
AWS
re:Invent

CON304

Amazon ECS on AWS Fargate deep dive with Affirm

Massimo Re Ferre
Principal Technologist
AWS

Greg Sterin
Senior Staff Software Engineer
Affirm

AWS re:Invent

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Agenda

Common batch workloads and requirements

Benefits of running batch on Amazon ECS / AWS Fargate

Building resilient serverless batch jobs

AWS Batch on Fargate (New!)

Optimizing costs with Fargate Spot

How Affirm uses Amazon ECS / Fargate to manage batch jobs



© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Related sessions

CON218: AWS Fargate: Roadmap and vision

CON216: AWS Fargate: Are serverless containers right for you?

CON206: Getting up and running with Amazon ECS

CON212: Securing your Amazon ECS applications: Best practices

CON210: Developing CI/CD pipelines with Amazon ECS and AWS Fargate

CON303: Amazon EKS on AWS Fargate deep dive

Common batch workloads



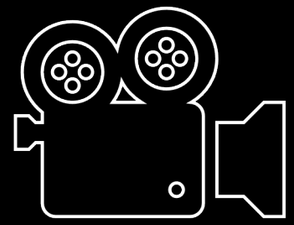
Financial Services

Analyze daily transactions, reporting, market performance



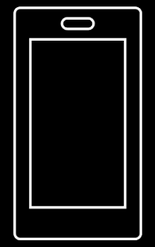
Life Sciences

Genomic sequencing, drug interaction screenings



Digital Media

Image and video transcoding / processing



Telecom Billing

Process call data records to calculate charges

Batch job: *"Jobs that can run without end user interaction, or can be scheduled to run as resources permit"*[1]

[1]https://www.ibm.com/support/knowledgecenter/zosbasics/com.ibm.zos.zconcepts/zconc_whatisbatch.htm

Batch processing requirements

Compute with high memory-to-vCPU ratio

Execution on trigger or schedule

Parallel with managed queues

Monitoring and alerts

Resilient to failures

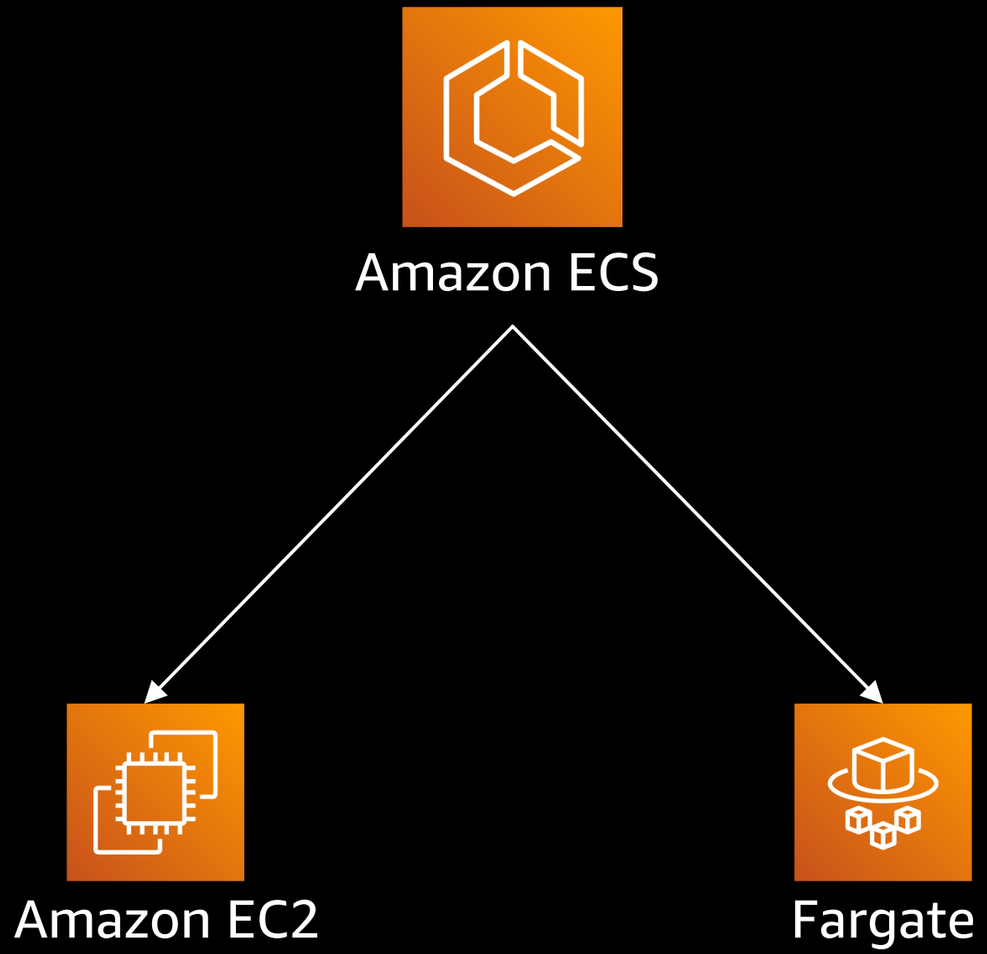
Secure and compliant

Flexible scheduling for lower costs

Amazon ECS and Fargate

Management
Deployment, scheduling, scaling, and management of containerized applications

Hosting
Where the containers run

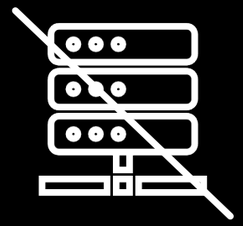


Why run batch workloads on Amazon ECS/Fargate?



Secure by default

Single tenant VM isolation, automatic patching, and encryption



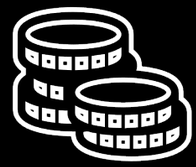
Serverless compute

No EC2 instances to provision, manage, or scale; on-demand, rightsized compute



Compliance

ISO, PCI, SOC, and HIPAA-eligible



Savings

On-demand and Spot pricing



Integrated

Integrates with AWS (e.g., AWS Batch, AWS Step Functions) and open source (e.g., Apache Airflow) workflow managers

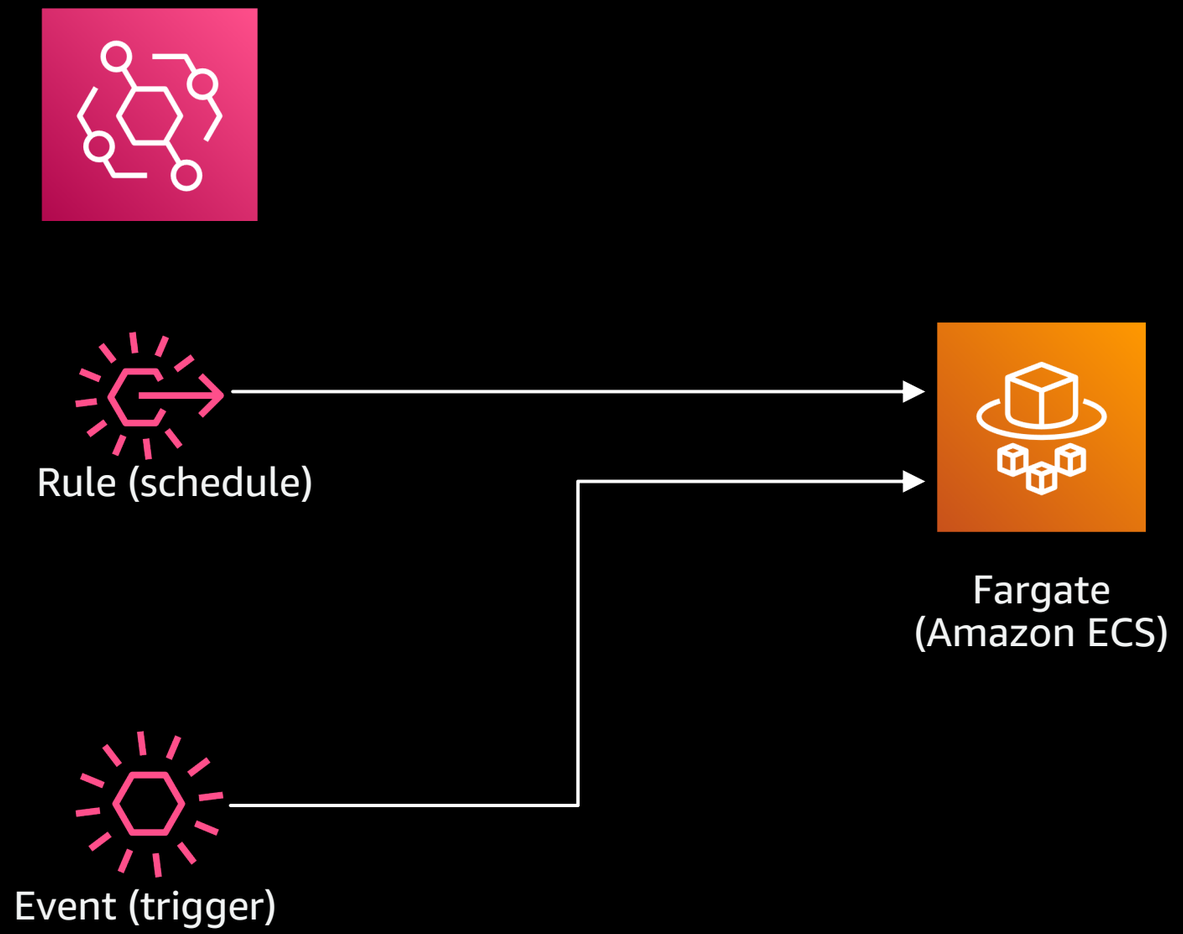
Fargate task CPU memory configurations

CPU	Memory
256 (.25 vCPU)	512MB, 1GB, 2GB
512 (.5 vCPU)	1GB, 2GB, 3GB, 4GB
1,024 (1 vCPU)	2GB, 3GB, 4GB, 5GB, 6GB, 7GB, 8GB
2,048 (2 vCPU)	Between 4GB and 16GB in 1GB increments
4,096 (4 vCPU)	Between 8GB and 30GB in 1GB increments

50 different CPU / memory configurations to choose from with up to 30GB memory
20GB of encrypted ephemeral storage by default

Scheduled or triggered batch jobs

AMAZON EVENTBRIDGE + FARGATE (AMAZON ECS)

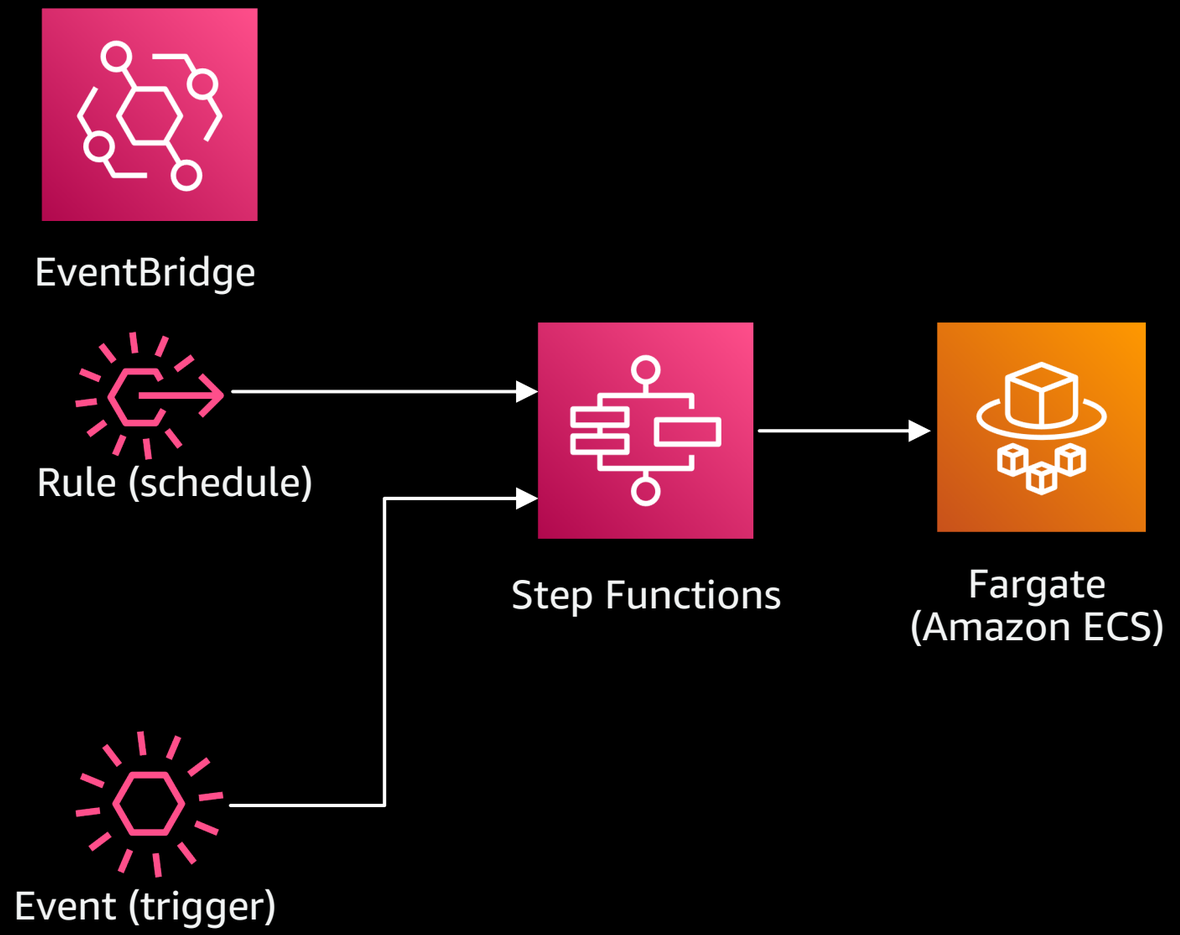


Advantages

- Automated
- Serverless
- Several AWS + partner trigger events supported
- Logging, metrics, and alerts with Amazon CloudWatch

Improve batch job resiliency

EVENTBRIDGE + STEP FUNCTIONS + FARGATE (AMAZON ECS)

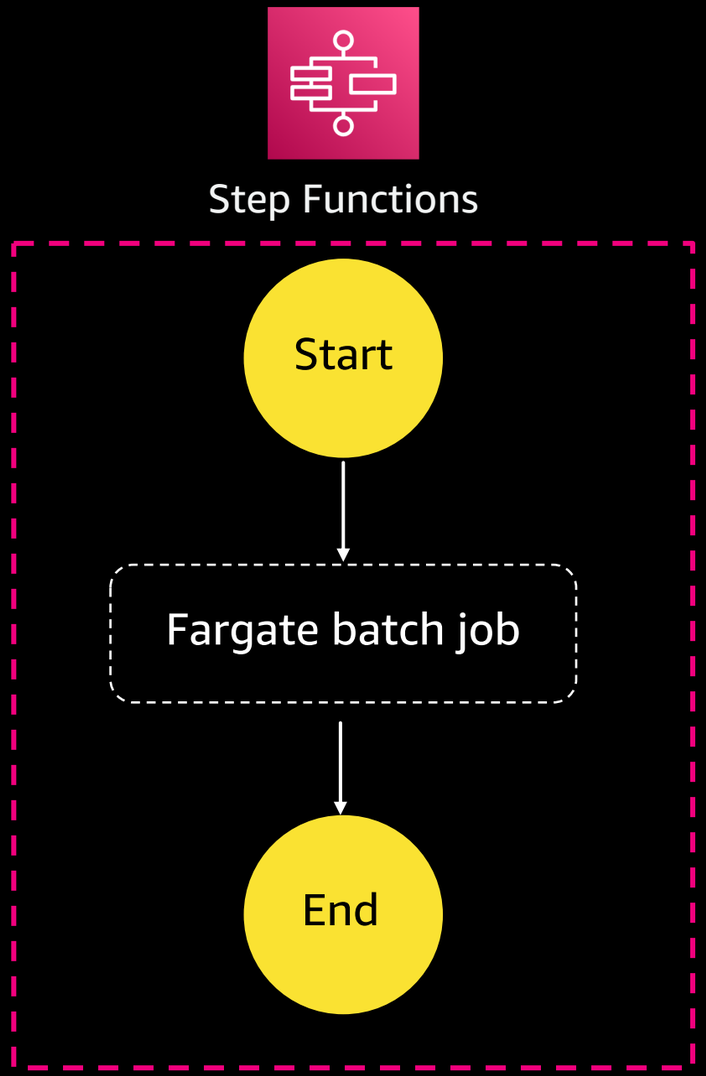


Additional benefits with Step Functions

- Define JSON, visualize in console, monitor execution
- State machine monitors job health
- Failed jobs can retry
- Define timeout
- Add notifications

Retry Fargate tasks

EVENTBRIDGE + STEP FUNCTIONS + FARGATE (AMAZON ECS)




```
"Retry": [
  {
    "ErrorEquals": [
      "States.TaskFailed"
    ],
    "IntervalSeconds": 3,
    "MaxAttempts": 3,
    "BackoffRate": 1.0
  },
  {
    "ErrorEquals": [
      "ECS.AmazonECSException"
    ],
    "IntervalSeconds": 10,
    "MaxAttempts": 3,
    "BackoffRate": 2.0
  }
],
```

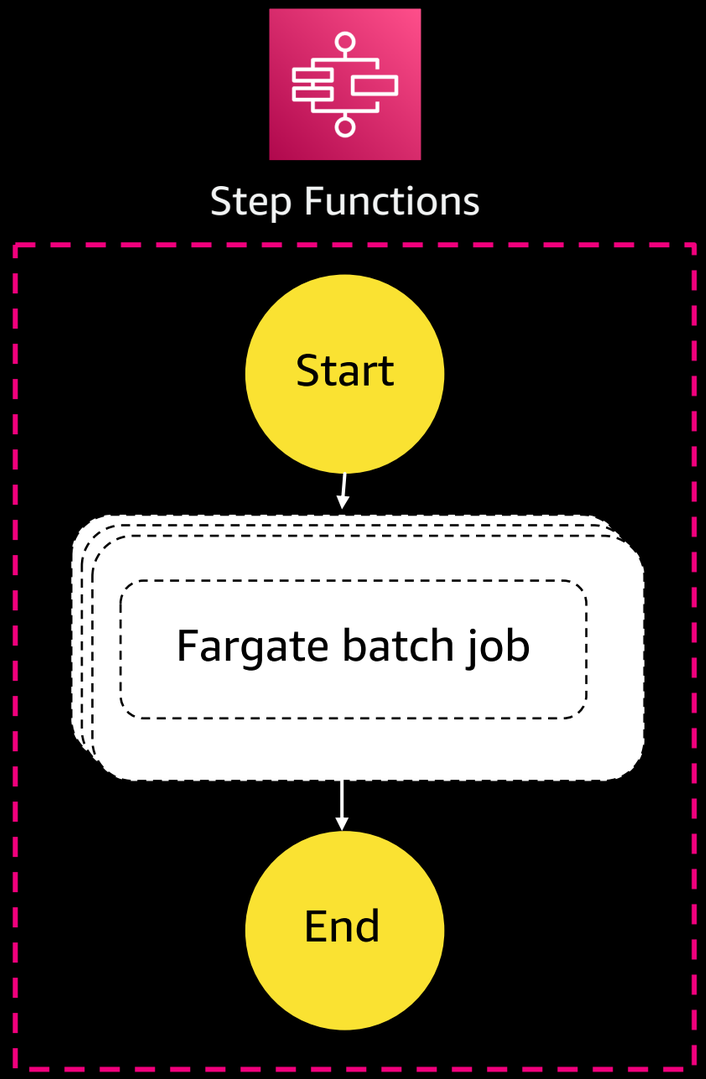
# of seconds before first retry attempt

Max # of retry attempts

Multiplier by which the retry interval increases during each attempt

Parallel batch jobs with resiliency

EVENTBRIDGE + STEP FUNCTIONS + FARGATE (AMAZON ECS)




```
"StartAt": "Map",
"States": {
  "Map": {
    "Type": "Map",
    "ItemsPath": "$.array",
    "ResultPath": "$.array",
    "MaxConcurrency": 0,
    "End": true,
    "Iterator": {
      "StartAt": "Fargate Batch Job",
      "States": {
        "Fargate Batch Job": {
          "Type": "Pass",
          "Result": "Done!",
          "End": true
        }
      }
    }
  }
}
```

"Map" state executes same steps for multiple entries of an array in parallel

Input array

# of parallel invocations of iterator (0 is default and places no quota)

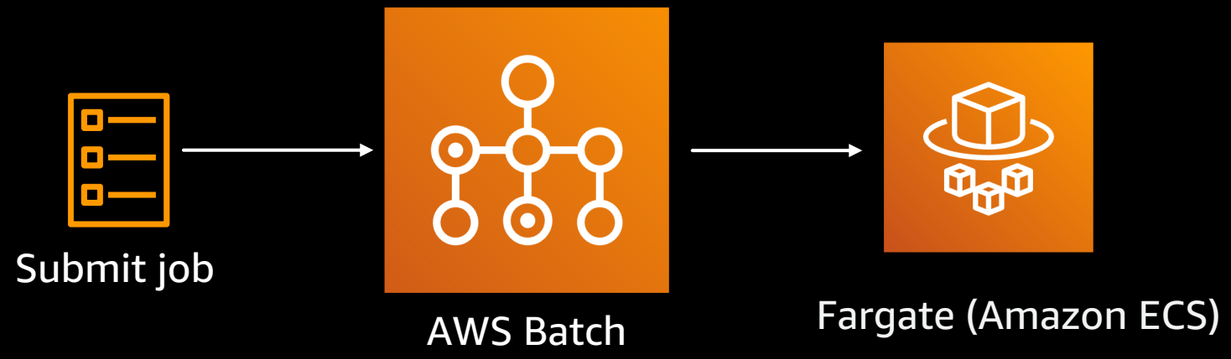
AWS Copilot CLI: Scheduled jobs (New)



- New Amazon ECS command line experience
- Rapid application deployment by automatically provisioning lower-level infrastructure
- Schedule jobs with built-in timeouts and retries

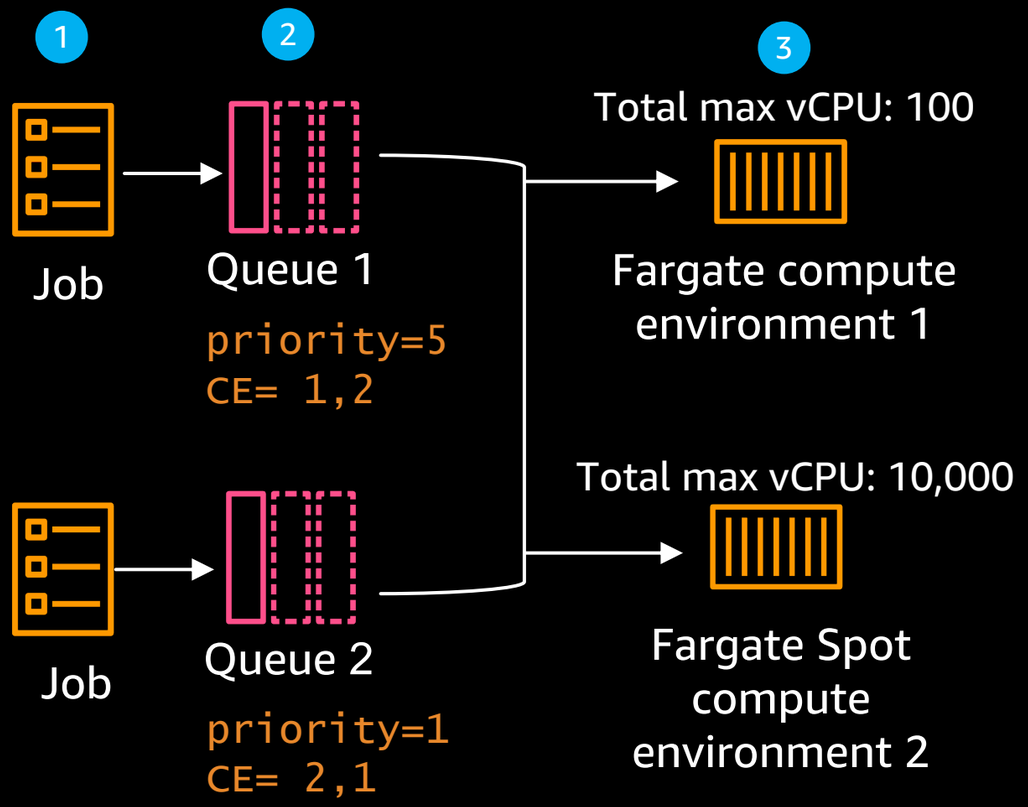
AWS Batch on AWS Fargate (New)

- Serverless batch computing
- AWS Batch provides managed batch queue, with priority, dependencies, retry configurations, and lifecycle management
- Utilize Fargate Spot with on-demand to optimize costs and availability
- Process thousands of jobs in parallel at scale with resiliency



How AWS Batch + Fargate works

1. Submit job to a preassigned queue
2. AWS Batch scheduler submits jobs by priority to Fargate compute environment (CE)
3. Once max CE vCPU limit is reached, jobs spill over to next CE
4. Any failed jobs will retry (if desired)
5. Users can specify dependencies that will be managed by AWS Batch



Optimize cost with Fargate Spot

- Unused capacity
- Prices up to 70% less than on-demand Fargate
- Can be reclaimed (with two-minute warning)
- Automatic diversification

Additional considerations

- AWS limits – VPC, concurrent tasks, task launch rate, task size
- New Amazon Resource Name (ARN) and resource ID format to enable tagging for cost allocation
- Max Step Functions concurrency = 40
- Use AWS Batch + Fargate to process thousands of jobs in parallel

Amazon ECS/Fargate 2020 launches

Storage

- Persistent task storage with Amazon EFS
- 20GB ephemeral storage
- Ephemeral storage encryption by default

Networking

- Dual-stack for IPv4/IPv6

Observability and Ease of Use

- Increased default concurrent task service quota
- Usage metrics against service quotas in CloudWatch

Looking ahead

Compute

- GPU
- Arm processor
- Larger task sizes

Storage

- Extensible ephemeral storage
- Persistent storage with Amazon EBS
- Encryption with customer managed keys

Other

- Amazon ECS exec
- Faster launch times and higher launch rate limit

Affirm: Batch jobs with Apache Airflow and Amazon ECS on Fargate



AWS re:Invent

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.



- **Mission**: “Make honest financial products that improve lives”
- We partner with thousands of online merchants at the point of sale, to provide users with a simple way to pay for an item over time, with no fine print



- Hundreds of jobs, thousands of daily task executions
- Run mission-critical batch pipelines
- Run many highly parallelized jobs
- Availability and reliable execution is critical
- Dynamic capacity – saves money, easier operationally

Batch job framework in a nutshell



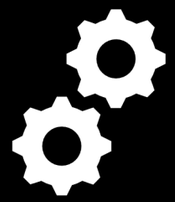
Definition

What to run



Scheduling

When to run it



Execution

Run it



Monitoring and notification

What happened

Batch job framework: Our implementation



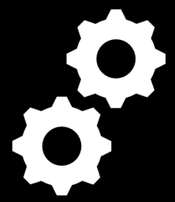
Definition

Airflow DAGs



Scheduling

Airflow scheduler



Execution

Amazon ECS and Fargate



Monitoring and notification

Airflow failure notifications

Airflow SLA alerting

Airflow runtime metrics

Why Amazon ECS on Fargate?

Dynamic

- Pay for what you use
- Spin up capacity on the fly
- Set CPU / memory allocation per job

Containerized

- We were already using Docker images as our core deployment artifact, so this was easy to plug in

No management of underlying infrastructure

- We don’t have to manage instances
- Provisioning and deprovisioning works out of the box

Why Amazon ECS on Fargate?

Dynamic Resource Provisioning

options

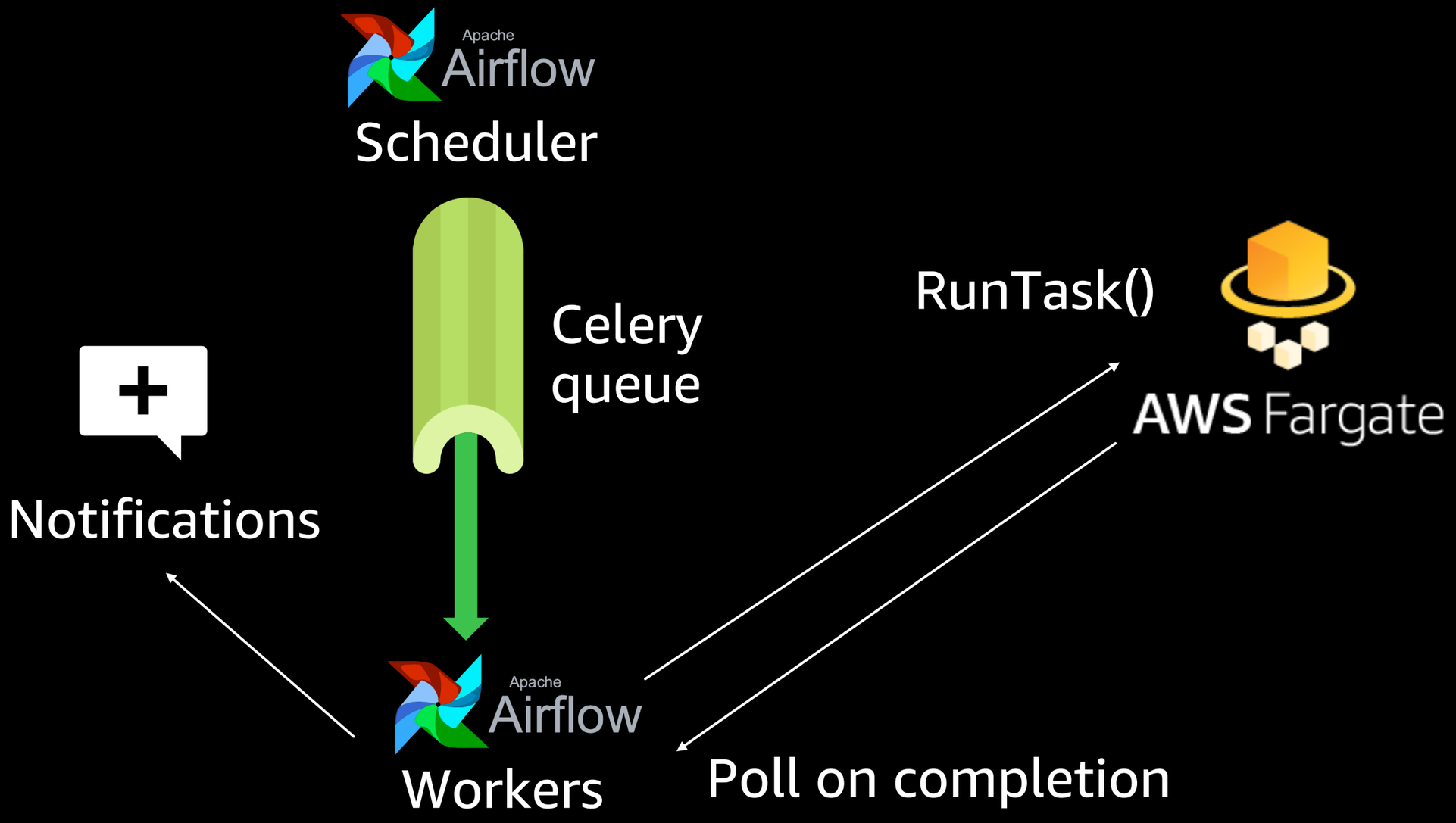
Technology	Pros	Cons
Docker Swarm	• Easy to integrate with existing docker setup • Support for CPU and memory limits via cgroups	• No autoscaling support, would have to roll our own • Designed around concept of always-up services rather than jobs
Kubernetes	• Robust ecosystem • Managed AWS version • Cloud agnostic solution	• Very heavy system - high infra management/maintenance cost
Amazon ECS	• AWS managed container scheduling/allocation • Support for CPU/memory allocation • Integration with AWS autoscaling	• Still have to configure AWS autoscaling rules, which is much trickier for batch jobs than a webserver. • Management of ECS worker nodes still our responsibility
Amazon ECS with Fargate	• Everything Amazon ECS has • No management of any instances, provisioning/deprovisioning works OOTB	• More expensive (per compute hour) than managing our own nodes. • Only available on AWS.

affirm

Why Airflow?

- Job definition, scheduling, and monitoring / notification all in one
- Active community
- Highly extensible

The design


Apache
Airflow
Scheduler
Celery
queue
RunTask()
AWS Fargate
Notifications
Apache
Airflow
Workers
Poll on completion

Registering task definitions: Happens on deploys

```
# Build ECS task definition
{%- for item in pillar.dynamic_jobs.ecs.cpu_memory_combinations %}
ecs-task-definition-fargate-{{ item.suffix }}:
boto_ecs.register_definition:
  - require:
    - test: success-image-exists
  - region: {{ grains.location }}
  # This is actually the name of the task definition
  - family: {{ task_family }}{{ item.suffix }}_fargate # Fargate only supports awsvpc
  - networkMode: awsvpc
  - requiresCompatibilities:
    - FARGATE
  # IAM Role ARN we will grant to the ECS Agent and the Docker Daemon
  - executionRoleArn: '{{ ecs.execution_role }}'
  - taskRoleArn: '{{ecs.task_role }}'
  - cpu: '{{ item.cpu }}'
  - memory: '{{ item.memory }}'
  - containerDefinitions:
      # Runs the main entrypoint for luigi jobs scheduled on fargate
```

AffirmECSOperator: Triggers Amazon ECS tasks

```
dag = DAG('example-task',
          default_args=default_args,
          schedule_interval='15 18 * * *',
          catchup=False)
task0 = AffirmECSOperator(task_definition='release-v1.2.3',
  cluster='dynamic-jobs',
  overrides={
    'containerOverrides': [{
      'name': 'affirm-docker',
      'command': ["MyTask", "--start-date", "{{
execution_date.strftime('%Y-%m-%d') }}"],
      'cpu': int(1024),
      'memory': int(4096),
      'environment': [
          {
              'name': 'JOB_IMPORTANCE',
              'value': 'normal'
```

Key benefits

- Less capacity we have to manage ourselves
- Less operational complexity
- Cost savings
- Security
- Airflow UI is great for operational tasks for developers

Things to be aware of

- Fargate limits
- We maintain a fleet of Amazon EC2 nodes running Amazon ECS agent to workaround
- Keep an eye on Fargate concurrent task limits
- Fargate uses a heterogenous warm pool of instances behind the scenes so performance may not be consistent
- Amazon ECS task history is not retained very long at all (<1 hour). Airflow helps us capture task histories. We link to the Amazon CloudWatch Logs from our Airflow logs to be able to discover logs for older task runs when they are no longer in the Amazon ECS console.

Want to learn more?

- https://aws.amazon.com/blogs/containers/how-affirm-uses-aws-fargate-and-apache-airflow-to-manage-batch-jobs/

Thank you!



AWS re:Invent

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.



Please complete the session survey

AWS re:Invent

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.