Amazon SNS – Pub/Sub Messaging

Full Form: Simple Notification Service
Type: Fully Managed Pub/Sub (Publish–Subscribe) Messaging Service

Amazon SNS is a fully managed messaging service used to send notifications or messages between applications, services, or users in a scalable and reliable way.

Core Concepts

- **Topic:**
  A logical access point through which messages are published.
  Publishers send messages to a topic, and all subscribers to that topic receive the messages.

- **Publisher:**
  An entity (application, service, or user) that publishes messages to the topic.
  Example: An application that sends order status updates after a purchase.

- **Subscriber:**
  An endpoint or service that receives messages from the topic.
  Subscriptions can be configured for various protocols like:

    - AWS Lambda (trigger functions)
    - Amazon SQS (queue messages)
    - HTTP/HTTPS endpoints (webhooks)
    - Email or SMS (notifications to users)

- **Message:**
  The content or data sent by publishers through a topic to subscribers.

Key Features

- **Fully managed service:** No need to provision or maintain servers.
- **Scalable:** Can handle millions of messages per second.

- **Reliable delivery:** Delivers messages across multiple endpoints.
- **Filtering:** Subscribers can apply message filters to receive only relevant data.
- **Security:** Integrated with IAM policies for fine-grained access control.
- **Durability:** Messages are stored redundantly across multiple Availability Zones.
- **Fan-out pattern:** One published message can be delivered to multiple subscribers at once.

Example Scenarios

1. **Order Processing (E-commerce):**
    - A customer places an order.
    - The order service publishes the order details to an *OrderTopic*.
    - Subscribers:
        - Inventory service (updates stock).
        - Billing service (initiates payment).
        - Notification service (sends order confirmation email/SMS).
2. **System Monitoring:**
    - CloudWatch alarm triggers when CPU usage exceeds 80%.
    - The alarm publishes a message to *MonitoringTopic*.
    - Subscribers:
        - DevOps email team (for alerts).
        - Lambda function (auto-scale EC2 instance).
3. **IoT Device Updates:**
    - IoT sensors publish temperature data to *SensorTopic*.

    - Subscribers:
        - Data analytics service (stores readings in DynamoDB).
        - Mobile app (shows live temperature updates).
4. **Application-to-Application Communication:**
    - Microservices can communicate asynchronously using SNS.
    - One service publishes a message; others subscribe to perform further tasks.

Message Publishing Example (CLI representation)

text

```
aws sns publish --topic-arn
arn:aws:sns:us-east-1:123456789012:MyTopic \
--message "Order ID 125 confirmed for shipment"
```

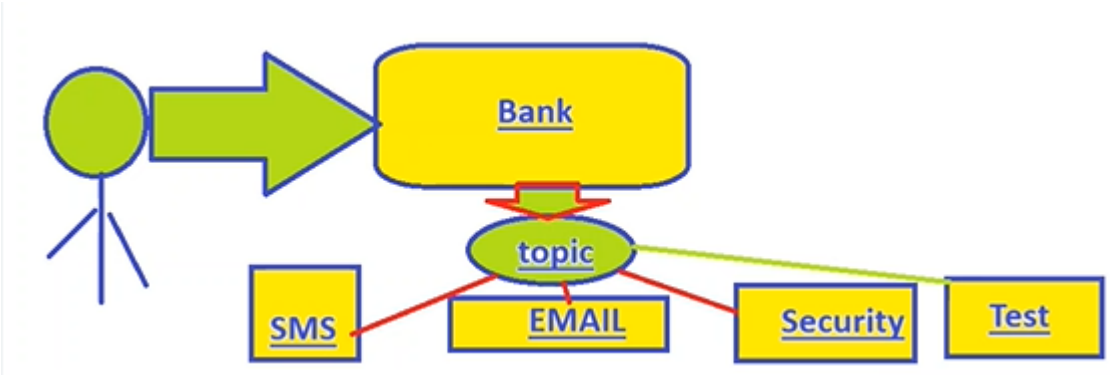
Subscription Example

- Subscribers can register like:
    - HTTP endpoint: `https://myapp.example.com/order-notifications`
    - Email: `customer@domain.com`
    - Lambda: `arn:aws:lambda:us-east-1:123456789012:function:NotifyUser`

Typical Workflow Summary

1. Create a Topic (e.g., `MyTopic`).
2. Add one or more Subscribers (Email, Lambda, HTTP, SQS, etc.).
3. Send a message to the Topic using a Publisher.

4. Subscribers automatically receive and process the message.



Steps to Use Amazon SNS: Create Topic, Subscribe Email, and Send Message

---

Step 1: Create an SNS Topic

- Navigate to the **Amazon SNS console** on AWS.
- Click on **Topics** in the sidebar.
- Click **Create topic**.
- Choose type:
    - Standard (for most use cases, supports multiple subscribers and high-throughput).
- Enter a **topic name** (e.g., `MyTopic`).
- (Optional) Add tags or configure settings if needed.
- Click **Create topic**.

Alternatively, via AWS CLI:

text
```
aws sns create-topic --name MyTopic
```

Step 2: Create a Subscription (Email with JSON Protocol)

- After the topic is created, select the topic from the SNS dashboard.
- Click **Create subscription**.
- Configure the subscription:
    - Protocol: Select **Email-JSON** (This delivers emails containing the message data in JSON format).
    - Endpoint: Enter the email address where notifications will be sent (e.g., your.email@example.com).
- Click **Create subscription**.
- The subscriber (email recipient) will receive a confirmation email.
- The recipient **must confirm the subscription** by clicking the confirmation link inside that email to activate the subscription.

Step 3: Publish a Message to the Topic

Using AWS Management Console:

- Go to the topic details page.
- Click **Publish message**.
- Enter subject (e.g., "Test Notification").
- Enter message body (may include text or JSON structure depending on use case).
- Click **Publish**.

Using AWS CLI to send message in JSON format:

text

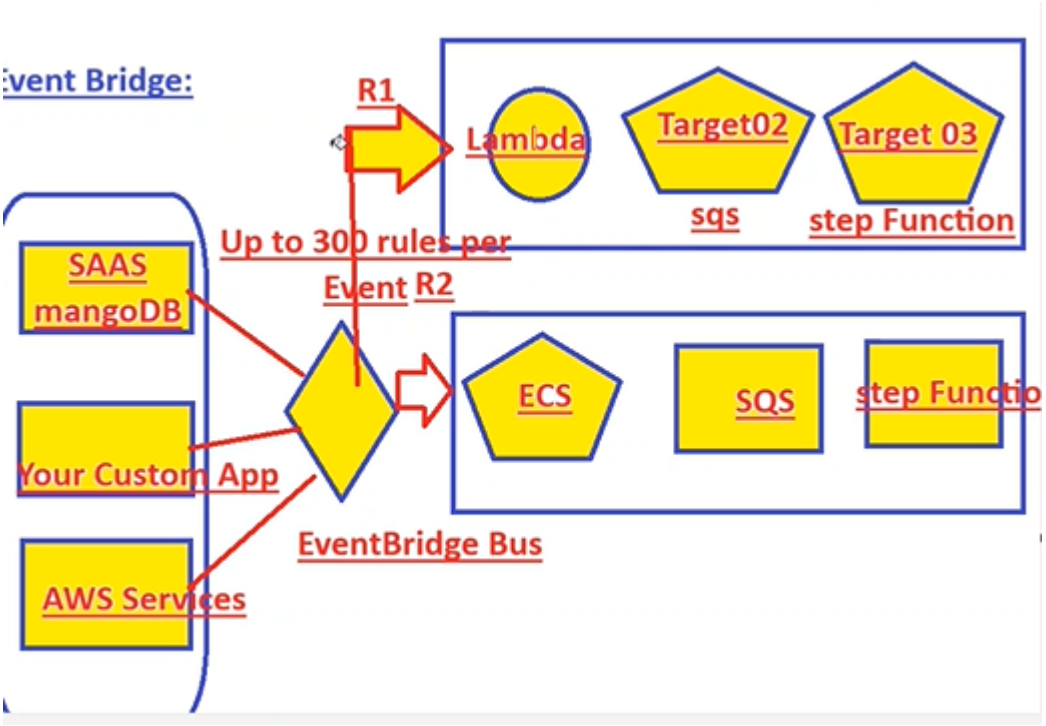
```
aws sns publish --topic-arn arn:aws:sns:region:account-id:MyTopic \
```

```
--message '{"default":"This is the default message","email":"This
message will be in JSON format"}' \
--message-structure json
```

- The email recipient will receive the message as a JSON payload, showing the message structure in detail.

Important Notes

- **Email-JSON protocol** sends the entire JSON message structure as content in the email, allowing for richer data representation.
- The email must be verified by the recipient to start receiving messages.
- Messages can be sent in different formats using `MessageStructure` parameter: `json` or `string`.
- Topics can have multiple subscribers with different protocols simultaneously.
- You can manage subscriptions (unsubscribe, confirm) via the SNS console or CLI.



AWS EventBridge Overview (From the Image)

EventBridge is an event bus service that allows SaaS (e.g., managed databases like MongoDB), custom applications, and AWS services to send and process real-time event data across your AWS environment.

Main Components in the Image

1. Event Sources

- **SAAS (mangoDB)**, **Your Custom App**, and **AWS Services** are all event sources.
- These systems publish events (such as data changes, status updates, or notifications) to the EventBridge bus.

2. EventBridge Bus

- The EventBridge bus acts as a central hub that collects all the incoming events.
- Each event is routed based on specific rules set on the bus.

3. Rules (Event R1, Event R2)

- You can define up to **300 rules per event bus**.
- Each rule examines events arriving at the event bus, filters them based on patterns (such as source, type), and routes them to specified targets.
    - **R1:** One such rule that targets Lambda, Target02 (SQS), and Target03 (Step Function).
    - **R2:** Another rule that routes events to ECS, SQS, and Step Function.

4. Targets

- Once a rule matches an event, the event is sent to one or more targets, such as:
    - **Lambda:** Serverless compute to process or transform events.
    - **ECS:** Run containers in response to events.
    - **SQS:** Store events in queues for later processing.

    - **Step Function:** Orchestrate workflows in response to events.
    - **Target02/Target03:** Represent other services like SQS/Step Functions for respective rules.

5. Event Flow Example

- Events from any source (e.g., SAAS app or custom app) enter the EventBridge Bus.
- Each rule (R1, R2) checks if the event matches its pattern.
- If it matches, the event is forwarded to all associated targets (multiple destinations supported for each rule).
- Example: A change in a MongoDB document (sent via SaaS) could simultaneously start a Lambda function, trigger a Step Function, send a message to SQS, and process something in ECS depending on the rule.

Key Points

- **Fan-out Architecture:** With rules and multiple targets, a single event can start workflows in various AWS services in parallel.
- **Automation & Decoupling:** EventBridge enables systems to operate independently, reacting to events without direct integration.
- **Scalability:** Supports complex event-processing and automation with minimal management.

Example Scenarios

- Database update in SaaS MongoDB triggers an alert (via Lambda) and stores a backup in SQS.
- Custom app pushes order completed event; EventBridge routes it for email notification (Lambda), records it in a data warehouse (ECS), and updates analytics pipeline (Step Function).

EventBridge Bus: Key Features

1. Content-Based Filtering

- Allows events to be filtered and routed to targets based on the content of the event message (such as field values, prefixes, suffixes, numeric comparisons, and more).aws.amazon+2

- You use **event patterns** defined in JSON to specify which events should trigger each rule.

- Example: Route only “order completed” events or only those from a particular service or with a specific detail.

- Supports advanced filtering options:

    - Prefix and suffix matching (example: a file ending in `.png`)

    - Numeric value matching and ranges

    - “Anything-but” (exclude specific values)

    - IP address or existence-based filtering

---

2. Event Archives, Replays, and Schema Registry

- **Event Archives:** Enables continuous archiving of events arriving on the bus for later auditing, debugging, or compliance use.

- You can set up an archive to retain all or filtered event data – these are immutable and can be stored for long durations.

- **Replay (Payback):** Archived events can be replayed to targets at any time, enabling reprocessing after correcting bugs or adding new features, without losing historical data.

- **Schema Registry:** Automatically catalogs the structure (schema) of events as they flow through the bus.

    - Developers can discover event formats and generate strongly-typed code bindings, accelerating event-driven application development.

---

3. Dead Letter Queues (DLQ) Per Target

- Each target of an EventBridge rule can be configured with its own **Dead Letter Queue** (DLQ), typically using SQS or SNS as the destination.dev
- If an event fails to be delivered to a target (due to errors, permission issues, or unavailability), it will be sent to the DLQ for later inspection or manual processing.
- This helps avoid data loss, ensures reliability, and aids in troubleshooting event failures on a per-target basis.

CONFIGURATION

1. Content-Based Filtering (Event Patterns)

How to Configure:

- In the AWS EventBridge console, create or edit a rule on your event bus.
- In the “Event pattern” section, define a JSON filter specifying attributes to match.
- Example: Match only ‘order completed’ events from your app.

Sample Event Pattern (JSON):

json

```
{
  "source": ["my.custom.app"],
  "detail-type": ["order-completed"],
  "detail": {
    "orderStatus": ["SUCCESS"],
    "amount": [{ "numeric": [">=", 1000] }],
    "customerEmail": [{ "prefix": "vip-" }]
  }
}
```

- Filter by event source, detail-type, and inner values.
- Operators include: `prefix`, `suffix`, `exists`, and numeric comparisons.aws.amazonyoutube

---

2. Event Archives and Replay

Archiving:

- In EventBridge, select “Archives” from the console menu.
- Click “Create Archive.”
    - Name and describe the archive.
    - Select the event bus.
    - Set retention (default: 90 days, max: 7 years).
    - Optionally, specify a filter pattern to archive only certain events.aws.amazon+2

Replay (Payback):

- After archiving, you can replay events for troubleshooting or reprocessing.
- In the console, go to “Replays.”
- Click “Start Replay,” select archive and rules, and specify time window.
- Events are re-injected to the bus and processed by matching rules; replayed events include a “replay-name” field in metadata for identification.aws.amazon+2

---

3. Dead Letter Queue (DLQ) Per Target

How to Configure:

- When creating or editing a rule, in the “Target” configuration, enable the DLQ option.

- Choose an SQS queue as the DLQ destination.
- EventBridge will automatically send undelivered or failed events for a specific target to the chosen DLQ.

DLQ Setup Example:

1. Create a target SQS queue (e.g., `myEventDLQ`).
2. While adding a target in EventBridge rule, specify this SQS queue under DLQ section.
3. Ensure permissions allow EventBridge to use `SendMessage` for the DLQ.[aws.amazon+1]

Permissions Example (SQS resource-policy):

json

```
{
  "Effect": "Allow",
  "Principal": { "Service": "events.amazonaws.com" },
  "Action": "sqs:SendMessage",
  "Resource": "arn:aws:sqs:us-west-2:123456789012:myEventDLQ"
}
```