german cooperation
DEUTSCHE ZUSAMMENARBEIT

Implemented by

giz Deutsche Gesellschaft für Internationale Zusammenarbeit (GIZ) GmbH

IBM

Make-IT in Africa
inspire. connect. develop.

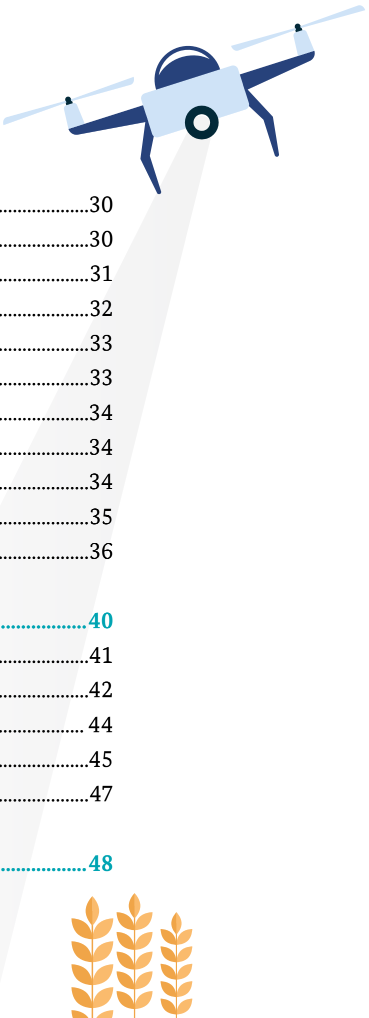
The Interoperability Playbook

How to integrate into a platform

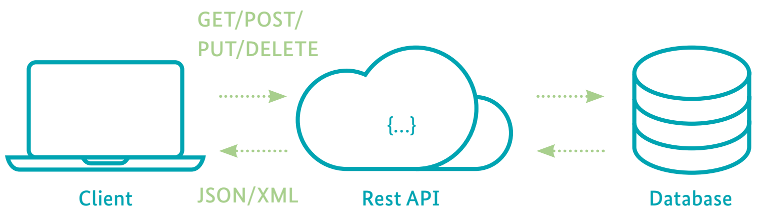
Interoperability Playbook

How to integrate into a platform

Contents



1.0 Interoperability – Basic



1

What is an API?
Am application programming interface (API) is a computing interface - a shared boundary between multiple software parties.

2

What do APIs do?
The API defines the interactions between the software, by specifying:

- The kinds of calls or requests that can be made
- How to make the request
- The data formats to use
- The conventions (protocols) to follow, etc.

3

Why do we use APIs?
APIs simplify programming by abstracting the underlying implemenation and only exposing objects or actions the developer needs.

Benefits of APIs

Application: APIs can access the app components, meaning that the delivery of services and information is more flexible.

Efficiency: When access is provided for an API, the content generated can be published automatically and is available for every channel. It allows it to be shared and distributed more easily.

Adaptation: Needs change over time, and APIs help to anticipate changes. When working with this technology, data migration is better supported, and the information is reviewed more closely. In short, APIs make service provision more flexible.

Scope: With an API, an application layer can be created and used to distribute information and services to new audiences, which can be personalised to create custom user experiences.

Integration: APIs allow content to be embedded from any site or application more easily. This guarantees more fluid information delivery and an integrated user experience.

Automation: With APIs, computers rather than people can manage the work. Through APIs, agencies can update workflows to make them quicker and more productive.

Personalisation: Through APIs any user or company can customise the content and services that they use the most.

API Endpoints

An API endpoint is basically a fancy word for a URL of a server or service. In simple terms, an API endpoint is the point of entry in a communication channel when two systems are interacting. It refers to touchpoints of the communication between an API and a server. The endpoint can be viewed as the means from which the API can access the resources it needs from a server to perform their task.

E.g. in the below API, this endpoint returns a random image from any dog breed:

https://dog.ceo/api/breeds/image/random

We all know that APIs operate through **requests** and **responses.** And when an API requests to access data from a web application or server, a response is always sent back. The location where the API sends a request and where the response emanates is what is known as an **endpoint**.

The endpoint is the most crucial part of the API documentation since it's what the developer will implement to make their requests. Endpoints help to depict the exact location of the resources to be accessed by the API and play a vital role in ensuring that the software, which is interacting with the API is functioning correctly.

An **API** refers to a set of protocols and tools that allow interaction between two different applications. In simple terms, it is a technique that enables third-party vendors to write programmes that can easily interface with each other.

Furthermore an **endpoint** is the place of interaction between applications. API refers to the whole set of protocols that allows communication between two systems, while an endpoint is a URL that enables the API to gain access to resources on a server.

Dog CEO API

All endpoints use the *https://dog.ceo/api/* part of the url:

- some use the **image** resource, e.g. *https://dog.ceo/api/breeds/image/random*
- some use the list **resource**, e.g. *https://dog.ceo/api/breeds/list/all*

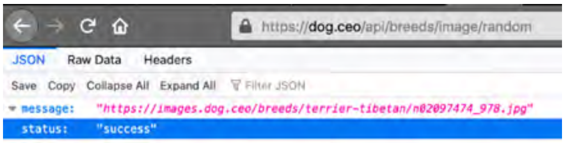
The API documentation can be found here:
https://github.com/ElliottLandsborough/dog-ceo-api

API Call

What is an API Call?
The moment you add an endpoint to a URL and send a request to a server, this is what counts as making an API call. For example, when you log on to any app or ask a question via a browser, you are making an API call.

In a nutshell, an API call is a process that takes place when you send a request after setting up your API with the correct endpoints. Your information is transferred, processed, and feedback is returned.

Using the previous example:
https://dog.ceo/api/breeds/image/random
The moment you enter this URL into your browser address window and press enter, you have made an API call.



```
▾ message:   "https://images.dog.ceo/breeds/terrier-tibetan/n02097474_978.jpg"
  status:    "success"
```

API Keys

An API key, or application programming interface key, is a code that gets passed in by computer applications. The programme or application then calls the API or application programming interface to identify its user, developer, or calling programme to a website.

Application programming keys are normally used to assist in tracking and controlling how the interface is being utilised. Often, it does this to prevent abuse or malicious use of the API in question.

An API key can act as a secret authentication token as well as a unique identifier. Typically, the key will come with a set of access rights for the API that it is associated with.

API keys increase security, enable you to monitor usage of your API, and prevents abuse of API endpoints by limiting who can access them.

HTTP Status Response Codes

HTTP status codes give you information about the request you have submitted and indicates success or failure.

All the Weather Company's APIs are REST web APIs and use HTTP as the communication protocol.

The most frequently used method for the Weather Company's APIs, is GET.
The HTTP status code descriptions can be found in the Weather Company API Common Usage Guide: *https://ibm.co/APICom*

v2/v3 API - HTTP Status Code	Description
200	Success.
204	Data Found for Specific Query. The 204 status Sode will have an empty response body.
400	Bad Request. The request could not be understood by the server due to malformed syntax. This is implemented for all API's. API will reject the request if any invalid parameters are supplied.
401	Unauthorised. The request requires authentication.
403	Forbidden. The server understood the request but the API key is not authorised to perform the requested operation.
404	Not Found. The endpoint requested is not found.
405	Method Not Allowed. For example, sending a POST instead of a GET.
406	Not Acceptable. For example, not accepting gzip compressed responses.
408	Request Timeout. Client did not produce a request within the time server was willing to wait.
500	Internal Server Error. The server encountered an unexpected condition which prevented it from fulfilling the request.
502-504	Service Unavailable Or Gateway Issue. These error codes are returned if the service is temporarily unavailable.

2.0 Interoperability – Intermediate

Introduction to Postman

Postman is a collaboration platform for API development.

It is also an API client that enables you to quickly send API requests directly within Postman.

Collections

In Postman, **collection** refers to a group of API requests that are already saved in the Postman and can be arranged into folders. Any number of folders can be created inside a collection.

Putting similar requests into folders and collections helps the client with better organisation and documentation of their requests.

All the API's requests can be stored and saved within a collection, and these collections can be shared amongst the team in the Postman workspace.

Environments

A collection of key-value pairs is called an **environment**. Each name of the variable represents its keys and referencing the name of the variable allows you to access its value.

Variables

Postman **variables** work in the same way as programming variables. You can store the values in variables and can use it throughout in requests, environments, collections and scripts.

Variables in Postman increase the user's efficiency to work and decrease the errors.

You use variables by putting the name of the variable in double curly brackets: {{variable_name}}

Install Postman

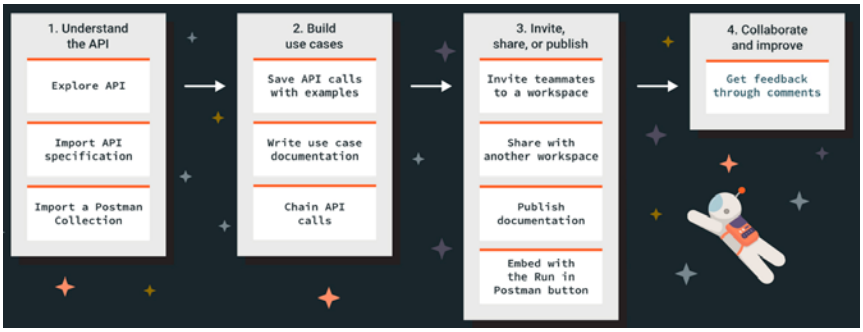
Postman is a collaboration platform for API development.
It is also an API client that enables you to quickly send API requests directly within Postman. That is mainly how we will use it in this course.

Installation Instructions

If you haven't already installed it, here are the installation instructions. Please install it now!

1. Go to *https://www.postman.com/downloads/*
2. Click on 'Download the app'
3. Download the right native app for your platform

API Exploration with Postman


1. Understand the API
Explore API
Import API specification
Import a Postman Collection
2. Build use cases
Save API calls with examples
Write use case documentation
Chain API calls
3. Invite, share, or publish
Invite teammates to a workspace
Share with another workspace
Publish documentation
Embed with the Run in Postman button
4. Collaborate and improve
Get feedback through comments

Import a Postman Collection and Environment:

1. Download and unzip the D4A Interop Intermediate - Masterclass.zip file
2. Open Postman
3. In the top left-hand corner, click 'Import'
4. Choose 'Upload Files'
5. Navigate to the GIZ_Postman folder and select all JSON files for import
6. Click 'OK'
7. A set of collections (folders) should appear on the left-hand side

Useful Links
Visualising GeoJSON files: *http://geojson.io/*

Rest and SOAP APIs

SOAP	REST
[**S**imple **O**bject **A**ccess **P**rotocol] is a messaging protocol used for exchanging structured information [XML data] over a network.	[**RE**presentational **S**tate **T**ransfer] is a standardised architectural style that can be used when creating a web API.

REST vs. SOAP Characteristics

SOAP APIs

SOAP has three primary characteristics:

Extensibility – The protocol allows for extensions that introduce more powerful features.

Neutrality – SOAP can operate over a wide range of protocols like UDP, JMS, SMTP, TCP, and HTTP.

Independence – Just about any programming language can use SOAP.

However, you may not use JSON with SOAP. The protocol is strict and the only option for data is XML. It's for this reason alone that just about everyone recommends REST instead of SOAP. JSON is easier to work with than XML, so REST becomes the preferred option.

REST APIs

REST is not a protocol, a tool, or library, but rather an architectural style of web service that provides a channel of communication between systems or computers on the internet.

Implementation – Convenient with JavaScript and allows easy implementation

Flexibility – Can use several standards like HTTP, URL, JSON, and XML

Bandwidth – Uses less bandwidth and resources since it deploys multiple standards

Data is not constrained to resources or methods. Therefore, it can make multiple types of calls and return various data formats.

Guiding Principles of REST

Client–server – By separating the user interface concerns from the data storage concerns, we improve the portability of the user interface across multiple platforms and improve scalability by simplifying the server components.

Stateless – Each request from client to server must contain all the information necessary to understand the request, and cannot take advantage of any stored context on the server. Session state is therefore kept entirely on the client.

Cacheable – Cache constraints require that the data within a response to a request be implicitly or explicitly labelled as cacheable or non-cacheable. If a response is cacheable, then a client cache is given the right to reuse that response data for later, equivalent requests.

Uniform interface – By applying the software engineering principle of generality to the component interface, the overall system architecture is simplified, and the visibility of interactions is improved. REST is defined by four interface constraints: identification of resources; manipulation of resources through representations; self-descriptive messages; and, hypermedia as the engine of application state.

Layered system – The layered system style allows an architecture to be composed of hierarchical layers by constraining component behaviour, such that each component cannot "see" beyond the immediate layer with which they are interacting.

Code on demand (optional) – REST allows client functionality to be extended by downloading and executing code in the form of applets or scripts. This simplifies things for clients by reducing the number of features required to be pre-implemented.

Why JSON?

JSON is based on the object notation of JavaScript. However, it does not require JavaScript to read or write because it is made in text format, which is language independent and can be run everywhere.

Fast and lightweight
JSON syntax is very easy to use. Since its syntax is very small and lightweight it parses and executes responses faster than XML.

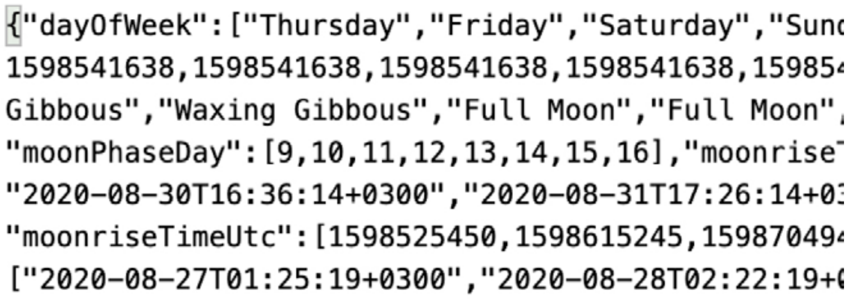
Human-readable
A chunk of JSON text can be made easily readable and understandable even for non-programmers.

Versatile for data exchange
JSON is a text format that is completely language independent but uses conventions that are familiar to programmers of the C-family of languages, including C, C++, C#, Java, JavaScript, Perl, Python, and many others. These properties make JSON an ideal data-interchange language.

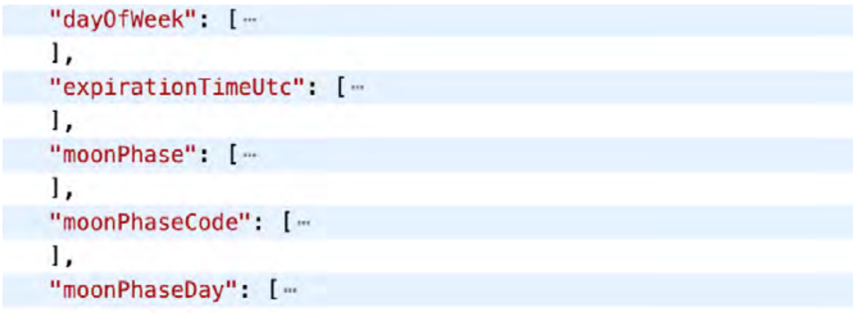
Parsing JSON

The word parsing can be used to mean interpreting. Parsing JSON means interpreting the data with the specific language that you are using at that moment. JSON is usually read as a string called the JSON string. This is a string that follows the specifications of JSON. When we parse JSON, it means we are converting the string into a JSON object by following the JSON specification, where we can then use it in any way we wish.

Before parsing JSON, you only have a regular string that you cannot access the data encoded within. After parsing, it is converted into a JavaScript object where that allows you to access the various data inside. Therefore, parsing JSON is a method of making it more accessible than it is before parsing.



```
{"dayOfWeek":["Thursday","Friday","Saturday","Sund
1598541638,1598541638,1598541638,1598541638,159854
Gibbous","Waxing Gibbous","Full Moon","Full Moon",
"moonPhaseDay":[9,10,11,12,13,14,15,16],"moonrise
"2020-08-30T16:36:14+0300","2020-08-31T17:26:14+03
"moonriseTimeUtc":[1598525450,1598615245,159870494
["2020-08-27T01:25:19+0300","2020-08-28T02:22:19+0
```



```
    "dayOfWeek": [⋯
    ],
    "expirationTimeUtc": [⋯
    ],
    "moonPhase": [⋯
    ],
    "moonPhaseCode": [⋯
    ],
    "moonPhaseDay": [⋯
```

HTTPS Methods

GET

GET is used to retrieve and request data from a specified resource in a server. GET is one of the most popular HTTP request techniques. In simple words, the GET method is used to retrieve whatever information is identified by the Request-URL.

HEAD

The HEAD technique requests a reaction that is like the GET request but doesn't have a message-body in the response. The HEAD request method is useful in recovering meta-data that is written according to the HTTP headers, without transferring the entire content. The technique is commonly used when testing hypertext links for accessibility, validity, and recent modification.

POST

Another popular HTTP request method is POST. In web communication, POST requests are utilised to send data to a server to create or update a resource. The information submitted to the server with the POST request method is archived in the request body of the HTTP request. The HTTP POST method is often used to send user-generated data to a server. One example is when a user uploads a profile photo.

3.0 Interoperability – Masterclass

An API Approach

APIs are a core building block of digital businesses. They enable a company to leverage existing assets and make them available in a variety of ways.

USERS

customers
partners
developers
internal users

DEVICES

tablets
handsets
dekstops

CHANNELS

in-store
online
call-centre
phone

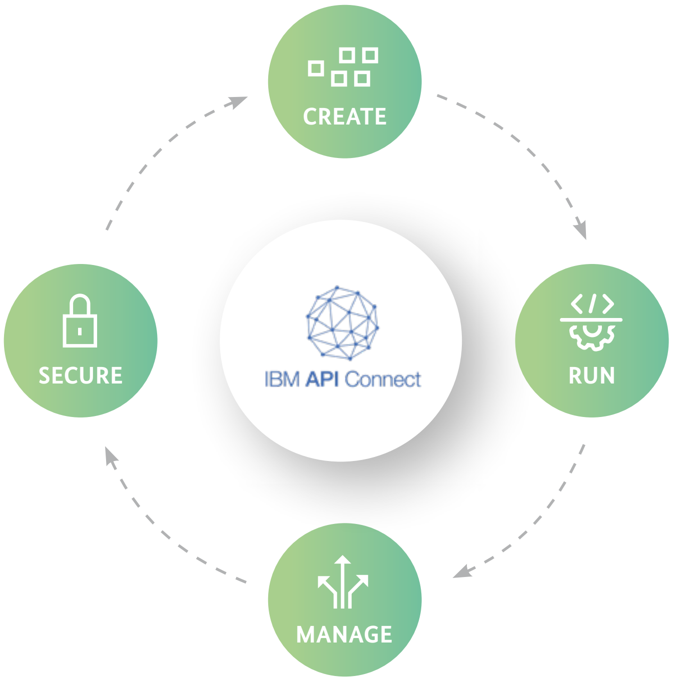
An API can become the primary entry point for enterprise services, for its own website and applications, as well as for partner and customer integrations. Some of the key areas that are driving API adoption are:

- Supporting mobile applications and enterprise app stores through APIs while hiding the complexity of underlying systems
- Extending business functionality and systems into the developer eco-system to drive development of apps with cloud and social mashups, which in turn fosters innovation
- Embracing APIs as a form of new revenue streams

API Platforms

APIs are best served through a platform that supports design, development, deployment, operations, and support. The platforms can be based on conventional (hosted, in-premise, or on cloud) as well as a pay-as-you-go cloud model.

An API platform is an API infrastructure that is ready to build and run APIs with minimum features and common services required of web APIs; with elements of the tech stack that digital enterprises need, such as caching and security, and preferably with a built-in support for API development and management. Ideally, it is configured in the standard way and validated.



An API platform comprises of API layer, API governance, API design time components, API discovery and documentation etc., as well as the ability to address varied demands on security, performance, governance, stability, flexibility and agility.

Examples of API platforms include Sencha, Mashery, MuleSoft and IBM App Connect, each catering to a different level of enterprise and developer needs.

CREATE

Create
Develop and write the API definition and implementation, and test the API

RUN

Run
Package and deploy the API. Ensure that the API is hosted securely on a stable platform.

MANAGE

Manage
Create and manage self-service portals that expose the API to API consumers. Monitor the set of rules and conditions that govern the API to ensure it is fulfilling its intended purpose, and make adjustments if necessary. Retire and archive the API when appropriate.

SECURE

Secure
Incorporate access control, monitoring, and logging to properly secure the API.

Principles of API Integration

When you are looking at integrating APIs into your solution, there are a few things to think about:

- Why are you looking to integrate this API?
- What's the value it adds to your business or your users?
- What will the API enable that you are not currently able to do?
- What costs will be associated with the API – both initial costs of integration, and ongoing costs of any licenses?
- What dependencies are you building into your solution by integrating this API?
- Does it drive revenue?
- Does it improve customer experience?
- Does it reduce costs or improve efficiencies?

API Design Principles

Think about it the way you think about an everyday object. Have empathy with your end user, and understand the context that you will be implementing the API in.

Purpose
- What is the purpose – what is it that your users want to do with the API?
- How complicated is your interface?
- Only expose the internal workings of your API when it makes sense to the user

Usability
- Name APIs and parameters in a straightforward manner
- Only request minimal and straightforward inputs
- Aggregate actions to minimise additional steps
- Make your API consistent in itself; consistent across the organisation, standards, and common practices

Constraints
- Design for your user context, e.g. latency over mobile networks
- Understand the kind of calls clients want to make (batch/bulk)

API Design

Designing APIs
If you are looking at an API-first business, you should also start looking at how you design APIs.

Your API is the first and most important way to access and interact with your product, the API needs to be managed and designed deliberately. **Just as you spend time on designing your graphical user interface, invest time to design your API**: what it exposes, what it does, and how it grows.

Your implementation will change as your application evolves and you optimise, refactor and grow the functionality. Your API, however, should not change frequently, but instead grow slowly and deliberately.

Documentation

Your API needs to be easily understood by people that have not been involved in its creation. That means documentation.

Usable API documentation is an essential prerequisite to making it consumable by humans.

When it comes to documentation for APIs, create structured documentation. Follow a standard pattern for URLs, resource types, request methods, headers, request parameters, and response formats. This will make it easier to explore and understand functionality.

Streaming Data

Streaming data refers to data that is **continuously generated**, usually in **high volumes**, and at **high velocity**. A streaming data source would typically consist of a stream of logs that record events as they happen – such as a user clicking on a link in a web page, or a sensor reporting the current temperature.

Common examples of streaming data include:

- IoT sensors
- Server and security logs
- Real-time advertising
- Click-stream data from apps and websites

In these cases, we have end devices that are continuously generating thousands or millions of records, forming a data stream – unstructured or semi-structured – and are most commonly JSON or XML key-value pairs.

A single streaming source will generate massive amounts of these events every minute. In its raw form, this data is very difficult to work with as the lack of schema and structure makes it difficult to query with SQL-based analytic tools; instead, data needs to be **processed, parsed, and structured** before any serious analysis can be done.

Benefits of Data Streaming Integration

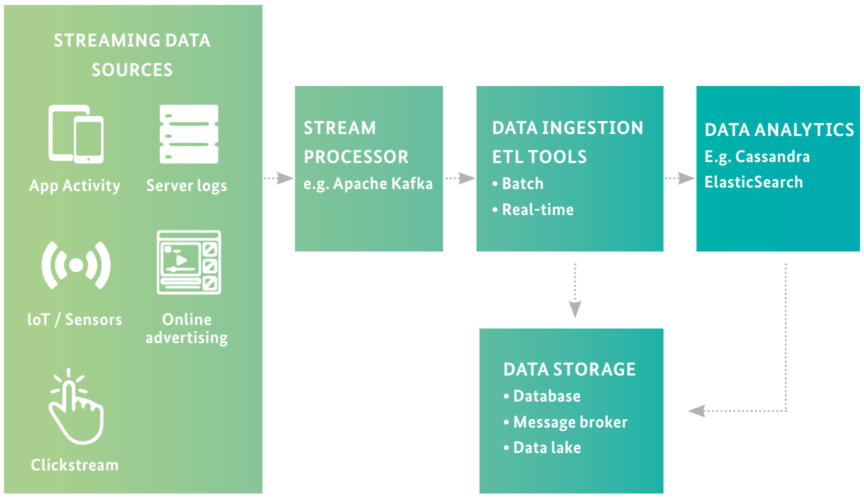
• **Able to deal with never-ending streams of events**—some data is naturally structured this way. Traditional batch processing tools require stopping the stream of events, capturing batches of data, and combining the batches to draw overall conclusions. In stream processing, while it is challenging to combine and capture data from multiple streams, it lets you derive immediate insights from large volumes of streaming data.

• **Real-time or near-real-time processing**—most organisations adopt stream processing to enable real time data analytics. While real time analytics is also possible with high performance database systems, often the data lends itself to a stream processing model.

• **Detecting patterns in time-series data**—detecting patterns over time, for example looking for trends in website traffic data, requires data to be continuously processed and analysed. Batch processing makes this more difficult because it breaks data into batches, meaning some events are broken across two or more batches.

• **Easy data scalability**—growing data volumes can break a batch processing system, requiring you to provision more resources or modify the architecture. Modern stream processing infrastructure is hyper-scalable and able to deal with gigabytes of data per second with a single stream processor. This allows you to easily deal with growing data volumes without infrastructure changes.

Data Streaming Integration Components



Key Building Blocks of Data Streaming Integration

1. The Message Broker / Stream Processor

This is the element that takes data from a source, called a producer, translates it into a standard message format, and streams it on an ongoing basis. Other components can then listen in and consume the messages passed on by the broker. Apache Kafka is a popular open source solution.

2. Batch and Real-time ETL (extract-transform-load) Tools

Data streams from one or more message brokers need to be aggregated, transformed, and structured before data can be analysed with SQL-based analytics tools. This would be done by an ETL tool or platform that receives queries from users, fetches events from message queues, and applies the query to generate a result – often performing additional joins and transformations on aggregations on the data. The result may be an API call, an action, a visualisation, an alert, or in some cases a new data stream. A few examples of open-source ETL tools for streaming data are Apache Storm, Spark Streaming and WSO2 Stream Processor.

3. Data Analytics / Serverless Query Engine
After streaming data is prepared for consumption by the stream processor, it must be analysed to provide value. There are many different approaches to streaming data analytics. Some of the most commonly used tools for streaming data analytics are Elasticsearch and Cassandra.

4. Streaming Data Storage
Given the low cost of storage, most organisations today are storing their streaming event data. This can be done in a database or data warehouse, in the message broker, or in a data lake.

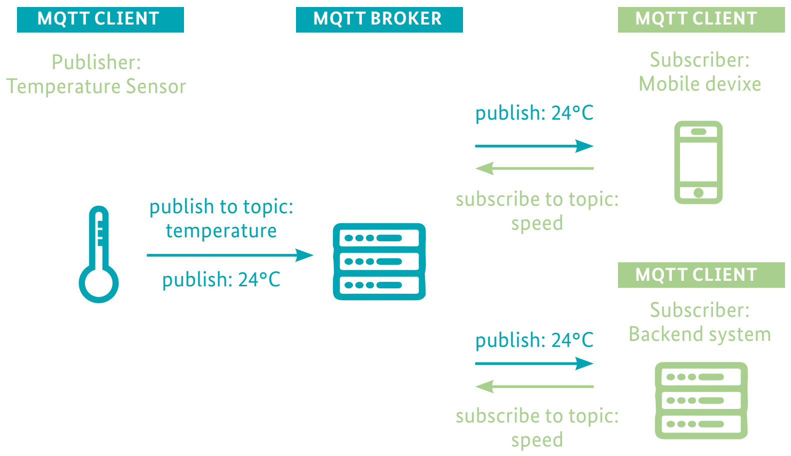
MQTT

MQTT is an open messaging protocol for the Internet of Things (IoT). It is designed as an extremely lightweight publish/subscribe messaging transport that is ideal for connecting remote devices with a small code footprint and minimal network bandwidth.

MQTT is no longer considered an acronym – it is simply the name of the protocol.

MQTT is used in a wide variety of industries, such as automotive, manufacturing, telecommunications, oil and gas, etc.

MQTT Architecture



MQTT vs. HTTP

It is different to HTTP in that a client doesn't have to pull the information it needs, but the broker pushes the information to the client, in the case that there is something new.

Therefore, each MQTT client has a permanently open connection to the broker. If this connection is interrupted by any circumstances, the MQTT broker can buffer all messages and send them to the client when it is back online.

As mentioned before, the central concept used in MQTT to dispatch messages, are topics.

A topic is a simple string that can have more hierarchy levels, which are separated by a slash. A sample topic for sending temperature data of the living room could be house/living-room/temperature.

The client can subscribe to the exact topic or use a wildcard. The subscription to house/+/temperature would result in all messages being sent to the previously mentioned topic house/living-room/temperature as well as any topic with an arbitrary value in the place of living room, for example house/kitchen/temperature.

The plus sign is a **single level wild card** and only allows arbitrary values for one hierarchy. If you need to subscribe to more than one level, for example to the entire subtree, there is also a **multilevel wildcard** (#). It allows you to subscribe to all underlying hierarchy levels. For example, house/# is subscribing to all topics beginning with house.

Why MQTT?

Lightweight and Efficient

MQTT clients are very small and require minimal resources so can be used on small microcontrollers. MQTT message headers are small to optimise network bandwidth.

Bi-directional Communications

MQTT allows for messaging between device to cloud and cloud to device. This makes for easy broadcasting messages to groups of things.

Scale to Millions of Things

MQTT can scale to connect with millions of IoT devices.

Reliable Message Delivery

Reliability of message delivery is important for many IoT use cases. Therefore, MQTT has defined three ‘quality of service’ levels: 0 – once at most; 1 – at least once; 2 – exactly once.

Support for Unreliable Networks

Many IoT devices connect over unreliable cellular networks. MQTT’s support for persistent sessions reduces the time to reconnect the client with the broker.

Security Enabled

MQTT makes it easy to encrypt messages using TLS and authenticate clients using modern authentication protocols, such as OAuth.

MQTT DEMO – Health and Safety IoT

https://ets.mybluemix.net/catalogue//demos/healthAndSafetyIoT/

Business Scenario
On a construction site there are many health and safety requirements that workers must follow, but it is often very difficult for a construction site manager to monitor and enforce these rules.

If any serious incidents occur on-site, workers may be at high risk of long-term injury if they are not seen to very quickly. Such an example would be if something fell onto the head of a worker and knocked them unconscious.

Business Value
Implementing a solution to address this problem shows the value of having a connected world. By utilising the sensor and IoT architecture, it is then easy to plug into such a system that can offer real insight into the performance and health and safety of the site's workers.

The Solution
Finding an IoT-based way to address this challenge involves sensors, a platform, and putting all the pieces together.

MQTT is used to send regular data from sensors that are embedded in the everyday equipment and clothing that the workers wear. This data is collated onto the IBM Watson IoT platform and run through NodeRED, to pick out the relevant data.

Events can then be sent to any platform of choice, whether that be a smartwatch being worn by the site manager, or in this case a centralised UI/Dashboard that the site manager can monitor.

Graph QL

GraphQL is a query language for APIs. GraphQL APIs are becoming more popular with developers because of some of the benefits they offer over RESTful APIs. One of the biggest benefits is that GraphQL allows for smarter and more precise querying which is especially useful when working with large APIs that return a lot of data.

GraphQL enables users to specify exactly what data they get back in their response – nothing more, and nothing less – and it allows querying for multiple fields in a single request.

GraphQL was developed to cope with the need for more flexibility and efficiency in client-server communication.

Data Fetching with GraphQL

Data Fetching with REST vs GraphQL
With a REST API, you would typically gather the data by accessing multiple endpoints. E.g. if you need to fetch information about a user of a service, you may need to call several endpoints. These could be:

- /users/<id> endpoint to fetch the initial user data
- /users/<id>/posts endpoint that returns all the posts for a user
- /users/<id>/followers that returns a list of followers per user

In GraphQL on the other hand, you'd simply send a single query to the GraphQL server that includes the concrete data requirements. The server then responds with a JSON object where these requirements are fulfilled.

GraphQL vs. REST

GraphQL can provide some significant querying efficiencies compared to REST. Send a GraphQL query to your API and get exactly what you need, nothing more and nothing less. It solves the problem of fetching too much data, or not enough data, and having to make multiple requests.

GraphQL queries always return predictable results. Apps using GraphQL are fast and stable because they control the data they get, not the server.

GraphQL was developed to cope with the need for more flexibility and efficiency. It solves many of the shortcomings and inefficiencies that developers experience when interacting with REST APIs.

GraphQL Development

GraphQL means a single endpoint for all the clients who want to get data from the server. That means you can fetch everything with a single request.

This query works by sending a POST request with all the data requirements for the client. This will be processed by the server and only exactly what is asked for will be returned.

This also allows for rapid iterations, as changes on the client-side can be made without any extra work on the server. Since clients can specify their exact data requirements, no backend engineer needs to adjust when the design and data needs on the frontend change.

GraphQL APIs only expose a single endpoint, and you use the query definition as a client to define the data you want.

GraphQL Schema Definition Structure

GraphQL uses a strong type system to define the capabilities of an API. All the types that are exposed in an API are written down in a **schema** using the GraphQL Schema Definition Language (SDL). This schema serves as the contract between the client and the server to define how a client can access the data.

Once the schema is defined, the teams working on the frontend and backends can do their work without further communication, since they both are aware of the definite structure of the data that's sent over the network.

Frontend teams can easily test their applications by mocking the required data structures. Once the server is ready, the switch can be flipped for the client apps to load the data from the actual API.

What are Webhooks?

Webhooks are API-like concepts that are being utilised to allow communication between applications. They are an incredibly essential and resource-light means of implementing event reactions.

Webhooks are restricted to, you guessed it, **the web**. This implies that they must communicate over a web protocol—HTTP in almost every case.

Webhooks allow you to receive real-time data from an app when an event happens. Web services use webhooks to provide other services with real-time information through HTTP POST when something happens.

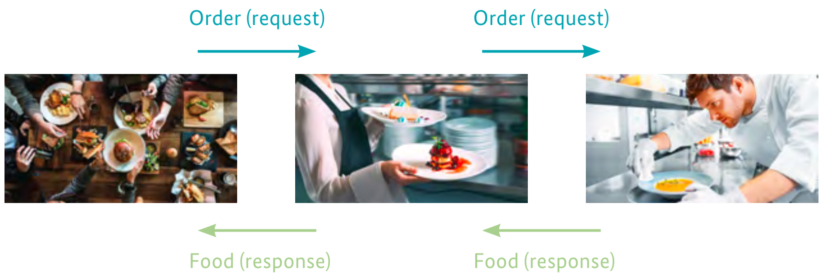
However, webhooks and APIs differ in how they make requests. For instance, APIs will place calls for data whether there's been a data update response, or not. While webhooks receive calls through HTTP POSTs only when the external system you're hooked to has a data update.

Webhooks are commonly used to perform smaller requests and tasks, however, there are situations where a webhook is more appropriate than an entire API.

One common scenario is when your app or platform demands real-time updates, but you don't want to waste your resources. In this instance, a webhook framework would be beneficial.

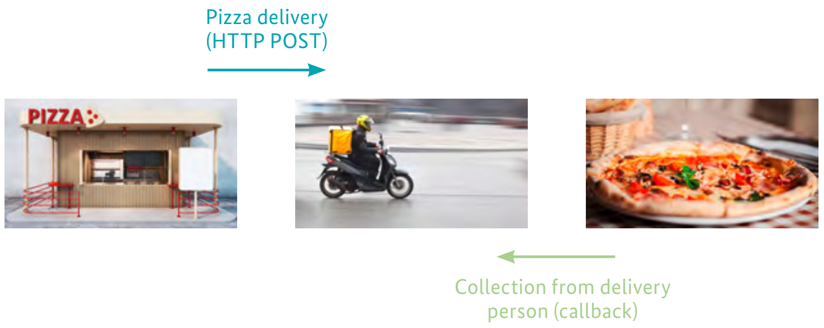
To implement webhooks all you must do is register a URL with the company providing the service you're requesting data from. That URL will accept data and can activate a workflow to turn the data into something useful. In most cases, you can even specify the situations in which your provider will deliver you the data.

Webhooks vs. APIs – Restaurant Example



APIs

1. When you (a programme) go to a restaurant, a waiter (the API) will take your order
2. The waiter goes to the kitchen (server) and delivers your order (API request)
3. The waiter returns to your table with the food (API response)


Pizza delivery (HTTP POST)
PIZZA
Collection from delivery person (callback)

Webhooks

1. You call up a restaurant and order pizza for delivery
2. You tell them exactly what pizza you want and give them your address (setting up the webhook)
3. The pizza is ready to be delivered (event)
4. The delivery person picks up the pizza, delivers it to your door and rings the doorbell (webhook provider collects data, ready to share along listener)
5. You come to the door to get your pizza (POST requests fires as callback)
6. You do what you want with the pizza (action within third party app)

Webhook Example

An Example: GitHub

A good example of webhooks in action is GitHub, which lets you set up webhooks to subscribe to various events that you might be interested in. So, if I want to be notified when a pull request is created, I can tell GitHub where my webhook is hosted, and then whenever a pull request is created on my project, GitHub will make an HTTP request to my callback URL, posting a JSON object that contains information about the pull request.

I can then write my own custom code that does whatever I want with that information; whether it's sending myself a text, or automatically accepting the pull request (don't do that), or triggering an automated CI build that will validate the quality of the pull request. The point is, I can do whatever I like because GitHub has provided me with an extensibility point.

Probably the main reason that webhooks have only recently become more ubiquitous, is that they require you to have a server to hand which you can use to listen for them.

While webhooks aren't a standard, most people implement them in the same way: when an event happens in your programme that another programme is subscribed to, loop though each subscription and check whether it has access to view that thing, and if so, send an HTTP POST request to your subscriber's URL.

It is then up to the subscriber to carry out an action on the webhook if it is needed.

We're going to show a super simple webhook example using GitHub and a webhook tester. This simplifies things because it means we don't have to set up a listener URL on a server.

We're going to use GitHub's webhooks. More specifically, we want **Webhook Tester** to be notified when we make a comment on a commit within a certain repository.

GitHub
POST request
Webhook.site

Workflow order:
1) Set up the webhook in GitHub
2) Test the webhook by commenting on commits in the repository
3) An HTTP POST request gets submitted to webhook.site
4) The request is visible in the call log

Should My Service Offer Webhooks?

Does your service let users initiate long-running operations? Are there events in your system that the users of your service might like to be notified in real-time about? If so, offering webhooks will make your system much easier for consumers to work with.

However, my recommendation is to **make webhooks optional** in your design. They should not provide any information that could not also be retrieved through polling. That way, users who don't want to (or can't) host a webhook will still be able to use your service.

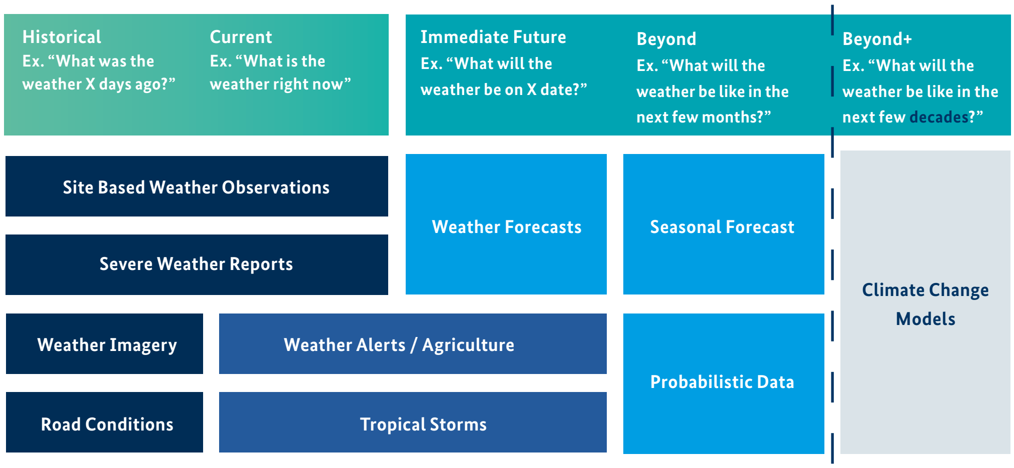
Also, if you're going to offer webhooks, **think carefully about security**. Make sure you follow the best practices and consider what the implications would be if a fake webhook was received by the system.

"Webhook equivalents" for GraphQL are called **subscriptions**.

API Documentation for Packages in This Workshop
Core: *https://ibm.co/TWCdac*
Agriculture: *https://ibm.co/TWCagr*
History on Demand: *https://ibm.co/TWChod*
Passport Advantage - Overall API Documentation: *https://docs.google.com/document/d/15Ru_3wdMgpbM4aOCm-4qNAnRfjx2w-Ruw3lnr8Hnodk/edit?usp=sharing*

Annex

Weather Data Packages


Historical
Ex. "What was the weather X days ago?"
Current
Ex. "What is the weather right now"
Immediate Future
Ex. "What will the weather be on X date?"
Beyond
Ex. "What will the weather be like in the next few months?"
Beyond+
Ex. "What will the weather be like in the next few decades?"
Site Based Weather Observations
Severe Weather Reports
Weather Forecasts
Seasonal Forecast
Climate Change Models
Weather Imagery
Weather Alerts / Agriculture
Probabilistic Data
Road Conditions
Tropical Storms

Available Packages for Testing

DATA PACKAGE: CORE

Core provides access to the most useful weather data to get started

What does it include?

2-Day Hourly Forecast – Forecasts for next 48 hours starting from the current time.

Daily Forecast – Forecast for 24-hour periods starting today for the next 3, 5, 7 and 10 days including forecasts for the daytime and nighttime segments.

Intraday Forecast – Forecasts for 6-hour periods starting today for the next 3,5,7 and 10 days including forecasts for the morning, afternoon, evening and overnight segments.

Site-Based Observations – Observed weather data (temperature, wind direction and speed, humidity, pressure, dew point, visibility, and UV Index) as well as a sensible weather phrase and its matching weather icon that are collected from METAR and SYNOP observation stations worldwide.

Time Series-Based Observations – Observed weather data from site based observation stations for the previous 24 hours.

Weather Alert Headlines And Details – Govermental-issued alert headlines and details.

Location-Mapping Services – Utility API for finding locations per zip code, geocode, city, internationalised state, region district, or province.

Current Conditions And Forecast Imagery Layers – detailed set of tile based raster products derived from observation and forecast data ready to be applied to your base map.

Radar And Satellite Layers – Provides selected radar and satellite raster products.

DATA PACKAGE: AGRICULTURE

This is a robust set of agriculture specific APIs providing hourly forecasts out to 15 days, built using a weighted blend of the following weather models: GFS Operational & Ensemble, ECM Operational, NAM & Deep Thunder.

What does it include?

Reference Evapotranspiration – Forecast based on idealised conditions. Utilising similar implementation to which is used by National Weather Service.

Core Specific Reference Evapotranspiration – 103 different crops supported Based on the % Crop Maturity and Crop Type specified in the API call.

Model Evapotranspiration – Forecast based on actual land use conditions.

Soil Moisture/Soil Temperature – User specifies the specific depth data is desired at. Data available down to 200cm below the surface.

User specifies latitude/longitude

Lifestyle APIs –
Degree Days: Growing
Degree Days: (GGD), Heating
Degree Days: (HDD), Cooling
Degree Days: (CDD)
Frost Index & Watering Needs

HISTORY ON DEMAND

What is it?

This package offers precise and accurate historical weather information, featuring a 22.5 km worldwide grid with gap-filled datasets and hourly historical information dating back to July 2011. HoD provide most common historical parameters in a consistent grid in a “buy-what-you-need” model. The data can for example be used to understand how weather has impacted demand for a product or to correlate weather to performance metrics of your organisation.

What does it include?

Direct API use with location and time inputs

Historical data back to 2011

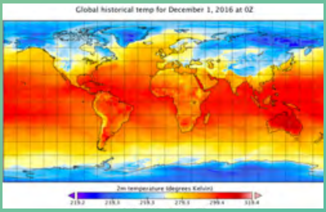
Support for multi-locations query and bounding box

22.5 km resolution

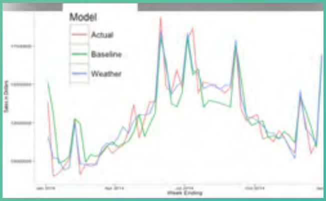
Hourly values for surface temperature, wind speed, wind direction, relative humidity, atmospheric pressure, and dewpoint rate.

Processes billions of observations and forecasts to build the industry’s most precise and accurate historical record of weather data.

Global snapshot of historical temps on a given date


Global historical temp for December 1, 2016 at 0Z
2m temperature (degrees Kelvin)

Data Science using historical weather data to train models and find busi-ness correlations



Weather API´s Documentation Structure

‘IBM Passport Advantage Offering’ – name of the packages that exist in the Weather Company’s full portfolio. We have access to:

- Weather Company Data – Core
- Weather Company Data – Agriculture
- Weather Company Data – History on Demand

‘Offering Summary Link’ – guide for how to use the APIs in this package.
‘Atomic API Name’ – the name of the unique API, sitting in a domain portfolio and domain. Each atomic API has several (atomic) endpoints.

The **API Common Usage Guide** contains information on API parameters that are consistent across end points (e.g. units of measure).

The **Atomic API Documentation** contains:

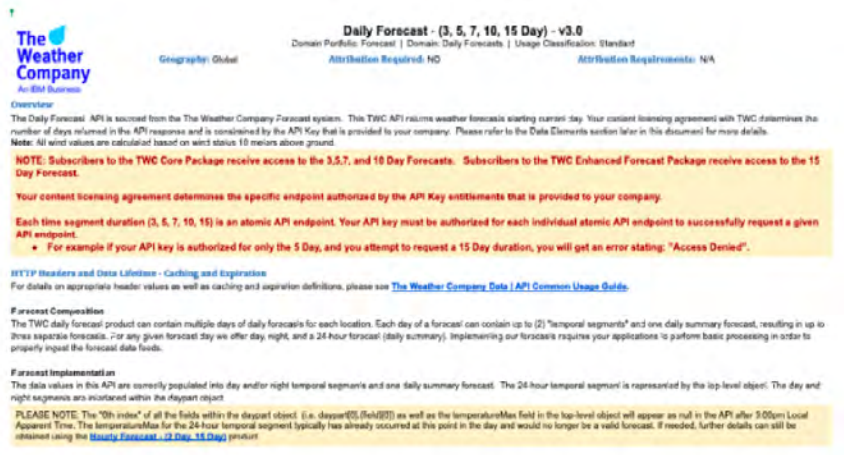
- List of available API endpoints
- URL construction examples, including required parameters
- Data elements and definitions
- Example response (JSON sample)

Weather API Documentation Links

API documentation for packages in this workshop:

- Core: *https://ibm.co/TWCdac*
- Agriculture: *https://ibm.co/TWCagr*
- History on Demand: *https://ibm.co/TWChod*

Passport Advantage:
*https://docs.google.com/
document/d/15Ru_3wdMgpbM4aOCm-4qNAnRfjx2w-Ruw3lnr8Hnodk/
edit?usp=sharing*



The Weather Company
An IBM Business

Daily Forecast - (3, 5, 7, 10, 15 Day) - v3.0

Domain Portfolio: Forecast | Domain: Daily Forecasts | Usage Classification: Standard

Geography: Global **Attribution Required:** NO **Attribution Requirements:** N/A

Overview

The Daily Forecast API is sourced from the The Weather Company Forecast system. This TWC API returns weather forecasts starting current day. Your content licensing agreement with TWC determines the number of days returned in the API response and is constrained by the API Key that is provided to your company. Please refer to the Data Elements section later in this document for more details.
Note: All wind values are calculated based on wind status 10 meters above ground.

NOTE: Subscribers to the TWC Core Package receive access to the 3,5,7, and 10 Day Forecasts. Subscribers to the TWC Enhanced Forecast Package receive access to the 15 Day Forecast.

Your content licensing agreement determines the specific endpoint authorized by the API Key entitlements that is provided to your company.

Each time segment duration (3, 5, 7, 10, 15) is an atomic API endpoint. Your API key must be authorized for each individual atomic API endpoint to successfully request a given API endpoint.

- **For example if your API key is authorized for only the 5 Day, and you attempt to request a 15 Day duration, you will get an error stating: "Access Denied".**

HTTP Headers and Data Lifetime - Caching and Expiration

For details on appropriate header values as well as caching and expiration definitions, please see **The Weather Company Data | API Common Usage Guide.**

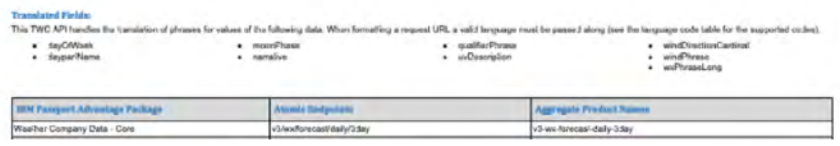
Forecast Composition

The TWC daily forecast product can contain multiple days of daily forecasts for each location. Each day of a forecast can contain up to (2) "temporal segments" and one daily summary forecast, resulting in up to three separate forecasts. For any given forecast day we offer day, night, and a 24-hour forecast (daily summary). Implementing our forecasts requires your applications to perform basic processing in order to properly ingest the forecast data feeds.

Forecast Implementation

The data values in this API are correctly populated into day and/or night temporal segments and one daily summary forecast. The 24-hour temporal segment is represented by the top-level object. The day and night segments are instanced within the daypart object.

PLEASE NOTE: The "0th index" of all the fields within the daypart object (i.e. daypart[0].{field}[0]) as well as the temperatureMax field in the top-level object will appear as null in the API after 3:00pm Local Apparent Time. The temperatureMax for the 24-hour temporal segment typically has already occurred at this point in the day and would no longer be a valid forecast. If needed, further details can still be obtained using the **Hourly Forecast - (2 Day, 15 Day)** product.



Translated Fields:

This TWC API handles the translation of phrases for values of the following data. When formatting a request URL a valid language must be passed along (see the language code table for the supported codes).

- dayOfWeek
- daypartName
- moonPhrase
- narrative
- qualifierPhrase
- uvDescription
- windDirectionCardinal
- windPhrase
- wxPhraseLong

IBM Passport Advantage Package	Atomic Endpoints	Aggregate Product Names
Weather Company Data - Core	v3/wxforecast/daily/3day	v3-wx-forecast-daily-3day

Weather API Integration Case Studies

Customised agriculture forecasts: *https://www.ibm.com/case-studies/emnotion-cloud*
Route management: *https://www.ibm.com/case-studies/xship*
Disaster management: *https://www.ibm.com/case-studies/satsure*

Publisher
Deutsche Gesellschaft für
Internationale Zusammenarbeit (GIZ) GmbH

Tech Entrepreneurship Initiative
“Make-IT in Africa”

Registered offices
Bonn and Eschborn

Friedrich-Ebert-Allee 32 + 36
53113 Bonn / Germany
T +49 228 44 60-0
F +49 228 44 60-17 66

Dag-Hammarskjöld-Weg 1-5
65760 Eschborn / Germany
T +49 61 96 79-0
F +49 61 96 79-11 15
E info@giz.de
I www.giz.de

Authors
Marc Hümmer
Cecilia Ivsen
Desiree Winges

Editor
creative republic/David Steel

Design
creative republic, Frankfurt a. M. / Germany

Illustrations
© shutterstock & creative republic

Photos Page 36
© shutterstock

GIZ is responsible for the content of this publication.

On behalf of the
German Federal Ministry for Economic Cooperation and Development (BMZ)

Division 112 - Digital technologies in development cooperation

Postal address of BMZ headquarters
Bonn and Berlin

Dahlmannstr. 4
53113 Bonn / Germany
T +49 228 99 535-0
F +49 228 99 535-3500

Stresemannstraße 94
10963 Berlin
T +49 30 18 535-0
F +49 30 18 535-2501

E poststelle@bmz.bund.de
I www.bmz.de

As of
December 2020