ORACLE

How to call backend services

Agenda

1. Backend integration overview
2. Built-in Rest service component
3. Custom dialog flow components
4. Entity event handler
5. Custom component deployment options
6. Best practices

Agenda

1. **Backend integration overview**
2. Built-in Rest service component
3. Custom dialog flow components
4. Entity event handler
5. Custom component deployment options
6. Best practices

About backend integration

> A digital assistant with no backend integration may be of no use.

Typically, digital assistants need to interact with a backend in one way or another

- FAQ digital assistants too

Access to business or data services

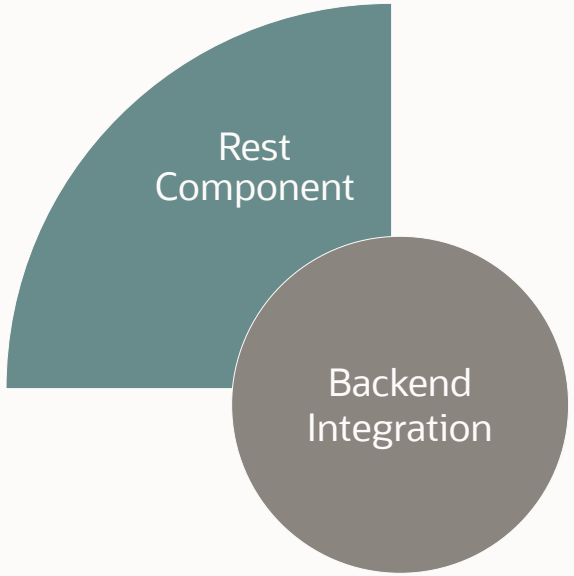
Access to the Internet

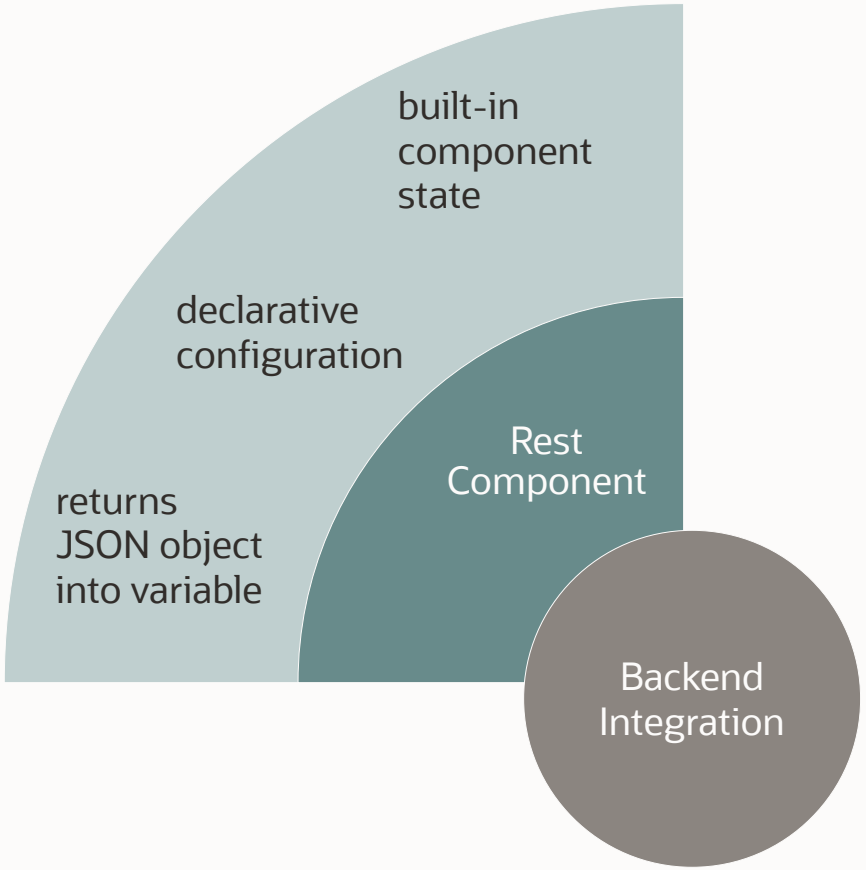
- "Who is the president of the USA?"

In Oracle Digital Assistant, developer access to backend services is through REST services

- May use OIC, OPA, or ORDS as intermediary

Backend Integration


Rest
Component
Backend
Integration

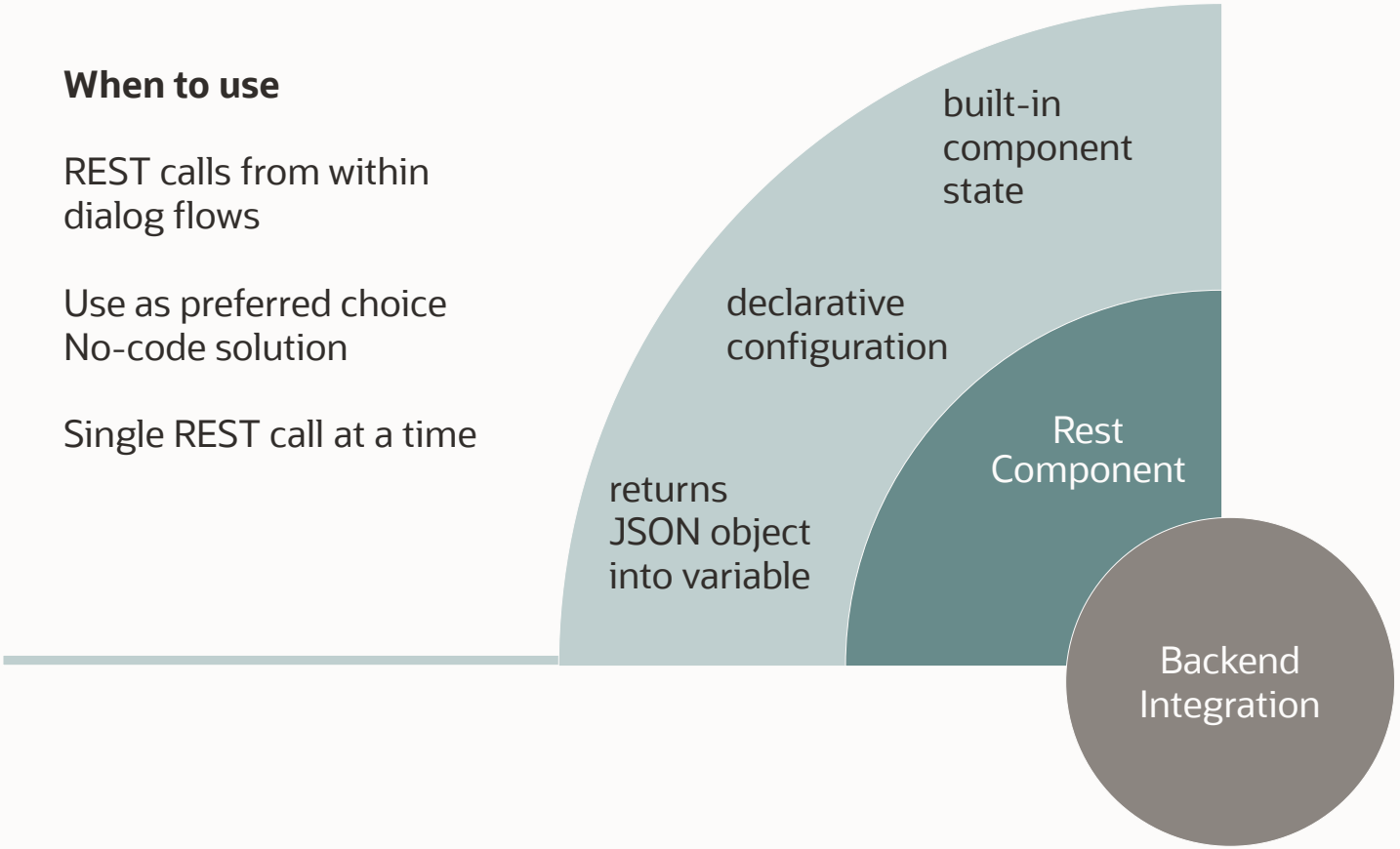

built-in
component
state
declarative
configuration
returns
JSON object
into variable
Rest
Component
Backend
Integration

When to use

REST calls from within dialog flows

Use as preferred choice
No-code solution

Single REST call at a time

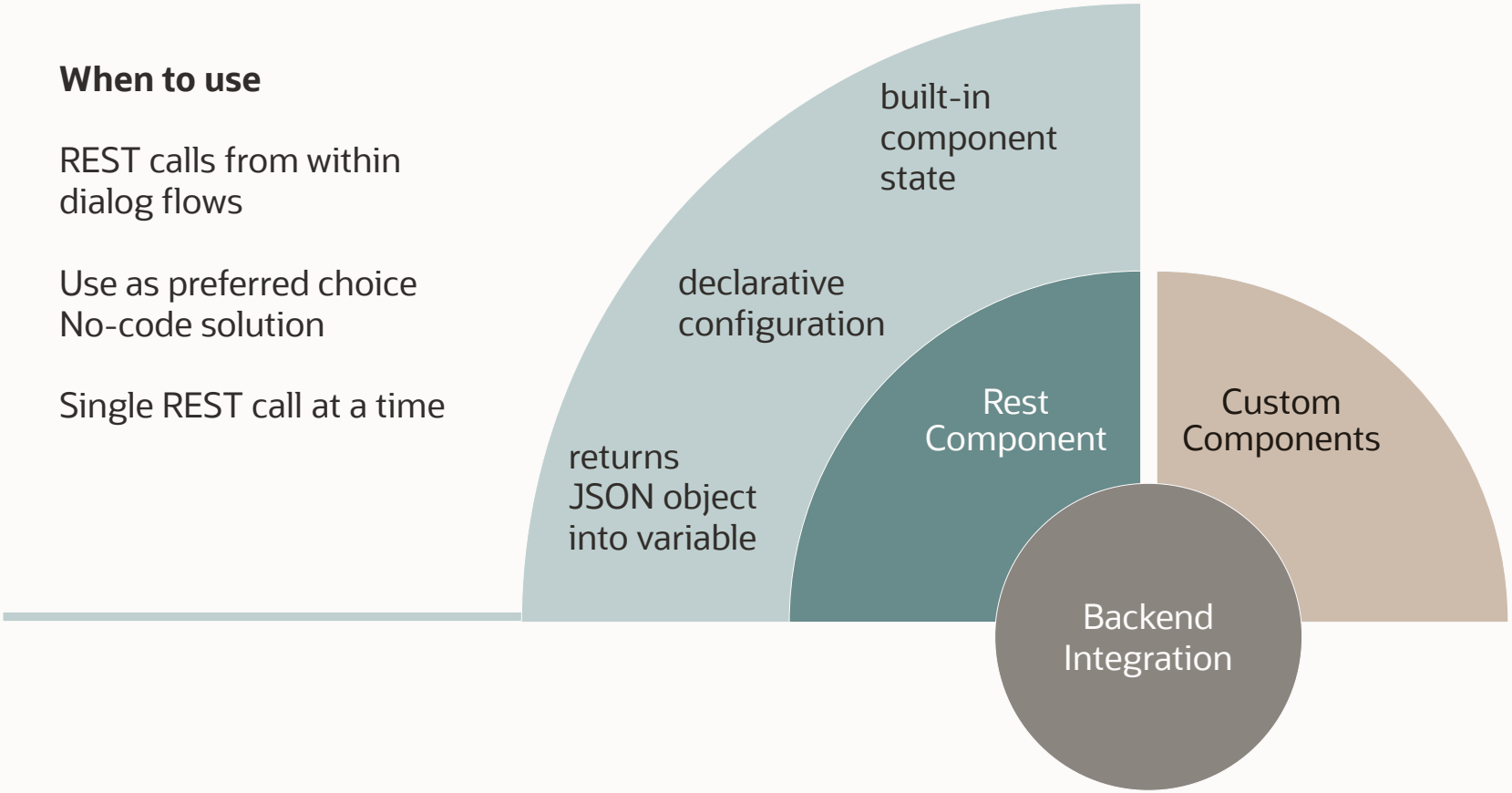


When to use

REST calls from within dialog flows

Use as preferred choice
No-code solution

Single REST call at a time

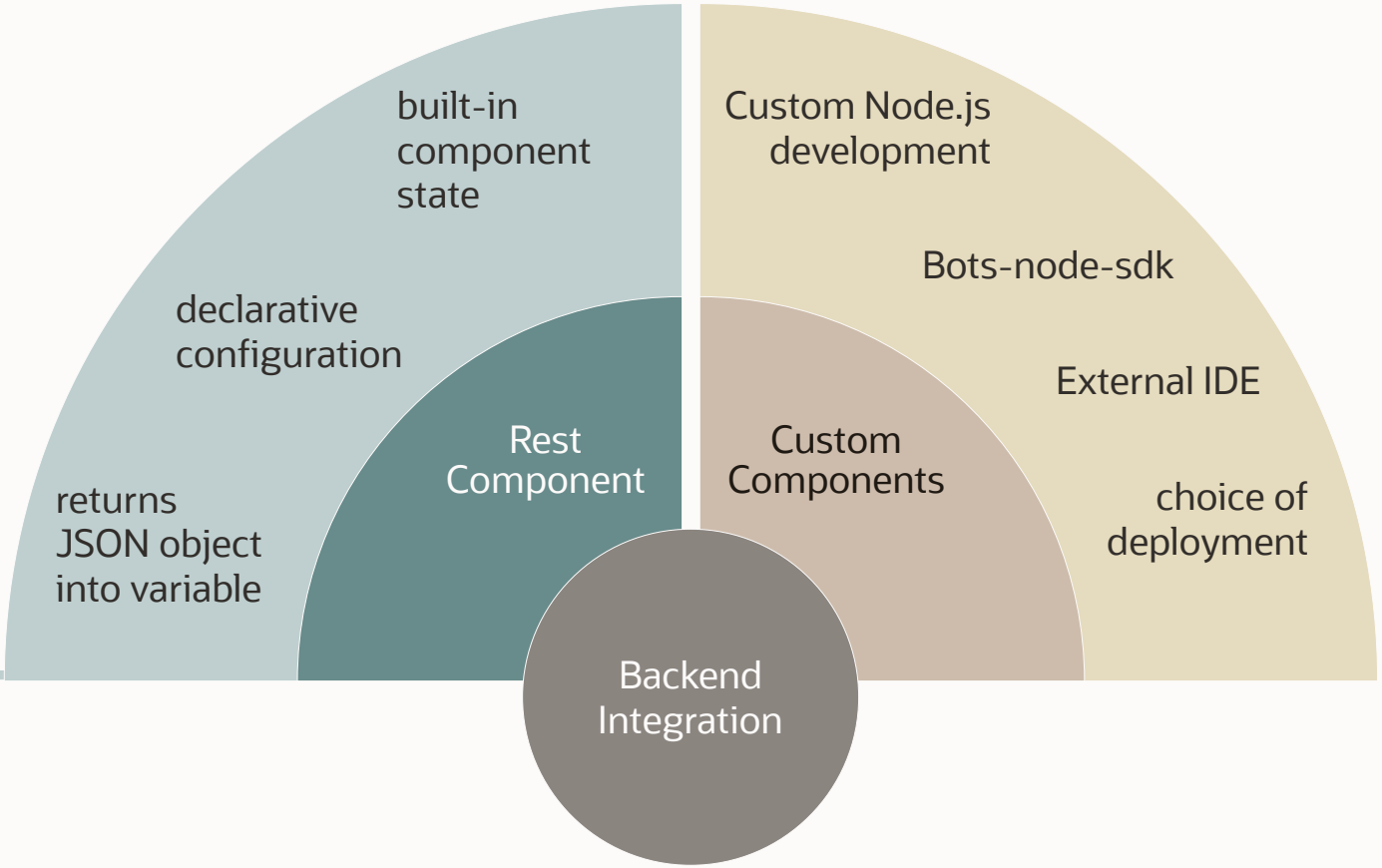


When to use

REST calls from within dialog flows

Use as preferred choice
No-code solution

Single REST call at a time

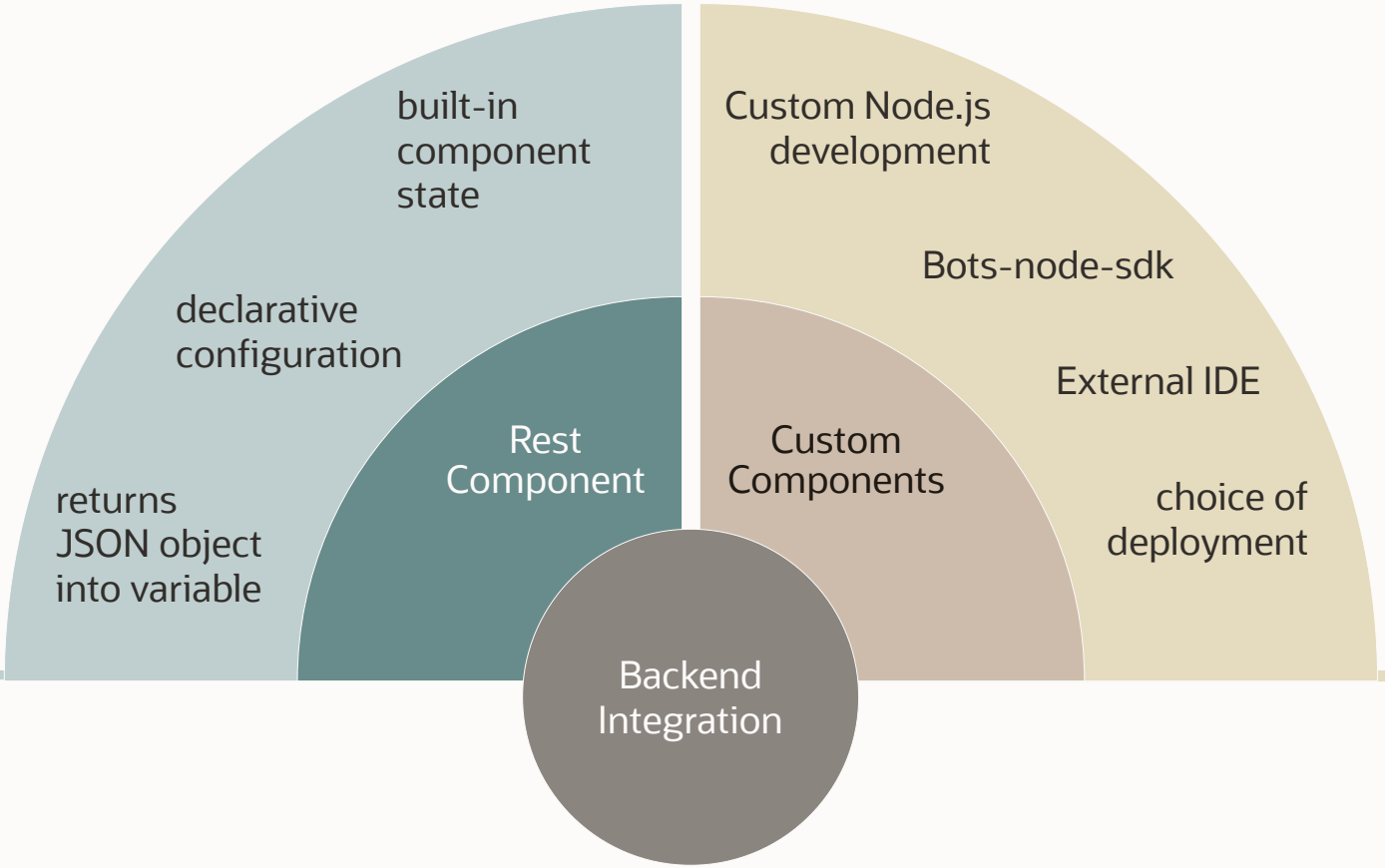


When to use

REST calls from within dialog flows

Use as preferred choice
No-code solution

Single REST call at a time



When to use

REST calls from within dialog flows

Multiple REST call within one invocation

Post processing of REST response

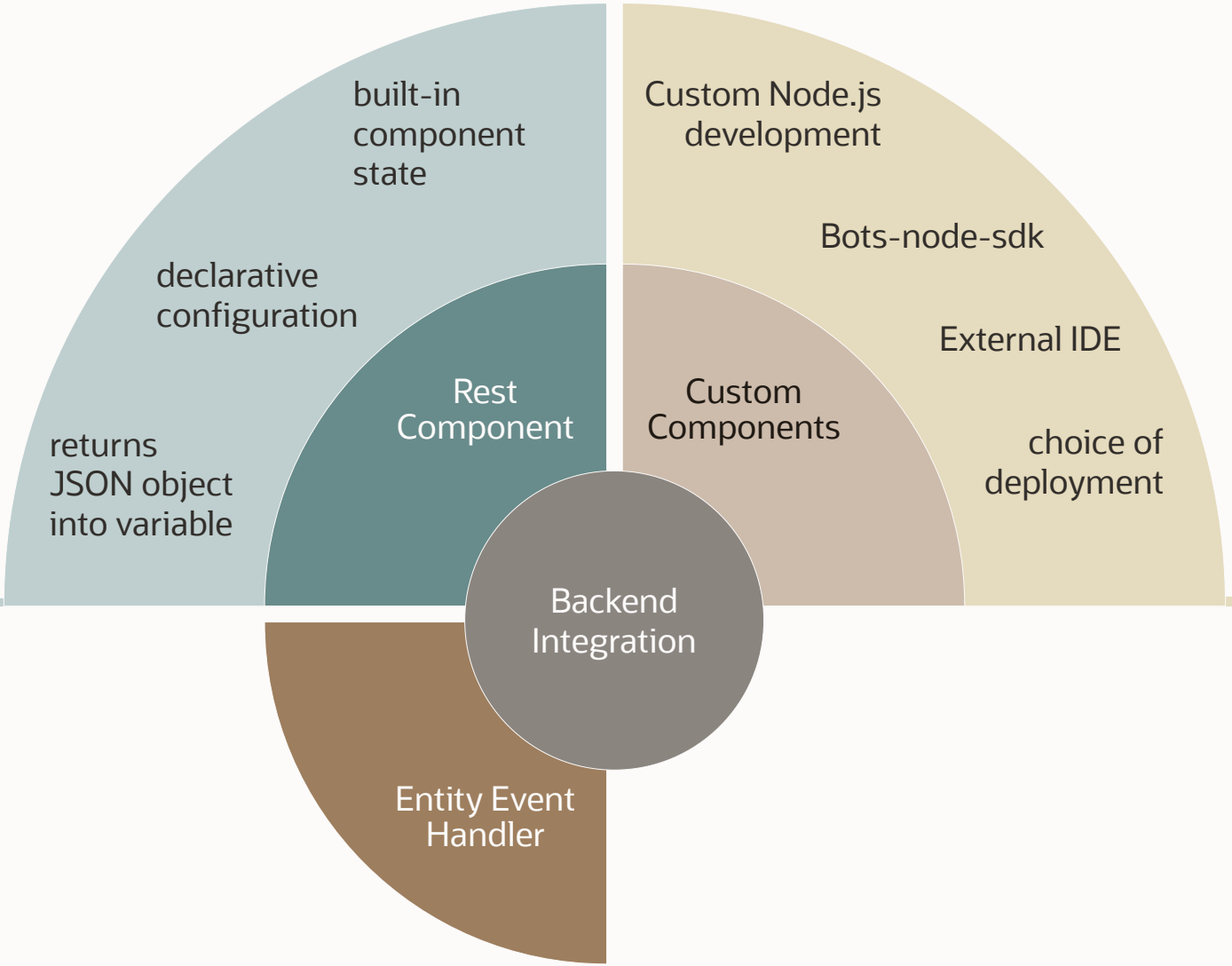
OCI integration

When to use

REST calls from within dialog flows

Use as preferred choice
No-code solution

Single REST call at a time



When to use

REST calls from within dialog flows

Multiple REST call within one invocation

Post processing of REST response

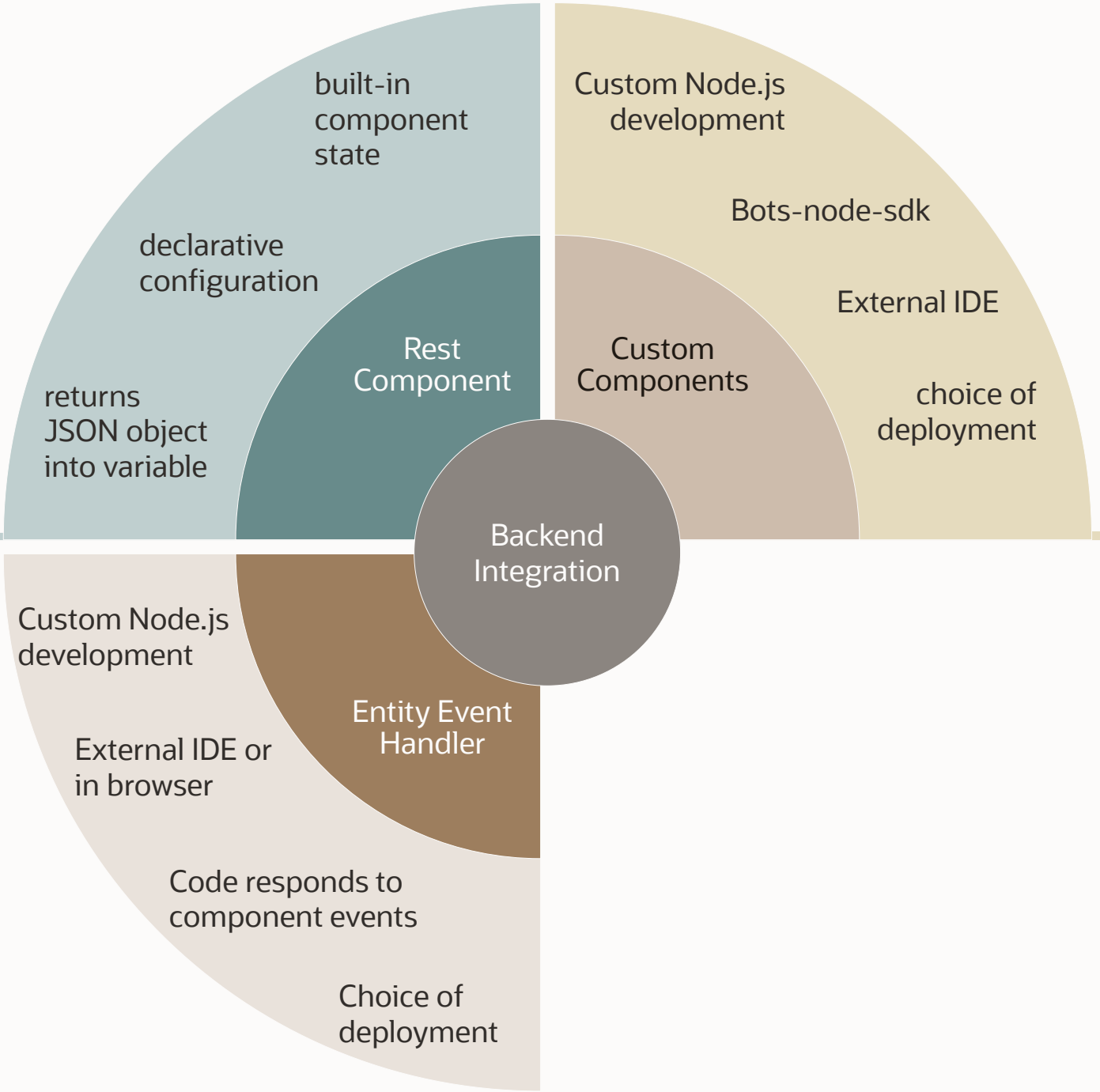
OCI integration

When to use

REST calls from within dialog flows

Use as preferred choice
No-code solution

Single REST call at a time



When to use

REST calls from within dialog flows

Multiple REST call within one invocation

Post processing of REST response

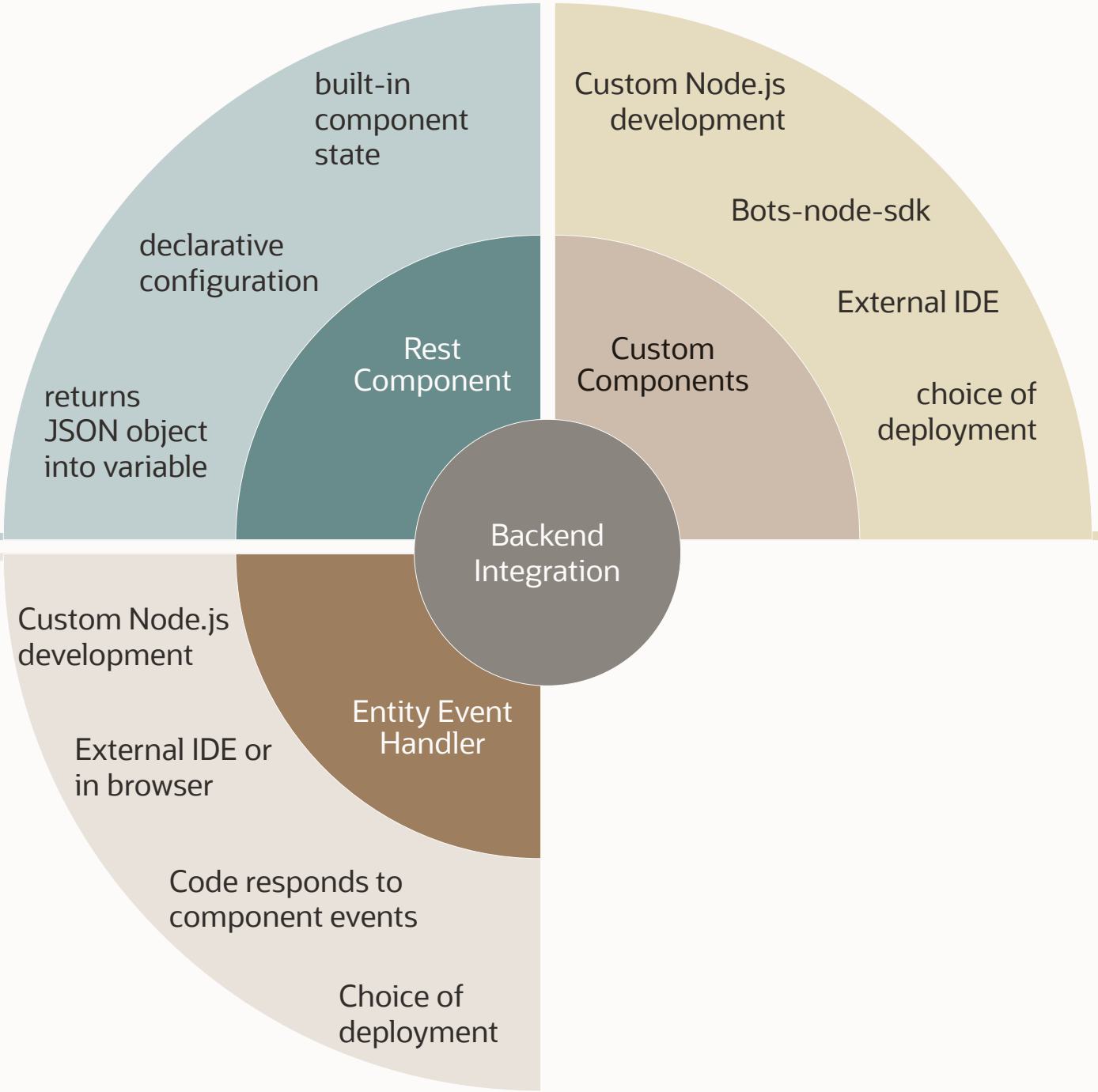
OCI integration

When to use

REST calls from within dialog flows

Use as preferred choice
No-code solution

Single REST call at a time



When to use

REST calls from within dialog flows

Multiple REST call within one invocation

Post processing of REST response

OCI integration

When to use

Use with composite bag entities

- No need to exit CBE for REST call

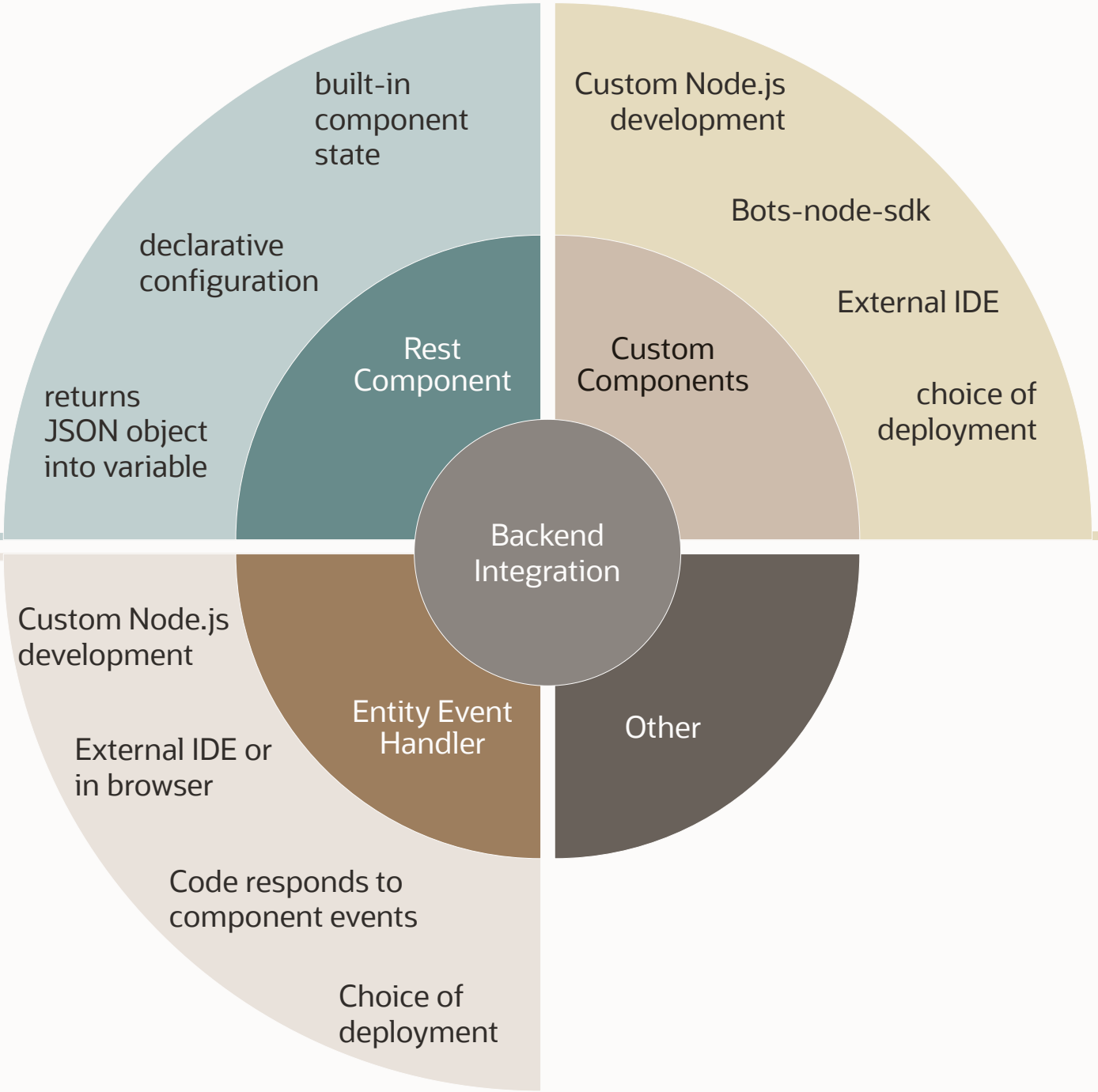
Use with model-driven conversations

When to use

REST calls from within dialog flows

Use as preferred choice
No-code solution

Single REST call at a time



When to use

REST calls from within dialog flows

Multiple REST call within one invocation

Post processing of REST response

OCI integration

When to use

Use with composite bag entities

- No need to exit CBE for REST call

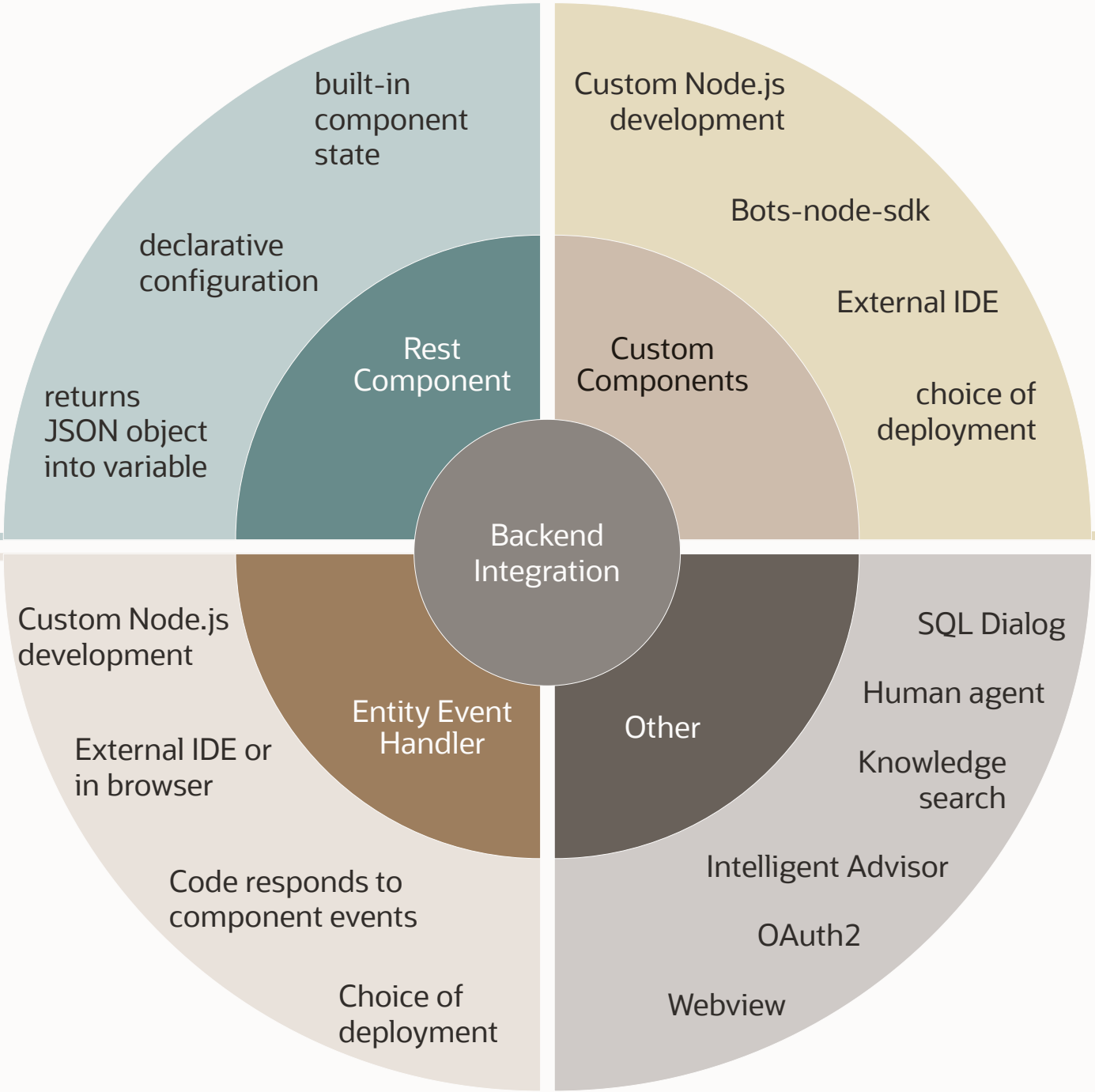
Use with model-driven conversations

When to use

REST calls from within dialog flows

Use as preferred choice
No-code solution

Single REST call at a time



When to use

REST calls from within dialog flows

Multiple REST call within one invocation

Post processing of REST response

OCI integration

When to use

Use with composite bag entities

- No need to exit CBE for REST call

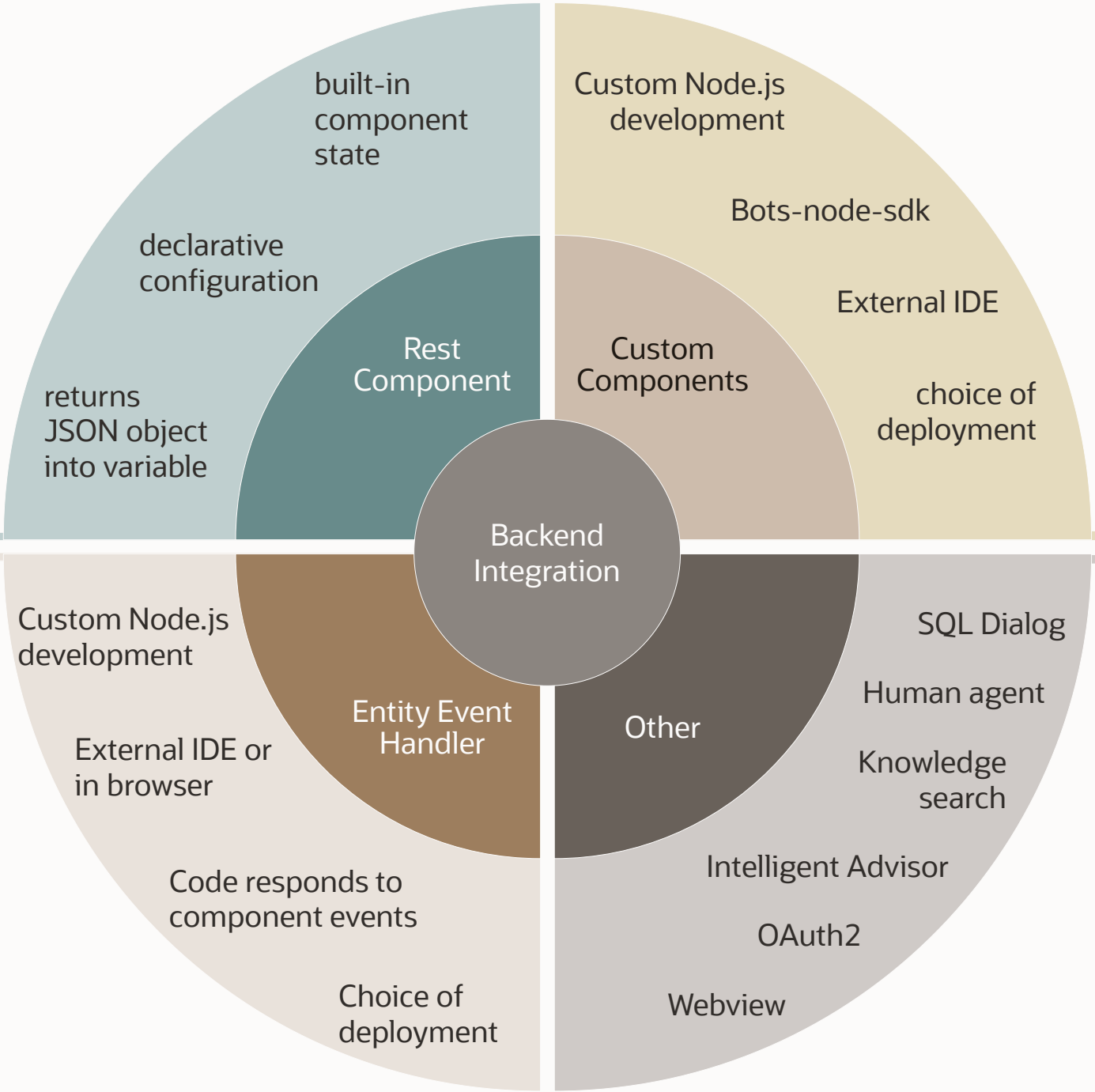
Use with model-driven conversations

When to use

REST calls from within dialog flows

Use as preferred choice
No-code solution

Single REST call at a time



When to use

REST calls from within dialog flows

Multiple REST call within one invocation

Post processing of REST response

OCI integration

When to use

Use with composite bag entities

- No need to exit CBE for REST call

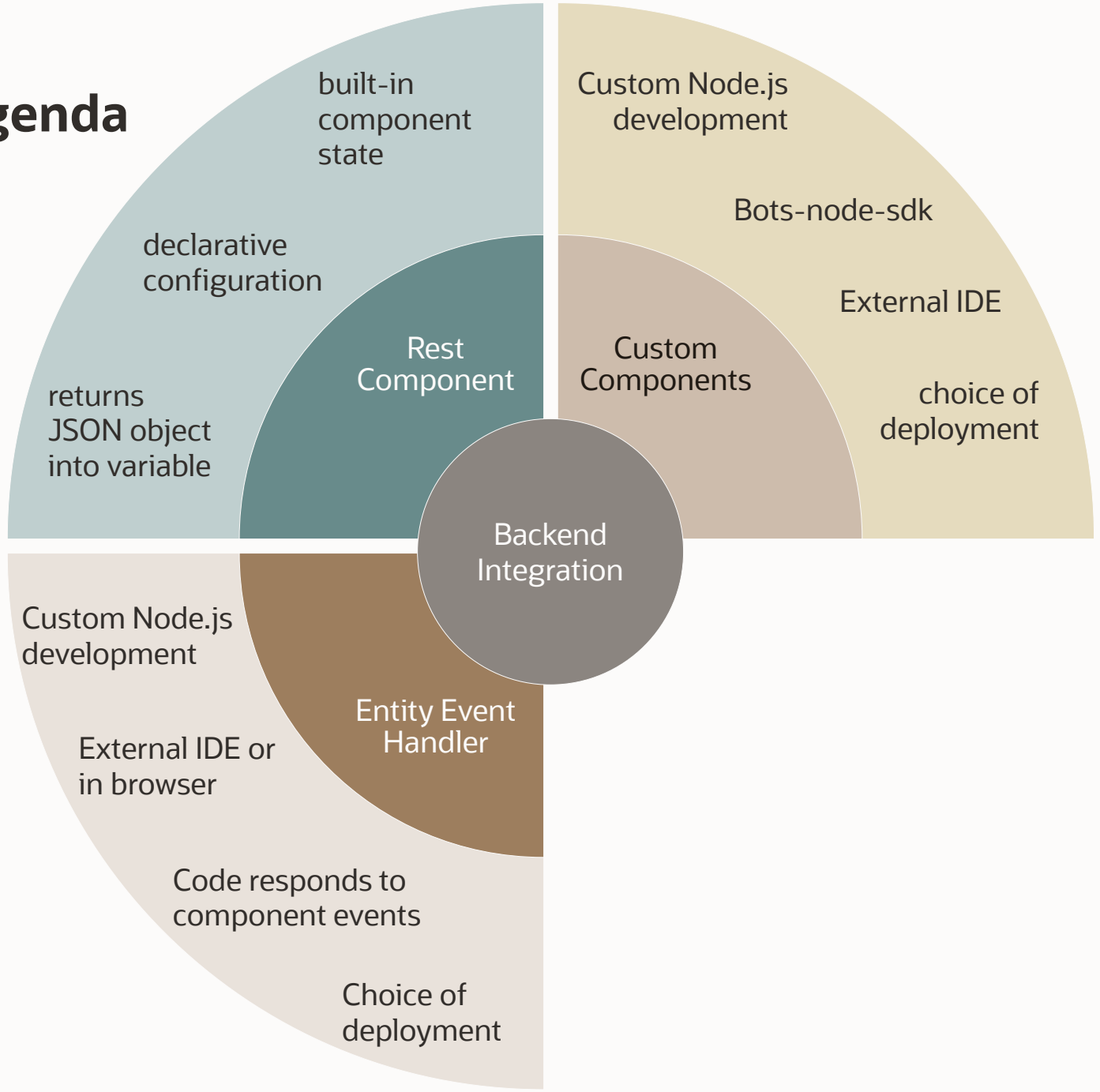
Use with model-driven conversations

When to use

Use for integration needs

Use when your requirement fits one of these out-of-the box solution

Remaining agenda



Agenda

1. Backend integration overview
2. **Built-in Rest service component**
3. Custom dialog flow components
4. Entity event handler
5. Custom component deployment options
6. Best practices

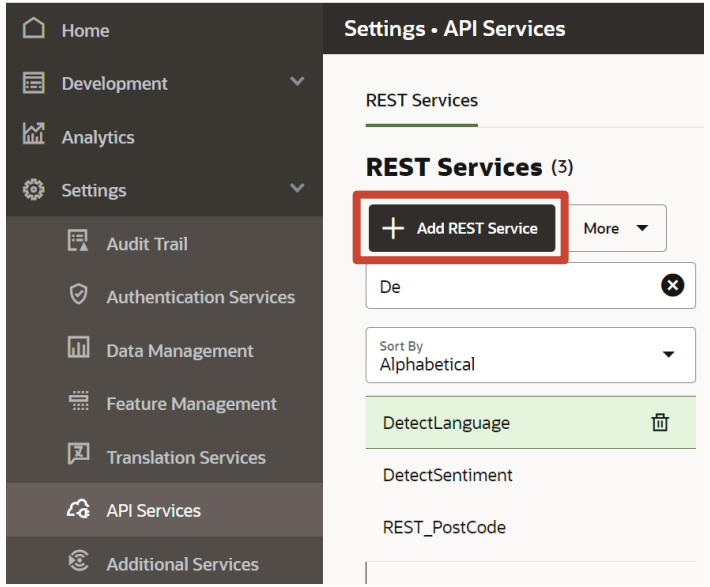
The REST service component

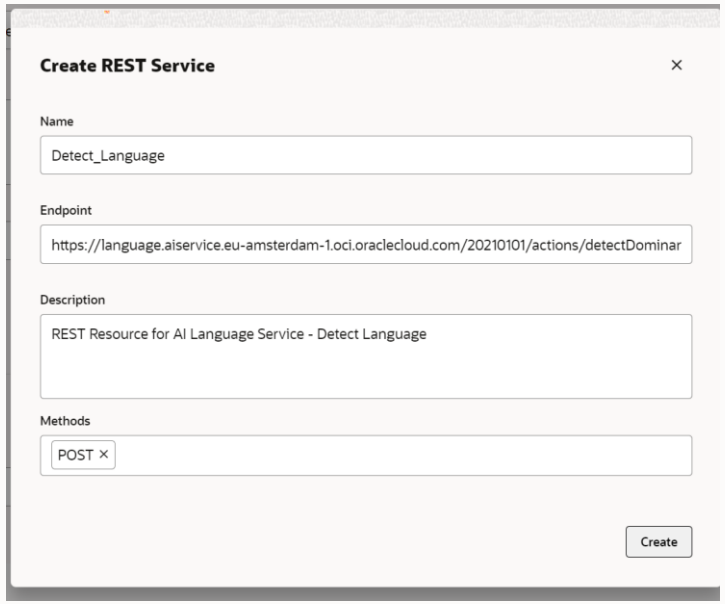
Built-in REST service component that you can use in Visual Dialog skills to send a request to a REST service's endpoint

Ideal for simple REST calls

- If you have a complex and large response, you may want to consider the custom component

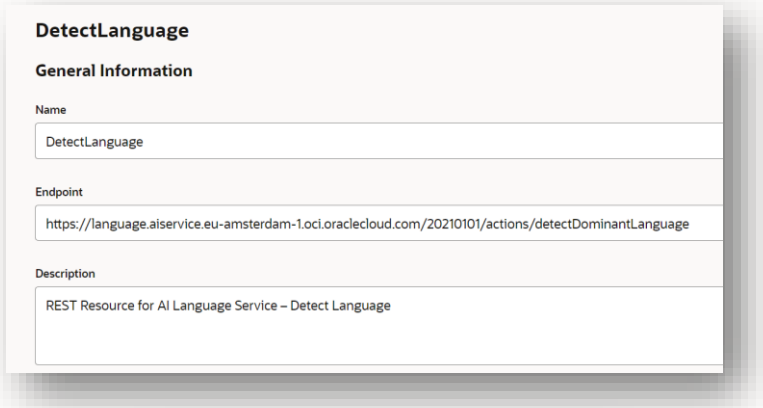
Add the new service under **Settings > API Services**





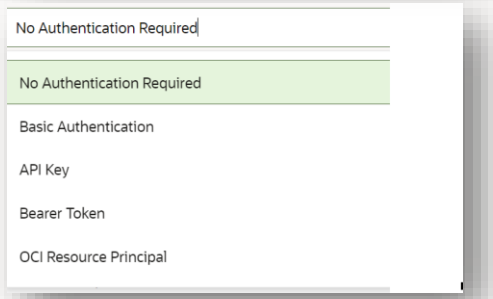
The REST service component

Configure the base endpoint, name and description



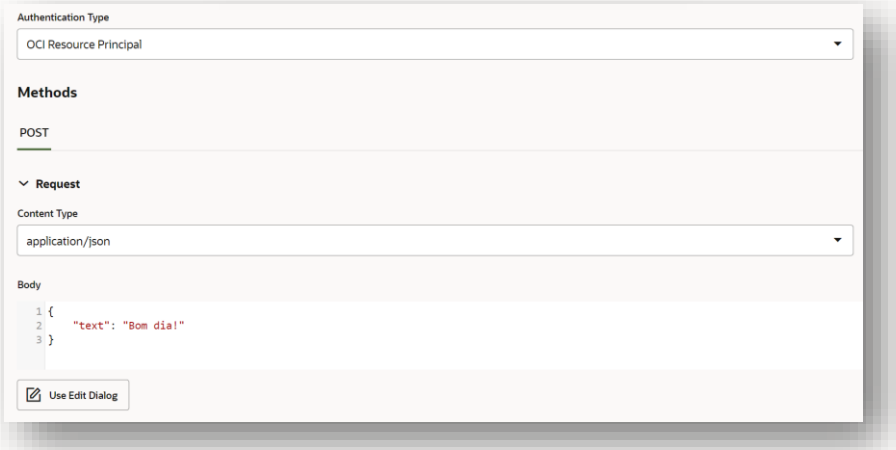
Choose the authentication type

- When using OCI Resource Principal, make sure the appropriate OCI policies are in place in the respective tenancy



Add the required methods

- Select the **content type**
- Fill the **Body** with a sample request



The REST service component

Define parameters if required

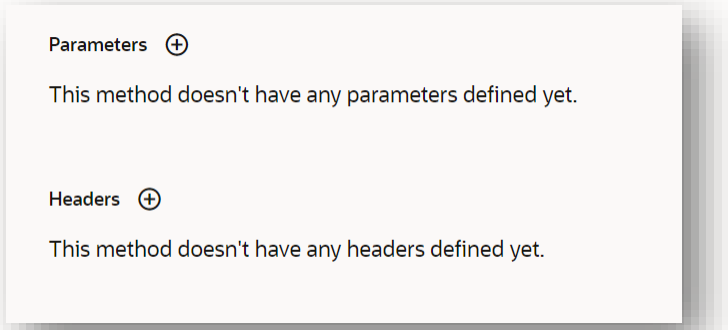
- You can configure **query** and **path** parameters

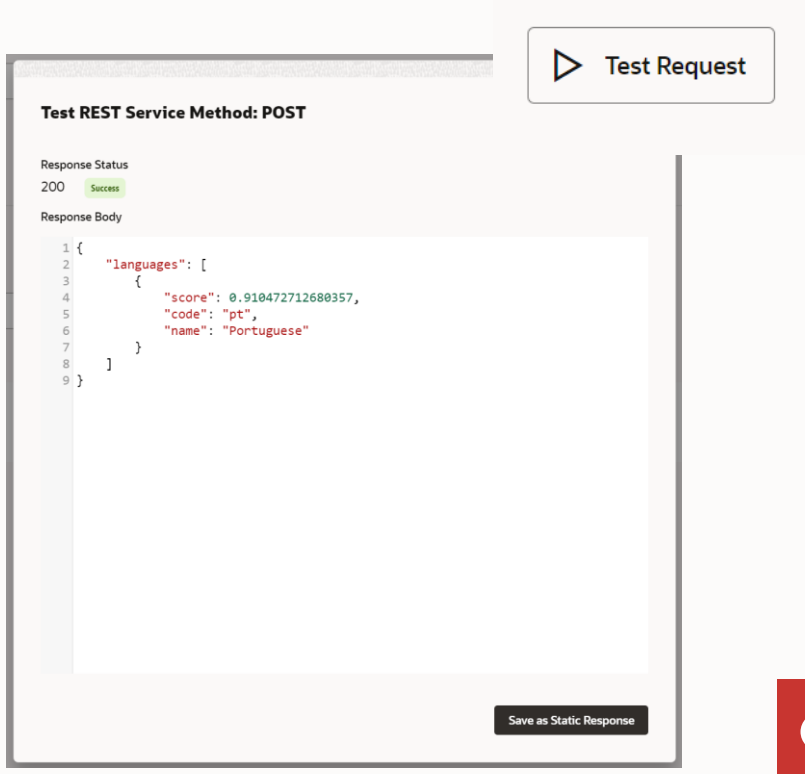
Test the request to confirm all settings are correct

You can save the response as a static value

- Useful to mock the response

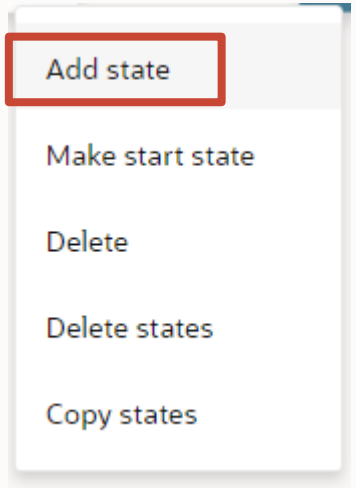
!Consider testing the API in Postman upfront!

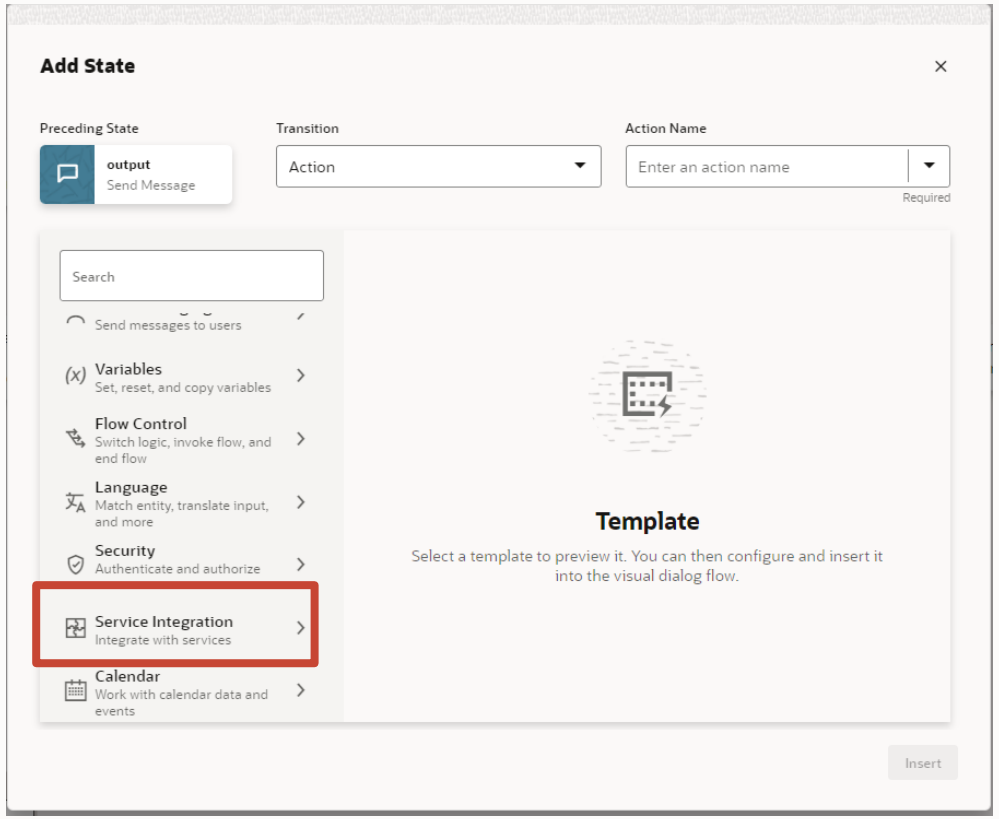


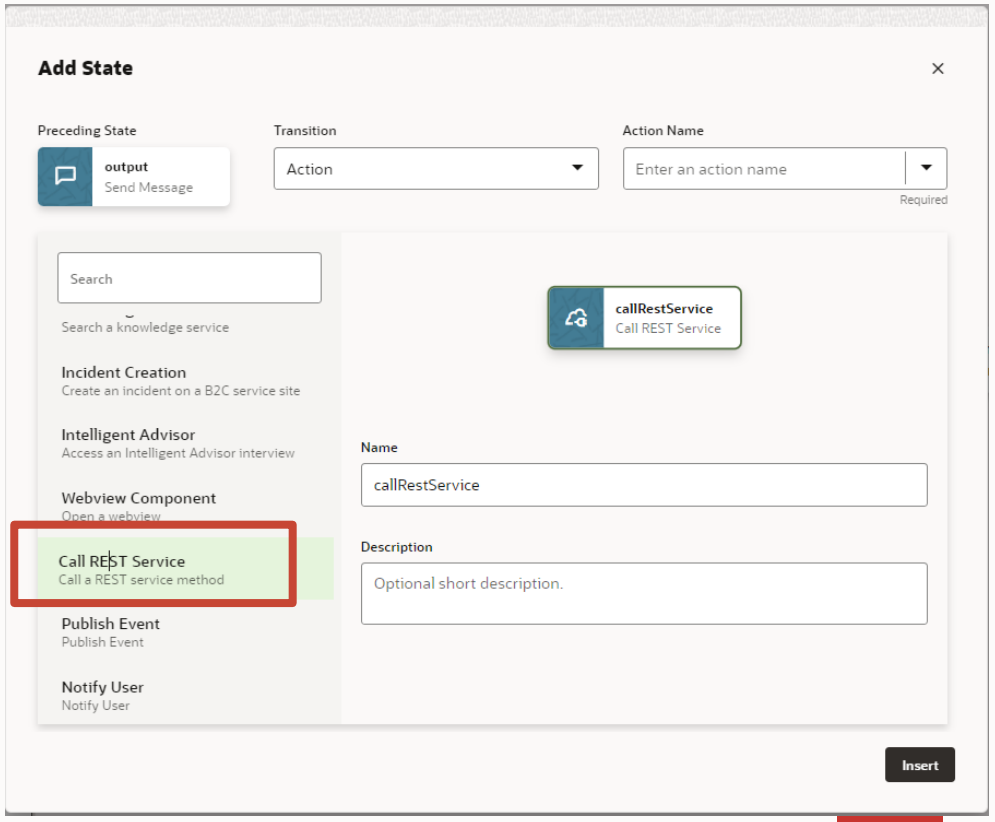


Invoke the REST service

The template Call REST Service exposes the existing configured API’s







Invoke the REST service

Pick the **REST Service** from the list of configured services

Choose the desired **Method**

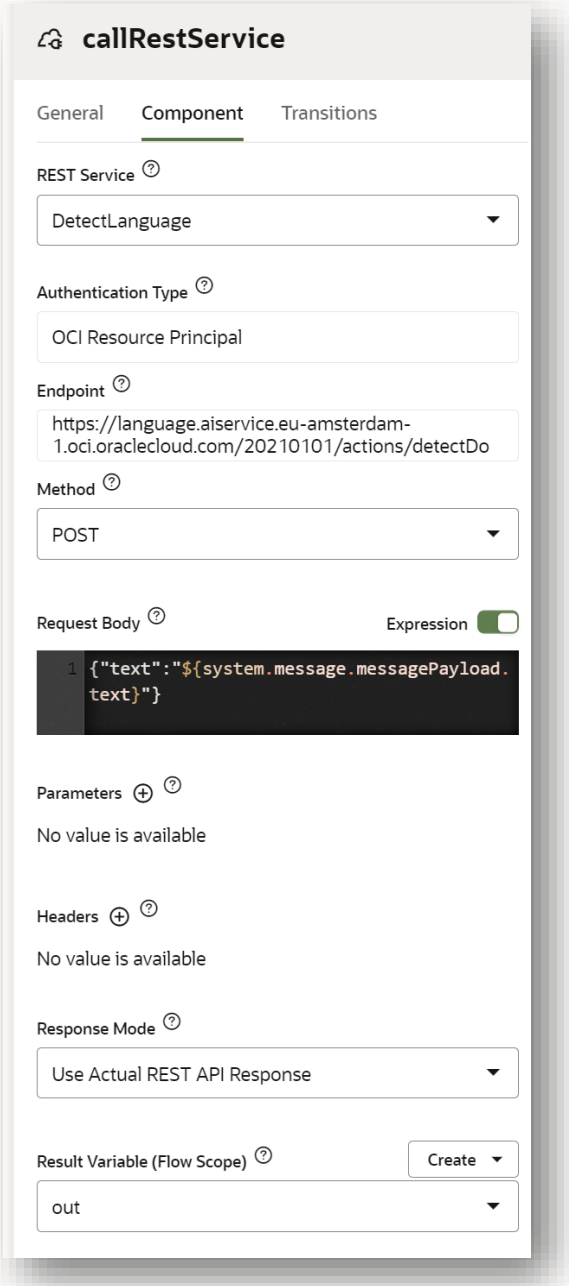
Populate the **request Body**

- With or without an expression

Map parameters and headers

Response Mode allows to mock it with the static response from the configuration page

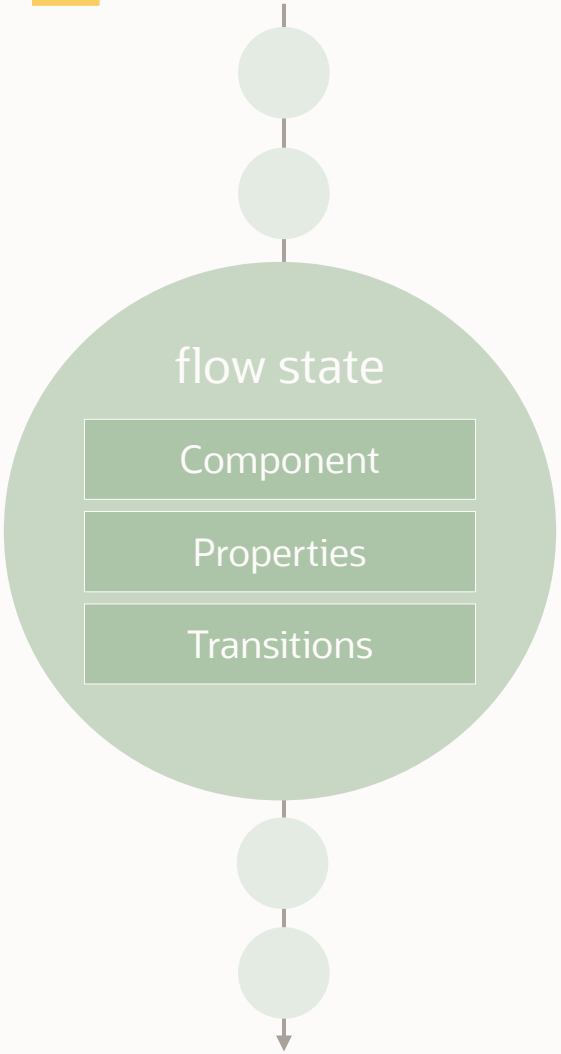
When using the actual RESP API response, a variable of type map is required to store it



Agenda

1. Backend integration overview
2. Built-in Rest service component
3. **Custom dialog flow components**
4. Entity event handler
5. Custom component deployment options

Dialog flows and components



Dialog flow states execute logic or render a user interface

- Uses components

Component types

- Built-in
- Custom

Properties

- Pass information into a component
- Used to update variables defined in a flow

Transitions

- next – navigates to a pre-defined next state
- action – conditionally navigates to a next state

About custom components

Node.js / JavaScript

Like built-in components, just custom

- Expose properties
- May return action strings
  - Used to determine navigation
- May render a user interface

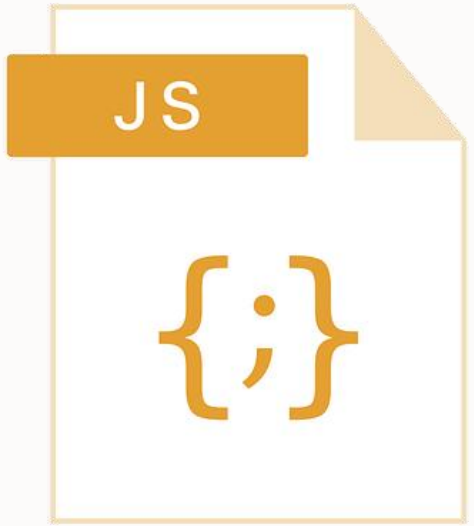


Image courtesy of pixabay.com

Oracle Bots Node SDK

As simple as it gets

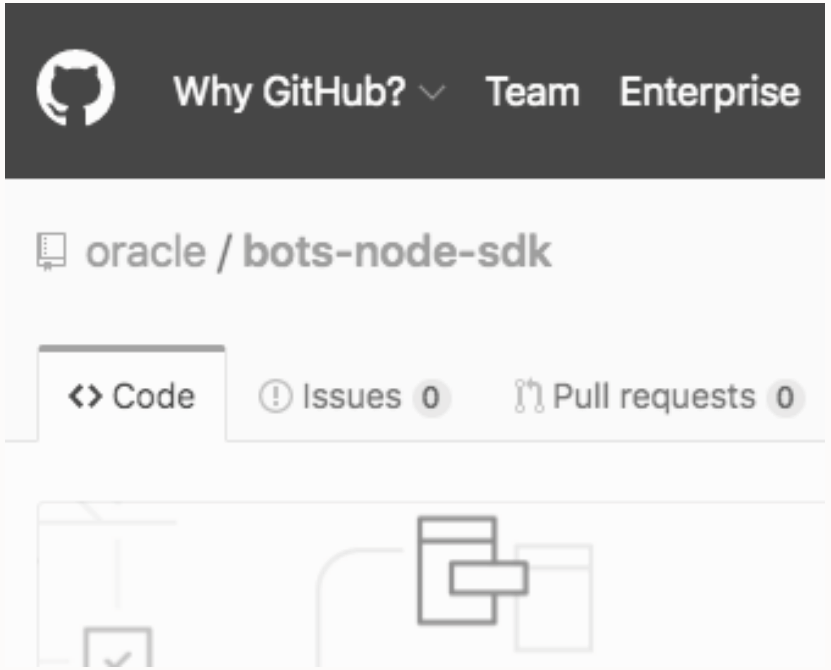
Open source on GitHub

- Command line to create new custom component projects
- Installed using Node Package Manager (NPM)

Contains custom component SDK

- Functions to access bot messages
- Functions to return messages to bot
- Functions to interact with user

Allows component service to be executed locally



github.com/oracle/bots-node-sdk

Step-by-step development process

Have node and npm installed (one time)

Install bots-node-sdk (one time)

- `npm install -g @oracle/bots-node-sdk`

Create custom component service

- Create folder and navigate into it
- `npm init -y`
- `bots-node-sdk init`

  or

  ```
  bots-node-sdk init --component-name
  HelloWorld
  ```

Develop your custom component

- Using JavaScript or TypeScript
  - … `--language typescript`
- Any JS IDE

When done coding, create deployable package

- `bots-node-sdk pack`

Optionally, you can run components locally

- `bots-node-sdk service`
- localhost:3000/components

Adding custom components to a project

```
bots-node-sdk init component <name> custom [components/<sub_dir>]
```

Example:

```
bots-node-sdk init component QueryOrders custom  components/orders
```

Hello World custom component example

```
'use strict';

module.exports = {
  metadata: () => ({
    name: 'HelloWorld',
    properties: {
      human: { required: true, type: 'string' },
    },
    supportedActions: ['weekday', 'weekend']
  }),


  invoke: async (context) => {
    // Retrieve the value of the 'human' component property.
    const { human } = context.properties();
    // Determine the current date
    const now = new Date();
    const dayOfWeek = now.toLocaleDateString('en-US', { weekday: 'long' });
    const isWeekend = [0, 6].indexOf(now.getDay()) > -1;
    // Send two messages, and transition based on the day of the week
    context.reply(`Greetings ${human}`)
      .reply(`Today is ${now.toLocaleDateString()}, a ${dayOfWeek}`)
      .transition(isWeekend ? 'weekend' : 'weekday');
  }
};
```

If you look at existing custom components, you will find a callback signature that is still supported. **It's not legacy, it's just old.**

```
invoke(conversation, done){ … }
```

Commonly used functions of the 'context' object

https://oracle.github.io/bots-node-sdk/CustomComponentContext.html

- Access to flow, skill, profile, user variables
  - `context.get|setVariable(name[,value])`
- Access to bot messages
  - `context.postback(), context.text(), context.rawPayload()`
- Keep or release control
  - `context.keepTurn(true|false)`
- Using resource bundles
  - `context.translate('name of rb key')`
- Print prompts and messages
  - `context.reply(String | message model message)`
- Transition to a next state in flow
  - `context.transition(), context.transition(string)`
- Access to component input parameter
  - `const {prop_name} = context.properties();`
- Access to MessageModel to, optionally, render rich UI
  - `Context.getMessageModel()`

REST service calls in custom components

The Node.js ecosystem offers many similar options for interacting with REST

- https (node core), axios (npm), node-fetch (npm), … and many more

Node fetch is integrated in the bots-node-sdk and recommended for this reason

- Asynchronous REST calls supporting all REST methods
- https://www.npmjs.com/package/node-fetch

Example: Currency Conversion

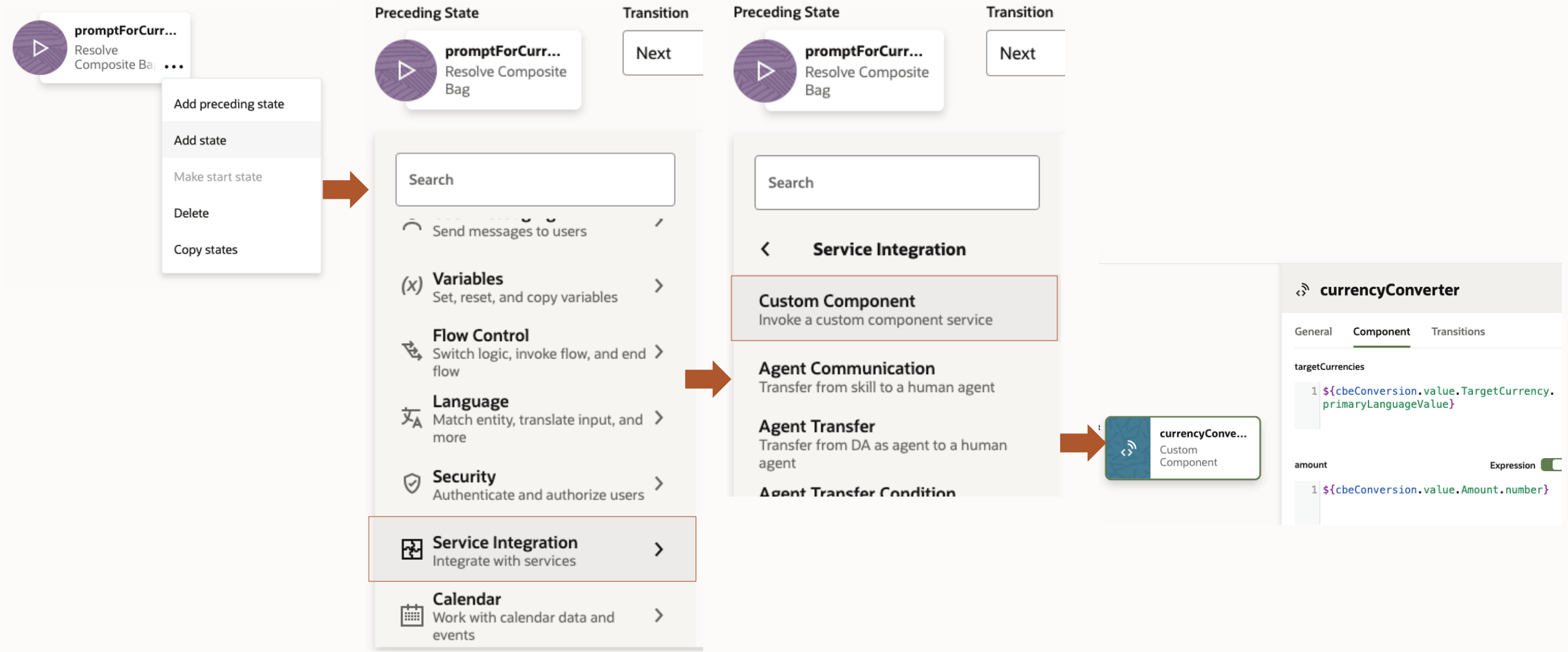
```
const fetch = require("node-fetch")
...

let reqUrl = converterBaseUrl+"?q="+baseCurrency+"_"+targetCurrency+"&compact=ultra&apiKey="+converterApiKey;

...
const restCall = await fetch(reqUrl,{ method: 'GET'});
const response =  await _ restCall.json();

const conversionRate = response[baseCurrency+"_"+targetCurrency];
const result = Math.round(((conversionRate * amount) + Number.EPSILON) * 100) / 100;
...
```

Adding custom components to the dialog flow



Agenda

1. Backend integration overview
2. Built-in Rest service component
3. Custom dialog flow components
4. **Entity event handler**
5. Custom component deployment options
6. Best practices

Entity event handler

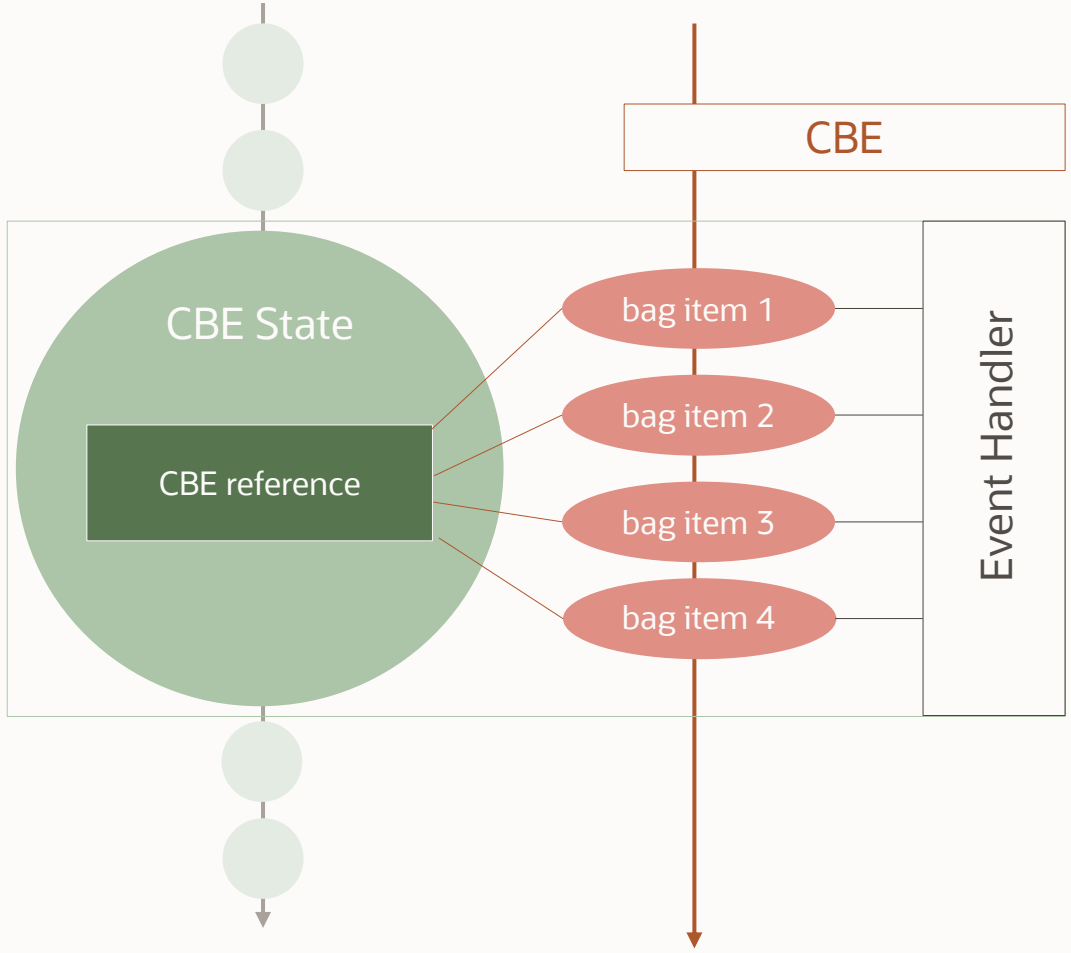
Event-driven approach to interact with custom components

Composite bag entity (CBE)

- Real world object
- Model-driven conversation

Entity event handler (EEH)

- (optional) registered with a CBE
- Events handled by functions in EEH
  - Function written in JavaScript (Node.js)
- Allow developers to interact with CBE



EEH Events

Entity Level

- init
- validate
- publishMessage
- maxPromptsReached
- resolved
- attachmentReceived
- locationReceived

Item Level

- shouldPrompt
- publishPromptMessage
- validate
- publishDisambiguateMessage
- maxPromptsReached

Custom

- postback event with custom payload

Entity event handler use cases

CRUD REST (backend) integration

Custom validation

- E.g., a valid date is not automatically a valid date
- E.g., cross bag-item validation
- E.g., validation based on backend queries

Customize prompts and messages

- E.g., use card layout instead of lists
- Add common buttons (e.g., "cancel button")
- Change content of a message (e.g., add slotted, queried or derived values)

Slot bag item values

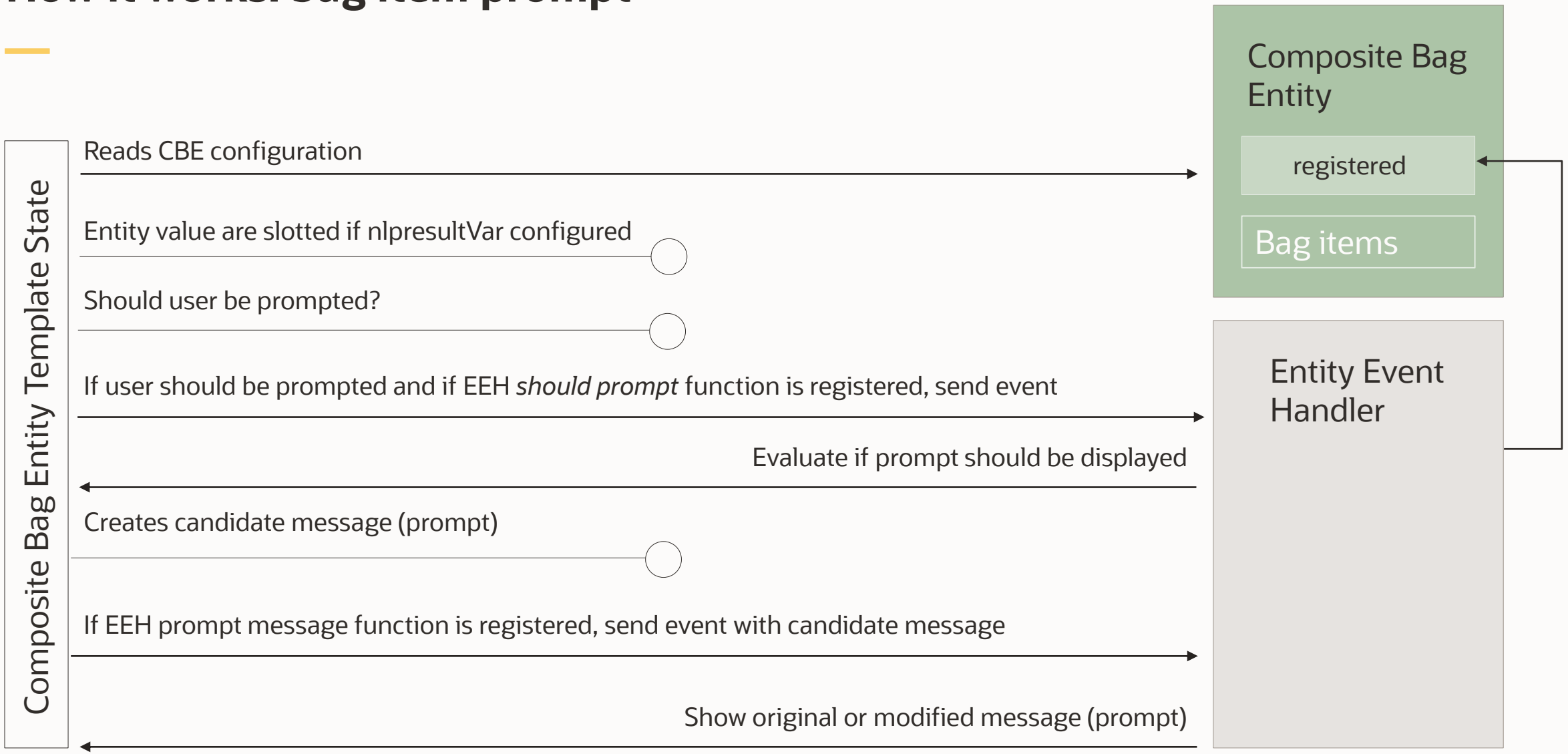
Determine next navigation target

Commonly used functions of the 'context' object

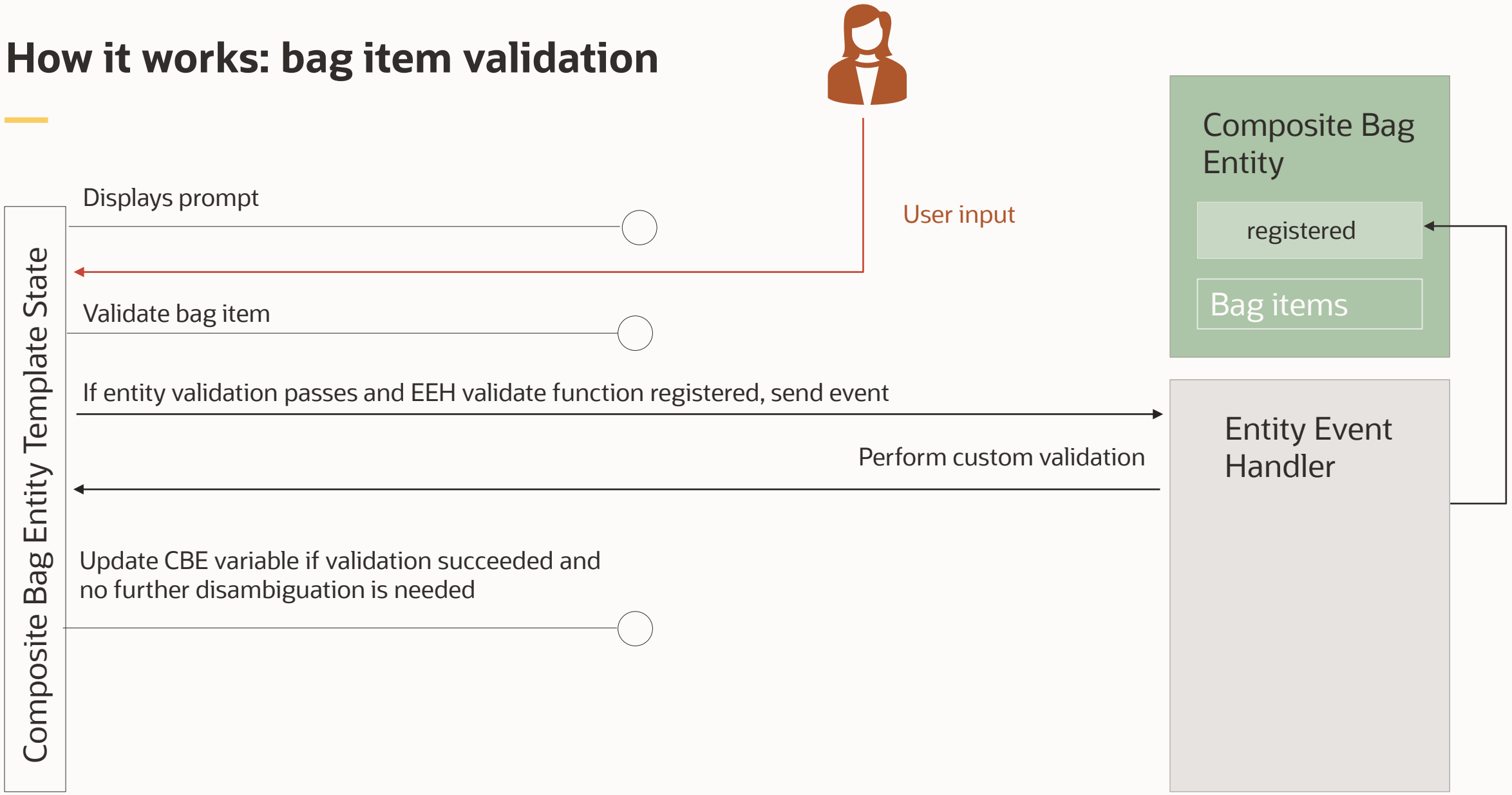
https://oracle.github.io/bots-node-sdk/EntityResolutionContext.html

- Access to flow, skill, profile, user variables
  - `context.get|setVariable(name[,value])`
- Access to bag item values
  - `context.get|setItemValue(name[,value])`
- Use of resource bundles
  - `context.translate('name of rb key')`
- Printing prompts and messages
  - `context.addMessage(string | message model, true | false)`
  - `context.getCandidateMessages(),`
    `context.addCandidateMessages()`
- Transition to a next state in flow
  - `Context.setTransitionAction(string)`
- Using custom properties
  - `context.set|getCustomProperty(name[,value])`
- Access to MessageModel to, optionally, render rich UI
  - `Context.getMessageModel()`

How it works: bag item prompt



How it works: bag item validation



Development choices

Browser development

Easy to use

Context dialog shows event functions that can be added for bag item or entity

Uses embedded container for deployment

Code is saved using "save" button

Source controlled via skill versioning

External JavaScript IDE

Advanced option

Uses bots-node-sdk to create the EEH project

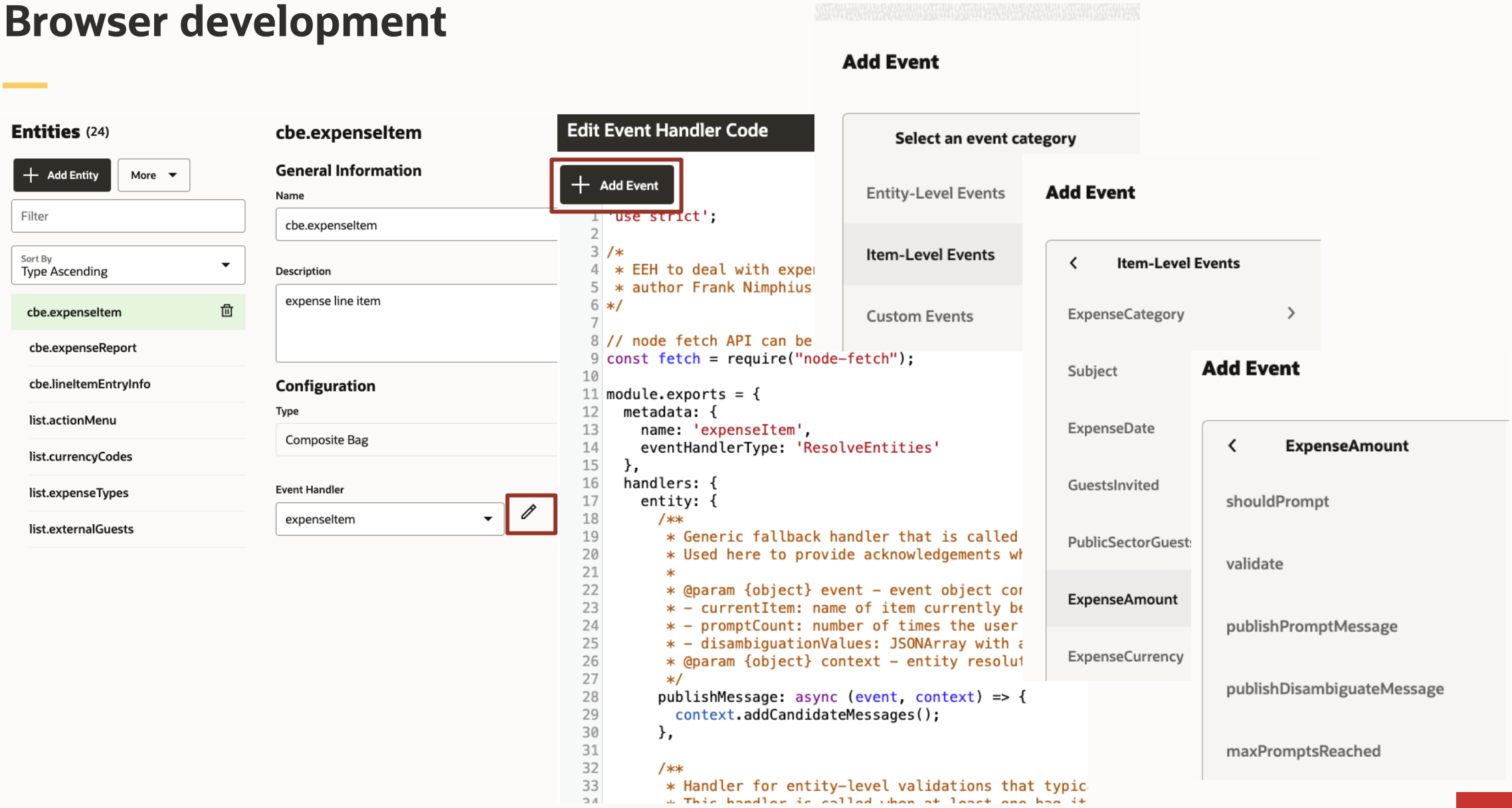
- Creates EEH components

Component needs to be packaged, deployed and registered with composite bag entity

Local code project can be source controlled

Easier to import and manage dependencies

Browser development



External IDE: Using bots-node-sdk to create an EEH project

Have node and npm installed (one time)

Install bots-node-sdk (one time)

- `npm install -g @oracle/bots-node-sdk`

Create EEH custom component service

- Create folder and navigate into it
- `npm init -y`
- `bots-node-sdk init --component-type entityEventHandler`

Develop your custom component

- Using JavaScript or TypeScript
  - … `--language typescript`
- Any JS IDE

When done coding, create deployable package

- `bots-node-sdk pack`

Optionally, you can run components locally

- `bots-node-sdk service`

External IDE: Adding EEH custom components to a project

```
bots-node-sdk init component <name> entityEventHandler [components/<sub_dir>]
```

Example:

```
bots-node-sdk init component OrderEEH entityEventHandler [components/<sub_dir>]
```

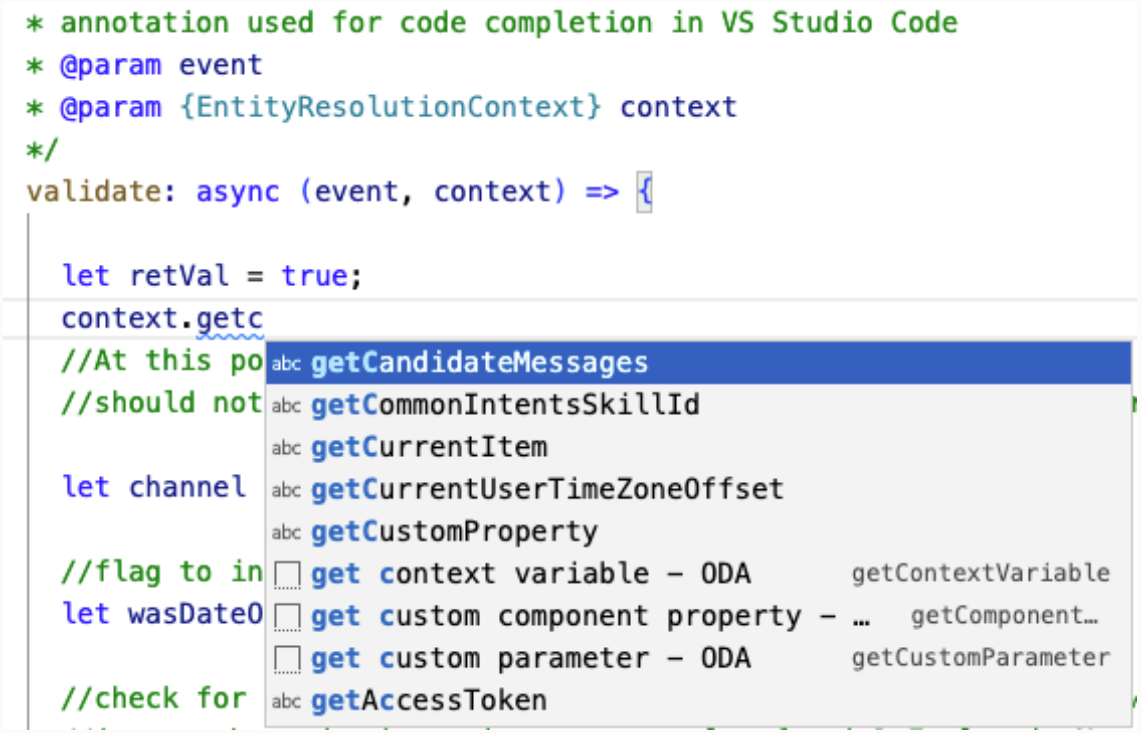
Code completion support in MS Visual Studio Code

Add the following code on top of your custom component (CCS or EEH)

```
// eslint-disable-next-line no-unused-vars
const { EntityResolutionContext } = require("@oracle/bots-node-sdk/lib");
```

Add the following comment on to each function

```
/**
* @param event
* @param {EntityResolutionContext} context
*/
validate: async (event, context) => {
```



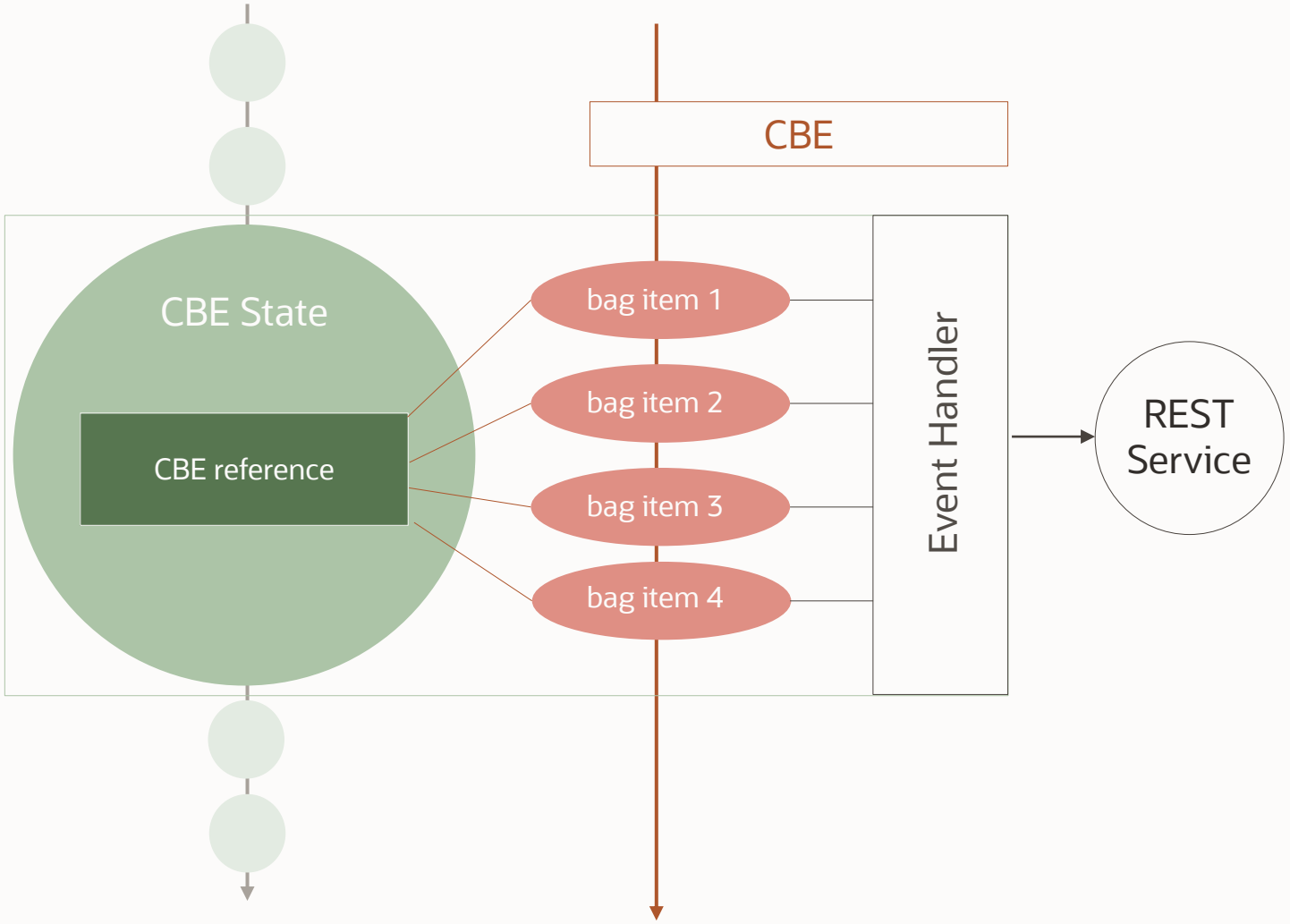
Rest Service calls in EEH

Use Node fetch module

- recommended

REST calls are same as with custom components

- Asynchronous calls
- No need to leave CBE



Agenda

1. Backend integration overview
2. Built-in Rest service component
3. Custom dialog flow components
4. Entity event handler
5. **Custom component deployment options**
6. Best practices

Deployment options & benefits

Built-in Container

- Embedded in skill
- Oracle functions deployed on Oracle tenancy
- Drag & drop deployment
- Ease of administration

External Oracle functions

- Customer tenant
  - Full control
  - Access to OCI services

External Container

- Customer tenant
  - Full control
  - Access to OCI service
  - No limits
- webviews and webhooks

Deployment options & things to consider

Built-in Container

- Embedded in skill
- Oracle functions deployed on Oracle tenancy
- Drag & drop deployment
- Ease of administration

- No administrative access
- Oracle function limits

External Oracle functions

- Customer tenant
  - Full control
  - Access to OCI services

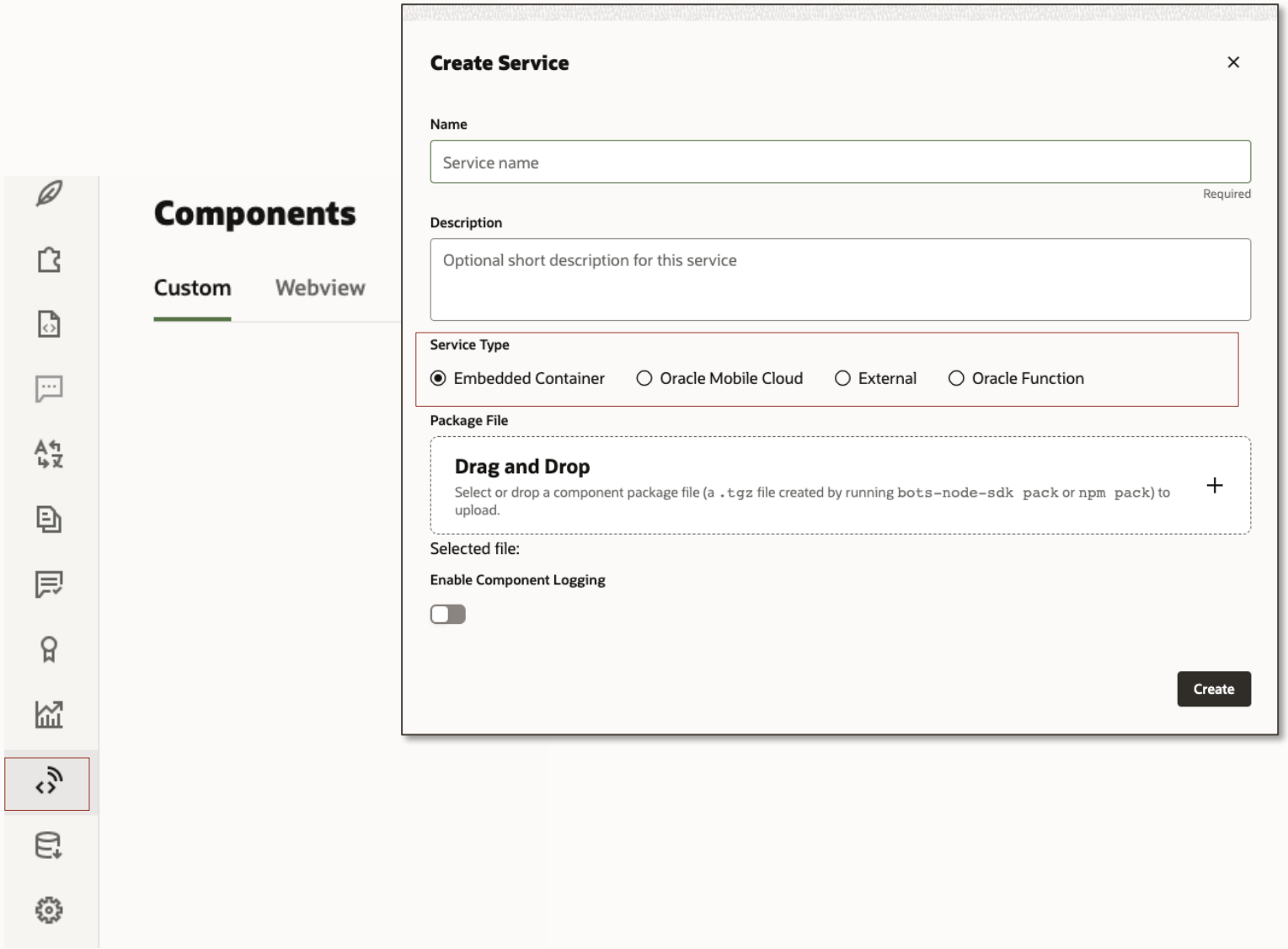
- Additional OCI costs
- Oracle function limits

External Container

- Customer tenant
  - Full control
  - Access to OCI service
  - No limits
- Can host webviews

- Additional OCI costs

Custom component registration



Agenda

1. Backend integration overview
2. Built-in Rest service component
3. Custom dialog flow components
4. Entity event handler
5. Custom component deployment options
6. **Best practices**

Question 1

> Custom components or entity event handlers,
>
> what is best for backend integration?

Custom components if you need backend integration from the dialog flow

Event handlers when that access is required from within the resolution of a composite bag

Question 2

> Custom or built-in REST service component,
> what is best for backend integration?

If you need to make several API calls, or to have a more complex response post-processing, then use custom components

REST service is ideal for single and simple API calls

Question 3

> If custom, where should I deploy to?

If you need to access OCI services from your custom component, then consider deploying it in your own tenancy (OCI Function, Kubernetes or external node server)

If you have requirements to avoid the “cold start”* from the embedded deployment, then a deployment in a Kubernetes cluster or an external node server is a good option

*Embedded deployments use internal OCI Functions which are serverless, hence they have a startup time the first time they are called


ORACLE