

Enabling Data Scientists to easily create and own Kafka Consumers

Stefan Krawczyk

Mgr. Data Platform - Model Lifecycle

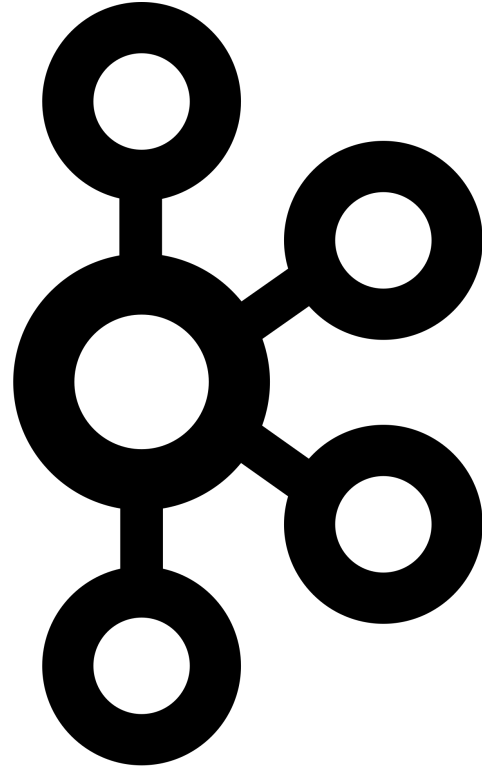
 @stefkrawczyk
 [linkedin.com/in/skrawczyk](https://www.linkedin.com/in/skrawczyk)

Try out Stitch Fix → goo.gl/Q3tCQ3

STITCH FIX



STITCH FIX



Agenda

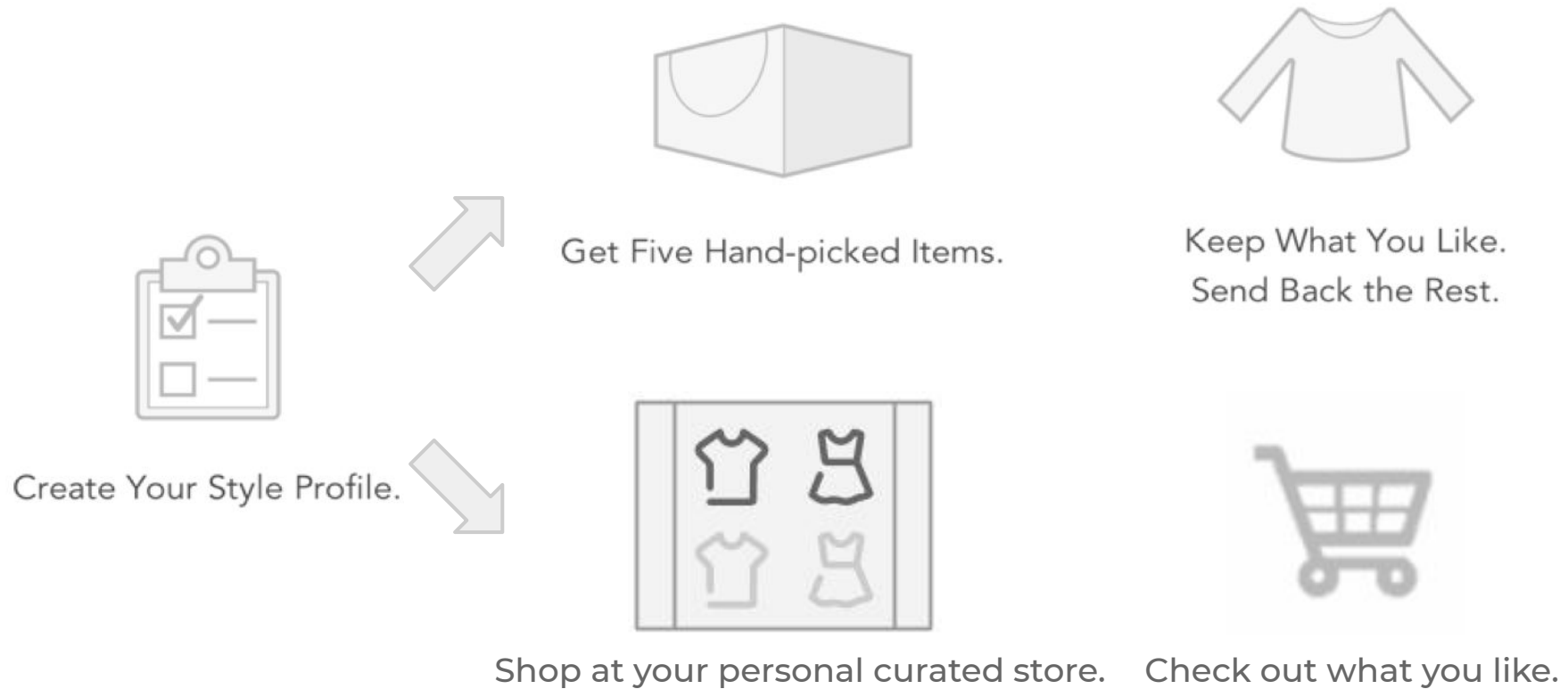
- What is Stitch Fix?
- Data Science @ Stitch Fix
- Stitch Fix's opinionated Kafka consumer
- Learnings & Future Directions

What is Stitch Fix

What does the company do?

STITCH FIX

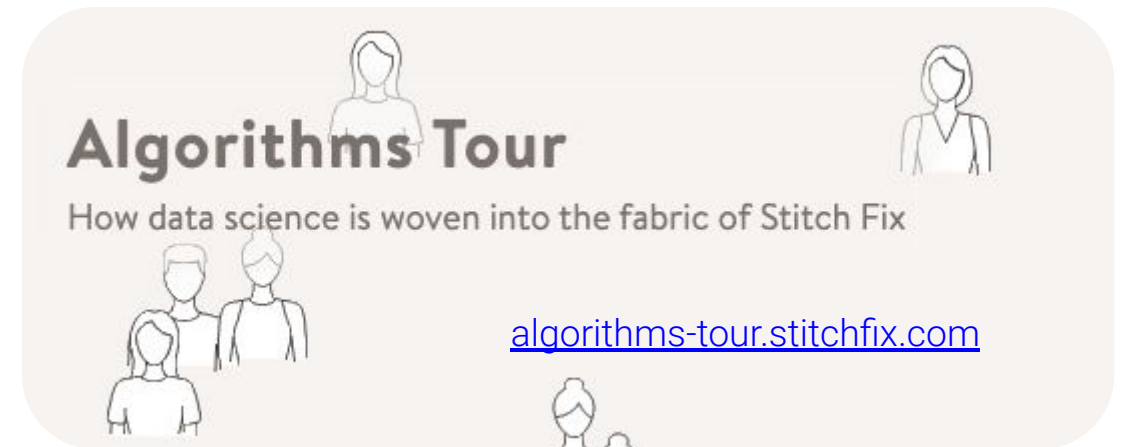
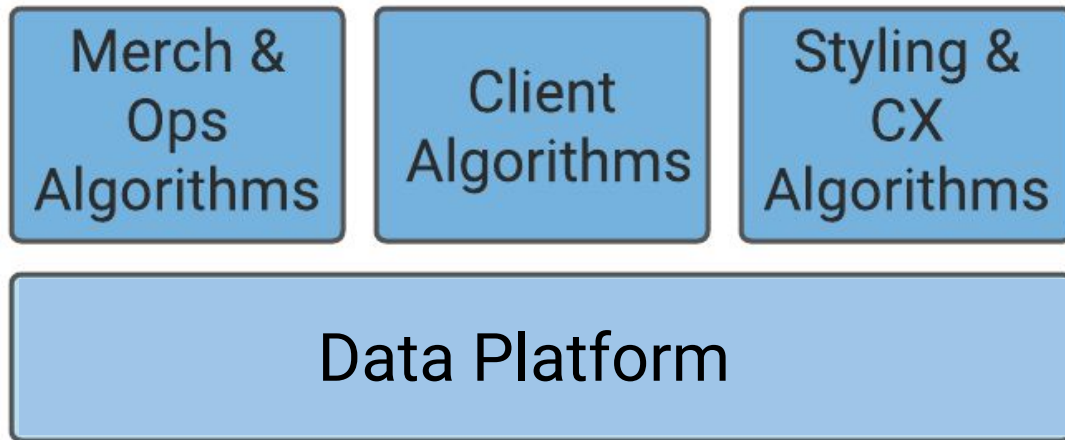
Stitch Fix is a personal styling service.



Data Science is behind everything we do.

Algorithms Org.

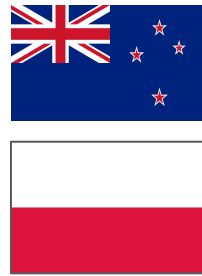
- 145+ Data Scientists and Platform Engineers
- 3 main verticals + platform



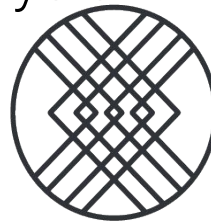
whoami



Pre-covid look



Stefan Krawczyk
Mgr. Data Platform - Model Lifecycle



STITCH FIX

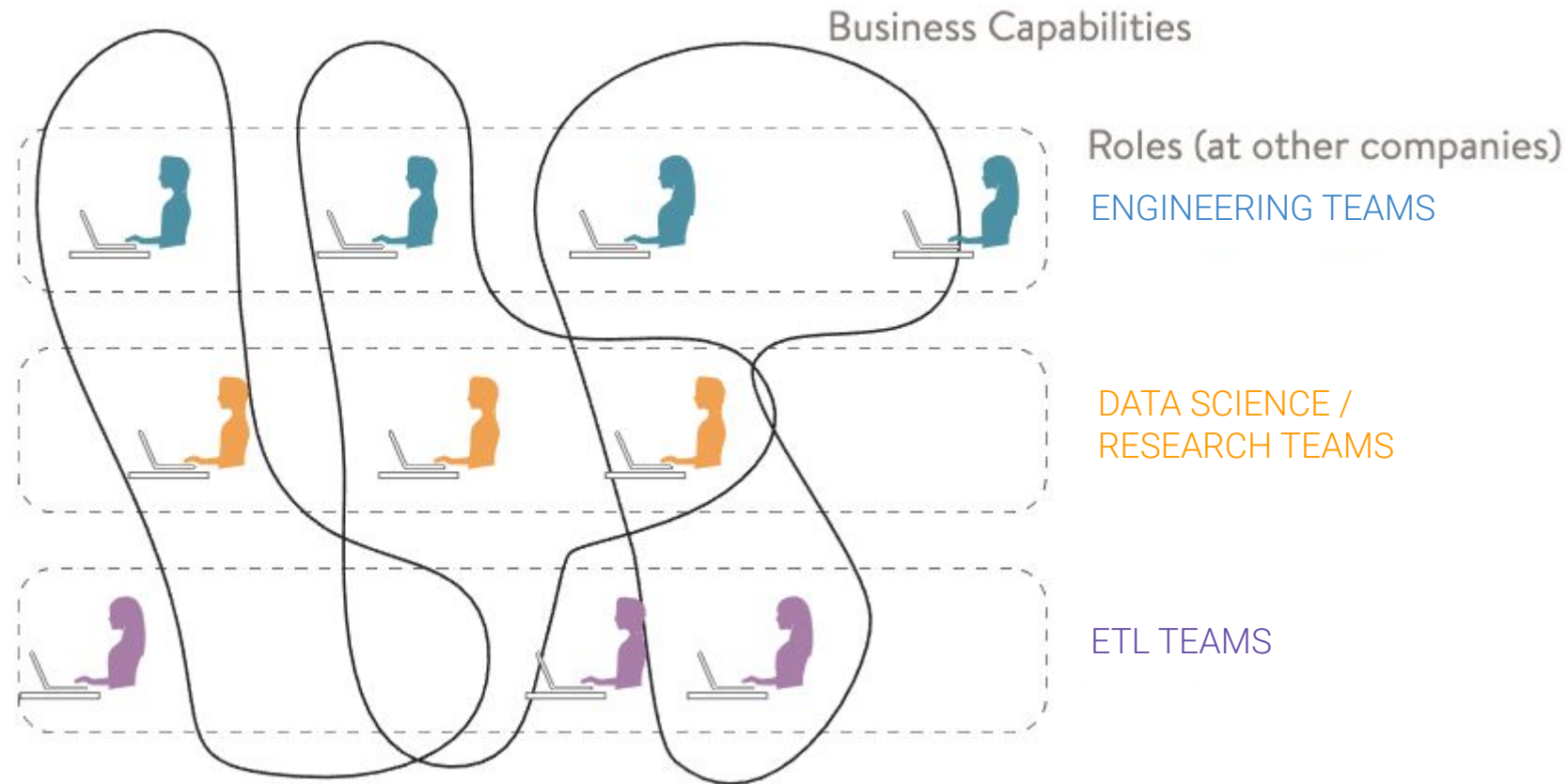
Data Science @ Stitch Fix

Expectations we have on DS @ Stitch Fix

Most common approach to Data Science

Typical organization:

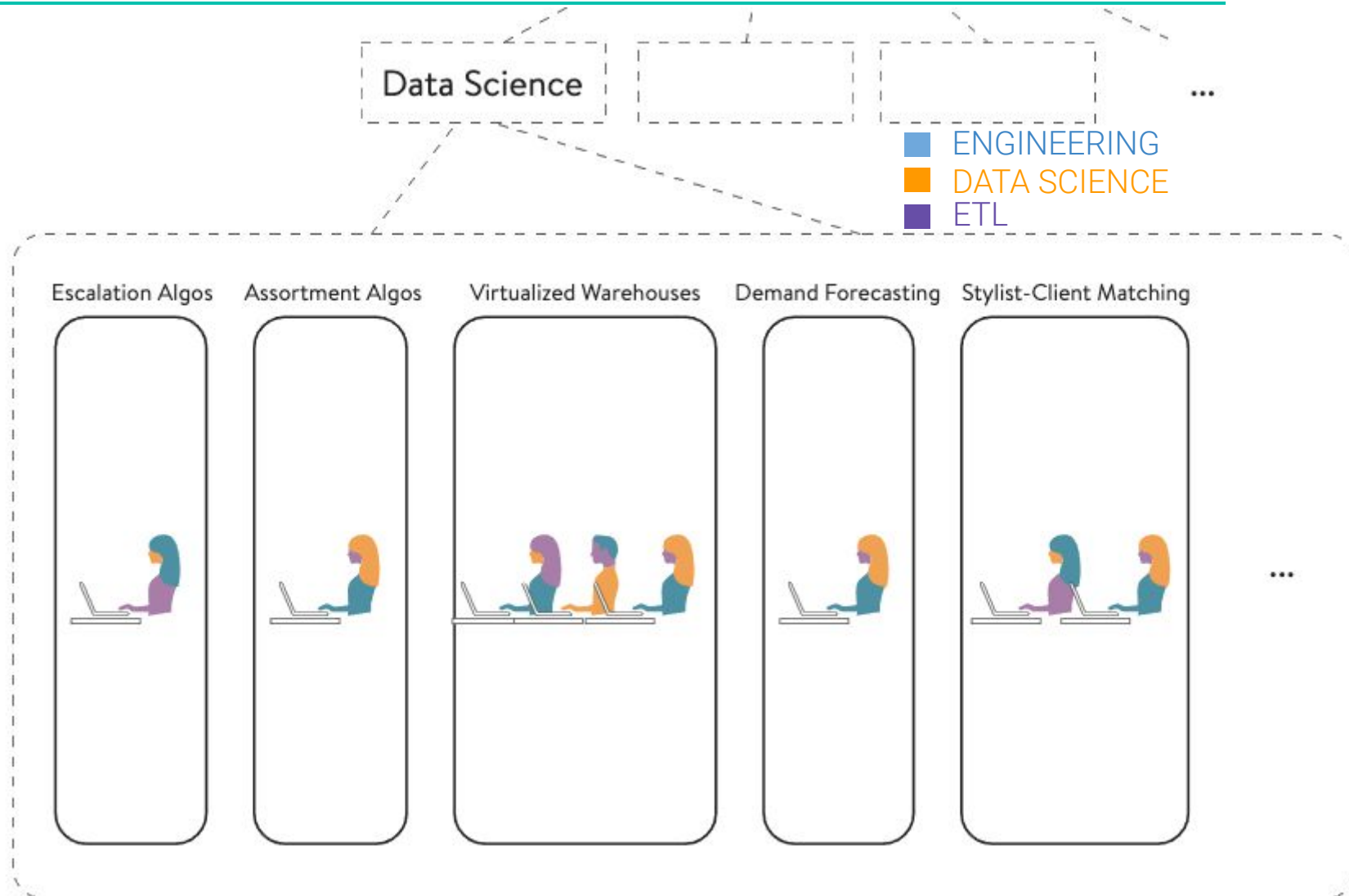
- Horizontal teams
- Hand off
- Coordination required



Full Stack Data Science

At Stitch Fix:

- Single Organization
- No handoff
- End to end ownership
- We have a lot of them!
- Built on top of data platform tools & abstractions.

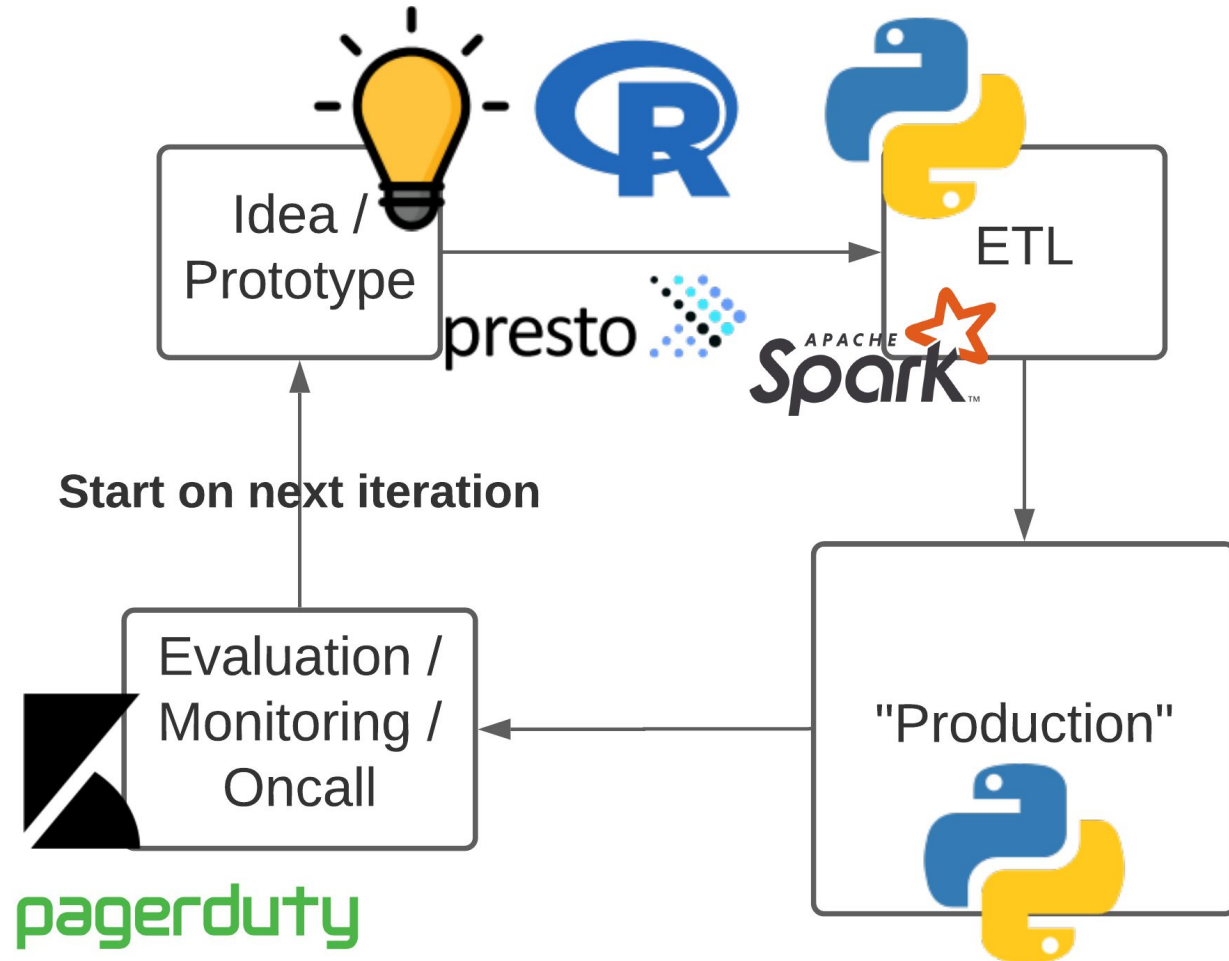


A typical DS flow at Stitch Fix

Full Stack Data Science

Typical flow:

- Idea / Prototype
- ETL
- "Production"
- Eval/Monitoring/Oncall
- Start on next iteration



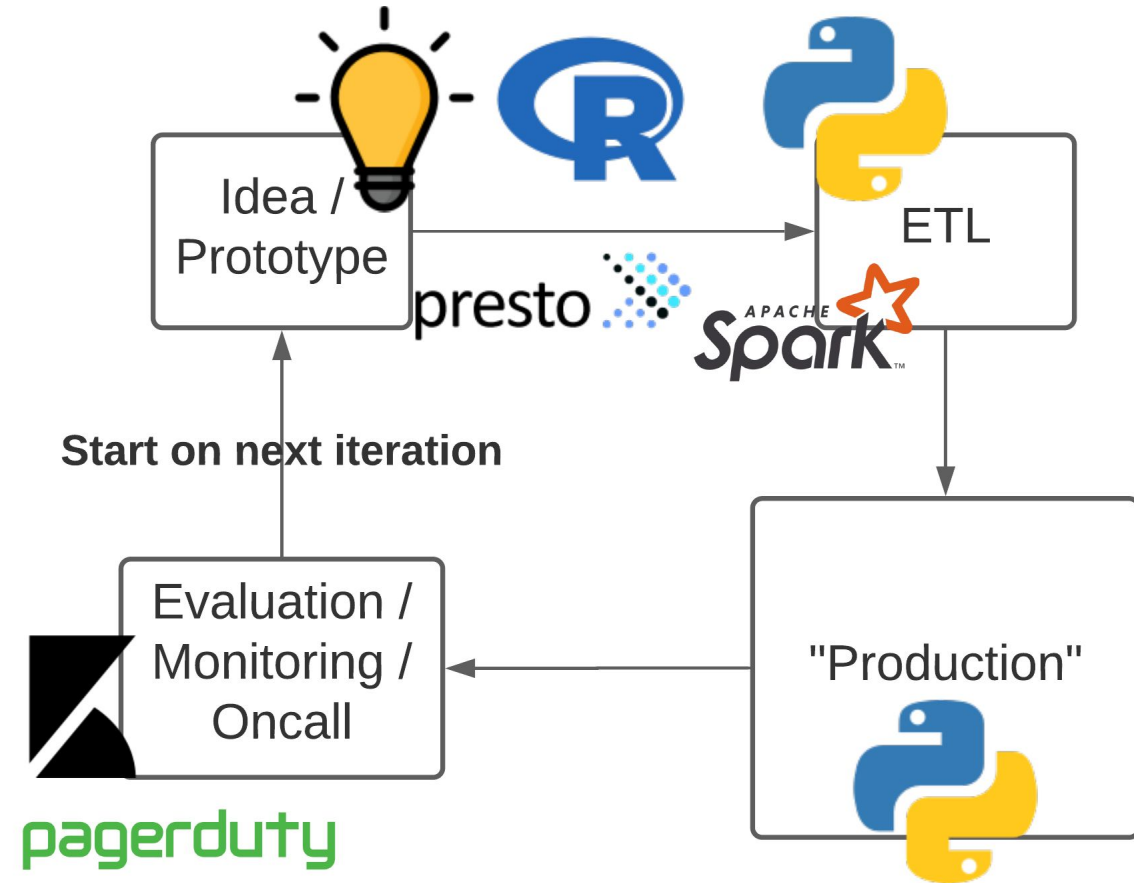
A typical DS flow at Stitch Fix

Full Stack Data Science

Production can mean:

- Web service
- Batch job / Table
- Kafka consumer

Heavily biased towards Python.



Example Kafka Consumers

Example use cases DS have built kafka consumers for

- A/B testing bucket allocation
- Transforming raw inputs into features
- Saving data into feature stores
- Event driven model prediction
- Triggering workflows

Stitch Fix's opinionated Kafka consumer

Code first, explanation second

Consumer Code []
Architecture []
Mechanics []

Our “Hello world”

Hello world consumer

A simple example

hello_world_consumer.py

```
import sf_kafka

@sf_kafka.register(kafka_topic='some.topic', output_schema={})
def hello_world(messages: List[str]) -> dict:
    """Hello world example
    :param messages: list of strings, which are JSON objects.
    :return: empty dict, as we don't need to emit any events.
    """
    list_of_dicts = [json.loads(m) for m in messages]
    print(f'Hello world I have consumed the following {list_of_dicts}')
    return {}
```

To run this:

> pip install sf_kafka

> python -m sf_kafka.server hello_world_consumer

Hello world consumer

A simple example

hello_world_consumer.py

```
import sf_kafka

@sf_kafka.register(kafka_topic='some.topic', schema={})
def hello_world(messages: List[str]) -> List[Dict[str, str]]:
    """Hello world example
    :param messages: list of messages
    :return: empty list to emit any events.
    """
    list_of_dicts = []
    for m in messages:
        print(f'Hello world I have consumed the following {list_of_dicts}')
    return {}
```

So what is this doing?

To run this:

> pip install sf_kafka

> python -m sf_kafka.server hello_world_consumer

Hello world consumer

A simple example

hello_world_consumer.py

```
import sf_kafka

@sf kafka.register(kafka topic='some.topic', output_schema={})
def hello_world(messages: List[str]) -> dict:
    """Hello world example
    :param messages: list of strings, which are JSON objects.
    :return: empty dict, as we don't need to emit any events.
    """
    list_of_dicts = [json.loads(m) for m in messages]
    print(f'Hello world I have consumed the following {list_of_dicts}')
    return {}
```

1. Python function that takes in a list of strings called messages.

Hello world consumer

A simple example

hello_world_consumer.py

```
import sf_kafka

@sf_kafka.register(kafka_topic='some.topic', output_schema={})
def hello_world(messages: List[str]) -> dict:
    """Hello world example
    :param messages: list of strings, which are JSON objects.
    :return: empty dict, as we don't need to emit any events.
    """
    list_of_dicts = [json.loads(m) for m in messages]
    print(f'Hello world I have consumed the following {list_of_dicts}')
    return {}
```

1. Python function that takes in a list of strings called messages.
2. **We're processing the messages into dictionaries.**

Hello world consumer

A simple example

hello_world_consumer.py

```
import sf_kafka

@sf_kafka.register(kafka_topic='some.topic', output_schema={})
def hello_world(messages: List[str]) -> dict:
    """Hello world example
    :param messages: list of strings, which are JSON objects.
    :return: empty dict, as we don't need to emit any events.
    """
    list_of_dicts = [json.loads(m) for m in messages]
    print(f'Hello world I have consumed the following {list_of_dicts}')
    return {}
```

1. Python function that takes in a list of strings called messages.
2. We're processing the messages into dictionaries.
- 3. Printing them to console. (DS would replace this with a call to their function())**

Hello world consumer

A simple example

hello_world_consumer.py

```
import sf_kafka

@sf kafka.register(kafka topic='some.topic', output schema={})
def hello_world(messages: List[str]) -> dict:
    """Hello world example
    :param messages: list of strings, which are JSON objects.
    :return: empty dict, as we don't need to emit any events.
    """
    list_of_dicts = [json.loads(m) for m in messages]
    print(f'Hello world I have consumed the following {list_of_dicts}')
    return {}
```

1. Python function that takes in a list of strings called messages.
 2. We're processing the messages into dictionaries.
 3. Printing them to console. (DS would replace this with a call to their function())
 - 4. We are registering this function to consume from 'some.topic' with no output.**
-

Consumer Code



Architecture

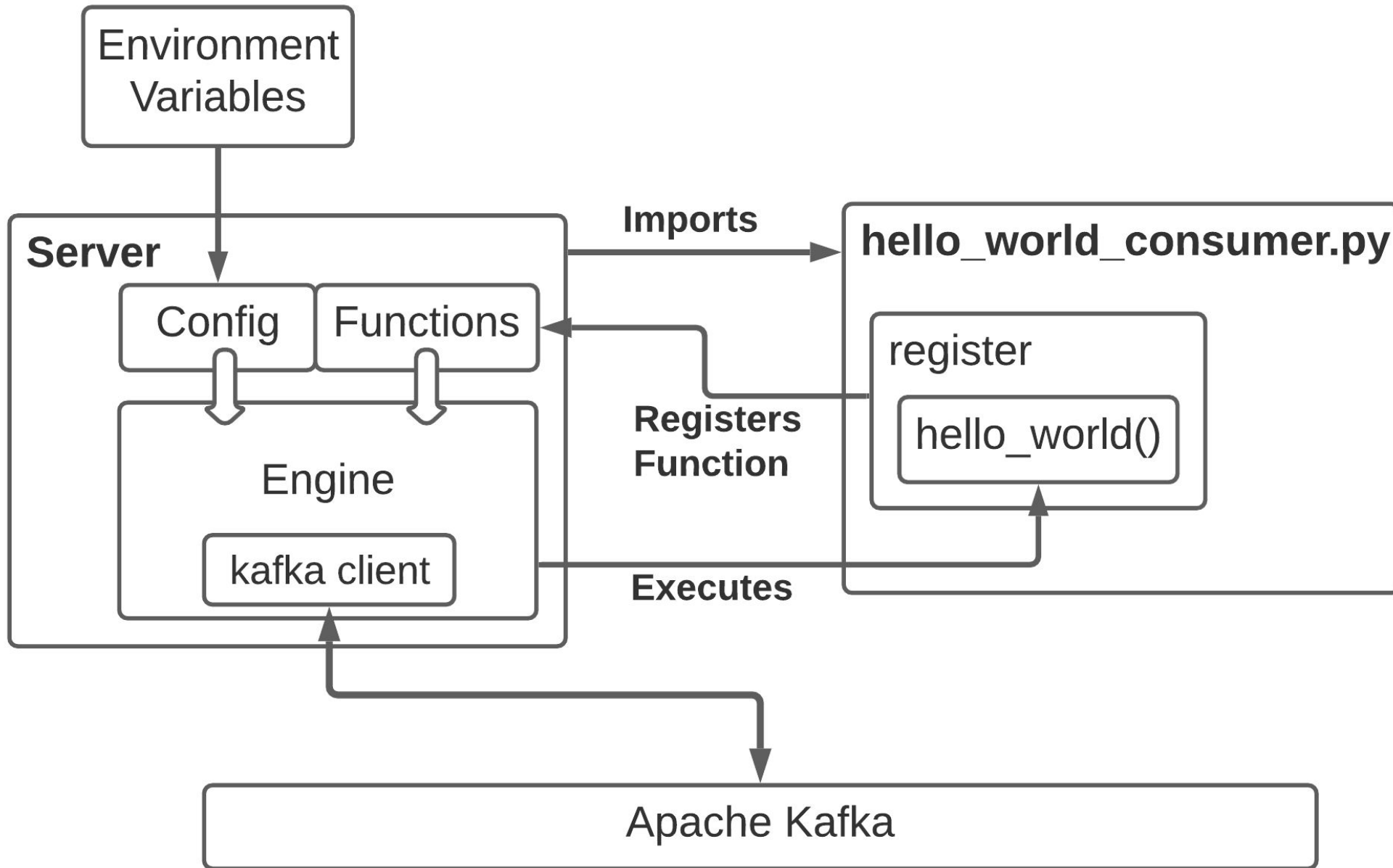


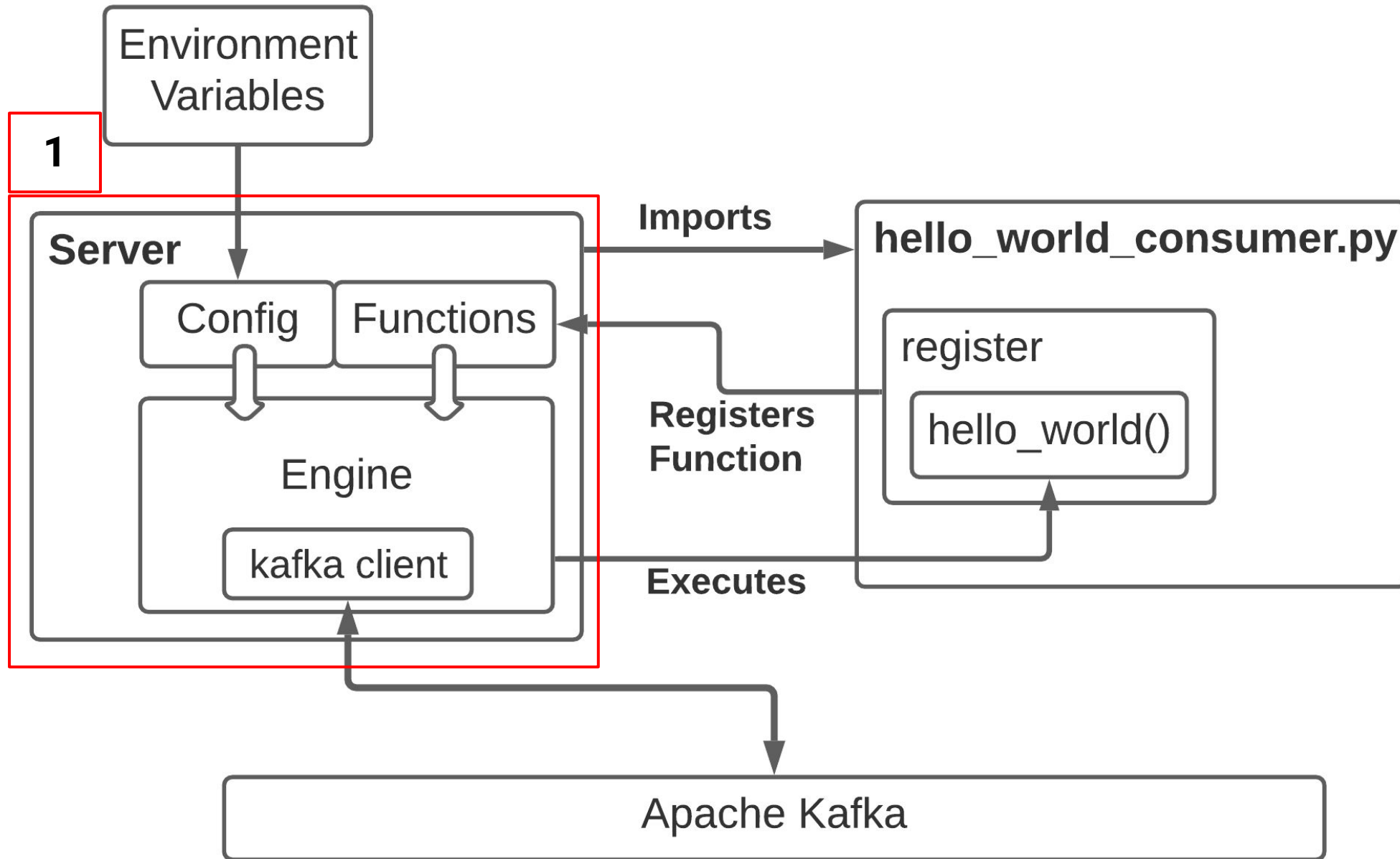
Mechanics

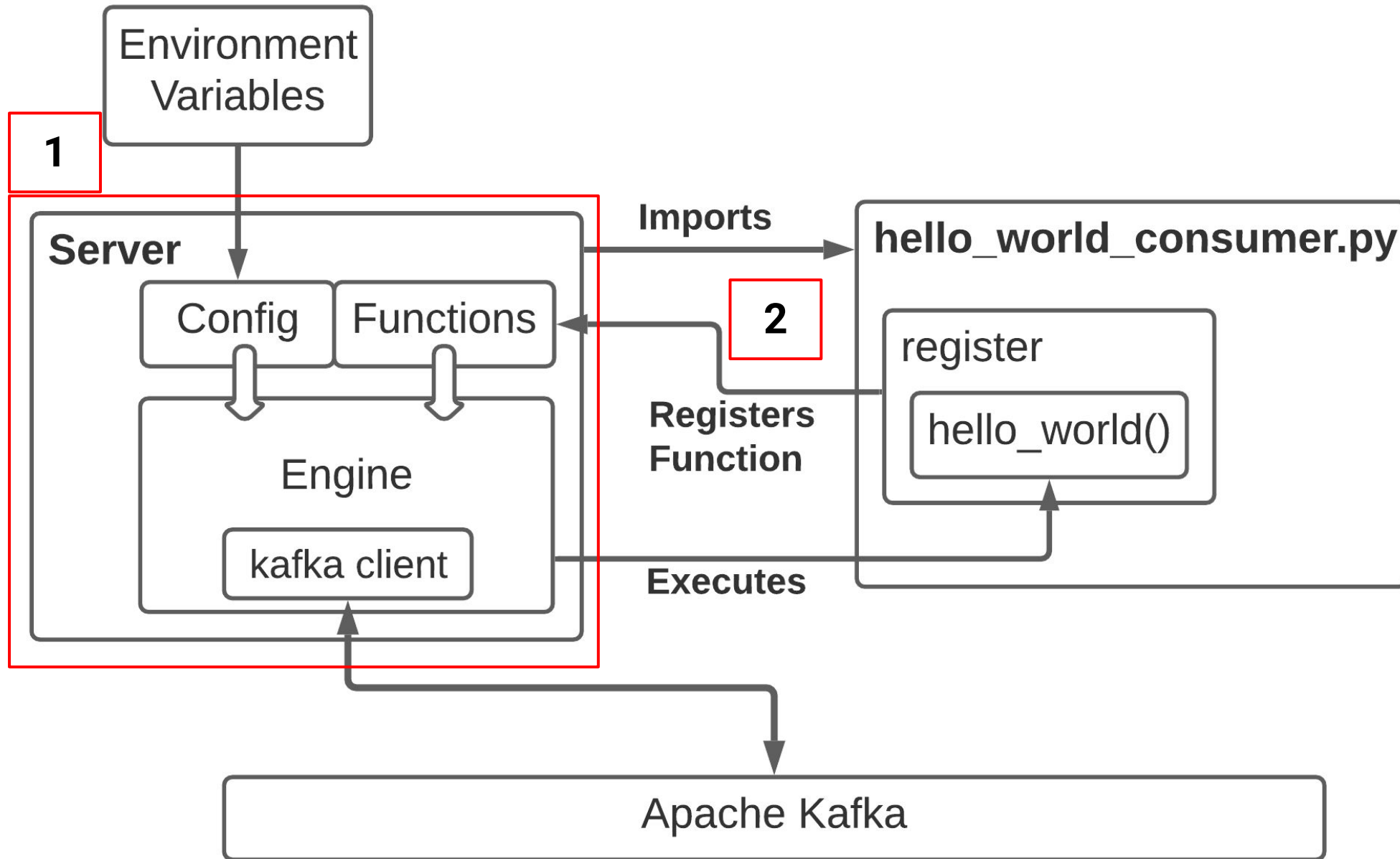


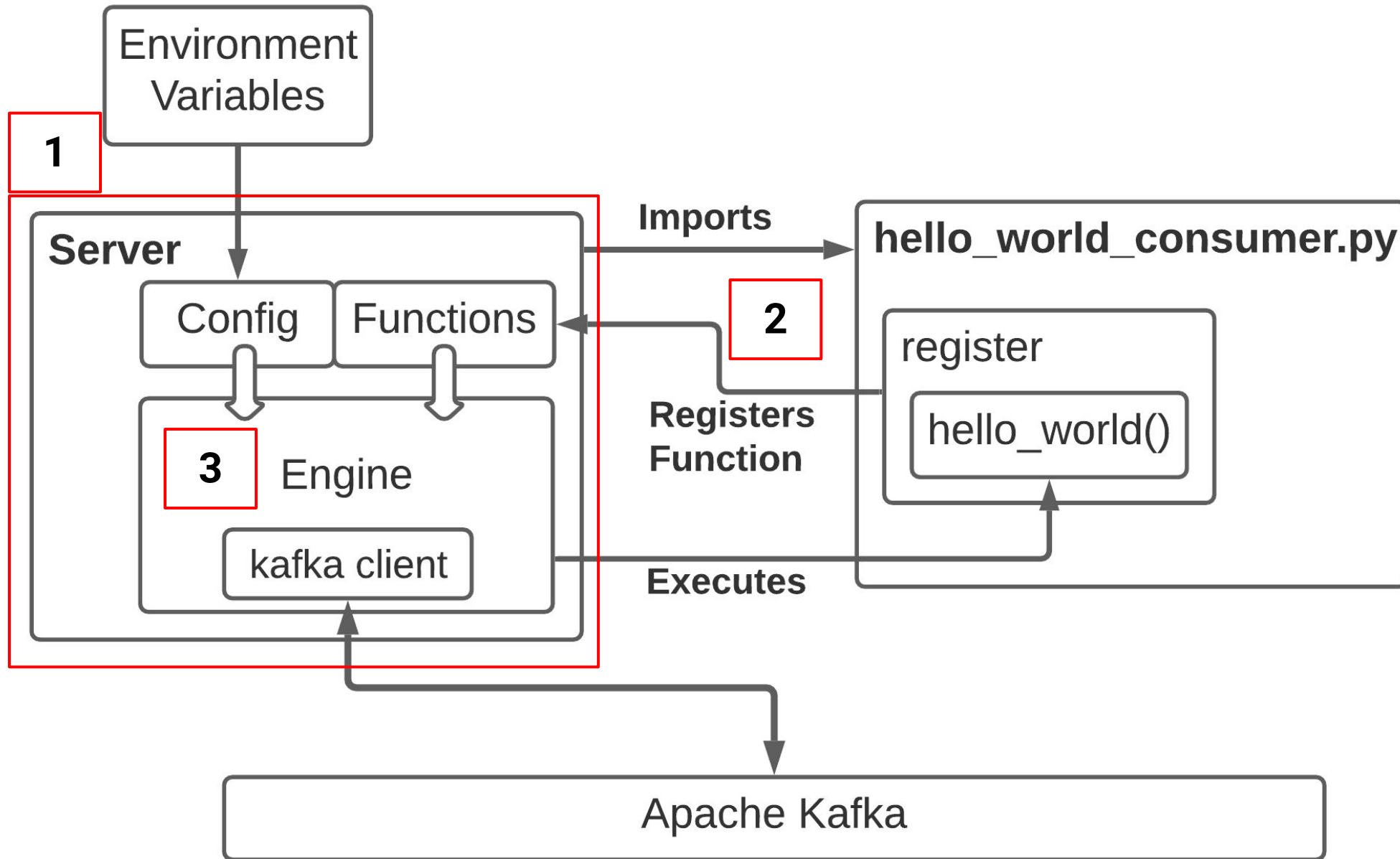
So what's really going on?

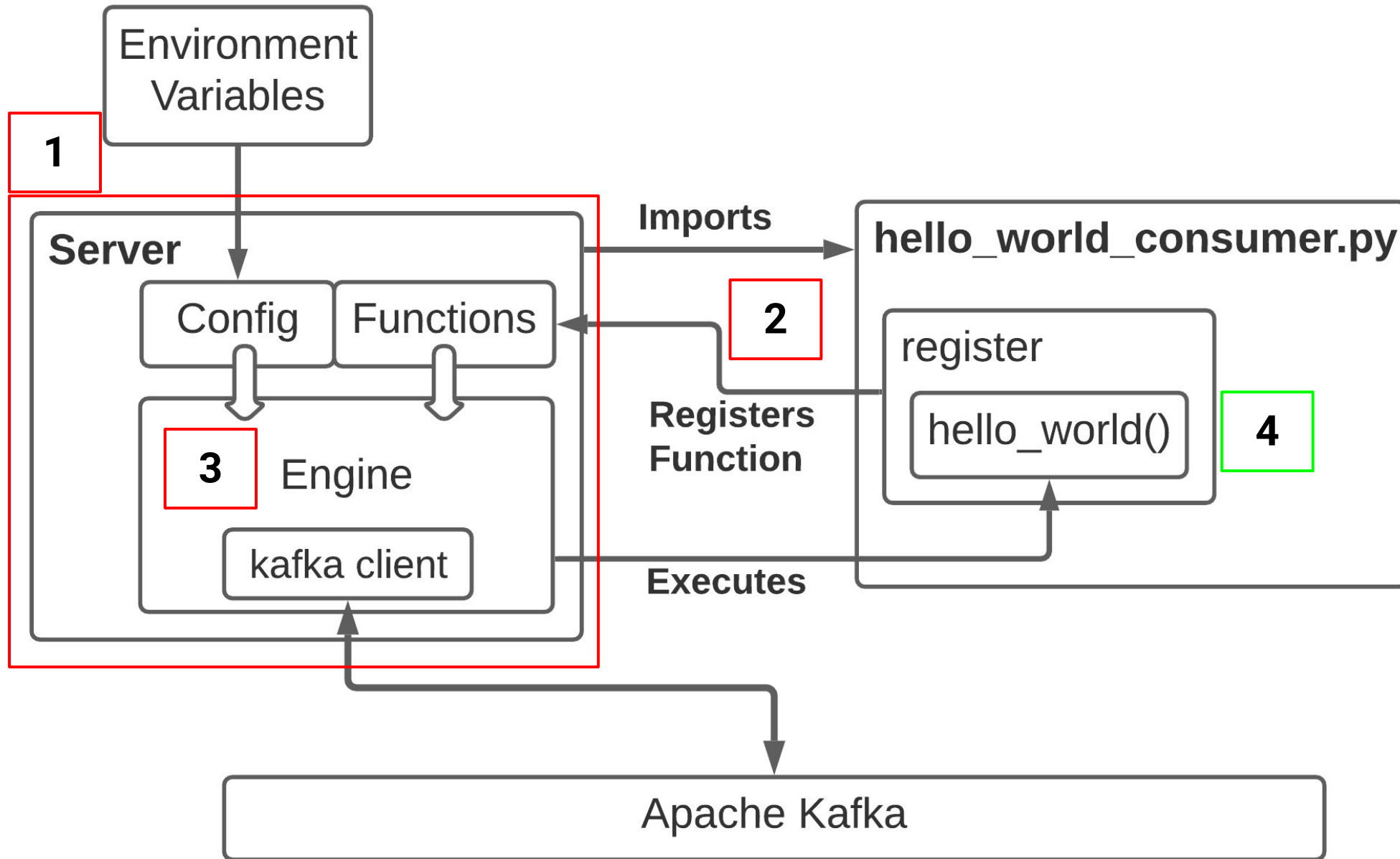
When someone runs `python -m sf_kafka.server hello_world_consumer`

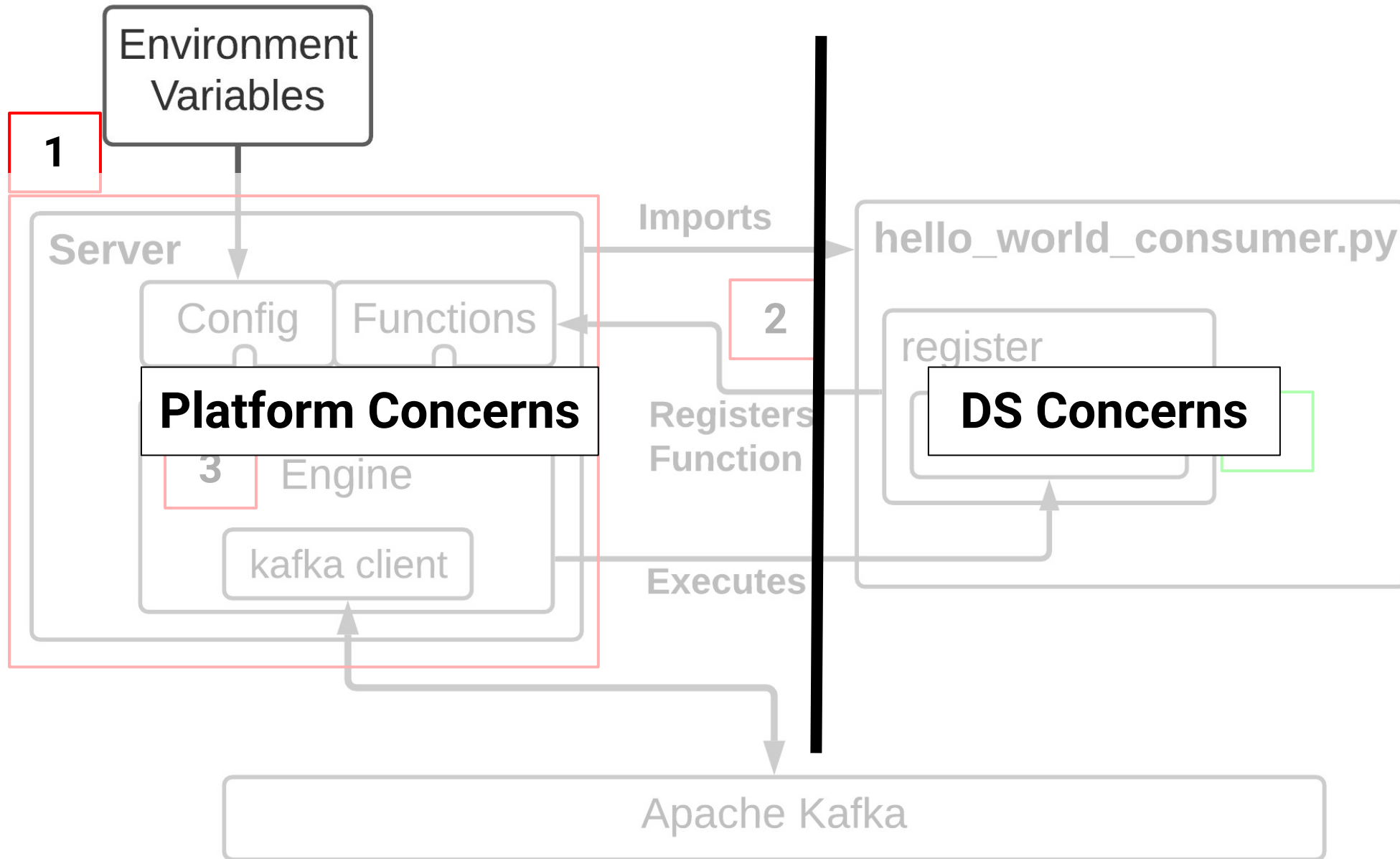












Consumer Code



Architecture



Mechanics



Platform Concerns vs DS Concerns

Platform Concerns vs DS Concerns

What does each side own

Platform Concerns	DS Concerns
<ul style="list-style-type: none">● Kafka consumer operation:<ul style="list-style-type: none">○ What python kafka client to use○ Kafka client configuration○ Processing assumptions<ul style="list-style-type: none">■ At least once or at most once○ How to write back to kafka<ul style="list-style-type: none">■ Direct to cluster or via a proxy?■ Message serialization format	

Platform Concerns vs DS Concerns

What does each side own

Platform Concerns	DS Concerns
<ul style="list-style-type: none">● Kafka consumer operation:<ul style="list-style-type: none">○ What python kafka client to use○ Kafka client configuration○ Processing assumptions<ul style="list-style-type: none">■ At least once or at most once○ How to write back to kafka<ul style="list-style-type: none">■ Direct to cluster or via a proxy?■ Message serialization format● Production operations:<ul style="list-style-type: none">○ Topic partitioning○ Deployment vehicle for consumers○ Monitoring hooks & tools	

Platform Concerns vs DS Concerns

What does each side own

Platform Concerns	DS Concerns
<ul style="list-style-type: none">● Kafka consumer operation:<ul style="list-style-type: none">○ What python kafka client to use○ Kafka client configuration○ Processing assumptions<ul style="list-style-type: none">■ At least once or at most once○ How to write back to kafka<ul style="list-style-type: none">■ Direct to cluster or via a proxy?■ Message serialization format● Production operations:<ul style="list-style-type: none">○ Topic partitioning○ Deployment vehicle for consumers○ Monitoring hooks & tools	<ul style="list-style-type: none">● Configuration:<ul style="list-style-type: none">○ App name [required]○ Which topic(s) to consume from [required]○ Process from beginning/end? [optional]○ Processing “batch” size [optional]○ Number of consumers [optional]● Python function that operates over a list● Output topic & message [if any]● Oncall

Platform Concerns vs DS Concerns

What does each side own

Platform Concerns	DS Concerns
<ul style="list-style-type: none">● Kafka consumer operation:<ul style="list-style-type: none">○ What python kafka client to use○ Kafka client configuration○ Processing assumptions<ul style="list-style-type: none">■ At least once or at most once○ How to write back to kafka<ul style="list-style-type: none">■ Direct to cluster or via a proxy?■ Message serialization format● Production operations:<ul style="list-style-type: none">○ Topic partitioning○ Deployment vehicle for consumers○ Monitoring hooks & tools	<ul style="list-style-type: none">● Configuration:<ul style="list-style-type: none">○ App name [required]○ Which topic(s) to consume from [required]○ Process from beginning/end? [optional]○ Processing “batch” size [optional]○ Number of consumers [optional]● Python function that operates over a list● Output topic & message [if any]● Oncall

■ Can change without DS involvement -- just need to rebuild their app.

Platform Concerns vs DS Concerns

What does each side own

Platform Concerns	DS Concerns
<ul style="list-style-type: none">● Kafka consumer operation:<ul style="list-style-type: none">○ What python kafka client to use○ Kafka client configuration○ Processing assumptions<ul style="list-style-type: none">■ At least once or at most once○ How to write back to kafka<ul style="list-style-type: none">■ Direct to cluster or via a proxy?■ Message serialization format● Production operations:<ul style="list-style-type: none">○ Topic partitioning○ Deployment vehicle for consumers○ Topic monitoring hooks & tools	<ul style="list-style-type: none">● Configuration:<ul style="list-style-type: none">○ App name [required]○ Which topic(s) to consume from [required]○ Process from beginning/end? [optional]○ Processing “batch” size [optional]○ Number of consumers [optional]● Python function that operates over a list● Output topic & message [if any]● Oncall

■ Requires coordination with DS

Salient choices we made on Platform

Platform Concern	Choice	Benefit
Kafka Client		
Processing assumption		

Salient choices we made on Platform

Platform Concern	Choice	Benefit
Kafka Client	python confluent-kafka (librdkafka)	librdkafka is very performant & stable.
Processing assumption		

Salient choices we made on Platform

Platform Concern	Choice	Benefit
Kafka Client	python confluent-kafka (librdkafka)	librdkafka is very performant & stable.
Processing assumption	At least once; functions should be idempotent.	Enables very easy error recovery strategy: <ul style="list-style-type: none">● Consumer app breaks until it is fixed; can usually wait until business hours.● No loss of events.● Monitoring trigger is consumer lag.

Salient choices we made on Platform

Platform Concern	Choice	Benefit
Message serialization format		
Do we want to write back to kafka directly?		

Salient choices we made on Platform

Platform Concern	Choice	Benefit
Message serialization format	JSON	Easy mapping to and from python dictionaries. Easy to grok for DS. * python support for other formats wasn't great.
Do we want to write back to kafka directly?		

Salient choices we made on Platform

Platform Concern	Choice	Benefit
Message serialization format	JSON	Easy mapping to and from python dictionaries. Easy to grok for DS. * python support for other formats wasn't great.
Do we want to write back to kafka directly?	Write via proxy service first.	Enabled: <ul style="list-style-type: none">● Not having producer code in the engine.● Ability to validate/introspect all messages.● Ability to augment/change minor format structure without having to redeploy all consumers.

Consumer Code



Architecture



Mechanics



What's missing?



Consumer Code



Architecture



Mechanics



What's missing?



Completing the production story

Consumer Code



Architecture



Mechanics



What's missing?



Self-service

Completing the production story

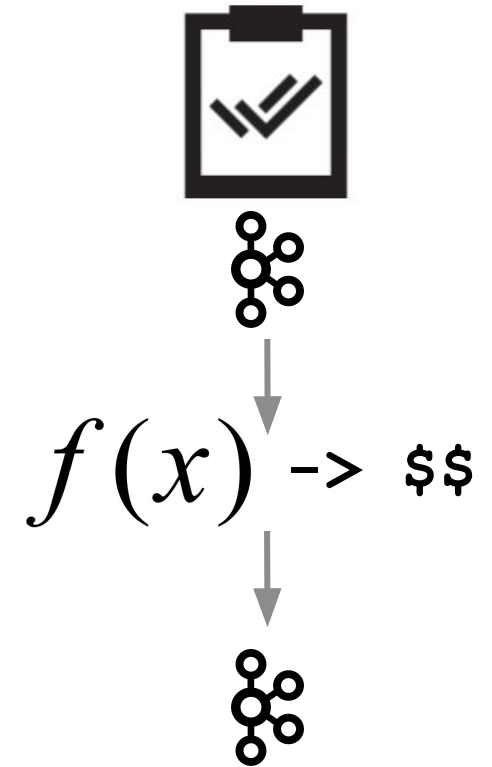
Completing the production story

The self-service story of how a DS gets a consumer to production

Example Use Case: Event Driven Model Prediction



1. Client signs up & fills out profile.
2. Event is sent - *client.signed_up*.
3. Predict something about the client.
4. Emit predictions back to kafka.
5. Use this for email campaigns.



Completing the production story

The self-service story of how a DS gets a consumer to production

1. **Determine the topic(s) to consume.**

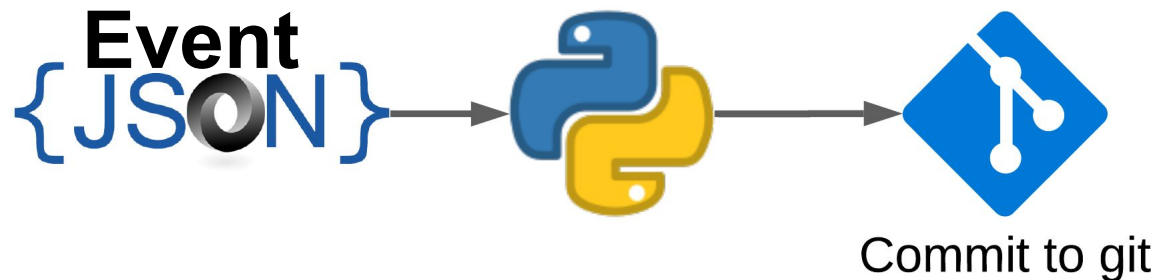
The screenshot shows the Confluent Topics page for a cluster named "DMV (prod)". The navigation bar includes "Topics" and "Connectors". Below the navigation bar, there is a "Topics" section with a "Filter by name:" input field. A table header is visible with the following columns: "Topic Name", "Messages / Minute", and "Consumers".

Topic Name	Messages / Minute	Consumers
------------	-------------------	-----------

Completing the production story

The self-service story of how a DS gets a consumer to production

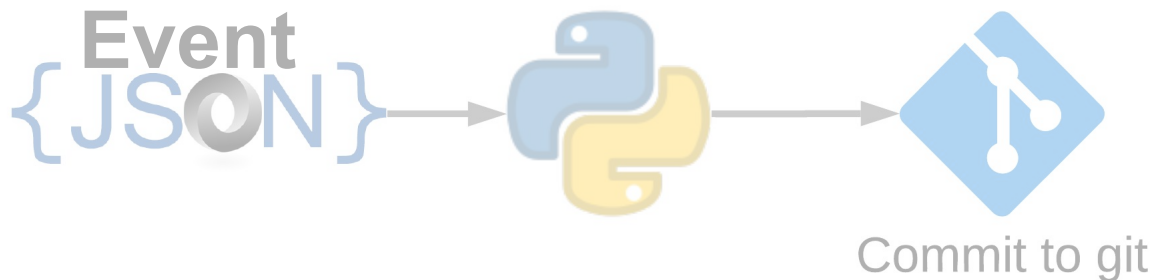
1. Determine the topic(s) to consume.
2. **Write code:**
 - a. Create a function & decorate it to process events
 - b. If outputting an event, write a schema
 - c. Commit code to a git repository



Completing the production story

The self-service story of how a DS gets a consumer to production

1. Determine the topic(s) to consume.
2. **Write code:**
 - a. *Create a function & decorate it to process events*
 - b. If outputting an event, write a schema
 - c. Commit code to a git repository



my_prediction.py

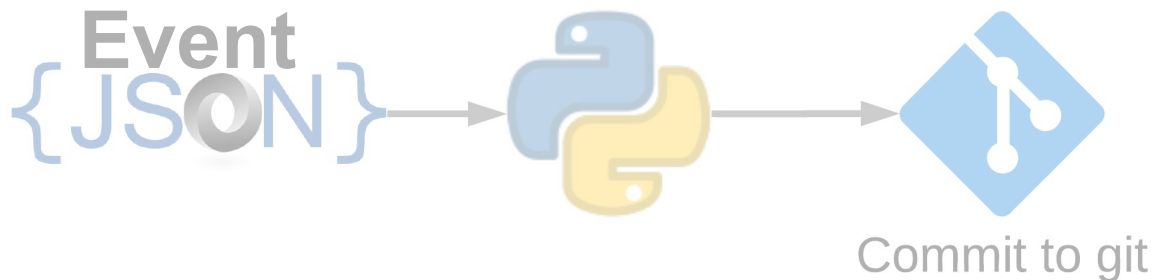
```
def create_prediction(client: dict) -> dict:
    # DS would write side effects or fetches here.
    # E.g. grab features, predict,
    # create output message;
    prediction = ...
    return make_output_event(client, prediction)

@sf_kafka.register(
    kafka_topic='client.signed_up',
    output_schema={'predict.topic': schema})
def predict_foo(messages: List[str]) -> dict:
    """Predict XX about a client. ..."""
    clients = [json.loads(m) for m in messages]
    predictions = [create_prediction(c)
                   for c in clients]
    return {'predict.topic': predictions}
```

Completing the production story

The self-service story of how a DS gets a consumer to production

1. Determine the topic(s) to consume.
2. **Write code:**
 - a. *Create a function & decorate it to process events*
 - b. If outputting an event, write a schema
 - c. Commit code to a git repository



my_prediction.py

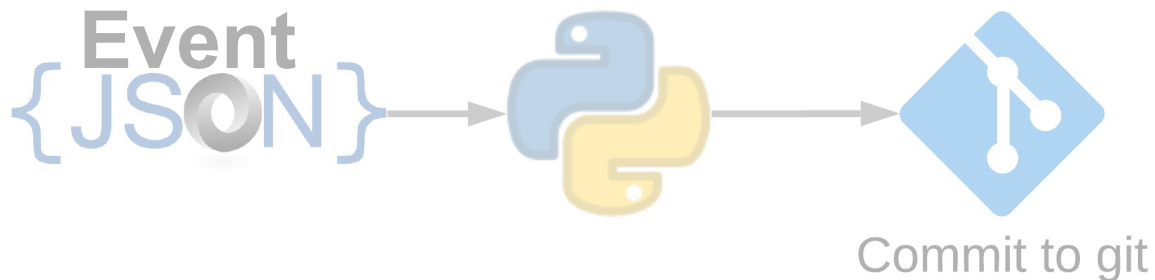
```
def create_prediction(client: dict) -> dict:
    # DS would write side effects or fetches here.
    # E.g. grab features, predict,
    # create output message;
    prediction = ...
    return make_output_event(client, prediction)
```

```
@sf_kafka.register(
    kafka_topic='client.signed_up',
    output_schema={'predict.topic': schema})
def predict_foo(messages: List[str]) -> dict:
    """Predict XX about a client. ..."""
    clients = [json.loads(m) for m in messages]
    predictions = [create_prediction(c)
                   for c in clients]
    return {'predict.topic': predictions}
```

Completing the production story

The self-service story of how a DS gets a consumer to production

1. Determine the topic(s) to consume.
2. **Write code:**
 - a. *Create a function & decorate it to process events*
 - b. If outputting an event, write a schema
 - c. Commit code to a git repository



my_prediction.py

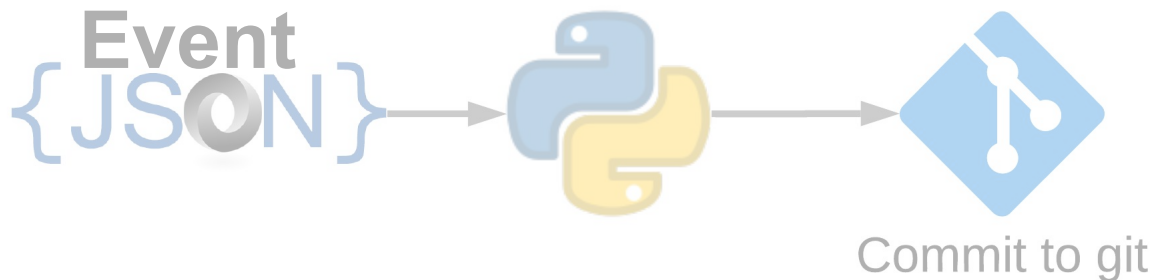
```
def create_prediction(client: dict) -> dict:
    # DS would write side effects or fetches here.
    # E.g. grab features, predict,
    # create output message;
    prediction = ...
    return make_output_event(client, prediction)

@sf_kafka.register(
    kafka_topic='client.signed_up',
    output_schema={'predict.topic': schema})
def predict_foo(messages: List[str]) -> dict:
    """Predict XX about a client. ..."""
    clients = [json.loads(m) for m in messages]
    predictions = [create_prediction(c)
                   for c in clients]
    return {'predict.topic': predictions}
```


Completing the production story

The self-service story of how a DS gets a consumer to production

1. Determine the topic(s) to consume.
2. **Write code:**
 - a. *Create a function & decorate it to process events*
 - b. If outputting an event, write a schema
 - c. Commit code to a git repository



my_prediction.py

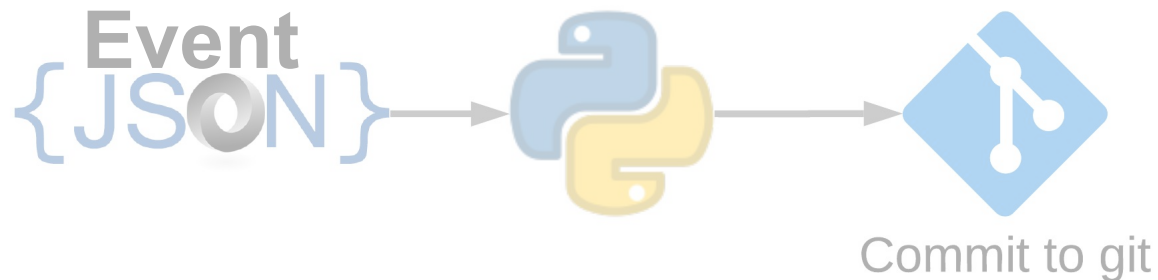
```
def create_prediction(client: dict) -> dict:
    # DS would write side effects or fetches here.
    # E.g. grab features, predict,
    # create output message;
    prediction = ...
    return make_output_event(client, prediction)

@sf_kafka.register(
    kafka_topic='client.signed_up',
    output_schema={'predict.topic': schema})
def predict_foo(messages: List[str]) -> dict:
    """Predict XX about a client. ..."""
    clients = [json.loads(m) for m in messages]
    predictions = [create_prediction(c)
                   for c in clients]
    return {'predict.topic': predictions}
```

Completing the production story

The self-service story of how a DS gets a consumer to production

1. Determine the topic(s) to consume.
2. **Write code:**
 - a. Create a function & decorate it to process events
 - b. *If outputting an event, write a schema*
 - c. Commit code to a git repository



my_prediction.py

```
"""Schema that we want to validate against."""
schema = {
    'metadata': {
        'timestamp': str,
        'id': str,
        'version': str
    },
    'payload': {
        'some_prediction_value': float,
        'client': int
    }
}
```

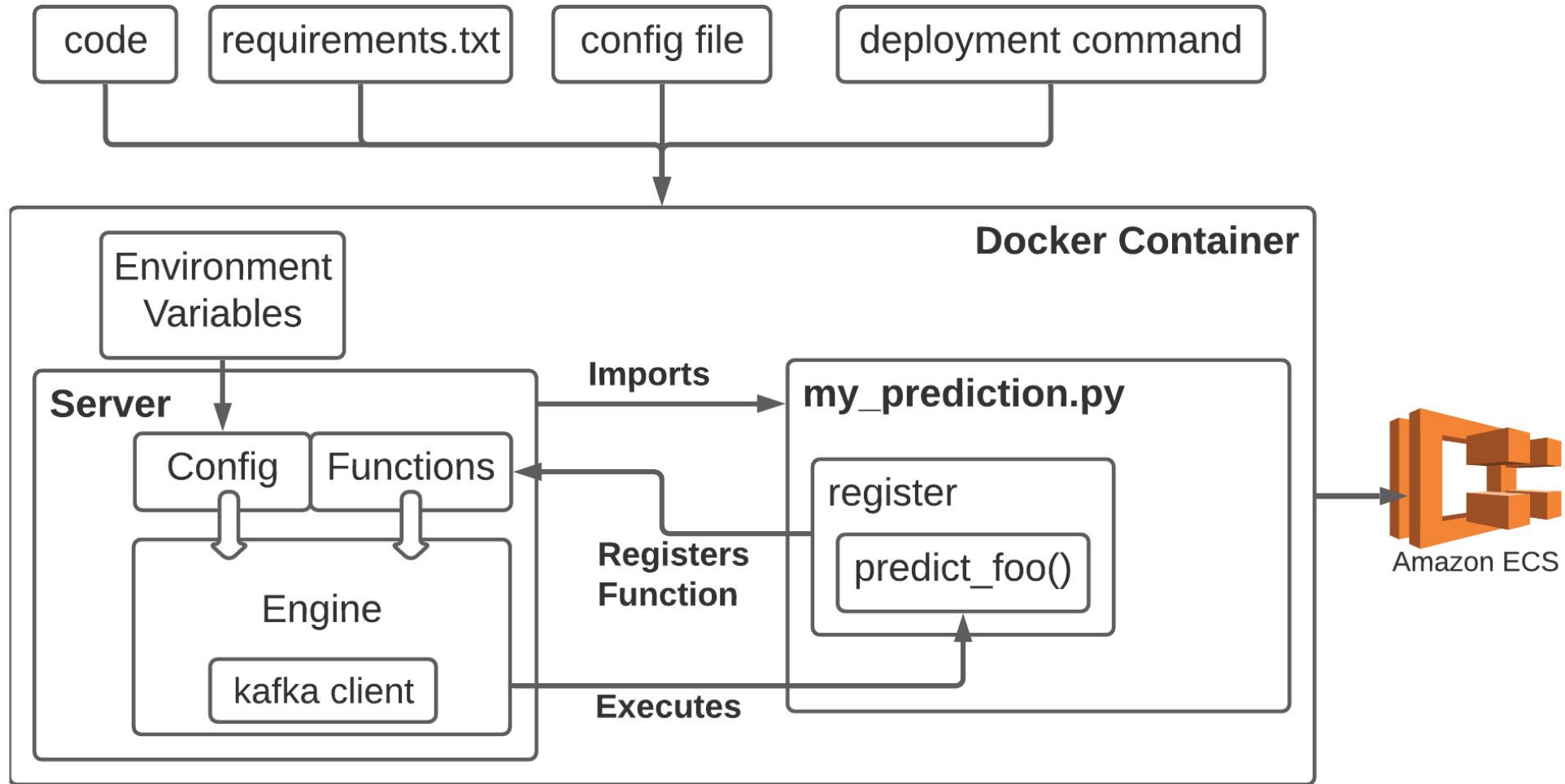
Completing the production story

The self-service story of how a DS gets a consumer to production

1. Determine the topic(s) to consume.
2. Write code
3. **Deploy via command line:**
 - a. Handles python environment creation
 - b. Builds docker container
 - c. Deploys

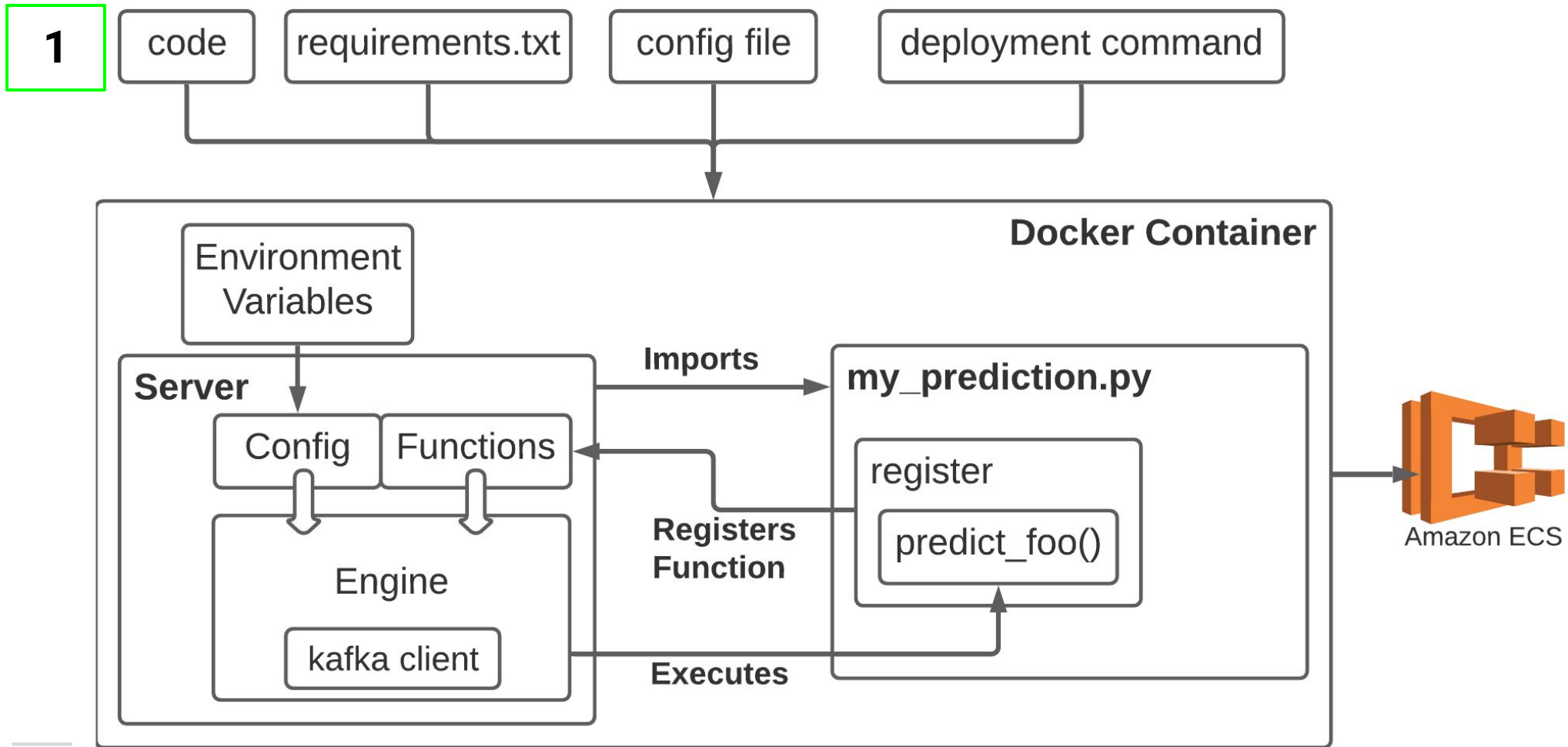
Completing the production story

Self-service deployment via command line



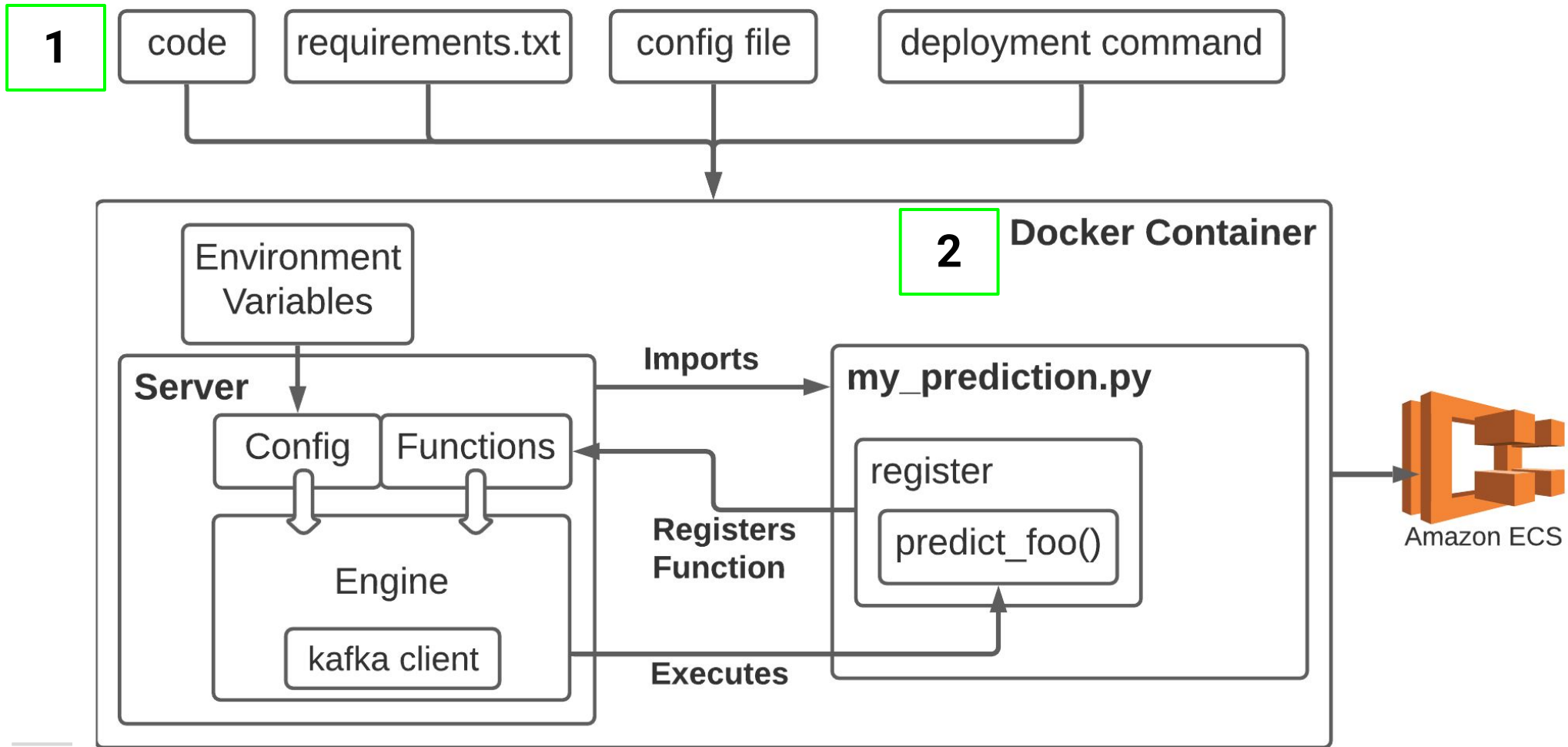
Completing the production story

Self-service deployment via command line



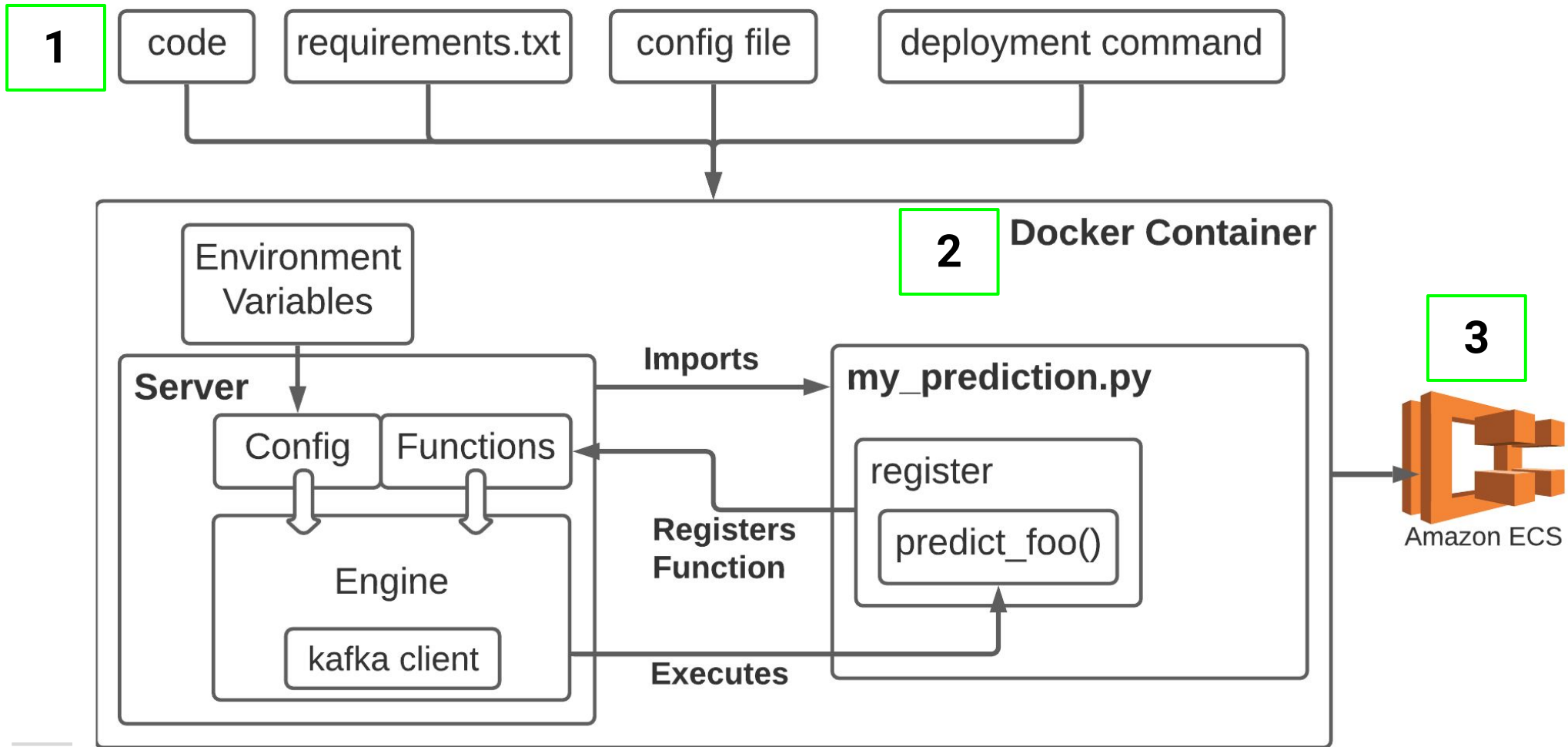
Completing the production story

Self-service deployment via command line



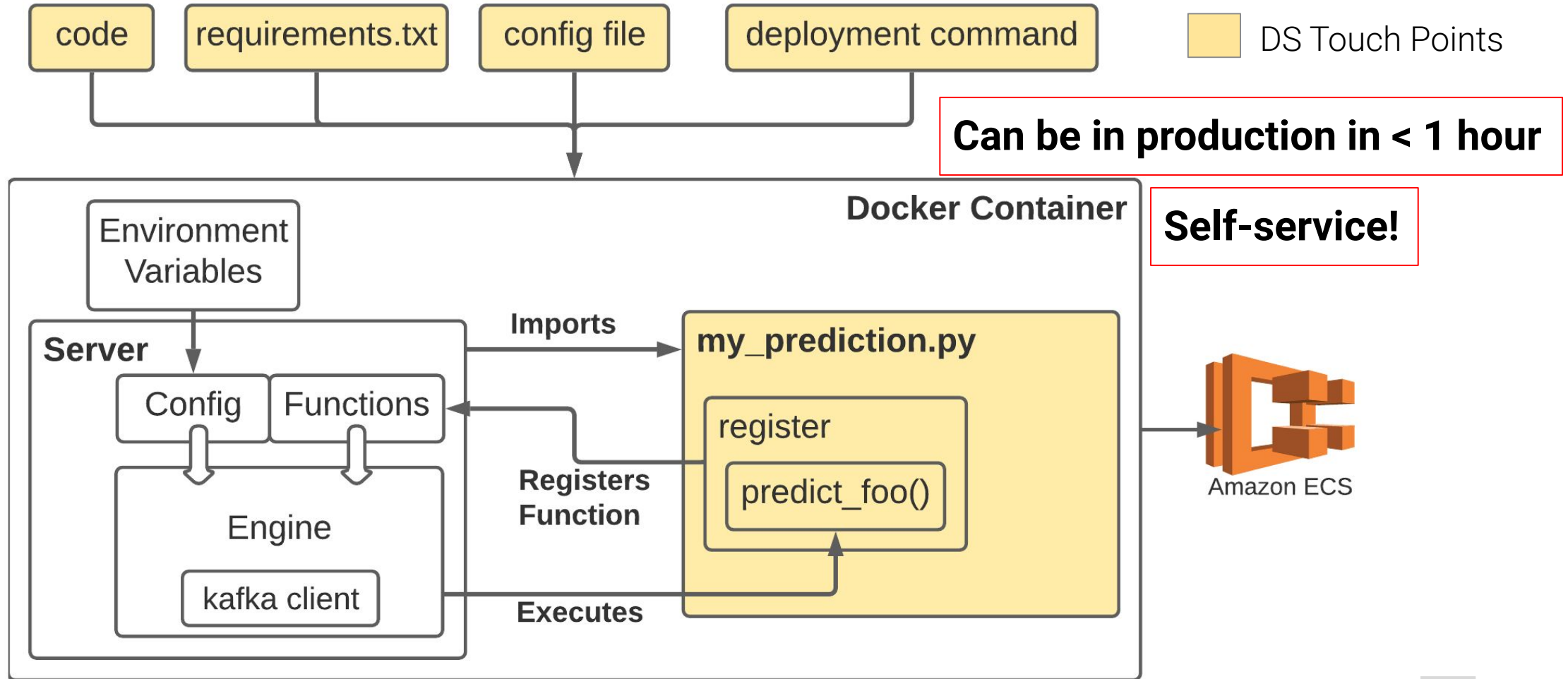
Completing the production story

Self-service deployment via command line



Completing the production story

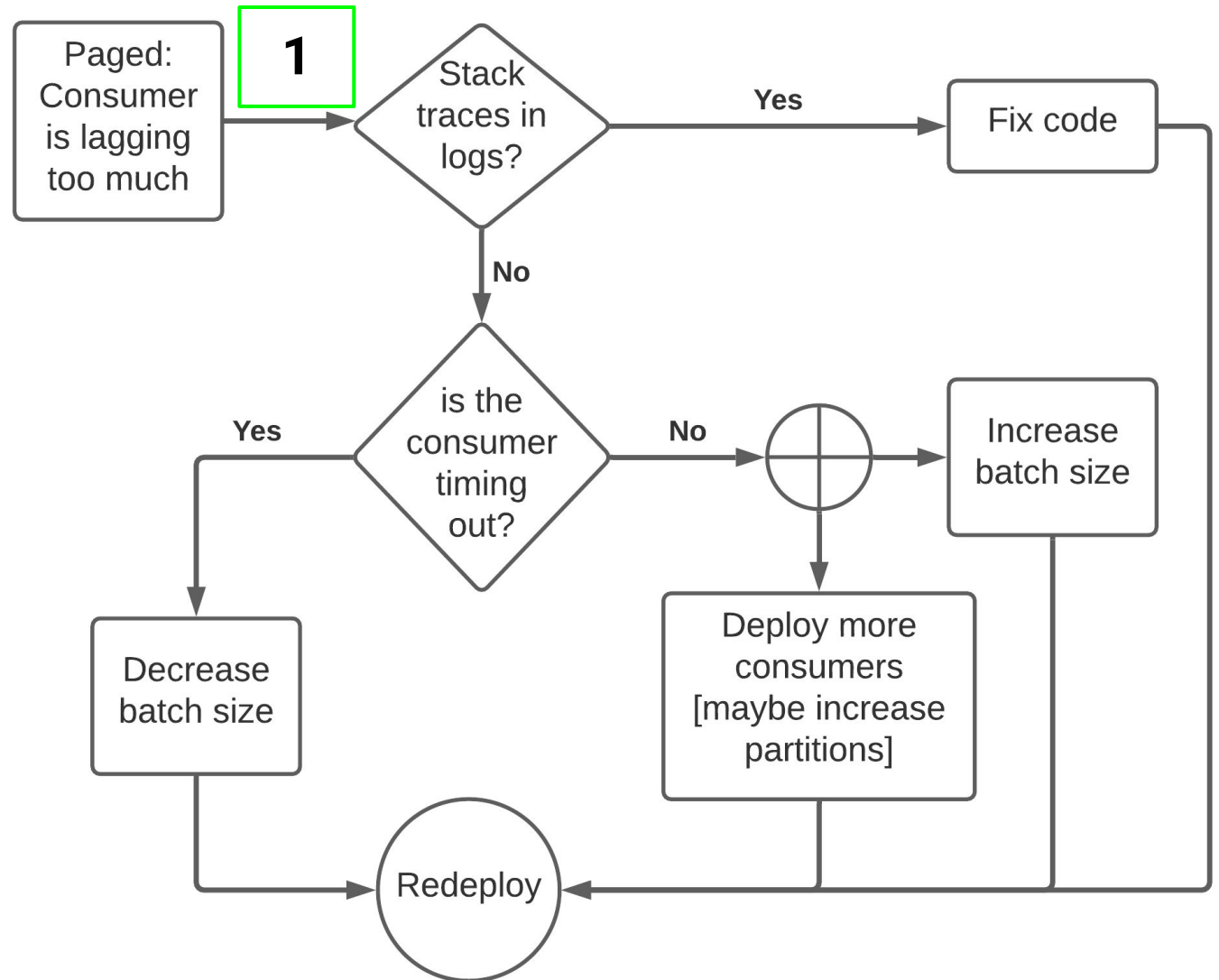
Self-service deployment via command line



Completing the production story

The self-service story of how a DS gets a consumer to production

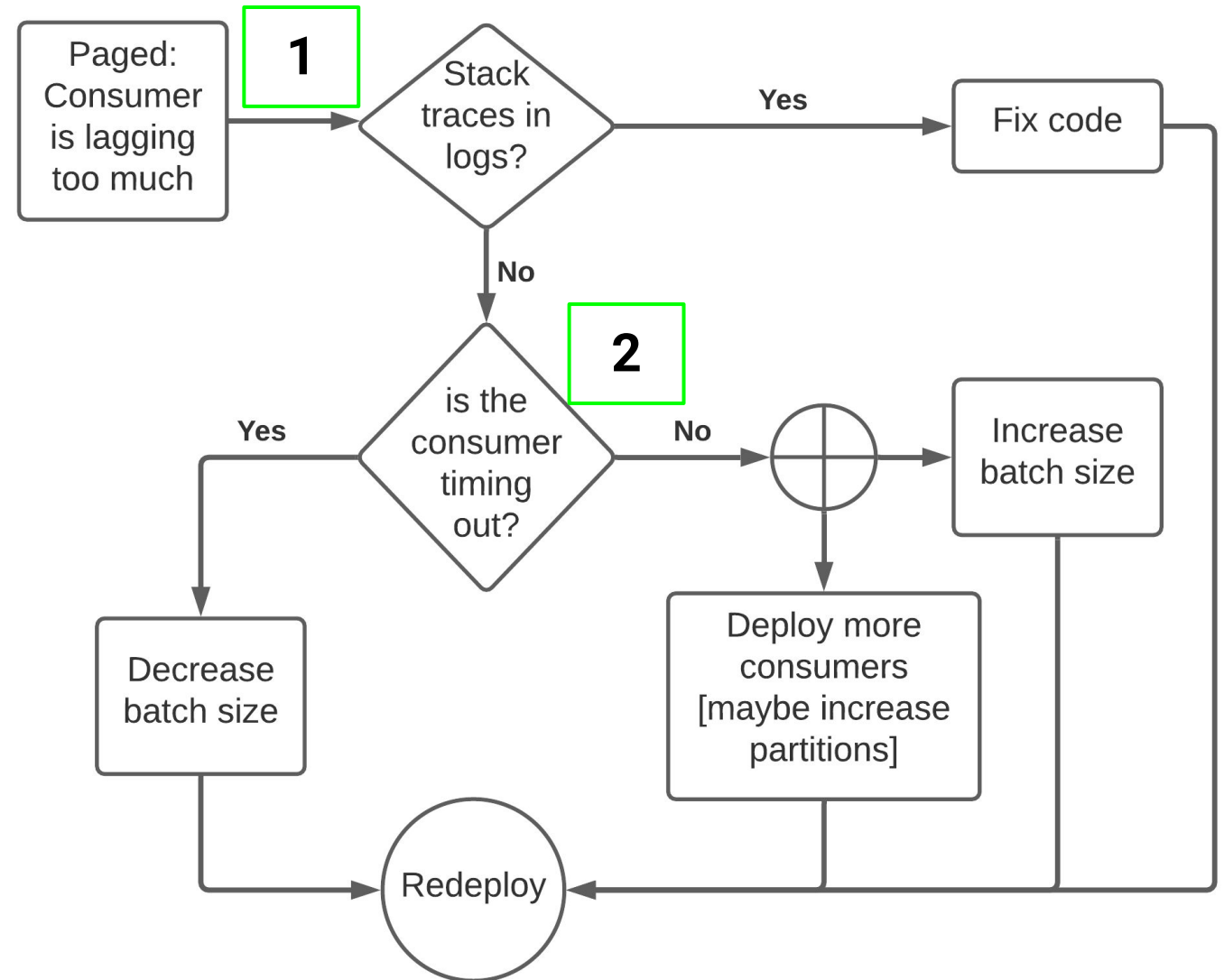
1. Determine the topic(s) to consume.
2. Write code
3. Deploy via command line
4. **Oncall:**
 - a. Small runbook



Completing the production story

The self-service story of how a DS gets a consumer to production

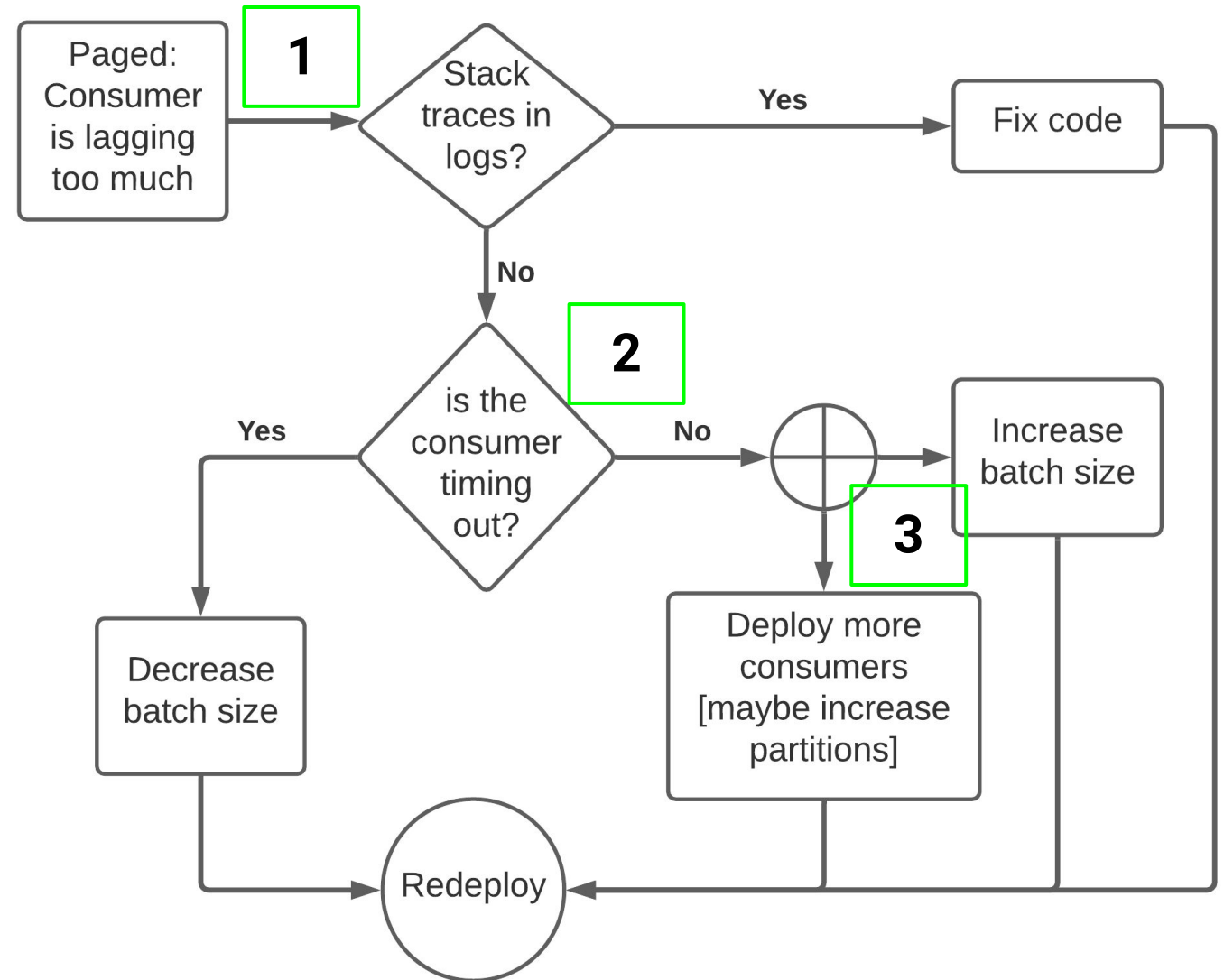
1. Determine the topic(s) to consume.
2. Write code
3. Deploy via command line
4. **Oncall:**
 - a. Small runbook



Completing the production story

The self-service story of how a DS gets a consumer to production

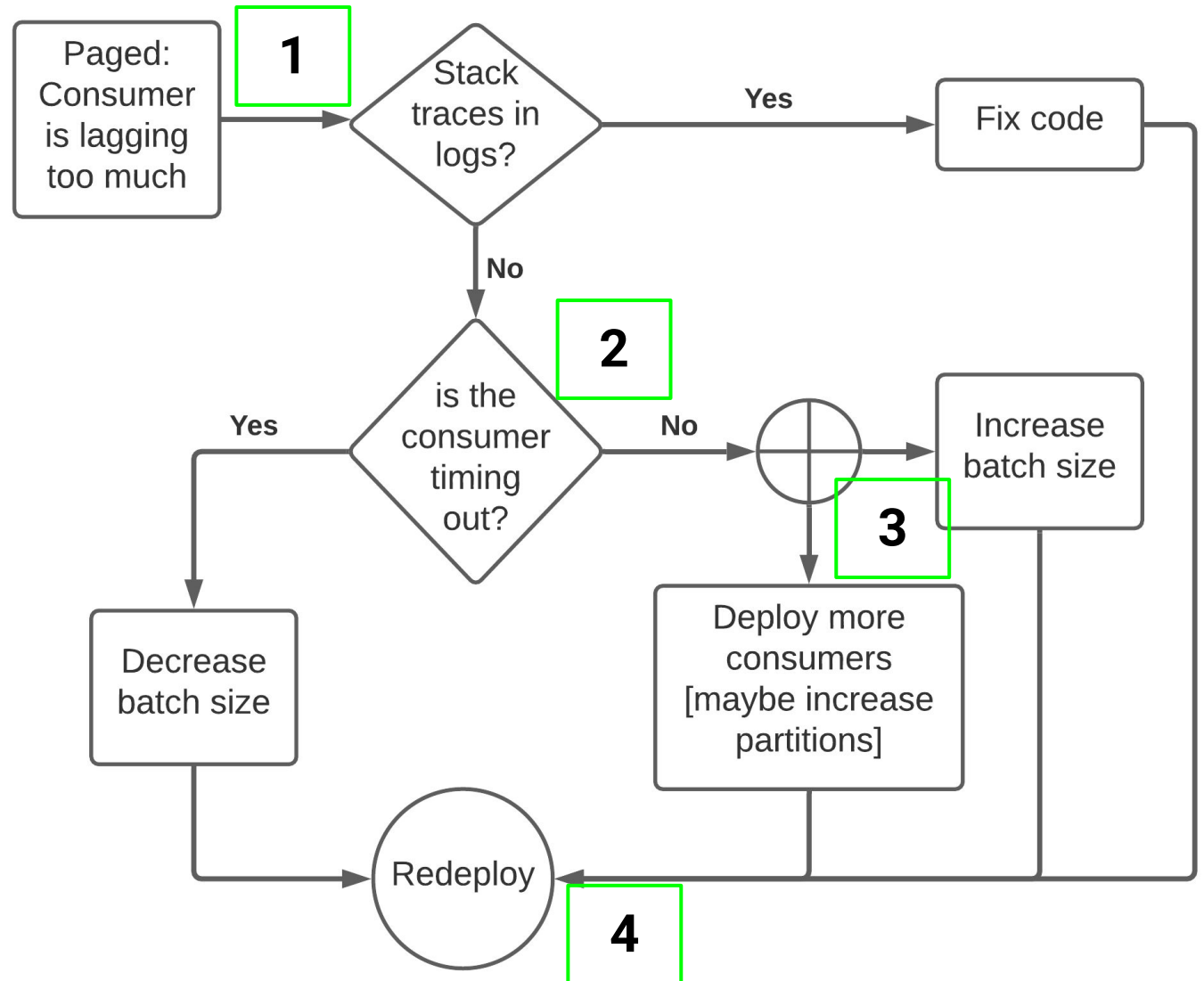
1. Determine the topic(s) to consume.
2. Write code
3. Deploy via command line
4. **Oncall:**
 - a. Small runbook



Completing the production story

The self-service story of how a DS gets a consumer to production

1. Determine the topic(s) to consume.
2. Write code
3. Deploy via command line
4. **Oncall:**
 - a. Small runbook





Learnings & Future Directions

What we learned from this and where we're looking to go.



Learnings - DS Perspective

What?	Learning
--------------	-----------------



Learnings - DS Perspective

What?	Learning
Do they use it?	 

Learnings - DS Perspective

What?	Learning
Do they use it?	 
Focusing on the function	<ol style="list-style-type: none">1. All they need to know about kafka is that it'll give them a list of events.2. Leads to better separation of concerns:<ol style="list-style-type: none">a. Can split driver code versus their logic.b. Test driven development is easy.

Learnings - DS Perspective

What?	Learning
Do they use it?	 
Focusing on the function	<ol style="list-style-type: none">1. All they need to know about kafka is that it'll give them a list of events.2. Leads to better separation of concerns:<ol style="list-style-type: none">a. Can split driver code versus their logic.b. Test driven development is easy.
At least once processing	<ol style="list-style-type: none">1. They enjoy easy error recovery; gives DS time to fix things.2. Idempotency requirement not an issue.

Learnings - Platform Perspective (1/2)

What?	Learning
--------------	-----------------

Learnings - Platform Perspective (1/2)

What?	Learning
Writing back via proxy service	<ol style="list-style-type: none">1. Helped early on with some minor message format adjustments & validation.2. Would recommend writing back directly if we were to start again.<ol style="list-style-type: none">a. Writing back directly leads to better performance.

Learnings - Platform Perspective (1/2)

What?	Learning
Writing back via proxy service	<ol style="list-style-type: none">1. Helped early on with some minor message format adjustments & validation.2. Would recommend writing back directly if we were to start again.<ol style="list-style-type: none">a. Writing back directly leads to better performance.
Central place for all things kafka	<p>Very useful to have a central place to:</p> <ol style="list-style-type: none">1. Understand topics & topic contents.2. Having “off the shelf” ability to materialize stream to a datastore removed need for DS to manage/optimize this process. E.g. elasticsearch, data warehouse, feature store.

Learnings - Platform Perspective (2/2)

What?	Learning
Using internal async libraries	Using internal asyncio libs is cumbersome for DS. Native asyncio framework would feel better.*

* we ended up creating a very narrow focused micro-framework addressing these two issues using aiokafka.

Learnings - Platform Perspective (2/2)

What?	Learning
Using internal async libraries	Using internal async libs is cumbersome for DS. Native async framework would feel better.*
Lineage & Lineage Impacts	If there is a chain of consumers*, didn't have easy introspection into: <ul style="list-style-type: none">● Processing speed of full chain● Knowing what the chain was

* we ended up creating a very narrow focused micro-framework addressing these two issues using aiokafka.

Future Directions

What?	Why?
<p>Being able to replace different subcomponents & assumptions of the system more easily.</p>	<p>Cleaner abstractions & modularity:</p> <ul style="list-style-type: none">• Want to remove leaking business logic into engine.• Making parts pluggable means we can easily change/swap out e.g. schema validation, or serialization format, or how we write back to kafka, processing assumptions, support async, etc.

Future Directions

What?	Why?
Being able to replace different subcomponents & assumptions of the system more easily.	Cleaner abstractions & modularity: <ul style="list-style-type: none">• Want to remove leaking business logic into engine.• Making parts pluggable means we can easily change/swap out e.g. schema validation, or serialization format, or how we write back to kafka, processing assumptions, support async, etc.
Exploring stream processing like kafka streams & faust.	Streaming processing over windows is slowly becoming something more DS ask about.

Future Directions

What?	Why?
Being able to replace different subcomponents & assumptions of the system more easily.	Cleaner abstractions & modularity: <ul style="list-style-type: none">• Want to remove leaking business logic into engine.• Making parts pluggable means we can easily change/swap out e.g. schema validation, or serialization format, or how we write back to kafka, processing assumptions, support asyncio, etc.
Exploring stream processing like kafka streams & faust.	Streaming processing over windows is slowly becoming something more DS ask about.
Writing an open source version	Hypothesis that this is valuable and that the community would be interested; <i>would you be?</i>

Summary

TL;DR:

STITCH FIX

Summary

TL;DR:

Kafka + Data Scientists @ Stitch Fix:

- We have a self-service platform for Data Scientists to deploy kafka consumers
- We achieve self-service through a separation of concerns:
 - Data Scientists focus on functions to process events
 - Data Platform provides guardrails for kafka operations

Questions?

Find me at:

[@stefkrawczyk](#)

[linkedin.com/in/skrawczyk/](#)

Try out Stitch Fix → goo.gl/Q3tCQ3

STITCH FIX