

ORACLE®

# Oracle Digital Assistant

## The Complete Training

### Custom Component Debugging

# Safe Harbor Statement

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

# Topic agenda

- 1 ➤ Debugging vs. logging
- 2 ➤ Debugging architecture
- 3 ➤ Debugging practices
- 4 ➤ Debugging with MS Visual Studio Code

# Topic agenda

- 1 Debugging vs. logging
- 2 Debugging architecture
- 3 Debugging practices
- 4 Debugging with MS Visual Studio Code

# Logging

- Logging is good to find problems that occur at runtime
  - Digital Assistant version 19.1.3 and later provide a diagnostic panel for displaying logs for locally deployed custom components
  - Mobile Hub and 3<sup>rd</sup> party Node containers have their own consoles for displaying logs
- Logging is not efficient enough for finding problems at design time
  - Not all problems are exceptions
    - Custom component doesn't render a response
    - Values returned by a custom component don't show in skill
    - Code logic is not getting executed
    - Broken dialog flow navigation after custom component was added

# Debugging

- Monitors bot-component interactions
  - Access to bot message payload
  - Insights to variable states
  - Steps through code execution
- Use your JavaScript IDE of choice
  - E.g. MS Visual Studio Code, JetBrains Webstorm
- Run custom component service from your local computer
  - Connect from Oracle Digital Assistant skill
- If it works, then it continues working after deployment

# Before you can debug custom component services

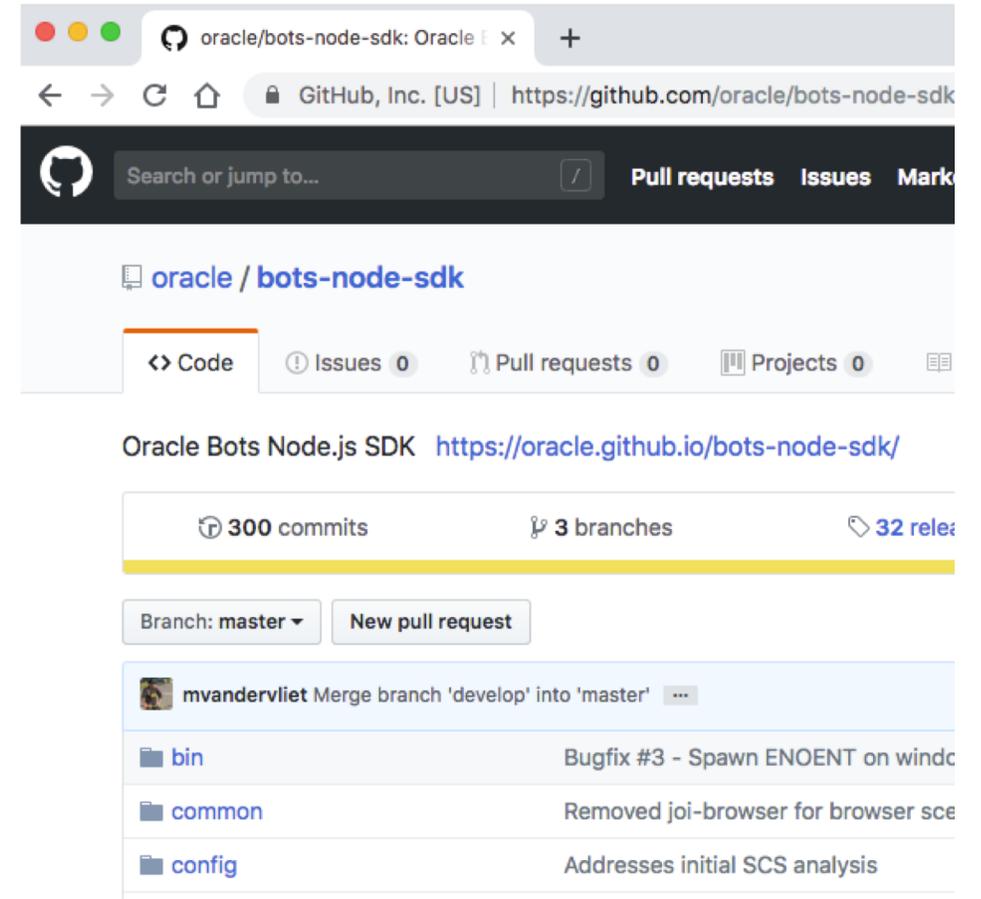
- Create local environment
  - Local runtime for custom component REST service
  - Install Oracle Bots Node.js
    - Command line to create custom component services
    - Includes local runtime
- Make local machine accessible from the Internet
  - Oracle Digital Assistant needs to access local component service runtime URL
- Have JavaScript IDE installed that supports Node debugging

# Topic agenda

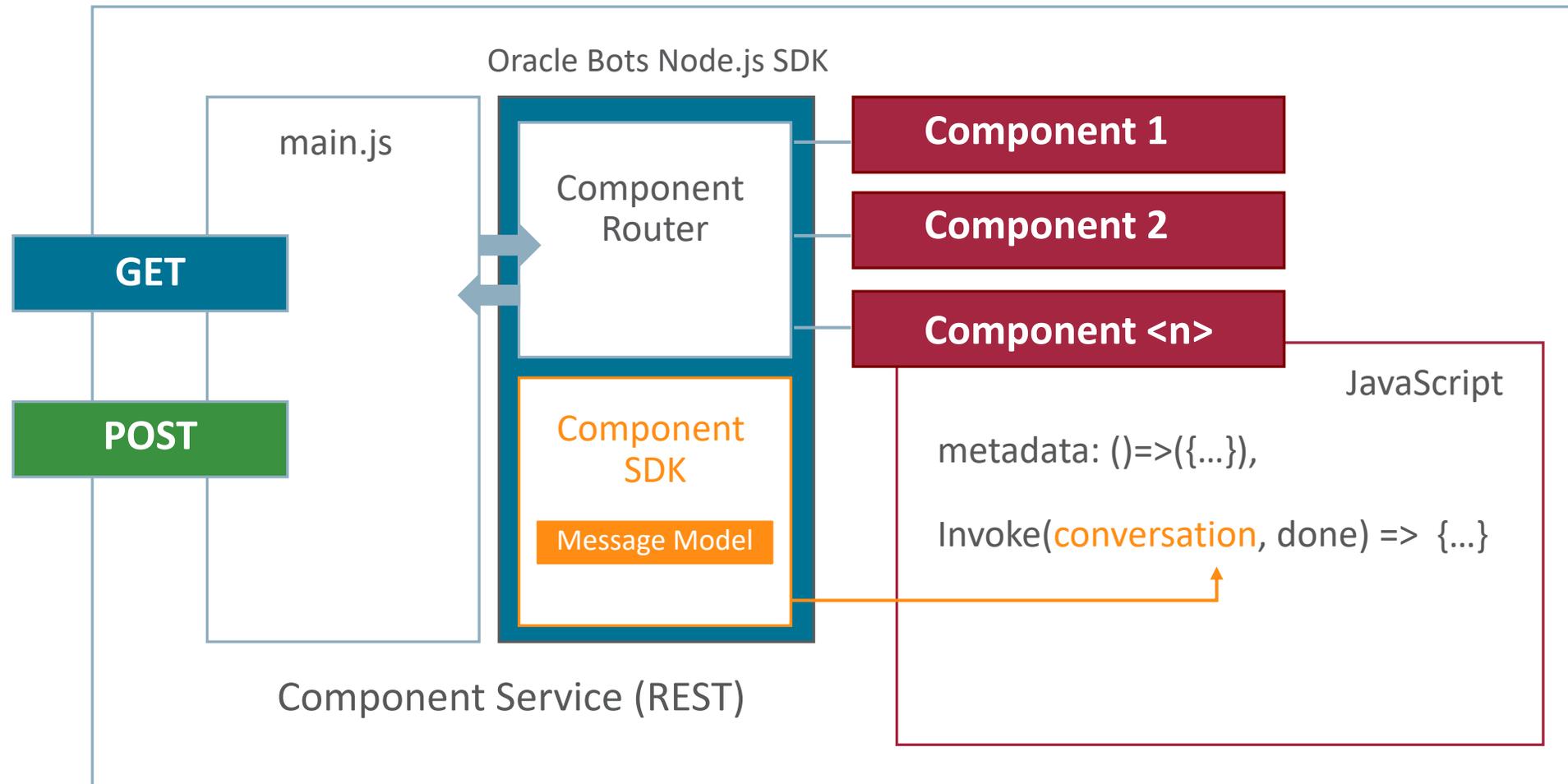
- 1 Debugging vs. logging
- 2 Debugging architecture**
- 3 Debugging practices
- 4 Debugging with MS Visual Studio Code

# About Oracle Bots Node.js SDK

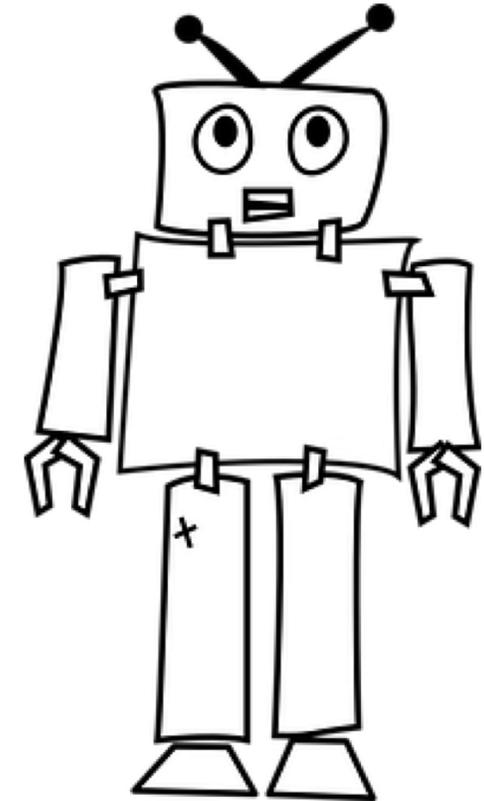
- Provides local environment for
  - Component development
  - Component debugging
  - Packaging for deployment
- NPM installable
- Command line to
  - Create custom component service project
  - Create custom components
- Adds bots custom component SDK



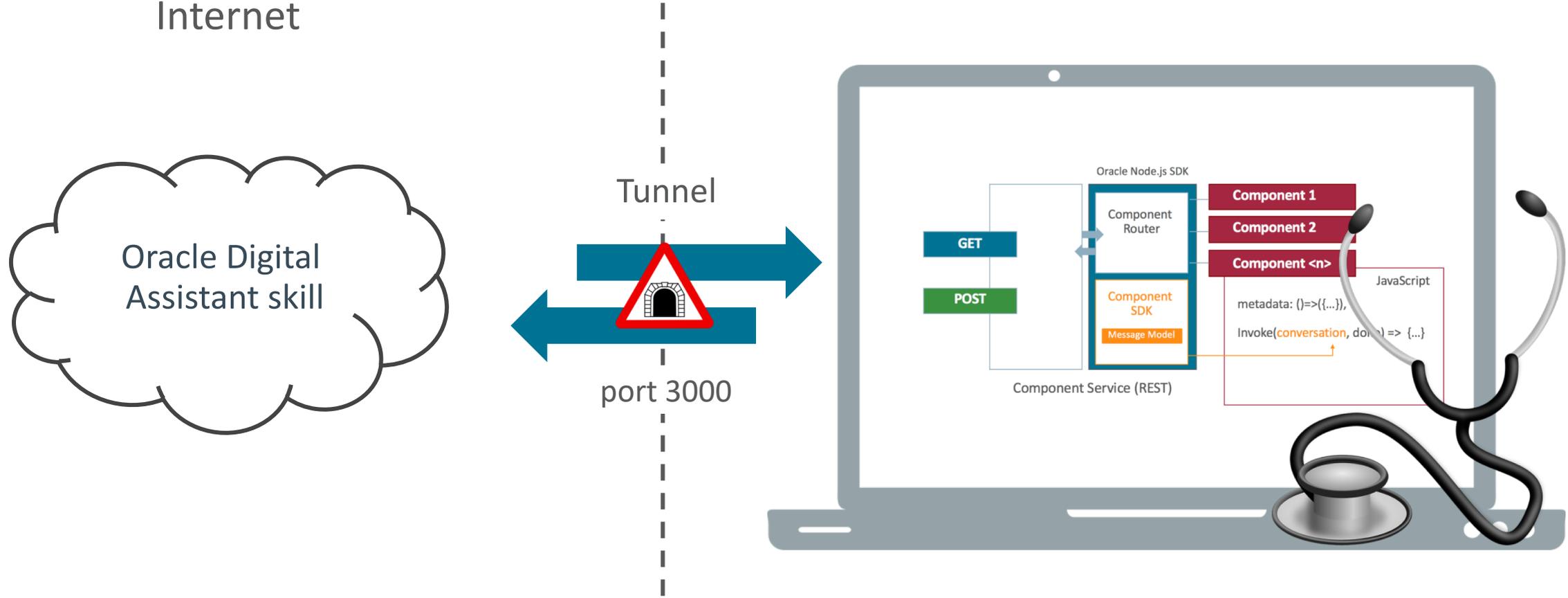
# Oracle Bots Node.js SDK component service architecture



For debugging, you **local machine must be accessible from** Oracle Digital Assistant running on **the Internet**

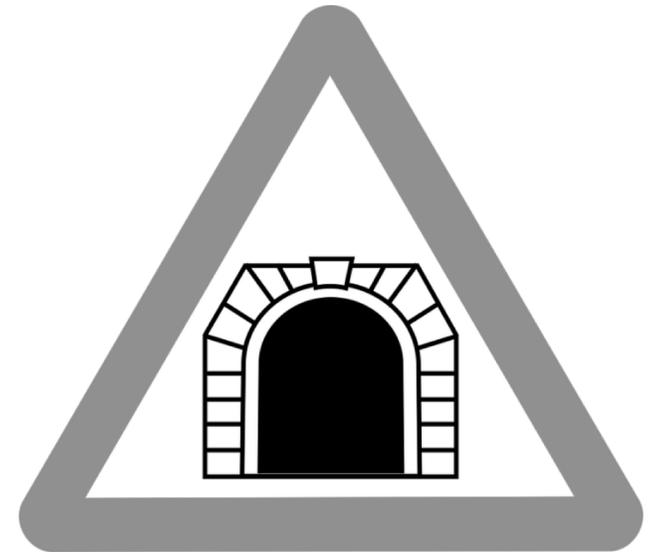


# Debugging Architecture



# Popular tunneling options

- ngrok
  - Well known
  - Doesn't require account
  - <https://ngrok.com/download>
- localtunnel
  - GitHub
  - Doesn't require account
  - <https://github.com/localtunnel/localtunnel>



# Hints and tips

- Generally speaking, ngrok works on Windows and Mac
  - Oracle Windows images (OBI) don't allow the use of ngrok
  - When ngrok doesn't work, use "localtunnel" instead instead
- Ensure you have direct Internet access
  - Public or home network
  - Mobile hotspot
- If you are behind a proxy
  - Get the external IP address of your proxy (<http://www.whatismyproxy.com/>).
  - In the terminal window where you'll be using ngrok enter
    - export https\_proxy=http://<external ip>:80
    - export http\_proxy=http://<external ip>:80

# Topic agenda

- 1 Debugging vs. logging
- 2 Debugging architecture
- 3 Debugging practices**
- 4 Debugging with MS Visual Studio Code

# How-to debug custom component services

- Run Node server in Oracle Bots Node.js SDK
- Configure tunnel to expose port 3000
- Register component service 'External' service
  - Local runtime is an 'External' deployment
  - `https://tunnel-url/components`
- Connect JavaScript IDE debugger to local node server process
- Set breakpoints and use conversation tester in skill to start the debug session

The screenshot shows a 'Create Service' dialog box with the following fields and options:

- Name:** HelloWorld
- Description:** My First Custom Component
- Deployment Options:** Embedded Container, Oracle Mobile Cloud, **External** (selected)
- Metadata URL:** https://47e39338.ngrok.io/components
- User Name:** none
- Password:** masked with dots
- Optional HTTP Headers:** (collapsed)
- Create** button

An arrow points from the 'Optional HTTP Headers' section to the Metadata URL field, indicating a configuration step.

# Starting the Node server on the local machine

- On local machine, start component service from parent folder
  - E.g. `bots-node-sdk service helloworld101`
- Open tunnel for port 3000
  - E.g. ngrok

The image shows two terminal windows. The top window is titled 'Desktop — node /usr/local/bin/bots-node-sdk service helloworld101 — 76x25'. It shows the command `bots-node-sdk service helloworld101` being executed, which results in the output: 'Component Service Ready (no auth): http://localhost:3000/components' and 'helloworld101 => HelloWorldComponent'. The bottom window is titled '/Applications — ngrok http 3000' and shows the ngrok interface. It displays session status as 'online', session expires in 7 hours, 32 minutes, and version 2.2.8. The forwarding section shows 'http://47e39338.ngrok.io -> localhost:3000' and 'https://47e39338.ngrok.io -> localhost:3000'. A table at the bottom shows connection statistics with columns for ttl, opn, rt1, rt5, p50, and p90, all with values of 0 or 0.00.

```
Desktop — node /usr/local/bin/bots-node-sdk service helloworld101 — 76x25
...bin/bots-node-sdk service helloworld101  /Applications — ngrok http 3000 ... +
[fnimphiu-orcl:Desktop fnimphiu$ ls
helloworld101
[fnimphiu-orcl:Desktop fnimphiu$ bots-node-sdk service helloworld101
-----
Component Service Ready (no auth):
http://localhost:3000/components
-----
helloworld101 => HelloWorldComponent

/Applications — ngrok http 3000  ...ots-node-sdk service helloworld101
ngrok by @inconshreveable (Ctrl+C to quit)
Session Status      online
Session Expires     7 hours, 32 minutes
Version              2.2.8
Region              United States (us)
Web Interface        http://127.0.0.1:4040
Forwarding           http://47e39338.ngrok.io -> localhost:3000
Forwarding           https://47e39338.ngrok.io -> localhost:3000
Connections          ttl    opn    rt1    rt5    p50    p90
                    0      0      0.00  0.00  0.00  0.00
```

# Testing the custom component service access in browser

- Type `https://<tunnel URL>/components`
  - Displays list of components hosted by component service
- Be aware
  - Tunnel URL changes with each re-start of the tunnel



 `https://47e39338.ngrok.io/components`

```
{"version": "1.1", "components": [{"name": "HelloWorldComponent", "properties": {"human": {"required": tr
```

# Register custom component service in skill

The image shows a two-step process. On the left, the 'Create Service' dialog is open. The 'Name' field is 'HelloWorld' and the 'Description' is 'My First Custom Component'. Under the 'Container' section, the 'External' radio button is selected and highlighted with a red box. The 'Metadata URL' is 'https://47e39338.ngrok.io/components', 'User Name' is 'none', and 'Password' is masked with dots. A 'Create' button is at the bottom right. An arrow points to the right, where the service is listed in the console. A green '+ Service' button is at the top left. Below it is a search filter. The 'HelloWorld' service is selected, showing its details: 'HelloWorldComponent' as the component name. Below this, a table shows the 'Properties' for the component.

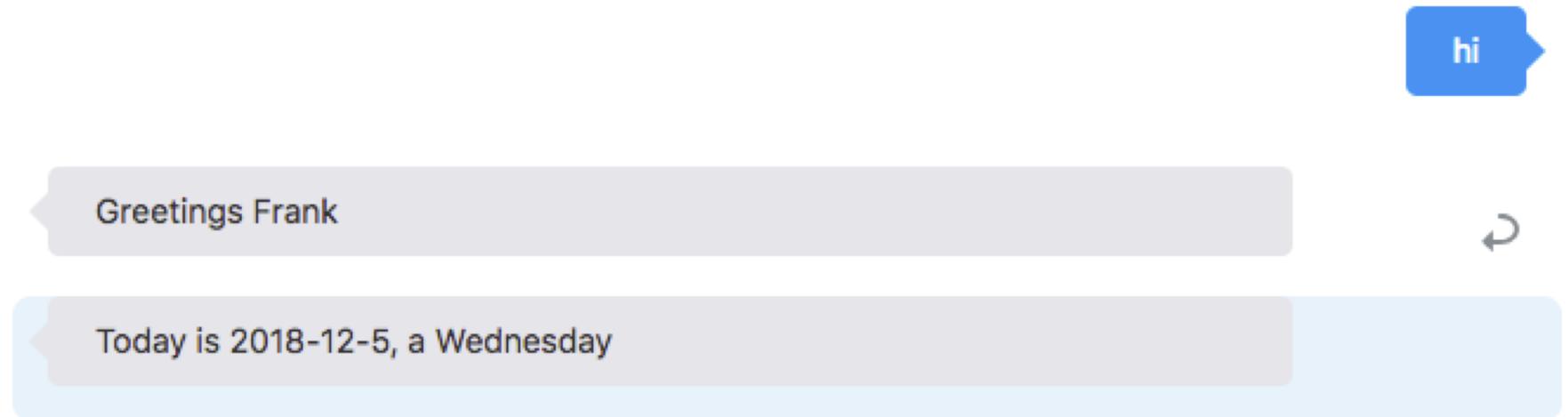
Property	Type	Required
human	string	true

Below the table, 'Supported Actions' are listed: 'weekday' and 'weekend'.

- Configure custom component service using ngrok or local tunnel URL
  - <https://47e39338.ngrok.io/components>
- No authentication required (just put something into the username and password field)

# Dialog Flow configuration & runtime output

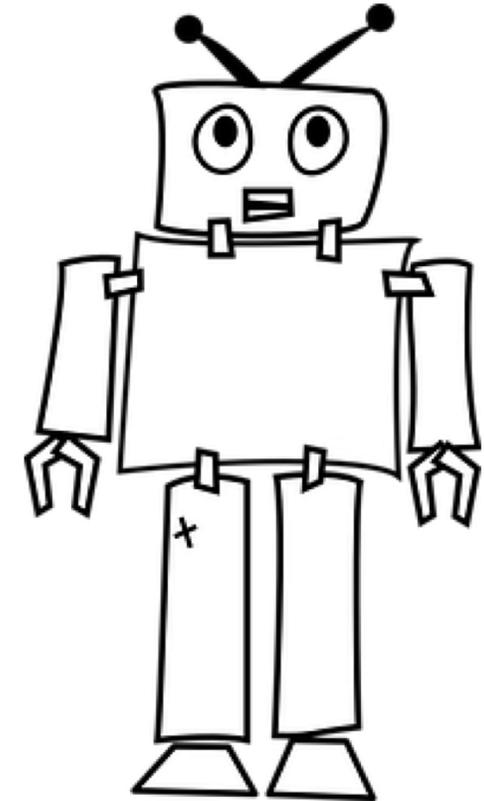
```
states:  
  askGreeting:  
    component: "HelloWorldComponent"  
    properties:  
      human: "Frank"  
    transitions:  
      return: "done"
```



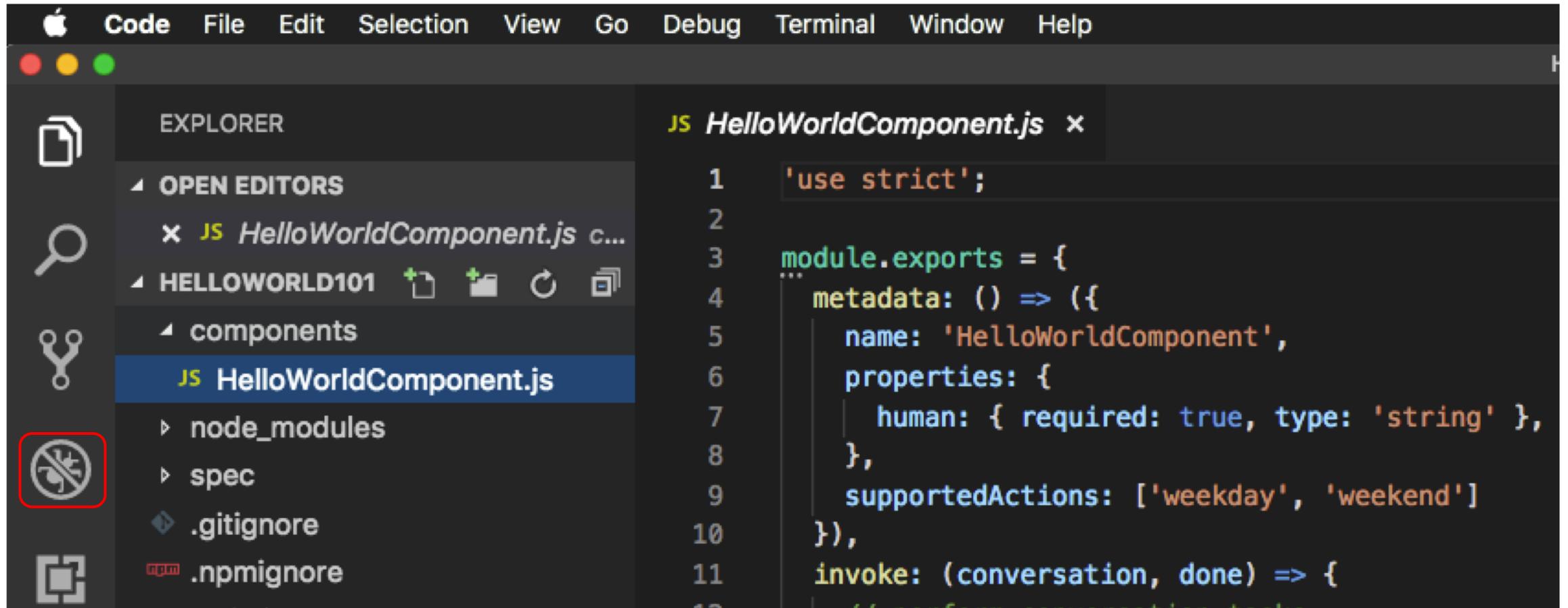
# Topic agenda

- 1 Debugging vs. logging
- 2 Debugging architecture
- 3 Debugging practices
- 4 Debugging with MS Visual Studio Code**

There is **no dependency to a specific JavaScript IDE**. The IDE must be able to debug local Node process.



# Open project in IDE and select debug option

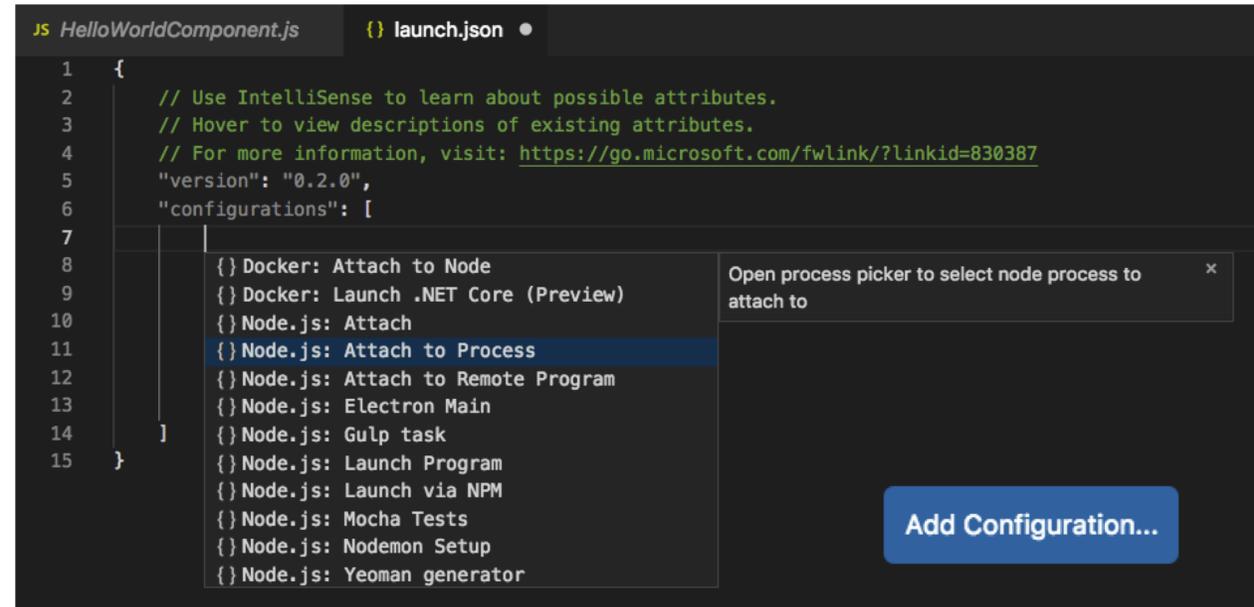
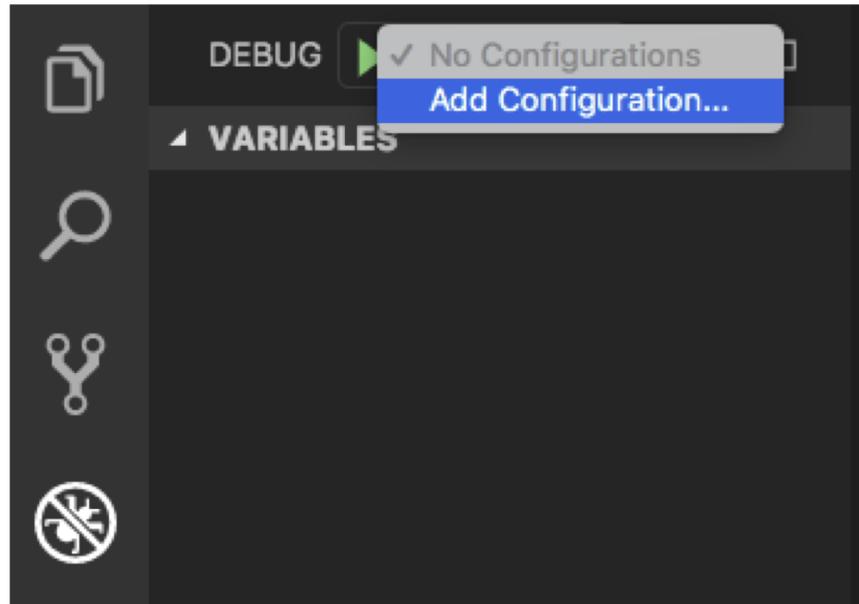


```
Code File Edit Selection View Go Debug Terminal Window Help

EXPLORER
├─ OPEN EDITORS
│   └─ JS HelloWorldComponent.js c...
├─ HELLOWORLD101
│   ├── components
│   │   └─ JS HelloWorldComponent.js
│   ├── node_modules
│   ├── spec
│   ├── .gitignore
│   └─ .npmignore
└─ [Debug Console Icon]

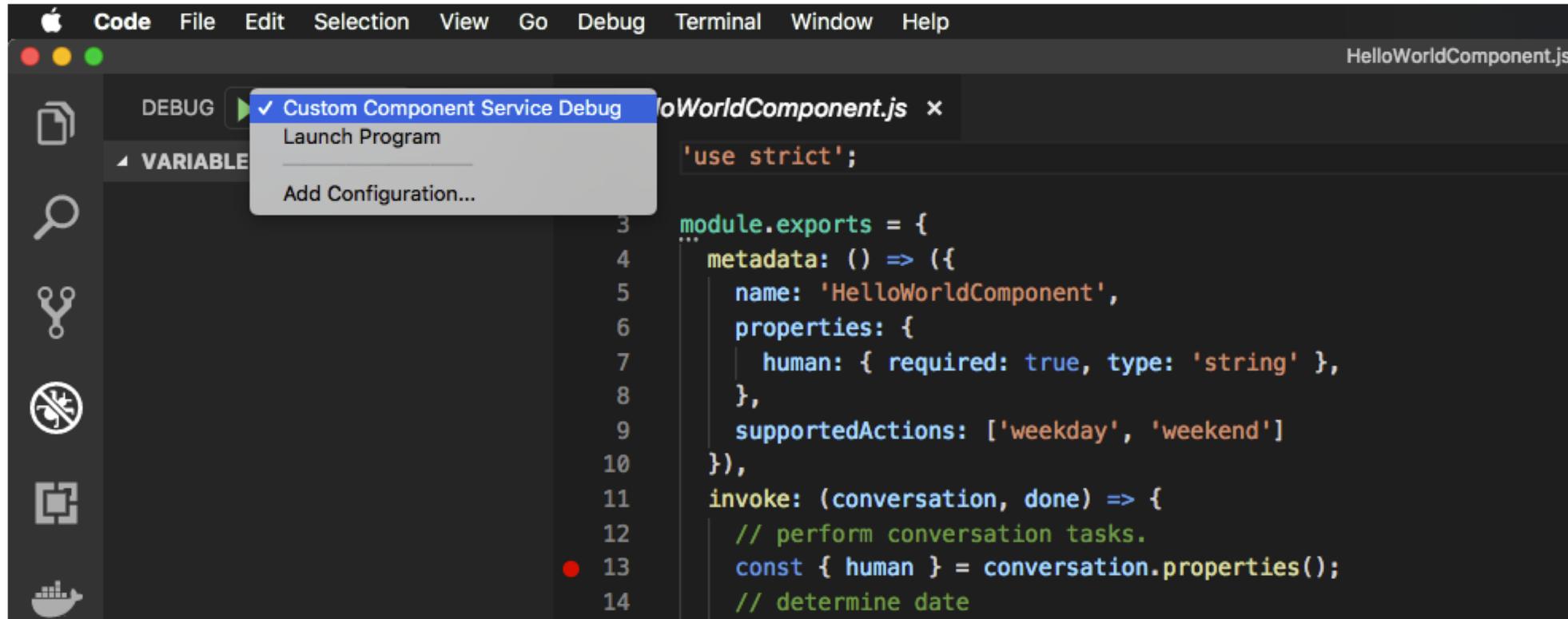
JS HelloWorldComponent.js x
1  'use strict';
2
3  module.exports = {
4  ...
5  metadata: () => ({
6    name: 'HelloWorldComponent',
7    properties: {
8      human: { required: true, type: 'string' },
9    },
10   supportedActions: ['weekday', 'weekend']
11  })),
12  invoke: (conversation, done) => {
13    // perform conversation logic
```

# Create debugging configuration



- Add a new debug configuration
  - Choose "Attach to Process" from "Add Configuration" button list
- Optionally, provide a custom name for the debug configuration

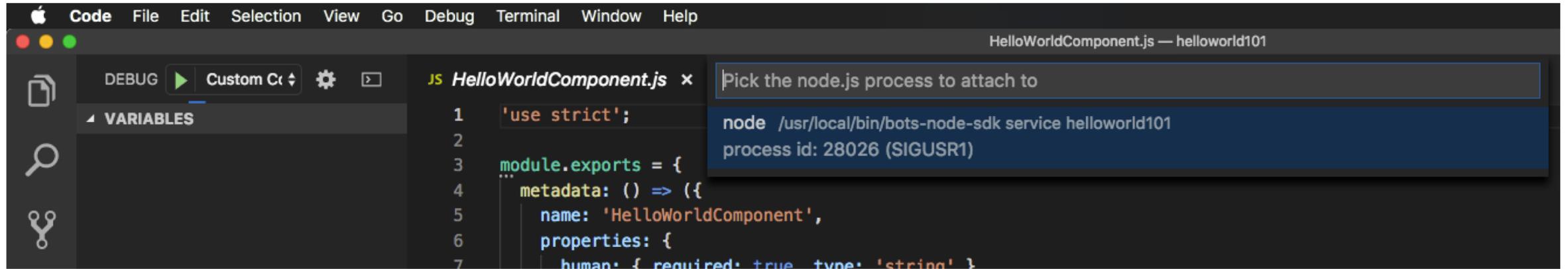
# Set breakpoint and start debugging session



```
Code File Edit Selection View Go Debug Terminal Window Help
HelloWorldComponent.js
DEBUG
  ✓ Custom Component Service Debug
  Launch Program
  Add Configuration...
VARIABLE
3
4
5
6
7
8
9
10
11
12
13
14
'use strict';
module.exports = {
  metadata: () => ({
    name: 'HelloWorldComponent',
    properties: {
      human: { required: true, type: 'string' },
    },
    supportedActions: ['weekday', 'weekend']
  }),
  invoke: (conversation, done) => {
    // perform conversation tasks.
    const { human } = conversation.properties();
    // determine date
```

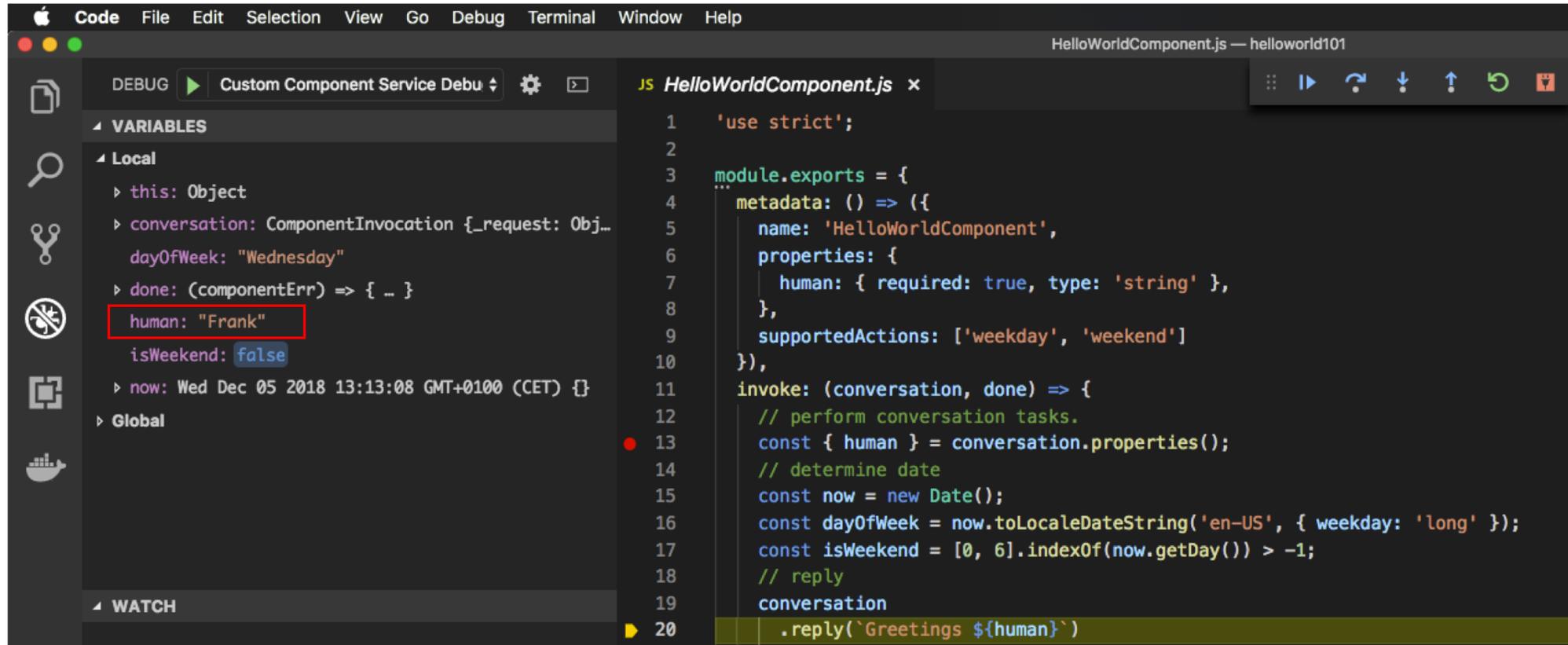
- Set break-points in the *invoke* function
- Select your custom debug configuration

# Select process



- Press the green run icon
- Select the Node process from the list

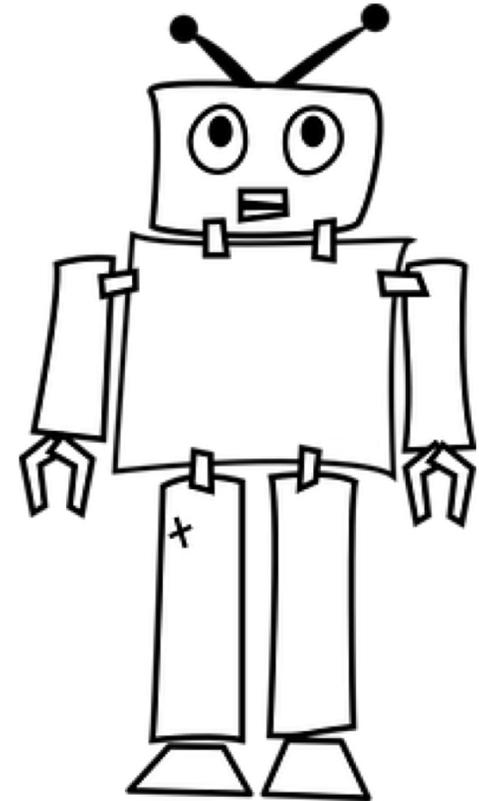
# Debug component service by running skill bot tester



The screenshot shows the Visual Studio Code interface during a debug session. The top menu bar includes 'Code', 'File', 'Edit', 'Selection', 'View', 'Go', 'Debug', 'Terminal', 'Window', and 'Help'. The title bar indicates the file is 'HelloWorldComponent.js' in a workspace named 'helloworld101'. The left sidebar contains several icons for navigation and debugging, with the 'VARIABLES' panel expanded. Under the 'Local' scope, the variable 'human' is set to the string 'Frank' and is highlighted with a red box. Other variables include 'this', 'conversation', 'done', 'isWeekend', and 'now'. The main editor displays the JavaScript code for 'HelloWorldComponent.js'. A red dot indicates a breakpoint is set at line 20, which is the final line of the 'invoke' function: `.reply(`Greetings ${human}`)`. The code defines a metadata object with a name, properties, and supported actions, and an invoke function that extracts the 'human' property from the conversation and replies with a greeting.

```
1 'use strict';
2
3 module.exports = {
4   metadata: () => ({
5     name: 'HelloWorldComponent',
6     properties: {
7       human: { required: true, type: 'string' },
8     },
9     supportedActions: ['weekday', 'weekend']
10  }),
11   invoke: (conversation, done) => {
12     // perform conversation tasks.
13     const { human } = conversation.properties();
14     // determine date
15     const now = new Date();
16     const dayOfWeek = now.toLocaleDateString('en-US', { weekday: 'long' });
17     const isWeekend = [0, 6].indexOf(now.getDay()) > -1;
18     // reply
19     conversation
20     .reply(`Greetings ${human}`)
```

Rerun the '**bots-node-sdk service**' command each time you change the component source code in a debug session for changes to take affect.



# Custom component deployment to local container

- Create a .tgz file for local container deployment
  - Run *npm pack* in the component service project folder
- Delete or disable the current custom component service configuration
- Create new service configuration using packaged component service
  - Name different if you kept the debugging configuration

**Create Service** [X]

\* **Name** HelloWorld\_Packaged

**Description** Optional short description for this service

Embedded Container  Oracle Mobile Cloud  External

? **Package File** ✓ helloworld101-1.0.1.tgz ready to be processed. [Change](#)

**Create**

# Integrated Cloud

## Applications & Platform Services

ORACLE®



# Oracle Digital Assistant Hands-On

---

TBD