

# Pipeline for user review translation and sentiment analysis

---

## Software Requirements Specifications

[SRS can be found here](#)

## Solution Presented

The solution presented fulfils the requirement of deploying a Python-based program on AWS, provided that the architecture is cost-efficient, time-constrained and follows best practices.

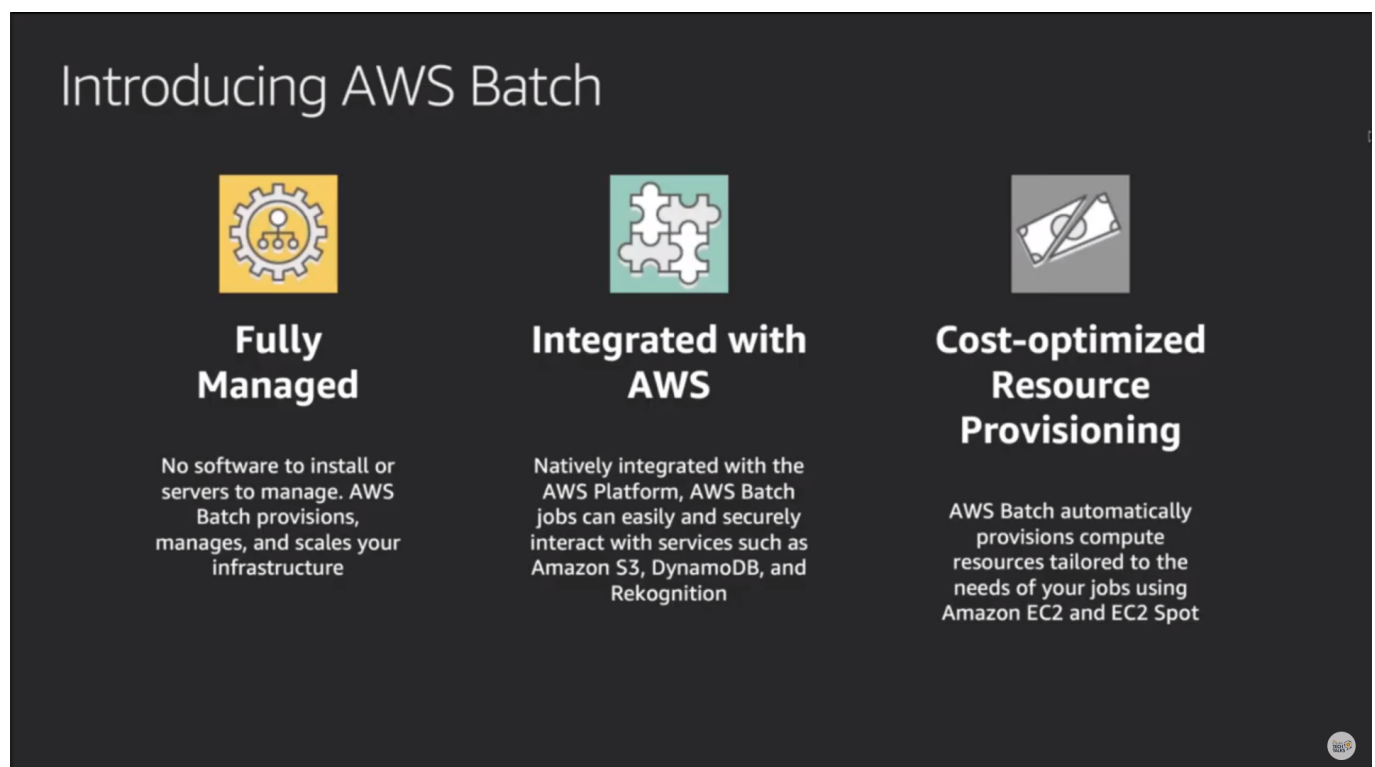
### AWS

#### AWS Batch + Spot Instances

Note: All images below are screenshots from these


- [Reference](#).
- [Reference](#)

#### What is AWS Batch?




The slide titled "Introducing AWS Batch" features a dark background with three columns of information. Each column has an icon at the top: a gear for "Fully Managed", puzzle pieces for "Integrated with AWS", and a dollar bill for "Cost-optimized Resource Provisioning". Below each icon is a bold heading and a descriptive paragraph. The "Fully Managed" section states that no software or servers need to be managed as AWS Batch handles the infrastructure. The "Integrated with AWS" section notes that AWS Batch is natively integrated with the AWS Platform and can interact with services like Amazon S3, DynamoDB, and Rekognition. The "Cost-optimized Resource Provisioning" section explains that AWS Batch automatically provisions compute resources tailored to the needs of the jobs using Amazon EC2 and EC2 Spot. A small AWS logo is in the bottom right corner.

## Introducing AWS Batch




### Fully Managed

No software to install or servers to manage. AWS Batch provisions, manages, and scales your infrastructure



### Integrated with AWS

Natively integrated with the AWS Platform, AWS Batch jobs can easily and securely interact with services such as Amazon S3, DynamoDB, and Rekognition

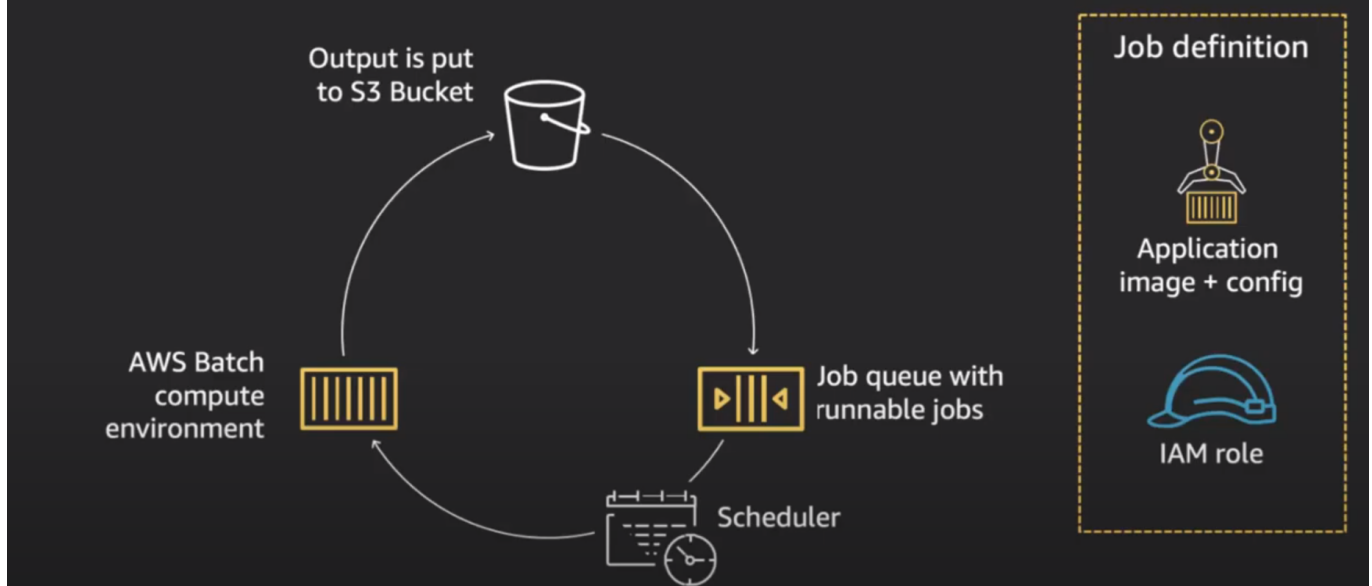


### Cost-optimized Resource Provisioning

AWS Batch automatically provisions compute resources tailored to the needs of your jobs using Amazon EC2 and EC2 Spot

## 2. AWS Batch Job Architecture

## Typical AWS Batch job architecture



### 3. Why Spot Instances?

- uses spare EC2 capacity that is available for less than the On-Demand price
- batch allocation strategy includes spot capacity optimized, which allows AWS to find available capacity pools, and launch instances accordingly
- suitable for a hybrid compute environment such that if on-demand resource's threshold is met, scaling would shift towards using spot instances, which in turn will scale as per requirements without much worry on the cost.

## Amazon EC2 purchase options

### On-Demand

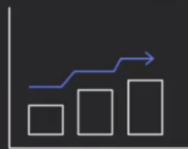
Pay-for-compute capacity **by the second** with no long-term commitments



Spiky workloads, to define needs

### Savings Plans & Reserved Instances

Make a commitment and receive a **significant discount** off compute



Committed & steady-state usage

### Spot Instances

Spare EC2 capacity at **savings of up to 90%** off On-Demand prices



Fault-tolerant, flexible, stateless workloads

### • spot interruptions:

- though spot interruptions may seem riskier at first, statistics show less than 5% of spot instance were interrupted in a span of 3 months

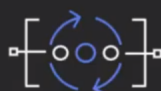
# Spot interruptions

## Minimal interruptions

Less than 5% of all running Spot Instances were interrupted in the last 3 months



The work you are doing to make your applications fault-tolerant also benefits Spot



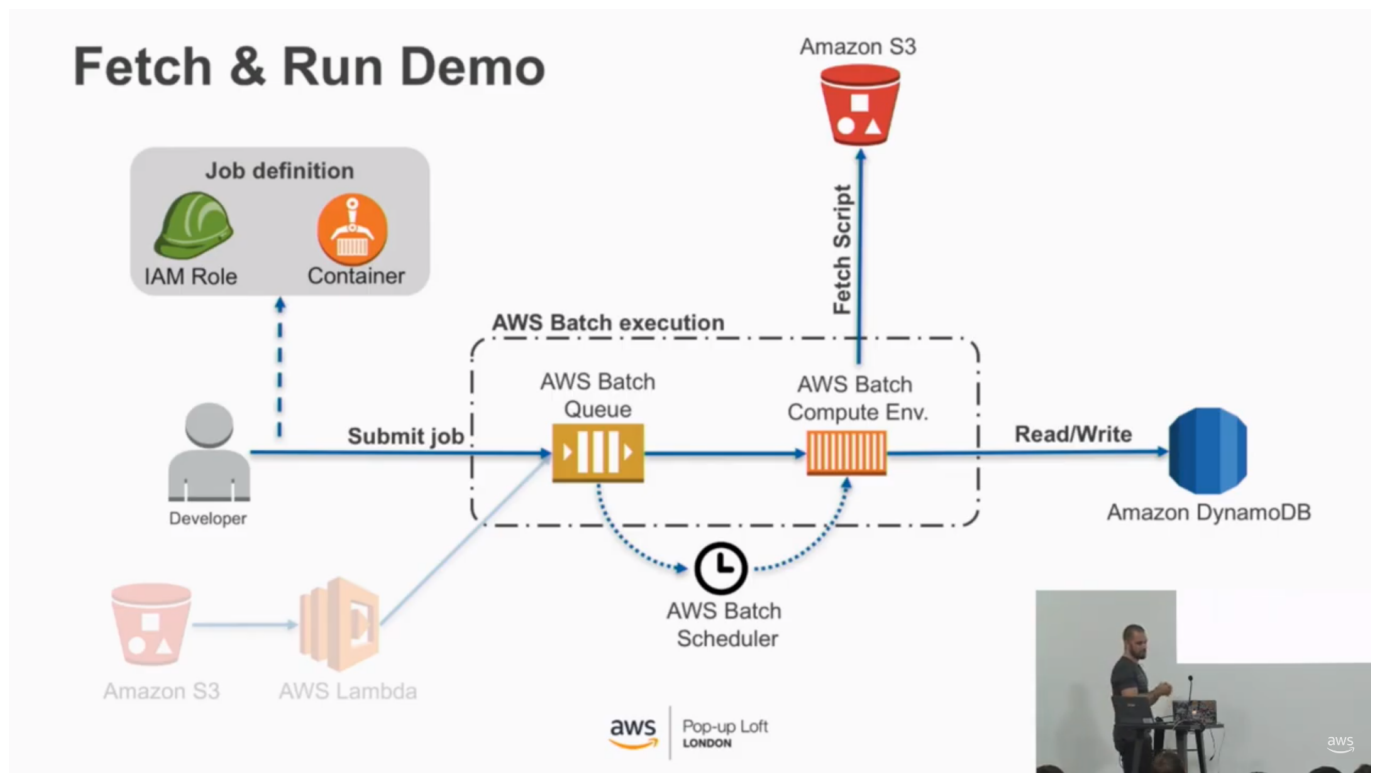
Spot is optimized for stateless, fault-tolerant, or flexible workloads

- ECS agent in spot instances trigger a 2 minute notice prior to interruptions if the resources are required, however this is handled by:
  - monitoring spot instances for notices, which should drain instance containers and stop scheduling any jobs to that instance
  - allowing flexibility in terms of subnets and instance types leads to an increased combination of spot instance across resources

## Best Practice Methodology

- Key steps:
  - ☐ Build a Docker image with the fetch & run script
  - ☐ Create an Amazon ECR repository for the image
  - ☐ Push the built image to ECR
  - ☐ Create a simple job script and upload it to S3
  - ☐ Create an IAM role to be used by jobs to access S3
  - ☐ Create a job definition that uses the built image
  - ☐ Submit and run a job that execute the job script from S3

The figure below shows the idea architecture for the solution, which is mainly to have a script that fetches jobs to run from S3, batch processess the data, and have the ready data written to a DB.



- create EC2 template and enable spot interruption handling

▼ Instance type
Info
Advanced

Instance type

t2.micro

Free tier eligible

▼

Family: t2    1 vCPU    1 GiB Memory  
On-Demand Linux pricing: 0.0116 USD per Hour  
On-Demand Windows pricing: 0.0162 USD per Hour

Compare instance types

▼ Key pair (login)
Info

You can use a key pair to securely connect to your instance. Ensure that you have access to the selected key pair before you launch the instance.

Key pair name

Don't include in launch template

▼

↺

Create new key pair

▼ Network settings

Networking platform
Info

☒ Virtual Private Cloud (VPC)  
Launch into a virtual network in your own logically isolated area within the AWS Cloud

☐ EC2-Classic  
Launch into a single flat network that you share with other customers.

Security groups

Select security groups

▼

default sg-01fb075ef5c43530d ✕

VPC: vpc-0e2bccb8492c2e25e

↺

- job retries enabled

- **Handling Spot Interruption:**

Advanced Settings > User Data > `echo "ECS_ENABLE_SPOT_INSTANCE _DRAINING=true" >> /etc/ecs/ecs.config`

- AWS Batch > create
  - compute environment

5 / 7

AWS Batch &gt; Compute environments &gt; Create compute environment

## Create compute environment

### Compute environment configuration

#### Compute environment type



##### Managed

AWS scales and configures your instances for you.



##### Unmanaged

You control and manage the instance configuration, provisioning, and scaling.

#### Compute environment name

Up to 128 letters (uppercase and lowercase), numbers, hyphens, and underscores are allowed.



Enable compute environment



Additional settings: service role, instance role, EC2 key pair



##### On-demand

EC2 instances that are billed per second.



##### Spot

Save money by using Spot instances but be aware your instances can be interrupted with a two minute notification when EC2 needs the capacity back.

#### Maximum % on-demand price

40

The maximum percentage of On-Demand pricing you want to pay for Spot resources. You will always pay the lowest Spot market price and never more than your maximum percentage. AWS Batch will use 100% as the default maximum price if you leave this field empty.

#### Minimum vCPUs

0

By keeping this set to 0 you will not have instance time wasted when there is no work to be run. If you set this above zero you will maintain that number of vCPUs at all times.

#### Maximum vCPUs

256

EC2 instance type limits will determine the maximum vCPUs you can have. [Learn more](#)

#### Desired vCPUs - optional

0

#### Allowed instance types

Choose options



optimal

Clear instance types

Optimal chooses the best fit of M4, C4, and R4 instance types available in the region. Jobs define their vCPU and memory requirements at submission and that information is used to match the job with the most appropriately sized instance.

#### Allocation strategy

SPOT\_CAPACITY\_OPTIMIZED



Allocation Strategies allow you to choose how Batch launches instances on your behalf. We recommend Best Fit Progressive for On-Demand CEs and Spot Capacity Optimized for Spot CEs. This will make it much more likely Batch will be able to secure the needed capacity for your workloads by pulling from diverse instance types. However, if you want to ensure Batch chooses only the lowest priced instance types appropriate to your jobs, you can select the Best Fit strategy.

- job queue
- job definition

- jobs

## Python

[Source code and documentation can be found here](#)