ECS Fargate Spot Auto-Recovery – POC Design Document

1. Purpose of this POC

This Proof of Concept (POC) demonstrates an **automated, centralized recovery mechanism for ECS Fargate Spot workloads across multiple AWS accounts**.

The goal is to ensure **service continuity** when Fargate Spot capacity becomes unavailable, **without manual intervention** and **without writing per-account custom code**.

---

2. Problem Statement

ECS services running on **Fargate Spot** may experience: - Capacity unavailability - Spot interruptions - Implicit scaling behavior without explicit failure signals

AWS does **not always emit reliable failure events** for these scenarios. As a result: - ECS may silently rebalance tasks - Spot failures may not trigger expected alarms - Manual intervention is often required to switch to On-Demand

This POC solves the problem by: - Centralizing detection and response - Automatically failing over to On-Demand capacity - Restoring Spot usage when capacity becomes stable

---

3. High-Level Architecture

Key Design Principles

- **Centralized automation**
- **Zero per-account code changes**
- **Event-driven architecture**
- **Multi-account scalability**
- **Idempotent and safe operations**

Account Structure

Central Account

Hosts: - EventBridge (central bus) - SQS queues - Lambda automation - DynamoDB state table

Workload Account(s)

Host: - ECS clusters and services - Fargate + Fargate Spot capacity providers - Minimal onboarding configuration only

---

4. Workload Account Onboarding Template

Each new workload account must perform **only the following three steps**:

Step 1: Create IAM Role

Create a role named:

```
CentralSpotAutomationRole
```

Trust policy allows the **central account** to assume this role.

Step 2: Attach Permissions

Minimum permissions: - ecs:DescribeServices - ecs:UpdateService - iam:PassRole (if required)

Step 3: Forward ECS Events

Create an EventBridge rule that forwards ECS events to the **central event bus ARN**.

✅No Lambda deployment ✅No DynamoDB ✅No per-service configuration

---

5. Central Account Components

> **Important:** The entire Central Account setup for this POC is deployed using **AWS CDK**. No manual resource creation is required in the Central Account. The CDK stacks provision EventBridge rules, SQS queues, Lambda functions, DynamoDB tables, IAM roles, and permissions in a repeatable and auditable manner.

5.1 EventBridge

- Central event bus receives ECS events from all workload accounts
- Supports both real and synthetic test events

5.2 SQS Queues

Queue	Purpose
fargate-spot-placement-failure-queue	Placement failure events
fargate-spot-interruption-queue	Spot interruption events

Each queue has a Dead-Letter Queue (DLQ).

---

6. Lambda Functions

All Lambda functions in the Central Account are deployed and managed using **AWS CDK**. Each Lambda has explicitly scoped IAM permissions attached via CDK constructs, ensuring least-privilege access and consistent configuration across environments.

6.1 Placement Failure Handler (Python)

Triggered by: SQS

Responsibilities: - Parse ECS event - Extract account ID, region, cluster, service - Assume workload account role - Capture current Capacity Provider strategy - Store state in DynamoDB - Switch service to **On-Demand only**

Fallback Strategy Applied: - FARGATE_SPOT → weight 0 - FARGATE → weight 10

6.2 Spot Interruption Handler (Node.js)

Triggered by: SQS

Responsibilities: - Handle real Spot interruption warnings - Apply failover logic - Ensure service continuity

---

6.3 Restore Handler (Python)

Triggered by: Scheduled EventBridge rule

Responsibilities: - Scan DynamoDB for pending restores - Assume workload account role - Restore original capacity provider strategy - Mark restore as complete - Emit audit event

---

7. DynamoDB State Table

Table Name

```
ecs_cp_restore_state
```

Primary Key

```
id = {account_id}::{cluster_name}::{service_name}
```

Attributes

- config → Original capacity provider strategy
- event → Original triggering ECS event
- scheduled → true / false

This design guarantees: - Multi-account isolation - Idempotent updates - Safe retries

---

8. Event Handling & Testing Strategy

Why Synthetic Testing Is Required

In real ECS behavior, Spot capacity loss does **not always emit explicit placement failure or interruption events**. ECS may simply reschedule or rebalance tasks automatically.

Because of this, for the POC we rely on **synthetic ECS events** to validate the automation end-to-end.

Synthetic Event Injection (POC Testing)

Use the following command from **any AWS account that has permission to put events to the central event bus**:

```
aws events put-events
  --region ap-south-1
  --entries '[
    {
      "Source": "test.aws.ecs",
      "DetailType": "ECS Service Action",
      "Resources": ["arn:aws:ecs:ap-south-1:903558039761:service/test-cluster2/
test-cluster-service"],
      "Detail": "{\"eventName\":\"SERVICE_TASK_PLACEMENT_FAILURE\",\"reason\":
```

```
\"RESOURCE:FARGATE\",\"capacityProviderArns\":[\"arn:aws:ecs:ap-
south-1:903558039761:capacity-provider/FARGATE_SPOT\"]}"
    }
  ]'
```

This event simulates a Fargate Spot placement failure and triggers the complete pipeline:

EventBridge → SQS → Lambda → DynamoDB → ECS Update

Verification Commands

After sending the synthetic event, validate each stage:

1. **Check SQS queue depth**

```
aws sqs get-queue-attributes
  --region ap-south-1
  --queue-url <QUEUE_URL>
  --attribute-names ApproximateNumberOfMessages
ApproximateNumberOfMessagesNotVisible
```

2. **Check Lambda execution logs**

```
aws logs tail /aws/lambda/fargate-spot-task-placement-failure-handler
  --region ap-south-1
  --since 5m
  --follow
```

3. **Verify DynamoDB state entry**

```
aws dynamodb scan
  --region ap-south-1
  --table-name ecs_cp_restore_state
```

Successful execution confirms that the Central Account automation is functioning correctly without any dependency on real Spot failures.

Synthetic Testing (POC)

Because ECS does not always emit failure events, synthetic events are used:

- `aws events put-events`
- Test placement failures
- Validate EventBridge → SQS → Lambda → DynamoDB flow

Verification Steps

- Check SQS queue depth
- Check Lambda CloudWatch logs
- Scan DynamoDB for stored state

---

9. Restore Scheduling

Restore Lambda runs on a schedule:

Examples: - Every 1 minute (DEV) - Every 5 minutes (PROD)

This ensures Spot capacity is reused when stable.

---

10. Security & IAM Model

- Central account uses **STS AssumeRole**
- No long-lived credentials
- No inbound access to workload accounts
- Least-privilege policies enforced

---

11. Scalability & Extensibility

This design supports: - Unlimited workload accounts - Multiple regions - Future extensions (notifications, metrics, dashboards)

No changes required in central automation when adding new accounts.

---

12. POC Success Criteria

The POC is considered successful if: - Spot failure triggers automated failover - Services continue running on On-Demand - Original strategy is restored automatically - No manual intervention is required - New workload accounts onboard with only 3 steps

13. Conclusion

This POC demonstrates a **robust, scalable, and production-ready pattern** for managing ECS Fargate Spot reliability across multiple AWS accounts using centralized automation.

It eliminates operational toil, improves service availability, and provides a clean path toward enterprise-grade Spot adoption.