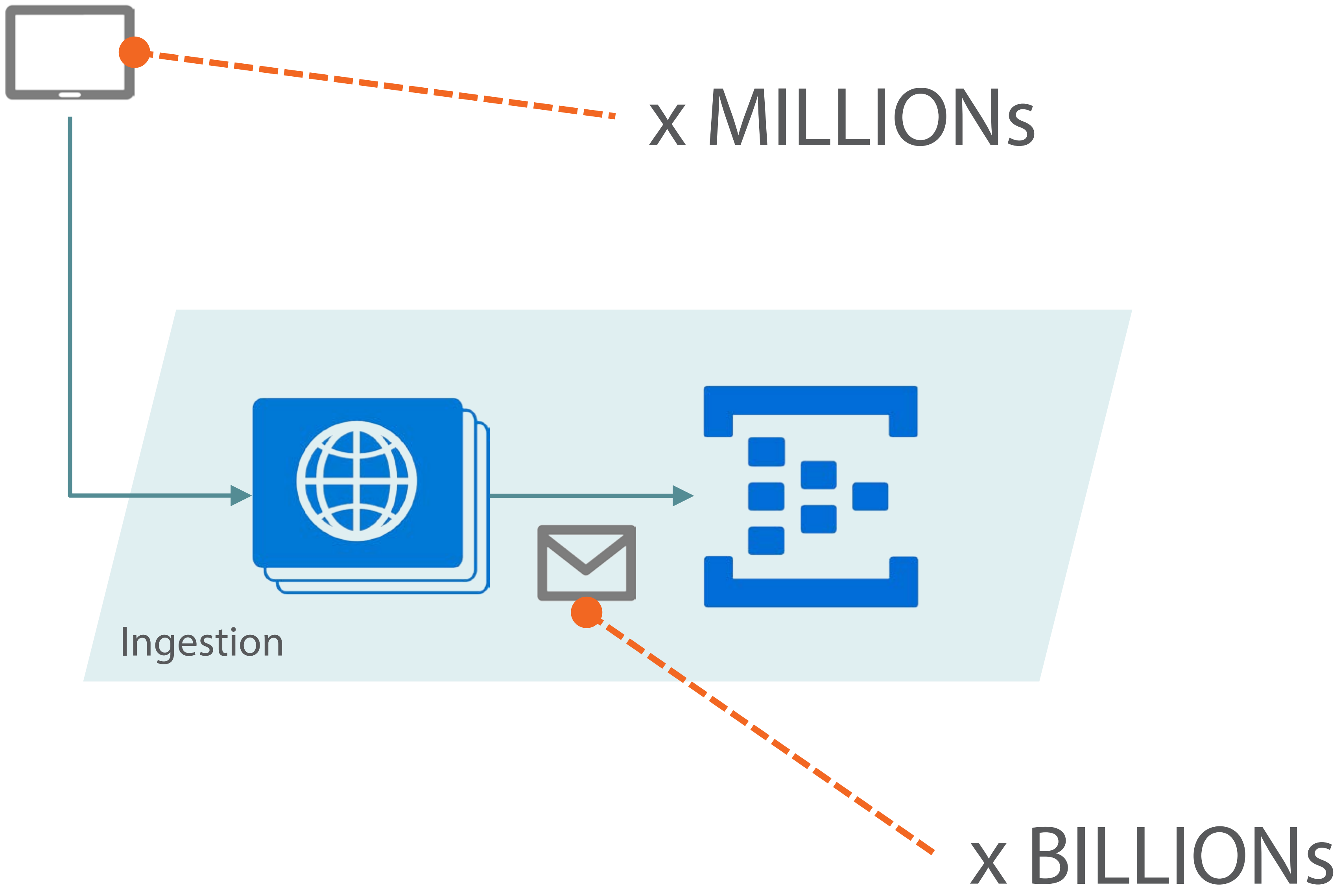
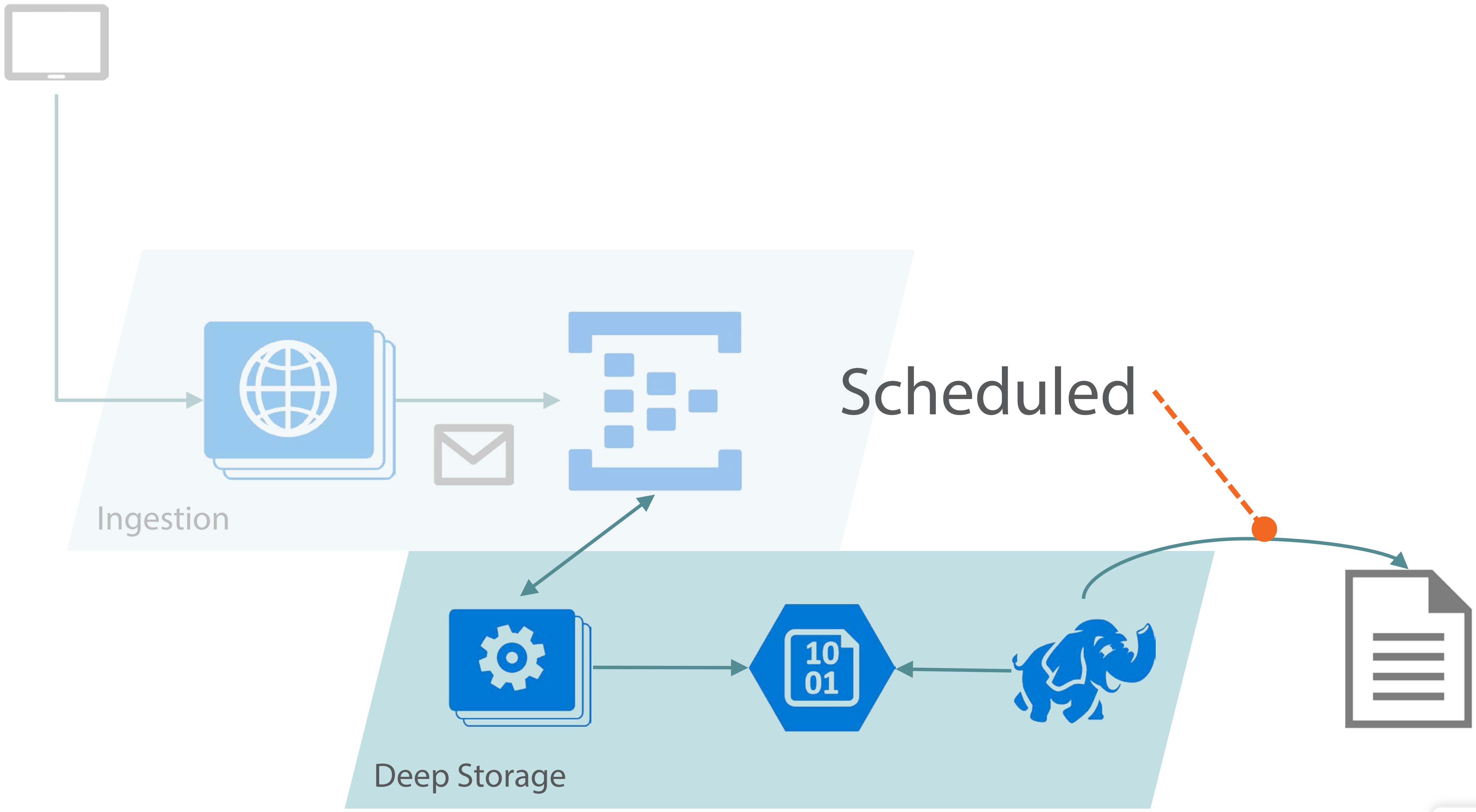
Real World Big Data in Azure

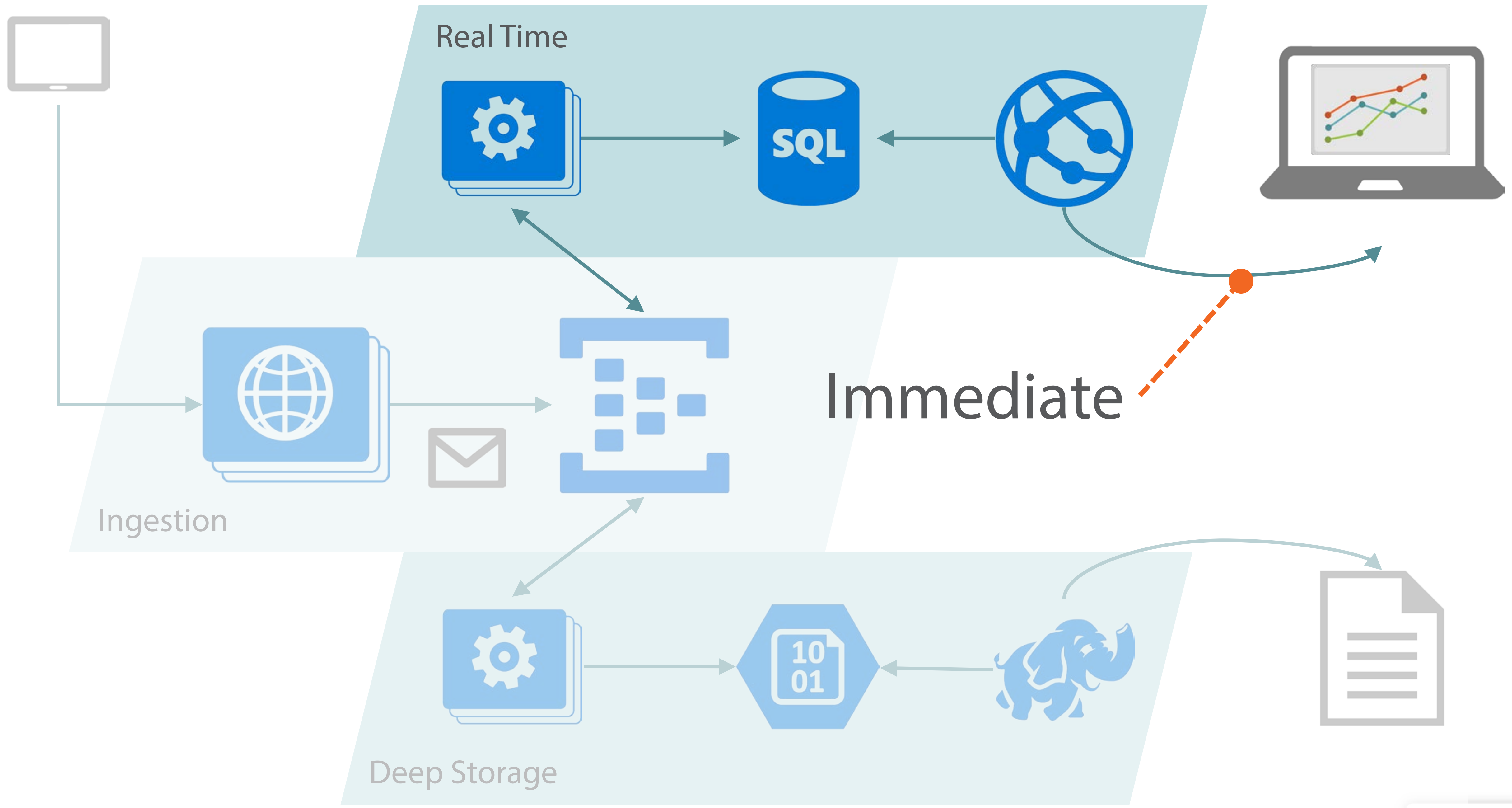
Understanding Big Data in Azure

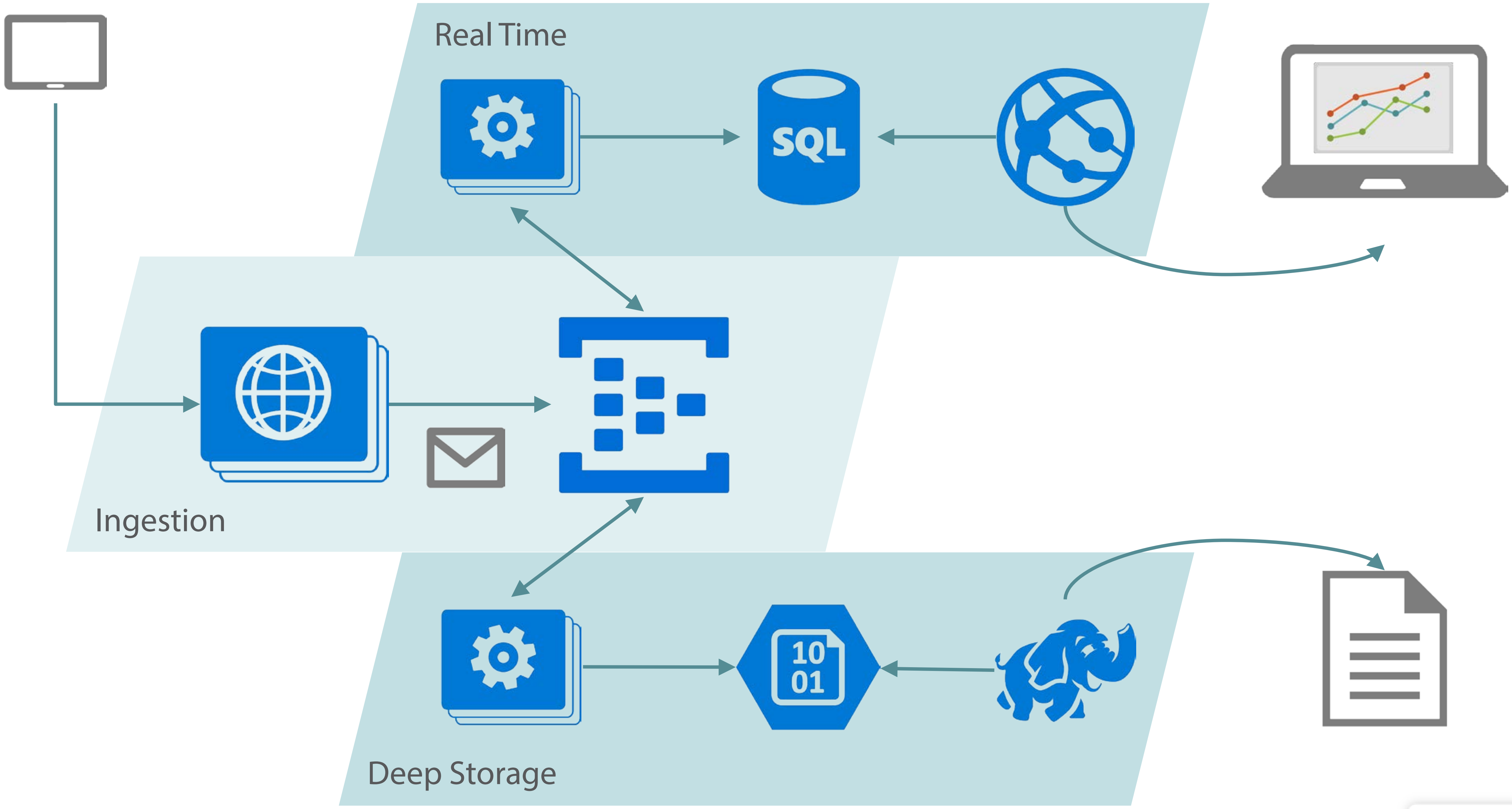
Elton Stoneman

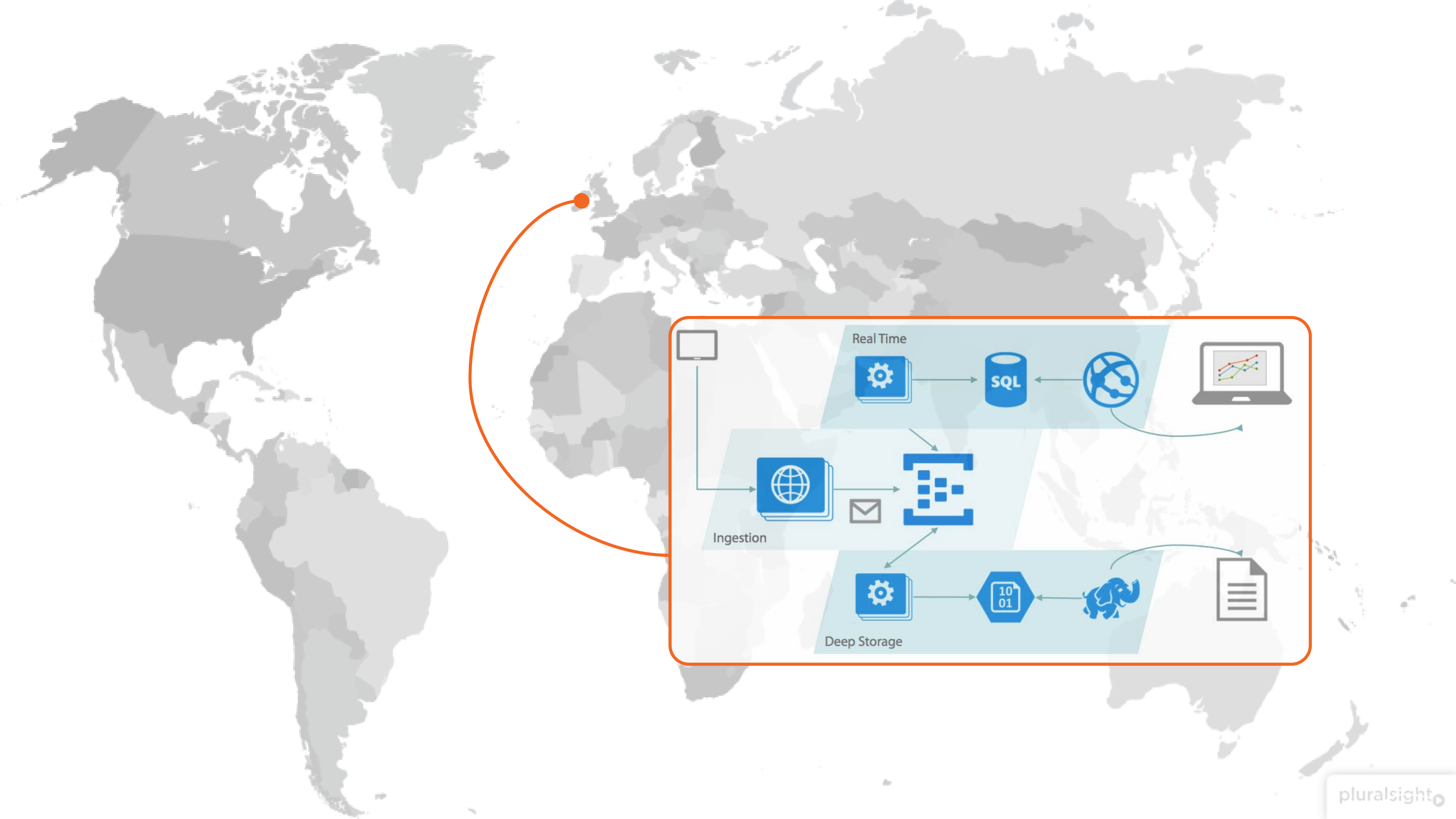
@EltonStoneman | www.geekswithblogs.net/eltonstoneman

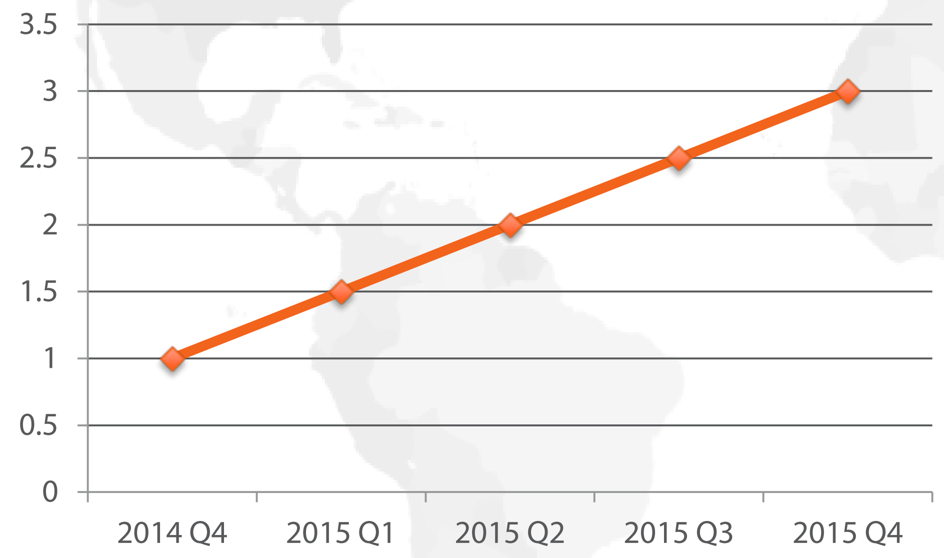

x MILLIONs
Ingestion
x BILLIONs


Ingestion
Scheduled
10
01
Deep Storage

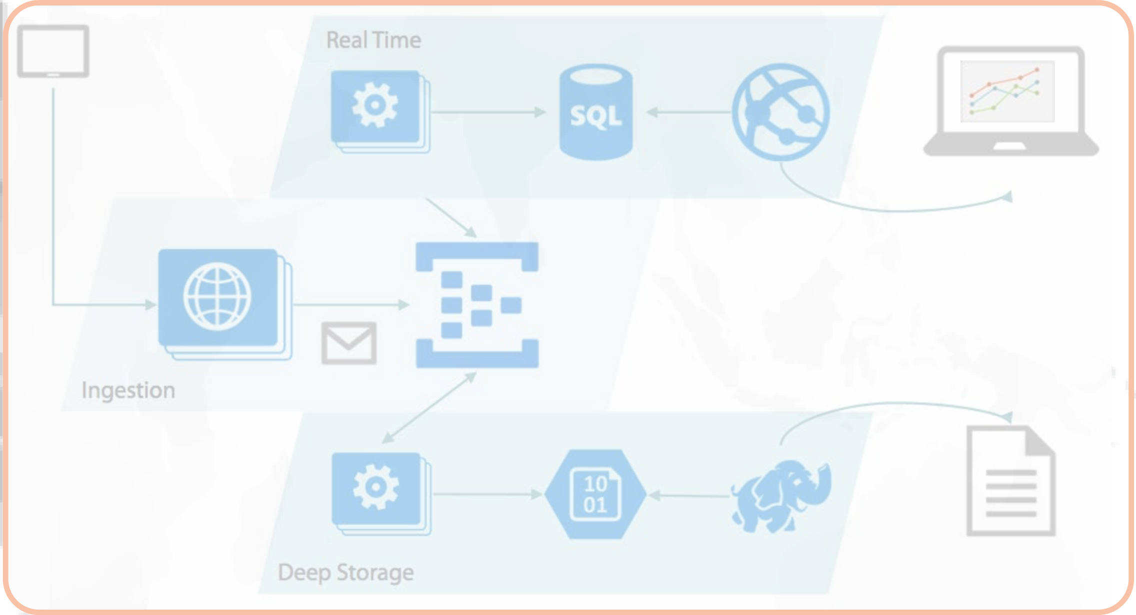

Real Time
SQL
Ingestion
10
01
Deep Storage
Immediate

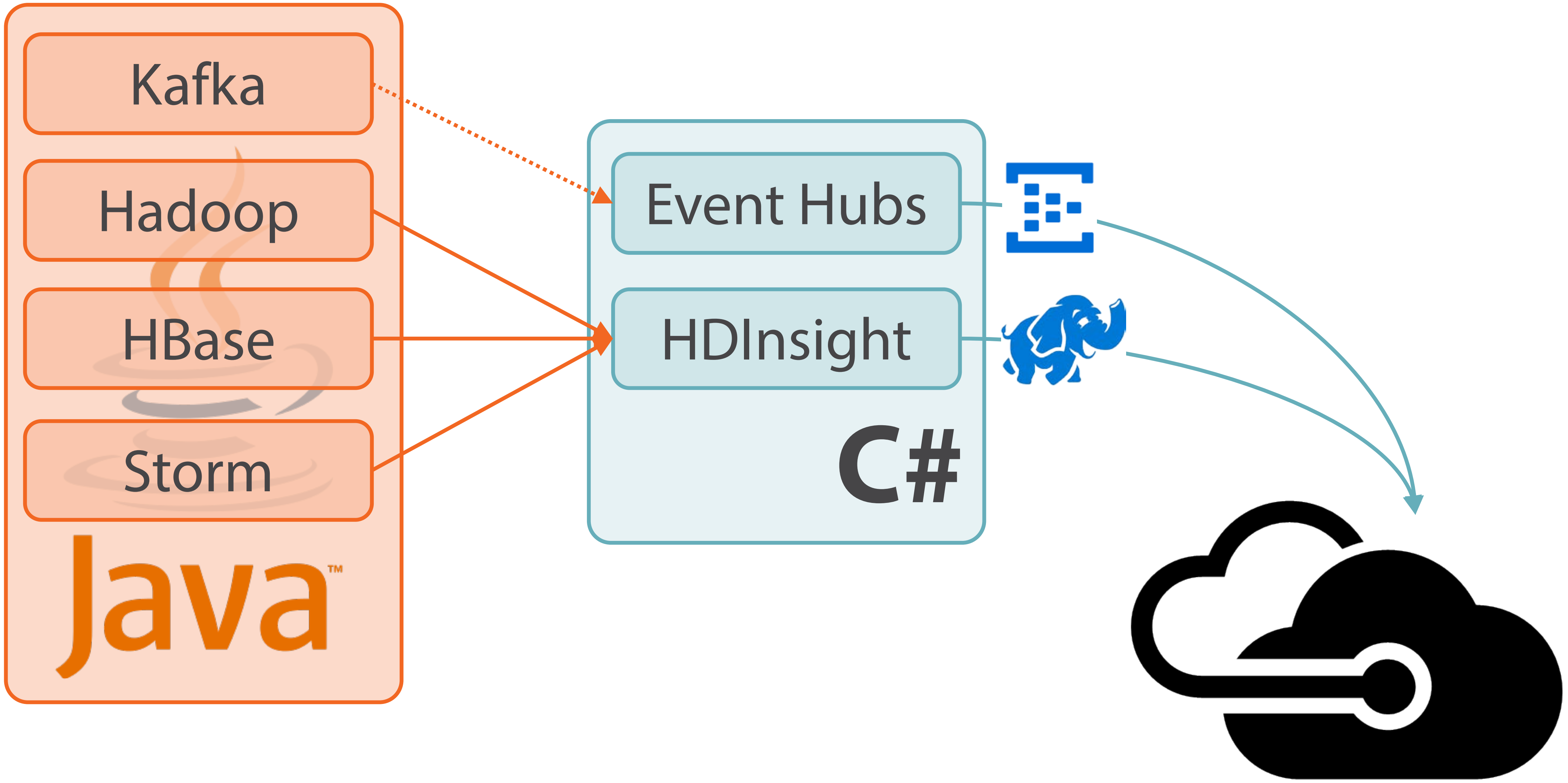

Real Time
SQL
Ingestion
10
01
Deep Storage

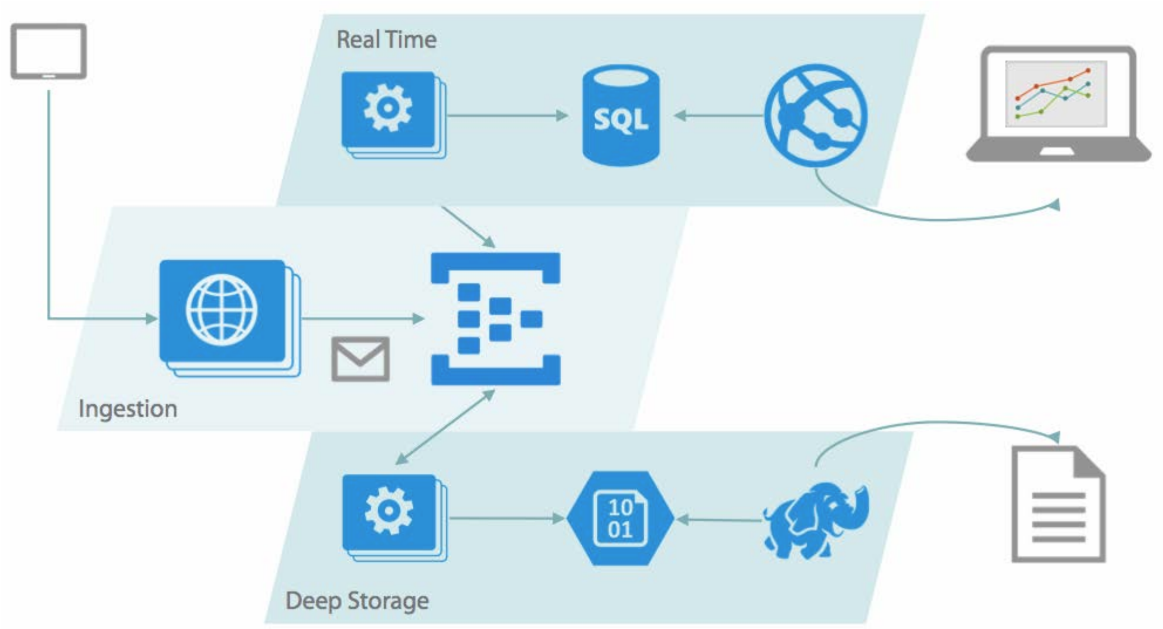

Real Time
SQL
Ingestion
10
01
Deep Storage


3.5
3
2.5
2
1.5
1
0.5
0
2014 Q4
2015 Q1
2015 Q2
2015 Q3
2015 Q4

Bn /day


Real Time
SQL
Ingestion
10
01
Deep Storage


Kafka
Hadoop
HBase
Storm
Java
Event Hubs
HDInsight
C#

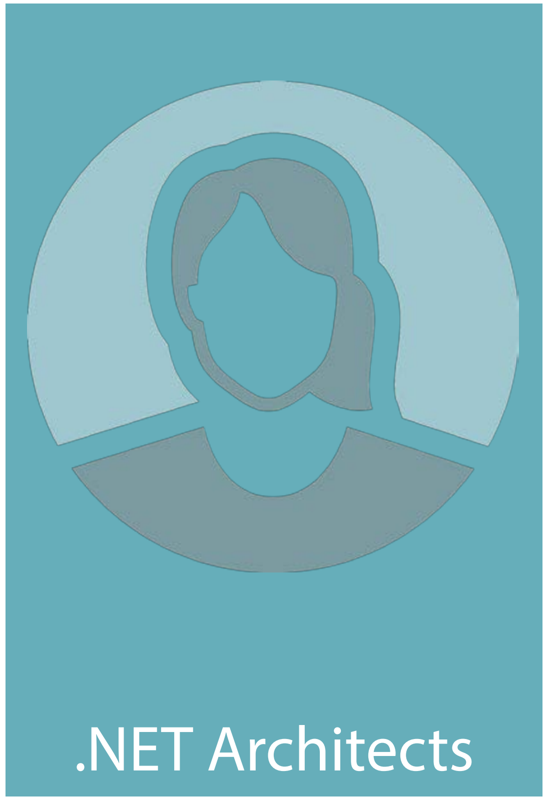


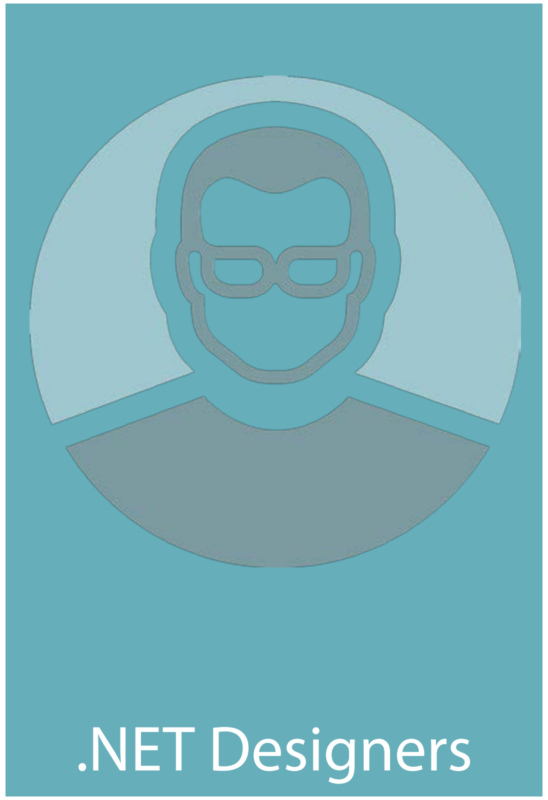
Scalable
Robust

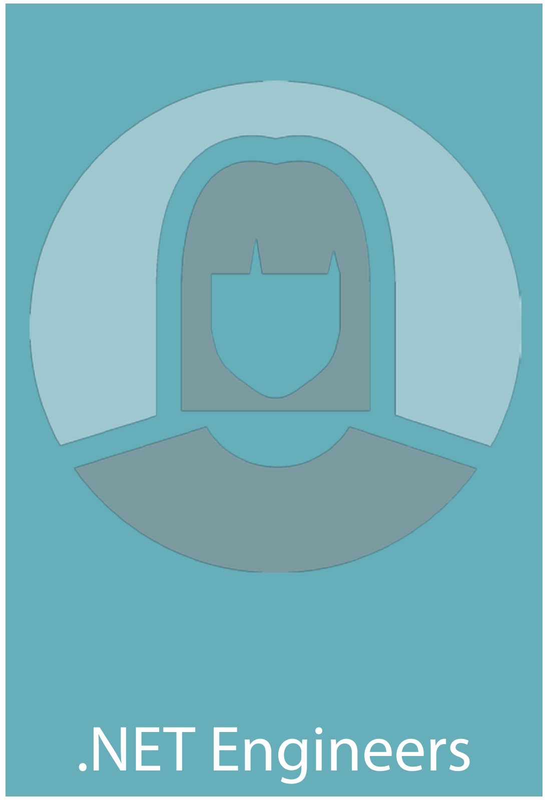
Fast
Efficient

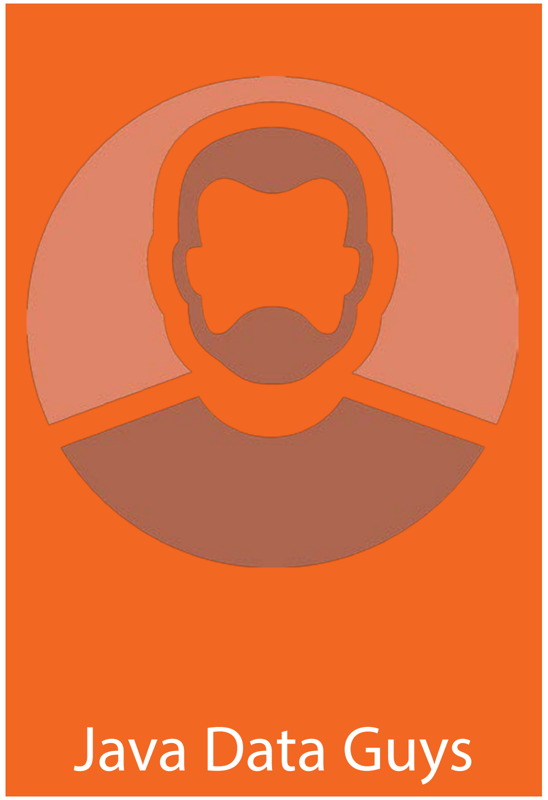
Supportable
Valuable

Is This Course for Me?





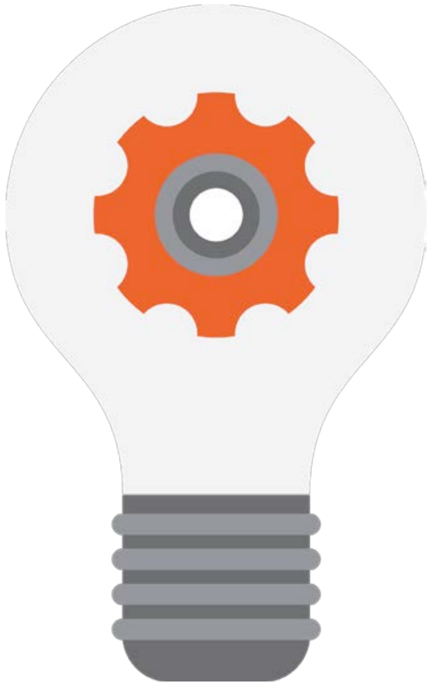




The Knowledge

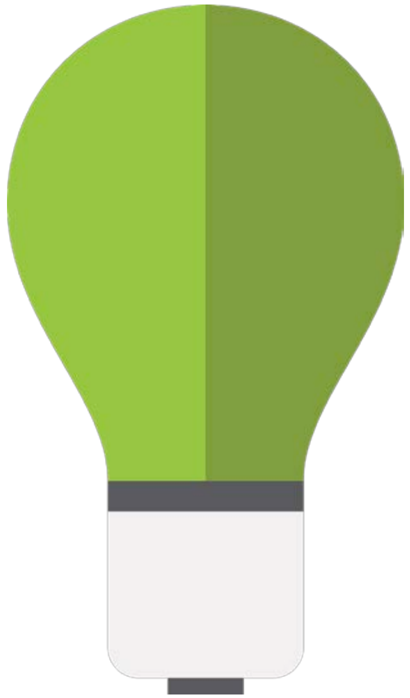
You should already know

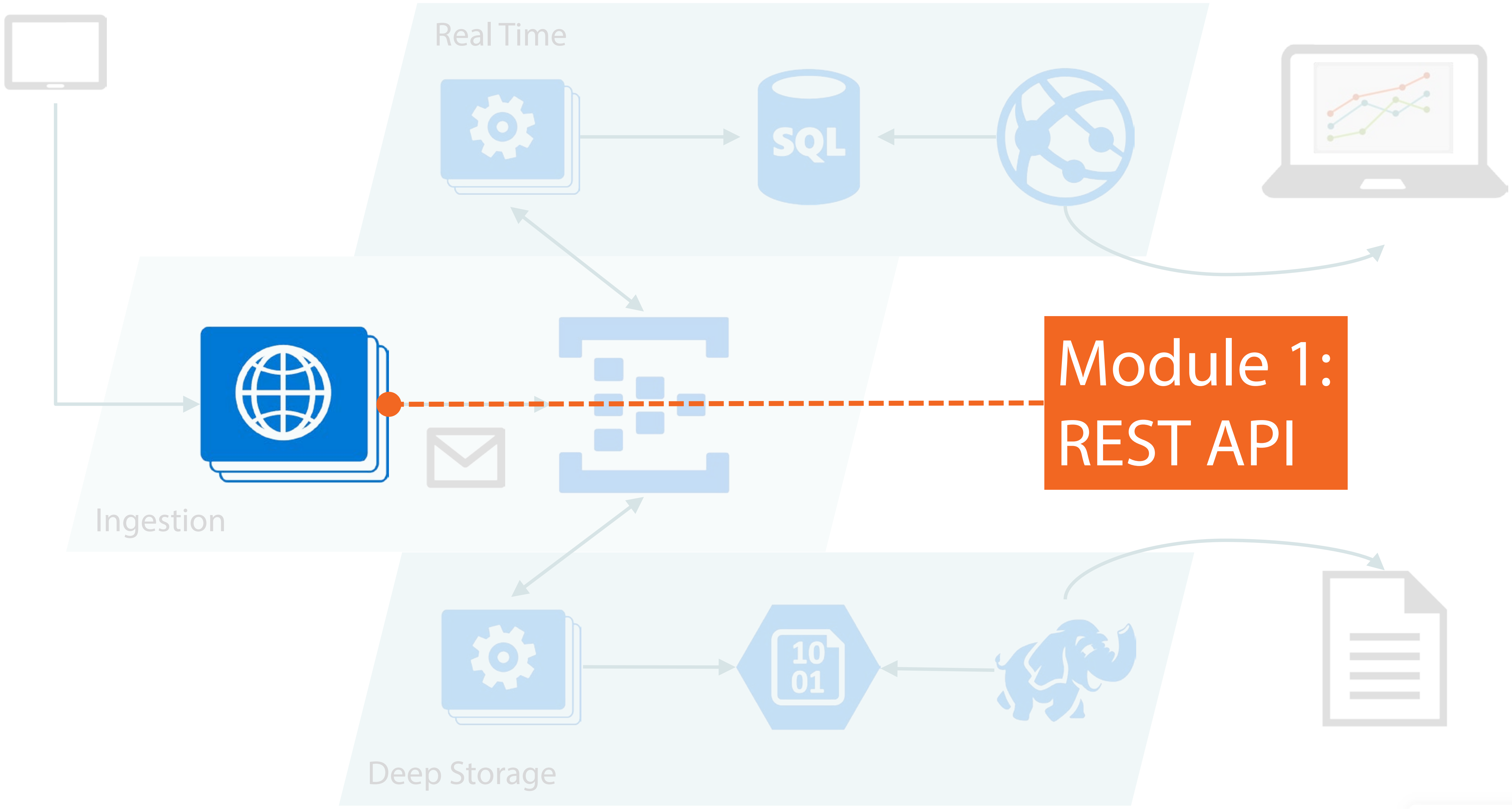
- C#
- SQL
- HTML
- Visual Studio
- (Some) Powershell



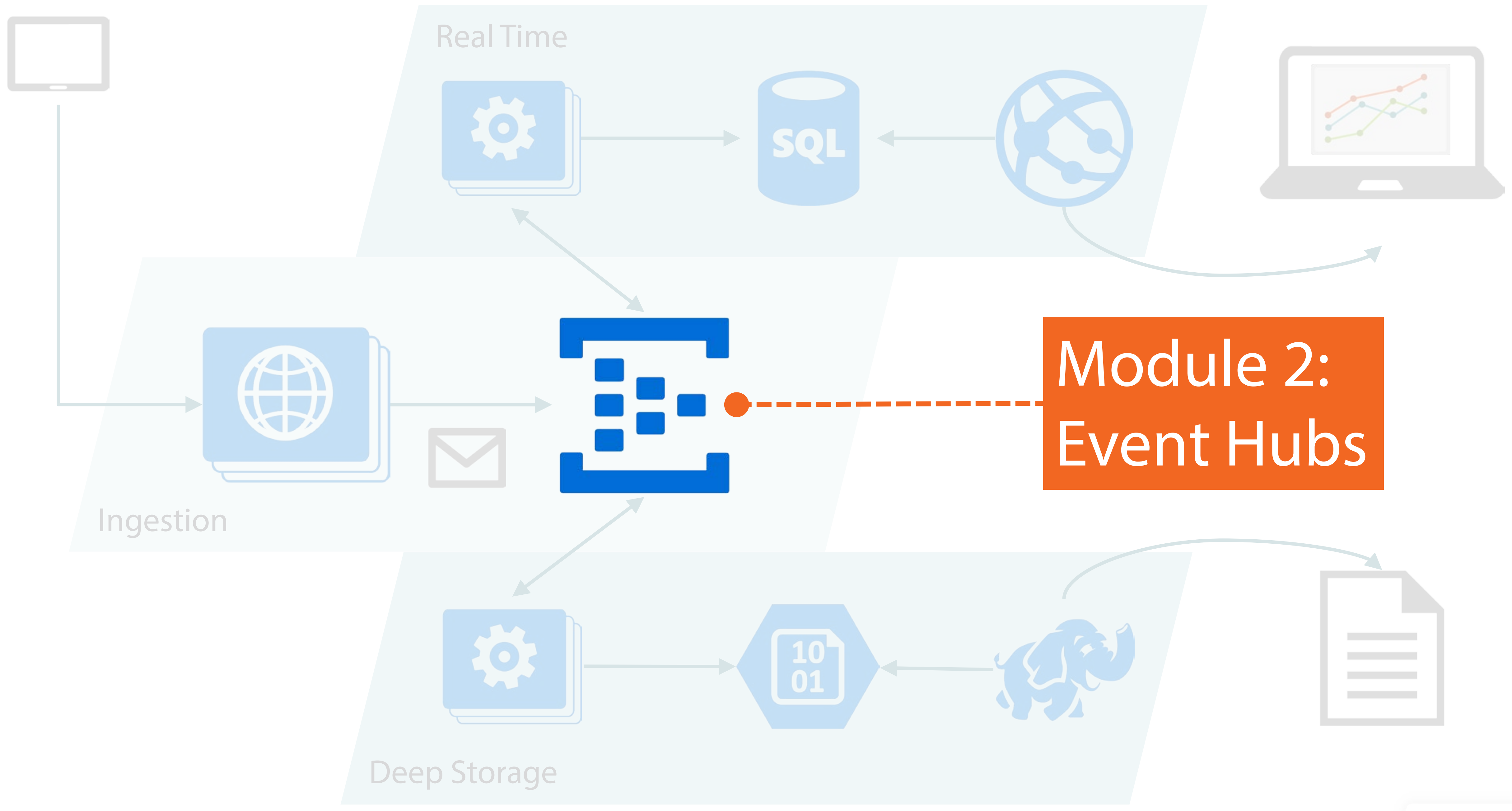
And you will learn

- Event Hubs
- Cloud Services
- Azure Websites
- SQL Azure
- Hadoop & Pig
- HBase & Storm

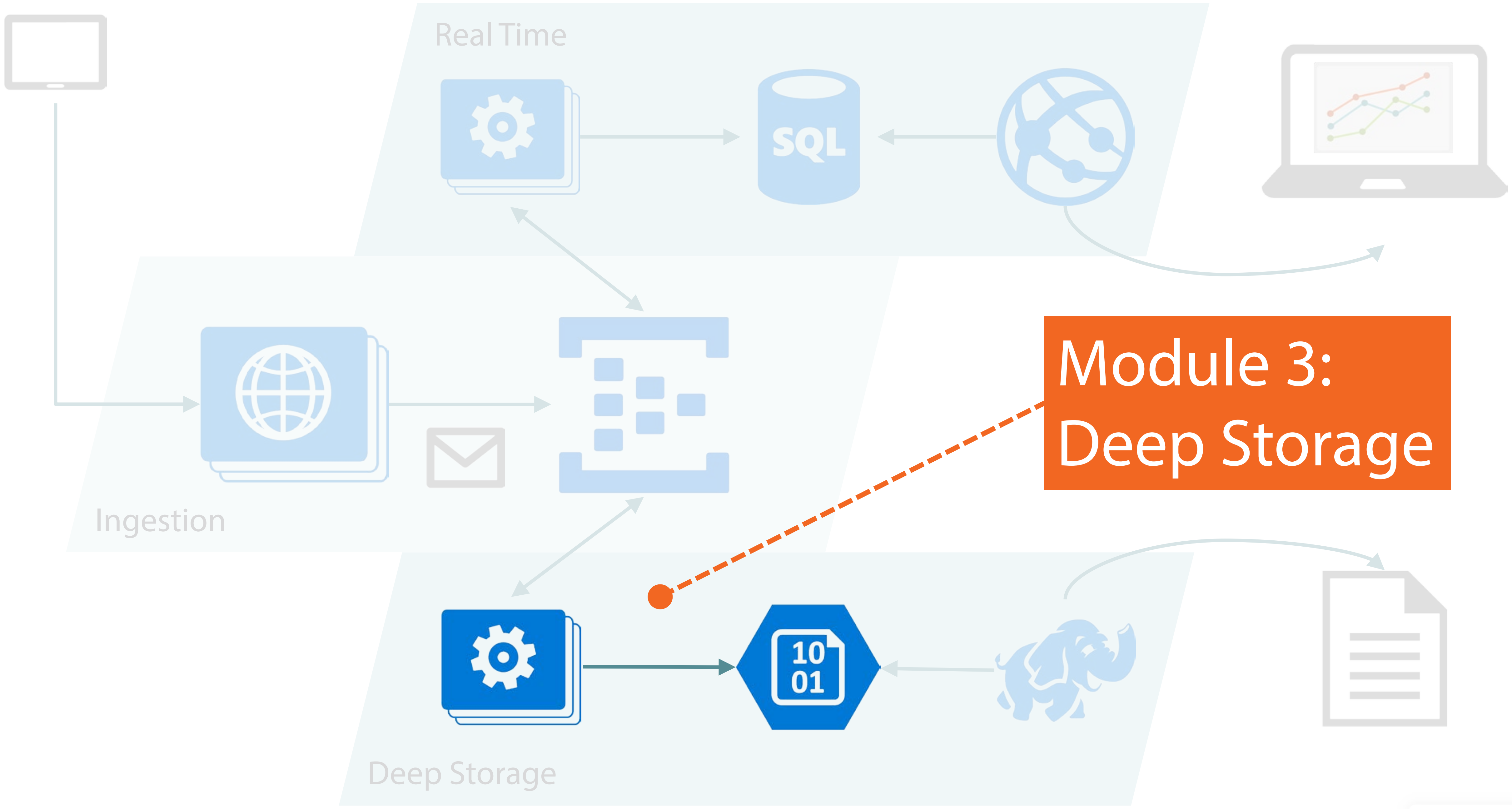




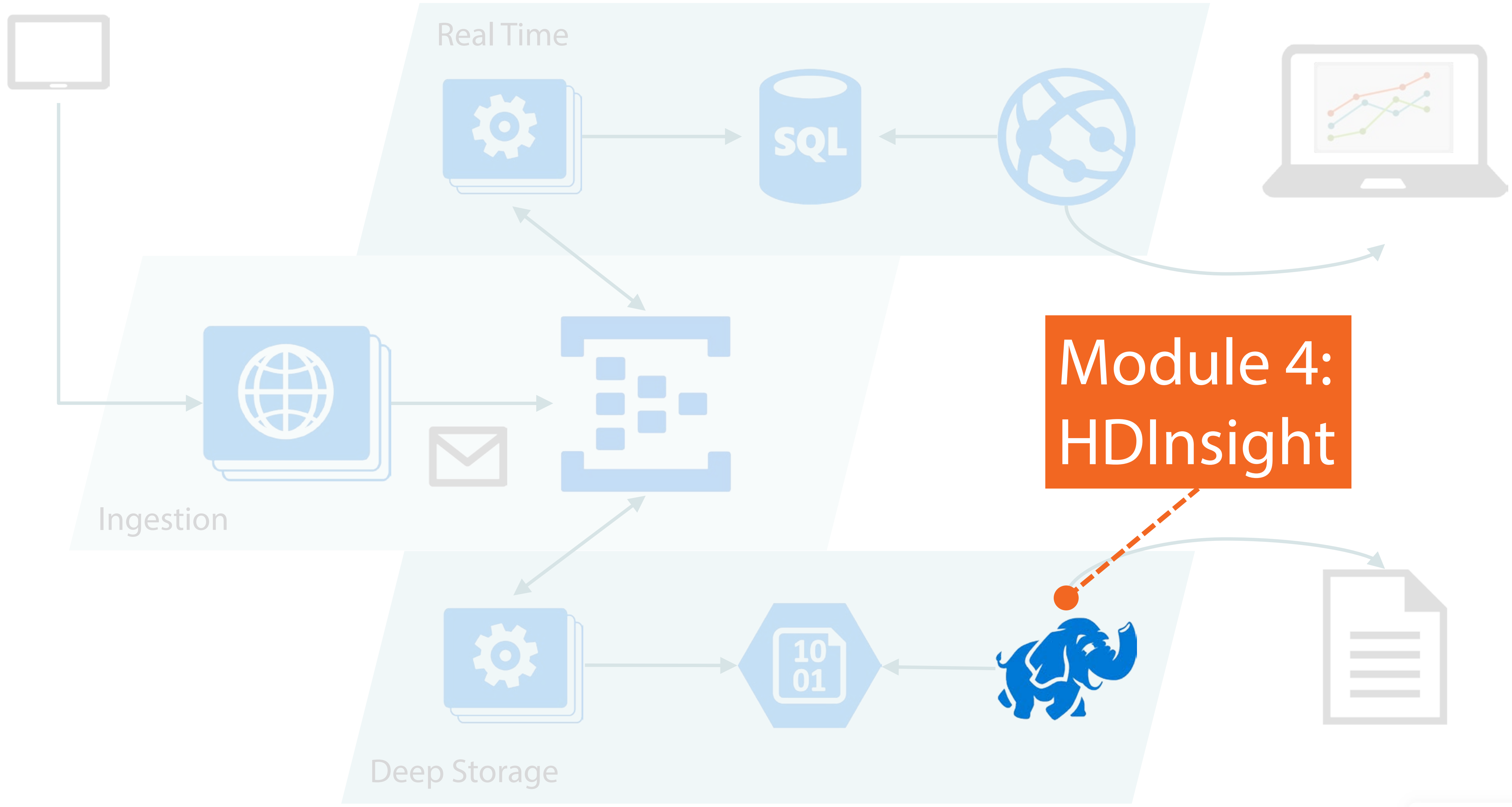
Module 1:
REST API



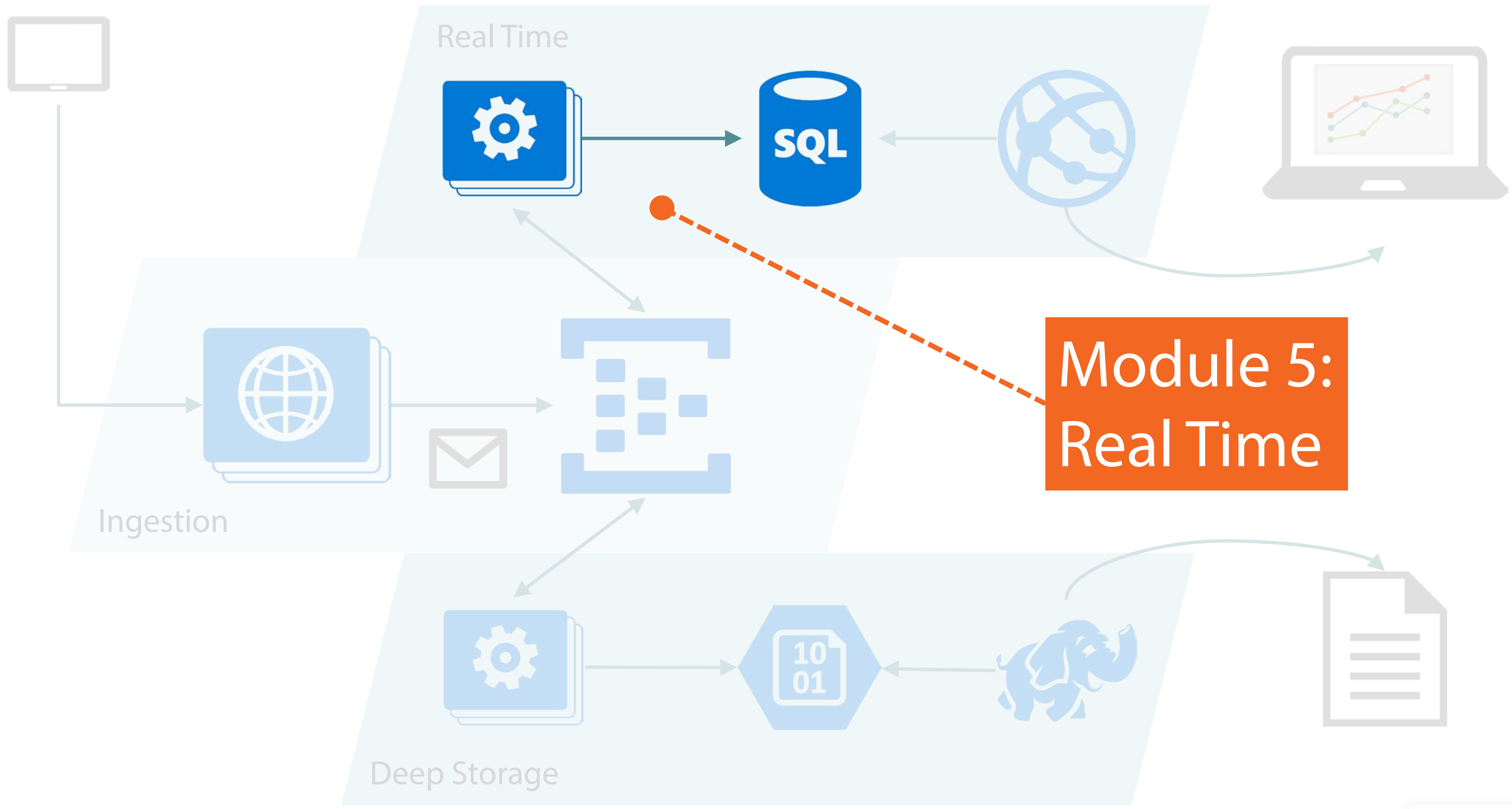
Module 2: Event Hubs



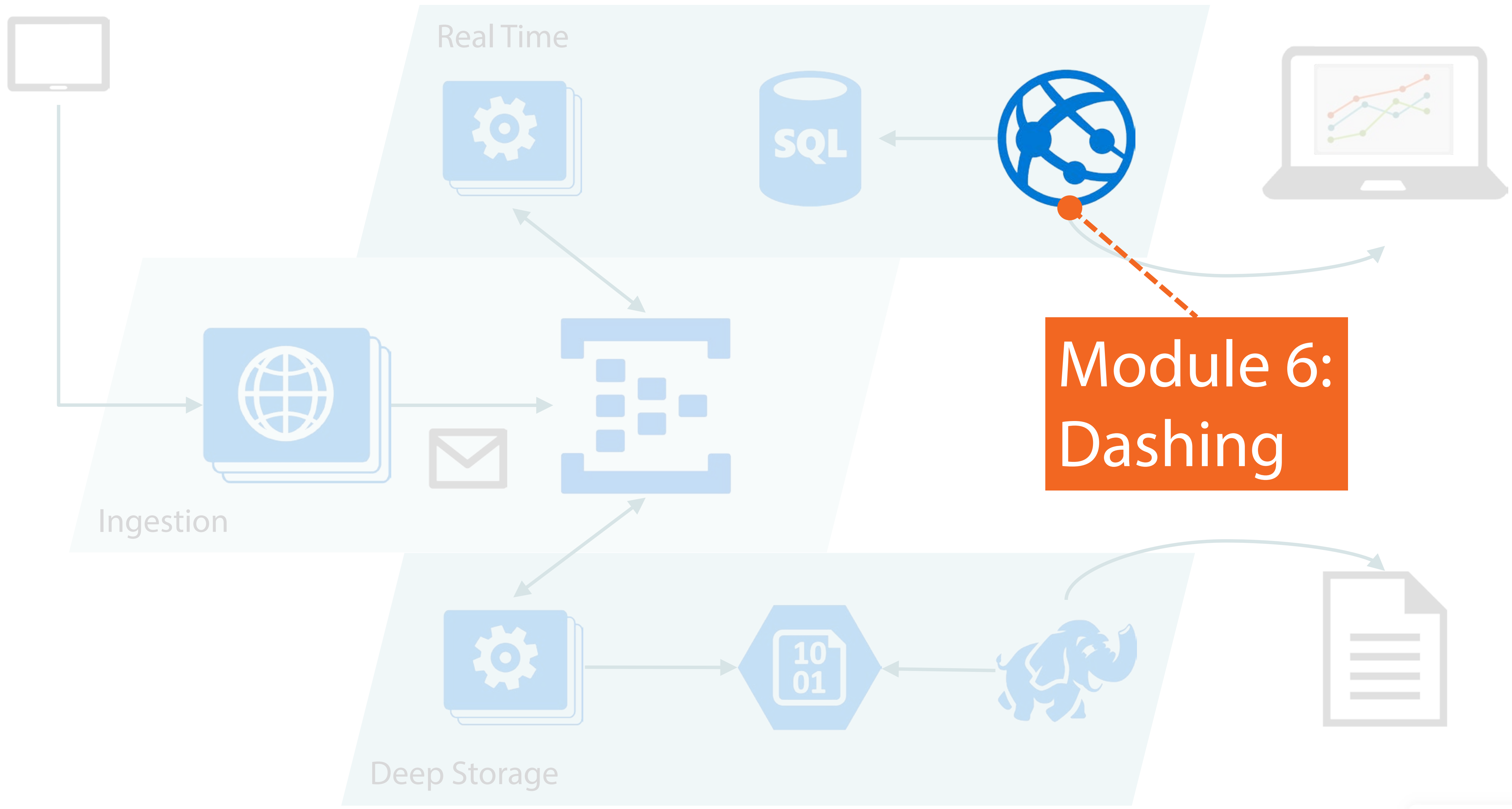
Module 3: Deep Storage



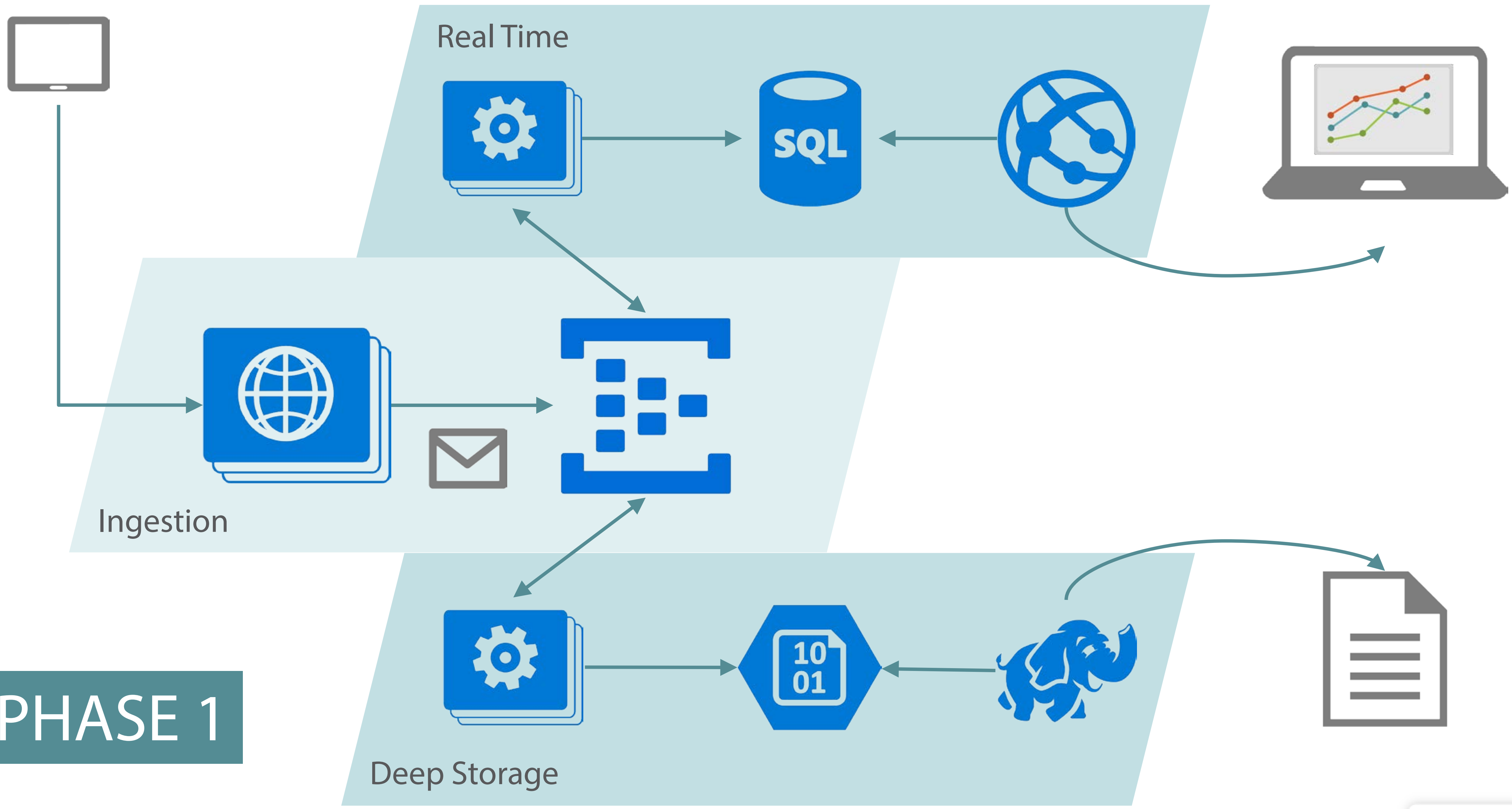
Module 4: HDInsight



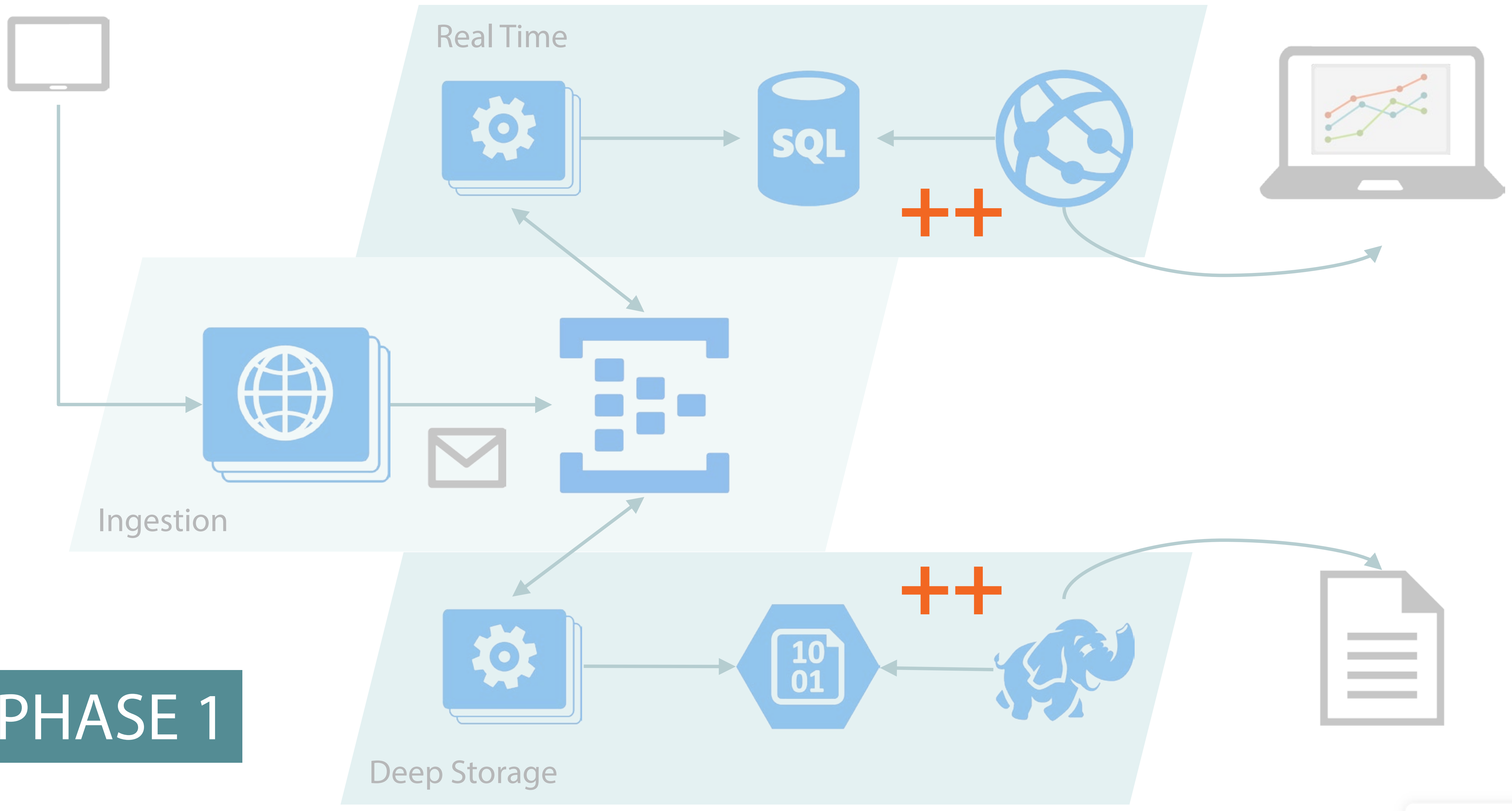
Module 5: Real Time



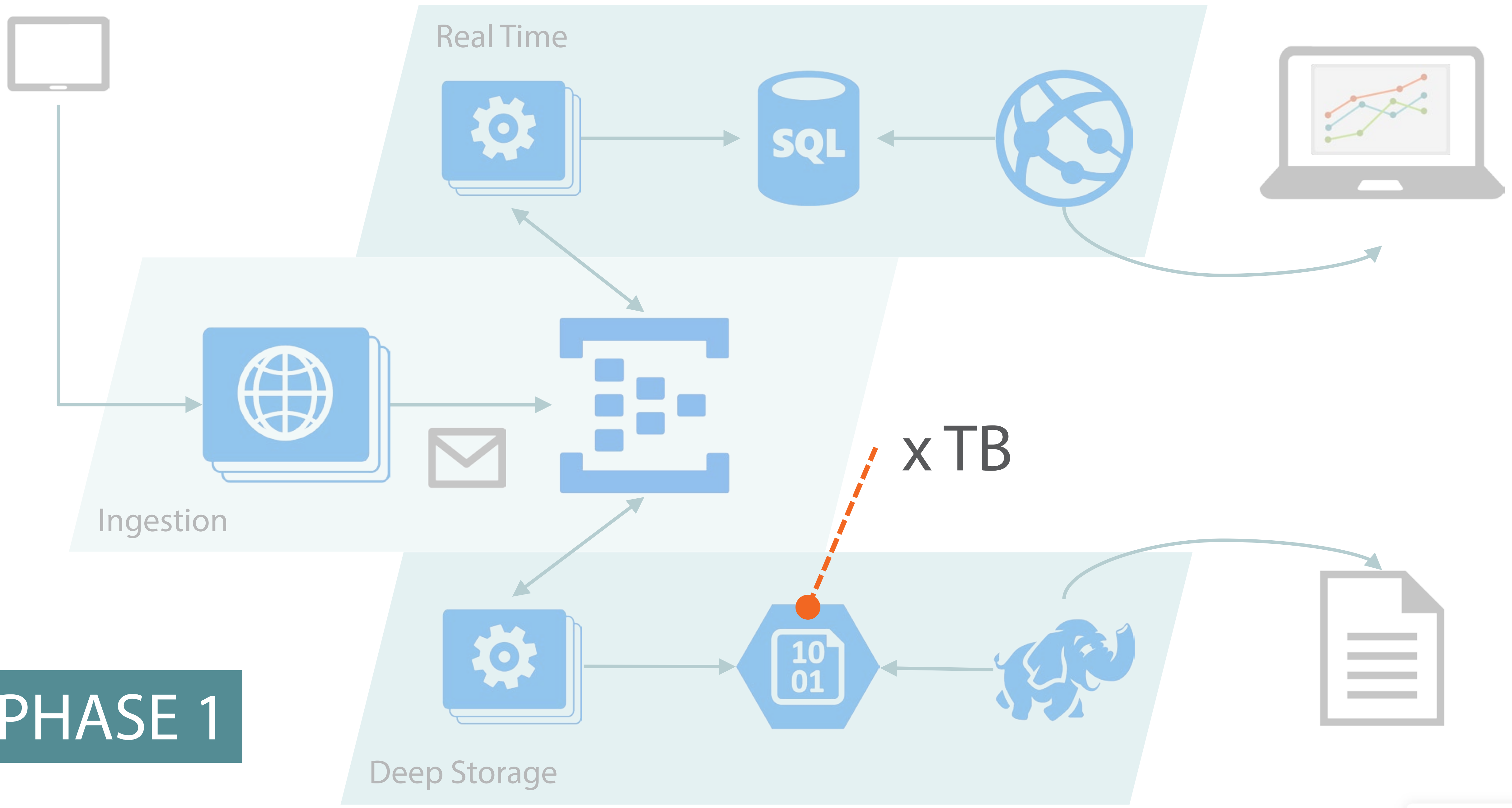
Module 6: Dashing



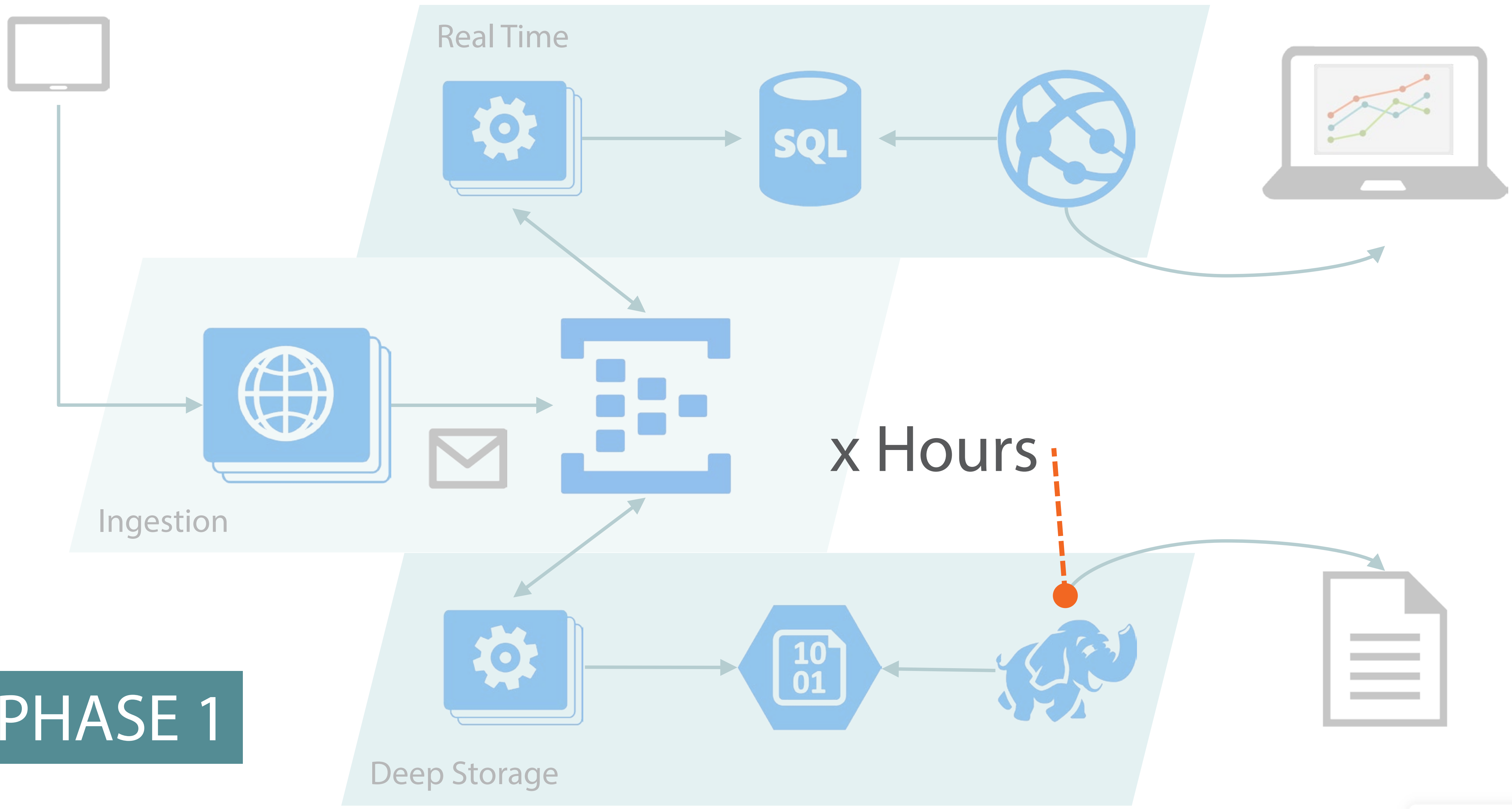
PHASE 1



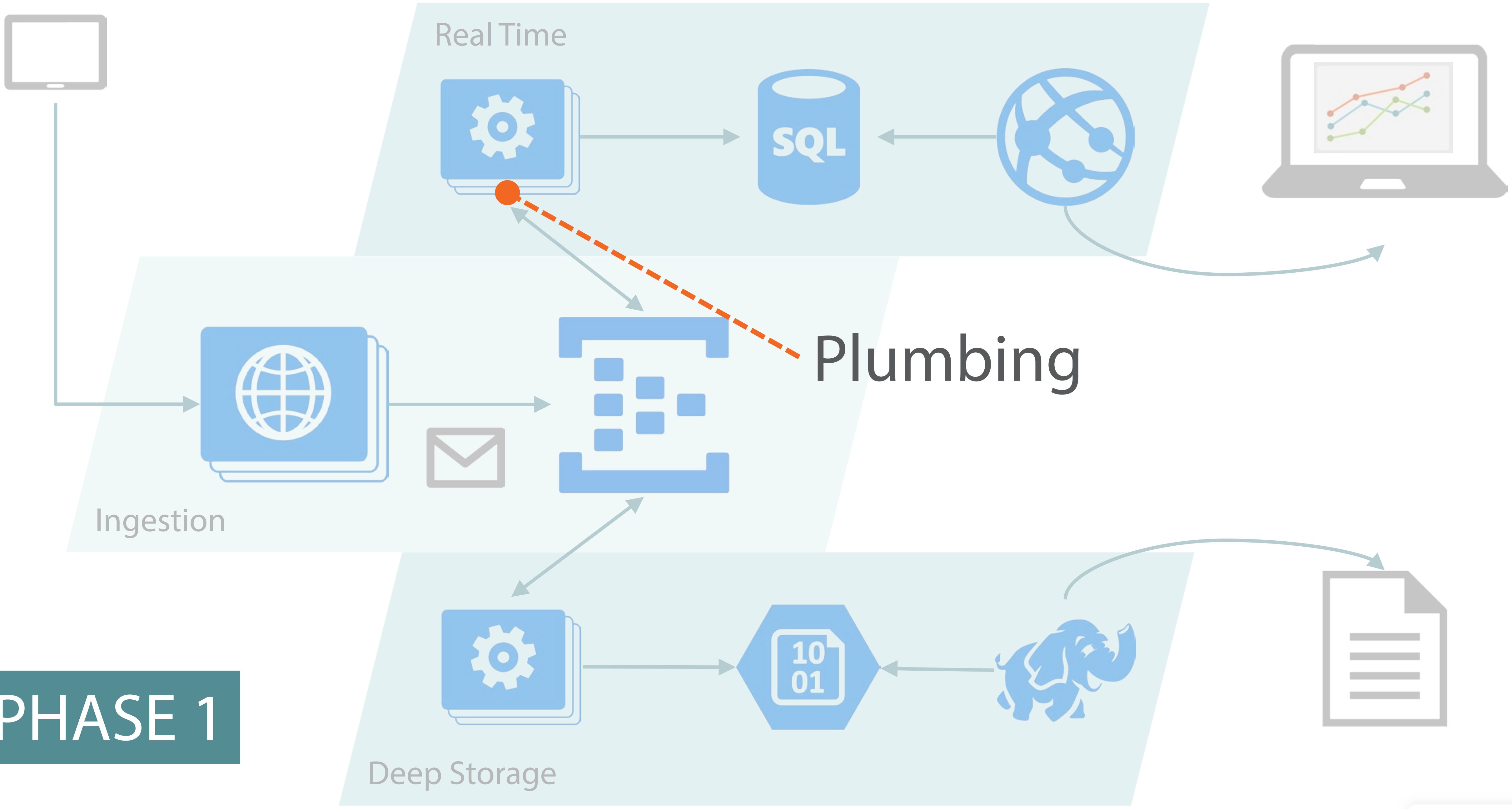
PHASE 1



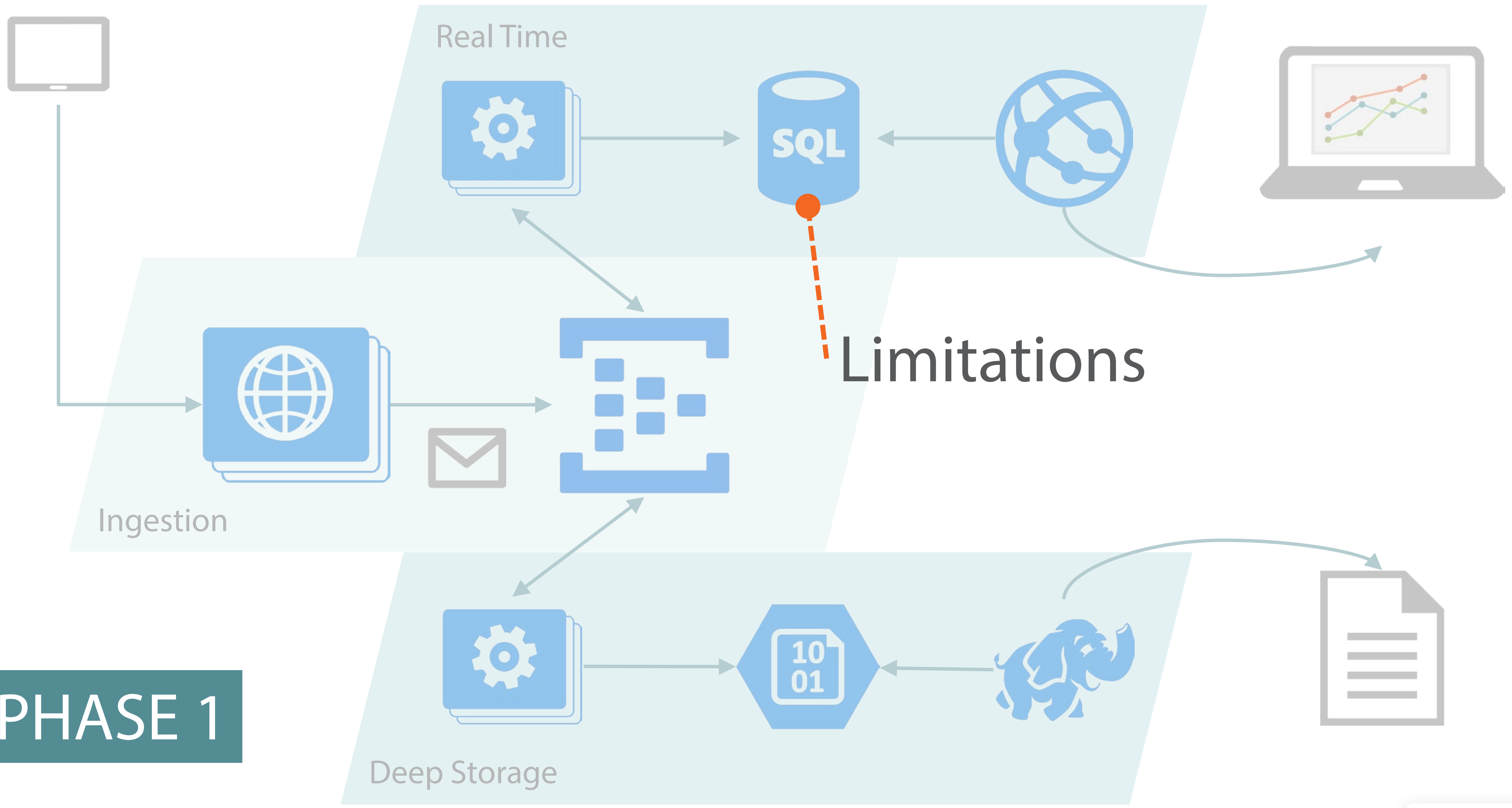
PHASE 1



PHASE 1

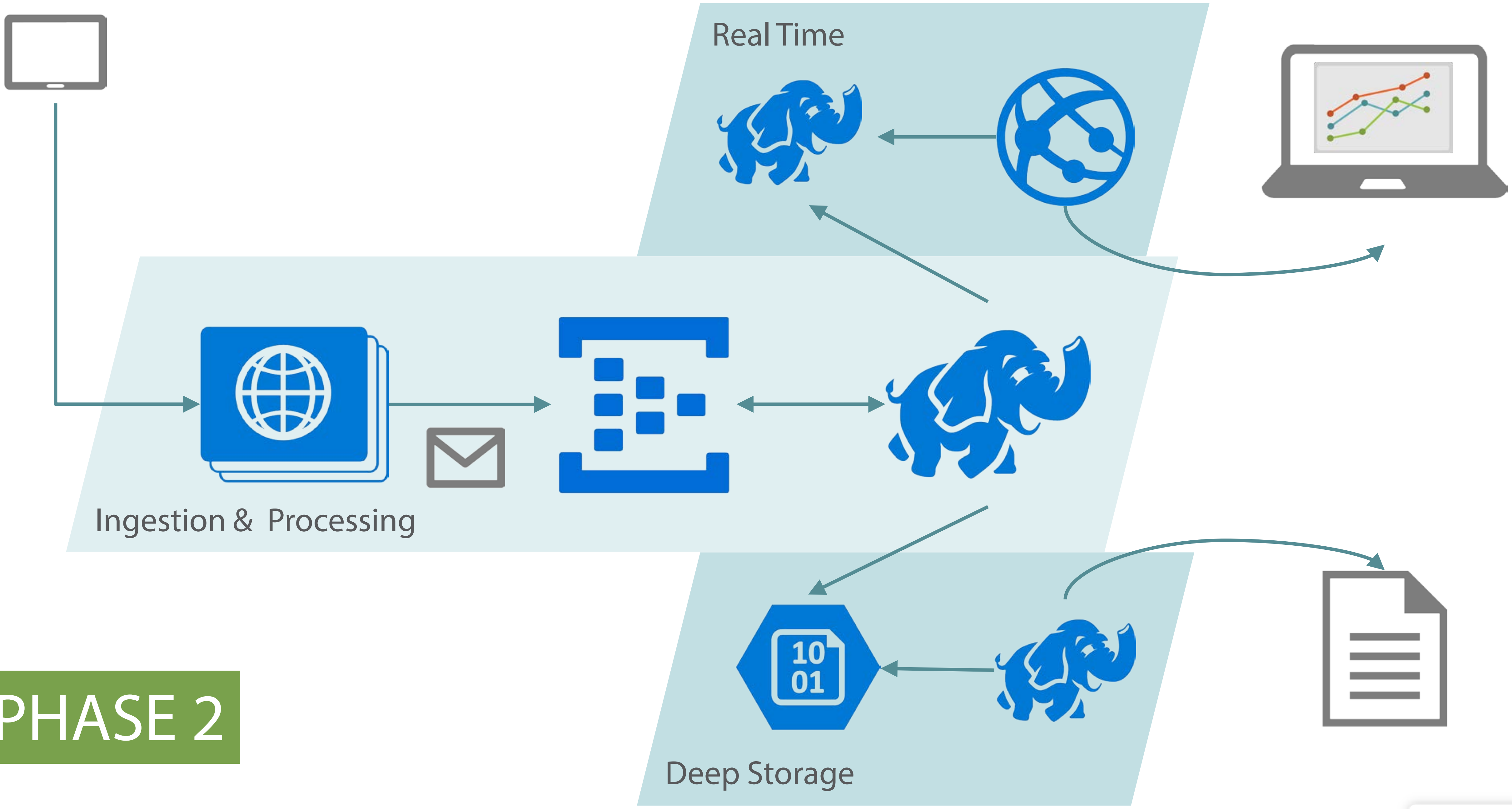


PHASE 1

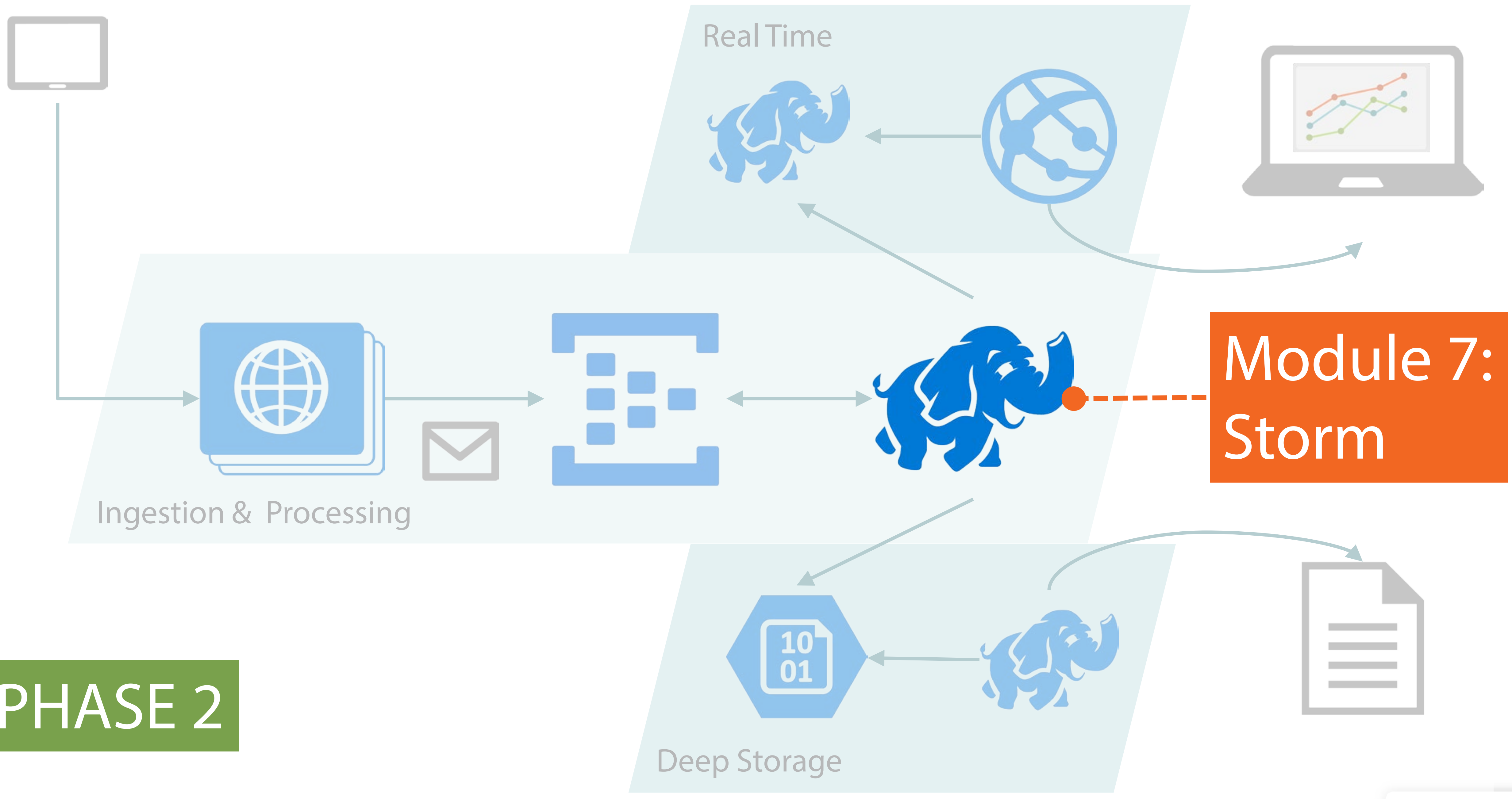


Limitations

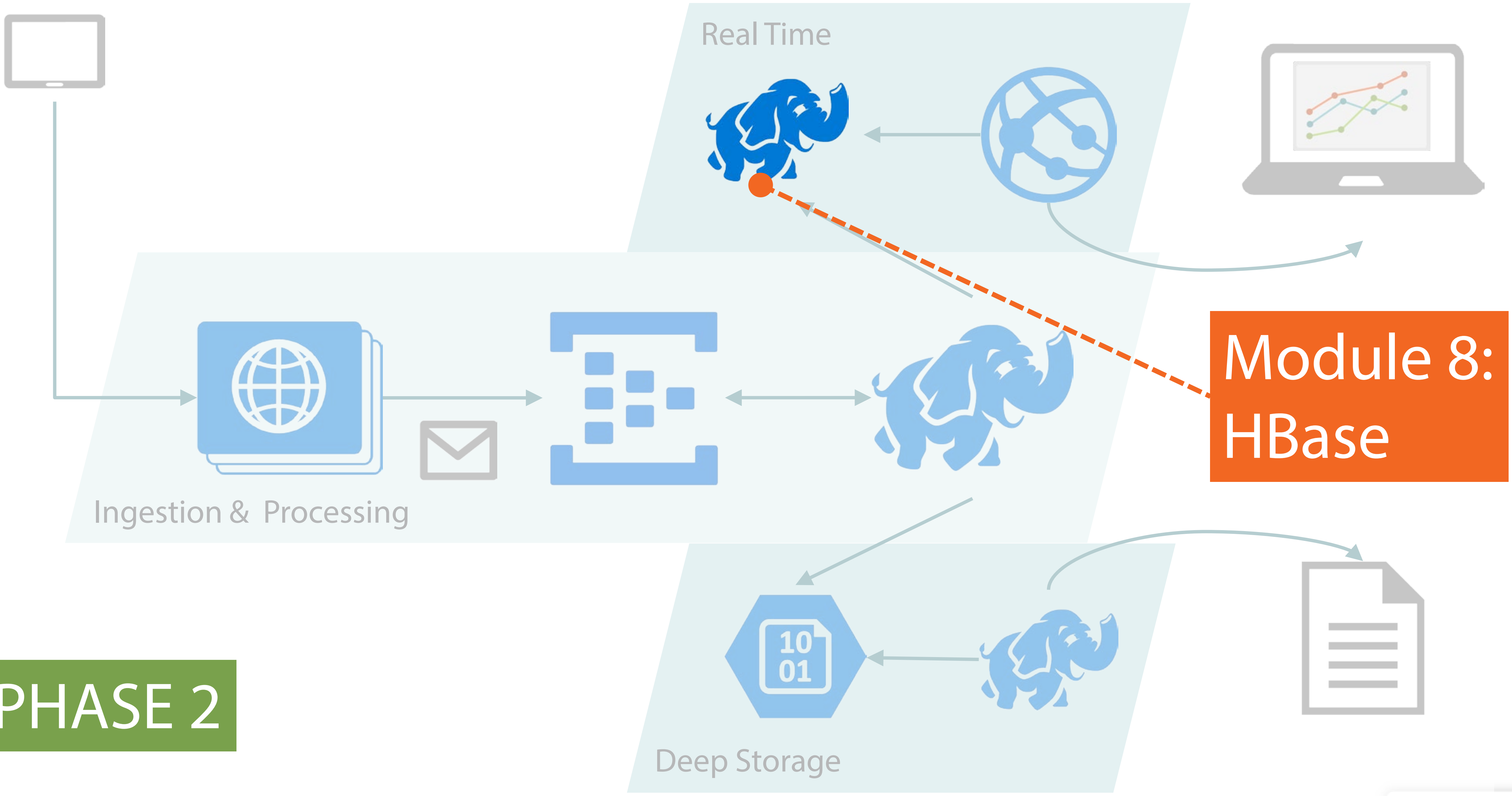
PHASE 1



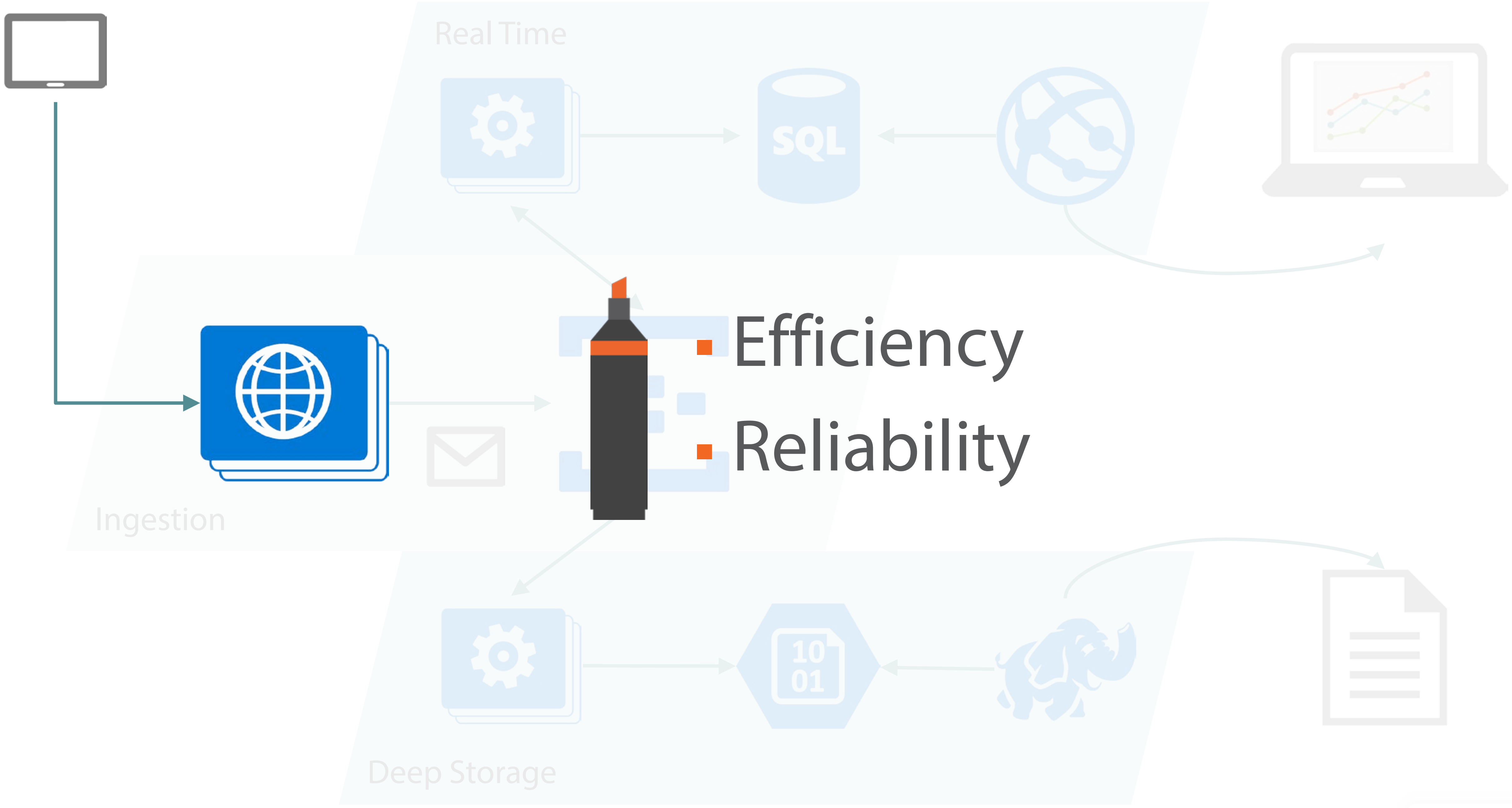
PHASE 2


Real Time
Ingestion & Processing
Deep Storage
10
01
Module 7: Storm

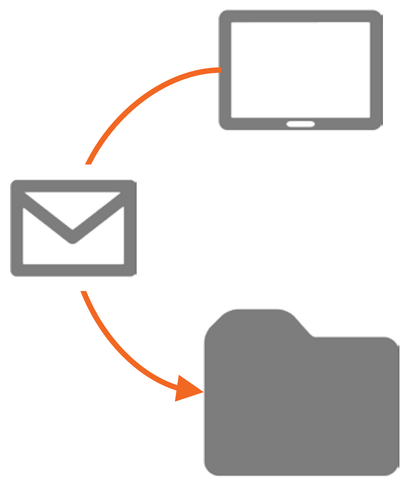
PHASE 2

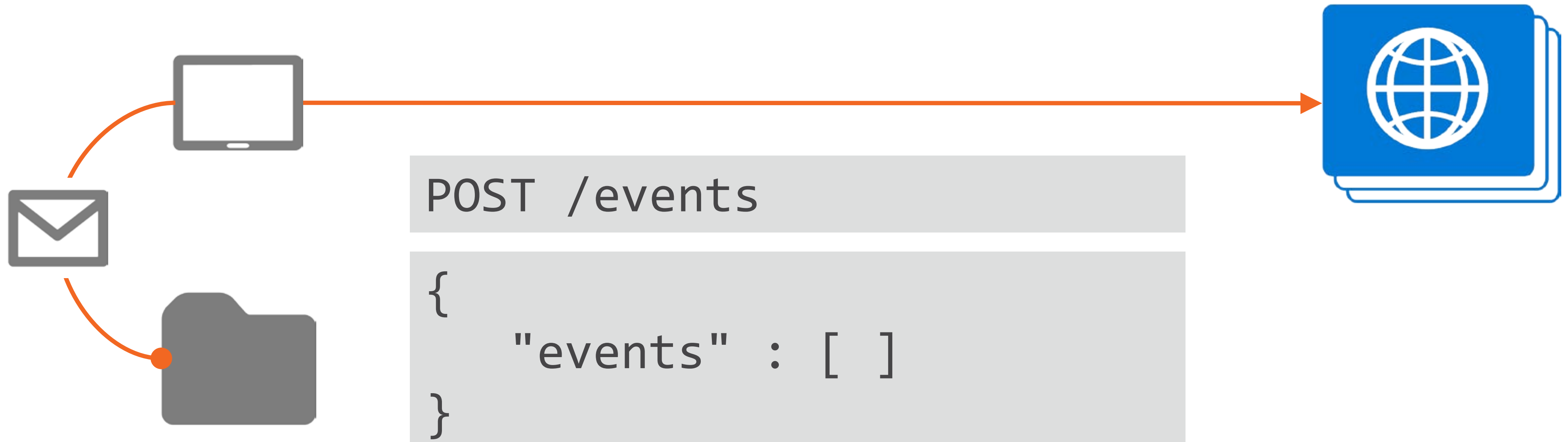

Real Time
Ingestion & Processing
Deep Storage
10
01
Module 8:
HBase

PHASE 2



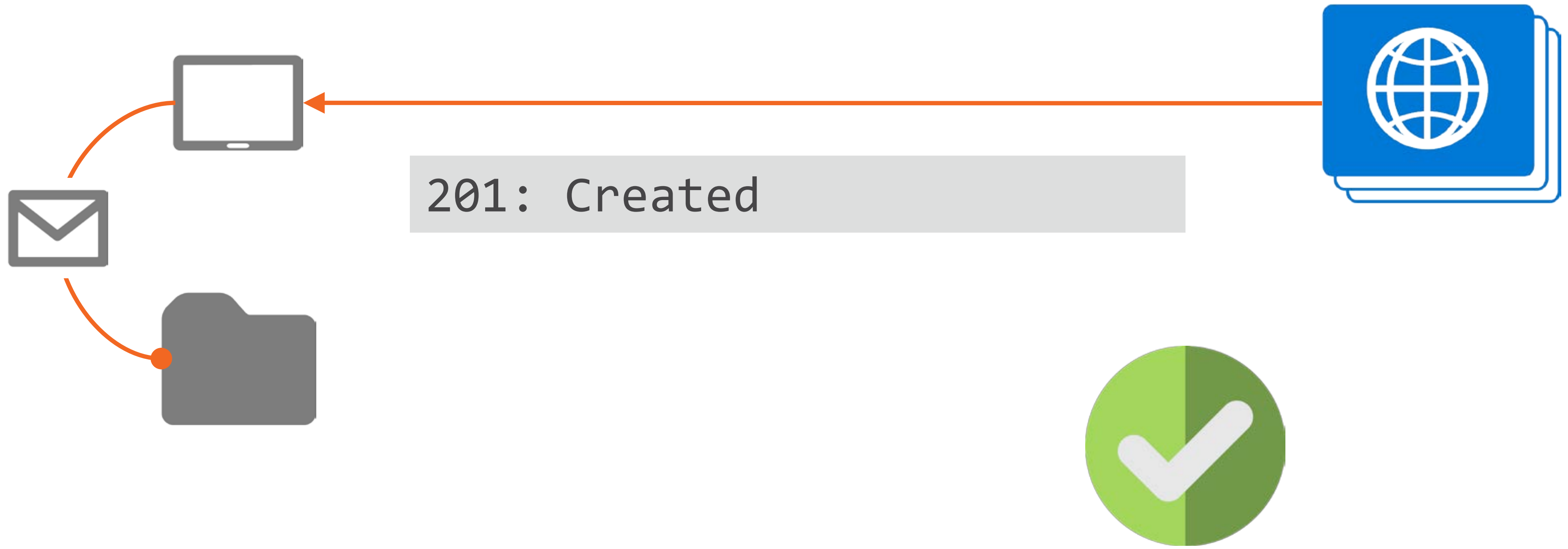
- Efficiency
- Reliability



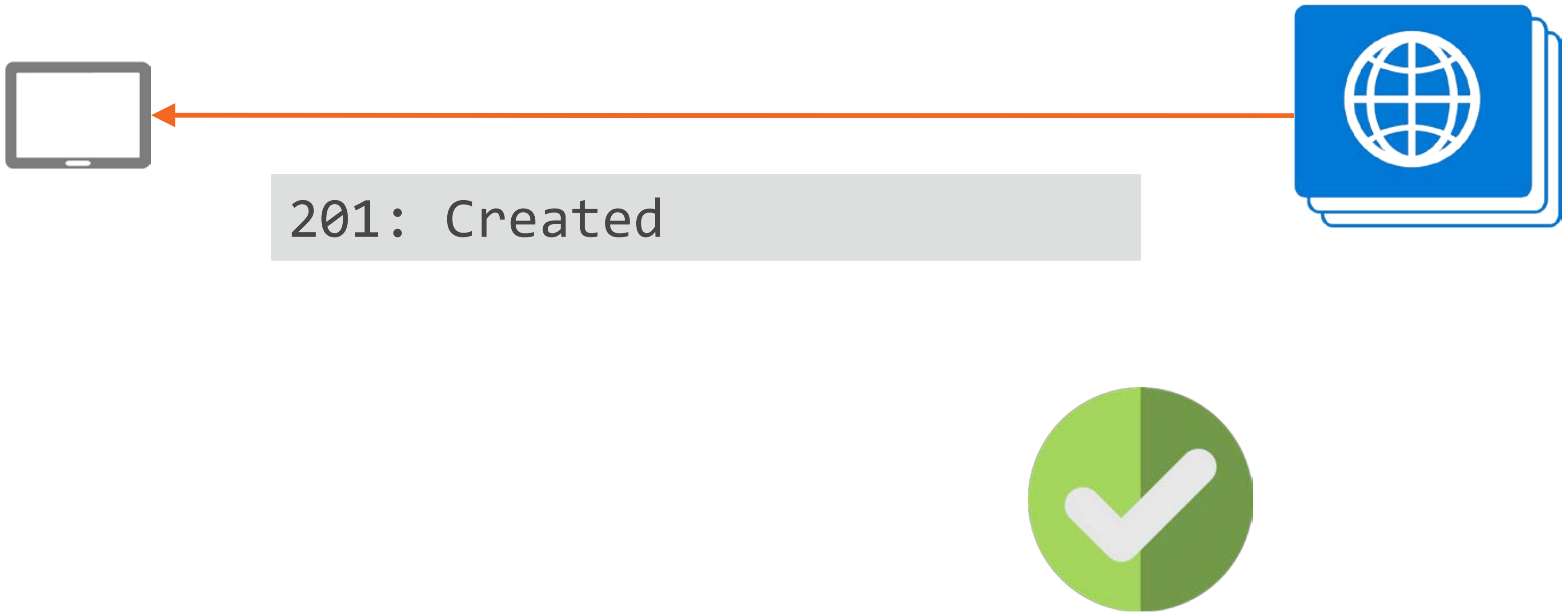


```
POST /events
```

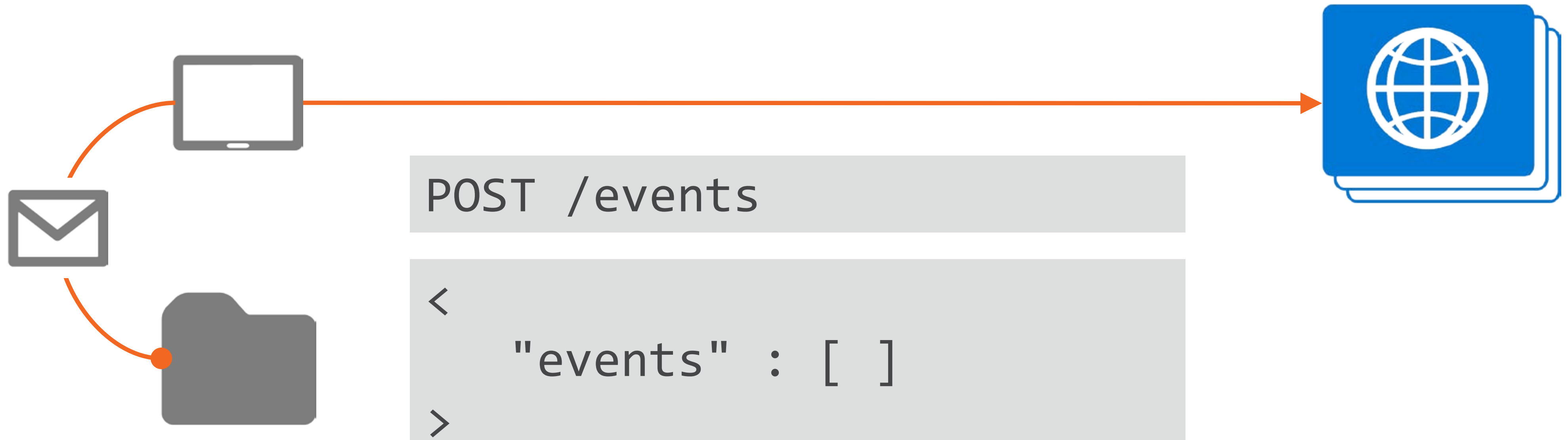
```
{
    "events" : [ ]
}
```



```
201: Created
```

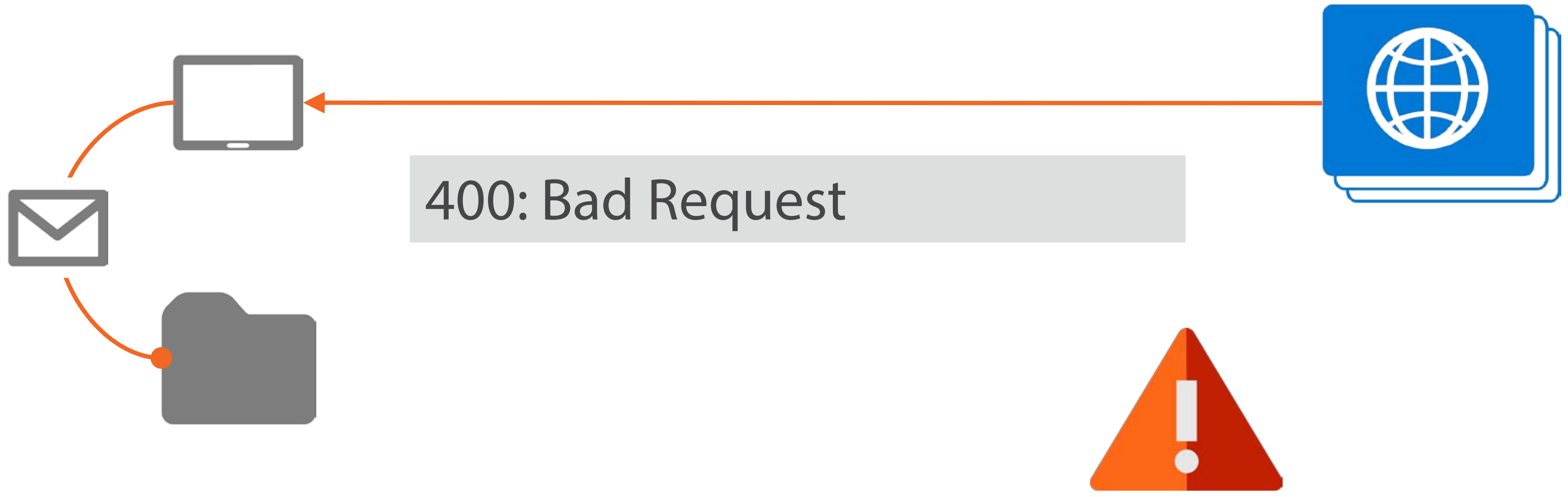


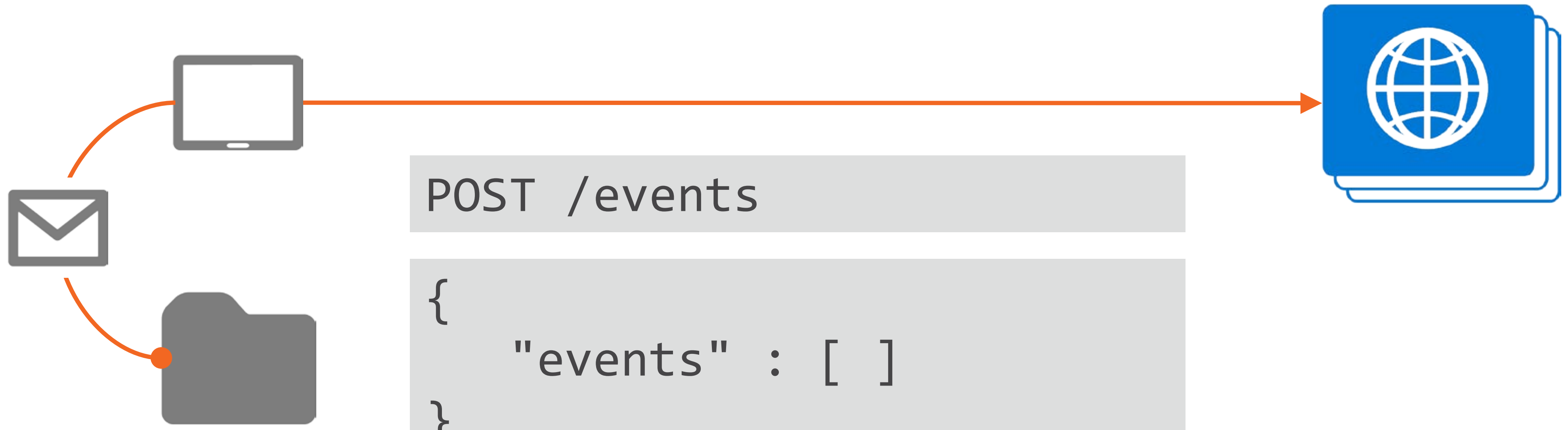
```
201: Created
```



```
POST /events
```

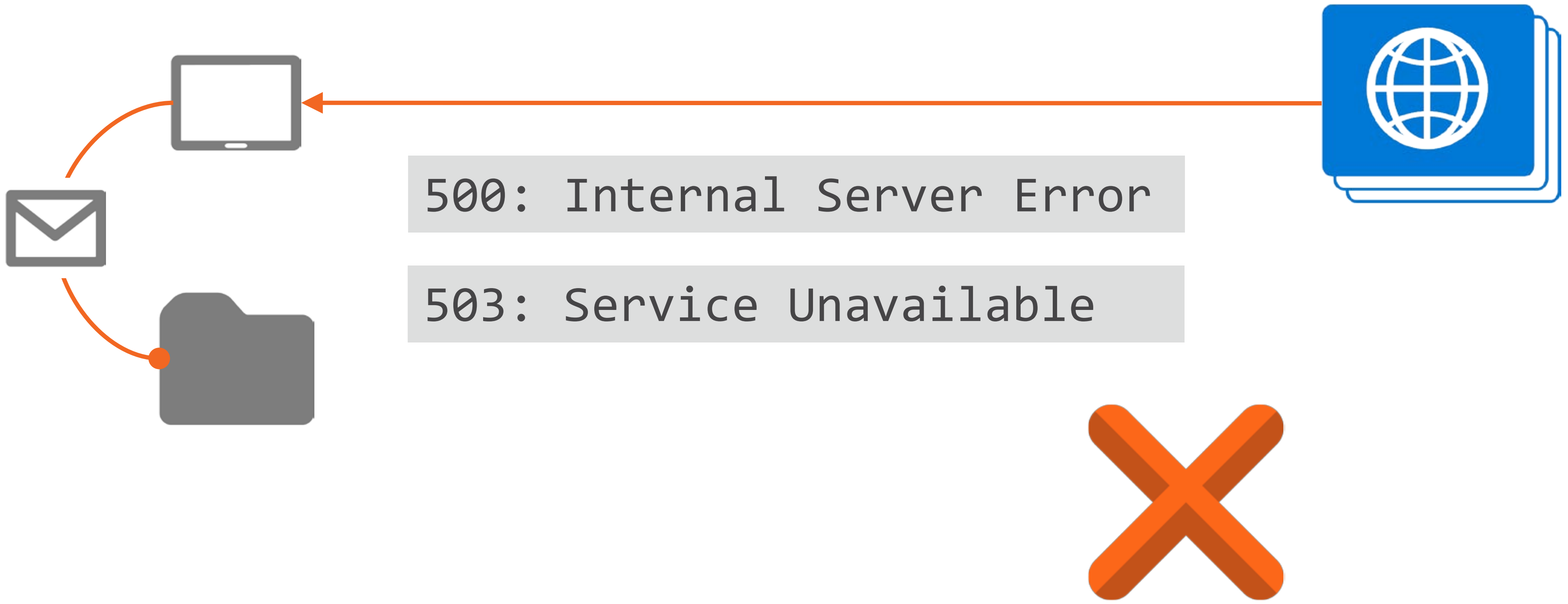
```
<
   "events" : [ ]
>
```


400: Bad Request



```
POST /events
```

```
{
    "events" : [ ]
}
```



```
500: Internal Server Error
```

```
503: Service Unavailable
```

Demo: EventsController

Dynamic JSON parsing

Exception handling

Logging

```
json = await requestMessage.Content.ReadAsStringAsync();
dynamic request = JObject.Parse(json);
events = (JArray)request.events;
```

Dynamic JSON

No fixed schema

```
{
    "deviceId": "def04e4c675d5f57f241b04484cfee62",
    "eventName": "device.gps.activated",
    "timestamp": 1406704244809
}
```

Sparse schema

Expected metadata

```
{
  "date":"2015-02-25 17:57:48.658 +00:00",
  "level":"Trace", "event": {},
  "logger":"EventsController", "loggerId":6419474
  "environment":"dev", "host":"SC-2013-DEV"
}
```

JSON logging

Used consistently

```
context.Response = new
HttpResponseMessage(HttpStatusCode.InternalServerError)
{
    ReasonPhrase = "Server error.",
    Content = new StringContent(content)
};
```

Logging exception filter

Trace errors from client to log

```
POST /Telemetry.Api/events
HTTP/1.1
Host: localhost
x-device-id: 123
```

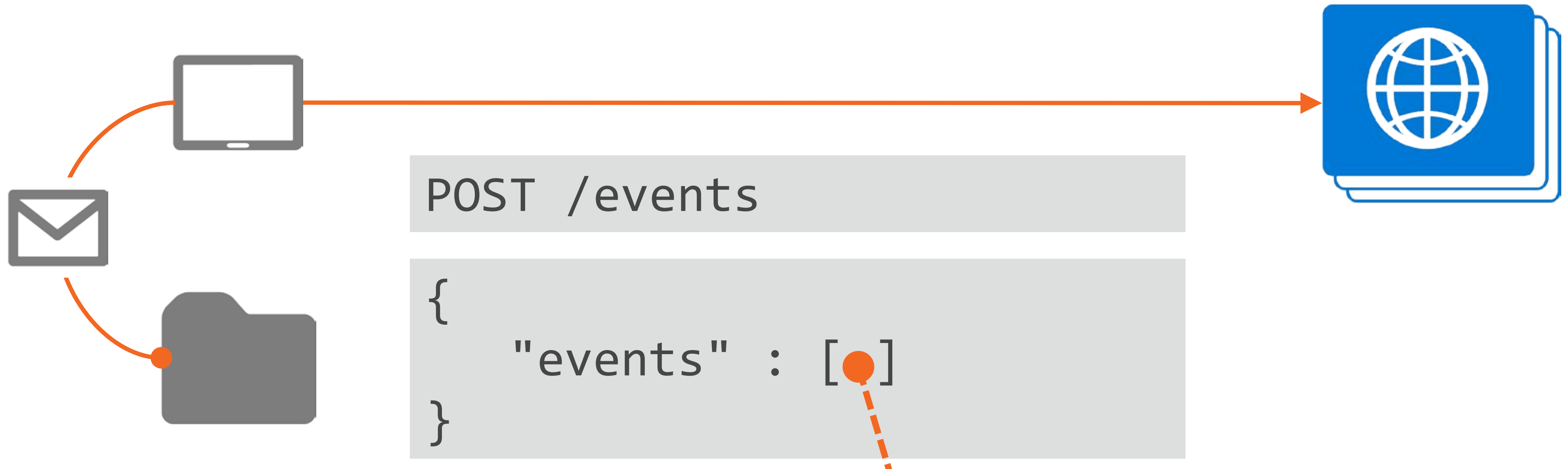
Custom request header

Identify client device

```
201 Created
x-api-version: 1.0
x-api-build: 1.0.0.0
```

Custom response headers

Identify API version

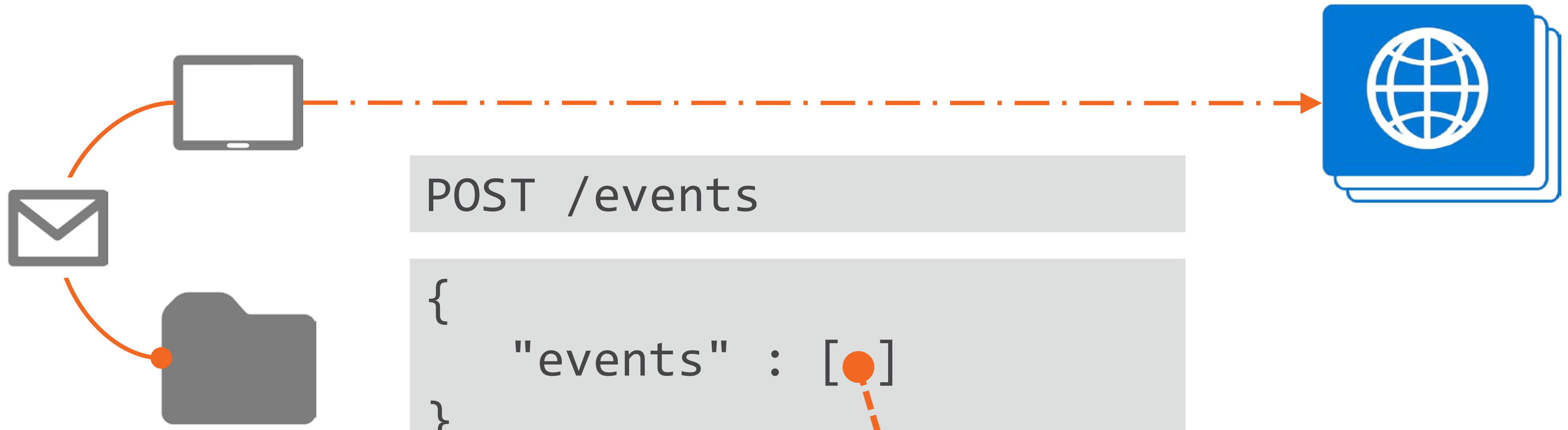


```
POST /events
```

```
{
    "events" : [●]
}
```

1,200 events

~300KB

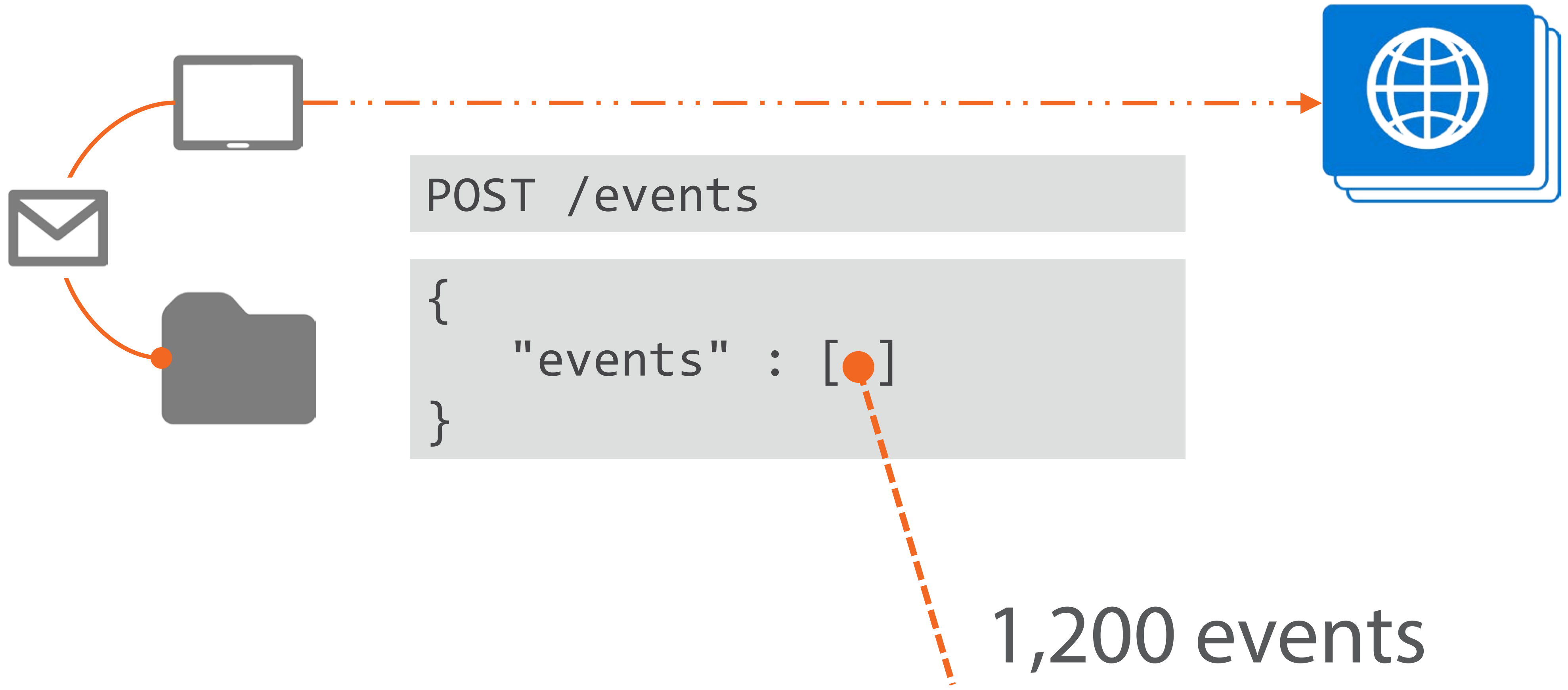


```
POST /events
```

```
{
    "events" : [●]
}
```

1,200 events

~300KB

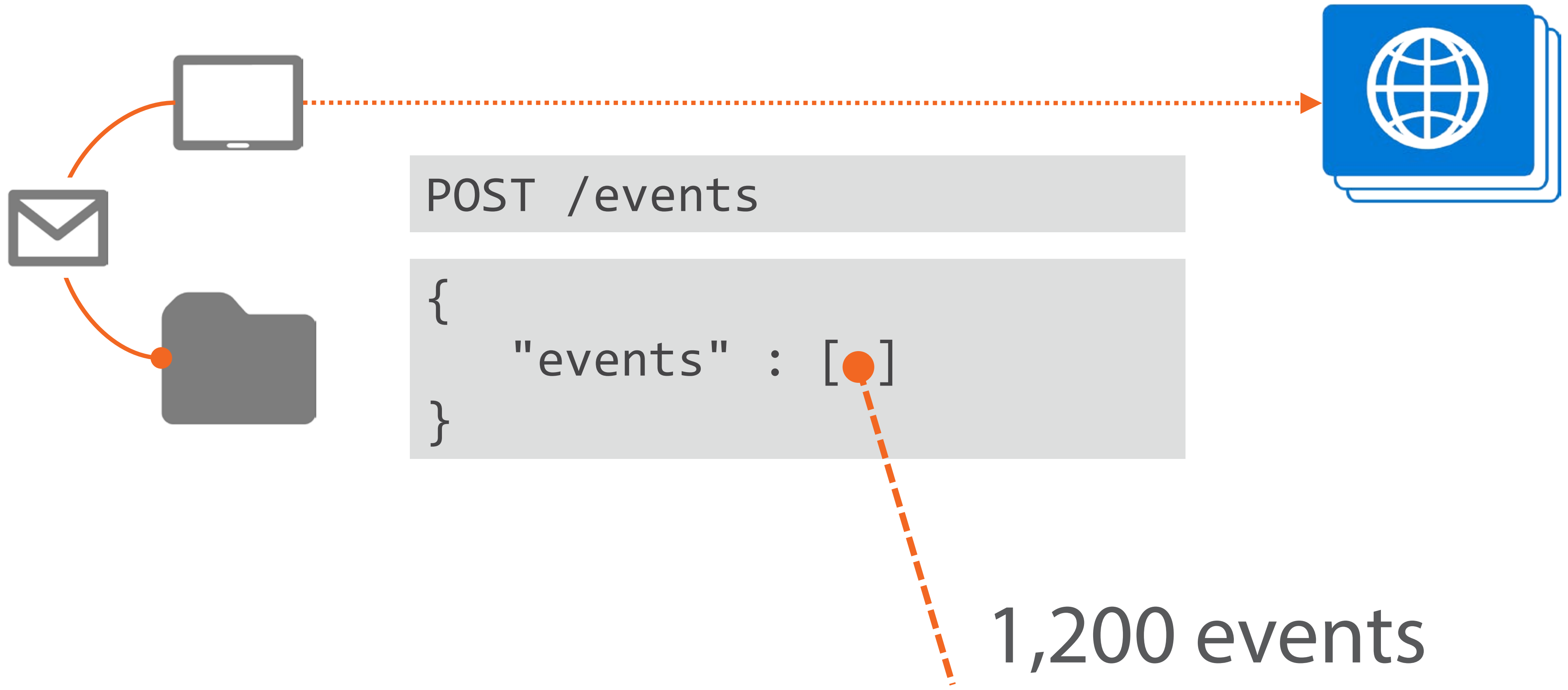


```
POST /events
```

```
{
    "events" : [•]
}
```

1,200 events

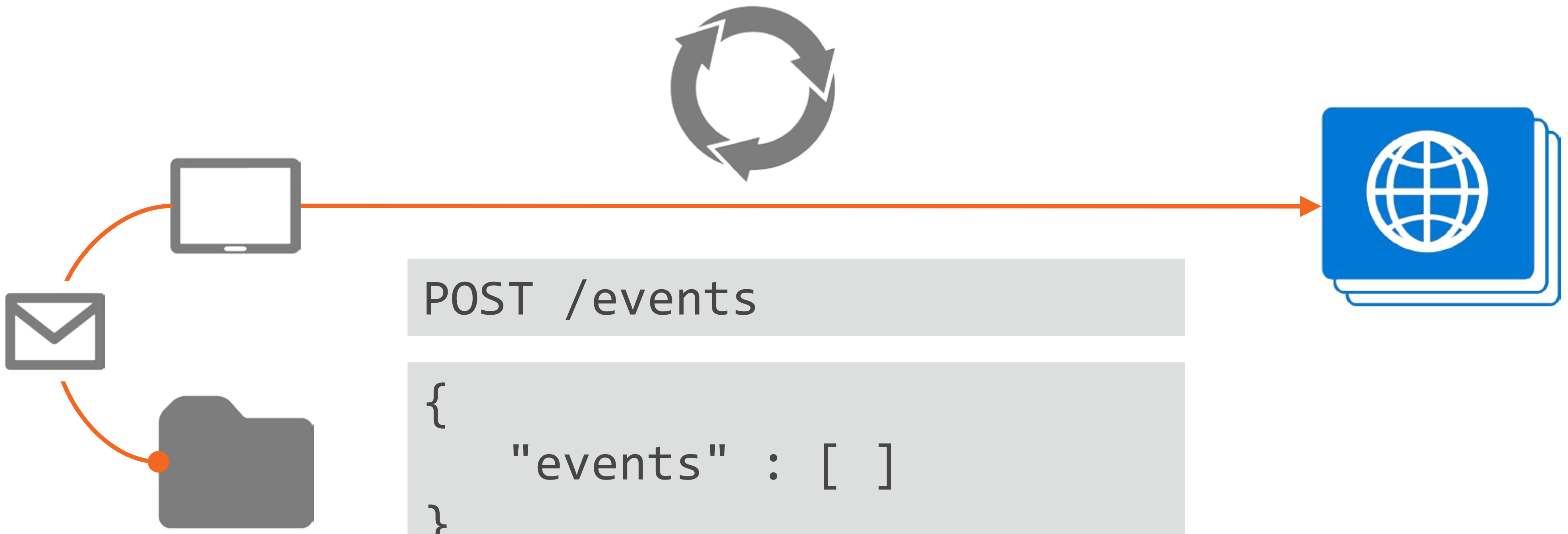
~300KB



```
POST /events
```

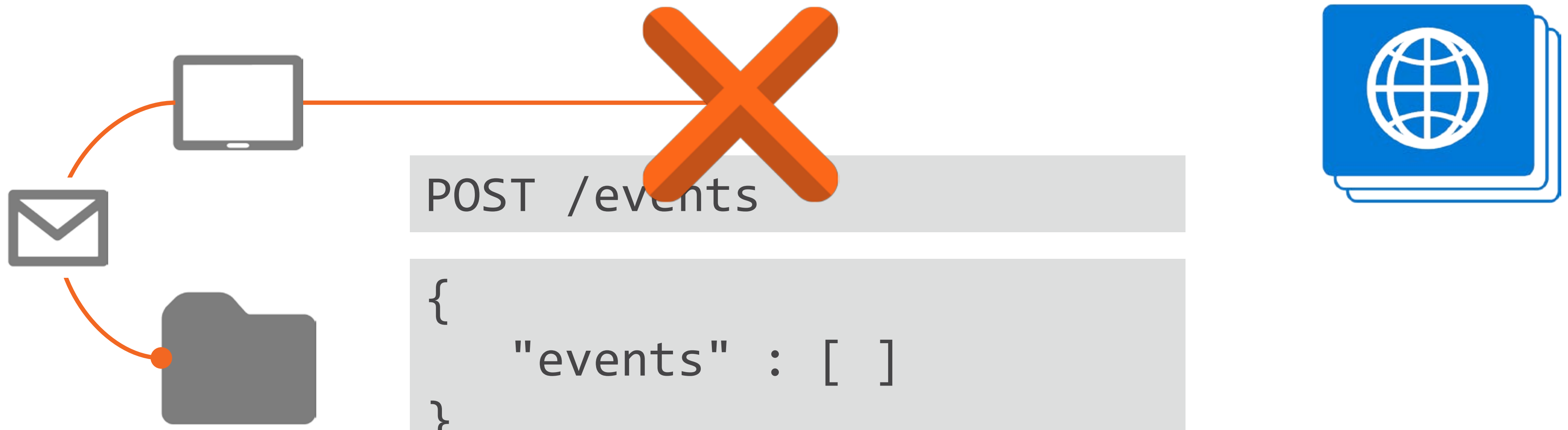
```
{
   "events" : [●]
}
```

1,200 events
~300KB



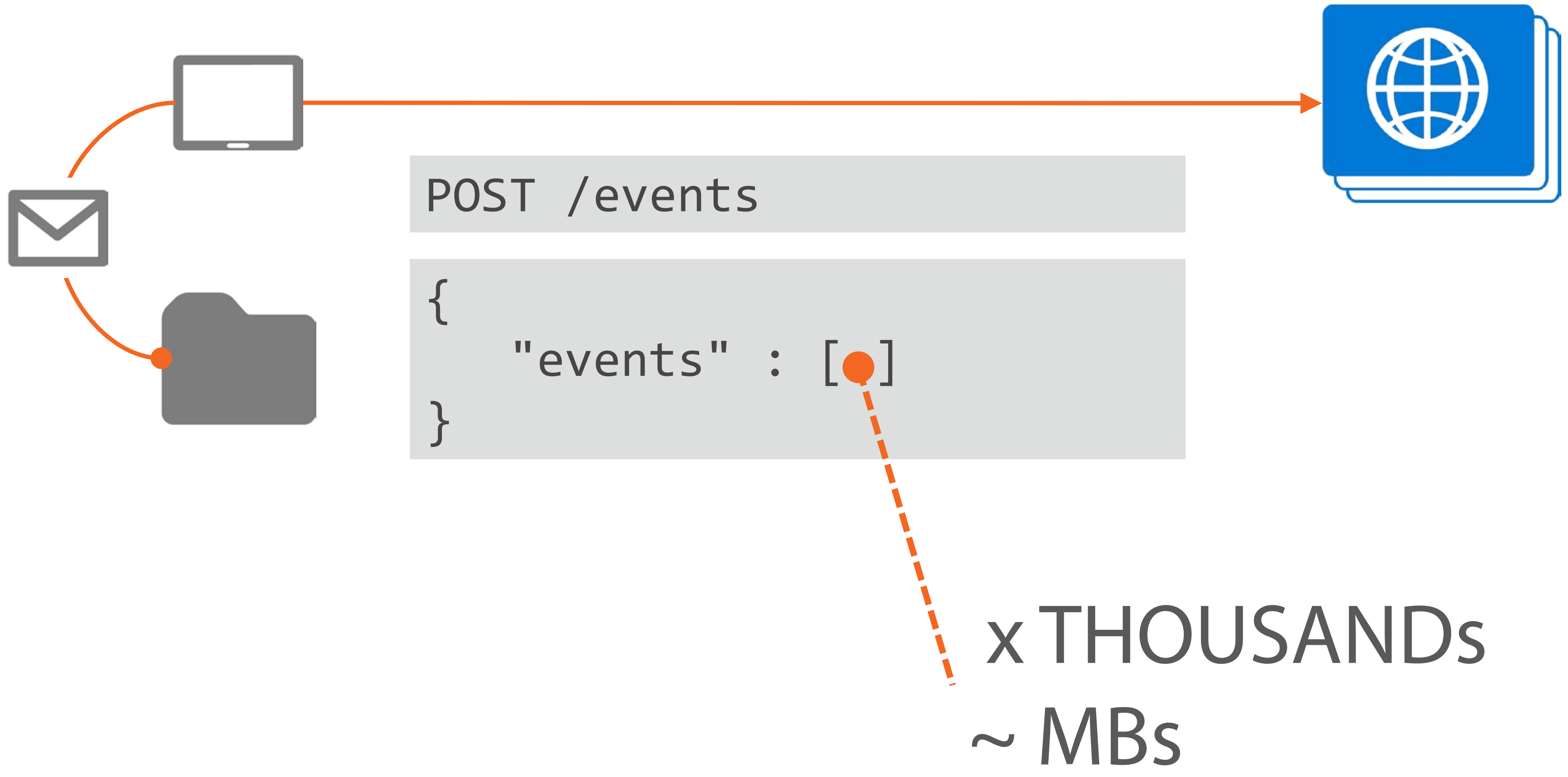
```
POST /events
```

```
{
   "events" : [ ]
}
```



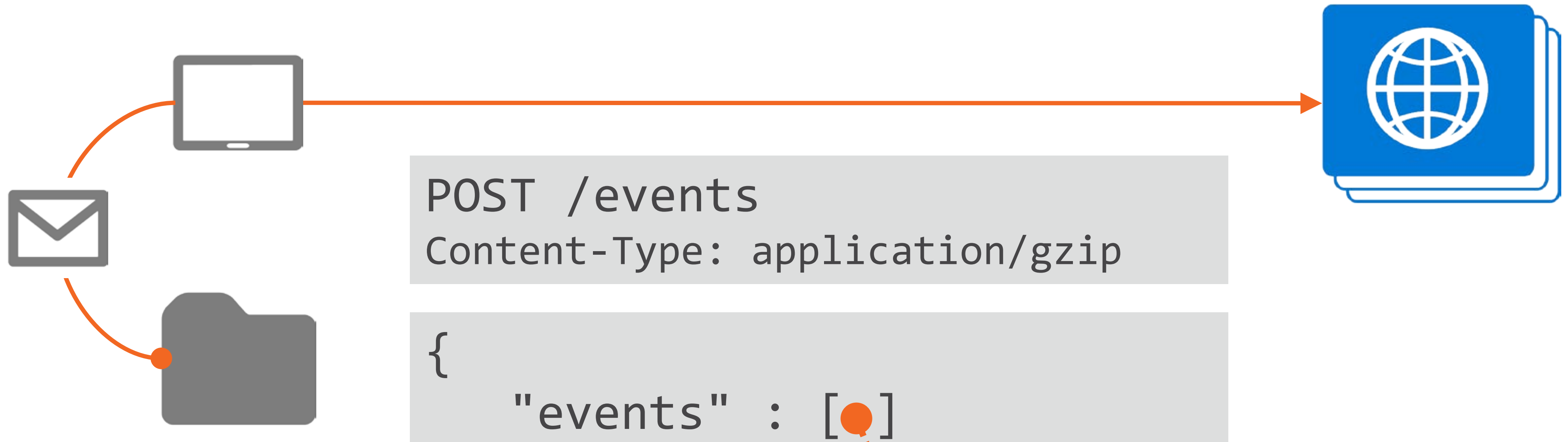
```
POST /events
```

```
{
   "events" : [ ]
}
```



```
POST /events
```

```
{
    "events" : [●]
}
```

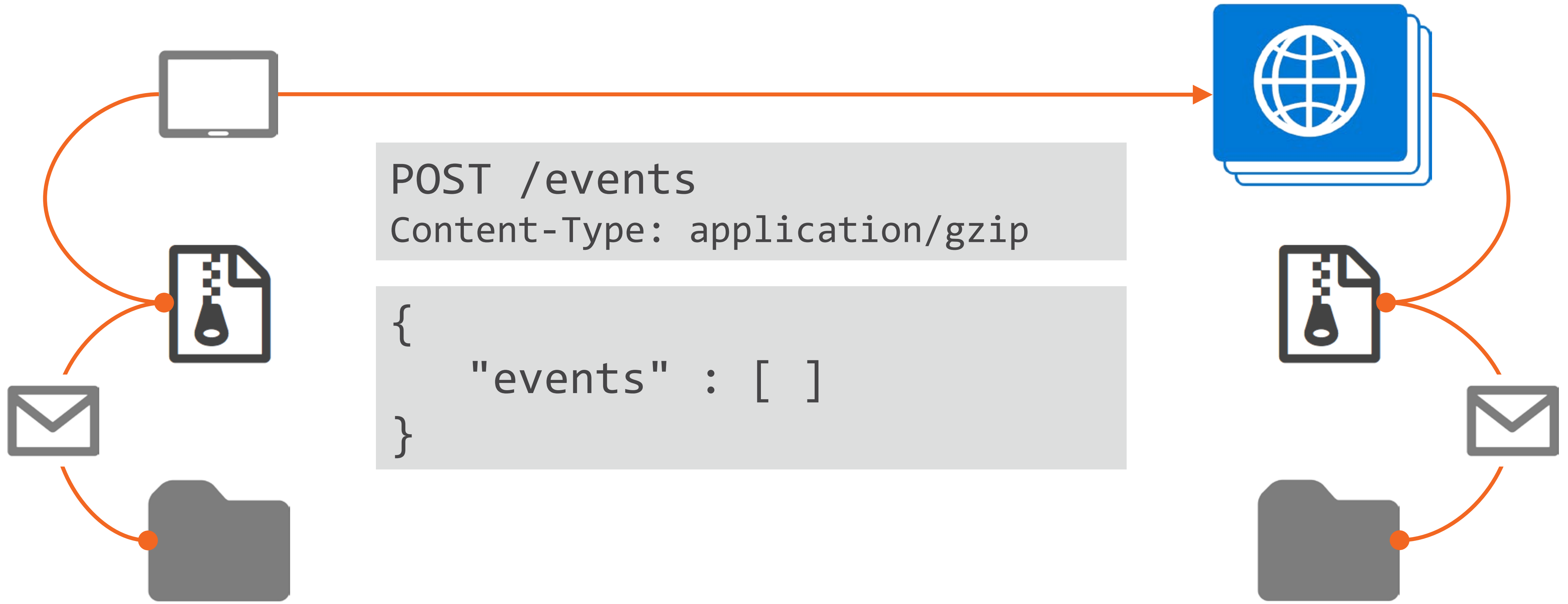


```
POST /events
Content-Type: application/gzip
```

```
{
    "events" : [●]
}
```

1,200 events

<30KB



```
POST /events
Content-Type: application/gzip
```

```
{
    "events" : [ ]
}
```

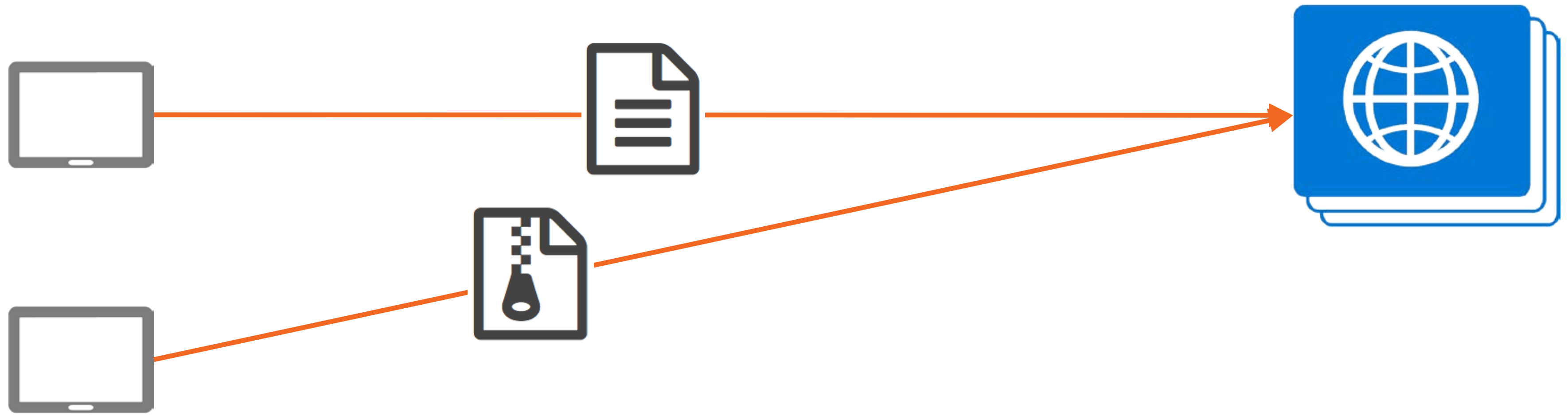
Demo: `GZipToJsonHandler`

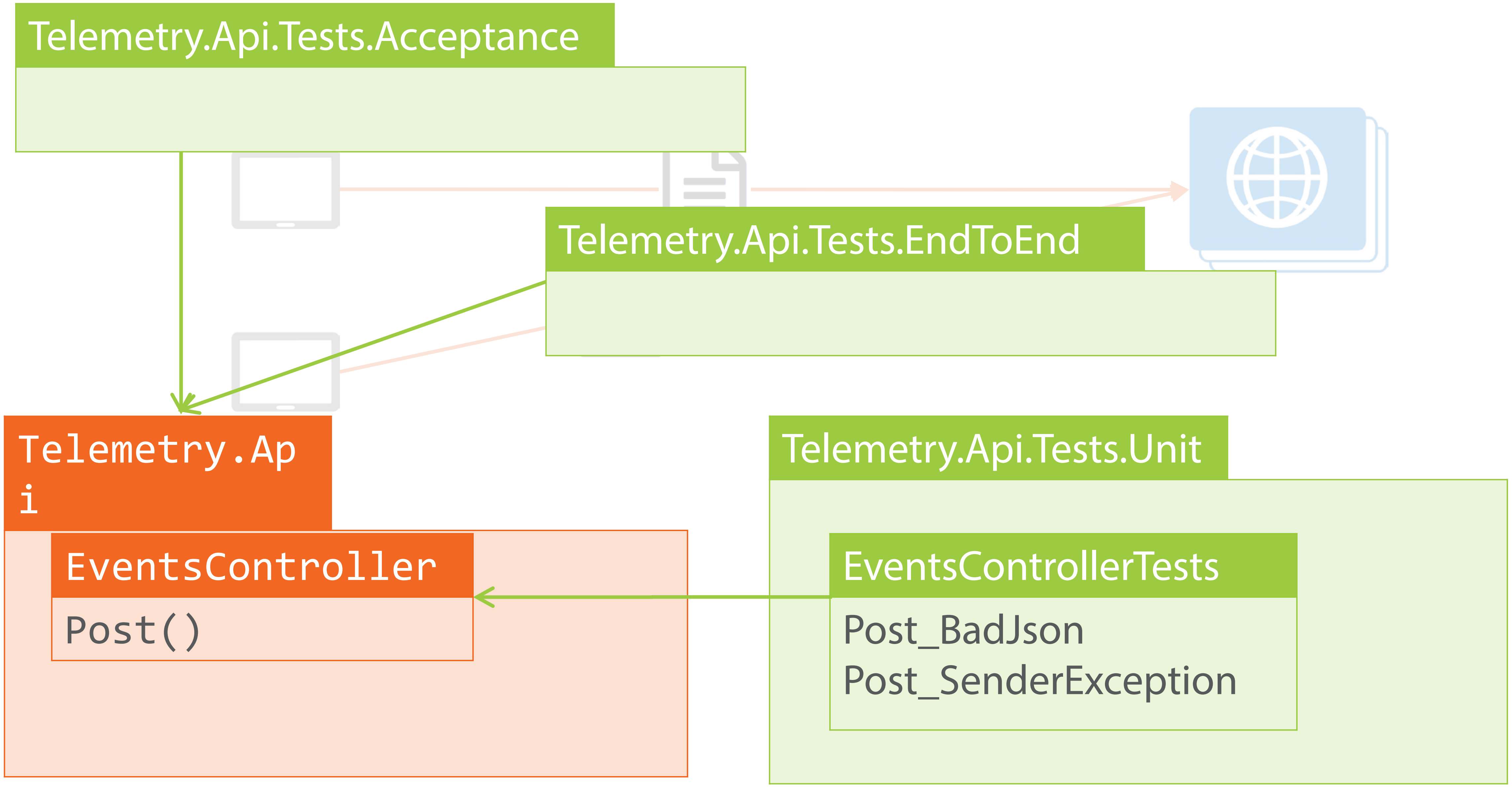
DelegatingHandler

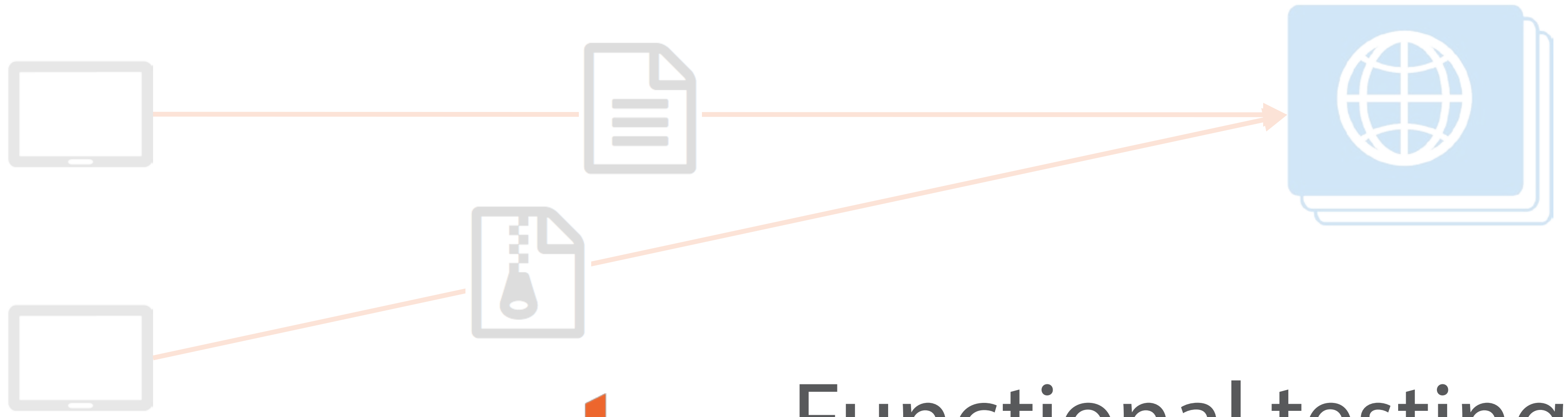
Add to controller route

Handler decompresses

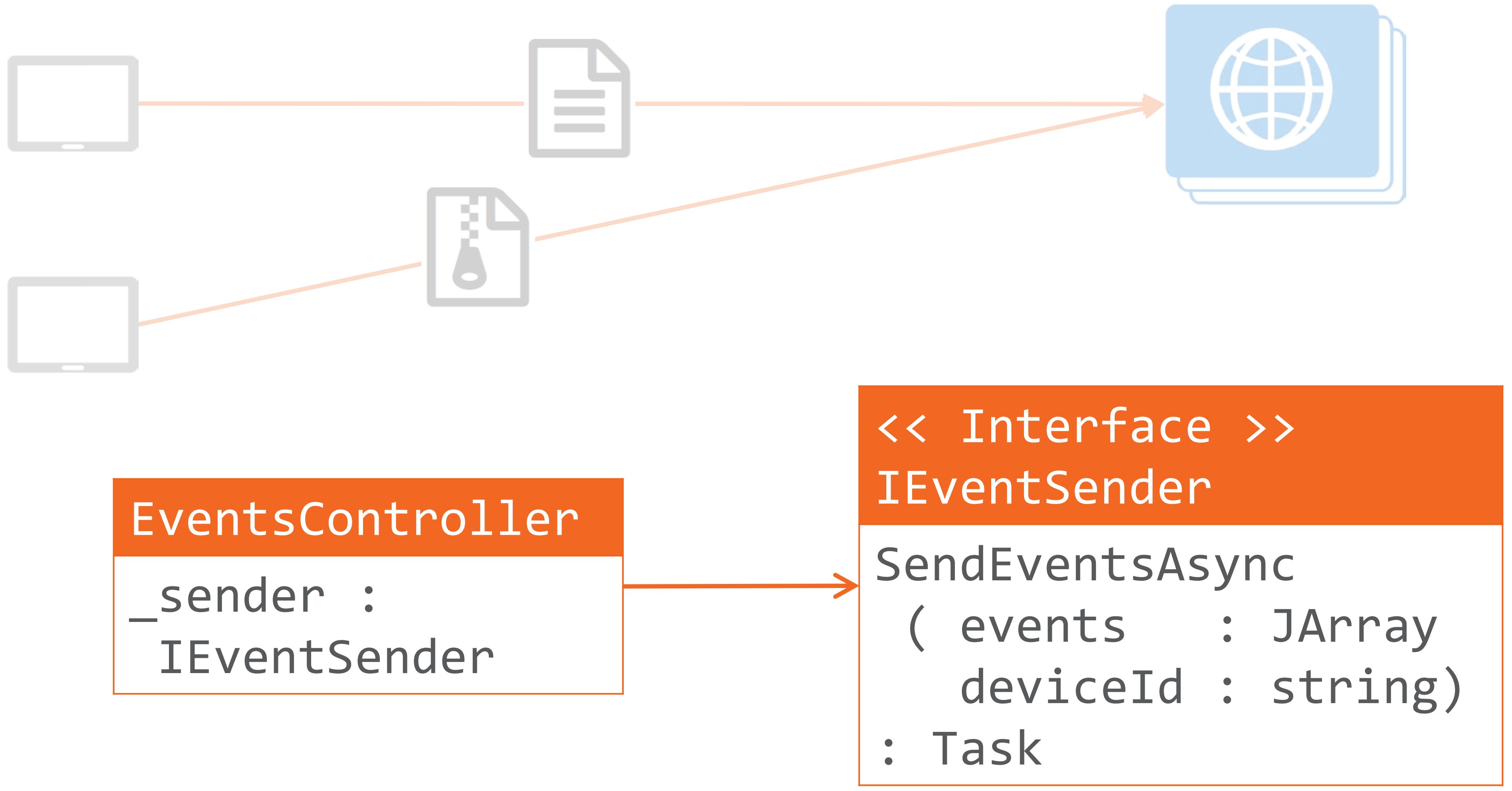
Controller gets JSON




Telemetry.Api.Tests.Acceptance
Telemetry.Api.Tests.EndToEnd
Telemetry.Ap
i
EventsController
Post()
Telemetry.Api.Tests.Unit
EventsControllerTests
Post_BadJson
Post_SenderException



- Functional testing
- IDisposable
- Exception handling
- Load testing



```
EventsController
_sender :
 IEventSender
```

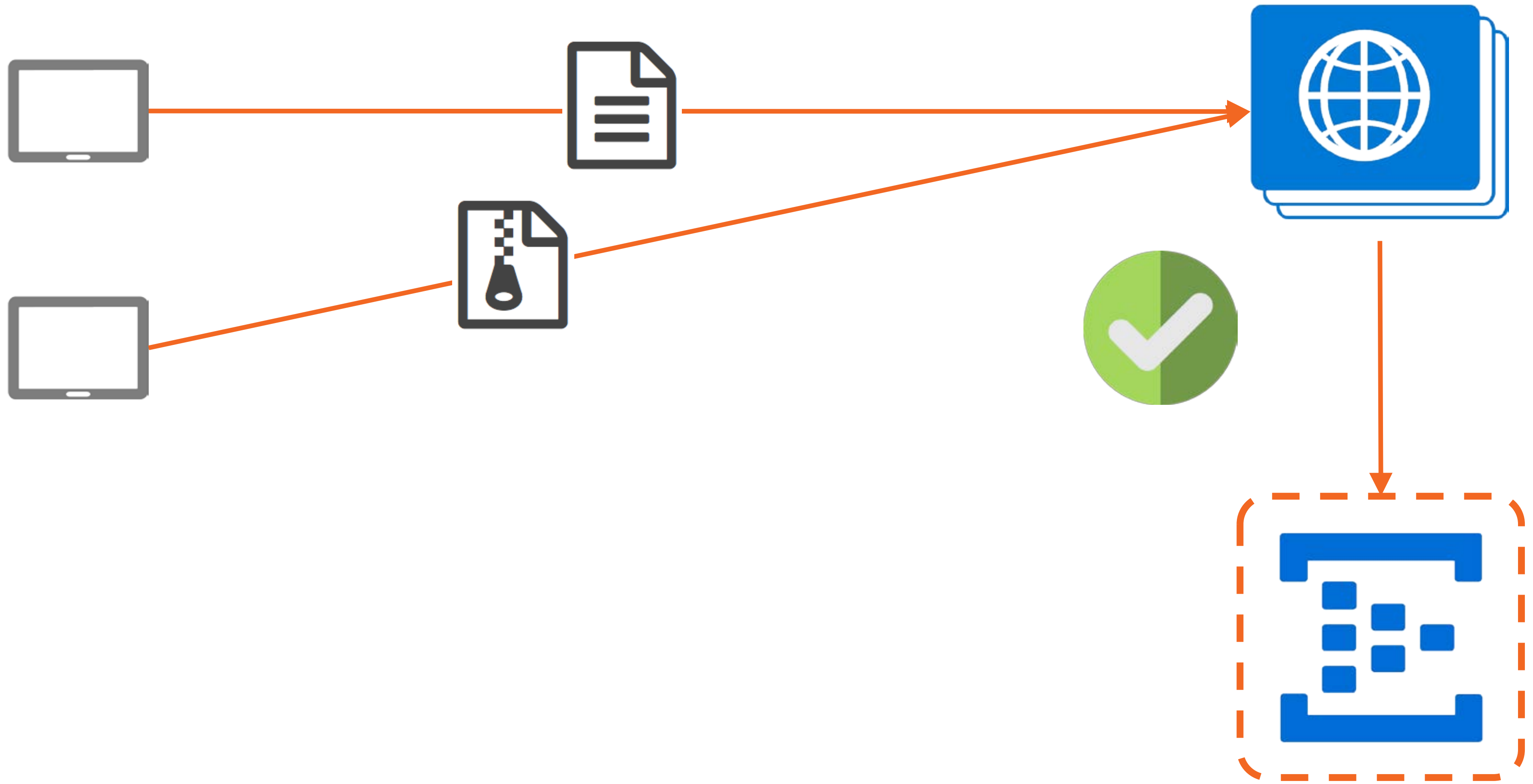
```
<< Interface >>
IEventSender
SendEventsAsync
 ( events   : JArray
   deviceId : string)
: Task
```

Demo: `IEventSender`

Abstracts Event Hubs

Simple interface

Injected into Controller



- Efficiency
- Reliability
- Traceability

Efficiency

High volume of messages

Batching & zipping reduces API load

Lean API

Cheaper to run, easier to support

Reliability

Keep all events

Client stores locally if API fails

Solid API

Avoid event processing backlog

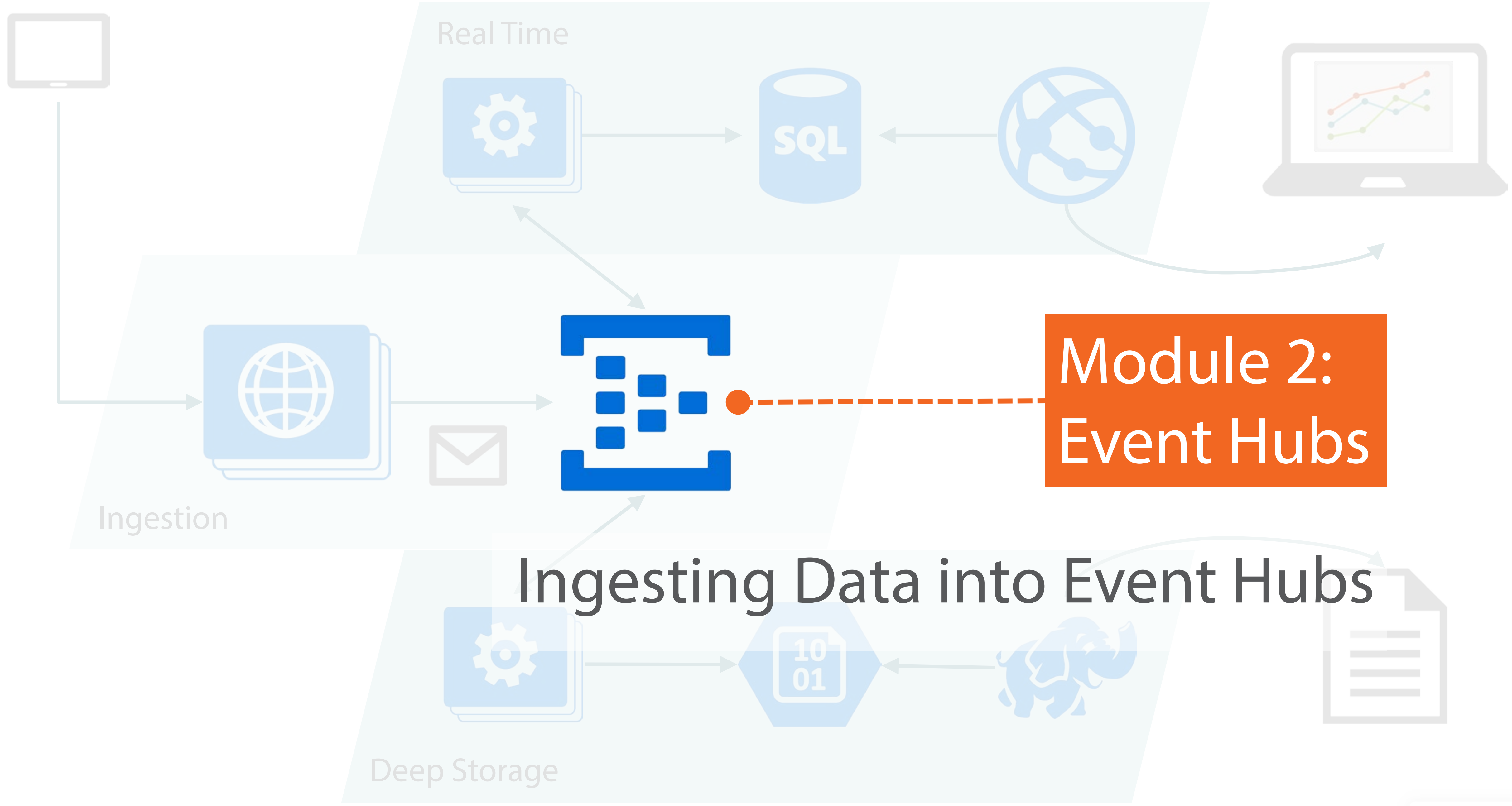
Traceability

Centralized logging

JSON for analysing logs

Exceptions and heartbeats

Know when things are wrong or right



Module 2:
Event Hubs

Ingesting Data into Event Hubs