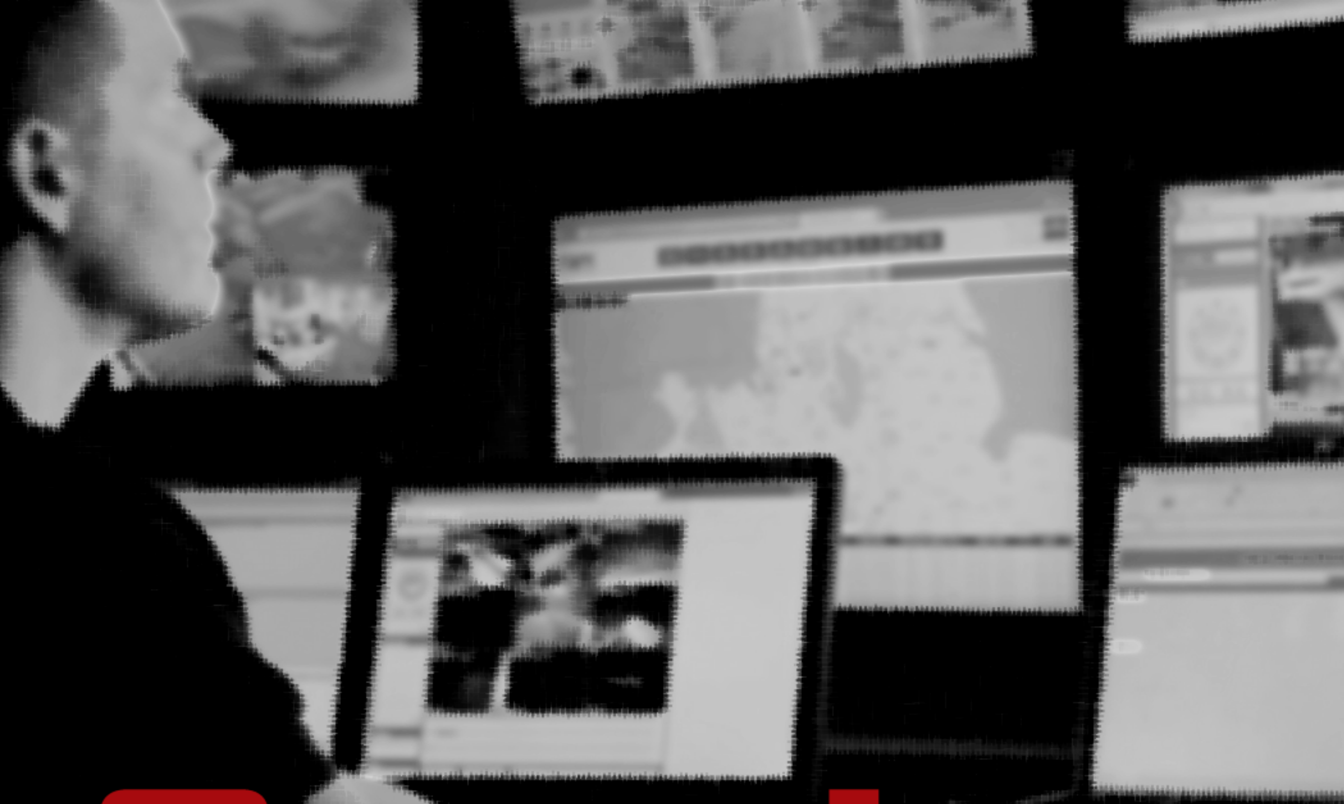


Graylog

Log Management with Graylog, Elasticsearch, MongoDB, Nginx, Fluentd, Vagrant and Docker

JORGE ACETOZI

Graylog

Log Management with Graylog, Elasticsearch, MongoDB, Nginx, Fluentd, Vagrant and Docker

Jorge Acetozi

This book is for sale at http://leanpub.com/graylog

This version was published on 2018-03-25

Tweet This Book!

Please help Jorge Acetozi by spreading the word about this book on Twitter!

The suggested tweet for this book is:

I just bought Graylog Book

The suggested hashtag for this book is #graylogbook.

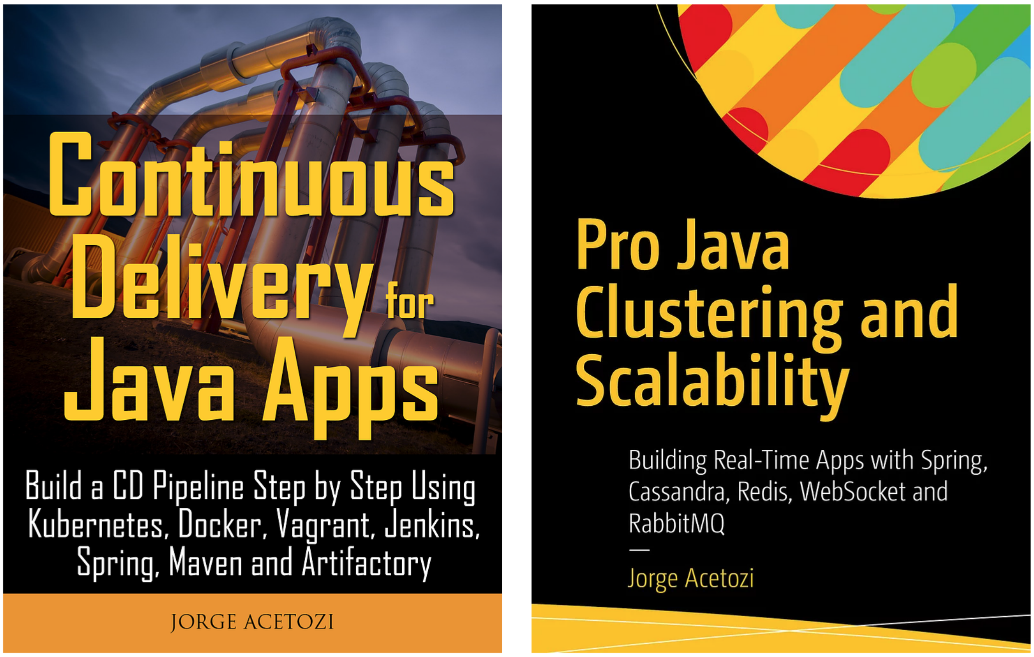
Find out what other people are saying about the book by clicking on this link to search for this hashtag on Twitter:

#graylogbook

Contents

About the Author

Jorge Acetozi is a software engineer who spends almost his whole day having fun with things such as AWS, Kubernetes, Docker, Terraform, Ansible, Cassandra, Redis, Elasticsearch, Graylog, New Relic, Sensu, Elastic Stack, Fluentd, RabbitMQ, Kafka, Java, Spring, and much more! He loves deploying applications in production while thousands of users are online, monitoring the infrastructure, and acting quickly when monitoring tools decide to challenge his heart's health!


Continuous Delivery for Java Apps
Build a CD Pipeline Step by Step Using Kubernetes, Docker, Vagrant, Jenkins, Spring, Maven and Artifactory
JORGE ACETOZI
Pro Java Clustering and Scalability
Building Real-Time Apps with Spring, Cassandra, Redis, WebSocket and RabbitMQ
Jorge Acetozi

Author's Books

You can reach him at:

- Web site[1]

[1] https://www.jorgeacetozi.com

- This Book's web site[2]
- GitHub[3]
- LinkedIn[4]
- Facebook[5]
- Twitter[6]
- Medium[7]

[2]https://www.graylogbook.com
[3]https://github.com/jorgeacetozi
[4]https://www.linkedin.com/in/jorgeacetozi
[5]https://www.facebook.com/jorgeacetozi
[6]https://twitter.com/jorgeacetozi
[7]https://medium.com/jorgeacetozi

Introduction

Development is Development. Production is Production. Typically, in order to deploy a new feature to production, the new code written is manually tested (by developers and sometimes a QA team) and automated tested (unit, integration, acceptance, performance, smoke, security tests) against several different environments like `development` and `staging` before eventually landing on `production`. Basically, all this process is done to ensure that when the `release candidate` is ready to be deployed to production, the team has a significant level of confidence that the code actually works as intended.

Well, although I wrote a 600-pages book called Continuous Delivery for Java Apps: Build a CD Pipeline Step by Step Using Kubernetes, Docker, Vagrant, Jenkins, Spring, Maven and Artifactory[8] guiding the reader on how to implement all these steps and much more in practice using top-notch technologies and deployment strategies like Canary Release, I have a little secret to confess: most of the steps of a continuous delivery pipeline actually take place in controlled environments (internal networks, predictable traffic, and so on), and as I mentioned at the beginning of this section: development is development and production is production. Many things can (and I bet they will) go wrong in production, especially if your application is accessible on the Internet. **When something crashes, guess who is your best friend? That's right: Logs**.

[8]https://leanpub.com/continuous-delivery-for-java-apps

Why do I need a Centralized Logging System Like Graylog?

Suppose that your application is designed on top of Microservices architecture[9] and a particular request goes through service A, B, and C. Each of these services run a load balancer that spreads requests among 10 different servers running application servers and reverse proxies (Nginx, for example). Also, these services use PostgreSQL and Redis instances running on Amazon RDS[10] and Amazon Elasticache[11] respectively.

Now answer me, how can you keep track of what's going on when something goes wrong in this particular scenario? Are you going to `ssh` 10 production servers from service A, 10 production servers from service B, 10 production servers from service C, and for each of them use commands like `tail -f logfile | grep "ERROR"` for the Nginx `access.log` and `error.log` as well as the application server logs? Don't forget that on top of that you would have to monitor the AWS logs for PostgreSQL and Redis as well.

Well, I think you got the idea. The bottom line is that the traditional way of visualizing logs (analyzing specific log files stored on the disk) doesn't scale at all. In fact, there are many other problems related to this traditional approach. Let's list out some of them:

- **Security issues**: you have to `ssh` production servers every time you want to see the logs, and as you know, some angry system administrators that fight on UFC for fun don't like that for security reasons, especially if you are not an infrastructure expert. As a result, you can get very hurt;
- **Logs are streams, not files**: you cannot assume that logs are being directly stored into files on the disk for cloud-native applications that follows the 12-Factor logging[12] practices. Instead, the only guarantee you have in the cloud

[9] https://www.nginx.com/blog/introduction-to-microservices/
[10] https://aws.amazon.com/rds/
[11] https://aws.amazon.com/elasticache/
[12] https://12factor.net/logs

is that the disk is ephemeral. Later on this book we will learn more about 12-Factor logging practices and this will become clearer;
- **Filtered content**: some log entries might contain sensitive information and must be hidden while others don't. How to control that and avoid that someone that is directly logged into the server could see the sensitive log entries?
- **Inefficiency**: when something goes wrong, the time spent on logging into many machines and `greping` log files one by one is probably the time that would take to query Graylog, figure out what's wrong and start working on the fix.

Now, let's list out some benefits you instantaneously get when you use Graylog as your centralized logging system:

- **Increased Security**: people don't need to have access to the production servers as the log visualization is displayed on the beautiful Graylog Web Interface through their preferred **modern** browser. Keep in mind that if you try to use something like Internet Explorer 6, your problem is not logs, my friend!;
- **Authentication and Authorization**: it's possible to manage users and roles very easily. Actually, it's even possible to integrate it with your existing LDAP solution (if you have one);
- **Powerful Querying Capabilities**: you can use a beautiful and powerful search syntax very close to the Apache Lucene syntax rather than relying on insanely ugly regular expressions;
- **Built-in Alerting**: Graylog has a build-in alerting mechanism, so you don't need to be all the time worried about your applications behavior and suffering with nightmares at night. Instead, you can define certain patterns that would trigger an alert and then get alerted on your mobile with a graceful message like this one: "Sorry to bother you, Mr. Jorge Acetozi, but it looks like someone attempted to `ssh` your production server (IP: XXX.XXX.XXX.XXX) for 5 consecutive times in less than a minute, so I was wondering here if you could take a look at it. By the way, your coffee is ready and waiting for you on your desk. Thank you very much for your time!";
- **Input Everything**: you don't have to restrict the centralized logging system to your application logs only. You can input every type of logs in it, such as the `sshd` daemon in the alerting example before, operating system logs, firewall logs… Well, pretty much everything!;

- **Custom Dashboards**: create custom dashboards and display them on televisions spread around your company's office;
- **Geolocation**: add geolocation metadata to log specific log entries (such as Nginx access.log) so that you can visualize where the requests are coming from;
- **Reports**: create reports that could even be useful on Marketing decisions, such as the more frequently accessed pages.

Well, and the list goes on!

However, everything comes at a price. Maintaining a production infrastructure of a centralized logging system like Graylog is neither easy nor cheap. As far as Graylog is concerned, it relies on different tools such as Elasticsearch[13] and MongoDB[14] to get the work done. In production, such tools should be deployed as clusters to allow high-availability, performance, and scalability, which of course leads to a increased complexity of the overall solution. Hopefully, this book will provide a good understanding of each technology involved in Graylog's architecture in a hands-on (and not boring) fashion so that you can get started with Graylog as soon as possible.

[13] https://www.elastic.co/
[14] https://www.mongodb.com/

Hands-on Introduction to Graylog

Graylog is a powerful open-source log management platform written in Java[15] that ingests logs from various sources and allows to search and visualize the logs in a beautiful web interface. It provides many useful built-in features like user management, alerting, custom dashboards, amazing querying capabilities, a REST API and others (most of them are covered in this book), which makes it ideal as the logging solution for your company (whether it's a small or huge one).

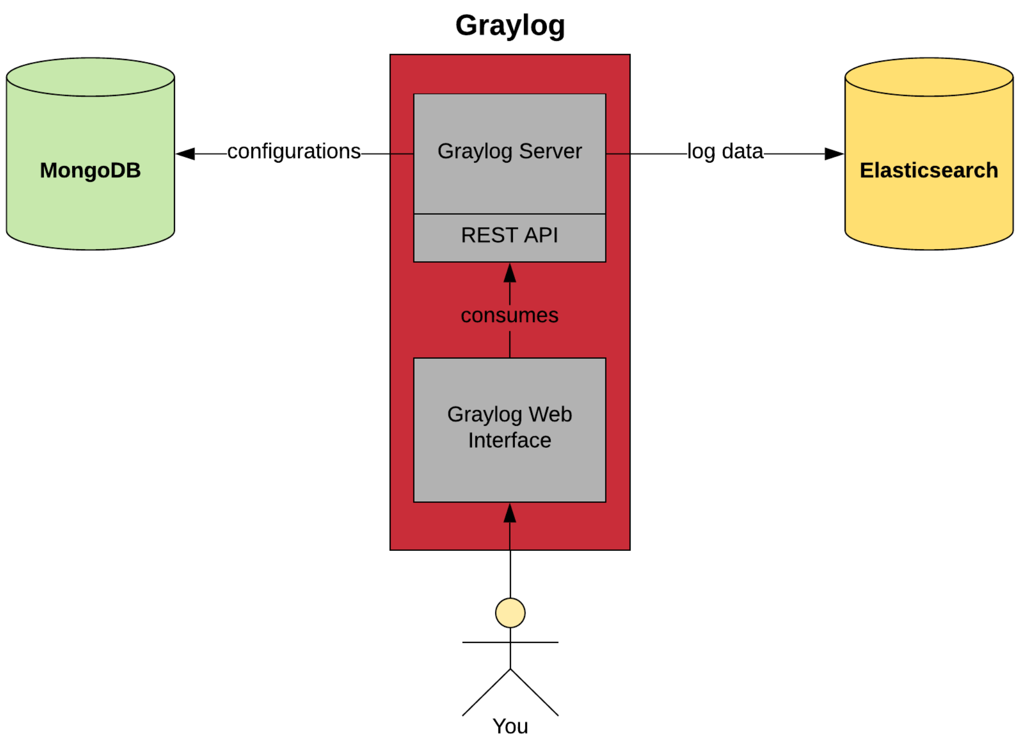
Graylog is built on top of a master-slave architecture, which means that slaves can be added for horizontal scalability. As we are going to learn later on this book, clustering Graylog is pretty straightforward.

Horizontal scalability happens when you add more nodes to your cluster of machines. Vertical scalability happens when you increase a machine's hardware power.

[15] https://github.com/Graylog2/graylog2-server

Graylog Components

Basically, Graylog stores the log data in Elasticsearch, which is a powerful open-source, RESTful, distributed search and analytics engine built on top of Apache Lucene. Besides, Graylog also uses MongoDB, which is a document-oriented NoSQL database[16], to store configuration data such as user information, inputs and streams configurations, and so on.



Graylog Components

So, as you can see, getting started with Graylog is as simple as setting up an instance for Elasticsearch, MongoDB, and Graylog itself. Basically, a Graylog instance is composed by two components:

[16]https://en.wikipedia.org/wiki/Document-oriented_database

- **Graylog Web Interface**: offers capabilities for searching and analyzing the indexed log data and allows configuring the Graylog environment (inputs, dashboards, etc). It communicates with the Graylog Server through the Graylog REST API;
- **Graylog Server**: encapsulates all the logic related to Graylog (a fancy way to say "does all the hard work") and provides the Graylog REST API.

Note that none of the log messages are ever stored in MongoDB. Thus, MongoDB does not have a big system impact, and you won't have to worry too much about scaling it even though Graylog is ingesting thousands of log messages per second. When it comes to scaling the system, the challenges typically involve scaling Graylog and Elasticsearch.

In this section, we are not going to worry about high-availability, scalability, performance, and others non-functional requirements (we will have the whole book to have fun and dive into these exciting subjects). Instead, it will provide a hands-on introduction to Graylog, showcasing some of its handy features so that you can actually understand in practice the concept of Centralized Logging System that we have discussed earlier. So, let's get our hands dirty!

Set up Graylog using Vagrant and Docker

Now that you've got the `ebook-graylog` directory on your machine, go into the `vagrant` subdirectory and check its structure:

```
$ cd vagrant
$ tree
├── Vagrantfile-fluentd-high-availability
├── Vagrantfile-fluentd-single-instance
├── Vagrantfile-hello-world
├── Vagrantfile-without-fluentd
├── Vagrantfile-without-fluentd-nginx-contentpacks
└── conf
    ├── fluentd
    │   ├── aggregator
    │   │   └── fluentd-aggregator.conf
    │   ├── forwarder
    │   │   └── fluentd-forwarder.conf
    │   └── single-instance
    │       └── fluentd-single-instance.conf
    ├── graylog
    │   ├── graylog-server-master
    │   │   ├── graylog-contentpacks.conf
    │   │   └── graylog.conf
    │   ├── graylog-server-slave
    │   │   ├── graylog-contentpacks.conf
    │   │   └── graylog.conf
    │   └── nginx
    │       ├── nginx-with-fluentd.conf
    │       ├── nginx-without-fluentd-contentpacks.conf
    │       └── nginx-without-fluentd.conf
    └── log-generator-app
        └── nginx
            ├── nginx-fluentd-forwarder-aggregator.conf
            ├── nginx-fluentd-single-instance.conf
            └── nginx-without-fluentd-contentpacks.conf
```

As you can see, there are five `Vagrantfiles` that we are going to use along this book. The idea is to start very simple and evolve our architecture according to the problems presented until we get to a really beautiful solution that is pretty much what you would be doing in production.

⚠ Note that in this book we are using Vagrant to orchestrate **local** virtual machines and Docker containers. So, obviously, this is not a production environment. However, the logging architecture we are going to implement and evolve throughout this book is pretty much what successful companies are doing in production, except that instead of local virtual machines, they run it on the cloud (most of them). Note that the concepts still exactly the same.

For this introductory example, we are going to use the `Vagrantfile-hello-world` as our Vagrantfile. By default, Vagrant will look for a file named `Vagrantfile` in the current directory, so let's instruct it to look for the file `Vagrantfile-hello-world` by setting up the `VAGRANT_VAGRANTFILE` environment variable instead:

```
$ export VAGRANT_VAGRANTFILE=Vagrantfile-hello-world
```

Let's check the contents of this file:

```
# -*- mode: ruby -*-
# vi: set ft=ruby :

$script = <<SCRIPT
sysctl -w vm.max_map_count=262144
SCRIPT

Vagrant.configure("2") do |config|
  config.vm.box = "ubuntu/trusty64"

  config.vm.define "vm_hello_world" do |helloworld|
    helloworld.vm.hostname = "helloworld"
    helloworld.vm.network "private_network", ip: "10.0.0.10"
    helloworld.vm.provider "virtualbox" do |vb|
      vb.memory = "4096"
    end

    helloworld.vm.provision "shell", inline: $script

    helloworld.vm.provision "docker" do |d|
      d.run "mongo",
        image: "mongo"
    end

    helloworld.vm.provision "docker" do |d|
      d.run "elasticsearch",
        image: "docker.elastic.co/elasticsearch/elasticsearch:5.6.8",
        args: "-e 'xpack.security.enabled=false'"
    end

    helloworld.vm.provision "docker" do |d|
      d.run "graylog",
        image: "graylog/graylog:2.4.3-1",
        args: "--link mongo \
               --link elasticsearch \
               -p 9000:9000 -p 12201:12201 -p 514:514 -p 5555:5555 \
               -e 'GRAYLOG_WEB_ENDPOINT_URI=http://10.0.0.10:9000/api'"

#               -e 'GRAYLOG_TRANSPORT_EMAIL_ENABLED=true' \
#               -e 'GRAYLOG_TRANSPORT_EMAIL_HOSTNAME=AWS_SMTP_SERVER' \
#               -e 'GRAYLOG_TRANSPORT_EMAIL_PORT=587' \
#               -e 'GRAYLOG_TRANSPORT_EMAIL_USE_AUTH=true' \
#               -e 'GRAYLOG_TRANSPORT_EMAIL_USE_TLS=true' \
#               -e 'GRAYLOG_TRANSPORT_EMAIL_USE_SSL=false' \
#               -e 'GRAYLOG_TRANSPORT_EMAIL_AUTH_USERNAME=AWS_SMTP_USERNAME' \
#               -e 'GRAYLOG_TRANSPORT_EMAIL_AUTH_PASSWORD=AWS_SMTP_PASSWORD'"
    end
  end

end
```

Basically, it instructs Vagrant to:

- Set up a single VM named `vm_hello_world` using the `ubuntu/trusty64` box;
- Configure the VM with the IP address `10.0.0.10`;
- Increase the VM memory to `4GB`;
- Executes the shell provisioner to increase the maximum map count check (needed for Elasticsearch[17]);
- Start three Docker containers (MongoDB, Elasticsearch, and Graylog) using the Docker provisioner[18].

[17] https://www.elastic.co/guide/en/elasticsearch/reference/5.6/_maximum_map_count_check.html
[18] https://www.vagrantup.com/docs/provisioning/docker.html

Note that there are a bunch commented Graylog environment variables in the Graylog container configuration. We are going to use it later to set up the alerting mechanism via e-mail.

Execute the `vagrant up` command and grab a cup of coffee while your VM is being started:

```
$ vagrant up
...
==> vm_hello_world: Running provisioner: shell...
    vm_hello_world: Running: inline script
    vm_hello_world: vm.max_map_count = 262144
==> vm_hello_world: Running provisioner: docker...
    vm_hello_world: Installing Docker onto machine...
==> vm_hello_world: Starting Docker containers...
==> vm_hello_world: -- Container: mongo
==> vm_hello_world: Running provisioner: docker...
==> vm_hello_world: Starting Docker containers...
==> vm_hello_world: -- Container: elasticsearch
==> vm_hello_world: Running provisioner: docker...
==> vm_hello_world: Starting Docker containers...
==> vm_hello_world: -- Container: graylog
```

Smooth! Let's `ssh` the VM to see our three containers up and running:

```
$ vagrant ssh vm_hello_world
```

List the containers:

```
vagrant@helloworld:~$ docker container ls
CONTAINER ID   CREATED         NAMES
cf62bb96df30   3 minutes ago   graylog
68f875c59858   4 minutes ago   elasticsearch
c4c3ec17e890   6 minutes ago   mongo
```

The `docker container ls` output was truncated to fit better in the book.

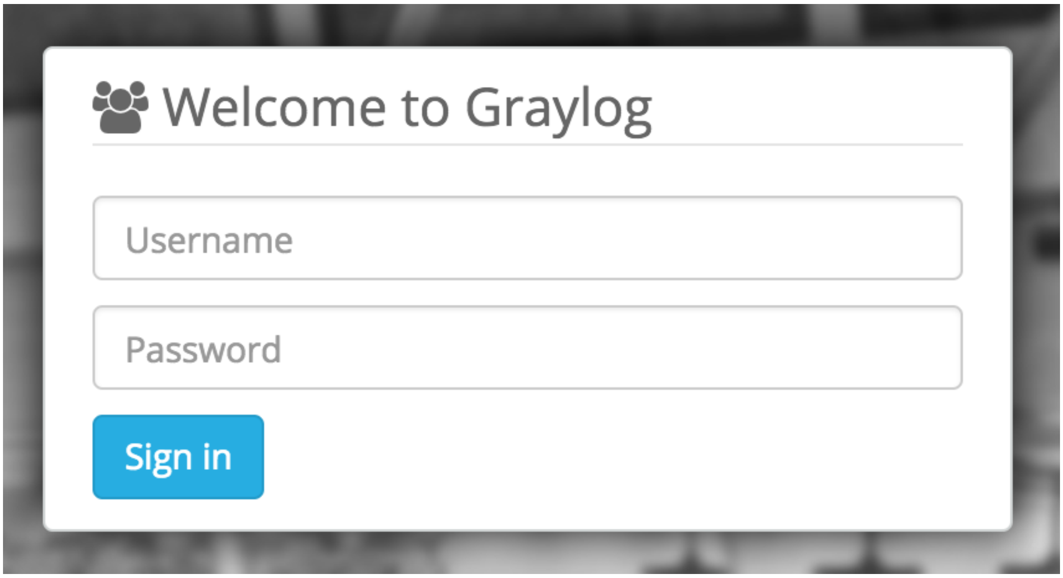
Let's check that the port 9000 (Graylog Web Interface) is waiting for connections:

```
vagrant@helloworld:~$ netstat -an | grep 9000
tcp6       0      0 :::9000                 :::*                    LISTEN
```

Awesome! Exit the VM:

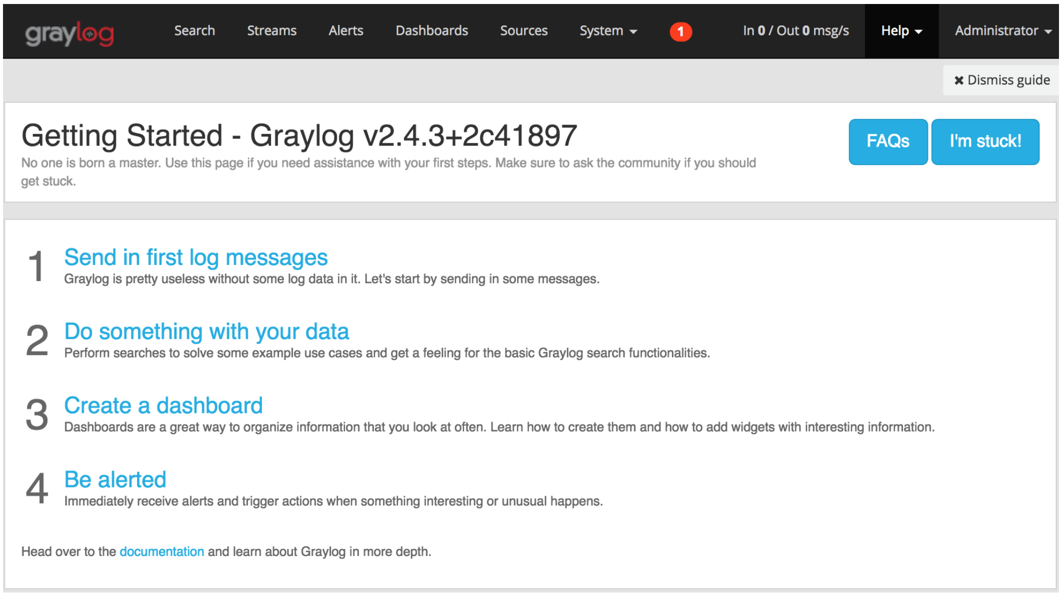
```
$ exit
```

Open your browser and navigate to `http://10.0.0.10:9000`. The Graylog Web Interface should pops up with an authentication form:



Login

Congratulations, your Graylog instance is up and running! Type in **admin** for the `username` and `password` fields and click on **Sign In**. You should see the Graylog Web Interface starting page:



Getting Started - Graylog v2.4.3+2c41897

No one is born a master. Use this page if you need assistance with your first steps. Make sure to ask the community if you should get stuck.

1 Send in first log messages
Graylog is pretty useless without some log data in it. Let's start by sending in some messages.

2 Do something with your data
Perform searches to solve some example use cases and get a feeling for the basic Graylog search functionalities.

3 Create a dashboard
Dashboards are a great way to organize information that you look at often. Learn how to create them and how to add widgets with interesting information.

4 Be alerted
Immediately receive alerts and trigger actions when something interesting or unusual happens.

Head over to the documentation and learn about Graylog in more depth.

Graylog Web Interface: Starting Page

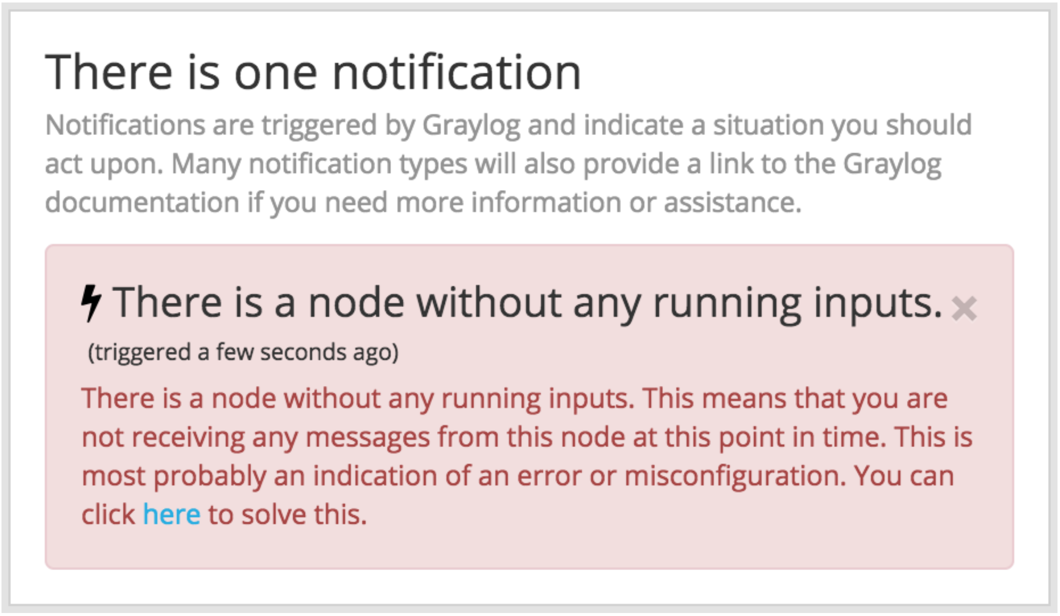
Basically, the top bar includes:

- The menu items;
- The amount of log messages being ingested per second;
- A counter with the system notifications (in red). When a relevant event takes place, a new notification will pop up on the top menu as shown.

Besides, a Getting Started guide is shown in the middle page, which is pretty much what we are going through right now.

Inputs

Back to the Graylog interface, click on the notification to see what is it about.



There is one notification

Notifications are triggered by Graylog and indicate a situation you should act upon. Many notification types will also provide a link to the Graylog documentation if you need more information or assistance.

There is a node without any running inputs.

(triggered a few seconds ago)

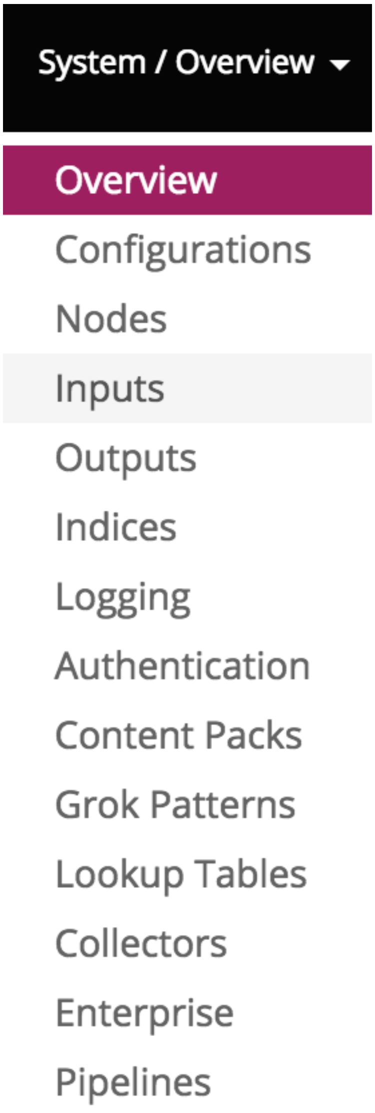
There is a node without any running inputs. This means that you are not receiving any messages from this node at this point in time. This is most probably an indication of an error or misconfiguration. You can click here to solve this.

Notification: No Inputs Found

As you may guess, Graylog is useless if no log messages are coming in. So, before you can start sending data to it, you have to set up the so called **Inputs**. Basically, inputs are the Graylog components responsible for accepting log messages.

As you have just created your Graylog instance, there are no inputs yet. As a result, Graylog friendly reminds you that you have to create an input if you want to start accepting log messages.

Let's create our first Input using the web interface. Navigate to the **System** -> **Inputs** menu item:

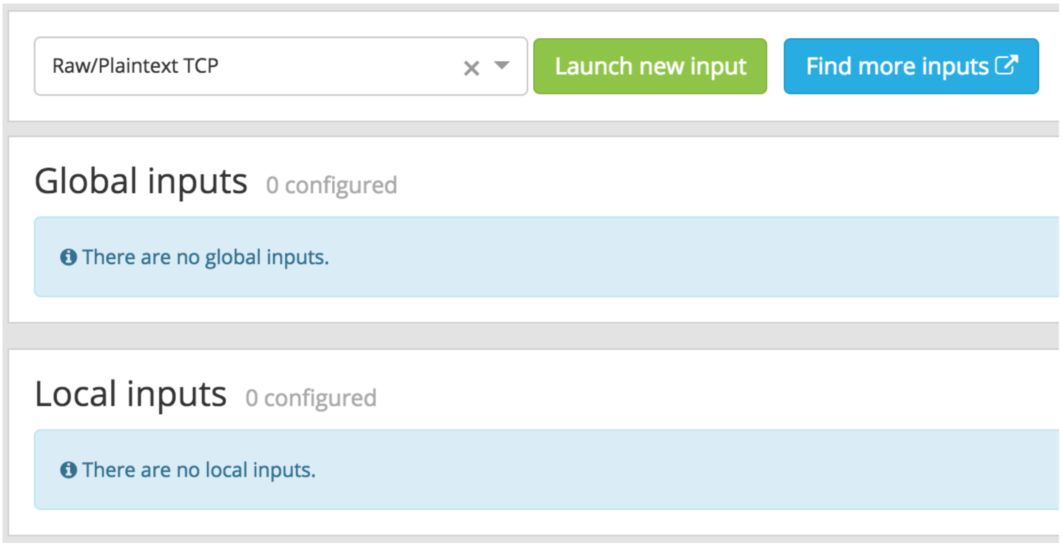


Menu: Inputs

It’s also possible to create Inputs using the REST API.

Note that we still have neither global nor local inputs configured. Select **Raw/Plain-**

text TCP and click on **Launch new input**:



Global inputs 0 configured

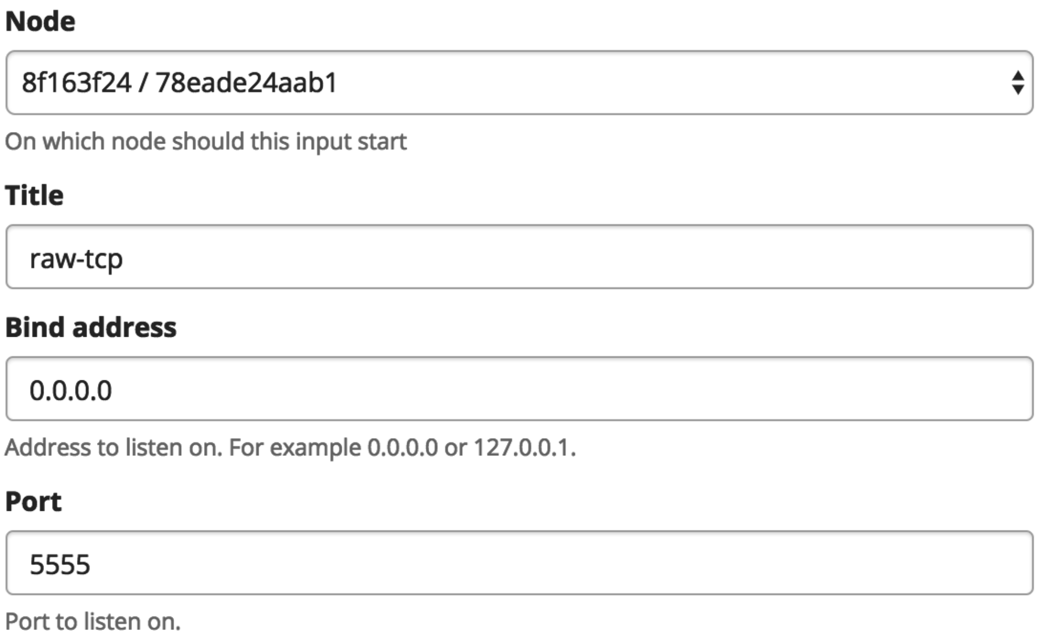
There are no global inputs.

Local inputs 0 configured

There are no local inputs.

Input: Launch New Raw/Plaintext TCP Input

Let's launch a `local` input, which is essentially one that runs on top of a specific Graylog node (as we currently have just one node, it doesn't really matters whether it's a `global` or a `local` input). Select the **Node** and give it the **Title** `raw-tcp`:


Node

8f163f24 / 78eade24aab1

On which node should this input start

Title

raw-tcp

Bind address

0.0.0.0

Address to listen on. For example 0.0.0.0 or 127.0.0.1.

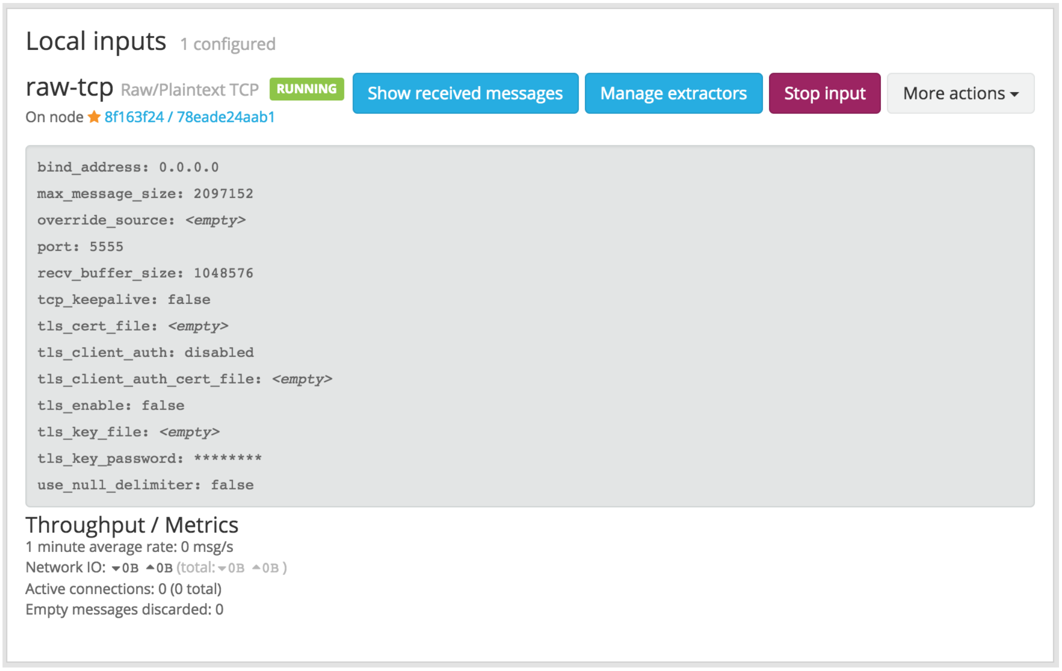
Port

5555

Port to listen on.

Raw/Plaintext TCP Input Configuration

Note that it will run on port `5555`. Click on **Save** and you should end up with the `raw-tcp` input in the **Running** status as follows:

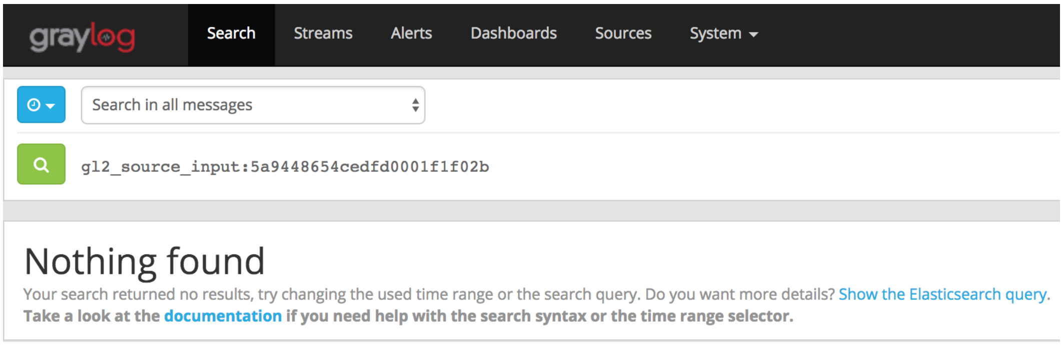

Local inputs 1 configured
raw-tcp Raw/Plaintext TCP RUNNING
Show received messages
Manage extractors
Stop input
More actions
On node 8f163f24 / 78eade24aab1

```
bind_address: 0.0.0.0
max_message_size: 2097152
override_source: <empty>
port: 5555
recv_buffer_size: 1048576
tcp_keepalive: false
tls_cert_file: <empty>
tls_client_auth: disabled
tls_client_auth_cert_file: <empty>
tls_enable: false
tls_key_file: <empty>
tls_key_password: ********
use_null_delimiter: false
```

Throughput / Metrics
1 minute average rate: 0 msg/s
Network IO: ▾0B ▴0B (total: ▾0B ▴0B)
Active connections: 0 (0 total)
Empty messages discarded: 0

Raw/Plaintext TCP Input Status: Running

Click on **Show received messages** so that you can visualize the log messages received by this input. Nothing should be found as we haven't sent anything yet:



```
gl2_source_input:5a9448654cedfd0001f1f02b
```

Nothing found
Your search returned no results, try changing the used time range or the search query. Do you want more details? Show the Elasticsearch query.
Take a look at the documentation if you need help with the search syntax or the time range selector.

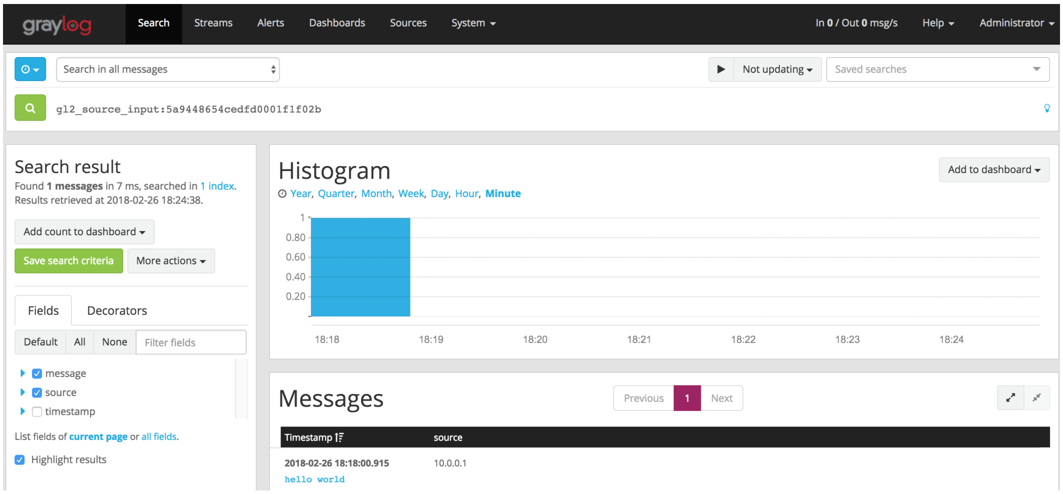
No Messages Found

As we have set up a Raw/Plaintext input, let's send a "hello world" plain-text message from the terminal using Netcat[19]:

```
$ echo "hello world" | nc 10.0.0.10 5555
```

After sending the message, if you quickly return to Graylog web interface you will be able to see the message throughput updating to **In 1 / Out 1 msg/s** on the top menu, indicating that the message was received and properly indexed in Elasticsearch.

Return to the browser and click on the green search icon:



```
gl2_source_input:5a9448654cedfd0001f1f02b
```

Search result
Found **1 messages** in 7 ms, searched in 1 index.
Results retrieved at 2018-02-26 18:24:38.

☑ message
☑ source
☐ timestamp

List fields of **current page** or all fields.

☑ Highlight results

Timestamp	source
2018-02-26 18:18:00.915	10.0.0.1
hello world	

Hello World Message

Here we go, the message was successfully received and indexed! On the left sidebar you can select which fields you want to visualize in the main panel. Of course this was a simple message with just few fields, but later we are going to send log messages with lots of fields, so it’s quite helpful to have the ability to filter by field.

[19] https://en.wikipedia.org/wiki/Netcat

Forwarding VM Logs using Rsyslog

The Syslog protocol[20] is a standard for message logging. Basically, it decouples the software that generates the logs, the system that stores them, and the software used to visualize and analyze them (which in our case is Graylog).

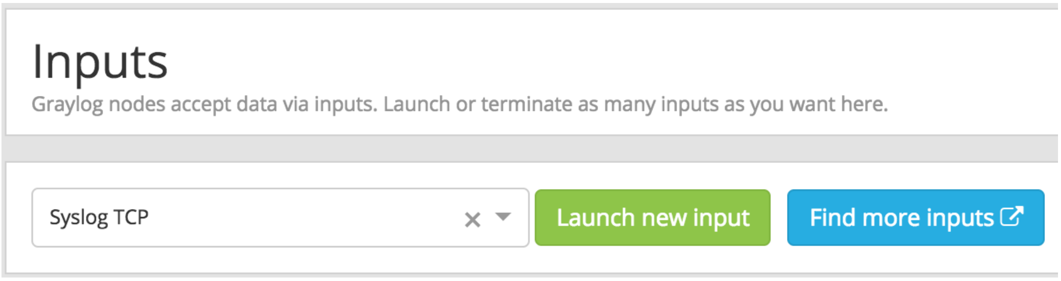
Rsyslog is the newest (out of three) implementations of the Syslog protocol and it's available by default on a number of Unix systems and Linux distributions. Besides implementing the Syslog protocol, it also extends its functionalities with:

- Support for buffered operation modes where messages are buffered locally if the receiver is not ready;
- Reliable transport using TCP;
- ISO 8601 timestamp with millisecond granularity and timezone information;
- Support for TLS;

And many other relevant features.

The other two implementations are Syslog (yes, this implementation has the same name as the protocol, so don't get confused!) and syslog-ng.

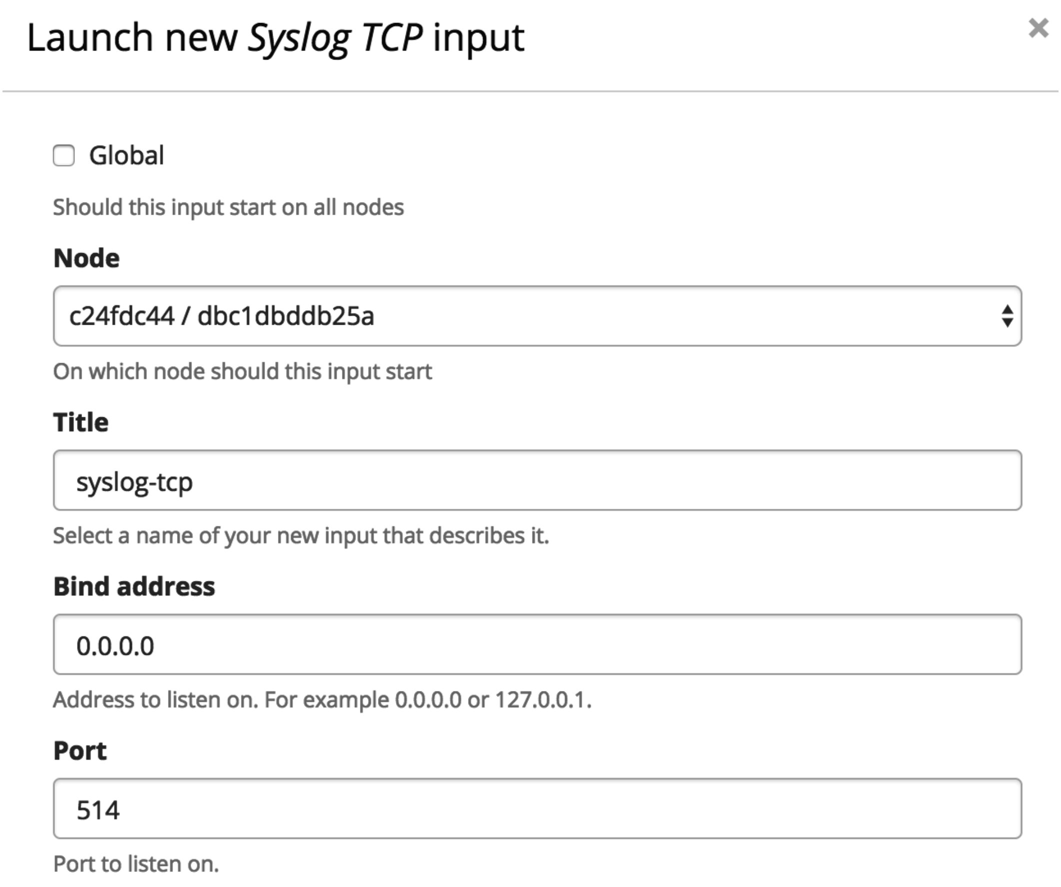
What's the first step we have to take in order get log messages in Graylog? That's correct, creating an Input! Navigate to **System** -> **Inputs** menu item, select **Syslog TCP** and click on **Launch new input**:


Inputs
Graylog nodes accept data via inputs. Launch or terminate as many inputs as you want here.

Input: Launch New Syslog TCP Input

[20] https://tools.ietf.org/html/rfc5424

Select the **Node**, give it the **Title** `syslog-tcp` and keep the port `514` as follows:



Launch new *Syslog TCP* input

☐ Global

Should this input start on all nodes

Node

c24fdc44 / dbc1dbddb25a

On which node should this input start

Title

syslog-tcp

Select a name of your new input that describes it.

Bind address

0.0.0.0

Address to listen on. For example 0.0.0.0 or 127.0.0.1.

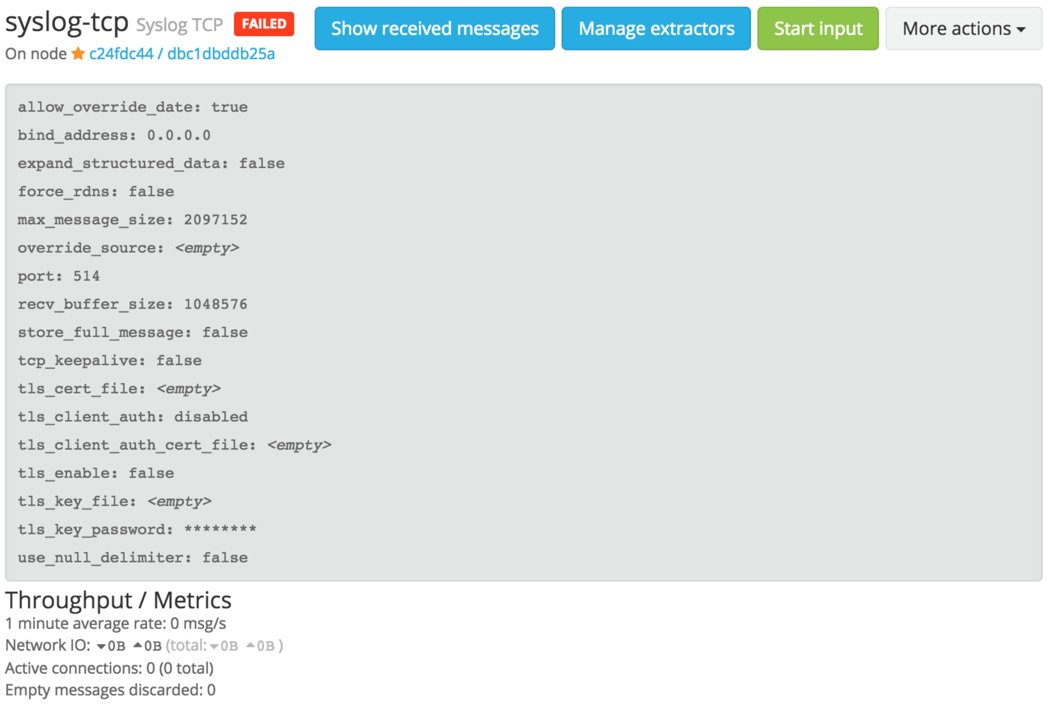
Port

514

Port to listen on.

Syslog TCP Input: Configuration

Finally, click on **Save**.



```
allow_override_date: true
bind_address: 0.0.0.0
expand_structured_data: false
force_rdns: false
max_message_size: 2097152
override_source: <empty>
port: 514
recv_buffer_size: 1048576
store_full_message: false
tcp_keepalive: false
tls_cert_file: <empty>
tls_client_auth: disabled
tls_client_auth_cert_file: <empty>
tls_enable: false
tls_key_file: <empty>
tls_key_password: ********
use_null_delimiter: false
```

Throughput / Metrics
1 minute average rate: 0 msg/s
Network IO: ▾0B ▴0B (total: ▾0B ▴0B)
Active connections: 0 (0 total)
Empty messages discarded: 0

Syslog TCP Input: Failed to Launch

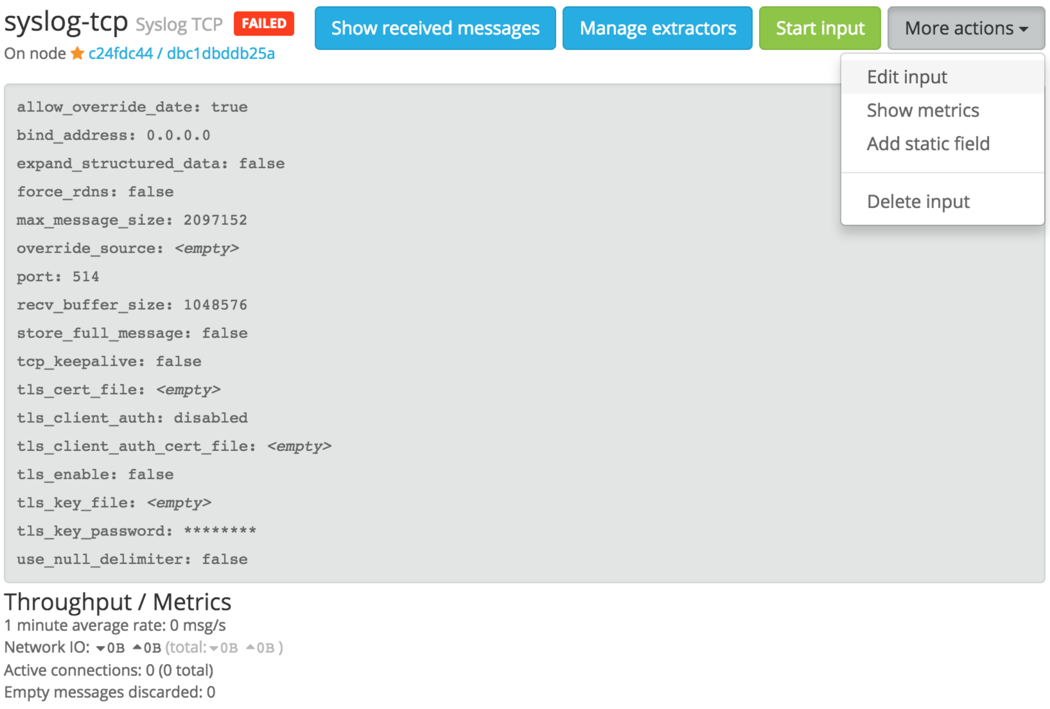
As you can see, the Input has failed to launch. Let's try to figure out why is that. First, `ssh` the `vm_hello_world`:

```
$ vagrant ssh vm_hello_world
```

Check the Graylog logs using the `docker container logs` command:

```
vagrant@helloworld:~$ docker container logs graylog
018-02-26 20:52:46,359 ERROR: org.graylog2.shared.inputs.InputLauncher - The [org.gra\
ylog2.inputs.syslog.tcp.SyslogTCPInput] input with ID <5a94739d4cedfd0001f21f10> misf\
ired. Reason: Permission denied.
org.graylog2.plugin.inputs.MisfireException: org.graylog2.plugin.inputs.MisfireExcept\
ion: org.jboss.netty.channel.ChannelException: Failed to bind to: /0.0.0.0:514
        at org.graylog2.plugin.inputs.MessageInput.launch(MessageInput.java:158) ~[graylog.j\
ar:?]
        at org.graylog2.shared.inputs.InputLauncher$1.run(InputLauncher.java:84) [graylog.ja\
r:?]
        at com.codahale.metrics.InstrumentedExecutorService$InstrumentedRunnable.run(Instrum\
entedExecutorService.java:176) [graylog.jar:?]
        at java.util.concurrent.Executors$RunnableAdapter.call(Executors.java:511) [?:1.8.0_\
151]
        at java.util.concurrent.FutureTask.run(FutureTask.java:266) [?:1.8.0_151]
        at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1149) [\
?:1.8.0_151]
        at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:624) [\
?:1.8.0_151]
        at java.lang.Thread.run(Thread.java:748) [?:1.8.0_151]
Caused by: org.graylog2.plugin.inputs.MisfireException: org.jboss.netty.channel.Chann\
elException: Failed to bind to: /0.0.0.0:514
        at org.graylog2.plugin.inputs.transports.NettyTransport.launch(NettyTransport.java:1\
55) ~[graylog.jar:?]
        at org.graylog2.plugin.inputs.MessageInput.launch(MessageInput.java:155) ~[graylog.j\
ar:?]
        ... 7 more
Caused by: org.jboss.netty.channel.ChannelException: Failed to bind to: /0.0.0.0:514
        at org.jboss.netty.bootstrap.ServerBootstrap.bind(ServerBootstrap.java:272) ~[graylo\
g.jar:?]
        at org.graylog2.plugin.inputs.transports.NettyTransport.launch(NettyTransport.java:1\
41) ~[graylog.jar:?]
        at org.graylog2.plugin.inputs.MessageInput.launch(MessageInput.java:155) ~[graylog.j\
ar:?]
        ... 7 more
Caused by: java.net.SocketException: Permission denied
        at sun.nio.ch.Net.bind0(Native Method) ~[?:1.8.0_151]
        at sun.nio.ch.Net.bind(Net.java:433) ~[?:1.8.0_151]
        at sun.nio.ch.Net.bind(Net.java:425) ~[?:1.8.0_151]
        at sun.nio.ch.ServerSocketChannelImpl.bind(ServerSocketChannelImpl.java:223) ~[?:1.8\
.0_151]
        at sun.nio.ch.ServerSocketAdaptor.bind(ServerSocketAdaptor.java:74) ~[?:1.8.0_151]
        at org.jboss.netty.channel.socket.nio.NioServerBoss$RegisterTask.run(NioServerBoss.j\
ava:193) ~[graylog.jar:?]
        at org.jboss.netty.channel.socket.nio.AbstractNioSelector.processTaskQueue(AbstractN\
ioSelector.java:391) ~[graylog.jar:?]
        at org.jboss.netty.channel.socket.nio.AbstractNioSelector.run(AbstractNioSelector.ja\
va:315) ~[graylog.jar:?]
        at org.jboss.netty.channel.socket.nio.NioServerBoss.run(NioServerBoss.java:42) ~[gra\
ylog.jar:?]
        at org.jboss.netty.util.ThreadRenamingRunnable.run(ThreadRenamingRunnable.java:108) \
~[graylog.jar:?]
        at org.jboss.netty.util.internal.DeadLockProofWorker$1.run(DeadLockProofWorker.java:\
42) ~[graylog.jar:?]
        ... 3 more
```

As the exception states, we got a **Permission denied** error. Basically, that happened because the ports ranging from **0-1023** are reserved for the root user. So, back to the Graylog web interface, click on **More actions** and edit the `syslog-tcp` input:

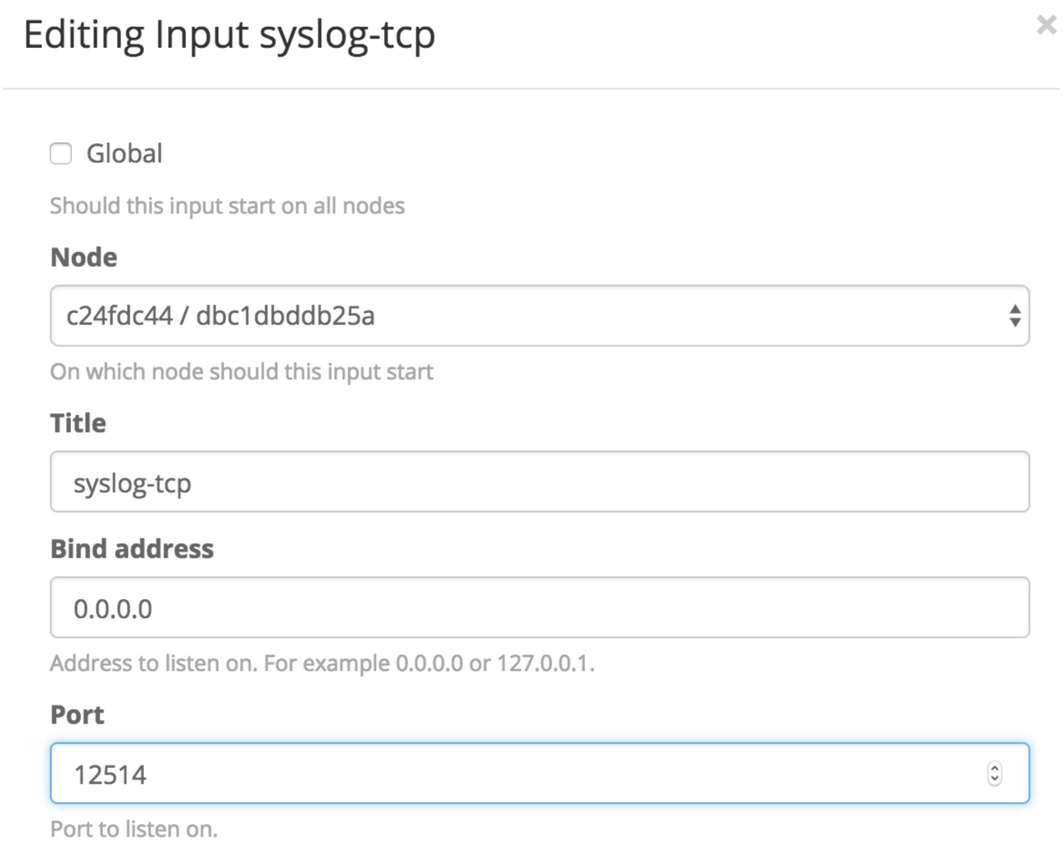


```
allow_override_date: true
bind_address: 0.0.0.0
expand_structured_data: false
force_rdns: false
max_message_size: 2097152
override_source: <empty>
port: 514
recv_buffer_size: 1048576
store_full_message: false
tcp_keepalive: false
tls_cert_file: <empty>
tls_client_auth: disabled
tls_client_auth_cert_file: <empty>
tls_enable: false
tls_key_file: <empty>
tls_key_password: ********
use_null_delimiter: false
```

Throughput / Metrics
1 minute average rate: 0 msg/s
Network IO: ▾0B ▴0B (total: ▾0B ▴0B)
Active connections: 0 (0 total)
Empty messages discarded: 0

Syslog TCP Input: Edit

Change the port to `12514` and save it.



Editing Input syslog-tcp

Global: [ ]

Should this input start on all nodes

Node: c24fdc44 / dbc1dbddb25a

On which node should this input start

Title: syslog-tcp

Bind address: 0.0.0.0

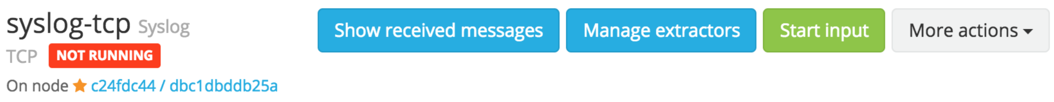
Address to listen on. For example 0.0.0.0 or 127.0.0.1.

Port: 12514

Port to listen on.

Syslog TCP Input: Port 12514

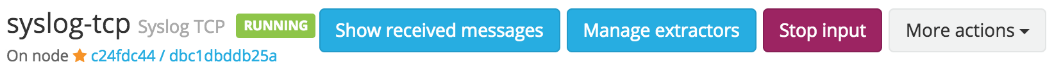
The input status should change from `Failed` to `Not Running`:



syslog-tcp Syslog
TCP NOT RUNNING
On node ★ c24fdc44 / dbc1dbddb25a

Syslog TCP Input Status: Not Running

Now just click on **Start Input** and voila!



Syslog TCP Input Status: Running

Click on **Show Received Messages** to see the incoming messages handled by our newly created Syslog TCP Input. As we have not set up the VM Rsyslog configuration to forward log messages to Graylog, obviously there should be no messages in it yet.

Let's set up the Rsyslog running on the VM so that it can forward the VM logs to the Graylog instance. You should probably already be logged into the VM as we were checking the Graylog container logs earlier, but if not, just `vagrant ssh` it again. Edit the Rsyslog configuration file as `sudo`:

```
vagrant@helloworld:~$ sudo vim /etc/rsyslog.conf
```

Scroll down to the bottom of the file, activate the `insertion mode` hitting `i` and add the following line in it:

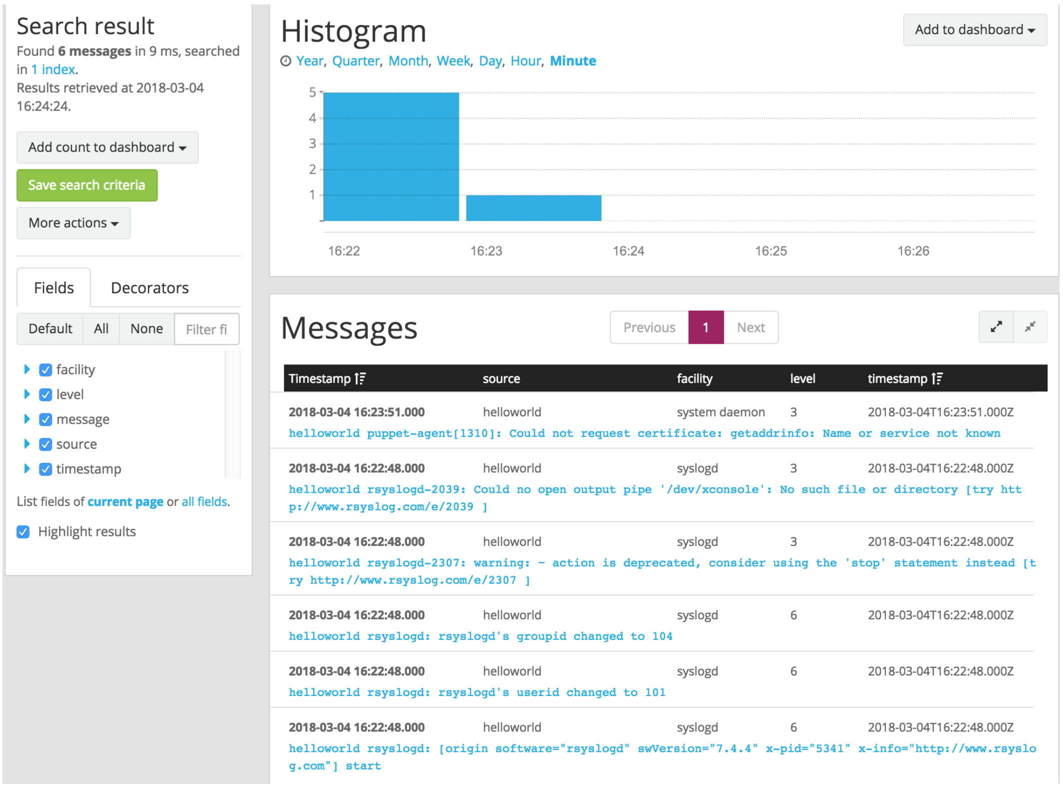
```
*.* @@10.0.0.10:12514;
```

Basically, this line means that we want to send every log message from every facility (`*.*`) using TCP as the transport protocol (`@@`) to server `10.0.0.10` on port `12514`.

Save it (`esc : wq` and hit `return`) and restart the Rsyslog service so that the changes take place:

```
vagrant@helloworld:~$ sudo service rsyslog restart
rsyslog stop/waiting
rsyslog start/running, process 5341
```

Return to the Graylog web interface, search for messages again and select all fields on the left sidebar:



Found **6 messages** in 9 ms, searched in 1 index.
Results retrieved at 2018-03-04 16:24:24.

List fields of **current page** or all fields.

Timestamp	source	facility	level	timestamp
2018-03-04 16:23:51.000	helloworld	system daemon	3	2018-03-04T16:23:51.000Z
2018-03-04 16:22:48.000	helloworld	syslogd	3	2018-03-04T16:22:48.000Z
2018-03-04 16:22:48.000	helloworld	syslogd	3	2018-03-04T16:22:48.000Z
2018-03-04 16:22:48.000	helloworld	syslogd	6	2018-03-04T16:22:48.000Z
2018-03-04 16:22:48.000	helloworld	syslogd	6	2018-03-04T16:22:48.000Z
2018-03-04 16:22:48.000	helloworld	syslogd	6	2018-03-04T16:22:48.000Z

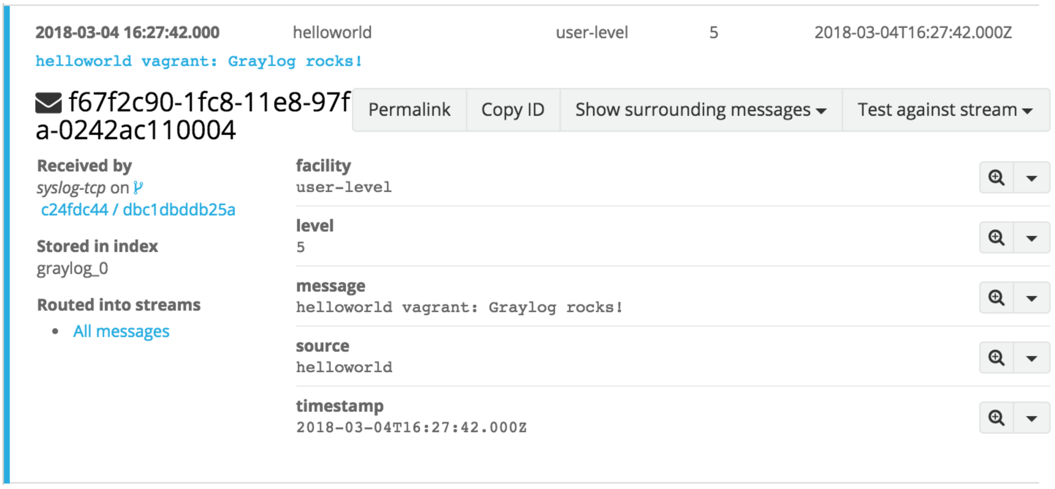
```
helloworld puppet-agent[1310]: Could not request certificate: getaddrinfo: Name or service not known
helloworld rsyslogd-2039: Could no open output pipe '/dev/xconsole': No such file or directory [try htt
p://www.rsyslog.com/e/2039 ]
helloworld rsyslogd-2307: warning: ~ action is deprecated, consider using the 'stop' statement instead [t
ry http://www.rsyslog.com/e/2307 ]
helloworld rsyslogd: rsyslogd's groupid changed to 104
helloworld rsyslogd: rsyslogd's userid changed to 101
helloworld rsyslogd: [origin software="rsyslogd" swVersion="7.4.4" x-pid="5341" x-info="http://www.rsyslo
g.com"] start
```

VM Logs

Awesome! The VM logs are now being sent to our Graylog instance. Use the `logger` shell utility to send a test message:

```
vagrant@helloworld:~$ logger "Graylog rocks!"
```

Search for messages again and click on the **Graylog rocks!** message to see its details:



```
2018-03-04 16:27:42.000    helloworld    user-level    5    2018-03-04T16:27:42.000Z
helloworld vagrant: Graylog rocks!
```

f67f2c90-1fc8-11e8-97fa-0242ac110004

Received by
syslog-tcp on c24fdc44 / dbc1dbddb25a

Stored in index
graylog_0

Routed into streams
- All messages

```
facility
user-level

level
5

message
helloworld vagrant: Graylog rocks!

source
helloworld

timestamp
2018-03-04T16:27:42.000Z
```

Message Details

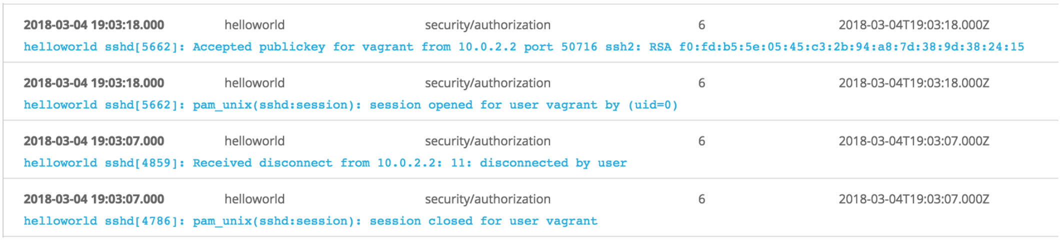
Note that the facility for this message is user-level (which makes sense, doesn't it?) and the message was stored in Elasticsearch index graylog_0.

Now, log out the VM and log in again:

```
vagrant@helloworld:~$ exit
logout
Connection to 127.0.0.1 closed.

$ vagrant ssh vm_hello_world
Welcome to Ubuntu 14.04.5 LTS (GNU/Linux 3.13.0-141-generic x86_64)
```

Return to the Graylog web interface and search for messages again. You should start seeing security/authorization facility messages popping up:



```
2018-03-04 19:03:18.000    helloworld    security/authorization    6    2018-03-04T19:03:18.000Z
helloworld sshd[5662]: Accepted publickey for vagrant from 10.0.2.2 port 50716 ssh2: RSA f0:fd:b5:5e:05:45:c3:2b:94:a8:7d:38:9d:38:24:15
2018-03-04 19:03:18.000    helloworld    security/authorization    6    2018-03-04T19:03:18.000Z
helloworld sshd[5662]: pam_unix(sshd:session): session opened for user vagrant by (uid=0)
2018-03-04 19:03:07.000    helloworld    security/authorization    6    2018-03-04T19:03:07.000Z
helloworld sshd[4859]: Received disconnect from 10.0.2.2: 11: disconnected by user
2018-03-04 19:03:07.000    helloworld    security/authorization    6    2018-03-04T19:03:07.000Z
helloworld sshd[4786]: pam_unix(sshd:session): session closed for user vagrant
```

sshd Messages

Awesome, isn’t it? Even though some bad guy gets ssh access to your servers and try to wipe off the trail out of the filesystem, the logs would have already been sent to our centralized logging system!

Appendices

Introduction to Docker

Let's forget for a while that this is a book about Graylog so that we can focus on learning Docker.

Have you ever heard the sentence "I don't know what is happening; it works on my machine"? What about the famous "this version was working perfectly on staging but it's not working on production"? One of the main reasons why these issues happen is because environments often become different over time, especially those managed manually. People are often installing different libraries on different versions in different environments (which can also run different operating systems, by the way). The result is that, for example, the staging environment is no longer a mirror from production over time. If the staging is not a mirror from production anymore, of course that the application running on these environments could behave different on each environment, what leads to bugs that happen in an environment but not in another.

In short, Docker allows you to easily run services (Graylog, Elasticsearch, MongoDB, MySQL, Jenkins, Nginx, your own application, etc) on a machine. Docker guarantees that these services will always be in the same state across executions, regardless of the underlying operating system or system libraries. In other words, if your staging environment is very different from your production environment (system libraries, operating systems, etc) and you run an application as a Docker container on both environments, it's guaranteed that the application would behave exactly the same on these environments regardless of their differences.

In my humble opinion, this is the most important benefit got from using Docker, but there are many more:

- It's easy to run services as Docker containers. Thus, it also helps a lot in the development phase because you don't have to waste time installing and configuring tools on your operating system.
- Docker is a highly collaborative tool. You can reuse Docker images that people build and share publicly.

- It encourages the infrastructure as code model because a Docker image is entirely described on a file called a Dockerfile that can (and should) be versioned in the source control.
- Docker has a great community, and it's expanding quickly.

Difference Between Container and Image

Docker images are binary files that contain everything needed to run a specific service. When you instantiate a service from a Docker image, you say that you create a Docker container. As an analogy, if a Docker image is a Java class, then a Docker container is an object. Many containers can be created from a single image.

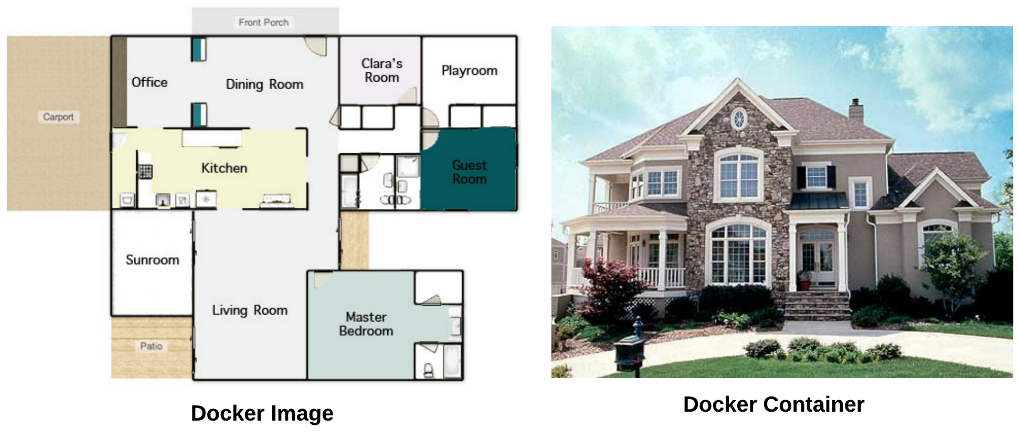


Image vs Container

As the figure above shows, an image is like a house scheme whereas the container is the concrete house where people actually live in.

Technically speaking, a container is a group of processes contained in an isolated environment, but running on the same kernel as the host operating system. This isolation is provided by concepts like `cgroups` and `namespaces`. For example, if you create a file inside a container, this file cannot be accessed from the host operating system (unless you explicitly specify this).

Docker images can be either created from a container state (like a snapshot of a running container) or describing the image state (like the operating system, libraries and applications) on a special file called `Dockerfile`. Both work, but which way do you think it's better? That's right, the `Dockerfile` way, because it encourages the infrastructure as code model, that is, the image state is easily tracked using a source control and safely evolves as every single change would go through a pull request, code review, and so on.

You create a Docker image from a `Dockerfile` using the **docker image build** command and create a Docker container by executing the **docker container run** command.