AWS re:Invent

CON218

How Amazon Lex uses Amazon ECS to process batches at scale for conversational bots

Steve Kendrex
Senior Product Manager
AWS Batch
Amazon Web Services

Harshal Pimpalkhute
Principal Product Manager
Amazon Lex
Amazon Web Services

Why is AWS great for ML?

Broadest and deepest platform choice

Categories

General purpose

Burstable

Compute intensive

Memory intensive

Storage (high I/O)

Dense storage

GPU compute

Graphics intensive

Capabilities

NEW! Choice of processor
(AWS, Intel, AMD)

Fast processors
(up to 4.0 GHz)

High memory footprint
(up to 12 TiB)

Instance storage
(HDD and NVMe)

Accelerated computing
(GPUs and FPGA)

NEW! Networking
(up to 100 Gbps)

Bare Metal

Size
(Nano to 32xlarge)

Options

Amazon Elastic Block Store
(Amazon EBS)

Amazon Elastic Graphics

NEW! Amazon
Elastic Inference

175

instance types

for virtually
every workload
and business need

AWS Batch

Introducing AWS Batch

Fully managed

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

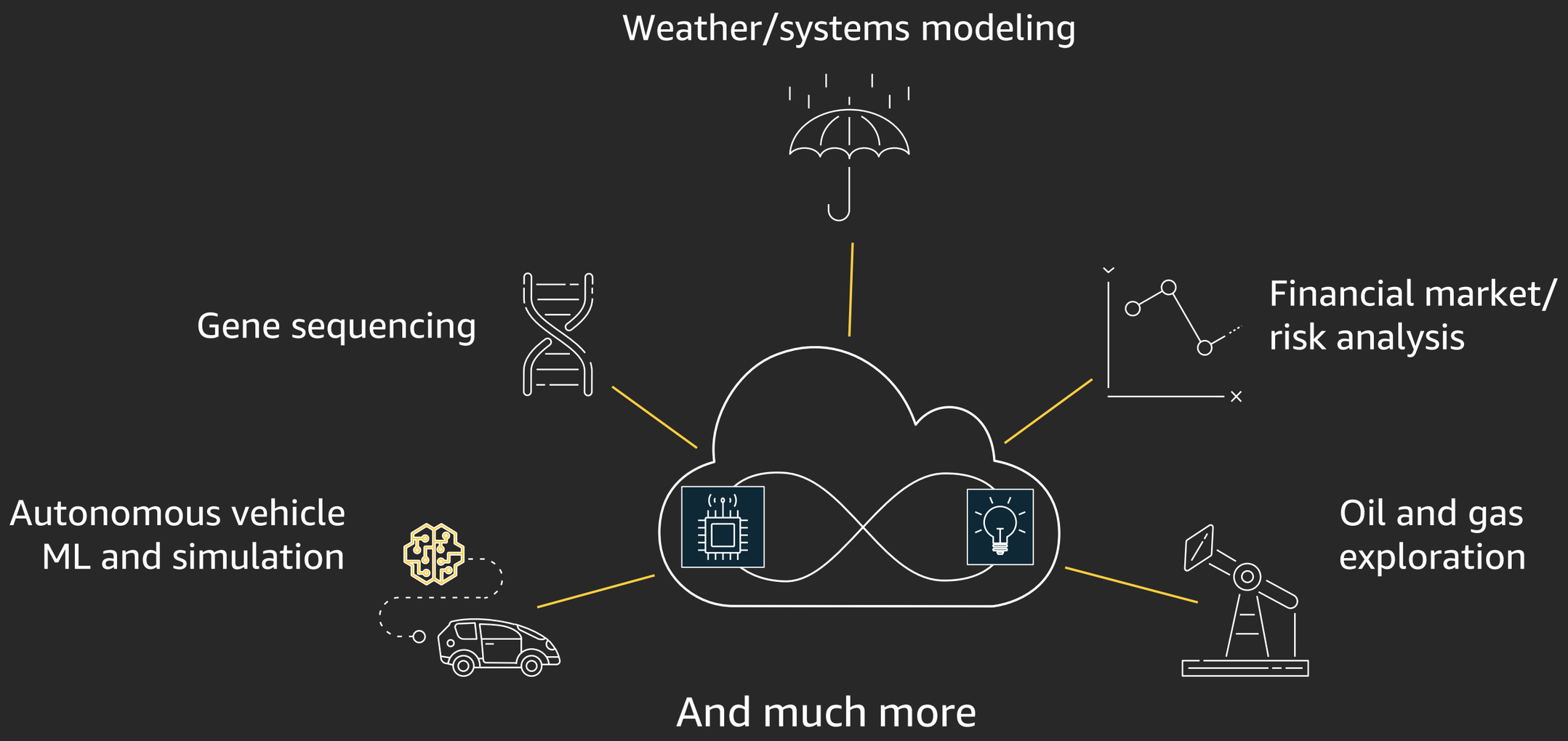
Integrated with AWS

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

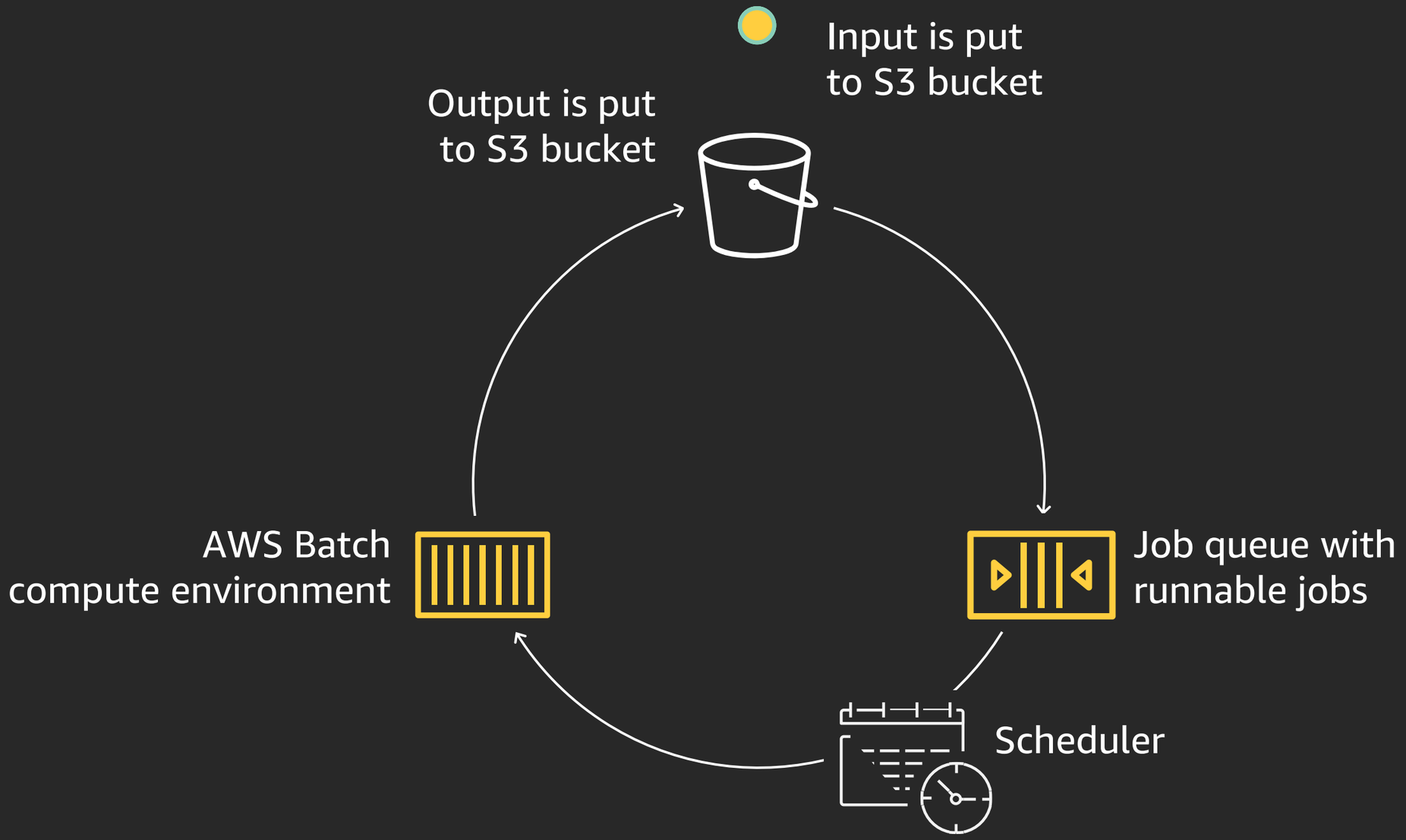
Cost-optimized resource provisioning

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

Who uses AWS Batch?



Typical AWS Batch job architecture



Job definition

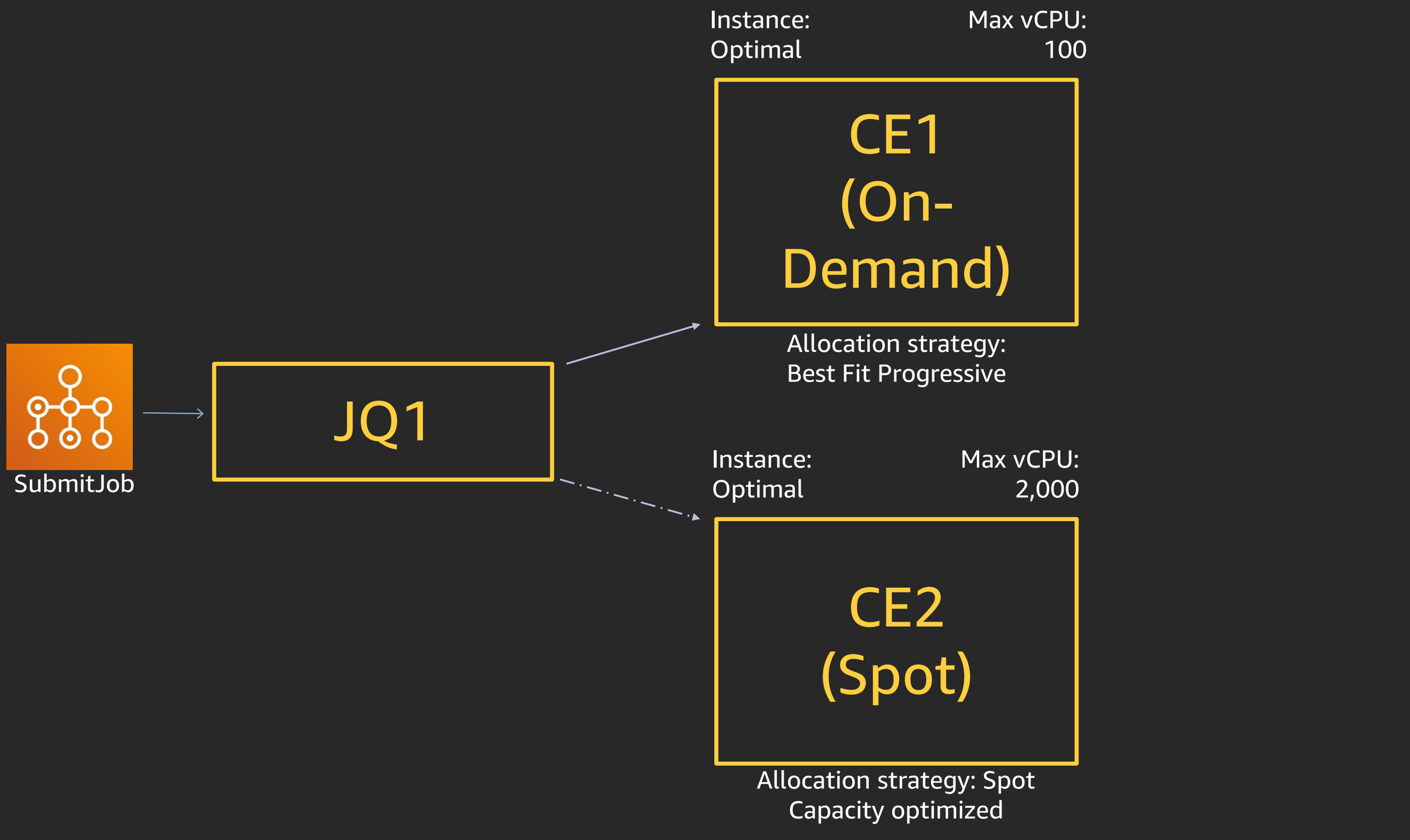
Application
image + config

IAM role

New: Allocation strategies for AWS Batch

Make capacity/throughput/cost tradeoffs

- Spot capacity optimized: Allow AWS to predict the deepest Spot capacity pools and launch instances accordingly. Available in Spot only.
- Best fit: Same behavior as previously; CEs that are created through SLI/SDK will default to this (to preserve backward compatibility). Spot or On-Demand CEs supported. Not recommended for most use cases.
- Best fit progressive: Same as best fit, but when we reach a capacity error (ICE, Reclaim, EC2 limit), AWS Batch will progressively sort through the list and pick the next best fit instance type. Recommended for OD CEs, or in Spot CEs with a specific use case.


Instance:
Optimal
Max vCPU:
100
CE1
(On-
Demand)
Allocation strategy:
Best Fit Progressive
SubmitJob
JQ1
Instance:
Optimal
Max vCPU:
2,000
CE2
(Spot)
Allocation strategy: Spot
Capacity optimized

Jobs

Jobs are the unit of work executed by AWS Batch as containerized applications running on Amazon Elastic Compute Cloud (Amazon EC2)

Containerized jobs can reference a container image, command, and parameters, or users can simply provide a .zip containing their application and we will run it on a default Amazon Linux container

```
aws batch submit-job --job-name sim-variation-1 \
                     --job-definition sim-sensors \
                     --job-queue high-mem-and-cpu
```

Job definitions

Similar to Amazon ECS task definitions, AWS Batch job definitions specify how jobs are to be run. While each job must reference a job definition, many parameters can be overridden. Some of the attributes specified in a job definition:

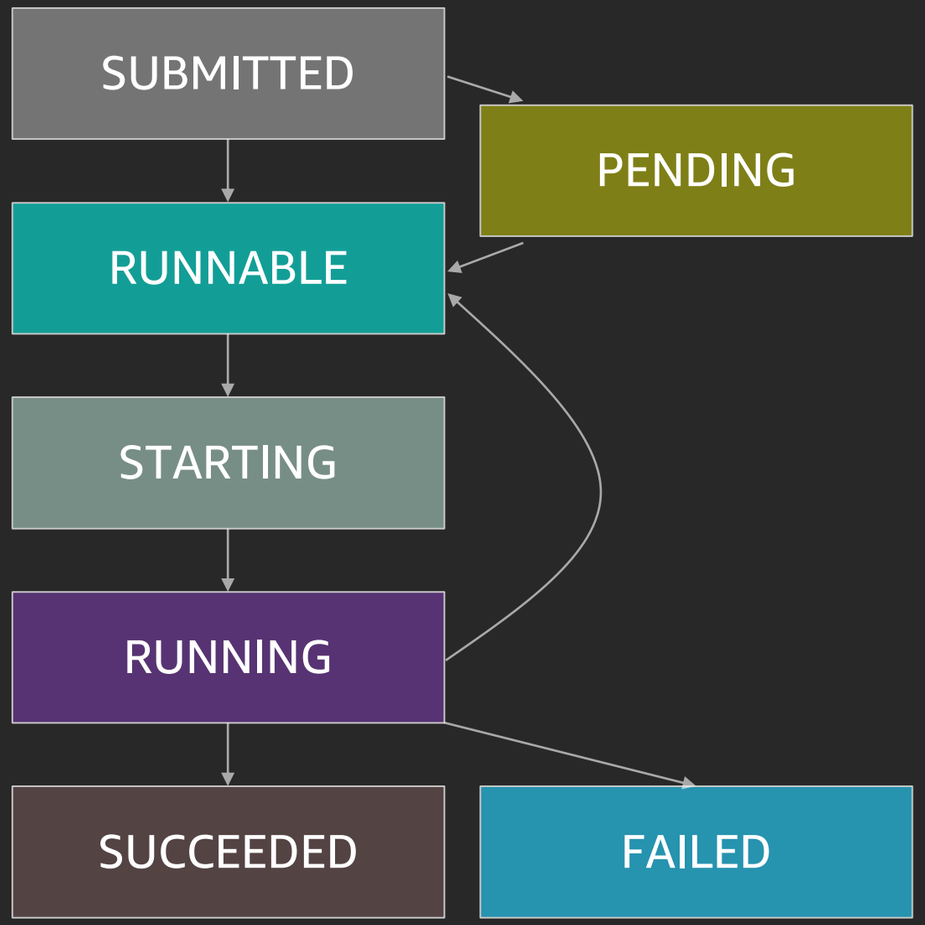
- AWS Identity and Access Management (IAM) role associated with the job
- vCPU and memory requirements
- Retry strategy
- Mount points
- Container properties
- Environment variables

```
aws batch register-job-definition --job-definition-name sim\
                                  --container-properties ...
```

Job states

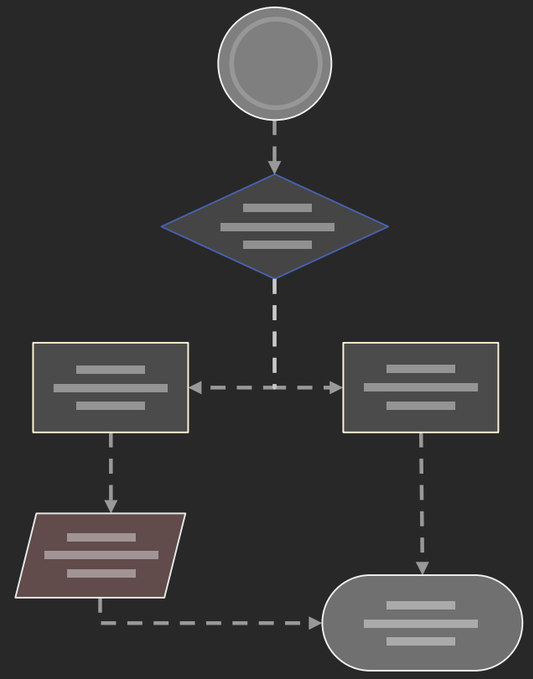
Jobs submitted to a queue can have the following states:

- **SUBMITTED:** Accepted into the queue, but not yet evaluated for execution
- **PENDING:** Your job has dependencies on other jobs which have not yet completed
- **RUNNABLE:** Your job has been evaluated by the scheduler and is ready to run
- **STARTING:** Your job is in the process of being scheduled to a compute resource
- **RUNNING:** Your job is currently running
- **SUCCEEDED:** Your job has finished with exit code 0
- **FAILED:** Your job finished with a non-zero exit code or was canceled or terminated



Workflows, pipelines, and job dependencies

Jobs can express a dependency on the successful completion of other jobs or specific elements of an array job

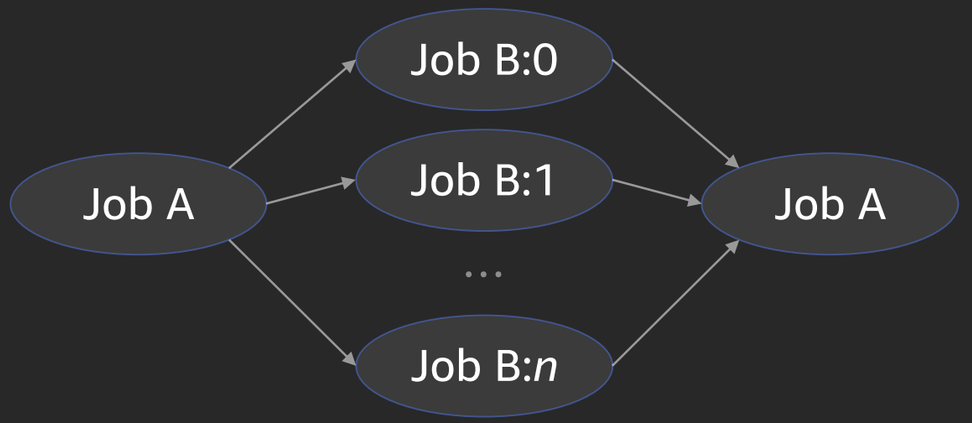


Use your preferred workflow engine and language to submit jobs. Flow-based systems simply submit jobs serially, while DAG-based systems submit many jobs at once, identifying inter-job dependencies.

```
aws batch submit-job --depends-on 606b3ad1-aa31-48d8-92ec-f154bfc8215f
```

Array jobs

Instead of submitting a large number of independent simple jobs, we also support array jobs that run many copies of an application against an array of elements



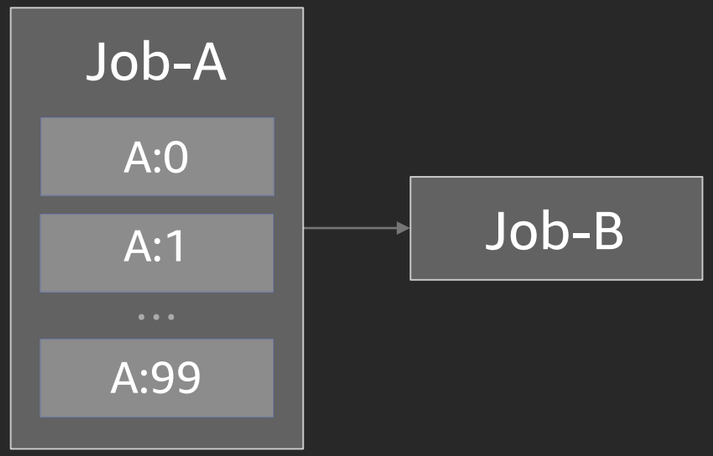
Array jobs are an efficient way to run:

- Parametric sweeps
- Monte Carlo simulations
- Processing a large collection of objects

```
aws batch submit-job --job-name sim-variation-1 \
                     --job-definition sim-sensors \
                     --job-queue high-mem-and-cpu \
                     --array-properties "{"size": 300}"
```

Array job dependency models

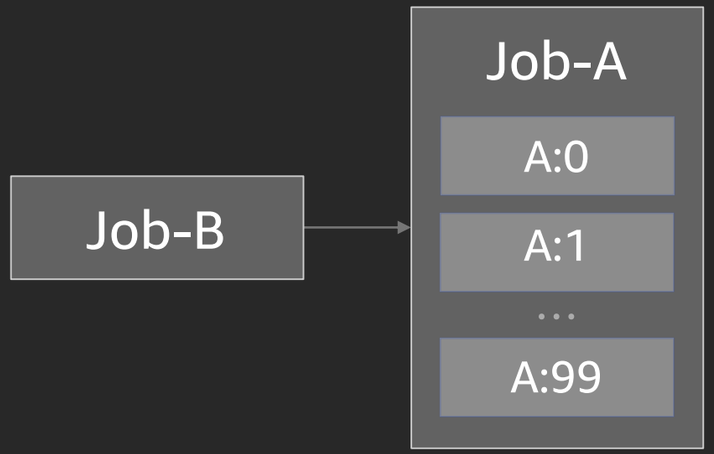
Job depends on array job




```
$ aws batch submit-job –cli-input-json file://./Job-A.json

<Job-A.json>
{
   "jobName": ”Job-A",
   "jobQueue": "ProdQueue",
   "jobDefinition": ”Job-A-Definition:1",
   ”arrayProperties":
   {
        “copies”: 100
   }
}

$ aws batch submit-job –cli-input-json file://./Job-B.json

<Job-B.json>
{
   "jobName": ”Job-B",
   "jobQueue": "ProdQueue",
   "jobDefinition": ”Job-B-Definition:1",
   "dependsOn": [
        {"jobId": "<job ID for Job A>"  }
   ]
}
```

Array job dependency models

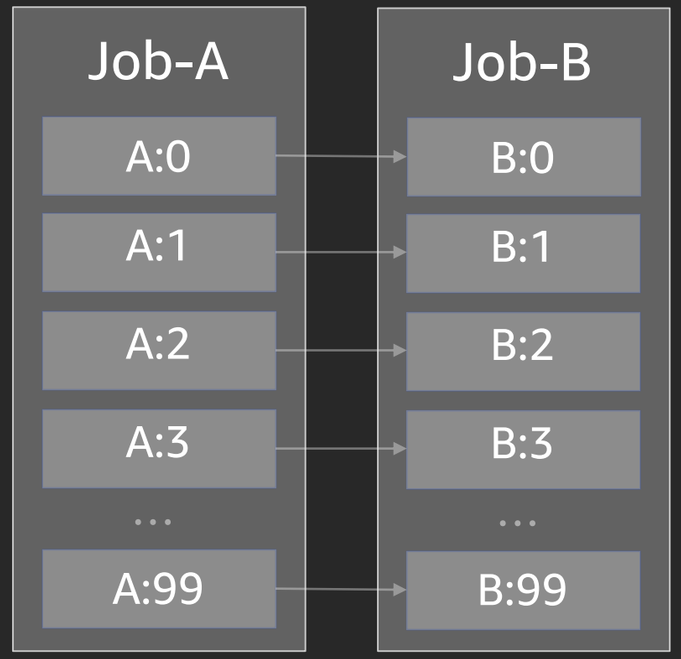
Array job depends on job




```
$ aws batch submit-job --job-name Job-A --job-queue
ProdQueue --job-definition job-A-Definition:1
{
        "jobName": "sequential-stress-10",
        "jobId": "7a6225f0-a16e-4241-9103-192c0c68124c”
}

<Job-B.json>
{
   "jobName": ”Job-A",
   "jobQueue": "ProdQueue",
   "jobDefinition": ”Job-A-Definition:1",
   ”arrayProperties":
   {
        “copies”: 100
   },
   "dependsOn": [
         {"jobId": "7a6225f0-a16e-4241-9103-192c0c68124c"
}
   ]
}
```

Array job dependency models

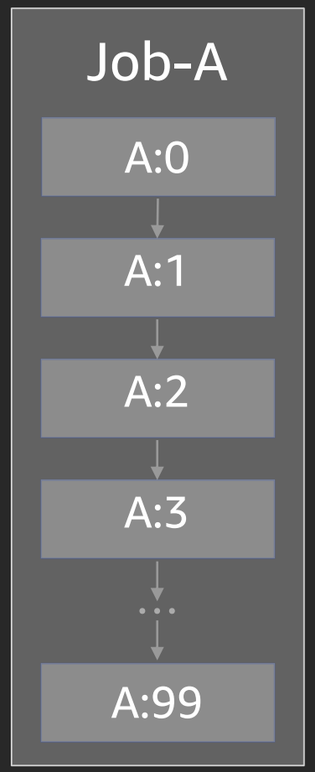
Two equally sized array jobs, a.k.a. “N-to-N”




```
$ aws batch submit-job --job-name Job-A --job-queue
ProdQueue --job-definition job-A-Definition:1 --array-
properties size=10000
{
        "jobName": ”Job-A",
        "jobId": "7a6225f0-a16e-4241-9103-192c0c68124c”
}

$ aws batch submit-job --job-name Job-B --job-queue
ProdQueue --job-definition job-B-Definition:1 --array-
properties size=10000 --depends-on jobId=7a6225f0-a16e-
4241-9103-192c0c68124c,type=N_TO_N
{
        "jobName": ”Job-B”,
        "jobId": "7f2b6bfb-75e8-4655-89a5-1e5b233f5c08”
}
```

Array job dependency models

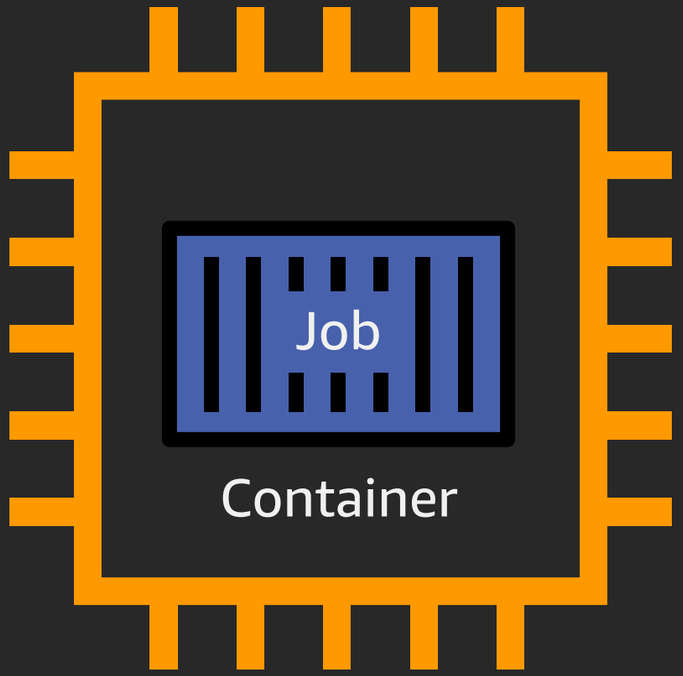
Array job depends on self, a.k.a. sequential job



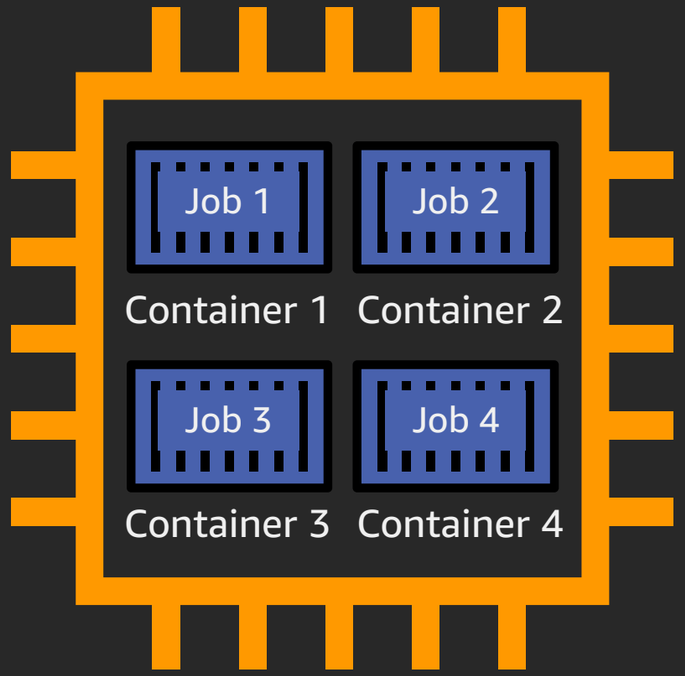

```
$ aws batch submit-job --job-name Job-A --job-queue
ProdQueue --job-definition job-A-Definition:1 --
array-properties size=10 --depends-on type=SEQUENTIAL
{
      "jobName": ”Job-A",
      "jobId": "7a6225f0-a16e-4241-9103-192c0c68124c”
}
```

Model example

C is dependent on A,
C N_TO_N dependency on B, same for D and C,
E and F depend on D



Instance

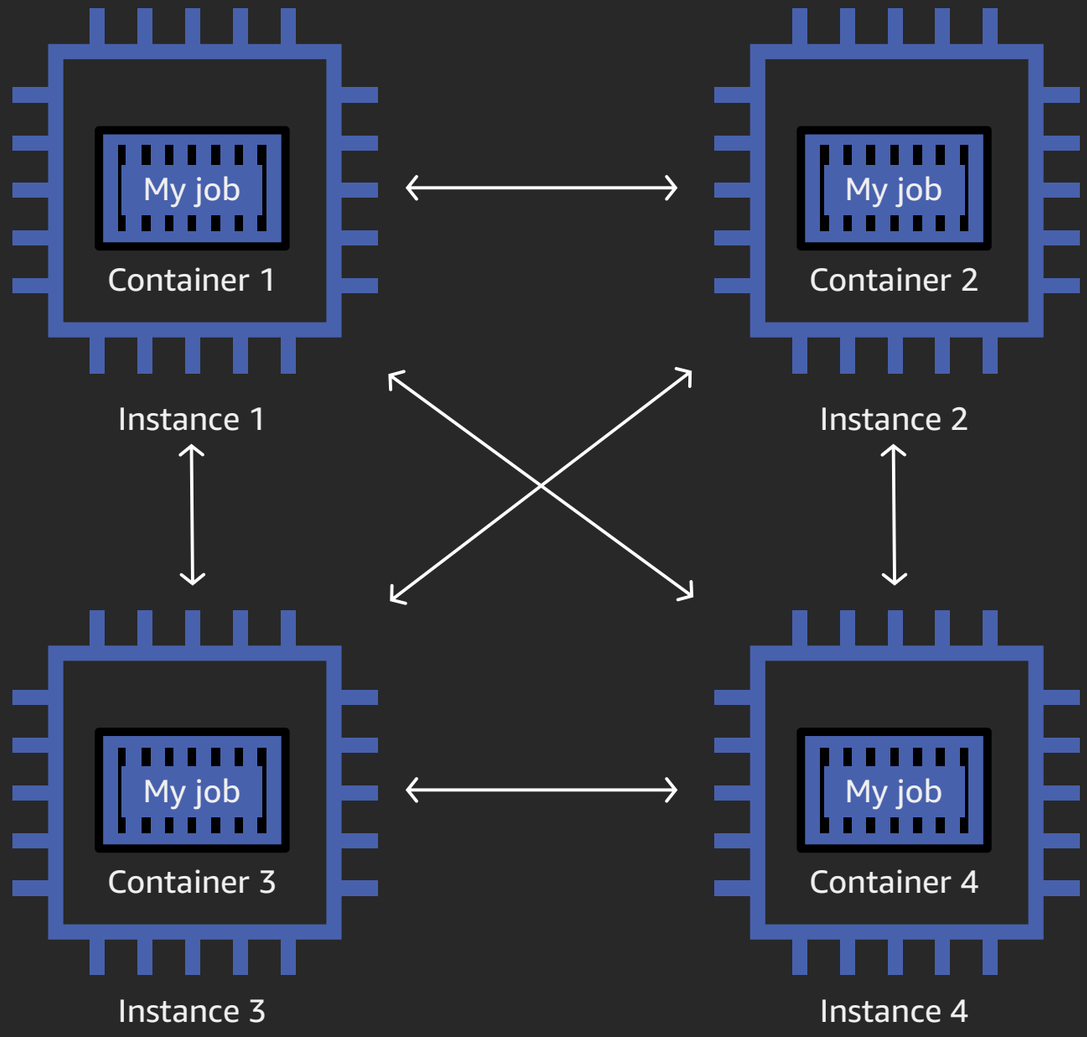


Instance

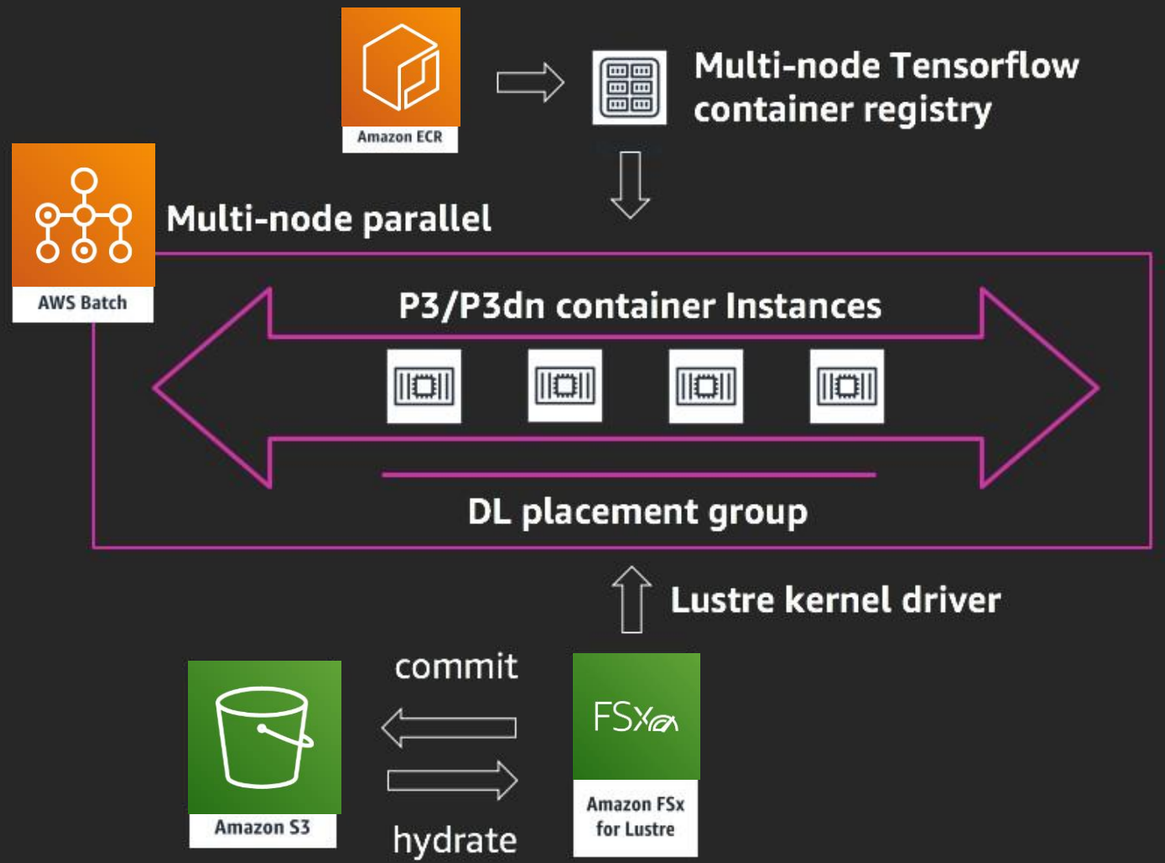
Multi-node parallel jobs on AWS Batch

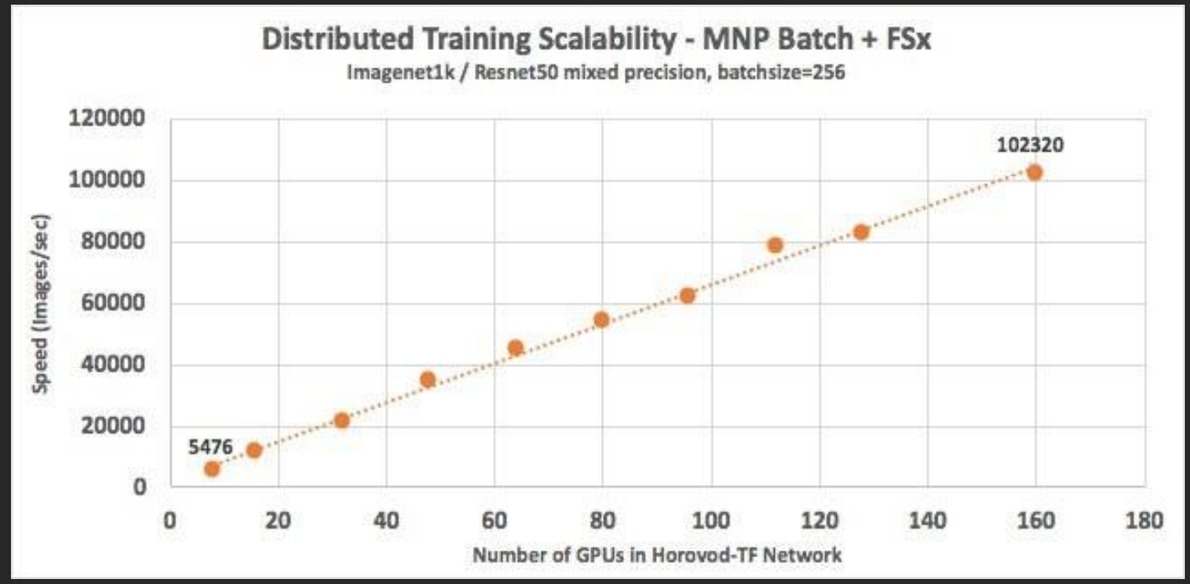
A multi-node parallel (MNP) job enables AWS Batch to run single jobs which span multiple EC2 instances

Integrated with the Elastic Fabric Adaptor for low latency between nodes



Deep neural network training

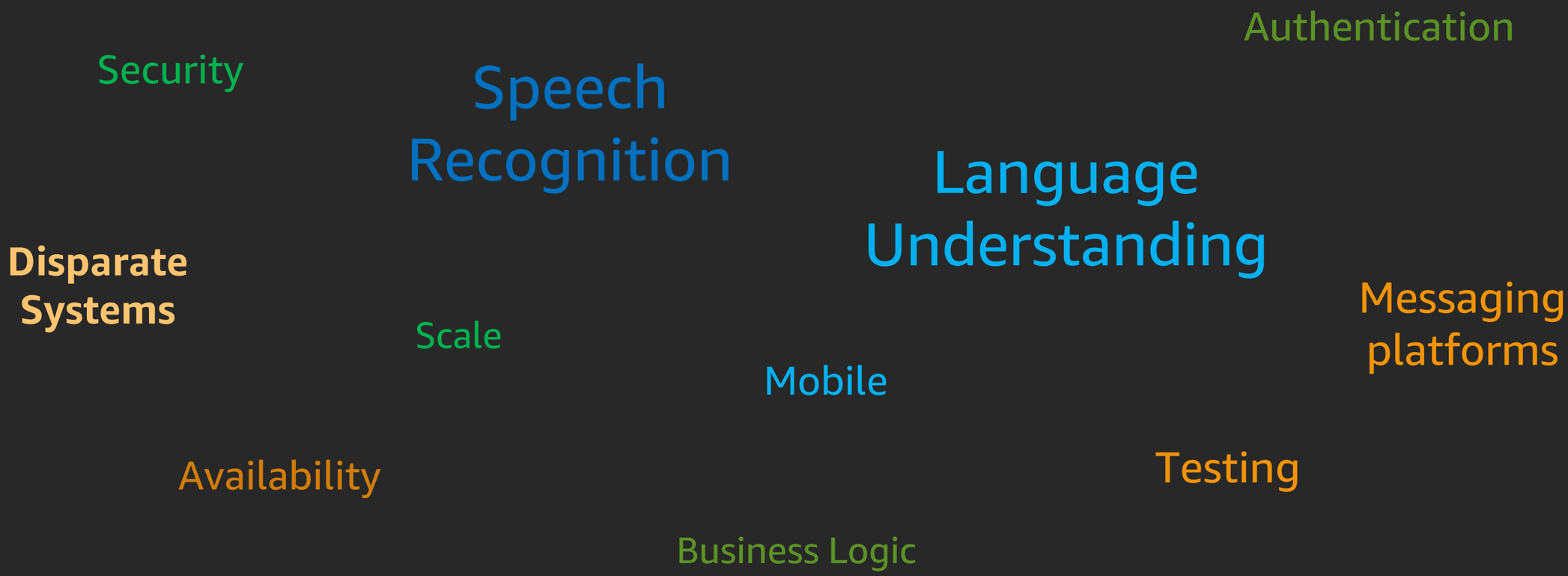




Amazon Lex

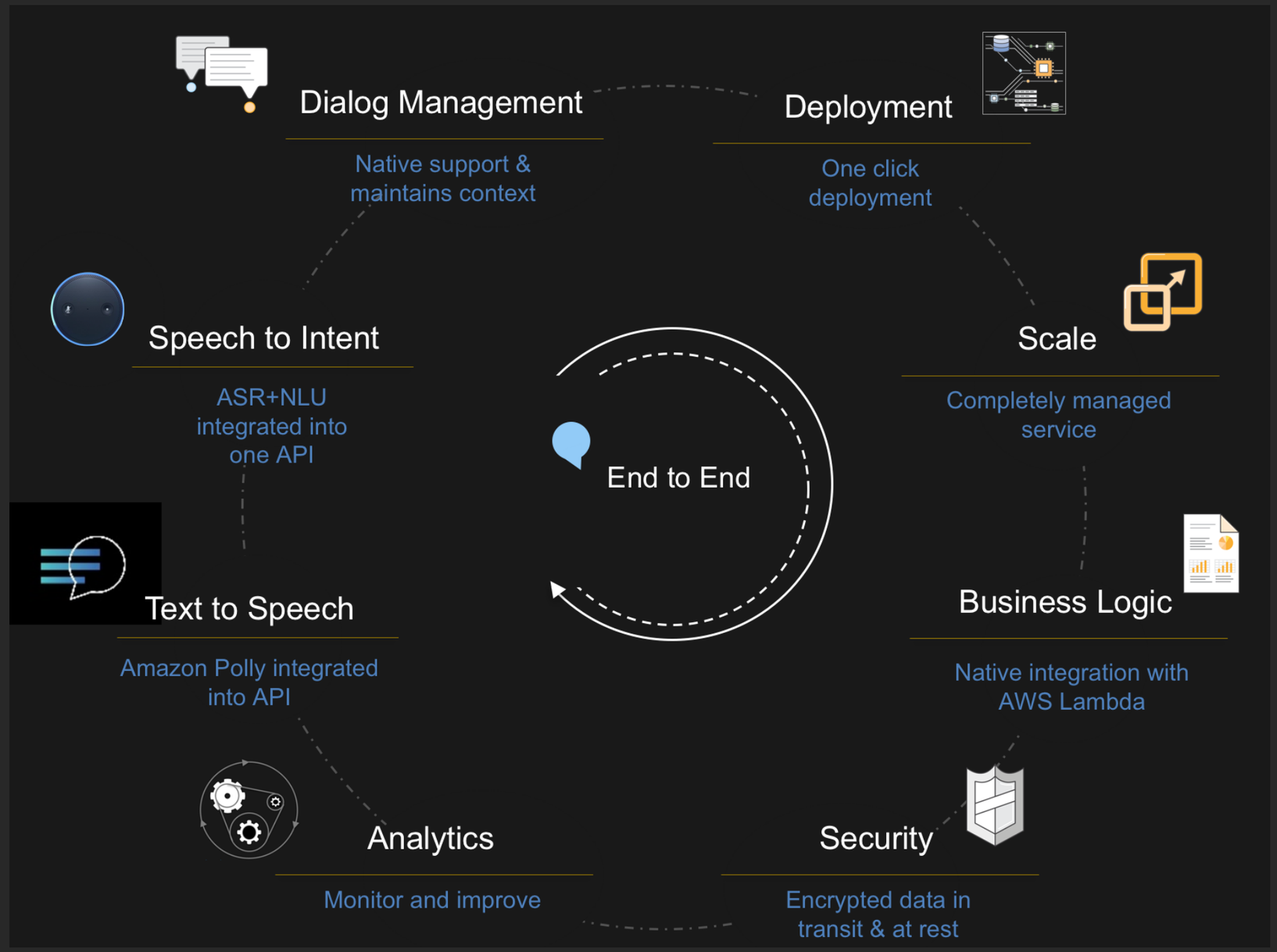
Quickly and easily build sophisticated, natural language conversational interfaces using voice and text

Challenges



Conversational interfaces need to combine a large number of sophisticated algorithms and technologies

Amazon Lex


Dialog Management
Native support & maintains context
Deployment
One click deployment
Speech to Intent
ASR+NLU integrated into one API
End to End
Scale
Completely managed service
Text to Speech
Amazon Polly integrated into API
Business Logic
Native integration with AWS Lambda
Analytics
Monitor and improve
Security
Encrypted data in transit & at rest

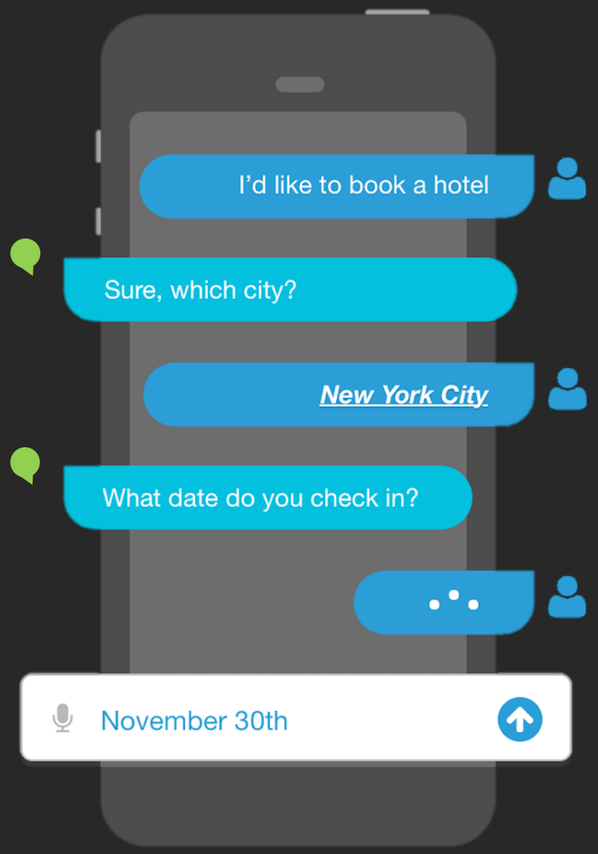
Amazon Lex

Speech recognition

Natural language understanding

Powered by the same deep learning technology as Alexa

Amazon Lex

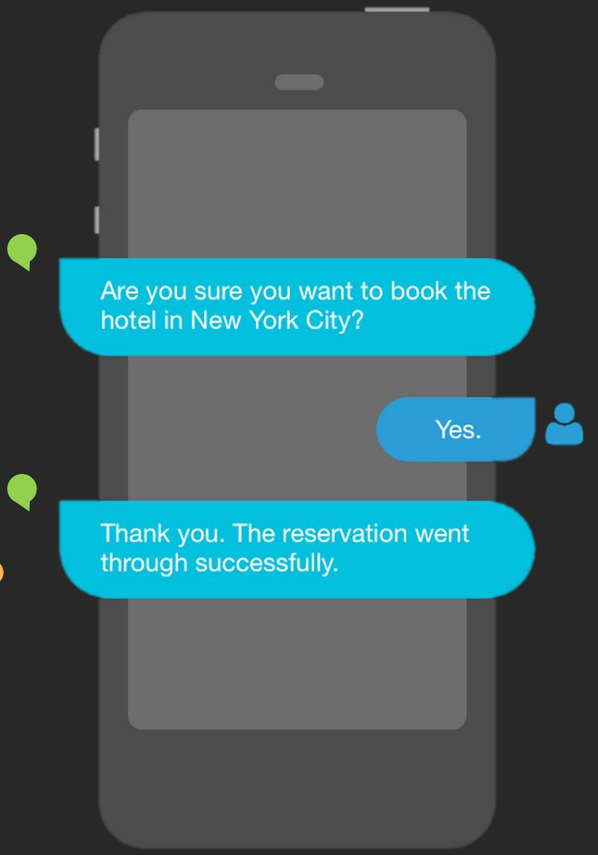


Intents
An intent performs an action in response to natural language user input

Utterances
Spoken or typed phrases that invoke your intent

Slots
Slots are input data required to fulfill the intent

Fulfillment
Fulfillment mechanism for your intent



“Book a hotel”

“Book a Hotel in NYC”

Book | a | Hotel | in | NYC

Automatic speech recognition

Hotel booking

New York City

Natural language understanding

Intent/Slot Model

Utterances

Hotel Booking	
City	New York City
Check In	April 19th
Check Out	April 21st

“Can I go ahead with the booking?

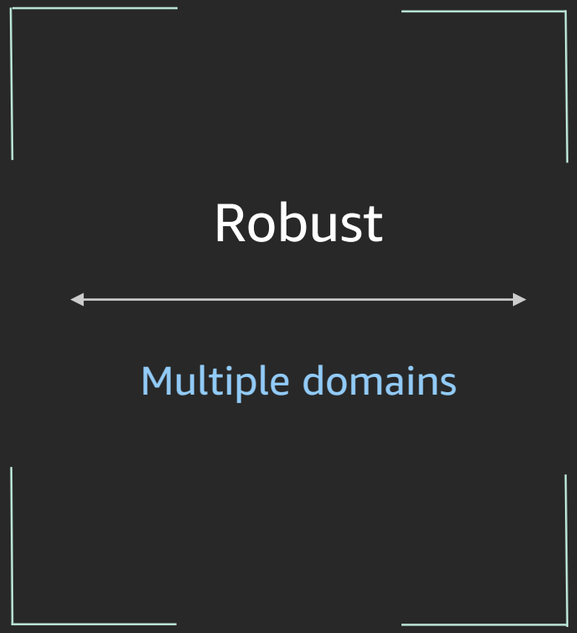
λ

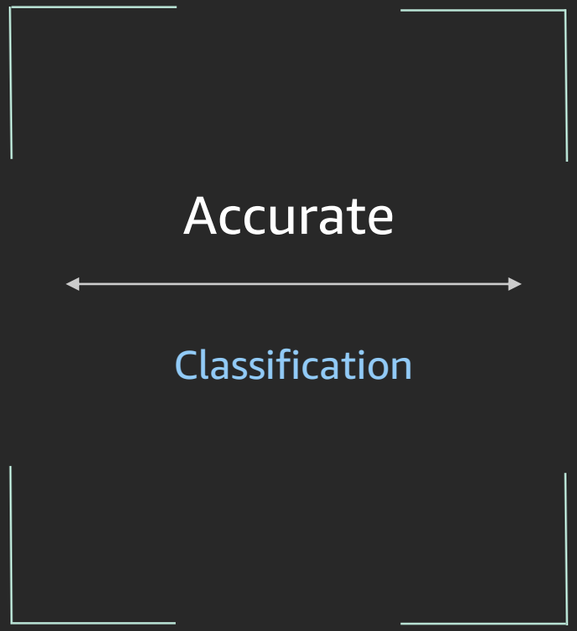
Confirmation: “Your hotel is booked for April 19th “

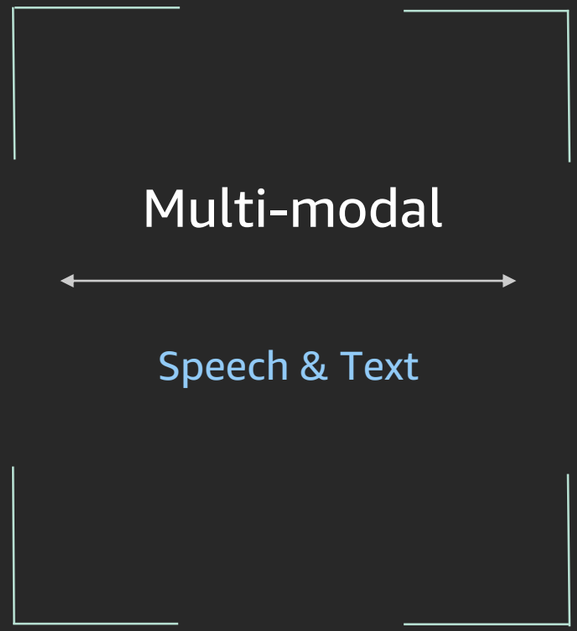
Amazon Polly

“Your hotel is booked for April 19th”

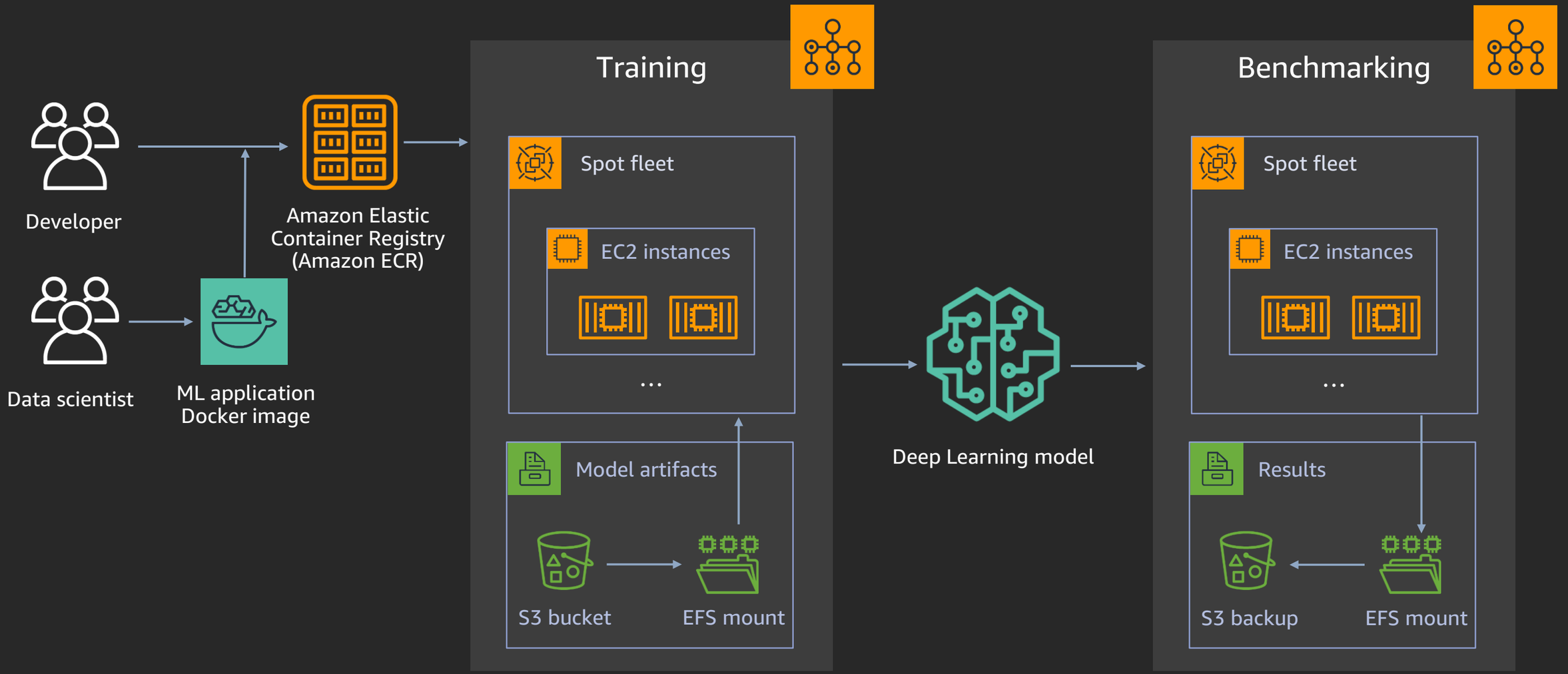
Natural language understanding







Amazon Lex modeling with AWS Batch



Thank you!

Steve Kendrex
kendrexs@amazon.com

Harshal Pimpalkhute
hhar@amazon.com

AWS re:Invent

aws

Please complete the session survey in the mobile app.