**Secret**

Privacy Center

개발표준가이드(배치서비스)

PRIVACY-CENTER-BATCH-001

Ver. 1.2

|  |  |
| --- | --- |
| 관리부서 : | |
|  |  |
|  |  |
| Copyright ⓒ LG CNS  LG CNS의 사전 승인 없이 본 내용의 전부 또는 일부에 대한 복사, 배포, 사용을 금합니다. | |

개 정 이 력

| 버전 | 작성일 | 변경내용[[1]](#footnote-1) | 작성자 | 승인자 |
| --- | --- | --- | --- | --- |
| V1.0 |  | 최초작성 | 신충일 |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

목 차

[1. 개요 1](#_Toc171411691)

[2. 개발표준 1](#_Toc171411692)

[2.1 디렉토리 구조 1](#_Toc171411693)

[2.2 패키지 구조 1](#_Toc171411694)

[2.3 명명 규칙 2](#_Toc171411695)

[3. 개발가이드 3](#_Toc171411696)

[3.1 요소 별 개발가이드 3](#_Toc171411697)

[3.1.1 Job 3](#_Toc171411698)

[3.1.2 Step 3](#_Toc171411699)

[3.1.3 Reader 4](#_Toc171411700)

[3.1.4 Processor 4](#_Toc171411701)

[3.1.5 Writer 5](#_Toc171411702)

[3.1.6 SQL 5](#_Toc171411703)

[4. 별첨 6](#_Toc171411704)

[4.1 Devon Management 6](#_Toc171411705)

[4.1.1 Job 등록 방법 6](#_Toc171411706)

[4.1.2 Job 배치파일 등록 방법 7](#_Toc171411707)

[4.2 Local 환경 Job 실행 7](#_Toc171411708)

[4.3 AWS Batch 9](#_Toc171411709)

[4.3.1 개요 9](#_Toc171411710)

[4.3.2 Job Definition 9](#_Toc171411711)

[4.3.1 Job 12](#_Toc171411712)

[4.3.1 Job Queue 14](#_Toc171411713)

[4.3.2 Compute Engine 14](#_Toc171411714)

[4.4 AWS Lambda 15](#_Toc171411715)

[4.4.1 Function 생성 15](#_Toc171411716)

[4.5 AWS Step Functions 17](#_Toc171411717)

[4.5.1 State Machine 생성 17](#_Toc171411718)

[4.5.2 State: Action: Lambda: Invoke 18](#_Toc171411719)

[4.5.3 State: Action: Batch: SubmitJob 19](#_Toc171411720)

[4.5.1 State: Flow: Choice 22](#_Toc171411721)

[4.6 AWS EventBridge 22](#_Toc171411722)

[4.6.1 Schedule 생성 23](#_Toc171411723)

[4.6.2 Schedule 생성 – 주기 설정 23](#_Toc171411724)

[4.6.3 Schedule 생성 – 대상 설정 24](#_Toc171411725)

[4.6.4 Cron 표현식 25](#_Toc171411726)

# 개요

본 문서는 서비스 빌링 플랫폼 프로젝트에 투입되는 개발자들에게 필요한 개발표준을 제공하기 위한 문서이다. 각 단계는 절차 중심으로 작성되어 개발자들이 쉽게 이해할 수 있도록 하였다.

# 개발표준

## 디렉토리 구조

DevOn Boot 사용 시 기본적으로 구성되는 모듈 구조이다.

[DevOn Boot]

| **폴더 구조** | | | | **용도** |
| --- | --- | --- | --- | --- |
| {프로젝트 모듈 경로} | /src/main/java | | | java 소스 |
| /src/main/resources | /sql | | SQL 파일(XML) |
| /META-INF | /spring | AutoConfiguration 파일 |
| application.yml | | 어플리케이션 설정 파일 |
| log4j2-spring.xml | | 로그 설정 파일 |
| build.gradle | | | gradle 파일 |

## 패키지 구조

패키지는 /src/main/java 하위에 생성하며, 영문 소문자를 사용하여 아래의 규칙에 따라 정의한다.

- level0: 서비스빌링플랫폼 프로젝트를 지칭하는 com.lgcns.svcp를 공통으로 사용한다.

- level1: 각 프로젝트를 식별하는 시스템코드를 사용한다. (4자리)

- level2: 채널 구분 용도로 batch 패키지를 고정으로 사용한다.

- level3: 업무 영역 Level로 설계 단계에서 정의한 도메인명을 기반으로 작성한다.

|  |
| --- |
| 예시)  svcp-bill-bat  ├─ src/main/java/com/lgcns/svcp \* default 패키지 (Level 0)  │ └─ bill \* 프로젝트 식별 영문약어명 (Level 1)  │ └─ batch \* 배치 개발 대상 (Level 2)  │ └─ xxxmgmt \* 업무컴포넌트 (Level 3)  │ ├─ job  │ ├─ listener  │ ├─ reader  │ ├─ processor  │ ├─ writer  │ ├─ tasklet  │ └─ dto |

## 명명 규칙

모든 Class와 Interface명은 표준 단어를 조합하여 사용하며, Pascal 표기법을 따른다.

| **구분** | **컴포넌트 구분** | **클래스명** | **컴포넌트 Postfix** |
| --- | --- | --- | --- |
| java | job | 컴포넌트명 + Config | XxxConfig |
| listener | 컴포넌트명 + Listener | XxxListener |
| processor | 컴포넌트명 + Processor | XxxProcessor |
| tasklet | 컴포넌트 내 의미있는 명 + Tasklet | XxxTasklet |
| reader | 컴포넌트 내 의미있는 명 + Reader | XxxReader |
| writer | 컴포넌트 내 의미있는 명 + Writer | XxxWriter |
| dto | 데이터객체 명사조합 + DTO | XxxDTO |
| mybatis | SQL 파일 | sql- + [테이블 영문명(Pascal표기)].xml | sql-xxxx.xml |

# 개발가이드

## 요소 별 개발가이드

### Job

|  |
| --- |
| @Configuration  @RequiredArgsConstructor  **public** **class** DbToDbConfig {  ...  @Bean(name = "DB\_TO\_DB")  **public** Job dbToDbJob(Step dbToDbStep) {  **return** jobBuilderFactory.get("DB\_TO\_DB")  .start(dbToDbStep)  .build();  }  ...  } |

- 클래스 상단에 @Configuration을 선언하여 Java Based Configuration으로 Job을 구성한다.

- Job은 @Bean어노테이션을 설정하고 name을 지정하여 bean과 job id가 동일하게 등록되도록 한다

### Step

|  |
| --- |
| @Configuration  @RequiredArgsConstructor  **public** **class** DbToDbConfig {  ...  **private** **final** DbToDbProcessor processor;  ...  @Bean  @JobScope  **public** Step dbToDbStep(PlatformTransactionManager transactionManager,  @Qualifier("employeeReader") ItemReader employeeReader,  @Qualifier("employeeOutWriter") ItemWriter employeeOutWriter  ) {  **return** stepBuilderFactory.get("dbToDbStep")  .chunk(10, transactionManager) // chunkSize  .reader(employeeReader)  .processor(processor)  .writer(employeeOutWriter).build();  }  ...  } |

실제 업무의 배치 로직이 들어가는 단위로서 1개의 Job에 1개 이상의 Step이 할당될 수 있다. XxxxJobConfig.java 내에 구성한다.

### Reader

파일, DB 등의 매체로부터 데이터를 읽어 들인다.

#### MyBatisCursorItemReader

|  |
| --- |
| @Configuration  @RequiredArgsConstructor  **public** **class** DbToDbConfig {  ...  @Bean  **public** MyBatisCursorItemReader<Employee> employeeReader() {  **return** **new** ItemCursorReaderBuilder<Employee>().queryId("Employee.retrieveEmployee")  .parameterValues(**new** HashMap(){{put("num", "30008");}})  .sqlSessionFactory(sqlSessionFactory).build();  }  ...  } |

- CursorReader는 read 메소드가 호출 될 때 마다 커서를 다음 행으로 이동시키고 처리 과정 중에 사용되는 맵핑된 객체를 반환하는 Reader 이다.

#### MyBatisPagingItemReader vs MyBatisCursorItemReader

일반적으로 적은 양의 데이터를 처리할 때는 MyBatisPagingItemReader를, 대량의 데이터를 처리할 때는 MyBatisCursorItemReader를 사용하는 것이 좋다.

|  |  |  |
| --- | --- | --- |
|  | MyBatisPagingItemReader | MyBatisCursorItemReader |
| 방식 | 페이징 기법을 사용하여 일정한 크기의 블록으로 나누어 읽어옵니다. 이를 위해 SQL 쿼리에 페이징 조건(LIMIT, OFFSET)을 사용해야 합니다. | 커서 기반 방식으로 데이터베이스 커서를 사용하여 한 번에 하나의 레코드를 읽어옵니다. |
| 병렬 처리 | 가능 | 불가능 (파티셔닝 기능을 활용해서 병렬 처리 가능) |

### Processor

Step 내에서 Reader에서 넘겨받은 Item 업무 로직으로 처리한 뒤에 Writer에 전달한다. 선택적으로 작성할 수 있으며 없는 경우 Reader에서 Writer로 전달된다.

|  |
| --- |
| @Configuration  **public** **class** DbToDbProcessor<I, O> **extends** AbstractProcessor<I, O> {  **public** O process(I item) {  **if**(employee.getNum() != **null**) {  **return** (O) EmployeeOut.*builder*()  .num(employee.getNum())  .name(employee.getName())  .birthdate(employee.getBirthdate())  .sex(employee.getGender())  .telephone(employee.getTelephone())  .address(employee.getAddress())  .postal(employee.getPostal())  .joblevelCode(employee.getJoblevelCode())  .divisionCode(employee.getDivisionCode())  .departmentCode(employee.getDepartmentCode())  .skillCode(employee.getSkillCode()).build();  }  **return** **null**;  }  } |

### Writer

Step 내에서 DB 혹은 File로 Item을 저장한다.

#### MyBatisBatchItemWriter

|  |
| --- |
| @Configuration  @RequiredArgsConstructor  **public** **class** DbToDbConfig {  ...  @Bean  **public** MyBatisBatchItemWriter<EmployeeOut> employeeOutWriter() {  **return** **new** MyBatisBatchItemWriterBuilder<EmployeeOut>()  .sqlSessionFactory(sqlSessionFactory)  .statementId("EmployeeOut.insert")  .build();  }  ...  } |

### SQL

SQL 파일은 개발표준을 준수하도록 하며 다음과 같이 작성한다.

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN" "http://mybatis.org/dtd/mybatis-3-mapper.dtd">  <mapper namespace="EmployeeOut">  ...  <insert id="insert">  <CDATA[  INSERT  INTO SAM\_EMPLOYEE\_OUT (NUM, NAME, BIRTHDATE, SEX, TELEPHONE, ADDRESS, POSTAL, JOBLEVEL\_CODE, DIVISION\_CODE, DEPARTMENT\_CODE, SKILL\_CODE)  VALUES (#{num}, #{name}, #{birthdate}, #{sex}, #{telephone}, #{address}, #{postal}, #{joblevelCode}, #{divisionCode}, #{departmentCode}, #{skillCode})  ]]>  </insert>  ...  </mapper> |

# 별첨

## Devon Management

### Job 등록 방법

devonboot 프레임워크 프로젝트를 관리하기 위한 별도 웹사이트가 존재하며, 해당 웹사이트에서 프로젝트 Batch Job, API 를 관리한다. 해당 웹사이트에 등록되지 않은 Job 를 호출하면 “Job Terminated in error: There is no matching job” Error Message가 출력된다.

접속 정보는 다음과 같다. (08:00 ~ 20:00 접속 가능)

|  |  |
| --- | --- |
| url | k8s-dtestdev-devonboo-c3c3e45a22-814130141.ap-northeast-2.elb.amazonaws.com/management |
| id | devon |
| password | q1w2e3R$ |

상단 메뉴 목록에서 [배치관리 - 배치작업관리] 페이지로 이동한다. 배치작업관리 페이지는 어플리케이션 별 등록된 Job 목록을 조회 및 관리할 수 있다.

![텍스트, 스크린샷, 소프트웨어, 번호이(가) 표시된 사진

자동 생성된 설명

\* job id : 어플리케이션에 개발한 Job의 bean name

\* 어플리케이션명 : 개발중인 시스템의 어플리케이션명

### Job 배치파일 등록 방법

파일 입출력이 필요한 경우 [배치관리 – 배치정보조회 – 배치파일정보조회]에서 등록이 가능하다.

텍스트, 스크린샷, 번호, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

\* 파일별명 : 입출력에 필요한 파일명

\* Job Id : Batch Job bean name

\* 파일경로폴더 : 입력의 경우 “./file/in/” 출력의 경우 “./file/out/”

\* 파일명 : 입출력에 필요한 파일명

\* IO SPEC

## Local 환경 Job 실행

텍스트, 스크린샷, 폰트, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

Boot Dashboard에서 내 프로젝트를 선택 후 open config

텍스트, 소프트웨어, 컴퓨터 아이콘, 웹 페이지이(가) 표시된 사진

자동 생성된 설명

ANSI console output 체크해제

\* 체크하지 않으면 jobExecution is null 에러 발생

텍스트, 스크린샷, 소프트웨어, 컴퓨터 아이콘이(가) 표시된 사진

자동 생성된 설명

Program arguments : [jobid] [param key]=[param value] (Ex. DB\_TO\_DB O\_DATE=20221007)

VM arguments : --add-opens=java.base/java.lang=ALL-UNNAMED --add-opens=java.base/java.time=ALL-UNNAMED

텍스트, 스크린샷, 소프트웨어, 컴퓨터 아이콘이(가) 표시된 사진

자동 생성된 설명

## AWS Batch

AWS Batch는 규모에 상관없이 배치 스타일 워크로드를 실행할 수 있는 컨테이너 기반의 완전 관리형 서비스입니다. 다음 흐름도는 AWS Batch가 각 작업을 실행하는 방법을 설명합니다.

1. IAM 역할, 메모리 및 CPU 요구 사항, 기타 구성 옵션을 제공하여 작업 실행 방법을 지정하는 작업 정의를 생성합니다.
2. 작업 처리를 위해 컴퓨팅 환경에 스케줄링될 때까지 작업이 상주하는 관리형 AWS Batch 작업 대기열에 작업을 제출합니다.
3. AWS Batch는 대기열에 있는 각 작업의 CPU, 메모리 및 GPU 요구 사항을 평가하고 컴퓨팅 환경에서 컴퓨팅 리소스를 확장하여 작업을 처리합니다.
4. AWS Batch 스케줄러는 적절한 AWS Batch 컴퓨팅 환경에 작업을 배치하여 처리합니다.
5. 작업이 종료되고 결과가 사용자 정의 스토리지에 기록됩니다.

### 개요

텍스트, 스크린샷, 폰트, 직사각형이(가) 표시된 사진

자동 생성된 설명

### Job Definition

Job Definition은 작업이 어떻게 실행될지 Job ID와 Job Parameter를 설정하고, CPU와 Memory를 얼마나 할당할 지를 지정한다. 작업 정의의 많은 항목은 Job을 제출할 때 수정이 가능하다.

#### Compute Engine Type 설정

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

Orchestration type : Fargate (생성된 Compute engnine과 동일한 type)

* Compute Engine의 Orchestration type 확인 방법

EC2 : 컴퓨팅 용량이 확장가능한 일종의 VM 서버

Fargate : EC2 대비 경량이고 기동 속도가 빠른 서버

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

#### Job Definition 정보 설정

텍스트, 폰트, 번호, 라인이(가) 표시된 사진

자동 생성된 설명

Name : Job Definition 이름 (batch-jd-[서비스 코드]-[job 명])

Execution Role : ecsInstanceRole

#### Container 설정

텍스트, 스크린샷, 폰트, 웹 페이지이(가) 표시된 사진

자동 생성된 설명

Image : ECR에 등록된 실행하고자 하는 배치의 Image 주소

Command : ["java","--add-opens=java.base/java.lang=ALL-UNNAMED","-jar","build/libs/app.jar","[Job Id]","O\_DATE=20240514"]

* **Image 주소 확인 방법**

[ECR의 해당 배치서비스 Repository] 접속하여 Image 목록을 조회할 수 있다.

스크린샷, 텍스트, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

* **Command**

“java” ~ “build/libs/app.jar” : java 소스 실행 명령어

“Job Id” : 실행하려는 Job Id

“O\_DATE=yyyymmdd” : Devon Boot Framework에서 지정한 필수 파라미터로 실행한 당일 날짜를 입력

이후에는 Job별로 필요한 파라미터 추가 입력이 가능

#### Logging (CloudWatch) 설정

텍스트, 스크린샷, 번호, 라인이(가) 표시된 사진

자동 생성된 설명

Log driver : awslogs

Options

awslogs-group : /aws/batch/svcp-dev

awslogs-stream-prefix : [서비스 코드]

보관주기 : 2개월

### Job

[Jobs – Submit New Job]에서 Job Definition을 template으로 작업을 제출하여 실행할 수 있다.

#### Job 구성

텍스트, 스크린샷, 번호, 폰트이(가) 표시된 사진

자동 생성된 설명

Name : batch-j-[시스템코드]-[Job ID]

Job definition : 생성한 job definition

Job queue : batch-jq-[시스템코드] (SWA 생성)

#### Job Definition 설정 수정

텍스트, 소프트웨어, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

Job Definition에서 사전에 설정한 Command를 불러올 수 있으며 수정 또한 가능하다.

#### Job 실행 결과

Dashboard의 JobQueue에서 제출된 Job의 진행상황을 알 수 있다.

텍스트, 스크린샷, 소프트웨어, 번호이(가) 표시된 사진

자동 생성된 설명

#### Logging (Cloudwatch) 확인

Job Definition에서 지정한 설정에 맞추어 로그가 Cloudwatch에 저장되고 Running 단계 이후에 로그 스트림 이름에 로그를 확인할 수 있는 url이 출력된다.

텍스트, 폰트, 번호, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

### Job Queue

사용자가 제출한 Job은 특정 Job Queue에 등록되어 Compute Engine에 예약될 때까지 상주한다.

### Compute Engine

작업을 실행하는데 사용되는 Fargate, EC2 환경을 의미한다. 기본적으로 Fargate를 사용하며 추가 구성이 필요한 경우 요청을 통해 EC2를 구성이 가능하다. (CPU, Memory 증설도 가능)

## AWS Lambda

AWS Lambda은(는) 서버를 프로비저닝하거나 관리하지 않고도 코드를 실행할 수 있게 해주는 컴퓨팅 서비스입니다. 고가용성 컴퓨팅 인프라에서 코드를 실행하고 서버와 운영 체제 유지 관리, 용량 프로비저닝 및 자동 조정, 코드 및 보안 패치 배포, 로깅 등 모든 컴퓨팅 리소스 관리를 수행합니다. Lambda를 사용하면 Lambda가 지원하는 언어 런타임 중 하나로 코드를 제공하기만 하면 됩니다.

### Function 생성

텍스트, 스크린샷, 소프트웨어, 웹 페이지이(가) 표시된 사진

자동 생성된 설명

workflow 구성을 위한 Step Functions 서비스에서 파라미터 데이터 생성을 위해 사용한다.

\* function name : ld-func-batch-[시스템코드]-[함수이름]

\* iam 정책으로 ld-func-batch-[시스템코드] 접두사 필수

\* Runtime : Node.js (aws console에서 직접 수정 가능한 언어, 다른 언어 지정 가능)

텍스트, 스크린샷, 소프트웨어, 번호이(가) 표시된 사진

자동 생성된 설명

자동으로 생성된 handler 함수로 배치 Parameter로 사용하기 위한 데이터 반환

(이후 4.4 StepFunctions – 4.4.2. State: Action: Lambda: Invoke Output에서 확인 가능)

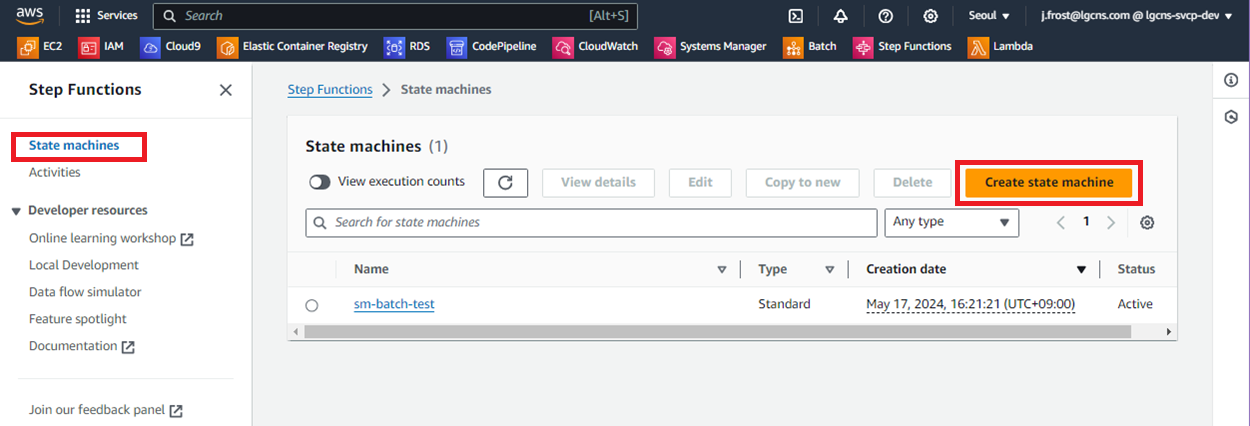
## AWS Step Functions

AWS Step Functions는 개발자가 AWS 서비스를 사용하여 분산 애플리케이션을 구축하고, 프로세스를 자동화하고, 마이크로서비스를 조정하고, 데이터 및 기계 학습(ML) 파이프라인을 생성할 수 있도록 지원하는 시각적 워크플로우 서비스입니다.

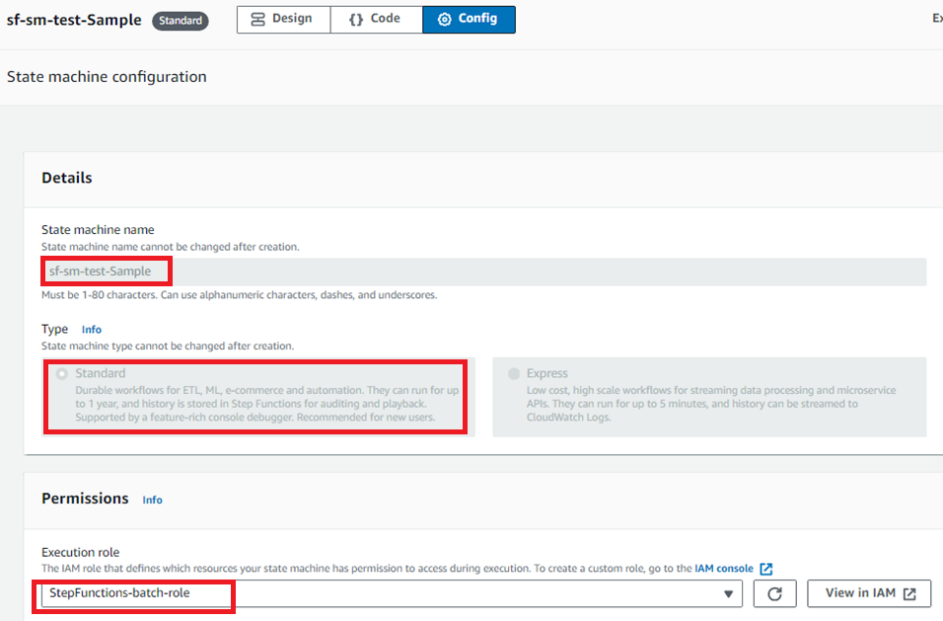
### State Machine 생성

State Machine은 State(상태)로 이루어져 있고State는 Action, Flow 두 종류로 구성되어 있다. Action은 Batch Job 실행, Lambda 함수 실행 등 서비스를 수행하는 상태이며

Flow는 Action간 흐름을 제어하는 상태이다.



[Step Functions – State machines – Create state machine]에서 blank template 선택 후 상태머신 생성



State machine name : 상태머신 이름

Type : Standard

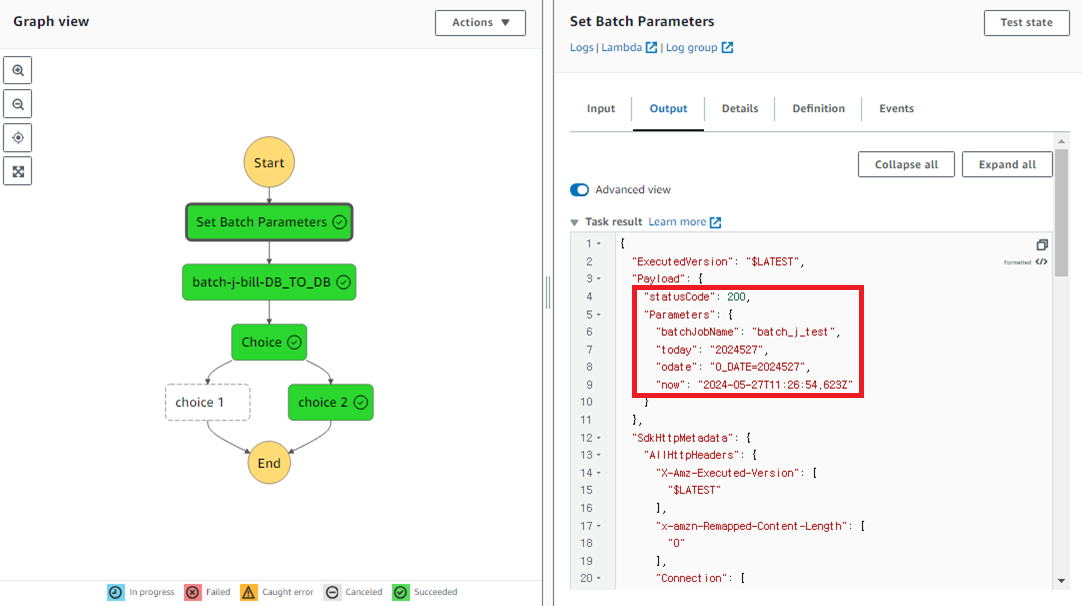
Execution role : StepFunctions-batch-role

### State: Action: Lambda: Invoke

텍스트, 스크린샷, 도표, 평행이(가) 표시된 사진

자동 생성된 설명

State name, API Parameters – Function name 설정



Output data 중 Payload 부분만 다음 state로 전달된다.

### State: Action: Batch: SubmitJob

텍스트, 스크린샷, 도표, 평행이(가) 표시된 사진

자동 생성된 설명

State name, Batch job name, Batch job definition, Batch job queue 지정 (AWS Batch 서비스의 목록 조회해서 지정 가능)

텍스트, 전자제품, 스크린샷, 디스플레이이(가) 표시된 사진

자동 생성된 설명

Edit as JSON 클릭 후 “Parameters.$”: “$.Parameters” 입력

텍스트, 스크린샷, 소프트웨어, 컴퓨터 아이콘이(가) 표시된 사진

자동 생성된 설명

Batch: SubmitJob Input Data

\* Input : Lambda 함수에서 전달된 Payload를 의미

\* Parameters : Input Data의 데이터를 $(Input)의 Parameters 부분을 Batch Parameter로 전달

(Configuration의 API Parameters에 “Parameters.$”: “$.Parameters” 세팅)

텍스트, 스크린샷, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

Batch: SubmitJob Output Data

### State: Flow: Choice

텍스트, 스크린샷, 도표, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

Workflow의 분리처리 가능

\* Input으로 전달받은 데이터를 활용하여 조건문 설정

## AWS EventBridge

Amazon EventBridge는 이벤트를 통해 애플리케이션 구성 요소를 서로 연결하여 개발자가 확장 가능한 이벤트 기반 애플리케이션을 쉽게 구축할 수 있도록 하는 서버리스 서비스입니다.

### Schedule 생성

텍스트, 스크린샷, 소프트웨어, 컴퓨터 아이콘이(가) 표시된 사진

자동 생성된 설명

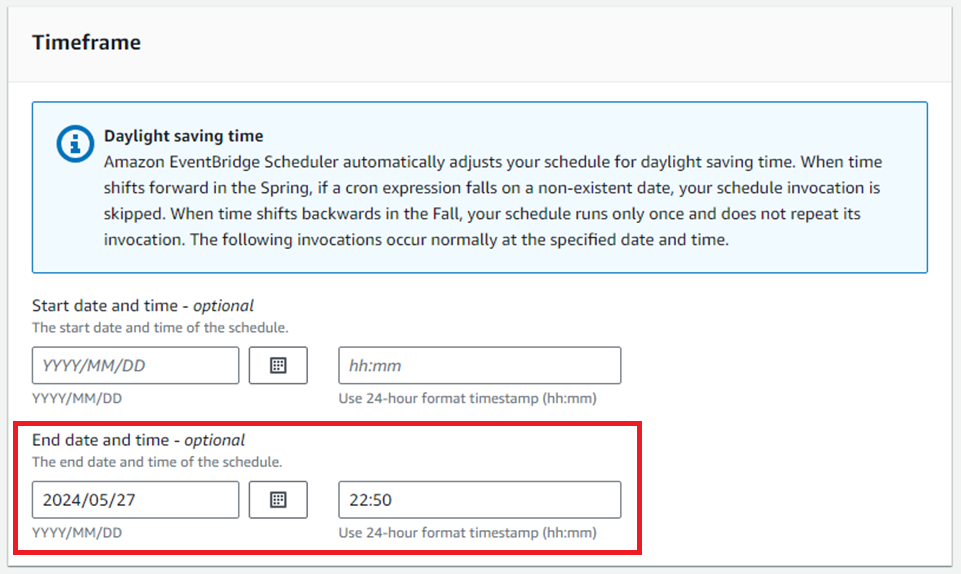
[AWS EventBridge – Scheduler – Schedules – Create Schedule]

### Schedule 생성 – 주기 설정

텍스트, 스크린샷, 번호, 폰트이(가) 표시된 사진

자동 생성된 설명

Cron 표현식에 의한 스케줄 주기 설정 (4.5.4 Cron 표현식 참고)



종료 시간 설정 필수

### Schedule 생성 – 대상 설정

텍스트, 스크린샷, 소프트웨어, 번호이(가) 표시된 사진

자동 생성된 설명

EventBridge로 실행시킬 대상 설정 (Step Functions StartExecution; 특정 StateMachine 실행)

텍스트, 스크린샷, 번호, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

실행할 Step Functions State Machine 설정

텍스트, 스크린샷, 번호, 폰트이(가) 표시된 사진

자동 생성된 설명

Execution role : Amazon\_EventBridge\_Scheduler\_SFN

### Cron 표현식

|  |  |  |
| --- | --- | --- |
| Field | Values | Wildcards |
| Minutes | 0-59 | , - \* / |
| Hours | 0-23 | , - \* / |
| Day-of-month | 1-31 | , - \* ? / L W |
| Month | 1-12 or JAN-DEC | , - \* / |
| Day-of-week | 1-7 or SUN-SAT | , - ? # |
| Year | 1970-2199 | , - \* / |

|  |  |  |
| --- | --- | --- |
| Wildcards | Description | |
| , (comma) | 나열 | The , (comma) wildcard includes additional values. In the Month field, JAN,FEB,MAR includes January, February, and March. |
| - (dash) | 범위 | The - (dash) wildcard specifies ranges. In the Day field, 1-15 includes days 1 through 15 of the specified month. |
| \* (asterisk) | 모든 값 | The \* (asterisk) wildcard includes all values in the field. In the Hours field, \* includes every hour. You can't use \* in both the Day-of-month and Day-of-week fields. If you use it in one, you must use ? in the other. |
| / (slash) | N번째 | The / (slash) wildcard specifies increments. In the Minutes field, you could enter 1/10 to specify every tenth minute, starting from the first minute of the hour (for example, the 11th, 21st, and 31st minute, and so on). |
| ? (question mark) | 모든 값 (요일) | The ? (question mark) wildcard specifies any. In the Day-of-month field you could enter 7 and if any day of the week was acceptable, you could enter ? in the Day-of-week field. |
| L | 마지막 날짜 | The L wildcard in the Day-of-month or Day-of-week fields specifies the last day of the month or week. |

1. 변경 내용: 변경이 발생되는 위치와 변경 내용을 자세히 기록(장/절과 변경 내용을 기술한다.) [↑](#footnote-ref-1)