BetterCloud
Powering Cloud IT

Dynamically Configured Stream Processing Using Flink & Kafka

David Hardwick
Sean Hester
David Brelloch

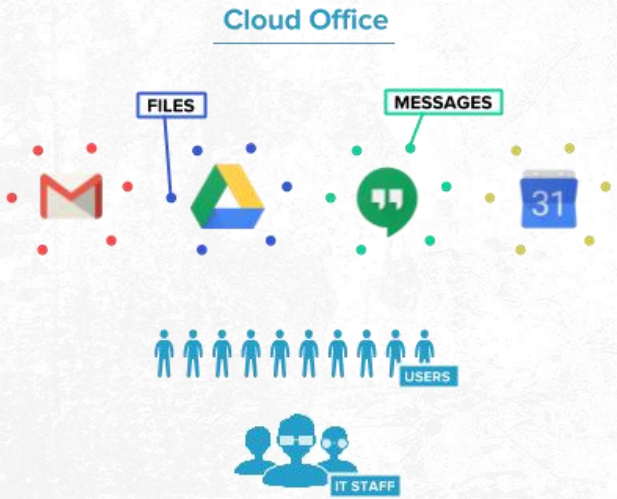
https://github.com/brelloch/FlinkForward2017

Multi-SaaS Management

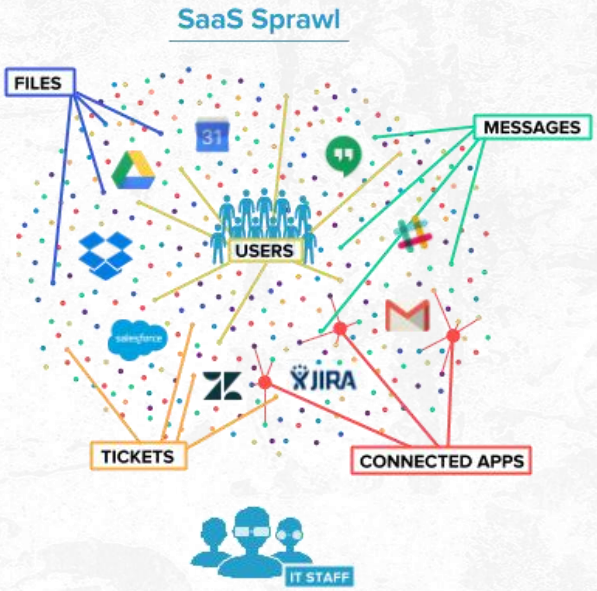
“What does that mean?”

Complex Interconnect of Apps, People and Data

A Few Years Ago

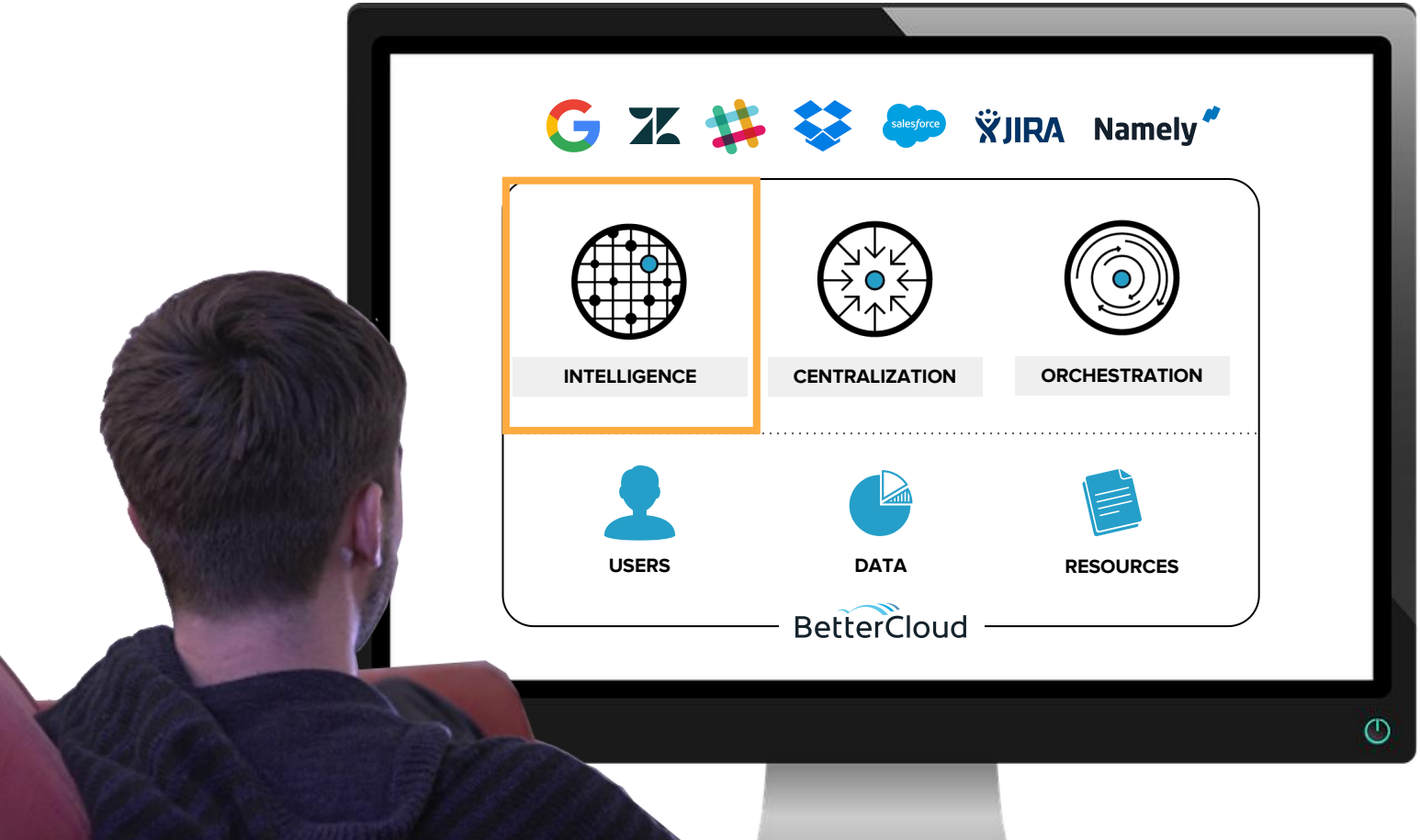


This is Now



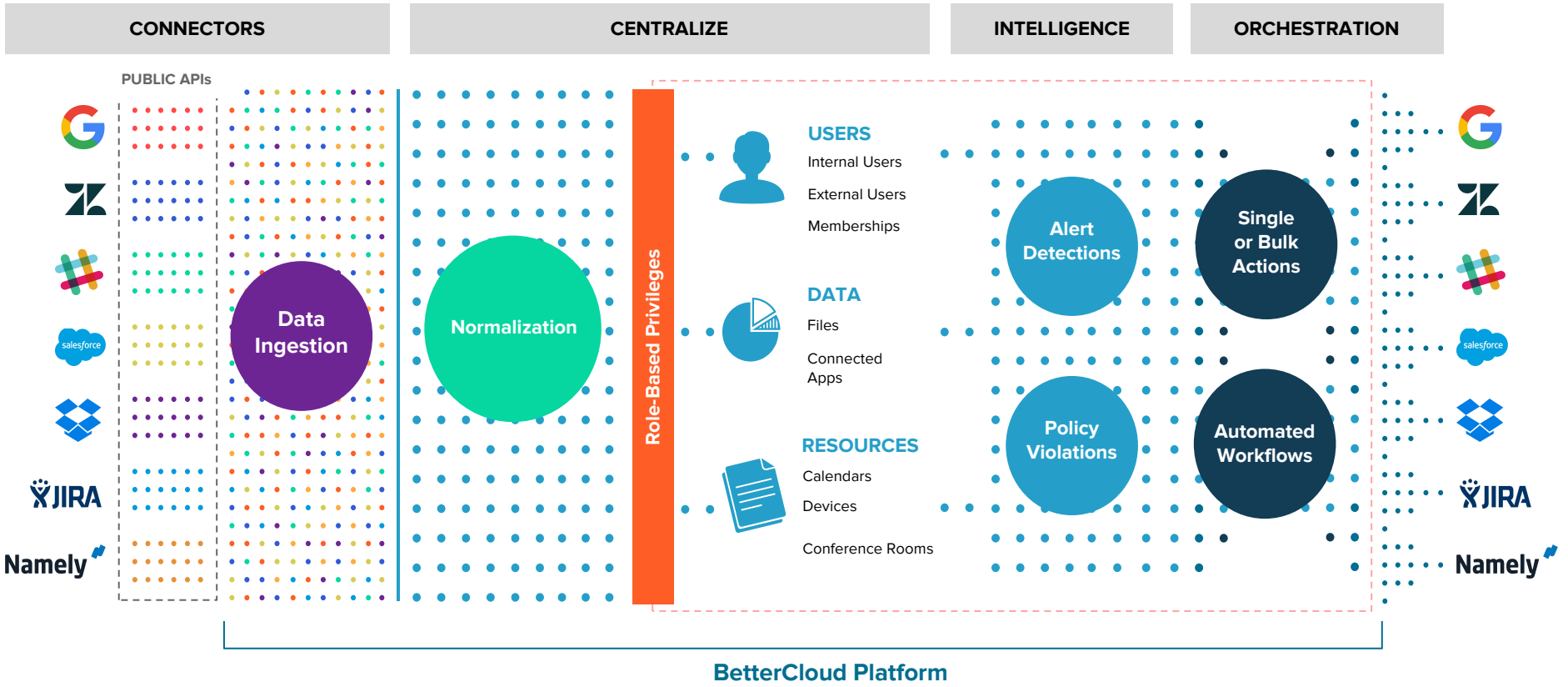
"Multi-SaaS Management" is like

Mission Control for IT Admins



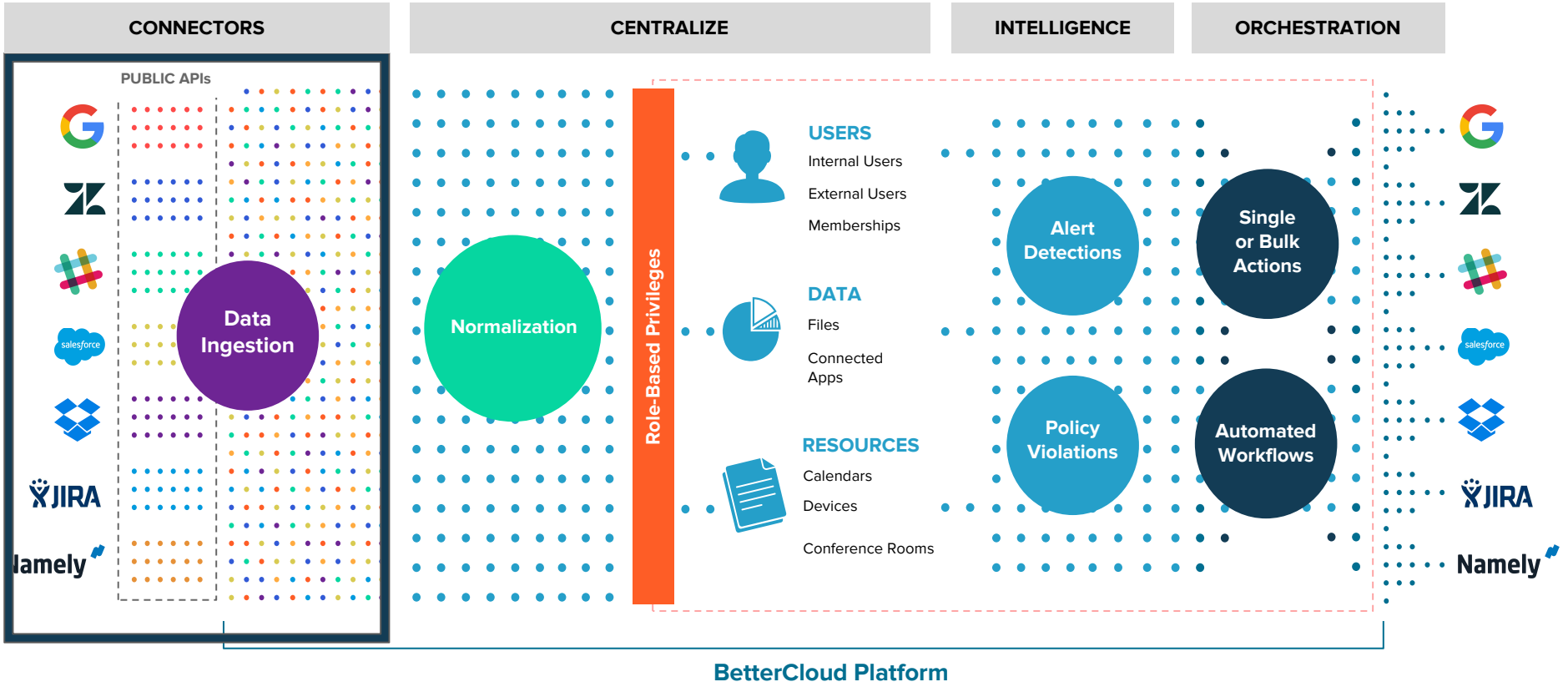
Multi-SaaS Management

We process 100s millions of messages daily



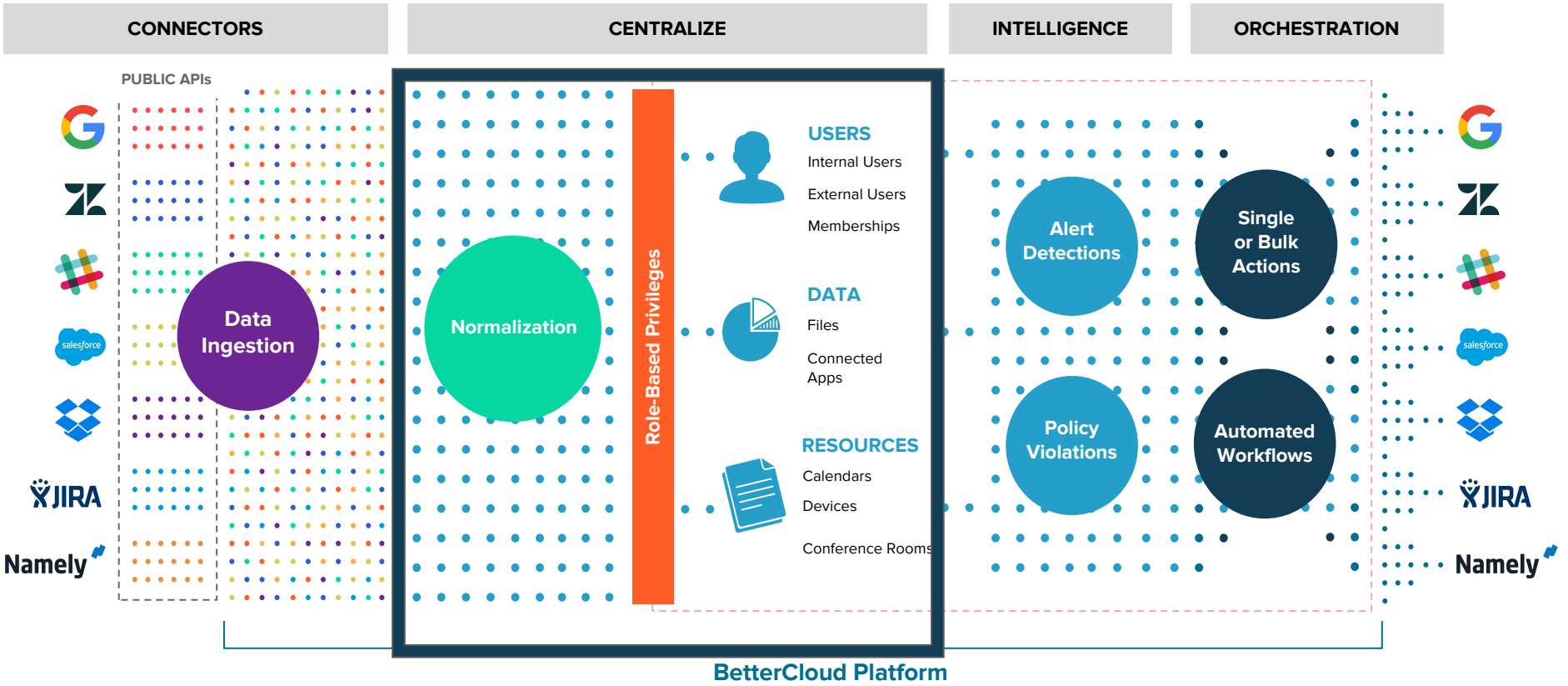
Multi-SaaS Management

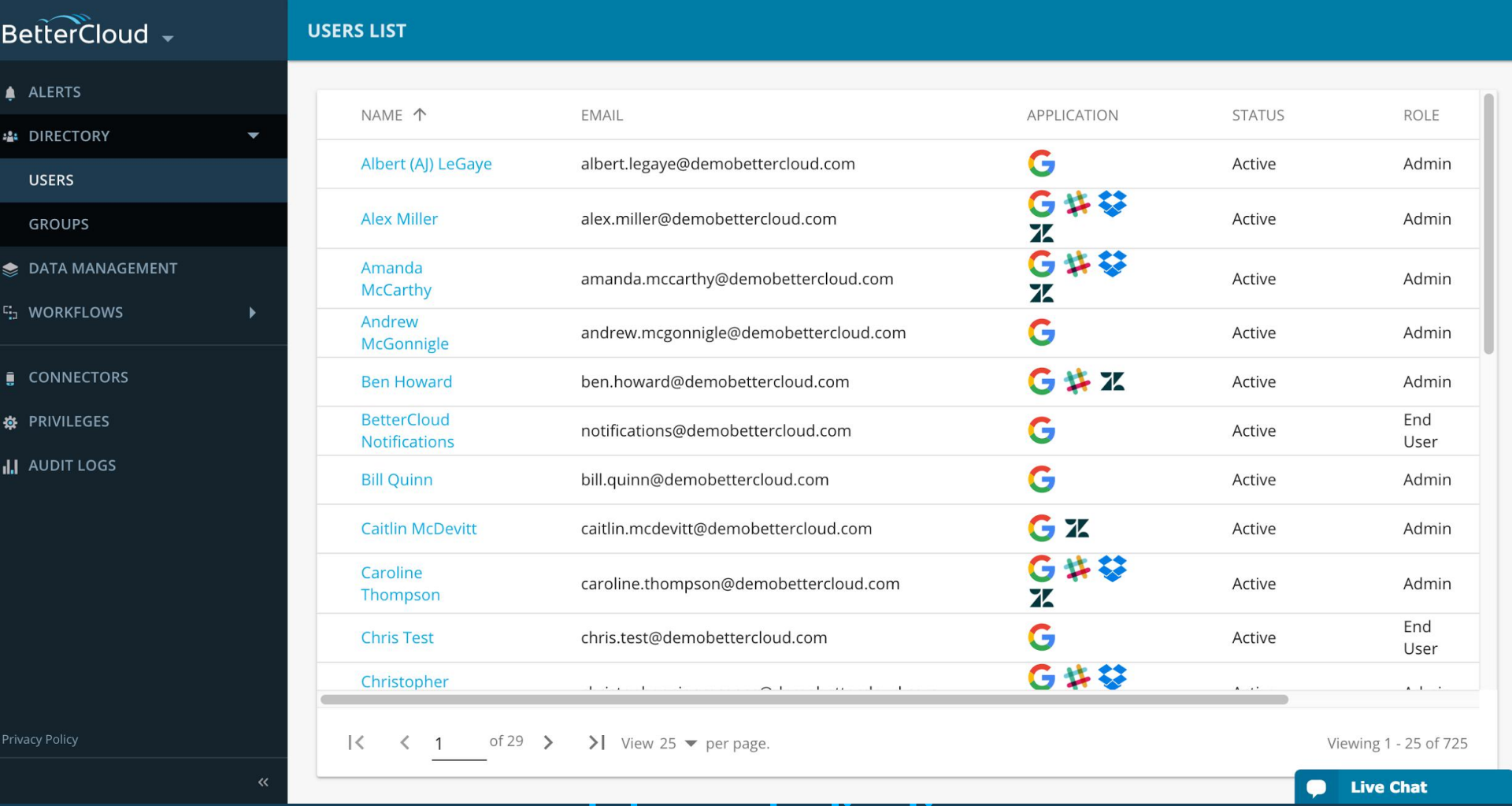
BetterCloud Platform



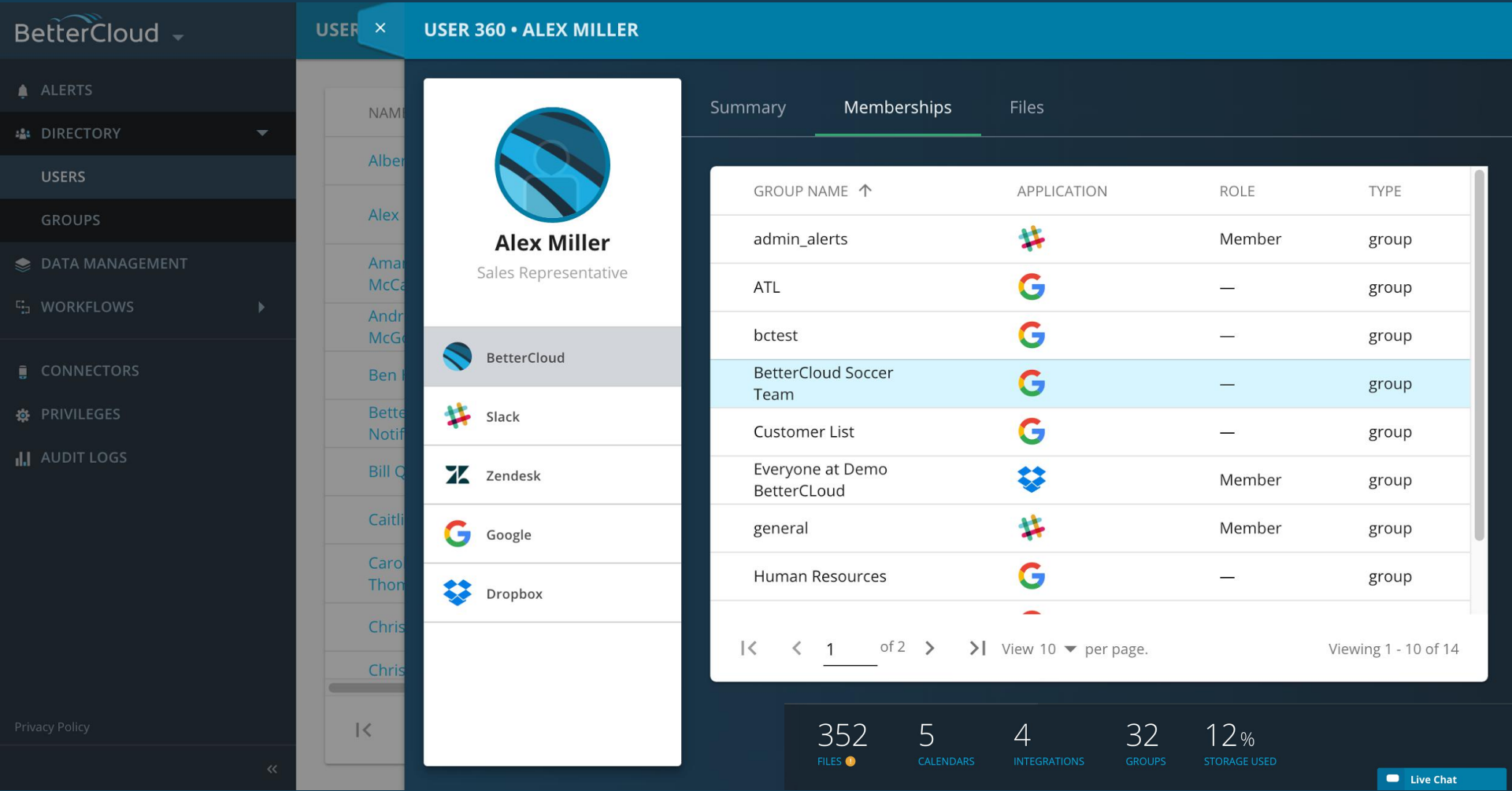
Multi-SaaS Management

BetterCloud Platform





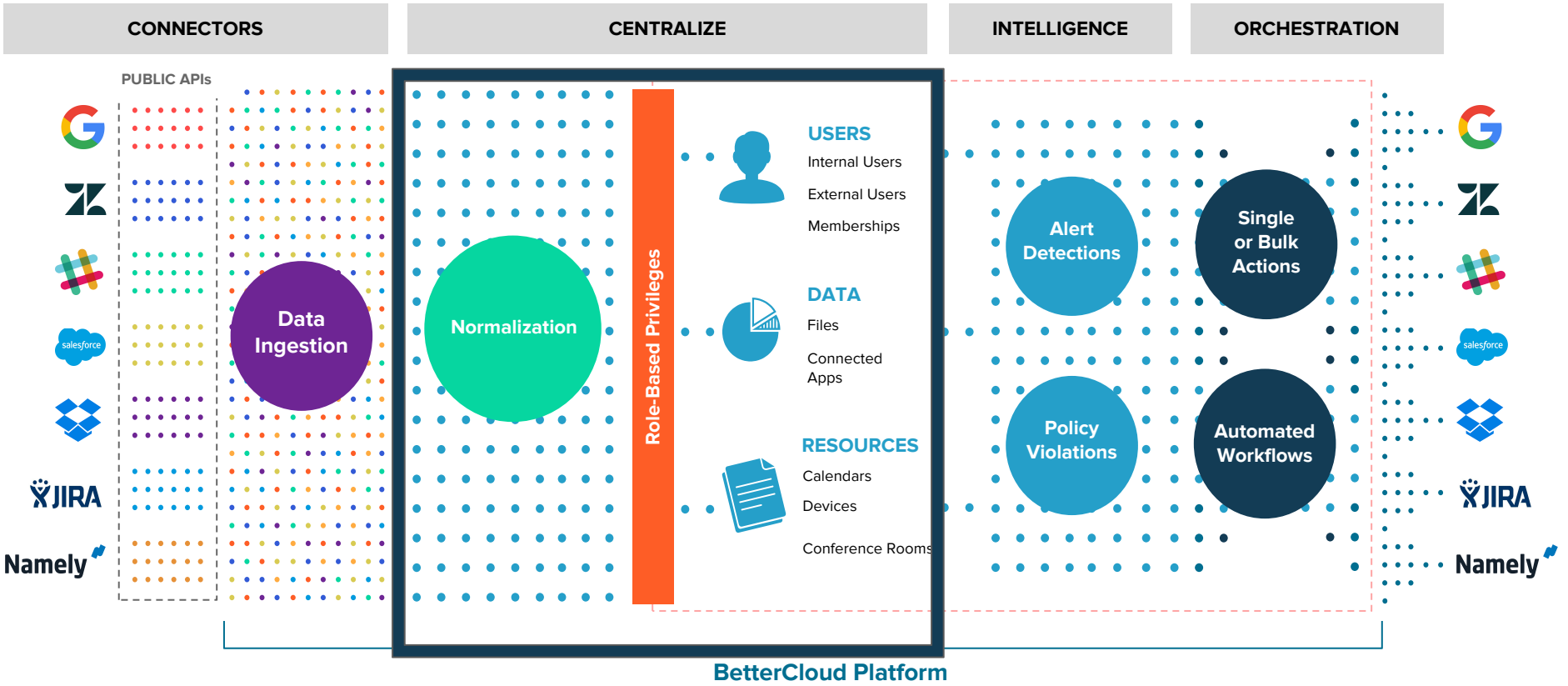
data centralization



data centralization

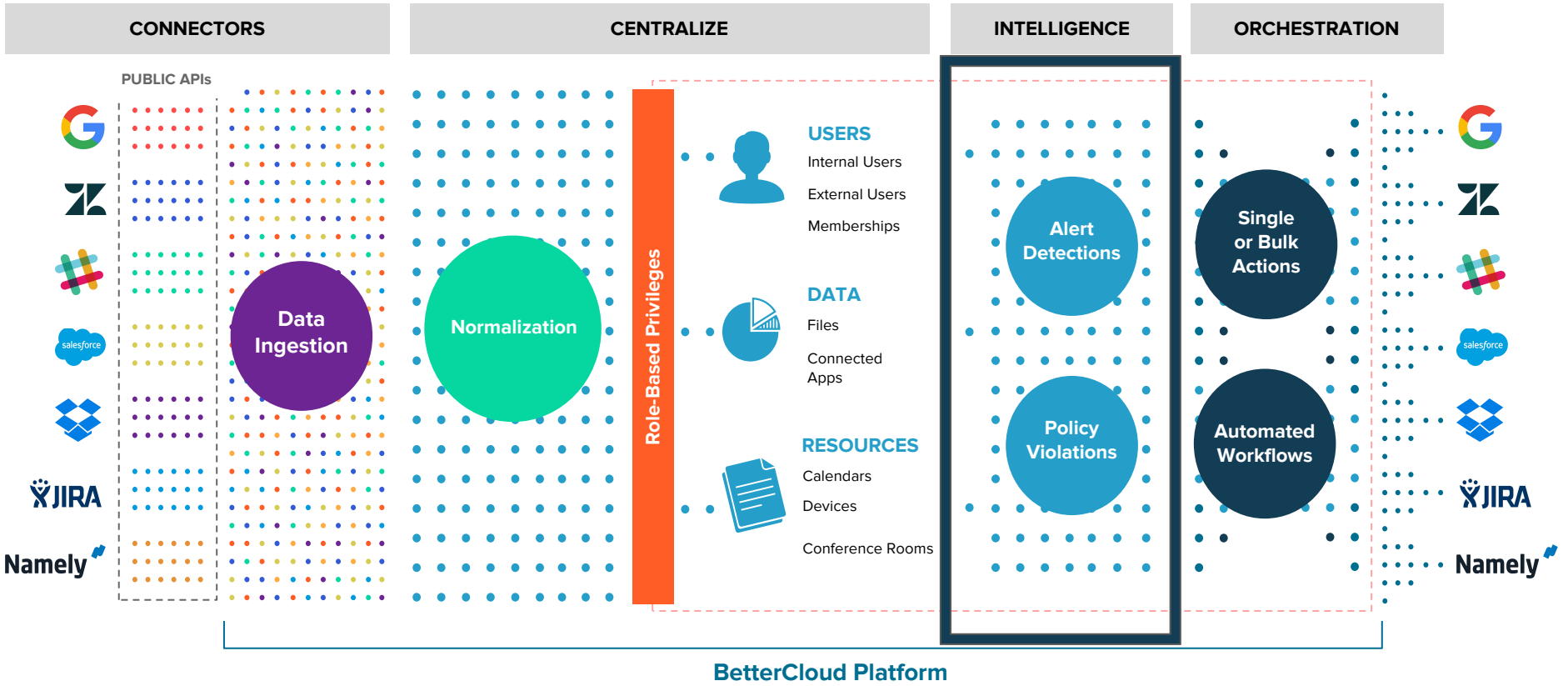
Multi-SaaS Management

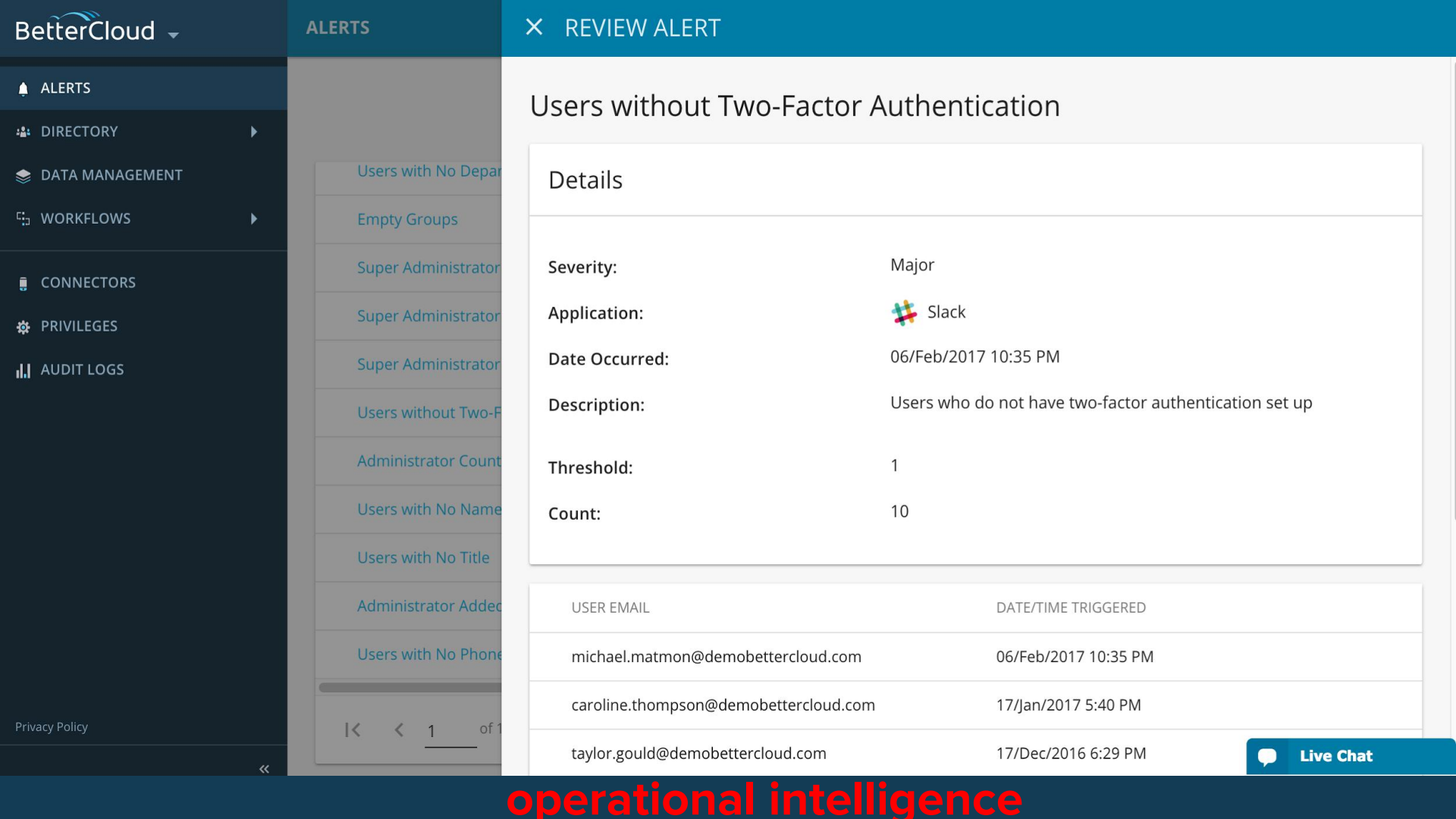
BetterCloud Platform



Multi-SaaS Management

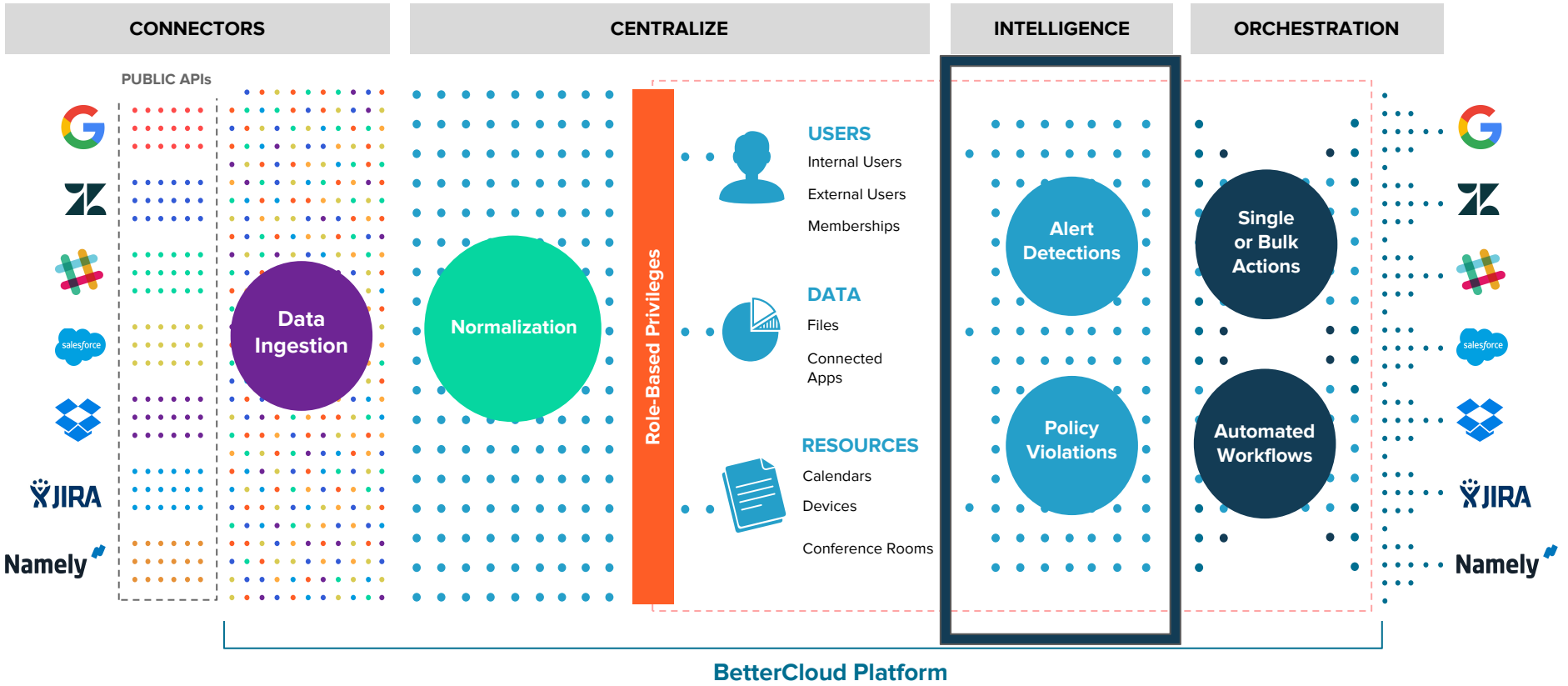
BetterCloud Platform




BetterCloud
ALERTS
DIRECTORY
DATA MANAGEMENT
WORKFLOWS
CONNECTORS
PRIVILEGES
AUDIT LOGS
Privacy Policy
ALERTS
Users with No Depar
Empty Groups
Super Administrator
Super Administrator
Super Administrator
Users without Two-F
Administrator Count
Users with No Name
Users with No Title
Administrator Addec
Users with No Phone
1
of 1
REVIEW ALERT
Users without Two-Factor Authentication
Details
Severity:
Major
Application:
Slack
Date Occurred:
06/Feb/2017 10:35 PM
Description:
Users who do not have two-factor authentication set up
Threshold:
1
Count:
10
USER EMAIL
DATE/TIME TRIGGERED
michael.matmon@demobettercloud.com
06/Feb/2017 10:35 PM
caroline.thompson@demobettercloud.com
17/Jan/2017 5:40 PM
taylor.gould@demobettercloud.com
17/Dec/2016 6:29 PM
Live Chat
operational intelligence

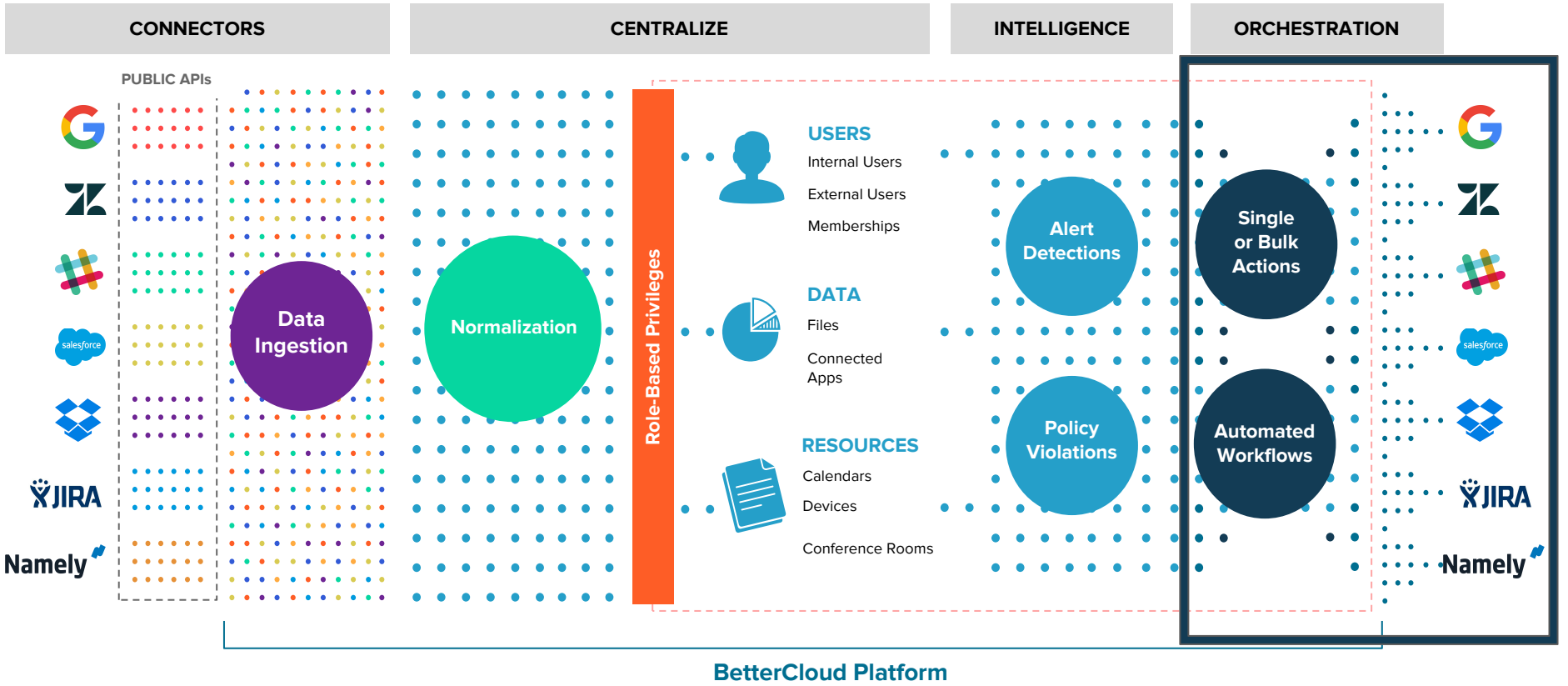
Multi-SaaS Management

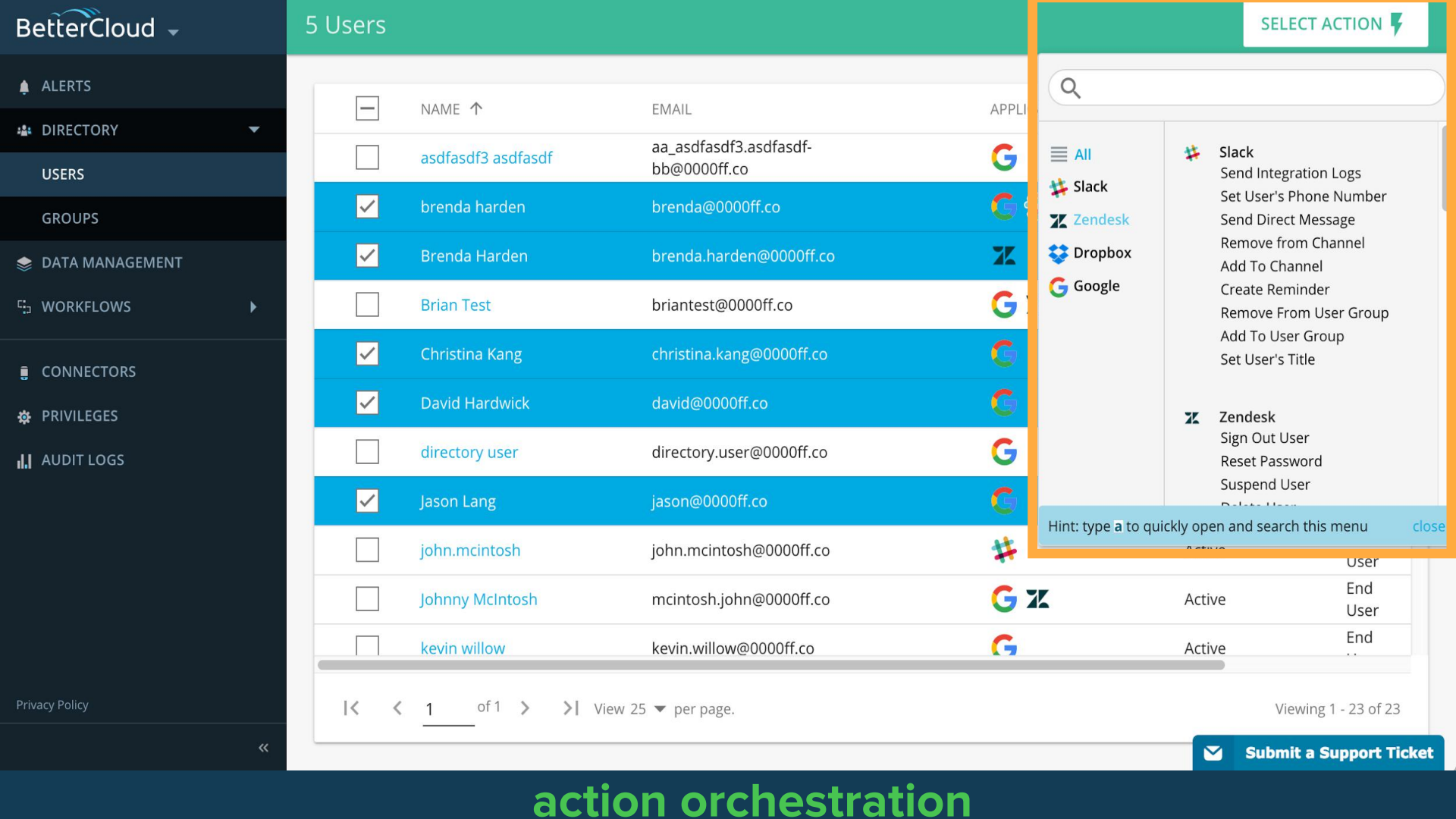
BetterCloud Platform

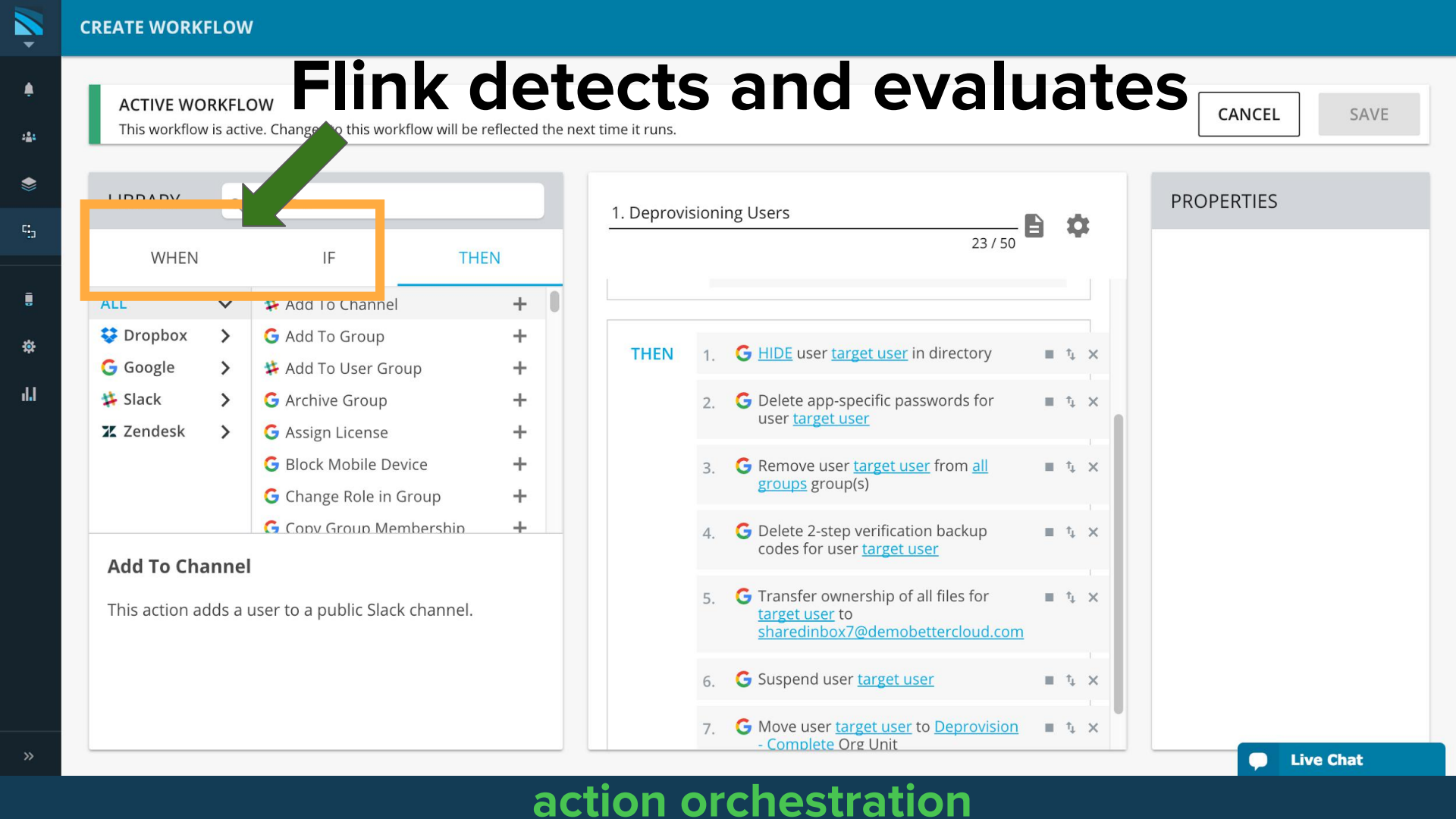


Multi-SaaS Management

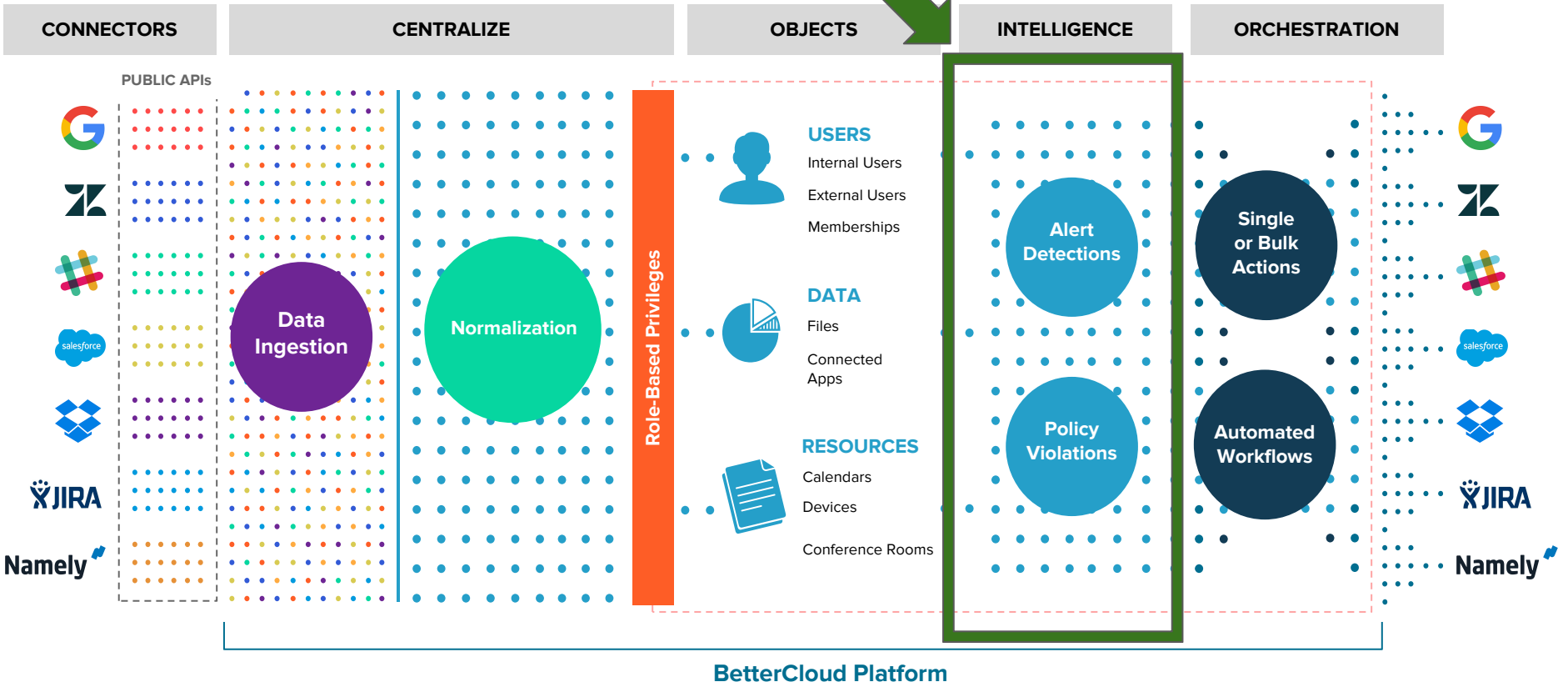
BetterCloud Platform




BetterCloud
ALERTS
DIRECTORY
USERS
GROUPS
DATA MANAGEMENT
WORKFLOWS
CONNECTORS
PRIVILEGES
AUDIT LOGS
Privacy Policy
5 Users
SELECT ACTION
NAME
EMAIL
APPLI
asdfasdf3 asdfasdf
aa_asdfasdf3.asdfasdf-bb@0000ff.co
brenda harden
brenda@0000ff.co
Brenda Harden
brenda.harden@0000ff.co
Brian Test
briantest@0000ff.co
Christina Kang
christina.kang@0000ff.co
David Hardwick
david@0000ff.co
directory user
directory.user@0000ff.co
Jason Lang
jason@0000ff.co
john.mcintosh
john.mcintosh@0000ff.co
Johnny McIntosh
mcintosh.john@0000ff.co
Active
End User
kevin willow
kevin.willow@0000ff.co
Active
End
All
Slack
Zendesk
Dropbox
Google
Slack
Send Integration Logs
Set User's Phone Number
Send Direct Message
Remove from Channel
Add To Channel
Create Reminder
Remove From User Group
Add To User Group
Set User's Title
Zendesk
Sign Out User
Reset Password
Suspend User
Hint: type a to quickly open and search this menu
close
1
of 1
View 25
per page.
Viewing 1 - 23 of 23
Submit a Support Ticket
action orchestration


CREATE WORKFLOW
Flink detects and evaluates
ACTIVE WORKFLOW
This workflow is active. Changes to this workflow will be reflected the next time it runs.
CANCEL
SAVE
WHEN
IF
THEN
ALL
Dropbox
Google
Slack
Zendesk
Add To Channel
Add To Group
Add To User Group
Archive Group
Assign License
Block Mobile Device
Change Role in Group
Add To Channel
This action adds a user to a public Slack channel.
1. Deprovisioning Users
23 / 50
THEN
1. HIDE user target user in directory
2. Delete app-specific passwords for user target user
3. Remove user target user from all groups group(s)
4. Delete 2-step verification backup codes for user target user
5. Transfer ownership of all files for target user to sharedinbox7@demobettercloud.com
6. Suspend user target user
7. Move user target user to Deprovision - Complete Org Unit
PROPERTIES
Live Chat
action orchestration

Heavy use of Flink here



Business Needs and Challenges

- Alert quickly — < 2s latency from ingest to alert
- Scale tremendously — ~100M events/day ...and growing
- Keep innovating — daily deployments to production
- Show the power — detect global alerts across all customers
- Let customers drive — custom alerts for every customer
- Replay history — “go back in time” for new alerts ...or alert bugs

“Okay, can you go deeper?”

Nope, but Sean and David can!

Foundational Prior Learnings

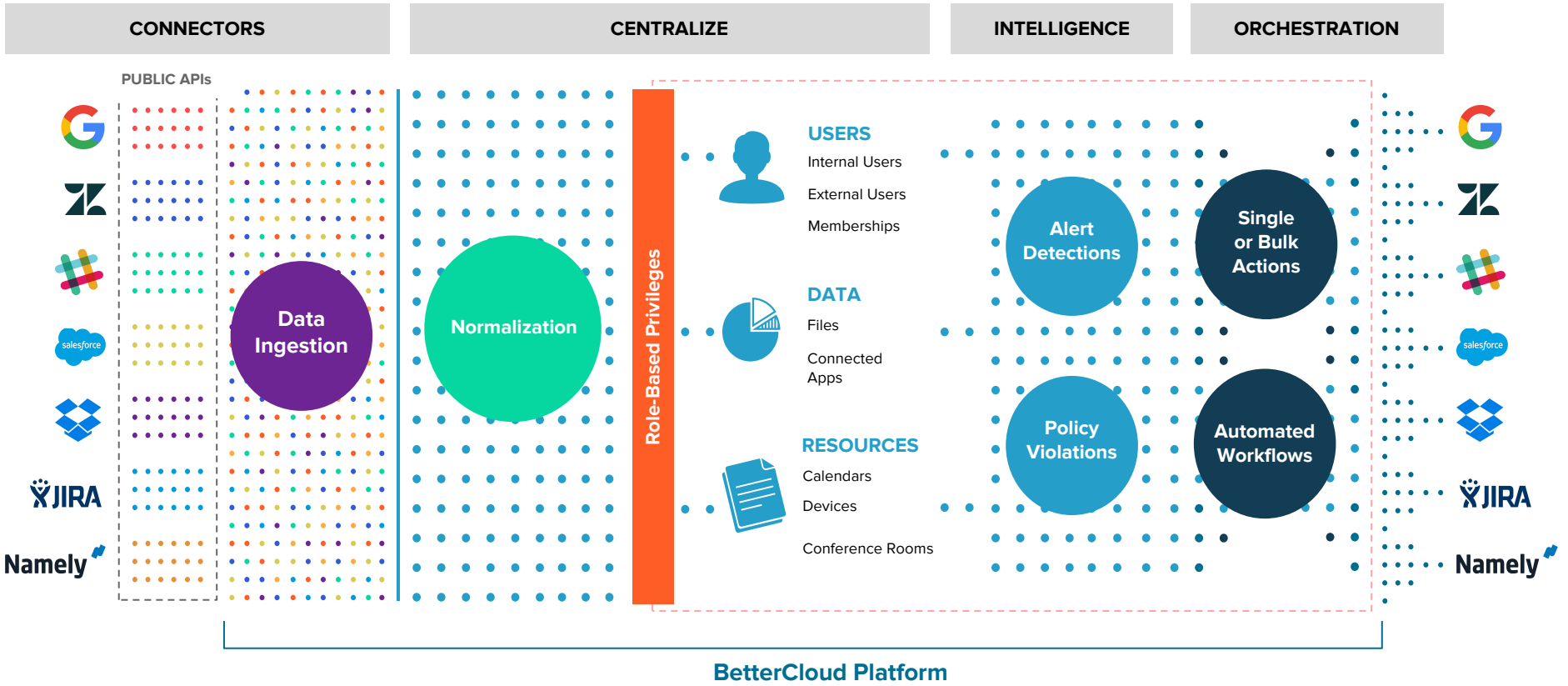
1. Event Stream Processing is a better model for Operational Intelligence than Ingest + Query

Foundational Prior Learnings

1. Event Stream Processing is a better model for Operational Intelligence than Ingest + Query
2. Rules Engines add significantly more value than hard-coded rules

Multi-SaaS Management

We process 100s of millions of messages daily



Foundational Prior Learnings

1. Event Stream Processing is a better model for Operational Intelligence than Ingest + Query
2. Rules Engines add significantly more value than hard-coded rules

JsonPath

- Jayway JsonPath: https://github.com/jayway/JsonPath
- Executes queries again JSON documents
- For us, simplifies the creation of a rules interface for our non-technical users

JsonPath Continued

```
{
    "store": {
        "book": [
            {
                "category": "reference",
                "author": "Nigel Rees",
                "title": "Sayings of the Century",
                "price": 8.95
            },
            {
                "category": "fiction",
                "author": "Evelyn Waugh",
                "title": "Sword of Honour",
                "price": 12.99
            }
        ],
        "bicycle": {
            "color": "red",
            "price": 19.95
        }
    },
    "expensive": 10
}
```

Example Paths and Output:

$.expensive => 10

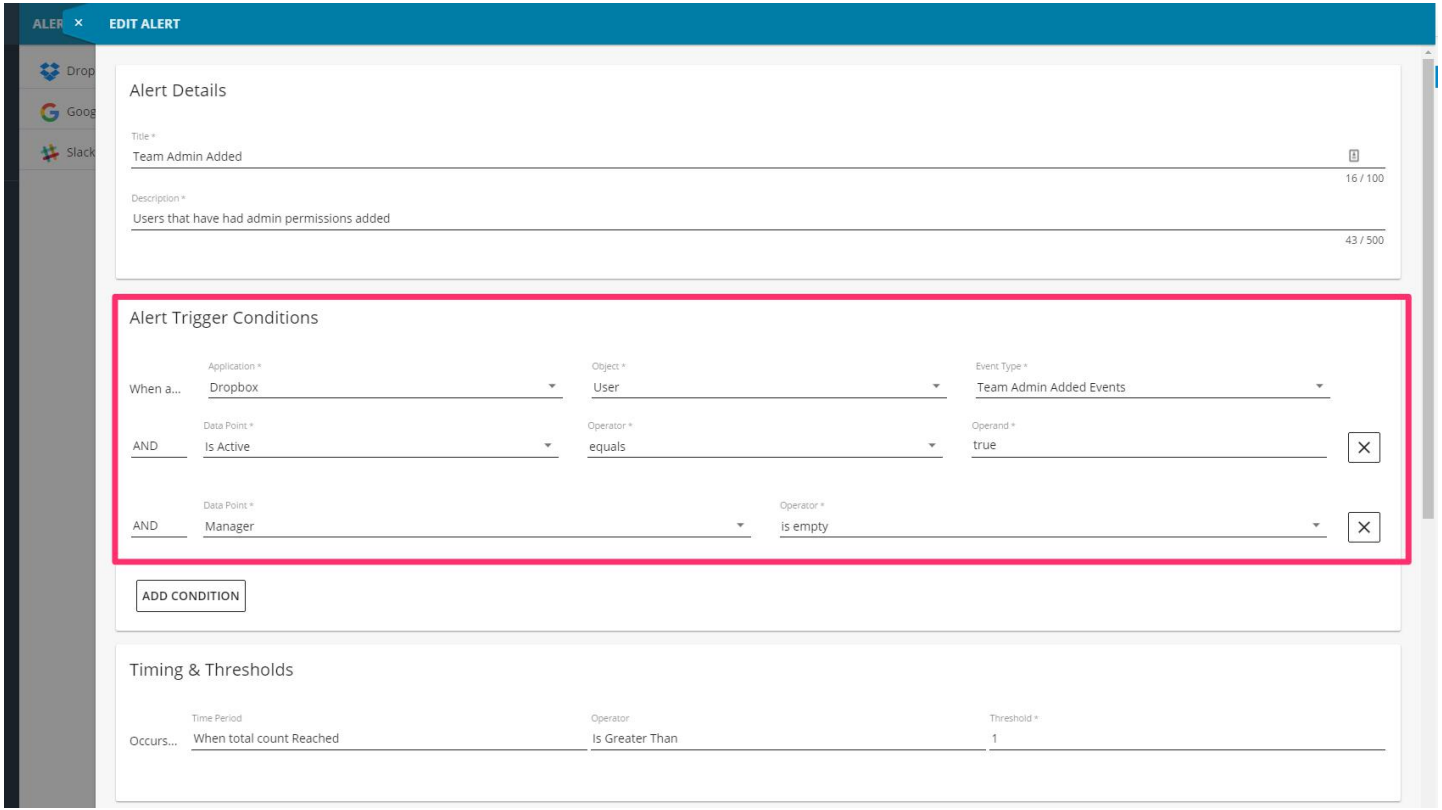
$.store.bicycle.price => 19.95

$.store.book[0].author => "Nigel Rees"

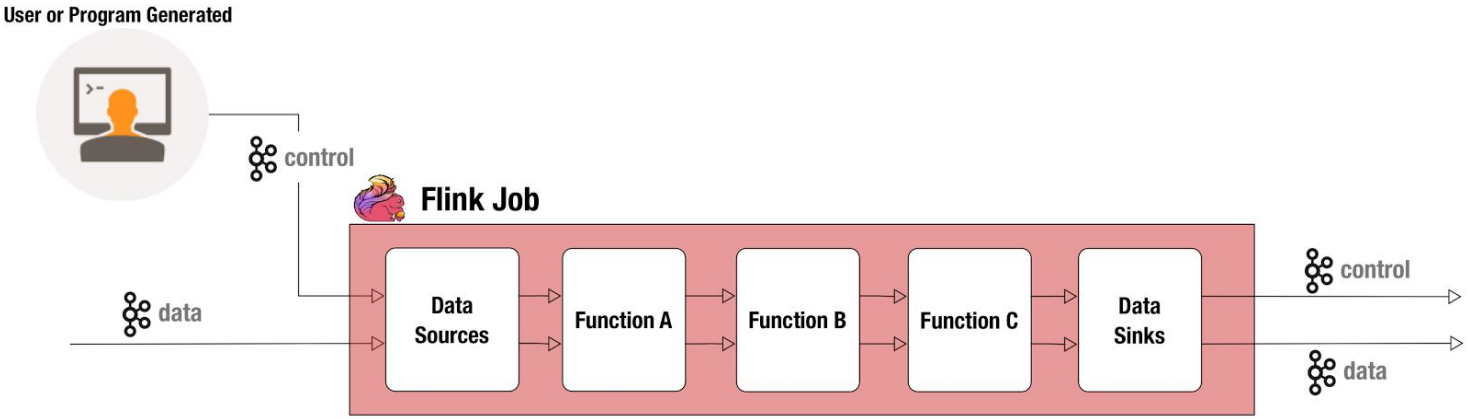
$.store.book.length > 0 => true

$.store.book[?(@.category == 'fiction')].price > 10.00 => { … }

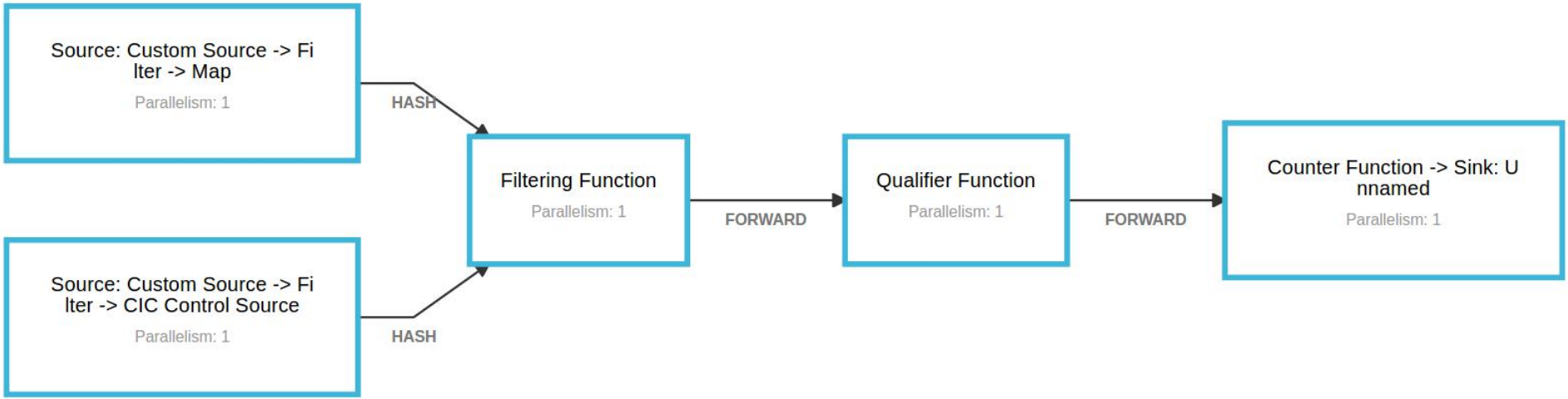
JsonPath Wrapped in a User Interface



The Control Stream



The Solution



Models

```
case class CustomerEvent(customerId: UUID, payload: String)

case class ControlEvent(customerId: UUID, alertId: UUID, alertName: String, alertDescription:
String, threshold: Int, jsonPath: String, bootstrapCustomerId: UUID)
```

Sources - Event Stream

```
val eventStream = env.addSource(new FlinkKafkaConsumer09("events", new CustomerEventSchema(),
properties))
 .filter(x => x.isDefined)
 .map(x => x.get)
 .keyBy((ce: CustomerEvent) => { ce.customerId } )
```

Sources - Control Stream

```
val controlStream = env.addSource(new FlinkKafkaConsumer09("controls", new ControlEventSchema(),
properties))
                        .filter(x => x.isDefined)
                        .map(x => x.get)
                        .name("Control Source")
                        .split((ce: ControlEvent) => {
                          ce.customerId match {
                            case Constants.GLOBAL_CUSTOMER_ID => List("global")
                            case _ => List("specific")
                          }
                        })
```

Sources - Control Stream Continued

```
val globalControlStream = controlStream.select("global").broadcast

val specificControlStream = controlStream.select("specific")
 .keyBy((ce: ControlEvent) => { ce.customerId })
```

Sources - Join the Streams

```
// Join the control and event streams
val filterStream = globalControlStream.union(specificControlStream)
 .connect(
   eventStream
 )
```

Work on Streams - Filtering

```
class FilterFunction() extends RichCoFlatMapFunction[ControlEvent, CustomerEvent, FilteredEvent] {
 var configs = new mutable.ListBuffer[ControlEvent]()
 override def flatMap1(value: ControlEvent, out: Collector[FilteredEvent]): Unit = {
   configs = configs.filter(x => (x.customerId != value.customerId) && (x.alertId != value.alertId))
   configs.append(value)
 }
 override def flatMap2(value: CustomerEvent, out: Collector[FilteredEvent]): Unit = {
   val eventConfigs = configs.filter(x => (x.customerId == x.customerId) || (x.customerId ==
Constants.GLOBAL_CUSTOMER_ID))
   if (eventConfigs.size > 0) {
     out.collect(FilteredEvent(value, eventConfigs.toList))
   }
 }
}
```

Work on Streams - Qualifying

```
class QualifierFunction extends FlatMapFunction[FilteredEvent, QualifiedEvent] {
 override def flatMap(value: FilteredEvent, out: Collector[QualifiedEvent]): Unit = {
   Try(JsonPath.parse(value.event.payload)).map(ctx => {
     value.controls.foreach(control => {
       Try {
         val result: String = ctx.read(control.jsonPath)
         if (!result.isEmpty) {
           out.collect(QualifiedEvent(value.event, control))
         }
       }
     })
   })
 }
}
```

Work on Streams - Counting

```
class CounterFunction extends FlatMapFunction[QualifiedEvent, ControlEvent] {
 var counts = scala.collection.mutable.HashMap[String, Int]()

 override def flatMap(value: QualifiedEvent, out: Collector[ControlEvent]): Unit = {
   val key = s"${value.event.customerId}${value.control.alertId}"
   if (counts.contains(key)) {
     counts.put(key, counts.get(key).get + 1)
     println(s"Count for ${key}: ${counts.get(key).get}")
   } else {
     counts.put(key, 1)
     println(s"Count for ${key}: ${counts.get(key).get}")
   }
 }
}
```

Wiring It Up

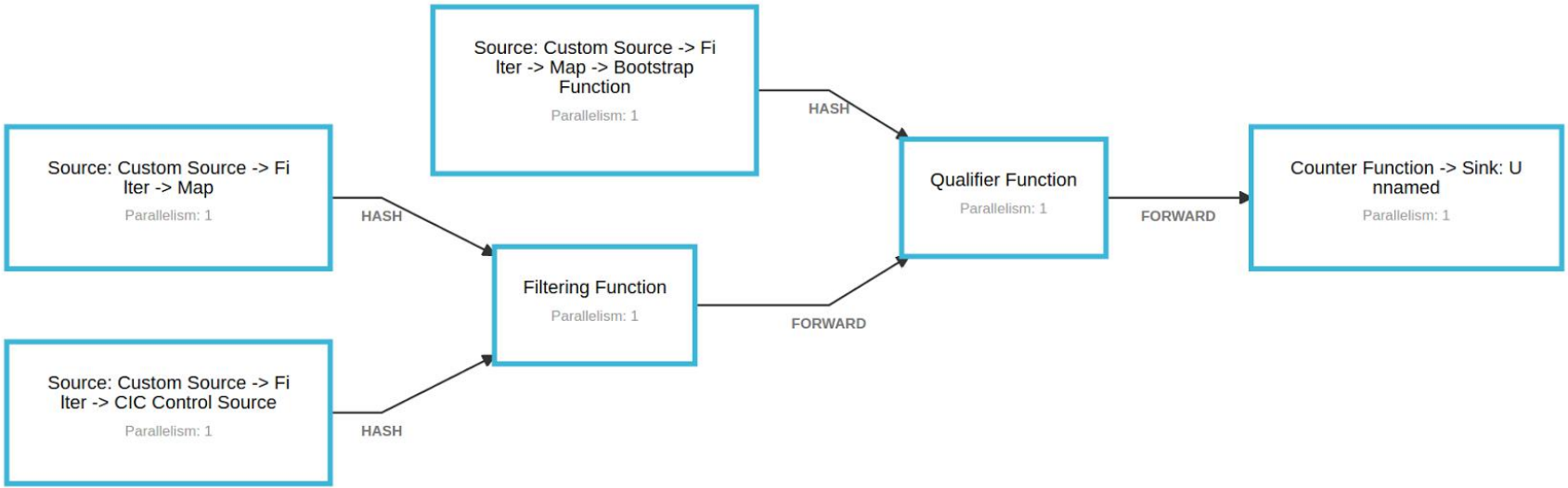
```
val filterFunction = new FilterFunction
val qualifierFunction = new QualifierFunction
val counterFunction = new CounterFunction

// Join the control and event streams
val filterStream = globalControlStream.union(specificControlStream)
 .connect(
   eventStream
 )
 .flatMap(filterFunction).name("Filtering Function")
 .flatMap(qualifierFunction).name("Qualifier Function")
 .flatMap(counterFunction).name("Counter Function")
```

What about historical data?

- The current solution works great against the live stream of data, but…

  how do you get to the current number when the events we need have already gone through the system?

Basic Implementation



Counter Function Updated

```
override def flatMap(value: QualifiedEvent, out: Collector[ControlEvent]): Unit = {
  val key = s"${value.event.customerId}${value.control.alertId}"
  if (counts.contains(key)) {
    counts.put(key, counts.get(key).get + 1)
    println(s"Count for ${key}: ${counts.get(key).get}")
  } else {
    val c = value.control
    counts.put(key, 1)
    out.collect(ControlEvent(c.customerId, c.alertId, c.alertName, c.alertDescription,
c.threshold, c.jsonPath, value.event.customerId))
    println(s"Bootstrap count for ${key}: ${counts.get(key).get}")
  }
}
```

Bootstrap Function ("Go back in time")

```
class BootstrapFunction extends FlatMapFunction[ControlEvent, FilteredEvent] {
 override def flatMap(value: ControlEvent, out: Collector[FilteredEvent]): Unit = {
   val stream = getClass.getResourceAsStream("/events.txt")

   Source.fromInputStream(stream)
     .getLines
     .toList
     .map(x => CustomerEvent(x))
     .filter(x => x.customerId == value.bootstrapCustomerId)
     .foreach(x => {
       out.collect(FilteredEvent(x, List(value)))
     })
 }
}
```

Wire up bootstrap ("Go back in time")

```
val bootstrapStream = env.addSource(new FlinkKafkaConsumer09("bootstrap", new ControlEventSchema(),
properties))
 .filter(x => x.isDefined).map(x => x.get).flatMap(bootstrapFunction).name("Bootstrap Function")
 .keyBy((fe: FilteredEvent) => { fe.event.customerId } )

val bootstrapSink = new FlinkKafkaProducer09("bootstrap", new ControlEventSchema(), properties)

val filterStream = globalControlStream.union(specificControlStream)
 .connect(eventStream)
 .flatMap(filterFunction).name("Filtering Function")
 .union(bootstrapStream)
 .flatMap(qualifierFunction).name("Qualifier Function")
 .flatMap(counterFunction).name("Counter Function")
 .addSink(bootstrapSink)
```

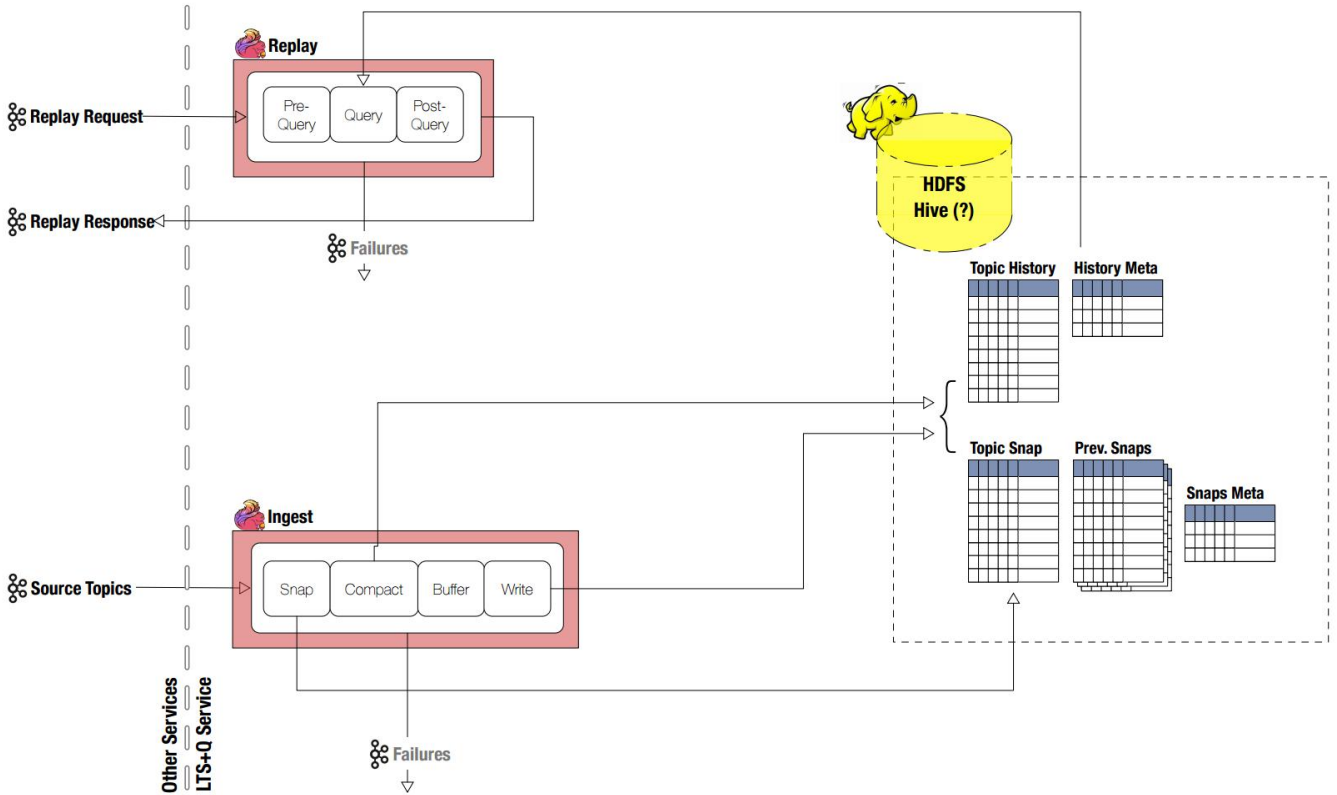
Bootstrap Function Notes ("Go back in time")

- Extremely simplified compared to what you would want in production
  - A real system requires both ingestion of all events and some kind of query system (API, database, kafka, etc…)
  - Be careful with how you implement the query system. It is easy to block the whole stream with an API or database (Async I/O!!!!).
- There should be some kind of tracking on the event which triggered to bootstrap
  - You will need to get all events up until that event from the storage system
- If ordering does not matter:
  - Can just replay all previous events as in the example code

Bootstrap Function Notes ("Go back in time")

- If ordering does matter:
  - You will need to collect and block all incoming events for that specific key until the replay of old events completes

Our Solution For Historical Data - Long Term Storage



TO THE DEMO!

BetterCloud

Further Areas for Improvement

- Most expensive operations for us are JsonPath operations. It may be worth moving towards broadcasting all alerts instead of just global and move the keyed stream to as late as possible (we are switching to this)
- State is not being saved. At a minimum the controls and counts should be saved in state
- The current models only support incrementing
- JsonPath supports different types so you can have more operators than equals (>, <, etc...)
- Only a single property path is supported

We are hiring (go figure, right?)!!!

Seriously, we are always hiring. If you are interested talk to David Hardwick or visit

https://www.bettercloud.com/careers/

Questions?

https://github.com/brelloch/FlinkForward2017