hostname:** Override default hostname, if empty use os.Hostname().

## Input Plugin

Input plugins collates data from the provided input metrics & aggregates the data that needs to be pushed to an output. While each input plugin has their separate configurations, common configurations for input plugins are:

- **interval:** How often to gather this metric. This will override the global configuration of interval for the plugin where specified .
- **name\_override:** Override the base name of the measurement. (Default is the name of the input).
- **tags:** A map of tags to apply to a specific input's measurements.

## Output Plugin

This will decide the database/queue where data collated by telegraf needs to be published. Mentioned is the output plugin for InfluxDB that will be used in our case:

### [outputs.influxdb]

```
urls = ["http://127.0.1.1:8086"] //influx server ip
database = "telegraf_growth" //influx database name
skip_database_creation = true //if database is already exists
timeout = "5s" //Timeout for HTTP messages
username = "test1" //database user
password = "test1"
retention_policy = "two_weeks" //default retention policy
```

## Measurement Filtering

To filter the fields that we intend to push to InfluxDB for a particular metrics, we need to do filtering in the input plugin. Few important attributes for that are:

- **namepass:** An array of glob pattern strings. Only points whose measurement name matches a pattern in this list are emitted.

- **namedrop:** The inverse of namepass. If a match is found the point is discarded. This is tested on points after they have passed the namepass test.
- **fieldpass:** An array of glob pattern strings. Only fields whose field key matches a pattern in this list are emitted.
- **fielddrop:** The inverse of field pass. Fields with a field key matching one of the patterns will be discarded from the point. This is tested on points after they have passed the fieldpass test.
- **tagpass:** A table mapping tag keys to arrays of glob pattern strings. Only points that contain a tag key in the table and a tag value matching one of its patterns is emitted.
- **tagdrop:** The inverse of tagpass. If a match is found the point is discarded. This is tested on points after they have passed the tagpass test.
- **taginclude:** An array of glob pattern strings. Only tags with a tag key matching one of the patterns are emitted. In contrast to tagpass, which will pass an entire point based on its tag, taginclude removes all non-matching tags from the point. This filter can be used on both inputs & outputs, but it is recommended to be used on inputs, as it is more efficient to filter out tags at the ingestion point.
- **tagexclude:** The inverse of tag include. Tags with a tag key matching one of the patterns will be discarded from the point.

## Starting Telegraf Agent

- *systemctl start telegraf* (//systemd installation)
- *telegraf -config test\_telegraf.conf -test* (// to test output of configuration)

## Sample Telegraf Configuration

*[global\_tags]*

*lob = "name of lob"*

*application = "name of application"*

*# Configuration for telegraf agent*

*[agent]*

*interval = "300s"*

*round\_interval = true*

```
metric_batch_size = 1000
metric_buffer_limit = 10000
collection_jitter = "0s"
flush_interval = "300s"
flush_jitter = "0s"
hostname = "10.5.78.42"
omit_hostname = false
```

#### ##### OUTPUT PLUGINS #####

```
[outputs]
[outputs.influxdb]
urls = ["http://127.0.1.1:8086"]
database = "telegraf_growth"
skip_database_creation = true
timeout = "5s"
username = "test1"
password = "test1"
```

#### ##### INPUT PLUGINS #####

```
[[inputs.cpu]]
percpu = false
totalcpu = true
collect_cpu_time = false
report_active = false
fieldpass = ["usage_idle"]
[[inputs.diskio]]
fieldpass = ["reads", "writes", "read_bytes", "write_bytes"]
[[inputs.mem]]
fieldpass = ["cached", "free", "used", "total"]
```

```
[[inputs.netstat]]
  fieldpass = ["tcp_established","tcp_time_wait"]
```

```
[[inputs.disk]]
```

```
[[inputs.system]]
```

```
[inputs.processes]]
```

```
[[inputs.swap]]
```

```
[[inputs.jolokia2_agent]]
```

```
urls = ["http://localhost:7778/jolokia"]
```

```
[inputs.jolokia2_agent.tags]
```

```
app = "at-backend"
```

```
[[inputs.jolokia2_agent.metric]]
```

```
name = "java_threading"
```

```
mbean = "java.lang:type=Threading"
```

```
paths = ["ThreadCount","PeakThreadCount","DaemonThreadCount","TotalStartedThreadCount"]
```

```
[[inputs.jolokia2_agent.metric]]
```

```
name = "java_memory_usage"
```

```
mbean = "java.lang:type=Memory"
```

```
paths = ["HeapMemoryUsage", "NonHeapMemoryUsage"]
```

```
[[inputs.jolokia2_agent.metric]]
```

```
name = "java_memory_pool_usage"
```

```
mbean = "java.lang:name=*,type=MemoryPool"
```

```
paths = ["Usage", "CollectionUsage"]
```

```
tag_keys = ["name"]
```

```
[[inputs.jolokia2_agent]]
```

```

urls = ["http://localhost:7777/jolokia"]
[inputs.jolokia2_agent.tags]
  app = "profile-manager" # Required in case of multiple applications are running on
  same VM.
[[inputs.jolokia2_agent.metric]]
  name = "java_memory_usage"
  mbean = "java.lang:type=Memory"
  paths = ["HeapMemoryUsage", "NonHeapMemoryUsage"]
[[inputs.jolokia2_agent.metric]]
  name = "java_memory_pool_usage"
  mbean = "java.lang:name=*,type=MemoryPool"
  paths = ["Usage", "CollectionUsage"]
  tag_keys = ["name"]

```

## Jolokia

The sole purpose of Jolokia agent is to expose JMX metrics over HTTP. Once JMX metrics are exposed over HTTP, it can be used for monitoring/alerting of these metrics.

In our use case, we ingest Telegraf from metrics exposed by Jolokia agent and finally push it to Influx. By default metrics are accessible on <http://127.0.0.1:8778/jolokia/>

## Download

Download the latest version <http://search.maven.org/remotecontent?filepath=org/jolokia/jolokia-jvm/1.5.0/jolokia-jvm-1.5.0-agent.jar> on the server

# Configuring Jolokia Agent

Generic configuration options can be found on

<https://jolokia.org/reference/html/agents.html#jvm-agent>

## Configuring on Tomcat

Add the following arguments in JAVA\_OPTS for the server in catalina.sh of running tomcat

```
-javaagent:./<<Installation Path>>/jolokia-jvm-1.6.0-agent.jar
```

## Configuring on WebLogic

Go to Environment > Servers > Configuration > Server Start

Add the following line in Arguments sections and restart the server:

```
-javaagent:./<<Installation Path>>/jolokia-jvm-1.6.0-agent.jar
```

## Configuring for Spring Boot

Add the mentioned line after the E28093jar command for starting the boot application:

```
-javaagent:./<<Installation Path>>/jolokia-jvm-1.6.0-agent.jar
```

## Configuring Multiple Jolokia Instances on Same VM

In this case run both agents on a separate port, as mentioned below:

```
-javaagent:./<<Installation Path>>/jolokia-jvm-1.6.0-agent.jar=port=7778
```

```
-javaagent:./<<Installation Path>>/jolokia-jvm-1.6.0-agent.jar=7777
```

## Configuring for cassandra

Add the following arguments in JAVA\_OPTS for the server in cassandra-env.sh

```
-javaagent:./<<Installation Path>>/jolokia-jvm-1.6.0-agent.jar
```

example:

```
JVM_OPTS="$JVM_OPTS -javaagent:/data/jolokia-jvm-1.6.0-agent.jar"
```

## Configuring for Kafka

Add the following snippet in kafka-server-start.sh and Restart the Kafka service

```
export KAFKA_JMX_OPTS="
```

```
-javaagent:/usr/share/java/kafka/jolokia-jvm-agent.jar=port=8778,host=localhost \
```



```
-Dcom.sun.management.jmxremote=true \  
-Dcom.sun.management.jmxremote.authenticate=false \  
-Dcom.sun.management.jmxremote.ssl=false \  
-Djava.rmi.server.hostname=localhost \  
-Dcom.sun.management.jmxremote.host=localhost \  
-Dcom.sun.management.jmxremote.port=9999 \  
-Dcom.sun.management.jmxremote.rmi.port=9999 \  
-Djava.net.preferIPv4Stack=true"
```

## ***Kafka + Zookeeper Monitoring***

*Step: 1. download + configure jolokia agent jar for kafka and restart kafka (as mentioned under jolokia doc)*

*step: 2. configure telegraf configuration for kafka input metrics.*

*Step: 3 restart telegraf service.*

### ***Telegraf conf for kafka input metrics***

```
[[inputs.jolokia2_agent]]  
  urls = ["http://localhost:8778/jolokia"]  
  
### This collects thread counts metrics.  
[[inputs.jolokia2_agent.metric]]  
  name = "java_threading"  
  mbean = "java.lang:type=Threading"  
  paths =  
["ThreadCount", "PeakThreadCount", "DaemonThreadCount", "TotalStartedThreadCount"]  
  
## This collects garbage collection metrics.  
[[inputs.jolokia2_agent.metric]]  
  name = "garbage_collector"
```

```
mbean = "java.lang:type=GarbageCollector,name=*"
paths = ["CollectionCount", "CollectionTime"]
tag_keys = ["name"]
```

### heap and nonheap memory usage

```
[[inputs.jolokia2_agent.metric]]
name = "java_memory_usage"
mbean = "java.lang:type=Memory"
paths = ["HeapMemoryUsage", "NonHeapMemoryUsage"]
```

####

```
[[inputs.jolokia2_agent.metric]]
name = "java_memory_pool_usage"
mbean = "java.lang:name=*,type=MemoryPool"
paths = ["Usage", "CollectionUsage"]
```

```
[[inputs.jolokia2_agent.metric]]
name = "kafka_ReplicaManager_UnderMinIsrPartitionCount"
mbean = "kafka.server:type=ReplicaManager,name=UnderMinIsrPartitionCount"
paths = ["Value"]
tag_keys = ["name"]
```

```
[[inputs.jolokia2_agent.metric]]
name = "kafka_ReplicaManager_UnderReplicatedPartitions"
mbean = "kafka.server:type=ReplicaManager,name=UnderReplicatedPartitions"
paths = ["Value"]
tag_keys = ["name"]
```

```
[[inputs.jolokia2_agent.metric]]
name = "kafka_Partition_UnderMinIsr"
mbean = "kafka.cluster:type=Partition,name=UnderMinIsr,topic=*,partition=*"
```

```
paths = ["Value"]
tag_keys = ["name"]
```

```
[[inputs.jolokia2_agent.metric]]
name = "kafka_KafkaController_OfflinePartitionsCount"
tag_keys = ["name"]
```

#### # Kafka Controller Metrics

```
[[inputs.jolokia2_agent.metric]]
name = "kafka_KafkaController_ActiveControllerCount"
mbean = "kafka.controller:type=KafkaController,name=ActiveControllerCount"
paths = ["Value"]
tag_keys = ["name"]
```

#### ###Kafka Server Broker Topic Metrics

```
[[inputs.jolokia2_agent.metric]]
name = "kafka_BrokerTopicMetrics_BytesInPerSec"
mbean = "kafka.server:type=BrokerTopicMetrics,name=BytesInPerSec"
paths = ["Count","MeanRate"]
tag_keys = ["name"]
```

```
[[inputs.jolokia2_agent.metric]]
name = "kafka_BrokerTopicMetrics_BytesInPerSec_topic"
mbean = "kafka.server:type=BrokerTopicMetrics,name=BytesInPerSec,topic=*"
paths = ["Count","MeanRate"]
tag_keys = ["topic"]
```

```
[[inputs.jolokia2_agent.metric]]
name = "kafka_BrokerTopicMetrics_BytesOutPerSec"
mbean = "kafka.server:type=BrokerTopicMetrics,name=BytesOutPerSec"
paths = ["Count","MeanRate"]
```

```
tag_keys = ["name"]
```

```
[[inputs.jolokia2_agent.metric]]
```

```
name = "kafka_BrokerTopicMetrics_BytesOutPerSec_topic"
```

```
mbean = "kafka.server:type=BrokerTopicMetrics,name=BytesOutPerSec,topic=*"
```

```
paths = ["Count", "MeanRate"]
```

```
tag_keys = ["topic"]
```

```
[[inputs.jolokia2_agent.metric]]
```

```
name = "kafka_BrokerTopicMetrics_TotalProduceRequestsPerSec"
```

```
mbean = "kafka.server:type=BrokerTopicMetrics,name=TotalProduceRequestsPerSec"
```

```
paths = ["Count", "MeanRate"]
```

```
tag_keys = ["name"]
```

```
[[inputs.jolokia2_agent.metric]]
```

```
name = "kafka_BrokerTopicMetrics_TotalFetchRequestsPerSec"
```

```
mbean = "kafka.server:type=BrokerTopicMetrics,name=TotalFetchRequestsPerSec"
```

```
paths = ["Count", "MeanRate"]
```

```
tag_keys = ["name"]
```

```
[[inputs.jolokia2_agent.metric]]
```

```
name = "kafka_BrokerTopicMetrics_FailedProduceRequestsPerSec"
```

```
mbean = "kafka.server:type=BrokerTopicMetrics,name=FailedProduceRequestsPerSec"
```

```
paths = ["Count", "MeanRate"]
```

```
tag_keys = ["name"]
```

```
[[inputs.jolokia2_agent.metric]]
```

```
name = "BrokerTopicMetrics_FailedFetchRequestsPerSec"
```

```
mbean = "kafka.server:type=BrokerTopicMetrics,name=FailedFetchRequestsPerSec"
```

```
paths = ["Count", "MeanRate"]
```

```
tag_keys = ["name"]
```

```
[[inputs.jolokia2_agent.metric]]
```

```
name = "kafka_ControllerStats_LeaderElectionRateAndTimeMs"
```

```
mbean = "kafka.controller:type=ControllerStats,name=LeaderElectionRateAndTimeMs"
```

```
paths = ["Count", "MeanRate"]
```

```
tag_keys = ["name"]
```

```
[[inputs.jolokia2_agent.metric]]
```

```
name = "kafka_ControllerStats_UncleanLeaderElectionsPerSec"
```

```
mbean = "kafka.controller:type=ControllerStats,name=UncleanLeaderElectionsPerSec"
```

```
paths = ["Count", "MeanRate"]
```

```
tag_keys = ["name"]
```

```
[[inputs.jolokia2_agent.metric]]
```

```
name = "kafka_ReplicaManager_PartitionCount"
```

```
mbean = "kafka.server:type=ReplicaManager,name=PartitionCount"
```

```
paths = ["Value"]
```

```
tag_keys = ["name"]
```

```
[[inputs.jolokia2_agent.metric]]
```

```
name = "kafka_ReplicaManager_LeaderCount"
```

```
mbean = "kafka.server:type=ReplicaManager,name=LeaderCount"
```

```
paths = ["Value"]
```

```
tag_keys = ["name"]
```

```
[[inputs.jolokia2_agent.metric]]
```

```
name = "kafka_ReplicaManager_OfflineReplicaCount"
```

```
mbean = "kafka.server:type=ReplicaManager,name=OfflineReplicaCount"
```

```
paths = ["Value"]
```

```
tag_keys = ["name"]
```

```
[[inputs.jolokia2_agent.metric]]
```

```
name = "kafka_ReplicaFetcherManager_MaxLag_Replica"
```

```
mbean = "kafka.server:type=ReplicaFetcherManager,name=MaxLag,clientId=Replica"
```

```
paths = ["Value"]
```

```
tag_keys = ["name"]
```

```
[[inputs.jolokia2_agent.metric]]
```

```
name = "kafka_KafkaRequestHandlerPool_RequestHandlerAvgIdlePercent"
```

```
mbean = "kafka.server:type=KafkaRequestHandlerPool,name=RequestHandlerAvgIdlePercent"
```

```
paths = ["Count","MeanRate"]
```

```
tag_keys = ["name"]
```

```
[[inputs.jolokia2_agent.metric]]
```

```
name = "kafka_SocketServer_NetworkProcessorAvgIdlePercent"
```

```
mbean = "kafka.network:type=SocketServer,name=NetworkProcessorAvgIdlePercent"
```

```
paths = ["Value"]
```

```
tag_keys = ["name"]
```

```
[[inputs.jolokia2_agent.metric]]
```

```
name = "kafka_RequestChannel_RequestQueueSize"
```

```
mbean = "kafka.network:type=RequestChannel,name=RequestQueueSize"
```

```
paths = ["Value"]
```

```
tag_keys = ["name"]
```

```
[[inputs.jolokia2_agent.metric]]
```

```
name = "kafka_RequestMetrics_TotalTimeMs_Produce_FetchConsumer_FetchFollower"
```

```
mbean = "kafka.network:type=RequestMetrics,name=TotalTimeMs,request={Produce|FetchConsumer|FetchFollower}"
```

```
paths = ["Count","Mean"]
```

```
tag_keys = ["name"]
```

```
[[inputs.jolokia2_agent.metric]]
```

```

    name =
"kafka_RequestMetrics_RequestQueueTimeMs_Produce_FetchConsumer_FetchFollower"

    mbean =
"kafka.network:type=RequestMetrics,name=RequestQueueTimeMs,request={Produce|
FetchConsumer|FetchFollower}"

    paths = ["Count","Mean"]
    tag_keys = ["name"]

[[inputs.jolokia2_agent.metric]]

    name = "kafka_RequestMetrics_LocalTimeMs_Produce_FetchConsumer_FetchFollower"
    mbean = "kafka.network:type=RequestMetrics,name=LocalTimeMs,request={Produce|
FetchConsumer|FetchFollower}"

    paths = ["Count","Mean"]
    tag_keys = ["name"]

[[inputs.jolokia2_agent.metric]]

    name = "kafka_RequestMetrics_RemoteTimeMs_Produce_FetchConsumer_FetchFollower"
    mbean = "kafka.network:type=RequestMetrics,name=RemoteTimeMs,request={Produce|
FetchConsumer|FetchFollower}"

    paths = ["Count","Mean"]
    tag_keys = ["name"]

[[inputs.jolokia2_agent.metric]]

    name =
"kafka_RequestMetrics_ResponseQueueTimeMs_Produce_FetchConsumer_FetchFollower"

    mbean =
"kafka.network:type=RequestMetrics,name=ResponseQueueTimeMs,request={Produce|
FetchConsumer|FetchFollower}"

    paths = ["Count","Mean"]
    tag_keys = ["name"]

[[inputs.jolokia2_agent.metric]]

    name =
"kafka_RequestMetrics_ResponseSendTimeMs_Produce_FetchConsumer_FetchFollower"

```

```
mbean =  
"kafka.network:type=RequestMetrics,name=ResponseSendTimeMs,request={Produce|  
FetchConsumer|FetchFollower}"
```

```
paths = ["Count", "Mean"]
```

```
tag_keys = ["name"]
```

```
[[inputs.jolokia2_agent.metric]]
```

```
name = "kafka_BrokerTopicMetrics_MessagesInPerSec"
```

```
mbean = "kafka.server:type=BrokerTopicMetrics,name=MessagesInPerSec"
```

```
paths = ["Count", "MeanRate"]
```

```
tag_keys = ["name"]
```

```
[[inputs.jolokia2_agent.metric]]
```

```
name = "kafka_BrokerTopicMetrics_MessagesInPerSec_topic"
```

```
mbean = "kafka.server:type=BrokerTopicMetrics,name=MessagesInPerSec,topic=*"
```

```
paths = ["Count", "OneMinuteRate"]
```

```
tag_keys = ["topic"]
```

```
[[inputs.jolokia2_agent.metric]]
```

```
name = "kafka_LogFlushStats_LogFlushRateAndTimeMs"
```

```
mbean = "kafka.log:type=LogFlushStats,name=LogFlushRateAndTimeMs"
```

```
paths = ["Count", "Mean"]
```

```
tag_keys = ["name"]
```

```
[[inputs.jolokia2_agent.metric]]
```

```
name = "kafka_ReplicaManager_IsrShrinksPerSec"
```

```
mbean = "kafka.server:type=ReplicaManager,name=IsrShrinksPerSec"
```

```
paths = ["Count", "MeanRate"]
```

```
tag_keys = ["name"]
```

```
[[inputs.jolokia2_agent.metric]]
```



```
name = "kafka_ReplicaManager_IsrExpandsPerSec"
mbean = "kafka.server:type=ReplicaManager,name=IsrExpandsPerSec"
paths = ["Count", "MeanRate"]
tag_keys = ["name"]
```

```
[[inputs.jolokia2_agent.metric]]
```

```
name = "kafka_DelayedOperationPurgatory_delayedOperation_Produce_PurgatorySize"
mbean =
```

```
"kafka.server:type=DelayedOperationPurgatory,delayedOperation=Produce,name=PurgatorySize"
```

```
paths = ["Value"]
```

```
tag_keys = ["name"]
```

```
[[inputs.jolokia2_agent.metric]]
```

```
name = "kafka_DelayedOperationPurgatory_delayedOperation_Fetch_PurgatorySize"
```

```
mbean =
```

```
"kafka.server:type=DelayedOperationPurgatory,delayedOperation=Fetch,name=PurgatorySize"
```

```
paths = ["Value"]
```

```
tag_keys = ["name"]
```

## # Zookeeper Metrics

```
[[inputs.jolokia2_agent.metric]]
```

```
name = "zookeeper_disconnects"
```

```
mbean = "kafka.server:type=SessionExpireListener,name=ZooKeeperDisconnectsPerSec"
```

```
paths = ["Count", "OneMinuteRate", "FiveMinuteRate", "FifteenMinuteRate", "MeanRate"]
```

```
[[inputs.jolokia2_agent.metric]]
```

```
name = "zookeeper_sync_connects"
```

```
mbean = "kafka.server:type=SessionExpireListener,name=ZooKeeperSyncConnectsPerSec"
```

```
paths = ["Count", "OneMinuteRate", "FiveMinuteRate", "FifteenMinuteRate", "MeanRate"]
```

```
[[inputs.jolokia2_agent.metric]]
```

```
name = "zookeeper_auth_failures"
```

```
mbean = "kafka.server:type=SessionExpireListener,name=ZooKeeperAuthFailuresPerSec"
```

```

    paths = ["Count", "OneMinuteRate", "FiveMinuteRate", "FifteenMinuteRate", "MeanRate"]
    [[inputs.jolokia2_agent.metric]]
    name = "zookeeper_readonly_connects"
    mbean =
"kafka.server:type=SessionExpireListener,name=ZooKeeperReadOnlyConnectsPerSec"
    paths = ["Count", "OneMinuteRate", "FiveMinuteRate", "FifteenMinuteRate", "MeanRate"]
    [[inputs.jolokia2_agent.metric]]
    name = "zookeeper_authentications"
    mbean =
"kafka.server:type=SessionExpireListener,name=ZooKeeperSaslAuthenticationsPerSec"
    paths = ["Count", "OneMinuteRate", "FiveMinuteRate", "FifteenMinuteRate", "MeanRate"]
    [[inputs.jolokia2_agent.metric]]
    name = "zookeeper_expires"
    mbean = "kafka.server:type=SessionExpireListener,name=ZooKeeperExpiresPerSec"
    paths = ["Count", "OneMinuteRate", "FiveMinuteRate", "FifteenMinuteRate", "MeanRate"]

```

## Redis Monitoring

*step: 1. configure telegraf configuration (telegraf.conf) for Redis input metrics.*

*Step: 2 restart telegraf service.*

### ***Telegraf conf for Redis input metrics***

```

[[inputs.redis]]

## [protocol://][:password]@address[:port]
## If no servers are specified, then localhost is used as the host.
## If no port is specified, 6379 is used

servers = ["tcp://localhost:6379"]

```

## RabbitMQ Monitoring

*Step: 1. configure telegraf configuration (telegraf.conf) for RabbitMQ*

*step: 2. restart telegraf service*

## ***Telegraf conf for Rabbitmq input metrics***

```
[[inputs.rabbitmq]]
# url = "http://localhost:15672"
## Credentials
# username = "guest"
# password = "guest"

## A list of queues to gather as the rabbitmq_queue measurement. If not
## specified, metrics for all queues are gathered.
# queues = ["telegraf"]

## A list of exchanges to gather as the rabbitmq_exchange measurement. If not
## specified, metrics for all exchanges are gathered.
# exchanges = ["telegraf"]
```

## **Aerospike Monitoring**

*aerospike plugin queries aerospike server(s) and get node statistics & stats for all the configured namespaces*

*step: 1. configure telegraf configuration (telegraf.conf )for aerospike*

*step: 2. restart telegraf service*

## ***Telegraf conf for Aerospike input metrics***

```
[[inputs.aerospike]]
## Aerospike servers to connect to (with port)
servers = ["localhost:3000"]

# username = "telegraf"
# password = "pa$$word"
```

## **ElasticSearch Monitoring**

*step: 1. configure telegraf configuration (telegraf.conf )for elasticsearch*

*step: 2. restart telegraf service*

## ***Telegraf conf for elasticsearch input metrics***

```
[[inputs.elasticsearch]]
  servers = ["http://localhost:9200"]
  http_timeout = "5s"
  local = true
  cluster_health = true
  cluster_stats = true
  cluster_stats_only_from_master = true
  indices_include = ["_all"]
  indices_level = "shards"
  node_stats = ["jvm", "http"]
```

## **MongoDB Monitoring**

*step: 1. configure telegraf configuration (telegraf.conf) for mongodb*

*step: 2. restart telegraf service*

## ***Telegraf conf for MongoDB input metrics***

```
[[inputs.mongodb]]
  ## "mongodb://" [user ":" pass "@"] host [ ":" port]
  ## For example:
  ## mongodb://user:auth_key@10.10.3.30:27017,
  servers = ["mongodb://127.0.0.1:27017"]
  gather_perdb_stats = true
```

## **MySQL Monitoring**

*step: 1. configure telegraf configuration (telegraf.conf) for mysql*

*step: 2. restart telegraf service*

## ***Telegraf conf for Mysql input metrics***

```
[[inputs.mysql]]
  ## [username[:password]@][protocol[(address)]]/?tls=[true|false|skip-verify]]
  ## servers = ["user:passwd@tcp(127.0.0.1:3306)/?tls=false"]
  ## servers = ["user@tcp(127.0.0.1:3306)/?tls=false"]
  #
  ## If no servers are specified, then localhost is used as the host.
  servers = ["tcp(127.0.0.1:3306)"]
```

# Prometheus Monitoring

*step: 1. configure telegraf configuration (telegraf.conf) for prometheus*

*step: 2. restart telegraf service*

## ***Telegraf conf for prometheus input metrics***

```
[[inputs.prometheus]]
  urls = ["http://localhost:9101/metrics"]
```

# Cassandra Monitoring

*Step: 1. download + configure jolokia agent jar for cassandra and restart cassandra (as mentioned under jolokia doc)*

*step: 2. configure telegraf configuration for cassandra input metrics.*

*Step: 3 restart telegraf service.*

## ***Telegraf conf for cassandra input metrics***

```
[[inputs.jolokia2_agent]]
  urls = ["http://127.0.0.1:8778/jolokia"]

[inputs.jolokia2_agent.tags]
  app = "cassandra_digital_internal_dc"

[[inputs.jolokia2_agent.metric]]
  name = "java_threading"
  mbean = "java.lang:type=Threading"
  paths =
["ThreadCount", "PeakThreadCount", "DaemonThreadCount", "TotalStartedThreadCo
unt"]

[[inputs.jolokia2_agent.metric]]
  name = "java_memory_usage"
  mbean = "java.lang:type=Memory"
  paths = ["HeapMemoryUsage", "NonHeapMemoryUsage"]

[[inputs.jolokia2_agent.metric]]
  name = "java_memory_pool_usage"
  mbean = "java.lang:name=*,type=MemoryPool"
  paths = ["Usage", "CollectionUsage"]
```

```
tag_keys = ["name"]
```

```
[[inputs.jolokia2_agent.metric]]
```

```
name = "cassandra_tables_matrix_MemtableOnHeapSize"
```

```
mbean = "org.apache.cassandra.metrics:type=Table,name=MemtableOnHeapSize"
```

```
paths = ["Value"]
```

```
tag_keys = ["name"]
```

```
[[inputs.jolokia2_agent.metric]]
```

```
name = "cassandra_tables_matrix_MemtableOffHeapSize"
```

```
mbean = "org.apache.cassandra.metrics:type=Table,name=MemtableOffHeapSize"
```

```
paths = ["Value"]
```

```
tag_keys = ["name"]
```

```
[[inputs.jolokia2_agent.metric]]
```

```
name = "cassandra_tables_matrix_MemtableLiveDataSize"
```

```
mbean = "org.apache.cassandra.metrics:type=Table,name=MemtableLiveDataSize"
```

```
paths = ["Value"]
```

```
tag_keys = ["name"]
```

```
[[inputs.jolokia2_agent.metric]]
```

```
name = "cassandra_tables_matrix_AllMemtablesOnHeapSize"
```

```
mbean =
```

```
"org.apache.cassandra.metrics:type=Table,name=AllMemtablesOnHeapSize"
```

```
paths = ["Value"]
```

```
tag_keys = ["name"]
```

```
[[inputs.jolokia2_agent.metric]]
```

```
name = "cassandra_tables_matrix_AllMemtablesLiveDataSize"
```

```
mbean =
```

```
"org.apache.cassandra.metrics:type=Table,name=AllMemtablesLiveDataSize"
```

```
paths = ["Value"]
```

```
tag_keys = ["name"]
```

```
[[inputs.jolokia2_agent.metric]]
```

```
name = "cassandra_tables_matrix_MemtableColumnsCount"
```

```
mbean =
```

```
"org.apache.cassandra.metrics:type=Table,name=MemtableColumnsCount"
```

```
paths = ["Value"]
```

```
tag_keys = ["name"]
```

```
[[inputs.jolokia2_agent.metric]]  
name = "cassandra_tables_matrix_ReadLatency"  
mbean = "org.apache.cassandra.metrics:type=Table,name=ReadLatency"  
paths = ["MeanRate","FiveMinuteRate"]  
tag_keys = ["name"]
```

```
[[inputs.jolokia2_agent.metric]]  
name = "cassandra_tables_matrix_WriteLatency"  
mbean = "org.apache.cassandra.metrics:type=Table,name=WriteLatency"  
paths = ["MeanRate","FiveMinuteRate"]  
tag_keys = ["name"]
```

```
[[inputs.jolokia2_agent.metric]]  
name = "cassandra_ThreadPools_matrix_ActiveTasks"  
mbean =  
"org.apache.cassandra.metrics:type=ThreadPools,path=*,scope=*,name=ActiveTasks"  
paths = ["Value"]  
tag_keys = ["name"]
```

```
[[inputs.jolokia2_agent.metric]]  
name = "cassandra_ThreadPools_matrix_PendingTasks"  
mbean =  
"org.apache.cassandra.metrics:type=ThreadPools,path=*,scope=*,name=PendingTasks"  
paths = ["Value"]  
tag_keys = ["name"]
```

```
[[inputs.jolokia2_agent.metric]]  
name = "cassandra_ThreadPools_matrix_CompletedTasks"  
mbean =  
"org.apache.cassandra.metrics:type=ThreadPools,path=*,scope=*,name=CompletedTasks"  
paths = ["Value"]  
tag_keys = ["name"]
```

```
[[inputs.jolokia2_agent.metric]]
name = "cassandra_ThreadPools_matrix_TotalBlockedTasks"
mbean =
"org.apache.cassandra.metrics:type=ThreadPools,path=*,scope=*,name=TotalBlockedTasks"
paths = ["Value"]
tag_keys = ["name"]
```

```
[[inputs.jolokia2_agent.metric]]
name = "cassandra_ThreadPools_matrix_CurrentlyBlockedTask"
mbean =
"org.apache.cassandra.metrics:type=ThreadPools,path=*,scope=*,name=CurrentlyBlockedTask"
paths = ["Count"]
tag_keys = ["name"]
```

```
[[inputs.jolokia2_agent.metric]]
name = "cassandra_Storage_Load"
mbean = "org.apache.cassandra.metrics:type=Storage,name=Load"
paths = ["Count"]
tag_keys = ["name"]
```

```
[[inputs.jolokia2_agent.metric]]
name = "cassandra_Storage_TotalHintsInProgress"
mbean = "org.apache.cassandra.metrics:type=Storage,name=TotalHintsInProgress"
paths = ["Count"]
tag_keys = ["name"]
```

```
[[inputs.jolokia2_agent.metric]]
name = "cassandra_Storage_Exceptions"
mbean = "org.apache.cassandra.metrics:type=Storage,name=Exceptions"
paths = ["Count"]
tag_keys = ["name"]
```

```
[[inputs.jolokia2_agent.metric]]
name = "cassandra_ClientRequest_Latency"
mbean =
"org.apache.cassandra.metrics:type=ClientRequest,name=Latency,scope=*"
paths = ["OneMinuteRate", "Count"]
```



```
tag_keys = ["scope"]
```

```
[[inputs.jolokia2_agent.metric]]
```

```
name = "cassandra_Client_connectedNativeClients"
```

```
mbean = "org.apache.cassandra.metrics:type=Client,name=connectedNativeClients"
```

```
paths = ["Value"]
```

```
tag_keys = ["name"]
```

```
[[inputs.jolokia2_agent.metric]]
```

```
name = "cassandra_CommitLog_PendingTasks"
```

```
mbean = "org.apache.cassandra.metrics:type=CommitLog,name=PendingTasks"
```

```
paths = ["Value"]
```

```
tag_keys = ["name"]
```

```
[[inputs.jolokia2_agent.metric]]
```

```
name = "cassandra_CommitLog_TotalCommitLogSize"
```

```
mbean =
```

```
"org.apache.cassandra.metrics:type=CommitLog,name=TotalCommitLogSize"
```

```
paths = ["Value"]
```

```
tag_keys = ["name"]
```

```
[[inputs.jolokia2_agent.metric]]
```

```
name = "cassandra_ClientRequest_Timeouts"
```

```
mbean =
```

```
"org.apache.cassandra.metrics:type=ClientRequest,name=Timeouts,scope=*"
```

```
paths = ["OneMinuteRate","Count"]
```

```
tag_keys = ["scope"]
```

```
[[inputs.jolokia2_agent.metric]]
```

```
name = "cassandra_Compaction_BytesCompacted"
```

```
mbean = "org.apache.cassandra.metrics:type=Compaction,name=BytesCompacted"
```

```
paths = ["Count"]
```

```
tag_keys = ["name"]
```

```
[[inputs.jolokia2_agent.metric]]
```

```
name = "cassandra_ColumnFamily_WriteLatency"
```

```
mbean = "org.apache.cassandra.metrics:type=ColumnFamily,name=WriteLatency"
```

```
paths = ["Count","OneMinuteRate"]
```

```
tag_keys = ["name"]
```

```
[[inputs.jolokia2_agent.metric]]
```

```
name = "cassandra_ColumnFamily_ReadLatency"
mbean = "org.apache.cassandra.metrics:type=ColumnFamily,name=ReadLatency"
paths = ["Count","OneMinuteRate"]
tag_keys = ["name"]
```

```
[[inputs.jolokia2_agent.metric]]
name = "cassandra_ColumnFamily_RangeLatency"
mbean = "org.apache.cassandra.metrics:type=ColumnFamily,name=RangeLatency"
paths = ["Count","OneMinuteRate"]
tag_keys = ["name"]
```

```
[[inputs.jolokia2_agent.metric]]
name = "cassandra_ColumnFamily_AllMemtablesLiveDataSize"
mbean = "org.apache.cassandra.metrics:type=ColumnFamily,scope=*,
name=AllMemtablesLiveDataSize,keyspace=*"
paths = ["Count"]
tag_keys = ["scope"]
```

# PORT +PING +URL Status via heartbeat/Shellscript

## Port +Health check URL Monitoring

### Purpose of Port/ping/url Monitoring:

1. port monitoring : whether your services/application port are up or not. One way to monitor application status is to check default or defined port status.
2. ping monitoring : to check connectivity and packet percentage loss (data loss in servers communication )
3. URL monitoring : if mentioned url is up or not. If application url is reachable or not.

### Monitoring Architecture

*In our monitoring architecture we use Heartbeat-elastic for this port/Ping + Url monitoring.*

**Heartbeat asks the simple question: Are you alive?** Heartbeat ships this information and response time to the rest of the Elastic Stack for further analysis.

*Heartbeat tells you whether your services are reachable.*

*Heartbeat currently supports monitors for checking hosts via:*

- **ICMP (v4 and v6) Echo Requests.** Use the icmp monitor when you simply want to check whether a service is available. This monitor requires root access.
- **TCP.** Use the tcp monitor to connect via TCP. You can optionally configure this monitor to verify the endpoint by sending and/or receiving a custom payload.
- **HTTP.** Use the http monitor to connect via HTTP. You can optionally configure this monitor to verify that the service returns the expected response, such as a specific status code, response header, or content.

**Exception: heartbeat can't monitor chunked url. For that we use shell script using curl.**

# Monitoring Via Heartbeat:

## Heartbeat Installation:

*Step 1: Download latest rpm version of heartbeat for RHEL servers (almost everywhere we have RHEL linux distribution).*

*Download RPM :*

```
wget https://artifacts.elastic.co/downloads/beats/heartbeat/heartbeat-7.14.1-x86\_64.rpm
```

*Install RPM:*

```
rpm -i heartbeat-7.14.1-x86_64.rpm
```

*Start Service:*

```
systemctl start heartbeat-elastic
```

*Check status :*

```
systemctl status heartbeat-elastic
```

*Test command file*

```
heartbeat -c testFileName.yml
```

## Heartbeat configuration:

Default path for heartbeat.yml ==> /etc/heartbeat/heartbeat.yml

we make all necessary changes for monitoring in /etc/heartbeat directory.

## Steps for PORT,PING,URL Heartbeat Setup

Heartbeat define a set of monitors to check your remote hosts.

Specify monitors either directly inside the heartbeat.yml config file, or in external dynamically loaded files located in the directory **/etc/heartbeat/monitor.d**

**In our environment, we make a file loaded with all monitoring information (i.e for port: IP+PORT, for Ping : domain or server IP, for URL : url ) in monitor.d/ directory.**

Step: 1.) make a file in monitor.d directory (For example : telnetmonitoring.yml)

Step: 2.) define that file path in heartbeat.yml file

```
(heartbeat.config.monitors:
```

```
  path: "/etc/heartbeat/monitors.d/*.yaml")
```

Step: 3.) run file manually to check for any syntax error (`heartbeat -c testFileName.yml` )

Step: 4.) restart heartbeat service --> systemctl restart heartbeat

**step: 1.) make a file in monitor.d directoy (For example : telnetmonitoring.yml)**

*for telnet/port Monitoing: vim /etc/heartbeat/moniter.d/telnet.yml*

```
- type: tcp
name: Telnet
hosts: ["application server ip1","application server ip 2"]
ports: [application port]
schedule: '@every 60s'
fields:
  Module: applicationModuleName/MicroserviceName
  Application: applicationName
  SourceIP: heartbeatServerIP
ipv4: true
ipv6: true
mode: any
timeout: 57s
```

*For ping monitoring : vim /etc/heartbeat/moniter.d/ping.yml*

```
-type: icmp
name: Ping
hosts: ["127.0.0.1"]
Schedule: '@every 60s'
fields:
  Module: applicationModuleName/MicroserviceName
  Application: applicationName
  SourceIP: heartbeatServerIP
Ipv4: true
Ipv6: false
mode: any
```

*For Curl/url monitoring : vim /etc/heartbeat/moniter.d/curl.yml*

```
- type: http
name: Curl
urls: ["https://sampleUrl"]
schedule: '@every 60s'
mode: any
ipv4: true
ipv6: false
ssl.verification_mode: none
fields:
```

```
Module: applicationModuleName/MicroserviceName
Application: applicationName
Source: heartbeatServerIP
```

*Make sure you will give you give enough file permissions to read and execute above files.*

***chmod +x \*.yml in monitor.d directory***

**Step: 2.) define that file path in heartbeat.yml file**

**heartbeat.yml conf**

```
im heartbeat.yml
heartbeat.config.monitors:
  path: "/etc/heartbeat/monitors.d/*.yml"
  reload.enabled: true
  reload.period: 30s

setup.template.settings:
  index.number_of_shards: 1
  index.codec: best_compression

output.elasticsearch:
  Hosts: [127.0.0.1:9200]
  Username: "elastic"
  Password: "elastic"

logging.level: info
logging.to_files: true
logging.files:
  path: /var/log/heartbeatlog
  name: heartbeat
  keepfiles: 7
  permissions: 0644
```

**Step: 3 and step 4 are self explanatory. You just need to check your configuration and restart heartbeat service with the command mentioned in installation part.**

### ***Heartbeat output setup***

In our working environment, we mostly use logstash output. And for debugging purpose ,use console output (output.console).

### Elasticsearch output:

for secure elasticsearch cluster we need to provide authentication (if authentication is not there you can remove user and password part)

```
output.elasticsearch:
  hosts: ["https://myEShost:9200"]
  username: "heartbeat_writer"
  password: "YOUR_PASSWORD"
```

With api key authentication

```
output.elasticsearch:
  hosts: ["https://myEShost:9200"]
  api_key: "ZCV7VnwBgnX0T19fN8Qe:KnR6yE41RrSowb0kQ0HWoA"
```

With PKI authentication (Public Key Infrastructure (PKI) certificates.)

```
output.elasticsearch:
  hosts: ["https://myEShost:9200"]
  ssl.certificate: "/etc/pki/client/cert.pem"
  ssl.key: "/etc/pki/client/cert.key"
```

### Logstash Output:

```
output.logstash:
  hosts: ["127.0.0.1:5044"]
```

### Redis OutPut:

Password will be there in conf if we have authentication on Redis

```
output.redis:
  hosts: ["localhost"]
  password: "my_password"
  key: "heartbeat"
  db: 0
  timeout: 5
```

### File Output:

```
output.file:
  path: "/tmp/heartbeat"
  filename: heartbeat
  #rotate_every_kb: 10000
  #number_of_files: 7
  #permissions: 0600
  #rotate_on_startup: true
```

## Console Output:

```
output.console:  
pretty: true
```

## Monitoring Via Shell Script:

1. make a file containing url which need to monitored
2. mention that part to script in UriList

```
#!/bin/bash  
date=$( date +%s)  
UriList=/home/arcos_root/PROXY_URL_MONITORING/fsUrlMonitoring  
sourceServer=`hostname -i`  
module="FinancialService"  
for url in $(cat $UriList); do  
    output=$(curl -o /dev/null --no-proxy '*' -s -w '%{http_code}', '%{time_total}', '%  
{url_effective}'\n' ${url} ;)  
    response_code=$(echo "$output" | cut -d "," -f1)  
    response_time=$(echo "$output" | cut -d "," -f2)  
    healthCheckUrl=$(echo "$output" | cut -d "," -f3)  
    if [ $response_code < 400 ]  
    then  
        resultCode=1i  
        Status="up"  
  
    else  
        resultCode=0i  
        Status="down"  
    fi  
    (curl --no-proxy '*' -XPOST "http://10.14.162.49:8086/write?db=heartbeatDb" --data-binary  
"http,application=FinancialService,module=$module,monitoringType=http,sourceServer=$source  
Server,status=$Status,healthCheckUrl=$healthCheckUrl  
resultCode=$resultCode,statusCode=$response_code,responseTime=$response_time")  
done
```



# Grafana Screenshot on telegram

## grafana screenshot by grafana-image-renderer plugin

**step: 1)** check for grafana “grafana-image-renderer” plugin (in airtel env we have already this plugin so no need to install. )

either check it from grafana UI or grafana plugin path on server

**grafana ->configuration ->plugins -> grafana-image-renderer**

**cd /var/lib/grafana/plugin**

if plugin is not there install plugin and give ownership grafana:grafana

**cd /var/lib/grafana/plugin (on grafana server) and run following command to install plugin online**

**grafana-cli plugins install grafana-image-renderer**

or download it manually from

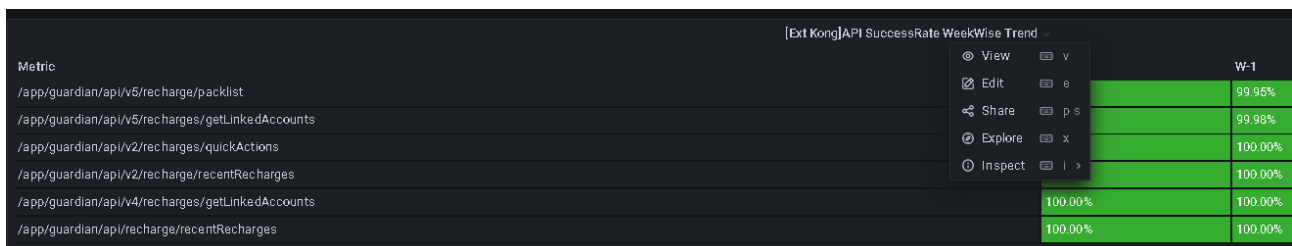
[https://grafana.com/api/plugins/grafana-image-renderer/versions/3.6.1/download?](https://grafana.com/api/plugins/grafana-image-renderer/versions/3.6.1/download?os=linux&arch=amd64)

**os=linux&arch=amd64**

and unzip plugin and move plugin to /var/lib/grafana/plugin

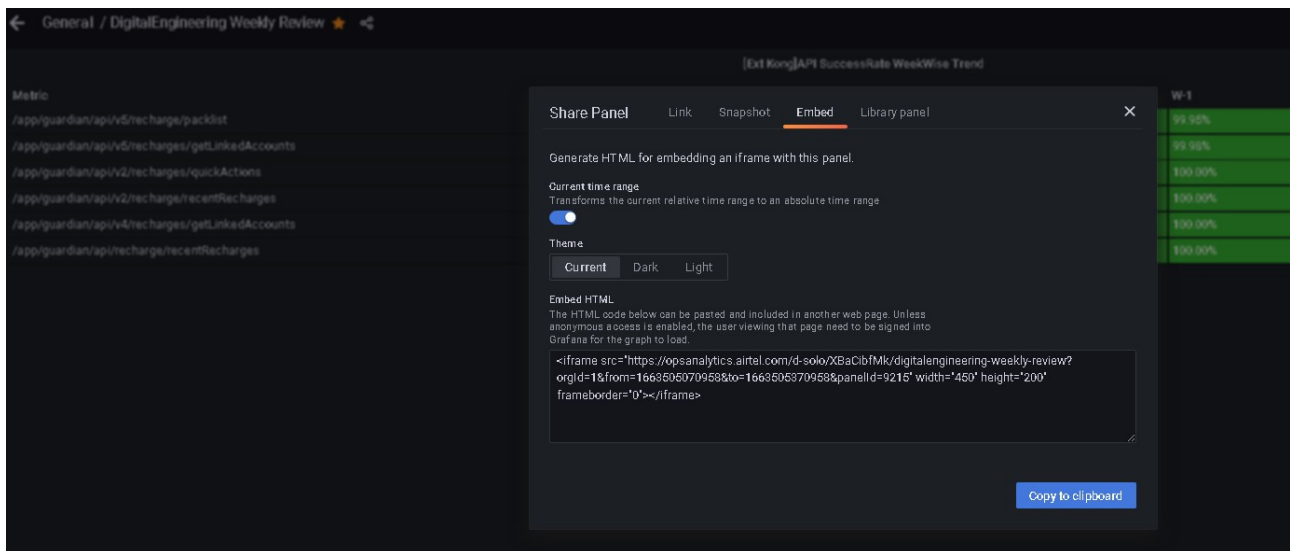
**chown -R grafana:grafana grafana-image-renderer**

**step: 2)** go to particular pannel and click for options as shown in image



Metric	SuccessRate	WeekWise	Trend	W-1
/app/guardian/api/v5/recharge/packlist				99.95%
/app/guardian/api/v5/recharges/getLinkedAccounts				99.98%
/app/guardian/api/v2/recharges/quickActions				100.00%
/app/guardian/api/v2/recharge/recentRecharges				100.00%
/app/guardian/api/v4/recharges/getLinkedAccounts	100.00%			100.00%
/app/guardian/api/recharge/recentRecharges	100.00%			100.00%

**step: 3) click on share and go Embed and copy link given in iframe tag and paste this link to give shell script**



```
#!/bin/sh
export http_proxy="use proxy"
export https_proxy="use proxy"
export no_proxy="opsanalytics.airtel.com"
prev=`date --date="yesterday" +"%b %d"`

rm -f /opt/grafanaSnaps/application;

curl -u "admin:Airtel@123" "http://opsanalytics.airtel.com/render/d-solo/Jo6qCfynz/digital-alert-
dashboard?orgId=1&width=1100&height=575&panelId=8&from=now-1d%2Fd&to=now-1d
%2Fd" > /opt/grafanaSnaps/application;

curl --silent -k -XPOST
"https://api.telegram.org/bot1113415807:AAHf42vGZEZLbTUBDepM7qQYIpyTEwxdyys/
sendPhoto" -F "chat_id=-1001229485850" -F "caption=caption can be anything but relevant
($prev)" -F "photo=@/opt/grafanaSnaps/application";
```