

CON304

AWS re:Invent

Batch Processing with Containers on AWS

Zaven Boni

Technical Manager, GoPro

Lee Baker

HERE, Sr. Architect, Highly Automated Driving

Tom Fuller

Principal Solutions Architect, AWS

Jamie Kinney

Principal Product Manager, AWS Batch and HPC

November 30, 2017

AWS re:Invent

© 2017 Amazon Web Services, Inc. or its Affiliates. All rights reserved.



Introductions



Zaven Boni: Technical Manager, GoPro



Lee Baker: Sr. Architect, Highly Automated Driving, HERE



Tom Fuller: Principal Solutions Architect, AWS

Jamie Kinney: Principal Product Manager, AWS Batch & HPC

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

Agenda

- Intro to Batch Processing
- Amazon ECS
- Migrating GoPro Plus to Amazon ECS
- AWS Batch: overview and recent launches
- HERE
- Q&A

What is Batch Computing?



Run jobs asynchronously and automatically across one or more computers.



Jobs may have dependencies, making the sequencing and scheduling of multiple jobs complex and challenging.

Challenges of Running Batch Workloads

- Typically resource intensive
- Time constraint for completion
- Potential impact to concurrent batch jobs
- Scaling infrastructure resources
- Ensuring effective resource utilization and cost savings
- Fragile and unreliable



What Batch Workloads Need



Reliability



Easy Development



Easy Deployment



High Throughput



Low Ops Load

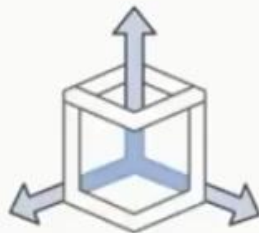


Cost Efficiency

Why the Cloud Makes Sense for Batch Workloads



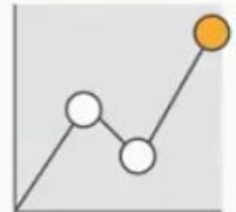
Reliable



Scalable



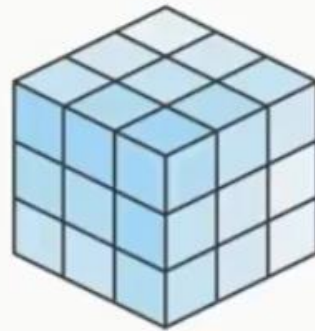
Infrastructure as
code



Pay as you go

Why Containers Make Sense for Batch Workloads

- Simple to model
- Polyglot
- Image is the version
- Do one thing well
- You build it, you run it
- Black box



Options for Batch Workloads on AWS



AWS Batch



Amazon ECS

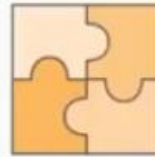
What is Amazon ECS?



Scalable Container Management



Flexible Container Placement



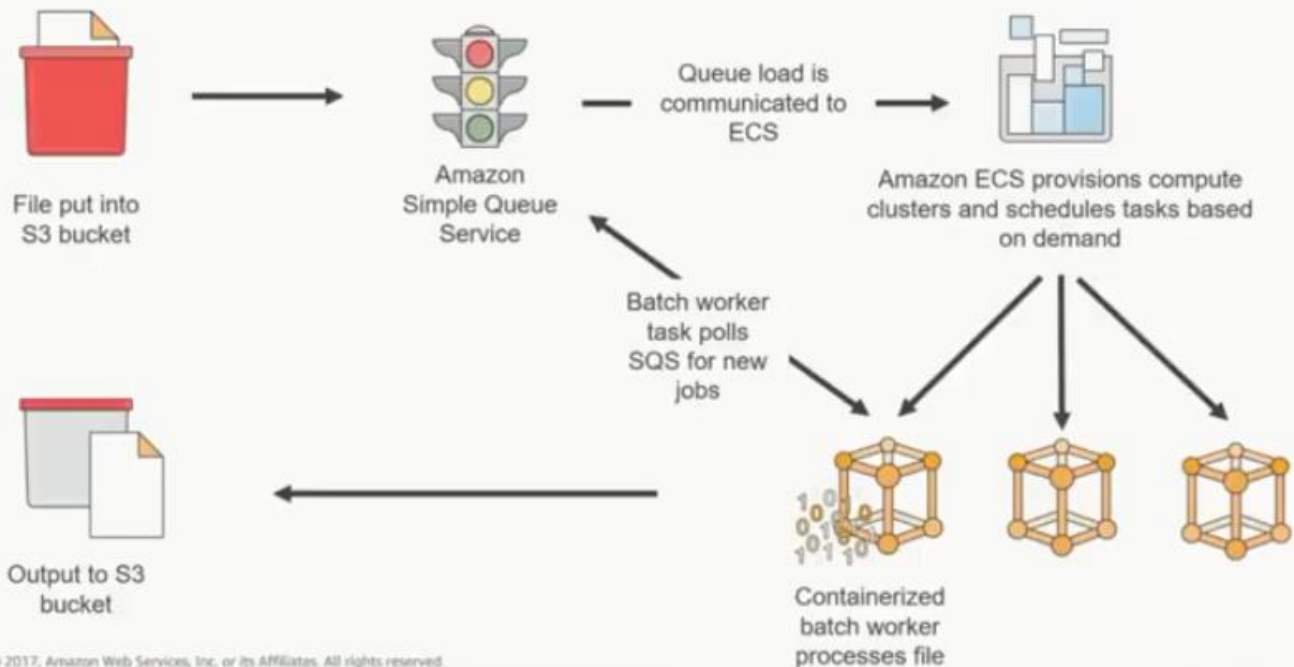
Native AWS Integrations



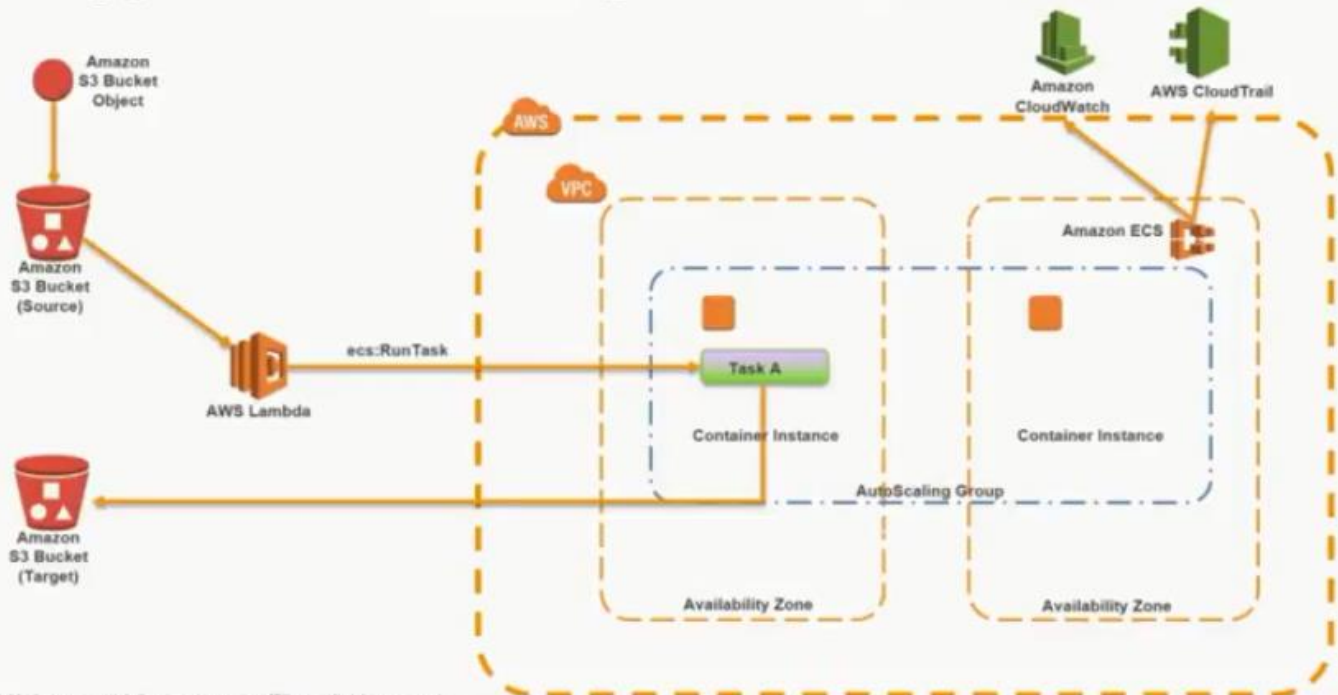
Extensible by Design

ECS is a way for you to bring your Docker containers and schedule those resources as tasks that get run on your ECS instances and clusters.

Basic Batch Workflow with ECS

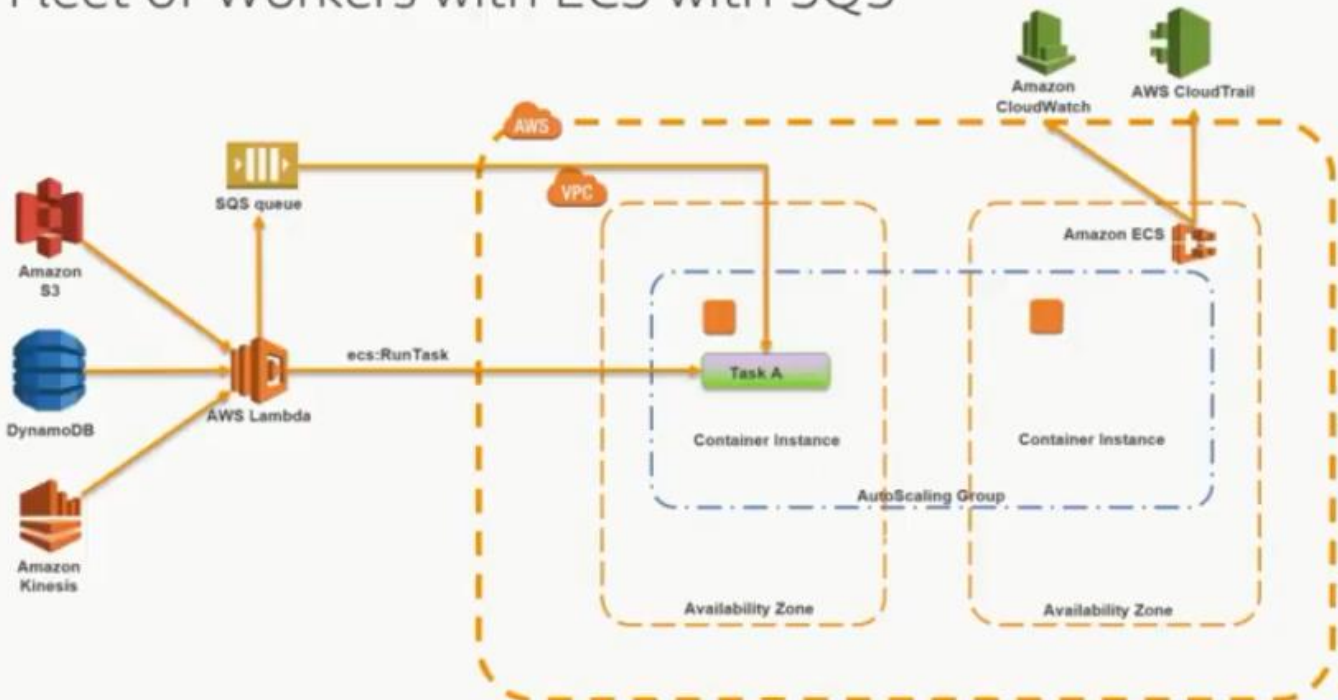


Trigger Batch Processing with AWS Lambda



© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

Fleet of Workers with ECS with SQS



© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

Long-Running Batch Jobs

- Utilize Spot Instances
- EC2 Spot Blocks for Defined-Duration Workloads
- ECS event stream for CloudWatch Events
- Service Scaling and Monitoring



AWS
re:Invent

© 2017, Amazon Web Services, Inc., or its Affiliates. All rights reserved.



Get the Best Value for EC2 Capacity – Spot Instances

- Since Spot instances typically cost 50-90% less than On-Demand, you can **increase your compute capacity by 2-10x within the same budget**
- Or, you could **save 50-90% on your existing workload**
- Either way, **it's worth it!**



Best Practices

- Store state and inputs, outputs in S3 or another datastore
- Minimize dependencies between task definitions (should be independent of each other)
- Use Spot Instances and Spot fleets for long-running batch jobs
- Monitor cluster state with ECS APIs
- Share pools of resources
- Auto Scaling, VPC, IAM, scheduled Reserved Instances



Zaven Boni: Technical Manager, GoPro

@opstastic

Migrating **GoPro Plus** to ECS



GoPro Plus



GoPro plus is a subscription service that makes it easy to automatically upload, edit, share, and delete photos from your GoPro devices. When a user uses this service, they will be interacting with the GoPro Plus EC2 containers running our different apps inside of ECS.

PLATFORM COMPONENTS



- Photo and video processing
- User and subscription management
- Device management
- Web front-ends
- Logging and analytics
- Infrastructure tools and utilities



PLATFORM SCALE

60+

ECS Services

500*

EC2 Instances

100M

API Requests/day

** and shrinking...*



Journey to ECS

CHALLENGES BEFORE ECS

APPS & MICRO-SERVICES

- Custom AMIs
- Long deployment times (AMI creation + EC2 instance boot time + application startup)
- Multi-step deployments = more places to fail
- One container per EC2 instance!

CHALLENGES BEFORE ECS

3RD-PARTY QUEUING AND PROCESS MANAGEMENT

Visibility

- Orchestration was a black box
- Scraped 3rd party API to create Cloudwatch custom metrics

Control

- Scaling parameters were abstracted away
- Docker bugs caused silent failures

Cost

- Operational overhead
- Inefficient use of resources

EVALUATING ECS



?

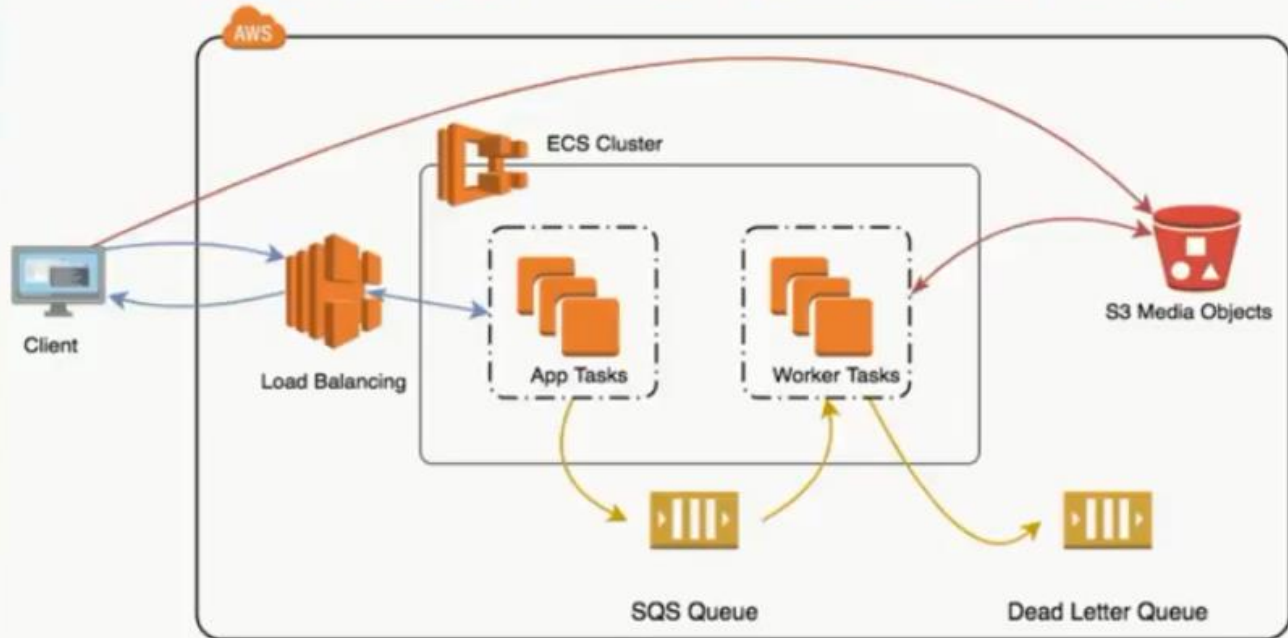


EVALUATING ECS

- IAM Roles specific to each service running on the cluster - principle of least privilege
- Familiar AWS interface and concepts
- Integration with AWS services like Cloudwatch, ELB
- Enterprise support
- Less cluster maintenance



MEDIA SERVICE ARCHITECTURE

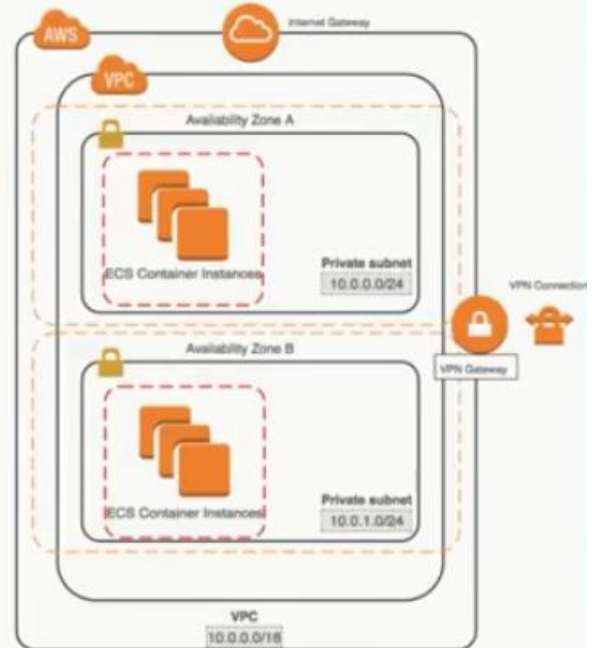


The Migration

INFRASTRUCTURE AS CODE

We Terraformed (almost) everything

- VPCs
- ECS clusters
- SQS queues
- ECS task definitions
- Resource reservations
- Docker image tags
- IAM per-service roles
- Environment variables
- Cloudwatch scaling thresholds
- Deployments?



Deploying (scaling and teardown) to ECS can be done using Terraform definitions (checked in with the code in GitHub) or CloudFormation scripts

INFRASTRUCTURE AS CODE

TF config and module structure

- DevOps team maintain base modules
- Development teams import modules into their app codebase
- Terraform definitions for each app are maintained along with the app code repository

A few project-specific variables

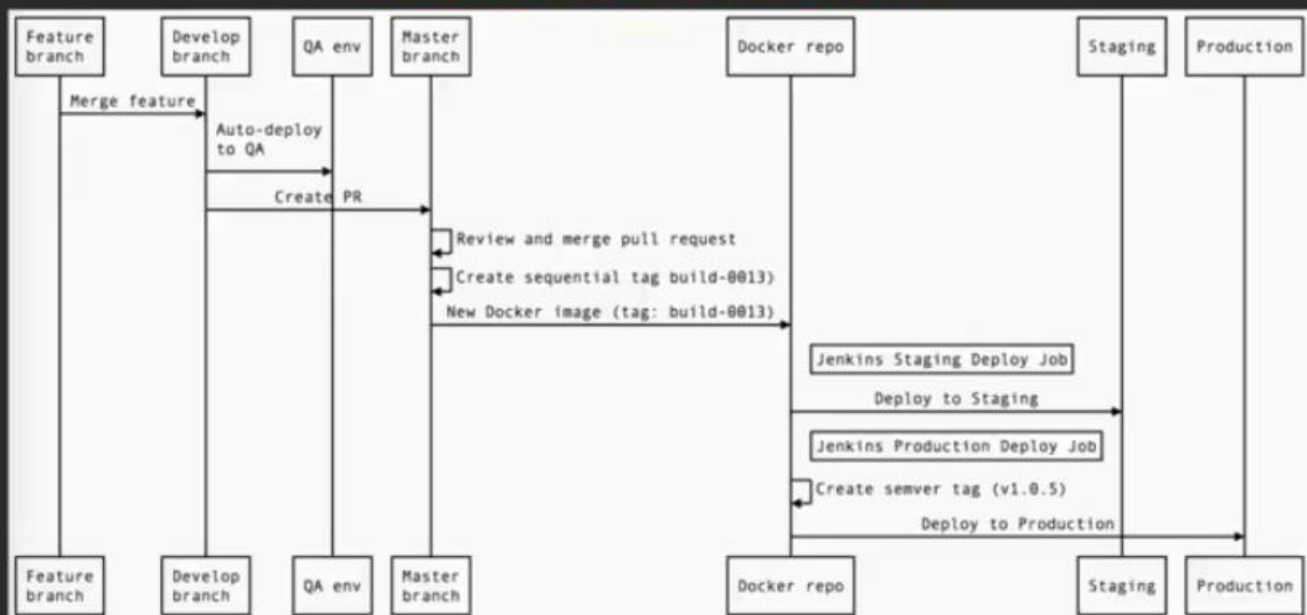
- Service name
- ECS Task definition
 - CPU Allocation
 - Hard/Soft memory limits
- Cloudwatch alarm thresholds
 - SQS max queue length

RELEASE TAGGING

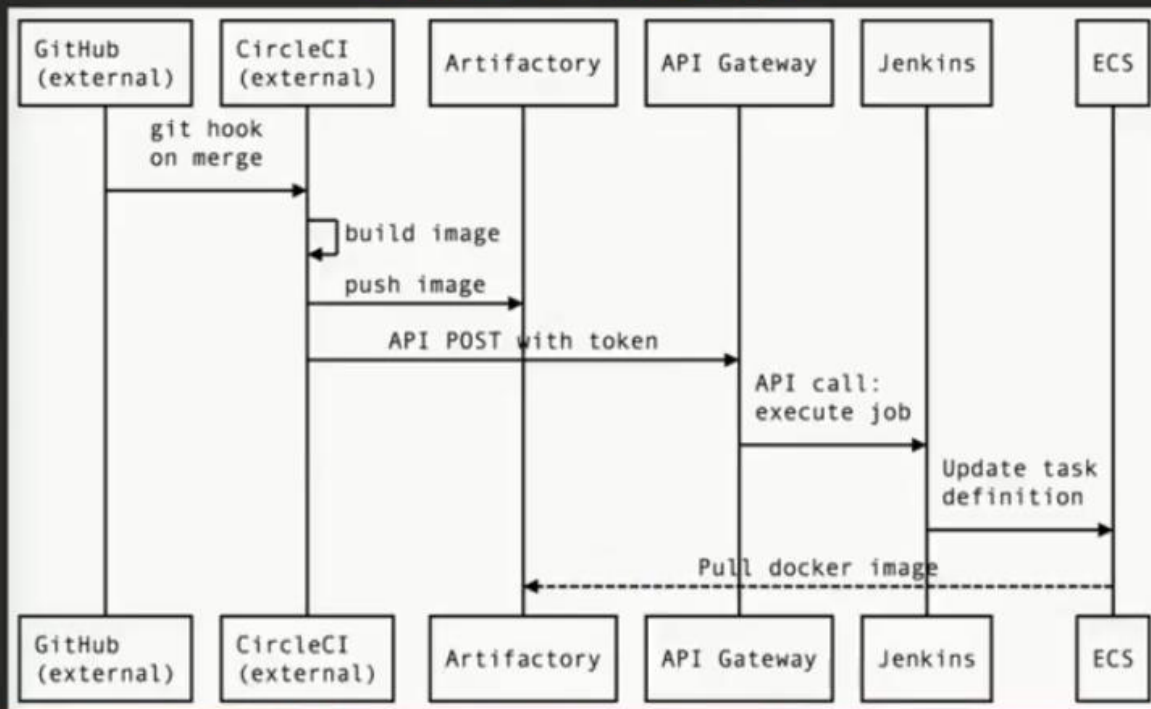
- Use a semantic versioning scheme to track releases through pre-prod and production deployments
- Apply the same tag to:
 - git commit, Docker image, and ECS task
- Ensures parity between environments
- Rollback with confidence
- Dashboard visibility into the deployed code versions



RELEASE TAGGING



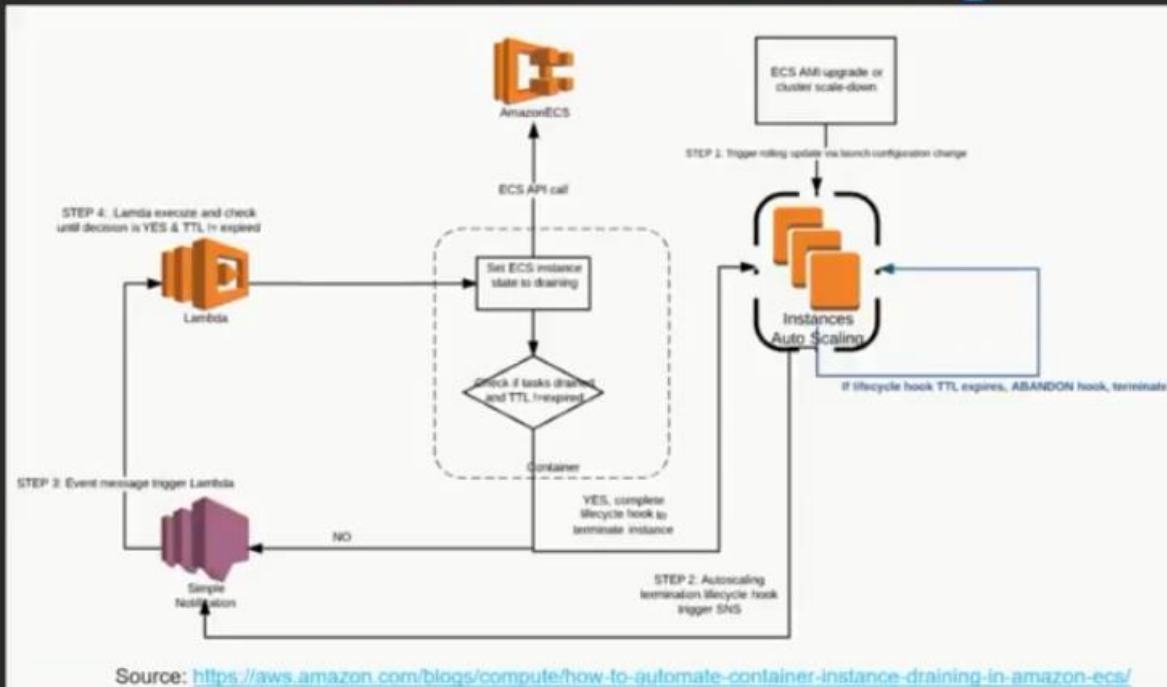
DEPLOYMENTS



AUTO SCALING

- ECS cluster scaling (horizontal)
 - CPU or Memory Reservation (not Utilization)
- Service scaling (vertical)
 - Workers - SQS queue length (*ApproximateNumberOfMessagesVisible*)
 - Services – CPUUtilization or MemoryUtilization
- Scaling down without interrupting tasks
 - Container Instance draining

Container Instance Draining



Lessons Learned

LESSONS: Scaling

1. Scale with headroom
 - Application startup time is a factor
1. Scale on custom metrics
 - Number of messages in the queue

LESSONS: Immutability

- Immutable images are good, and...
 - App code will still break with incorrect ENV
 - App code will still break with environmental differences
 - Lesson: post-deployment testing and automated roll-back


LESSONS: IAM

- Access policies can be complex.
 - If resource tagging isn't universal, tough to provide partial access
 - Isolate your environments in separate AWS accounts

LESSONS: Infra-As-Code

- Executing terraform apply / cloudformation deploy probably requires privileges across many AWS services
 - Solution: same as before
- When updating a task definition, it's all too easy to apply an older container image and unintentionally roll-back
 - Solution: wrapper script checks the ECS API for current version

Realized Benefits



OPERATIONAL BENEFITS

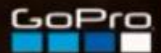
- Simplified deploy pipeline
- Greatly reduced deploy time – 30m to 30s
- Predictability and visibility
- Increased efficiency for mixed workloads
- Simplified ops

In summary...

- ECS is the foundation of the GoPro cloud platform
- In two quarters, we migrated 60+ services
- We're realizing:
 - Cost savings, better deploys, stability
 - Ops and developer happiness :-)

Zaven Boni
@opstastic

Thank you!



Principal Product Manager, AWS Batch & HPC

Jamie Kinney



AWS
re:Invent

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

aws

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

AWS
re:Invent

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

aws

AWS Batch allows you to simply submit your jobs that run in a Docker container to a queue and watch it run in a managed manner. You submit your job using a Job Definition file with the correct metrics like instance types, memory needed, on-demand/spot, etc to a job queue, you can have many job queues for different purposes.



AWS Batch Concepts

- Jobs
- Job Definitions
- Job Queue
- Compute Environments
- Scheduler

2017 Launches



AWS Batch Regional Expansion

AWS Batch is available in the following regions:

- us-east-1 (N. Virginia)
- us-east-2 (Ohio)
- us-west-2 (Oregon)
- eu-west-1 (Ireland)
- eu-west-2 (London)
- eu-central-1 (Frankfurt)
- ap-northeast-1 (Tokyo)
- ap-southeast-2 (Sydney)
- ap-southeast-1 (Singapore)





Improved Managed Compute Environments

Custom AMIs:

- Auto-mount EFS and other shared filesystems
- Larger/faster/encrypted EBS
- GPU/FPGA drivers

New instance families

- G3
- F1
- P3
- C5

Faster scale-up/scale-down and per-second billing

Automated tagging of EC2 Spot instances



© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.



Manageability & Performance

- Automated Job Retries: Easily recover from application errors, hardware failures, or EC2 Spot terminations
- Scheduling throughput: run jobs as short as 5 seconds with ~90% scheduling efficiency
- Native AWS CloudFormation Support for AWS Batch Resources
- Amazon CloudWatch Events for Job State Transitions



© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.



HIPAA Eligibility



AWS re:Invent

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

aws

Workflows, Pipelines, and Job Dependencies



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



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

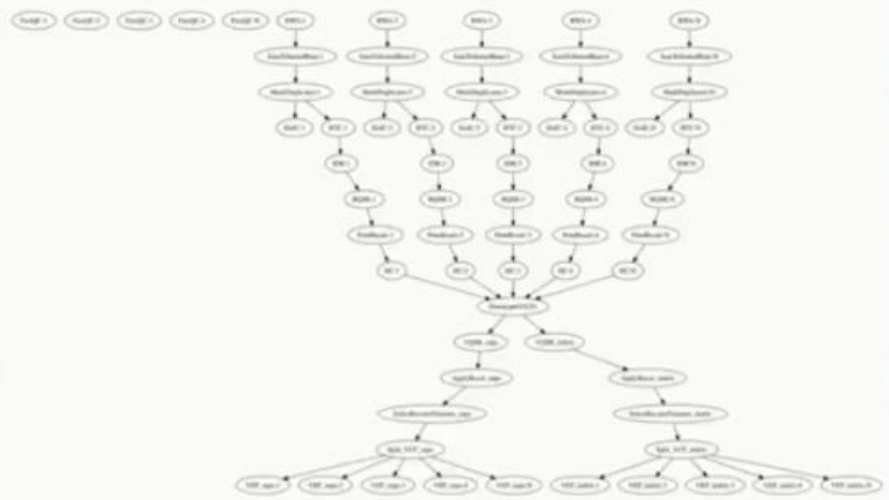
You can also use **AWS Step Functions** or other workflow systems to submit jobs. Flow-based systems submit jobs serially, while DAG-based systems submit many jobs at once, identifying inter-job dependencies.

AWS re:Invent

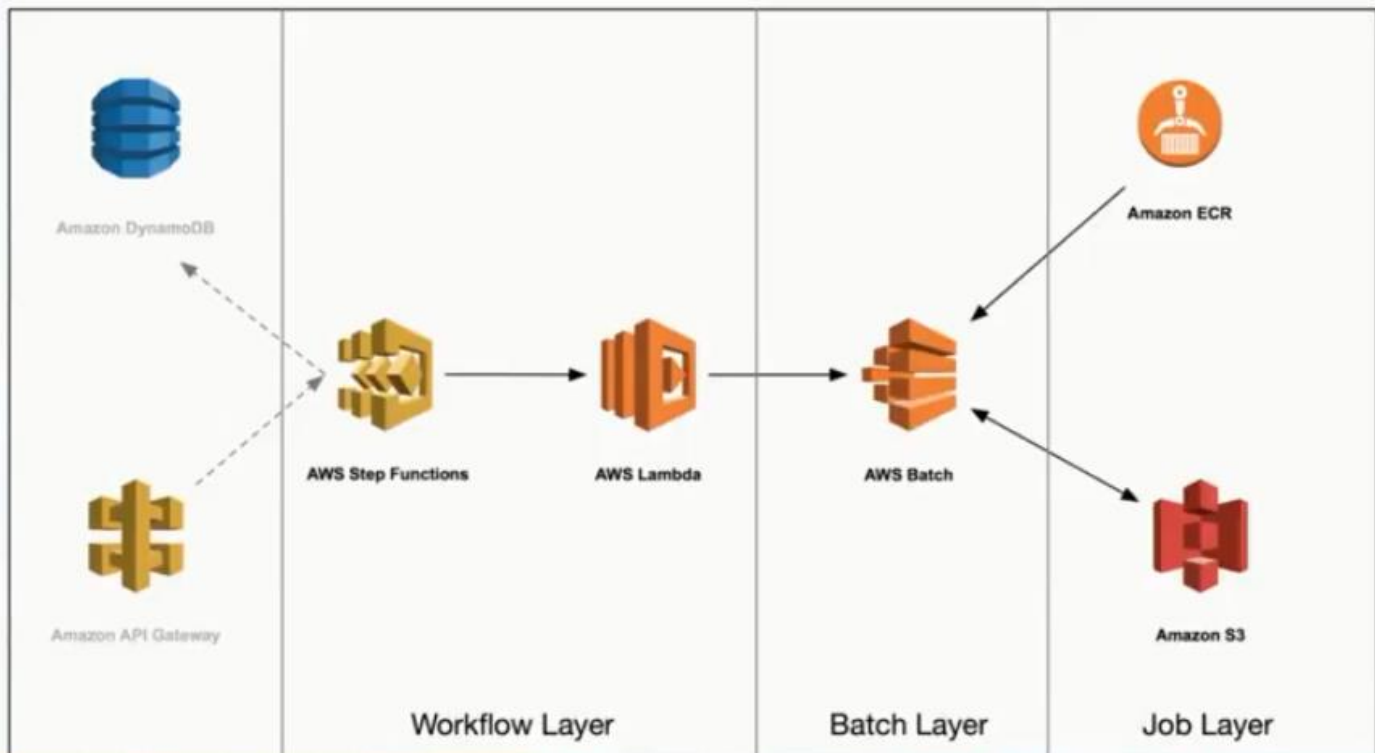
© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

aws

Workflows and Job Dependencies



<https://github.com/aws-labs/aws-batch-genomics>



The above architecture shows how you would use Step Functions to submit Jobs to AWS Batch through a Lambda Function. Using a template that is available in the lambda console, your jobs can be running images stored in ECR, Docker Hub or your private Docker Repository and also interact with services like S3.

New! Array Jobs - Easily run parallel jobs



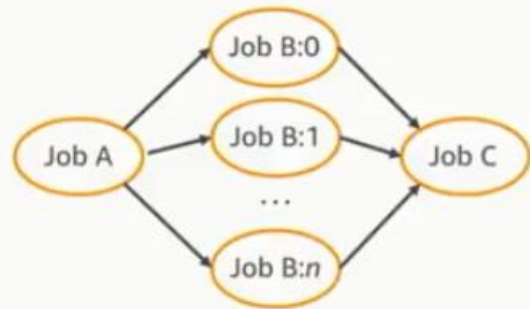
Array jobs allow you to run up to 10,000 copies of an application. AWS Batch creates child jobs for each element in the array.

Array jobs are an efficient way to run:

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

Also includes:

- Extended Dependency Model
- Enhancements to Job APIs



AWS re:Invent

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.



This can be used to run a job that is running the same command against many different objects inside an S3 bucket, you also get an extra environment variable that tells you the index of the job within that array of jobs. You can have dependencies between jobs like saying Job C should only be started once all the elements of Job B have completed successfully.

Submit an AWS Batch Job

Select a job definition to run and apply any command, parameter, or environment variable overrides.

Job run-time

Job name: distributed-pi

Job definition: distributed-pi-digits:1

Job queue: Select a job queue

Job Type: ☐ Single ☒ Array

Array Size: 1000

Job depends on: optional: enter a jobid this will depend on

N-To-N job dependencies: optional: enter a jobid this will depend on

Run children sequentially: ☐

Environment

Command: Space delimited JSON

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

Example Array Job Submission

```
$ aws batch submit-job --job-name BigBatch --job-queue ProdQueue --job-definition monte-carlo:8 --array-properties "size=10000" ...  
{  
  "jobName": "BigBatch",    "jobId": "350f4655-5d61-40f0-aa0b-03ad787db329"  
}
```

Job Name: BigBatch
Job ID: 350f4655-5d61-40f0-aa0b-03ad787db329

Job Name: BigBatch
Job ID: 350f4655-5d61-40f0-aa0b-03ad787db329:0

Job Name: BigBatch
Job ID: 350f4655-5d61-40f0-aa0b-03ad787db329:1

...

Job Name: BigBatch
Job ID: 350f4655-5d61-40f0-aa0b-03ad787db329:9999



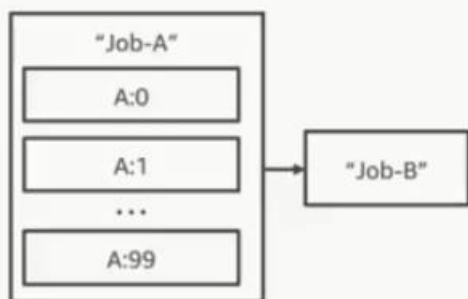
© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

This is the syntax to use, every child job will have the same jobId with an additional colon value for the index of the child within that array of jobs.

Array Job Dependency Models



Job Depends on Array Job



AWS
re:Invent

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

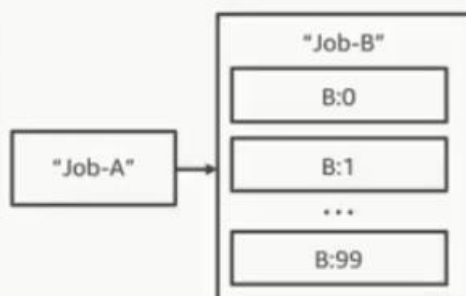
```
$ aws batch submit-job -cli-input-json file://./Job-A.json  
  
<Job-A.json>  
{  
  "jobName": "Job-A",  
  "jobQueue": "ProdQueue",  
  "jobDefinition": "Job-A-Definition:1",  
  "arrayProperties":  
  {  
    "copies": 100  
  }  
}  
  
$ aws batch submit-job -cli-input-json file://./Job-B.json  
  
<Job-B.json>  
{  
  "jobName": "Job-B",  
  "jobQueue": "ProdQueue",  
  "jobDefinition": "Job-B-Definition:1",  
  "dependsOn": [  
    {  
      "jobId": "<job ID for Job A>"  
    }  
  ]  
}
```

This is how you can submit jobs with different dependencies within an array of jobs, Job B will only be run once all the Job A child jobs are completed because of the way we have set up the dependencies as above. If you call **'describe jobs'** on an array of 10,000 jobs, you will be able to see a summary of how many child jobs are completed and what states the whole job array is in at the point in time.



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" }
  ]
}
```

aws
re:Invent

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

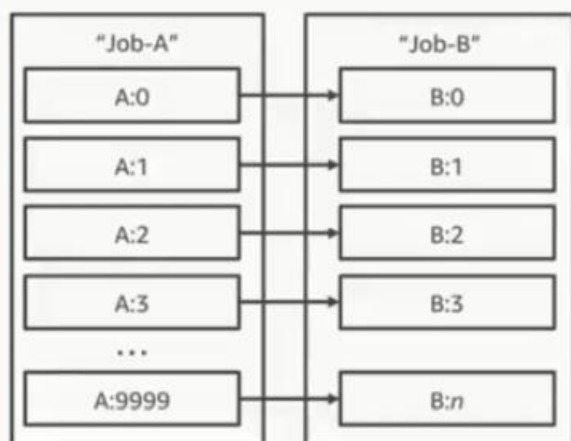


You can also use the '*list jobs*' API to get a particular listing of the jobs or a particular job with their state.



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"
}
```

aws
re:Invent

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

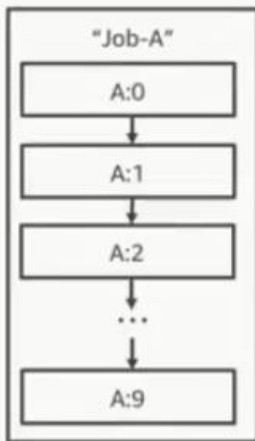


You can also have a **N-to-N dependency model** as above, such as for an image processing pipelines where you need to first validate that the file was not corrupted in transit in the 1st stage, the 2nd stage is to do a rectilinear correction, the 3rd stage is where you want to make image recognition calls or try to do some computer vision analysis of an image. You can easily submit this and process 10,000 images by just making 3 API calls, one call for each stage as above. This means that as element A:42 of job 1 completes, then element B:42 of job 2 will be considered runnable and the whole pipeline can proceed in this way.



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"  
}
```

aws
re:Invent

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

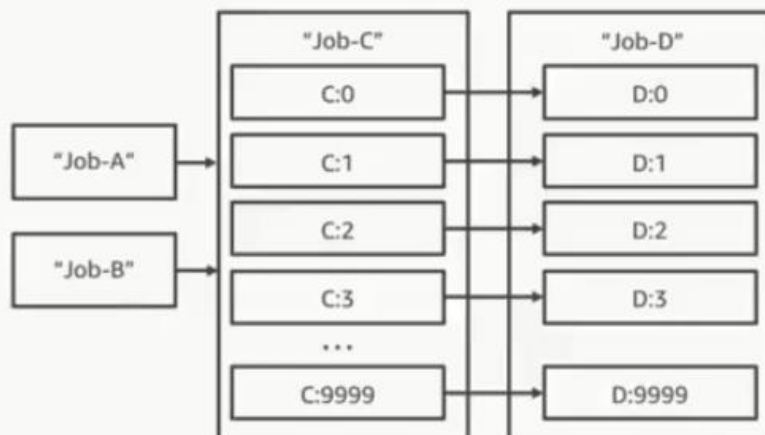


You can also express a sequential relationship between the elements of a single array job as above, the child jobs will then be processed sequentially.



Another Example

C is dependent on A and B
D has an N_TO_N dependency on C



aws
re:Invent

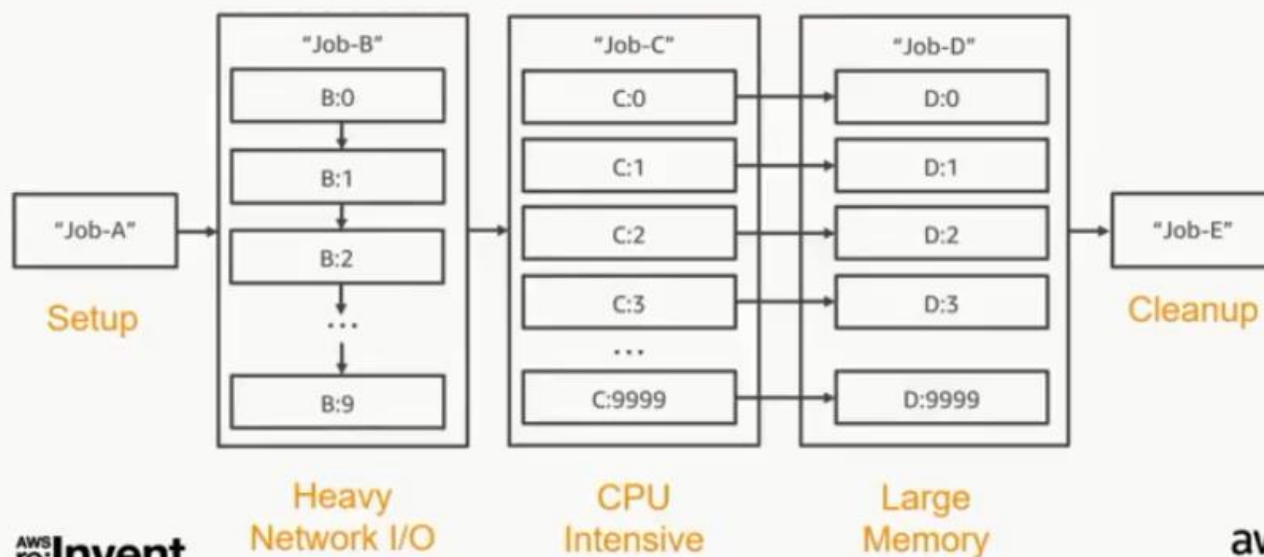
© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.



You can also express dependencies on individual elements of an array of jobs like jobs 3, 7 and 42, only when those are completed can you move on to other jobs in the array.



These Models Can be Combined



aws
re:Invent

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

aws

This becomes really useful when you have pipelines for each stage that might require different instance types, CPU/GPU, FPGA configurations or memory requirements. You can also submit these jobs and the correct instance types will be selected for you, used, and scaled down as needed automatically.

Roadmap



Let us discuss what the roadmap is and what you can expect from us going forward into the future.



What Can You Expect in 2018?

- Significant improvements to the AWS Batch console
- Automatically submit AWS Batch jobs in response to CloudWatch Events
- CloudTrail auditing of AWS Batch API calls
- Consumable resources
- Additional job types
- Further regional expansion

Important Links

Product Details:

<https://aws.amazon.com/batch/details/>

Getting Started:

<https://aws.amazon.com/batch/getting-started/>

Sample code for AWS Batch + Step Functions Integration:

<https://github.com/aws-labs/aws-batch-genomics>

Compute Blog Post Describing How to Use Batch + FPGAs:

<https://aws.amazon.com/blogs/compute/accelerating-precision-medicine-at-scale/>

Deep Learning on AWS Batch:

<https://aws.amazon.com/pt/blogs/compute/deep-learning-on-aws-batch/>

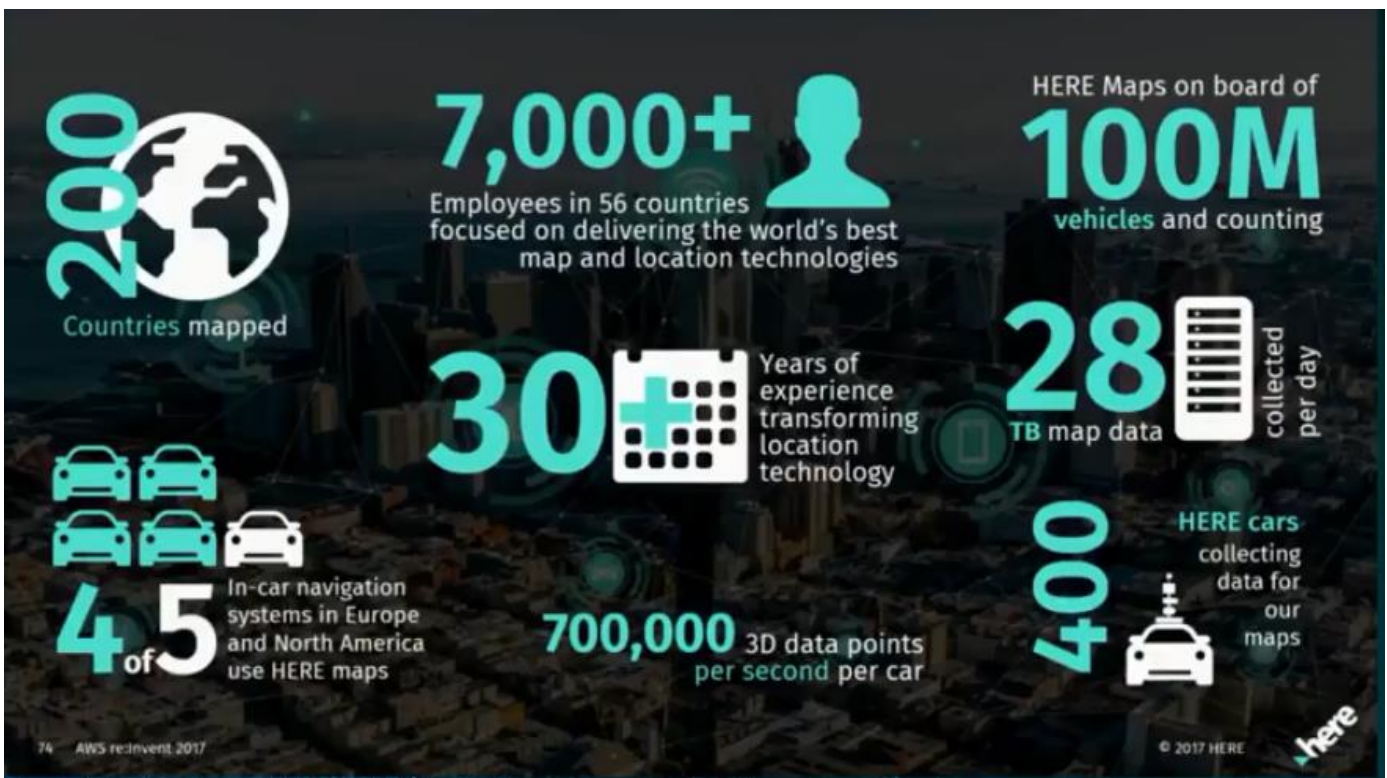
AWS
re:Invent

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.





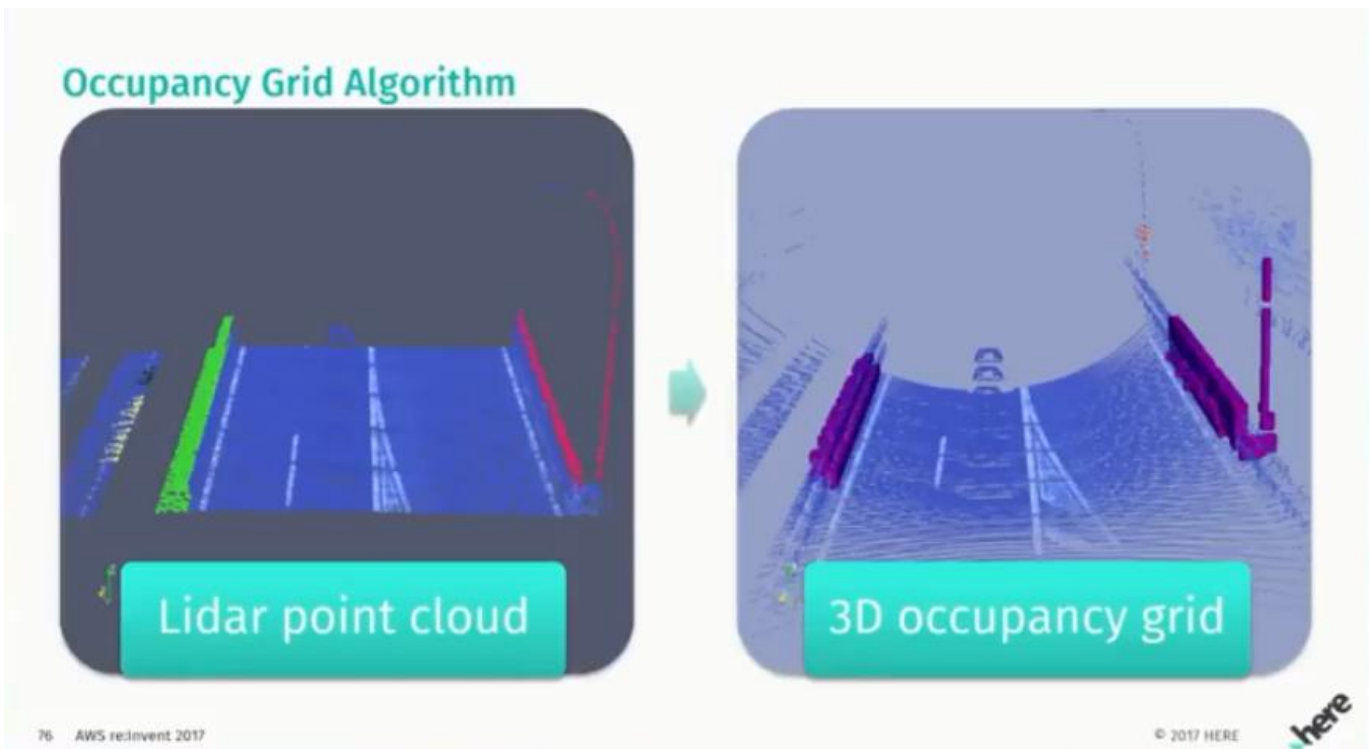
Let us now see how HERE Technologies is using AWS Batch jobs to run their automated driving platform.



To generate our map data, we have over 400 cars driving around the world collecting high resolution imagery, GPS, lidar data around the clock. This generates over 28TB of map data per day.

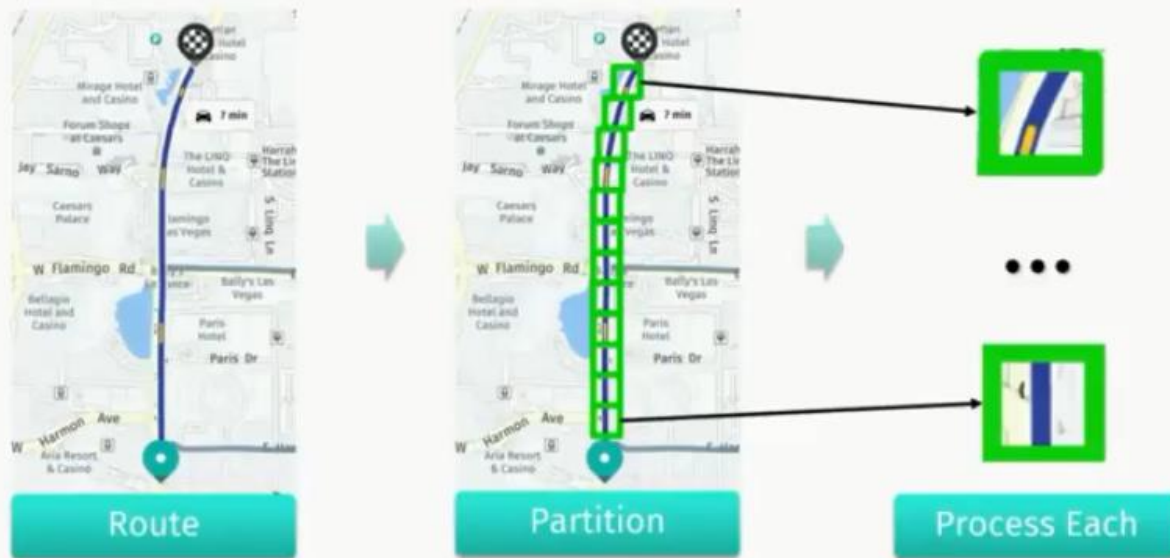


We provide a lot of products to several companies all based around the concept of location.



This is useful for self-driving cars to detect where obstacles are. This algorithm is a microservice offering that we provide to our customers to submit map lidar data for use to process and give them the obstacles within those data as above as a service. We had to create a pipeline for this system as below.

Occupancy Grid Pipeline



77 AWS reInvent 2017

© 2017 HERE



A customer submits a route, the route is then partitioned into equal sizes, and we can then process each of the partitions in parallel workloads.

Requirements & Considered Architectures

1. 1-month deadline
2. Researcher productivity
3. Generic
4. Heterogenous languages
5. Auto Scaling
6. Minimal care and feeding

Deep Learning

- Best for structured data (images), not unstructured (point clouds)
- Researcher decision



Spark

- No native C++ support



ECS

- Manual Auto Scaling tuning
- No instance type optimization/bin packing
- No queue management



Kubernetes

- No team experience

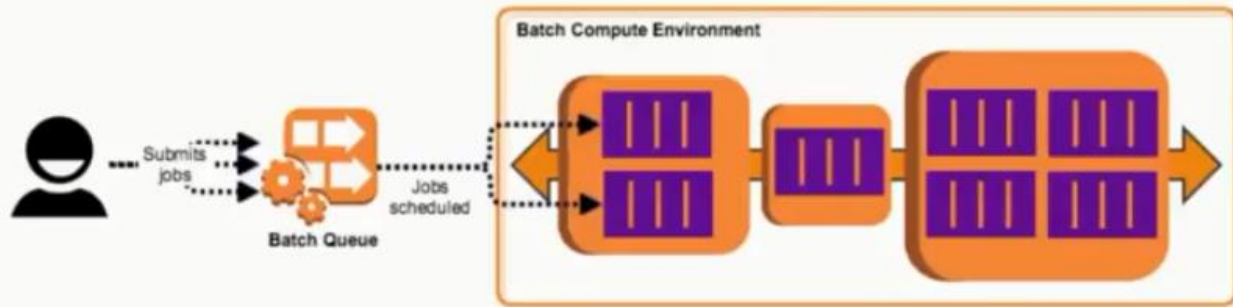


78 AWS reInvent 2017

© 2017 HERE



System Architecture



This is what we came up with using AWS Batch. We use the managed batch compute environment that picks the correct instance type and learns also to get better over time.

Challenge: Orchestration

- Option 1: Batch Job Dependencies
 - Limit of 20

Job run-time

Job name

Job definition

Job queue

Job depends on

We needed a way to orchestrate the fan-in and fan-out of jobs to several thousands at a time.

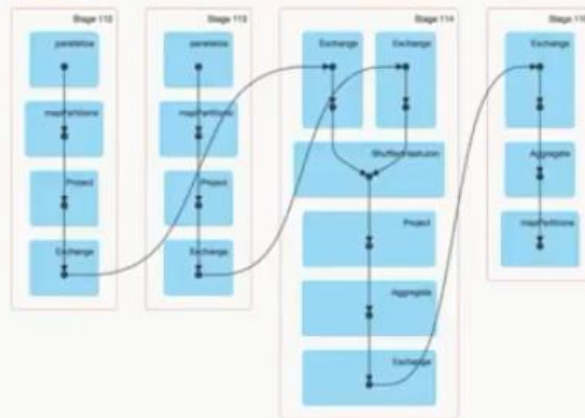
Challenge: Orchestration

- Option 1: Batch Job Dependencies
 - Limit of 20
- Option 2: Spark
 - Overkill for simple orchestration

Details for Job 8

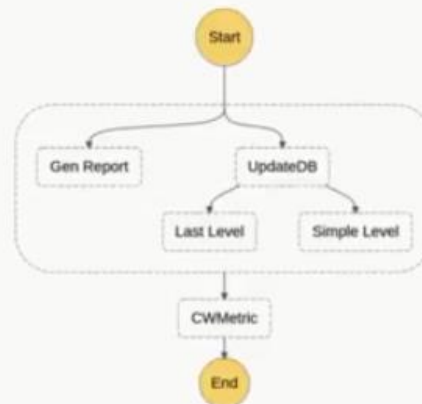
Status: SUCCEEDED
Completed Stages: 4

• Event Timeline
• DAG Visualization



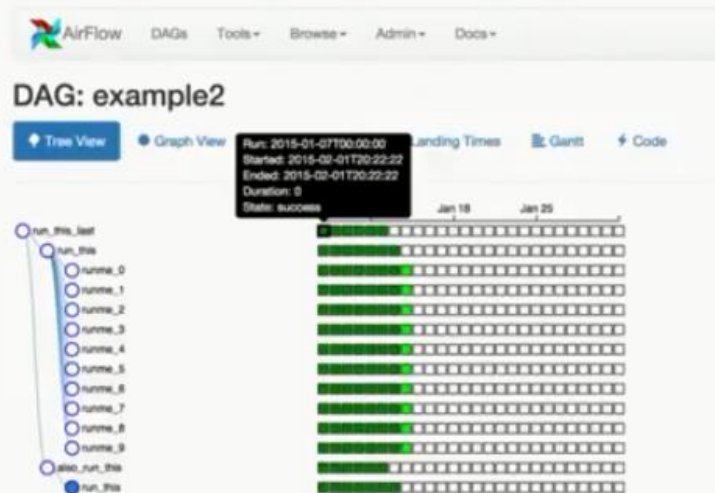
Challenge: Orchestration

- Option 1: Batch Job Dependencies
 - Limit of 20
- Option 2: Spark
 - Overkill for simple orchestration
- Option 3: Step Functions
 - Long tasks = Custom Activities



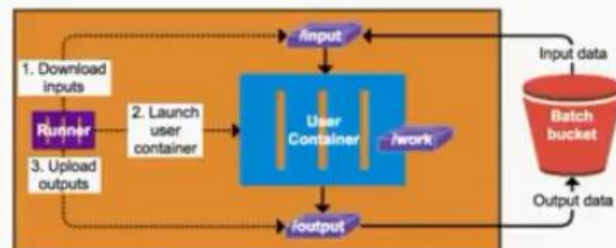
Challenge: Orchestration

- Option 1: Batch Job Dependencies
 - Limit of 20
- Option 2: Spark
 - Overkill for simple orchestration
- Option 3: Step Functions
 - Long tasks = Custom Activities
- Solution: Apache Airflow
 - Winner



Challenge 2: Separating Algorithm & Data Transfer

- Option 1: Require library inclusion
 - Change → many image updates
 - Library per language
- Option 2: Base Docker image
 - Change → many image updates
 - Limits flexibility
 - Risky
- Option 3: EFS
 - Not suited for 100% temporary files
- Solution: "Runner" container



We are using Docker containers that can spin up secondary Docker containers to do the jobs.

Other Challenges

- Storage
 - Initial: Custom AMI w/large EBS volume
 - In-Progress: “Auto Scaling” EBS volumes
- Managed vs. Unmanaged Compute Environment
 - Use Managed unless you absolutely can’t

Next Steps

- Explore array jobs (announced November 28)
- Migrate other projects



Fully Managed



Integrated with AWS



**Cost-optimized
Resource Provisioning**