
AWS
re:Invent

API302

Building event-driven architectures

Stephen Liedig
Sr. Serverless Specialist Solutions Architect
AWS

AWS re:Invent

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Agenda

Amazon EventBridge - a quick recap

Where do I start?

Event bus patterns and multi-account design considerations

Recommended practices



© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Amazon EventBridge recap



AWS re:Invent

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

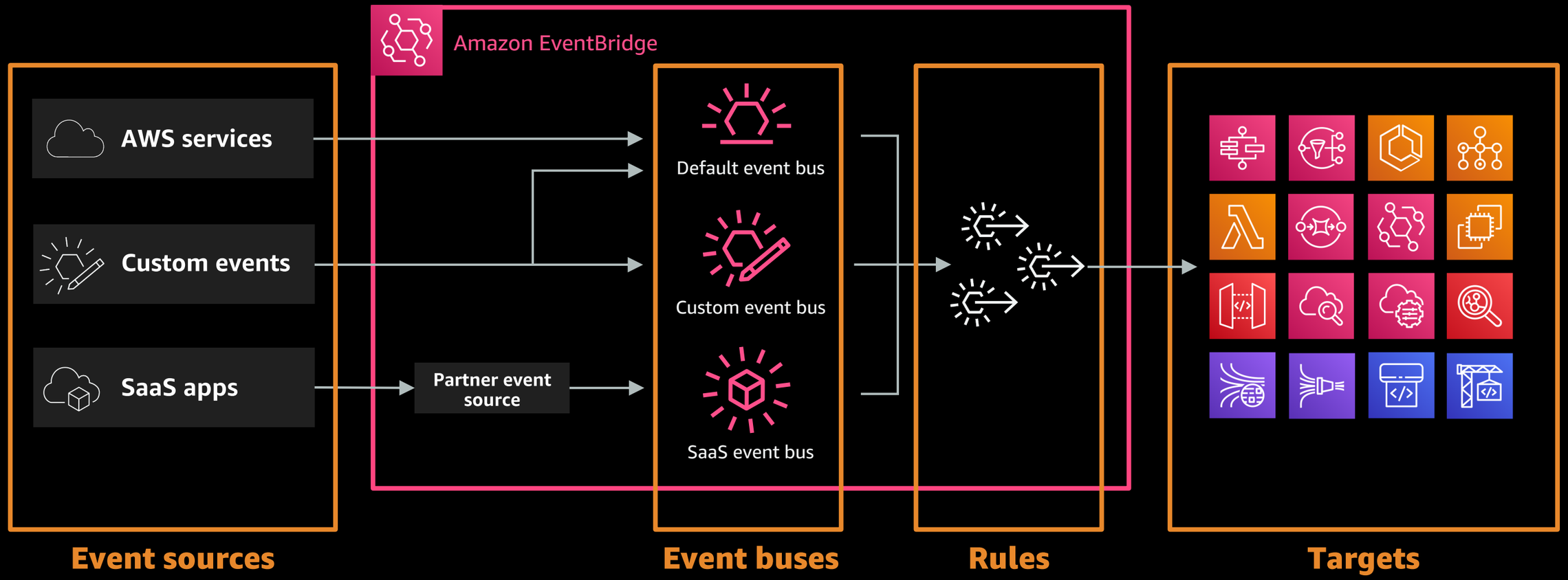


Amazon EventBridge

A serverless event bus service for AWS services, your own applications, and SaaS providers

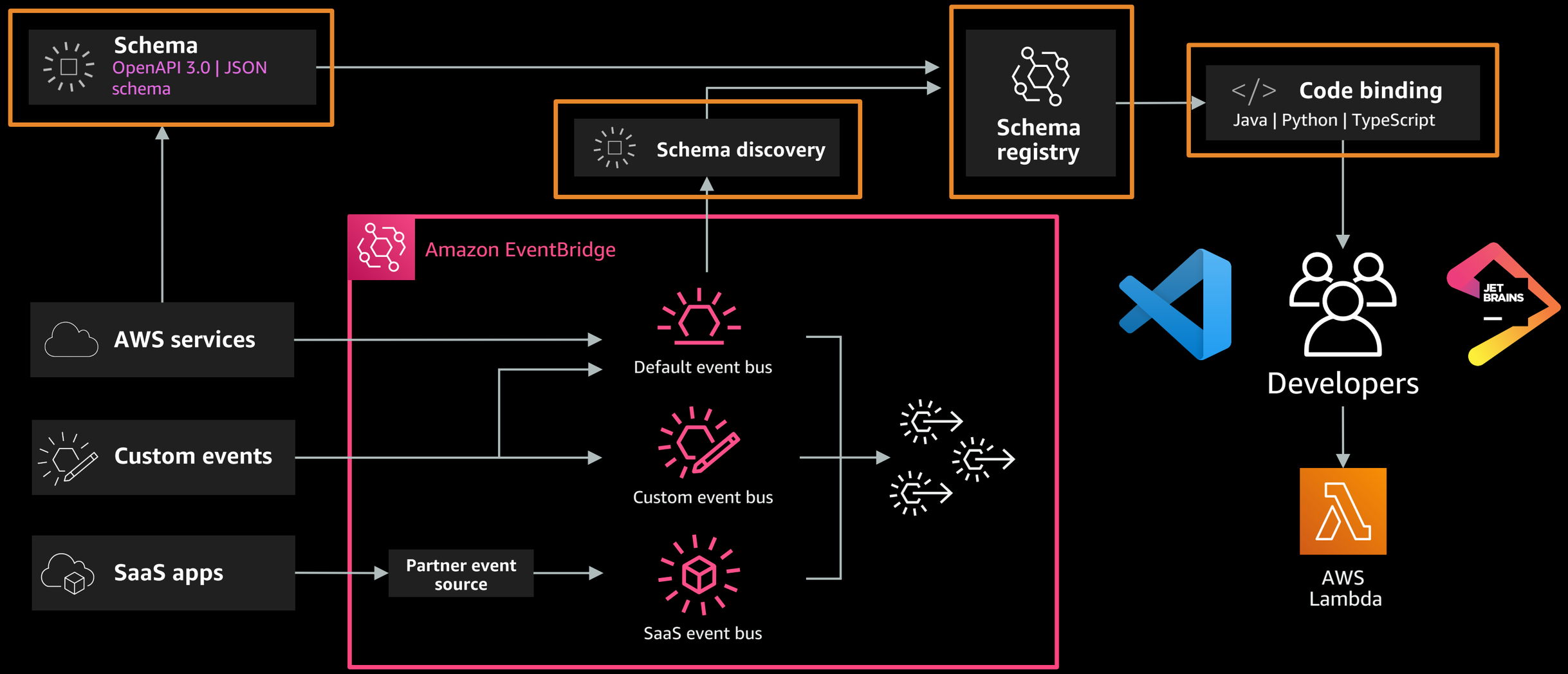
- Serverless—pay only for the events you process
- Simplified scaling avoids increasing costs to sustain and manage resources
- No upfront investments, ongoing licensing, or maintenance costs
- No specialist knowledge needed

Amazon EventBridge architecture

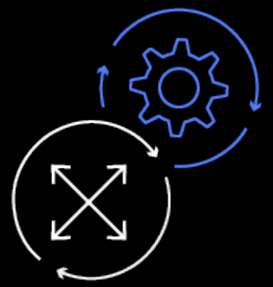


Amazon EventBridge architecture

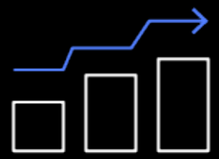
AMAZON EVENTBRIDGE SCHEMA REGISTRY



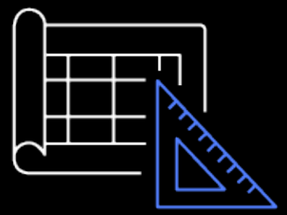
What are the benefits?



Loose coupling



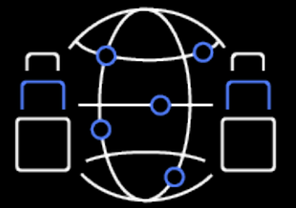
Scalability



Extensibility



Complexity



Availability

Where do I start?

Do I use a single bus?

Single account?

Multi-account?

Do I use multi-buses?

How do my choices affect team structure?

What about security?

What “best practices” should I be adopting?

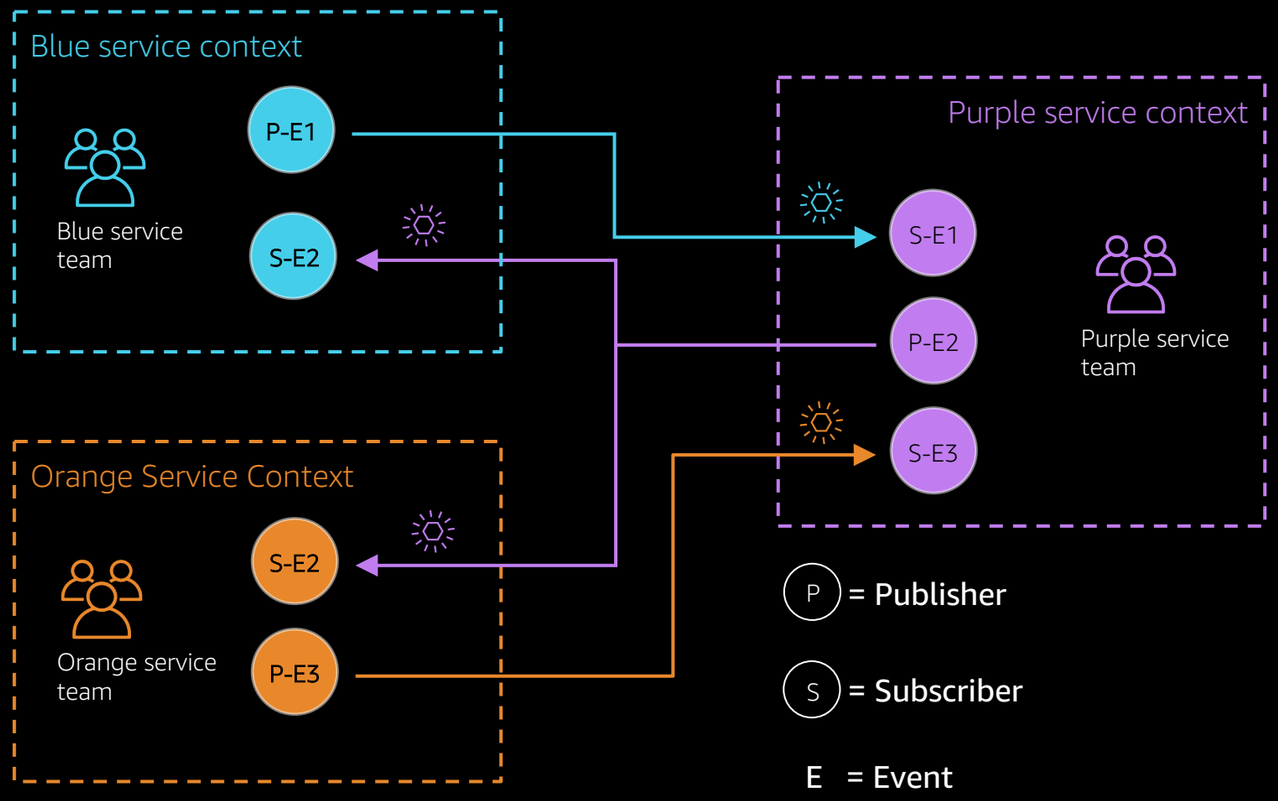
Event bus patterns



AWS re:Invent

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Logical architecture



Identify business events, processes, actors, etc.

Cluster-related concepts

Define bounded contexts and sub-domains

Identified and modeled their integration requirements

At this point, there is no physical architecture

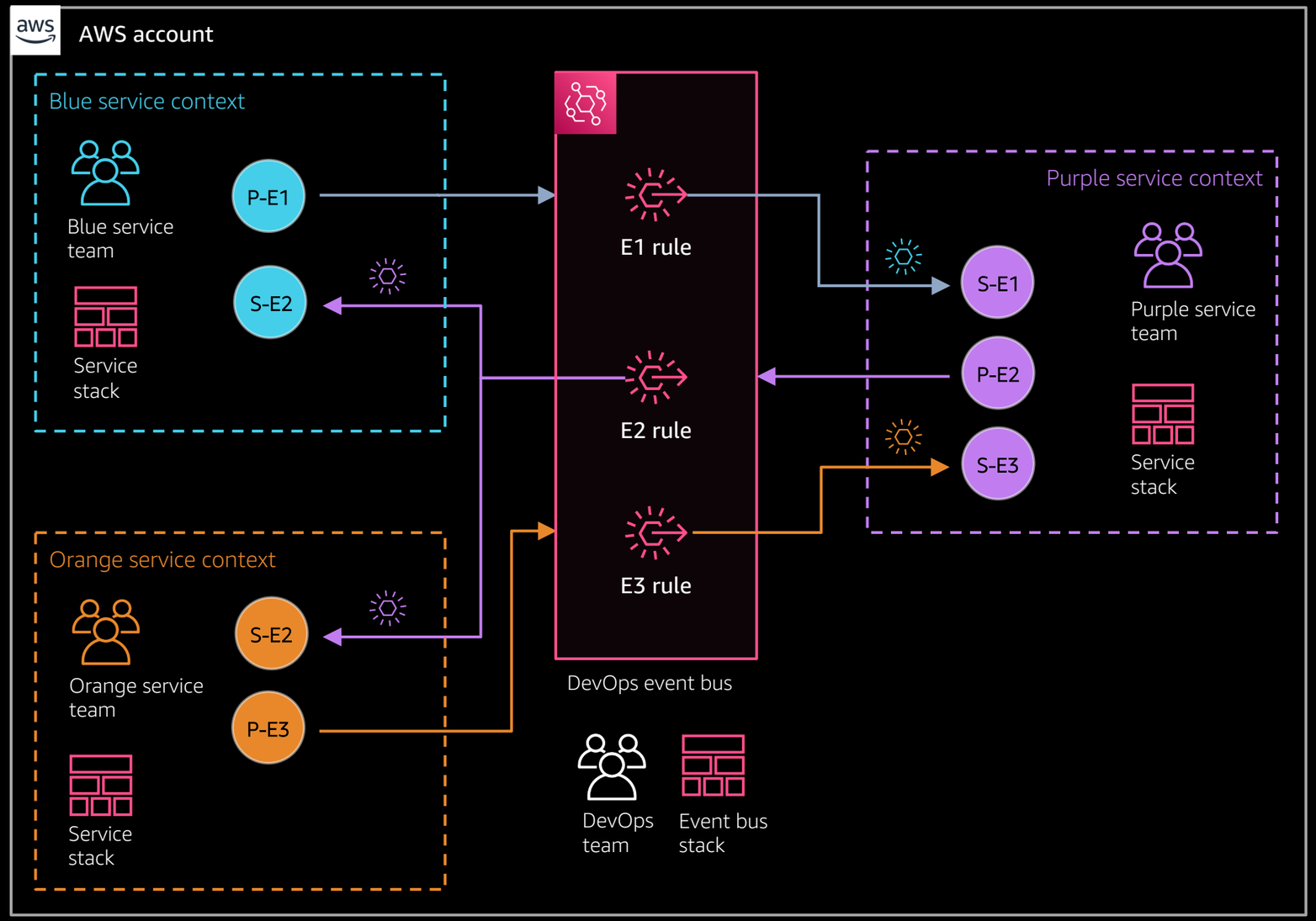
Architectural approaches

CENTRALIZED VS DISTRIBUTED

	Advantages	Disadvantages
Centralized	• All routing centralized, concentrating all communications to a single event bus • Enables central resource management, permissions, and deployment • Allows you to easily integrate applications with minimal changes	• All routing centralized • Prevents apps from gaining autonomy, leaky business logic • Single point of failure
Distributed	• Publisher owns the event bus • Designed to allow independent evolution of sources and destinations • No single point of failure • Greater support service autonomy	• More difficult to design distributed solutions than centralized ones • More resources to manage

Single-bus, single-account pattern

CENTRALIZED



Characteristics

- All events from services routed through a single bus in an account
- Target services can be invoked easily

Team

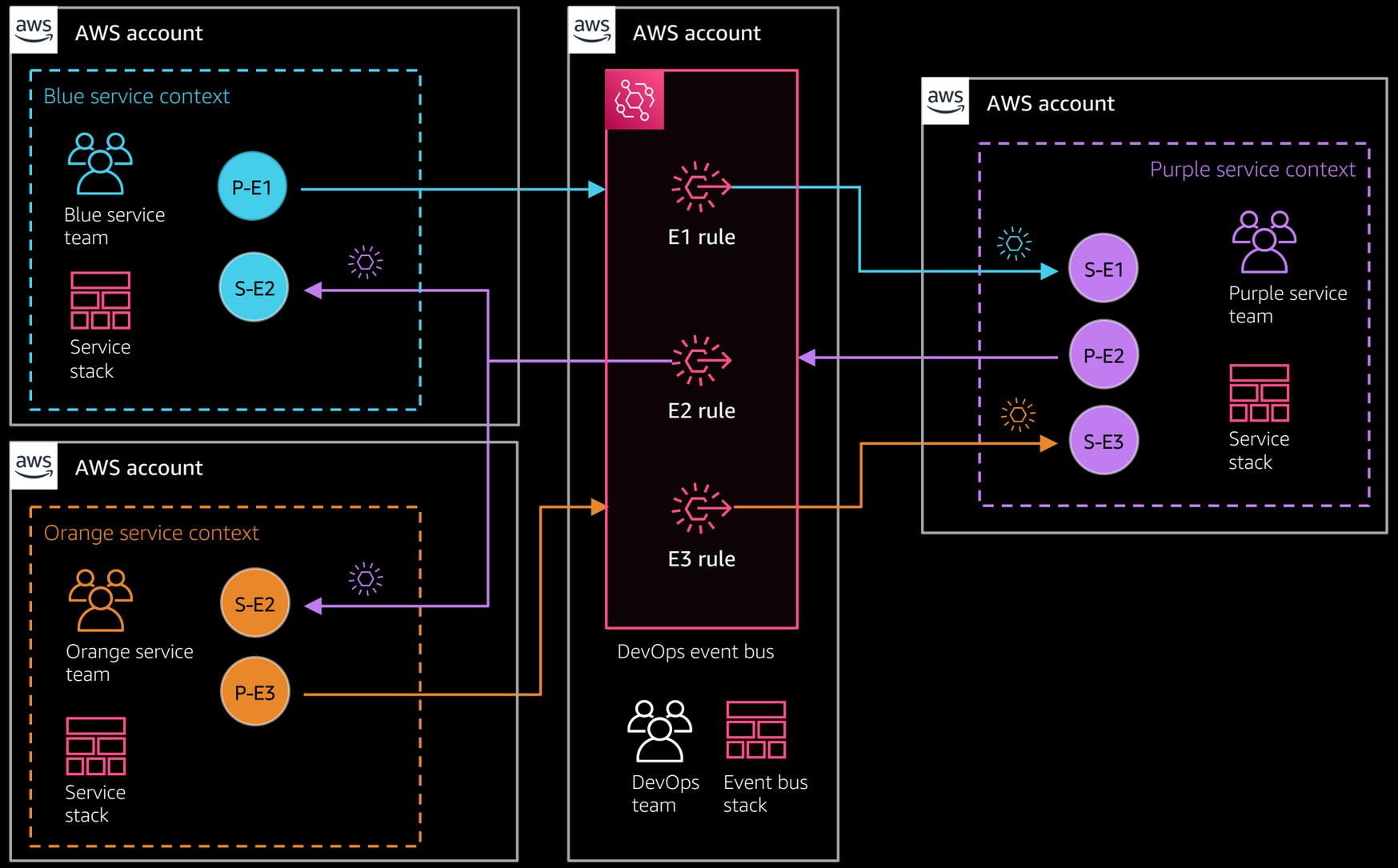
- Event bus resource stack managed by “DevOps” team
- Service teams collaborate with DevOps teams to coordinate resource needs

Considerations

- Tradeoff resource management with development velocity

Single-bus, multi-account pattern

CENTRALIZED



Characteristics

- Architecturally identical, except for AWS account boundary

Team

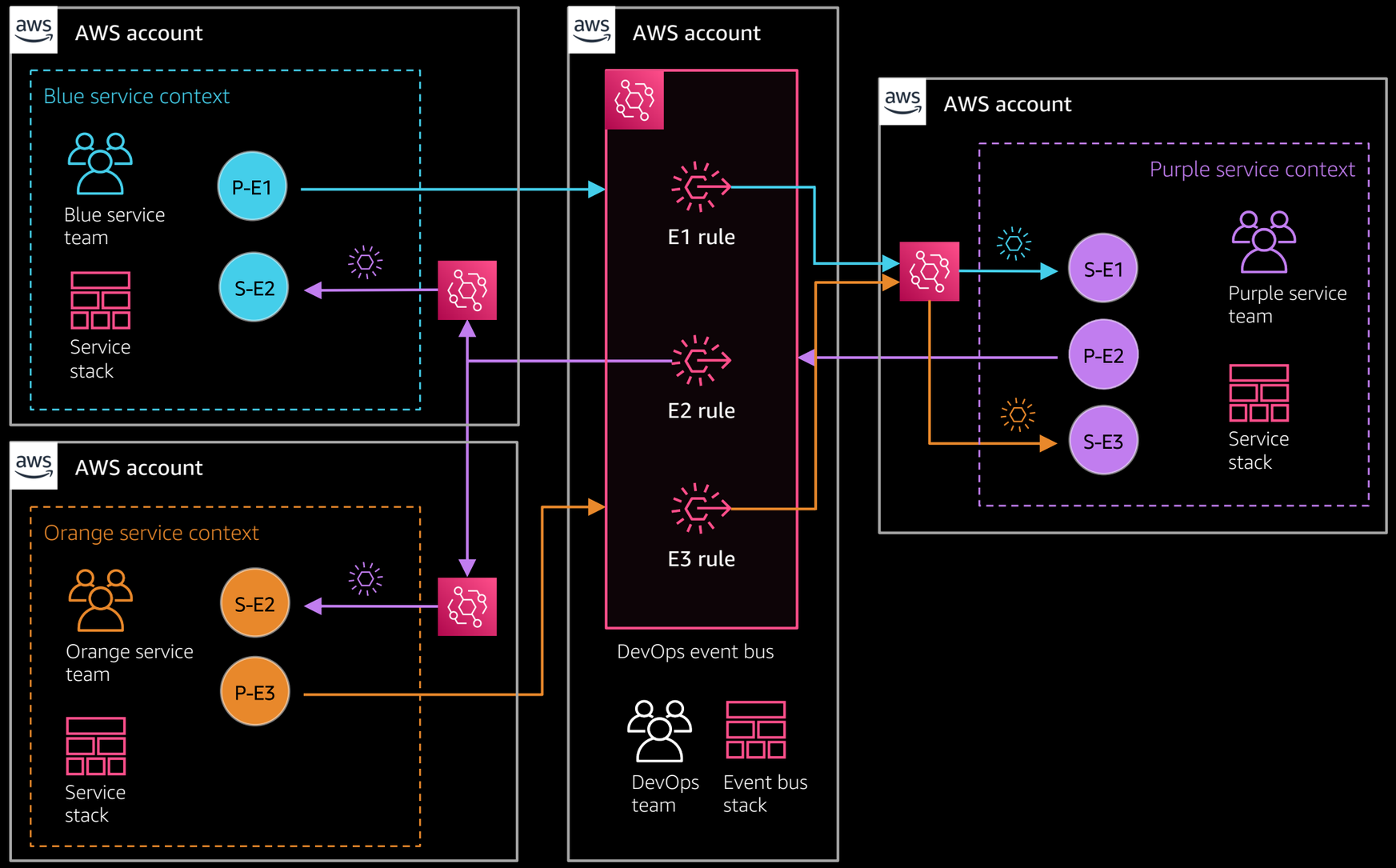
- Event bus resource stack is managed by “DevOps” team
- Service teams collaborate with DevOps teams to coordinate resource needs

Considerations

- Tradeoff - Ease of management with development velocity

Single-bus*, multi-account pattern

CENTRALIZED



Characteristics

- Introduces the need for multiple event buses to transfer event between accounts
- Routing rules still via central bus
- Target rules migrate to service account event bus

Team

- Service teams manage target configurations, but not routings

Considerations

- Additional cross-account policy management

EventBridge resource policies

Previously:

- All EventBridge API calls were scoped to caller account. Caller had to assume an IAM role to interact with event bus or rule in another account.
- Event bus resource policy only supported *PutEvents* action.
- Resource policy only used for cross-account event bus targets.

Now:

- API calls (e.g., *PutEvents*, *PutRule*, *PutTarget*) can reference an event bus in another account.
- Resource policy supports all API actions (except *PutPermissions*).
- Support for organizations and tags in resource policy.

Event bus resource policy - *PutRule*

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "LimitedRuleCreation",
      "Effect": "Allow",
      "Action": "events:PutRule",
      "Principal": {
        "AWS": "[PURPLE-SERVICE-ACCOUNT]"
      },
      "Resource": [
        "arn:aws:events:us-east-1:[DEVOPS-ACCOUNT]:rule/devops-event-bus/*"
      ],
      "Condition": {
        "StringEquals": {
          "events:creatorAccount": "${aws:PrincipalAccount}",
          "events:detail-type": "E2",
          "events:source": "purple.service.context"
        }
      }
    }
  ]
}
```

Give Purple Team account permissions to create rules on “DevOps” event bus.

Limit to rules that match against the “E2” event from “com.purple.service”

Event bus resource policy - *PutEvents*

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "CrossAccountPublish",
      "Effect": "Allow",
      "Action": [
        "events:PutEvents"
      ],
      "Principal": {
        "AWS": "[PURPLE-SERVICE-ACCOUNT]"
      },
      "Resource": "arn:aws:events:us-east-1:[DEVOPS-ACCOUNT]:event-bus/devops",
      "Condition": {
        "StringEquals": {
          "events:detail-type": "E2",
          "events:source": "purple.service.context"
        }
      }
    }
  ]
}
```

Give Purple Team account permissions to publish events to “DevOps” event bus.

Only allow the Purple Team to publish “E2” events from “com.purple-service.context”

Event bus resource policy – organizations & tags

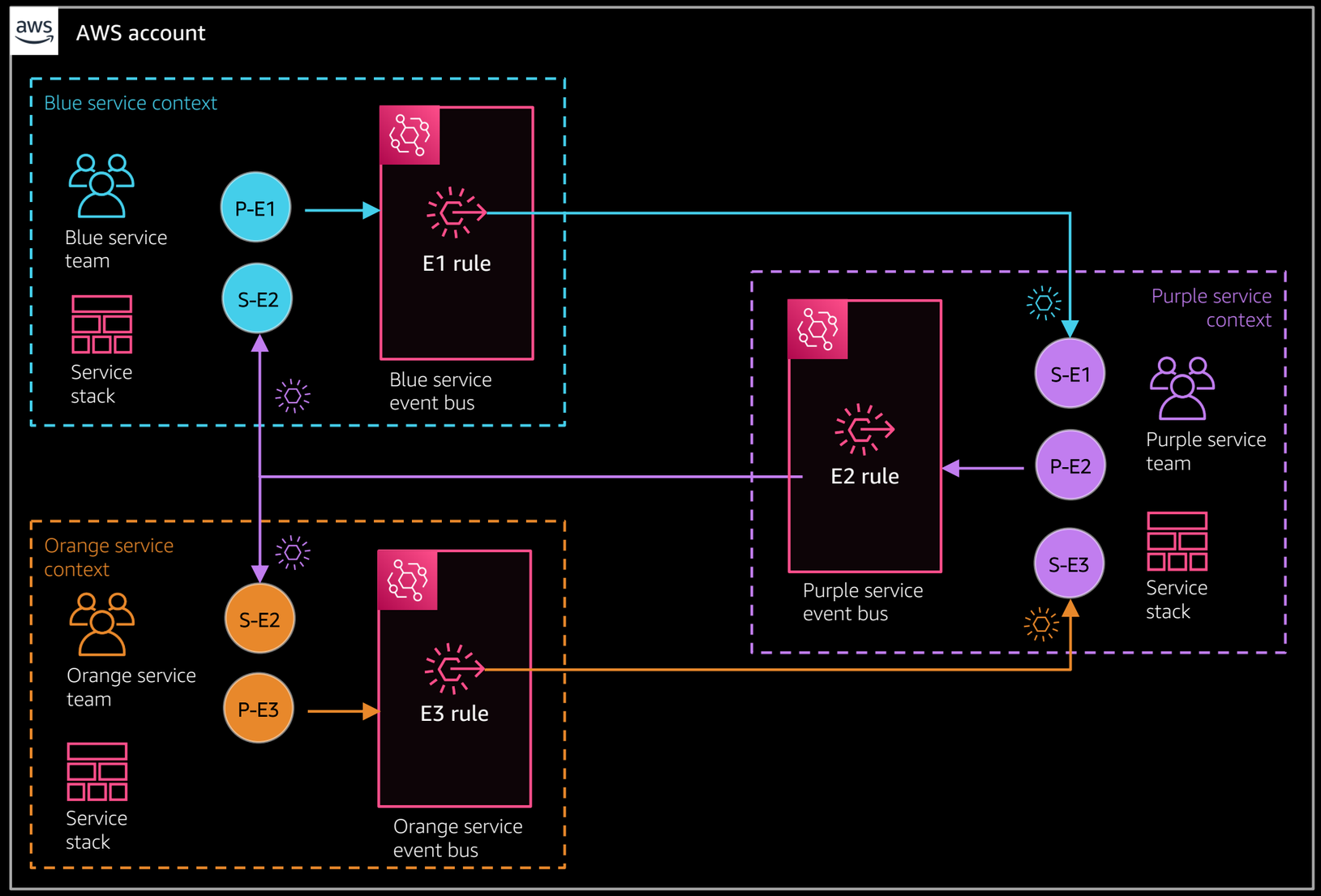
```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "CrossAccountPublish",
      "Effect": "Allow",
      "Action": [
        "events:PutEvents"
      ],
      "Principal": {
        "AWS": "[PURPLE-SERVICE-ACCOUNT]"
      },
      "Resource": "arn:aws:events:us-east-1:[DEVOPS-ACCOUNT]:event-bus/devops",
      "Condition": {
        "StringEquals": {
          "aws:PrincipalOrgID": "o-a1b2c3d4e5",
          "aws:PrincipalTag/Production": "true"
        }
      }
    }
  ]
}
```

Give any account permissions to publish events to “DevOps” event bus.

Limit publishing to accounts in the “o-a1b2c3d4e5” organization, and that are also tagged as “Production” accounts.

Multi-bus, single account pattern

DISTRIBUTED



Characteristics

- Multiple event buses owned by the service teams
- No centralized routing

Team

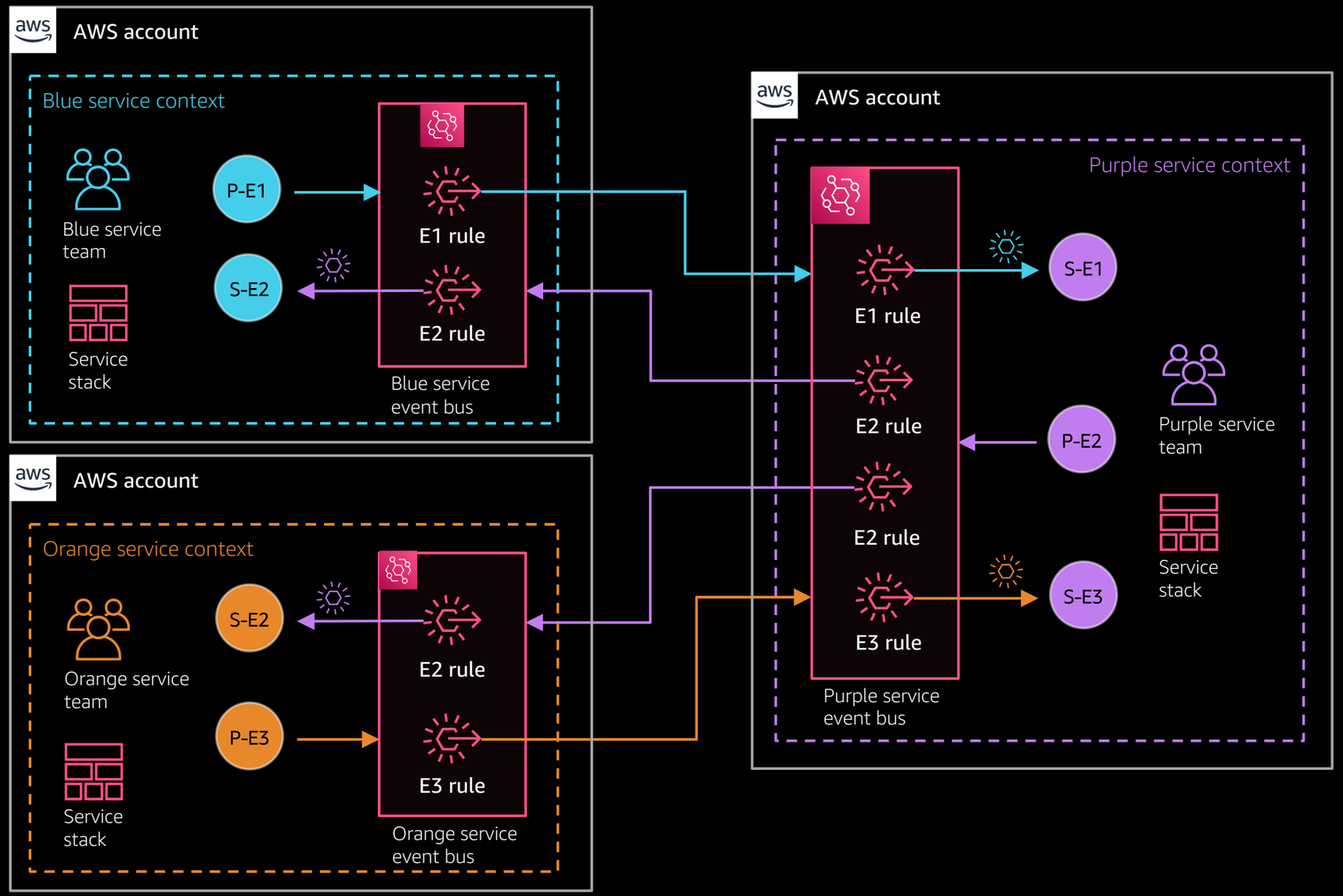
- Teams manage their service event buses
- Service teams collaborate with each other to coordinate resource needs

Considerations

- Increased application and team autonomy
- Increased development velocity

Multi-bus, multi-account pattern

DISTRIBUTED



Characteristics

- Architecturally identical, except for AWS account boundary
- No additional buses required to facilitate cross account event delivery

Team

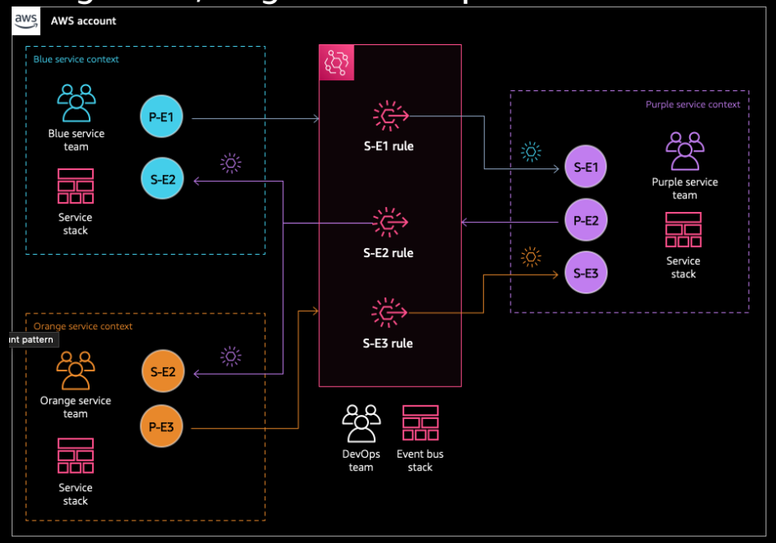
- Service teams manage their service event buses and all resources for sending and receiving events

Considerations

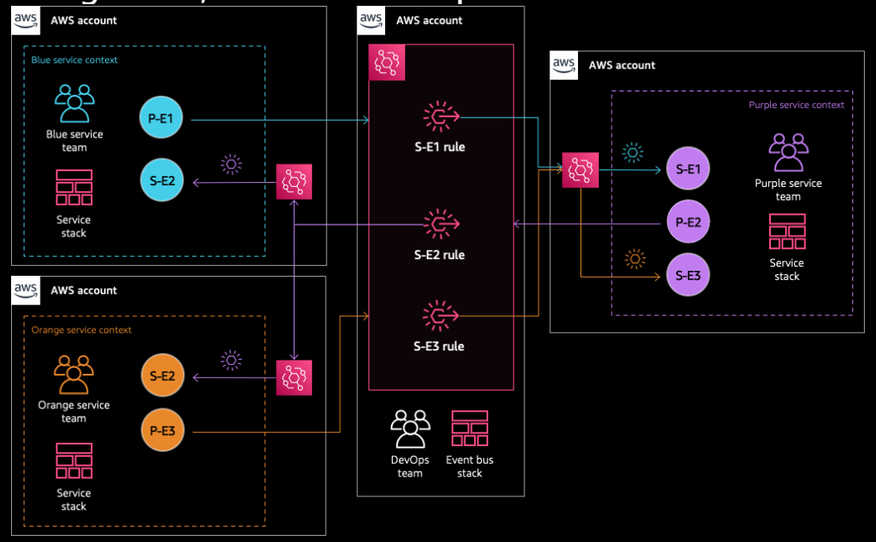
- Additional overhead in managing distributed rules

Event bus patterns

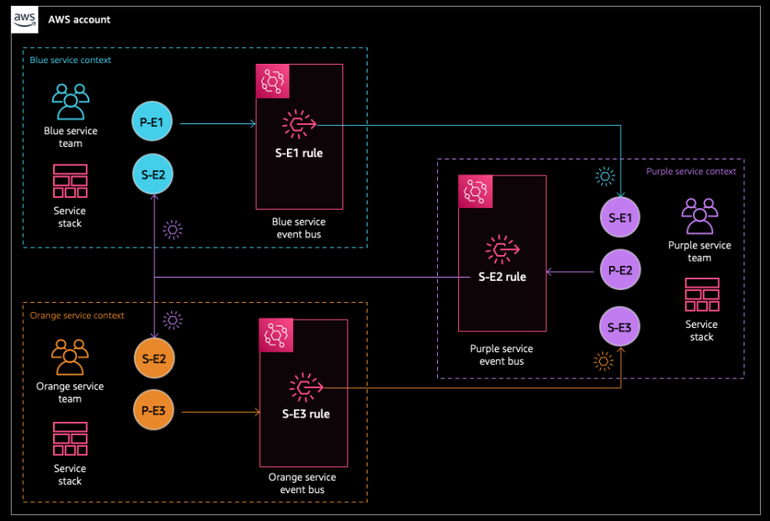
Single-bus, single-account pattern



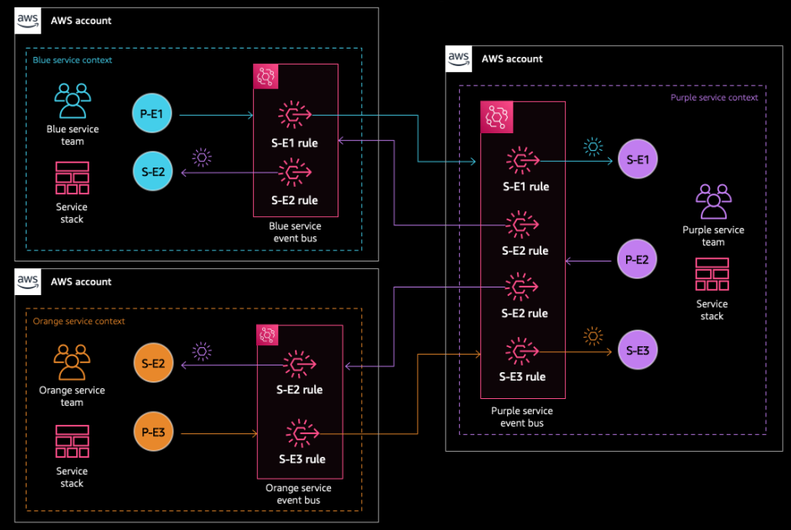
Single-bus, multi-account pattern



Multi-bus, single account pattern

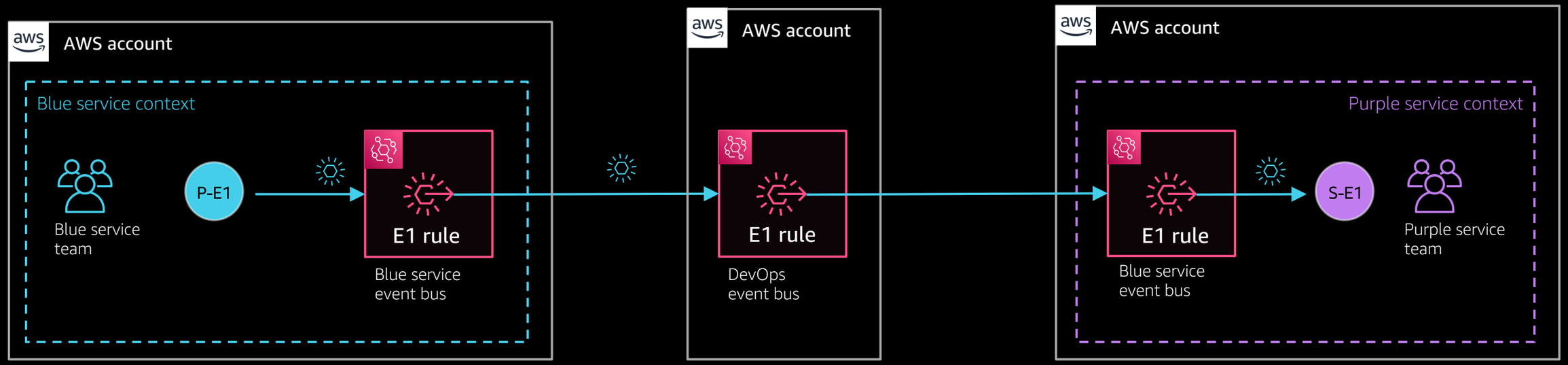


Multi-bus, multi-account pattern



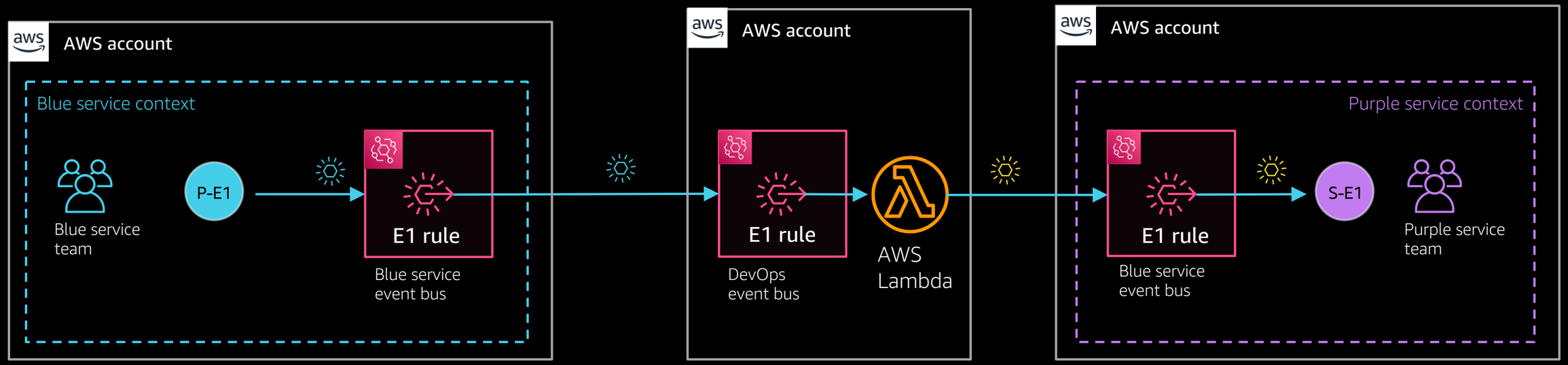
Cross account loop (bus hopping)

Amazon EventBridge does not support transitive routing



Cross account loop (bus hopping)

Amazon EventBridge does not support transitive routing



Events need to be republished by proxy (AWS Lambda) making it a new event with new *id*, *source*, *account* ≈ new event!

Recommended practices



© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Recommended practices using EventBridge

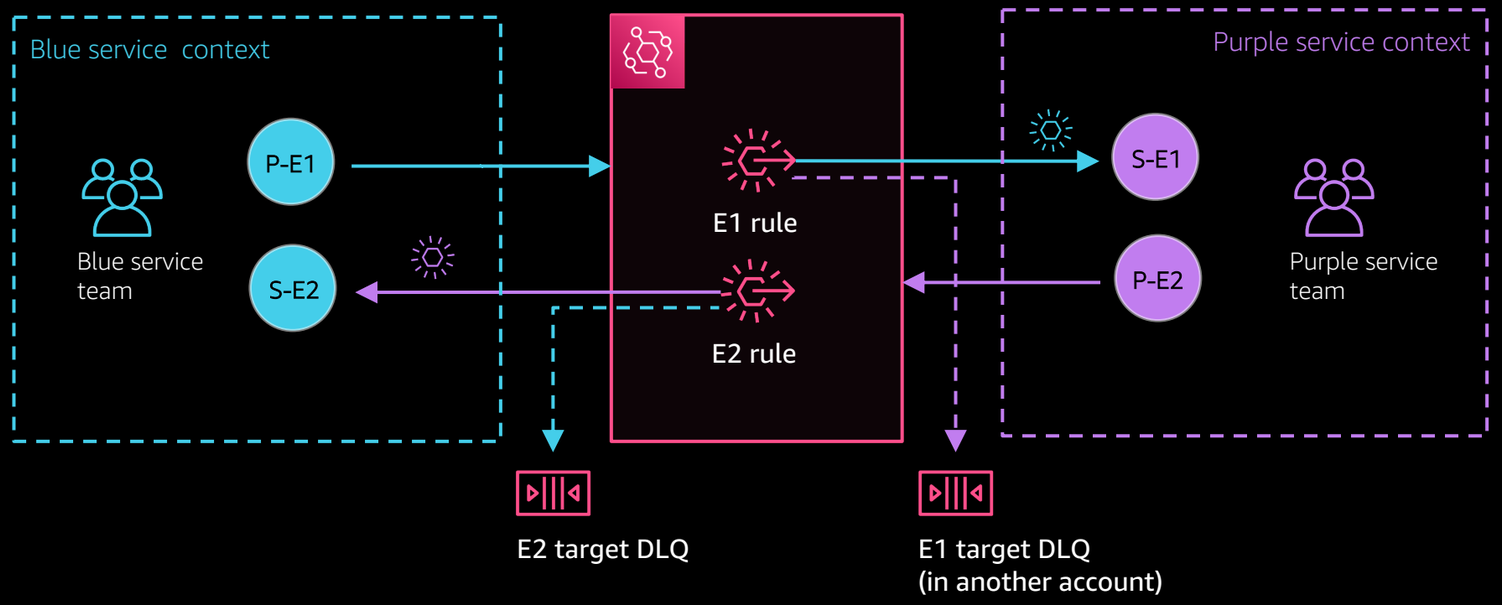
UNLESS YOU KNOW BETTER ONES

- Use dead letter queues
- Avoid using the default event bus for custom application events
- Create “subscriptions”—one target per rule

Amazon EventBridge dead letter queues

DON'T LOSE EVENTS AND UNDERSTAND ROOT CAUSE

EventBridge now supports DLQ and custom retry policy (maximum # of retries or the maximum event age of the event) via customer-managed Amazon SQS queue



Possible root causes?

- Permissions not correct
- Service availability
- Deleted resource
- Throttling
- Cross account loop
- Invalid parameters

Amazon EventBridge dead letter queues

DON'T LOSE EVENTS AND UNDERSTAND ROOT CAUSE

The DLQ and Customer Retry Policy are configured per EventBridge target and via the `PutTargets` API

DLQs for Amazon EventBridge come with AWS Console, AWS CLI and AWS CloudFormation support

```
# Create your favourite rule
aws events put-rule \
  --event-bus-name blue-service-bus \
  --name E1-rule \
  --event-pattern "{\"source\": [\"E1\"]}" \
  --region us-east-1

# Create your favorite target with DLQ/Retry Policy configured
aws events put-targets \
  --rule E1-rule \
  --targets '{"Id":"1", "Arn": "arn:aws:lambda:us-east-1:123456789012:function:non-existing",
    "DeadLetterConfig": {"Arn": "arn:aws:sqs:us-east-1:123456789012:blue-service-dlq"},
    "RetryPolicy": {"MaximumRetryAttempts": 1, "MaximumEventAgeInSeconds": 300 }}' \
  --region us-east-1
```

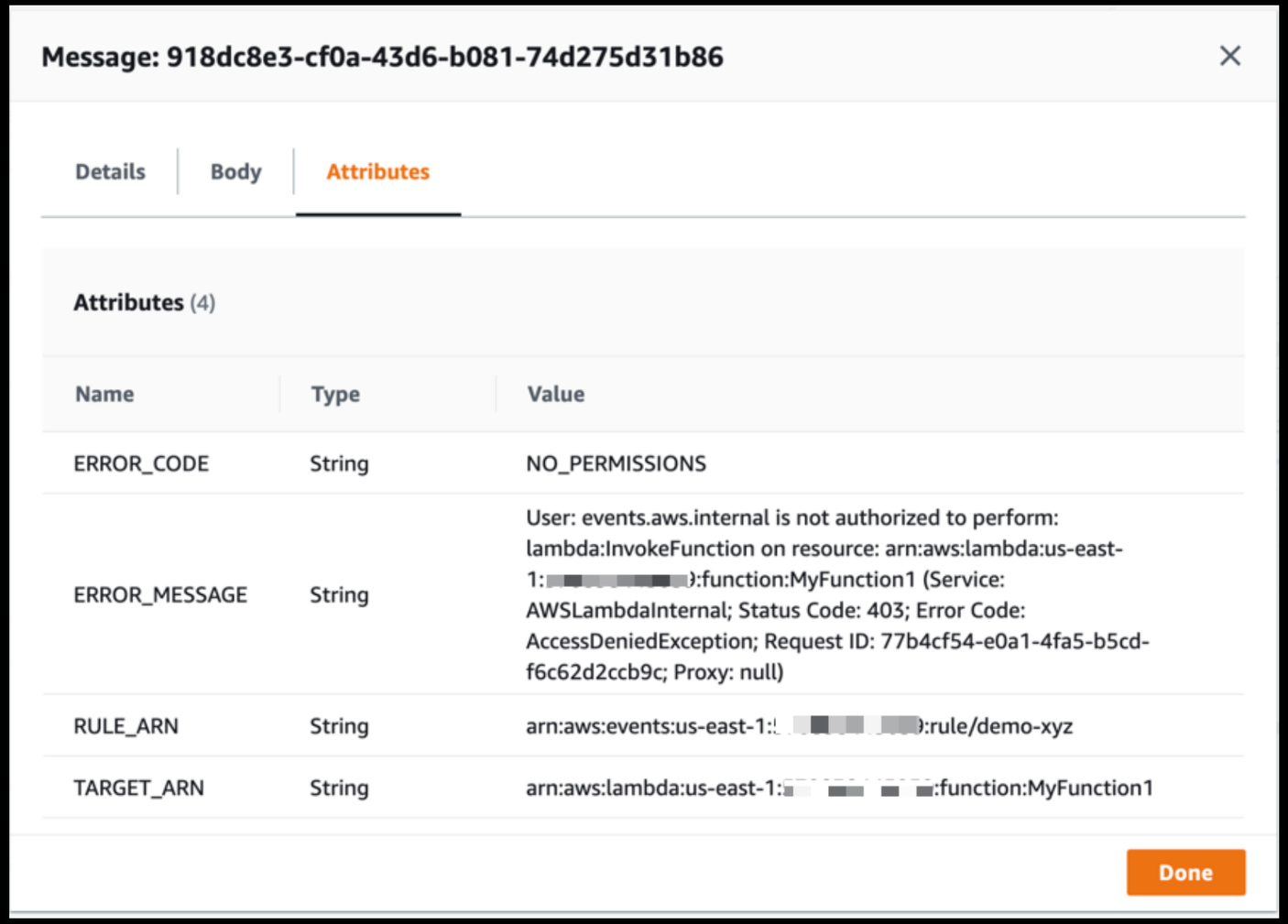
```
Resources:
  BlueServiceBusRule:
    Type: 'AWS::Events::Rule'
    Properties:
      Description: Rule to consume P-E1
      Name: E1-Rule
      EventPattern:
        source:
          - E1
      State: ENABLED
      Targets:
        - Arn: 'arn:aws:sqs:us-east-1:123456789012:function:non-existing'
          Id: Id1234
          RetryPolicy:
            MaximumRetryAttempts: 4
            MaximumEventAgeInSeconds: 400
          DeadLetterConfig:
            Arn: 'arn:aws:sqs:us-east-1:123456789012:blue-service-dlq'
```

Amazon EventBridge dead letter queues

DON'T LOSE EVENTS AND UNDERSTAND ROOT CAUSE

Inspecting and re-processing messages on the DLQ made simple with message attributes

- ERROR_CODE
- ERROR_MESSAGE
- RULE_ARN
- TARGET_ARN



Recommended practices using EventBridge

UNLESS YOU KNOW BETTER ONES

- Use dead letter queues
- Avoid using the default event bus for custom application events
- Create “subscriptions”—one target per rule

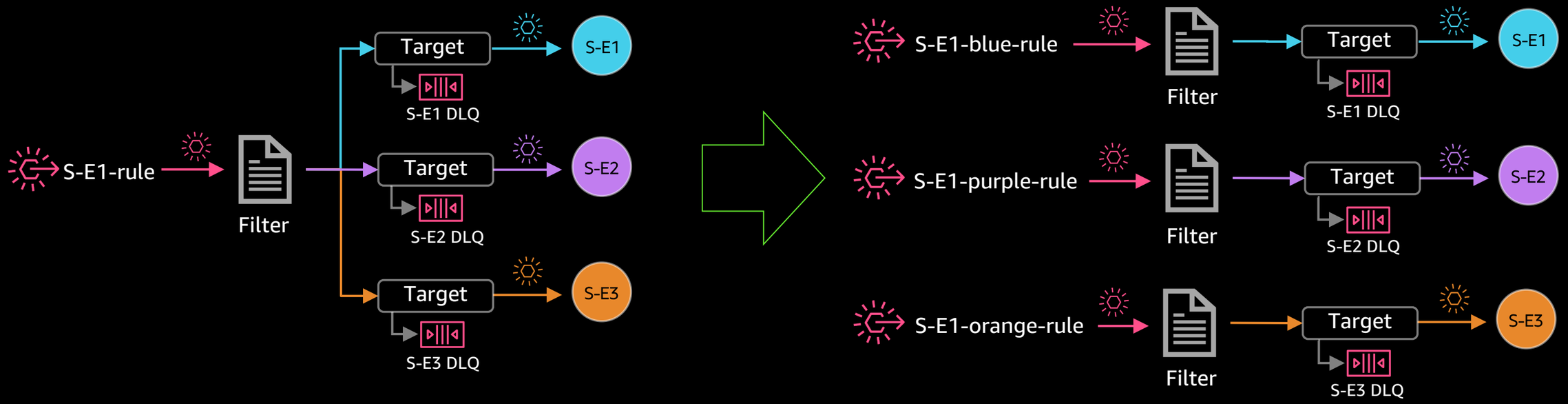
Recommended practices using EventBridge

UNLESS YOU KNOW BETTER ONES

- Use dead letter queues
- Avoid using the default event bus for custom application events
- Create “subscriptions”—one target per rule

Create subscriptions

TARGET IS NOT INDEPENDENT OF THE RULE



- Coupling is on the filter not the event
- Adding additional dependencies, i.e., targets to a rule introduces complexity

- Coupling is on the event not the filter
- Filters can change independently of target

Summary



© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Summary

Focus on logical architecture first, this is most important

There is no right or wrong, use the pattern or combination of patterns that makes sense to your problem domain

Think not only about the physical implementation but also the effect it has on team structure and development—beware Conway’s law

Thank you!



AWS re:Invent

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.



Please complete the session survey

AWS re:Invent

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.