{ “hands-on” }

Chatbot pour IBM i

v1.2

Ce hands-on vous permet de découvrir comment développer un prototype de chatbot permettant d’interroger l’état des ressources système d’un IBM i.
Il utilise les services Node-RED et Watson de Bluemix pour le chatbot, et Slack pour l’interface utilisateur. L’état des ressources système de l’IBM i sera exposé sous forme de web services implémentés en Node.js.

Tous ces composants seront étudiés et implémentés.

Auteur : C. Lalevée (lalevee@fr.ibm.com)
Version 1.2 du 15 mai 2017

Agenda

Présentation du Hands-on

Information

Pour pouvoir faire ce hands-on, vous devez posséder un compte Bluemix. Si vous n'en avez pas encore, vous pouvez le créer ici : http://ibm.biz/Bluemix-BC.

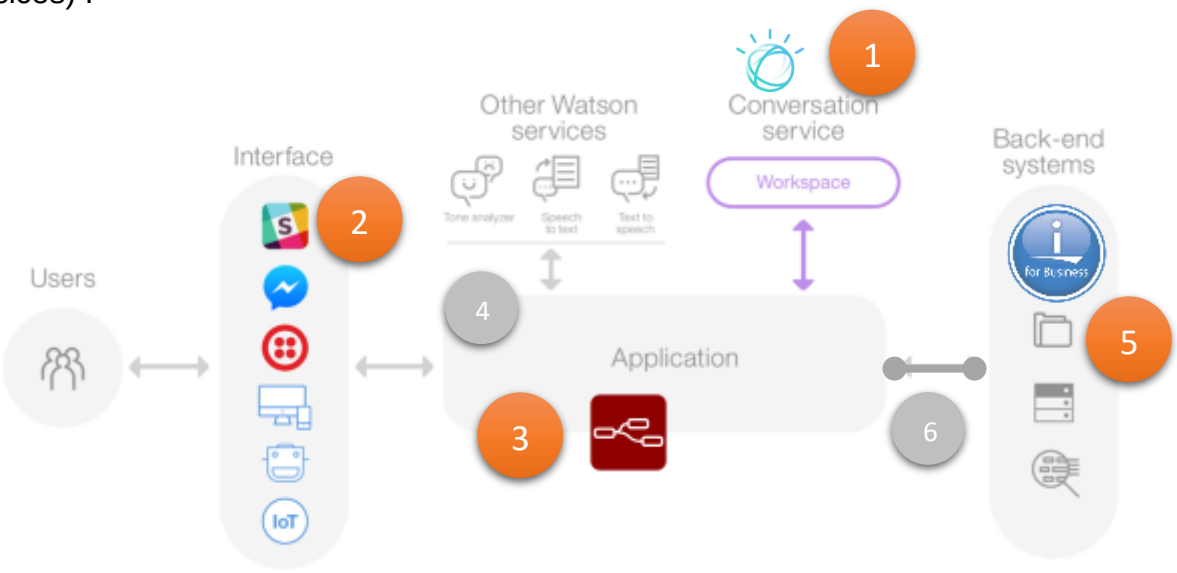
Dans cet hands-on, vous allez créer un chatbot qui permet d'interroger, en langage naturel, l'état des ressources système (CPU, ASP) d'un IBM i, en utilisant Slack comme interface utilisateur.

Ce hands-on est conçu comme un développement de prototype.
Vous utiliserez donc Node-RED, une interface de développement graphique idéale pour implémenter rapidement une application, sans presque écrire de code.
Dans ce prototype, 2 demandes utilisateur seulement seront traitées : dire bonjour et demander le taux d'utilisation du processeur et de l'ASP système. Cette dernière demande vous amènera à créer 2 web services sur l'IBM i en utilisant SQL et Node.js.

Enfin, le prototypage étant la clé de voûte du développement itératif et agile, la disponibilité immédiate d'une plateforme adéquate pour supporter ce type de développement est fondamentale. Vous utiliserez donc Bluemix Platform et ses nombreux services innovants.

L'IBM i utilisé est hébergé dans un Datacenter IBM à Montpellier, France. Pour que vous puissiez y accéder, nous avons installé sur votre poste de travail, un logiciel de VPN : OpenVPN. Votre application chatbot, dans Bluemix, utilisera, elle, un service Bluemix d'intégration : Secure Gateway. Le déploiement et la configuration de ce service ne fait pas partie de ce hands-on. Vous utiliserez une instance préconfigurée pour vous.

La vue globale de ce hands-on est donc (les numéros correspondent au numéro des exercices) :



Vous aurez 4 composants à déployer et configurer :

1) Le service Watson Conversation dans Bluemix en charge de la reconnaissance des intentions et de la gestion du dialogue
2) Le logiciel Slack, l'interface utilisateur, au sein duquel vous allez créer un « **Bot Slack** », composant servant à interfacer Slack à votre propre programme de Chatbot (3)
3) Votre programme de Chatbot, développé en Node-RED dans Bluemix. C'est le cœur de la solution. Il fait le lien entre le Service Watson Conversation (1), l'interface utilisateur (2), et des sources de données comme les APIs REST de l'IBM i (5). C'est le composant l'« **application logic** ».
5) Un programme Node.js sur IBM i pour exposer l'état du système (« WRKSYSSTS ») sous forme d'APIs REST. Durant les étapes 1 à 4 (4 étant la phase de test), vous utiliserez un programme Node.js existant et mis à disposition.
6) Le service d'intégration Bluemix, la Secure Gateway, qui permet de mettre en place une communication sécurisée (type VPN) entre votre programme Chatbot s'exécutant sur Bluemix (Cloud Public) et votre programme IBM i Node.js (Datacenter interne IBM – Montpellier, privé). Le déploiement et la configuration de ce service ne font pas partie de cet exercice.

0. Connection au poste de travail

Connectez-vous à votre machine virtuelle à l’aide du logiciel "Remote Desktop Connection".

1. Bluemix : déploiement et configuration de Watson Conversation

Dans cette première partie, vous allez déployer, configurer et tester le service Bluemix Watson qui sera en charge du dialogue avec l'utilisateur du chatbot.

Section 1. Connexion et configuration Bluemix

__ 1. Connectez-vous à votre compte Bluemix (http://www.bluemix.net) à l'aide de votre IBM ID. Si nécessaire, suivez la procédure pour créer un nouveau compte.

__ 2. Choisissez la région Royaume-Uni et l'espace dans lequel vous voulez travailler. Vous pouvez créer un nouvel espace si vous le souhaitez (lien « créer un espace »)

Section 2. Déploiement de Watson Conversation

__ 1. Afficher le catalogue des services (menu en haut à droite)

__ 2. Dans la catégorie Watson, retrouvez le service Conversation, et cliquez dessus. Vous obtenez la page de configuration du service.

__ 3. Renseignez le nom du service

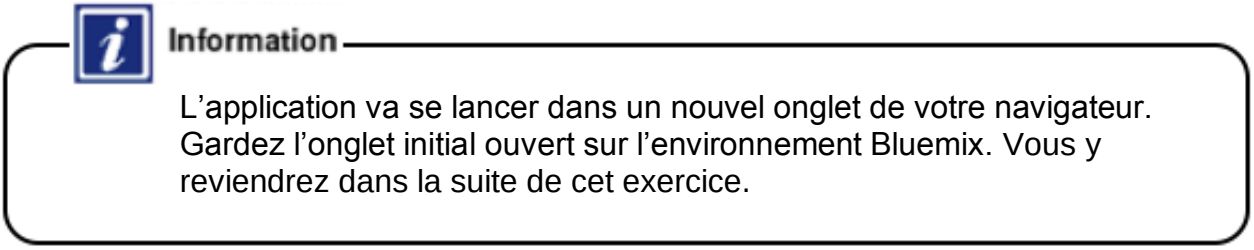
__ 4. Consulter les différents plans de facturation et sélectionnez le plan gratuit (1 000 requêtes d'API par mois, jusqu'à 3 espaces de travail, jusqu'à 25 intentions, Cloud public partagé).
Notez qu'il existe un plan « Premium » permettant d'avoir une instance dédiée de services Watson Developer Cloud, pour répondre aux besoins d'isolation maximum que vous pourriez avoir.

__ 5. Créez le service (bouton « **Créer** » au bas de la page)

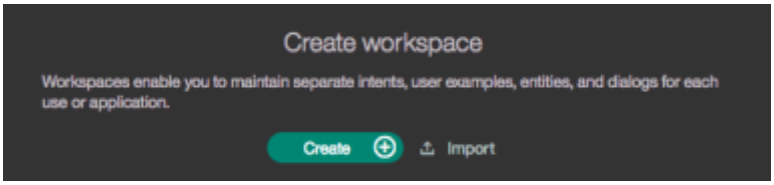
Section 3. Configuration initiale de Conversation

__ 1. Le service créé, vous obtenez la page d'accueil de votre service Conversation.
A partir du menu « **Gérer** », cliquez sur le bouton

pour lancer l'application de configuration du service.



__ 2. Cette application permet de gérer et configurer plusieurs « **workspaces** ». Un « **workspace** » est un environnement à l'intérieur du service Conversation, vous permettant de gérer de manière indépendante les artefacts d'un dialogue pour un usage ou une application particulière.
Créez un nouveau « **workspace** »



__ 3. Donnez un nom à ce nouveau workspace et sélectionnez la langue dans laquelle vous voulez gérer le dialogue.

__ 4. Vous arrivez sur la page vous permettant de créer de nouvelles intentions.
Un dialogue se compose de 3 types de données :

- Les intentions : « le verbe »
  Les intentions représentent le but de l'entrée d'un utilisateur. Vous pouvez considérer les intentions comme les actions que vos utilisateurs pourraient vouloir effectuer avec votre application.
  Exemple d'intention : Connaître la valeur d'une ressource du système

- Les entités : « le nom »
  Une entité représente un terme ou un objet dans la texte de l'utilisateur, qui fournit des éclaircissements ou un contexte spécifique pour une intention particulière. En reconnaissant les entités qui sont mentionnées dans l'entrée de l'utilisateur, le service Conversation peut choisir les actions spécifiques à prendre pour réaliser une intention donnée.
  Exemple d'entité : Ressource système (CPU, Disque, Mémoire, …)

- Le dialogue lui-même
  Le dialogue définit le flux de votre conversation sous la forme d'un arbre logique. Chaque nœud de l'arbre a une condition qui le déclenche, en fonction de l'entrée de l'utilisateur.
  Le but du dialogue est de conduire à la réponse à une question ou à l'exécution d'une action / commande.

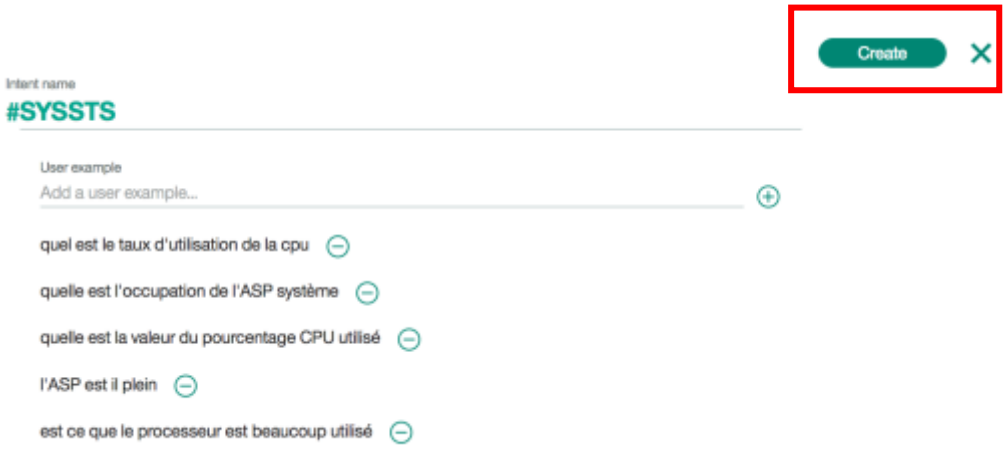
__ 5. Créez une nouvelle intention (bouton « **create new** »)

__ 6. Le but sera d'identifier l'intention d'un utilisateur voulant connaitre le taux d'utilisation d'une ressource système.
Nommez l'intention. (SYSSTS par exemple).

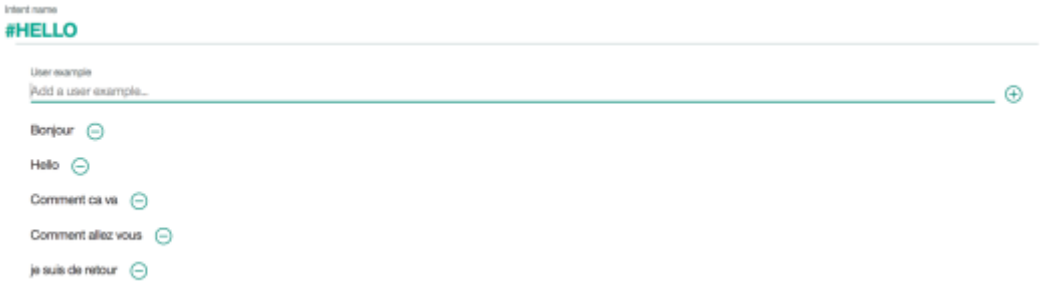
__ 7. Bien entendu, Conversation ne détectera pas l'intention sur des phrases prédéfinies, mais utilisera des algorithmes de type « **machine learning** » pour reconnaître les intentions. Énumérez plusieurs façons de poser la même question (min. 5) pour aider votre robot à reconnaître l'intention.
Exemples :

- Quel est le taux d'utilisation de la cpu
- Quelle est la valeur du pourcentage CPU utilisé
- Quelle est l'occupation de l'ASP système

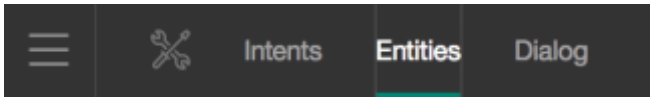
__ 8. Une fois les exemples entrés, créez l'intention en cliquant sur le bouton « **Create** »



__ 9. De la même manière, créez une intention #HELLO permettant de comprendre que notre utilisateur dit bonjour.

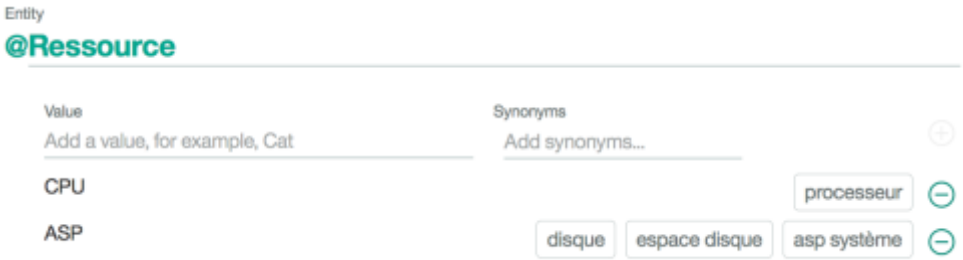


__ 10. Nous allons maintenant créer une entité permettant d'identifier quelle valeur système l'utilisateur veut voir avec cette intention.
Dans la barre de menu, cliquez sur « **Entities** »



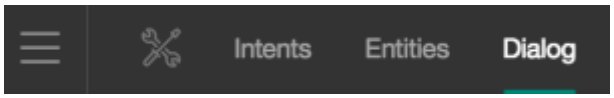
__ 11. Il existe 2 types d'entités : les entités système, « génériques », prédéfinie par le service Conversation (nombre, date, …) et devant simplement être activées, et les entités utilisateur que vous devez créer pour vos besoins spécifiques.
Créer une nouvelle entité : bouton « **Create new** »

__ 12. Le but est d'identifier à quelle ressource l'utilisateur s'intéresse : CPU ? ASP ?
Nous allons donc créer une entité « **ressource** », pouvant prendre les valeurs CPU ou ASP ou les synonymes de ces valeurs.
Saisissez valeurs et synonymes pour les ressources CPU et ASP, comme dans l'exemple ci-dessous.



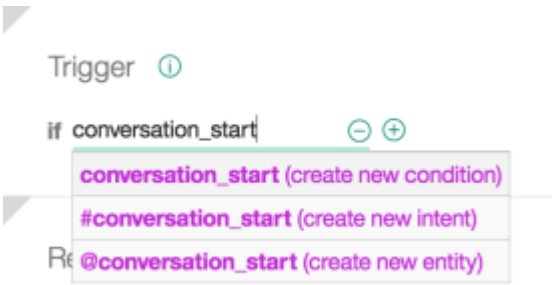
__ 13. Créer la nouvelle entité en cliquant sur le bouton « create ».

__ 14. Nous allons maintenant créer le flux permettant de dialoguer avec l'utilisateur et de lui fournir les réponses demandées.
Dans la barre de menu, cliquez sur « **Dialog**»



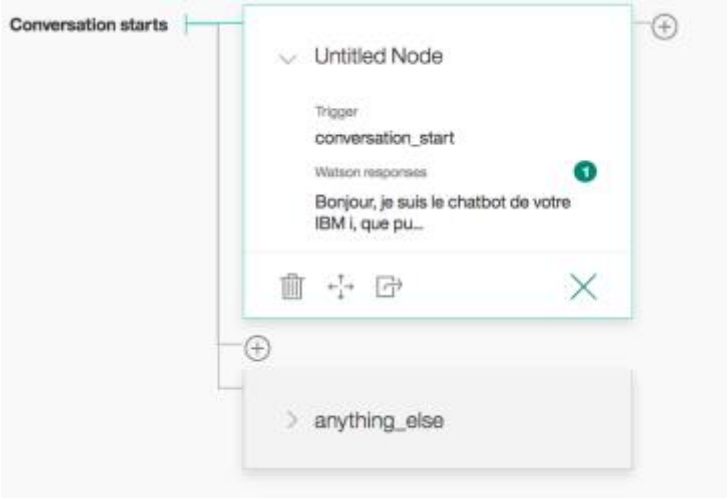
__ 15. Nous allons créer un dialogue simple.
Créer un nouveau dialogue : bouton « **Create** ». Vous verrez apparaître un premier nœud.

__ 16. Tout d'abord, créons le nœud pour démarrer le dialogue : un message d'accueil.
Dans ce premier nœud qui est apparu, entrez « `conversation_start` » dans le champ « `Enter a condition` ». Sélectionnez ensuite « **`conversation_start`** `(create new condition)` ».



__ 17. Dans le champ « `Enter a response` », entrez un message d'accueil puis <Enter>. Par exemple :

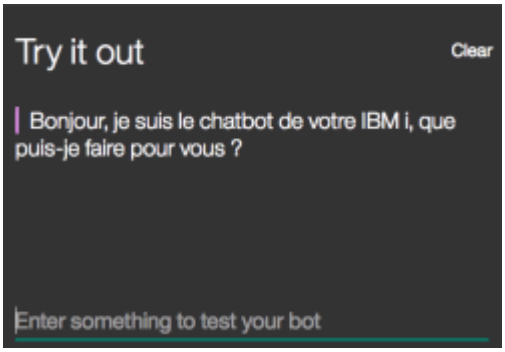
__ 18. Un autre nœud « `anything_else` » a été automatiquement créé, permettant de toujours répondre à l'utilisateur. Il sera utilisé quand le robot n'aura pas de réponse à fournir ou qu'il ne comprendra pas la saisie de l'utilisateur.



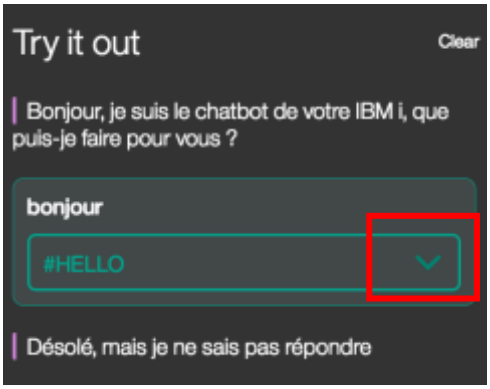
__ 19. Ouvrez le en cliquant sur « **>** » et saisissez une réponse.
Par exemple : « Désolé, mais je ne sais pas répondre à ça »

__ 20. Nous pouvons maintenant tester notre dialog.
En haut à droite de la fenêtre, cliquez sur l'icône

__ 21. Un panneau s'ouvre, contenant votre message d'accueil. Vous pouvez également dialoguez avec votre robot (champ « `Enter something to test your bot` »).



__ 22. Essayez de lui dire bonjour. Le robot reconnaît bien l'intention #HELLO (en vert) mais il n'est pas (encore) configuré pour répondre à cette intention. Il utilise donc le nœud par défaut (« anything_else ») pour répondre.

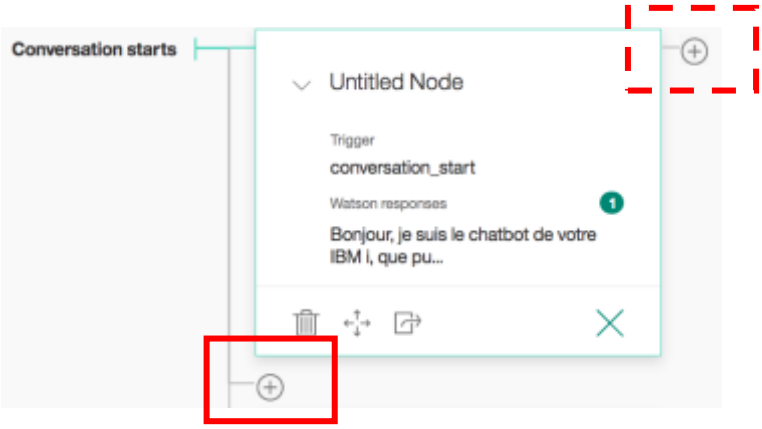


Notez, que ce panneau permet aussi d'entrainer Watson Conversation. Si #HELLO n'avait pas été la bonne intention, en cliquant sur la flèche du menu déroulant, nous aurions pu lui indiquer ce qu'il aurait dû reconnaître.

Section 4. Gestion des intentions

__ 1. Nous allons maintenant créer les nœuds permettant de gérer les intentions #HELLO et #SYSSTS.

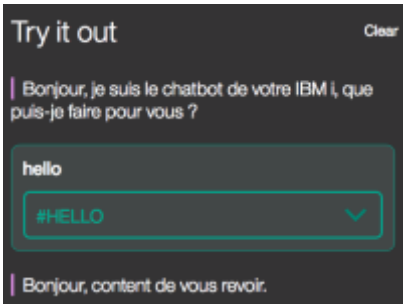
__ 2. Cliquez sur le premier node « **conversation_start** ».
Un + apparaît sous le nœud (pour le moment, ne tenez pas compte de celui apparu à droite). Cliquez sur le + du bas pour ajouter un nouveau nœud.



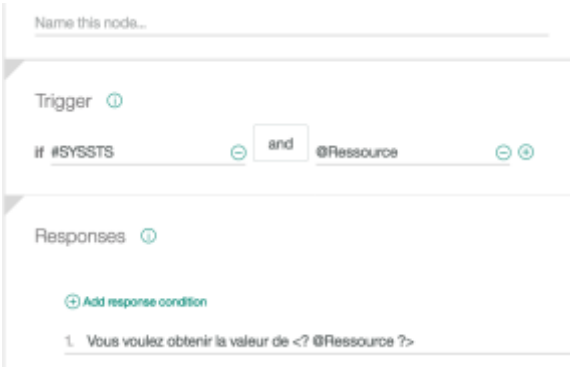
__ 3. Un panneau apparaît sur la droite.
Renseignez les différents champs de manière à obtenir.

Note : pour définir la condition d'exécution de ce nœud (« **trigger** »), entrez seulement « # » puis sélectionnez une intention.
Dans un dialogue, le préfixe # identifie toujours une intention.

__ 4. Testez votre dialogue. Vous devez maintenant obtenir une réponse pertinente lorsque vous lui dites bonjour.



__ 5. Gérons maintenant l'intention permettant d'obtenir la valeur courante d'une ressource système.
Pour cela nous devons identifier l'intention #SYSSTS et l'entité sur laquelle elle porte (CPU ou ASP).
Créez de la manière suivante, un nouveau nœud #SYSSTS, suivant le nœud gérant l'intention #HELLO (+ au bas du nœud #HELLO) :



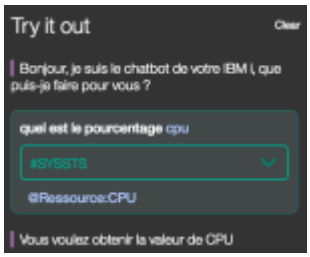
La condition d'exécution de ce nœud est :

- Que l'intention identifiée soit obtenir un taux d'utilisation d'une ressource (#SYSSTS)
- ET que la ressource demandée soit spécifiée dans la phrase, c'est à dire que la phrase contienne une entité @Ressource

Note : pour définir la condition d'exécution de ce nœud (« **trigger** »), entrez seulement « # » puis sélectionnez l'intention #SYSSTS. Cliquez sur le (+) pour ajouter une condition. Dans le nouveau champ, entrez « @ », puis sélectionnez dans la liste déroulante @Ressource.
Dans un dialogue, le préfixe @ identifie toujours une entité.

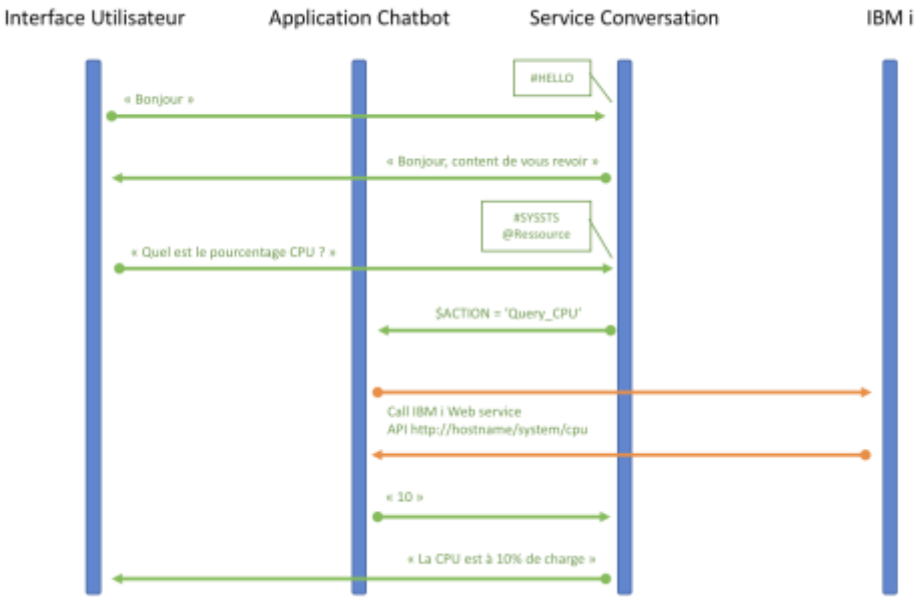
Dans le champ « `Enter a response` », saisissez une réponse, comme par exemple « `Vous voulez obtenir la valeur de <? @Ressource ?>` » (Cela ne sera utile que pour les tests préliminaires). Notez la manière d'écrire la variable entité à l'aide des balises « **<?** » et « **?>** ».

__ 6. Testez votre dialogue. Vous devez obtenir…



L'intention #SYSSTS et l'entité @Ressource ont bien été reconnues et gérées.

__ 7. A ce stade du dialogue, Conversation va devoir laisser la main à l'application chatbot afin qu'elle interroge l'IBM i pour obtenir le taux d'utilisation de la CPU (ou de l'ASP système).
Le diagramme de séquence sera donc :



Pour l'intention #HELLO, le service Conversation répond en langage naturel à l'utilisateur après avoir identifié l'intention. L'application chatbot (que vous développerez ensuite) n'est donc ici qu'un simple intermédiaire.

Pour l'intention #SYSSTS, le service Conversation, après avoir identifié l'intention, demande à l'application chatbot de retrouver la valeur de la CPU (et non de répondre à l'utilisateur).
Pour cela, vous initialiserez une variable dans le « **contexte** » de cette conversation ($ACTION). L'application chatbot devra tester cette variable pour savoir si il faut interroger l'IBM i et ce qu'il vaut récupérer comme valeur (en fonction de la valeur de la variable ACTION : cpu vs asp).
L'application chatbot retournera la valeur au service Conversation qui pourra alors formuler la réponse en langage naturel à l'utilisateur.

Notez que, pour le service Conversation, il n'y a pas de différence entre une réponse provenant d'un utilisateur ou du programme chatbot.

__ 8. Pour définir la variable ACTION qui sera interceptée et testée par le programme chatbot, vous allez éditer le contexte de la conversation.
Ce contexte est défini en format JSON. Il contient toutes les informations permettant au service Conversation d'être « **connection less** », c'est à dire qu'il n'y a pas de connexion permanente entre le programme chatbot et le service Conversation, mais des appels successifs.

Cliquez sur les 3 points à droite du message que vous avez défini, puis sélectionnez « **JSON** ».

Vous obtenez alors :

```
{
  "output": {
    "text": {
      "values": [
        "Vous voulez obtenir la valeur de <? @Ressource ?>"
      ],
      "selection_policy": "sequential"
    }
  }
}
```

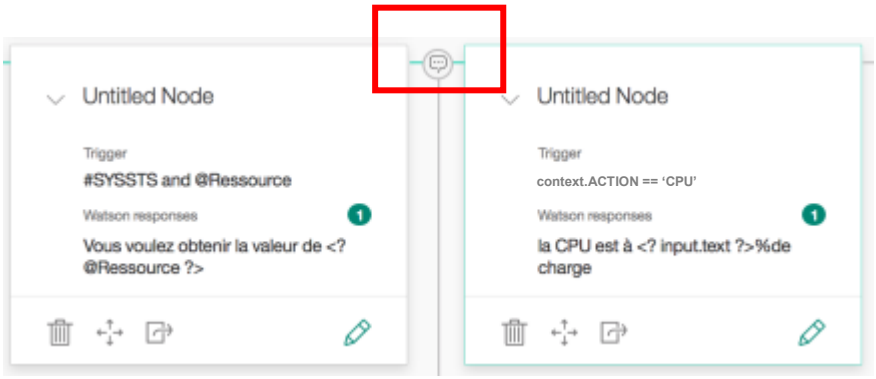
Après la première accolade, ajoutez le texte suivant :

```
"context": {
  "ACTION": " <? @Ressource ?>"
},
```

On sauvegarde dans la variable ACTION l'entité trouvée : elle déterminera l'API à appeler. Vous obtenez alors :

```
{
  "context": {
    "ACTION": "<? @Ressource ?>"
  },
  "output": {
    "text": {
      "values": [
        "Vous voulez obtenir la valeur de <? @Ressource ?>"
      ],
      "selection_policy": "sequential"
    }
  }
}
```

__ 9. Vous allez créer un nouveau nœud comme ci-dessous.

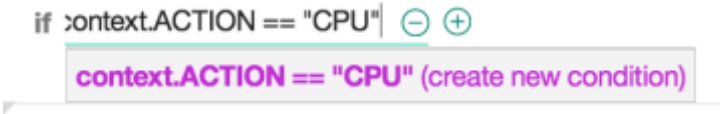


En cliquant sur le nœud précédemment créé (#SYSSTS and @Ressource) faites apparaitre un (+) en haut à droite du nœud. Cliquez dessus pour ajouter un nœud qui s'exécutera à la suite du nœud courant.

L'icône  , entre les 2 nœuds, représente une saisie utilisateur. Concrètement, cela revient à répondre au programme du chatbot, et à récupérer, de sa part, une réponse retournée par un utilisateur ou un programme.

Créez le nouveau nœud de la manière suivante :

La condition d'exécution sera « context.ACTION == "CPU" » = gérer la réponse du programme chatbot suite à la demande de la valeur CPU.
Tapez `context.ACTION == "CPU",` puis sélectionnez « `context.ACTION == "CPU" (create new condition)` ».



En passant en mode d'affichage JSON, tapez le texte suivant :

```
{
  "context": {
    "ACTION": null
  },
  "output": {
    "text": {
      "values": [
        "la CPU est à <? input.text ?>% de charge"
      ],
      "selection_policy": "sequential"
    }
  }
}
```

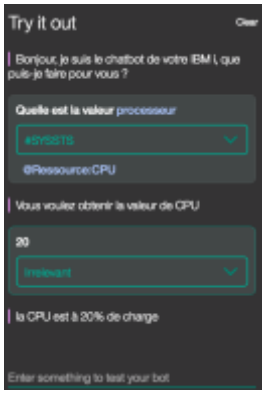
Cela permet de :

- Annuler la demande d'appel à l'IBM i puisque, à ce stade, elle vient d'être faite (ACTION : null)
- Répondre en générant un message incluant la valeur de la CPU retrouvée, « **<? input.text ?>** » (retourné par votre programme chatbot).

__ 10. En cliquant sur le nœud faites apparaitre un (+) en bas du nœud. Ajouter un nœud de la manière suivante :

Tous les nœuds de la même colonne forment un « **switch case** ». La condition « **true** » permet de faire une condition « toujours vraie » : le traitement par défaut. Dans cet exemple simple, si on n'a pas demandé la charge CPU, c'est que l'on a demandé l'occupation de l'ASP. La réalité serait plus probablement complexe…

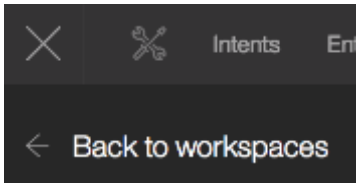
__ 11. Testez votre dialogue.
Après la première réponse de Conversation (« Vous voulez obtenir la valeur de CPU »), simulez la réponse du programme de chatbot en saisissant vous-même la valeur de la CPU telle quelle pourrait être retournée par l'API IBM i.
Vous obtenez alors :



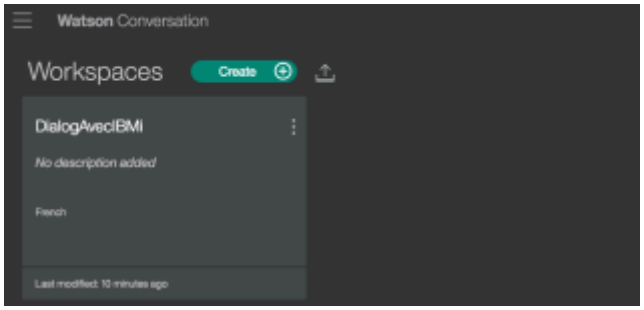
__ 12. Afin que le programme Chatbot puisse tester la variable ACTION pour chaque intention (besoin d'un appel d'une source de donnée externe ?), ajoutez la variable action dans le nœud #HELLO, afin d'obtenir :

Faites de même sur le nœud « **Conversation_start** ».

__ 13. Vous avez fini la configuration de votre premier dialogue.
Pour revenir sur la page d'accueil de l'outil de configuration de Conversation, cliquez sur le menu « burger » puis « **Back to workspace** ».



__ 14. Vous reviendrez sur cet espace dans l'exercice 2 pour retrouver l'identifiant de ce workspace (gardez cette page ouverte dans votre navigateur).



Vous allez maintenant créer l'interface utilisateur.

Information

Notez qu'à partir de ce menu « Burger », vous pouvez accéder à l'interface d'entrainement de Conversation, afin de le rendre plus efficace et pertinent dans l'identification des intentions et entités : « Improve » > « User conversations ».

2. Slack : création et configuration d'un team

Votre application de chatbot sera accessible au travers de Slack : l'interface utilisateur.

Slack est une plate-forme de communication collaborative propriétaire lancée en 2014.

Slack fonctionne à la manière d'un chat IRC organisé en canaux correspondant à autant de sujets de discussion. La plateforme permet également de conserver une trace de tous les échanges (« Slack » est l'acronyme de « Searchable Log of All Conversation and Knowledge »), permet le partage de fichiers au sein des conversations et intègre en leur sein des services externes comme GitHub, Box, …. Slack dispose de clients natifs sur la plupart des plateformes mobiles (iOS, Android, Windows Phone) ainsi que sur MacOS, Windows, Linux, et via un navigateur internet. La documentation en français est disponible ici : https://get.slack.help/hc/fr-fr.

Slack permet également de s'interface avec des programmes extérieurs. C'est ce mécanisme que vous utiliserez pour créer un « **bot Slack** », communicant avec votre programme chatbot, lui-même communiquant avec Watson Conversation.

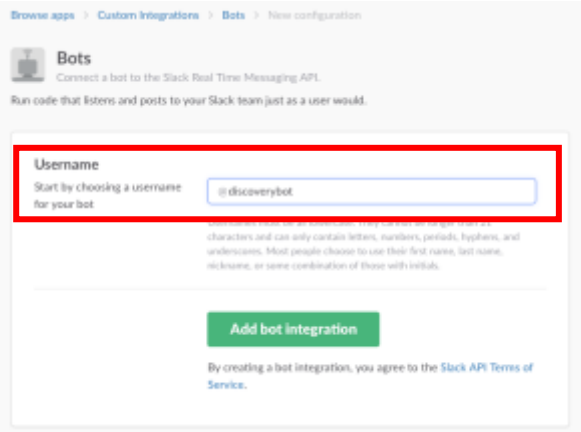
Section 1. Création d'un team Slack

__ 1. Ouvrez un nouvel onglet dans votre navigateur et créez un groupe Slack (https://slack.com/create) ou utilisez un groupe existant si vous avez suffisamment de privilèges. Référez-vous à la documentation en ligne pour savoir comment créer un groupe : https://get.slack.help/hc/en-us/articles/206845317-Create-a-Slack-team.

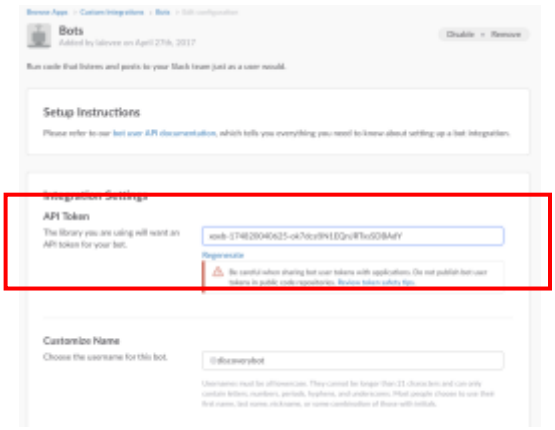
Section 2. Création d'un bot Slack

__ 1. Pour ajouter un Bot Slack, allez sur la page Slack de configuration des applications de votre groupe : https://<slack_group>.slack.com/apps/new/A0F7YS25R-bots

__ 2. Donner un nom à votre Bot Slack

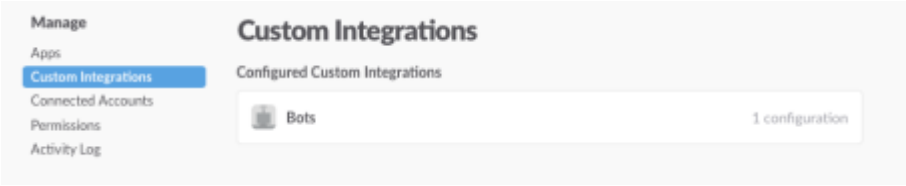


__ 3. Copiez le token (« **API token** ») généré. Vous en aurez besoin lors de la création de votre programme de chatbot.

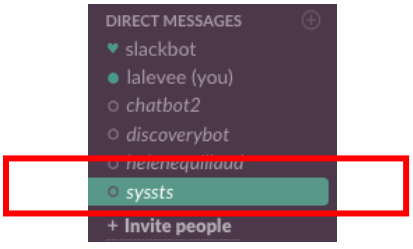


__ 4. Sauvegardez

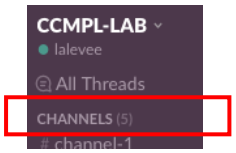
__ 5. Vous pouvez maintenant voir votre Bot Slack et sa configuration en vous rendant sur la page https://slack_group.slack.com/apps/manage, et en cliquant sur le menu de gauche : « **Custom Integrations** »



__ 6. Connectez-vous à votre groupe Slack : https://slack_group.slack.com/messages. Vous pouvez voir, dans le menu de gauche, votre nouveau Bot Slack comme un utilisateur « non connecté », (il apparaitra comme « connecté » lorsque nous aurons créé le programme chatbot).

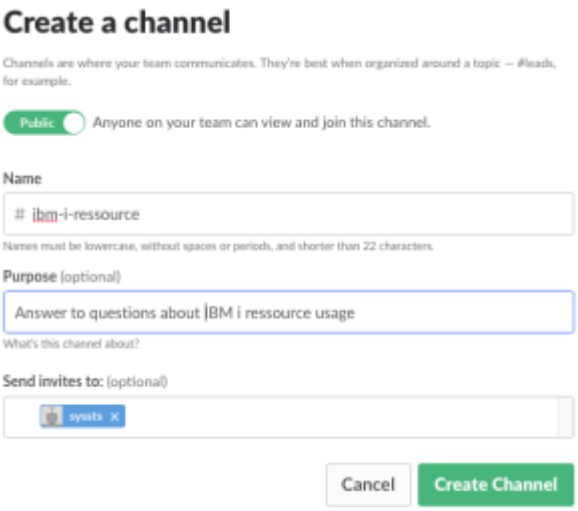


__ 7. Si vous souhaitez que votre Bot Slack réponde aux questions posées dans un Channel, créez un nouveau Channel (sinon, vous pourrez vous adresser directement à lui).
Cliquez sur « Channel » dans le menu Slack de gauche



__ 8. Sur la nouvelle page qui s'ouvre, cliquez sur…

__ 9. Saisissiez un nom pour votre Channel et invitez votre Bot Slack.



__ 10. Créez le Channel…

Votre Bot Slack est prêt à dialoguer. Vous allez maintenant créer le programme chatbot (« **application logic** ») pour faire la liaison entre le Bot Slack et le service Watson Conversation.

Information

Gardez la fenêtre ou l'onglet Slack ouvert dans votre navigateur. Vous y reviendrez dans la suite de cet exercice.

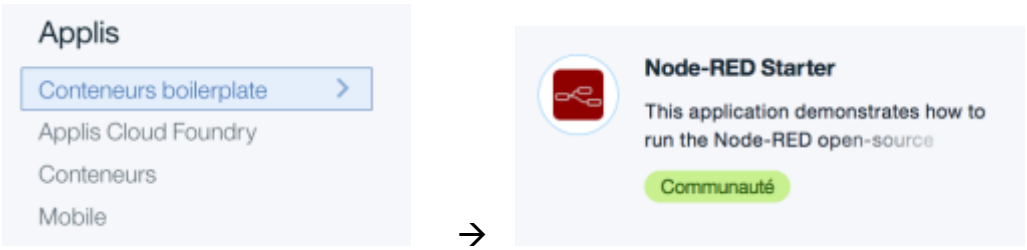
3. Bluemix : déploiement de Node-RED

Pour créer le programme de chatbot, vous allez utiliser Node-RED.

Node-RED est un logiciel initialement développé par IBM pour l'interfaçage de matériels, d'API et de services en ligne (« Cloud ») dans le cadre de l'Internet des objets. Il fournit un éditeur de flux, accessible via un navigateur, qui peut être utilisé pour créer des fonctions JavaScript. Les éléments des applications (« nodes »), déployés graphiquement, peuvent être sauvegardés ou partagés pour être réutilisés. Node-RED s'appuie sur Node.js.
En 2016, IBM a apporté Node-RED en tant que projet open source à la JS Foundation.

Section 1. Déploiement et exécution d'un boilerplate Node-RED

__ 1. Retournez sur l'environnement Bluemix, dans l'espace où vous avez déployé le service Conversation.
Affichez le catalogue de services, et sélectionnez « **Boilerplate** » dans la catégorie « **Applis** ». Cliquez sur « **Node-RED Starter** ».



Notes : Un boilerplate est une application prête à l'emploi, intégrant la partie code mais également les services nécessaires à son fonctionnement. Le boilerplate Node-RED est composé d'un runtime Node.JS et d'une base de données Cloudant. Consultez les plans de tarifications. Vous avez bien les plans de tarifications pour les 2 services inclus dans ce boilerplate.

__ 2. Donnez un nom à votre application Node-RED et créez-la :

__ 3. L'application va être construire (« **Build** ») puis déployée (« **Deploy** ») et enfin exécutée. Cela peut prendre quelques minutes.
Profitez-en pour vous familiariser avec les différents menus composant l'environnement de cette application. Consultez notamment les logs traçant le déploiement de l'application (Menu « **journaux** »).

__ 4. Une fois l'application déployée, cliquez sur le lien « **Visit App URL** ».

Cela va ouvrir un nouvel onglet contenant l'application Node-RED.
Si vous obtenez un message de la forme…

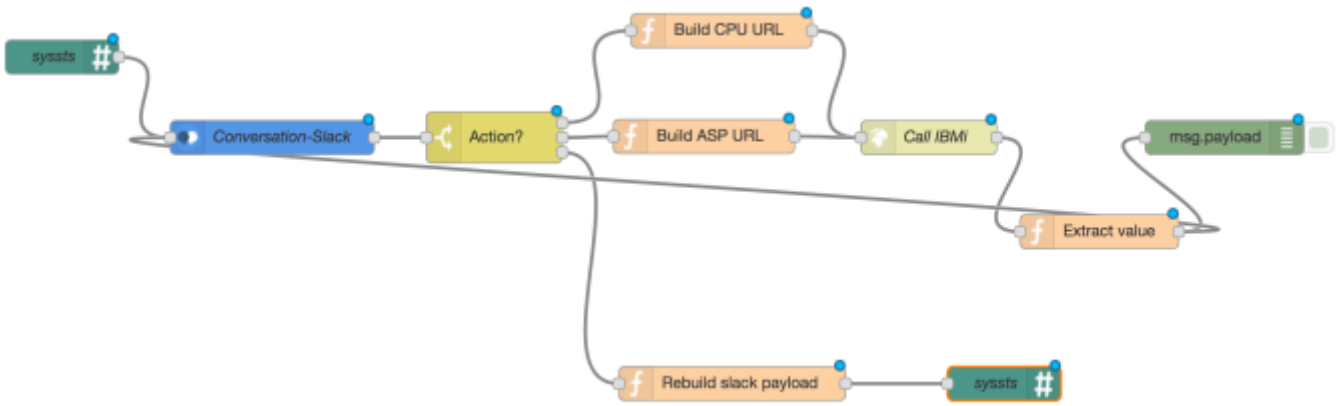
```
404 Not Found: Requested route ('chatbot4i-v2.eu-
gb.mybluemix.net') does not exist.
```

… c'est que l'application n'est pas encore déployée, toujours en cours de lancement.

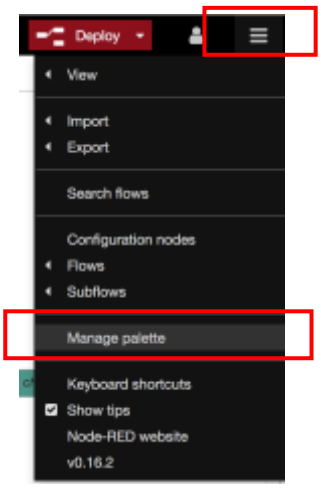
Section 2. Configuration de l'environnement Node-RED

__ 1. Configurez l'authentification Node-RED à l'aide de l'assistant puis cliquez sur « **Go to your Node-RED flow editor** » pour ouvrir l'éditeur de flux.

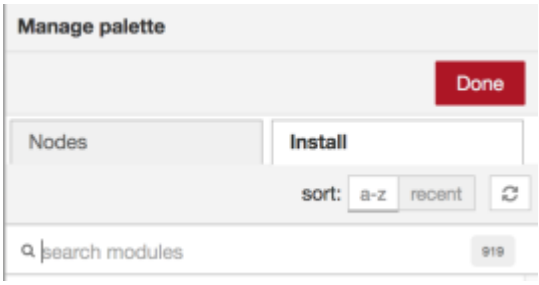
__ 2. Node-RED permet de construire des applications à l'aide d'un éditeur graphique, en connectant ensemble les blocs (ou nœuds) dont on a besoin. Il suffit simplement de glisser et déposer les blocs du menu de gauche (la palette) dans l'espace de travail au centre de l'écran et les connecter ensemble pour créer un nouveau flux : un programme.
Voici à quoi ressemblera le flux que vous allez créer (par la suite, nous reviendrons en détail sur les différents nœuds interconnectés) :



__ 3. La palette de gauche contient déjà les nœuds correspondant aux services Watson. Par contre, elle ne contient pas ceux permettant de se connecter à Slack. Vous allez donc l'enrichir.
Ouvrez le menu « burger » en haut à droite de votre fenêtre, et cliquez sur « **Manage palette** ».

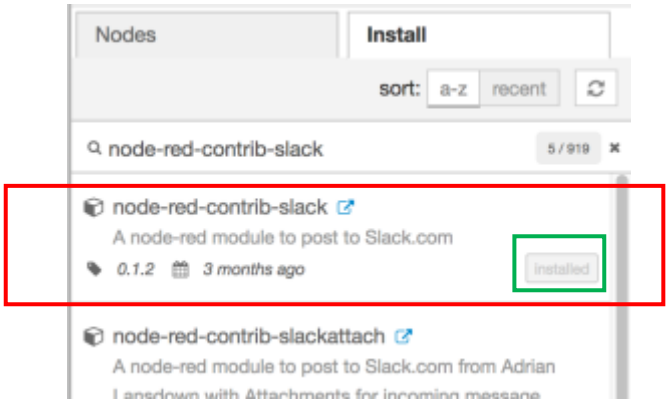


__ 4. Un panneau se déploie alors sur le côté gauche. Cliquez sur l'onglet « **Install** ».



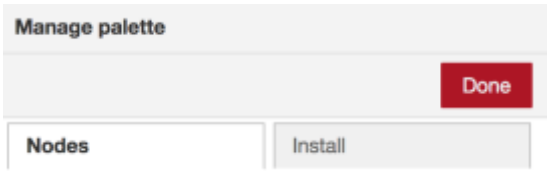
__ 5. Dans le champs « **search modules** », entrez le nom du module Slack recherché : `node-red-contrib-slack`.

__ 6. Une liste apparaît. Cliquez sur le bouton « **install** » en face du module désiré.



__ 7. Un message de mise en garde peut apparaître, indiquant qu'un redémarrage de l'application Node_RED peut être nécessaire.
Cliquez sur le bouton « **install** ».

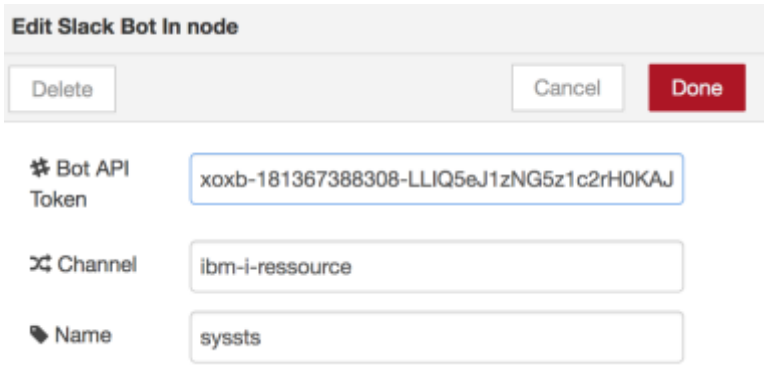
__ 8. Vous devez maintenant voir le module Slack installé disponible dans la liste des modules (onglet « **Nodes** »).
Fermez le panneau « **Manage Palette** » en cliquant sur le bouton « **Done** ».



Section 3. Création du flux – étape 1

__ 1. Dans la palette, retrouvez le nœud « **Slack Bot In** », et glissez-le sur l'espace de travail

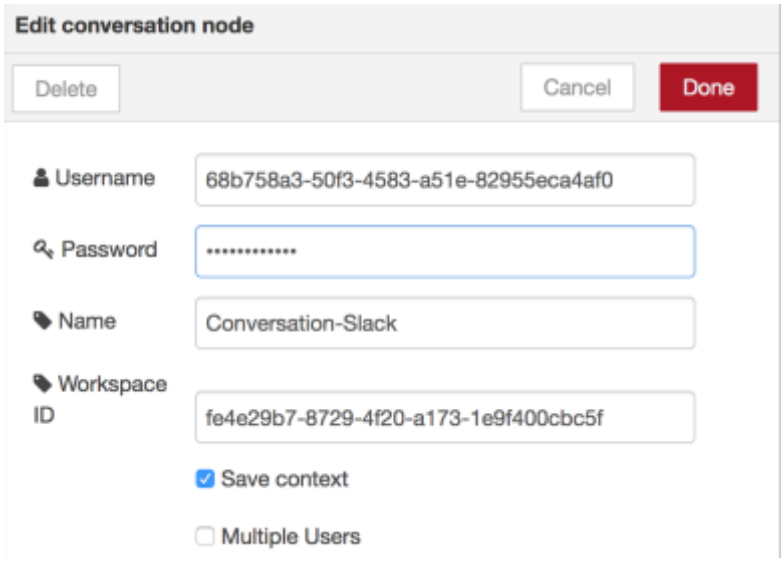
__ 2. Double cliquez dessus. Renseignez les champs « **Bot API Token** » et « **Channel** » définis à l'étape précédente. Indiquez également un nom du nœud.



Note : vous pouvez retrouver les informations liées à votre Bot Slack à partir de l'URL suivante : https://votre_slack_group.slack.com/apps/manage, puis « **Custom Integration** », « **Bots** », puis cliquez sur l'icône en fac votre bot Slack.

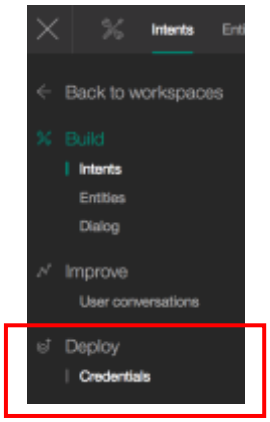
__ 3. Dans la palette, catégorie « **IBM_Watson** », retrouvez le nœud « **Conversation** », et glissez-le sur l'espace de travail. Créez un lien entre le nœud Slack et le nouveau nœud Conversation.

__ 4. Double cliquez sur le nœud Conversation et renseignez les paramètres demandés :

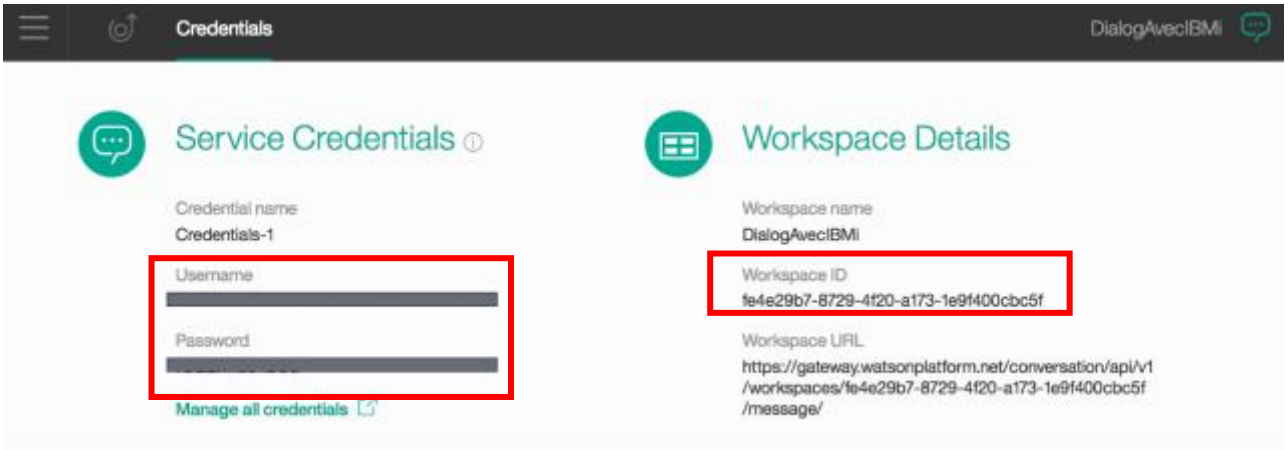


Notes :

__ a. Les « **Username** » et « **Password** » sont définis au niveau du service Conversation que vous avez créé à l'étape 1. Le « Workspace ID » est défini dans l'outil de configuration du service Conversation (utilisé à l'étape 2 de cet exercice). Vous pouvez retrouver tous ces éléments à partir de l'outil de configuration du service Conversation.
Ouvrez la page de votre navigateur correspondant à de l'outil de configuration du service Conversation. Cliquez sur le workspace ID pour l'ouvrir. A partir du menu « burger » de gauche, sélectionnez « **Deploy > Credentials** ».



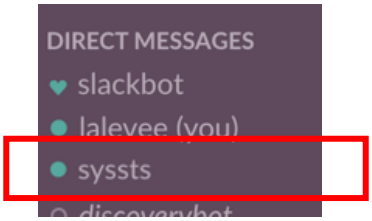
__ b. Copiez Username, Password et Workspace ID et collez les dans le noeud Node-RED Conversation.



__ 5. Cliquez sur le bouton « **Done** » pour valider la création de ce nouveau nœud Node-RED.

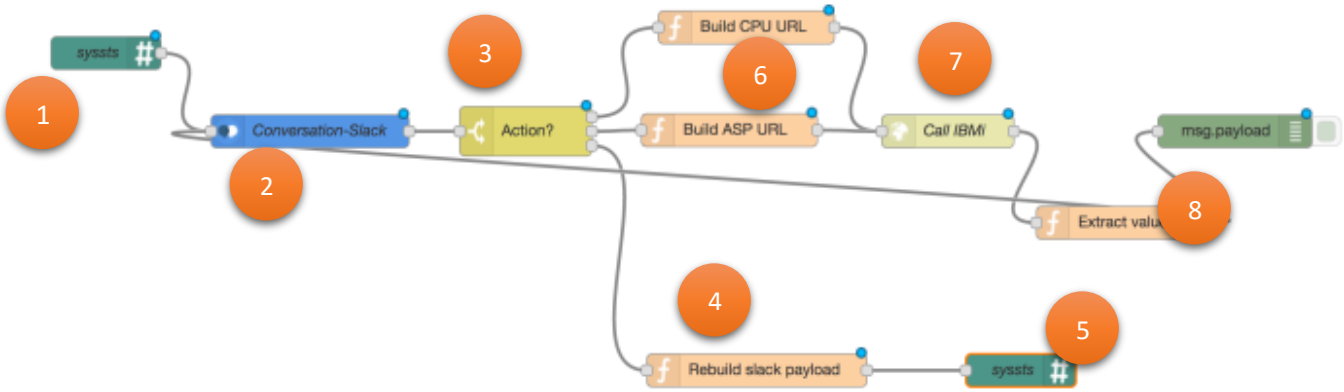
__ 6. Cliquez sur « **Deploy** » pour déployer votre Flux

__ 7. Si vous ouvrez la page de votre navigateur correspondant à votre Team Slack (ou groupe- Slack, vous devez maintenant voir l'utilisateur correspondant à votre Bot Slack connecté (point vert).



Section 4. Création du flux – étape 2

Nous allons maintenant créer les autres nœuds du flux.

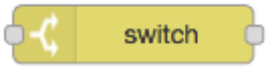


1. Récupération du texte saisi dans Slack (déjà créé)
2. Appel du service Conversation qui identifie les intentions, les entités et mène le dialogue (déjà créé à l'étape 1)
3. Test de la variable ACTION dans le contexte de la conversation en cours
   a. Si context.ACTION = CPU, construction et appel de l'API IBM i pour retrouver la charge CPU courante (6)
   b. Si context.ACTION = ASP, construction et appel de l'API IBM i pour retrouver l'occupation de l'ASP Système (6)
   c. Sinon, renvoi du texte généré par le service Conversation à l'utilisateur Slack (5)
4. Dans le contexte de la conversation en cours, extraction du texte à retourner à l'utilisateur Slack
5. Renvoi d'un texte à l'utilisateur de Slack (en fait, renvoi au Bot Slack, qui interagit avec les utilisateurs Slack)

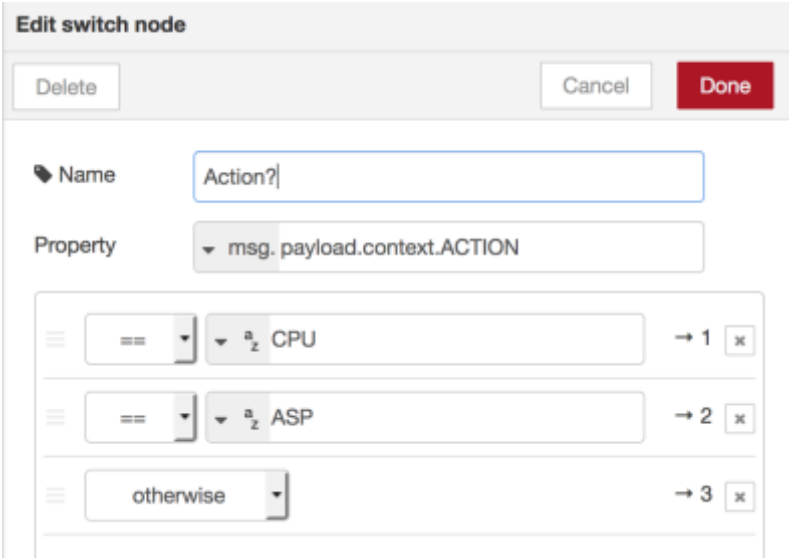
6. Selon le type d'information à retrouver sur l'IBM i (CPU ou ASP), construction des URLs d'appel de l'API sur l'IBM i
7. Appel de l'API IBM i et récupération du résultat (format JSON)
8. Extraction de la valeur du JSON et réponse au service Conversation

__ 1. Test de la variable ACTION

La variable ACTION que vous avez créée précédemment, se retrouve dans l'objet JSON « **msg** » échangé entre les nœuds Node-RED, dans l'objet « **payload** ». Vous allez donc pouvoir tester sa valeur.
A partir de la palette, glissez/déposez un nœud de type « **switch** » et reliez le à la sortie du nœud Conversation.



Configurez-le de la manière suivante



3 sorties seront créées pour ce nœud.

__ 2. A partir du contexte de la conversation en cour, extraction du texte à retourner à l'utilisateur Slack.

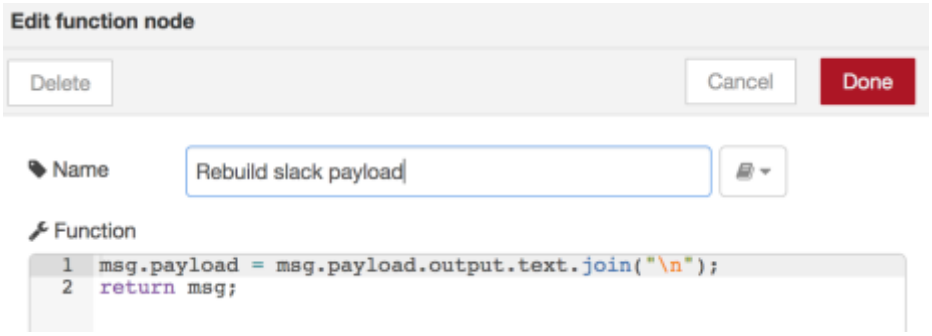
S'il n'y a aucune action à exécuter, il faut extraire et formater le texte (« **output.text** ») renvoyé par le service Conversation pour l'utilisateur Slack. Il se trouve dans l'objet « **msg.payload** » échangé entre les nœuds Node-RED.

A partir de la palette, glissez/déposez un nœud de type « **function** » et reliez-le à la 3e sortie du nœud précédent **« Action ? »**.

Copiez le texte suivant (code Javascript)…

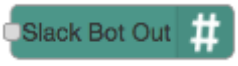
```
msg.payload = msg.payload.output.text.join("\n");
return msg;
```

… et donnez un nom au nœud de manière à obtenir :

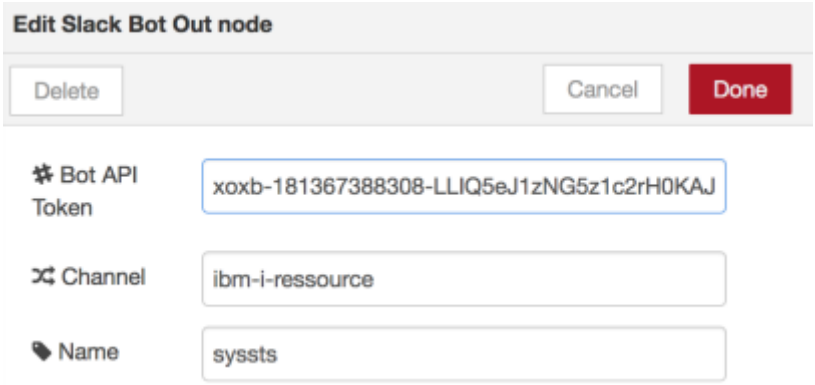


__ 3. Renvoi d'un texte à l'utilisateur Slack (en fait, renvoi au Bot Slack, qui interagit avec les utilisateurs Slack).

A partir de la palette, glissez/déposez un nœud de type « **Slack Bot Out** » et reliez-le à la sortie du nœud « **rebuild slack payload** ».



Configurez-le à l'aide des paramètres de votre Bot Slack (comme le nœud Slack précédent) de manière à obtenir :



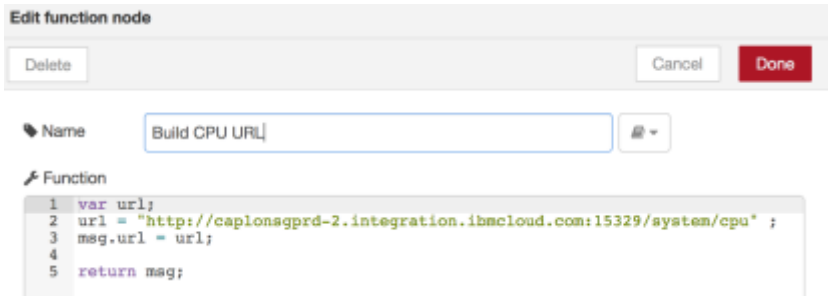
__ 4. Vous allez maintenant traiter les cas où il faut appeler une API REST sur l'IBM i pour obtenir les valeurs de CPU ou d'ASP.

__ a. Pour construire l'URL du web service permettant de retrouver la valeur courante de l'utilisation CPU, glissez/déposez un nœud de type « **function** » et reliez-le à la première sortie du nœud « **Action?** » (switch).

Ouvrez le nœud, donnez-lui un nom et copiez le code suivant :

```
var url;
url = "http://caplonsgprd-2.integration.ibmcloud.com:15329/system/cpu";
msg.url = url;
return msg;
```

Vous devez obtenir :



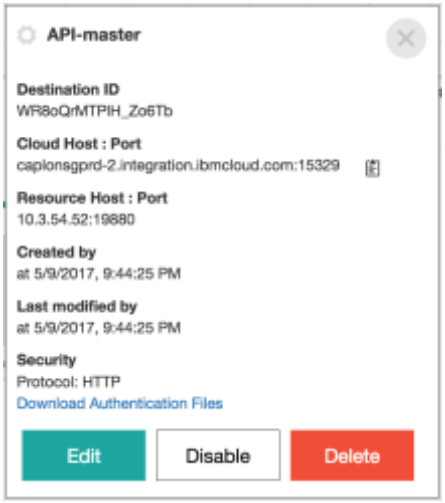
__ b. Pour construire l'URL du web service permettant de retrouver le taux d'utilisation de l'ASP système, glissez/déposez un nœud de type « **function** » et reliez-le à la deuxième sortie du nœud « **Action?** » (switch).

Ouvrez le nœud, donnez-lui un nom et copiez le code suivant :

```
var url;
url = "http://caplonsgprd-2.integration.ibmcloud.com:15329/system/asp";
msg.url = url;
return msg;
```

Vous devez obtenir :

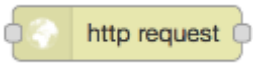
Note : l'URL utilisée ne correspond à pas aux hostname de l'IBM i car il n'est pas accessible à partir d'Internet. Vous utilisez ici la translation d'adresse (« NAT ») fournie par un service Bluemix d'intégration : Secure Gateway.
Sa configuration est la suivante :



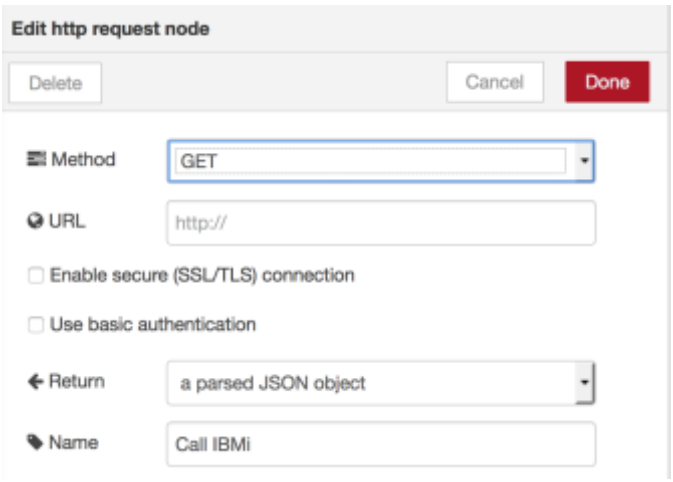
Pour plus d'information sur la Secure Gateway :
https://console.ng.bluemix.net/docs/services/SecureGateway/secure_gateway.html

__ 5. Appel de l'API IBM i et récupération du résultat (format JSON)

A partir de la palette, glissez/déposez un nœud de type « **http request** » et reliez-le aux sorties des nœuds « **Build CPU URL** » et « **Build ASP URL** ».



Ouvrez le nœud et configurez-le de manière à obtenir



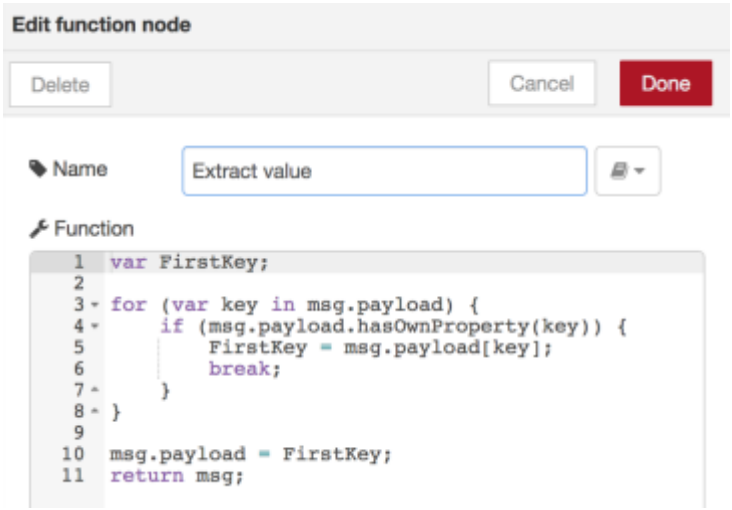
__ 6. Extraction de la valeur du JSON et réponse au service Conversation.

L'API IBM i renvoie un objet JSON (Exemple : `{ SYSTEM_ASP_USED: "27.48" })` mais nous devons renvoyer une valeur numérique au service Conversation. Nous allons donc l'extraire du JSON.
A partir de la palette, glissez/déposez un nœud de type « **function** » et reliez-le à la sortie du nœud « **Call IBMi** ».
Ouvrez le nœud et copiez le code suivant :

```
var FirstKey;
for (var key in msg.payload) {
   if (msg.payload.hasOwnProperty(key)) {
       FirstKey = msg.payload[key];
       break;
   }
}
msg.payload = FirstKey;
return msg;
```

Note : sans rentrer dans les détails, ce code permet d'extraire la valeur d'un objet JSON, composé d'une seul paire clé/valeur. Il est nécessaire car selon l'API appelée la clé de la paire n'est pas la même (`SYSTEM_ASP_USED` vs `ELAPSED_CPU_USED`).
Il faudrait également gérer le cas où l'API renvoie un code d'erreur… mais cela ne sera pas abordé dans cet exercice.

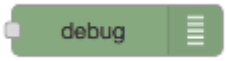
Vous devez obtenir :



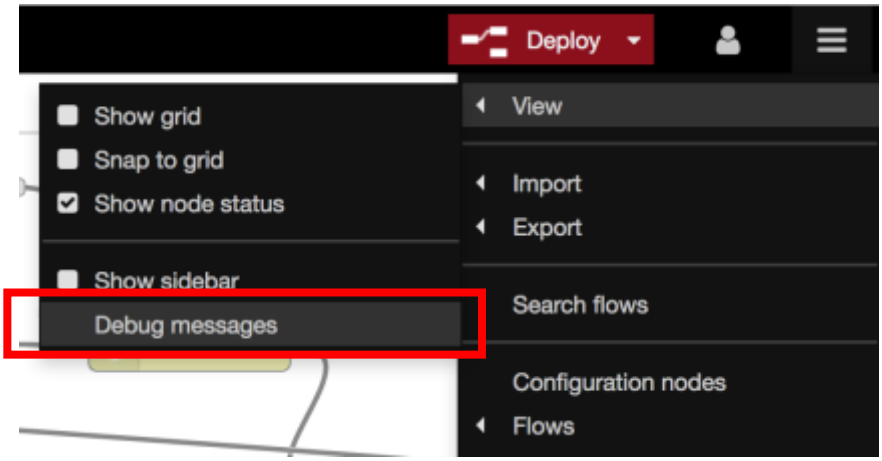
Connectez la sortie de ce nœud à l'entrée du nœud Conversation pour prendre en compte cette valeur comme une entrée utilisateur.

__ 7. Nous avons fini l'implémentation de notre flux.
Cependant, avant de le tester, nous allons voir comment ajouter une trace pour éventuellement debugger le flux.

A parti de la palette, glissez/déposez un nœud de type « **Debug**» et reliez-le à la sortie du nœud « **Extract Value?** ».



A partir du menu « burger » en haut à droite, afficher le panneau des messages de debug : lors de l'exécution, vous y verrez apparaître les messages en sortie du nœud « **Extract Value?** », et donc ce qui est renvoyé au nœud Conversation.

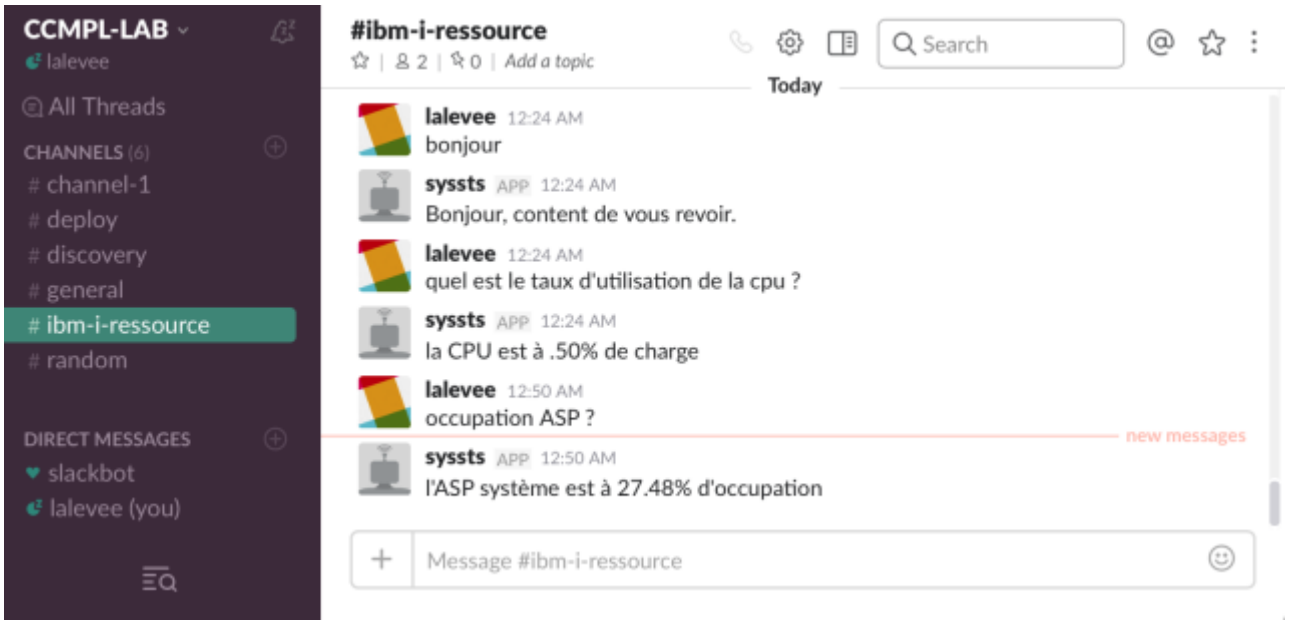


__ 8. Déployez votre flux

4. Slack : test du dialogue

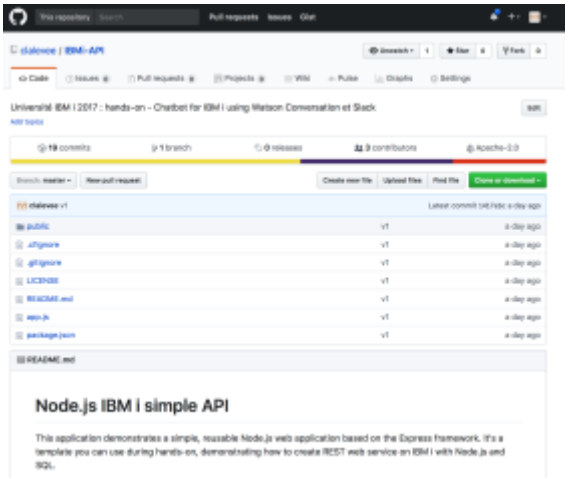
__ 1. Ouvrez la page de votre navigateur positionnée sur les messages Slack de votre team (https://slack_group.slack.com/messages)

__ 2. A partir du channel dédié, dialoguez avec votre chatbot !



5. Option - IBM i : création des APIs REST en Node.JS

__ 1. Ouvrez une nouvelle fenêtre dans votre navigateur, et entrez l'URL suivante : https://github.com/clalevee/IBMi-API.

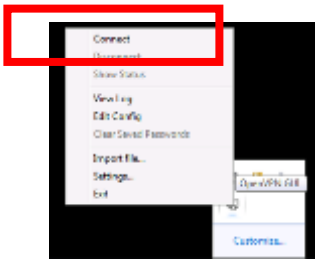


C'est ce « repository Git » que vous allez déployer sur l'IBM i.

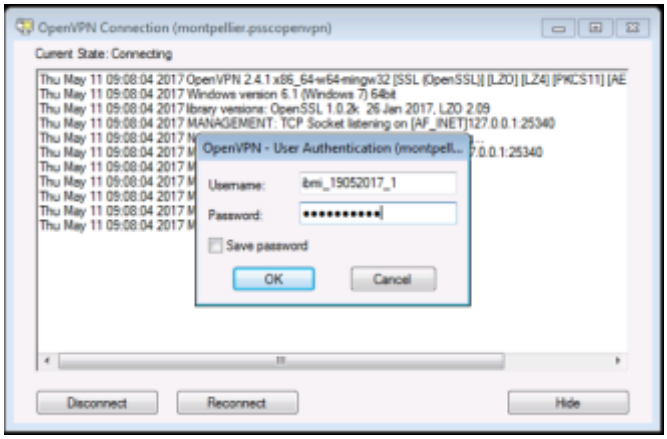
Section 1. Connection OpenVPN

__ 1. Connectez-vous au Datacenter hébergeant l'IBM i à l'aide du logiciel OpenVPN (icône dans la barre des taches Windows)

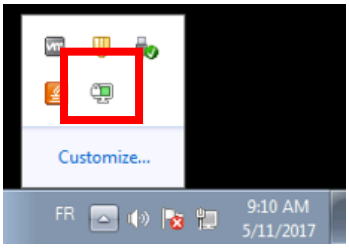
Click bouton droit puis « Connect »



Utiliser l'identifiant OpenVPN et le mot de passe attribués à votre team et donné en annexe 1 de ce document.



Une fois connecté, l'icône OpenVPN passe au vert

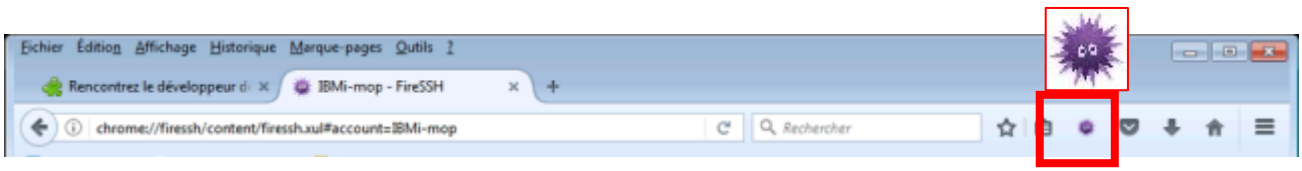


Section 2. Connexion ssh à l'IBM i et clonage du code (Git)

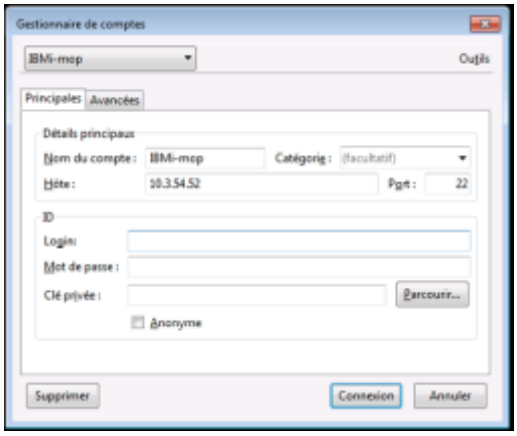
__ 1. Connectez-vous en ssh à l'IBM i.

Vous pouvez utiliser le plugin Firefox FireSSH ou Putty

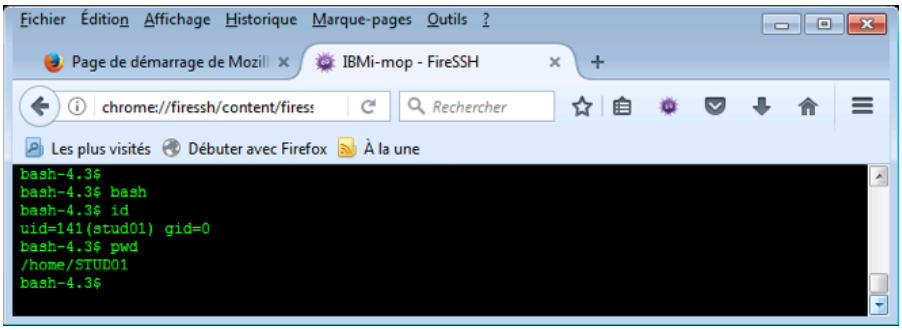
__ a. FireSSH : cliquez sur l'icône FireSSH dans la barre d'adresse de Firefox



Configurez l'adresse IP = 10.3.54.52 et votre identifiant = studxx (où xx est votre numéro de team). Le mot de passe est « password ».



Cliquez sur le bouton « Connexion » : vous êtes connecté.



__ b. Putty : cliquez sur l'icône Putty se trouvant sur le bureau et configurez une nouvelle connexion ssh comme ci-dessous

- Adresse IP : 10.3.54.52
- Identifiant : studxx (où xx est votre numéro de team)
- Mot de passe : password

__ 2. Dans la session ssh, si vous le souhaitez, lancez un shell bash pour un environnement plus… « user-friendly ».

```
$ bash
bash-4.3$
```

__ 3. Positionnez-vous dans le répertoire « **/home/orion** », puis, à l'aide de la commande Git ci-dessous, clonez le projet clalevee/IBMi-API dans le répertoire studxx (où xx est votre numéro de team).

```
$ cd /home/orion
$ git clone -c http.sslVerify=false https://github.com/clalevee/IBMi-API
stud01
 Cloning into 'stud01'...
 remote: Counting objects: 81, done.
 remote: Compressing objects: 100% (51/51), done.
 remote: Total 81 (delta 31), reused 69 (delta 19), pack-reused 0
 Unpacking objects: 100% (81/81), done.
```

La syntaxe de la commande est :

```
git clone URL_Source Répertoire_Local_Cible
```

(`-c http.sslVerify=false` est là pour gérer un problème de certificat IBM i auto-signé…).

__ 4. Positionnez-vous dans le répertoire /home/orion/stuxx (où xx est votre numéro de team), puis, à l'aide de la commande **npm** ci-dessous, installez les dépendances du programme Node.js (les dépendances sont déclarées dans le fichier package.json).

```
$ cd stud01
$ npm install
 express@4.13.4 node_modules/express
 ├── escape-html@1.0.3
 ├── array-flatten@1.1.1
 ├── utils-merge@1.0.0
 . . . .
  ├──── accepts@1.2.13 (negotiator@0.5.3, mime-types@2.1.15)
  └──── serve-static@1.10.3 (send@0.13.2)
$
```

__ 5. Afin que vous puissiez éditer à l'aide de l'éditeur Web Orion, les fichiers se trouvant dans ce répertoire, vous allez donner les droits au profil QTMHHTTP

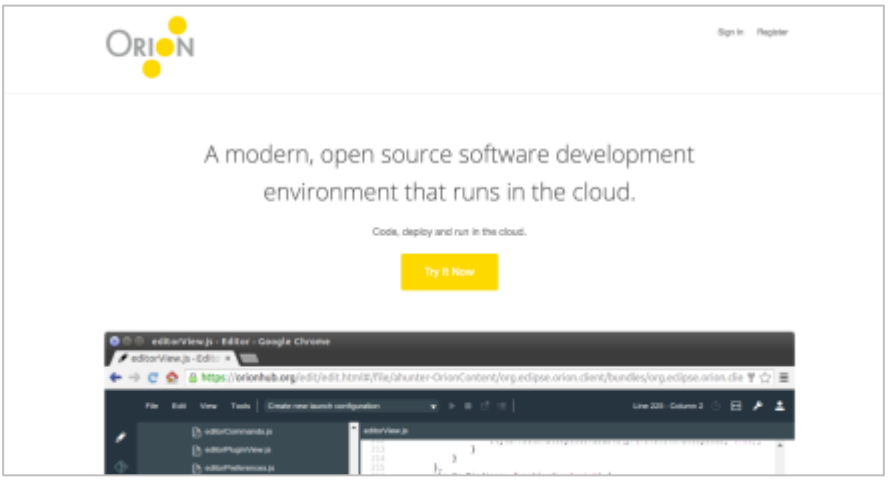
```
 $ system "CHGAUT OBJ('/home/orion/stud01') USER(QTMHHTTP) DTAAUT(*RWX)
OBJAUT(*ALL) SUBTREE(*ALL)"
```

Section 3. Modification du code Node.js

__ 1. Vous allez maintenant utiliser Orion pour customiser le code Node.js cloné.
Orion est une application en ligne de la fondation Eclipse dont le but est de fournir un environnement de développement intégré en mode hébergé, permettant, par exemple, de développer directement sur le cloud. Le but que s'est fixé l'équipe développant Orion est de permettre le développement d'applications web, sur le web (https://wiki.eclipse.org/Orion).
Dans notre cas, l'application est installée sur l'IBM i (5733OPS, Opt. 8).

Note : si vous ne souhaitez pas utiliser Orion, vous pouvez également utiliser Notepad++ à partir de votre poste de travail. Pour cela, consulter l'annexe 2 de ce document.

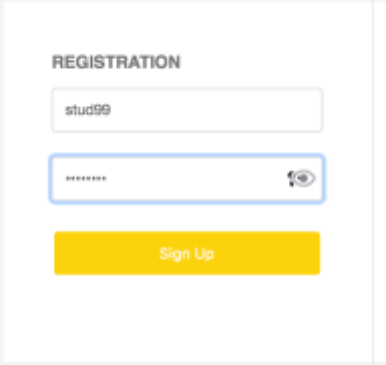
Dans une nouvelle fenêtre de votre navigateur, connectez-vous à l'URL suivante : http://10.3.54.52:2025



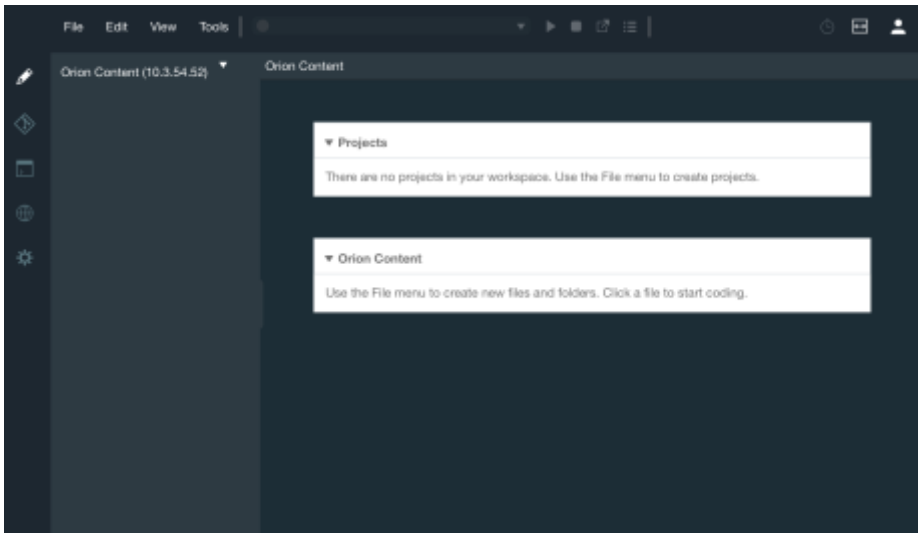
__ 2. Cliquez sur le bouton

__ 3. Entrez un identifiant identique à votre identifiant IBM i (studxx) et un mot de passe contenant un chiffre et un caractère spécial.

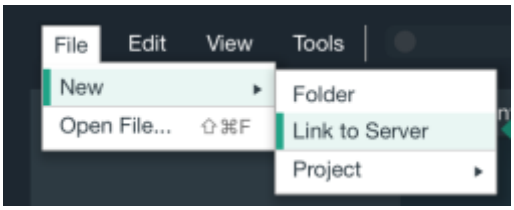
Note : Les identifiants Orion ne sont pas les *USRPRF de l'IBM i. De plus, nous n'avons pas limité la création de compte aux seuls administrateurs. C'est pourquoi vous pouvez vous-même vous créer votre identifiant Orion.



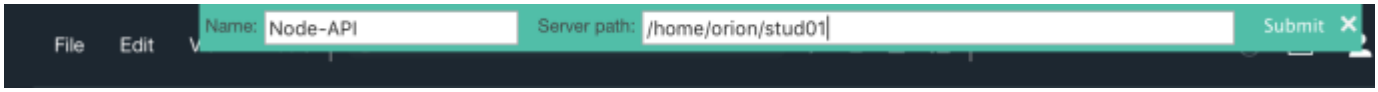
__ 4. L'éditeur s'ouvre



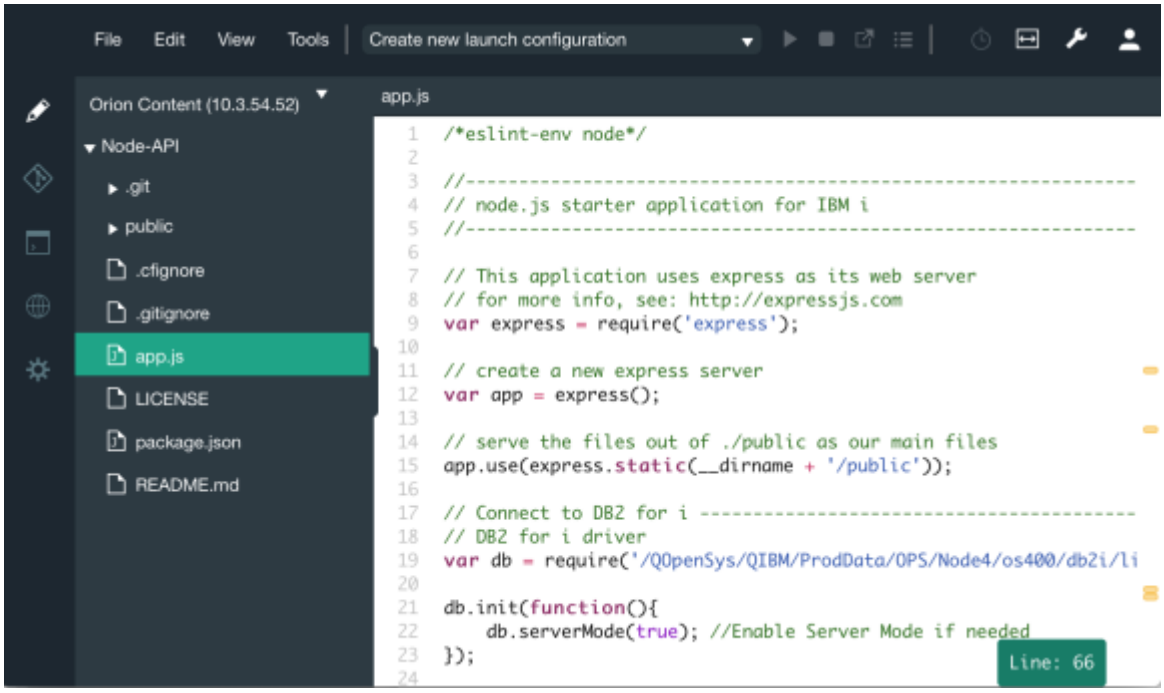
__ 5. A partir du menu « **File** », ouvrez le répertoire que nous venons de cloner : « **New** » > « **Link to Server** »



__ 6. Donnez un nom au nouveau projet « **Node-API** » et initialisez le « **Server path** » avec le chemin du répertoire dans lequel vous venez de cloner le repository Git : « **/home/orion/studxx** » (où xx est votre numéro de team).



__ 7. Vous pouvez maintenant éditer le fichier « app.js »



__ 8. Retrouvez le code du web service #1

```
// Web Service #1 ------------------------------------------------------

app.get('/system/cpu', function(req, res, next) {
    // replace following line with you Web Service code
    res.json(204);
});
```

… et modifier le de la manière suivante

```
// Web Service #1 ------------------------------------------------------------

app.get('/system/cpu', function(req, res, next) {
    var result = {};
    var sql = "SELECT elapsed_cpu_used FROM QSYS2.SYSTEM_STATUS_INFO";

    try {
        console.log("SQL: " + sql);
        db.exec(sql, function(rs) {
                console.log (JSON.stringify(rs));
                        if(rs.length != 0) {
                            res.json(rs[0]);
                        } else {
                            res.json(204);
                        }
        });
     } catch(e) {  // Exception handler
        console.log("Error: " + e);
        res.json(500);
     }
});
```

Analysez ce code qui expose une API REST (méthode GET) permettant de retrouver le pourcentage processeur utilisé, via une simple requête SQL, et, de le renvoyer format JSON.
Retrouvez en début de programme, ce qui permet d'exécuter du SQL sur DB2 for i à partir de Node.js.

__ 9. Retrouvez le code du web service #2

```
// Web Service #2 ------------------------------------------------------

app.get('/system/asp, function(req, res, next) {
    // replace following line with you Web Service code
    res.json(204);
});
```

et modifier le de la manière suivante

```
// Web Service #2 ------------------------------------------------------------

app.get('/system/asp, function(req, res, next) {
    var result = {};
```

```
    var sql = "SELECT system_asp_used FROM QSYS2.SYSTEM_STATUS_INFO";

    try {
        console.log("SQL: " + sql);
        db.exec(sql, function(rs) {
                console.log (JSON.stringify(rs));
                        if(rs.length != 0) {
                            res.json(rs[0]);
                        } else {
                            res.json(204);
                        }
        });
     } catch(e) {  // Exception handler
        console.log("Error: " + e);
        res.json(500);
     }
});
```

Analysez ce code qui expose une deuxième API REST permettant de retrouver le pourcentage d'ASP système utilisé, via une simple requête SQL, et, de le renvoyer au format JSON.

__ 10. Vous allez maintenant modifier le port TCP sur lequel se mettra à l'écoute votre programme.

```
// hereunder, replace 19880 by TCP port you want to use
var ServerPort = 19880;
```

Remplacez le numéro de port par 19880 + votre numéro de team (19881 pour le team 1, 19882 pour le team 2, etc…).

Section 4. Exécution du programme Node.js et test

__ 1. Retournez à votre interface SSH (Firefox FireSSH ou Putty) et positionnez-vous dans le répertoire de votre application

```
$ cd /home/orion/stud01
$
```

__ 2. Exécutez le programme à l'aide la commande « **npm start** » (ou « **node app.js** »)

```
$ npm start
> Nodejs4iStarterApp@0.0.1 start /home/orion/stud01
> node app.js
DB2 init done
DB2 connect done
Server starting on 19881
```

__ 3. Ouvrez une nouvelle fenêtre de votre navigateur et saisissez l'URL suivante : http://10.3.54.52:<votre_numero_de_port>/system/cpu.

Vous devez obtenir le résultat suivant

De la même manière, testez la deuxième API. Quelle est son URL ?

Vous avez terminé la mise en œuvre des 2 REST APIs, en Node.js, sur IBM i.

6. Conclusion

Félicitation !

Vous avez terminé cet exercice.

L'étape suivante serait la configuration du service d'intégration Bluemix, la Secure Gateway, afin de prendre en compte la translation du nouveau port TCP que vous avez mis en œuvre. Mais, cela ne fait pas partie de cet exercice.
Parlez-en avec l'instructeur si vous voulez plus d'informations.

7. Annexe 1 : identifiants OpenVPN

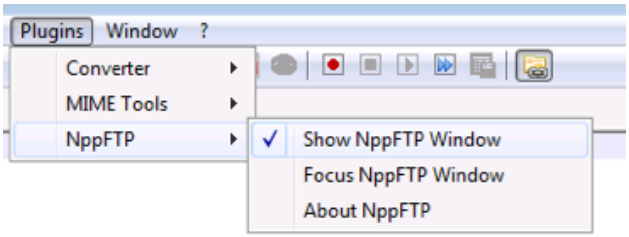
{ expired }

8. Annexe 2 : Utilisation de Nodepad++

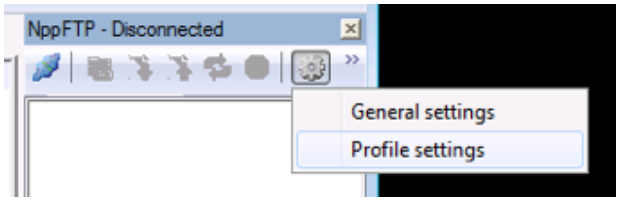
Si vous ne souhaitez pas utiliser Orion, pour pouvez utiliser le logiciel Windows Nodepad++ installé sur votre poste de travail, et configuré avec le plugin NppFTP, permettant l'édition de fichiers à distance.

__ 1. Ouvrez le programme Nodepad++ à partir du menu Windows.

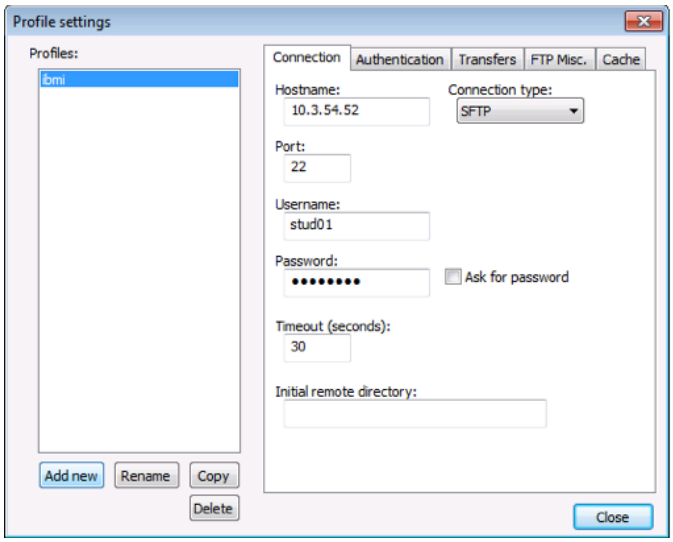
__ 2. A partir du menu « **Plugins** » > « **NppFTP** » > « **Show NppFTP Windows** »…



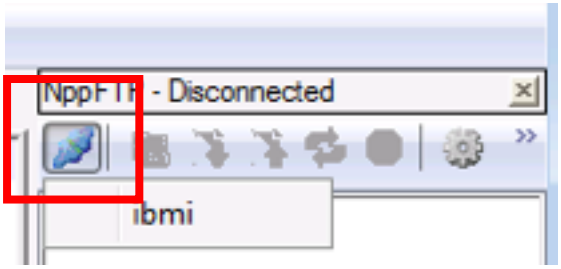
… ouvrez la fenêtre de configuration des profils : « **Profile settings** »



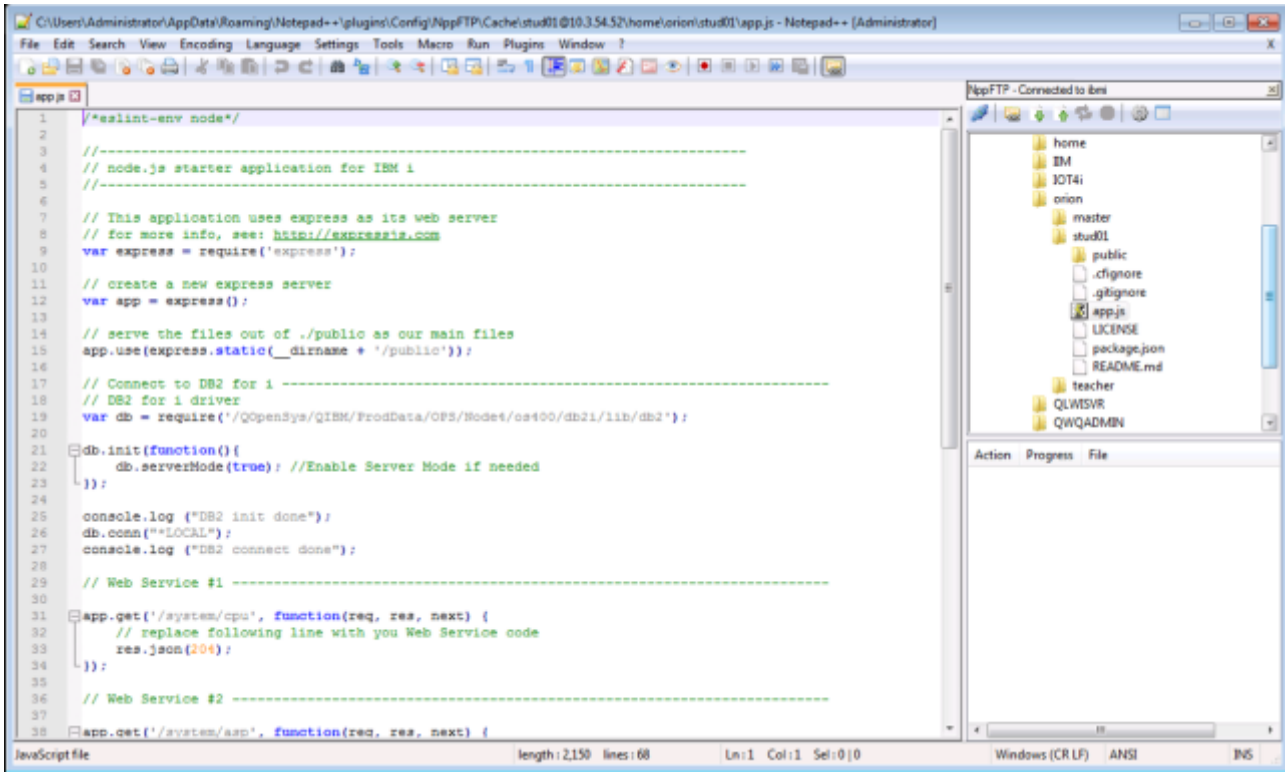
… et configurez la connexion à l'IBM i de la manière suivante. Fermez cette fenêtre.



__ 3. Vous pouvez maintenant vous connecter à l'IBM i



__ 4. A partir de l'explorer de droite, sélectionnez le fichier à éditer.



9. Annexe 3 : Code source programme Node.js IBM i

```
//-------------------------------------------------------------------------
// node.js starter application for IBM i
//-------------------------------------------------------------------------

// This application uses express as its web server
// for more info, see: http://expressjs.com
var express = require('express');

// create a new express server
var app = express();

// serve the files out of ./public as our main files
app.use(express.static(__dirname + '/public'));

// Connect to DB2 for i ----------------------------------------------------

// DB2 for i driver
var db = require('/QOpenSys/QIBM/ProdData/OPS/Node4/os400/db2i/lib/db2');

db.init(function(){
        db.serverMode(true); //Enable Server Mode if needed
});

console.log ("DB2 init done");
db.conn("*LOCAL");
console.log ("DB2 connect done");

// Web Service #1 ----------------------------------------------------------

app.get('/system/cpu', function(req, res, next) {
    var result = {};
    var sql = "SELECT elapsed_cpu_used FROM QSYS2.SYSTEM_STATUS_INFO";

    try {
        console.log("SQL: " + sql);
        db.exec(sql, function(rs) {
                console.log (JSON.stringify(rs));
                        if(rs.length != 0) {
                                res.json(rs[0]);
                        } else {
                            res.json(204);
                        }
        });
     } catch(e) {  // Exception handler
        console.log("Error: " + e);
        res.json(500);
     }
});
```

```
// Web Service #2 ------------------------------------------------------------

app.get('/system/asp', function(req, res, next) {
    var result = {};
    var sql = "SELECT system_asp_used FROM QSYS2.SYSTEM_STATUS_INFO";

    try {
        console.log("SQL: " + sql);
        db.exec(sql, function(rs) {
                console.log (JSON.stringify(rs));
                        if(rs.length != 0) {
                                res.json(rs[0]);
                        } else {
                            res.json(204);
                        }
        });
     } catch(e) {  // Exception handler
        console.log("Error: " + e);
        res.json(500);
     }
});

// ---------------------------------------------------------------------------

// hereunder, replace 19880 by TCP port you want to use
var ServerPort = 19880;

// start server on the specified port and binding host
app.listen(ServerPort, '0.0.0.0', function() {
  // print a message when the server starts listening
  console.log("Server starting on " + ServerPort);
});

// Handle exit events --------------------------------------------------------

process.on('SIGINT', function () {
  console.log('SIGINT fired')
  process.exit(1)
});

process.on('exit', function () {
  console.log('Exit fired');
  console.log ("Close DB connection...");
  db.close();
});
```