Linux

- How to Create and Use MacVLAN Network in Docker
- Markdown Cheat Sheet (1)
- RegEx Tutorial
- RegEx Reference (todo)

How to Create and Use MacVLAN Network in Docker

In Docker, a common question that usually comes up is **“How do I expose my containers directly to my local physical network?”** This is especially so when you are running monitoring applications that are collecting network statistics and want to connect container to legacy applications. A possible solution to this question is to create and implement the macvlan network type.

Macvlan networks are special virtual networks that allow you to create “clones” of the physical network interface attached to your Linux servers and attach containers directly your LAN. To ensure this happens, simple designate a physical network interface on your server to a macvlan network which has its own subnet and gateway.

In this guide, we will demonstrate how you can create and use mavlan networks in Docker. But before you get started, here are a few things that you should keep in mind:

NOTE:

- Macvlan networks are usually blocked by most Cloud service providers. Hence, you need physical access to your server.
- The macvlan network driver only works on Linux hosts. It is not supported on Windows or mac devices.
- You need to be running on Linux kernel 4.0 and later.

In this guide, we will use Ubuntu 20.04 to demonstrate how to create and use macvlan networks. As a prerequisite, we have Docker installed. We have a guide on how to install Docker on Ubuntu 20.04.

Creating a macvlan network

A macvlan network can be created either in **bridge mode** and or **802.1q trunk mode**.

In bridge mode, the macvlan traffic is channeled through the physical interface on the Linux host.

In the 802.1q trunk bridge mode, traffic passes through an 802.1q sub-interface which is created by Docker. This allows for controlled routing and filtering at a granular level.

With that out of the way, let us now see how you can create each of the macvlan networks.

Bridge mode

In our example, we have a physical network interface **enp0s31f6** on the **192.168.123.0/24** network and the default gateway of **192.168.123.1**.The default gateway is the IP address of the router.

Now, we will create a macvlan network called demo-macvlan-net with the following configuration.

```
### create mac-vlan network
docker network create \
  -d macvlan \
  --subnet=192.168.123.0/24 \
  --gateway=192.168.123.1  \
  -o parent=enp0s31f6 \
  demo-macvlan-net
```

NOTE:

- The **subnet & gateway values need to match those of the Docker host network interface**. Simply put, the subnet and default gateway for your **macvlan network** should mirror that of your Docker host. Here, the **--subnet=** option specifies the subnet and the **--gateway** option defines the gateway which is the router's IP. Modify these values to accommodate your environment.
- The **-d option** specifies the **driver name**. In our case, the -d option specifies the **macvlan driver**.
- The **-o parent option** specifies the parent interface which is your NIC interface. In our case, the parent interface is **enp0s31f6**.
- Finally, we have specified the name of our macvlan network which is **demo-macvlan-net** .

To confirm that the newly added macvlan network is present, run the command:

```
### check network is created
docker network ls
```

Next, we will create a container image and attach is to the macvlan network using the **--network option**. The **-itd option** allows you to run the container in the background and also to attach to it. The **--m** option removes the container once it is stopped. We have also assigned the **IP 192.168.123.111** to our container. Be sure to specify an IP that is **not within** your DHCP IP range to avoid instances of an IP conflict.

```
### run an image and assign the static IP
docker run --rm -itd \
  --name alp1 \
```

```
  --network=demo-macvlan-net \
  --ip=192.168.123.111 \
  alpine:latest \
  /bin/sh

### check the container is created
docker ps

### find container ID and
### view details of the container
docker container inspect <ID1>
```

Let us now create a second container as follows. In this case, the container will **automatically be assigned an IP by Docker**.

```
### create new container and let DHCP assign the IP
docker run  --rm -itd \
  --name alp2 \
  --network=demo-macvlan-net \
  alpine:latest \
  /bin/sh

### find ID of the second container
docker ps
```

Next, we will try and establish if the containers can ping each other. In this case, we are testing connectivity of the first container from the second container.

You can achieve this using a single command as follows.

```
### from the second container ping the first one
docker exec -it <ID2> ping 192.168.123.111 -4
```

Alternatively, you can access the shell of the container and run the ping command.

```
### you can exec into second container and ping the first one
docker exec -it <ID2> /bin/sh
# ping 192.168.123.111 -4
```

Try the same command from the other container and you will find out that at this point, the containers can communicate with each other.

However, the Docker host cannot communicate with the containers and vice-versa. If you try pinging the host from the container or the other way round, you will find out that that host and the containers cannot communicate with each other.

From the output shown, we cannot reach one of the containers using the ping command.

```
### both containers can ping each other
docker exec -it alp1 ping alp2 -4
docker exec -it alp2 ping alp1 -4

##### this does not work

### ping host from any container
docker exec -it alp1 ping 192.168.123.100 -4
docker exec -it alp2 ping 192.168.123.100 -4

### ping from host to any of the containers
ping alp1 -4
ping alp2 -4
```

For the containers to communicate with the host, we need to create a macvlan interface on the Docker host and configure a route to the macvlan interface.

Create new macvlan interface on the host

Next, we are going to create a **macvlan interface** using **ip command**.In this example, we have created an interface called **mycool-net**. Feel free to give it any name you deem fit.

```
sudo ip link add mycool-net link enp0s31f6 type macvlan mode bridge
```

Then assign a unique IP to the interface. **Ensure to reserve this IP on your router.**

```
### Ensure to reserve this IP on your router.
sudo ip addr add 192.168.123.50/32 dev mycool-net
```

Bring up the macvlan interface.

```
sudo ip link set mycool-net up
```

The last step is to instruct our Docker host to use the interface in order to communicate with the containers. For this, we will **add a route to the macvlan network**.

```
sudo ip route add 192.168.123.0/24 dev mycool-net
```

Now, with the route in place, the host and the containers can communicate with each other. You can verify the routes using the **ip route command**.

```
ip route
```

- I’m able to ping the host from the one container.
- I can also well ping the host from the second container.
- I can also ping one of the containers from the host.

```
### both containers can ping each other
docker exec -it alp1 ping alp2 -4
docker exec -it alp2 ping alp1 -4

### ping host from any container
docker exec -it alp1 ping 192.168.123.100 -4
docker exec -it alp2 ping 192.168.123.100 -4

### working : ping container from host by ip
ping 192.168.123.111 -4
ping 192.168.123.2 -4

### not working : ping from host to any of the containers
ping alp1 -4
ping alp2 -4
```

Markdown Cheat Sheet (1)

This Markdown cheat sheet contains the most-used Markdown elements. To keep this cheat sheet brief, I will sometimes link to more detailed explanations in separate articles.

Markdown Headers

There are two ways to create a header in Markdown: by using hashes or by using underlines.

Hashes

The first option, creating headers with hash signs, is more compact and quicker to type:

```
# Primary header (similar to HTML H1 tag)

## Secondary header (similar to HTML H2 tag)

### Tertiary header (similar to HTML H3 tag)

#### Quaternary header (similar to HTML H4 tag)
```

Underlined headers

Alternatively, you can ‘underline’ your header. It’s a bit more work, but some people prefer it since it stands out more when looking at the plain text. Note that you can only use this for primary and secondary headers:

```
Primary header (similar to a HTML H1 tag)
=========================================

Secondary header (similar to a HTML H2 tag)
-------------------------------------------
```

Emphasis (bold, italic)

There are three ways to emphasize a piece of text that we'll show in this markdown sheet cheat:

1. By making it bolder than the surrounding text
2. Using italics
3. Or combining bold and italics

Make bold text with markdown (strong emphasis)

To create bold text, also called strong emphasis, surround it with two asterisks or two underscores. What you use is a matter of taste.

```
This is how you create **bold** text
```

This is how you create **bold** text

When converted to HTML or PDF, it will result in: "This is how you create **bold** text".

This is the same, but with underscores:

```
This is how you create __bold__ text
```

Italics with markdown (emphasis)

To create italic text, also called emphasis, support the text with single asterisks or underscores. Again, it's a matter of taste which one you choose:

```
This is how you create *italic* text
```

This is how you create *italic* text

When converted to HTML or PDF, it will result in:: "This is how you create *italic* text."

The same, but with underscores:

```
This is how you create _italic_ text
```

This is how you create *italic* text

Combine bold and italics

You can combine both, resulting in ***bold italics*** text. You can either mix and match or use triple underscores or triple asterisks like this:

```
This is how you create **_bold italics_** text.
This is how you create ___bold italics___ text.
This is how you create ***bold italics*** text.
```

The result when converted to HTML or PDF:

This is how you create ***bold italics*** text.

Strikethrough

You can strikethrough your text as follows:

```
This is how you create ~~strikethrough~~ text.
```

This results in: “This is how you create ~~strikethrough~~ text.”

Markdown Lists

You can either create ordered (numbered) lists or unordered lists. Both types of markdown lists look very natural in plain text, as you’ll see in the examples in this markdown sheet cheat.

Ordered lists

To create an ordered list, create a text list with numbers, one per line:

```
1. Apples
2. Bananas
3. Peanut butter
```

1. Apples
2. Bananas
3. Peanut butter

I numbered the individual items properly, but you don’t have to. In fact, any number will do. However, properly numbering the items will look better in plain text, but it can be a pain sometimes. For example, if you need to add items in the middle of a long list, you need to renumber all of them.

In the end, use whatever suits you best. I tend to number my lists properly since most lists are small anyway, and it’s easier on the eyes when looking at the source text.

All markdown converters I know of will convert the following into exactly the same list as the one above:

```
1. Apples
1. Bananas
1. Peanut butter
```

The result:

1. Apples
2. Bananas
3. Peanut butter

Unordered lists

Unordered lists can be made in two ways:

- with asterisks
- or by using dashes

The following lists give an identical output when converted into another document format like HTML or PDF:

```
- Apples
- Bananas
- Peanut butter
```

and:

```
* Apples
* Bananas
* Peanut butter
```

Both result in:

- Apples
- Bananas
- Peanut butter

Checklists (or Task Lists, TODO lists)

Some but not all markdown parsers support checklists. E.g. this works on GitHub and with several markdown plugins for VS Code, for example, but it won't work within the WordPress Gutenberg editor:

```
- [x] Apples
- [ ] Bananas
- [ ] Peanut butter
```

- ☑ Apples
- ☐ Bananas
- ☐ Peanut butter

You can use these to keep track of work (TODO list) or tasks that need to be completed.

Links in Markdown

You can use Markdown to link to a website or local file. In fact, most markdown converters will automatically convert a URL to a link, but you better not rely on it. In case of doubt, it's best to try and see what happens in your specific case.

A basic link

```
This is an inline-style link to our [markdown cheat sheet](/markdown-cheat-sheet)
You can also create a relative link: [code repo](../repo/code)
Or an absolute, inline-style link to [Google](https://google.com)
URLs like <https://google.com>, and sometimes https://google.com
or even google.com, get converted to clickable links.
Inline-style link with a title attribute to [Markdown Land](https://markdown.land "Markdown
Land")
```

This is an inline-style link to our markdown cheat sheet

You can also create a relative link: code repo

Or an absolute, inline-style link to Google

URLs like https://google.com, and sometimes https://google.com or even google.com, get converted to clickable links.

Inline-style link with a title attribute to Markdown Land

Reference style links

In addition to regular links, markdown also supports reference-style links. We can define these links somewhere in a file, usually at the bottom, and reference them in the text anywhere.

```
[This website about Markdown][Case-insensitive reference link to Markdown.land]
[The Python.land tutorial with a numbered link][1]
Or leave it empty and use the link text itself, like this: [Python Land].

Text and other Markdown markup, to demonstrate that the reference links can be put anywhere.

Usually, they are placed at the bottom of a document. Because they are reference links, they
won't show up by themselves. So you won't see the links below.

[Case-insensitive reference link to Markdown.land]: https://www.markdown.land
[1]: https://python.land/python-tutorial
[Python Land]: http://python.land
```

This website about Markdown

The Python.land tutorial with a numbered link Or leave it empty and use the link text itself, like this: Python Land.

Text and other Markdown markup, to demonstrate that the reference links can be put anywhere.

Usually, they are placed at the bottom of a document. Because they are reference links, they won't show up by themselves. So you won't see the links below.

Tables

I created a separate page explaining all the details about Markdown tables, but for quick reference, here's an example of a Markdown table with the alignment of the price and availability columns:

```
| Item         | Price | # In stock |
|--------------|:-----:|-----------:|
| Juicy Apples |  1.99 |        739 |
| Bananas      |  1.89 |          6 |
```

Head over to the Markdown table page for all the details, including column alignment and links to handy table generators to save you time and effort.

Emoji's

Let me start with the disclaimer: use emojis sparingly to not annoy your readers or look like a child □□

There are two ways to add emojis to your Markdown file:

1. by copying the Unicode character and pasting it in your file
2. By using an emoji shortcode

Copy and pasting emojis into Markdown

A quick and easy method is to copy and paste the emoji. Many sites help you find and copy emojis. Here's one, and here's another one. Select the emoji you like, hit control + c, and paste it in your markdown file with control + v. If you're on a mac, that would be cmd+c and cmd+v.

Using shortcodes

I don't recommend this method since it's less readable. But in rare cases, you might prefer it. E.g., if your text document is not stored in Unicode format. Not all parses will support this method, but you can always try.

An emoji shortcode is a word surrounded by colons like these:

- □□
- :laughing:
- □□
- … etcetera

The list of available words depends heavily on the Markdown parser. Here's a list of shortcodes that work on Github.

RegEx Tutorial

Summary

Welcome to the regeEx Tutorial. This template is used to help better understand regex and their different uses. Regex or Regular Expressions are defined as a sequence of special characters that describe a search pattern. The regex that will be used in this tutorial is **Matching a HTML tag**: or `/^<([a-z]+)([^<]+)*(?:>(.*)<\/\1>|\s+\/>)$/`.

Regex Component

The regular expression or regex is characterizied as a literal, which means the sequence has to be surrounded by slash characters "/". If you take a look at the **Matching a HTML tag** below you will see notice how its surrounded by slash characters :

`/^<([a-z]+)([^<]+)*(?:>(.*)<\/\1>|\s+\/>)$/`

Anchors

In the regular expression take notice of the `^` and the `$` these symbols are known as anchors. The `^` anchor represents a string that starts with the characters that follow it. This could be in one of two ways: . An exact string match, like **^The**, with strings "The" or "The dog" will be a match but remember regex is case sensitive so "the" and "the dog" will not work. . The second way is a scope of possible matches, presented using bracket expressions.

The `$` anchor represents a string that closes with the characters that begin it. Similiar to the `^` anchor, it can be begun with an exact string or scope of possible matches. In our **Matching a HTML tag** regex, the string begins and ends with values that match the pattern below: `<([a-z]+)([^<]+)*`

Quantifiers

Quantifiers are important beucase they impose the restraints on the string that your regex matches. They are known to incorporate the lowest and highest number of characters that your regex is searching for. It's worth noting that quantifiers are selfish in the sense that they match with as many events of certains patterns as possible. In our example we can name some

quantifiers listed : `+` matches the sequence one or more times `*` matches the pattern zero or more times

Grouping Constructs

Grouping Constructs act by examining many parts of a string to decipher the necessary sections that meet the needs of different requirements. The standard way of utilizing the grouping constructor is denoted with the `(())`. Also important to note is every section in the paranthesis is described as a sub expression.

Bracket Expressions

Values that are inside square brackets signifies an assortment of chracters we want to match. While they are called bracket expressions they are also recognized as positive character group, because they highlight the characters we want to incorporate. It's also important to note that its common practice to use a hyphen between alphanumeric characters to signify a scope of those possible characters. So for example in our "Matching an HTML tag" regex you would notice the bracket expression [a-z]. This signifies the string may contain any lowercase letter between a-z. Its imporant to note that this applies to only lowercase letters. `[a-z]` or `[abcdefghijklmnopqrstuvwxyz]` written out.

Character Classes

In regex character classes are declared as a group of characters, that can reside in an input string for fulfilling a match. In our example for Matching an HTML tag regex we see two distinct instances of the character classes. We see the `.` which means matches all characters with an exception to the newline character. We also see the `\s` which means matches a single whitespace character, incorporating tabs and line breaks.

The OR Operator

The OR operator acts as another way of writing logic. The OR operator is typcially used with grouping one or different grouping conventions. For example our regex we could use or operator for `[a-z]` and say `(a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|v|w|x|y|z)`

Flags

Flags are positioned at the closing of a regex, proceeding the second slash, and they explicate the additional functionality or edge for the regex. The Matching an HTML tag regex dosn't include flags

but something to note is there are six optional flags either singular or combined and any in ary order. However there are three that are the most common and thats `g` which stands for the global search , `i` stands for the case insensitive search, and lastly `m` which stands for the multi-line search.

Character Escapes

Character escapes are denoted by the `\` backslash. Character escapes function by escaping characters that alternaitvely would be interpreted literally. In our example we can see that the character escape is used `\/\1` `\s+\/`.

RegEx Reference (todo)

Based on modern regex-directed engines with features like lazy quatifiers and backreferences.

Useful

```
^(?=.*?\bbubble\b)(?=.*?\bgum\b).*
```

```
^(?!.*bubble).*
```

```
^(?!(?:foo|bar)$)\w+$
```

Matching

[cols="^1,4,3"] |=== |RegEx |Matching|Example

|[abc]|A single character: a, b or c| |[^abc]|Any single character but a, b, or c| |[a-z]|Any single character in the range a-z| |[A-Z] | Uppercase letters from A to Z| |[a-zA-Z]|Any single character in the range a-z or A-Z| |[0-9] | Digits from 0 to 9 | |^|Start of line| |$|End of line| |\A|Start of string| |\z|End of string| |\b | Word boundary| |\B | Not word boundary| |< | Start of word| |> | End of word| |. | anything | |\s|Any whitespace character| |\S|Any non-whitespace character| |\d|Any digit| |\D|Any non-digit| |\w|Any word character (letter, number, underscore)| |\W|Any non-word character| |a | a character | |ab | ab string | |a|b | a or b | |a?|Zero or one of a| |a*|Zero or more of a| |a+|One or more of a| |a{3}|Exactly 3 of a| |a{3,}|3 or more of a| |a{3,6}|Between 3 and 6 of a| |i | ignore case | |\d | one digit 0-9 | `ie: foo_\d\d = file_55` |\D | non-digit | |\w | one word,letters, digits or _, | `ie: \wfoo- = foo-, foo-bar` |\W | non-word| |\s | whitespace | `ie: foo\sbarr = foo bar` |\S | non-whitespace | |\x | Hexadecimal digit| |\O | Octal digit| |===

Flags

Regex may have flags that affect the search. For example there are 6 of them in JavaScript:

[cols="^1,8"] |=== |Flag | Explanation

|i | With this flag the search is case-insensitive: no difference between A and a.

|g |With this flag the search looks for all matches, without it - only the first match is returned.

|m |Multiline mode.

|s |Enables "dotall" mode, that allows a dot . to match newline character \n.

|u |Enables full Unicode support. The flag enables correct processing of surrogate pairs.

|y |“Sticky” mode: searching at the exact position in the text. |===

Character

[cols="^1,4,3"] |=== |\t | tab| |\r | return| |\n | new line| |\v | Vertical tab| |\f | Form feed| |\xxx | Octal character xxx| |\xhh | Hex character hh| |\ | escapes regex special character, |`ie: \.\*\+\?\$\^\/\\ = .*\|+? $^/\` |===

Quantifer

[cols="^1,2,4"] |=== |+ | once or more, |`ie: \w-\d+ = img-1, img-23, img-1001` |? | once or none, |`ie: \w-\d? = img-1, img-2, img-` |* | zero or more, |`ie: \w*-\d = imgimg-1, imgimg-2` |{3} | three times, |`ie: \w{3}-\d = imgimgimg-1` |{2,} | two or more times, |`ie: \w-\d{2,} = img-23, img-1001` |[123] | matches class, |`ie: [1-3] = 1,2,3` |q[^x] | negates matches, |`ie: q[^x] = question but not iraq` |===

Anchor

[cols="^1,2,4"] |=== |^ | beginning of line, not in [] |`ie: ^app = appMain.js, appAuth.\|js` |$ | end of line, |`ie: .*?Auth.js$ = appAuth.js` |\A | beginning of string| |\Z | end of string| |===

Logic

[cols="^1,2,4"] |=== |() | capturing group, |`ie: A(nt\|pple) = pple in Apple` |(?) | non-capuring group| |===

Modifier

[cols="^1,4,4"] |=== |(?i) | case-insensitive | `ie: (?i)Monday = Monday, monday, monDAY` |(?m) | multiple lines so ^ and $ match in multiple places, | `ie: (?m)1\r\n\^2$\r\n\^3$ = 1\n2\n3` |(?x) | whitespace mode| |===

Lookaround

[cols="^1,2,4"] |=== |(?= | positive lookahead | `ie: (?=\d{10})\d{5} = 01234 in 0123456789` |(?<= | positive lookbehind | `ie: (?<=\d)cat = cat in 1cat` |(?! | negative lookahead | `ie: (?!theatre)the\w+ = theme` |(?<! | negative lookbehind | `ie: \w{3}(?<!mon)ster = Munster` |===

Replacement

[cols="^1,2,4"] |=== $foo - inserts foo $& - inserts entire match $` - inserts preceding string $' - inserts following string $Y - insert Y'th captured group |===

REGULAR EXPRESSIONS YOU SHOULD KNOW:

[cols="2,6"] |=== |Maching | RegEx

|Username| /^[a-z0-9_-]{3,16}$/ |Password| /^[a-z0-9_-]{6,18}$/ |Hex Value| /^#?([a-f0-9]{6}|[a-f0-9]{3})$/ |Slug| /^[a-z0-9-]+$/ |Email| /^([a-z0-9_.-]+)@([\da-z.-]+).([a-z.]{2,6})$/ |URL| /^(https?://)?([\da-z.-]+).([a-z.]{2,6})([/\w .-]*)/?$/ |IP Address| /^(?:(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?).){3}(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)$/ |HTML Tag| /^<([a-z]+)([^<]+)*(?:>(.*)</\1>|\s+/>)$/ |===

Example & Breakdown

.find html hexidecimal colors like #fff, #aa0099, #CCCCCC [source,bash]

^#(?:[0-9a-fA-F]{3}){1,2}$

[cols="1,4"] |=== |^|anchor for start of string |#|the literal # |(|start of group ^|?:|indicate a non-capturing group that doesn't generate backreferences ^| [0-9a-fA-F]|hexadecimal digit ^| {3}|three times |)|end of group |{1,2}|repeat either once or twice |$|anchor for end of string |===

Examples

[cols="4,3"] |=== |sep[ae]r[ae]te | find `separate` even if it's misspelled seperate |<div\b[^>]>(.?) | match open and closing html `div` tag |\b[A-Z0-9._%+-]+@[A-Z0-9.-]+.[A-Z]{2,4}\b. | match valid email |(?(?=condition)(then1|then2|then3)|(else1|else2|else3)). | if/then/else conditionals |===

.example [source,bash]

[a-f0-9]{3}

- Matches any string consisting of lowercase characters between `a-f` and digits between `0-9`
- At 3 total characters `{3}`.

.example [source,bash]

#?

- Another Quantifier ?,
- matches between 0 and 1 characters of the preceeding expression boundaries #

.example [source,bash]

[a-f0-9]{6}|[a-f0-9]{3}

- Matches any string consisting of lowercase characters between `a-f` and digits between `0-9` At 6 total characters `{6}`
-

OR +

- Matching any string consisting of lowercase characters between `a-f` and digits between `0-9` At 3 total characters `{3}`.

.example [source,bash]

([a-f0-9]{6}|[a-f0-9]{3})

This specifies within the brackets that we are looking for and extracting strings consisting of `a-z` and `0-9` criteria and plating them as Capture Groups.

.example [source,bash]

(/^([a-z0-9_.-]+)@([\da-z.-]+).([a-z.]{2,6})$/)

- In the above pattern, `/^([a-z0-9_\.-]+)` says that email must begin with alpha-numeric characters with only lower case, it may also include ., _, -.
- `@` meaning there must be @ sign after first chars.
- The pattern `[\da-z\.-]+` says that after @ char there must be lowercase alpahabet and may also contain ., -.
- `\.` indicates that there should be a period to separate domain and sub-domain names in the email.
- The pattern `[a-z\.]{2,6})$/` says that the email should end with 2 to 6 lowercase chars. Here 2 indicates the minimum number and 6 indicates the maximum number of characters.

TODO https://gist.github.com/bashworthj/95eee63656342321c248463e1d052970

References:

http://www.regular-expressions.info/quickstart.html

http://www.rexegg.com/regex-quickstart.html

http://en.wikibooks.org/wiki/Regular_Expressions/syntax/posix_basic_regular_expression

http://blog.codinghorror.com/excluding-matches-with-regular-expressions/