Cisco Spark API アドバンスド ラボ v1

最終更新日：2018 年 3 月 13 日

コラボレーション VT メンバーとのコラボレーションによる作成。

このラボについて

この事前設定済みラボの内容は次のとおりです。

要件

次の表に、この事前設定済みのラボ演習の要件を示します。

表 1. 要件

必須	オプション
• Cisco AnyConnect がインストールされているラップトップ	• なし

このシスコ ソリューションについて

Cisco Spark の概要

Cisco Spark は、完結したコラボレーション スイートを提供するアプリケーション セントリックなクラウドベースのサービスです。チームは同じ場所にいても、離れていても、作成、会議、メッセージング、コール、ホワイトボードの利用、共有を行うことができます。これにより、会議前、会議中、会議後まで統合された継続的なワーク ストリームが実現します。Cisco Spark は、チームのシームレスな連携を実現するために作られました。仕事の質を高める、シンプル、セキュア、かつオープンな、完結したサービスです。

コミュニケーションがあるべき姿は、アジャイル、モバイル、コラボレーティブ。これらすべては、モバイル デバイスや、インフラストラクチャとアプリケーションにおけるイノベーションの進化のおかげです。Cisco Spark サービスは、業界をリードするコミュニケーション ツールを緊密に統合して、瞬時のコミュニケーションとリアルタイムの会議を可能にし、これまでにないコラボレーション体験を実現します。こうしたメリットを提供できるのは、Cisco Cloud Collaboration だけです。
Cisco Spark にはオープン API が用意されており、開発者が Spark ボットやインテグレーションを作成することができます。そのため、既存のアジャイル プラットフォームの効率性をさらに向上させ、価値を高めることができます。このラボの完了後は、https://developer.ciscospark.com/ [英語] でさらに詳細に学習することができます。

トポロジ

このコンテンツには、ソリューションの機能を実例で示すために事前設定されたユーザとコンポーネントが含まれています。コンポーネントのほとんどは、事前定義の管理ユーザ アカウントを使用して任意の設定が可能です。コンポーネントへのアクセスに使用する IP アドレスとユーザ アカウントのクレデンシャルは、アクティブ セッションの [トポロジ(Topology)] メニューにあるコンポーネント アイコンをクリックして確認するか、その情報が必要なシナリオ内の手順で確認できます。

図 1. dCloud のトポロジ

表 2. サーバの詳細

名前	説明	ホスト名(FQDN)	IP アドレス	ユーザ名	パスワード	RDP
AD1	Active Directory、DNS、ADFS3.0	ad1.dcloud.cisco.com	198.18.133.1	dcloud\administrator	C1sco12345	○
WKST1	Windows 7	wkst1.dcloud.cisco.com	198.18.133.36	dcloud\cholland	dCloud12345!	○

はじめに

プレゼンテーションの前に

Cisco dCloud では、実際の対象者の前でプレゼンテーションを行う前に、アクティブなセッションを使用して、このドキュメントのタスクを実施しておくことを強く推奨します。そうすることで、ドキュメントとコンテンツの構成に慣れることができます。

場合によっては、環境を元の構成にリセットするため、このガイドに従った後に新しいセッションをスケジュールする必要があります。

プレゼンテーションを成功させるためには、入念な準備が不可欠です。

次の手順に従ってコンテンツのセッションをスケジュールし、プレゼンテーション環境を設定します。

dCloud セッションを開始します。[手順を見る]

注: セッションがアクティブになるまで、最長で **10 分**かかることがあります。

最適なパフォーマンスを得るために、**Cisco AnyConnect VPN** [手順を見る] および**ラップトップのローカル RDP クライアント**[手順を見る] を使用してワークステーションに接続します。

- Workstation 1: **198.18.133.36**、ユーザ名: **DCLOUD\cholland**、パスワード: **dCloud12345!**

シナリオ 1.　Cisco Spark オウム ボット

この例では、Cisco Spark ボットの作成方法をデモンストレーションします。このボットは、ユーザがボットに呼びかけると、同じテキストを返します。つまり、メッセージがボットに送信されると、アプリケーションがそのメッセージからテキストを抽出し、同じメッセージをユーザのスペースに返すということです。

このラボでは、ドラッグアンドドロップ方式のビジュアル アプリケーション開発ツールである、Node-Red を使用してゼロからアプリケーションを開発します。このツールを使用すれば、プログラミングの経験がない人でもラボを完了できます。

ボットの開発には 2 つのパートがあります。最初のパートでは、Spark for Developers で提供されるマシン アカウントを利用して、Cisco Spark に「ユーザ」アカウントを作成します。2 つ目のパートで、このアカウントでの頭脳にあたるプログラムを開発します。

手順

1. まだ、Cisco Spark アカウントを持っていない場合は、Cisco Spark にサインアップします。サインアップの方法については、こちらの動画で手順を確認できます。

基本的な Cisco Spark ボットを作成する

このセクションの目的は、ロジックのないボットを作成し、ボット ツールの配置に慣れることです。

1. この動画で、Cisco Spark for Developers Web サイトにボットを作成する方法を確認できます。
2. ボットを作成します。[ボット名（Bot Name）] は任意の名前、[ボット電子メールアドレス（Bot Email）] には **training_username_bot@sparkbot.io** を指定します。すでに使われている場合は名前に番号を付けるなどして一意になるようにします。

注：ボット詳細ページから移動する前に**認証トークンをテキスト ファイルにコピーする**のを**忘れない**ようにしてください。

ボットに Hi と呼びかける

1. Cisco Spark アプリケーションを開き（自分のアカウントでサインイン）、ボットと 1 対 1 のメッセージングを開始します。

当然ですが、ボットは（まだ）スマートではありません。ロジックを組み込んでいないからです。そこで、ボットにいくつかロジックを追加します。

ngrok を始める

ngrok とは何でしょうか。

ngrok は、一般のエンドポイント（インターネットなど）からローカルで実行中のネットワーク サービス（自社のアプリケーション）にセキュアなトンネルを確立する、トンネリング用のリバース プロキシ ソフトウェアです。

ngrok は以下の手順で開始します。

1. Workstation 1 のデスクトップに ngrok アプリケーションへのショートカットがあることを確認します。このショートカットをダブルクリックして、コマンド プロンプトを起動します。
2. **ngrok http 1880** と入力します。

3. *https* を使用した*転送* URL をコピーします（例：https://8296ecae.ngrok.io）。今後この URL を **targetUrl** として参照します。今後このセクションで使用します。

注：ngrok コマンド プロンプトを閉じないようにしてください。

ngrok が機能していることを確認します。

4. ブラウザ（Google Chrome）を開き、targetUrl を貼り付けます。ngrok コマンド プロンプトに切り替え、「GET / 200 OK」メッセージが表示されていることを確認します。これは、targetUrl に送信されたデータを ngrok が取得し、アプリケーションに接続しようとしていることを示しています。

ウェブフックの機能

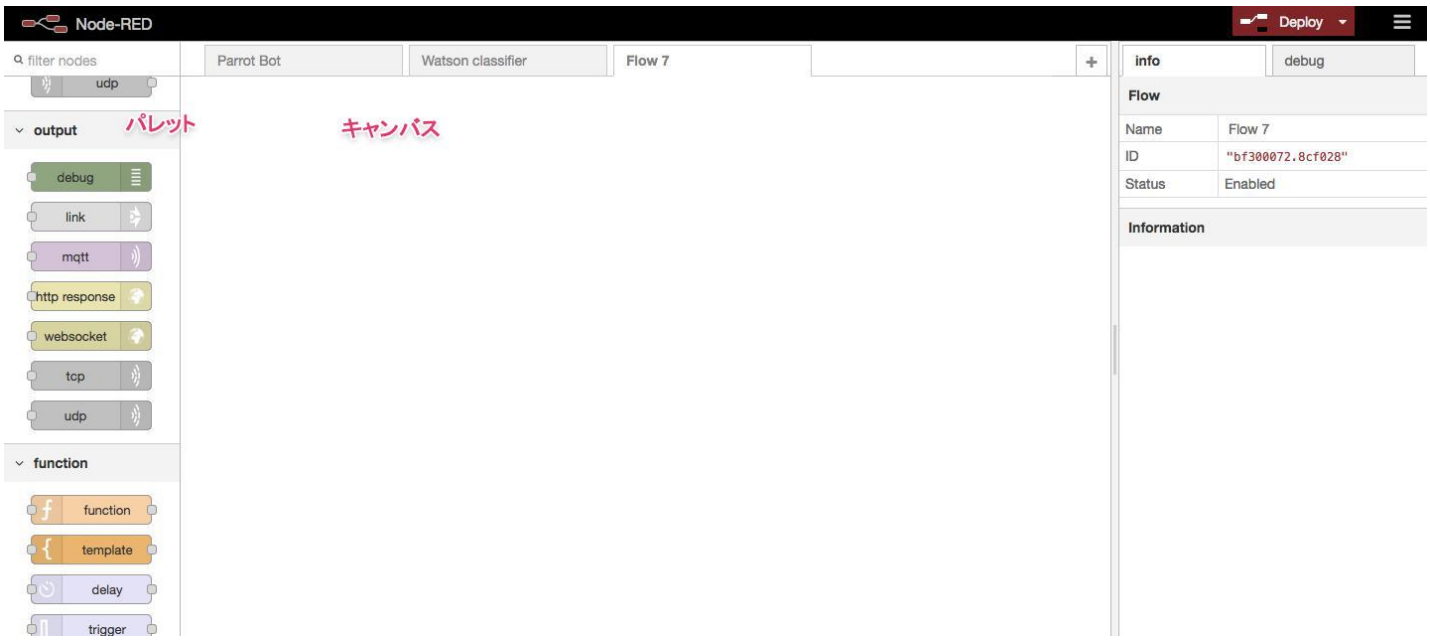
ウェブフックは、Cisco Spark で特定のアクティビティが発生した時の状態の変化をアプリケーションに通知します。指定された URL に対する HTTP コールバックまたは HTTP POST で、Cisco Spark プラットフォーム上のリソースのいずれかにおいて特定のアクティビティまたは「イベント」が発生したときにアプリケーションに通知します。これで targetUrl の必要性がわかったと思います。Spark は、ボット用のウェブフックを作成する API を提供しています。このラボでは Node-RED を使用していますので、Node-RED を利用して、ボット用のウェブフックをプログラミングします。

Node-RED

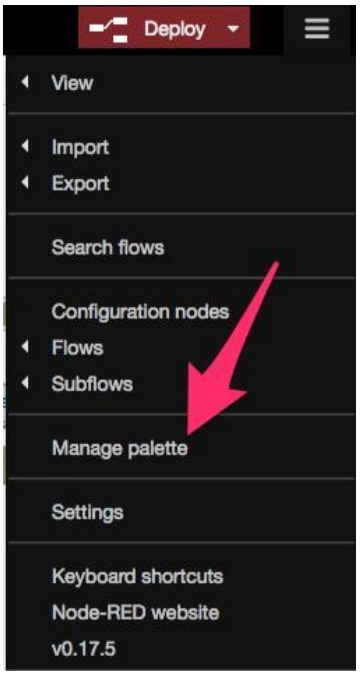
Node-RED は、API とオンライン サービスを、新たな興味深い方法で連携させるプログラミング ツールです。ブラウザベースのエディタで、パレット内のさまざまなノードを使用してフローを結びつけ、シングル クリックでランタイムに展開することが簡単にできます。

1. Node-RED を起動します（まだ開いていない場合）。ブラウザの新しいタブで http://localhost:1880 と入力すれば起動できます。

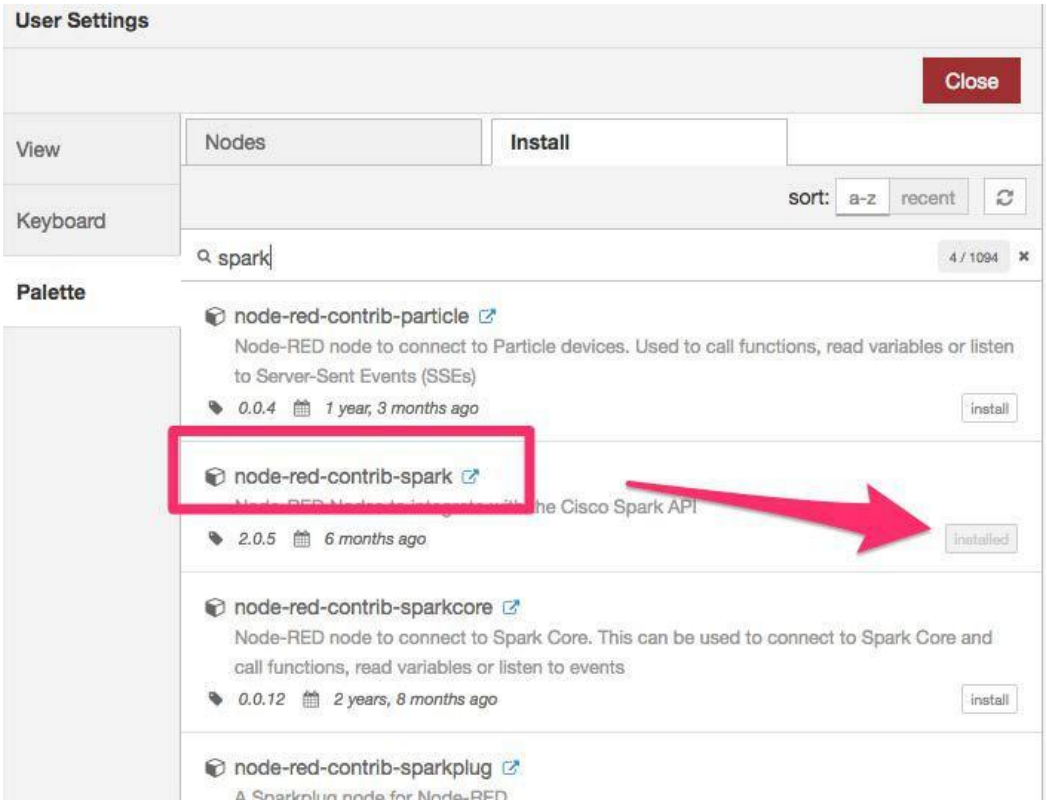
2. Node-RED インターフェイスは、3 つのパートで構成されています。左側は、ノードが表示されるパレットで、ここからノードを中央にドラッグすることができます。キャンバスは、ボットを作成する部分です。右側には、現在選択しているノードの情報や、フロー実行時のデバッグ情報が表示されます。



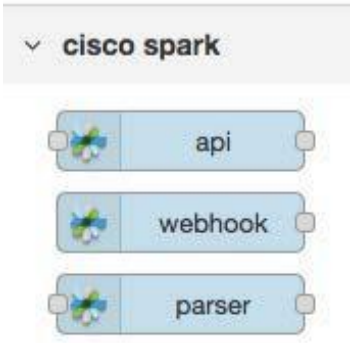
3. Cisco Spark ノードを Node-RED のパレットに追加します。画面右隅の [導入（Deploy）] メニューのメニュー アイコン [ ] をクリックし、[パレットの管理（Manage palette）] をクリックします。



4. [インストール（Install）] タブを選択し、**Spark** で検索して、Cisco Spark モジュール **node-red-contrib-spark** を見つけます。[インストール（Install）] を 2 回クリックして追加します。インストールが完了するまで数分かかります。

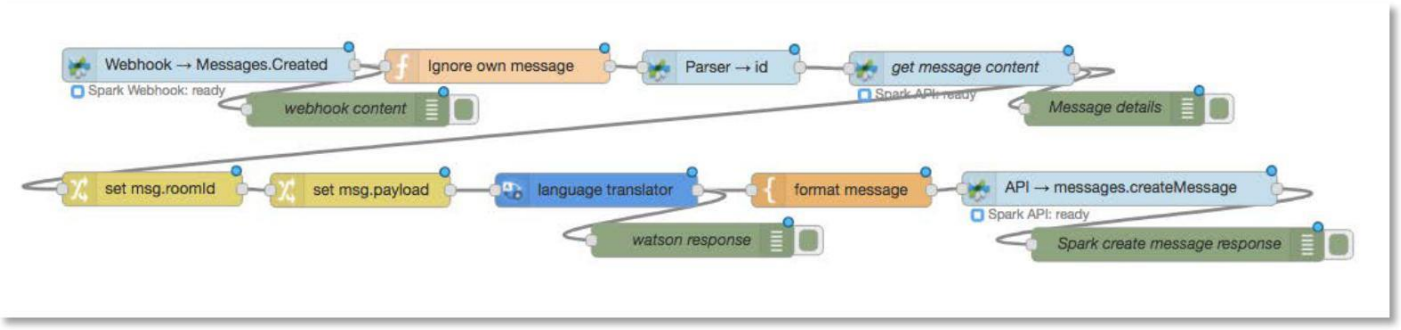


5. インストールが完了したら、Node-RED の左側のノード パレットを下にスクロールして、**Cisco Spark** ノードを確認します。



api ノードは、Spark API に対するアクションを実行します。すべての API が 1 つのノードに含まれています。アクションを選択するには、ノードを設定し、必要なタイプを選択します。
webhook ノードは、アプリケーションへのエントリ ポイントを示します。このノードによって Cisco Spark へのウェブフック サブスクリプションが設定され、そのウェブフックがイベントを送信するルートがセットアップされます。
parser ノードは、Spark API からの結果を、Node-RED で使用できる変数に変換するために使用されます。

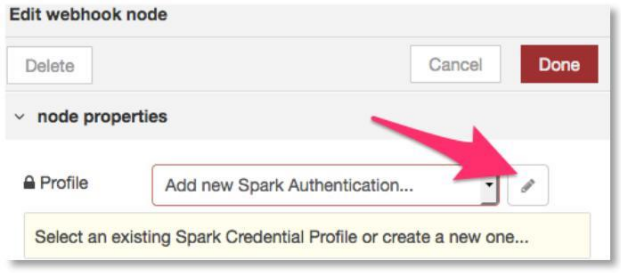
設定が終了した時点で、変換アプリケーションは以下のようになります。



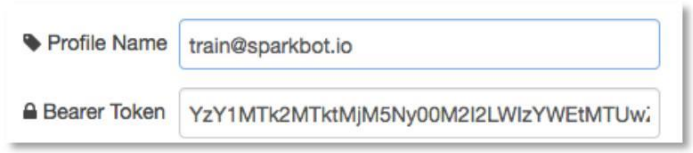
緑のノードはデバッグ ノードで、省略可能です。フローで各ステップを実行している時に、フロー内のデータを確認するために使用します。フローが正しく動作しない場合は、デバッグ ノードをいくつか追加すれば、アプリケーション実行中に何が発生しているかを確認することができます。

アプリケーションの構築

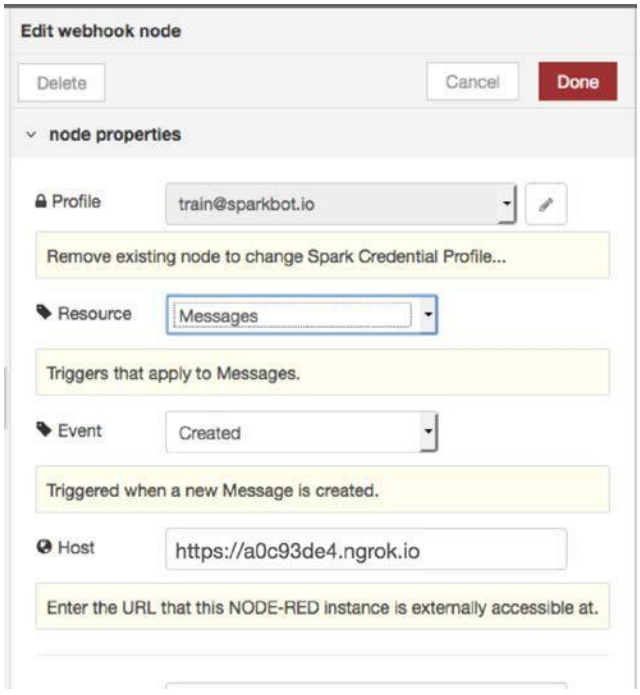
1. Cisco Spark **webhook** ノードをキャンバスにドラッグして、フローに追加します。
2. ノードをダブルクリックして設定を編集します。[プロファイル（Profile）] ドロップ ダウンから [新しいSpark認証を追加（Add new Spark Authorization）] を選択します。
3. [編集（Edit）] アイコンをクリックします。



4. [プロファイル名（Profile Name）] に先ほど作成したボットのアドレスを入力します。アドレスは、*<アドレス>@sparkbot.io* という形式になります。[ベアラートークン（Bearer Token）] にボットの認証トークンを入力し、[追加（Add）] をクリックします。



5. [完了（Done）] をクリックします。
6. Webhook ノードを設定します。[リソース（Resource）] に [メッセージ（Messages）] を選択します。[イベント（Event）] には、[作成時（Created）] を選択します。[ホスト（Host）] フィールドに Node-RED アプリケーションの URL を入力します。最後のスラッシュは除き、https://<文字列>.ngrok.io のように指定します。



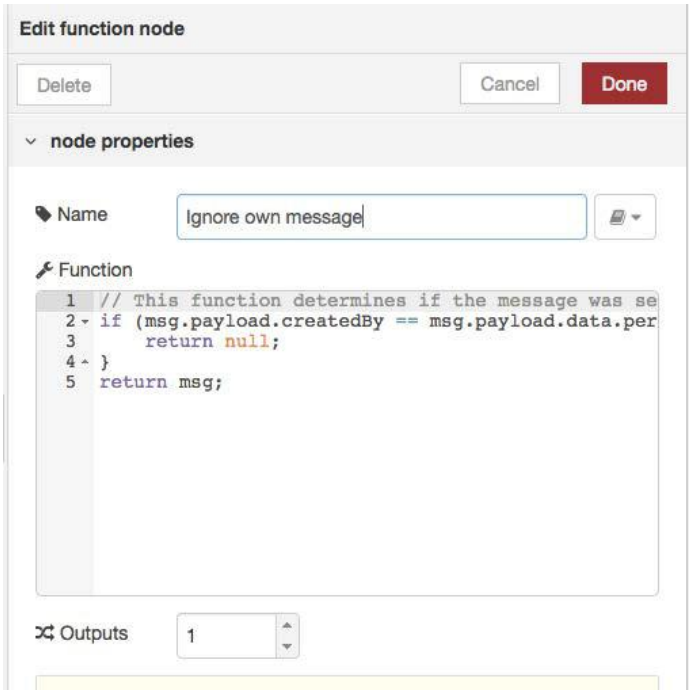
7. 次の手順では、Spark ボットが自分自身のメッセージに返信しないようにする関数を追加します。この関数は、ウェブフックを作成したアカウントと同じアカウントから送信されたメッセージかどうかを確認し、同じ場合はフローを終了します。パレットの [関数(Functions)] カテゴリから [関数(function)] ノードをキャンバスにドラッグします。

注:このラボ ガイドをダウンロードした dCloud カタログの同じ所から **Spark for Developers.txt** ファイルをダウンロードし、任意のコードスニペットをコピーして貼り付けてください。

8. Spark Webhook から関数ノードに接続をドラッグし、関数ノードの設定を編集します。関数のテキスト エディタで、次のように入力します。

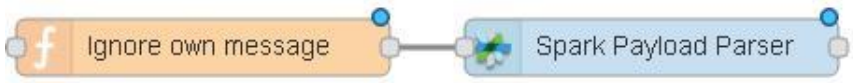
```
// was message was sent by someone other than the Spark bot
if (msg.payload.createdBy == msg.payload.data.personId) {
    return null;
}
return msg;
```

9. ウィンドウが以下の画像のようになっているはずです。

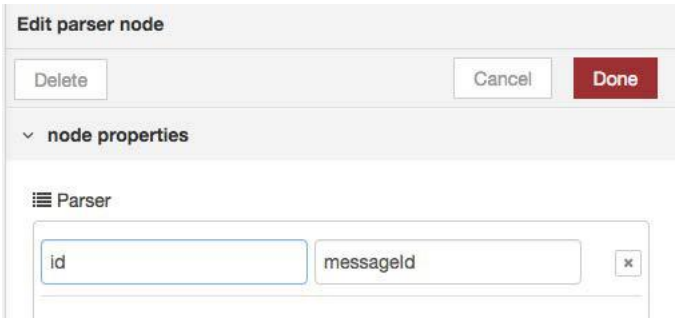


このコードは、「ウェブフックの作成者が、メッセージを送信したユーザと同じ Spark personID を持っている場合、フローをキャンセルする。それ以外の場合は、メッセージ オブジェクト変更せずに返し、何も発生しなかったかのようにフローを続行する」ということを示しています。

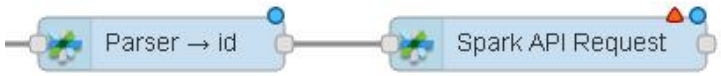
10. [完了（Done）] をクリックします。
11. Cisco Spark Parser ノードをフローにドラッグし、今追加した関数ノードに接続します。



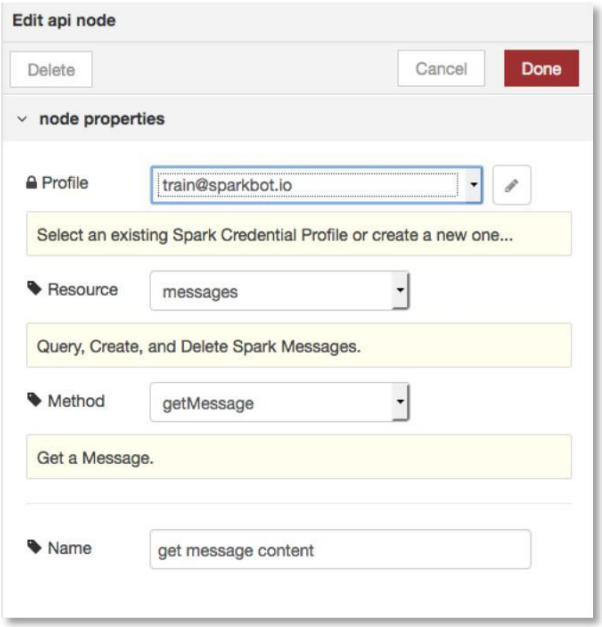
12. パーサー ノードを編集し、[プロパティ（Property）] フィールドに **id** と入力し、[名前（Name）] フィールドには、**messageId** と入力して、[完了（Done）] をクリックします。



13. Cisco Spark api ノードをキャンバスにドラッグし、作成したパーサー ノードに接続します。



14. api ノードをクリックして編集し、[リソース（Resource）] に [メッセージ（Messages）] を選択します。[メソッド（Method）] には [getMessage] を選択します。似たようなメソッドが 2 つあるので注意してください。1 つは複数形の *getMessages* で、もう 1 つは単数形の *getMessage* です。単数形の getMessage を選択していることを確認します。

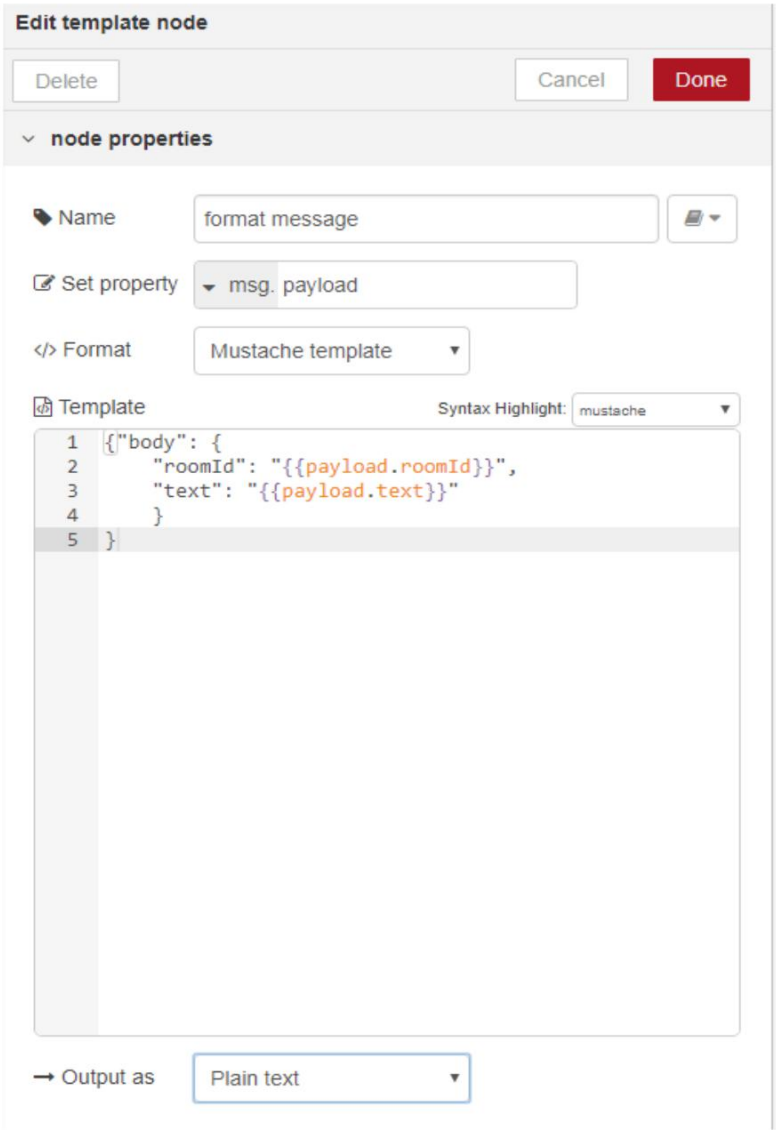


次のいくつかの手順では、ユーザにデータを送り返すために、フロー内のデータのフォーマットを再設定します。Node-RED は、*msg* というオブジェクトを使用します。各ノードがこのオブジェクトに追加したり、アプリケーションが実行される際にはこのオブジェクトから読み取ったりします。

15. アプリケーションは、Spark が判読できる形式で API コールにメッセージをフォーマットする必要があります。[関数（Function）] パレットから [テンプレート（Template）] ノードをドラッグしてフローに追加し、作成したトランスレータ ノードに接続します。
16. テンプレート ノードを編集し、[セットプロパティ（Set property）] フィールドを [msg.payload] に変更し、[テンプレート（Template）] エディタに以下のように入力します。

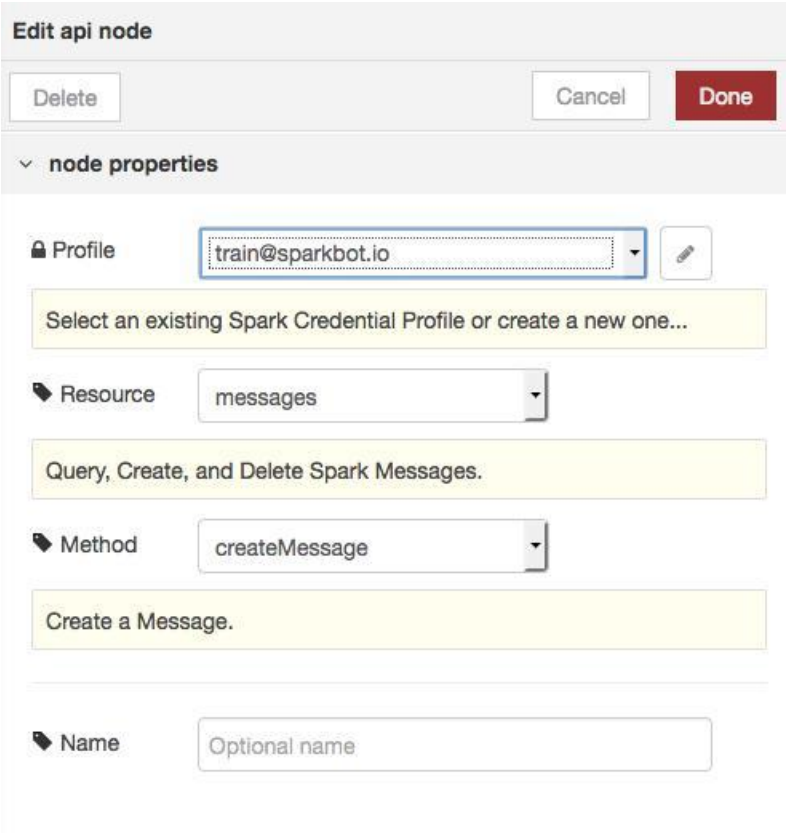
```
{"body": {
    "roomId": "{{payload.roomId}}",
    "text": "{{payload.text}}"
    }
}
```

17. 画面が以下の画像のようになっているはずです。[完了（Done）] をクリックして終了します。

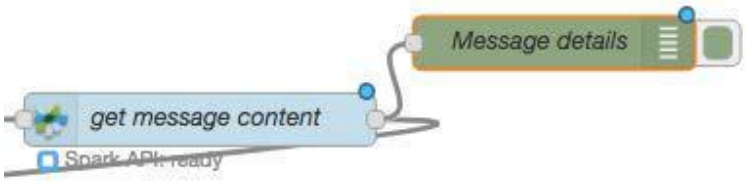


18. Cisco Spark api ノードをフローにドラッグして、今作成したテンプレート ノードに接続します。

19. API ノードを [編集(Edit)] して、[リソース(Resource)] に [メッセージ(messages)] を選択します。[メソッド(Method)] には [createMessage] を選択します。[完了(Done)] をクリックします。

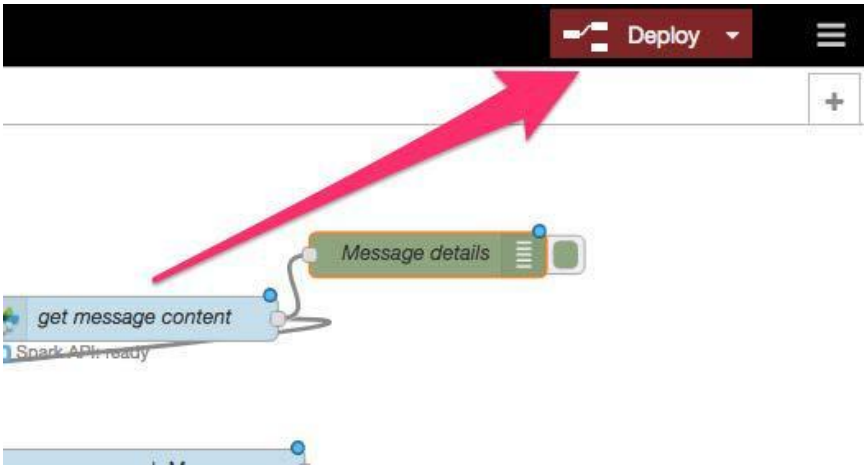


これでアプリケーションの作成が完了しました。必要に応じて、デバッグ ノードをフローにドラッグし、アプリケーションの各ノードの出力に接続します。



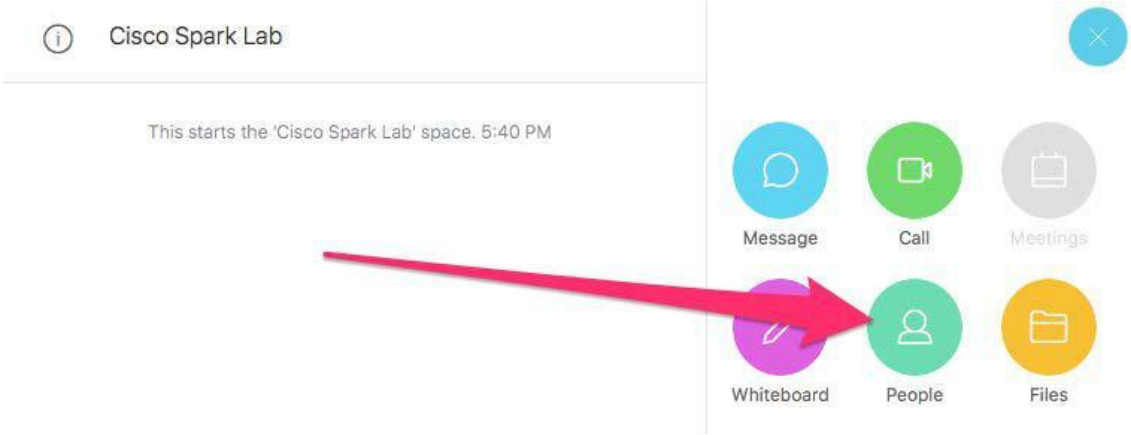
デバッグ ノードを接続することで、アプリケーションの状態がデバッグ パネルに出力されるため、アプリケーションで発生している内容を確認するのに役立ちます。

20. [導入（Deploy）] ボタンをクリックしてアプリケーションを導入します。

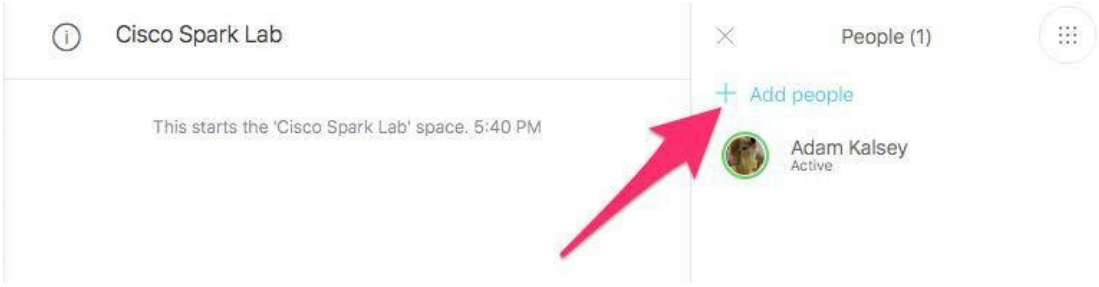


21. [アクティビティサークル（Activity Circle）] アイコンをクリックして、Cisco Spark Space にボットを追加します。

22. 次に、[ユーザ（People）] アクティビティをクリックします。



23. [ユーザの追加（Add People）] をクリックします。



24. ボットのアドレスを入力します。

25. @mention を利用してボットにメッセージを送信します。ボットが同じメッセージを返してくるのを確認します。

注: ボットと直接通信する場合は @mention は必要ありません。

26. ボットに送信したメッセージと同じ内容がそのまま返されるのを確認します。このラボでは、送信したメッセージに応答する、シンプルなボットを構築するために必要なインフラストラクチャのデモンストレーションを行います。次のラボでは、このボットを変更し、ユーザがボットに送信する短いメッセージを理解する会話型ボットを作成します。

シナリオ 2.　Cisco Spark 会話型ボット

この例では、メッセージの意図を理解し、適切な応答を見つけ出す Cisco Spark ボットを作成する方法を示します。ボットにメッセージが送信されると、アプリケーションは、メッセージからテキストを抽出し、IBM Watson Conversation にそのメッセージを送信して意図を判断します。意図に合った応答を取得したら、送信者のスペースに応答メッセージを返します。

このラボでは、オウム ボット（ユーザにメッセージをオウム返しするボット）を拡張して会話型ボットにします。そのため、前の演習で作成したコンポーネントのいくつかを利用します。Node-RED を使用すれば、プログラミングの経験がない人でもラボを完了できます。

手順

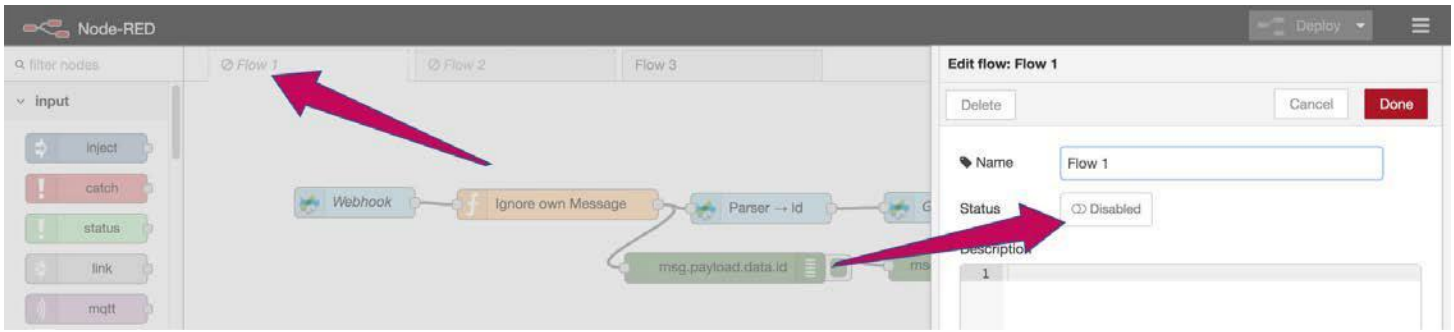
次の手順では、Box インテグレーションを、シナリオ 1：Cisco Spark オウム ボットで作成したスペースに追加し、ファイルを追加するか、Box ノートを作成してテストします。

開始する前に、オウム ボット演習を完了し、完全に機能していることを確認します。ブラウザから http://localhost:1880/ にアクセスして Node-RED を開き、フローの 1 つとしてオウム ボット フローが開かれていることを確認します。

前の Node-RED フローを無効にする

オウム ボットは、Spark ボットが機能していることを確認するのに適しています。そのため、前のフローに変更を加えるのではなく、この演習用に新しいフローを作成します。ただし、この演習にも同じボット トークンを再利用するため、前のフローを無効にしないと、両方のフローがメッセージに応答して、1 つのメッセージに複数の応答が返ってしまいます。そのため、先に進める前に、オウム ボット フローを無効にする必要があります。

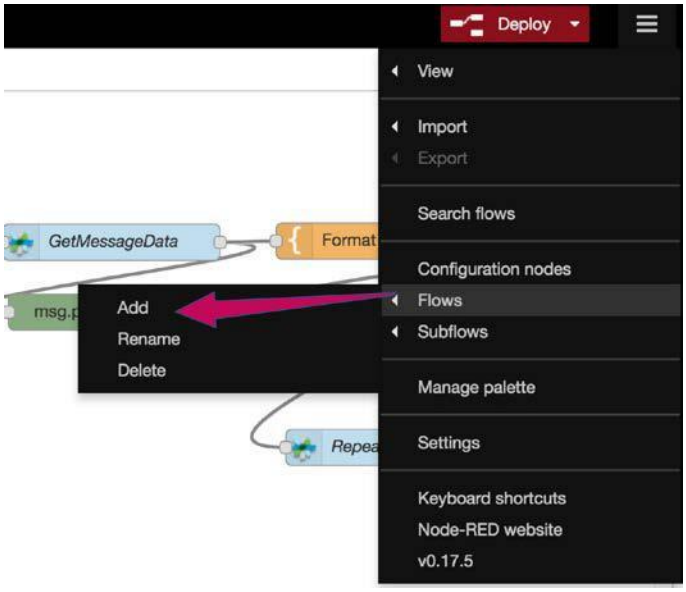
1. [フロー（Flow）] タブをダブル クリックします。[フロー編集（Edit Flow）] ダイアログがポップアップし、フローを [無効化（Disable）] するオプションを確認できます。[ステータス（status）] フィールドを切り替えて、オウム ボット フローを無効にします。[完了（Done）] をクリックします。



新しい Node-RED フローを作成する

この演習用の新しいフローを作成します。

1. 画面の右上隅のメニューを開き、[フロー（Flows）] -> [追加（Add）] の順にクリックして新しいフローを追加します。



フローから別のフローに内容をコピーする

オウム ボットと新しい会話型ボットの共通点を確認しましょう。この演習ではさらに以下を実施する必要があります。

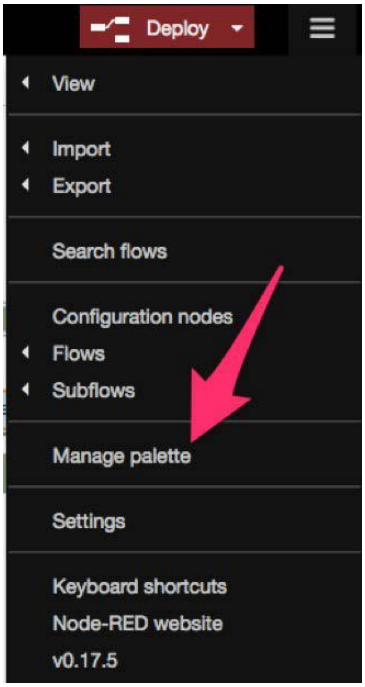
- ウェブフックを作成する
- ボット自体から送信されたメッセージを無視する
- ウェブフックを解析する
- Spark からの実際のメッセージを取得する

この時点で、会話型ボットは、Spark Space に同じメッセージを返す代わりに、IBM Watson Conversation にメッセージを送信する必要があります。フローを始めるために、オウム ボットから上記の手順をコピーして、新しいフローに貼り付けます。

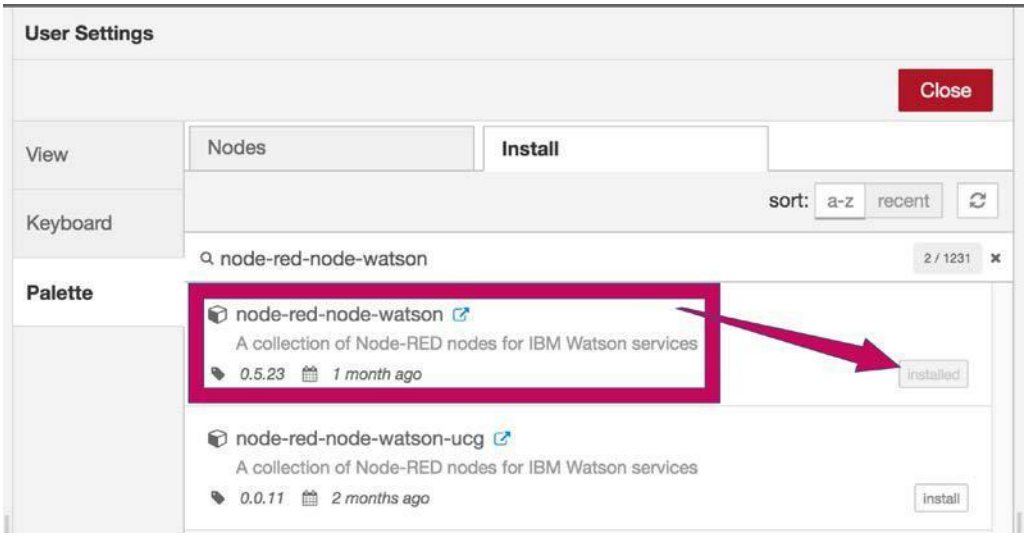
1. オウム ボット フローのブロックを [選択（Select）] してコピーし（**Ctrl + C**）、新しいフローに貼り付けます（**Ctrl + V**）。

Node-RED パレットに Watson を追加する

1. 画面の右上隅のメニューを開き、[パレットの管理（Manage palette）] をクリックして、Cisco Spark ノードを Node-RED に追加します。

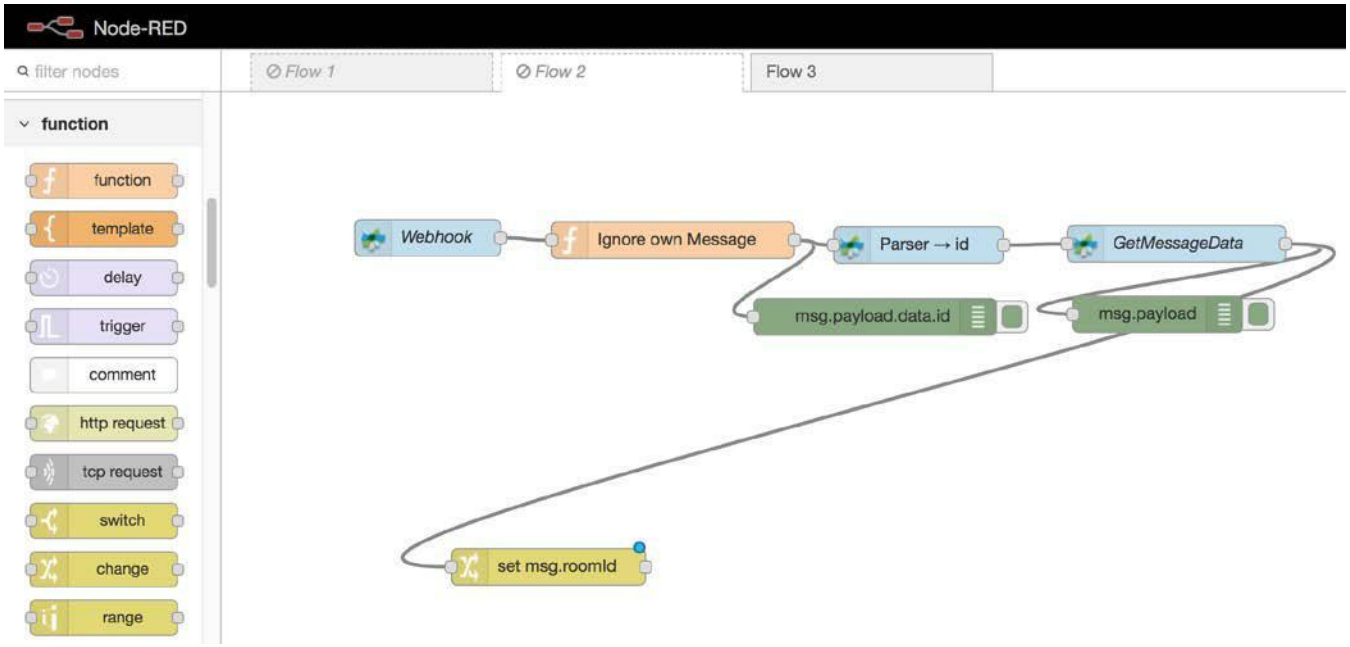


2. [インストール（Install）] タブを選択して、IBM Watson を見つけます。IBM Watson モジュール **node-red-node-watson** を検索します。見つけたらクリックして追加します。インストールが完了するまで数分かかります。

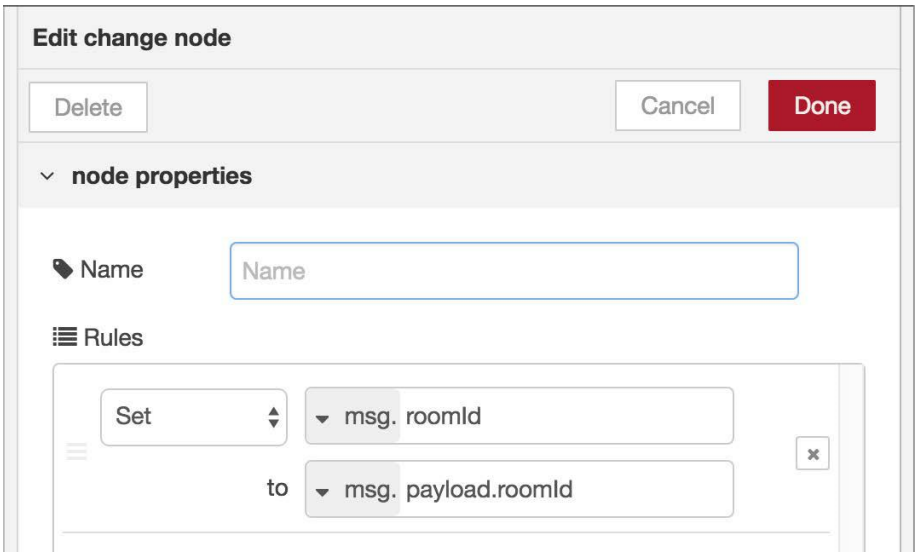


アプリケーションの作成を続ける

1. 次の手順では、後で参照できるように、別の変数に roomId を保存する関数を追加します。メッセージ プロパティを設定/移動するには、[変更（change）] ノードを使用する必要があります。
2. パレットの [関数（Functions）] カテゴリから [変更（change）] ノードをキャンバスにドラッグします。

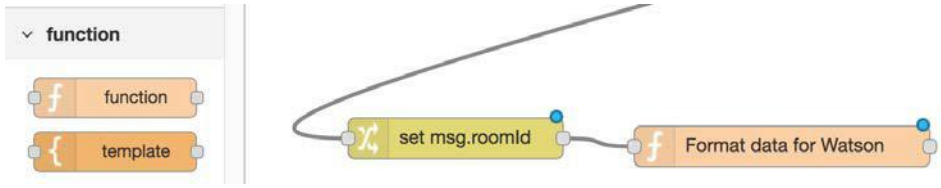


3. ノードを [編集（Edit）] し、[プロパティ（Property）] で、[設定（Set）]、[msg.roomId] to [msg.payload.roomId] と指定します。[完了（Done）] をクリックします。



次に、IBM Watson Conversation に合う形式にメッセージをフォーマットします。

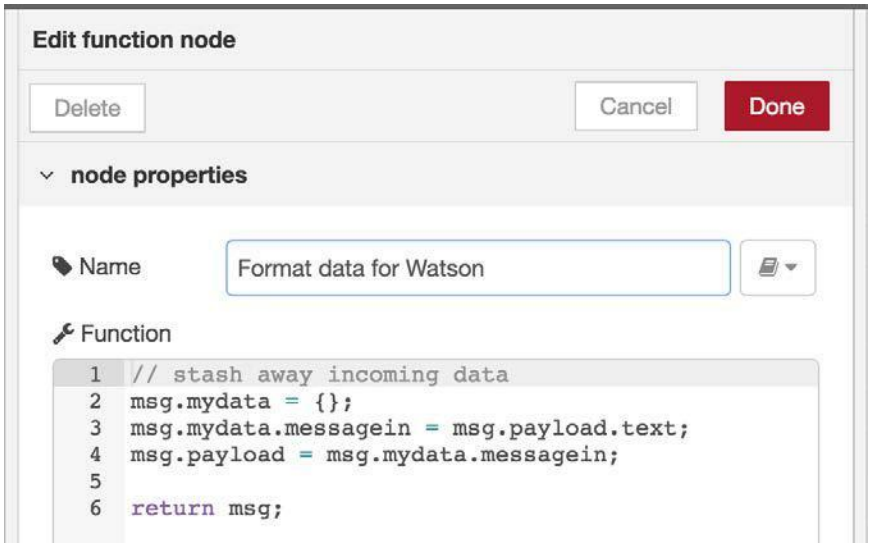
4. パレットの [関数（Functions）] カテゴリから [関数（function）] ノードをキャンバスにドラッグします。



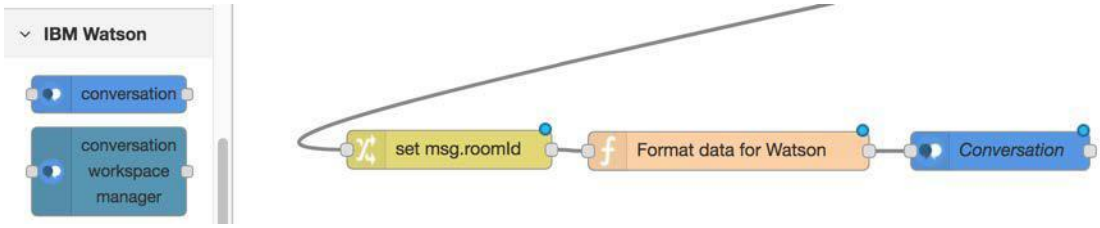
5. メッセージをフォーマットするために、以下のコードを [関数（function）] ノードに追加する必要があります。

```
msg.mydata = {};
msg.mydata.messagein = msg.payload.text;
msg.payload = msg.mydata.messagein;
return msg;
```

6. ノードを [編集（Edit）] し、[関数（Function）] セクションに上記コードを貼り付けます。[完了（Done）] をクリックします。



7. パレットの [IBM Watson] カテゴリから [カンバセーション（conversation）] ノードをキャンバスにドラッグします。

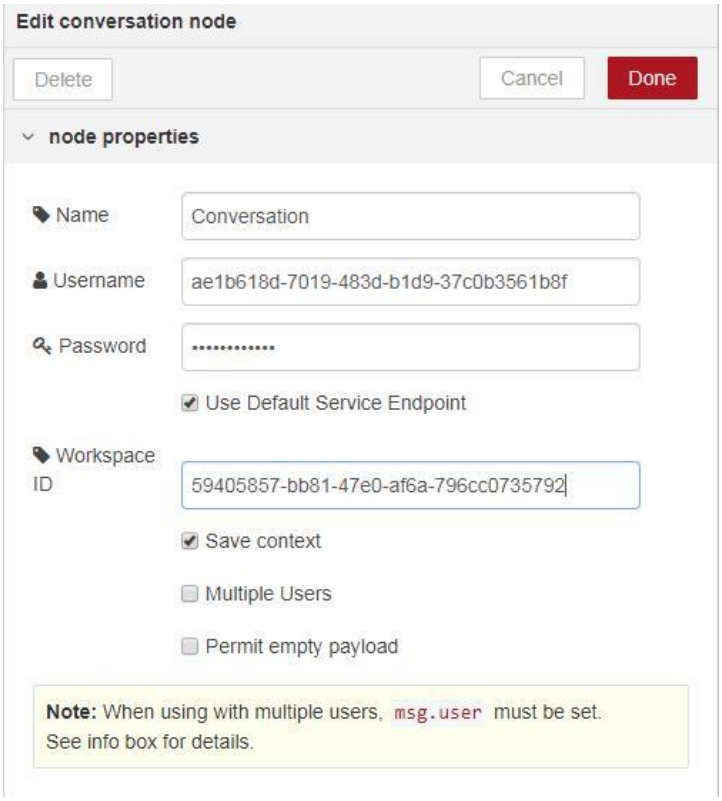


8. ノードを編集し、[プロパティ（Property）] の、[ユーザ名（Username）]、[パスワード（Password）]、[ワークスペースID（Workspace ID）] に Watson アカウントの該当する情報を設定します。この演習用に、Watson Conversation Workspace が事前に設定されています。次の詳細情報を使用して、この演習を完了させます。[完了（Done）] をクリックします。

[ユーザ名（Username）]：ae1b618d-7019-483d-b1d9-37c0b3561b8f

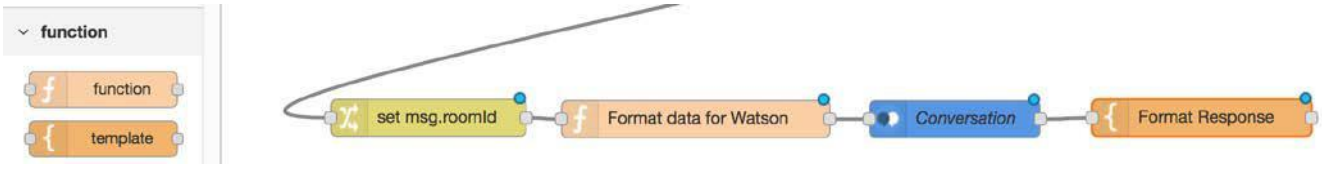
[パスワード（Password）]：k4XBNplDxIBl

[ワークスペースID（Workspace ID）]：59405857-bb81-47e0-af6a-796cc0735792



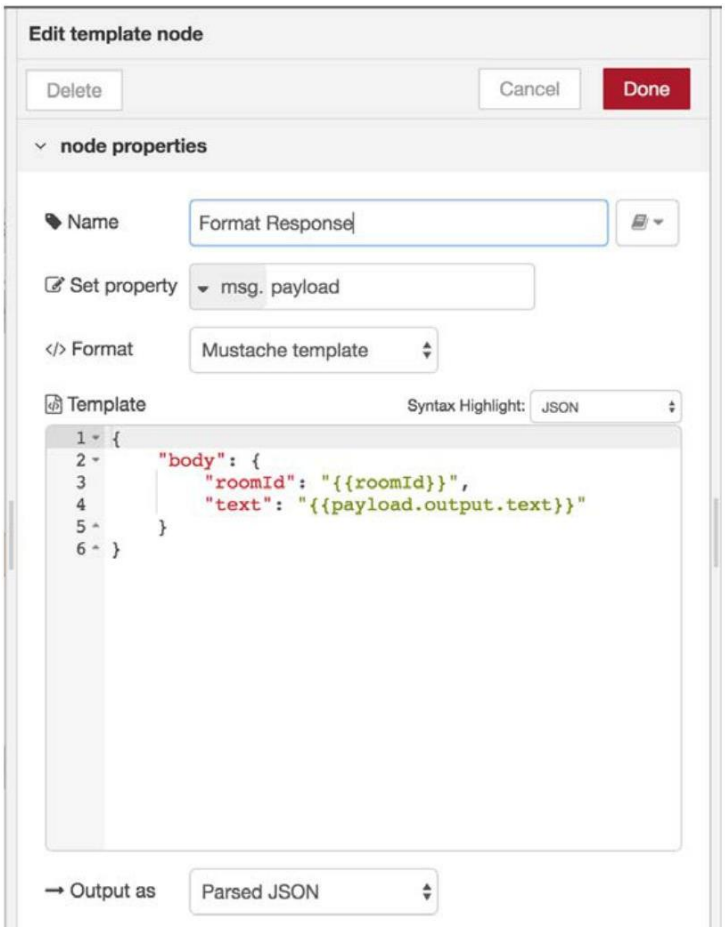
この時点で、ユーザのメッセージが IBM Watson Conversation に送信され、Watson から JSON 応答を受け取ることになります。この JSON は、会話の意図、各意図の確実性、エンティティ、応答内容、会話のコンテキストなどのさまざまな有用な情報を保持しています。これらすべてのデータを使用して、複雑なボットを作成することもできます。この演習では、「output.text」だけを使用して、詳細な分析はせずに、Watson から返ってきた応答内容をそのまま返信します。

9. パレットの [関数（Functions）] カテゴリから [テンプレート（template）] ノードをキャンバスにドラッグします。

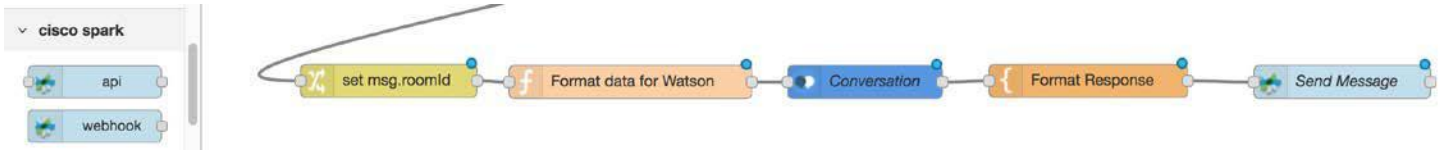


10. ノードを編集し、[セットプロパティ（Set property）] で、ドロップダウンから [msg.payload] を選択します。[フォーマット（Format）] に [Mustacheテンプレート（Mustache Template）] を選択します。[シンタックス指定（Syntax Highlight）] に [JSON] を設定します。[テンプレート（Template）] の body 部分に、次の JSON コードをコピーして貼り付けます。[完了（Done）] をクリックします。

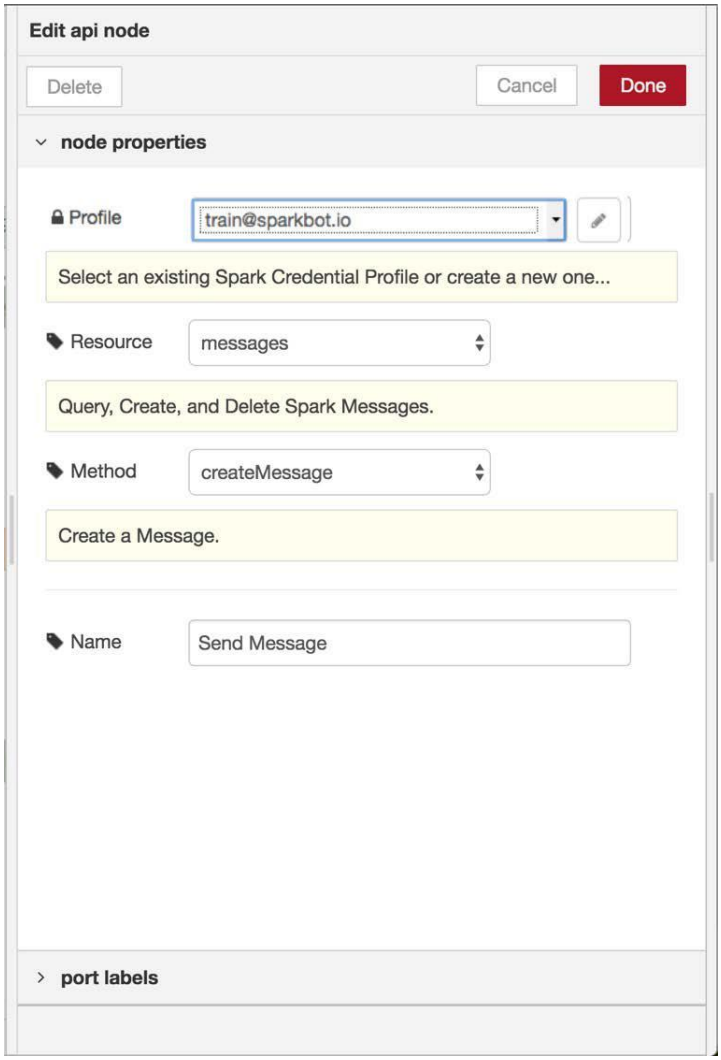
```
{
    "body": {
        "roomId": "{{roomId}}",
        "text": "{{payload.output.text}}"
    }
}
```



11. パレットの [Spark] カテゴリから [api] ノードをキャンバスにドラッグします。

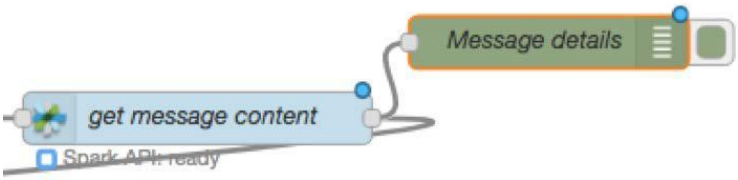


12. ノードを [編集(Edit)] します。[プロファイル(Profile)] で *<自分のボット>* を選択します。[リソース(Resource)] に [メッセージ(messages)] を選択し、[メソッド(Method)] に [createMessage] を選択します。[完了(Done)] をクリックします。

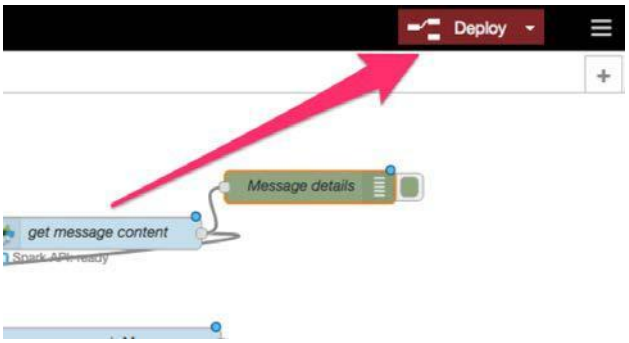


アプリケーションをテストする

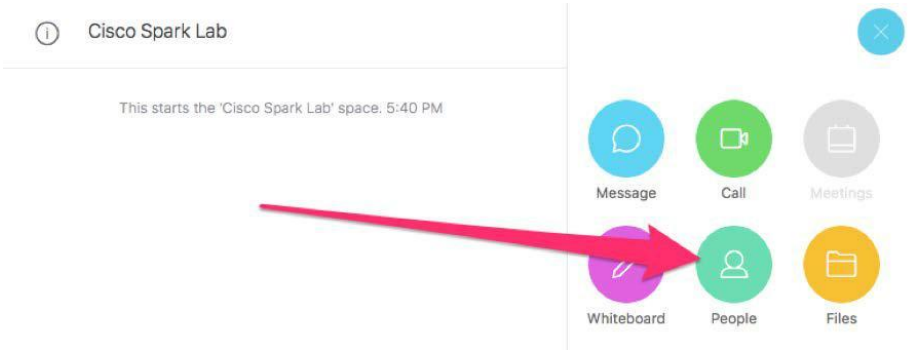
1. これでアプリケーションの作成が完了しました。必要に応じて、デバッグ ノードをフローにドラッグし、アプリケーションの各ノードの出力に接続します。



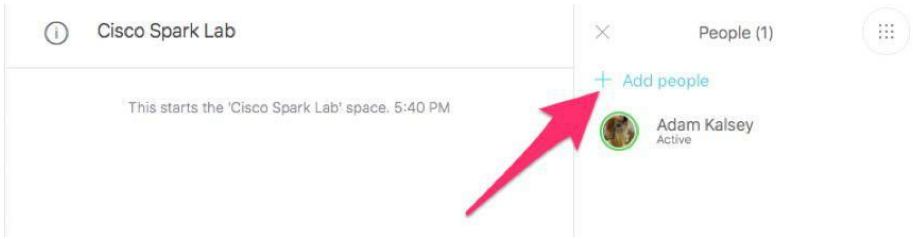
2. 上部右側の [導入（Deploy）] ボタンをクリックしてアプリケーションを導入します。



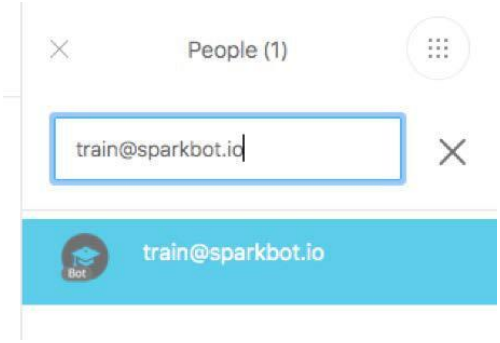
3. [ユーザアクティビティサークル（People Activity Circle）] アイコンをクリックして、Cisco Spark Space にボットを追加します。



4. [ユーザの追加（Add People）] をクリックします。



5. ボットのアドレスを入力し、表示されるオプションをクリックします。



6. @mention を利用してメッセージを送信します（グループ スペースにボットを追加した場合）。まだオウム ボットを使用しているため、同じメッセージが返されます。また、Watson カンバセーション サービスからのメッセージも返ってきます。たとえば、「hello」と送信すると、「「hello」と一緒に「Hi!」というメッセージも返ってきます。

注：ボットと直接通信する場合は @mention は必要ありません。

このラボでは、シンプルなボットをどんな会話でも一般的なほぼすべての意図に対応できる会話型ボットに拡張するために必要なインフラストラクチャのデモンストレーションを行います。

シナリオ 3.　Cisco Spark ビジョン ボット

このラボでは、画像の属性を把握する Cisco Spark ボットの作成方法をデモンストレーションします。このボットに PNG または JPG 画像の URL を送信すると、アプリケーションがメッセージから URL（テキスト）を抽出して Google Vision に転送し、画像の属性を識別して、ユーザのスペースに応答を返します。

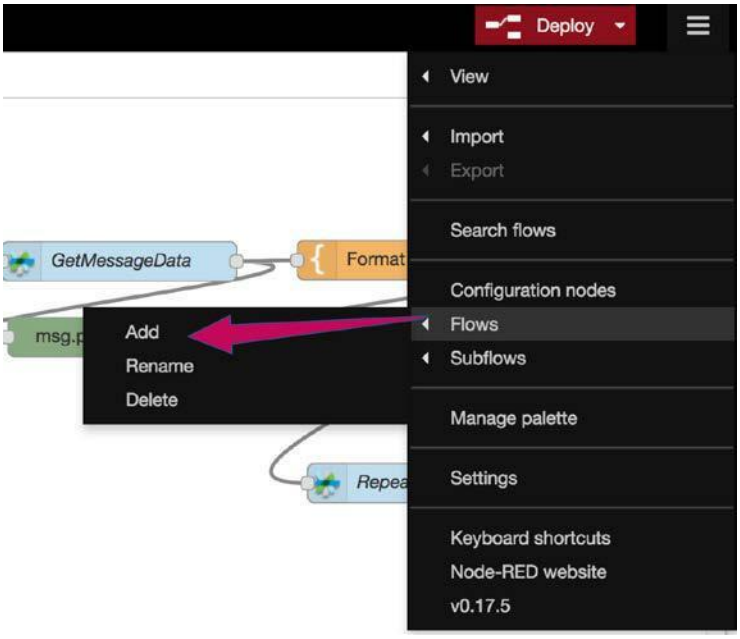
手順

1. 開始する前に、オウム ボット演習を完了し、完全に機能していることを確認します。ブラウザから http://localhost:1880/ にアクセスして Node-RED を開き、フローの 1 つとしてオウム ボット フローが開かれ、無効化されていることを確認します。オウム ボットを基にしてビジョン ボットを構築します。

新しい Node-RED フローを作成する

この演習用の新しいフローを作成します。

1. 画面の右上隅のメニューを開き、[フロー（Flows）] -> [追加（Add）] の順にクリックして新しいフローを追加します。



フローから別のフローに内容をコピーする

オウム ボットと新しいビジョン ボットの共通点を確認しましょう。この演習ではさらに以下を実施する必要があります。

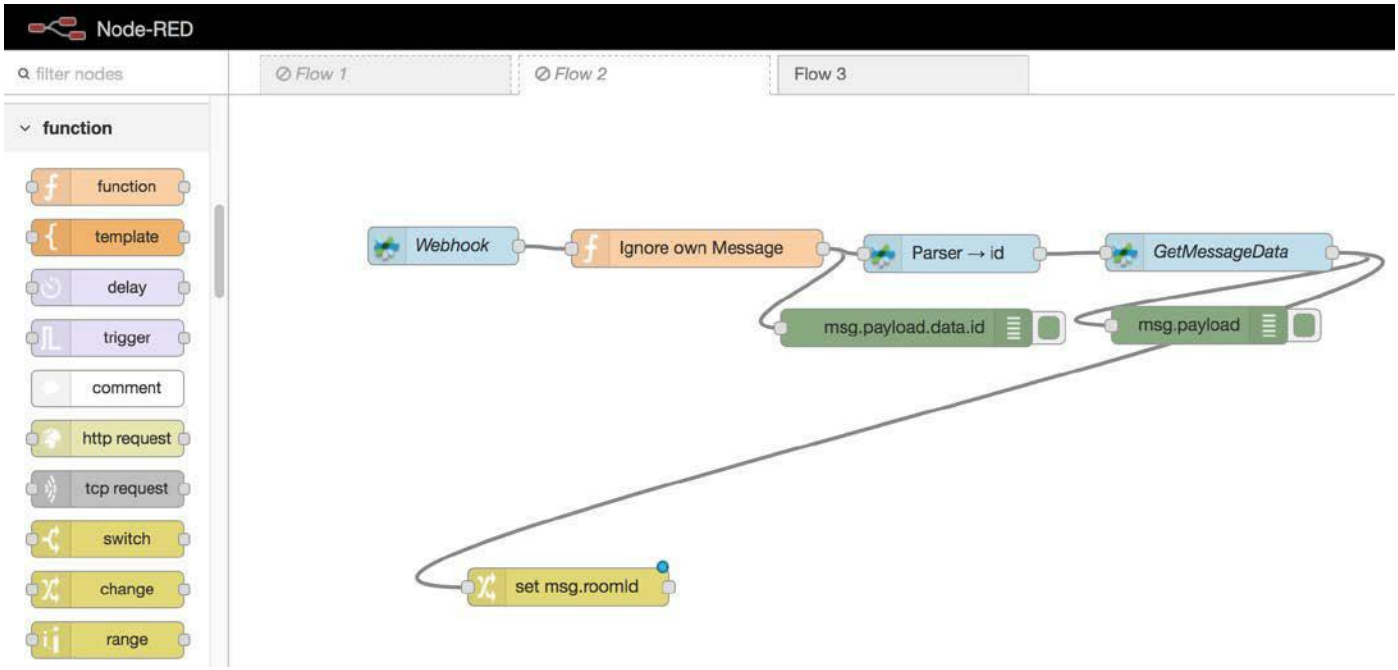
- ウェブフックを作成する
- ボット自体から送信されたメッセージを無視する
- ウェブフックを解析する
- Spark からの実際のメッセージを取得する

この時点で、会話型ボットは、Spark Space に同じメッセージを返す代わりに、Google Vision Service にメッセージを送信する必要があります。フローを始めるために、オウム ボットから上記の手順をコピーして、新しいフローに貼り付けます。

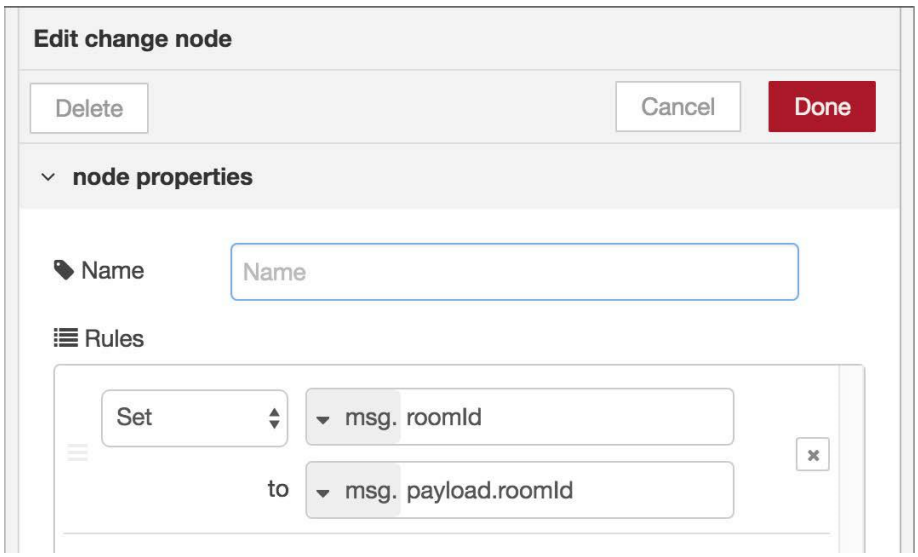
1. オウム ボット フローからブロック（Spark Webhook：Messages.created、関数：Ignore own message、Parser → id、Spark API：get message）を選択します。新しいフローにコピー（**Ctrl + C**）して、貼り付けます（**Ctrl + V**）。コピーしたノードの数を示す確認ポップアップが表示されます。

アプリケーションの作成を続ける

1. 次の手順では、後で参照できるように、別の変数に roomId を保存する関数を追加します。メッセージ プロパティを設定/移動するには、[変更（change）] ノードを使用する必要があります。
2. パレットの [関数（Functions）] カテゴリから [変更（change）] ノードをキャンバスにドラッグします。

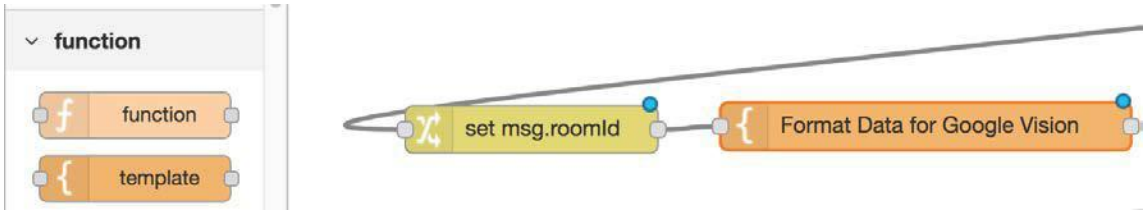


3. ノードを [編集(Edit)] し、[プロパティ(Property)] で、[設定(Set)]、[msg.roomId] to [msg.payload.roomId] と指定します。[完了(Done)] をクリックします。



次に、Google Vision に合う形式にメッセージをフォーマットします。

4. パレットの [関数(Functions)] カテゴリから [テンプレート(template)] ノードをキャンバスにドラッグします。

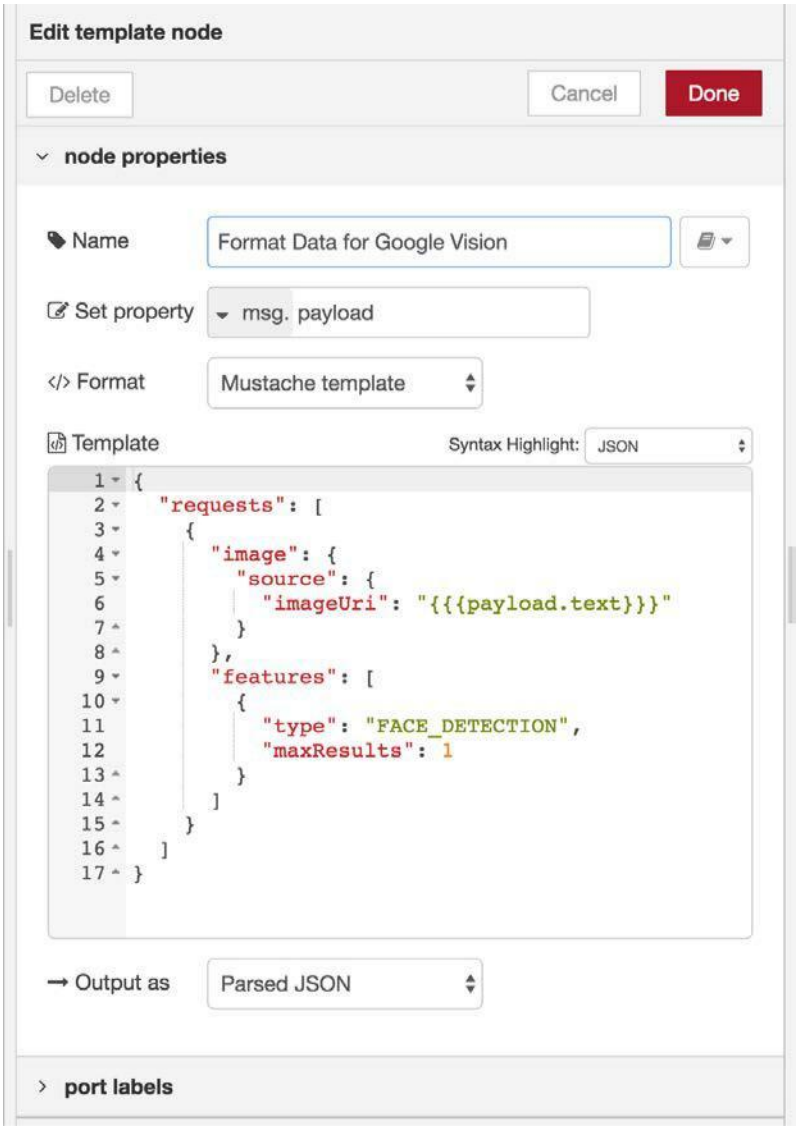


5. メッセージをフォーマットするために、以下のコードを [関数(function)] ノードに追加する必要があります。

```
{
    "requests": [
      {
        "image":
          { "source":
          {
            "imageUri": "{{{payload.text}}}"
          }
        },
        "features": [
          {
            "type": "FACE_DETECTION",
            "maxResults": 1
          }
        ]
      }
    ]
}
```

1 つ注意して欲しい点は、**"imageUri": "{{{payload.text}}}"** の中括弧が 2 重ではなく、3 重になっていることです。これは、mustache ではデフォルトですべての HTML エンティティを代替変数内でエスケープ処理するからです。たとえば、*https://* がエスケープ処理されると https:// となりますが、その場合、不正な URL が送信されることになります。それを避けるために、中括弧を 3 重に使用するのです。

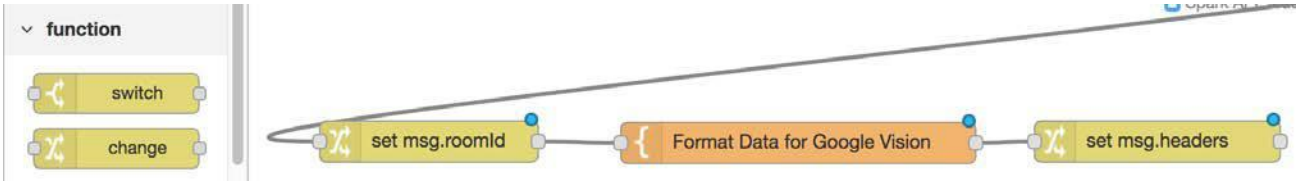
6. ノードを [編集（Edit）] し、[テンプレート（Template）] フィールドに上記のコードを以下のように追加します。[完了（Done）] をクリックします。



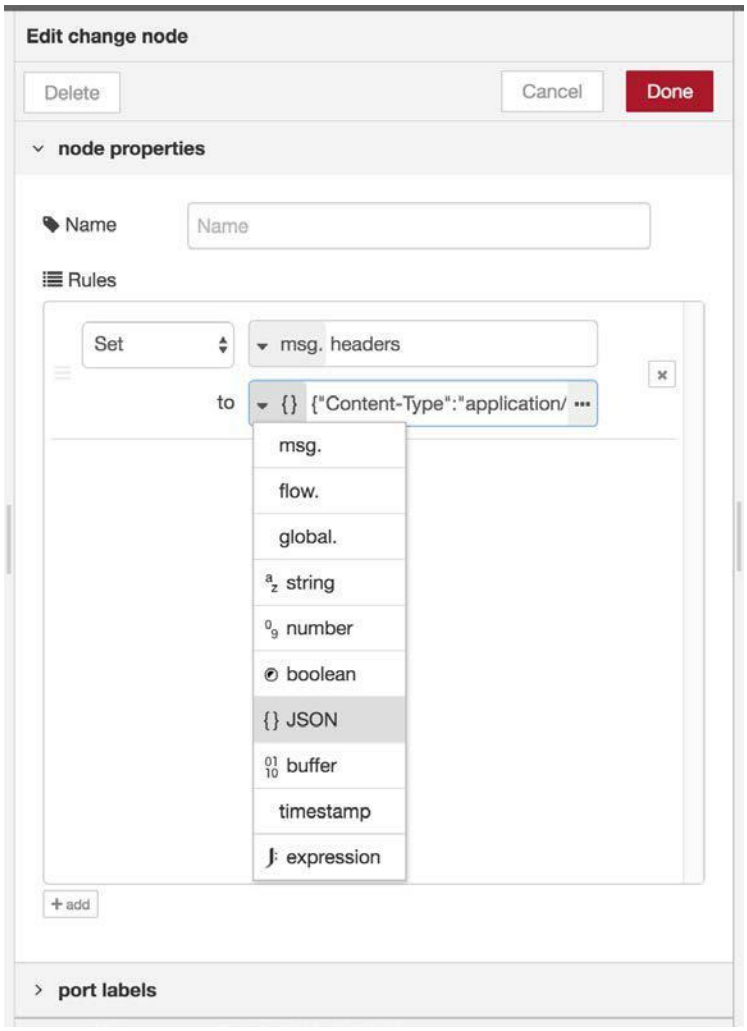
Node-RED の IBM Watson Component を使用した会話型ボットの演習とは異なり、Google Vision の事前構築済みコンポーネントはない前提で進めます。実際にはこのようなケースは頻繁に発生します。ただし、使用するサービスが、標準 API（REST や SOAP など）を公開している限り、Node-RED のコンポーネントがなくても、その API と容易に通信できます。

Google Vision の場合、上記の手順は、API サービスに JSON データを送信することを示します。そのため、まず、送信されるデータが JSON であることを示すヘッダーを設定する必要があります。

7. パレットの [関数（Functions）] カテゴリから [変更（change）] ノードをキャンバスにドラッグします。

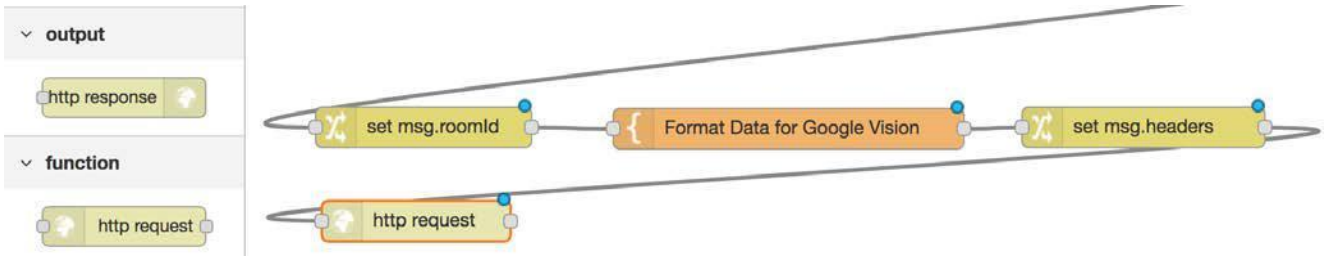


8. ノードを編集し、[プロパティ（Property）] で、[設定（Set）]、[msg.headers] to [{"Content-Type":"application/json"}] のように指定します。このフィールドの種類に JSON を選択していることを確認します。[完了（Done）] をクリックします。



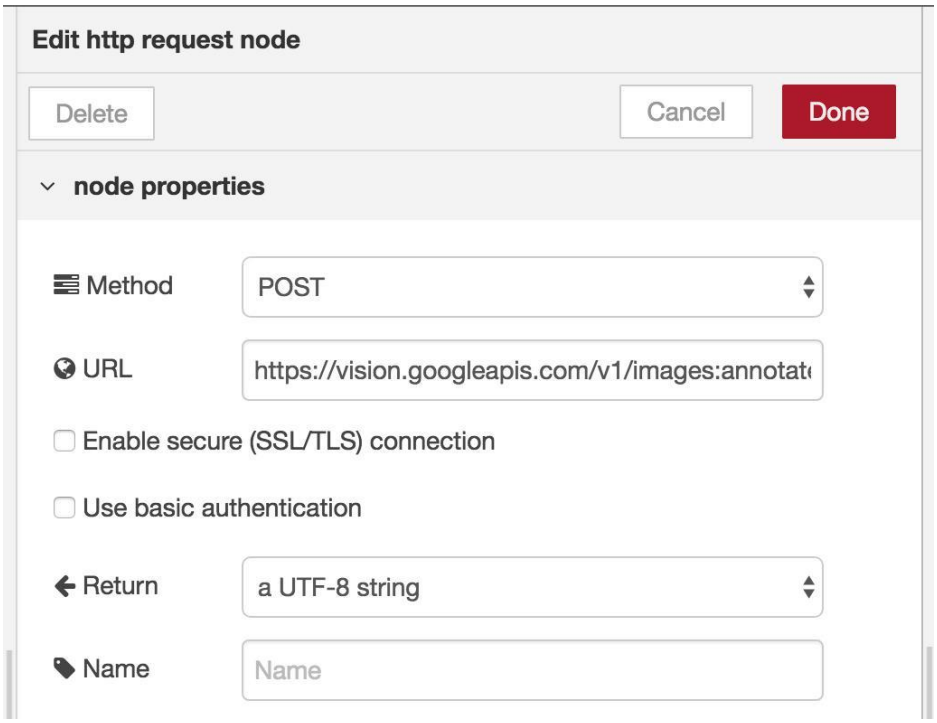
次のステップは、Google Vision API を呼び出すことです。ただし、前に述べたように、Google Vision 用の Node-RED コンポーネントはありません。代わりに、[関数（Functions）] の下にある、Node-RED の [http request] 汎用コンポーネントを使用します。「http request」コンポーネントを使用すれば、GET、PUT、POST、DELETE の各操作を送信できます。この汎用コンポーネントを使用するのは、サービスが HTTP ベースの API を公開しているが、Node-RED コンポーネントは用意されていない場合であることに注意してください。

9. パレットの [関数（Functions）] カテゴリから [http request] ノードをキャンバスにドラッグします。



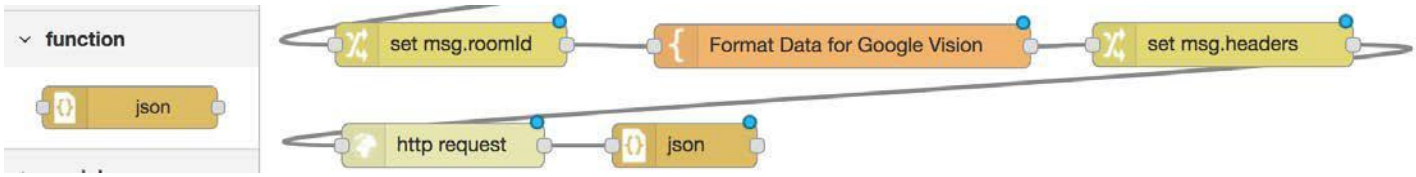
10. ノードを [編集（Edit）] し、[メソッド（Method）] には [POST] を、[URL] には、以下のように Google Vision API の URL を設定します。このラボでは、Google Vision API サービスはすでに有効化されており、キーも生成されています。次の詳細情報を使用して、この演習を完了させます。[完了（Done）] をクリックします。

URL：https://vision.googleapis.com/v1/images:annotate?key=AIzaSyBI9RX2yStT68KhrbYspLDmvrZSqtVD_Yw



この時点で、ユーザのメッセージが Google Vision に送信され、Google から応答を受け取ることになります。応答が文字列として返される場合もあります（JSON オブジェクトの場合を含む）。そのため、この JSON 文字列を JSON オブジェクトに変換する必要があります。

11. パレットの [関数（Functions）] カテゴリから [json] ノードをキャンバスにドラッグします。

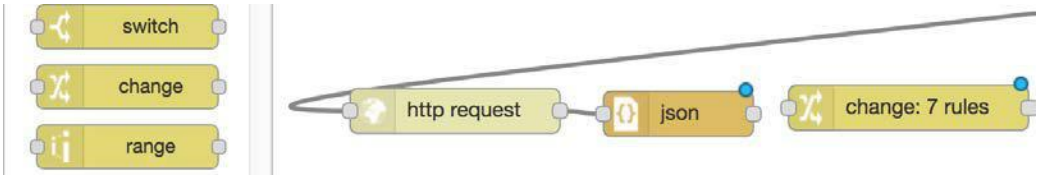


この JSON オブジェクトには、joyLikelihood、sorrowLikelihood、angerLikelihood、surpriseLikelihood や、LEFT_EYE、RIGHT_EYE、RIGHT_EYEBROW ポジションなどのランドマーク情報をはじめとした、さまざまな貴重な情報が含まれています。これらすべてのデータを使用して、複雑なボットを作成することもできます。この演習では、いくつかの基本的な表情変数だけを使用して、詳細な分析はせずに、Google Vision から返ってきた応答内容をそのまま返信します。

ヒント:** デバッグ ** ノードを使用すれば、** json ** ノードの出力結果をプリントして全体を確認することができます。

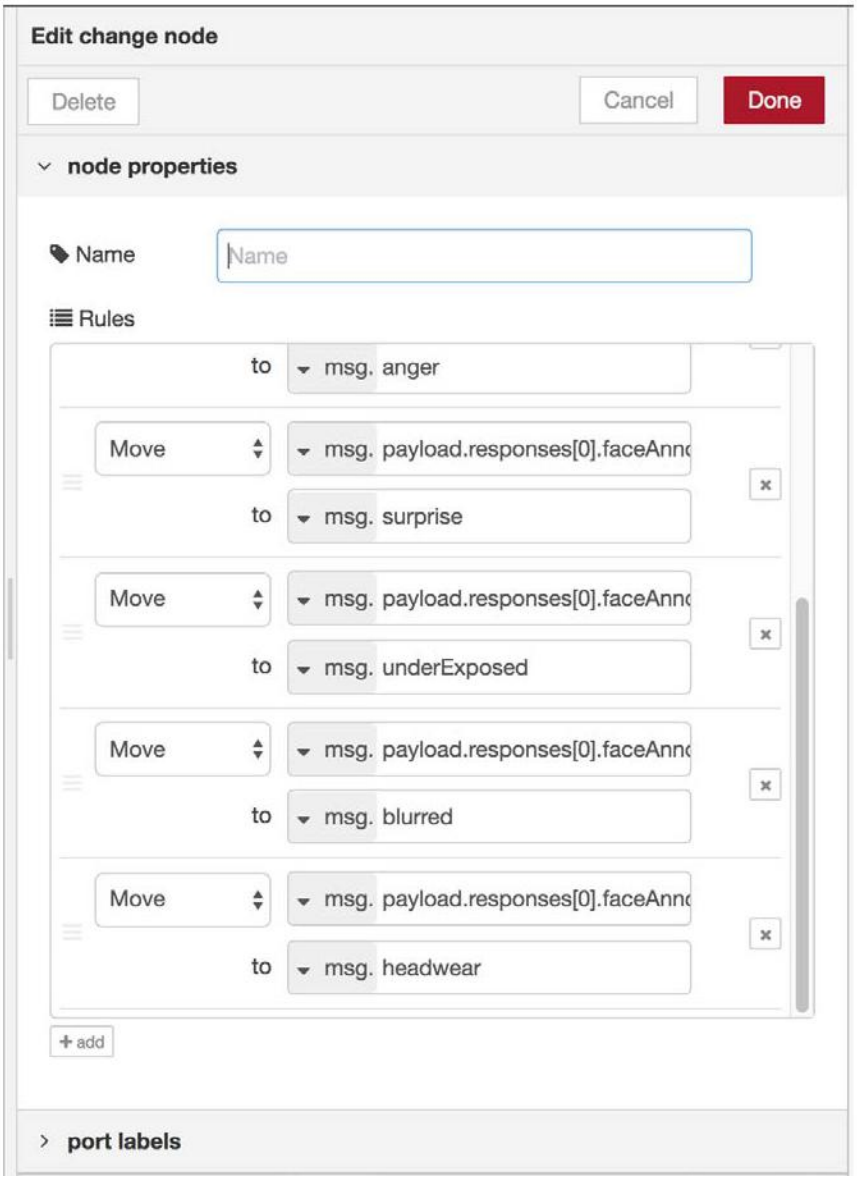
Spark に送信されるメッセージをフォーマットする前に、JSON オブジェクトのこれらの重要な情報の一部を抽出して別の変数に移動します。そのために [変更（change）] ノードを使用します。

12. パレットの [関数（Functions）] カテゴリから [変更（change）] ノードをキャンバスにドラッグします。



13. ノードを [編集（Edit）] し、[ルールプロパティ（Rules Property）] で、[移行（Move）] オプションを選択し、[移動（Move）]、[msg.payload.responses[0].faceAnnotations[0].joyLikelihood] to [msg.joy] と指定します。[追加（add）] ボタンをクリックして、その他の情報をすべて該当の変数に [移動（move）] します。[完了（Done）] をクリックします。

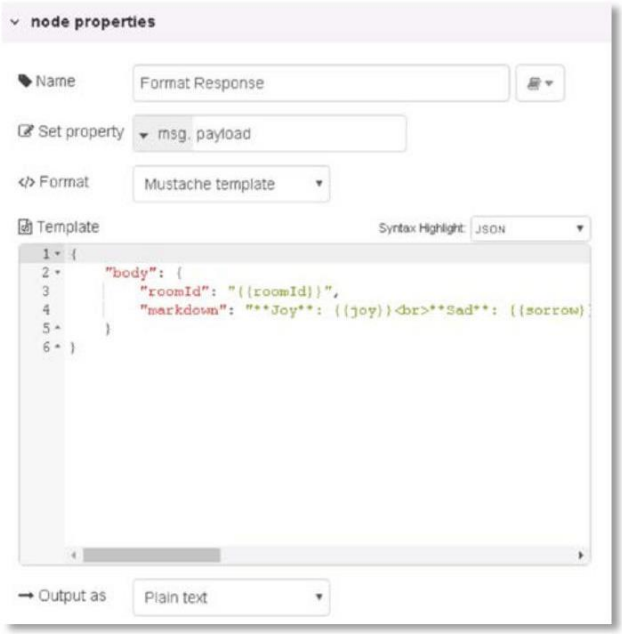
Move	*msg.payload.responses[0].faceAnnotations[0].joyLikelihood*	*msg.joy*
Move	*msg.payload.responses[0].faceAnnotations[0].*sorrowLikelihood	*msg.sorrow*
Move	*msg.payload.responses[0].faceAnnotations[0].*angerLikelihood	*msg.anger*
Move	*msg.payload.responses[0].faceAnnotations[0].*surpriseLikelihood	*msg.surprise*
Move	*msg.payload.responses[0].faceAnnotations[0].underExposedLikelihood*	*msg.underExposed*
Move	*msg.payload.responses[0].faceAnnotations[0].blurredLikelihood*	*msg.blurred*
Move	*msg.payload.responses[0].faceAnnotations[0].headwearLikelihood*	*msg.headwear*



14. パレットの [関数（Functions）] カテゴリから [テンプレート（template）] ノードをキャンバスにドラッグします。

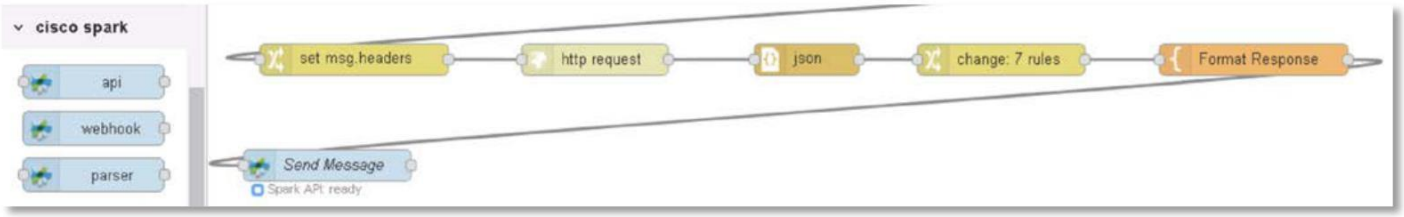
15. ノードを [編集（Edit）] し、[セットプロパティ（Set property）] で [msg.payload] を選択します。[フォーマット（Format）] に [Mustacheテンプレート（Mustache Template）] を選択します。[シンタックス指定（Syntax Highlight）] に [JSON] を設定します。[テンプレート（Template）] の本体部分に、次の JSON コードを挿入します。[完了（Done）] をクリックします。テキストの一部は画面に表示されない場合がありますが、右にスクロールすると表示できます。

```
{
    "body": {
        "roomId": "{{roomId}}",
        "markdown": "**Joy**: {{joy}}<br>**Sad**: {{sorrow}}<br>**Angry**: {{anger}}<br>**Surprised**:
{{surprise}}<br>**Under Exposed**: {{underExposed}}<br>**Blurred**: {{blurred}}<br>**Headwear**:
{{headwear}}"
    }
}
```

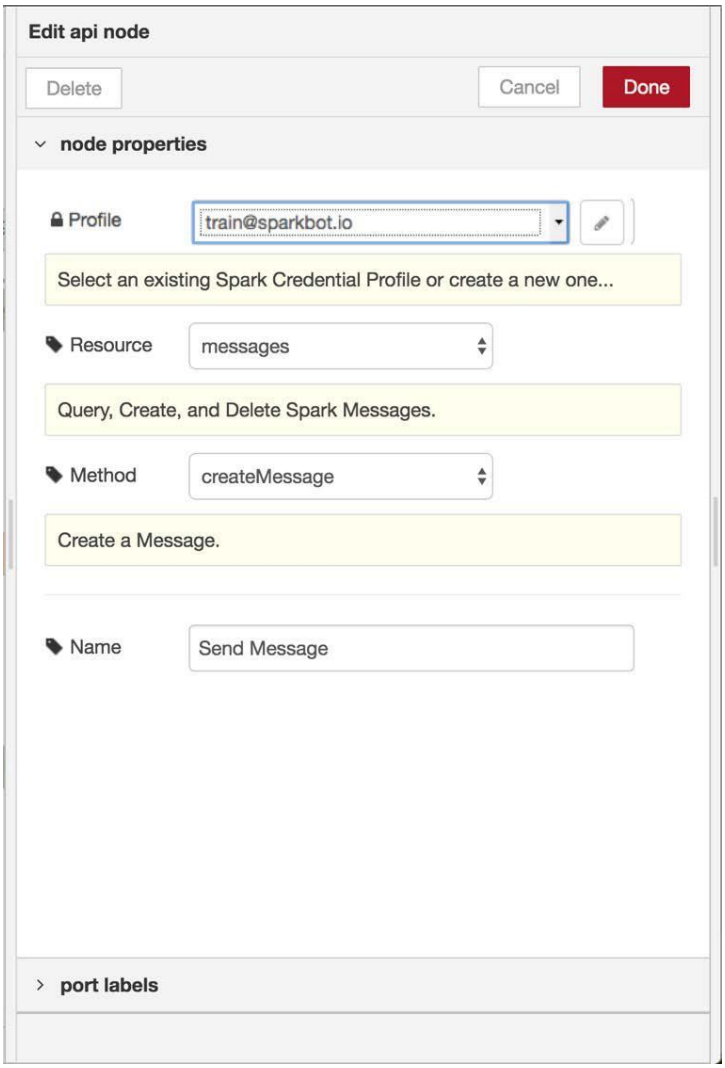


markdown が何を意味するかご存知ですか。markdown は、メッセージをリッチ テキスト形式に変換する優れた方法です。モバイル、デスクトップ、Web などあらゆるプラットフォームに対応しています。markdown に関する記事を読んで、自分独自のリッチ テキスト形式を見つけてください。

16. パレットの [Spark] カテゴリから [api] ノードをキャンバスにドラッグします。

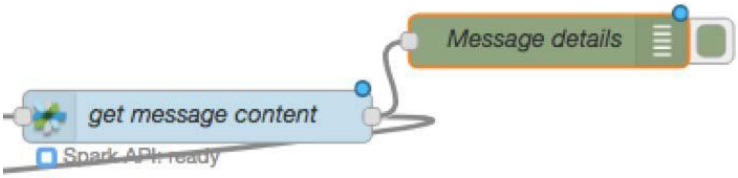


17. ノードを [編集（Edit）] します。[プロファイル（Profile）] に *<使用するボット名>* を設定します。[リソース（Resource）] に [メッセージ（messages）] を選択し、[メソッド（Method）] に [createMessage] を設定します。[完了（Done）] をクリックします。

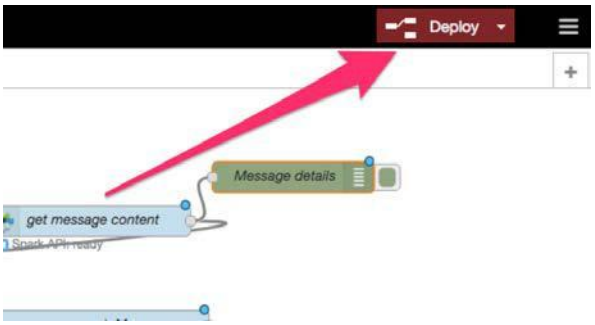


アプリケーションをテストする

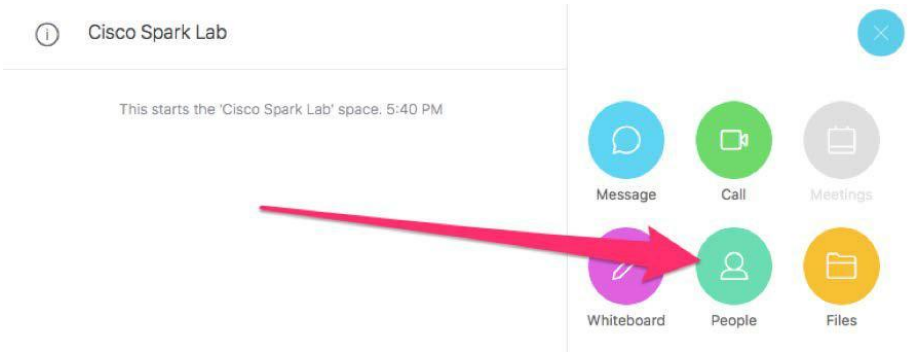
1. これでアプリケーションの作成が完了しました。必要に応じて、デバッグ ノードをフローにドラッグし、アプリケーションの各ノードの出力に接続します。



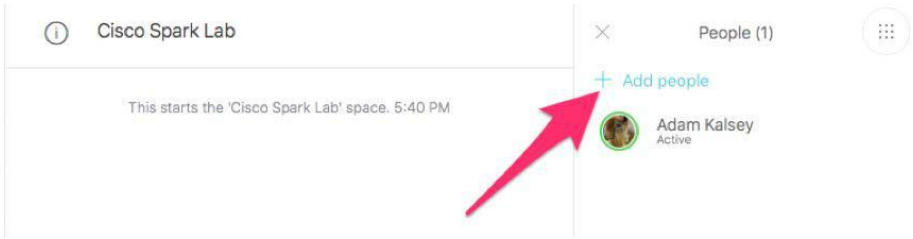
2. [導入（Deploy）] ボタンをクリックしてアプリケーションを導入します。



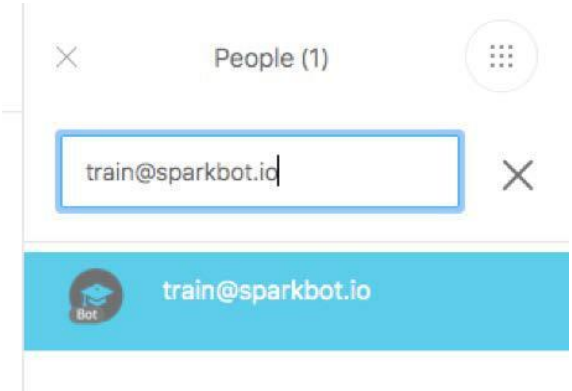
3. [ユーザアクティビティサークル（People Activity Circle）] アイコンをクリックして、Cisco Spark Space にボットを追加します。



4. [ユーザの追加（Add People）] をクリックします。



5. ボットのアドレスを入力し、表示される検索結果をクリックします。



6. @mention を利用して URL（ユーザの写真の画像）をボットに送信します（ボットをグループ スペースに追加している場合）。ボットが画像のプロパティを返してくるのを確認します。

注：ボットと直接通信する場合は @mention は必要ありません。

7. ビジョン ボットのテストに利用できるサンプル URL を以下に示します。

- https://www.biography.com/.image/t_share/MTE1ODA0OTcxODAxNTQ0MjA1/mother-teresa-9504160-1-402.jpg（帽子をかぶった喜びの表情）
- https://media2.popsugar-assets.com/files/2013/11/07/831/n/1922398/9d62fd4fa48c4072_25.gif（驚いた様子）
- http://media.dishnation.com/content/uploads/2015/09/Dish-Nation-smith.jpg（笑顔）一度に 1 つづつ URL を送信し、結果を確認します。

これで、写真の人物の画像属性の一部がボットから返されるのを確認できます。このラボでは、シンプルなボットをビジョン ボットに拡張するために必要なインフラストラクチャのデモンストレーションを行いました。

お問い合せ先

シスコシステムズ合同会社
〒107 - 6227 東京都港区赤坂9-7-1 ミッドタウン・タワー
http://www.cisco.com/jp