Orchestrating Data Workflows Using a Fully Serverless Architecture
About Me

- Data Engineer at Fundbox
- Technology enthusiast - especially data, ML and serverless
- Tennis (go Roger!), Football (Chelsea)
- Beer
About Fundbox

Products
Data-driven business credit solutions

Customers
>150k US-based small businesses

Founded
2013

Locations
• San Francisco: headquarters
• Tel Aviv: R&D
• Dallas: credit and support
• 225 employees and 12 dogs
THE OPPORTUNITY

Tectonic shifts in data landscape

- Expensive: ~$3,500 per underwrite
- Inaccurate: low approval rate
- Overly reliant on personal credit
Tectonic shifts in data landscape

THE OPPORTUNITY

- Expensive: ~$3,500 per underwrite
- Inaccurate: low approval rate
- Overly reliant on personal credit

☑ Scalable: ~$0 cost per underwrite
☑ Accurate: high approval rate
☑ Comprehensive view of business
Data Scientist

Data Engineer

Data Analyst

Data
Motivation
Motivation

SSH
Our goal:

Let users **think less about how their jobs are run** on our infrastructure, and instead focus on their core inquiry.
Meet BIJO
Principles

Simple
Easy to use and maintain

Monitored
Metrics collection
Logs collection

Extensible
Multiple runtime support
Runtime params
Low level API
Simple
3-step job creation

1. Create a folder in Github

User creates a job definition folder in a dedicated Github repository
3-step job creation

1. Create a folder in Github
   - Branch: master + dataops-bijo / job-definitions / integration-test
   - User creates a job definition folder in a dedicated Github repository

2. Add job files and config
   - 1. type: sql
   - 2. entrypoint: report.sql
   - 3. schedule: 0 6 * * *
   - User adds his python or sql file to the resource folder and created a config.yaml for his job
3-step job creation

1. Create a folder in Github
   - User creates a job definition folder in a dedicated Github repository
   - Example repository: `dataops-bijo / job-definitions / integration-test`

2. Add job files and config
   - User adds his python or sql file to the resource folder and created a config.yaml for his job
   - Example config.yaml:
     ```yaml
     1 type: sql
     2 entrypoint: report.sql
     3 schedule: 0 6 * * *
     ```

3. Get job notifications
   - User get notified on job events such as job started and finished
   - Example notification: "BIJO dataops/audit prod job failed"
Runtime Parameter

BIJO stores parameters in a secured key-value service called AWS Parameter Store. Parameters such as Snowflake credentials and BIJO API address.

Parameters available (additional parameter can be defined upon request):

<table>
<thead>
<tr>
<th>Parameter name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>/&lt;env&gt;/dataops/bijo/s3bucket</td>
<td>String Default S3 bucket, can be used by jobs to store/read data</td>
</tr>
<tr>
<td>/&lt;env&gt;/dataops/bijo/snowflake/account</td>
<td>Snowflake account name</td>
</tr>
<tr>
<td>/&lt;env&gt;/dataops/bijo/snowflake/password</td>
<td>Snowflake password</td>
</tr>
<tr>
<td>/&lt;env&gt;/dataops/bijo/snowflake/user</td>
<td>Snowflake user name</td>
</tr>
<tr>
<td>/&lt;env&gt;/dataops/bijo/snowflake/warehouse</td>
<td>Snowflake default warehouse</td>
</tr>
<tr>
<td>/&lt;env&gt;/dataops/bijo/api/url</td>
<td>REST API endpoint URL</td>
</tr>
</tbody>
</table>

If your job type is either `python` or `docker` you can read parameters directly:

```python
import os
import boto3

ssm = boto3.client('ssm')
env = os.environ['ENV']

snowflake_account = ssm.get_parameter(Name="/<env>/dataops/bijo/snowflake/account")
snowflake_user = ssm.get_parameter(Name="/<env>/dataops/bijo/snowflake/user")
snowflake_password = ssm.get_parameter(Name="/<env>/dataops/bijo/snowflake/password")

print(snowflake_account['Parameter']['Value'])
print(snowflake_user['Parameter']['Value'])
```

ENV environment variable is available for all jobs.
```yaml
1 type: sql
2 entrypoint: report.sql
3 schedule: 0 6 * * *
```
GitHub

1. type: sql
2. entrypoint: report.sql
3. schedule: 0 6 * * *
1 type: sql
2 entrypoint: report.sql
3 schedule: 0 6 * * *
CloudWatch Cron Schedule

GitHub

1. type: sql
2. entrypoint: report.sql
3. schedule: 0 6 * * ? *

Jenkins

AWS Fargate

ECR - Container Registry
CloudWatch Cron Schedule

```yaml
1  type: sql
2  entrypoint: report.sql
3  schedule: 0 6 * * ? *
```
CloudWatch Cron Schedule

GitHub

```yaml
1  type: sql
2  entrypoint: report.sql
3  schedule: 0 6 * * *
```

AWS Fargate

Jenkins

CloudWatch Logs

ECR - Container Registry
Python Support

```python
1  type: python
2  entrypoint: run_worker.py
3  schedule: 0 5 * * ? *
4
```

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>runtime_migrations_dir</td>
<td>add prospects to databases</td>
</tr>
<tr>
<td><strong>init</strong>.py</td>
<td>add prospects to databases</td>
</tr>
<tr>
<td>prospects_features_worker_logic.py</td>
<td>prospects_features_worker_logic.py</td>
</tr>
<tr>
<td>requirements.txt</td>
<td>upgrade mlops</td>
</tr>
<tr>
<td>run_worker.py</td>
<td>upgrade mlops</td>
</tr>
</tbody>
</table>
Python Support

1  type: python
2  entrypoint: run_worker.py
3  schedule: 0 5 * * ? *
Python Support

1. `type: python`
2. `entrypoint: run_worker.py`
3. `schedule: 0 5 * * ? *`

```
1. `type: python`
2. `entrypoint: main.py`
3. `schedule: 0 3 * * ? *`
4. `container-memory: 16384`
5. `container-cpu: 4096`
```
Monitored
Monitoring jobs

BIJO uses AWS StepFunctions to manage job state and catch failures.
CloudWatch Cron Schedule

1. type: sql
2. entrypoint: report.sql
3. schedule: 0 6 ** ? *
SNS – Notification Service

A message

SNS Topic

Email Address

SMS
SNS – Notification Service

A message

SNS Topic

Email Address

SMS

Lambda Function
Monitoring

CloudWatch Cron Schedule

Slack Lambda

#bijo_notifications

http post

Job A has finished

Wednesday, September 18th

bijo APP 10:39 AM
BIJO dataops/audit prod job failed
Step function name: re-run-of-2019.09.17.21.07.59.553543-S3
Manager service info: [{'manager_service_request_id': 'dd014c80-c189-405e-aa8c-a6e7f37264a1', 'manager_service_log_stream_name': '2019/09/17/[$LATEST]874f78990153436cb208bbdc2aa9d043', 'origin': 'S3', 'origin_request_id': '05104ef7-be74-4406-8509-7726fde2609d'}]
@here

CloudWatch Logs

docker

Pull docker image

ECR - Container Registry

Fundbox
Adi Levhar 11:11 AM
we need dependencies between jobs

Tomer Levi 11:11 AM
shit 😓

Message Adi Levhar
StepFunctions 🔄 → SNS Topic 📥 → Sns-to-Slack Lambda 🐍 → #bijo_notifications

AWS Fargate 🐒 → CloudWatch Logs 📊

Job dependency service
Job dependency service

- StepFunctions
- SNS Topic
- Sns-to- Slack Lambda
- #bijoNotifications
- CloudWatch Logs
- AWS Fargate
- Job-Dependency Lambda
- DynamoDB table
- http post
Job dependency service

- StepFunctions
- SNS Topic
- Sns-to- Slack Lambda
- #bijo_notifications
- CloudWatch Logs
- AWS Fargate
- Job-Dependency Lambda
- DynamoDB table

http post
Dependency

defends-on: analysts/simple-analysts-job-example
1 type: python
2 entrypoint: entrypoint.py
3 depends-on: dataops/s3-job OR dataops/on-demand-job
type: python
entrypoint: entrypoint.py
depends-on: dataops/s3-job OR dataops/on-demand-job

type: python
entrypoint: entrypoint.py
depends-on:
type: s3
bucket: default_bucket
regex: "IPV[\S]*.csv"
Job dependency service

- S3: New Object creation
- StepFunctions
- CloudWatch Logs
- AWS Fargate
- SNS Topic
- Sns-to-slack Lambda
- http post
- #bijo_notifications
- Job-Dependency Lambda
- DynamoDB table
- REST API
- API Gateway
Job dependency service

- S3: New Object creation
- StepFunctions
- SNS Topic
- Sns-to- Slack Lambda
- #bijo_notifications
- CloudWatch Logs
- AWS Fargate
- Job-Dependency Lambda
- DynamoDB table
- REST API
- API Gateway
- http request
Job dependency service

- S3
- StepFunctions
- AWS Fargate
- CloudWatch Logs
- SNS Topic
- Sns-to-slack Lambda
- #bijo_notifications
- Job-Dependency Lambda
- DynamoDB table
- REST API
- API Gateway

New Object creation → StepFunctions → SNS Topic → Sns-to-slack Lambda → #bijo_notifications

Run job → Job-Dependency Lambda → DynamoDB table

REST API

http request → API Gateway
Provide information about the target backend that this method will call and whether the incoming request data should be modified.

**Integration type**
- Lambda Function
- HTTP
- Mock
- AWS Service
- VPC Link

**Use Lambda Proxy integration**

**Lambda Region**
us-east-1

**Lambda Function**
dataops-bijo-job-dependency-function-test

**Execution role**

**Invoke with caller credentials**

**Credentials cache**
Do not add caller credentials to cache key

**Use Default Timeout**
Provide information about the target backend that this method will call and whether the incoming request data should be modified.

<table>
<thead>
<tr>
<th>Integration type</th>
<th>Lambda Function</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>HTTP</td>
</tr>
<tr>
<td></td>
<td>Mock</td>
</tr>
<tr>
<td></td>
<td>AWS Service</td>
</tr>
<tr>
<td></td>
<td>VPC Link</td>
</tr>
</tbody>
</table>

Use Lambda Proxy integration

<table>
<thead>
<tr>
<th>Lambda Region</th>
<th>us-east-1</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Lambda Function</th>
<th>dataops-bijo-job-dependency-function-test</th>
</tr>
</thead>
</table>

Execution role

<table>
<thead>
<tr>
<th>Invoke with caller credentials</th>
<th>Do not add caller credentials to cache key</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use Default Timeout</td>
<td></td>
</tr>
</tbody>
</table>
Extensible
Principles

REST API
- Python Client

Custom Runtime
- Custom runtime support using Docker containers
Custom runtime support through Docker

Support for any runtime environment and configuration using Docker

```
1 type: docker
2 command: python query_snowflake.py query.sql
3 depends-on: dataops/simple-dataops-job-example
```
Custom runtime support through Docker

Support for any runtime environment and configuration using Docker

```
1 type: docker
2 command: python query_snowflake.py query.sql
3 depends-on: dataops/simple-dataops-job-example
```
REST API

Python Client
BIJO APIs

**Manager Service**
- Run jobs on demand
- Get job execution history
- Get job status
- More…

**Notifications**
- Send Slack notifications easily
- Plots and images support
Manager API - Job execution

```python
from fundbox.data.bijotools.client.management import ManagementClient

client = ManagementClient()
client.run_job(source_name='test', job_group_name='dataops', job_name='sql-job')
```
Job Manager Service

S3 New Object creation

StepFunctions

Sns-to- Slack Lambda #bijonotifications

http post

Job-Manager Lambda

API Gateway

Client

http request

Job-Dependency Lambda

DynamoDB table

AWS Fargate

CloudWatch Logs

Key Value Parameter Store

New Object creation

Sns-to- Slack Lambda

http post

Job-Manager Lambda

API Gateway

Client

http request
Notifications API

```python
from fundbox.data.bijotools.client.notifications import NotificationClient

NotificationClient().send_slack_notification(
    slack_target_name='#bijo_notifications',
    body_text='Strata Data Conference New York: rocks!! :guitar: ')
```
Simple v2
BIJO Slack Bot

BIJO mlups/prospects-features-runner

Step function name: 2019.09.19.05.00.

Manager service info: "manager_service"

/bijo logs bi/daily-attribution
61
BIJO mlops/prospects-features-runner prod
Step function name: 2019.09.19.05.00.14.4
Manager service info: {'manager_service_reg:
/bijo history bi/daily-attribution
<table>
<thead>
<tr>
<th>Date/Time</th>
<th>Execution Duration</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>2019-09-05T12:30</td>
<td>32.4910</td>
<td>SUCCEEDED</td>
</tr>
<tr>
<td>2019-09-05T11:30</td>
<td>32.5870</td>
<td>SUCCEEDED</td>
</tr>
<tr>
<td>2019-09-05T10:30</td>
<td>32.7910</td>
<td>SUCCEEDED</td>
</tr>
<tr>
<td>2019-09-05T09:30</td>
<td>33.0140</td>
<td>SUCCEEDED</td>
</tr>
<tr>
<td>2019-09-05T08:30</td>
<td>32.7830</td>
<td>SUCCEEDED</td>
</tr>
<tr>
<td>2019-09-05T07:30</td>
<td>32.3640</td>
<td>SUCCEEDED</td>
</tr>
<tr>
<td>2019-09-05T06:30</td>
<td>32.5830</td>
<td>SUCCEEDED</td>
</tr>
<tr>
<td>2019-09-05T05:30</td>
<td>32.8380</td>
<td>SUCCEEDED</td>
</tr>
<tr>
<td>2019-09-05T04:30</td>
<td>32.7410</td>
<td>SUCCEEDED</td>
</tr>
<tr>
<td>2019-09-05T03:30</td>
<td>32.4790</td>
<td>SUCCEEDED</td>
</tr>
<tr>
<td>2019-09-05T02:30</td>
<td>32.6820</td>
<td>SUCCEEDED</td>
</tr>
<tr>
<td>2019-09-05T01:30</td>
<td>32.5690</td>
<td>SUCCEEDED</td>
</tr>
<tr>
<td>2019-09-05T00:30</td>
<td>33.1430</td>
<td>SUCCEEDED</td>
</tr>
<tr>
<td>2019-09-04T23:30</td>
<td>32.8410</td>
<td>SUCCEEDED</td>
</tr>
<tr>
<td>2019-09-04T22:30</td>
<td>32.7290</td>
<td>FAILED</td>
</tr>
<tr>
<td>2019-09-04T21:30</td>
<td>32.5470</td>
<td>SUCCEEDED</td>
</tr>
<tr>
<td>2019-09-04T20:30</td>
<td>32.4610</td>
<td>SUCCEEDED</td>
</tr>
<tr>
<td>2019-09-04T19:30</td>
<td>32.6670</td>
<td>SUCCEEDED</td>
</tr>
<tr>
<td>2019-09-04T18:30</td>
<td>32.5180</td>
<td>SUCCEEDED</td>
</tr>
<tr>
<td>2019-09-04T17:30</td>
<td>32.6370</td>
<td>SUCCEEDED</td>
</tr>
</tbody>
</table>
StepFunctions → SNS Topic → Job-Manager Lambda → Job-Dependency Lambda

Sns-to-Slack Lambda

Bot Lambda

#bijo_notifications

S3

New Object creation

CloudWatch Logs

AWS Fargate

Key Value Parameter Store

DynamoDB table

API Gateway

Client

http post

http request
StepFunctions

SNS Topic

Sns-to- Slack Lambda

#bijo_notifications

CloudWatch Logs

Key Value Parameter Store

AWS Fargate

Job-Manager Lambda

Job-Dependency Lambda

API Gateway

Client

http request

Bot Lambda

http post

DynamoDB table

Bot calls

Execute job

Get history

Execute job

New Object creation
Wait ....
```python
1 type: python
2 entrypoint: entrypoint.py
3 depends-on:
4   type: s3
5    bucket: default_bucket
6    regex: "IPV[\\S]*\.csv"
```
S3 object creation == 1 job

1. S3
2. New Object creation
3. StepFunctions
4. SNS Topic
5. Job-Dependency Lambda
6. AWS Fargate
S3 object creation == 1 job

- S3
- New Object creation
- StepFunctions
- SNS Topic
- Job-Dependency Lambda
- AWS Fargate

Fundbox
S3 object creation == 1 job

S3

New Object creation

StepFunctions

SNS Topic

Job-Dependency Lambda

AWS Fargate
S3 object creation == 1 job

- S3
- New Object creation
- StepFunctions
- SNS Topic
- Job-Dependency Lambda
S3 object creation == 1 job

- S3
- New Object creation
- StepFunctions
- SNS Topic
- Job-Dependency Lambda
- DynamoDB table
S3 object creation == 1 job

- S3
- New Object creation
- StepFunctions
- SNS Topic
- Job-Dependency Lambda
- AWS Fargate
- DynamoDB table
S3 object creation == 1 job

- S3
- New Object creation
- StepFunctions
- SNS Topic
- Job-Dependency Lambda
- AWS Fargate
- Docker
- DynamoDB table
S3 object creation == 1 job
S3 object creation == 1 job

- S3
- New Object creation
- StepFunctions
- SNS Topic
- Job-Dependency Lambda
- AWS Fargate
- docker
- DynamoDB table
S3 object creation == 1 job

- S3
- New Object creation
- StepFunctions
- SNS Topic
- Job-Dependency Lambda
- AWS Fargate
- DynamoDB table

Number of ECS Fargate tasks is limited
**S3 object creation == 1 job**

- S3
- New Object creation
- StepFunctions
- SNS Topic
- Job-Dependency Lambda
- DynamoDB table
S3 object creation == 1 job
S3 object creation == 1 job

S3

New Object creation

StepFunctions

SNS Topic

Job-Dependency Lambda

AWS Fargate

DynamoDB table
S3 object creation == 1 job

- S3
- StepFunctions
- SNS Topic
- Job-Dependency Lambda
- DynamoDB table
- Job-Manager Lambda

New Object creation

AWS Fargate
S3 object creation == 1 job

- S3
- New Object creation
- StepFunctions
- SNS Topic
- Any available resources?
- Job-Dependency Lambda
- Job-Manager Lambda
- AWS Fargate
- DynamoDB table
S3 object creation == 1 job

- S3
- New Object creation
- StepFunctions
- AWS Fargate
- SNS Topic
- Job-Dependency Lambda
- Job-Manager Lambda
- Yes
- DynamoDB table
S3 object creation == 1 job

- S3
- New Object creation
- StepFunctions
- SNS Topic
- Job-Manager Lambda
- Job-Dependency Lambda
- DynamoDB table
- Run job
S3 object creation == 1 job

- S3
- New Object creation
- StepFunctions
- Any available resources?
- SNS Topic
- Job-Manager Lambda
- Job-Dependency Lambda
- AWS Fargate
- DynamoDB table
S3 object creation == 1 job

- S3
- New Object creation
- StepFunctions
- AWS Fargate
- SNS Topic
- Job-Manager Lambda
- Job-Dependency Lambda
- No :(
- DynamoDB table
S3 object creation == 1 job

- S3
- New Object creation
- StepFunctions
- SNS Topic
- Job-Manager Lambda
- Job queue
- Job-Dependency Lambda
- DynamoDB table
- Job X done

Fundbox
S3 object creation == 1 job

- S3
- New Object creation
- StepFunctions
- SNS Topic
- Job-Dependency Lambda
- DynamoDB table
- Job-Manager Lambda
- Job queue
- AWS Fargate
- Run job
- DynamoDB table
Principles

Simple  
Easy to use and maintain

Monitored  
Metrics collection  
Logs collection

Extensible  
Multiple runtime support  
Runtime params  
Low level API
Impact

35

$0.01

>4000
The Future

- Dynamic job resource allocation
- UI Dashboard
- Release as an open-source project
Thank you

@tomer_levi
tomer.levi@fundbox.com
Rate today’s session

Cyberconflict: A new era of war, sabotage, and fear

9:30am-10:10am Wednesday, March 27, 2019
Location: Ballroom
Secondary topic: Security and Privacy

We’re living in a new era of constant sabotage, misinformation, and fear, in which everyone is a target, and you’re often the collateral damage in a growing conflict among states. From crippling infrastructure to sowing discord and doubt, cyber is now the weapon of choice for democracies, dictators, and terrorists.

David Sanger explains how the rise of cyberweapons has transformed geopolitics like nothing since the invention of the atomic bomb. Moving from the White House Situation Room to the dens of Chinese, Russian, North Korean, and Iranian hackers to the boardrooms of Silicon Valley, David reveals a world coming face-to-face with the perils of technological revolution—a conflict that the United States helped start when it began using cyberweapons against Iranian nuclear plants and North Korean missile launches. But now we find ourselves in a conflict we’re uncertain how to control, as our adversaries exploit vulnerabilities in our hyperconnected nation and we struggle to figure out how to deter these complex, short-of-war attacks.

David Sanger
The New York Times

David E. Sanger is the national security correspondent for the New York Times as well as a national security and political contributor for CNN and a frequent guest on CBS This Morning, Face the Nation, and many PBS shows.