

Building Real-Time Visualizations at Scale

Mike Barry @msb5014

Kevin Robinson @krob

Hello!

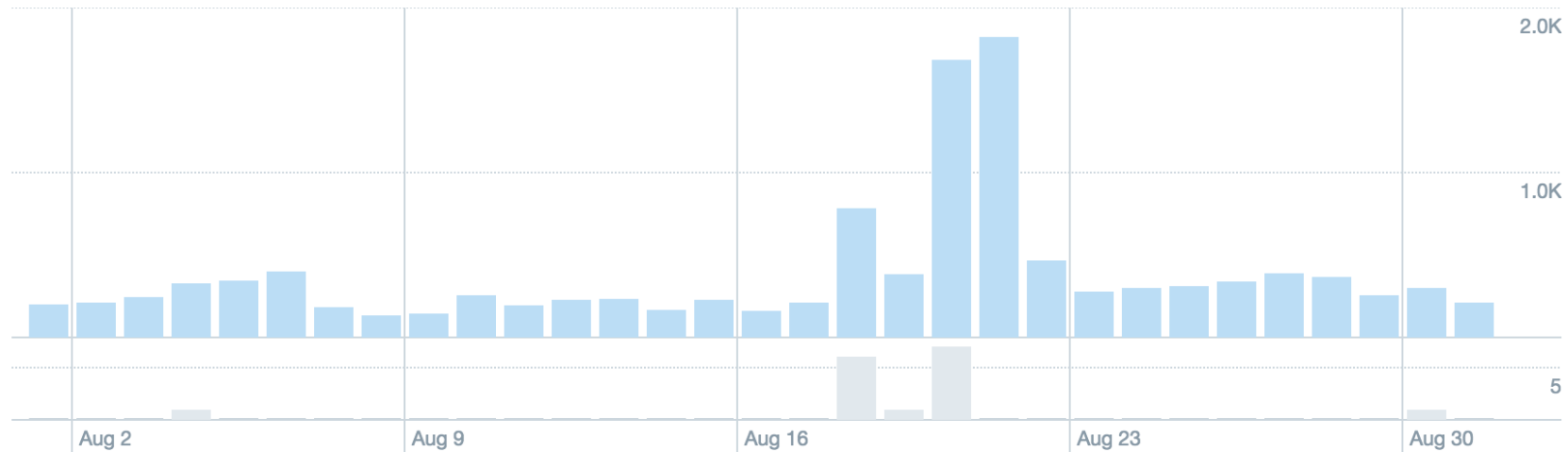


Hello!



analytics.twitter.com

Your Tweets earned **11.8K impressions** over this **31 day** period



Hello!



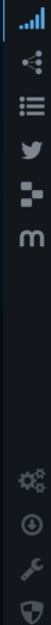
Answers

iOS

Application 1

ios.com.crashlytics.app1

Admin
Crashlytics, Inc.



7,384

ACTIVE USERS RIGHT NOW



DAILY ACTIVE USERS

211.0k

▲ 2.8%

DAILY NEW USERS

5.0k

▲ 4.2%

MONTHLY ACTIVE USERS

665.3k

▲ 3.6%

CRASH-FREE USERS

96.3%

▲ 5.0%

SESSIONS

633.6k

▲ 7.0%

PURCHASED ITEM

211.0k

▲ 2.8%

211.0k ▲ 2.8%

DAILY ACTIVE USERS



Building Real-time Visualizations

Real-time

Actionable

User-focused



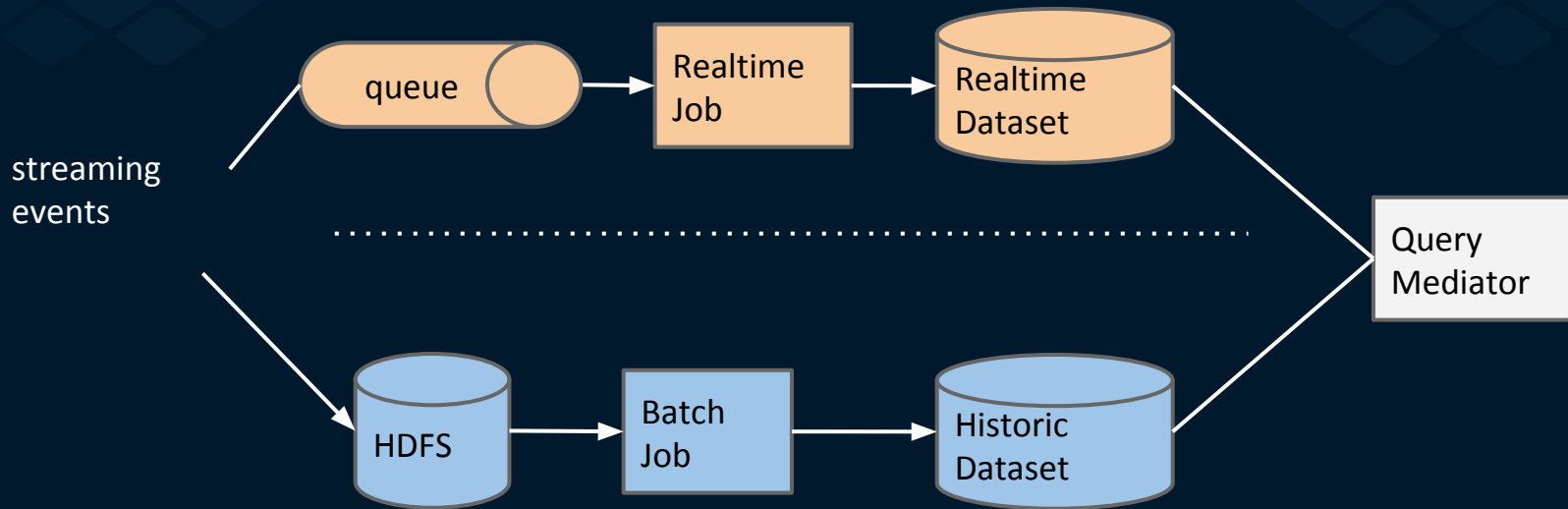
Analytics at Twitter

Architecture

Higher-level abstractions

Human flexibility





Typical Analytics Pipeline

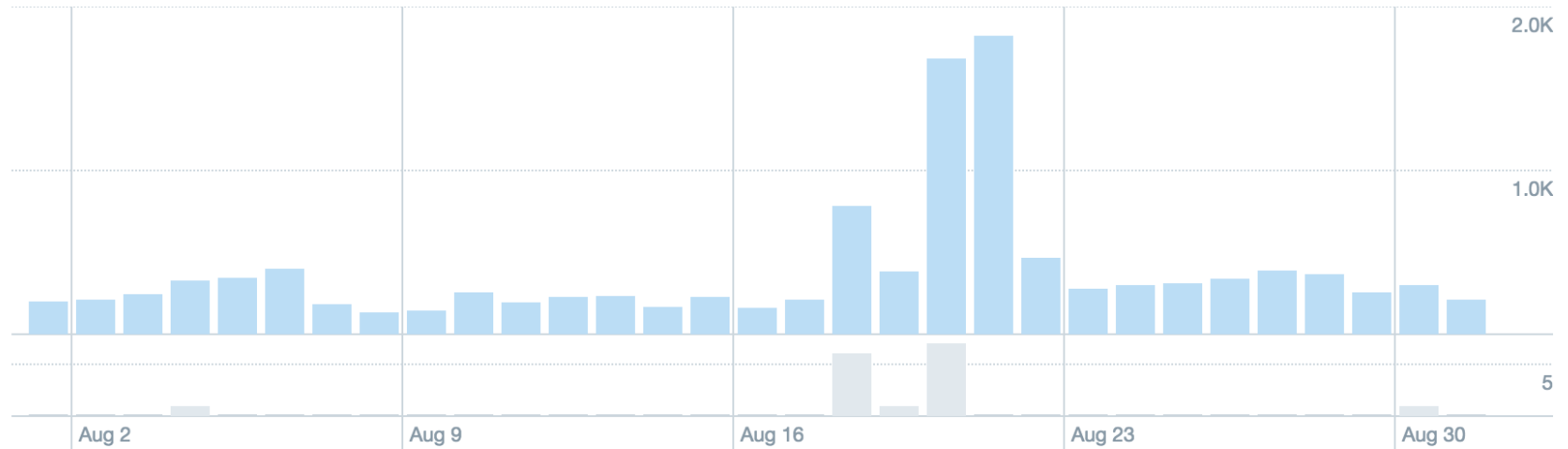


**Do more work on write
so that reads are fast**

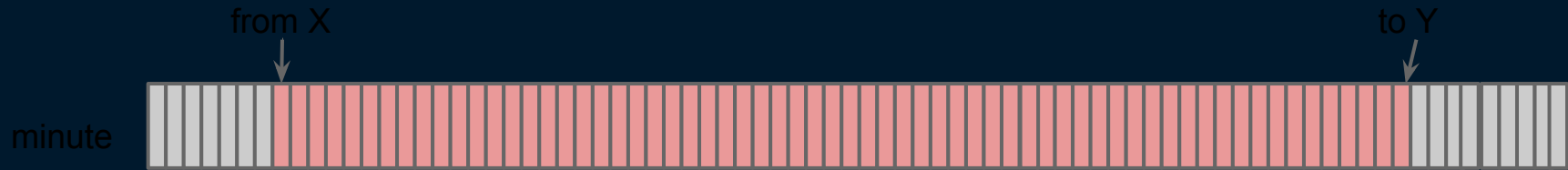


How many impressions from X to Y?

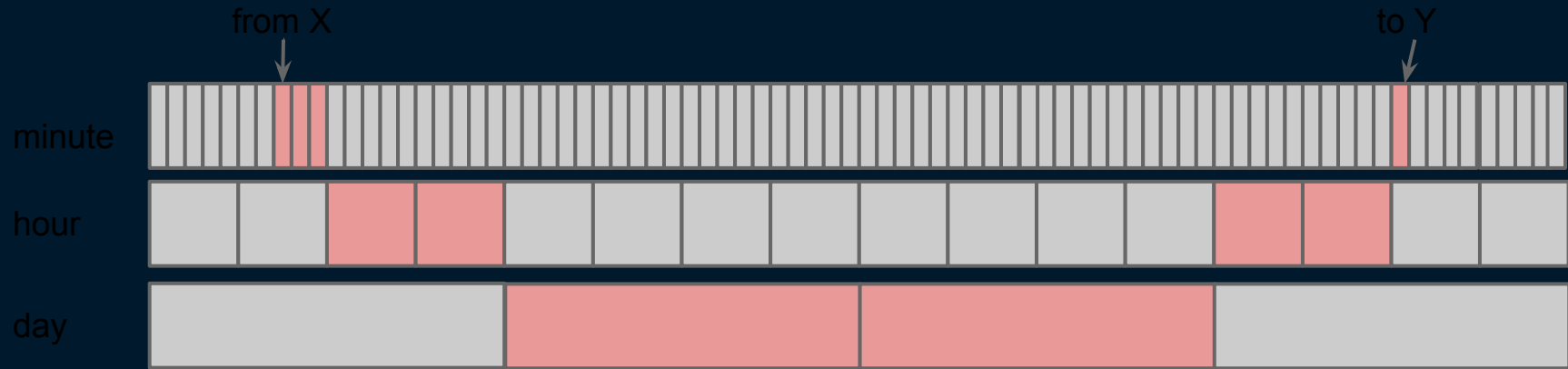
Your Tweets earned **11.8K impressions** over this **31 day** period



How many impressions from X to Y?

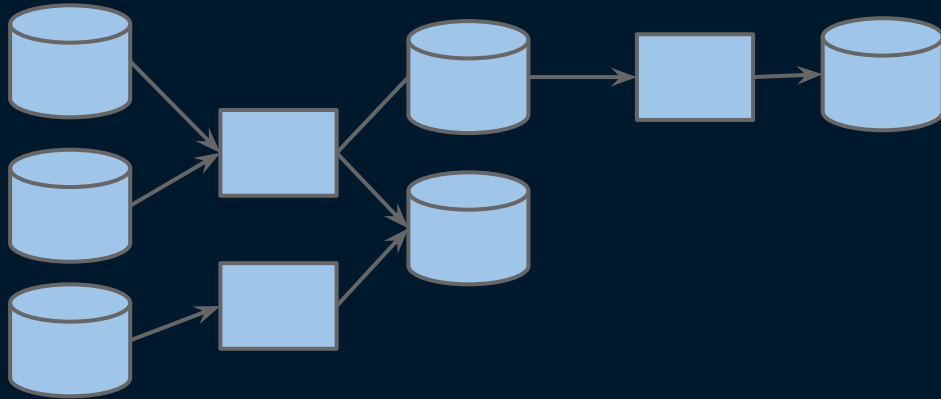
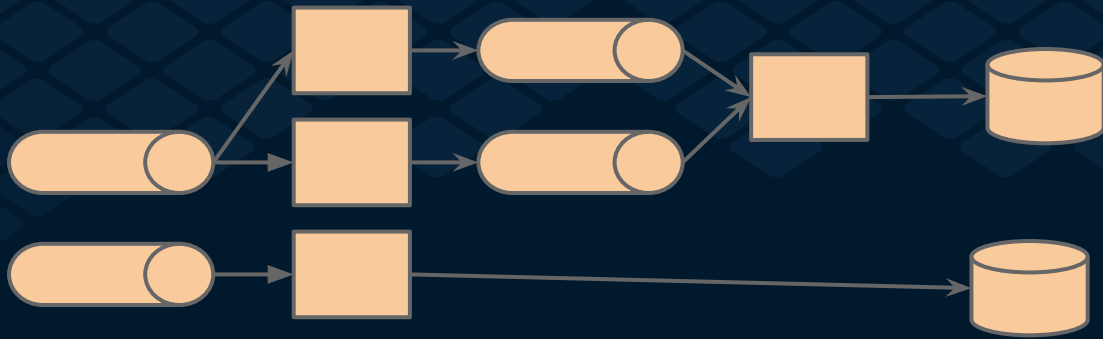


How many impressions from X to Y?



Abstractions





Scaling up



**Communicate Fearlessly
to Build Trust**

Human Flexibility

Globally-available data

+ Flexible individuals

+ Hack weeks

= innovation



Analytics at Twitter

Architecture

Higher-level abstractions

Human flexibility



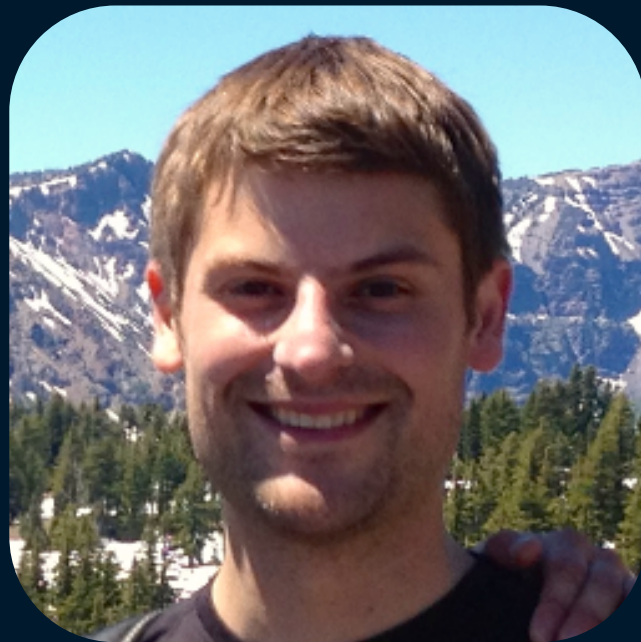




answers

No analysis required.

Finally, mobile app analytics you don't need to analyze.





7,384

ACTIVE USERS RIGHT NOW



DAILY ACTIVE USERS ?

211.0k

▲ 2.8%

DAILY NEW USERS ?

5.0k

▲ 4.2%

MONTHLY ACTIVE USERS ?

665.3k

▲ 3.6%

CRASH-FREE USERS ?

96.3%

▲ 5.0%

SESSIONS ?

633.6k

▲ 7.0%

PURCHASED ITEM ?

211.0k

▲ 2.8%

211.0k ▲ 2.8%

DAILY ACTIVE USERS



200K



○ answers **events**



Initial assumptions

Shortest path to usefulness

Real users and data change everything



Initial assumptions

Shortest path to usefulness

Real users and data change everything



No data!

Existing data sources? nope

Predictable usage or distributions? nope

hmm...

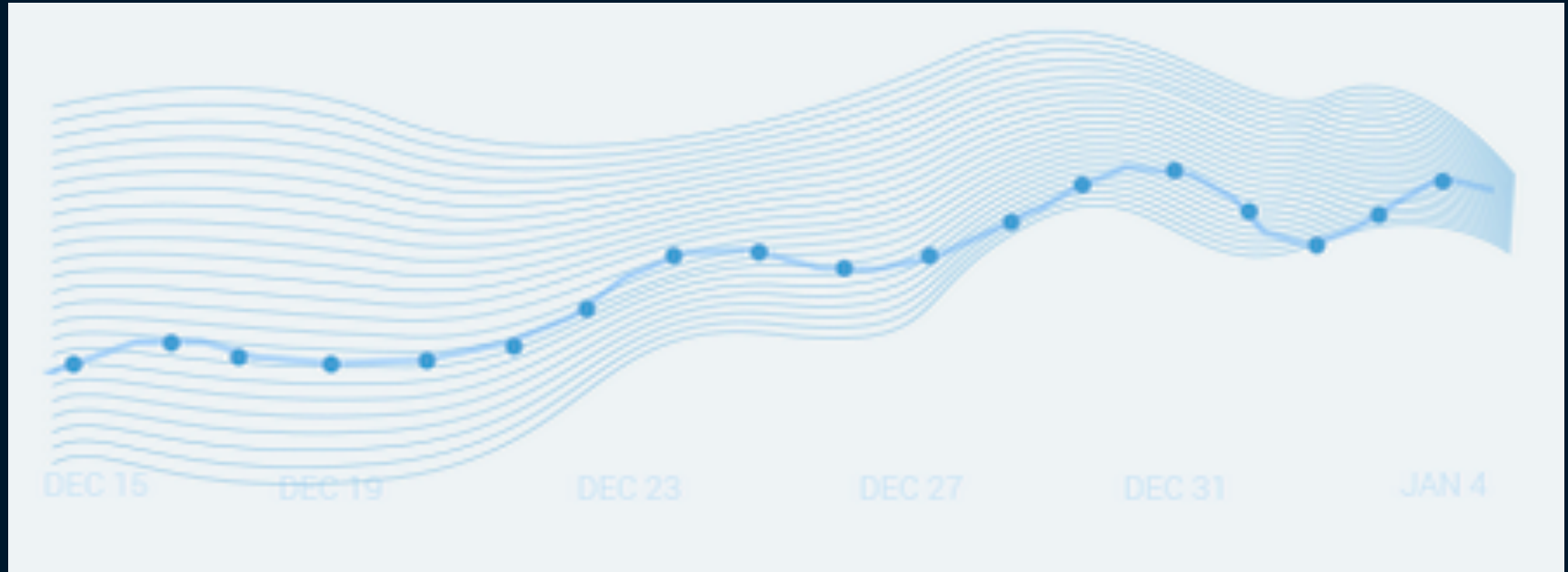


THE DATA

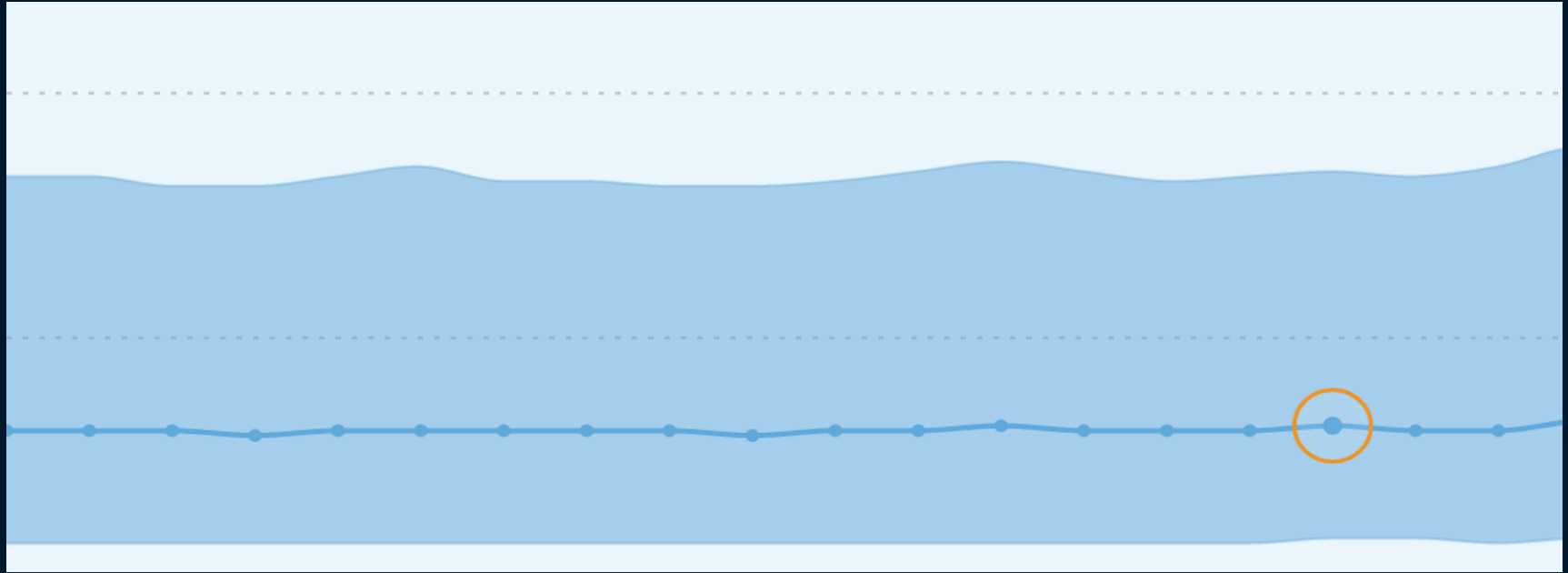
MUST FLOW



Assumptions about data



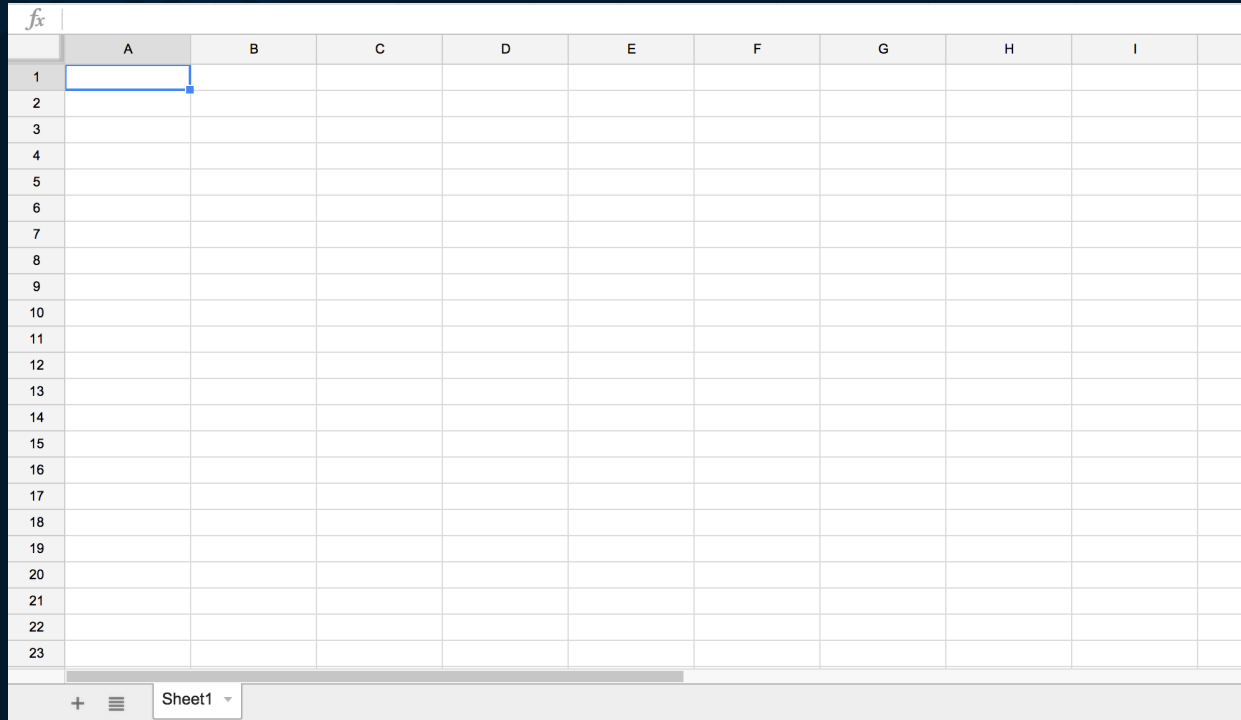
Assumptions about data



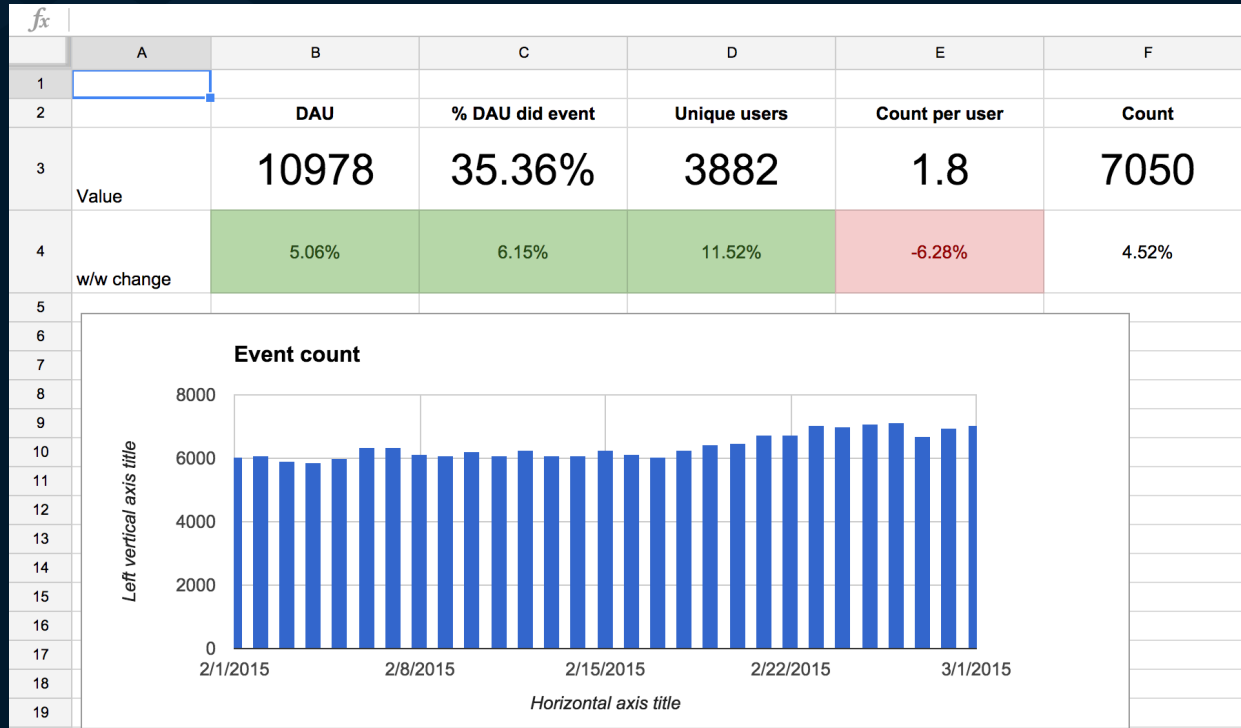
How can we make them explicit?



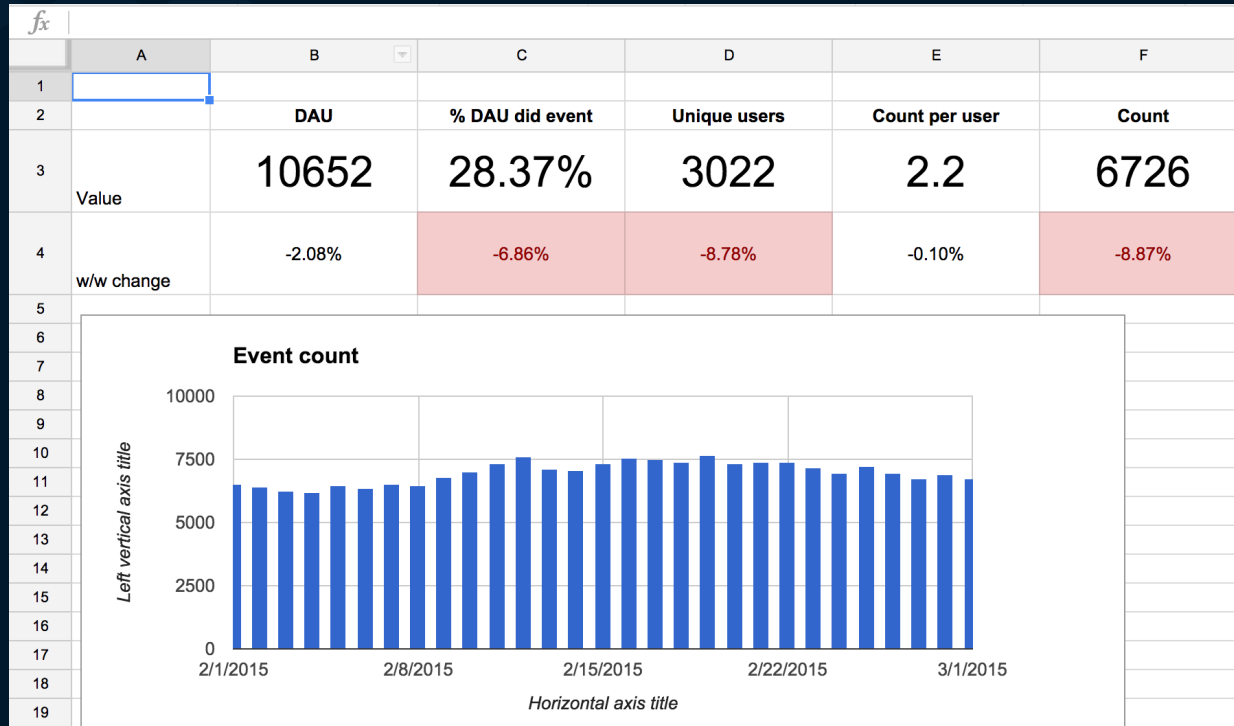
How can we make them explicit?



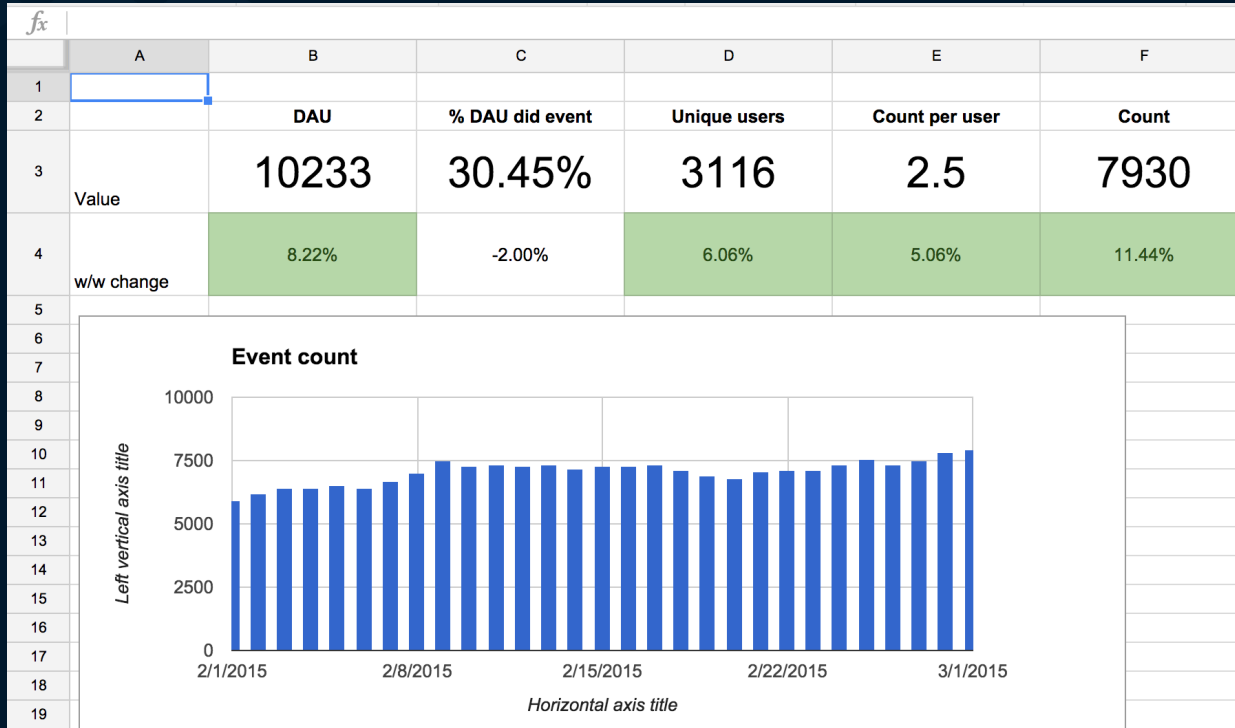
Excel prototype



Excel prototype



Excel prototype



Initial assumptions about data

Shortest path to usefulness

Real users and data change everything



Initial assumptions about data

Shortest path to usefulness

Real users and data change everything



Let's build!



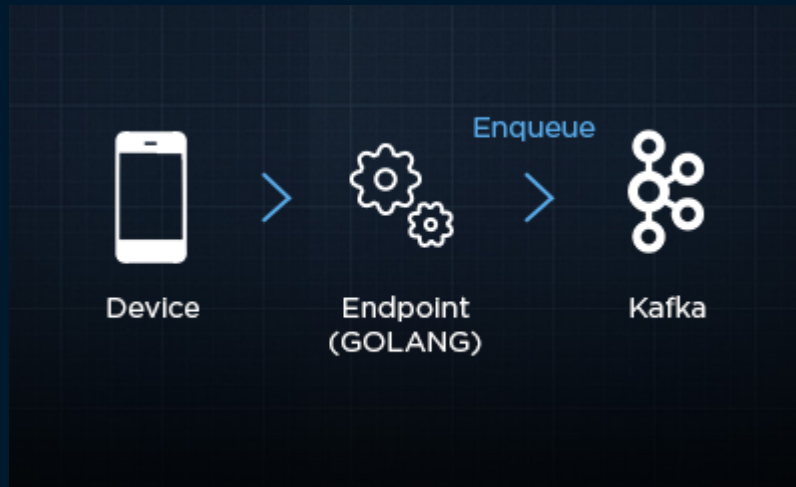
Let's build!



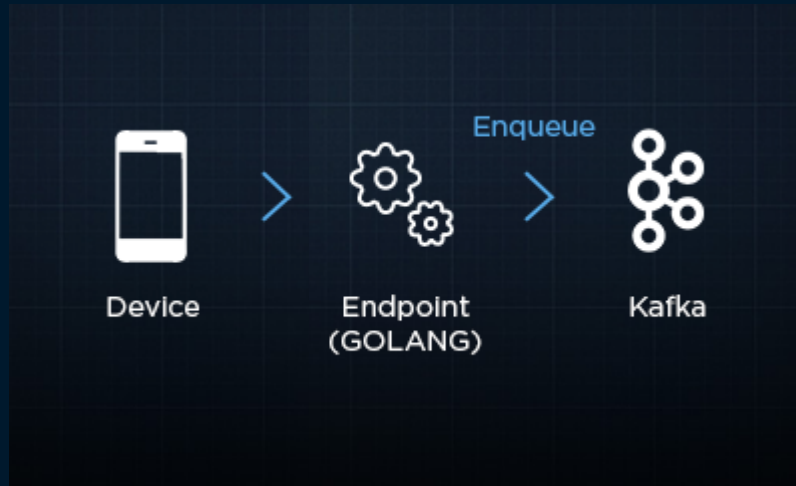
Device



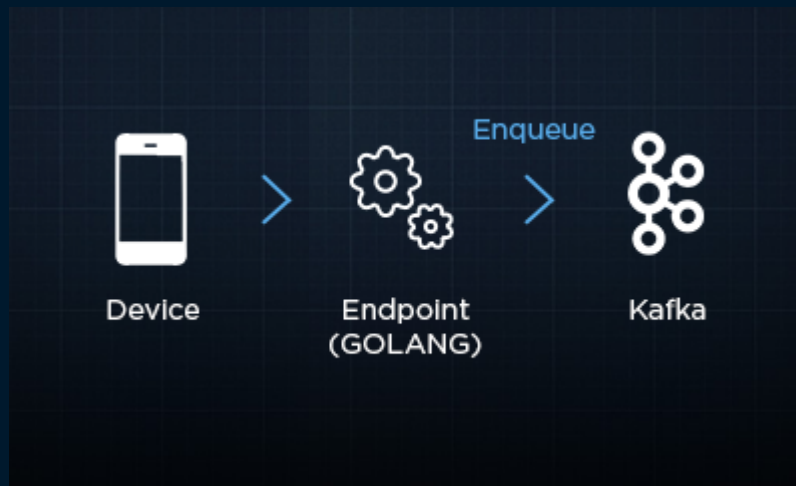
Let's build!



Real-time computation



Real-time computation



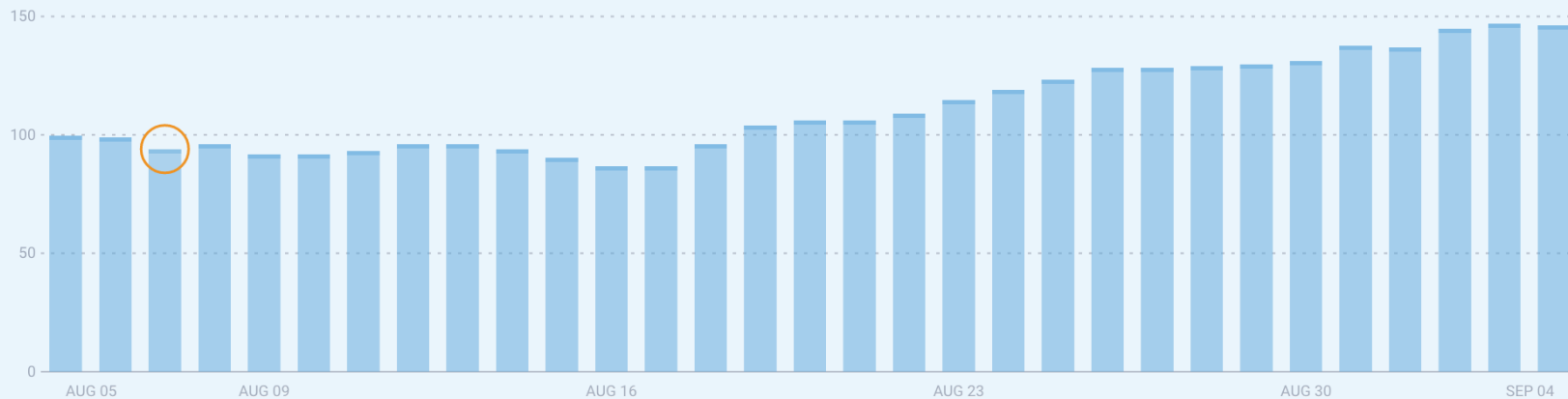
Let's build: Prototype feature



Prototype feature

103.5k ▲ 3.4%

MONTHLY ACTIVE USERS

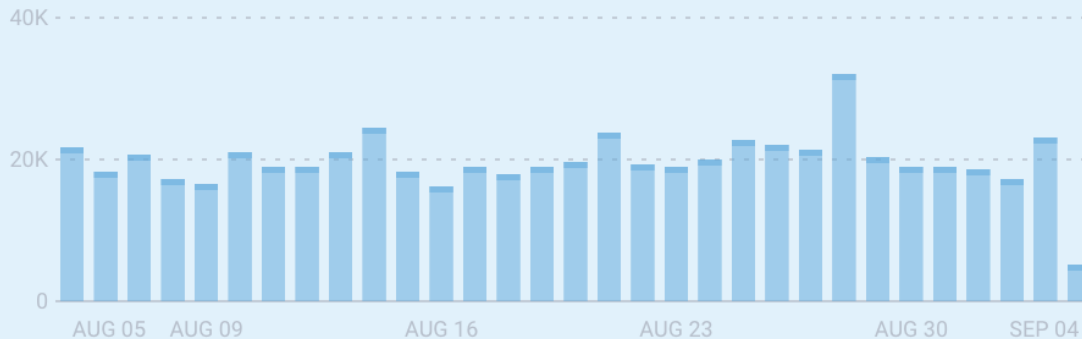


Prototype feature

4.8k ▲ 7.0%



EVENT COUNT



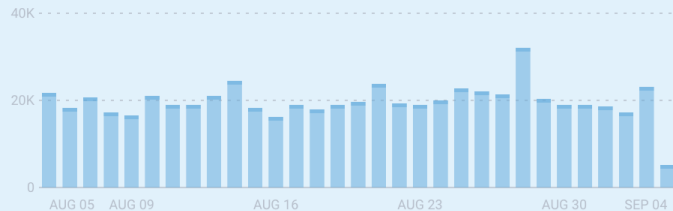
Prototype feature



Share

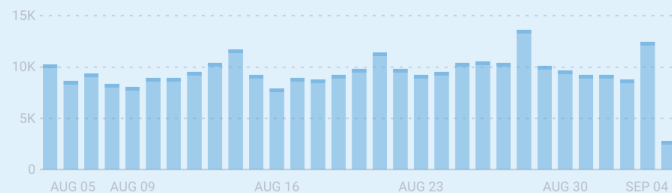
4.8k ▲ 7.0%

EVENT COUNT



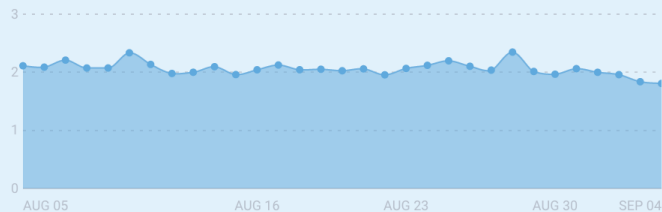
2.7k ▲ 6.7%

EVENT USERS



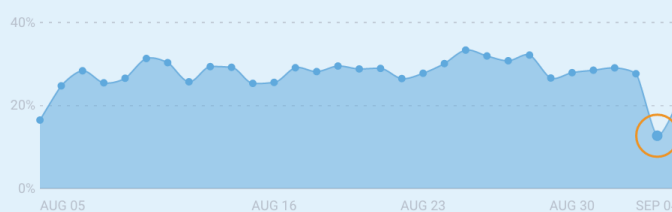
2.3

AVERAGE COUNT PER EVENT USER



31.2%

PERCENT OF DAILY ACTIVE USERS



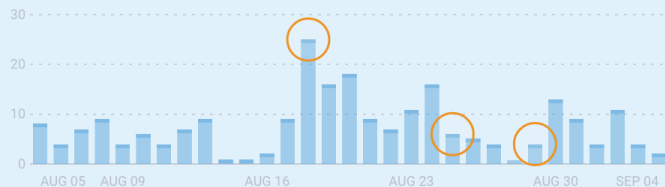
Prototype feature



Share

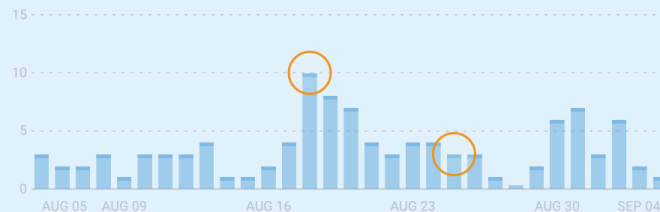
2

EVENT COUNT



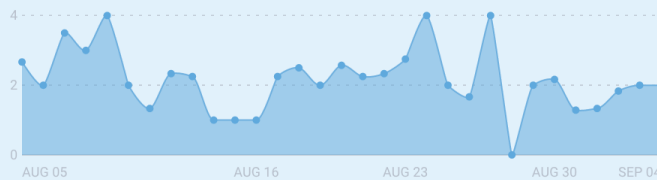
1

EVENT USERS



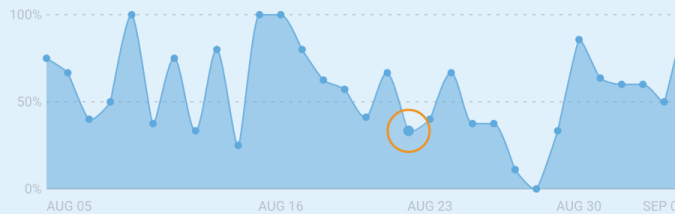
2.0

AVERAGE COUNT PER EVENT USER



100.0%

PERCENT OF DAILY ACTIVE USERS



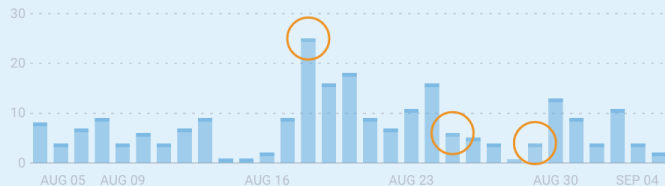
Prototype feature



Share

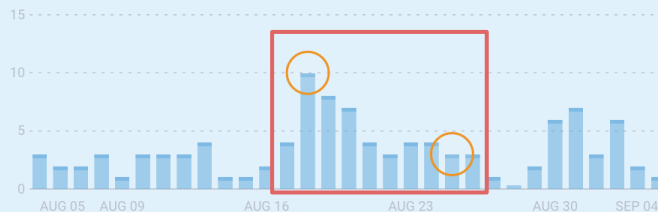
2

EVENT COUNT



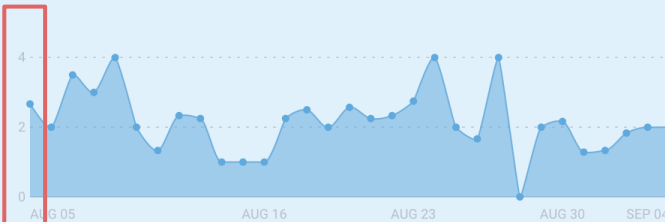
1

EVENT USERS



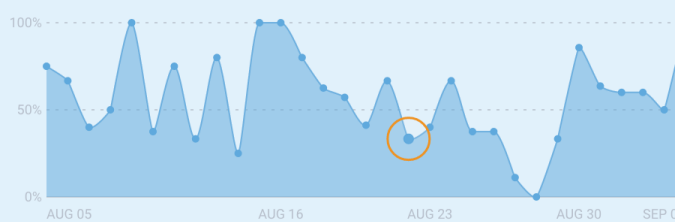
2.0

AVERAGE COUNT PER EVENT USER



100.0%

PERCENT OF DAILY ACTIVE USERS



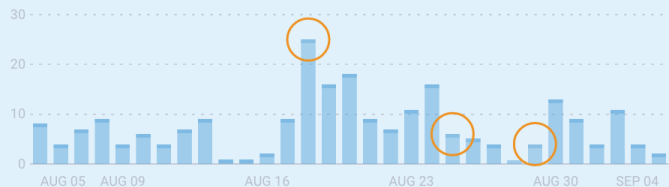
Production feature



Share

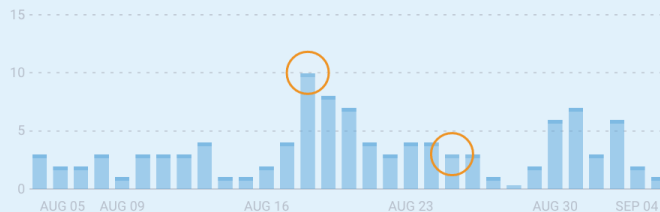
2

EVENT COUNT



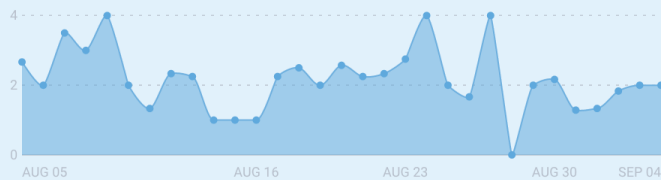
1

EVENT USERS



2.0

AVERAGE COUNT PER EVENT USER



100.0%

PERCENT OF DAILY ACTIVE USERS



More fault-tolerance



More fault-tolerance

Local Cascading jobs

Subsets or samples of real data

In-memory tests



More fault-tolerance

Local Cascading jobs

Subsets or samples of real data

In-memory tests

More data only a command line away



Ready for real users!



Initial assumptions about data

Shortest path to usefulness

Real users and data change everything



Initial assumptions about data

Shortest path to usefulness

Real users and data change everything



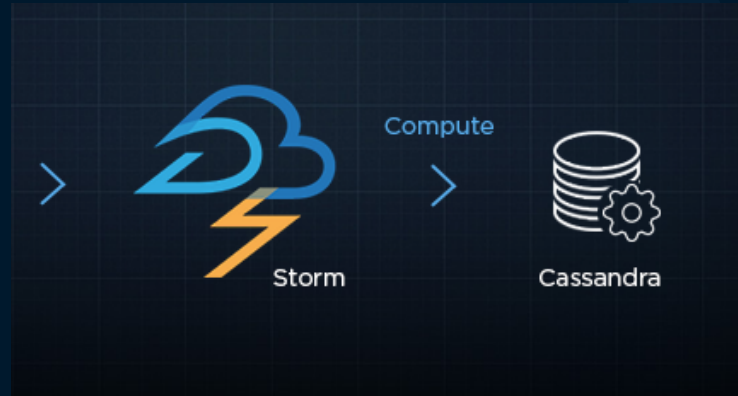
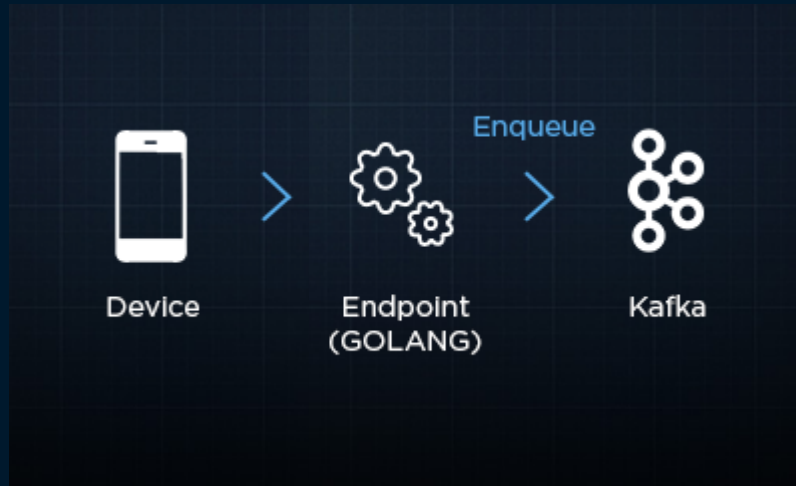
High-touch feedback



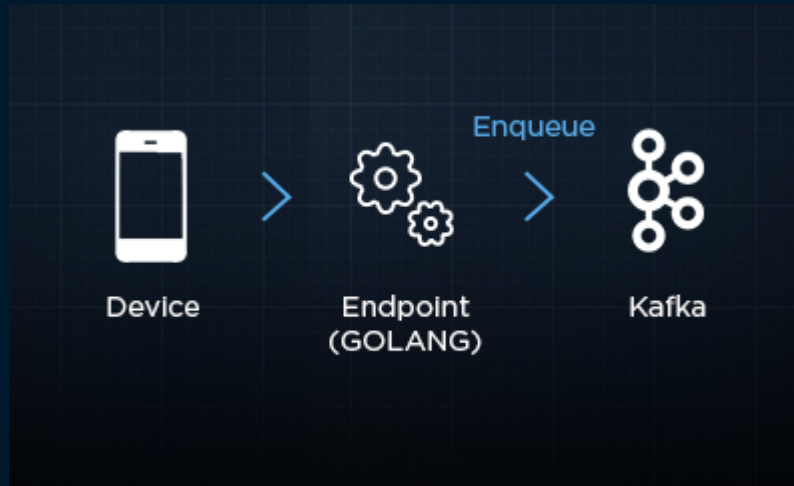
Exploring the data



Real-time prototyping



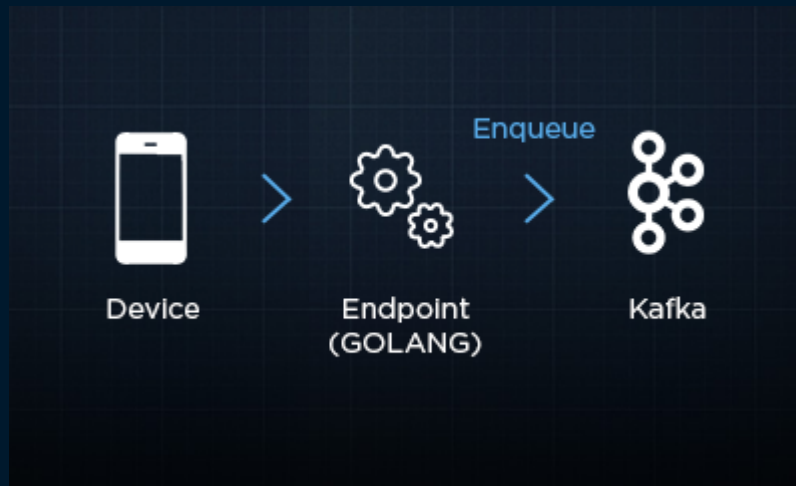
Real-time prototyping



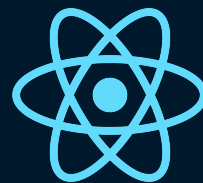
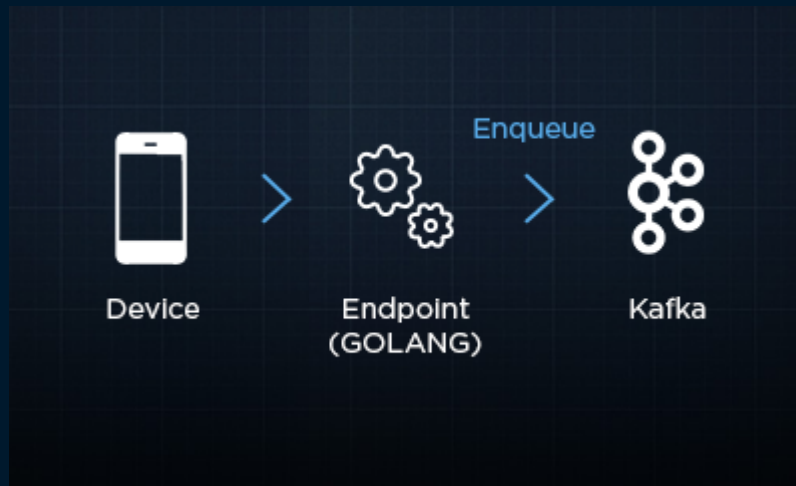
kafka logs gif



Real-time prototyping



Real-time prototyping



Real-time prototyping

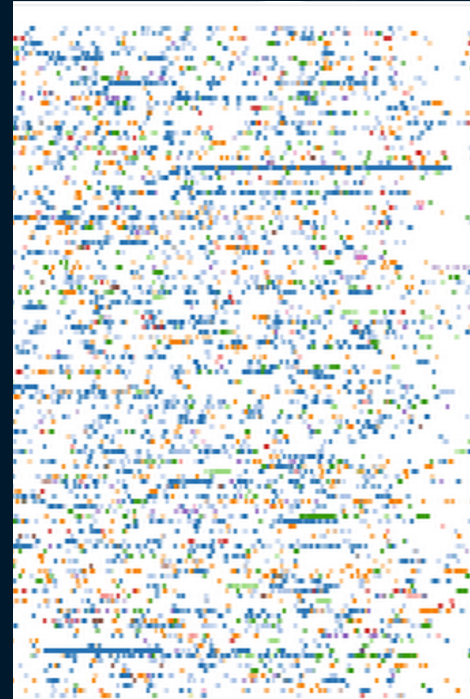
```
CanvasTimeline = React.createClass(  
  displayName: 'CanvasTimeline'  
  
  render: ->  
    dom.canvas {  
      ref: 'canvas'  
      width: @props.width  
      height: @props.height  
      onMouseMove: @whenMouseMoveInCanvas  
      onClick: @whenCanvasClicked  
    }  
  
  componentDidMount: ->  
    @ctx = @refs.canvas.getDOMNode().getContext '2d'  
    @redrawCanvas()  
  
  redrawCanvas: ->  
    {x, y, binHeight, minBarWidth, nowMs} = @props.chartParams  
  
    @ctx.clearRect 0, 0, @props.width, @props.height  
    @props.filteredData.forEach (d) =>  
      @ctx.fillStyle = @props.blockColor d  
      @ctx.fillRect [  
        (x(d.startTime) + @props.xOffset) *  
        y(d) * @props.yScale  
        Math.max(minBarWidth, x(d.endTime) -  
        binHeight * @props.yScale  
      ]...  
  
      nowX = (x(nowMs) + @props.xOffset) * @pr  
      @ctx.strokeStyle = '#ccc'  
      @ctx.beginPath()  
      @ctx.moveTo(nowX, y.range()[0])  
      @ctx.lineTo(nowX, y.range()[1])  
      @ctx.stroke()
```



Real-time prototyping

```
CanvasTimeline = React.createClass(  
  displayName: 'CanvasTimeline'  
  
  render: ->(  
    dom.canvas {  
      ref: 'canvas'  
      width: @props.width  
      height: @props.height  
      onMouseMove: @whenMouseMoveInCanvas  
      onClick: @whenCanvasClicked  
    }  
  
    componentDidMount: ->(  
      @ctx = @refs.canvas.getDOMNode().getContext '2d'  
      @redrawCanvas()  
  
      redrawCanvas: ->(  
        {x, y, binHeight, minBarWidth, nowMs} = @props.chartParams  
  
        @ctx.clearRect 0, 0, @props.width, @props.height  
        @props.filteredData.forEach (d) =>  
          @ctx.fillStyle = @props.blockColor d  
          @ctx.fillRect [  
            (x(d.startTime) + @props.xOffset) *  
              y(d) * @props.yScale  
            Math.max(minBarWidth, x(d.endTime) -  
              binHeight * @props.yScale  
          ]...  
  
          nowX = (x(nowMs) + @props.xOffset) * @pr  
          @ctx.strokeStyle = '#ccc'  
          @ctx.beginPath()  
          @ctx.moveTo(nowX, y.range()[0])  
          @ctx.lineTo(nowX, y.range()[1])  
          @ctx.stroke()  
        )  
      )  
    )  
  )  
)
```

sublime samples scale



Real-time prototyping

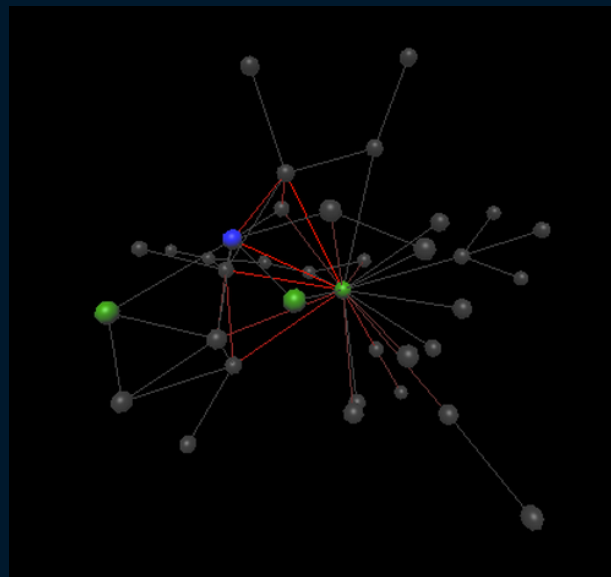
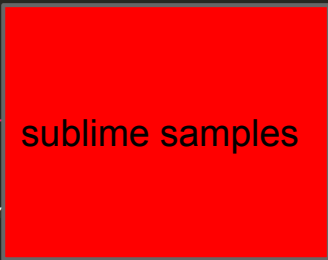
```
CanvasTimeline = React.createClass(  
  displayName: 'CanvasTimeline'  
  
  render: ->(  
    dom.canvas {  
      ref: 'canvas'  
      width: @props.width  
      height: @props.height  
      onMouseMove: @whenMouseMoveInCanvas  
      onClick: @whenCanvasClicked  
    }  
  
    componentDidMount: ->(  
      @ctx = @refs.canvas.getDOMNode().getContext '2d'  
      @redrawCanvas()  
  
      redrawCanvas: ->(  
        {x, y, binHeight, minBarWidth, nowMs} = @props.chartParams  
  
        @ctx.clearRect 0, 0, @props.width, @props.height  
        @props.filteredData.forEach (d) =>  
          @ctx.fillStyle = @props.blockColor d  
          @ctx.fillRect [  
            (x(d.startTime) + @props.xOffset) *  
              y(d) * @props.yScale  
            Math.max(minBarWidth, x(d.endTime) -  
              binHeight * @props.yScale  
            ]...  
  
            nowX = (x(nowMs) + @props.xOffset) * @pr  
            @ctx.strokeStyle = '#ccc'  
            @ctx.beginPath()  
            @ctx.moveTo(nowX, y.range()[0])  
            @ctx.lineTo(nowX, y.range()[1])  
            @ctx.stroke()
```

sublime samples scale



Real-time prototyping

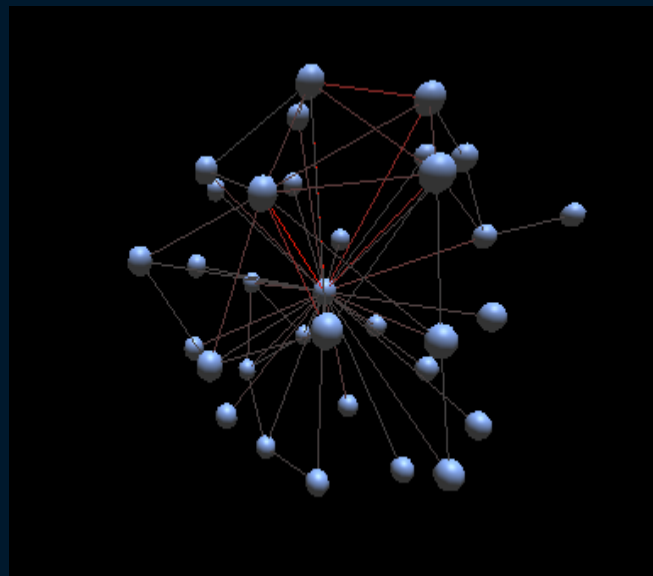
```
CanvasTimeline = React.createClass(  
  displayName: 'CanvasTimeline'  
  
  render: ->  
    dom.canvas {  
      ref: 'canvas'  
      width: @props.width  
      height: @props.height  
      onMouseMove: @whenMouseMoveInCanvas  
      onClick: @whenCanvasClicked  
    }  
  
  componentDidMount: ->  
    @ctx = @refs.canvas.getDOMNode().getContext '2d'  
    @redrawCanvas()  
  
  redrawCanvas: ->  
    {x, y, binHeight, minBarWidth, nowMs} = @props.chartParams  
  
    @ctx.clearRect 0, 0, @props.width, @props.height  
    @props.filteredData.forEach (d) =>  
      @ctx.fillStyle = @props.blockColor d  
      @ctx.fillRect [  
        (x(d.startTime) + @props.xOffset) *  
        y(d) * @props.yScale  
        Math.max(minBarWidth, x(d.endTime) -  
        binHeight * @props.yScale  
      ]...  
  
      nowX = (x(nowMs) + @props.xOffset) * @pr  
      @ctx.strokeStyle = '#ccc'  
      @ctx.beginPath()  
      @ctx.moveTo(nowX, y.range()[0])  
      @ctx.lineTo(nowX, y.range()[1])  
      @ctx.stroke()
```



Real-time prototyping

```
CanvasTimeline = React.createClass(  
  displayName: 'CanvasTimeline'  
  
  render: ->  
    dom.canvas {  
      ref: 'canvas'  
      width: @props.width  
      height: @props.height  
      onMouseMove: @whenMouseMoveInCanvas  
      onClick: @whenCanvasClicked  
    }  
  
  componentDidMount: ->  
    @ctx = @refs.canvas.getDOMNode().getContext '2d'  
    @redrawCanvas()  
  
  redrawCanvas: ->  
    {x, y, binHeight, minBarWidth, nowMs} = @props.chartParams  
  
    @ctx.clearRect 0, 0, @props.width, @props.height  
    @props.filteredData.forEach (d) =>  
      @ctx.fillStyle = @props.blockColor d  
      @ctx.fillRect [  
        (x(d.startTime) + @props.xOffset) *  
        y(d) * @props.yScale  
        Math.max(minBarWidth, x(d.endTime) -  
        binHeight * @props.yScale  
      ]...  
  
      nowX = (x(nowMs) + @props.xOffset) * @pr  
      @ctx.strokeStyle = '#ccc'  
      @ctx.beginPath()  
      @ctx.moveTo(nowX, y.range()[0])  
      @ctx.lineTo(nowX, y.range()[1])  
      @ctx.stroke()
```

sublime samples



What's the TL;DR?



Answers Events



Opinionated

The screenshot displays the 'Answers Events' configuration screen. At the top, there is a star icon and the title 'Answers Events' with the subtitle 'What would you like to track in Crashlytics?'. Below this, a vertical sidebar on the left contains various navigation icons. The main content area is organized into four columns: E-COMMERCE, CONTENT, USERS, and GAMING. Each column contains several event cards, each with an icon and a text label. The 'E-COMMERCE' column includes 'Purchase', 'Add to Cart', and 'Start Checkout'. The 'CONTENT' column includes 'Content View', 'Search', 'Share', and 'Rated Content'. The 'USERS' column includes 'Sign Up', 'Log In', and 'Invite'. The 'GAMING' column includes 'Level Start' and 'Level End'.

E-COMMERCE	CONTENT	USERS	GAMING
Purchase	Content View	Sign Up	Level Start
Add to Cart	Search	Log In	Level End
Start Checkout	Share	Invite	
	Rated Content		



Opinionated



E-COMMERCE

Purchase

Add to Cart

Start Checkout

CONTENT

Content

Search

Share

Recommend

Answers Events



2

Getting started with Answers Events

Implementing a Purchase event allows you to see your revenue in real-time, understand how many users are making purchases, how popular your items are, and track plenty of other important purchase-related metrics.

Use the following to

```
[Answers logPurchaseWithPrice:[NSDecimalNumber decimalValue:13.50]
currency:@"USD"
success:@YES
itemName:@"Answers Shirt"
itemType:@"Apparel"
itemId:@"sku-350"
attributes:@{}];
```

3

Recommended attributes

We recommend including the following attributes to get a better understanding of your users' purchase-related activity. These are all included in the example above.

itemPrice
Amount in currency

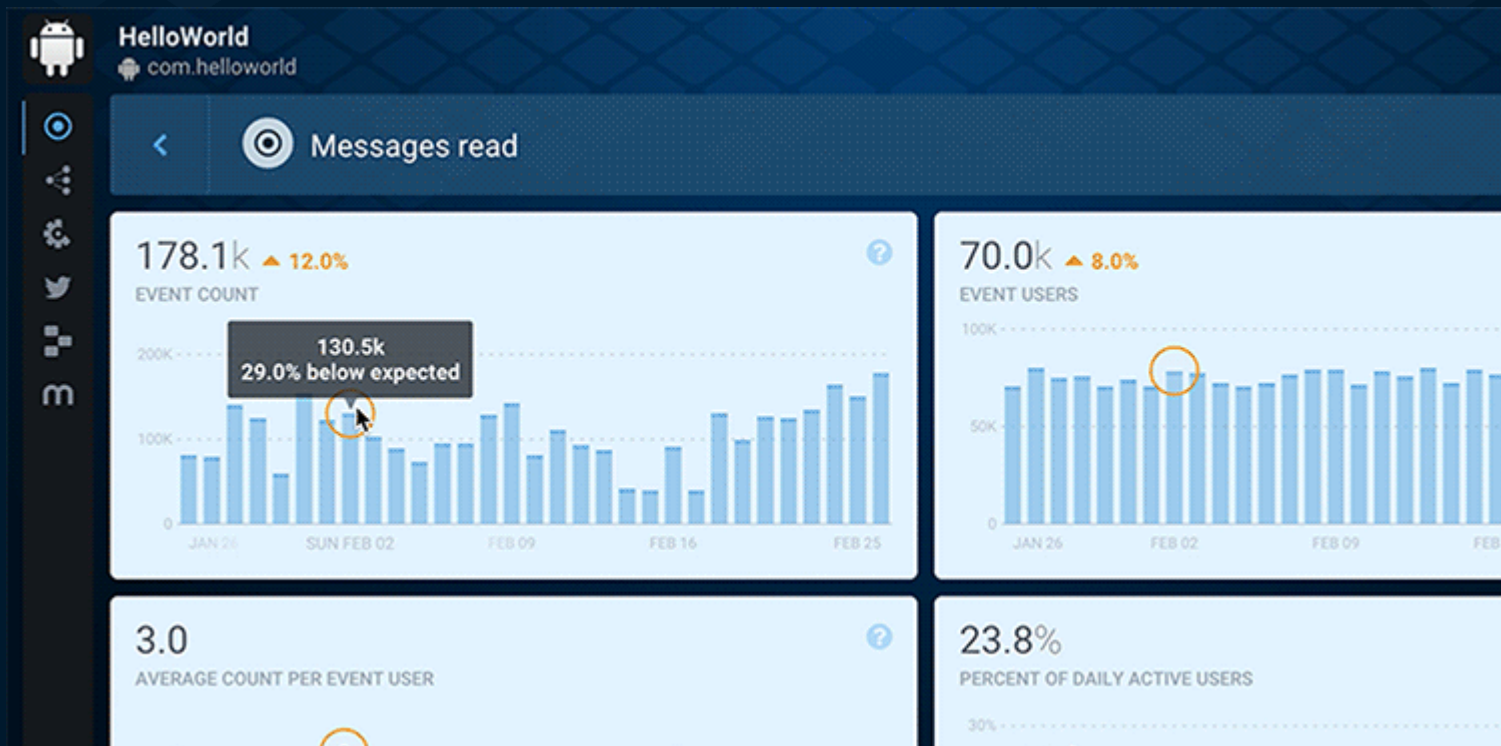
itemName
Human-readable name for the item



TL;DR



TL;DR



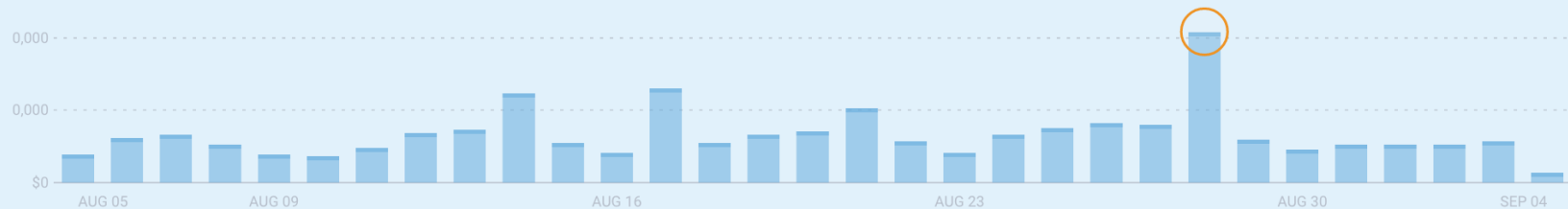
TL;DR



Purchase

\$928.71

DAILY REVENUE



INTRODUCING



answers
events

Initial assumptions about data

Shortest path to usefulness

Real users and data change everything

<https://blog.twitter.com/2015/handling-five-billion-sessions-a-day-in-real-time>



Conclusion

Lambda architecture

Opening everything enables re-use

Higher-level abstractions

Full-stack iteration



Thanks!

