

---

# Cloudpipes Documentation

*Release*

**Cloudpipes**

September 20, 2016



<b>1</b>	<b>Contents:</b>	<b>3</b>
1.1	Getting Started . . . . .	3
1.2	Templating (field mapping and conversion) . . . . .	14
1.3	Blueprints . . . . .	20
1.4	Field Colors Guide . . . . .	25
1.5	Scheduling . . . . .	26
1.6	Flow Control . . . . .	29
1.7	Linking Items . . . . .	36
1.8	Current Time and Date/Time Arithmetic . . . . .	46
1.9	Integration Examples . . . . .	47
1.10	Channel Documentation . . . . .	86
1.11	Frequently Asked Questions . . . . .	98
1.12	List of Tier-1 and Tier-2 channels . . . . .	101
<b>2</b>	<b>Indices and tables</b>	<b>105</b>



This is the official Cloudpipes documentation repository.



---

## Contents:

---

## 1.1 Getting Started

### 1.1.1 What is Cloudpipes?

**Cloudpipes** enables you to automate tasks between web application like Salesforce, Slack and Trello. Cloudpipes gives you the ability to execute commands in one system, when something happens in another. For example when a Lead gets updated in Salesforce post a new message to Slack, or when a Card is created in Trello add a new row to a Google Spreadsheet.

Cloudpipes offers a simple visual interface to design “*pipelines*” - flowcharts that specify what and how data flows between *apps*. In addition to it's simplicity Cloudpipes empowers you with the freedom and flexibility to build complex pipelines that span many disconnected systems, have as many pipes (steps) in them, use *logical conditions* (if-then-else), iterate over lists of items with *for-each loops*, *schedule* pipeline execution at selected intervals or just do one-off tasks that would be too tedious to execute manually. Thanks to it's unique features, like *Linking* you can achieve full bi-directional continuous data sync between supported systems in as little as two pipelines!

**Cloudpipes** is the most powerful and flexible services integration platform in existence today !

### 1.1.2 Glossary

#### Pipeline

A **Pipeline** is constructed from simple steps called **pipes**. A Pipeline is the link between your applications. For example - send a “*New Message on Slack*” when a “*New Lead is created in Salesforce*” is what is contained in a pipeline.

#### Trigger

A **Trigger** is an event that can make a pipeline run. For example, if you want to automate sending a slack message every time a new Lead is created in Salesforce, then “New Lead Created” is the Trigger. Cloudpipes also supports pipelines that do not have trigger. Such pipelines can be started either manually, for on-off tasks or can be **scheduled** to be automatically run on set intervals.

#### Action

**Actions** do stuff - send messages, create records in Salesforce, translate text to another language or just about anything that your apps are capable of. For example, if you're sending that slack message, every time a new Lead is created in

Salesforce, the “Post Message to Slack” is the Action.

### Pipe

**Pipes** are the building blocks of pipelines. Each pipe belongs to a certain **channel** and is responsible for doing something - creating a record in Salesforce, uploading a picture or sending an email with Google.

A Pipe can be either a **Trigger** or an **Action**. For example “*Send a Message on Slack*” is an action pipe and “*New Lead Created in Salesforce*” is a trigger pipe.

### Blueprint

**Blueprints** are pre-composed pipelines that are ready to use. You just need to add them to your **Dashboard** and add your **Accounts** for the relevant services. Blueprints save time, eliminate pipeline building errors and help tremendously in getting your apps talk to each other.

### Account

An **Account** allows Cloudpipes to access your data inside another app. Each service like Slack, Salesforce or GMail needs to have an account configured in Cloudpipes in order to be used in building pipelines.

### Field

**Fields** are the attributes of data flowing through the pipelines. A Trello Card name for example is a field, so is the Value of a Salesforce Opportunity and the Due Date of an Asana Card. In the Editor Fields are color-coded by type. For reference see [Field Colors Guide](#).

### The Editor

The Cloudpipes **Editor** is where you create and edit your pipelines. The Editor is simple to use, allowing you to build complex integrations by simply dragging and dropping **Triggers** and **Actions** from the palette on the right.

### Dashboard

The **Dashboard** is where you have an overview of all your existing pipelines, your account and billing data and usage reports.

### Application

An **Application** in the context of Cloudpipes is a web service or application. Salesforce, Slack, Trello and Asana are all web applications. Cloudpipes supports hundreds such applications, allowing you to connect and automate them.

### Channel

A **Channel** represents an Application that has support implemented in Cloudpipes. A Channel is a collection of pipes that represent the Triggers and Actions supported by the Application implementing them. For example Salesforce is a channel in Cloudpipes - a collection of Pipes that represent what you can with Salesforce, the Application in Cloudpipes. So are Slack, GMail, Trello and all the other Applications Cloudpipes supports.



Channels on Cloudpipes belong to either the **Standard**, **Tier-1** or **Tier-2** groups. Using channels from a certain group requires you to be on one of our paid plans. Click [here](#) for a list of channel per tier.

## Built-in Channel

A **Built-in Channels** is a channel that does not have a backing web application, but instead has functionality provided entirely by Cloudpipes. RSS Feeds, NLP based text sentiment analysis, Webhooks are all examples of such channels. For a full guide on built-in channels see [Built-in Channels Guide](#)

## Transaction

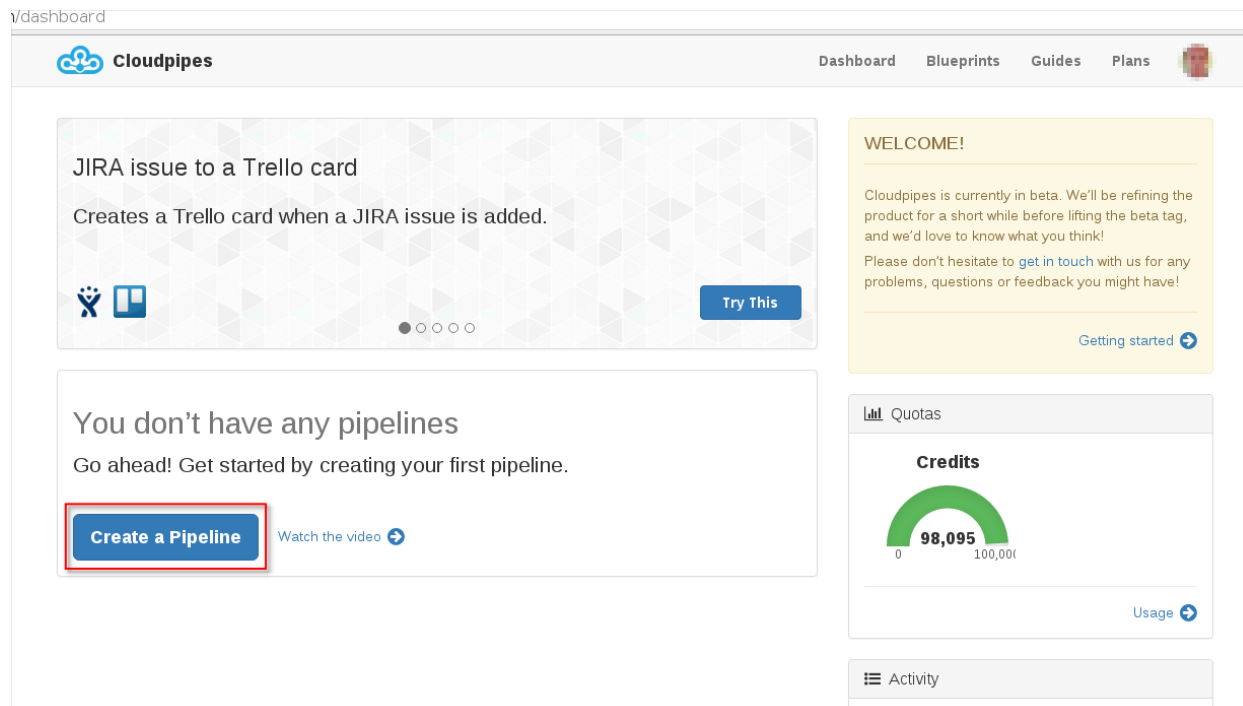
A **transaction** is a successful execution of an action. For instance, if your pipeline creates new users in Intercom when there are new accounts in Salesforce, each action (user created) in that pipeline would count as a single transaction.

### 1.1.3 2 Minute Howto

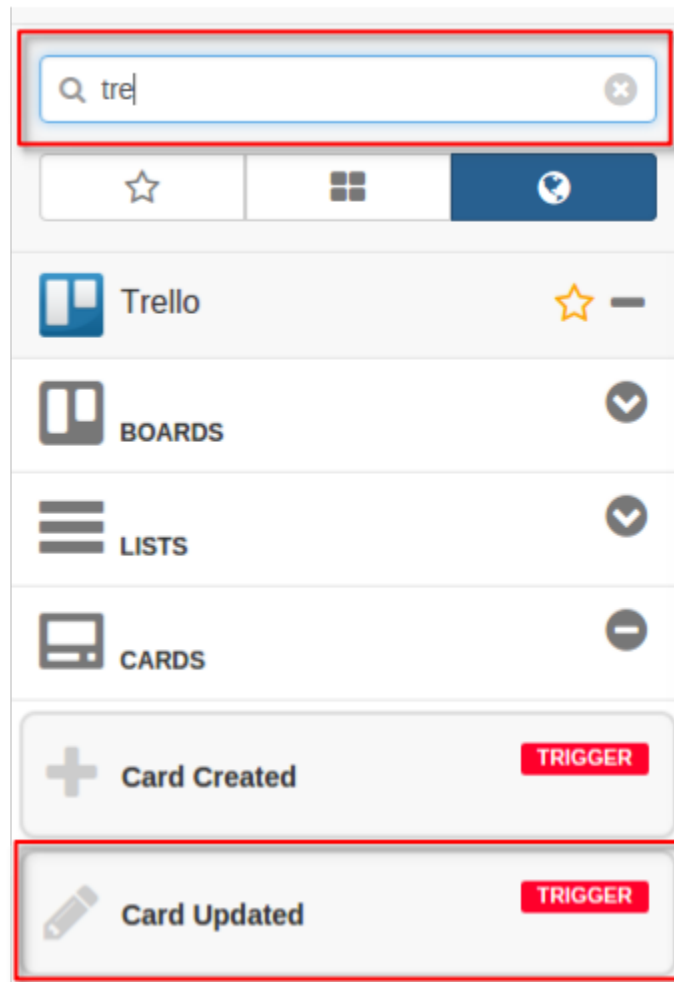
This is a simple tutorial on how to build a *pipeline* with Cloudpipes. In this example we will build a pipeline that will pop a desktop notification whenever a Trello Card on a Board of your choice gets updated.

#### Here's how to do it

First - Make a new pipeline by clicking the big **Create a Pipeline** button on the *dashboard*.

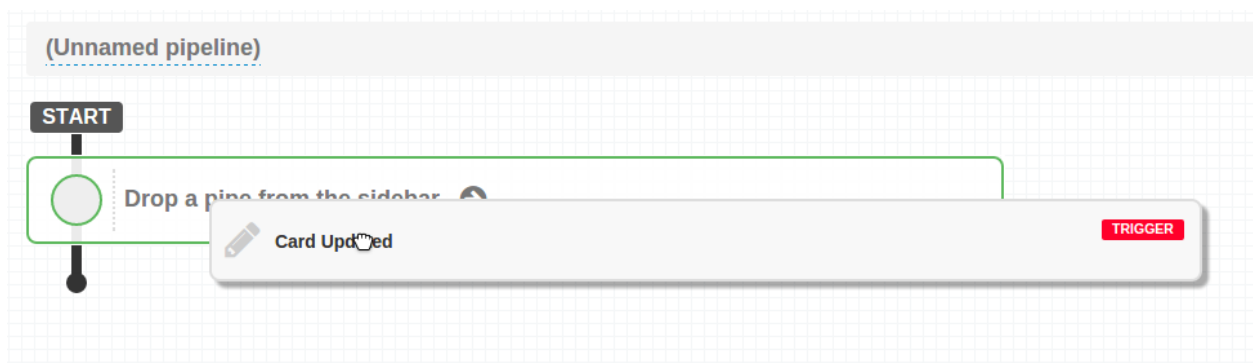


This will open the *Cloudpipes Editor*. To find Trello in the *Channels drawer* to the right, type “tre” in the filter text-box at the top, then expand *Cards* to see the *pipes* related to that.

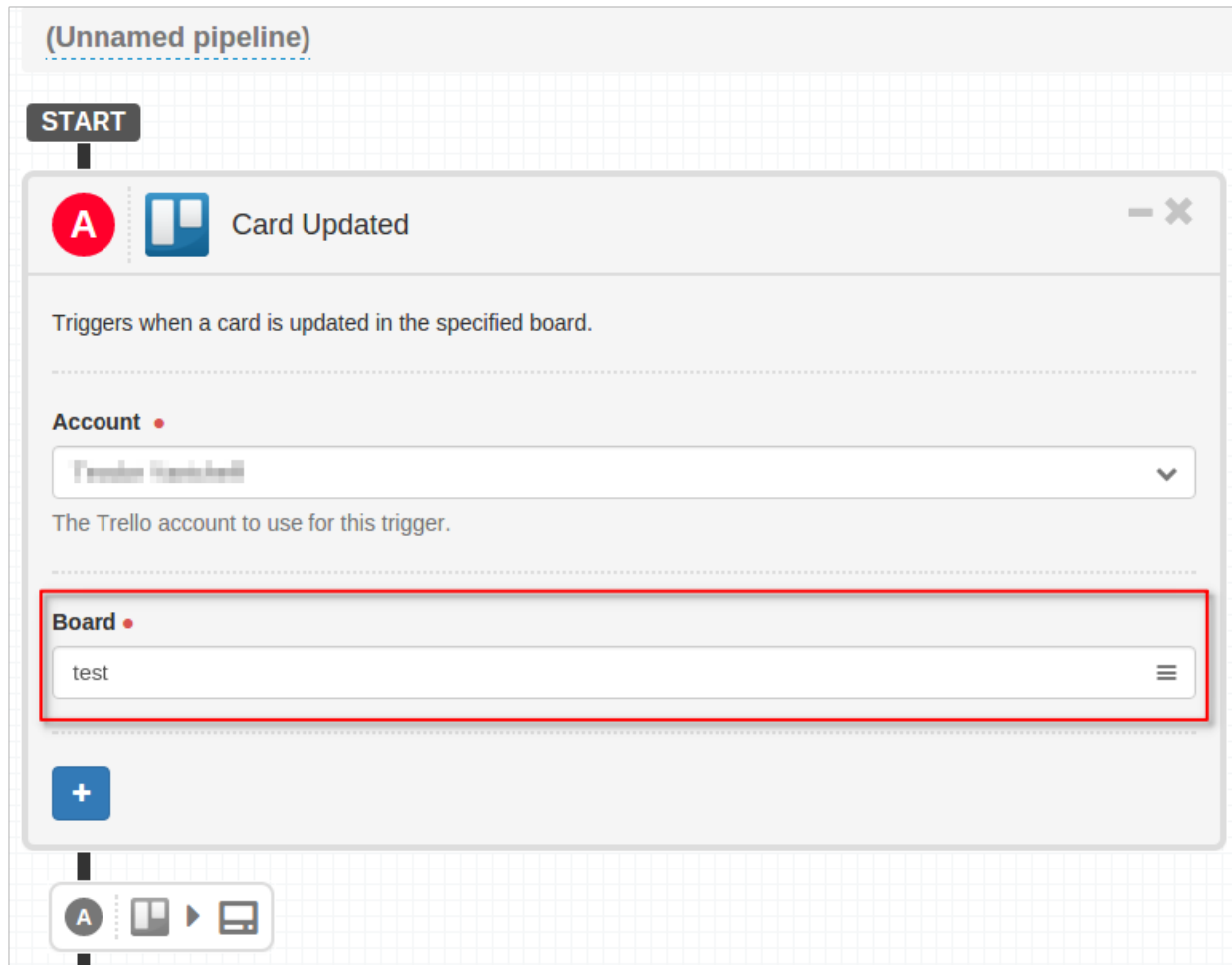


**Note:** If you haven't connected your Trello [account](#) you will see a **Connect** button that will allow you to connect your Trello account to Cloudpipes.

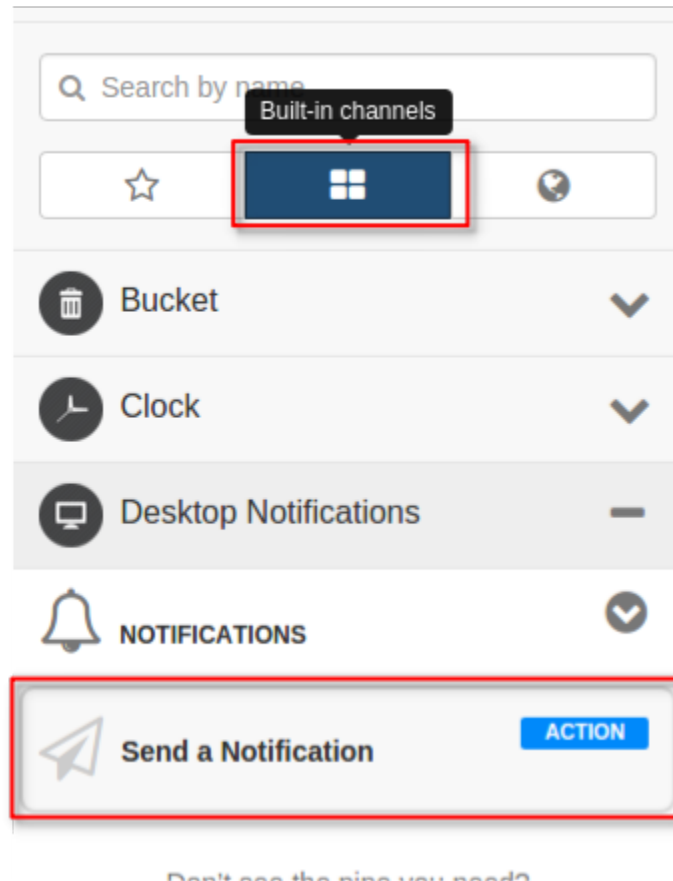
Then drag the *Card Updated trigger pipe* and drop it onto the first slot and currently only slot in your pipeline. A **trigger pipe** is the event that will initiate the execution of your pipeline. It is the starting point of the pipeline. In this example we want the pipeline to start when a Card in Trello was updated.



In the Card Updated pipe select the Trello Board you want to monitor for updated cards :



Next, select the *Built-in Channels* tab in the channels drawer, expand *Desktop notifications* and drag the *Send a Notification* pipe onto the next available slot in the pipeline. This is an *Action* pipe – a one that actually does something.



After you drop it after the Trello Card Updated, you should have something similar to this:

The screenshot shows a pipeline editor for an "Unnamed pipeline". It starts with a "START" node, followed by a "Card Updated" trigger (labeled A). Below this is a "Send a Notification" action (labeled B). The notification action has fields for Title, Body, Icon, and URL. The Title field contains "Cloudpipes". The Body field is empty. The Icon field contains a URL: "https://www.cloudpipes.com/assets/logos/128x128.png". The URL field is empty. To the right of the notification action is a "Fields List" panel, which is highlighted with a red border. This panel contains a list of fields available for drag-and-drop into the notification action's input fields. The fields are organized into sections: "Card", "Board", and "List".

**Fields List:**

- Card**
  - ID: {{a:id}}
  - Name: {{a:name}}
  - Description: {{a:description}}
  - Due Date: {{a:due\_date}}
  - Position: {{a:position}}
  - Updated At: {{a:updated\_at}}
- Board**
  - ID: {{a:board.id}}
  - Name: {{a:board.name}}
  - Background Color: {{a:board.background\_color}}
  - URL: {{a:board.url}}
  - Short URL: {{a:board.short\_url}}
  - Closed: {{a:board.closed}}
- List**
  - ID: {{a:list.id}}
  - Name: {{a:list.name}}

The area to the right of the Desktop Notification action is called the *Fields List*. It gives you access to data that is available in all the previous pipes you have added to the current pipeline you are editing. Drag and drop the Card's name field into the Title parameter for the Desktop notification and Card's Description into the Body textbox. This way your notification will have as title the name of the updated Trello card and it's Body will be the Description of the Card.

(Unnamed pipeline)

START

A Card Updated

B Send a Notification

Sends a desktop notification.

Title •

Updated: "{{a:name}}"

The title that will be shown within the notification.

Body

Text representing an extra content to display within the notification.

Icon

https://www.cloudpipes.com/assets/logos/128x128.png

The URL of an image to be used as an icon by the notification.

URL

The URL to open if the notification is clicked.

More

Card

Toggle samples

ID {{a:id}}

Name {{a:name}}

Description {{a:description}}

Due Date {{a:due\_date}}

Position {{a:position}}

Updated At {{a:updated\_at}}

Board

ID {{a:board.id}}

Name {{a:board.name}}

Background Color {{a:board.background\_color}}

URL {{a:board.url}}

Short URL {{a:board.short\_url}}

Closed {{a:board.closed}}

List

ID {{a:list.id}}

Name {{a:list.name}}

All that's left to do is to give the pipeline a name by clicking on "Unnamed Pipeline" near the upper left corner of the Editor...

Desktop notification on Tre|

START

A Card Updated

... and turn it **ON** with the switch control to the right.

Cloudpipes

(Unnamed pipeline) ON ↺ ?

**START**

**A** Card Updated

**B** Send a Notification

Sends a desktop notification.

**Title**

Updated: "{{a:name}}"

The title that will be shown within the notification.

**Body**

{{a:description}}

Text representing an extra content to display within the notification.

**Icon**

https://www.cloudpipes.com/assets/logos/128x128.png

The URL of an image to be used as an icon by the notification.

**URL**

The URL to open if the notification is clicked.

[More](#)

**Card**

[Toggle samples](#)

**ID** {{a:id}}

**Name** {{a:name}}

**Description** {{a:description}}

**Due Date** {{a:due\_date}}

**Position** {{a:position}}

**Updated At** {{a:updated\_at}}

**Board**

**ID** {{a:board.id}}

**Name** {{a:board.name}}

**Background Color** {{a:board.background\_color}}

**URL** {{a:board.url}}

**Short URL** {{a:board.short\_url}}

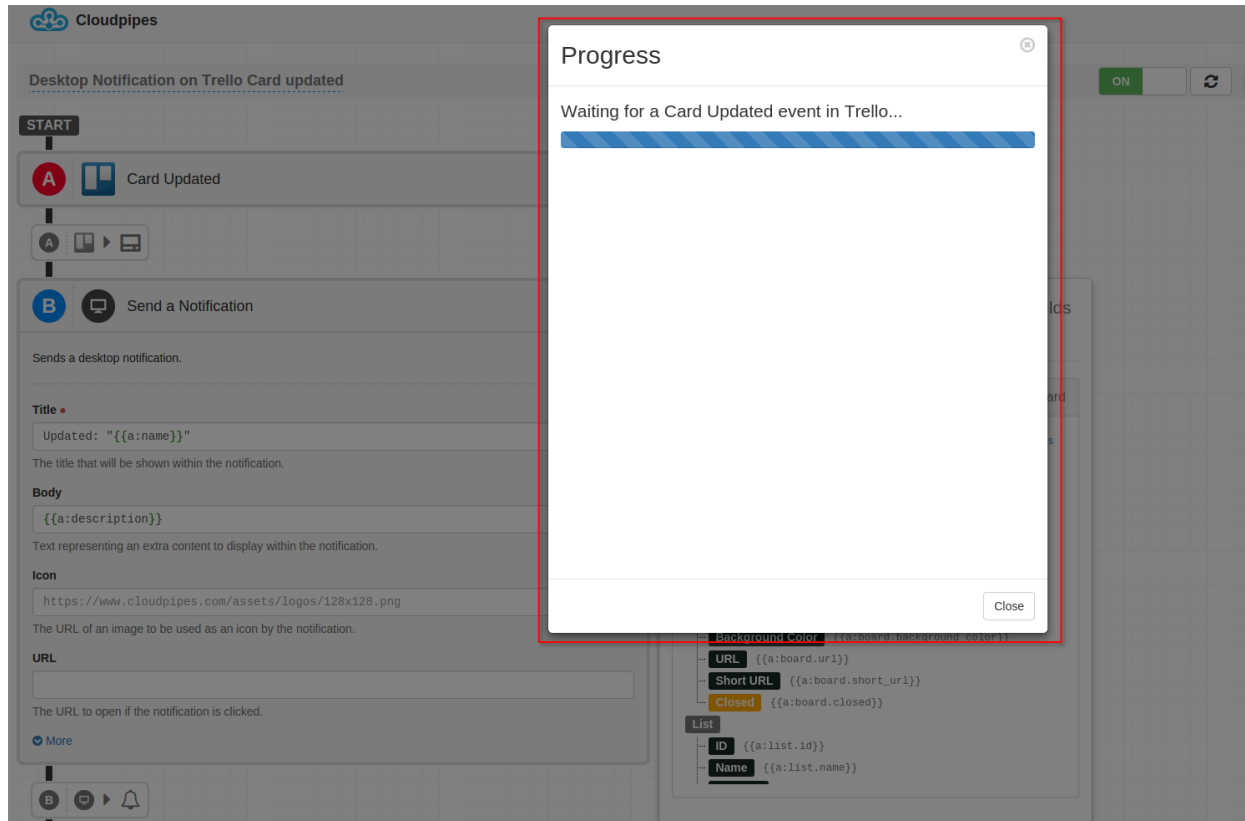
**Closed** {{a:board.closed}}

**List**

**ID** {{a:list.id}}

**Name** {{a:list.name}}

The window that will pop is called the **Progress Dialog**. It is used to debug pipelined execution. In this simple tutorial there is no need to delve into too much detail, just know that it exists.



Now, it's time to see that all that we've created so far works as expected. Go to the Trello Board you have configured in the pipeline previously and update a card. Go back to the Progress dialog in the Editor. Shortly after updating the Card you should see something like this :



# Progress

Pipeline running.

Card Updated

1 credit

description	Description
labels	(Empty List)
list	566c0d39301d1b2e09d068b0
updated_at	2016-02-08T09:56:36.241000+00:00
board	566c0d2735414c354c9e8a9c
members	(Empty List)
position	163840
closed	No
id	569fb0d435e684690517e9d7
name	dfgdfgdfgdfg

Sent a Notification

10 credits

body	Description
sticky	No

Close

**Congratulations!** You have now successfully completed your first pipeline with Cloudpipes!

You can now close the Progress Dialog and go back to the Dashboard. Your pipeline is now complete and will continue to run every time a Card gets updated in Trello.

### Conclusion

This is all there is to it really. In this very brief tutorial we've covered all the basics you need to know to be able to effectively use Cloudpipes – Channels, Pipes – Triggers and Actions, Fields and also an overview of the interface.

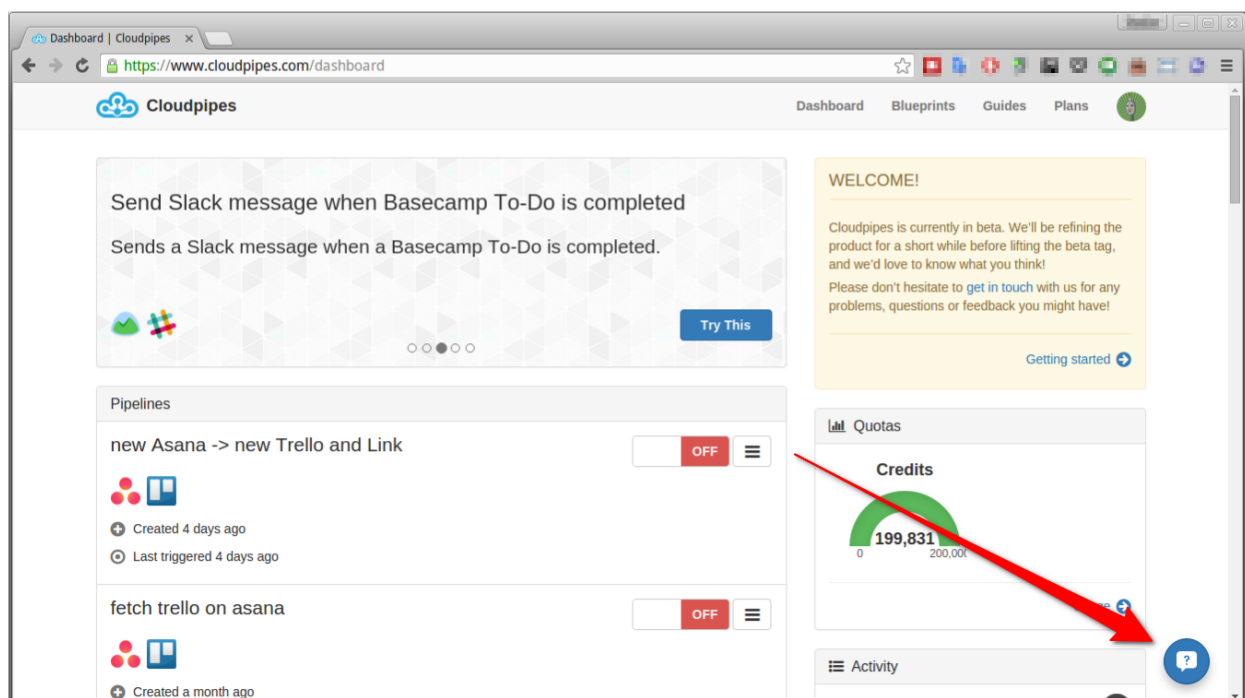
Should you need any assistance, as usual just drop us a line at [support@cloudpipes.com](mailto:support@cloudpipes.com) or reach us in the in-app chat.

Happy integrating with **Cloudpipes**!

### 1.1.4 Contacting Us

The best way to contact us is using [support@cloudpipes.com](mailto:support@cloudpipes.com).

If you already have an account we are even closer to you - just click on the chat-widget you will find throughout the app



If you're contacting us about a specific pipeline you're having problems with or need advice, please don't delete or modify the pipeline! That helps us troubleshoot the problem with your pipeline.

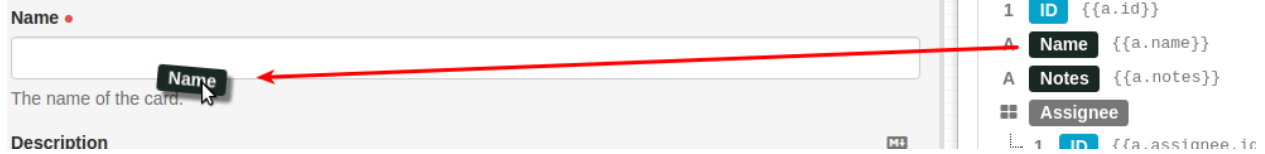
Currently we offer phone support only on our Enterprise Plan.

## 1.2 Templating (field mapping and conversion)

Cloudpipes uses [jinja2](#) templating language for pipe and pipe filter fields. This allows to do complex transformation on data when passing information between channels. The simplest way to use a template template is to just drag and drop a field from the exchange picker.

For example dragging the `name` field:

have specified the board by explicit ID (or dropdown) above. If you need to specify both board and list dynamically, you will need to use the respective lookup pipes, to fetch the board and lists by name, and use the exported IDs here.



results in the following template in the Name field of the Pipe:



You can either drag and drop fields, or just type the jinja2 syntax yourself.

You can add text to the template like:

```
{{a.name}} Some Text After the name
```

Or even combine two fields into one like this:

```
{{a.name}}
{{a.description}}
```

Assuming the `a.name` is “Hello World!” and `a.description` is “Hi there!” the last template will result in:

```
Hello World!
Hi there!
```

You can read more on the general jinja2 syntax in the official documentation [here](#). The sections below describe some features which are useful for Cloudpipes pipelines, and some of the Cloudpipes-specific jinja objects and filters.

### 1.2.1 Filters

**Filters** are used to transform the output. You can use pipe character `|` in the curly braces after the value to apply a filter. For example you can use the built-in jinja2 filter `upper` to make a string uppercase:

```
{{a.name}}
{{a.name|upper}}
```

This results in:

```
Hello World!
HELLO WORLD!
```

Cloudpipes support all the built-in jinja2 filters, for which you can read in the [official documentation](#). The following sections give of both Cloudpipes-specific filters and built-in filters, useful for building your pipelines.

### 1.2.2 Numbers - calculation and conversion

You can do simple arithmetic in templates like:

```
Incremented by 1 is: {{a.count + 1}}
Discounted value is: {{a.sum - (a.sum * a.discount_percents / 100)}}
```

Some channels return numbers as text fields, which will result in an error if you try to use them for calculations. For these you can use the `int` and `float` filters, which convert text to an integer or floating-point number:

```
text to integer plus one is: {{a.text_integer|int + 1}}
text to float times 1.35 is: {{a.text_float|float * 1/35}}
```

### 1.2.3 Date and Time arithmetic

See [Current Time and Date/Time Arithmetic](#).

### 1.2.4 HTML fields and extracting text from HTML

#### Markdown fields

Some channels expect HTML in some of their fields. If you see the `M` symbol this means that we have added markdown support to this field, and any markdown will be automatically converted to HTML:



You can read more about markdown syntax [here](#)

#### Remove HTML tags

Some channels return fields which contain HTML tags. If you want to transfer these fields to a channel which does not support HTML, you may want to extract just the text. There are two filter `text`, which just removes all html tags and formatting, and `html2text`, which converts the text to markdown, which can be use in Markdown fields. Since markdown is also relatively human-readable you may want to use `html2text` anyway to keep some of the formatting. As a rule of thumb use `text` for single-line fields and `html2text` for html fields with more than one line.

```
Body no formatting:
{{a.body|text}}

Markdown (formatted) body:
{{a.body|html2text}}
```

#### Escape HTML

Some channels expect HTML input in their fields, but some characters are invalid in HTML unless escaped. To convert the characters `&`, `<`, `>`, `'`, and `"` in string `s` to HTML-safe sequences use the `escape` filter. For example imagine `a.name` is "Johnson & Son". If you use just `{{a.name}}` in an HTML field, you may receive an error about bad encoding of the `&` character. In order to avoid this error you can use:

```
{{a.name|escape}}
```

Which will result in:

```
Johnson & Son
```

### 1.2.5 Appending to list field in an Update pipe

Some pipes have list fields such a tags, which you don't want to replace completely in your update pipe, but you prefer to append to the existing list. For example if you have a resource which has Tags field and you want to add a new tags, if you just write the new tag in the field it will replace all existing tags:

```
new_tag
```

In order to “add” the tag to the existing ones you can use the `append` filter like this:

```
{{a.tags|append('new_tag')}}}
```

You can also append multiple tags like:

```
{{a.tags|append('new_tag1', 'new_tag2')}}}
```

### 1.2.6 Template If-Then-Else

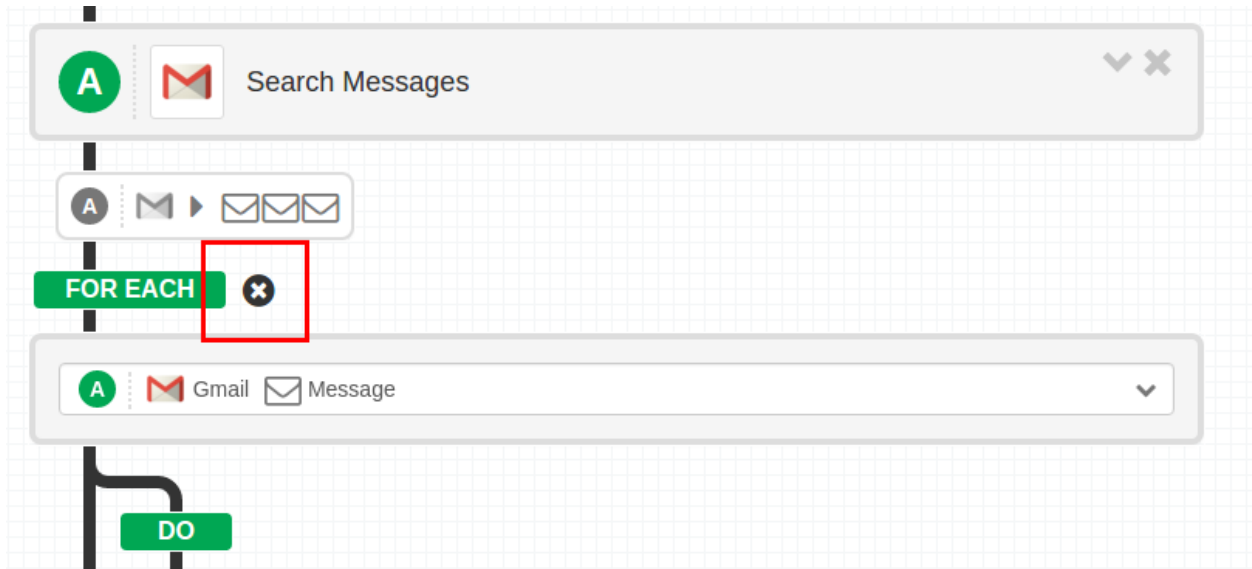
Jinja supports if-then-else flow-control (click [here](#) for their documentation). You can use if-else to conditionally put one value or another:

```
{% if a.first_name %}
{{a.last_name}}
{% else %}
{{a.first_name}}
{% endif %}
```

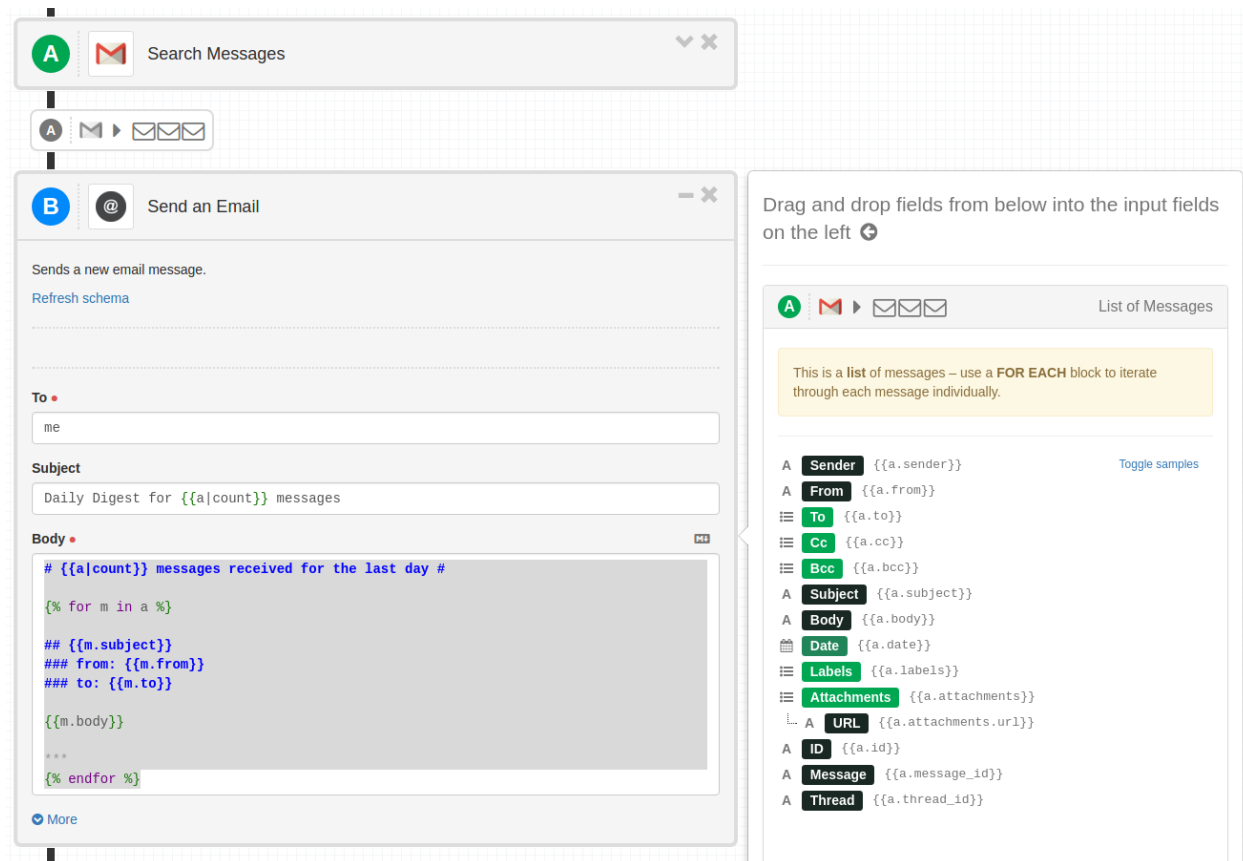
This will put `a.first_name` in the field if it set (non-empty), otherwise it will put `a.last_name`

### 1.2.7 Summarize Search Result to a single field

You can use [jinja for-each loop](#) to summarize information of a search pipe to a single field. Here is an example from our [Daily digest from a Gmail label](#) blueprint (more about blueprints [here](#)). In order to summarize the information from the search you need to remove the Pipeline for-each loop by clicking on the X in the editor:



You can then use the for-each loop in the body of the *Send an Email* pipe, and combine it with markdown syntax to achieve a formatted summary of all results from the Search Pipe:



This will create an email which contains all the emails found from the Gmail pipe. The template code for the Body field is:

```
# {{a|count}} messages received for the last day #

{% for m in a %}
```

```
## {{m.subject}}
### from: {{m.from}}
### to: {{m.to}}

{{m.body}}

***
{% endfor %}
```

Notice that in this case `a` is the full export of the search pipe, which means that it is a list. We use the `count` filter to find out how many elements there are in the list `{{a|count}}` messages received for the last day.

In the iteration over message we create a temporary variable `m`: `{% for m in a %}`, so in any further references to the a single message we use `m` instead of `a`, like `{{m.body}}`.

### 1.2.8 Clear a field in an Update pipe

Cloudpipes ignores fields with empty values in pipe's mapping, i.e. it does not send them to the remote service, to avoid clearing a remote field accidentally. This however is a problem when you want to explicitly clear this field in an Update pipe. For example the following pipe won't do anything to change the Trello card's description:

**Update a Card**

Updates an existing card.

➔ There must be a Trello pipe returning a Card higher up in the pipeline.

[Refresh schema](#)

---

**Card** •

**A** Trello Card

The target card on which to act upon.

---

**Description** [Remove](#)

The description of the card.

**+**

Instead to clear the description you need to specify the special template value `{{CLEAR}}` like this:

B

Update a Card

Updates an existing card.

There must be a Trello pipe returning a Card higher up in the pipeline.

Refresh schema

Card

A

Trello

Card

The target card on which to act upon.

Description

{{ CLEAR }}

The description of the card.

+

Link (advanced)

Select a resource

Link this card to another resource instance, so that later you can fetch one from the other and vice-versa. More information [here](#).

You can combine the clear value with conditions:

```
{% if a.priority != 'no priority' %}{{a.priority}}{% else %}{{CLEAR}}{% endif %}
```

In this case one channel has explicit value of “no priority” when nothing is selected, and the other expects empty string for priority when nothing is selected.

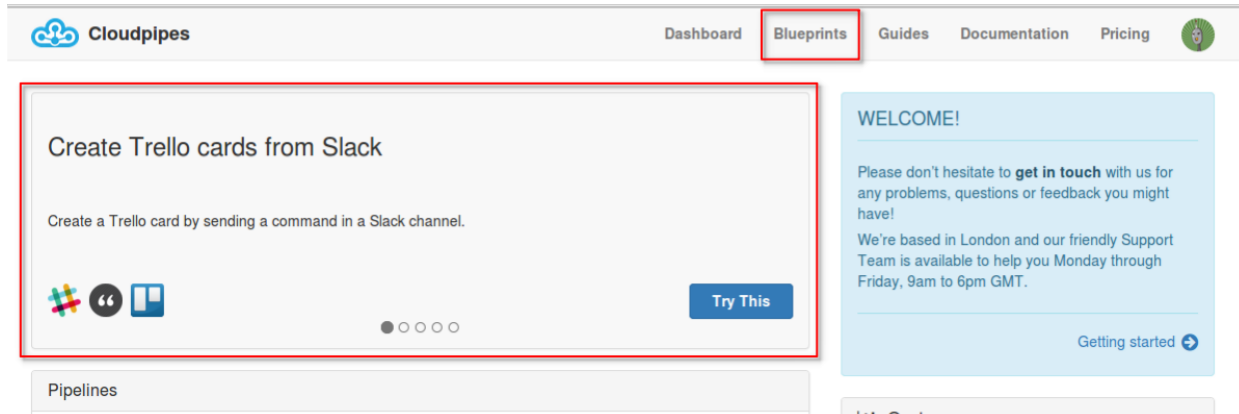
## 1.3 Blueprints

**Blueprints** are pre-composed *pipelines* that are ready to use. You just need to add them to your dashboard and configure your *accounts* for the relevant services. Blueprints save time, eliminate pipeline building errors and help tremendously in getting your apps talk to each other. Cloudpipes offers a large collection of blueprints for most of the pipelines Cloudpipes supports.

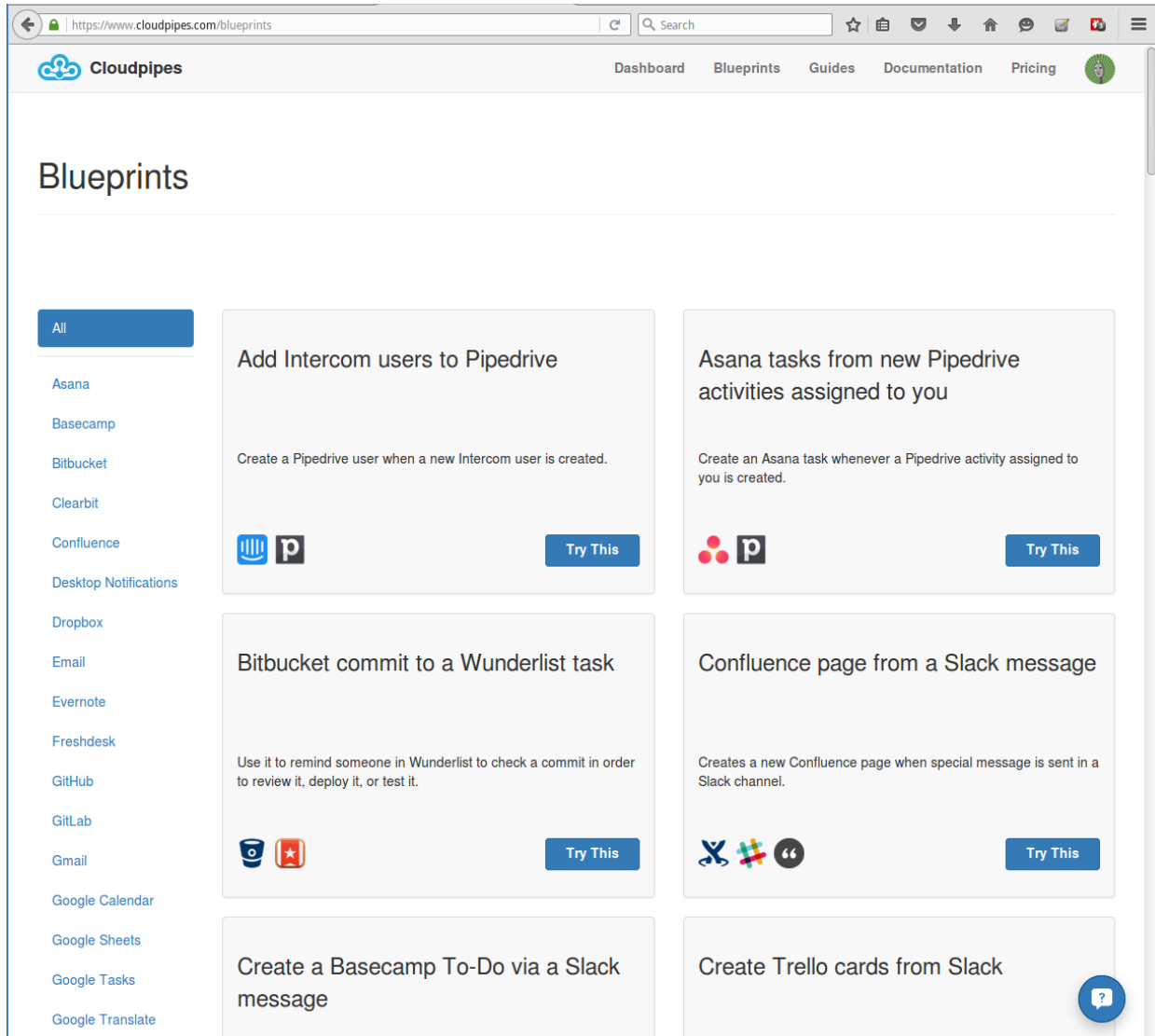


### 1.3.1 Adding a pipeline from a blueprint to your account

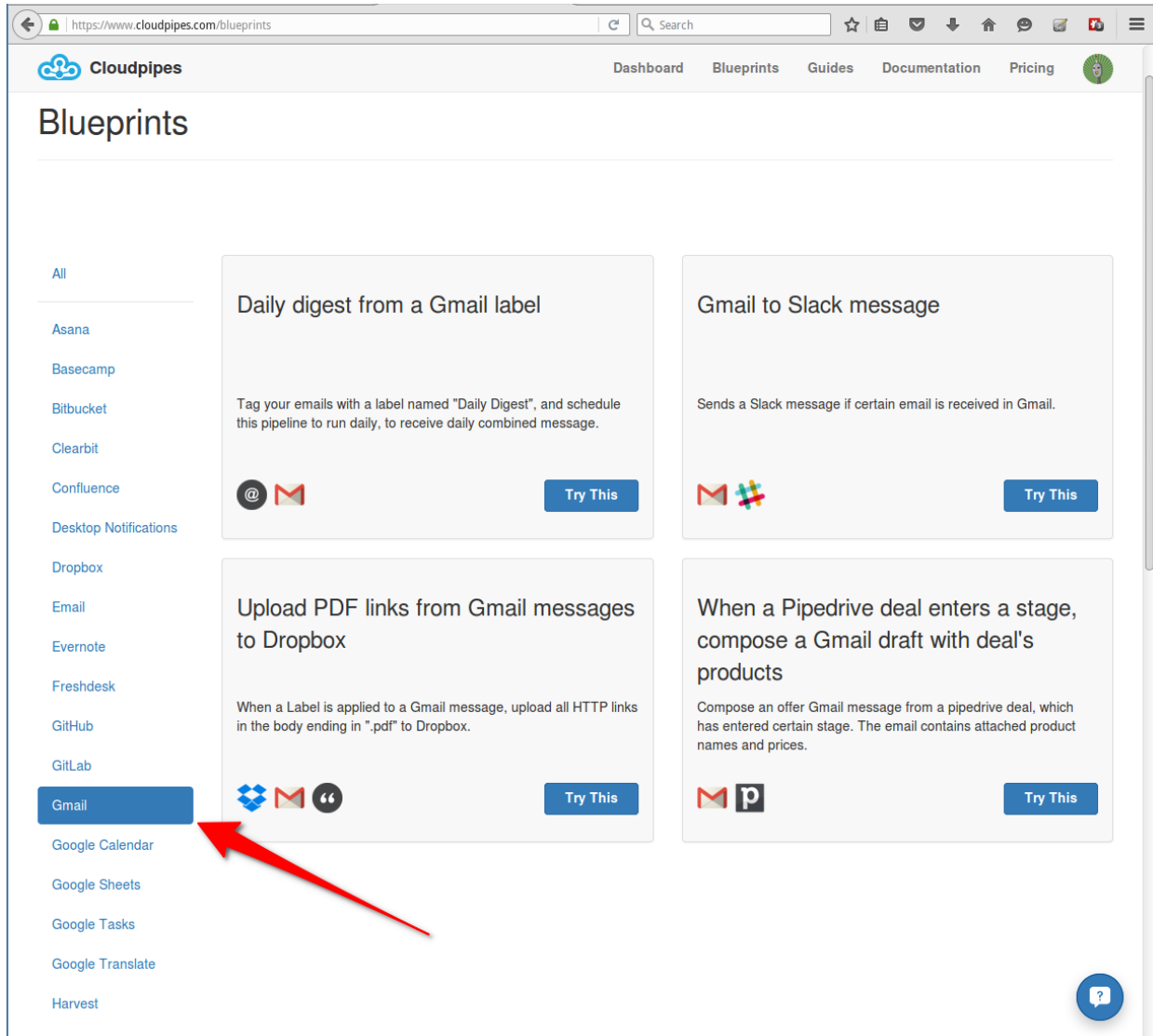
Adding a pipeline base on a blueprint to your account is easy. For easy access we have added a carousel demo containing blueprints that you may find useful on the *Dashboard*. If you do like one of those just click on the **Try This** button to have it immediately in your account. In order to get to the full library of blueprints just click on **Blueprints** from the header navigation bar :



Once you have the Blueprints page open you will see a list of the channels on the left and a list of the available blueprints on the right. All blueprints have a short descriptive name that explains in brief what the blueprint implements, and also a longer description that goes in to more detail. Each of the blueprint also has icons of the channels used in this blueprint and most importantly a **Try This** button that will make a pipeline based on that blueprint in your account.



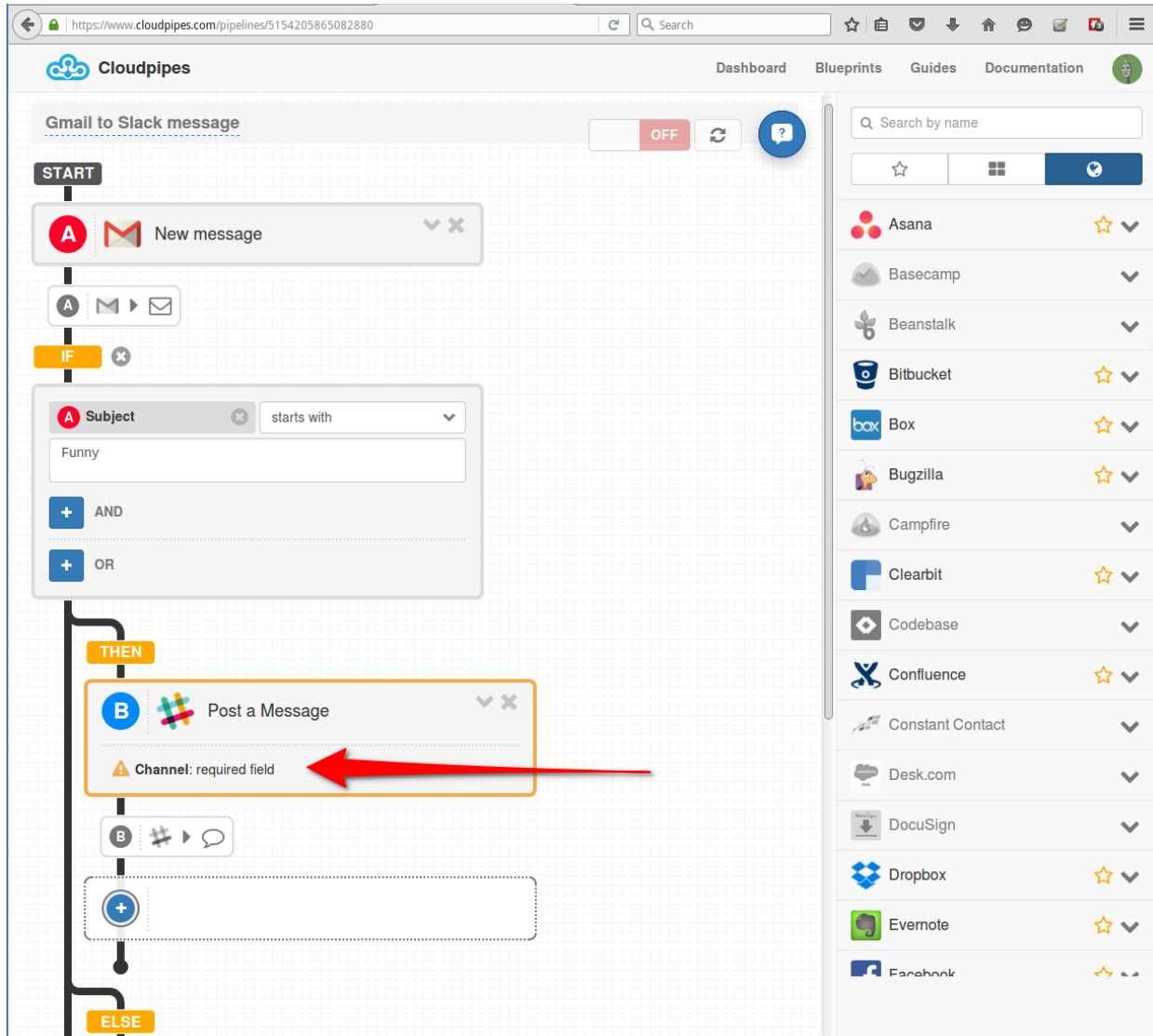
The channels list on the left is also a filter for the list of blueprints, so you are interested in say blueprints that involve say *GMail*, all you need to do is to find *GMail* in the list and click it. This will filter the blueprints shown on the right to only the ones that have the *GMail* channel in them.



As an example we are going to try the “GMail to Slack message” blueprint, by clicking on the **Try This** button. Once you click it, a pipeline with the same name will be created in your account and will open in the *Editor*.

### 1.3.2 Configuring the necessary fields

There’s only one thing left to do in order to start using this pre-assembled pipeline - select the *accounts* you want to use and configure personal details to the pipes that need it.



Adding a blueprint based pipeline to your account will try to automatically select accounts for the pipes in the blueprint for which you already have accounts configured. If you need to use a different account than the one automatically chosen, just open the relevant pipe and select another account. You will also see that some pipes have yellow exclamation marks to them – this shows you that some personalised information needs to be configured. In the example above you will need to select the Slack channel you want the messages posted to. Also note that I decided to go with the default choice of accounts to use for both GMail and Slack and since I had them already configured I did not have to do anything other than select the Slack channel to post the messages to.

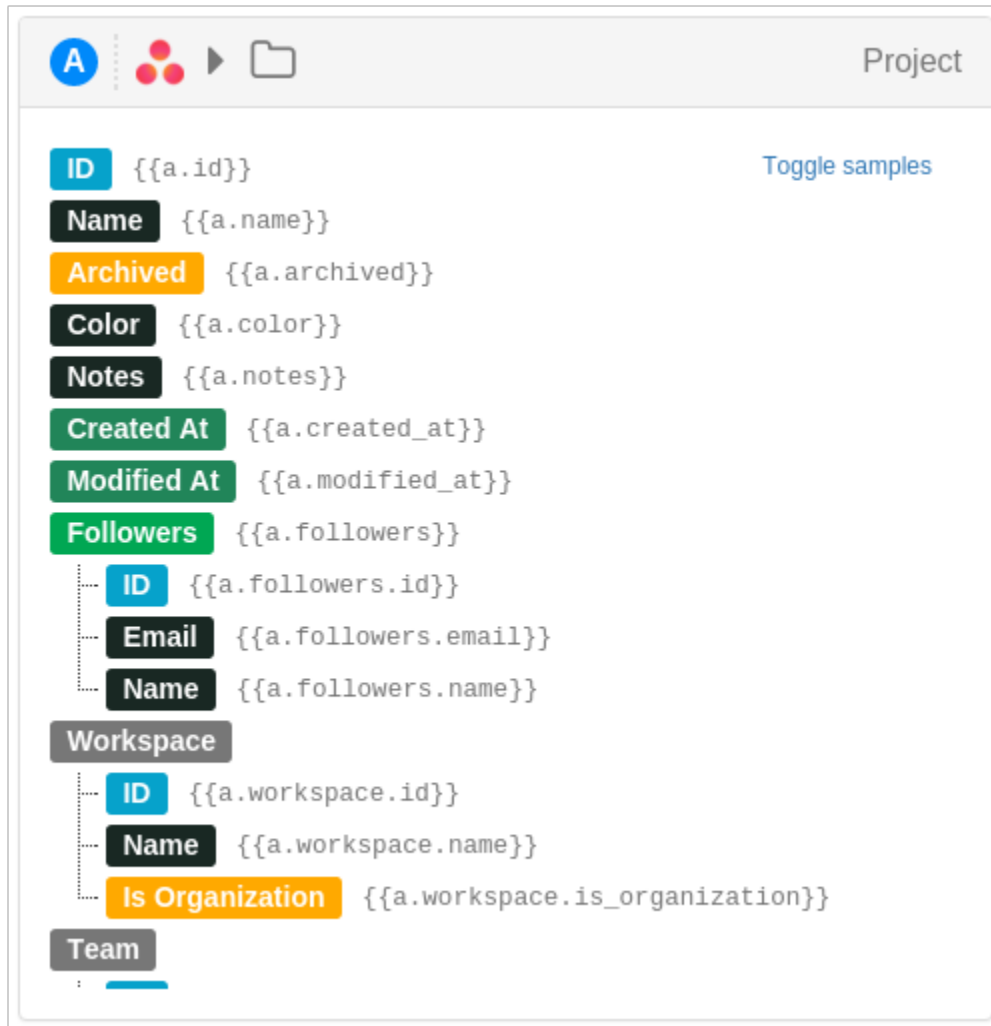
### 1.3.3 Conclusion

This is all there is to it really. By using a Blueprint we have created a full-blown pipeline in only a few clicks. In this brief tutorial we saw how to access the Blueprints page, browsed through the catalog of Blueprints, built a pipeline based on a blueprint and configured the needed personalised options.

Should you need any assistance, as usual just drop us a line at [support@cloudpipes.com](mailto:support@cloudpipes.com) or reach us in the in-app chat.

Happy integrating with **Cloudpipes**!

## 1.4 Field Colors Guide



In the fields list boxes that have the parameters you can drag into your pipes you will notice that various fields have different colors. Cloudpipes uses color to present the **type** of the respective field.

### 1.4.1 Field Colors

Color	Example Field	Meaning	Example
Black	Name	String	"Card Name"
Blue	ID	Number	123.5
Orange	Archived	Boolean	True   False
Dark Green	Created At	Date/Time	2016-04-01T22:44:55.0000
Gray	Workspace	Nested Object	
Light Green	Followers	List of items	

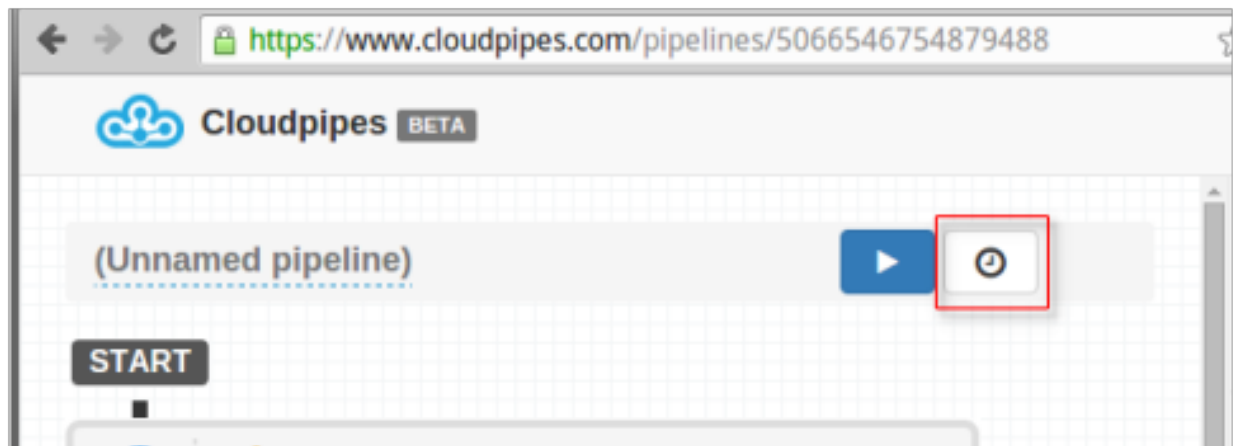
Should you need any assistance, as usual just drop us a line at [support@cloudpipes.com](mailto:support@cloudpipes.com) or reach us in the in-app chat. Happy integrating with **Cloudpipes**!

## 1.5 Scheduling

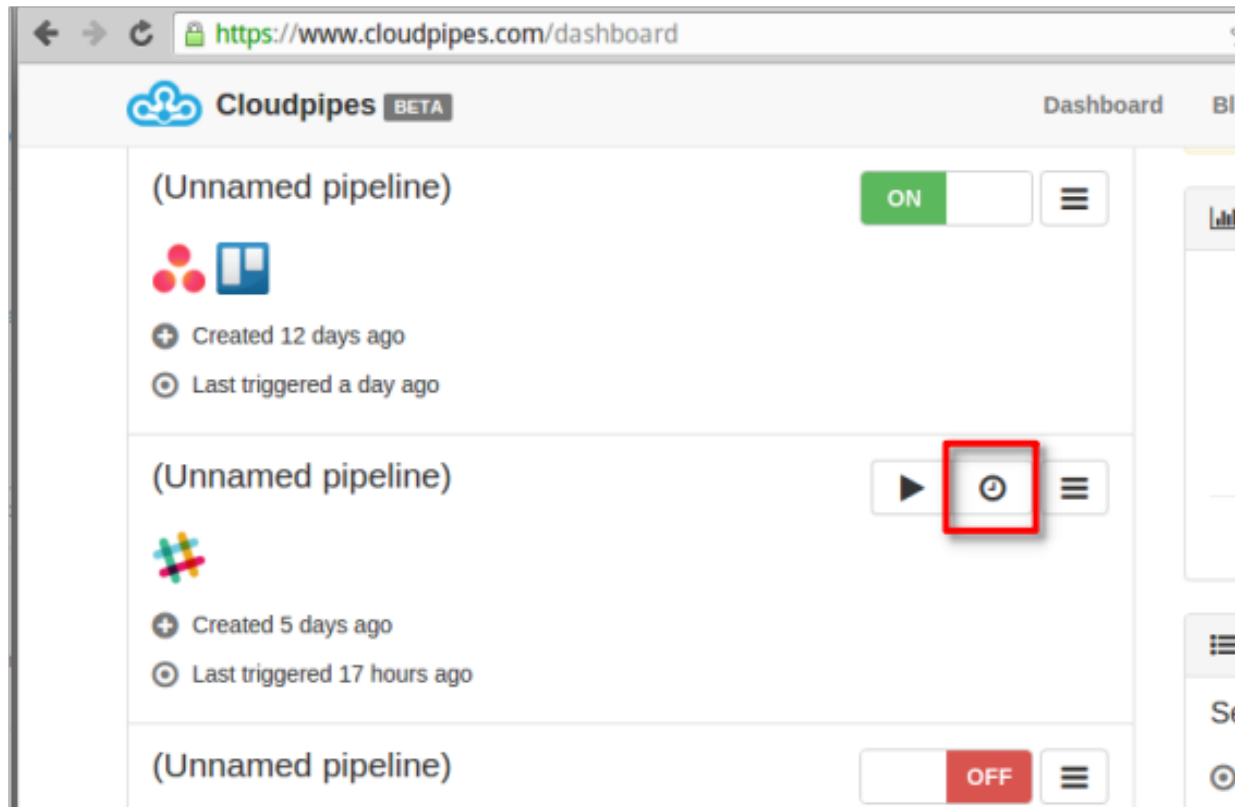
*Pipelines* that do not have a trigger, only actions in them can be started manually with the *play* button you will see next to the pipeline name on the *dashboard* or near the upper right corner while you are in the *editor*. You can also **Schedule** a pipeline – have it execute automatically in defined intervals. Without scheduling you will have to manually start pipelines that do not have a trigger in them.

### 1.5.1 How to do it

The scheduling dialog is available either from within the editor :

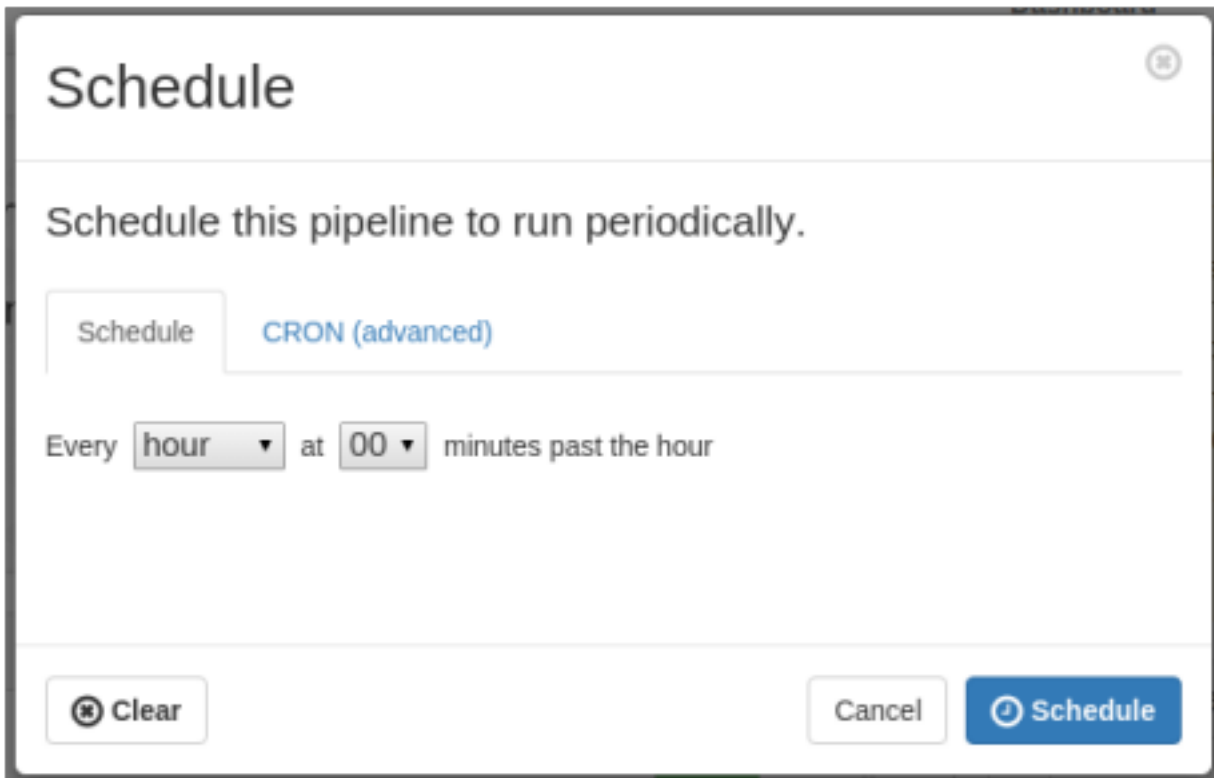


Or from the dashboard, where you have all your pipelines listed :



Clicking on the *clock* button will open the **schedule** dialog. There are two ways in which you can schedule a pipeline to run

## The simple way



**Schedule**

Schedule this pipeline to run periodically.

Schedule [CRON \(advanced\)](#)

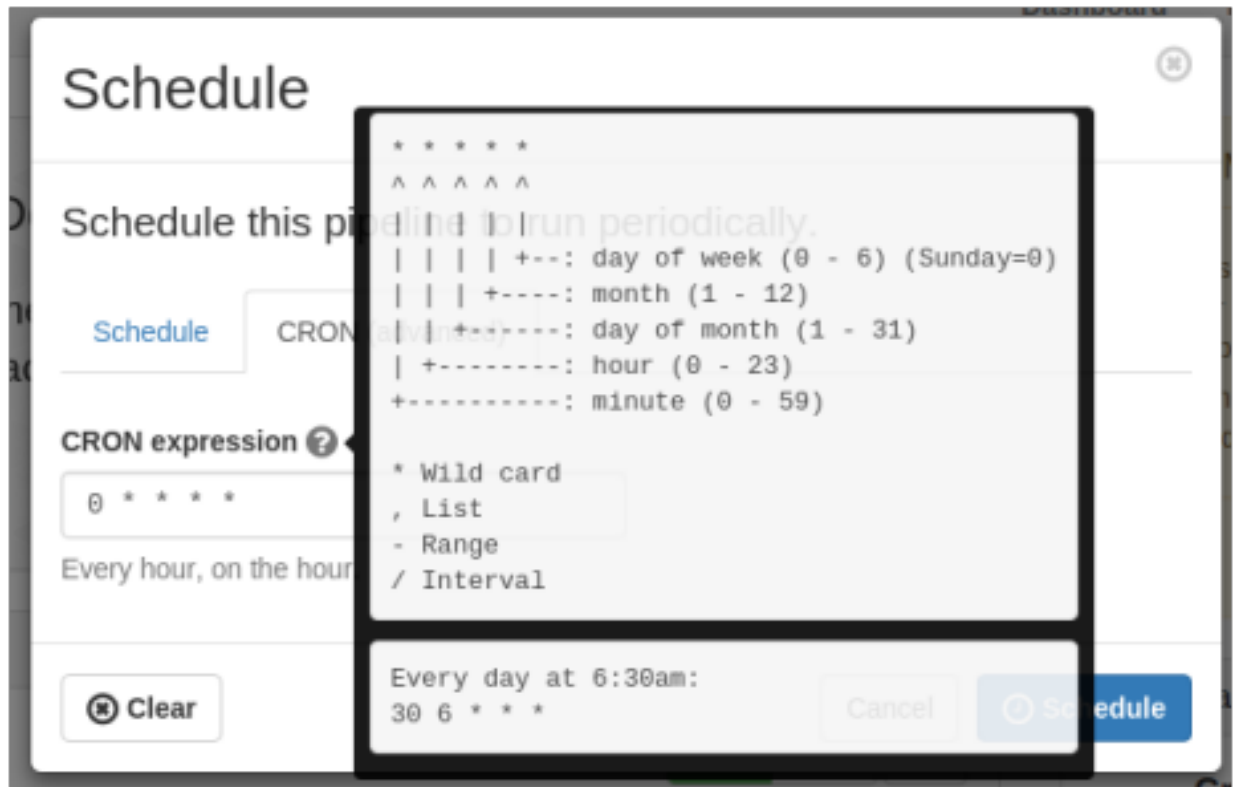
Every hour at 00 minutes past the hour

✕ Clear Cancel 🕒 Schedule

Using the simple scheduling option allows you to have the pipeline run every *minute*, *hour*, *day*, *week*, *month* and *year* and gives you the flexibility to select when exactly, within the given interval, you want your pipeline to be started.



## The CRON-like scheduler



Cloudpipes also supports a full-blown CRON syntax scheduler. Using it will allow you to have the ultimate flexibility in defining when do you want the pipeline started.

### 1.5.2 Conclusion

Scheduling trigger-less pipelines is a very powerful concept that also adds to the already broad set of functions Cloudpipes provides. It is also one of the way to ensure you integrations will execute without any intervention by you.

Should you need any assistance, as usual just drop us a line at [support@cloudpipes.com](mailto:support@cloudpipes.com) or reach us in the in-app chat.

Happy integrating with **Cloudpipes**!

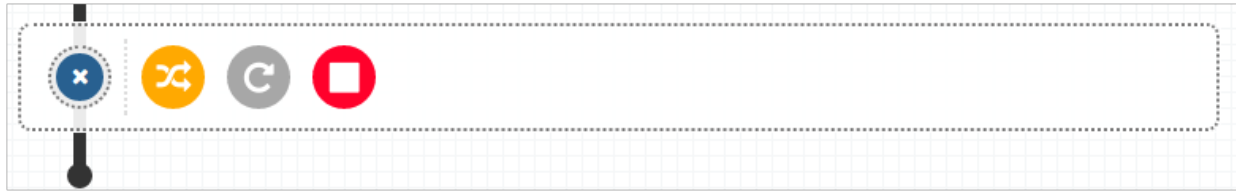
## 1.6 Flow Control

One of the more advanced features **Cloudpipes** supports is pipeline execution flow control. It allows you to implement logical conditions in the form of if-then-else blocks, loop over lists of items (for-each) and even forcibly stop the execution of a *pipeline*.

These are the basic building blocks of any computer program and they give you the ultimate power and flexibility in designing and building pipelines.

## 1.6.1 Adding a flow control statement to a pipeline

Available flow control blocks are hidden under the plus icon on any empty *pipe* block in the pipeline *editor*. Hovering with the mouse over the “+” will expand the flow-control blocks list.

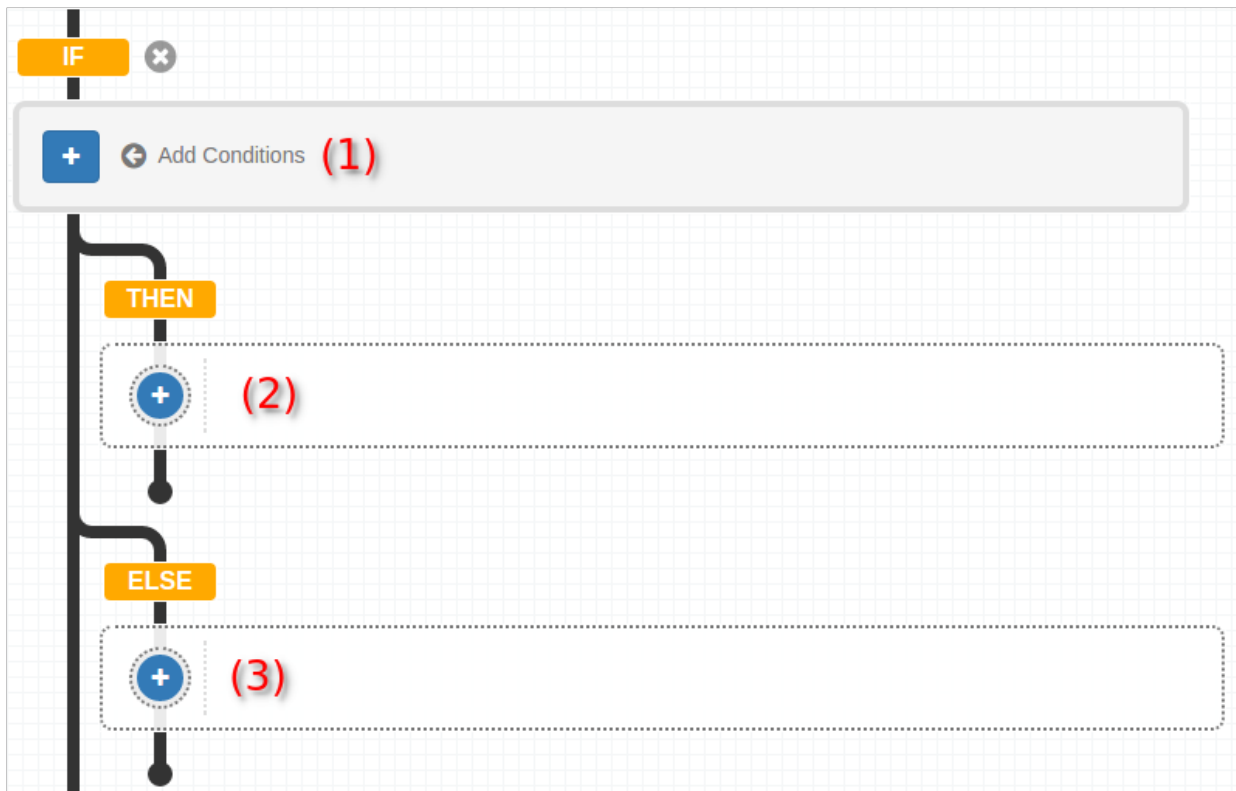


**Note:** The very first pipe slot in a pipeline will not have these.

## 1.6.2 Logical Conditions

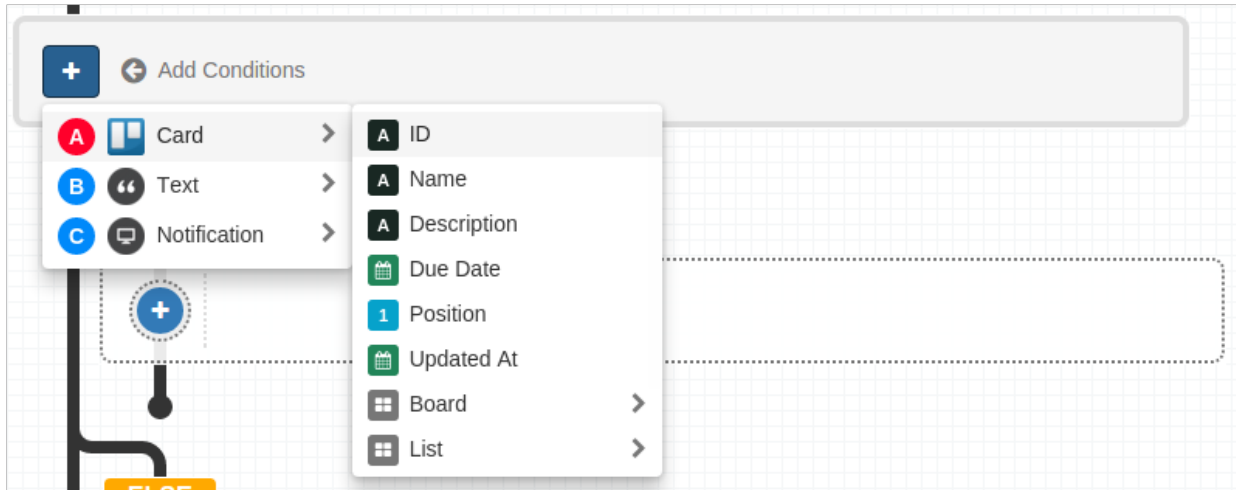


Clicking on this icon will insert a logical condition block in your pipeline.

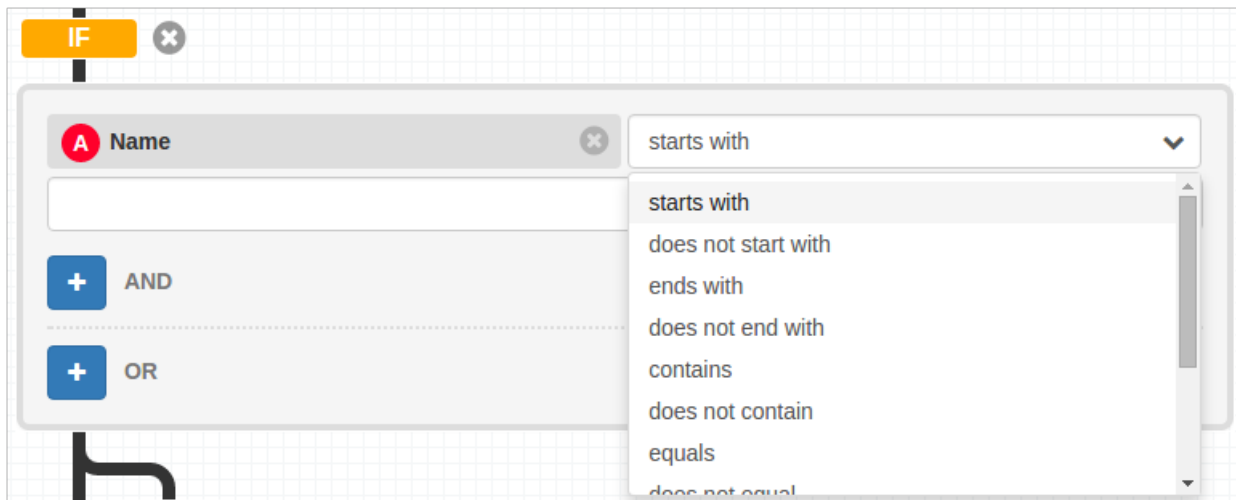


There are three key entities to pay attention to :

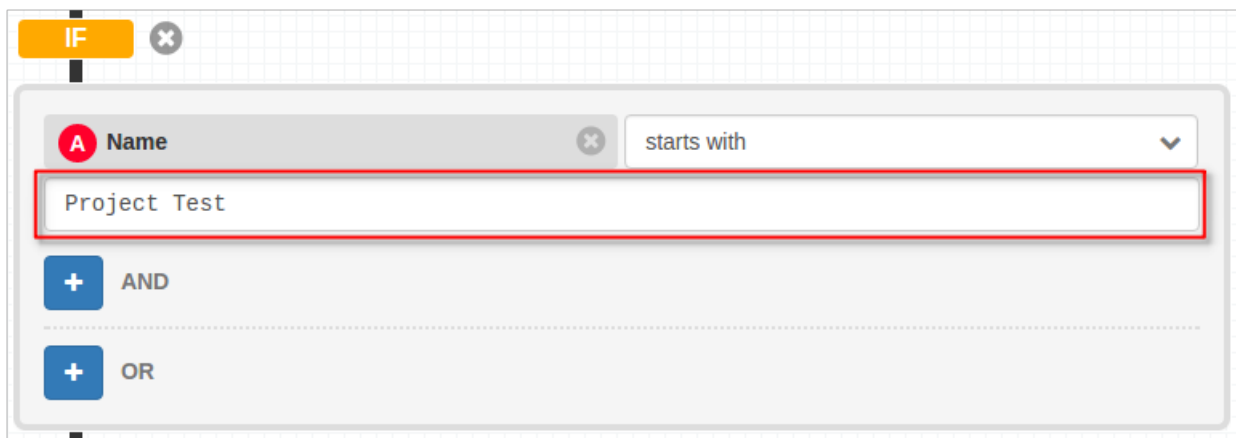
- (1) - Condition block. A **condition** is a statement that evaluates to either **true** or **false**.
- (2) - Pipe(s) that will execute when the condition is **true**.
- (3) - Pipe(s) that will execute when the condition is **false**.



Clicking on the blue “plus” button (**Add Filter**) allows you to select the fields you want from what pipes before the if-then-else block export.



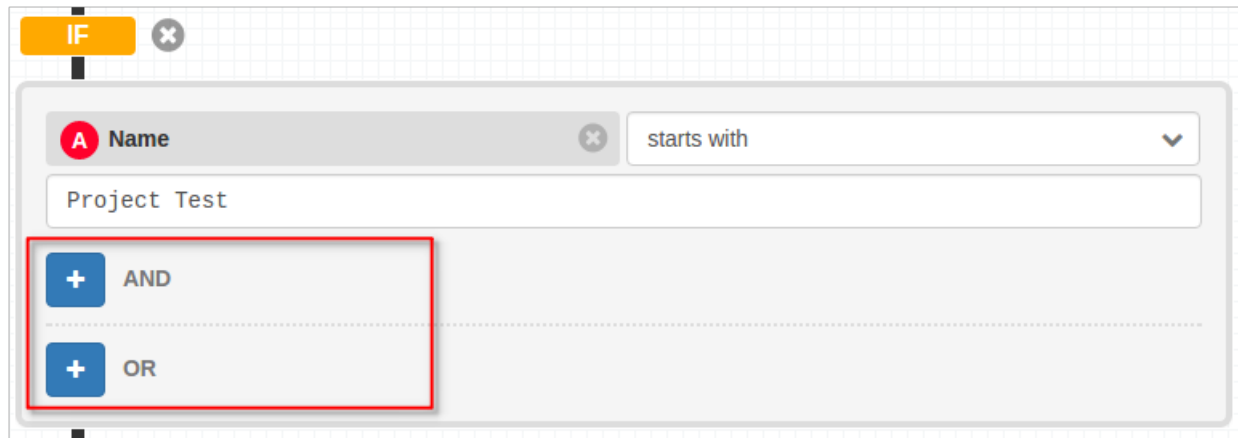
In the next drop-down you need to select the test you want to apply to the selected field. Tests available will vary based on the type of field you are testing.



Finally enter the value to test against. In the example here the block reads as:

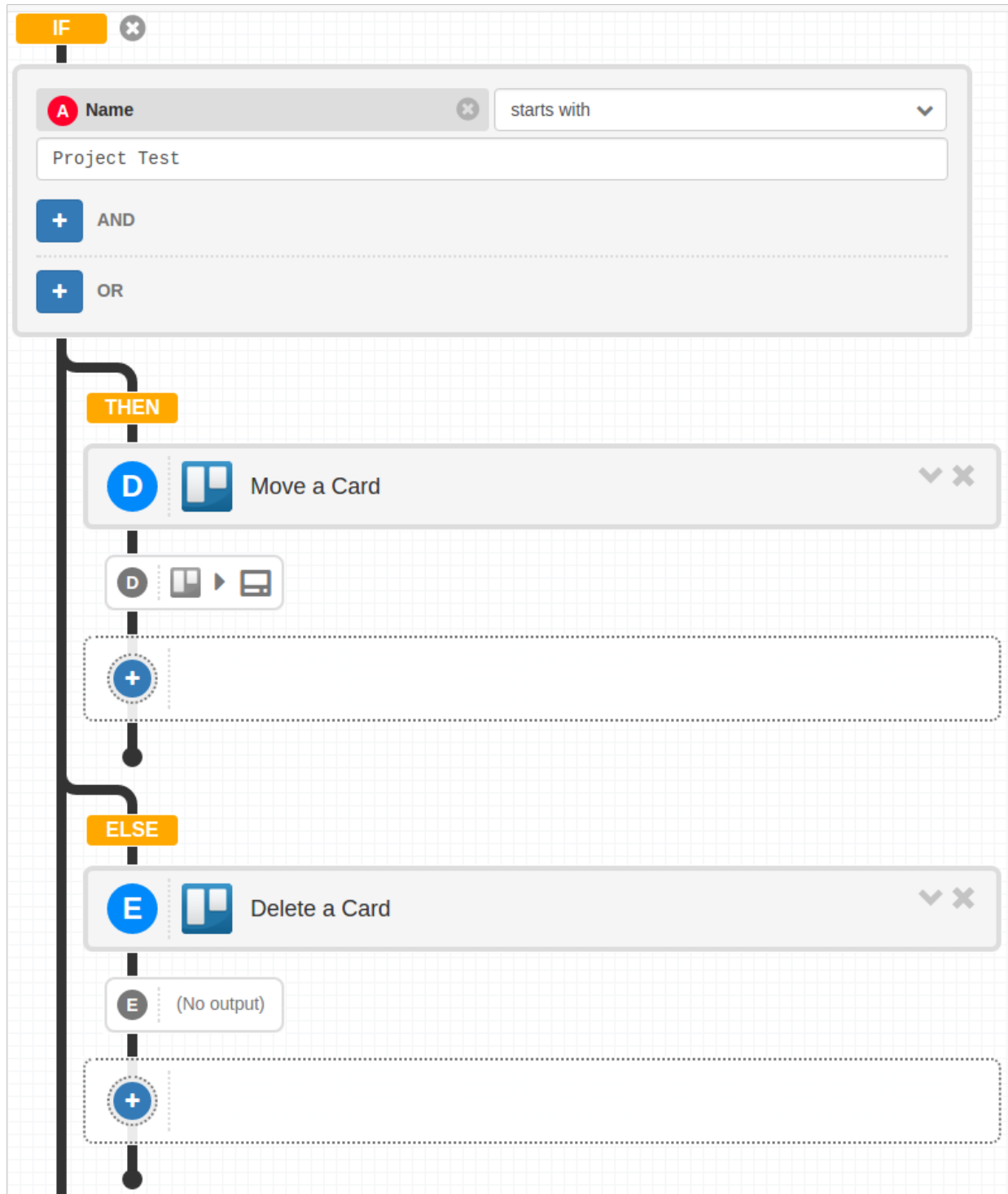
*if card's name starts with 'Project Test'*

In the cases where more complex logic is needed you can also use the provided **AND** and **OR** button to add filters to the **condition** of the block.



The screenshot shows the configuration for an 'IF' block in the Cloudpipes interface. The block is titled 'IF' in an orange tab. Below the title, there is a condition configuration area. It features a red circle with a white 'A' icon, followed by the text 'Name'. To the right of this is a dropdown menu currently showing 'starts with'. Below the dropdown is a text input field containing the text 'Project Test'. At the bottom of the configuration area, there are two buttons: a blue button with a white plus sign and the text 'AND', and a blue button with a white plus sign and the text 'OR'. These two buttons are enclosed in a red rectangular box.

After you have specified the logical condition it's time to specify what should happen if that condition is met – it evaluates to **true**. This means adding pipes to the **THEN** and **ELSE** branches. Pipes that you put in the **THEN** will execute if the logical condition is met (it evaluates to **true**). On the other hand, pipes that you add to the **ELSE** section will be executed only when the logical condition evaluates to **false** (is *not* true).



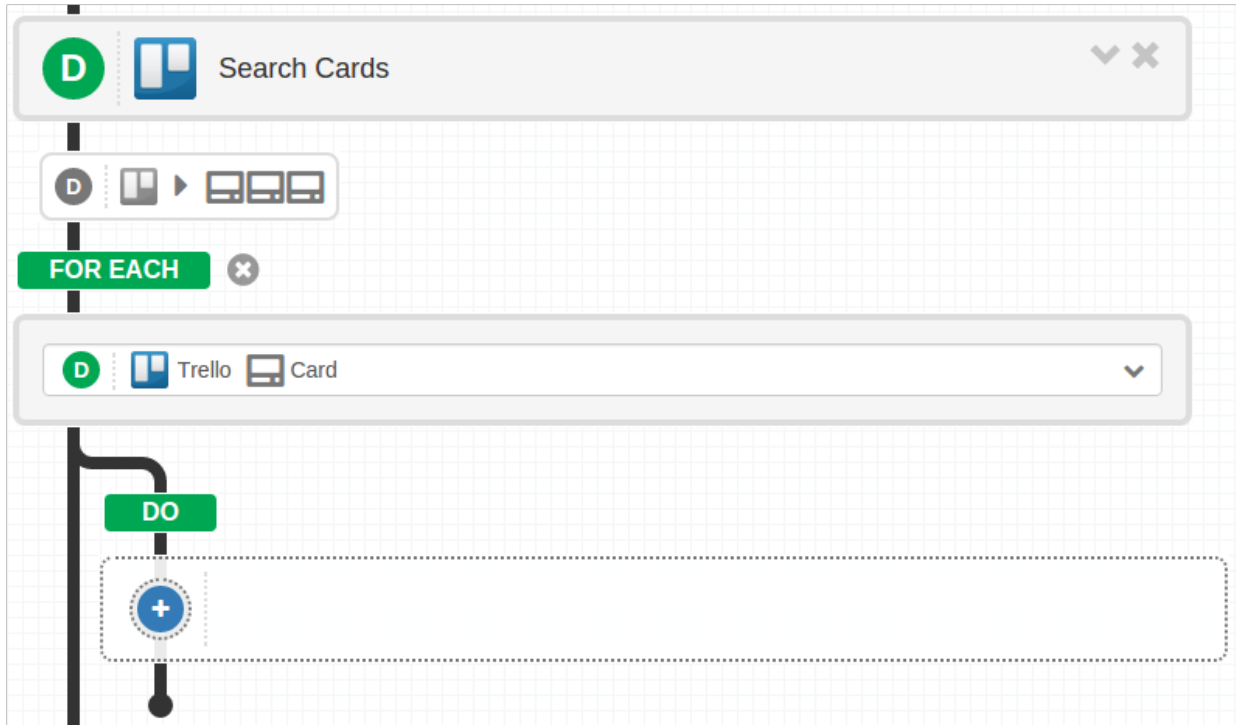
As an example - let's say that for the cases when the card's name does start with "Project Test" we want to move the cards to another list and for the cards with names that **do not** start with the given string - delete the cards.

There is no limitation on adding just one pipe in these blocks. You can add as many as needed and also you can have more conditional blocks, loops and stop blocks.

### 1.6.3 Iterating over lists

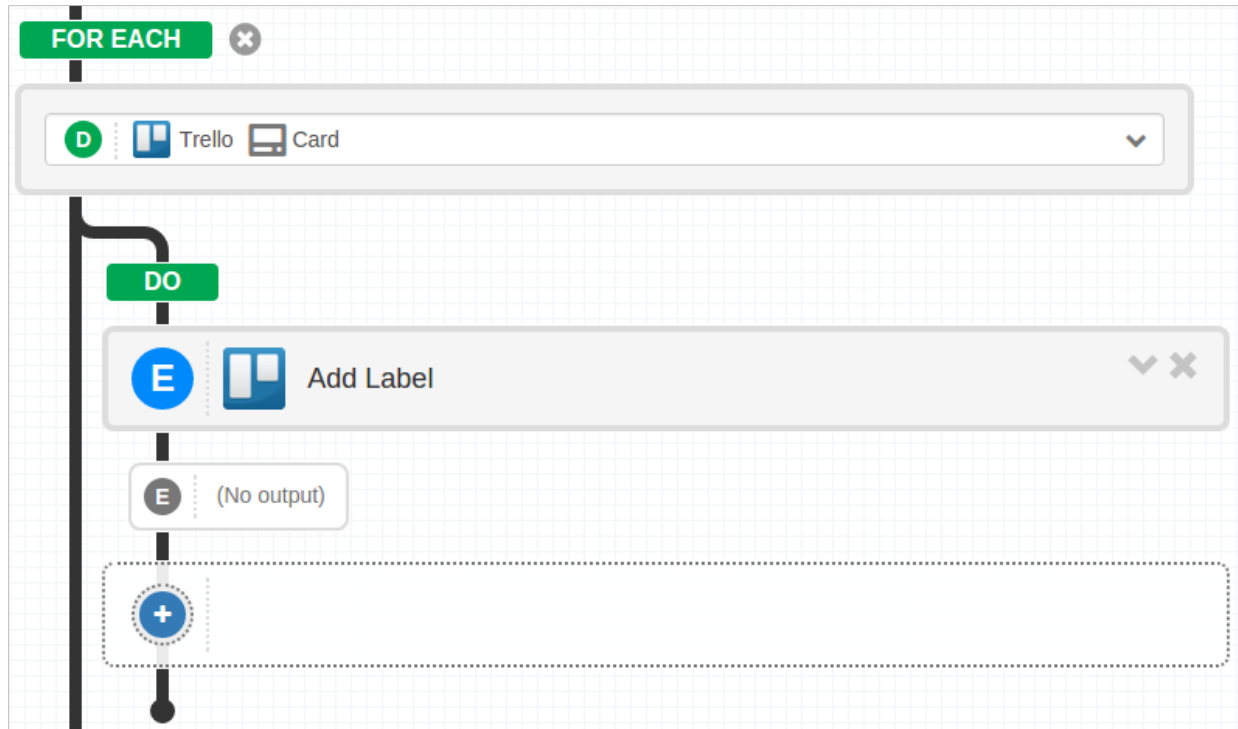


Clicking on this icon will insert a **for-each** block in your *pipeline*. Usually you'll want to do this after *pipes* that export a list of items in their outputs. Many of the “**Search ...**” pipes are examples of such.



Using the for **for-each** construct you will be able to execute pipes for each item in the list that is being iterated over. In this example we are Searching (listing) Trello Cards.

In the **DO** section you add the pipes that you want to have executed for each item. Like :



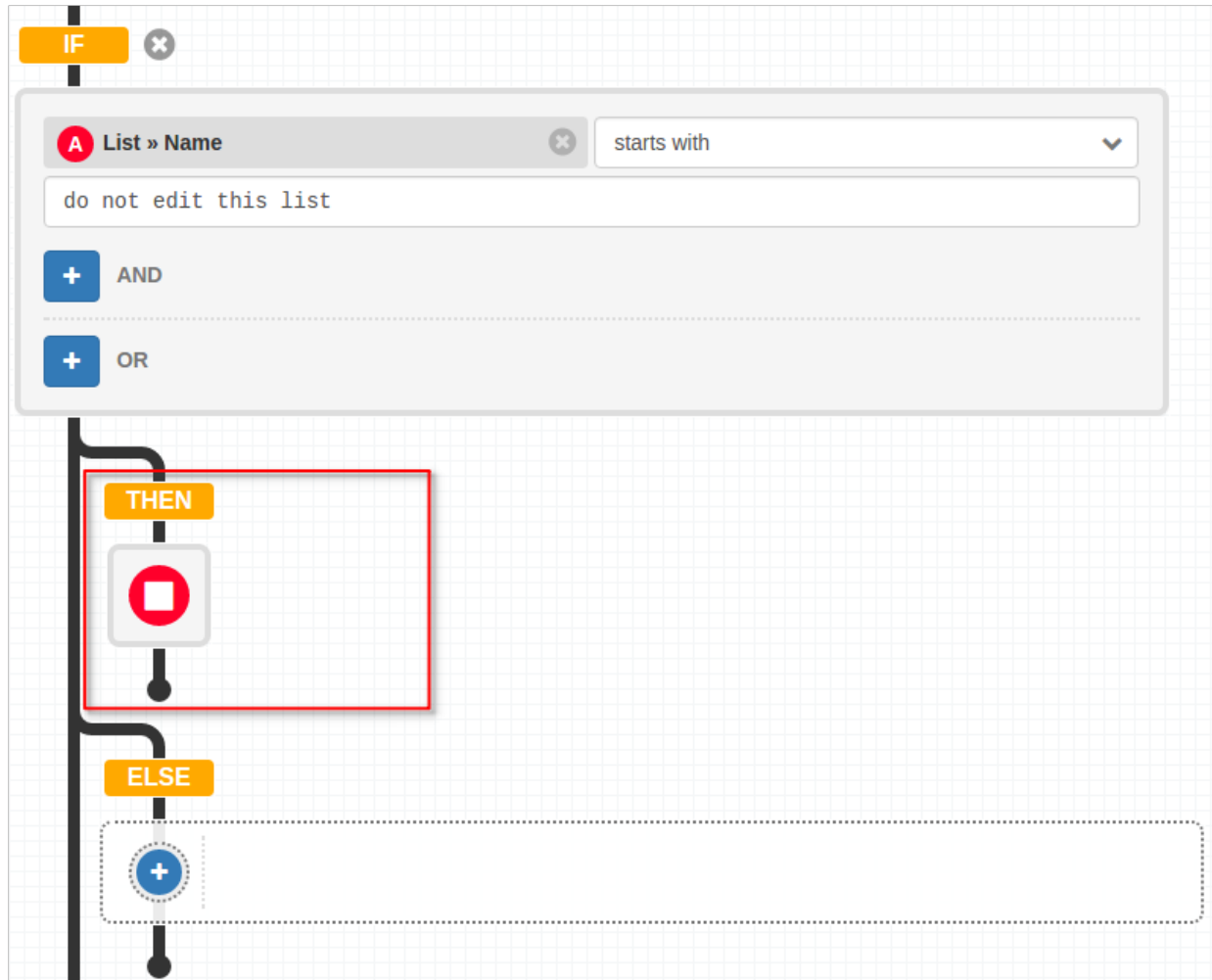
In our example we want to add a label to all the cards in the list, so we simply add an “Add Label” pipe, and select the label we want added in the pipe configuration.

Just like with the if-then-else blocks, there is no limitation on adding just one pipe in the **DO** section. You can add as many as needed and also you can have more conditional blocks, loops and stop blocks.

#### 1.6.4 Stop Pipeline Execution



The **Stop** block terminates *pipeline* execution once that block is reached.



For example if we have a Trello Card that belongs to a list that is not to be modified and given after the if-then-else block there are pipes that would modify the card, we can implement a pipeline like the above. The **stop** block will terminate pipeline execution as soon as it's reached and no further pipes will be run.

### 1.6.5 Conclusion

Flow control blocks are very powerful and flexible features of Cloudpipes. They allow you to implement logical conditions (if-then-else)s, iterate over lists of items (for-each) and also terminate the execution of a pipeline. Using these constructs you have at your disposal a set of tools that empower you to implement pipelines of arbitrary complexity.

Try them out and let us know what you think. Should you need any assistance, as usual just drop us a line at [support@cloudpipes.com](mailto:support@cloudpipes.com) or reach us in the in-app chat.

Happy integrating with **Cloudpipes**!

## 1.7 Linking Items

Linking is a feature that allows you to set a *link* between objects from two separate systems so that later on, on as needed basis, be able to *fetch* one of the linked objects knowing the other. For example you may want to create new Asana Tasks whenever a new Trello Item is created and later, when you update the Trello Item easily know which the



related Asana Task is. Another example maybe linking Salesforce Opportunities to Rows in a Google Sheet so when you update the Google Sheet Row be able to quickly fetch the *linked* Salesforce Opportunity and modify it as needed.

We try to make as many resources linkable as possible and most of the channels you see in the editor do support linking items from them. Should you need linking on a resource that you do not see in the editor, drop us a line at [support@cloudpipes.com](mailto:support@cloudpipes.com) or reach us in the in-app chat and we'll be happy to add it.

There are two essential actions that are relevant to *linking* - **creating the link** between the objects you want to link and **fetching** an already existing link.

### 1.7.1 How to do it

First of all in order to be able to use linking you need to have both object in the current pipeline. In this example we create a new Trello Card whenever a new Asana Task was created.

A

Task Created

A

▶

✓

B

Create a Card

Creates a new card in the specified board and list.

Account

Teodor Yantcheff

The Trello account to use for this action.

Board

test

List

list

Name

{{a:name}}

The name of the card.

Description

{{a:notes}}

M4

The description of the card.

More

Link (advanced)

A

Asana

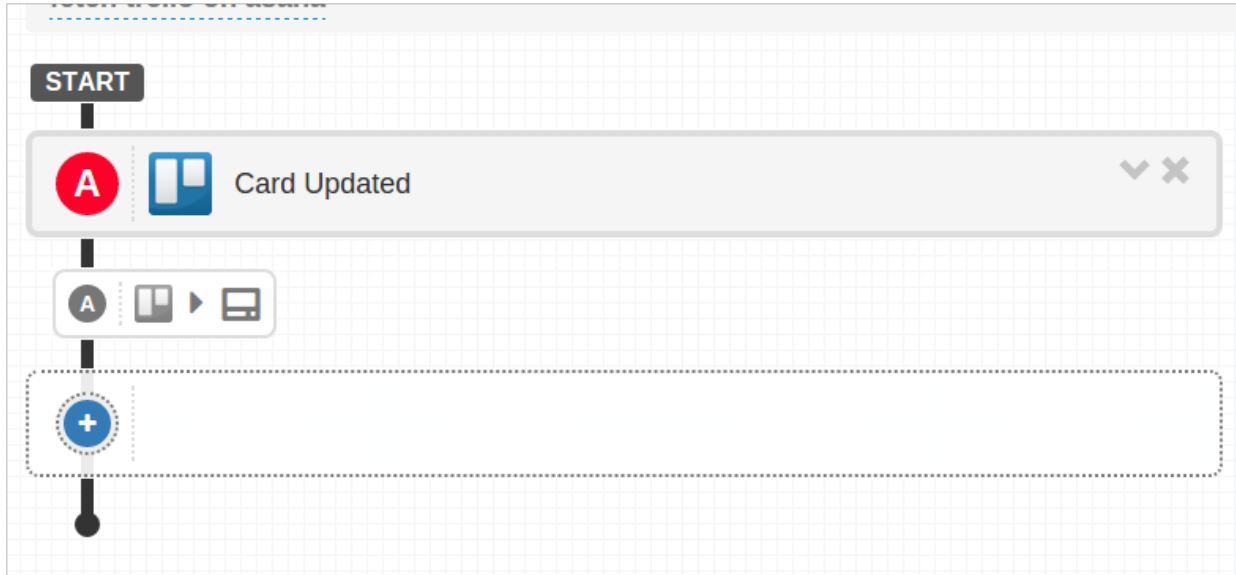
✓ Task

Link this card to another resource instance, so that you can later fetch one from the other and vice-versa.

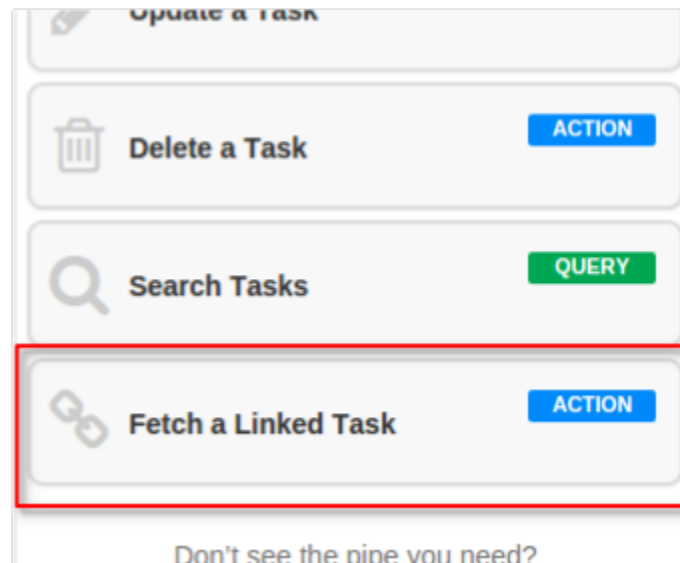
In the “Create A Card” pipe for Trello you can see the **Link** option, near the end of the pipe box, populated with the Asana Task we want to link to. In that dropdown you will be able to select which exact object from the current pipeline you want to link to in the case there is more than one linkable entity in the pipeline.

In this case the only items in the pipeline that allow linking are the Asana Task from the first pipe (A) and the Trello Card from the second (B).

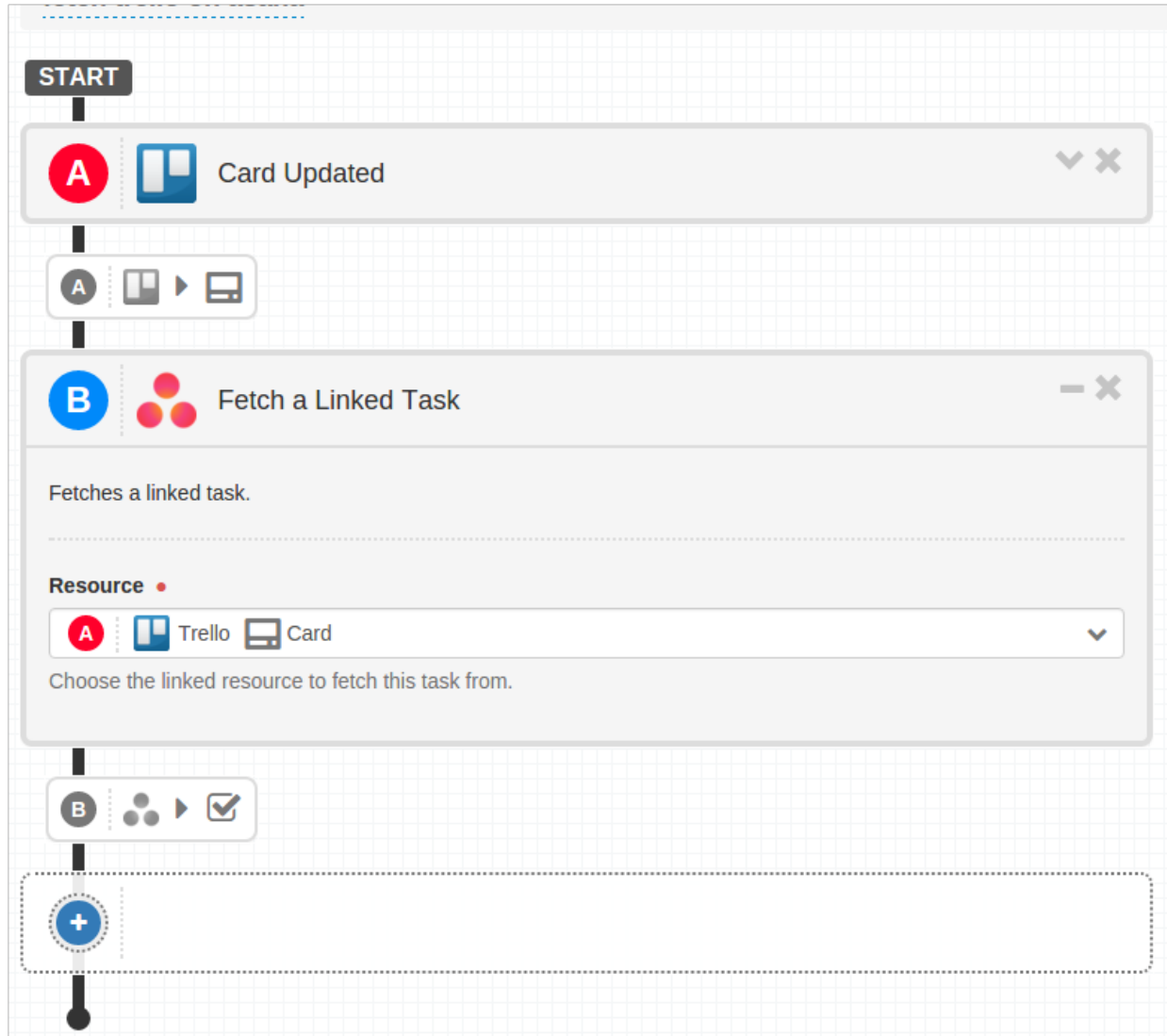
So now, after we have created the link we can use that link in another pipeline. Let's assume we want to update the linked Asana Task whenever the linked Trello Task gets updated. We start by creating a new pipeline and adding the "Card Updated" trigger from Trello:



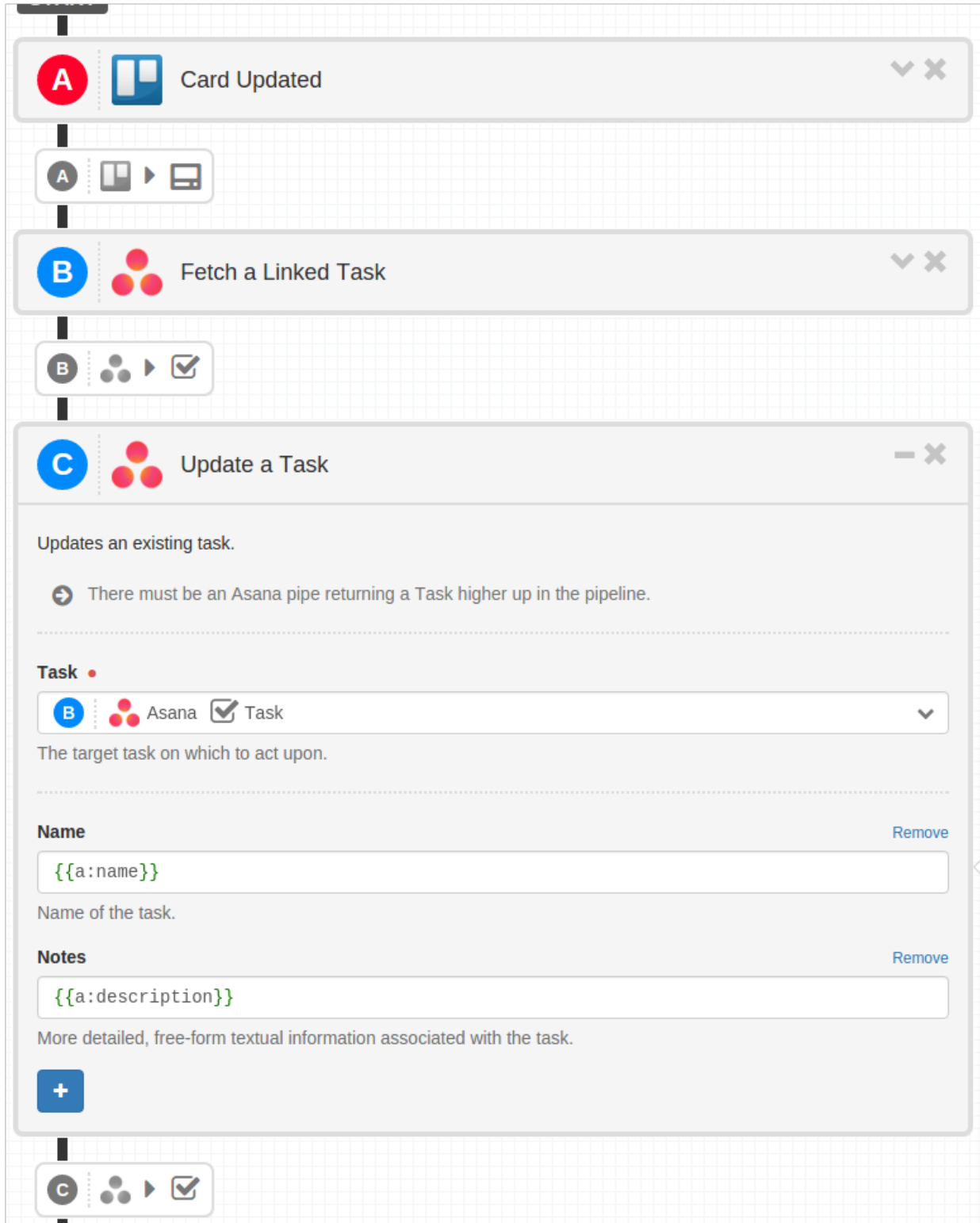
Next we need to find the Fetch Link pipe under the Asana drawer and drag and drop it as the next step in the pipeline.



This pipe will output the linked Asana Task and will make it accessible further down in the current pipeline.



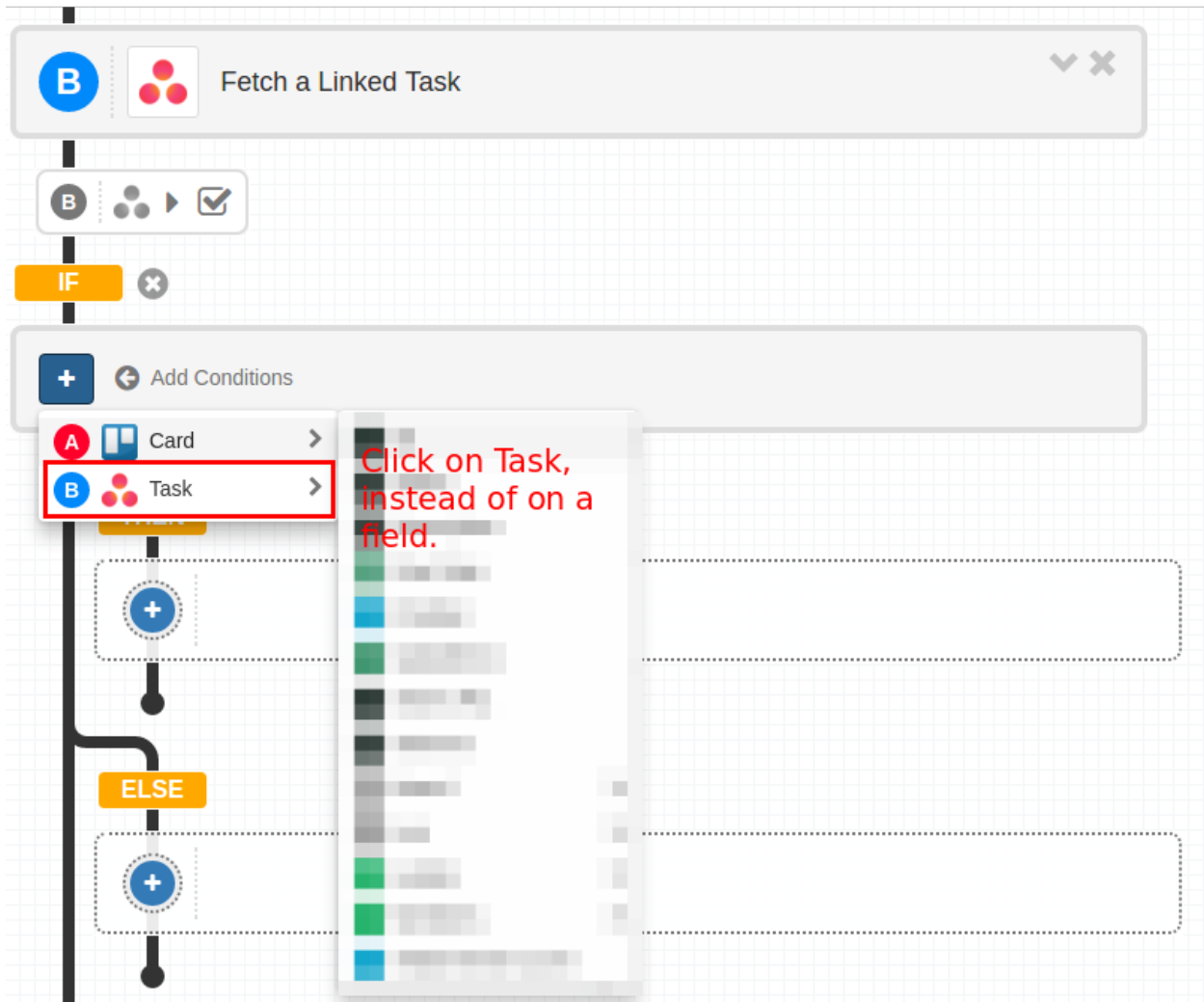
We can now add an “Update Task” pipe from the Asana collection and actually update it. In this example we update the Task’s *name* and *notes* to be the same as the linked Trello Card has.



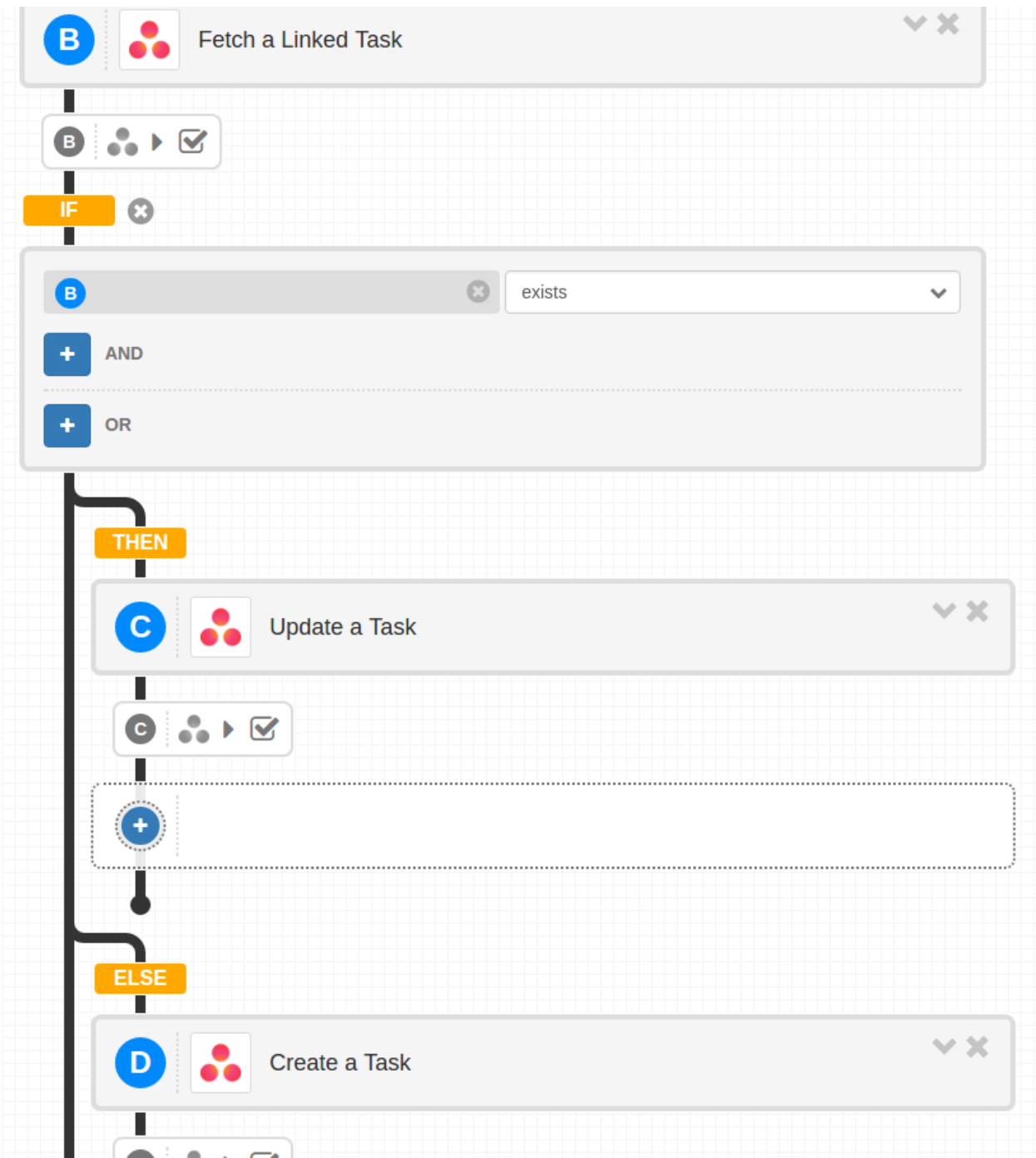
And this is really all there is to it. From that point on, using both pipelines we've created in this tutorial you have set up auto creation of new Trello cards and more importantly have **linked** those Cards to the corresponding Asana Tasks.

## 1.7.2 Checking if linked object exists

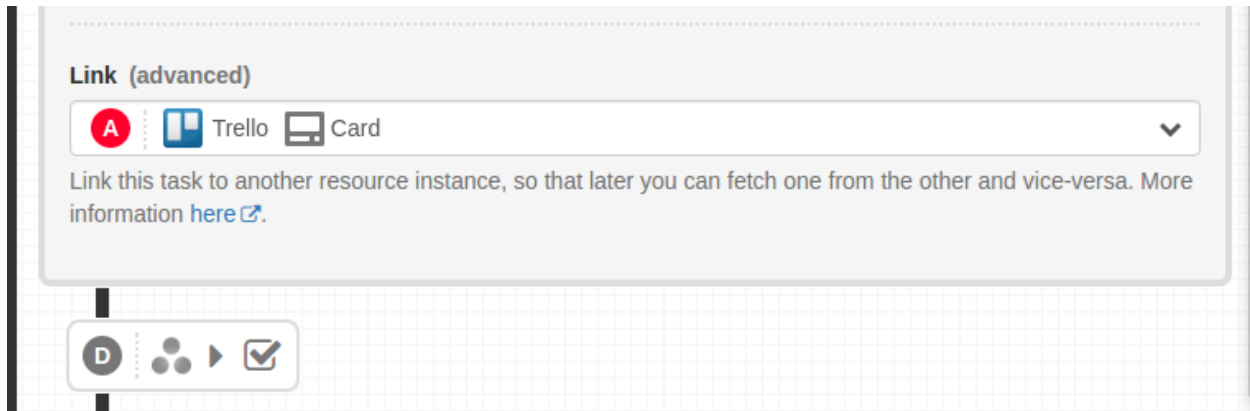
Sometimes your pipelines may trigger on objects which have not been linked yet. In this case you need to check whether the linked object already exists and update it, or if it does not exist to create it. To do that you need to put an IF condition like this (see also *Logical Conditions*):



The in the THEN branch add an *Update a Task* pipe, and in the ELSE branch add a *Create a Task* pipe.



**Don't forget** to add linking in the *Create a Task* pipe.



### 1.7.3 Troubleshooting

Make sure the options of the pipe from which an object is linked are the same as the ones from which it is retrieved. For example, if you have set *issue\_type* when creating a jira issue, you need to set it on the trigger which produces one side of the link, so that options match. For example, lets assume you have a Jira *Create an Issue* pipe, with the following options, which links to a trello card:

The following pipeline won't be able to fetch the linked card because the options differ (*Project* and *Issue Type* are not set):



**Issue Created**

Triggers when a new issue is created in the selected project.

[Refresh schema](#)

**Account** •

Kaloyan Kanev

The JIRA account to use for this trigger.

**Project**

**Issue Type**

Select the Issue Type if you need to access dynamic custom fields

**Fetch a Linked Card**

You will need to set both *Project* and *Issue Type* to the values from the *Create an Issue* pipe in order for the fetch link to work.

### 1.7.4 Conclusion

Linking is a very powerful concept that can be applied in many cases and for various reasons. Try it out and tell us what you think. Should you need any assistance, as usual just drop us a line at [support@cloudpipes.com](mailto:support@cloudpipes.com) or reach us in the in-app chat.

Happy integrating with **Cloudpipes**!

## 1.8 Current Time and Date/Time Arithmetic

### 1.8.1 Current Time

To put current date-time in a field you need the following template:

```
{{time.now}}
```

You can also get the time at the beginning of the date Today (00:00:00) with: `time today`

```
{{time.today}}
```

### 1.8.2 Time Arithmetic

Sometimes you need to get a date in the future or in the past. Usually this will be a due date or similar. To do that you can use the `time.now` and `time.today` together with `time.delta`. `time.delta` is a time offset object which you can add or subtract from a datetime to get a datetime in the future or the past.

#### Relative Time Delta

`time.delta` can be used with relative arguments to get relative time offset. The available arguments are *years*, *months*, *weeks*, *days*, *hours*, *minutes*, *seconds*. Note that all of them a plural, as opposed to singular *year*, *month*, etc., which are used for absolute time delta (replacement of current value).

For example to get the same time one hour from now you can use:

```
{{time.now + time.delta(hours=1)}}
```

Similarly to get a time at the beginning of day one week from now you can do:

```
{{time.today + time.delta(days=7)}}
```

Note that months and years takes into account the current date, so `time.now + time.delta(months=1)` actually results in the same date one month from now, unless next month has less days than the current day of the month in which case it results in last day of next month. For example if today is 2016-01-27 and we add `time.delta(months=1)` the end result is 2016-02-27, but if today is 2016-01-31, since February 2016 has 29 days the result will be 2016-02-29.

#### Absolute Time Delta

Similar to relative delta above, you can give the following arguments to `time.delta` to get an absolute offset, that is replace the current value with the one from the delta: *year*, *month*, *day*, *hour*, *minute*, and *second*.

For example to get same datetime as now in 2012 you can do:

```
{{time.now + time.delta(year=2012)}}
```

### 1.8.3 Timezone Filter

All datetimes which flow through Cloudpipes are in UTC. In general you don't need to do any timezone conversion yourself if you map between datetime fields, even between different channels. Sometimes however you may want to print the time in a text field in a user-friendly timezone. You can do that with the `timezone` filter.

```
{{time.now|timezone}}
{{time.now|timezone('UTC')}}
{{time.now|timezone('EST')}}
{{time.now|timezone('Australia/Darwin')}}}
```

renders to:

```
2016-07-28 13:18:26.889542+00:00
2016-07-28 13:18:26.889571+00:00
2016-07-28 08:18:26.889584-05:00
2016-07-28 22:48:26.889603+09:30
```

The default timezone is UTC, so `{{time.now|timezone}}` is equivalent to `{{time.now|timezone('UTC')}}}`. You can find your timezone's name in the *TZ* column in this [Wikipedia Article](#).

## 1.8.4 Date Format Filters

**Warning: do not use this filter to transfer dates between date-typed fields. We handle the remote system date format. This filter is intended for formatting dates into string-typed (human-readable) fields.**

Date are usually formatted as timestamp for inside Cloudpipes. Sometimes you might need to format the date in another format or display just the date options. You can use these filters to do that: `date_ymd`, `date_dmy`, and `date_mdy`.

```
{{time.now|date_ymd}}
{{time.now|date_mdy}}
{{time.now|date_dmy}}
```

renders to:

```
2016-09-20
09-20-2016
20-09-2016
```

Note that there is an optional separator argument to the filter. The default separator is `-`.

```
{{time.now|date_ymd('')}
{{time.now|date_mdy('/')}}
{{time.now|date_dmy('.')}}
```

renders to:

```
20160920
09/20/2016
20.09.2016
```

Note that when separator is not given `-` is used and that you can specify empty string with two single quotes `''`.

## 1.9 Integration Examples

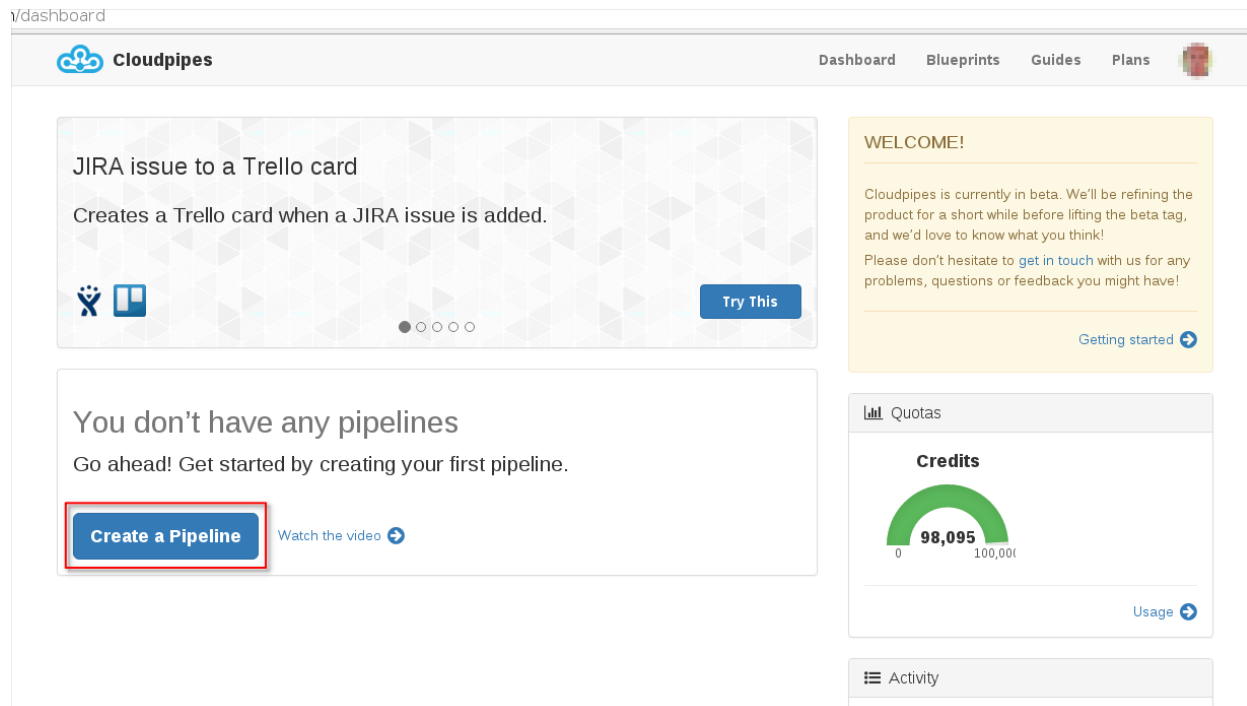
Here you will find examples and guides on how to implement useful integrations between Apps commonly used by our user

## 1.9.1 Using Slack Command to Lookup Opportunities in Salesforce

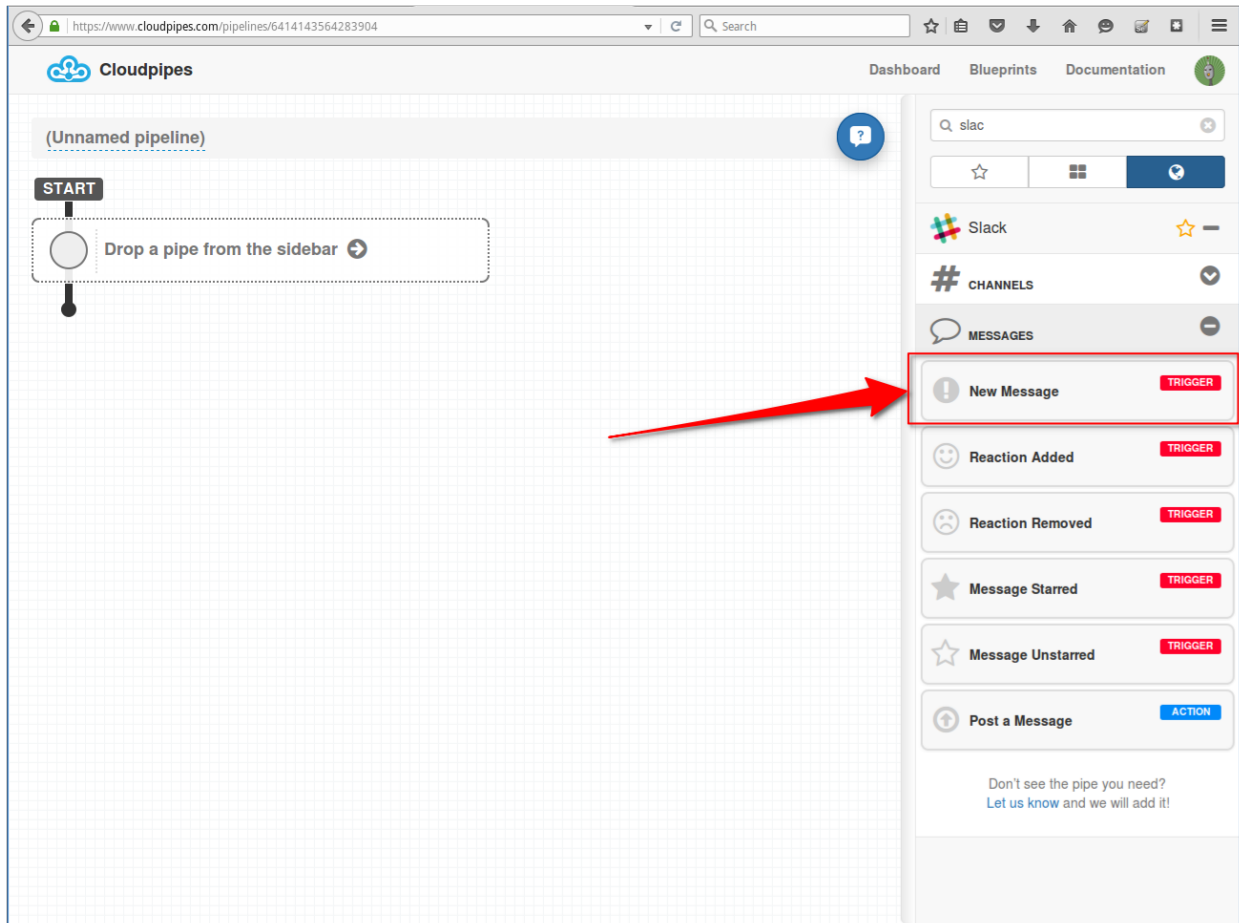
In this tutorial we are going to build a pipeline that listens on new messages in Slack that start with “search:”. It then searches Salesforce Opportunities that have matching names and posts selected information back to the Slack channel.

### How to do it

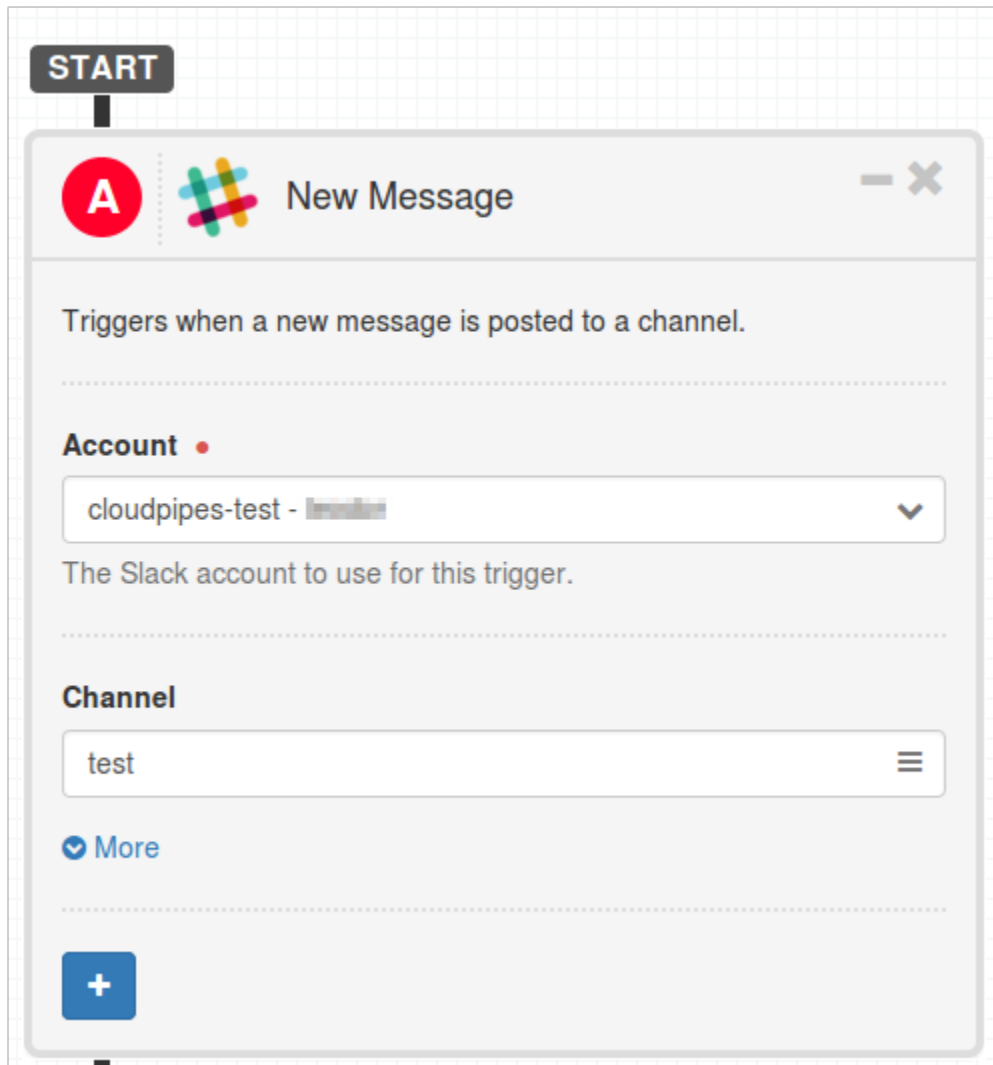
Let’s start by creating a new, blank *pipeline* by clicking the **Create a Pipeline** button on the dashboard.



The first pipe that needs to be added to the pipeline is a *trigger*, that will fire whenever a new message is posted to Slack. We need to locate Slack in the channels list on the right, then drag and drop the “New Message” trigger.



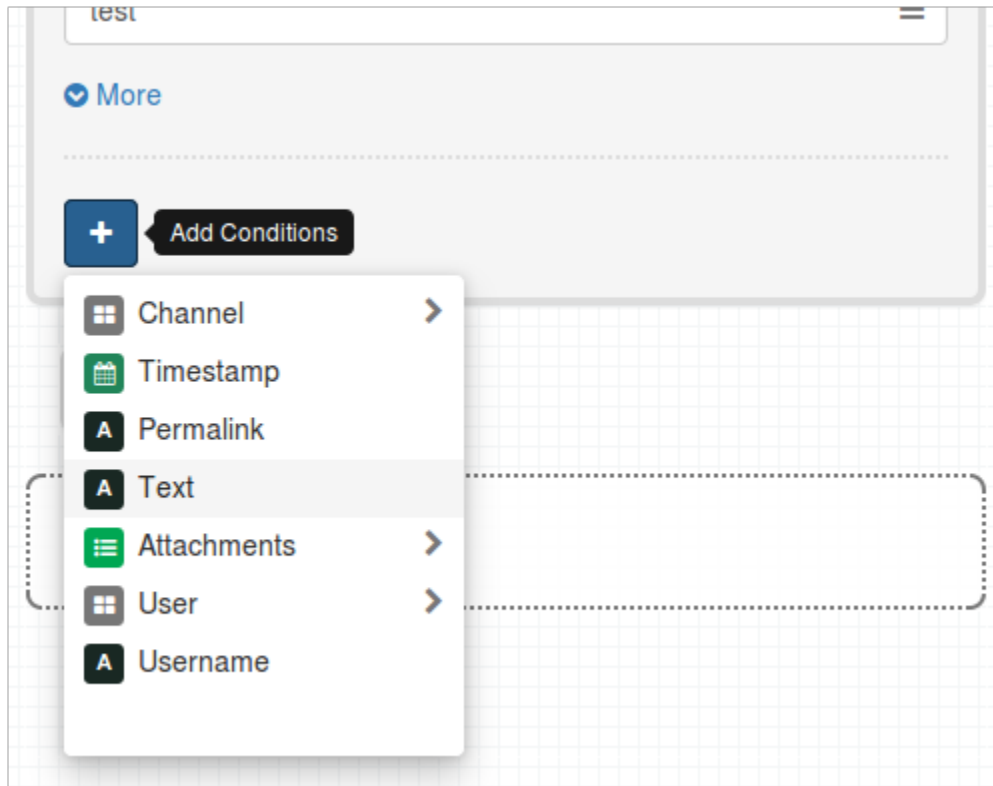
After dropping the *pipe* in the first slot of the pipeline, select your *account* (or connect a new one if needed), then select the Slack channel you are interested in monitoring.



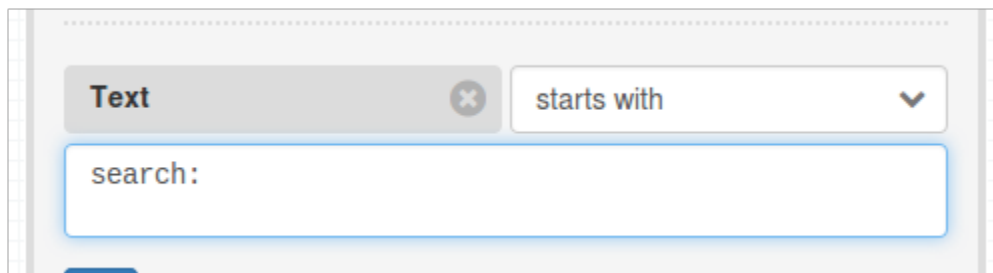
Here we want to act on Slack messages that start with “search:” so we need to add a **filter** to the pipe. **Filters** make sure that only events that match will fire the trigger and execution of the pipeline will continue. Filters are supported on pipes that have the **Add filter** button near the end (blue *plus* icon)

### Adding a Filter

Click the **Add filter** button (blue *plus* icon) and select *Text* as the field to filter on.



We are only interested in messages that start with “search:”, so in the filter condition drop-down select *starts with* and in the filter value enter “search:”. This will make sure that the trigger will fire only on messages that start with “search:”

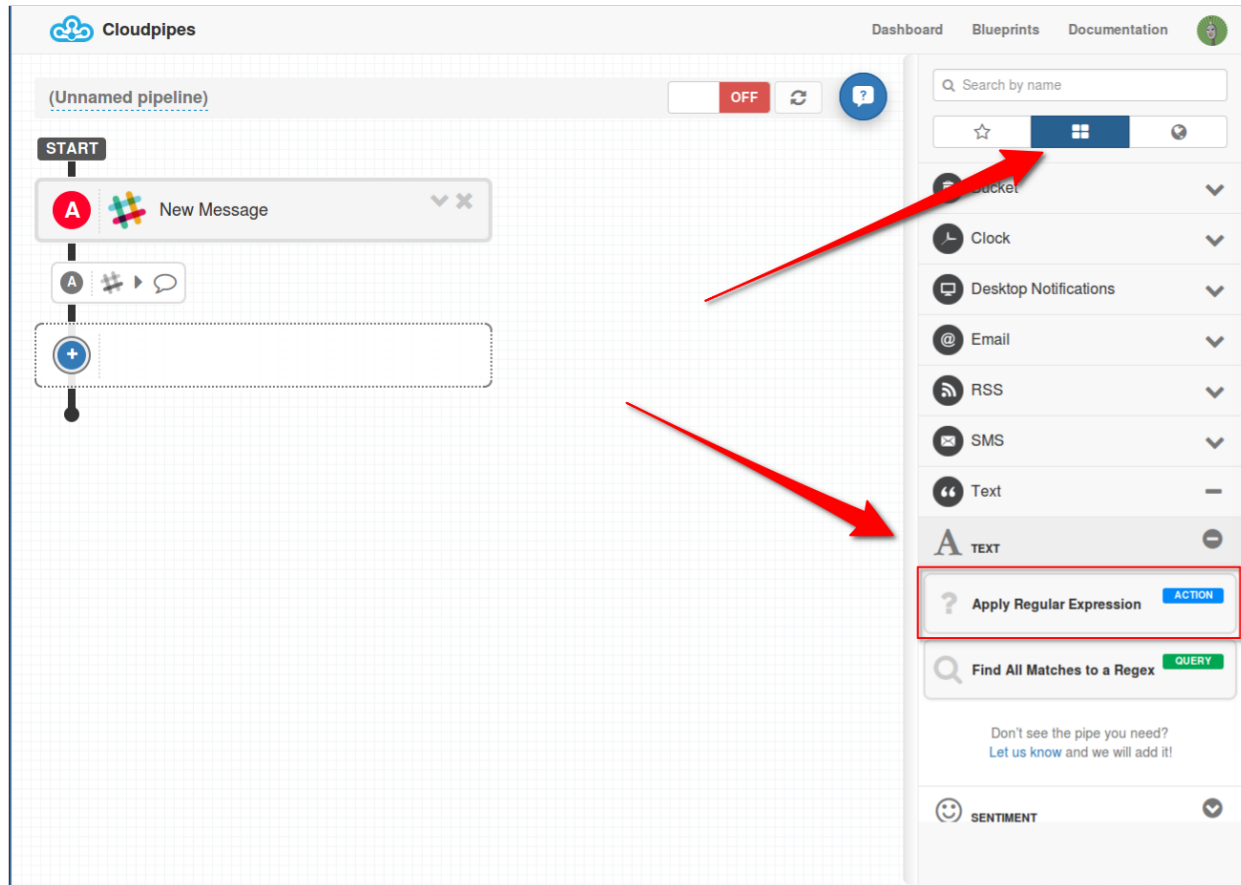


Now, since the *New Message* pipe outputs the entire message that was sent to Slack, we need to split it into two parts - “search:” which we’ll call “command” and the string that was searched for, which we call “term”.

### Splitting Text With a Regular Expression

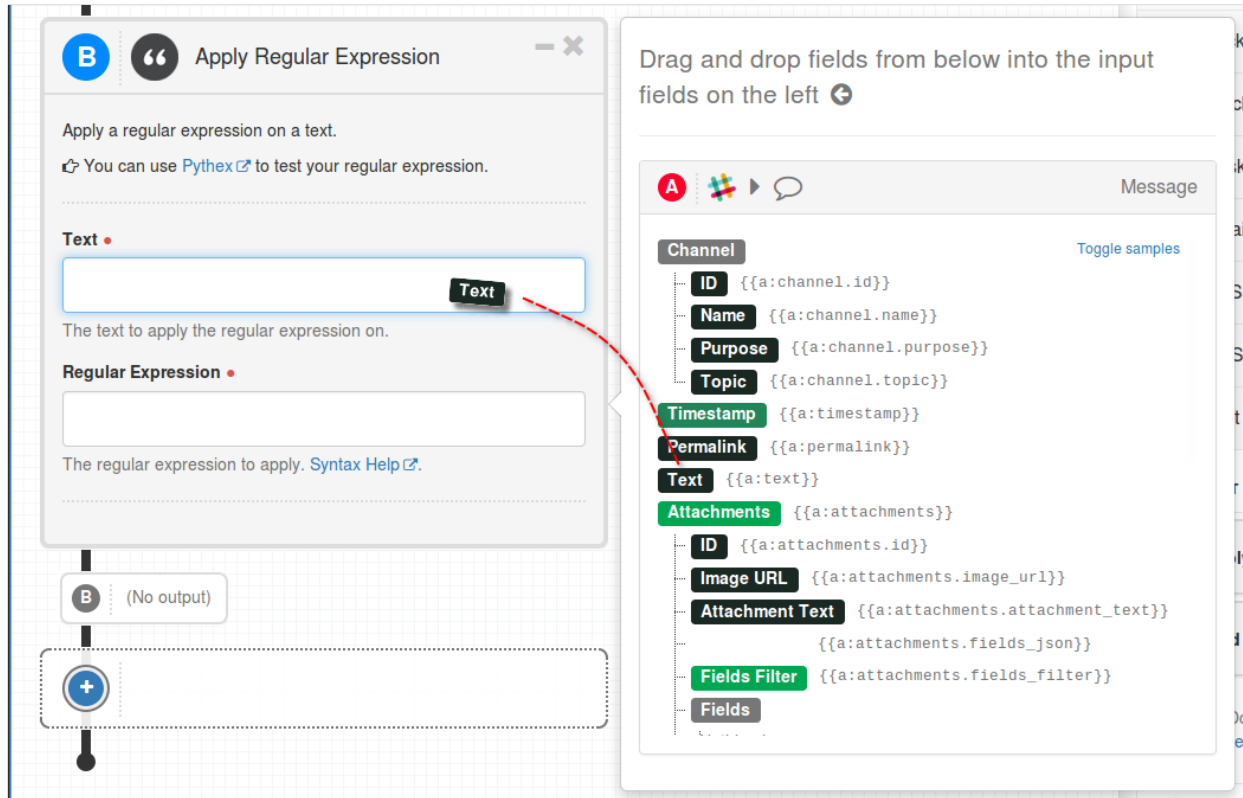
In order to do that we will use a *Regular Expression*. Regular Expressions (also called “regexes”) are extremely flexible and powerful and allow you to do all kinds of magic with text. In our example we are going to use one to split the message we have from Slack into two parts – the “command” used (“search” in this example) and the search term.

Locate the “*Apply Regular Expression*” pipe under *Built-in channels* -> *Text* and add it as the next step in the pipeline.



*Apply Regular Expression* has only two parameters – *Text*, the text to apply the regular expression on, and *Regular Expression*, the regex to apply. Since we are interested in splitting the Slack message from the fields list drag and drop *Text*.



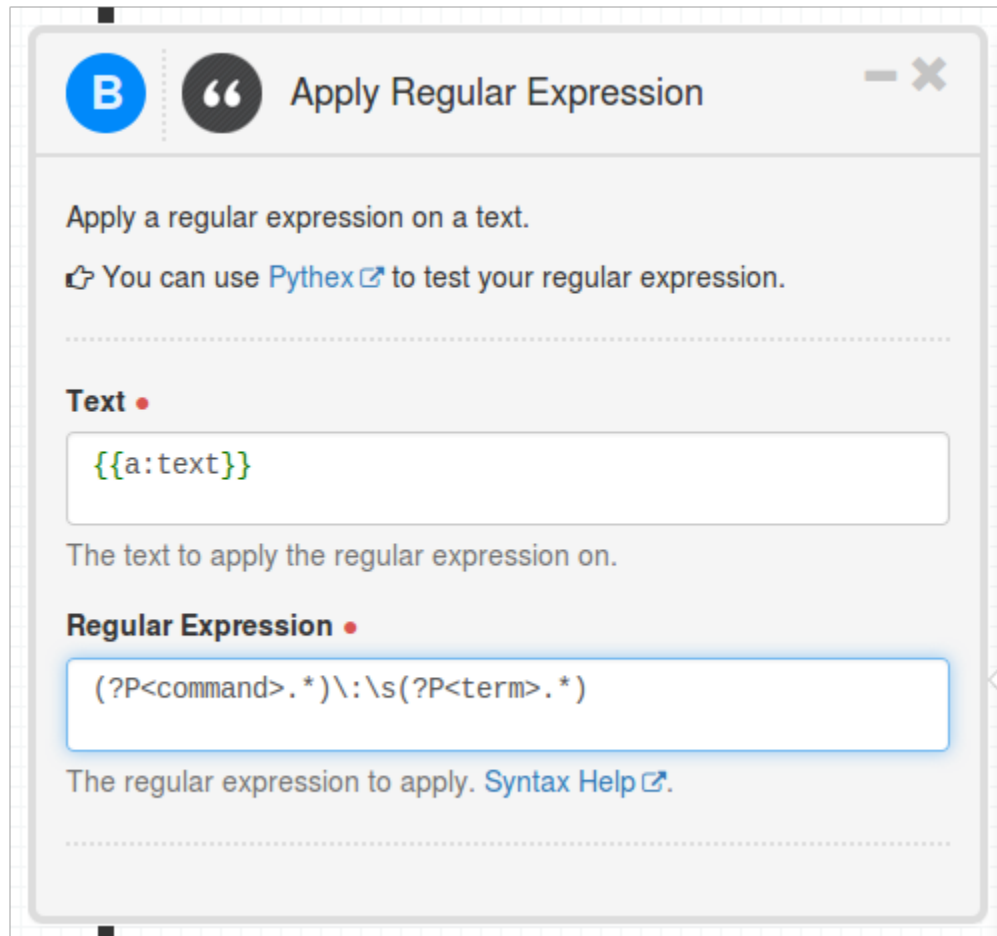


In the *Regular Expression* textbox enter: `(?P<command>.*)\:\s(?P<term>.*)`

This will make sure that in the pipe output we will have two named fields - *command* and *term*. We consider all that's before the colon in the Slack message to be "command" and any characters after to be "term".

**Note:** For a reference of the supported syntax please refer to [this document](#). You can also use [Pythex](#) to test your regular expression.

After you are done the "Apply Regular Expression" pipe should look like this :



The screenshot shows a dialog box titled "Apply Regular Expression" with a blue "B" icon and a quote icon. The dialog contains the following text and fields:

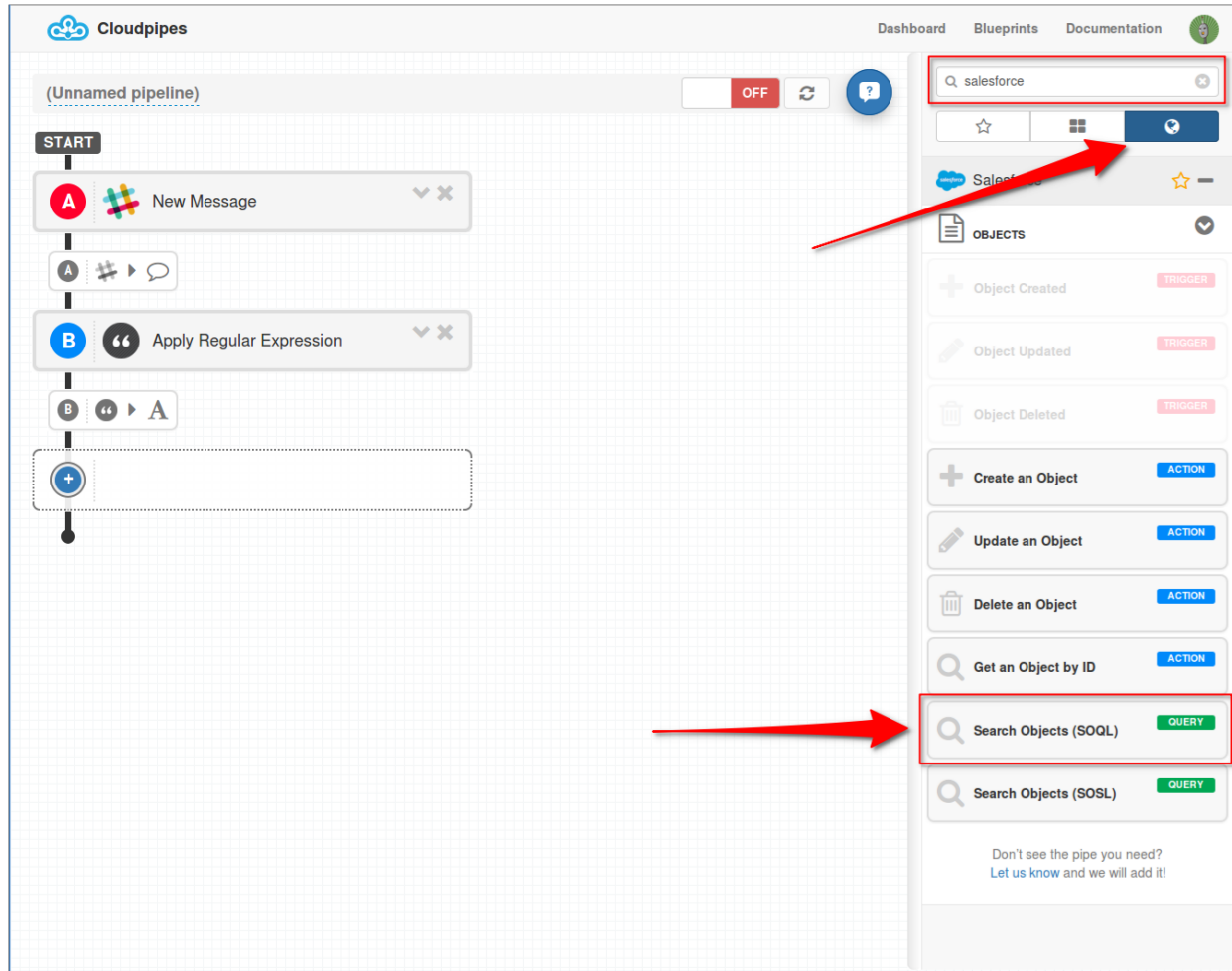
- Instruction: "Apply a regular expression on a text."
- Tip: "You can use [Pythex](#) to test your regular expression."
- Section: "Text" with a red dot icon.
- Text input field: Contains the placeholder text `{{a:text}}`.
- Label: "The text to apply the regular expression on."
- Section: "Regular Expression" with a red dot icon.
- Text input field: Contains the regular expression `(?P<command>.*)\:\s(?P<term>.*)`.
- Label: "The regular expression to apply. [Syntax Help](#)."

At this point we now have the message that was posted in Slack split in *command* and *term* which are available in the output of "Apply Regular Expression". Note that because of the filter that we added in the "New Message" trigger the *command* will always be "search".

So now all that's left is to search Trello and post back to Slack.

### Searching Opportunities in Salesforce

Select the *All Channels* tab on the channels list and locate *Salesforce*. Note that you can also use the text-box above to search by name for speedier access. Then add the *Search Objects (SOQL)* pipe as the next step in the pipeline :



In the *Search Objects (SOQL)* pipe select the Salesforce account that you want to use or if needed connect a new one. For *Object Type* select **Opportunity** from the drop-down and in the *Query* textbox enter: `WHERE Name LIKE '%{{b:term}}%'`

Search Objects (SOQL)

Search for objects in Salesforce using Salesforce Object Query Language (SOQL). Only object types queryable by the user are shown. The query sent to Salesforce is `SELECT <ALL FIELDS> FROM <Object Type> WHERE <Query> .`

Account

The Salesforce account to use for this query.

Object Type

Opportunity

The Salesforce object type (can be a custom object).

Query

WHERE Name LIKE '%{{b:term}}%'

An optional [SOQL](#) WHERE clause for filtering the object (ex. Name LIKE '%Co%' AND MailingCity = 'London'). If no WHERE clause is given, all objects are retrieved.

Order By

Select a field

(unordered)

Optional sort order.

Limit

(unlimited)

Optional limit of the number of objects returned.

Drag and drop fields from below into the input fields on the left

A

Message

B

A

Text

Full match

{{b:group\_0}}

Toggle samples

Group 2

{{b:group\_2}}

Group 1

{{b:group\_1}}

Named group 'term'

{{b:term}}

Named group 'command'

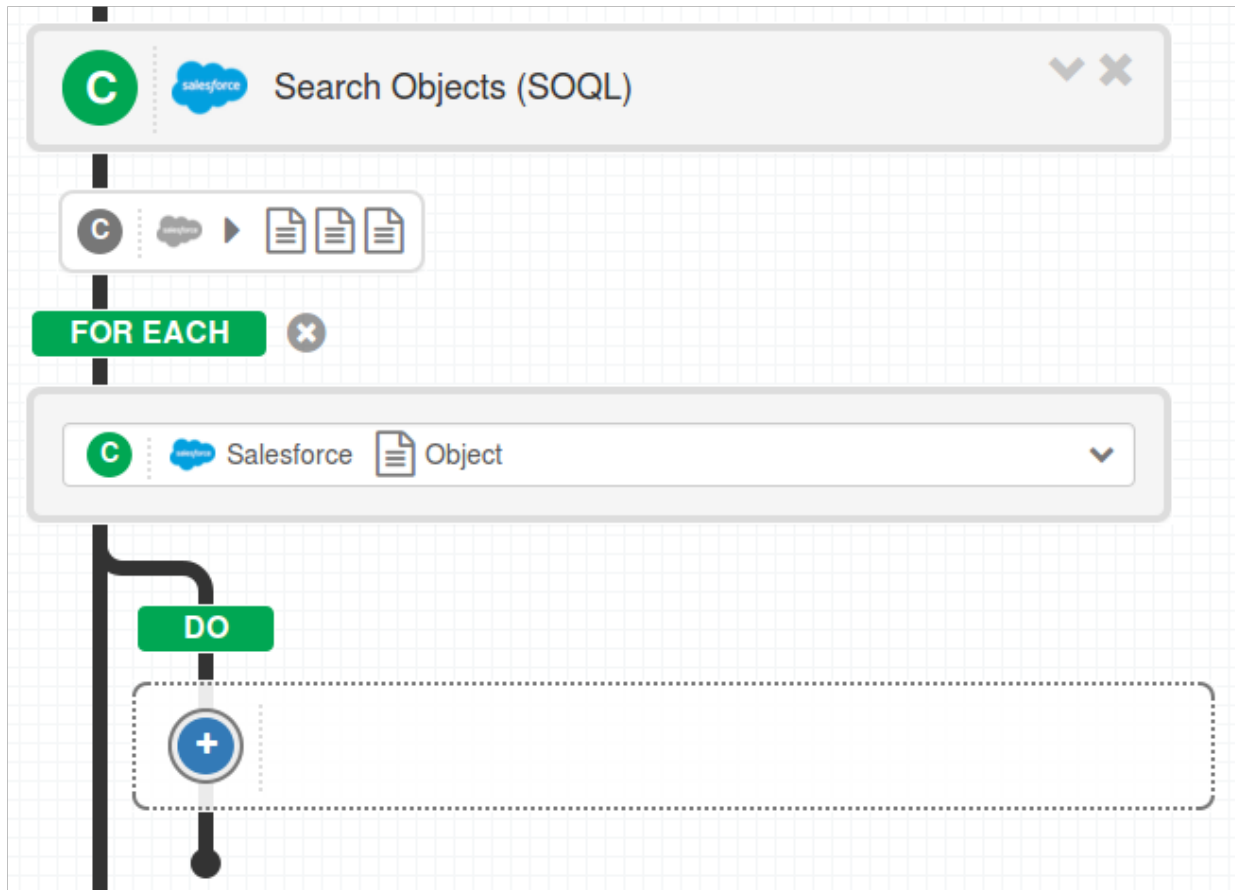
{{b:command}}

Having *Search Objects* configured like this will search for Opportunities that contain our *term* search field in their *Name* property.

The last thing we need to do is to post back to Slack the results we've found.

## Posting to Slack

Adding a *Search* pipe automatically adds a *for-each iteration* block after it. Using this we can act on each individual result found.



So now all we need to do is add a *Post Message* action pipe from Slack. In the channels list locate *Slack*. Drag the *Post Message* pipe and put it in the slot after *DO*.

The screenshot shows the Cloudpipes interface for configuring a 'Post a Message' action. The action is part of a 'DO' block within a 'FOR EACH' loop. The 'Post a Message' action has the following fields:

- Account:** A dropdown menu with 'cloudpipes-test' selected.
- Channel:** A dropdown menu with 'Select a Channel' selected.
- Text:** A text input field.
- Post as User:** A dropdown menu with 'No' selected.
- Username:** A text input field with 'Cloudpipes' entered.

On the right, a panel titled 'Drag and drop fields from below into the input fields on the left' shows a list of fields to be dragged into the 'Text' field:

- Account ID
- Amount
- Campaign ID
- Close Date
- Closed
- Created By ID
- Created Date
- Current Generator(s)
- Deleted
- Delivery/Installation Status
- Description
- Expected Amount
- Fiscal Period
- Fiscal Quarter
- Fiscal Year

You can now select a specific channel in your Slack that you want the results posted to. Alternatively drag and drop *ID* from the list of fields to the right after expanding the first list of fields (the one from the “A” pipe).

From the fields list drag and drop the fields you want posted to Slack in response to our query. Note that since we are posting to Slack we can make use of the [markdown-like formatting](#) Slack supports.

For our example in the *Text* textbox we are going to enter this:

```
*name: {{c:name}} ({{c:probability}})*
*amount:* {{c:amount}}
*stage:* {{c:stage_name}}
*last activity:* _{{c:last_activity_date}}_
*descr:* "{{c:description}}"
*-----*
```

As you can see we've added several fields from Salesforce's Opportunity result and used Slack formatting.

One optional step we can take is to add one more *Post Message* pipe to the very end of the pipeline that simply says “— **End List** —”, so even for the cases when the search returns no results you will get a note in Slack and know that it was an empty list.

## Conclusion

This is all there is to it really. In this brief tutorial we've covered all that's needed to build quite a sophisticated pipeline that automates your workflow for Salesforce by using Slack to search for Opportunities and query data. We've used a regular expression to correctly split “commands” in Slack and also *filters* to only act on commands we are interested in.

Should you need any assistance, as usual just drop us a line at [support@cloudpipes.com](mailto:support@cloudpipes.com) or reach us in the in-app chat.

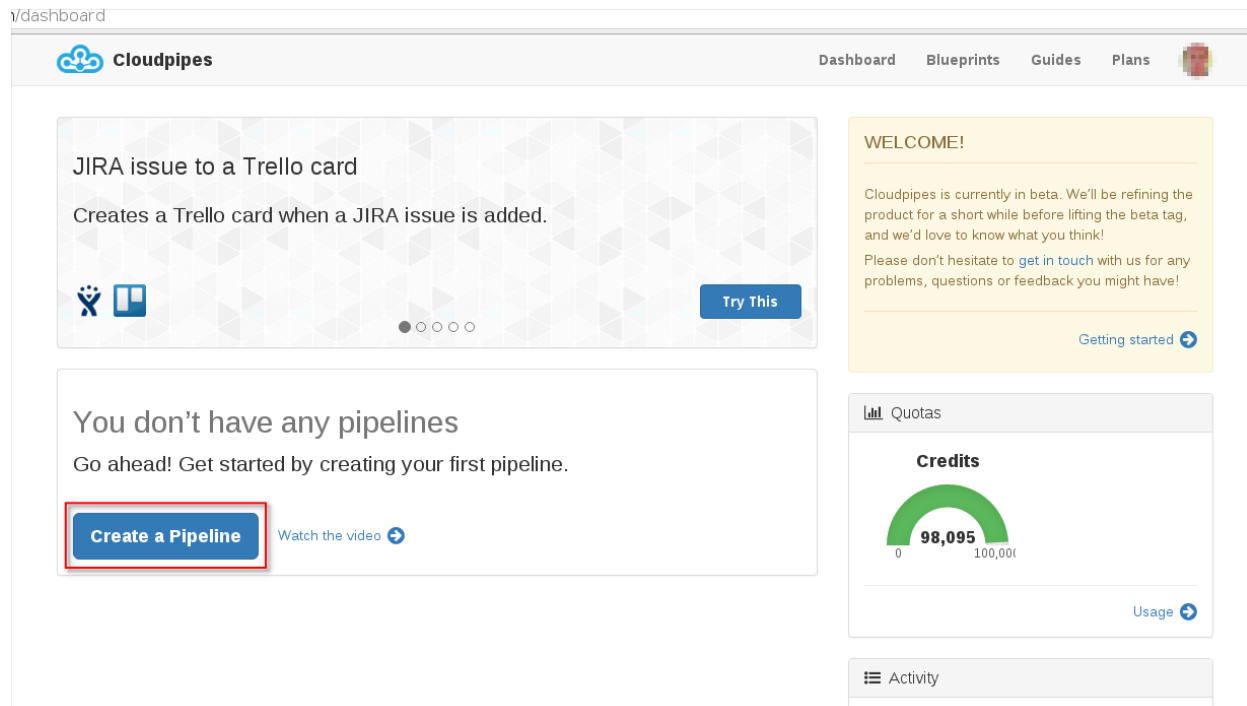
Happy integrating with **Cloudpipes**!

## 1.9.2 Search Intercom Users Using a Command in Slack

In this tutorial we are going to build a pipeline that listens on new messages in Slack that start with “search:”. It then searches Intercom users that have matching names and posts user information back to the Slack channel.

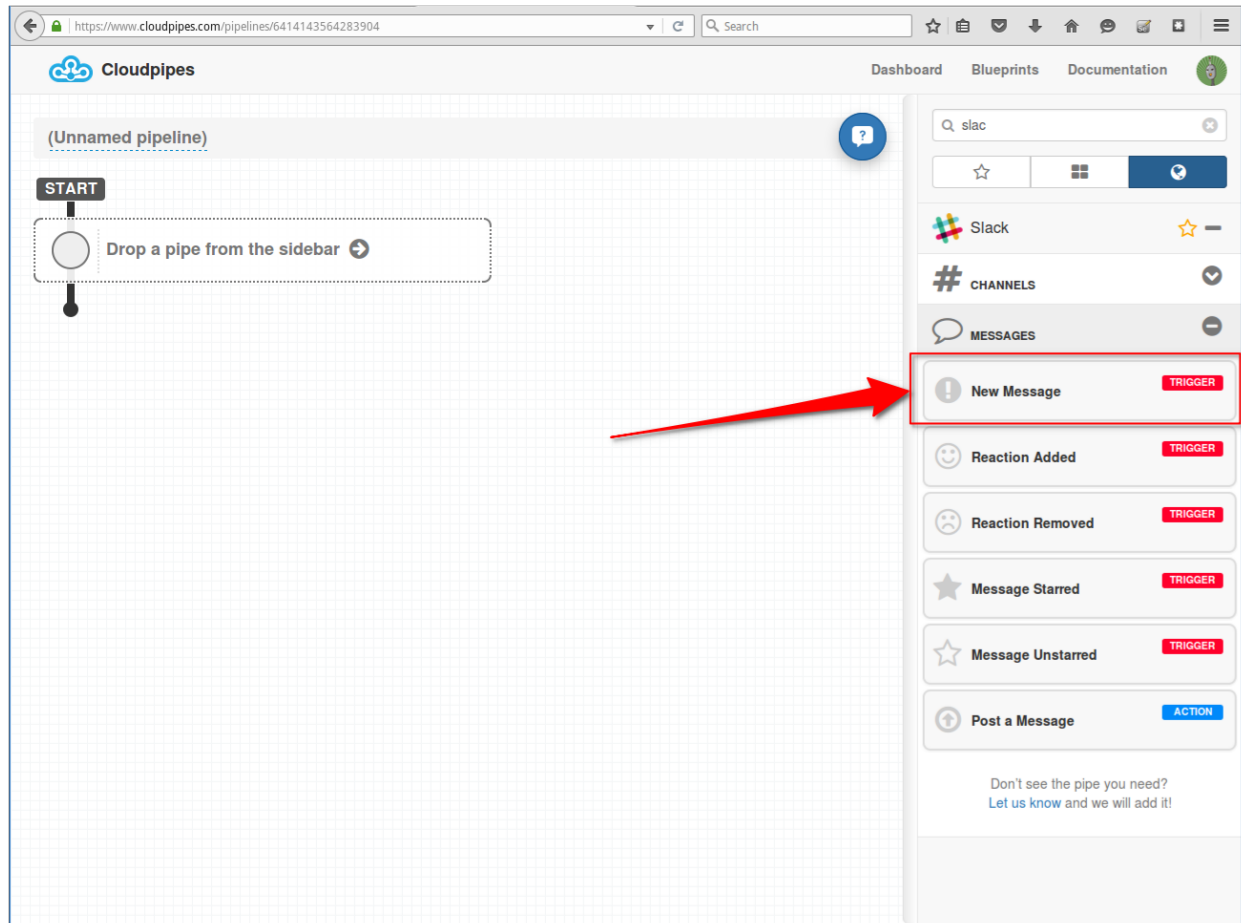
### How to do it

Let’s start by creating a new, blank *pipeline* by clicking the **Create a Pipeline** button on the dashboard.

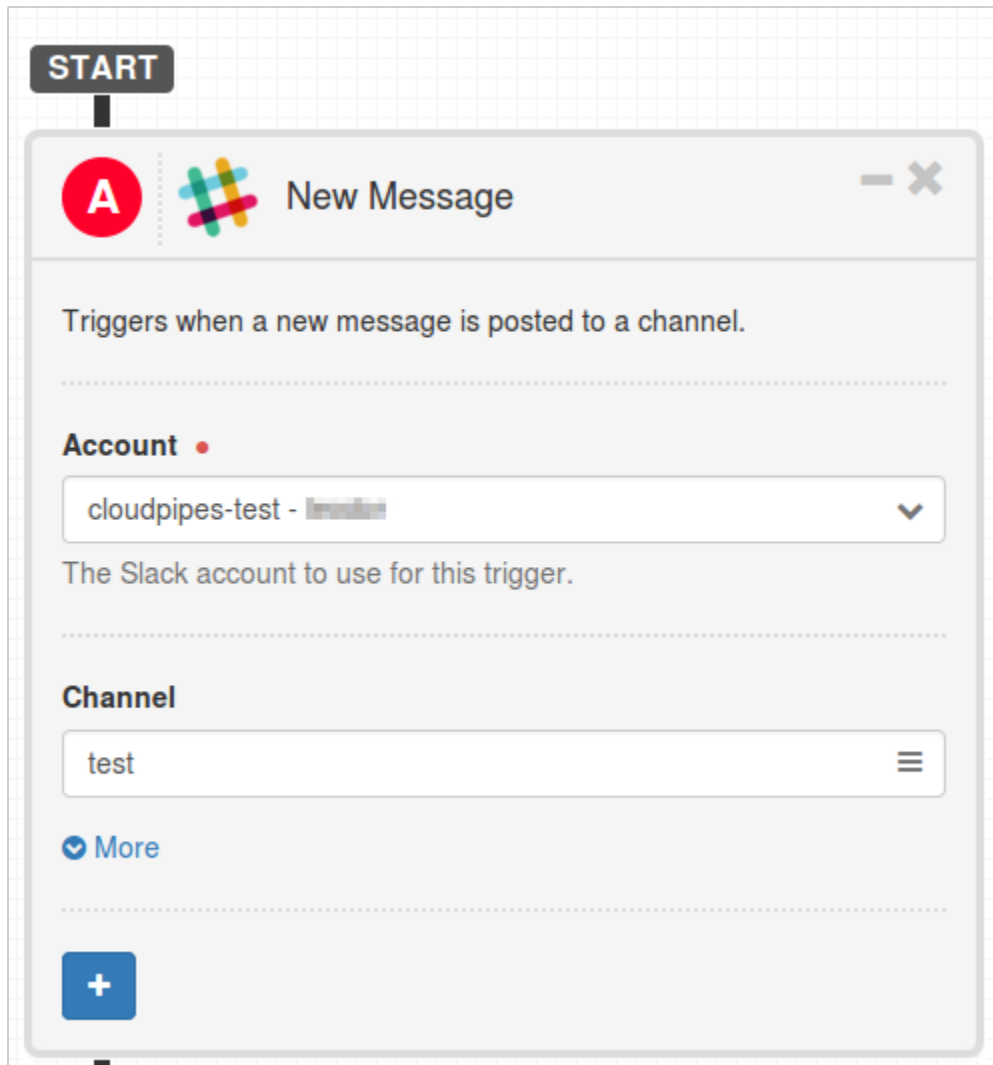


The first pipe that needs to be added to the pipeline is a *trigger*, that will fire whenever a new message is posted to Slack. We need to locate Slack in the channels list on the right, then drag and drop the “New Message” trigger.





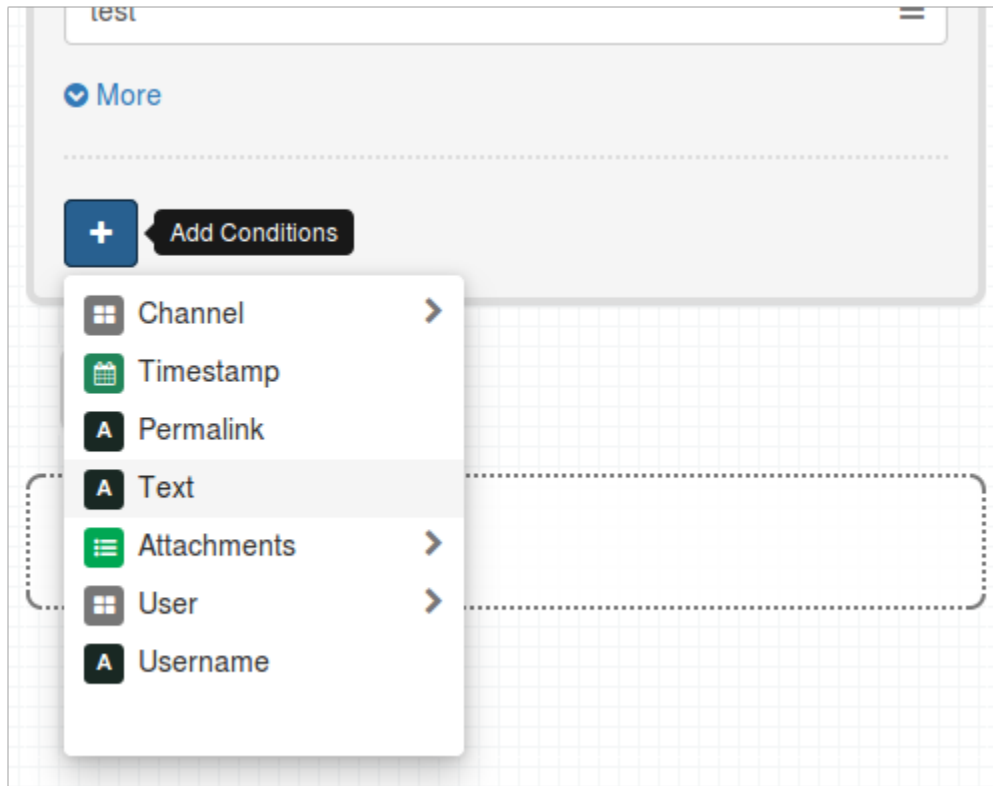
After dropping the *pipe* in the first slot of the pipeline, select your *account* (or connect a new one if needed), then select the Slack channel you are interested in monitoring.



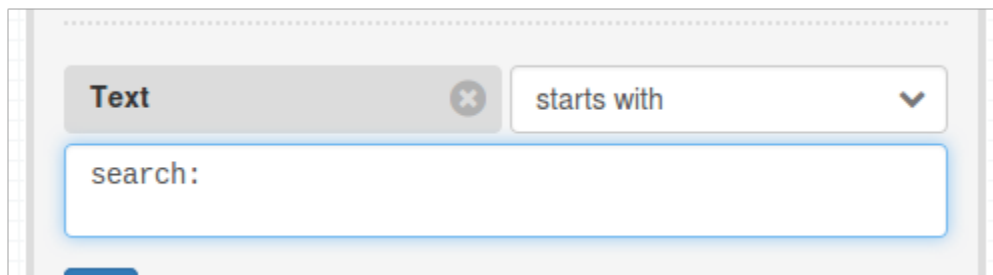
Here we want to act on Slack messages that start with “search:” so we need to add a **filter** to the pipe. **Filters** make sure that only events that match will fire the trigger and execution of the pipeline will continue. Filters are supported on pipes that have the **Add filter** button near the end (blue *plus* icon)

### Adding a Filter

Click the **Add filter** button (blue *plus* icon) and select *Text* as the field to filter on.



We are only interested in messages that start with “search:”, so in the filter condition drop-down select *starts with* and in the filter value enter “search:”. This will make sure that the trigger will fire only on messages that start with “search:”

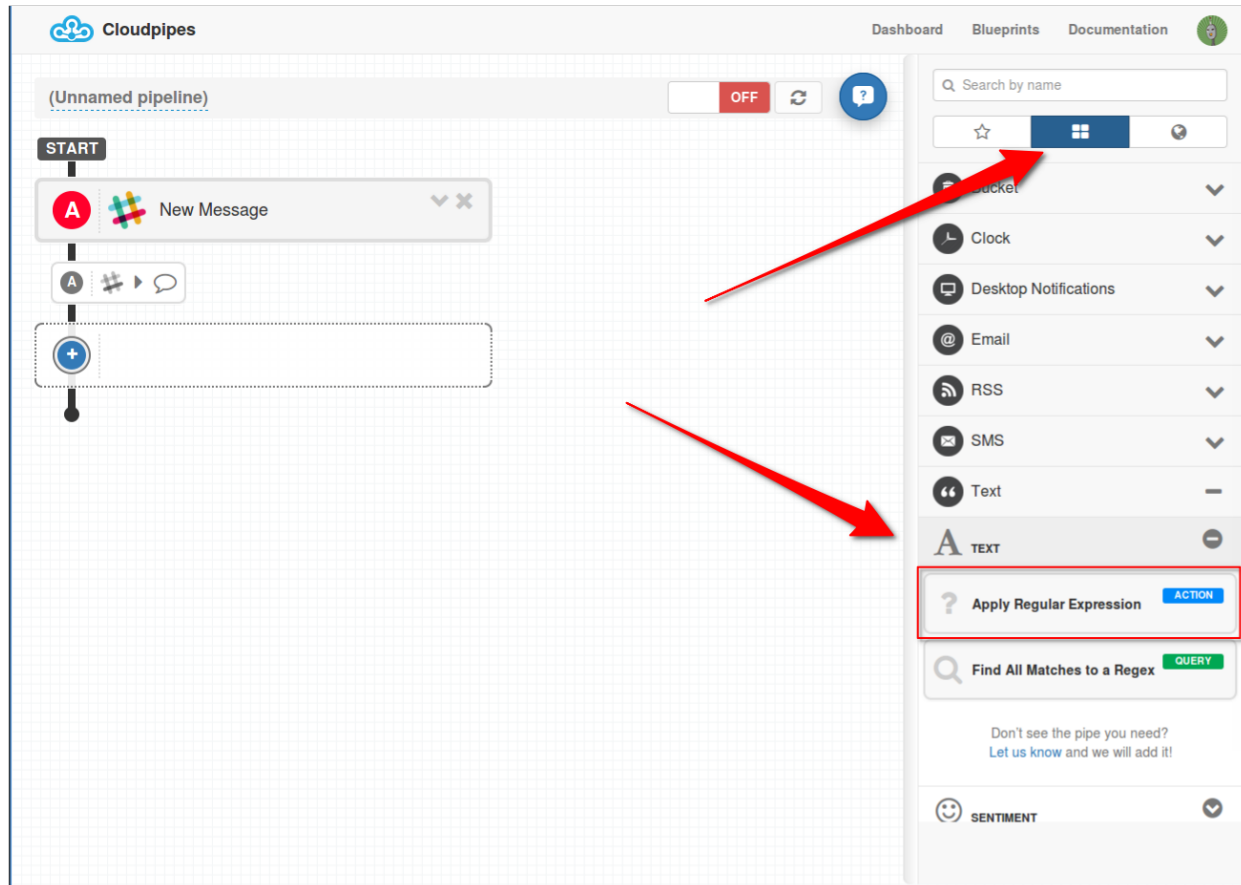


Now, since the *New Message* pipe outputs the entire message that was sent to Slack, we need to split it into two parts - “search:” which we’ll call “command” and the string that was searched for, which we call “term”.

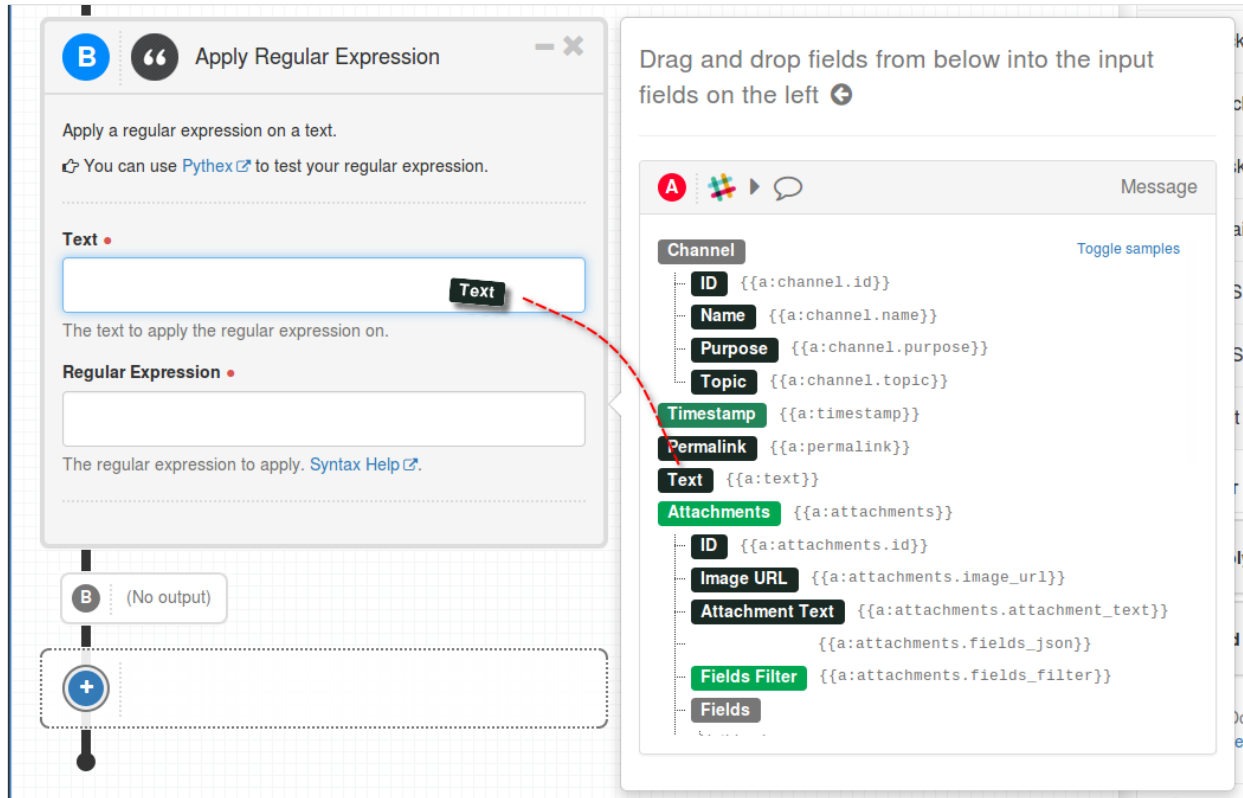
### Splitting Text With a Regular Expression

In order to do that we will use a *Regular Expression*. Regular Expressions (also called “regexes”) are extremely flexible and powerful and allow you to do all kinds of magic with text. In our example we are going to use one to split the message we have from Slack into two parts – the “command” used (“search” in this example) and the search term.

Locate the “*Apply Regular Expression*” pipe under *Built-in channels* -> *Text* and add it as the next step in the pipeline.



*Apply Regular Expression* has only two parameters – *Text*, the text to apply the regular expression on, and *Regular Expression*, the regex to apply. Since we are interested in splitting the Slack message from the fields list drag and drop *Text*.

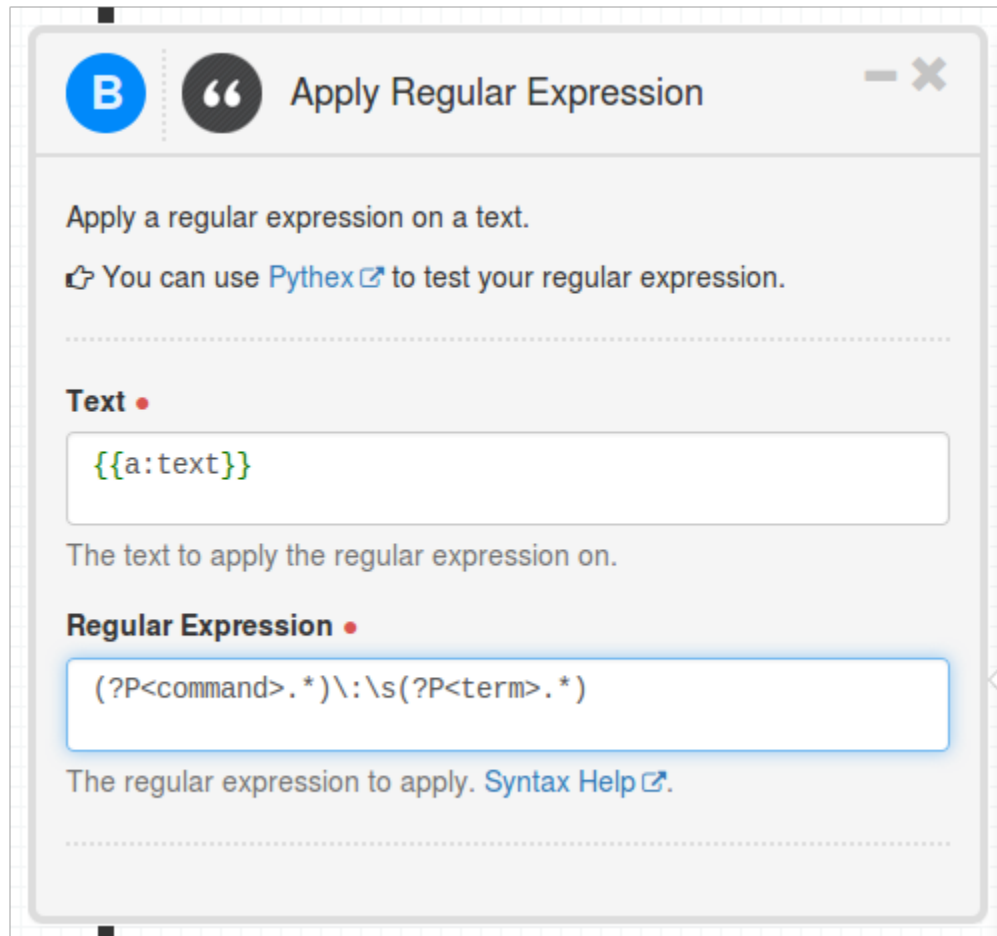


In the *Regular Expression* textbox enter: `(?P<command>.*)\:\s(?P<term>.*)`

This will make sure that in the pipe output we will have two named fields - *command* and *term*. We consider all that's before the colon in the Slack message to be "command" and any characters after to be "term".

**Note:** For a reference of the supported syntax please refer to [this document](#). You can also use [Pythex](#) to test your regular expression.

After you are done the "Apply Regular Expression" pipe should look like this :



The screenshot shows a dialog box titled "Apply Regular Expression" with a blue "B" icon and a quote icon. The dialog contains the following text and fields:

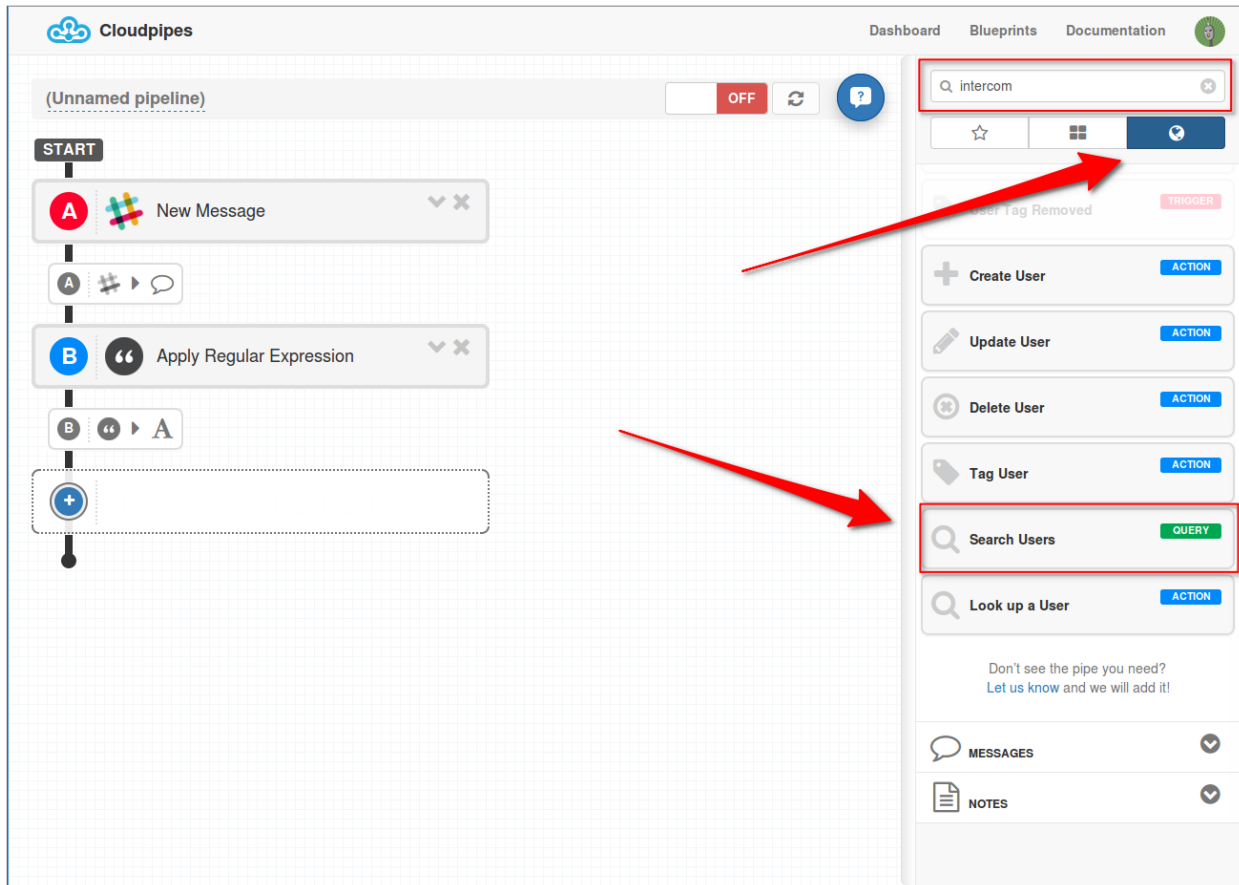
- Instruction: "Apply a regular expression on a text."
- Tip: "You can use [Pythex](#) to test your regular expression."
- Section: "Text" with a red dot icon.
- Text input field: Contains the placeholder text `{{a:text}}`.
- Text: "The text to apply the regular expression on."
- Section: "Regular Expression" with a red dot icon.
- Text input field: Contains the regular expression `(?P<command>.*)\s(?:P<term>.*)`.
- Text: "The regular expression to apply. [Syntax Help](#)."

At this point we now have the message that was posted in Slack split in *command* and *term* which are available in the output of "Apply Regular Expression". Note that because of the filter that we added in the "New Message" trigger the *command* will always be "search".

So now all that's left is to search Intercom and post back to Slack.

### Searching Users in Intercom

Select the "All Channels" tab on the channels list and locate *Intercom*. Note that you can also use the text-box above to search by name for speedier access. Then add the "Search Users" pipe as the next step in the pipeline :



In the “*Search Users*” select the Intercom account that you want to use. If needed connect a new one. Then click the **Add Filter** button (blue *plus* icon) and select *Name*.

**Search Users**

**Account**

[Account Name]

The Intercom account to use for this query.

**Order By**

[Select a field] (unordered)

Optional sort order.

**Limit**

(unlimited)

Optional limit of the number of users returned.

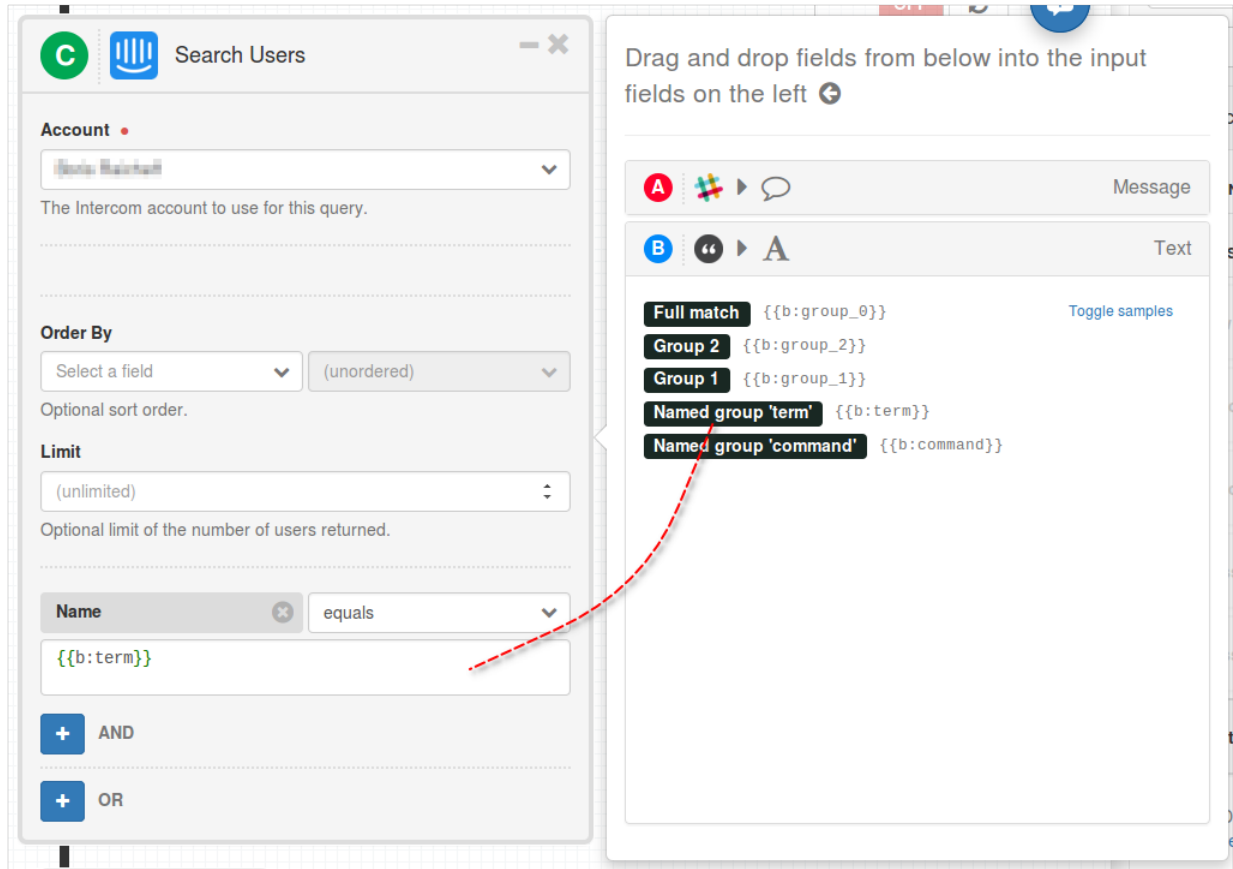
**+ Add Conditions**

- Intercom ID
- User ID
- Email
- Name** (The user's full name.)
- IP
- User Agent
- Created
- Updated

[Filter Value]

In the *filter value* textbox drag and drop the *term* field form the previous pipe.



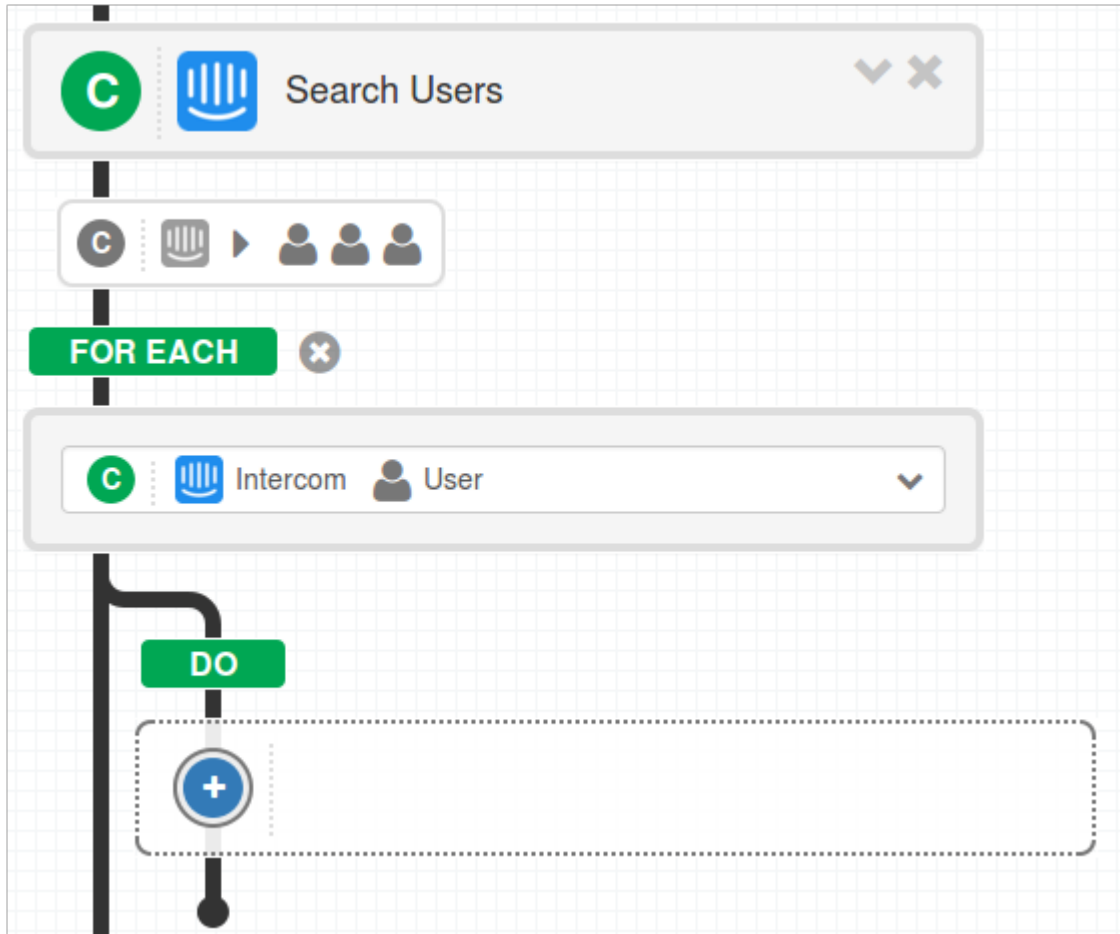


Having “*Search Users*” configured like this will search for user records in Intercom that have their *Name* property match our *term* search field.

The last thing we need to do is to post back to Slack the results we’ve found.

## Posting to Slack

Adding a *Search* pipe automatically adds a *for-each iteration* block after it. Using this we can act on each individual result found.



So now all we need to do is add a *Post Message* action pipe from Slack. In the channels list locate *Slack*. Drag the “*Post Message*” pipe and put it in the slot after “*DO*”.

**FOR EACH** (x)

Intercom User

**DO**

**D** Post a Message

Posts a new message to a public channel, private group, or IM channel.

**Account** •

cloudpipes-test - [REDACTED]

The Slack account to use for this action.

**Channel** •

Select a Channel

Channel to send message to. Can be a public channel, private group or IM channel. Can be an encoded ID, or a name.

**Text**

[Text Input]

Text of the message to send.

**Post as User**

No

Whether to post the message as a user (yourself), instead of as a

Drag and drop fields from below into the input fields on the left ➡

**Message**

**Text**

**User**

Intercom ID {{c:id}} [Toggle samples](#)

User ID {{c:user\_id}}

Email {{c:email}}

Name {{c:name}}

IP {{c:ip}}

User Agent {{c:user\_agent}}

Created {{c:created\_at}}

Updated {{c:updated\_at}}

Last Request {{c:request\_at}}

Signed Up {{c:signed\_up\_at}}

Time Zone {{c:timezone}}

Segments {{c:segments}}

Tags {{c:tags}}

Social Profiles {{c:social\_profiles}}

You can now select a specific channel in your Slack that you want the results posted to. Alternatively drag and drop *ID* from the list of fields to the right after expanding the first list of fields (the one from the “A” pipe).

From the fields list drag and drop the fields you want posted to Slack in response to our query. Note that since we are posting to Slack we can make use of the [markdown-like formatting](#) they support.

Post a Message

Posts a new message to a public channel, private group, or IM channel.

Account

cloudpipes-test

The Slack account to use for this action.

Channel

{{a:channel.id}}

Channel to send message to. Can be a public channel, private group or IM channel. Can be an encoded ID, or a name.

Text

```

*name: {{c:name}}*
*email:* {{c:email}}
*created:* {{c:signed_up_at}}
*last updated:* {{c:updated_at}}
*social:* {% for sprofile in c.social_profiles %}
    *{{sprofile.name}}* - _{{sprofile.username}}_
{% else %}
    _no social profiles_
{% endfor %}
*-----*

```

Text of the message to send.

Post as User

No

Whether to post the message as a user (yourself), instead of as a bot.

Username

Cloudpipes

Name of bot.

More

Drag and drop fields from below into the input fields on the left

A

Message

B

Text

C

User

Updated

{{c:updated\_at}}

Toggle samples

Last Request

{{c:request\_at}}

Signed Up

{{c:signed\_up\_at}}

Time Zone

{{c:timezone}}

Segments

{{c:segments}}

Tags

{{c:tags}}

Social Profiles

{{c:social\_profiles}}

ID

{{c:social\_profiles.id}}

Name

{{c:social\_profiles.name}}

Username

{{c:social\_profiles.username}}

URL

{{c:social\_profiles.url}}

Companies

{{c:companies}}

ID

{{c:companies.id}}

Name

{{c:companies.name}}

Company ID

{{c:companies.company\_id}}

Created At

{{c:companies.created\_at}}

For our example in the *Text* textbox we are going to enter this:

```

*name: {{c:name}}*
*email:* {{c:email}}
*created:* {{c:signed_up_at}}
*last updated:* {{c:updated_at}}
*social:* {% for sprofile in c.social_profiles %}
    *{{sprofile.name}}* - _{{sprofile.username}}_
{% else %}
    _no social profiles_
{% endfor %}
*-----*

```

As you can see we've used many fields from Intercom's result and used Slack formatting. We also made use of advanced field operations, but that's for another tutorial and another time.

One optional step we can take is to add one more "*Post Message*" pipe to the very end of the pipeline that simply says "**End List**", so even for the cases when the search returns no results you will get a note in Slack and know that it was an empty list.

## Conclusion

This is all there is to it really. In this brief tutorial we’ve covered all that’s needed to build quite a sophisticated pipeline that automates your workflow for Intercom by using Slack to search for Users and query user data. We’ve used a regular expression to correctly split “commands” in Slack and also *filters* to only act on commands we are interested in.

Should you need any assistance, as usual just drop us a line at [support@cloudpipes.com](mailto:support@cloudpipes.com) or reach us in the in-app chat.

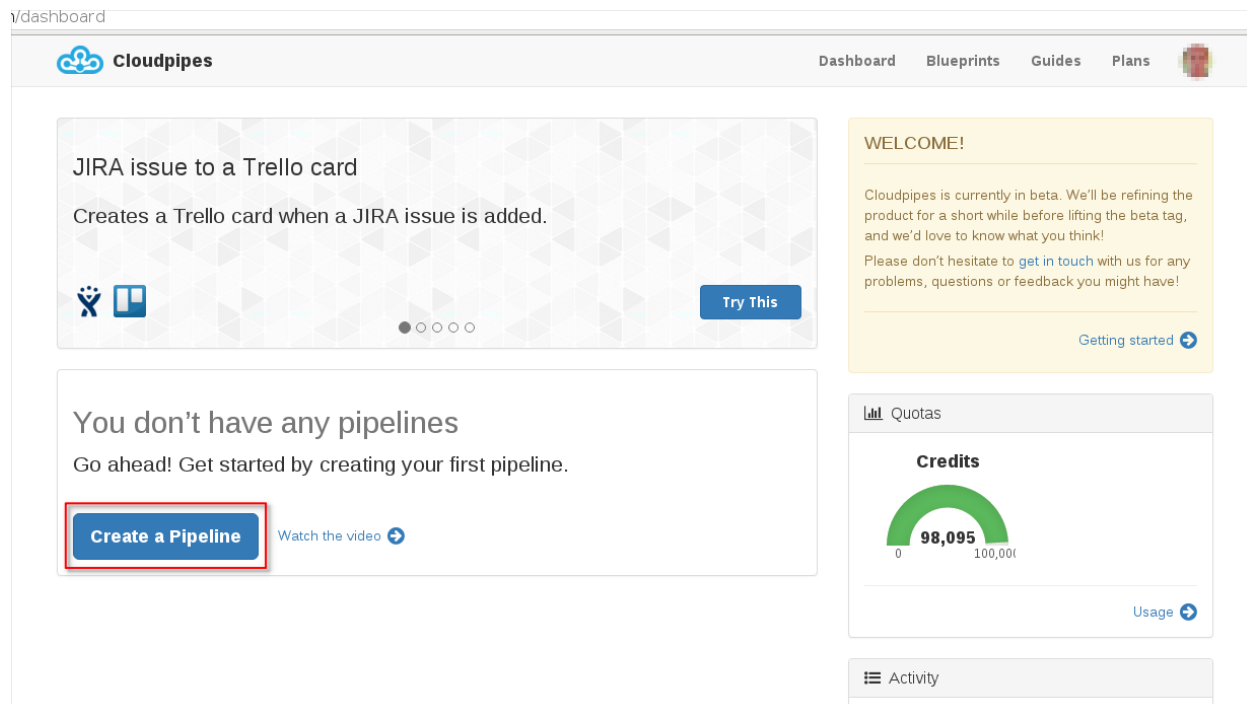
Happy integrating with **Cloudpipes**!

### 1.9.3 Query Trello Cards Using a Command in Slack

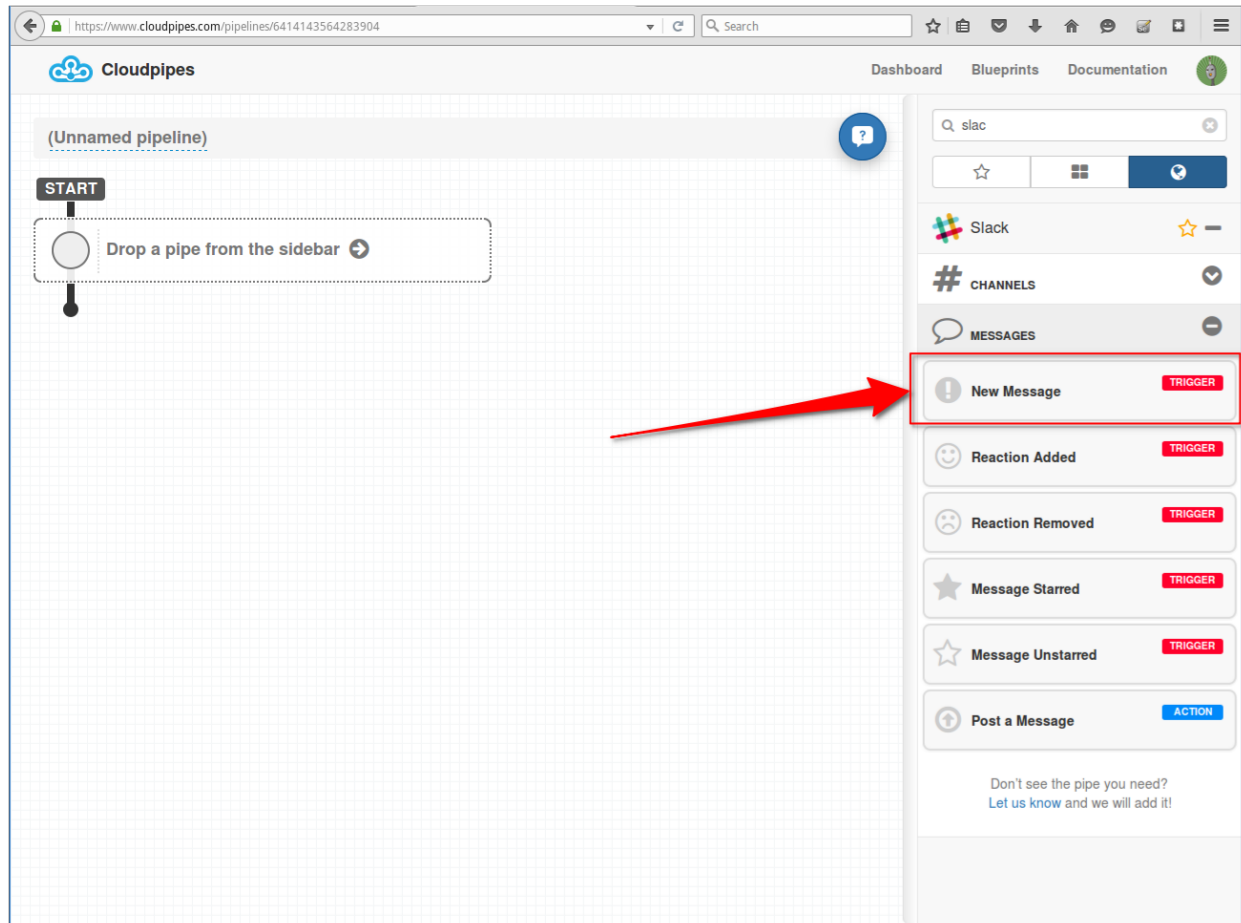
In this tutorial we are going to build a pipeline that listens on new messages in Slack that start with “search:”. It then searches Trello Cards that have matching names and posts selected information back to the Slack channel.

#### How to do it

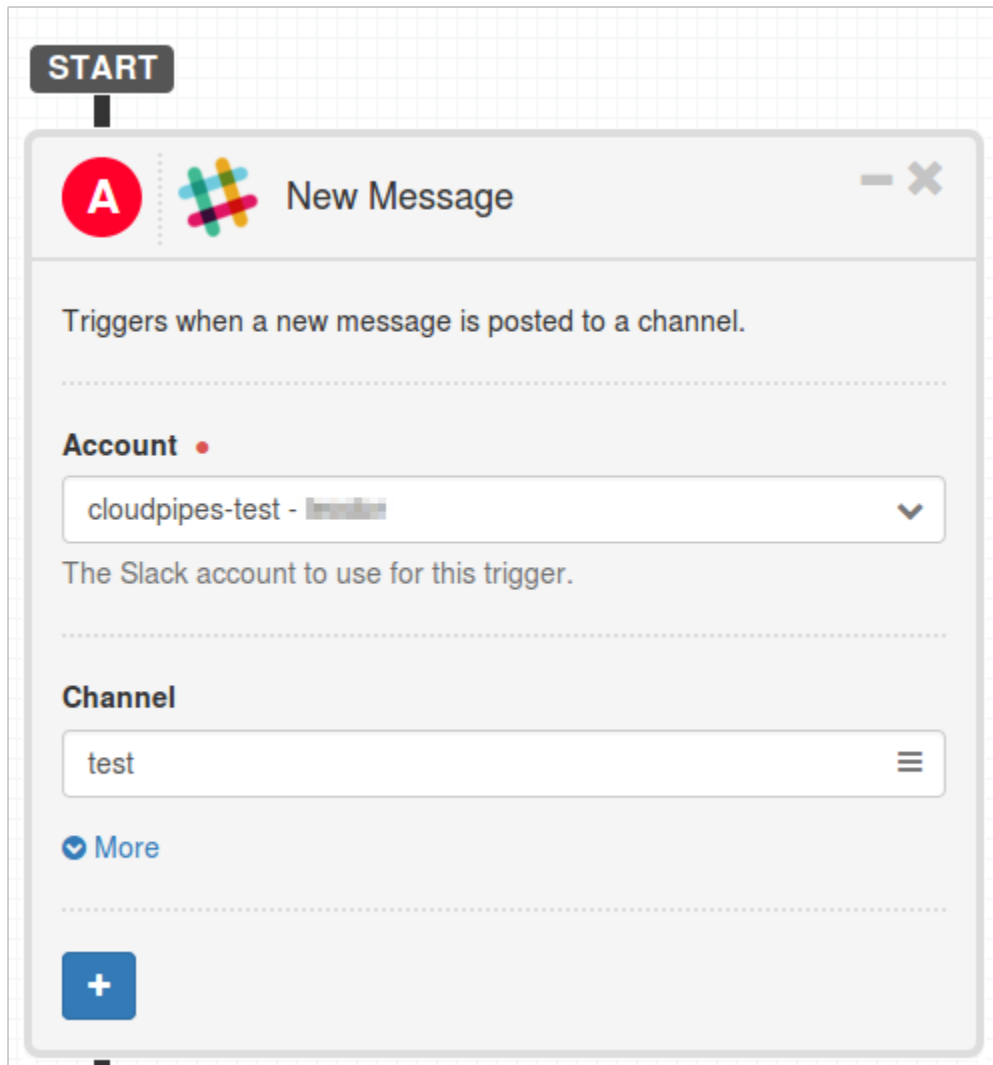
Let’s start by creating a new, blank *pipeline* by clicking the **Create a Pipeline** button on the dashboard.



The first pipe that needs to be added to the pipeline is a *trigger*, that will fire whenever a new message is posted to Slack. We need to locate Slack in the channels list on the right, then drag and drop the “New Message” trigger.



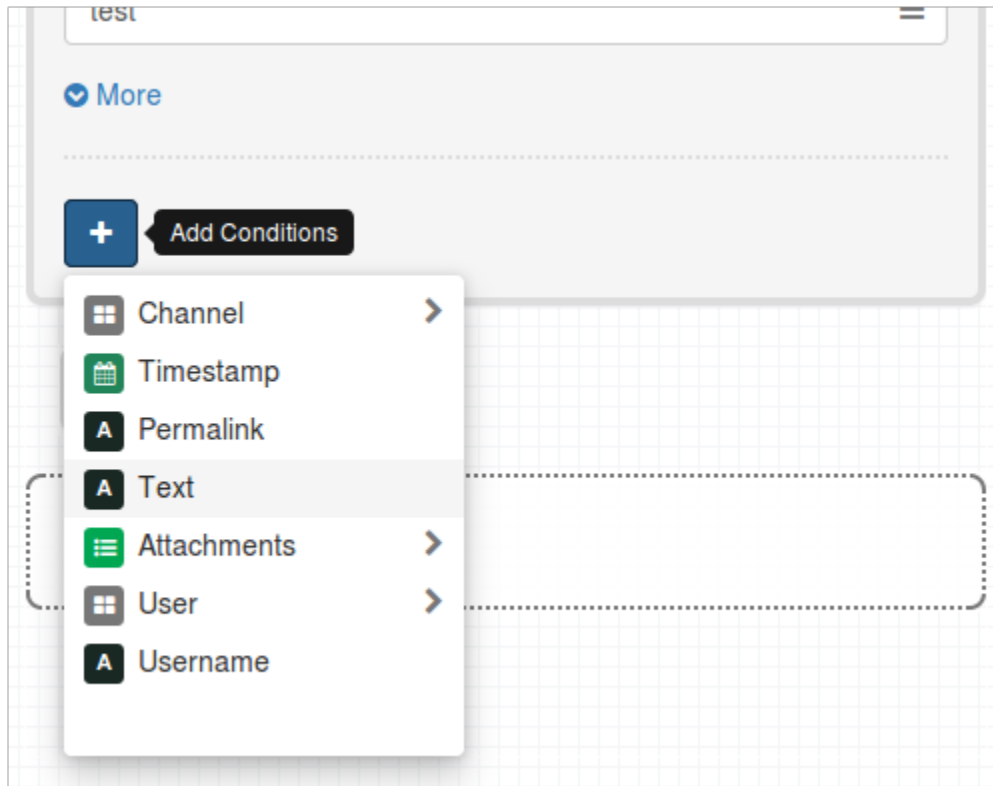
After dropping the *pipe* in the first slot of the pipeline, select your *account* (or connect a new one if needed), then select the Slack channel you are interested in monitoring.



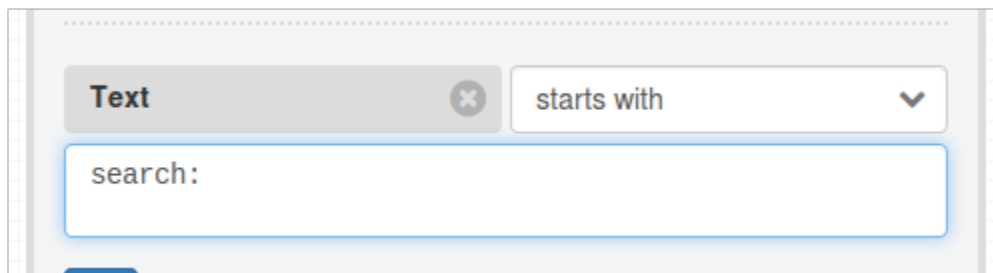
Here we want to act on Slack messages that start with “search:” so we need to add a **filter** to the pipe. **Filters** make sure that only events that match will fire the trigger and execution of the pipeline will continue. Filters are supported on pipes that have the **Add filter** button near the end (blue *plus* icon)

### Adding a Filter

Click the **Add filter** button (blue *plus* icon) and select *Text* as the field to filter on.



We are only interested in messages that start with “search:”, so in the filter condition drop-down select *starts with* and in the filter value enter “search:”. This will make sure that the trigger will fire only on messages that start with “search:”



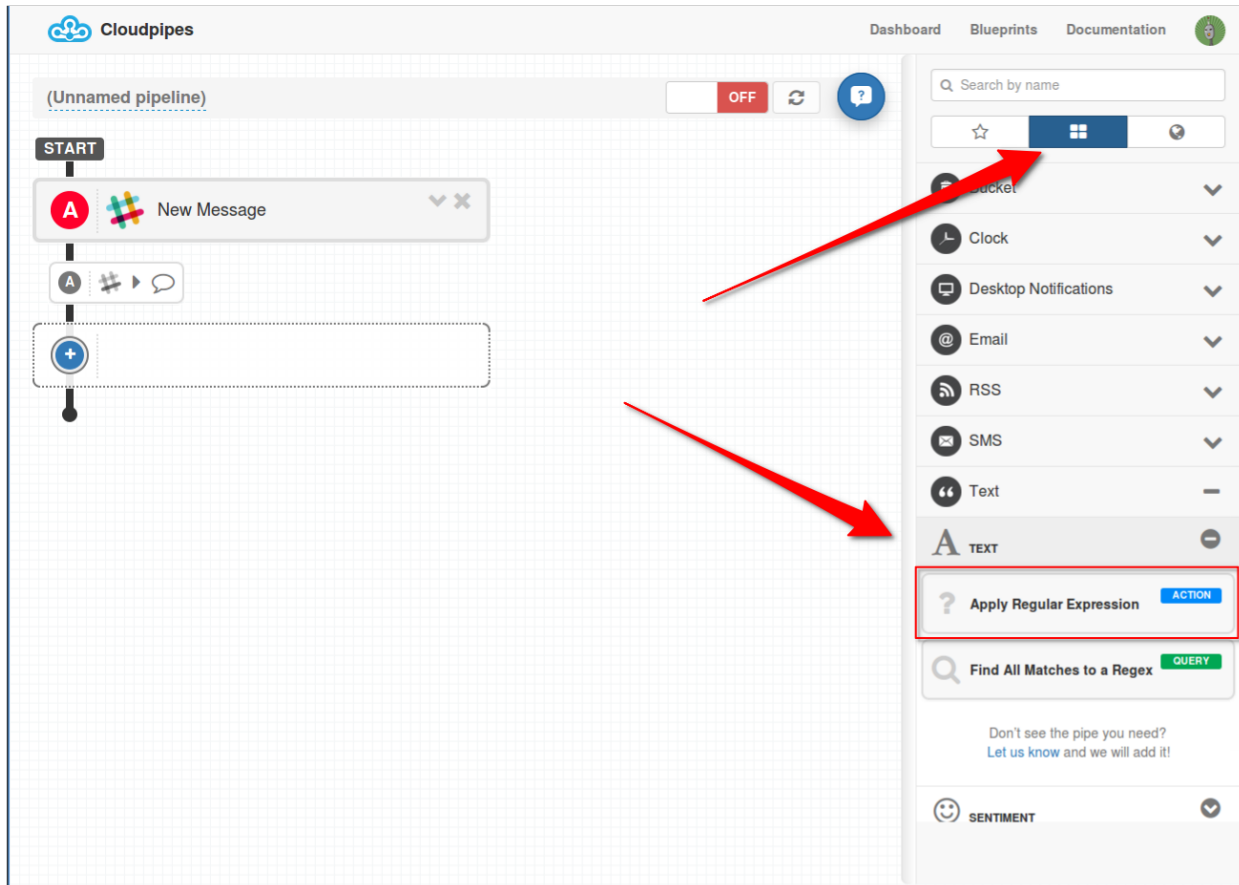
Now, since the *New Message* pipe outputs the entire message that was sent to Slack, we need to split it into two parts - “search:” which we’ll call “command” and the string that was searched for, which we call “term”.

### Splitting Text With a Regular Expression

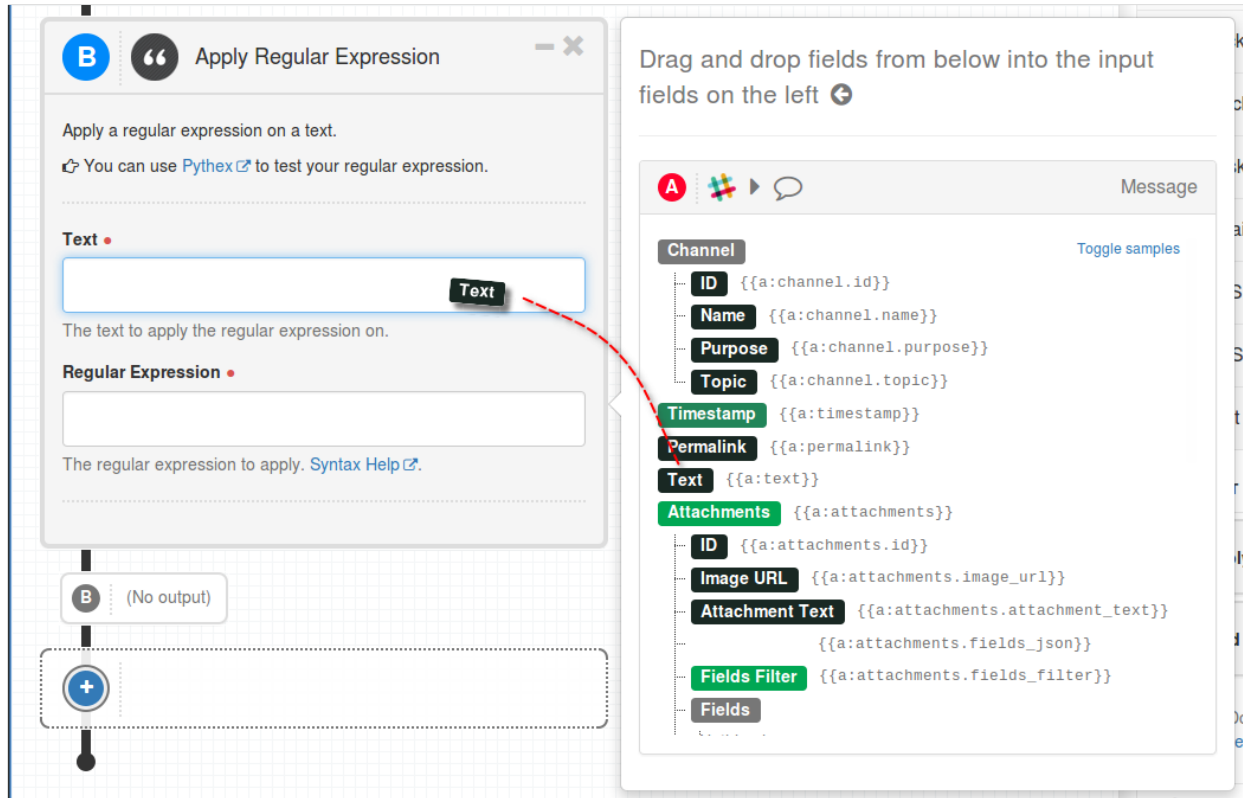
In order to do that we will use a *Regular Expression*. Regular Expressions (also called “regexes”) are extremely flexible and powerful and allow you to do all kinds of magic with text. In our example we are going to use one to split the message we have from Slack into two parts – the “command” used (“search” in this example) and the search term.

Locate the “*Apply Regular Expression*” pipe under *Built-in channels* -> *Text* and add it as the next step in the pipeline.





*Apply Regular Expression* has only two parameters – *Text*, the text to apply the regular expression on, and *Regular Expression*, the regex to apply. Since we are interested in splitting the Slack message from the fields list drag and drop *Text*.



In the *Regular Expression* textbox enter: `(?P<command>.*):\s(?P<term>.*)`

This will make sure that in the pipe output we will have two named fields - *command* and *term*. We consider all that's before the colon in the Slack message to be "command" and any characters after to be "term".

**Note:** For a reference of the supported syntax please refer to [this document](#). You can also use [Pythex](#) to test your regular expression.

After you are done the "Apply Regular Expression" pipe should look like this :

**B** “ Apply Regular Expression

Apply a regular expression on a text.

🔗 You can use [Pythex](#) to test your regular expression.

---

**Text** •

`{{a:text}}`

The text to apply the regular expression on.

**Regular Expression** •

`(?P<command>.*)\:\\s(?P<term>.*)`

The regular expression to apply. [Syntax Help](#).

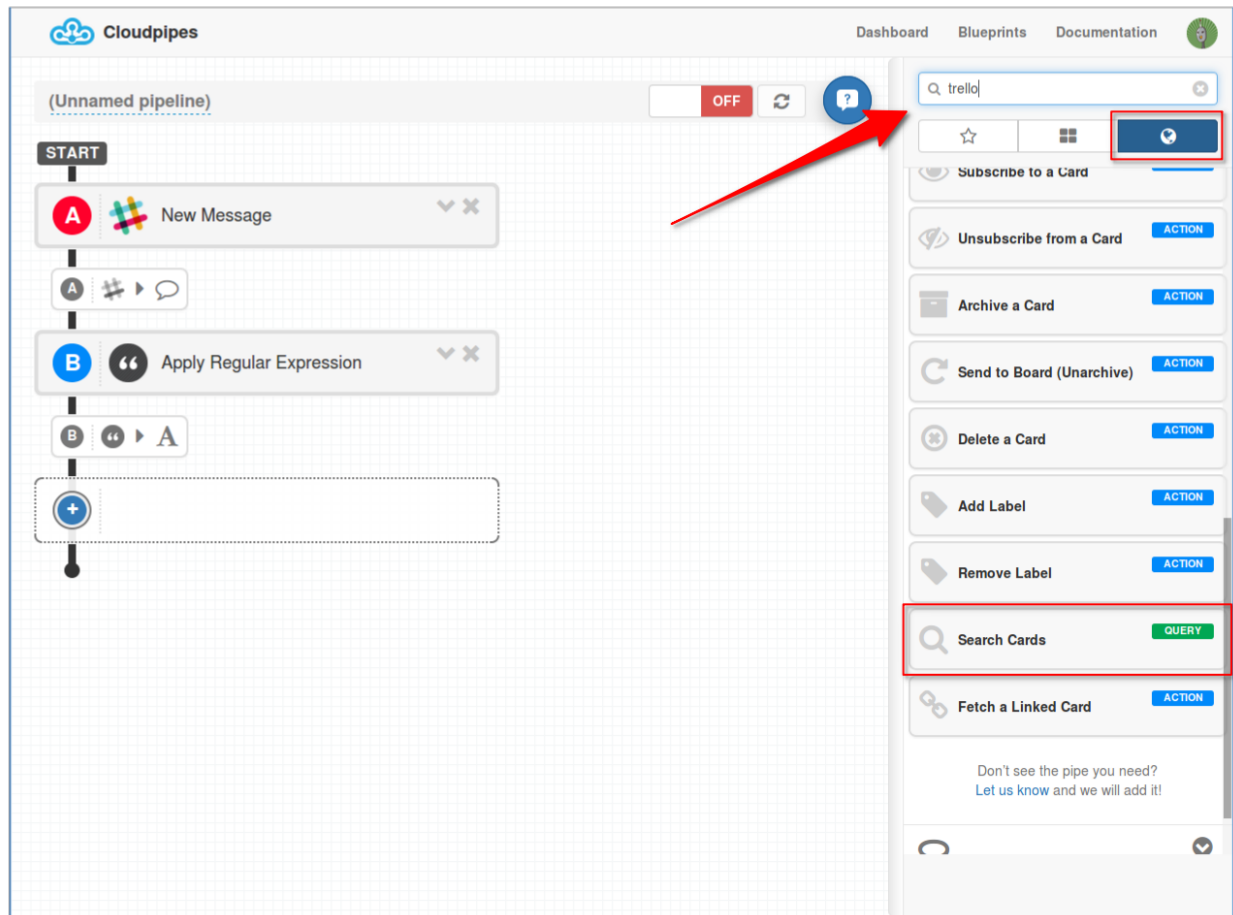
---

At this point we now have the message that was posted in Slack split in *command* and *term* which are available in the output of “Apply Regular Expression”. Note that because of the filter that we added in the “New Message” trigger the *command* will always be “search”.

So now all that’s left is to search Trello and post back to Slack.


### Searching Cards in Trello

Select the *All Channels* tab on the channels list and locate *Trello*. Note that you can also use the text-box above to search by name for speedier access. Then add the *Search Cards* pipe as the next step in the pipeline :




In the *Search Cards* pipe select the Trello account that you want to use (If needed connect a new one) and the Board to search in. Then click the “Add Filter” button (blue “plus” icon) and select *Name*.

C



Search Cards



Lists cards in the given board and list, applying optional filter conditions.

Account

The Trello account to use for this query.

Board

test

List

Order By

Select a field

(unordered)

Optional sort order.

Limit

(unlimited)

Optional limit of the number of cards returned.


+


Add Conditions


A ID

A Name

A Description

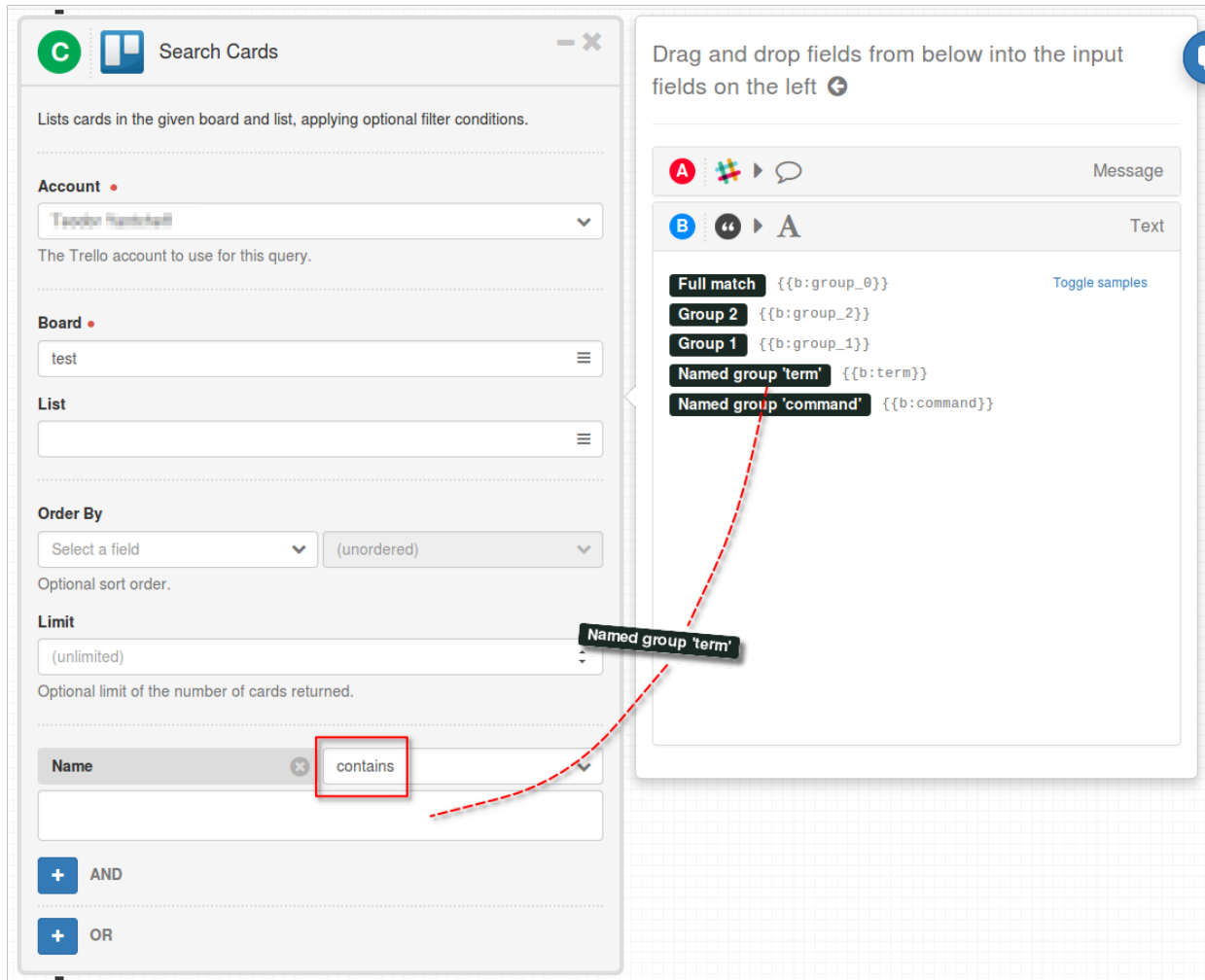
 Due Date

 Position

 Updated At

The name of the card.

From the *filter condition* drop-down select *contains* then in the *filter value* textbox drag and drop the “*term*” field from the previous pipe.

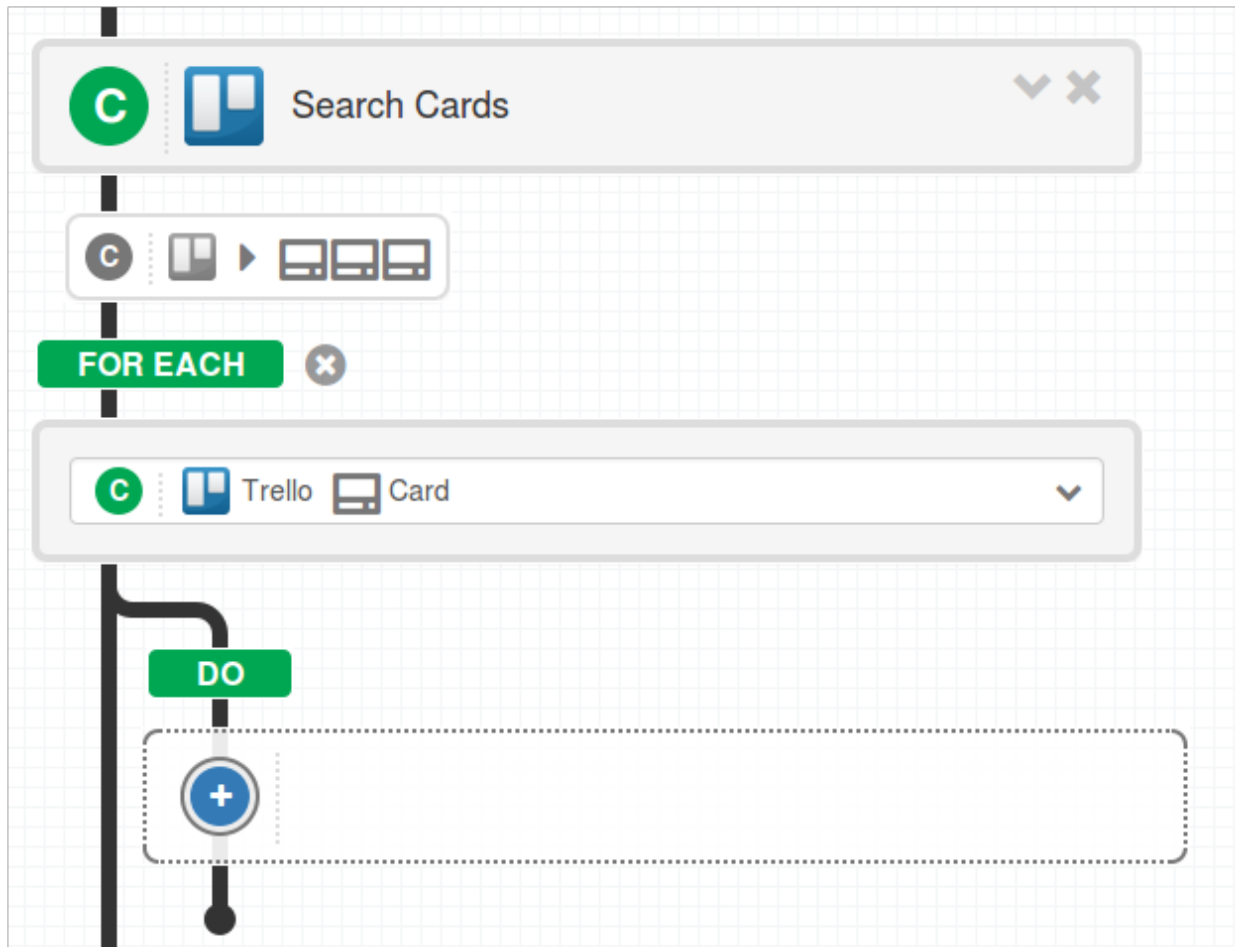


Having *Search Cards* configured like this will search for Cards in the Board you selected in Trello that contain our *term* search field in their *Name* property.

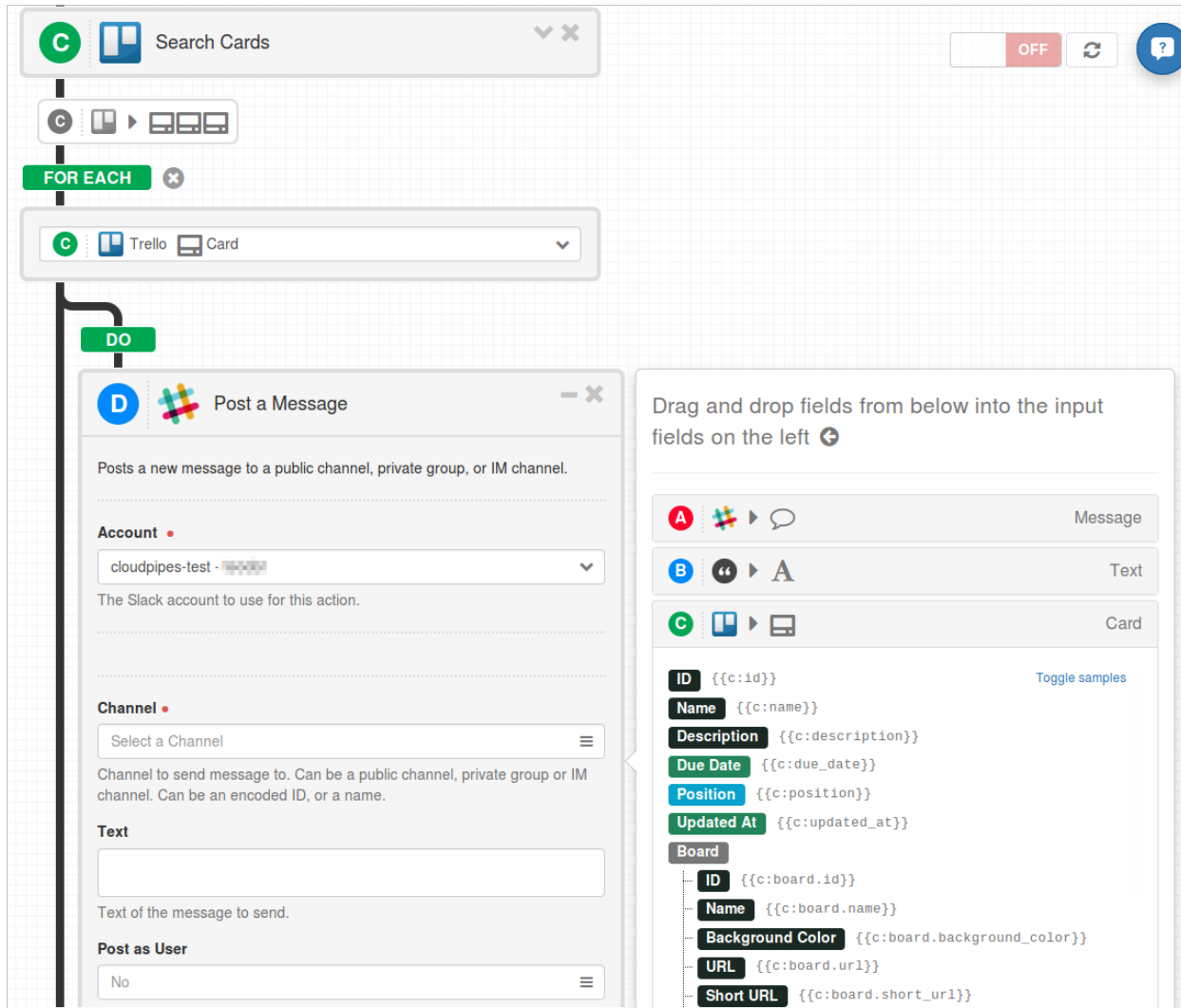
The last thing we need to do is to post back to Slack the results we’ve found.

## Posting to Slack

Adding a *Search* pipe automatically adds a *for-each iteration* block after it. Using this we can act on each individual result found.



So now all we need to do is add a “*Post Message*” action pipe from Slack. In the channels list locate *Slack*. Drag the “*Post Message*” pipe and put it in the slot after “*DO*”.



You can now select a specific channel in your Slack that you want the results posted to. Alternatively drag and drop *ID* from the list of fields to the right after expanding the first list of fields (the one from the “A” pipe).

From the fields list drag and drop the fields you want posted to Slack in response to our query. Note that since we are posting to Slack we can make use of the [markdown-like formatting](#) Slack supports.



**DO**

**D** **Post a Message**

Posts a new message to a public channel, private group, or IM channel.

**Account**

cloudpipes-test

The Slack account to use for this action.

**Channel**

{{a:channel.id}}

Channel to send message to. Can be a public channel, private group or IM channel. Can be an encoded ID, or a name.

**Text**

Cards matching "{{b:term}}":

```
*name: {{c:name}}
*description: {{c:description}}
*due: {{c:due_date}}
*list: {{c:list.name}}
*labels: {% for label in c.labels %}
  *{{label.name}}* - _{{label.color}}_
{% else %}
  no labels
{% endfor %}
```

Text of the message to send.

**Post as User**

No

Whether to post the message as a user (yourself), instead of as a bot.

**Username**

Cloudpipes

Name of bot.

Drag and drop fields from below into the input fields on the left

**Message**

**Text**

**Card**

**List** [Toggle samples](#)

- ID** {{c:list.id}}
- Name** {{c:list.name}}
- Position** {{c:list.position}}
- Closed** {{c:list.closed}}
- Board ID** {{c:list.board}}

**Labels** {{c:labels}}

- ID** {{c:labels.id}}
- Name** {{c:labels.name}}
- Color** {{c:labels.color}}
- Uses** {{c:labels.uses}}

**Members** {{c:members}}

- ID** {{c:members.id}}
- Username** {{c:members.username}}
- Full Name** {{c:members.full\_name}}

For our example in the *Text* textbox we are going to enter this:

```
*name: {{c:name}}*
*description: {{c:description}}
*due: {{c:due_date}}
*list: {{c:list.name}}
*labels: {% for label in c.labels %}
  *{{label.name}}* - _{{label.color}}_
{% else %}
  no labels
{% endfor %}
*-----*
```

As you can see we've added many fields from Trello's result and used Slack formatting. We also made use of advanced field operations, but that's for another tutorial and another time.

One optional step we can take is to add one more "Post Message" pipe to the very end of the pipeline that simply says "End List", so even for the cases when the search returns no results you will get a note in Slack and know that it was an empty list.

### Conclusion

This is all there is to it really. In this brief tutorial we've covered all that's needed to build quite a sophisticated pipeline that automates your workflow for Trello by using Slack to search for Cards and query data. We've used a regular expression to correctly split "commands" in Slack and also *filters* to only act on commands we are interested in.

Should you need any assistance, as usual just drop us a line at [support@cloudpipes.com](mailto:support@cloudpipes.com) or reach us in the in-app chat.

Happy integrating with **Cloudpipes**!

## 1.10 Channel Documentation

Here you will find guides specific to the relevant channels.

### 1.10.1 Clock Syntax Guide

You can use the syntax below to specify the amount of time to wait.

Supports expressions like the following:

```
10s
25seconds
32m
2h32m
3d2h32m
1w3d2h32m
1w 3d 2h 32m
1 w 3 d 2 h 32 m
4:13
4:13:02
4:13:02.266
2:04:13:02.266
2 days, 4:13:02
2 days, 4:13:02.266
5hr34m56s
5 hours, 34 minutes, 56 seconds
5 hrs, 34 mins, 56 secs
2 days, 5 hours, 34 minutes, 56 seconds
1.2 m
1.2 min
1.2 mins
1.2 minute
1.2 minutes
172 hours
172 hr
172 h
172 hrs
172 hour
1.24 days
5 d
5 day
5 days
5.6 wk
5.6 week
5.6 weeks
```

**Note:** No suffix means time in seconds. E.g. “15” means “15 seconds”.

As you see these are pretty much self-explanatory. Should you need any assistance, as usual just drop us a line at [support@cloudpipes.com](mailto:support@cloudpipes.com) or reach us in the in-app chat.

Happy integrating with **Cloudpipes**!

## 1.10.2 Codebase Search Guide

You can use the syntax below to search for tickets in your project.

### Search Fields

#### Tickets assigned to user

```
assignee:username | me | none | any
assignee:dave
assignee:me
assignee:none
assignee:any
```

#### Tickets by status

```
status:status_name | open | closed
status:completed
status:open
status:closed
```

#### Tickets by category

```
category:category_name
category:application
```

#### Tickets by type

```
type:type_name
type:bug
```

#### Tickets by priority

```
priority:priority_name
priority:high
```

#### Tickets by milestone

```
milestone:milestone_name
milestone:"Initial Release"
```

### Sorting and Ordering

#### Sort field

```
sort:number
      type
      status
      subject
      resolution
      priority
      category
      assignee
      milestone
      created_at
      updated_at
      deadline
```

#### Sort order

```
sort:asc
      desc
```

### Other search options

#### Negative values

Add a **not** before the field name

```
not status:completed
```

#### Multiple values

Comma separate values:

```
staus:new, accepted
```

Multiple words - Encapsulate any query value with more than one word in double quotes:

```
milestone:"Initial Release"
```

### 1.10.3 Telegram Bot Guide

To add our bot on telegram to your rooster do this :

1. First you need to find the bot on telegram. There's a convenient link in the "Send Message" pipe for telegram **(1)**.
2. Then you need to send to the bot the message that is shown **(2)**

https://telegram.me/cloudpipesbot or search @cloudpipesbot to find our bot.' The URL is highlighted with a red box and a red number '1'. Below this, there is another line of text: 'In order to authorize the bot, send it the message "/connect 5854021822709760".' This message is also highlighted with a red box and a red number '2'. At the bottom, there is a section labeled 'Channel' with a dropdown menu that says 'Select a Channel'."/>

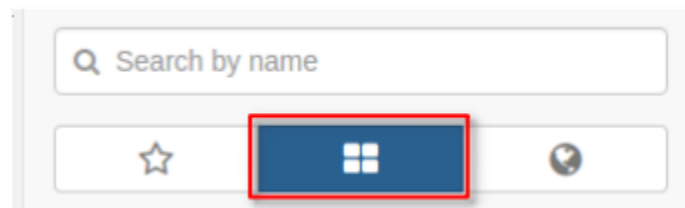
Simple as that!

Should you need any assistance, as usual just drop us a line at [support@cloudpipes.com](mailto:support@cloudpipes.com) or reach us in the in-app chat.

Happy integrating with **Cloudpipes**!

### 1.10.4 Built-in Channels Guide

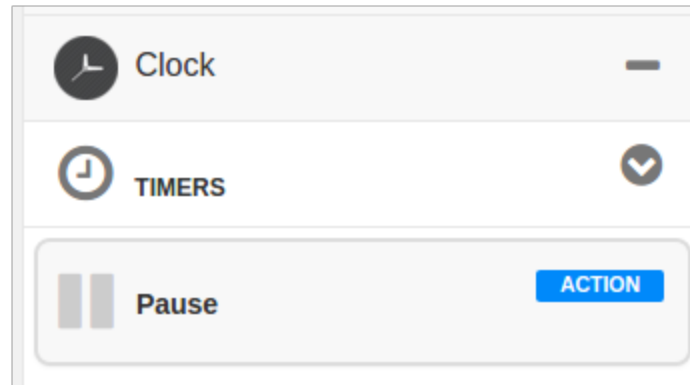
**Built-in Channels** are channels that do not have a 3rd party web application that provides their functionality, but instead have functionality provided entirely by Cloudpipes. Built-in channels can be found in the middle tab on the sidebar.



#### Clock

In the **Clock** section there is currently only one Pipe – **Pause**. **Pause** stops (pauses) pipeline execution for the specified amount of time and then resumes. Think of it as a ‘*sleep*’ function. You can use the **Pause** pipe whenever you need to wait for some known amount of time. For example to avoid updating objects too fast, or to avoid race conditions in your pipelines.

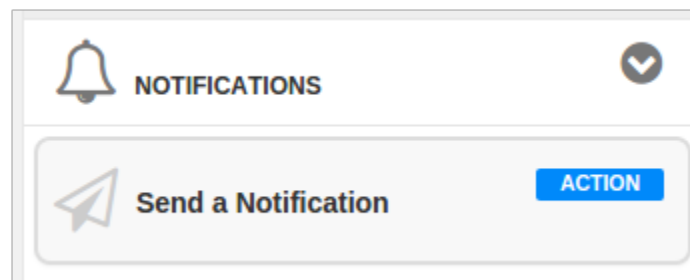
For a detailed description of the supported syntax refer to [Clock Syntax Guide](#).



**Note:** If you need a pipeline to be executed at set-times and regular intervals see [Scheduling](#).

## Desktop Notification

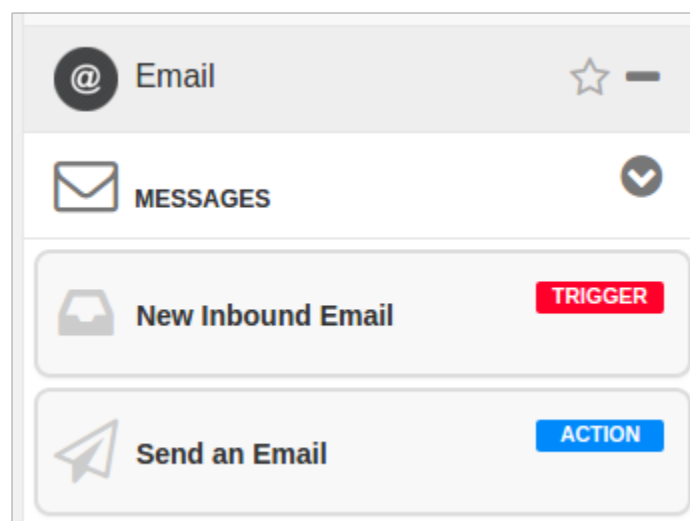
**Send a Notification** pipe will pop a browser desktop notification balloon on screen. Among other things this pipe is also useful for debugging purposes.



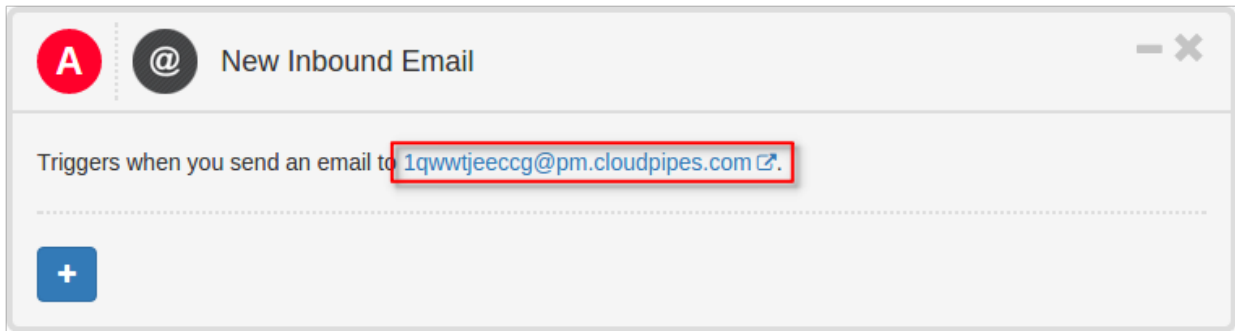
You can configure the balloon *Title*, *Body*, *Icon* and a *URL* you want to be opened once the balloon gets clicked.

## Email

In **Email** you will find pipes that work with a mail server provided by Cloudpipes.



- **[Trigger]** New Inbound Email - Will trigger when a message has been received at the given address:

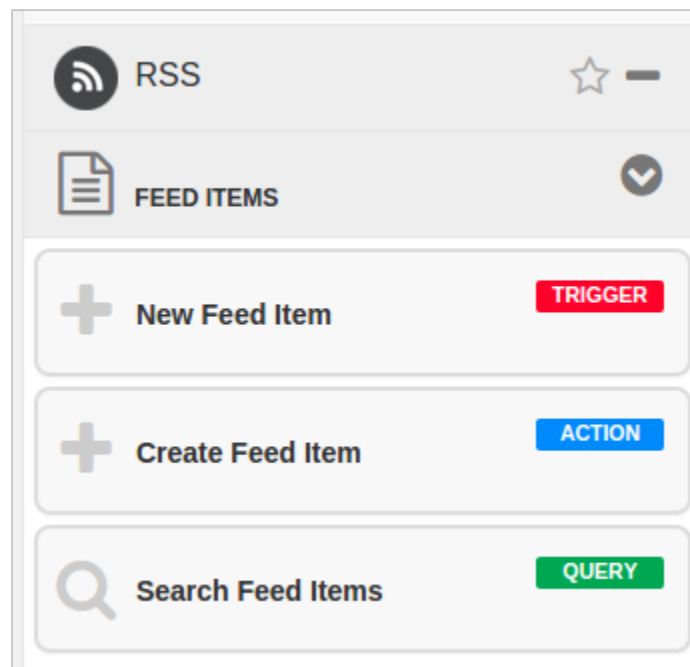


**Note:** Just like with any other pipe that has the blue '+' filter button you can set filter conditions.

- **[Action]** Send an Email - Sends an email. You can set *To*, *Subject*, *Body* and after expanding the *more* fold - *CC* and *BCC*.

**Note:** **Send an Email** is rate limited so that it cannot be used for malicious purposes. Also the number of recipients in the *To*, *CC* and *BCC* are administratively capped for the same reasons.

## RSS



- **[Trigger]** New Feed Item - Will trigger when an item is added to the given feed. *Feed URL* is configurable and after the '*more*' fold you can also specify authentication parameters if needed.

**Note:** Just like with any other pipe that has the blue '+' filter button you can set filter conditions.

- **[Action]** Create Feed Items - is a pipe that allows you to add items to an RSS feed that is served by Cloudpipes. You can name your feed and choose a custom slug. The *URL* at which your feed will be available is also visible in the create dialog:

### RSS Feed

**Title**

**Slug**

**URL**

Close

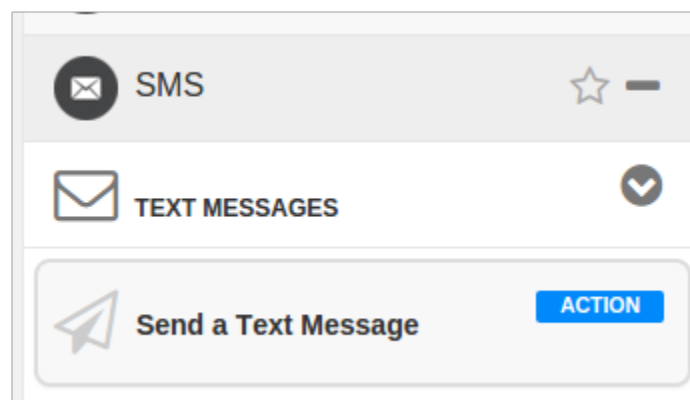
Create

In the 'Create Feed Items' pipe you can specify *Feed*, *Title*, *Description*, *Link* and *Publish Date*. After the *more* fold *Author*, *GUID* and *Updated at* become available as fields.

- **[Query]** Search Feed Items - Allows you to fetch (list) the feed items available at the given *URL*. You can order them by a specific field and limit the number of items to fetch. The blue '+' filter button, as usual, allows you to set filter conditions as usual.

## SMS

The **Send a Text Message** pipe will send a text. You can specify number to send *To* and the *Body* of the text. *Body* is limited in length to 1600 characters.

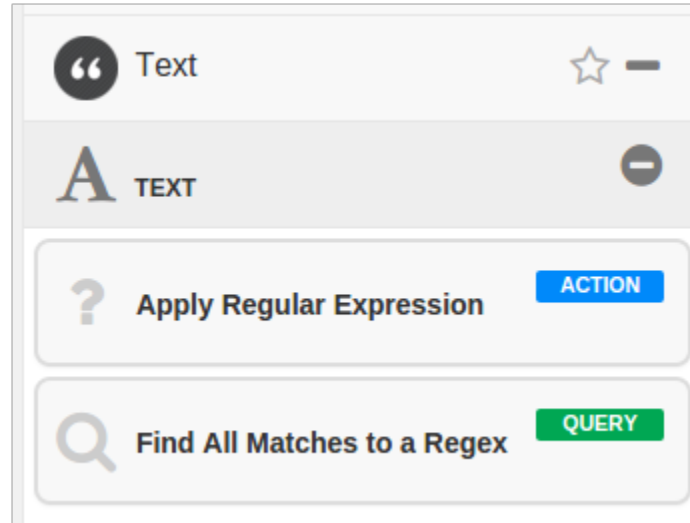


**Note:** Currently only UK numbers can be texted.



## Regular Expression

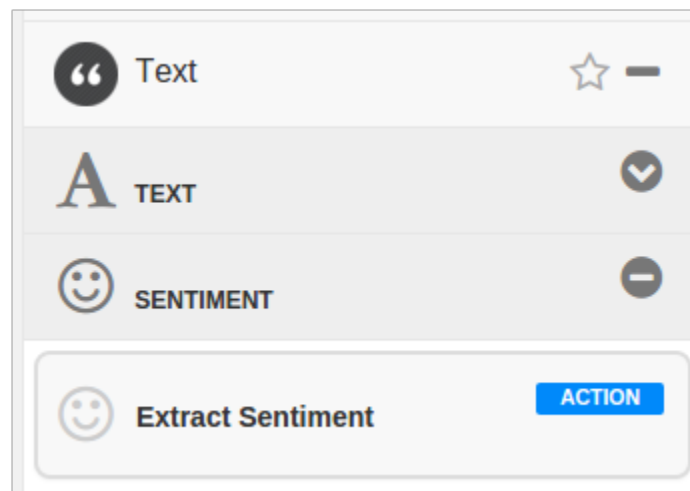
A **Regular Expression** is a sequence of symbols and characters expressing a string or pattern to be searched for within a longer piece of text. **Regular Expressions** are extremely useful for when you have a single field that contains information you need split out into multiple fields. You may only want part of that field to come through.



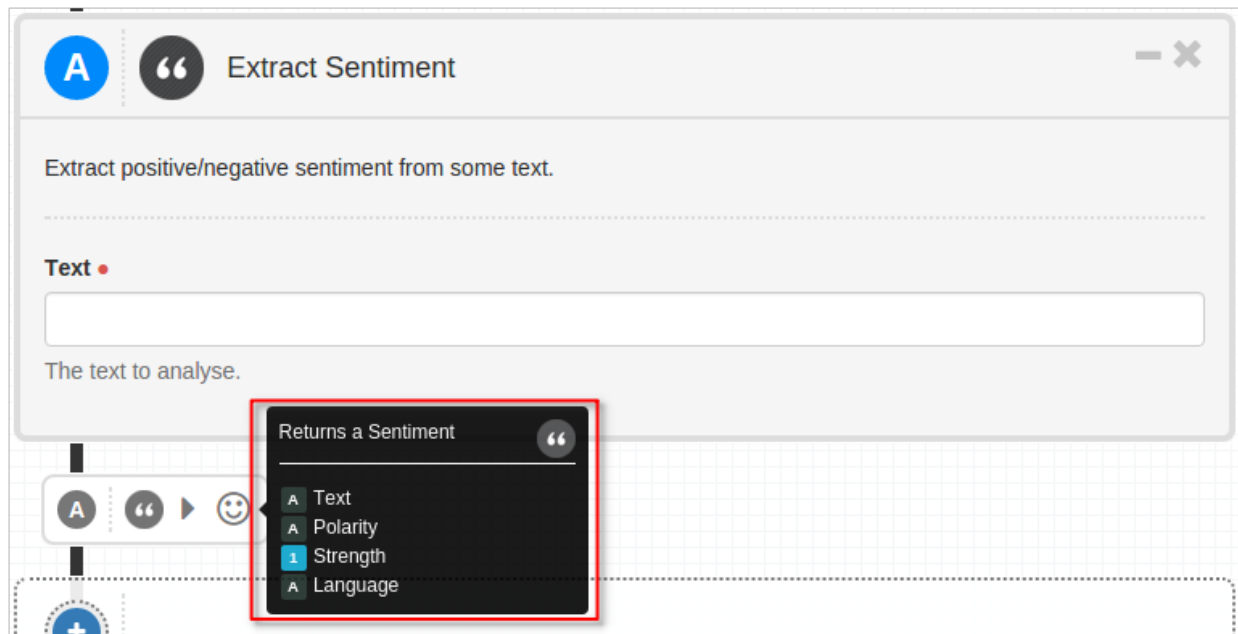
- The **Apply Regular Expression** pipe applies the regular expression given in *Regular Expression* to the text provided in the *Text* field. For a reference of the supported syntax please refer to [regex syntax](#). You can also use [Pythex](#) to test your regular expression.
- The **Find All Matches to a Regex** query pipe will apply a regular expression (specified in *Regular Expression*) on a text (*Text*) and return all non-overlapping matches of the regex. As with all the other *Search* pipes Cloudpipes offers you can *Order by* a specified field and set a *Limit* on the number of results you are interested in.

## Sentiment Analysis

**Extract Sentiment** is a pipe that performs natural language processing (NLP) on the supplied text and extracts a positive/negative sentiment.



As visible in the *output* of the **Extract Sentiment** pipe :

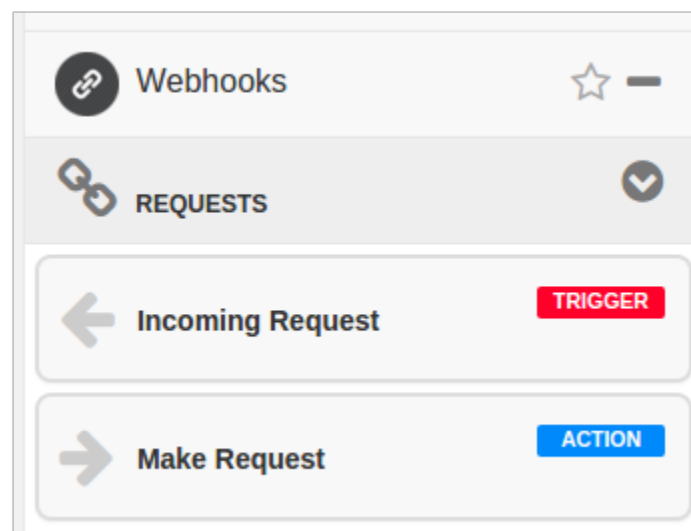


Among other fields you get the extracted *Strength* and *Polarity* of the given *Text*.

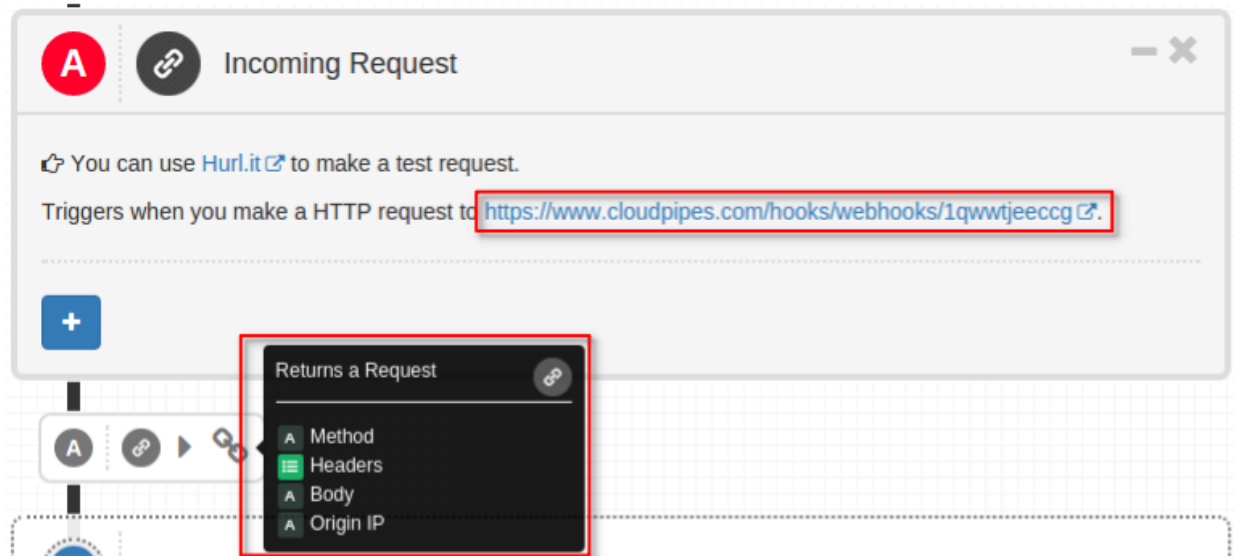
**Note:** We use [Alchemy](#) as the NLP engine for **Extract Sentiment**.

## Webhooks

A **webhook** is an HTTP callback - an HTTP POST that occurs when something happens. Webhooks are used as a simple event-notification mechanism via HTTP POST requests.



- **[Trigger] Incoming Request** will fire when an HTTP POST request is made to the provided address:



The *URL* to make requests to in order for the trigger to fire is shown in the pipe. From the *output* of **Incoming Request** you get *HTTP Method*, *Headers*, *Body* and *Origin IP* of the request made.

**Note:** You can use [Hurl.it](https://hurl.it) to make a test request and make sure all works as expected.

- [Action] **Make a Request** is the pipe to use when you want to make a webhook request :

A

Make Request

Makes a HTTP request.

You can use [RequestBin](#) to see what this pipe is sending or to inspect and debug webhook requests.

Authentication

Optional authentication scheme.

URL •

Method •

GET

Headers (list)

Name

Value

Content Type

Body

Less

Returns a Request

Status Code

Headers

Content

Elapsed

Json

Parameters you can configure include *Authentication Method*, *URL* to make the request to, *HTTP Method* to use, *Content-Type*, the *Body* of the request and an optional list of *Headers* that is available after unfolding *more*.

In the *Output* of **Make a Request** you get *Status Code* of the response to the request made, a list of *Headers*, time *Elapsed*, response *Content* as text and also a *json* object that will have accessible fields for the cases when the remote service returns a JSON object.

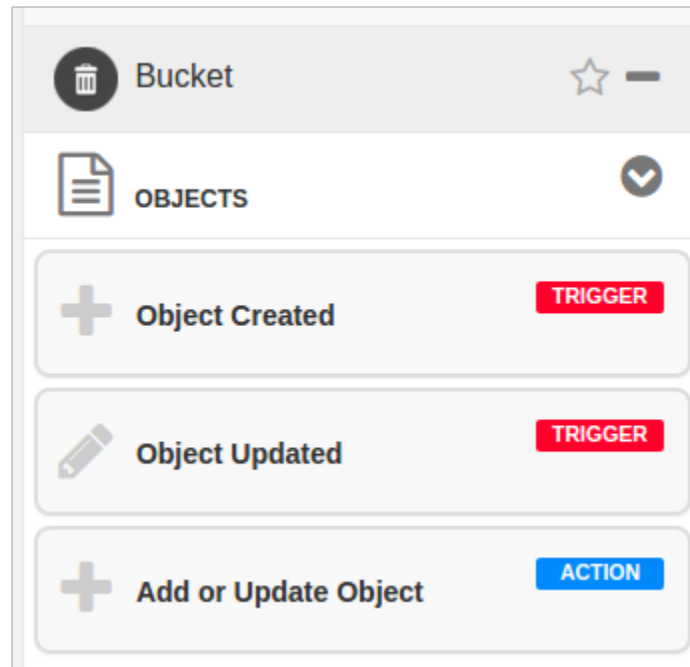
#### Hints:

- You can use [RequestBin](#) or [httpbin](#) to see what this pipe is sending or to inspect and debug webhook requests.
- You can use `to_json` filter to obtain json representation of any object, or part of it. for example doing `{{a|to_json}}`, where pipe A exports a Trello card will result in:

```
{
  "id": "57adaedfdf614dca6af0c77d",
  "description": "This is the description.\nC.",
  "members": [
    "5576b2fd9f60f708ad286470"
  ],
  "position": 2048,
  "short_id": "6",
  "attachments_count": 0,
  "short_url": "https://trello.com/c/g8GRSMOe",
  "closed": false,
  "name": "Example Name",
  "labels": [
    {
      "uses": 3,
      "id": "55b9e9ed19ad3a5dc2f3cf58",
      "color": "green",
      "name": null
    },
    {
      "uses": 3,
      "id": "55b9e9ed19ad3a5dc2f3cf5d",
      "color": "blue",
      "name": null
    }
  ],
  "updated_at": "2016-08-12T11:11:27.099000+00:00",
  "board": "55b9e9ed5ae0424917b5eec6",
  "list": "55b9e9ed5ae0424917b5eec7",
  "due_date": "2016-08-12T21:00:00+00:00"
}
```

## Bucket

The **Bucket** allows you to store arbitrary items and retrieve them later. You can have as many Buckets as you may need. Buckets can be named and also can hold an arbitrary number of items.



- **[Trigger]** Object Created - Will trigger when an item is added to the Bucket having the specified name.
- **[Trigger]** Object Updated - Will trigger when an item is updated in the Bucket having the specified name.
- **[Action]** Add or update object - Adds or updates the specified item (Resource) to the Bucket and assigns the specified **ID** to it for later retrieval.

## 1.11 Frequently Asked Questions

### 1.11.1 General

#### What's this all about?

**Cloudpipes** enables you to automate tasks between web application like Salesforce, Slack and Trello. Cloudpipes gives you the ability to execute commands in one system, when something happens in another. For example when a Lead gets updated in Salesforce post a new message to Slack, or when a Card is created in Trello add a new row to a Google Spreadsheet.

To get started check out our [Getting Started](#) guide.

#### Pipelines, Pipes, Channels....

To get up to speed with the terminology take a look at the [Glossary](#).

#### I just want to get my feet wet

Sure, see the [2 Minute Howto](#).

### What about the more advanced stuff?

Absolutely - you can use [Flow Control And Loops](#), implement [Item Linking](#), [Schedule execution](#) and save time by using [Blueprints](#)

### Any channel specific guides?

Yep, check out [Some common examples](#).

### I want to see it running

When you want to execute your pipeline, just click on the Play button in [The Editor](#). It will present you a Progress window, where you can see the execution of each step in the pipeline. You will also be able to see the actual field values when executing the relevant pipe.

### What if I'm not around to click the Play button?

There are several way to [trigger](#) execution of a pipeline. The simplest one, is interactively by using the Play button by the pipeline. Once you have such pipeline and you are happy with its operation, you can schedule its execution periodically on a certain interval or time.

Other pipelines can begin with a trigger pipe. These types of pipes react on events from the external service and trigger the pipeline when such even occurs. Imagine update of an issue in a bug tracking system or receiving an email or a Twitter message. You can enable and disable the trigger based pipelines in order to control if they react on trigger events or just ignore them.

### How often do my pipelines run?

Checking for changes depends on what the specific channel API offers us. When possible we use webhooks, so changes propagate immediately to our system. For channels that do not support this, Cloudpipes will poll every ~5 minutes for new/changed data.

### What if my pipelines fail to run, how do i get notified?

You will get notified by email when a certain threshold in terms of number of failures is reached. We have it like that so intermittent connectivity problems do not case a flurry of messages.

### How can I see what my pipelines are doing?

Your [Activity stream](#) is where each pipe run is recorded with a timestamp and further details.

### Typing Into Dropdowns

In many of your pipes you have the option of inserting a custom value in dropdowns that have populated values to choose from. Many times, these custom values are the causes of errors, so try using the populated values instead if you do get errors when using the custom values.

### What do the field colors mean?

In the fields list you will see that various fields have different colors. For a full explanation refer to [Field Colors Guide](#).

### 1.11.2 Billing and pricing

#### How much does it cost?

Not that much. For details see our [Pricing and plans](#) page.

#### What are transactions?

A **transaction** is a successful execution of an action. For instance, if your pipeline creates new users in Intercom when there are new accounts in Salesforce, each action (user created) in that pipeline would count as a single transaction.

#### What is being charged?

- Each pipe executed successfully is 1 transaction even if filters on a *trigger pipe* do not match.
- For query pipes one execution is 1 credit, regardless of whether the query matches any items.
- Fetch a [linked resource](#) is 1 transaction.

#### What are Standard, Tier-1 and Tier-2 channels?

On our [channels](#) page you can see a list of all the *channels* Cloudpipes supports and also which tier the respective channel is assigned to. *Channels* on Cloudpipes belong to either the **Standard**, **Tier-1** or **Tier-2** groups. Using channels from a certain group requires you to be on one of our paid plans. Click [here](#) for a list of channel per tier.

#### What does “Multiple Accounts per Channel” mean?

On our **Personal** plan you will only be able to connect one *account* per *application*. On our higher plans - Startup, Business and so on, you will be able to connect multiple accounts for the same App. You need multiple accounts per channel if you want to interact with more than one system of that kind. e.g. if you have two JIRA instances which are completely separate and you need to create tickets in both, then you need to connect two accounts to the JIRA channel and select the respective one in each JIRA pipe.

#### What does “Cloudpipes branding” mean?

On the Startup plan Cloudpipes may add branding text to selected fields in you pipelines. For example way may add “*Powered by Cloudpipes*” to a tweet you post via Cloudpipes.

#### What does “Static outgoing IP” mean?

If you are on our Enterprise plan we will dedicate an IP to your pipelines and all the requests Cloudpipes makes will be coming from that IP. For systems that are hosted on-premises this will guarantee you can allow access to the system for that specific IP.



## What is the difference between Standard and Premium support?

The difference between Standard and Premium support is the response time and resolution period for any issues that you may encounter.

With Premium support we aim to respond within 12 hours (in most cases immediately, depending on load) and resolve the issue within 1-3 working days (in most cases, depending on severity), and we also offer consultation and assistance with setting up your integrations.

With Standard support the response time is not guaranteed (but no longer than 48 hours), and resolution is at our discretion. We don't usually offer consultation to customers with Standard support.

### 1.11.3 Still having questions?

Should you need any assistance, as usual just drop us a line at [support@cloudpipes.com](mailto:support@cloudpipes.com) or reach us in the in-app chat.

Happy integrating with **Cloudpipes**!

## 1.12 List of Tier-1 and Tier-2 channels

*Channels* on Cloudpipes belong to either the Standard, Tier-1 or Tier-2 groups. Using channels from a certain group requires you to be on one of our paid plans. Here's a list of channels that are in each group.

### 1.12.1 Tier-1 Channels

Using channels from the **Tier-1** group requires you to be on our [Business Plan](#).

- Act-On
- Clearbit
- DocuSign
- DueDil
- EchoSign
- Eloqua
- Factual
- HubSpot
- KISSmetrics
- Magento
- Marketo
- Microsoft Dynamics CRM
- Microsoft Exchange
- Microsoft SharePoint
- Pardot
- QuickBase
- QuickBooks Online

- Salesforce
- Skype for Business

### 1.12.2 Tier-2 Channels

Using channels from the **Tier-2** group requires you to be on our [Startup](#) or higher plan.

- Active Collab
- Amazon Payments
- Ambassador
- AWeber
- Bamboo
- Basecamp
- Box
- BugHerd
- Campaign Monitor
- Chargify
- Chartbeat
- Chatter
- CircleCI
- Confluence
- Constant Contact
- Crucible
- DataSift
- Desk.com
- Diffbot
- Eventbrite
- Evernote Business
- FastSpring
- FogBugz
- FreeAgent
- FreshBooks
- Freshdesk
- FullStory
- Geckoboard
- GetResponse
- Google Sheets
- Google Translate

- GoSquared
- GoToWebinar
- Harvest
- HootSuite
- Infusionsoft
- Insightly
- Intercom
- JIRA
- Kiln
- Leftronic
- MailChimp
- Microsoft Translator
- Moz
- OCR
- PayPal
- Phone
- Pipedrive
- Podio
- Recurly
- Semantria
- Shopify
- Smartsheet
- SMS
- Stash
- Stripe
- SugarCRM
- Teamwork.com
- TestRail
- Text
- TFS
- Unbounce
- UserVoice
- Xero
- YouTrack
- Zendesk
- Zoho CRM

### 1.12.3 Standard Channels

All channels not in one of the groups above is a **Standard** channel and can be used even on our **Free** plan.

---

## Indices and tables

---

- `genindex`
- `modindex`
- `search`