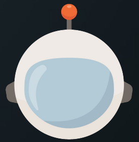
Using Async API schema to define event driven architecture with AWS SNS



Presented by

Ayush Goyal

Software Engineer II
Integrations Squad,
Postman

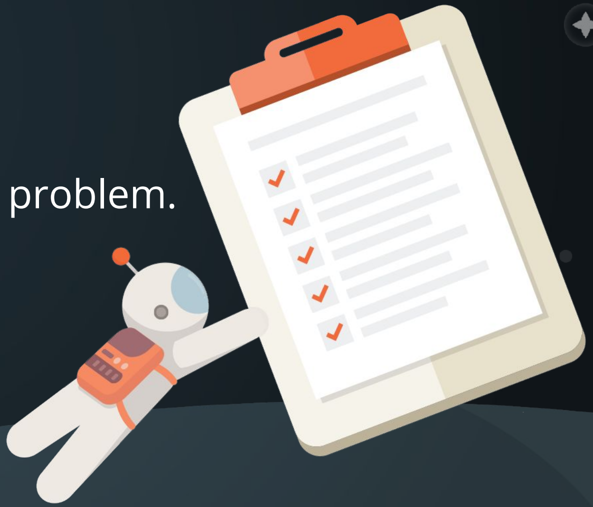
Agenda

1 | What is an event and event-driven architecture

2 | What is AWS Simple Notification Service(SNS)

3 | Use case we had at Postman

4 | How Async API helped us to quickly solve our problem.



What is an Event?

What is Event-Driven Architecture?

1 | Using events for communication

2 | Components of event-driven architecture

- Event Producer
- Event Consumer
- Event Router

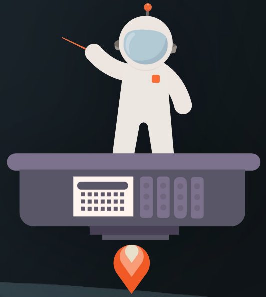
3 | Decoupled producer and consumer

Benefits of Event-Driven Architecture?

1 | Parallel Processing

2 | Versatility in choosing technical stack

3 | Hassle free cross-team dependencies

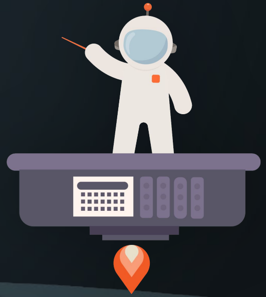


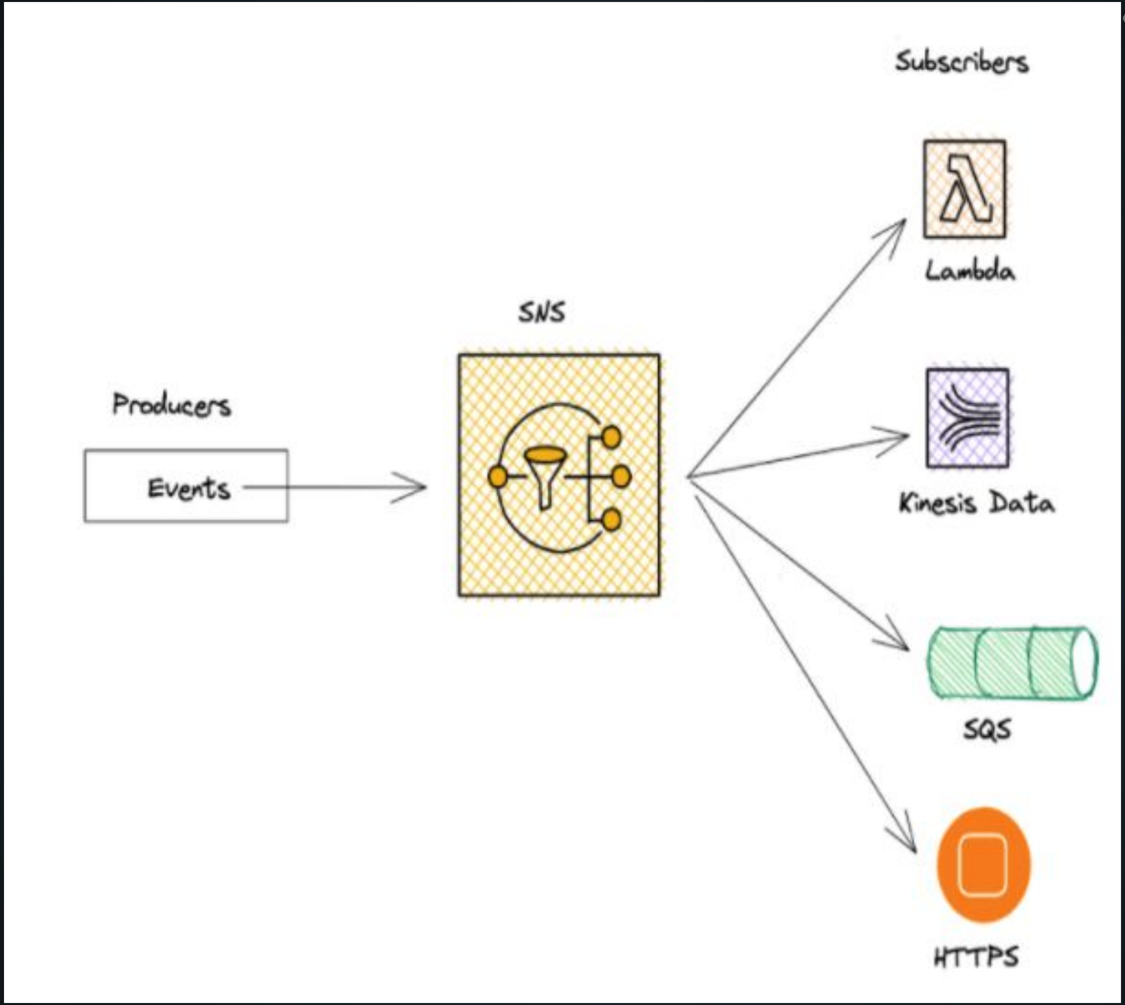
What is SNS (Simple Notification Service) ?

1 | An Event router which provides message delivery

2 | In-built attribute based message filtering mechanism

3 | How the communication works?




Subscribers
Lambda
SNS
Producers
Events
Kinesis Data
SQS
HTTPS

How does the message filtering work on SNS?

1 | Boolean logic - it either matches filter policy, or not

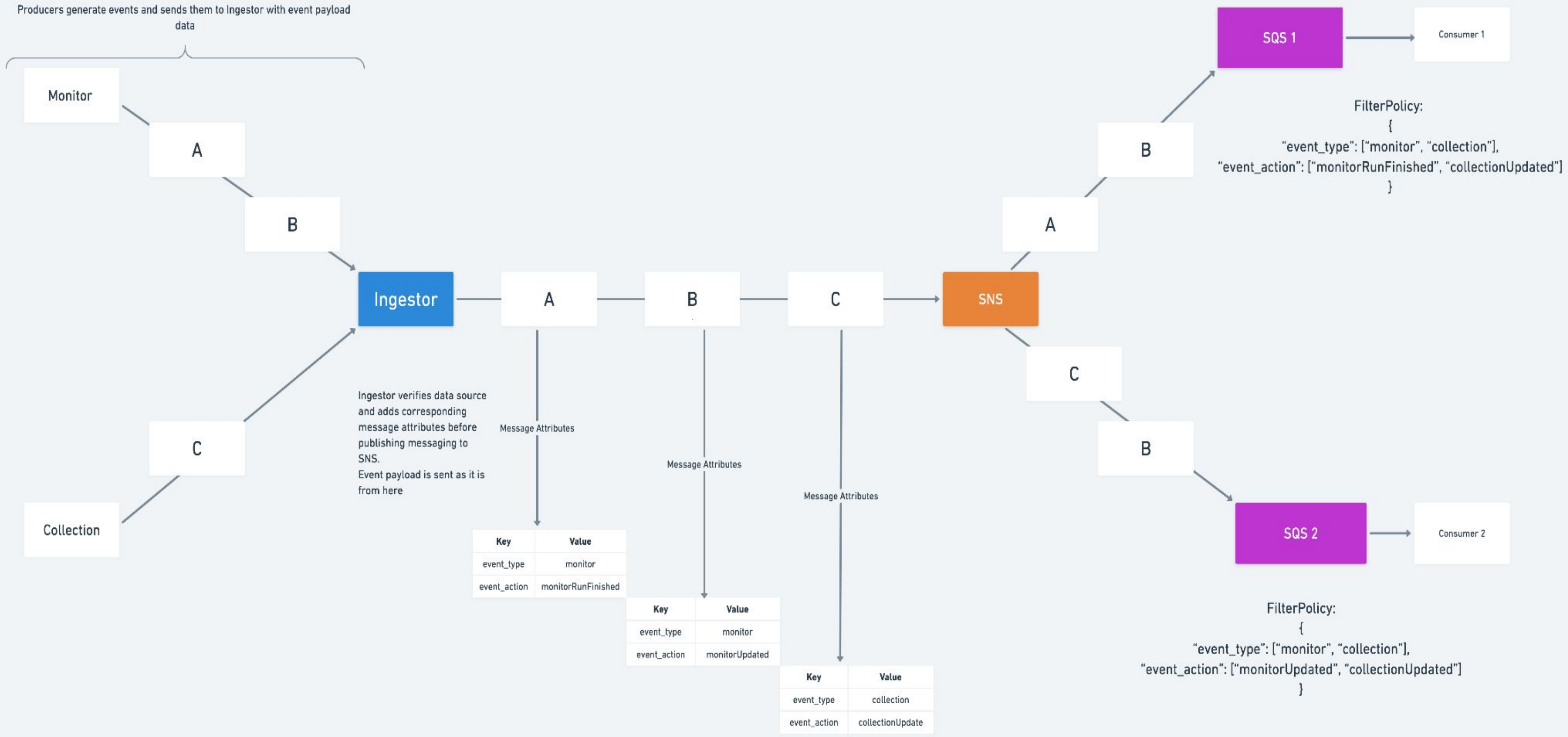
2 | For it to match a message, the message must contain all the attribute keys listed in the policy.

3 | Attributes of the message not mentioned in the filtering policy are ignored.

4 | The matching is exact (character-by-character), without case-folding or any other string normalization.

5 | Number matching is at the string representation level. 300 != 300.0

How attribute based message filtering works at SNS



Ingestor verifies data source and adds corresponding message attributes before publishing messaging to SNS.
Event payload is sent as it is from here

Key	Value
event_type	monitor
event_action	monitorRunFinished

Key	Value
event_type	monitor
event_action	monitorUpdated

Key	Value
event_type	collection
event_action	collectionUpdate

```
FilterPolicy:
{
"event_type": ["monitor", "collection"],
"event_action": ["monitorRunFinished", "collectionUpdated"]
}
```

```
FilterPolicy:
{
"event_type": ["monitor", "collection"],
"event_action": ["monitorUpdated", "collectionUpdated"]
}
```

Use case at Integrations Squad in Postman

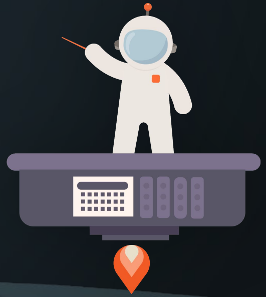
1 | Get in-flow of event messages from various other teams

2 | Process those messages to trigger relevant integrations

3 | The service should be completely decoupled

4 | Adding new service which can start publishing events should be easy and fast

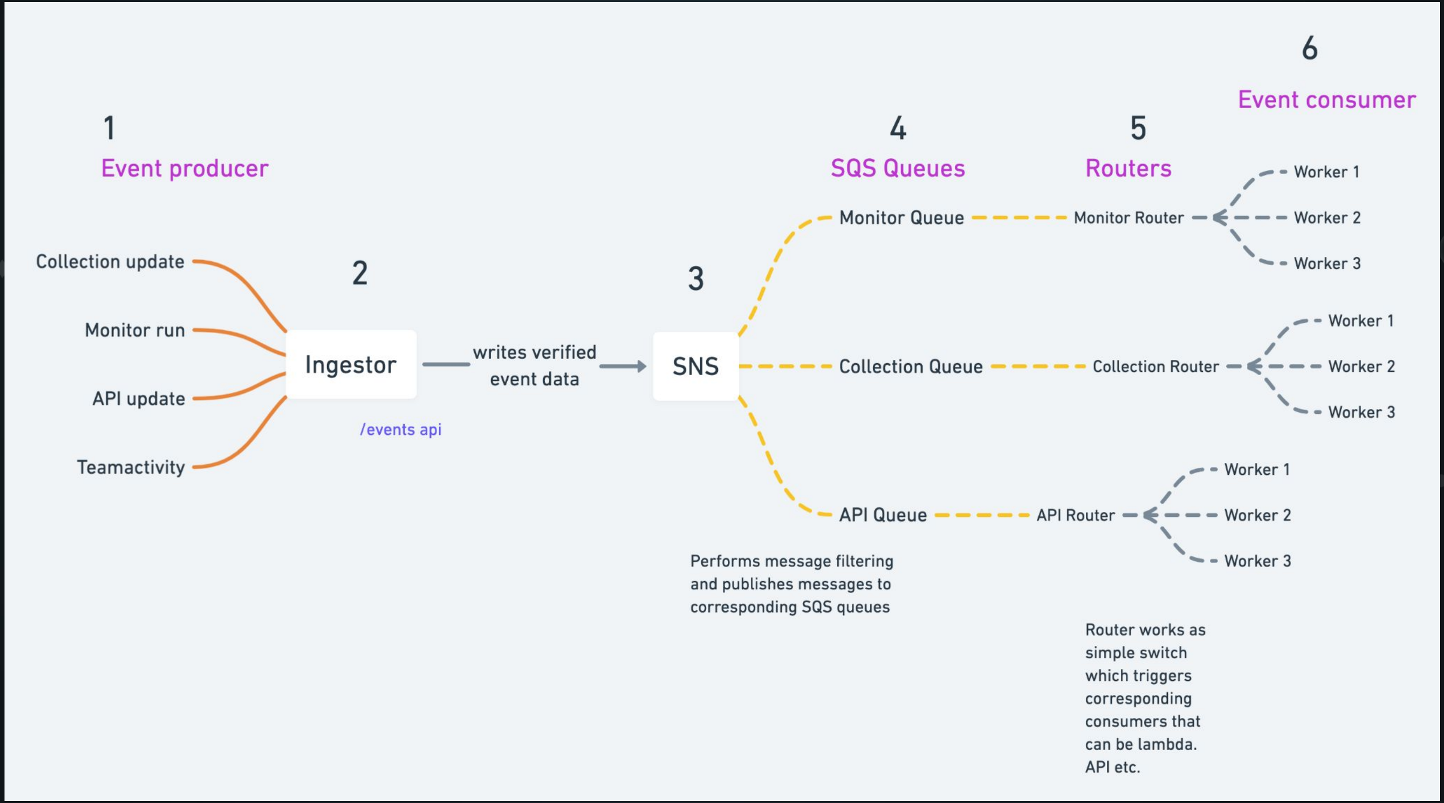
5 | A new team should be able to start consuming these events easily



Different approaches for the solution

Approach 1

Use SNS to publish to squad specific queues and then forward events to consumer

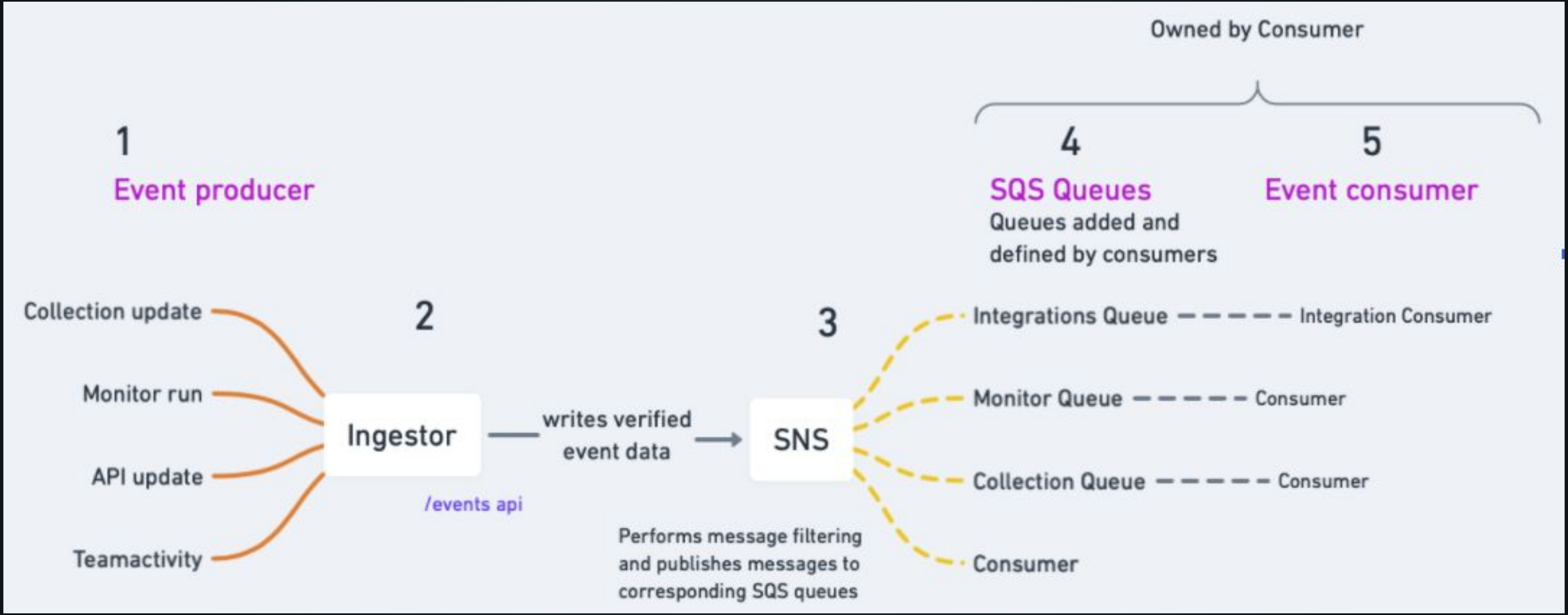


Performs message filtering and publishes messages to corresponding SQS queues

Router works as simple switch which triggers corresponding consumers that can be lambda. API etc.

Finalised Approach

Consumer to own complete infrastructure which connects to SNS

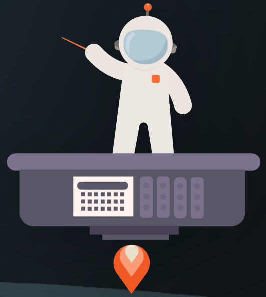


How Async API helped us

1 | Made it easy to review the payload and event structures

2 | Allowed both the involved teams to start working parallely with 100% confidence

3 | Provided a one-stop repository for future teams to know the event structure to easily start event consumption for new use cases



What if - teams don’t use Async API

Defining essential components of the Async API Schema

Servers

```
servers:
 production:
   url: <production url>
   protocol: https
   description: <small description>
   security:
     - basic_auth: []
 beta:
   url: <beta testing url>
   protocol: https
   description: <small description>
   security:
     - basic_auth: []
```

Security

```yaml
components:
  securitySchemes:
    basic_auth:
      type: userPassword
      description: <description>
```

Channel & Schemas

- **Channel Object:**
  - Holds the relative paths to the individual channel and their operations
  - The path will be relative to server
  - Also known as "topics", "routing keys", "event types" or "paths".

```
user/signedup:
 subscribe:
   message:
     $ref: "#/components/messages/userSignedUp"
```

- **Schema Object**:
  - Allows the definition of input and output data types
  - Can be objects but also primitives and arrays
  - Can be used as value in reference object

```
$ref: '#/components/schemas/Pet'
```

Thank You

defcon_007

DefCon-007

https://www.defcon007.com/