
AWS
re:Invent
NOV. 28 – DEC. 2, 2022 | LAS VEGAS, NV

API307-R

Designing event-driven integrations using Amazon EventBridge

Stephen Liedig (he/him)

Principal Serverless Specialist Solutions Architect
Amazon Web Services

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Agenda

What we've learned about multi-account topologies

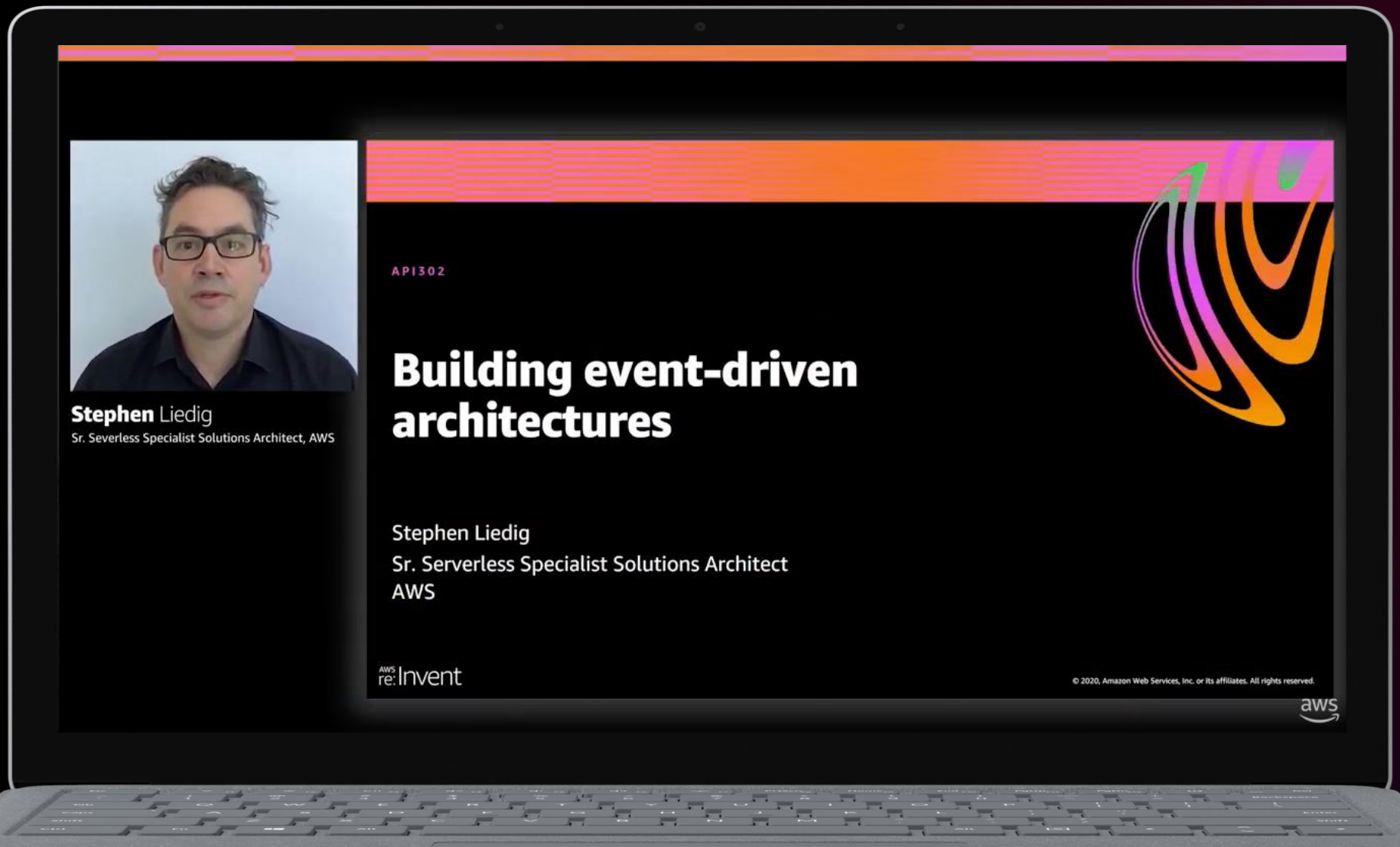
Coupling and events

Roles and responsibilities in an event-driven architecture

Designing a Serverless, event-driven event bus using Amazon EventBridge

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Previously on “Building event-driven architectures with Amazon EventBridge”



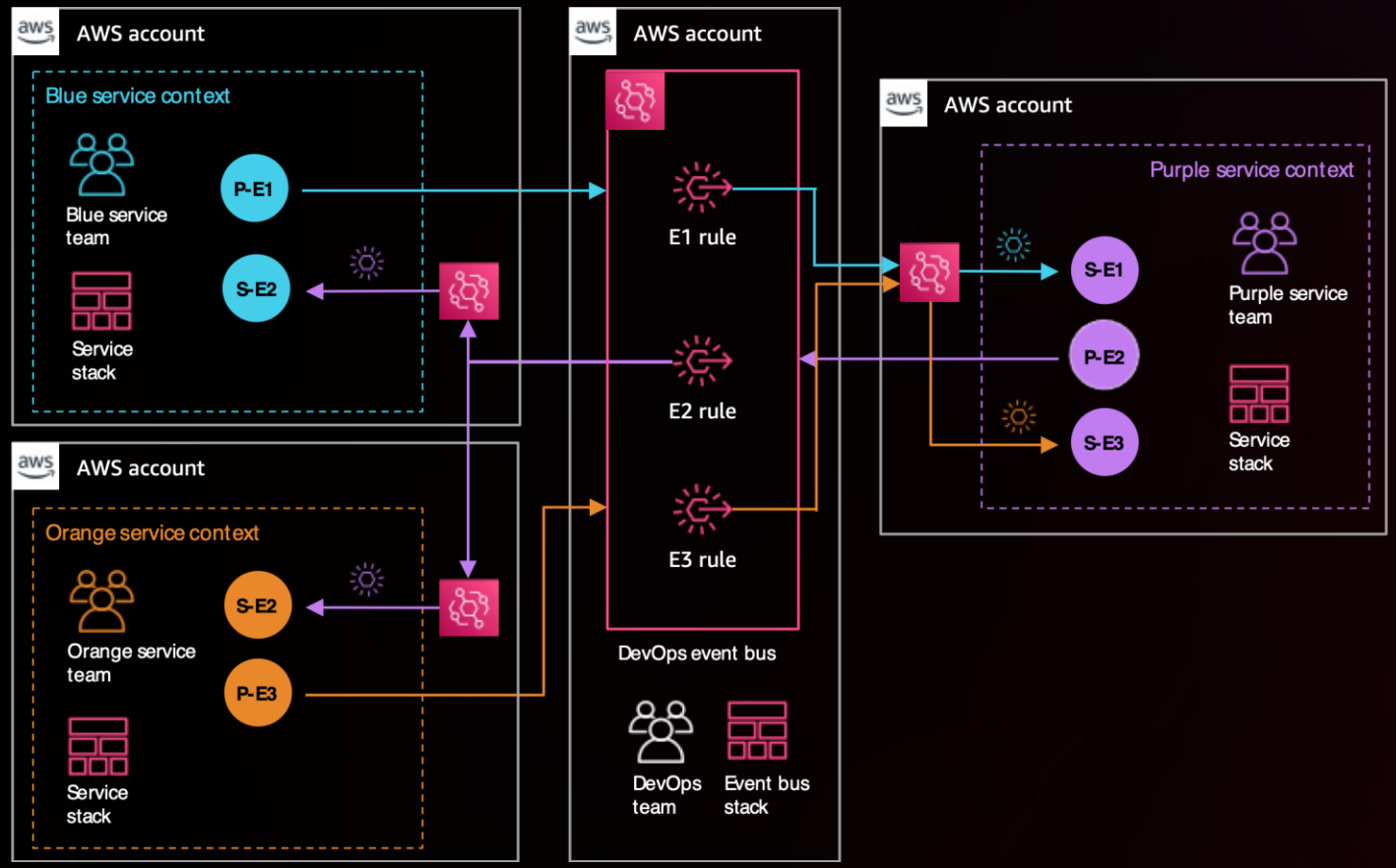
Which topology should I choose?

- Single event bus in a single account?
- Multiple event buses in a single account?
- Single event bus, multiple accounts?
- Multiple event buses across multiple accounts?

aws

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Single-bus, multi-account pattern



Characteristics

- All services push events to a single or “central” event bus.
- Rules for all services are created on the central event bus.

Team

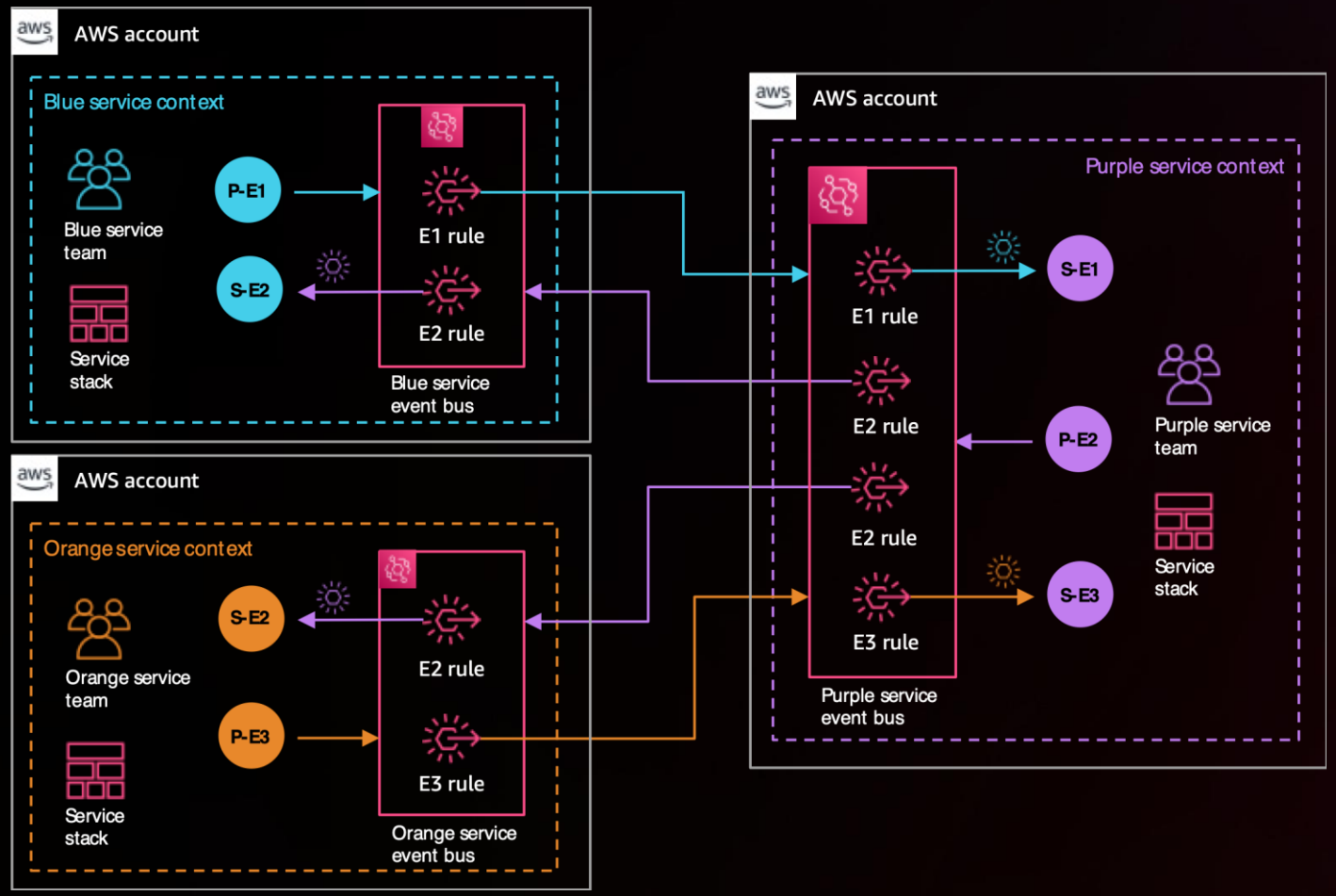
- DevOps teams manage access policies, delegate resource management to teams
- Service teams collaborate with DevOps teams to coordinate resource needs

Considerations

- Additional cross-account policy management. Resource policies help here!
- Consider potential impacts on development velocity

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Multi-bus, multi-account pattern



Characteristics

- Services publish events to an event bus that they own and manage
- Only has rules from services that are subscribing to the events the service is publishing

Team

- Service teams manage their service event buses and all resources for sending and receiving events

Considerations

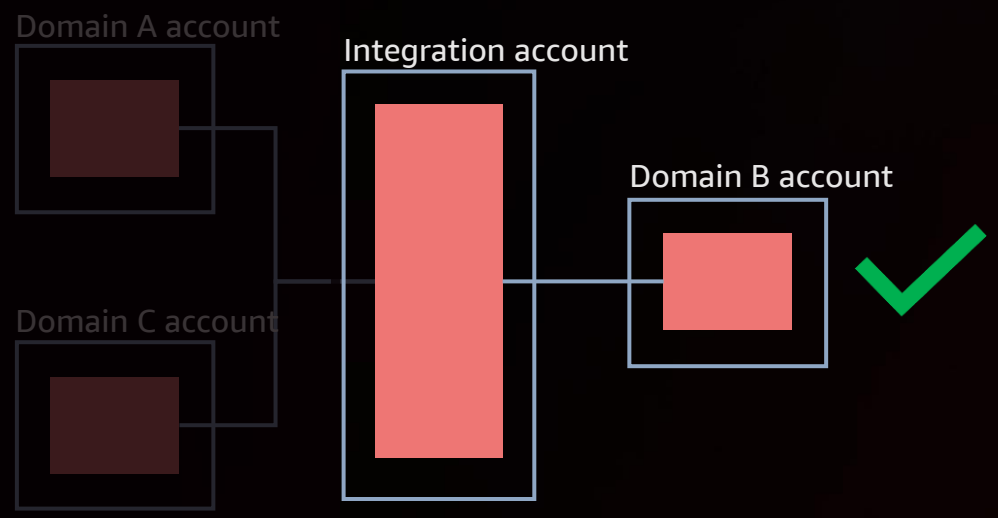
- Additional overhead in managing distributed rules

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Multi-account bus topology comparison

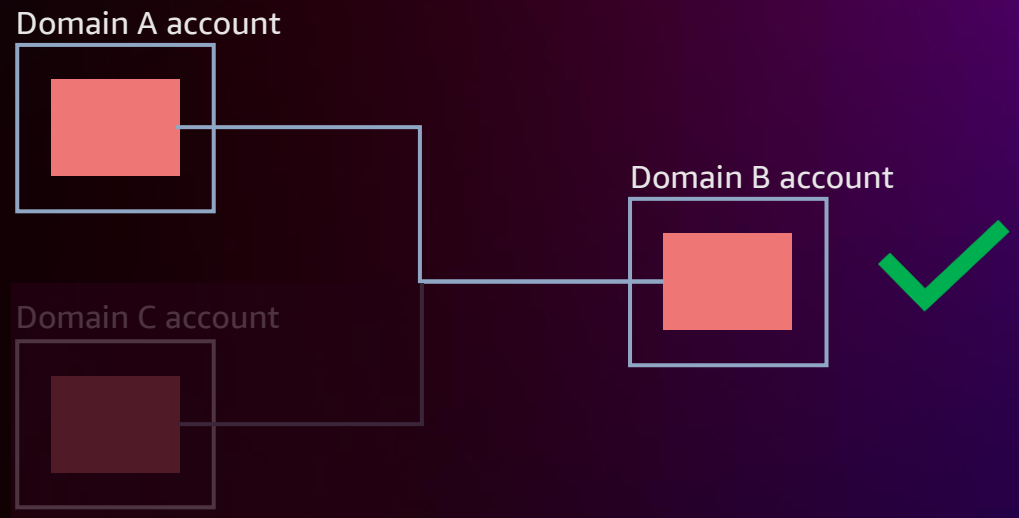
RESOURCE DISTRIBUTION

Single-bus topology



Centralized

Multi-bus topology



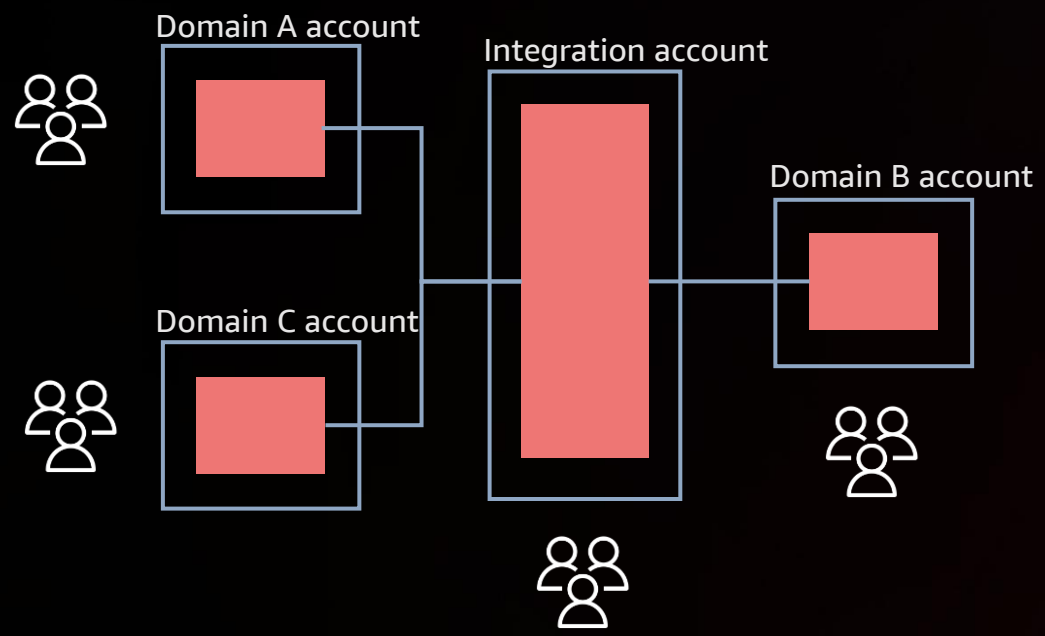
Decentralized

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Multi-account bus topology comparison

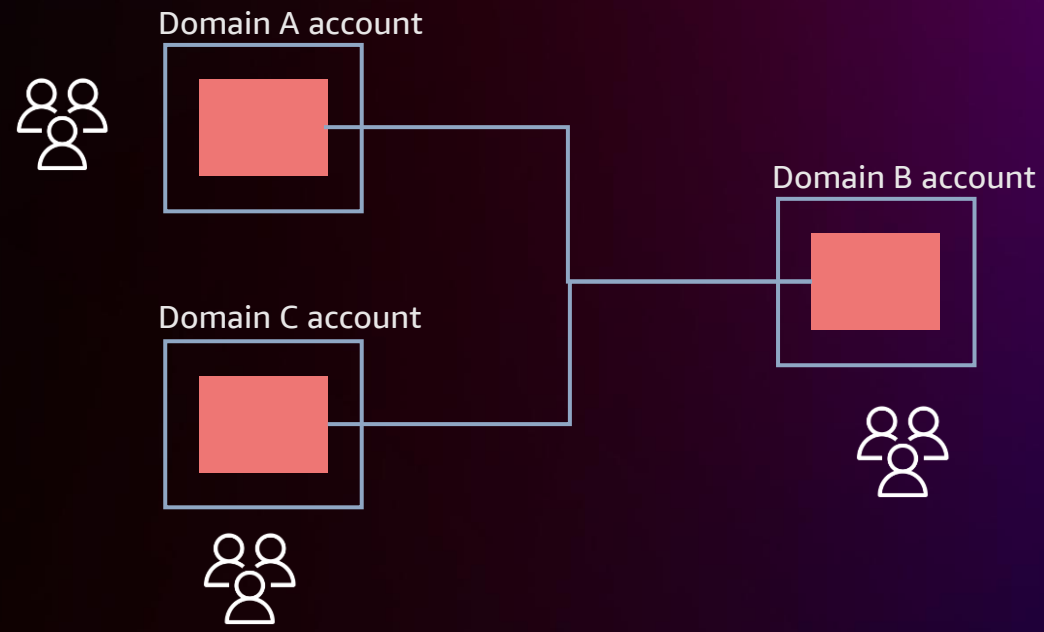
RESOURCE MANAGEMENT

Single-bus topology



Centralized (n+1)

Multi-bus topology



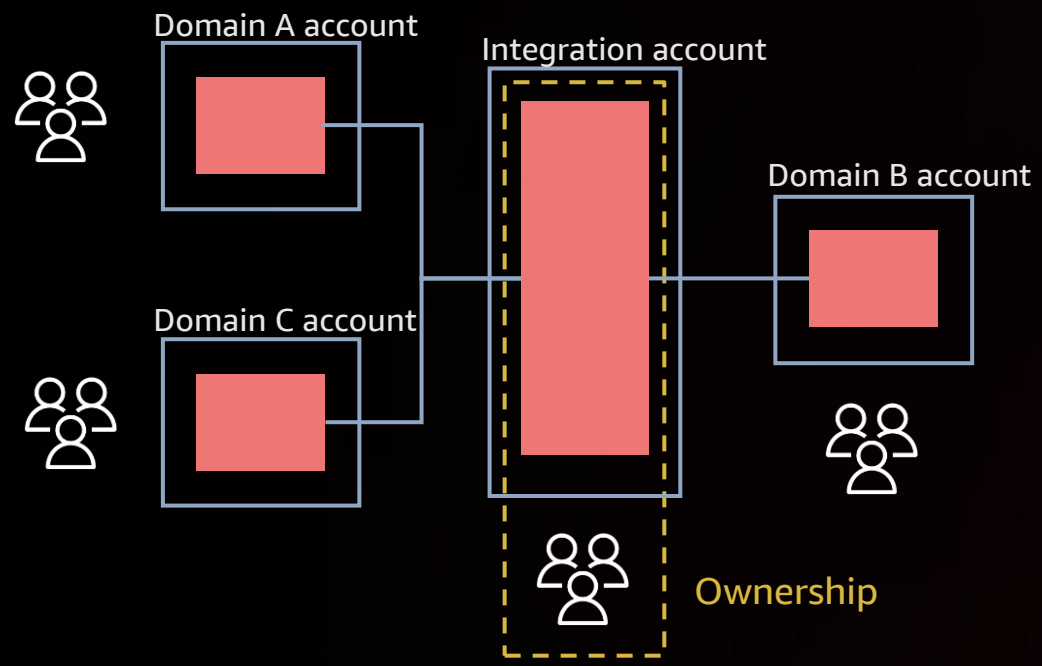
Decentralized

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Multi-account bus topology comparison

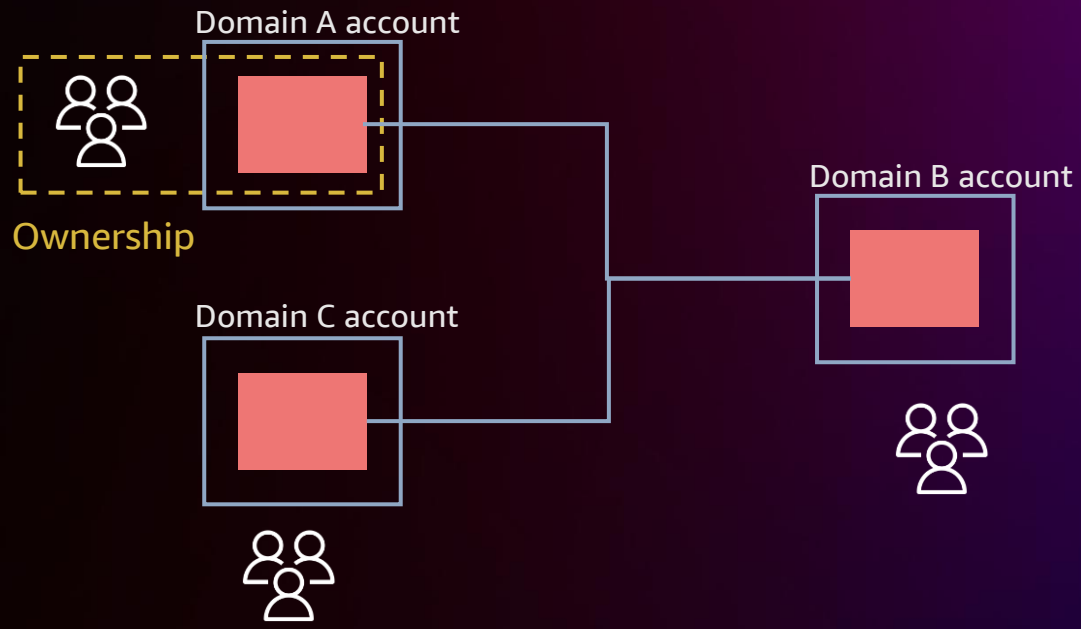
RESOURCE MANAGEMENT

Single-bus topology



Centralized (n+1)

Multi-bus topology



Decentralized

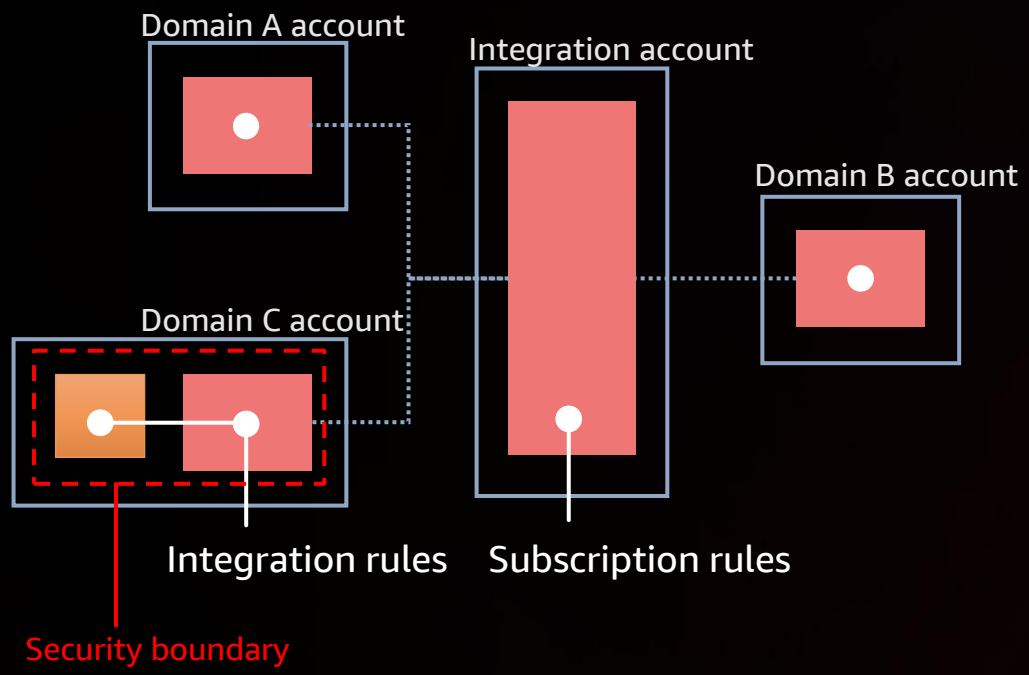
aws

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

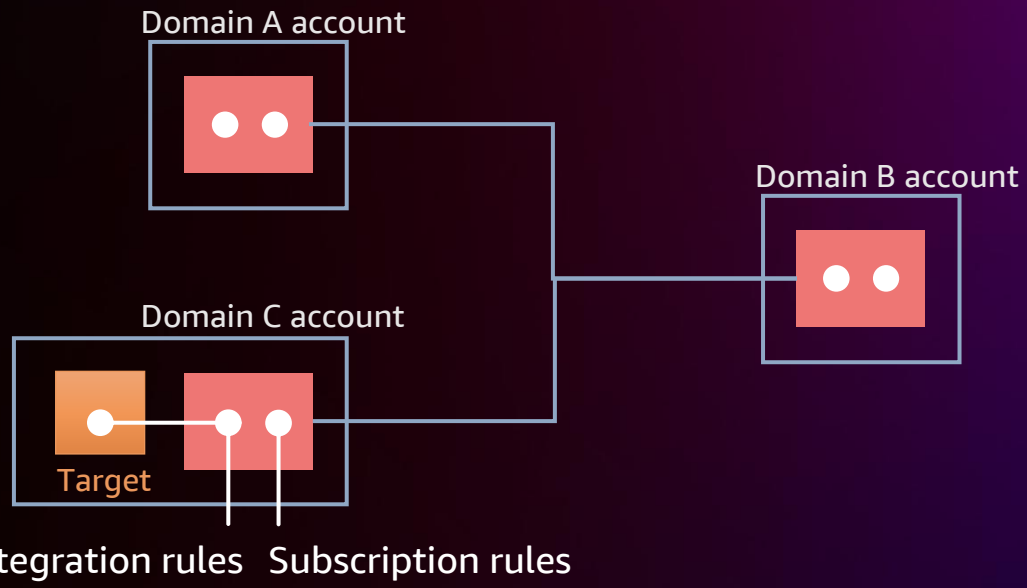
Multi-account bus topology comparison

RULE DISTRIBUTION

Single-bus topology rule distribution



Multi-bus rule distribution



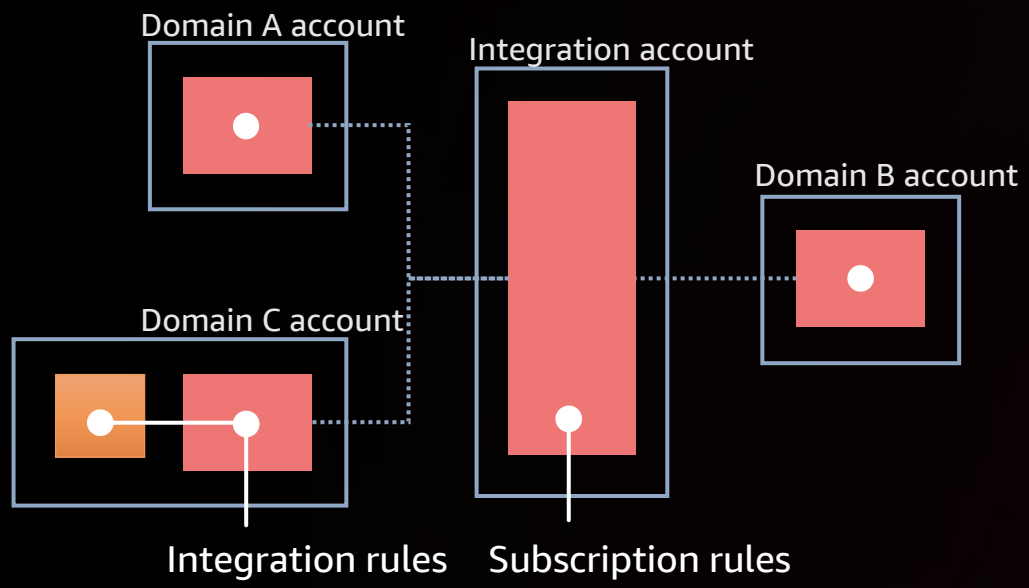
Rules are distributed in both cases, just in different ways

aws
© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Multi-account bus topology comparison

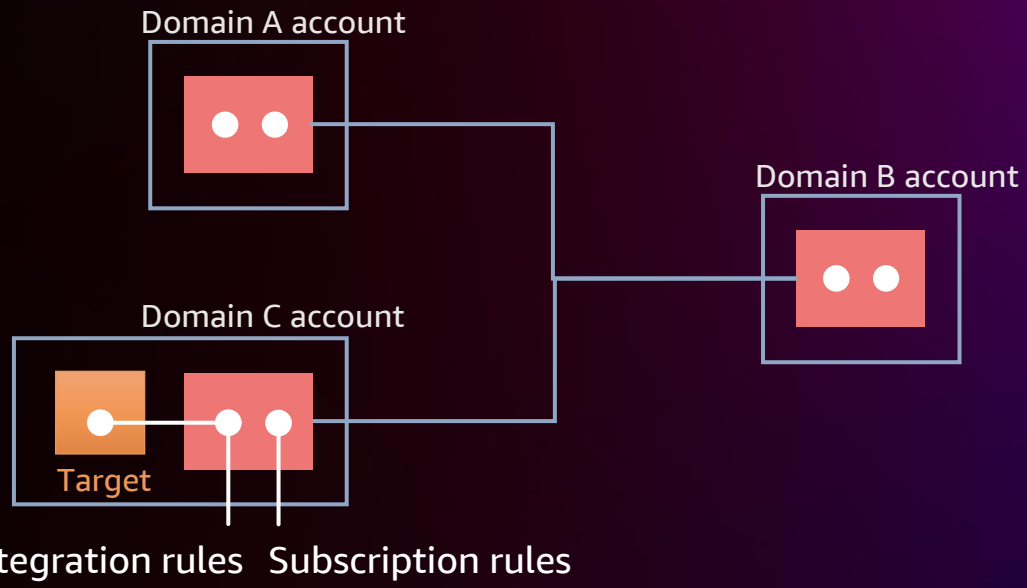
RULE DISTRIBUTION

Single-bus topology rule distribution



All subscriptions in one place

Multi-bus rule distribution



Only subscriptions from interested consumers

Rules are distributed in both cases, just in different ways

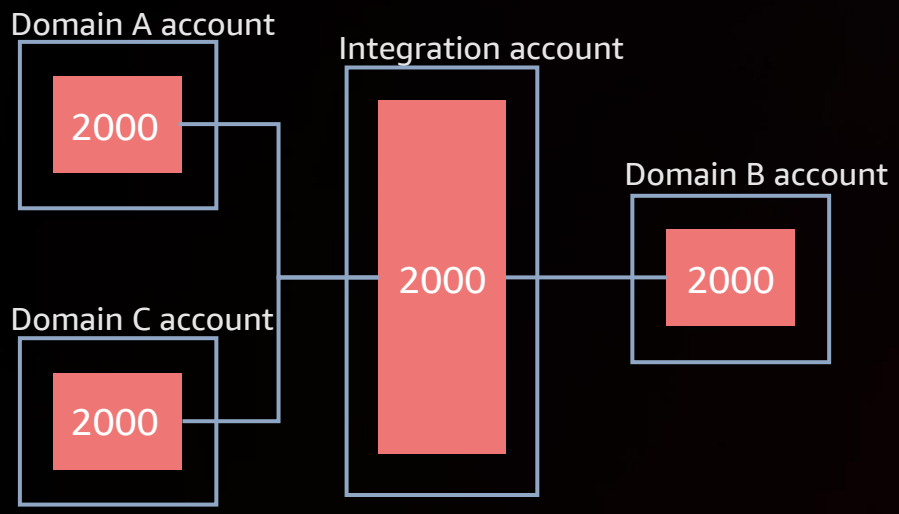
aws

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Multi-account bus topology comparison

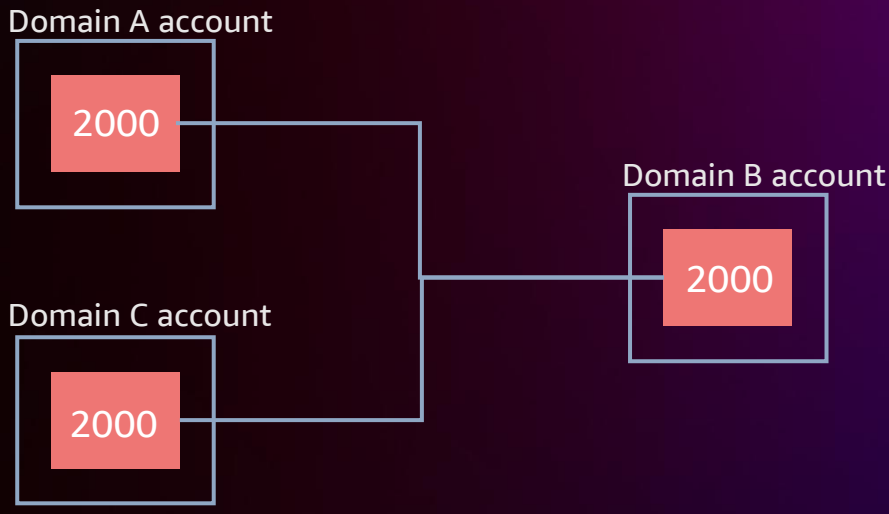
DISTRIBUTION OF SERVICE LIMITS

Single-bus rule limit distribution



Limits apply to all subscription rules on central event bus

Multi-bus rule limit distribution

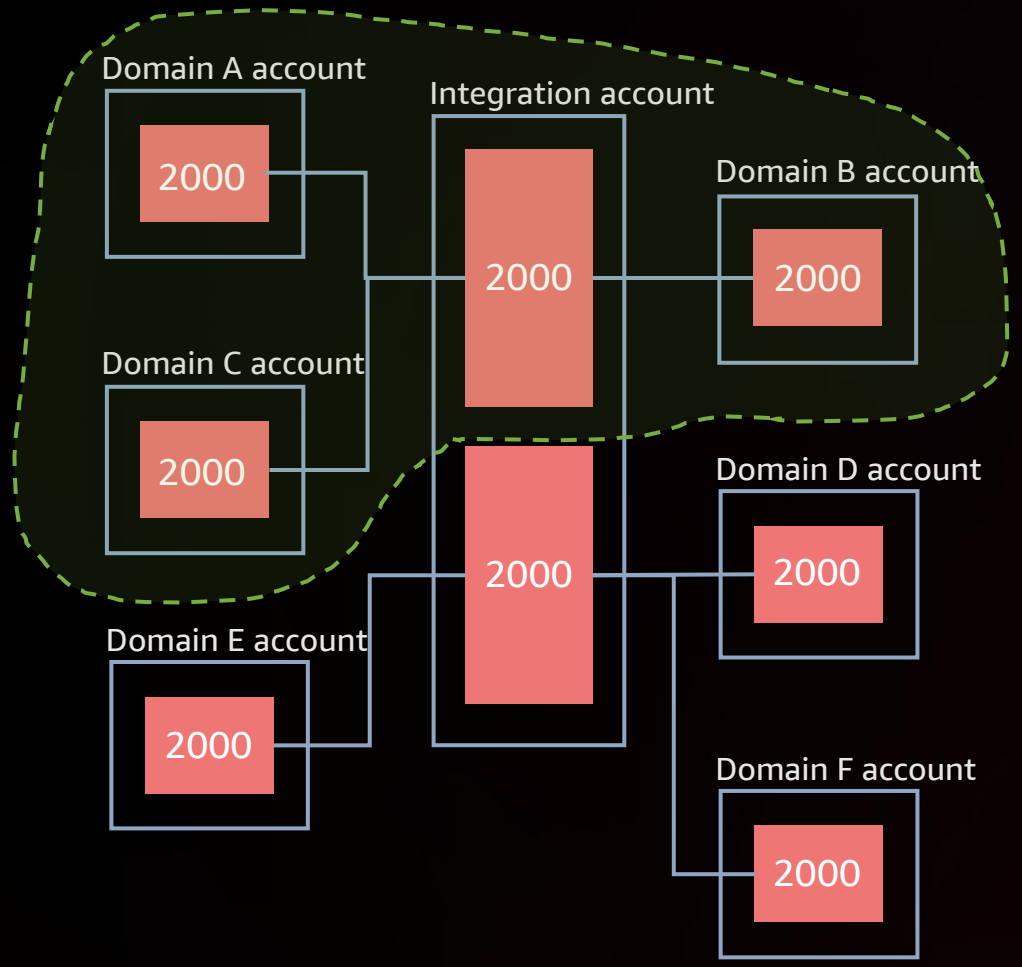


Limits apply to subscription rules for specific domain

aws
© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Dealing with rule growth

DISTRIBUTION OF SERVICE LIMITS

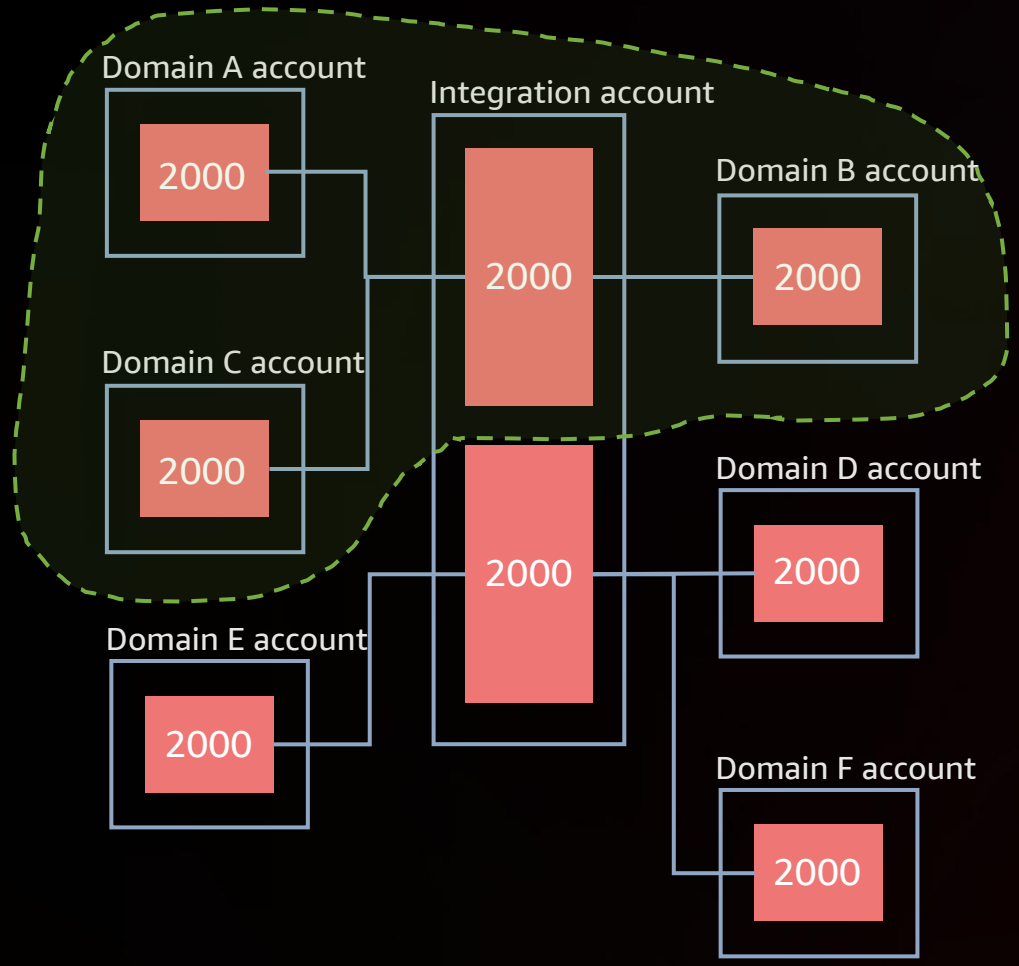


- **Distribute domains** across multiple event buses in central account
- Logical **domain groups** can publish to a specified event bus
- **Consumers** need to be aware of **where an event is published to**
- **Bus policies** need to allow event consumers to put rules across central event buses

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Dealing with rule growth

DISTRIBUTION OF SERVICE LIMITS




```
DomainARule:
  Type: AWS::Events::Rule
  Properties:
    Name: DomainA.DomainATarget.CustomerStatusChanged
    Description: Domain A subscription for event from Domain B
    EventBusName: "CentralEventBus1"
    EventPattern:
      source:
        - com.aws.DomainB
      detail-type:
        - CustomerStatusChanged
    State: ENABLED

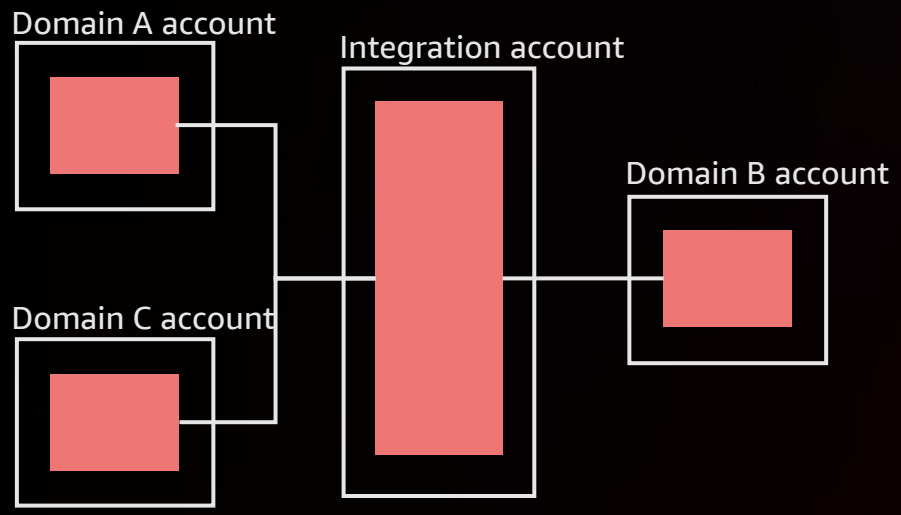
DomainARule2:
  Type: AWS::Events::Rule
  Properties:
    Name: DomainA.DomainATarget.MakeCustomerPreferred
    Description: Domain A subscription for event from Domain D
    EventBusName: "CentralEventBus2"
    EventPattern:
      source:
        - com.aws.DomainD
      account:
        - MakeCustomerPreferred
    State: ENABLED
```

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

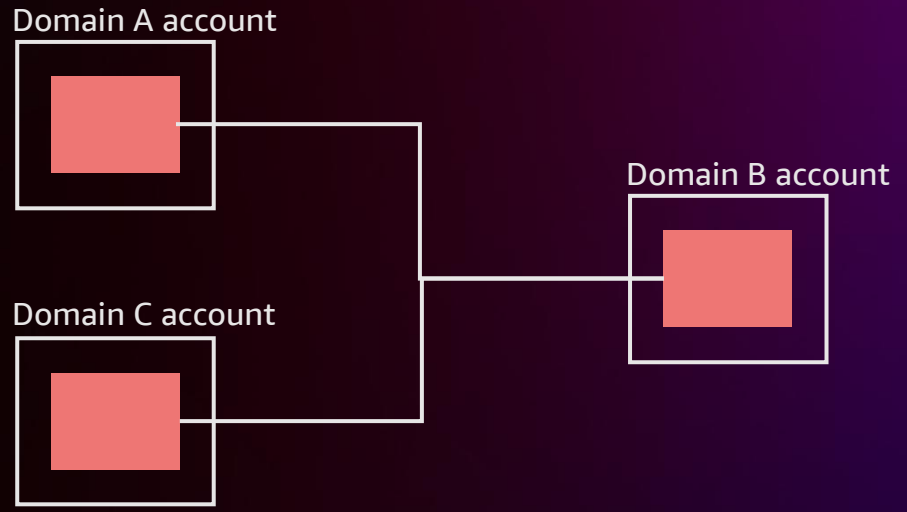
Multi-account bus topology comparison

WHICH IS BEST?

Single-bus topology



Multi-bus topology



It depends !!!

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

What you can do with one line of code?

```
aws events create-event-bus --name MyDomainEventBus
```

- Powerful, lightweight integration service, using non-proprietary protocols and tools
- Ability to route hundreds of thousands of messages per second
- Create a centralized or decentralized routing topology
- Send and replicate events across accounts and regions
- No infrastructure to manage

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Coupling

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

The many facets of coupling

- Technology dependency: Java versus C++
- Location dependency: IP addresses, DNS
- Data format dependency: Binary, XML, JSON, ProtoBuf, Avro
- Data type dependency: int16, int32, string, UTF-8, null, empty
- Semantic dependency: Name, Middlename, ZIP
- Temporal dependency: sync, async
- Interaction style dependency: messaging, RPC, query-style (GraphQL)
- Conversation dependency: pagination, caching, retries

aws
© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Source: EnterpriseIntegrationPatterns.com

“The appropriate level of coupling depends on the level of control you have over the endpoints.”

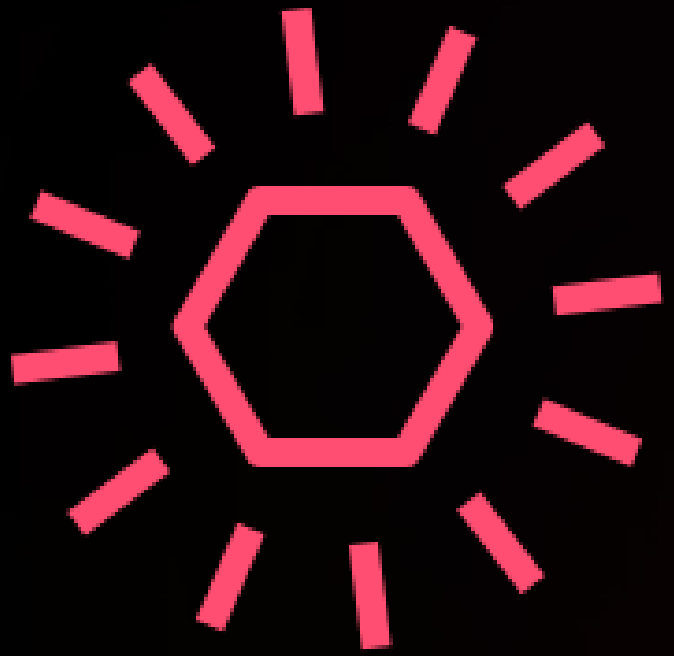
Gregor Hohpe

in *Enterprise Integration Patterns*

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Events

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.



Event

ɪˈvɛnt noun

An object in a system that has changed

aws

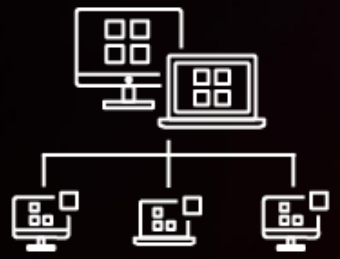
© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Event types



Notification events

- Notify that something important has happened
- Minimal information
- May need to query producer for additional details



Event-carried state transfer events

- Publishing changes in data
- Does not need to query producer for additional details
- Example: Amazon DynamoDB streams
- Useful for replicating data across domains without a direct dependency



Domain events

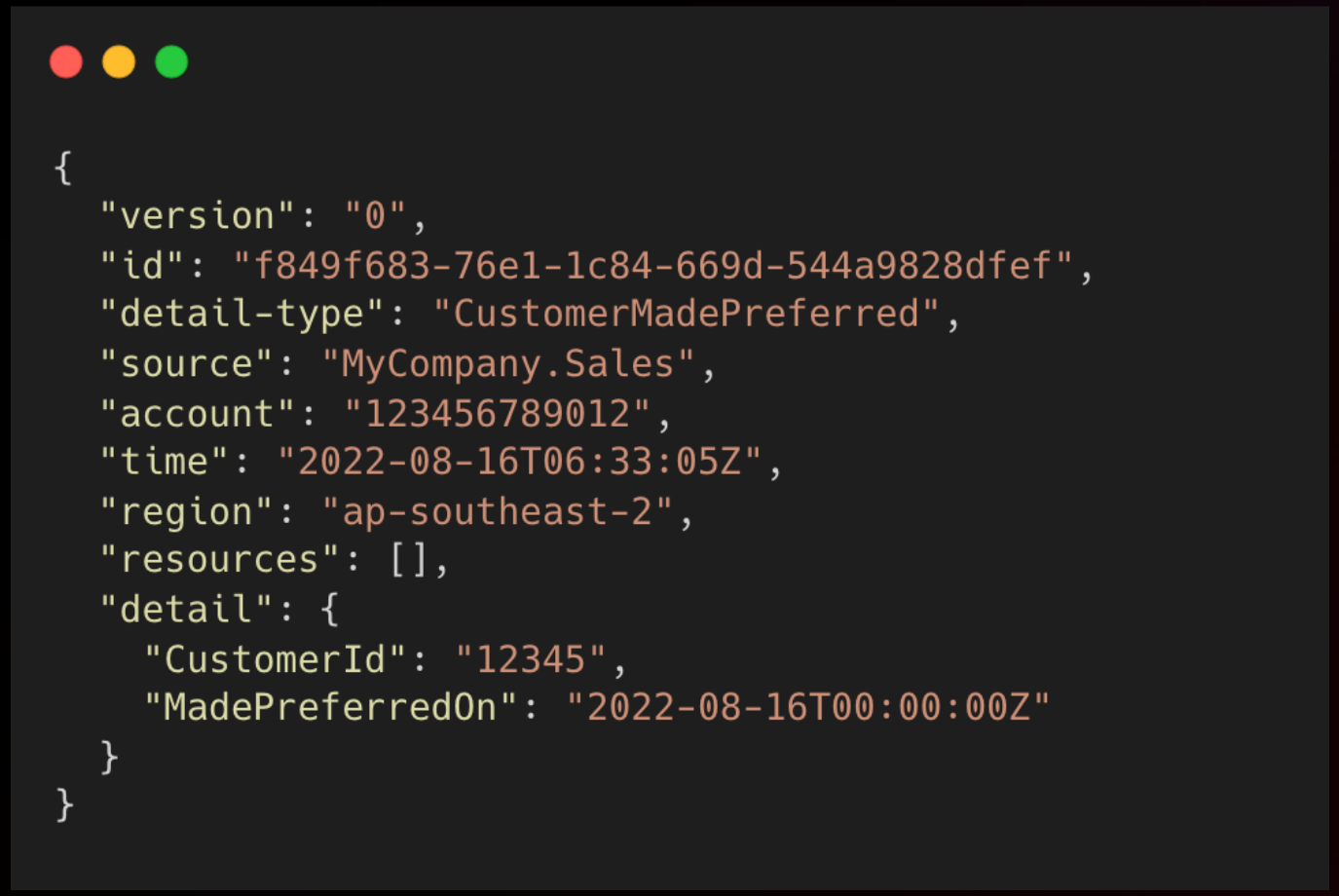
- Combination of notification and state transfer events
- Modeled as close as possible to the business domain
- Stateless, immutable, source of truth

Source: https://martinfowler.com/articles/201701-event-driven.html

aws

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Amazon EventBridge events



```
{
  "version": "0",
  "id": "f849f683-76e1-1c84-669d-544a9828dfef",
  "detail-type": "CustomerMadePreferred",
  "source": "MyCompany.Sales",
  "account": "123456789012",
  "time": "2022-08-16T06:33:05Z",
  "region": "ap-southeast-2",
  "resources": [],
  "detail": {
    "CustomerId": "12345",
    "MadePreferredOn": "2022-08-16T00:00:00Z"
  }
}
```

- Context is added to data to provide for consumers
- Fields `detail-type` and `source` scope events to a bounded context; together they **describe the intent of the event** and its details
- Core to establishing **homogenous integration between heterogeneous systems** and applications

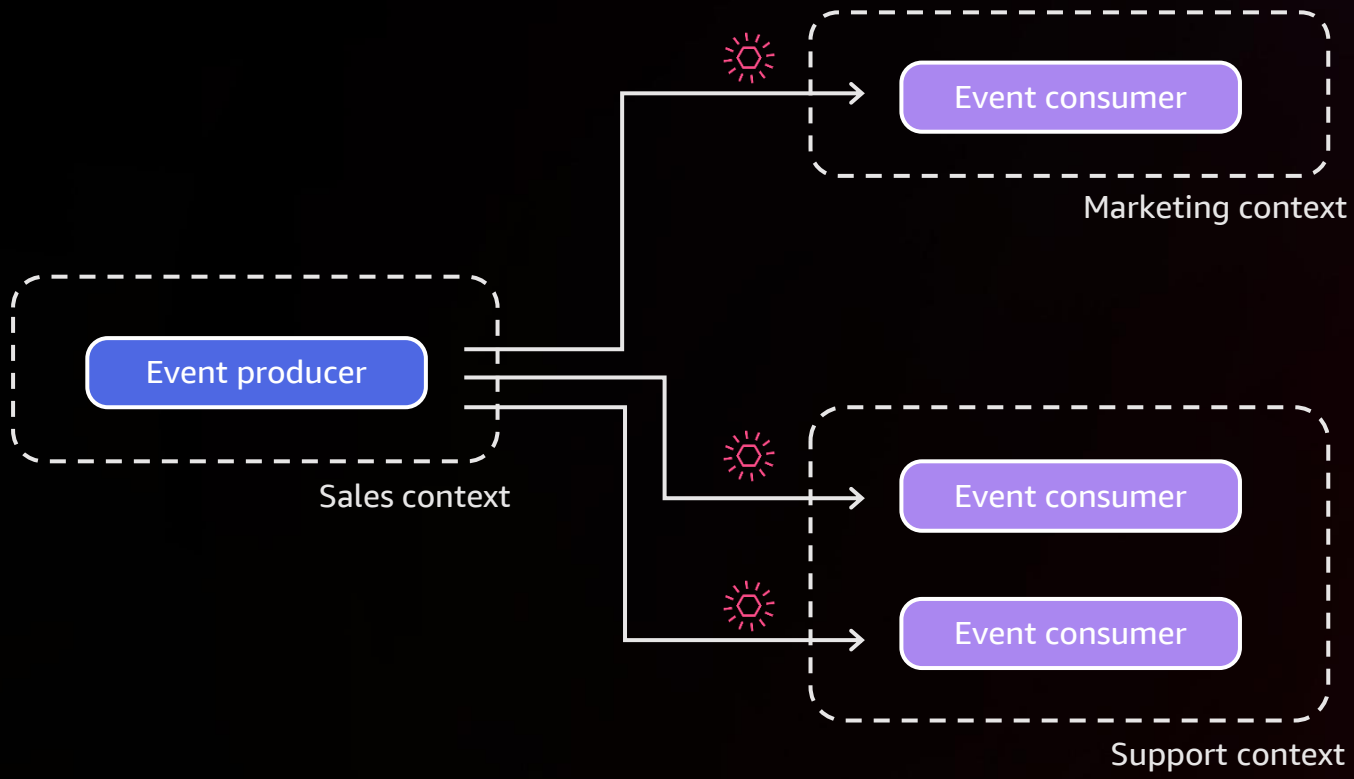
aws

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Roles and responsibilities

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Publish/subscribe integration pattern

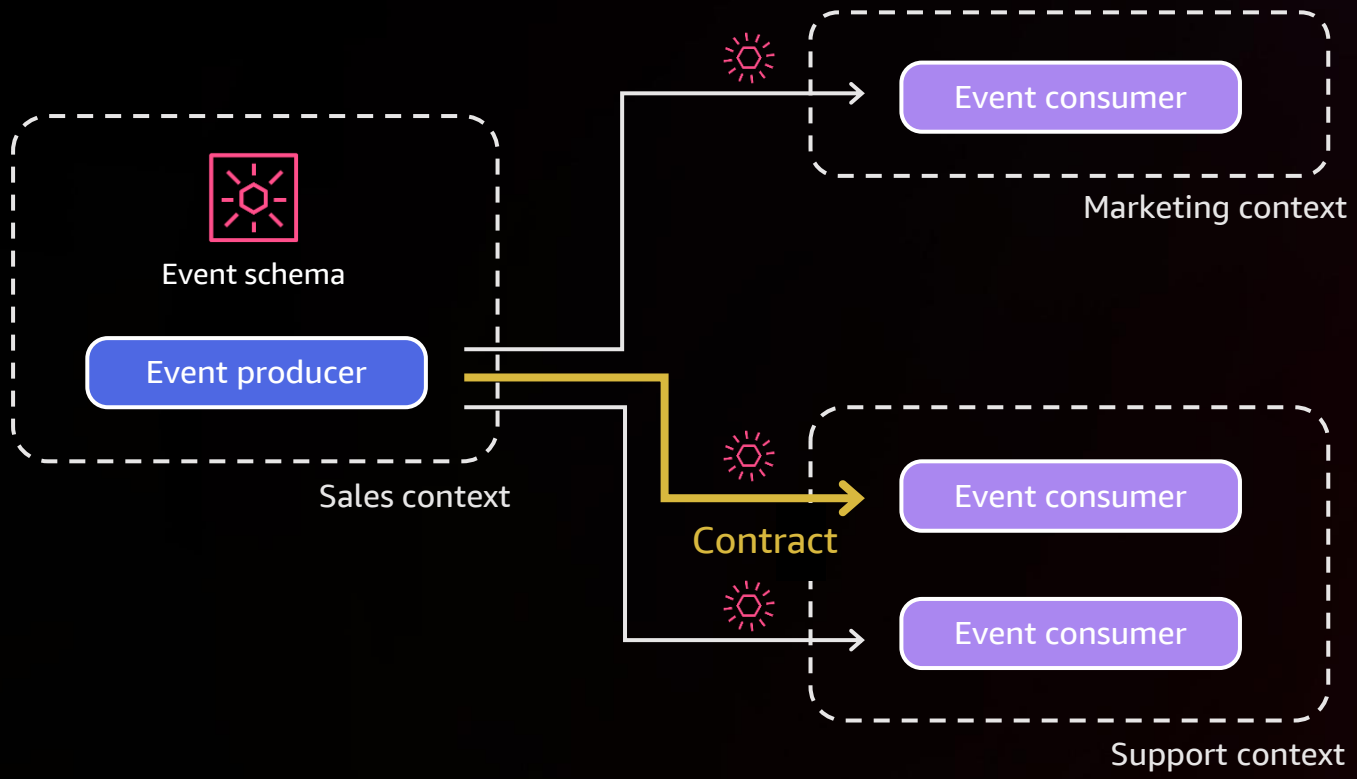


Publish subscribe

- One logical publisher for a given **domain event,** one or more consumers
- Enforces a **consistent boundary** between our bounded contexts
- Addresses **multiple types of coupling** – temporal, spatial and platform

aws
© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Publish/subscribe integration pattern

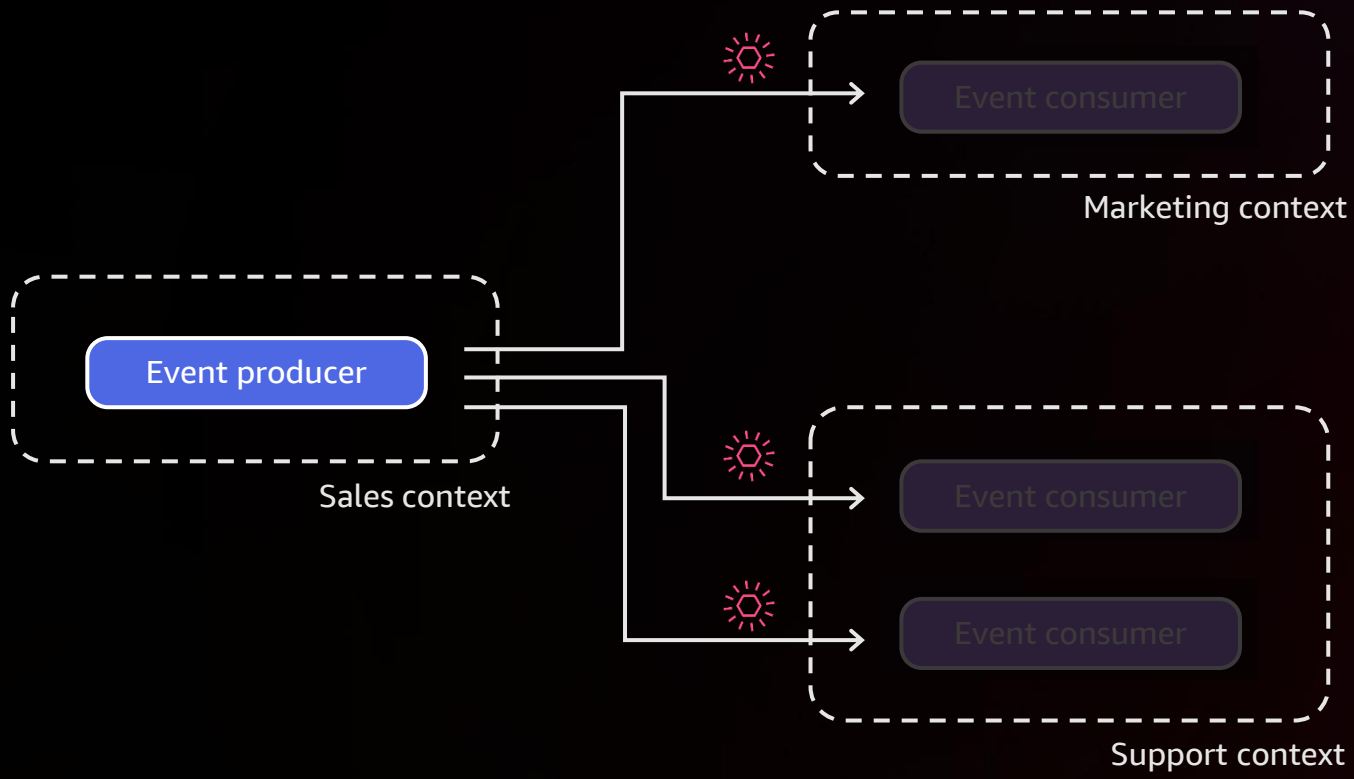


Establishing contracts

- Not all coupling is removed – the **good coupling** is still there
- The **business context** establishes a **"contract"** between producers and consumers
- Contacts can be represented as an **EventBridge schema**

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Producer responsibilities

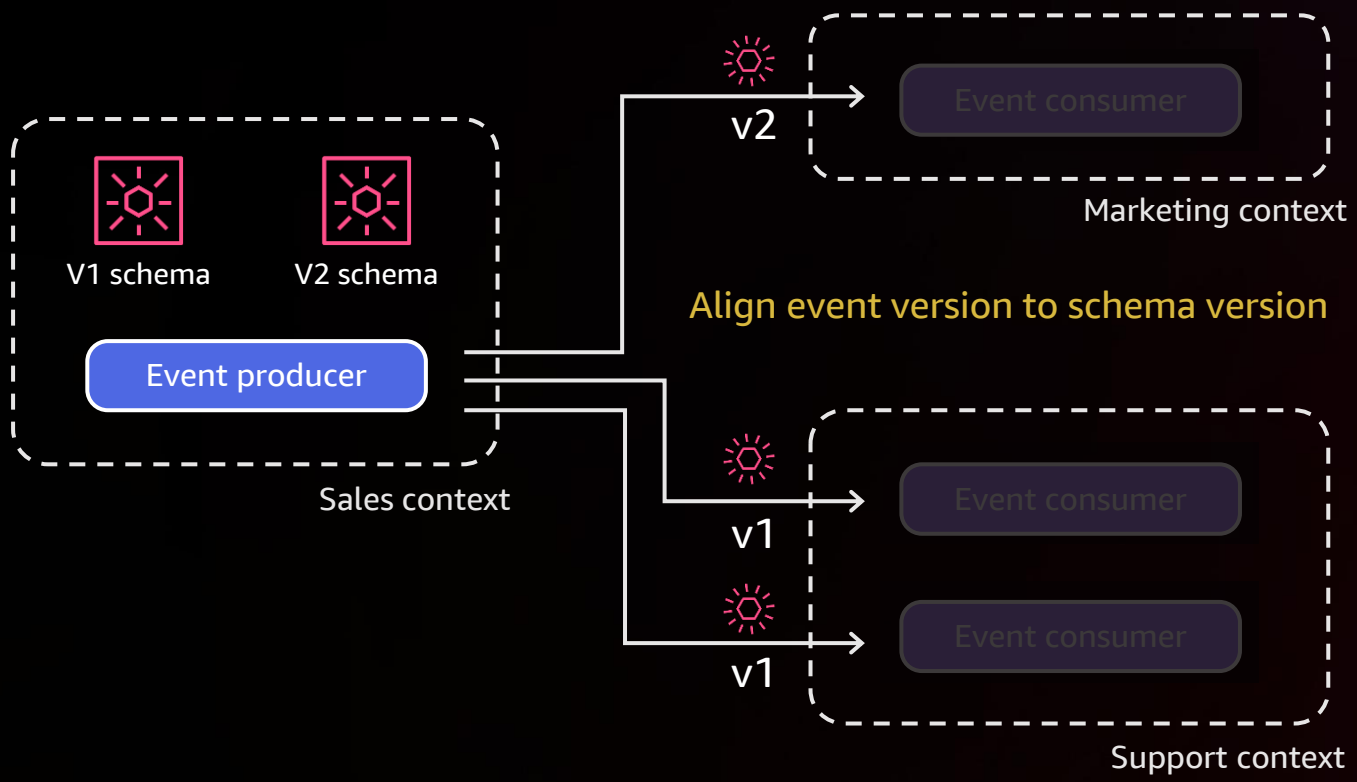


Producers ensure

- The events they publish are **stable and consistent** with the business processes that define them
- The events that they publish **conform to an agreed upon contract**/definition

aws
© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Producer responsibilities



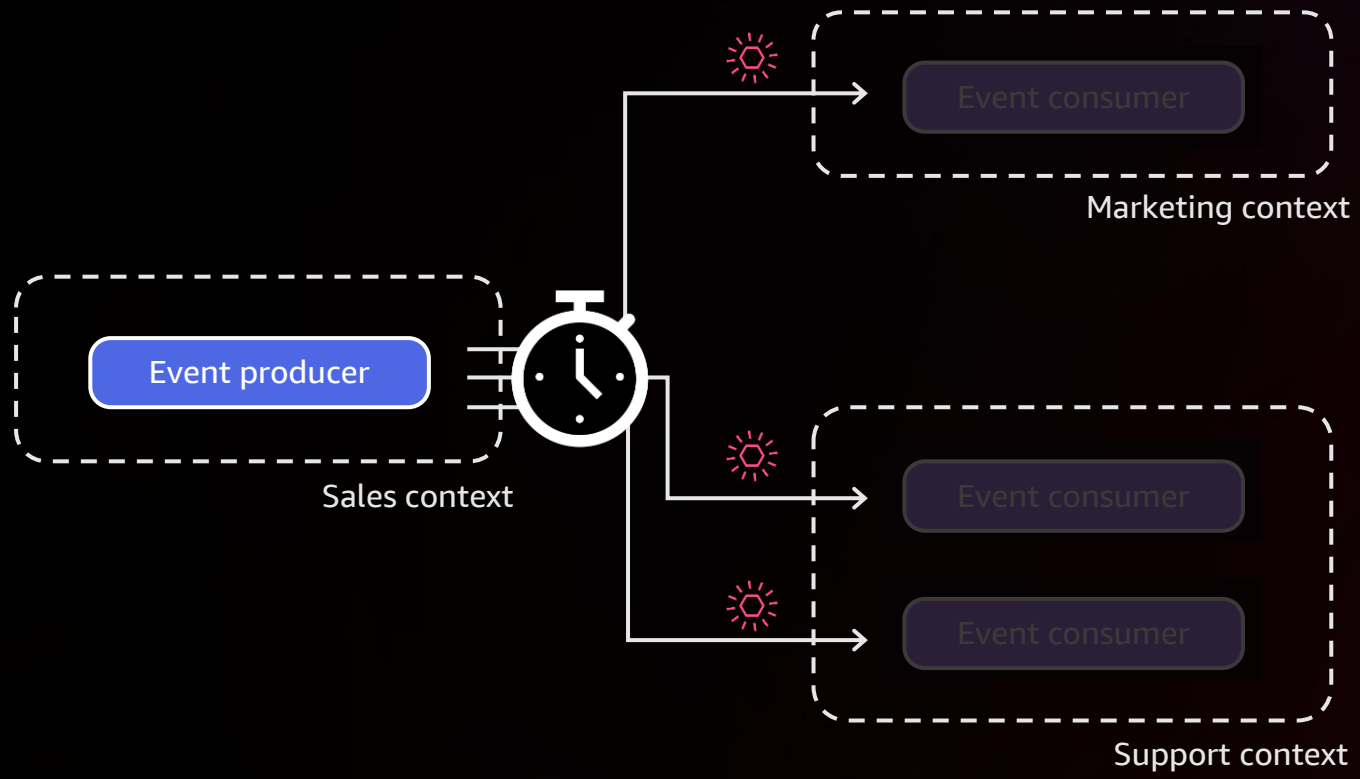
Producers ensure

- The events they publish are **stable and consistent** with the business processes that define them
- The events that they publish **conform to an agreed upon contract**/definition

This also means ensuring backwards compatibility as events definitions evolve

aws
© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Producer responsibilities

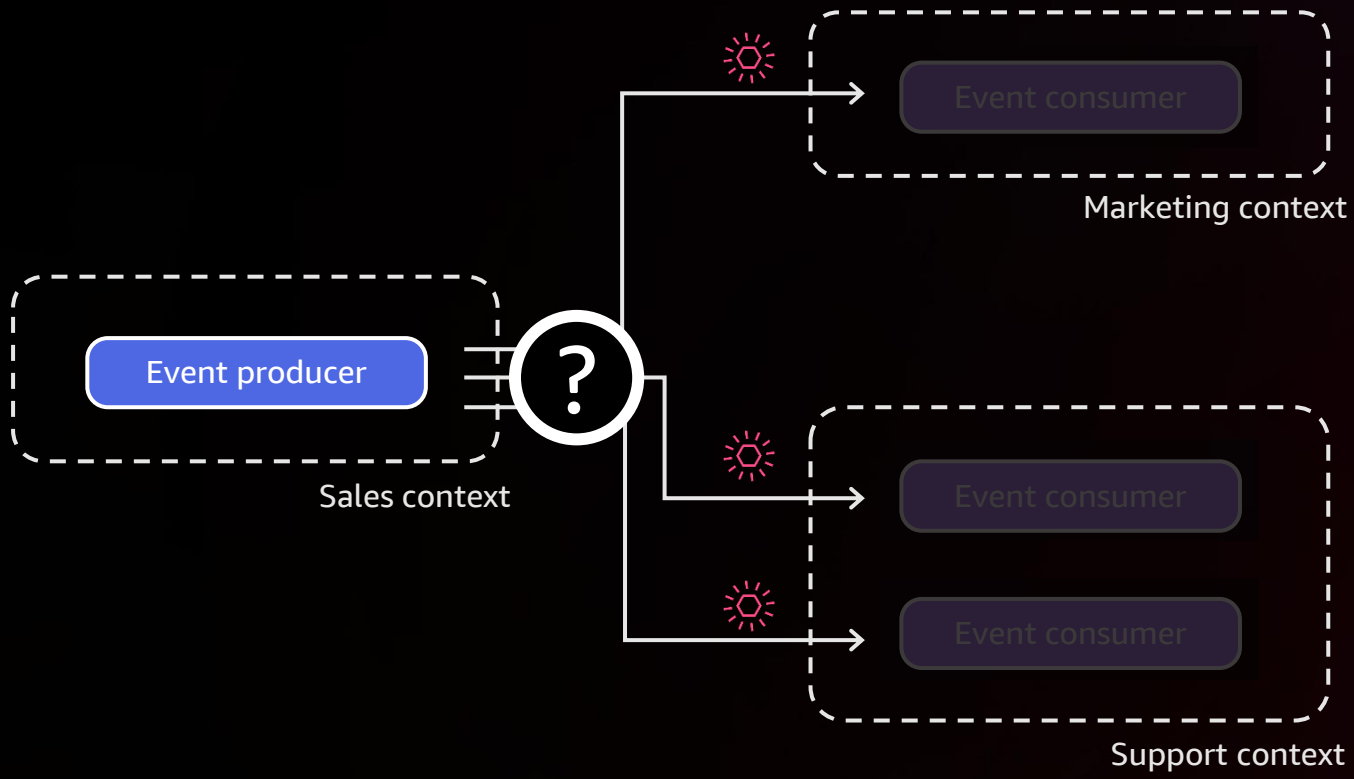


Producers ensure

- The events they publish are **stable and consistent** with the business processes that define them
- The events that they publish **conform to an agreed upon contract**/definition
- The **delivery of messages occurs as close to real time as possible**, independent of the needs of the consumers of the event

aws
© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Producer responsibilities

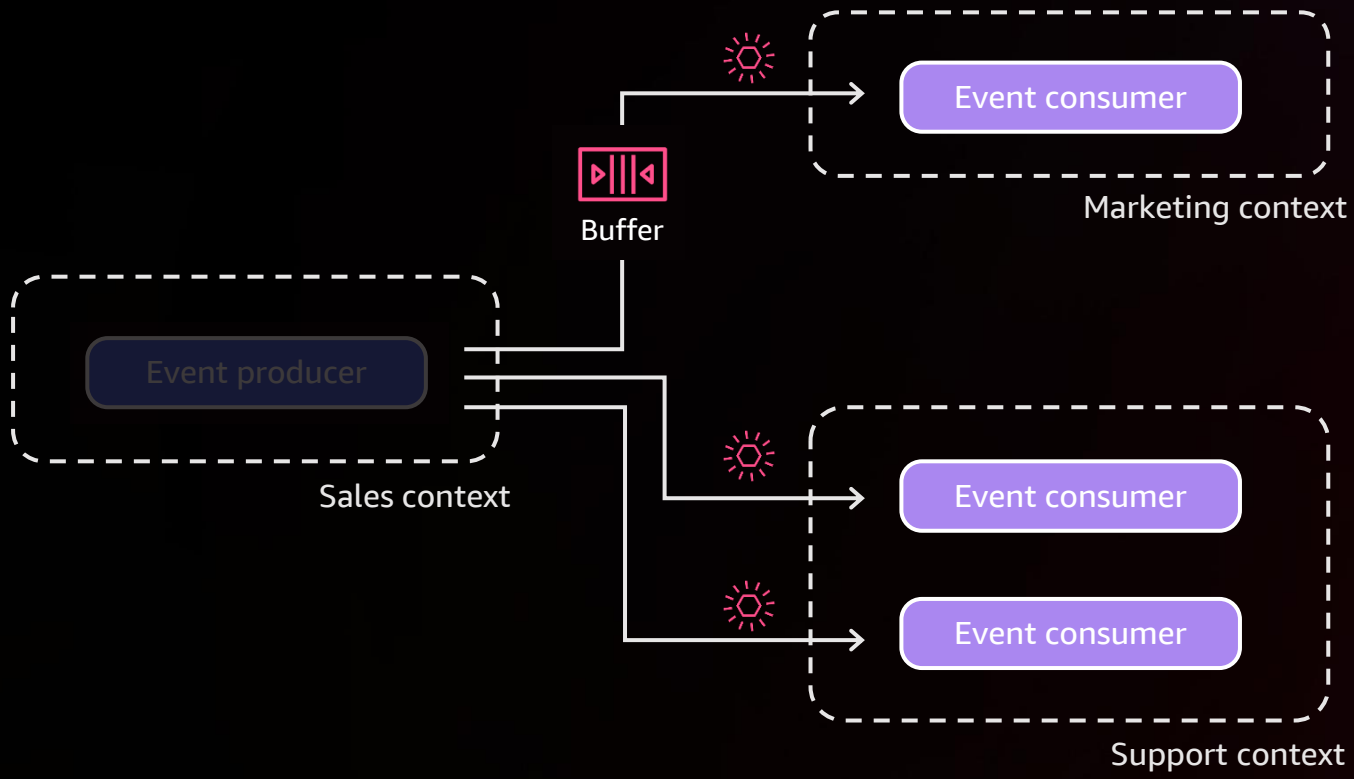


Producers are NOT responsible for

- **How** consumers are using their events
- **Knowing consumer constraints**, for example, the rate at which events are processed by consumers
- Understanding how **consumers need to transform events** prior to processing

aws
© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Consumer responsibilities

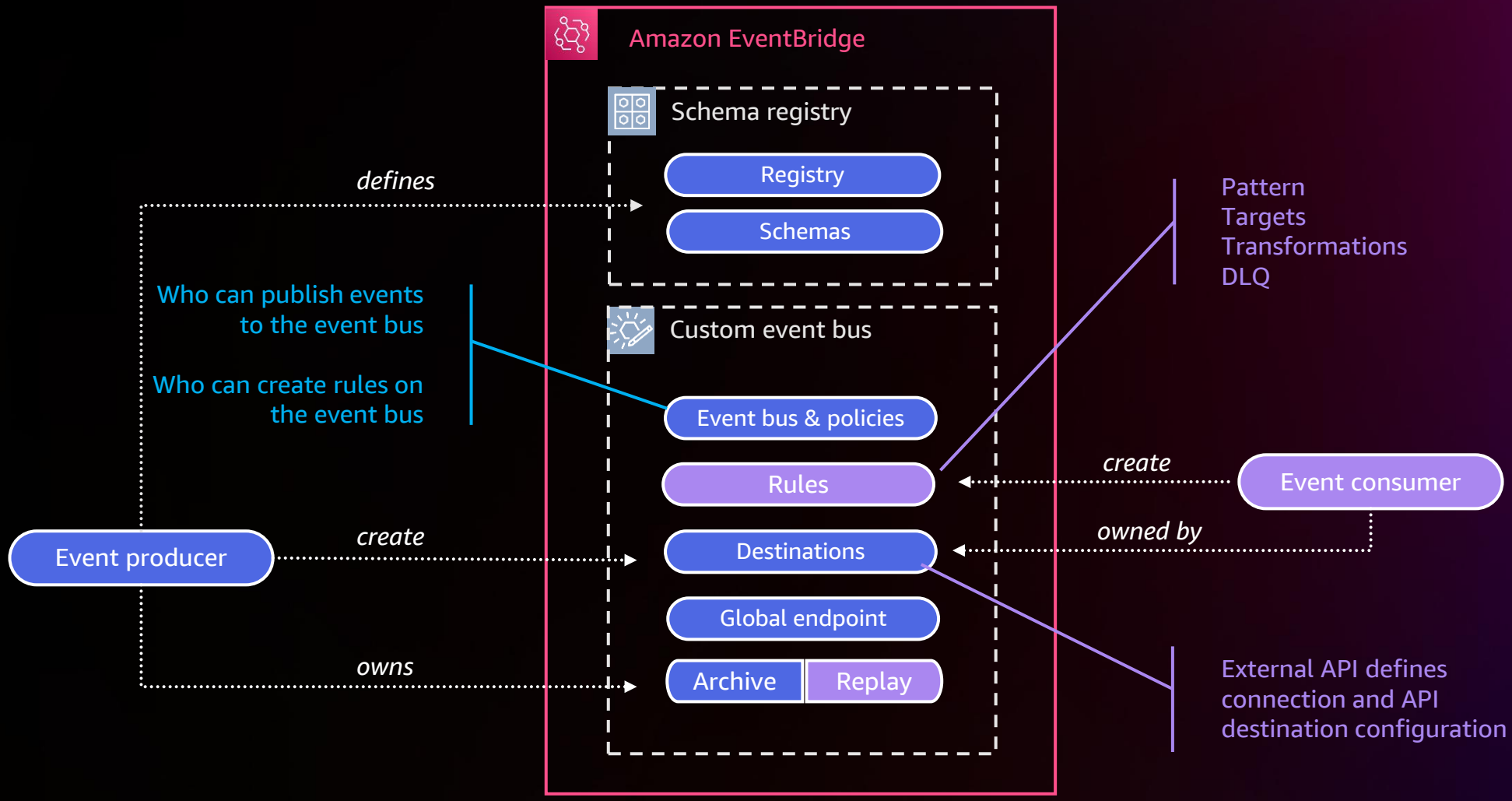


Consumers are responsible for

- Specifying **which events they want to receive**; additionally through selective filtering
- Determining **rate at which at which they process events**, for example, incoming event rate > consumer's ability to process
- Determining **transformation** needs

aws
© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Who owns what?

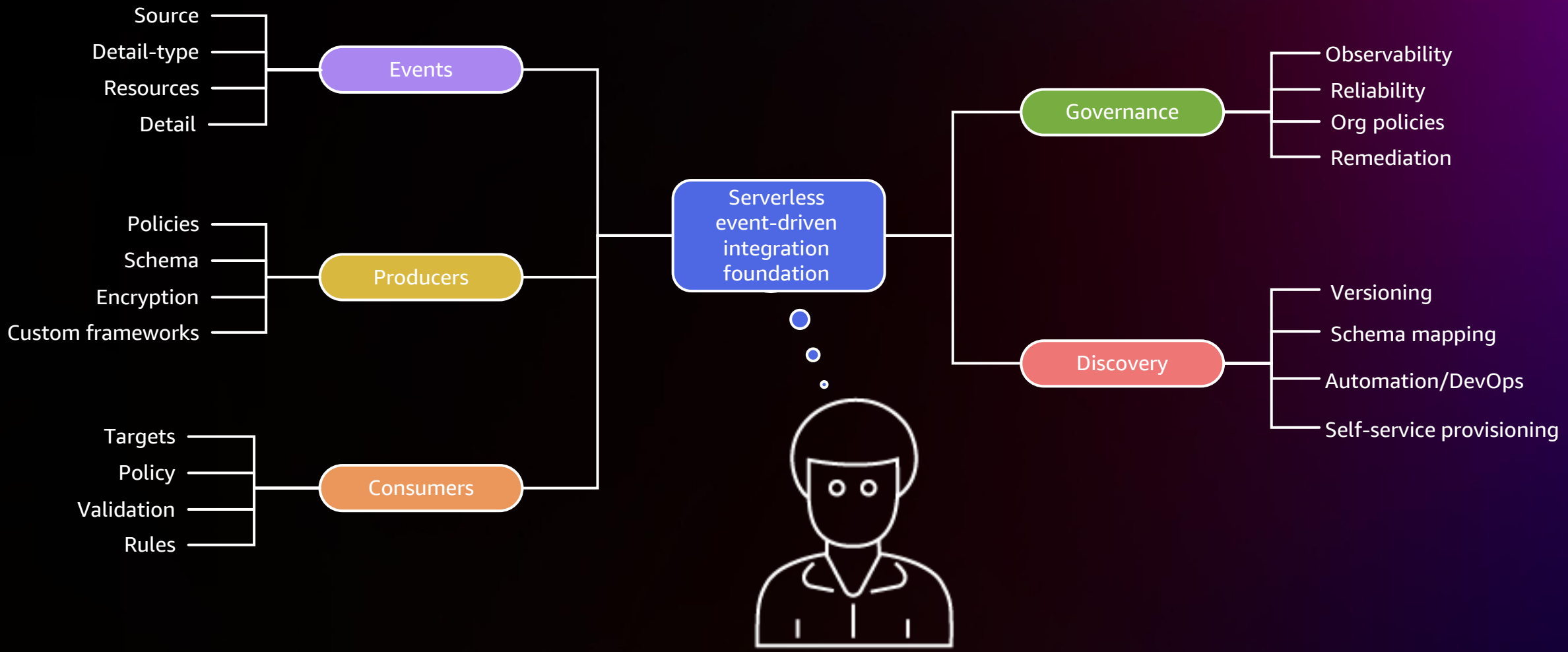


© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Foundations of a serverless event-driven integration using Amazon EventBridge

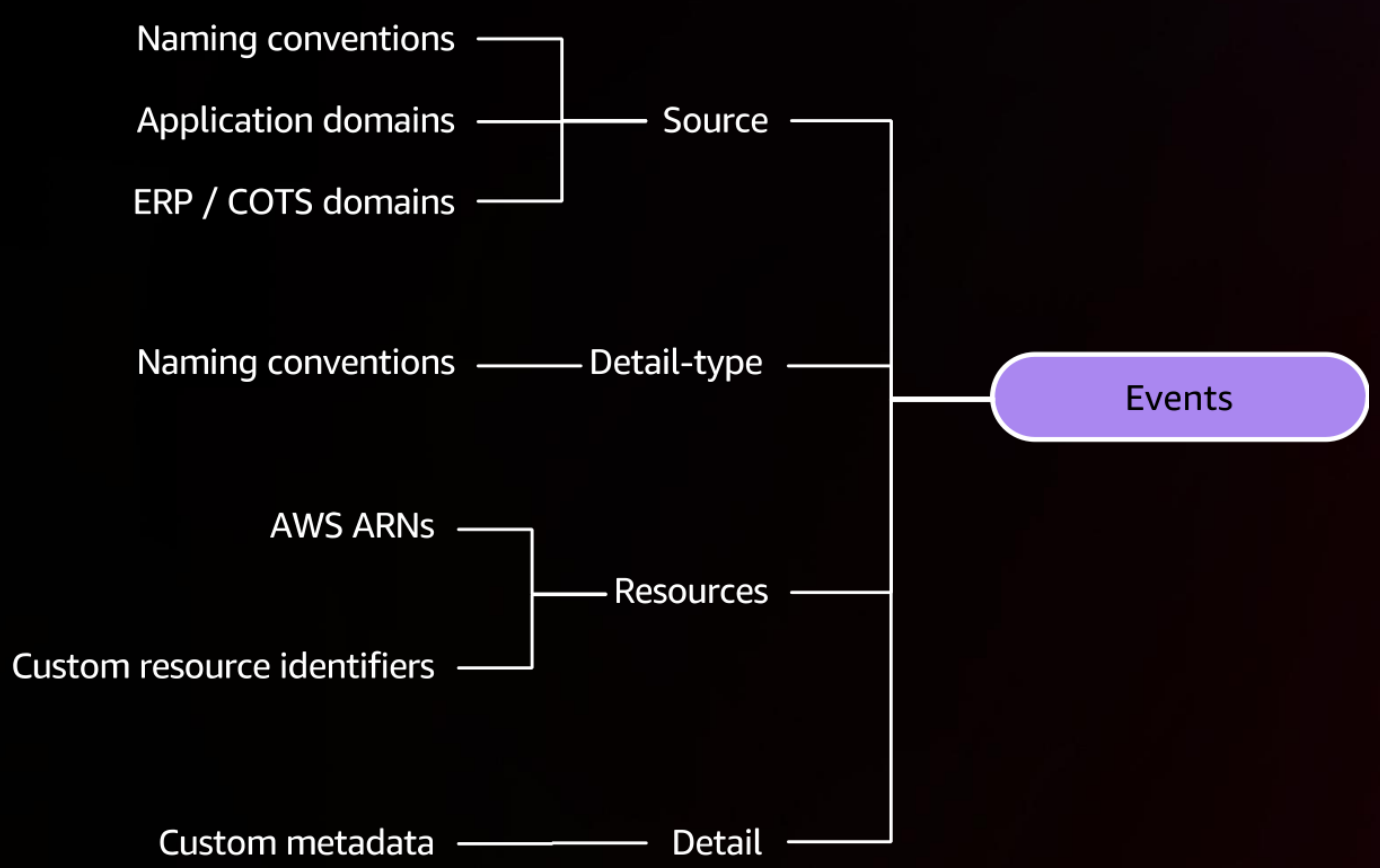
© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Event-driven integrations – a mind map



© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

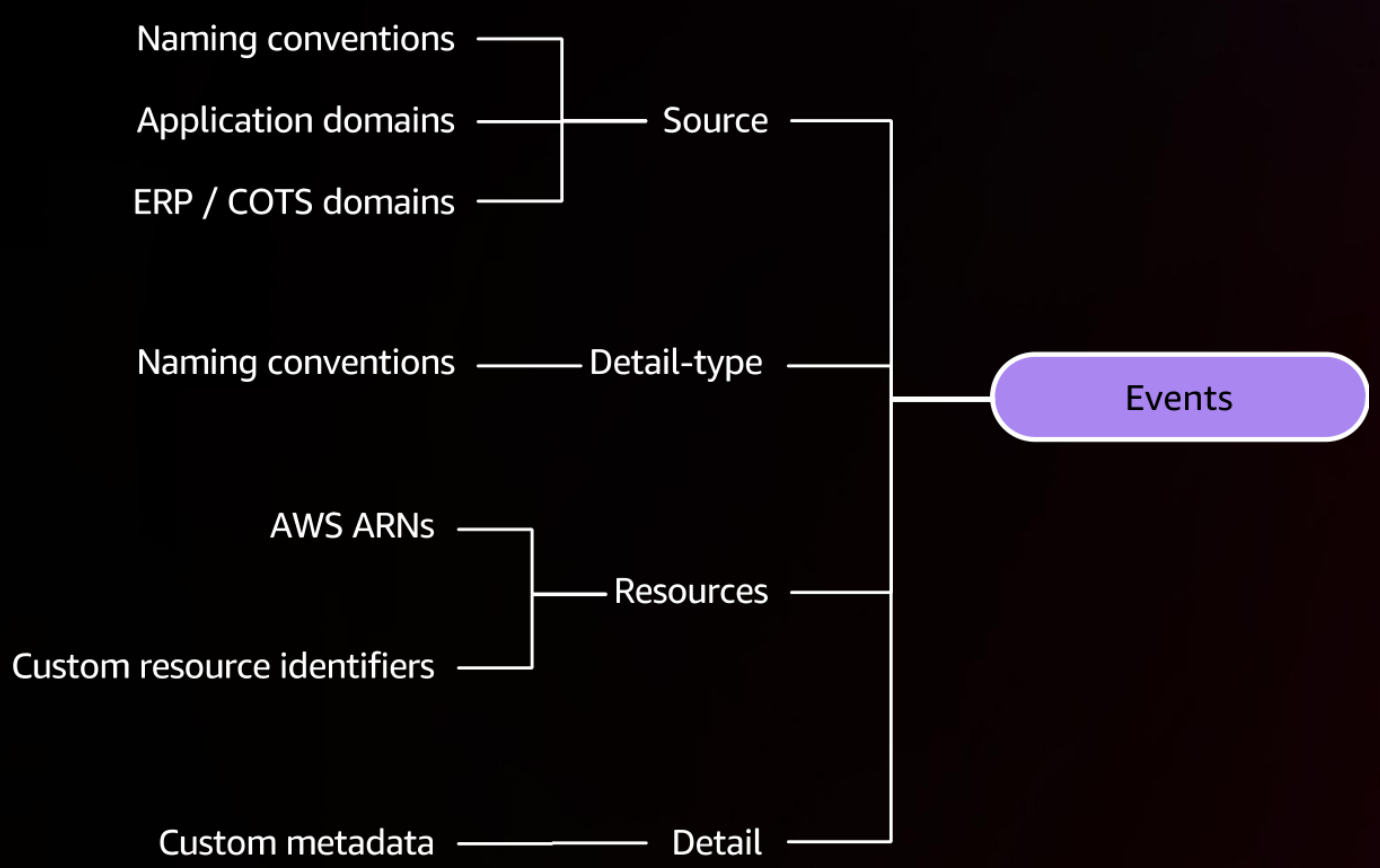
Events



- **Consistency is key** to uniquely identifying events; operational benefits
- Establish **naming conventions** that align to clearly defined domains
- Spend time thinking about how to **define different event types**
- Define **custom metadata to add additional context** to the event

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

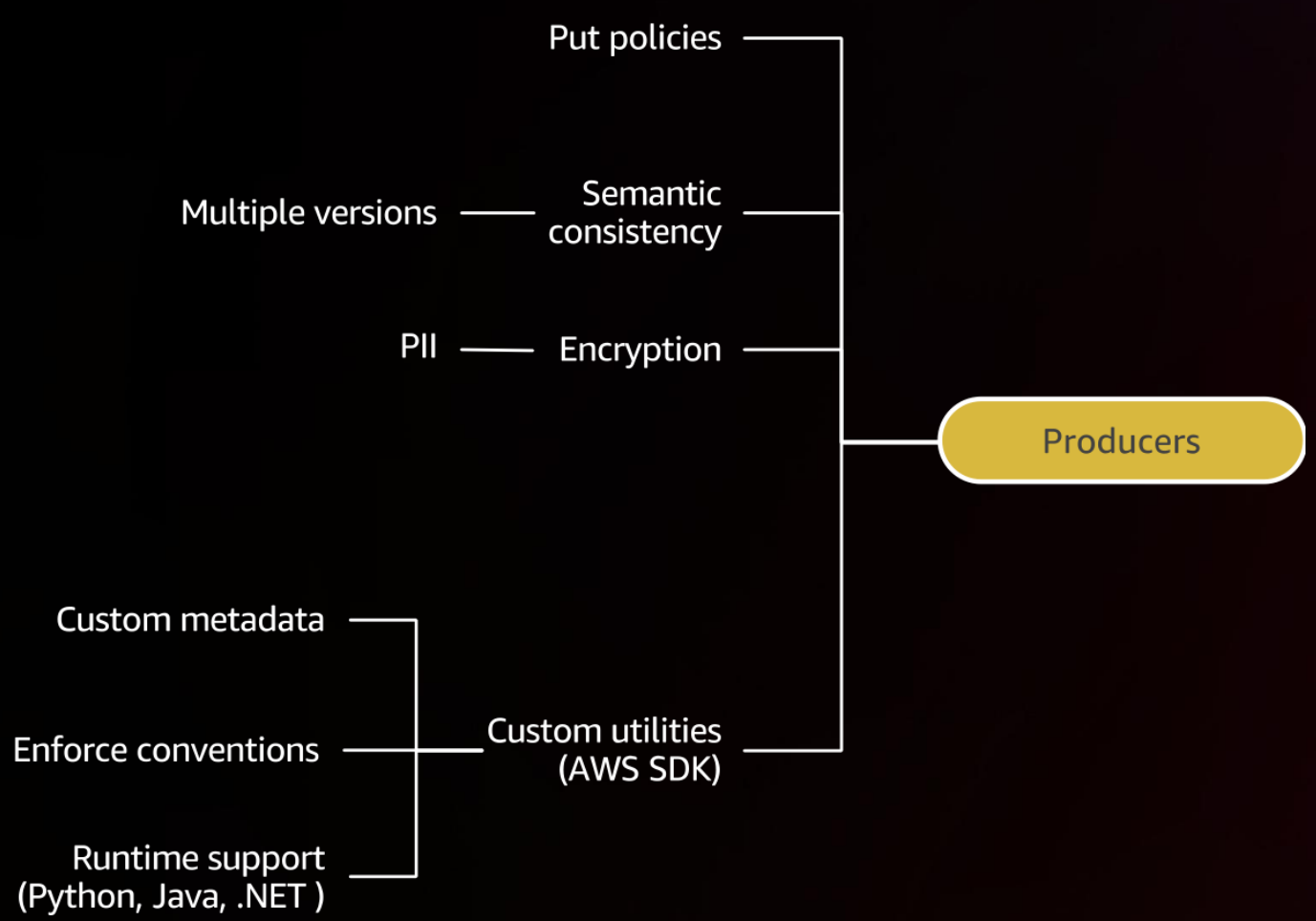
Events



- **Consistency is key** to uniquely identifying events; operational benefits
- Establish **naming conventions** that align to clearly defined domains
- Spend time thinking about how to **define different event types**
- Define **custom metadata to add additional context** to the event

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Producers



- Ensure the integrity of the event by enforcing **semantic consistency**
- Encryption of sensitive information needs to be at the client level – **Shared secrets are part of the contract that is established with consumer**
- Consider building **custom utilities** that initialize events with metadata properties and validate input
- Standardize across organization through **multiple runtimes**

aws
© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Build your own messaging utilities



```
{
  "version": "0",
  "id": "f849f683-76e1-1c84-669d-544a9828dfef",
  "detail-type": "CustomerMadePreferred",
  "source": "MyCompany.Sales",
  "account": "123456789012",
  "time": "2022-08-16T06:33:05Z",
  "region": "ap-southeast-2",
  "resources": [],
  "detail": {
    "Metadata": {
      "EventId": "12c37f32-6247-4d10-9527-193ef6b90f1d",
      "EventTime": "2022-08-16T06:11:36Z",
      "Location": "AU-876202721"
    },
    "Data": {
      "CustomerId": "12345",
      "MadePreferredOn": "2022-08-16T00:00:00Z",
    }
  }
}
```

- **Avoid having development teams implement their own solutions** to include metadata in event payloads
- Create **utility libraries** that can enforce event consistency and structure
- Support the **runtimes** that your organization uses
- **Distribute** via private artifact repositories for ease of use

aws

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Messaging utilities example in C#

```
/// <summary>
/// Event that represents when a customer was
/// made a preferred customer.
/// </summary>
public class CustomerMadePreferred
{
    public string CustomerId { get; }
    public DateTime MadePreferredOn { get; }

    public CustomerMadePreferred(string customerId,
                                 DateTime madePreferredOn)
    {
        CustomerId = customerId;
        MadePreferredOn = madePreferredOn;
    }
}
```

- Producer defines a `CustomerMadePreferred` class represents the event details
- The event is modeled to reflect the business domain
- Does not include `Metadata`

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Separate domain logic from integration needs

```
/// <summary>
/// Event that represents when a customer was
/// made a preferred customer.
/// </summary>
public class CustomerMadePreferred
{
    public string CustomerId { get; }
    public DateTime MadePreferredOn { get; }

    public CustomerMadePreferred(string customerId,
                                 DateTime madePreferredOn)
    {
        CustomerId = customerId;
        MadePreferredOn = madePreferredOn;
    }
}
```



© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Messaging utilities example in C#

```
namespace MyCompany.Shared.Eventing;

/// <summary>
/// IPublishEvents interface defines implementation to
/// publish any event class.
/// </summary>
public interface IPublishEvents
{
    Task PublishEvent<TEvent>(TEvent @event,
                              string source,
                              string detailType,
                              string eventBusName);
}
```

The utility library defines a generic interface `IPublishEvents` to publish any event class via the implementation

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Messaging utilities example in C#

```
public class Detail<T>
{
    public Metadata Metadata { get; private set; }
    public T Data { get; private set; }

    public Detail(T data)
    {
        Metadata = new Metadata(Guid.NewGuid(), DateTime.Now);
        Data = data;
    }
}
```

```
public class Metadata
{
    public Guid EventId { get; }
    public DateTime EventTime { get; }
    public string? Location { get; }

    public Metadata(Guid eventId, DateTime eventTime)
    {
        EventId = eventId;
        EventTime = eventTime;
        Location = GetLocation();
    }

    private static string? GetLocation()
    {
        return Environment.MachineName;
    }
}
```

The `Detail` class is also generic and includes a `Metadata` property; the Metadata class defines the structure and initializes the properties for every event that gets sent via the `IPublishEvents` implementation

aws

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Messaging utilities example in C#

```
public void PublishEvent<TEvent>(TEvent e, string source, string eventBusName)
    {
        // Use the name of the event class "e" to define the DetailType
        var detailType = e.GetType().Name;

        // Construct the Detail payload for the event with metadata
        var detail = new Detail<TEvent>(e);

        var jsonSerializerOptions = new JsonSerializerOptions { IncludeFields = true };

        // Create a new PutEvent Request
        var request = new PutEventsRequest
        {
            Entries = new List<PutEventsRequestEntry>
            {
                new()
                {
                    Source = source,
                    Detail = JsonSerializer.Serialize(detail, jsonSerializerOptions),
                    DetailType = detailType,
                    EventBusName = eventBusName,
                }
            }
        };

        // Using the PutEvent call in Amazon EventBridge SDK
        _amazonEventBridgeClient.PutEventsAsync(request);
    }
```

- Finally, the implementation of the interface `IPublishEvents` sends the events via the AWS SDK

- The `PublishEvent` method takes the `CustomerMadePreferred` event parameter and creates a new `Detail` object which is serialized and used to construct a new `PutEventsRequestEntry`

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Build your own messaging utilities

Why stop at injecting metadata?

- Add validation (against published schema)
- Durable publisher – deal with errors gracefully and ensure events are delivered “eventually”
- Mechanisms for delivering multiple versions of an event
- Enforce naming conventions
- Validate event size before it's delivered

aws
© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Working with sensitive data

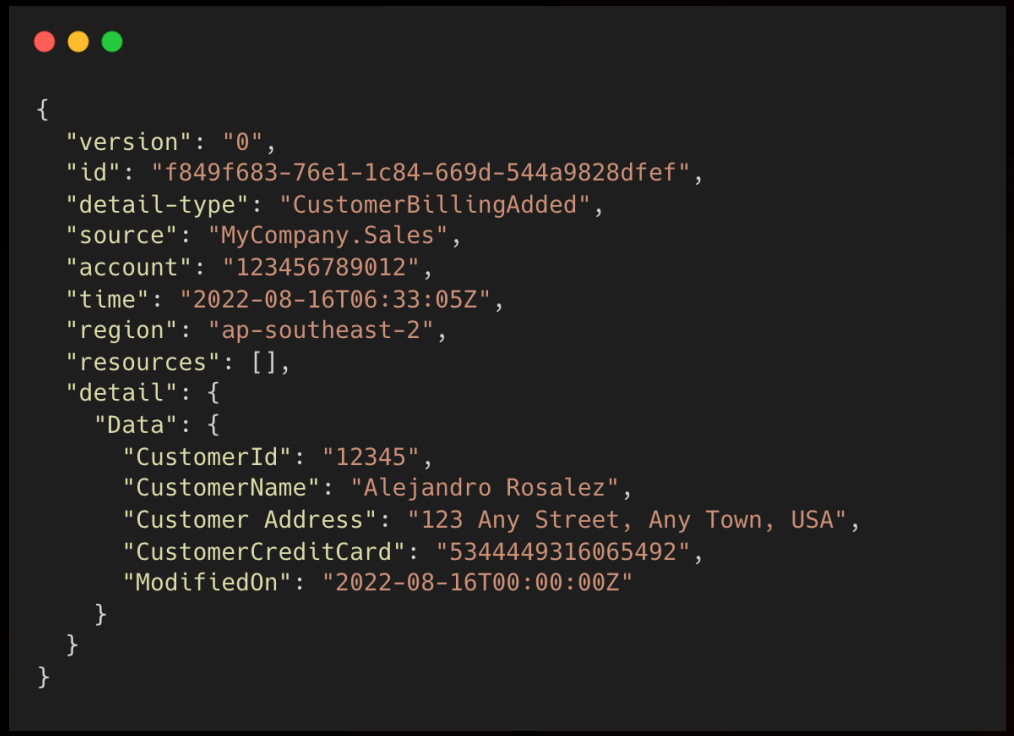
- Personally identifiable information in events requires **encryption**
- Amazon **EventBridge already protects events in transit and at rest**, but this is only point-to-point
- Where is the encryption applied?
- Do I handle it the middleware? Offload the responsibility to a service?



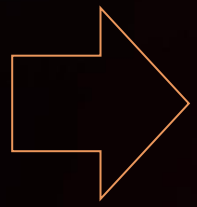
aws
© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

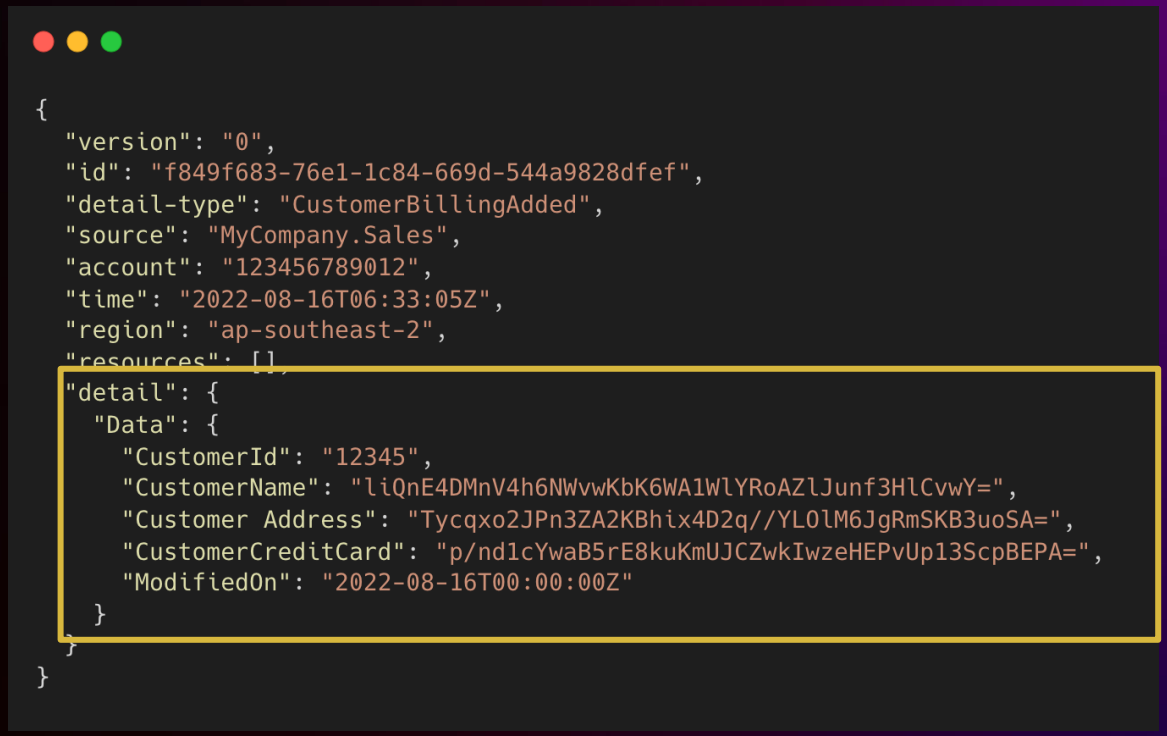
Working with sensitive data

ENCRYPTED PAYLOAD



```
{
  "version": "0",
  "id": "f849f683-76e1-1c84-669d-544a9828dfef",
  "detail-type": "CustomerBillingAdded",
  "source": "MyCompany.Sales",
  "account": "123456789012",
  "time": "2022-08-16T06:33:05Z",
  "region": "ap-southeast-2",
  "resources": [],
  "detail": {
    "Data": {
      "CustomerId": "12345",
      "CustomerName": "Alejandro Rosalez",
      "Customer Address": "123 Any Street, Any Town, USA",
      "CustomerCreditCard": "5344449316065492",
      "ModifiedOn": "2022-08-16T00:00:00Z"
    }
  }
}
```

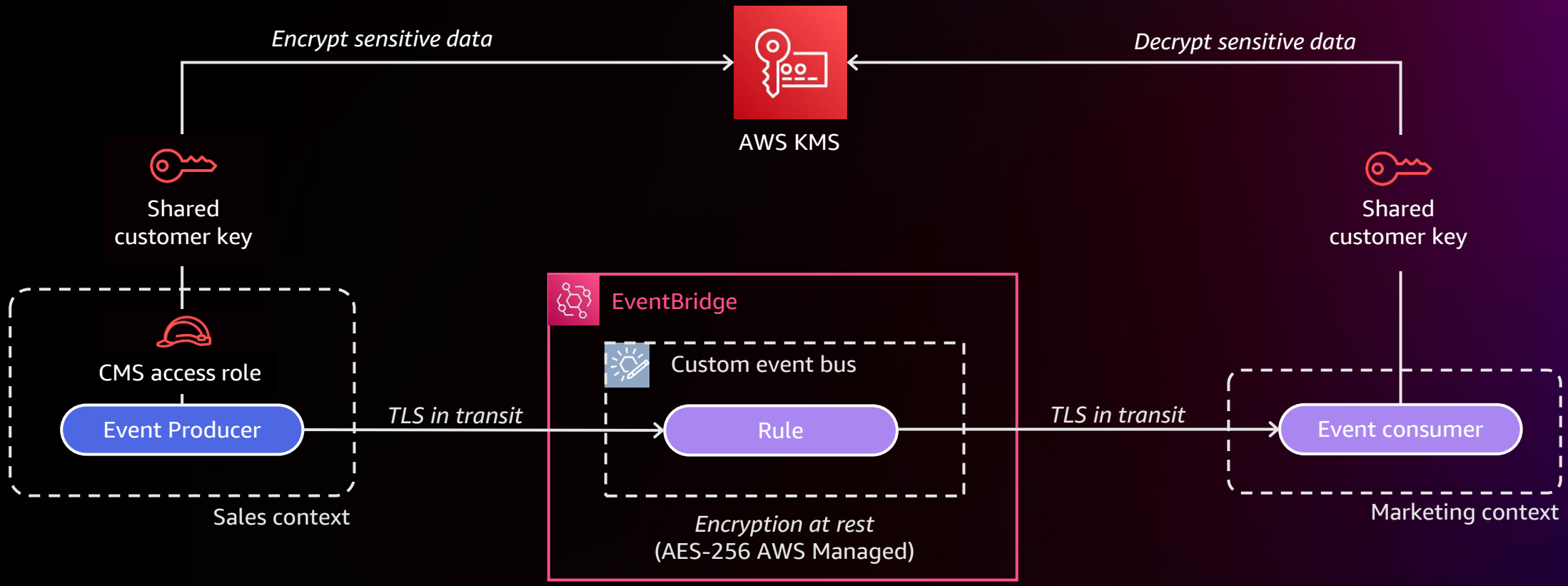




```
{
  "version": "0",
  "id": "f849f683-76e1-1c84-669d-544a9828dfef",
  "detail-type": "CustomerBillingAdded",
  "source": "MyCompany.Sales",
  "account": "123456789012",
  "time": "2022-08-16T06:33:05Z",
  "region": "ap-southeast-2",
  "resources": [],
  "detail": {
    "Data": {
      "CustomerId": "12345",
      "CustomerName": "liQnE4DMnV4h6NWvwKbK6WA1WlYRoAZlJunf3HlCvwY=",
      "Customer Address": "Tycqxo2JPn3ZA2KBhix4D2q//YLOlM6JgRmSKB3uoSA=",
      "CustomerCreditCard": "p/nd1cYwaB5rE8kuKmUJCZwkIwzeHEPvUp13ScpBEPA=",
      "ModifiedOn": "2022-08-16T00:00:00Z"
    }
  }
}
```

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Working with sensitive data

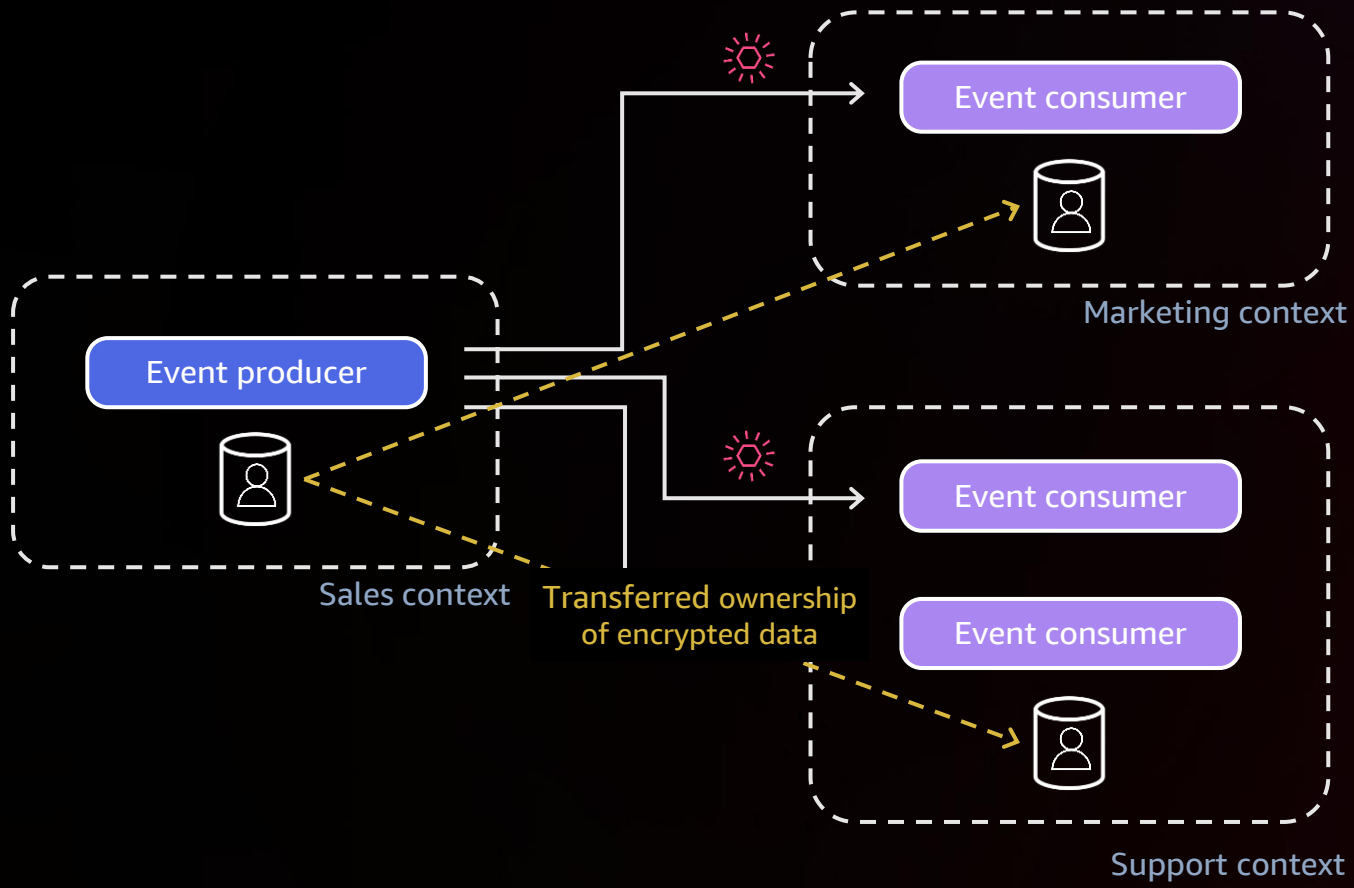


Extend the “contract” by explicitly sharing KMS CMK

Re-encrypt data with a key that the consumer manages

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Working with sensitive data

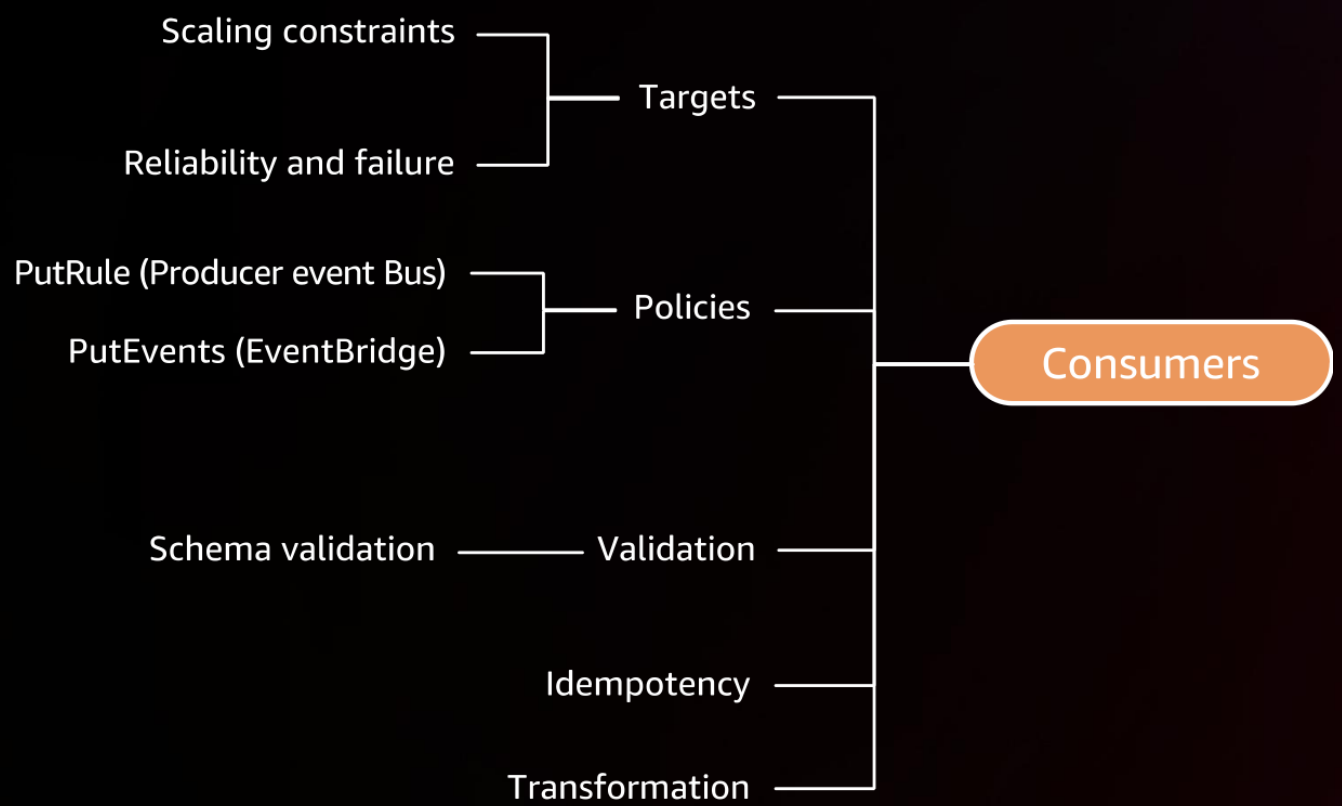


Ask yourself: Is sharing sensitive data absolutely necessary?

- Duplicating sensitive information **across multiple domain** boundaries
- The custodianship of the sensitive information becomes the **responsibility of multiple parties**
- Be mindful of your **domain boundaries** – who owns the data? Have you modeled correctly?

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Consumers



- Create **self-describing rule names**
- Plan your **ingestion rate** – you have no control over how quickly producers emit events
- If you need to validate events, **use the published schema, cache locally**
- Plan for failure! This includes retries; make your consumers **idempotent**

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Naming conventions for event bus rules

- Rule names need to be unique! If you don’t have a strategy for this your deployments will fail
- Operations can be assisted with rule tagging, but because rule tags cannot be set using Amazon CloudFormation or AWS CDK, this has limited value at this point in time
- Design a naming convention for the organization to adopt
- Not just rules – also event buses, target names and so on; be mindful that you will be deploying these across different environments (development, test, production)

aws © 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Naming conventions for event bus rules

Who is subscribing to the event

Where has the event come from



What target is being invoked

What is the name of the event

```
com.marketing.CmpHandlerFunction-com.sales.CustomerMadePreferred
```

(Max 64 character limit for rule names)

aws

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

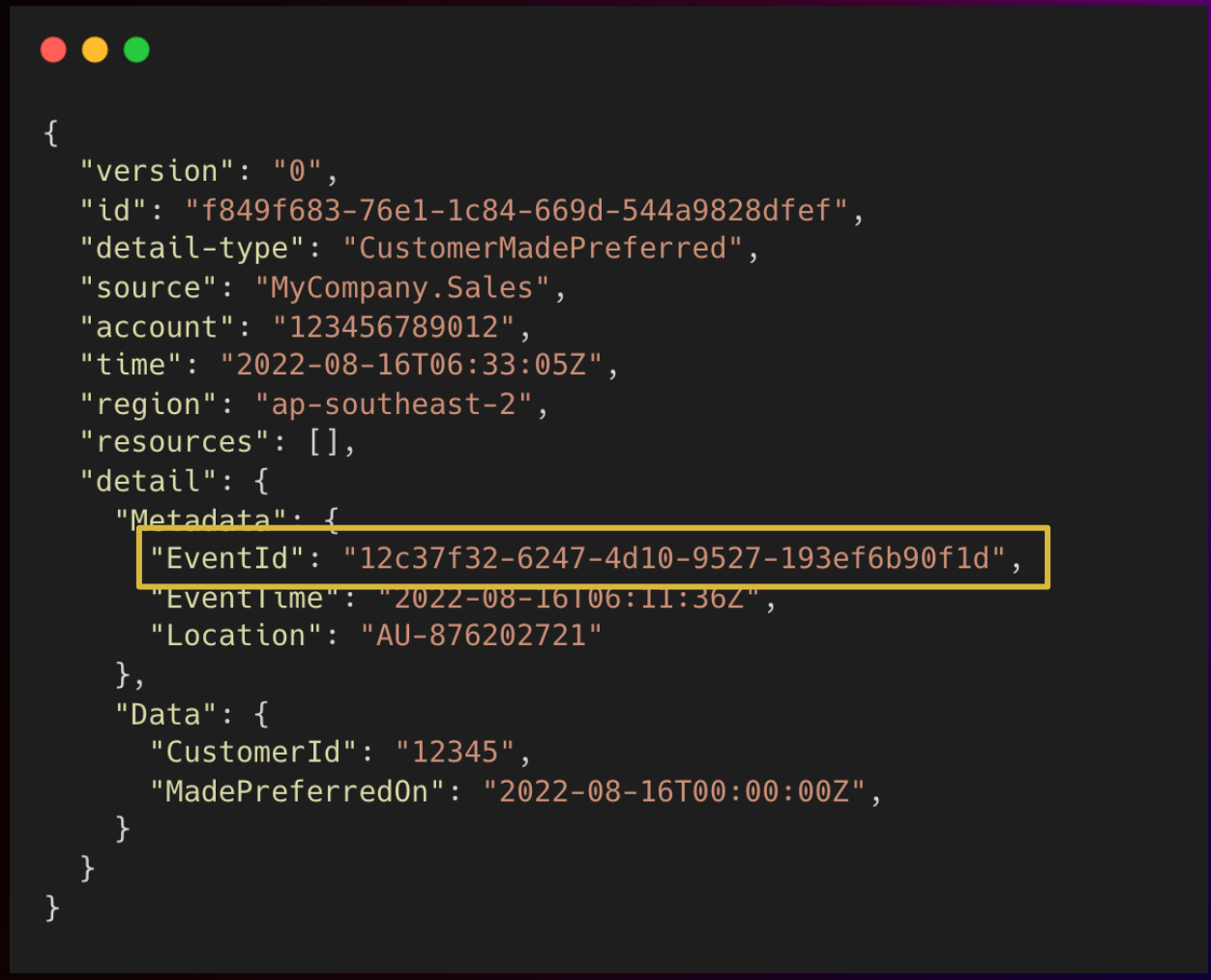
Event uniqueness

Idempotency

Operation will return the same results whether it is called once or multiple times (EIP)

Idempotency Key

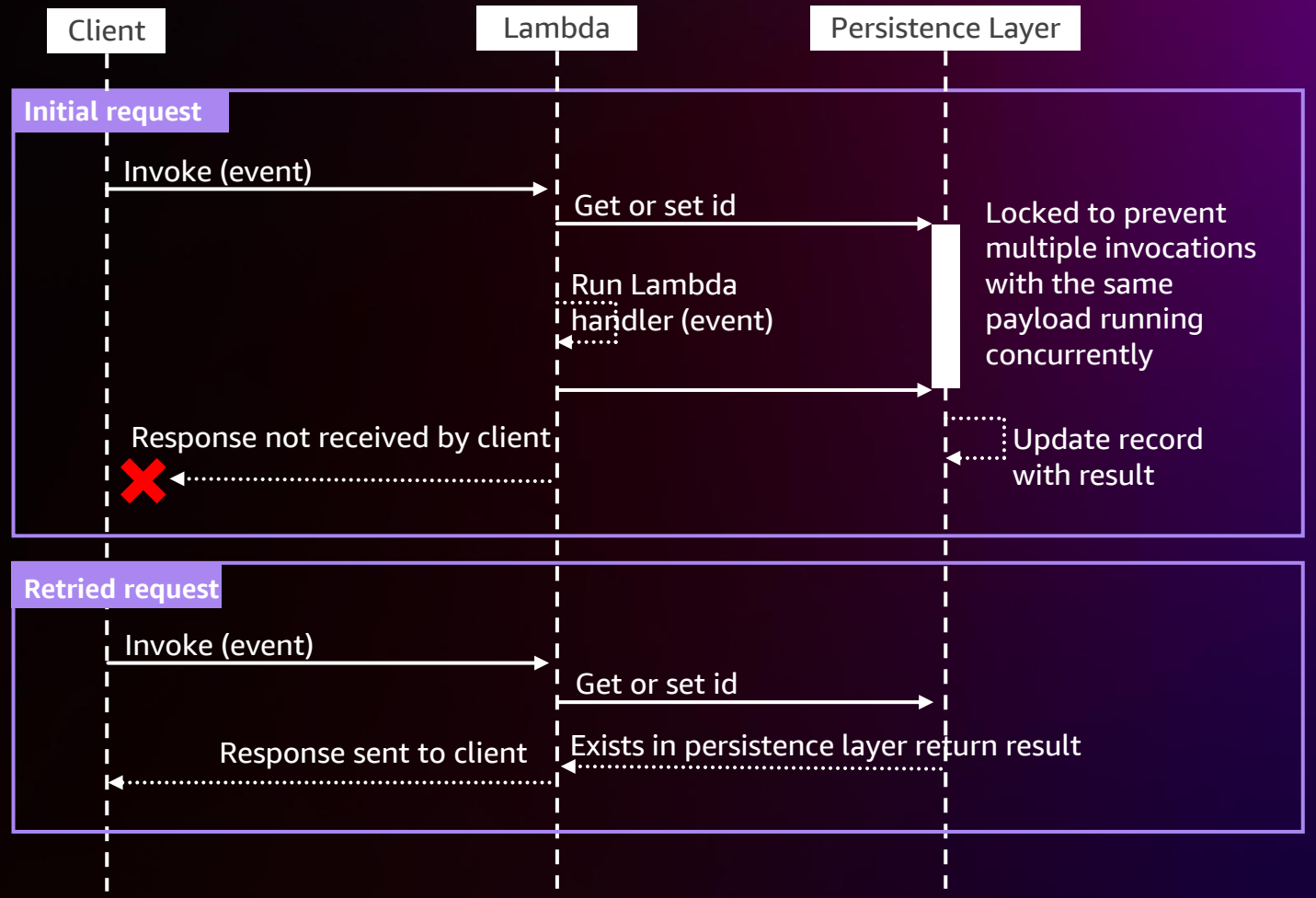
Assigned to the message by the sender to simplify deduplication by the receiver



© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Event deduplication strategies

- Manage idempotent state in an external data store
- Specify a time window for deduplication
- AWS Lambda Powertools Python Idempotency library
- Making retries safe with idempotent APIs (AWS Builders' Library)

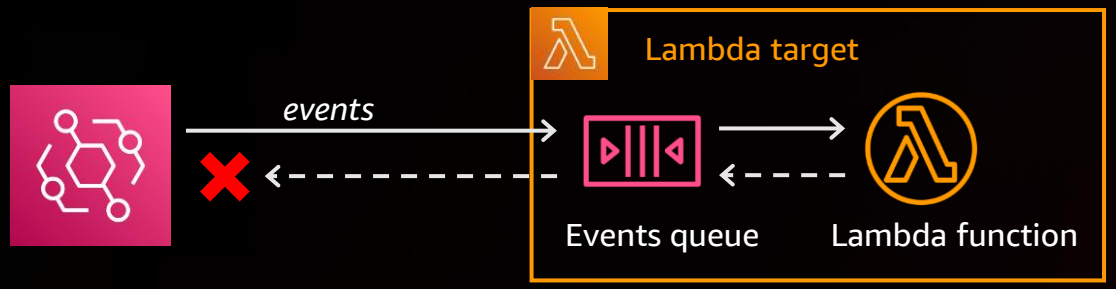


(Lambda Powertools Python Idempotency, https://s12d.com/powertools-py-idempotency)

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

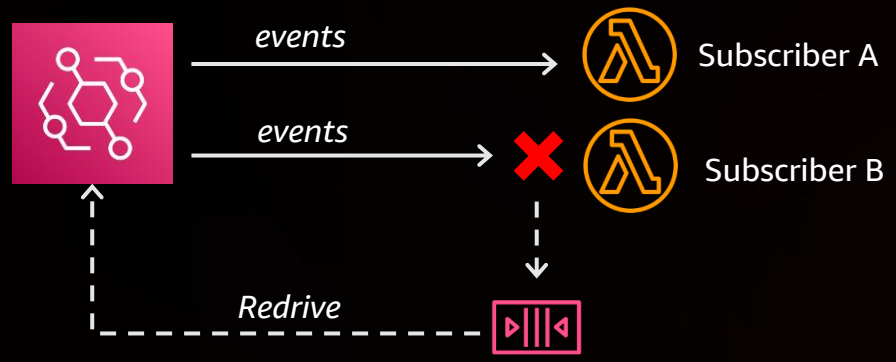
Idempotency from an EventBridge perspective

At runtime (idempotency window < 5 min)

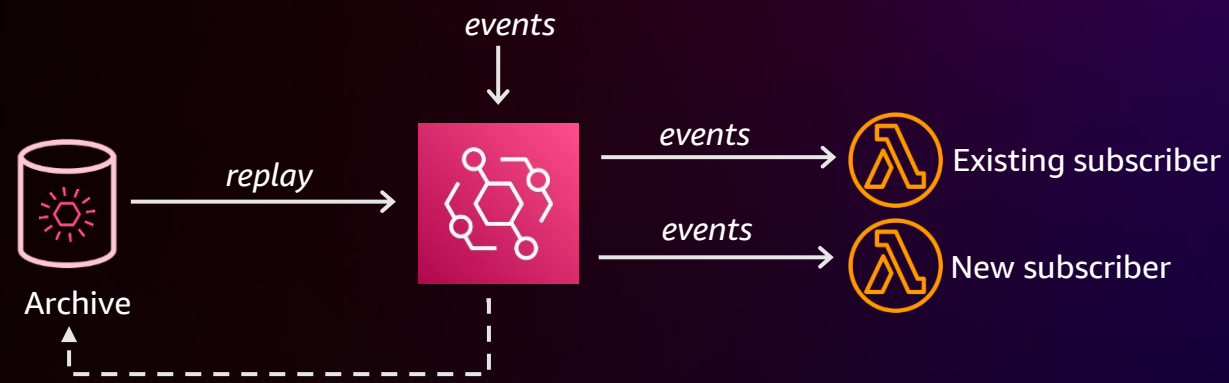


Idempotency window may vary depending on the use case

DLQ redrive (idempotency window > 5 min)



Replaying events (idempotency window > 5 min)



aws
© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Idempotency from an EventBridge perspective

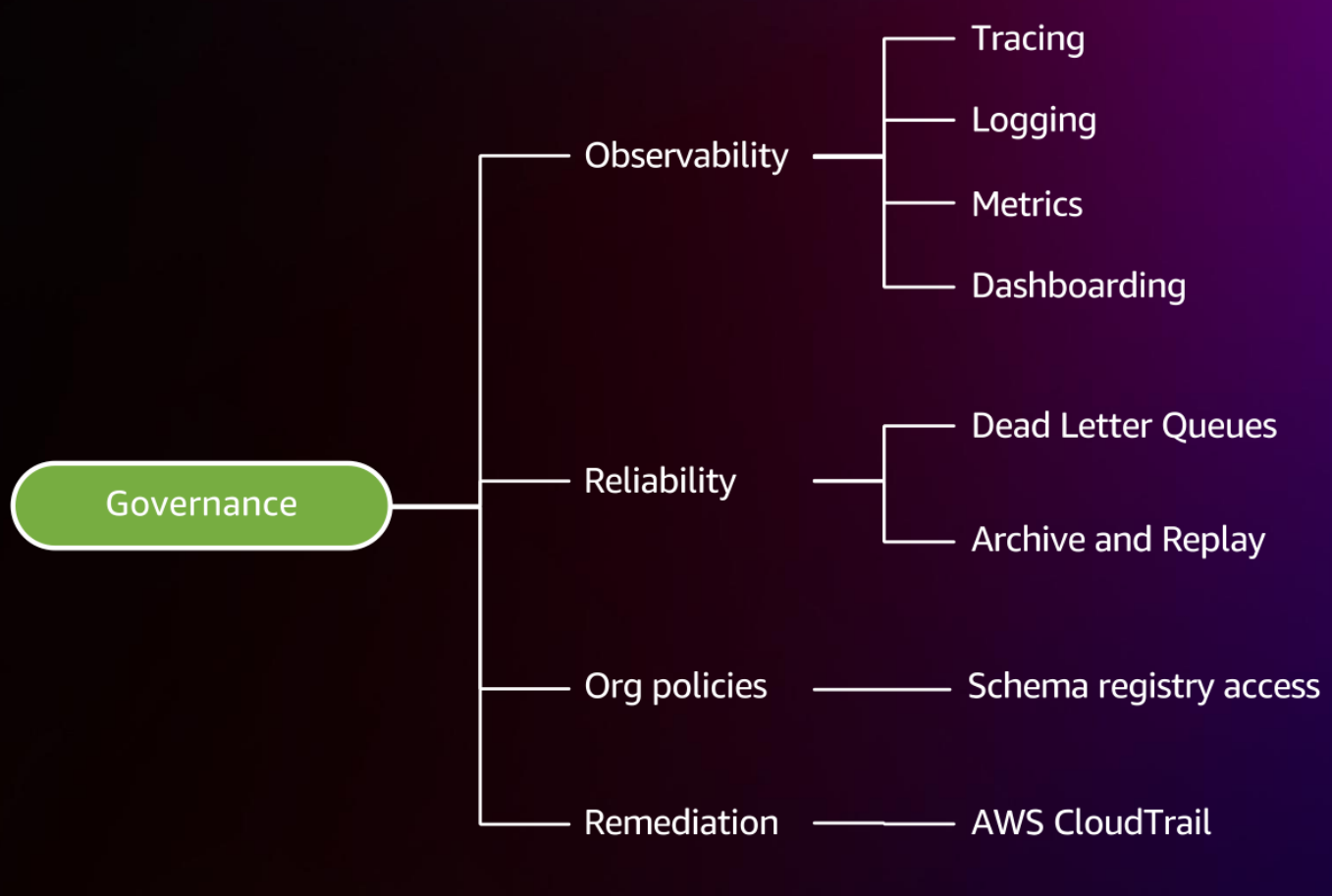
- Always assume that you will process messages **more than once**
- **Partitions don’t only occur on the network** – also in database layers and disk IO
- **Deduplication and ordering is not a guarantee** – they help, but don’t forget to look at your business processes for answers around how to deal with events that get processed more than once
- Apply idempotency controls on the critical parts of your system – **Sometimes it’s easier to just deal with duplicates!**

aws

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Governance

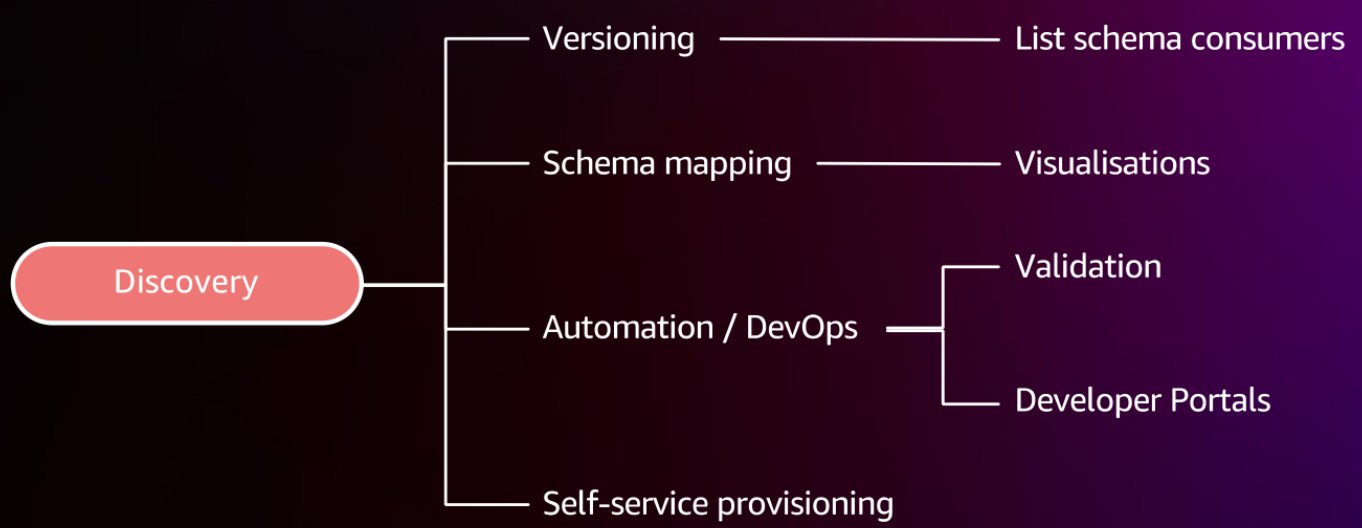
- Leverage utility libraries like AWS Lambda Powertools (Python, Java, TypeScript, .NET) and apply best practices for tracing, logging and metrics
- EventBridge supports AWS X-Ray trace propagation allowing you to capture requests as they flow through your event bus across different stacks
- Audit resource management through AWS CloudTrail – this can be automated to remediate violations in company standards



© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

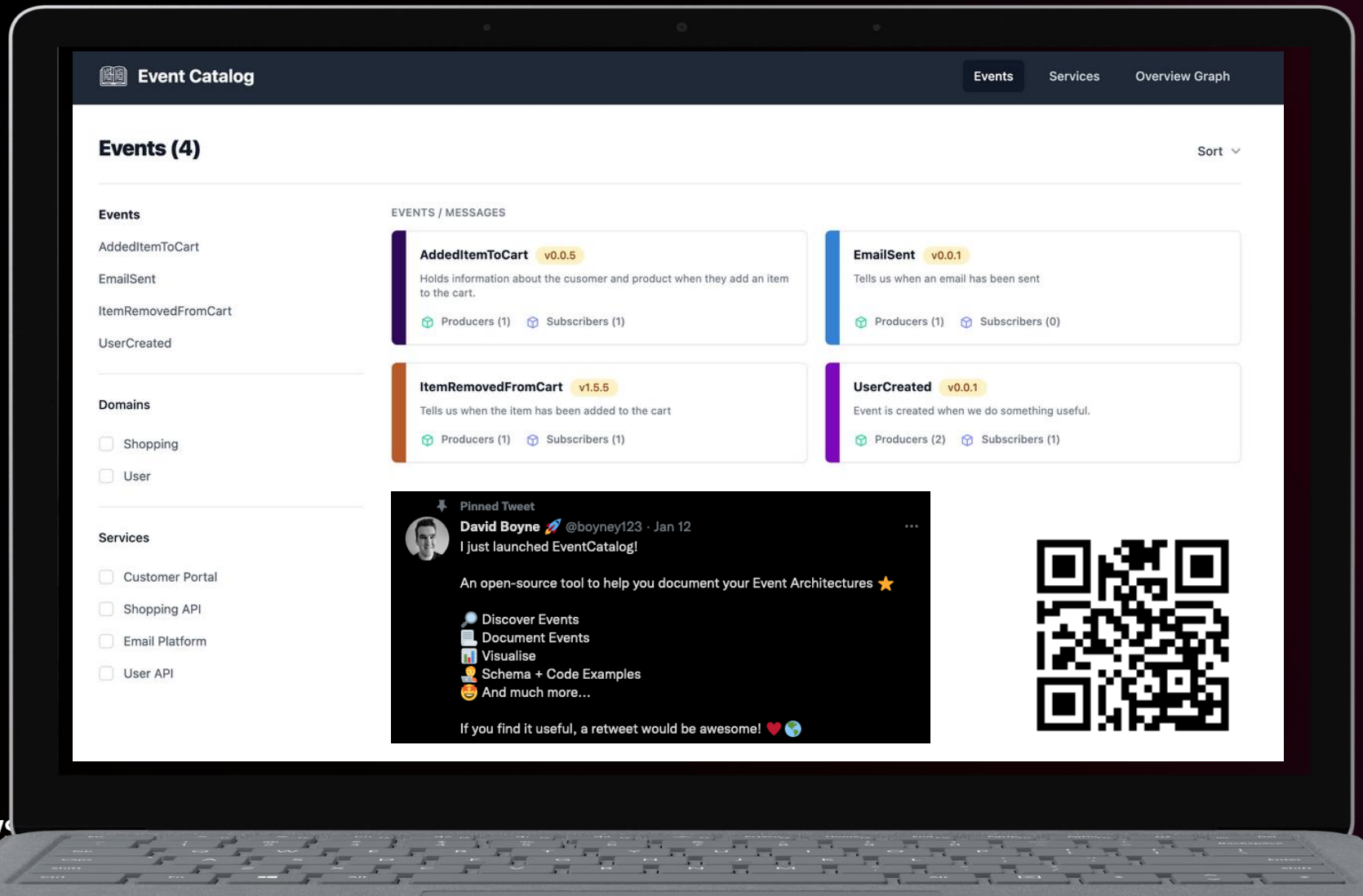
Discovery

- **Schemas** play a vital role in describing the structure of events
- They help in **communicating and understanding the relationships** of services
- **Explicit contacts** help us build dependency graphs to ensure consistency across system boundaries
- Lots more to do in this space



aws
© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Open source projects – Event Catalog



Features

- Transforms your Amazon EventBridge schemas, targets and rules into documentation
- Document and capture your events, services, schemas, code examples and much more
- Visualizer allows you to see the relationships between all events, publishers and subscribers
- Open source by David Boyne @boyney123

Key takeaways

- Really think about coupling! You're building a distributed event-driven system – there are trade-offs to make
- Pick the right type of event – spend time modeling events
- Focus on the needs and responsibilities of producers as well as how you consume events
- Make it easy for teams to integrate – create reusable libraries that abstract the integration requirements away and allow you to enforce consistency
- Create standards and policies that can be applied across the organization
- Think about how you want to document your event-driven architectures.

aws
© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Additional resources



Enterprise Integration Patterns

The book *Enterprise Integration Patterns*, written by Gregor Hohpe and Bobby Woolf, describes 65 patterns for the use of enterprise application integration and message-oriented middleware in the form of a pattern language; this website contains the catalogue of patterns and many other great resources



Use Amazon EventBridge to Build Decoupled, Event-Driven Architectures

Get an overview of EventBridge, the basics of event-driven architectures, a deeper dive into EventBridge capabilities, and how best to use it in your serverless applications



Using AWS X-Ray with Amazon EventBridge

Example application shows how to use AWS X-Ray with Amazon EventBridge, using the AWS Serverless Application Model (AWS SAM)



Amazon EventBridge resource policy samples

GitHub repository containing sample implementations for Amazon EventBridge resource policies, multi-account design patterns and cross-Region event routing

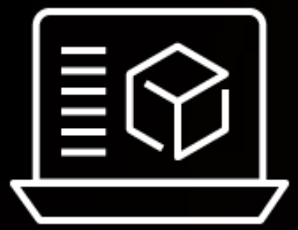
AWS Lambda Powertools for Python

A suite of utilities for AWS Lambda functions that makes tracing with AWS X-Ray, structured logging and creating custom metrics asynchronously easier; Powertools is available for Python, Java, TypeScript, and .NET

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Continue your AWS Serverless learning

Learn at your own pace



Expand your Serverless skills with our Learning Plan on AWS Skill Builder

Increase your knowledge



Use our Ramp-Up Guides to build your Serverless knowledge

Earn AWS Serverless badge



Demonstrate your Knowledge by achieving digital badges

https://s12d.com/serverless-learning

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Check out these other sessions

- API311: Building next-gen applications with event-driven architectures
  Monday (Nov 28) @ 10:00 AM – Wynn Encore Lafite 5

- API302: Building global event-driven applications
  Wednesday (Nov 30) @ 7:45 PM – Mandalay Bay Jasmine F

- API308: Are you integrating or building distributed applications?
  Wednesday (Nov 30) @ 1:45 PM – Caesar's Forum Summit 228

- SVS312: Building Serverlesspresso: Creating event-driven architectures
  Thursday (Dec 1) @ 3:30 PM – MGM Grand Chairman's Ballroom 355

aws
© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Thank you!

Stephen Liedig
@sliedigaws



Please complete the session survey in the **mobile app**

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.