

CON404

# AWS re:INVENT

## Deep Dive into Container Scheduling with Amazon ECS

Anthony Suarez—GM, Amazon ECS & ECR

Shubha Rao—Senior Product Manager, Amazon ECS

November 29, 2017

AWS  
re:INVENT

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



As your application's infrastructure grows and scales, well-managed container scheduling is critical to ensuring high-availability and resource optimization. In this session, we will deep dive into the challenges and opportunities around container scheduling, as well as the different tools available within Amazon ECS and AWS to carry out efficient container scheduling. We will discuss patterns for container scheduling available with Amazon ECS and the Blox scheduling framework

## Getting started with containers is easy!

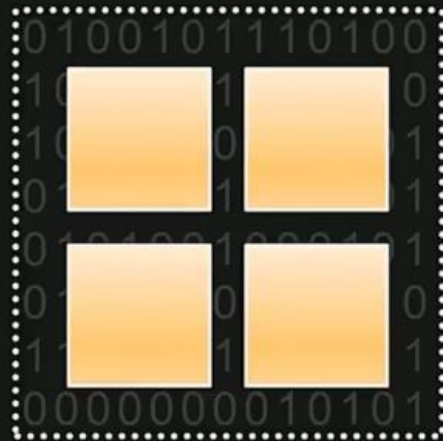


AWS  
re:INVENT

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



# From one container...

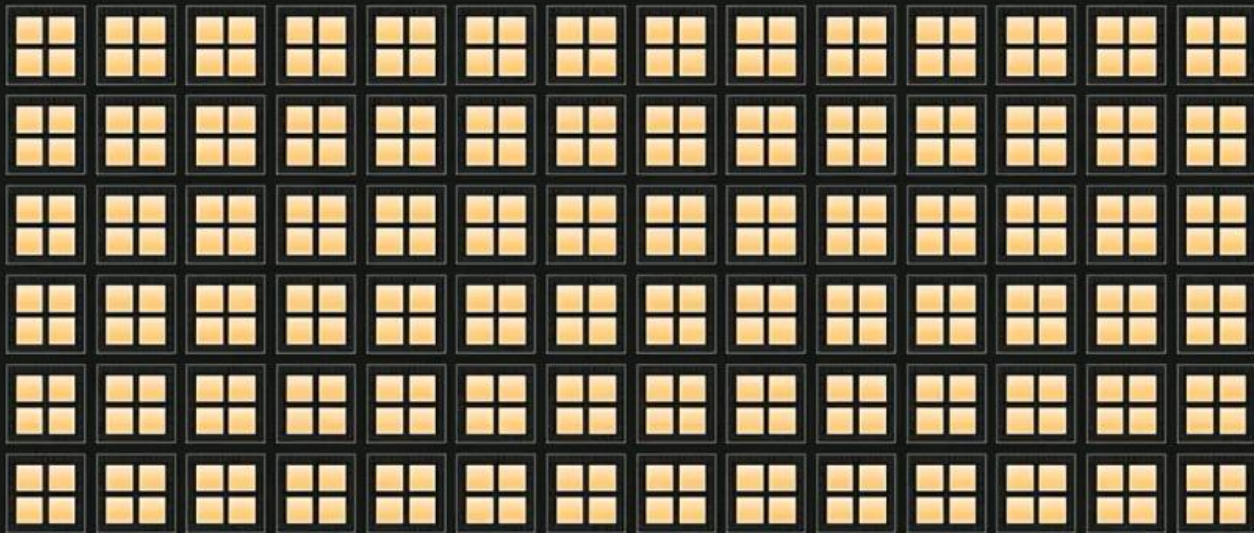


**AWS**  
**re:Invent**

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



# ...to hundreds...

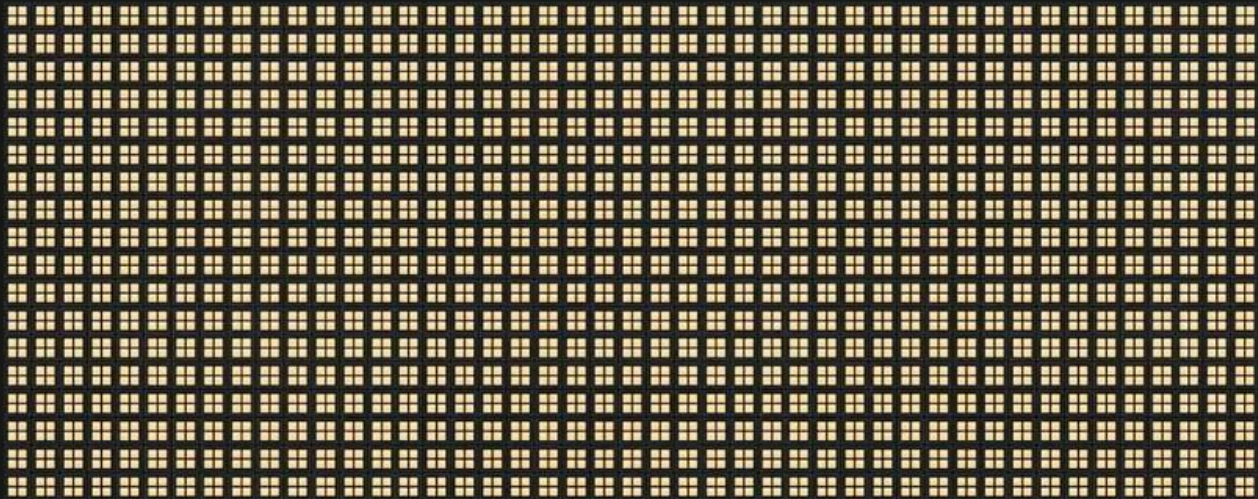


**AWS**  
**re:Invent**

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



...to thousands...

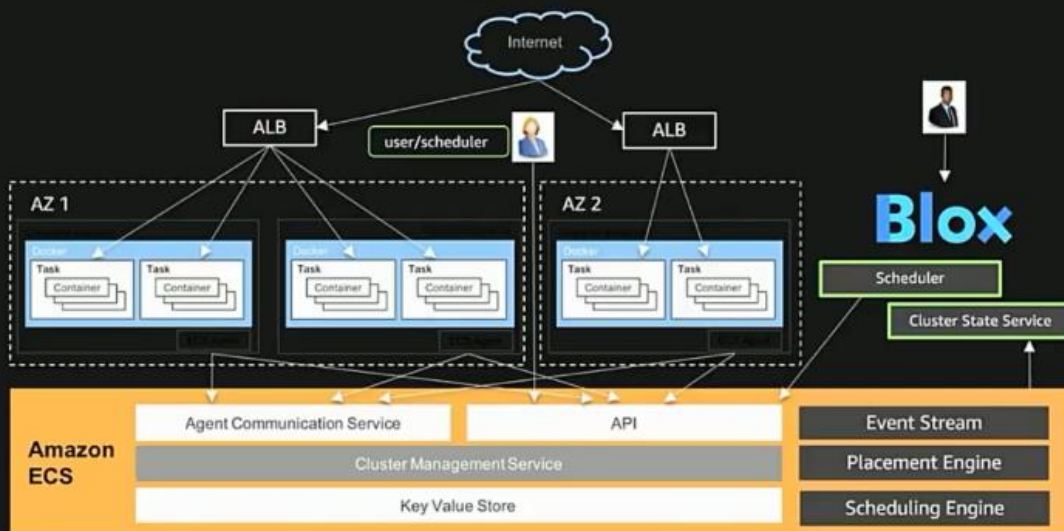


AWS  
re:Invent

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

aws

## Amazon ECS manages clusters at scale!



AWS  
re:Invent

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

aws



# Supporting massive scale



**450+%**  
growth



**Hundreds of millions**  
of containers started each week  
**Millions**  
of container instances

**AWS**  
**re:Invent**

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



## Core components



Scheduling engine



Placement engine



Extensions



## Scheduling engines

A task is simply a logical group of containers that need to be deployed together, you define tasks in a task definition file.

# Schedulers



Services



Batch



Events



Daemon

Batch workloads can be run using a Batch Scheduler, it has integration with things like Spot Fleet, long running, short running jobs. AWS Batch gives you a UI for running this job tasks based on some requirements you have. The Events Scheduler can run when you want like daily, etc. Where you can do this based on an event using CloudWatch events. The Daemon Scheduler helps make sure that some specific task is running on each node within your cluster everytime, it is good for things like logging. Work is continuing on ***the Daemon scheduler under the OSS scheduling framework Blox.***

## Open-source schedulers?



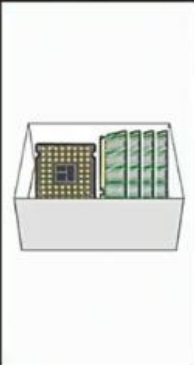
- An open-source project for container scheduling
- These OSS schedulers will power Amazon ECS
- Actively working on a daemon scheduler
- Follow along, comment, or contribute!

<http://blox.github.io>  
<https://github.com/blox>



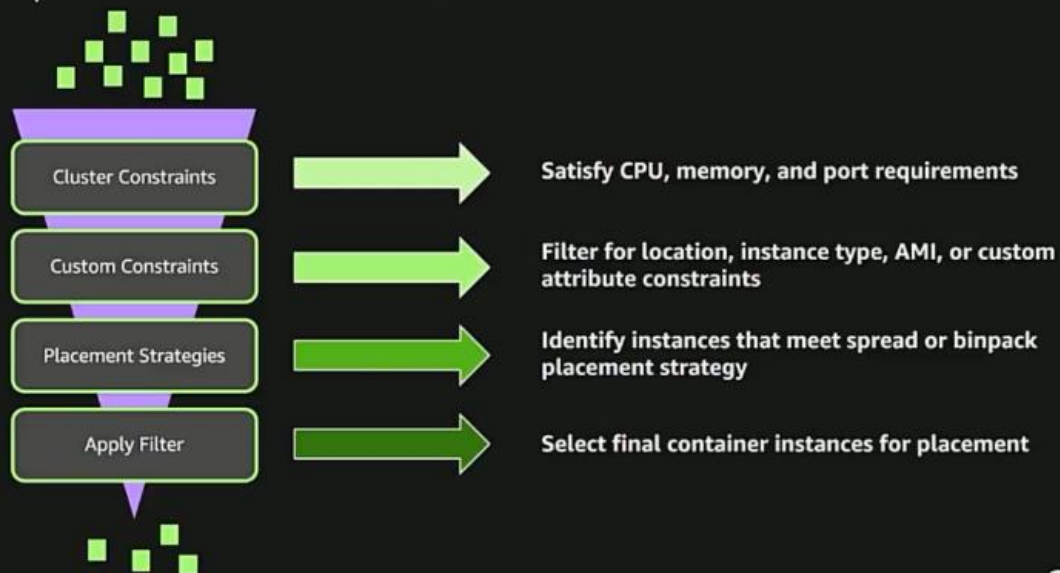
Placement engine

# Task placement engine

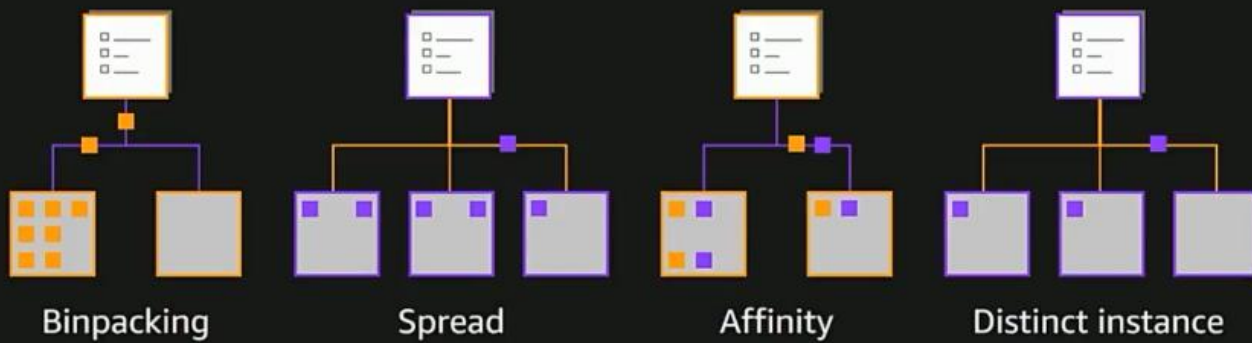


Name	Example
✓ AMI ID	<code>attribute:ecs.ami-id == ami-eca289fb</code>
✓ Availability Zone	<code>attribute:ecs.availability-zone == us-east-1a</code>
✓ Instance type	<code>attribute:ecs.instance-type == t2.small</code>
✓ Distinct instances	<code>type="distinctInstance"</code>
✓ Custom	<code>attribute:stack == prod</code>

## Task placement selection

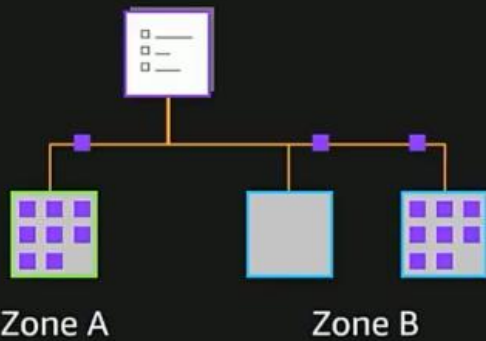


## Supported placement strategies



These are the default strategies offered with ECS today

## Task placement chaining



Spread tasks across  
zones *and* binpack  
within each zone

# Task placement: Easy to get started

## Task Placement

Determines how the individual tasks are scheduled to instances within your cluster. Different placement strategies are available to optimize for availability and efficiency.

Placement Templates

AZ Balanced Spread

Edit

AZ Balanced Spread

AZ Balanced BinPack

BinPack

One Task Per Host

Custom

Optional configuration

Elastic load balancing

Service Auto Scaling

# Task placement: Easy to get started

## Task Placement

Determines how the individual tasks are scheduled to instances within your cluster. Different placement strategies are available to optimize for availability and efficiency.

Placement Templates

Custom

Edit

This template will balance tasks across AZs and within the AZ spread across instances. [Learn more.](#)

### Strategy

You can use multiple placement strategies together and the order is used to find the best match. If there is a tie with the first strategy the next is used to tiebreaker and this continues down the strategies until match is found.

	Order	Type	Field	
⋮	1	Spread	attribute:ecs:availability-zone	⊗
⋮	2	Spread	instanceid	⊗
⋮	3	BinPack	Memory	⊗

[Add strategy](#)

### Constraint

Constraints allow you to filter the instances used for your placement strategies. The scheduler first applies any constraints to filter down to only instances that match the constraints and then the placement strategy is evaluated to find a match for the task to be placed.

Type	Expression	
MemberOf	attribute:ecs:instance-type == g2.*	⊗

[Add constraint](#)

AWS  
re:Invent

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

aws



# Example: Spread placement

```
aws ecs run-task --cluster ecs-demo --task-definition myapp --count 9 --placement-strategy  
type="spread",field="attribute:ecs.availability-zone"
```



# Example: Spread & binpack

```
aws ecs run-task --cluster ecs-demo --task-definition myapp --count 9 --placement-strategy  
type="spread",field="attribute:ecs.availability-zone" type="binpack",field="memory"
```



# Example: Multiple services & strategies



```
aws ecs create-service --service-name svc-binpk --cluster ecs-demo --task-definition myapp binpk
--desired-count 5 --placement-strategy type="binpack",field="memory"
```



```
aws ecs create-service --service-name svc-spread --cluster ecs-demo --task-definition myapp-spread
--desired-count 6 --placement-strategy type="spread",field="attribute:ecs:availability-zone"
```



## Extensions

## Cluster query language

```
aws ecs list-container-instances --cluster ecs-demo --filter "attribute:ecs:instance-type matches t2.*"
{
  "containerInstanceArns": [
    "arn:aws:ecs:us-east-1:123456789000:container-instance/3ced5d42-537c-40b4-9551-b9022cc13b78",
    "arn:aws:ecs:us-east-1:123456789000:container-instance/442d988b-4b00-40bf-85ae-34e0819454f2",
    "arn:aws:ecs:us-east-1:123456789000:container-instance/5dellede-6c22-411e-a469-8301eeebae0f",
    "arn:aws:ecs:us-east-1:123456789000:container-instance/6bfb12c8-1c3c-4d4a-976c-ce3c2c79b031",
    "arn:aws:ecs:us-east-1:123456789000:container-instance/7eb87781-abab-4a6a-9a0d-602a4da59549",
    "arn:aws:ecs:us-east-1:123456789000:container-instance/af8d48ba-73c4-409a-b40f-66596aa86c5d",
    "arn:aws:ecs:us-east-1:123456789000:container-instance/b5c08e3e-bd25-4ec9-9a88-celd53640542",
  ]
}
```

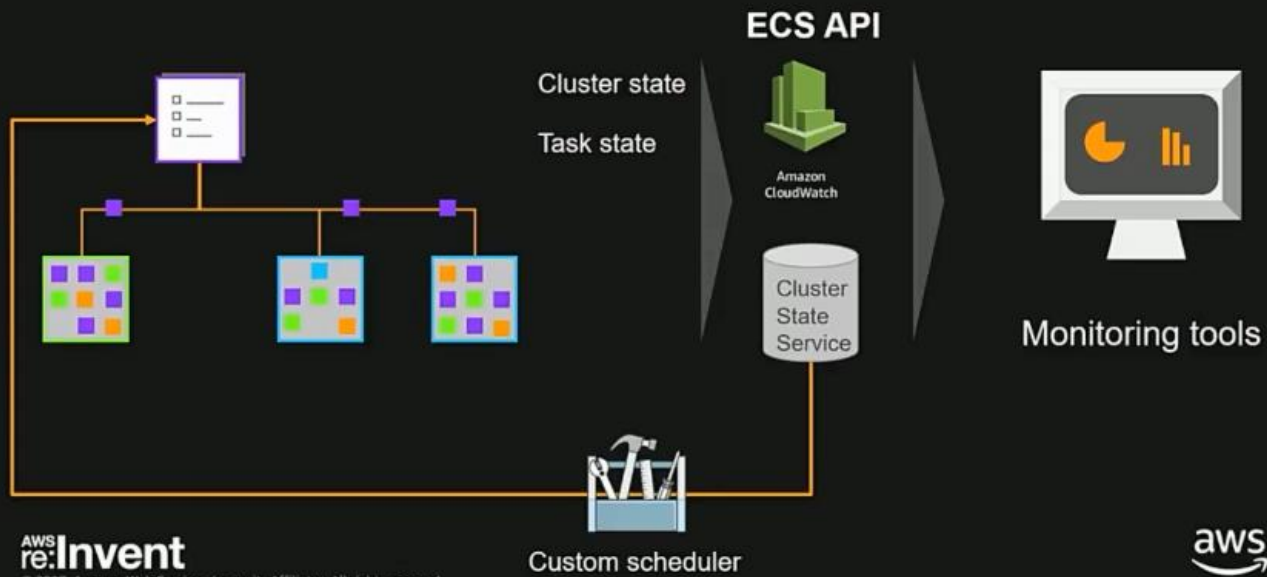
```
aws ecs list-container-instances --cluster ecs-demo --filter "attribute:ecs:instance-type matches t2.*"
{
  "containerInstanceArns": [
    "arn:aws:ecs:us-east-1:123456789000:container-instance/3ced5d42-537c-40b4-9551-b9022cc13b78",
    "arn:aws:ecs:us-east-1:123456789000:container-instance/442d988b-4b00-40bf-85ae-34e0819454f2",
    "arn:aws:ecs:us-east-1:123456789000:container-instance/5dellede-6c22-411e-a469-8301eeebae0f",
    "arn:aws:ecs:us-east-1:123456789000:container-instance/6bfb12c8-1c3c-4d4a-976c-ce3c2c79b031",
    "arn:aws:ecs:us-east-1:123456789000:container-instance/7eb87781-abab-4a6a-9a0d-602a4da59549",
    "arn:aws:ecs:us-east-1:123456789000:container-instance/af8d48ba-73c4-409a-b40f-66596aa86c5d",
    "arn:aws:ecs:us-east-1:123456789000:container-instance/b5c08e3e-bd25-4ec9-9a88-celd53640542",
  ]
}
```

```
aws ecs associate-attributes --cluster ecs-demo --target-id 3ced5d42-537c-40b4-9551-b9022cc13b78
--target-type container-instance --attribute name=stack, value=prod
```

```
aws ecs list-container-instances --cluster ecs-demo --filter "attribute:stack != prod"
{
  "containerInstanceArns": [
    "arn:aws:ecs:us-east-1:123456789000:container-instance/442d988b-4b00-40bf-85ae-34e0819454f2",
    "arn:aws:ecs:us-east-1:123456789000:container-instance/5dellede-6c22-411e-a469-8301eeebae0f",
    "arn:aws:ecs:us-east-1:123456789000:container-instance/6bfb12c8-1c3c-4d4a-976c-ce3c2c79b031",
    "arn:aws:ecs:us-east-1:123456789000:container-instance/af8d48ba-73c4-409a-b40f-66596aa86c5d",
  ]
}
```

We have added a cluster query language that lets you use the API to make calls to the cluster for listing container instances, while doing things like passing in certain constraints through the cluster query language. You can then use the startTask API to target a specific container instance ID.

## ECS event stream



We have also built an event stream for reactive use-cases for thing like a state within our cluster changed. The events stream allows you to filter for events and display real-time dashboards in a push-based model.

## Placement of containers



## ECS placement constraints example

Matt Callanan  
Engineering Manager/Tech Lead  
"Cloud Acceleration Team"  
Expedia  
Brisbane, Australia

mcallanan@expedia.com  
linkedin.com/in/matthewcallanan  
@mcallana



## Avoid relocating tasks to instances about to be drained during cluster update

### Problem:

- Tasks can get rescheduled to another old instance in the Auto Scaling group that is about to be replaced—so tasks can get bumped from instance to instance until all instances are replaced

### Solution:

- Deploy services with a placement constraint on the task definition

```
placement_constraints = [ {  
    type: 'memberOf',  
    expression: 'attribute:state != pre-drain'  
}]
```

- This means that a service won't be placed on an instance that has an attribute named "state" with value "pre-drain"
- At cluster replacement time, we will stand up all new clusters, place old clusters into the "pre-drain" state, and terminate the old instances in batches
- Relocated tasks will only be placed on new instances, avoiding the default "thundering herd" scenario

**AWS**  
**re:Invent**

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



We don't want to deploy apps to EC2 instances in the pre-drain state



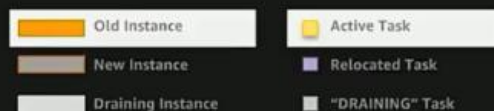
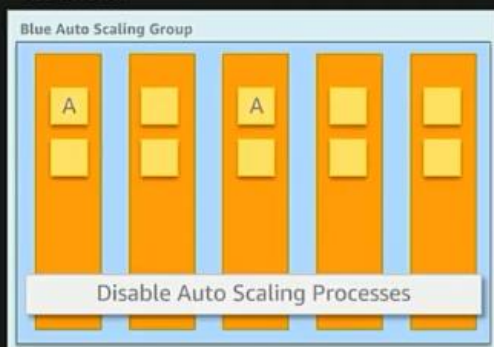
# Cluster update—Phase 1: Expand

Blue CFN Stack

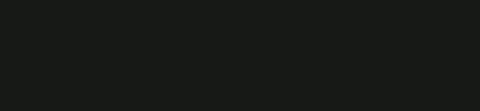
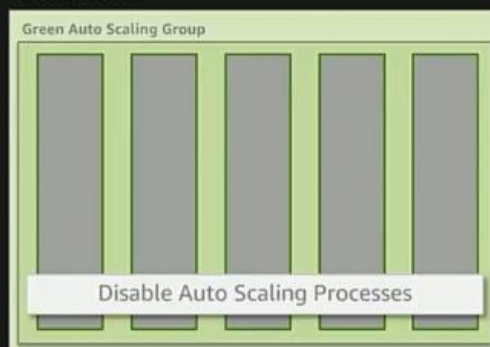


# Cluster update—Phase 1: Expand

Blue CFN Stack



Green CFN Stack

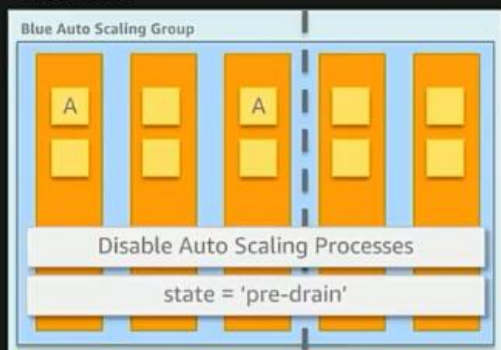




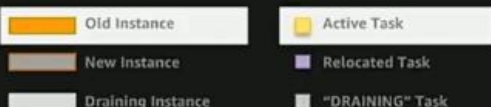
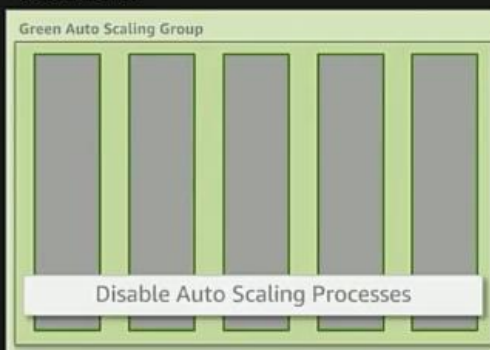
# Cluster update—Phase 2: Relocate tasks

(Batches of 3 instances)

Blue CFN Stack



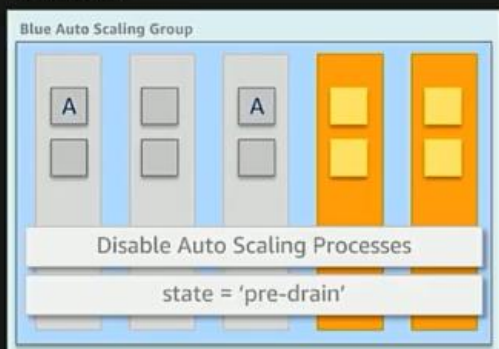
Green CFN Stack



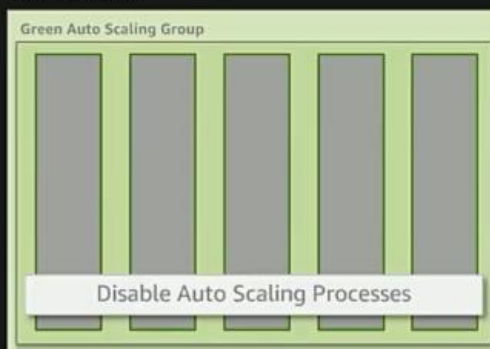
# Cluster update—Phase 2: Relocate tasks

(Batches of 3 instances)

Blue CFN Stack



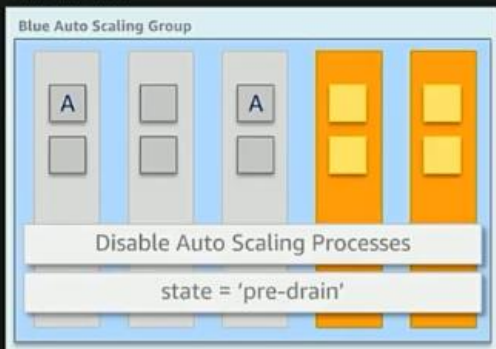
Green CFN Stack



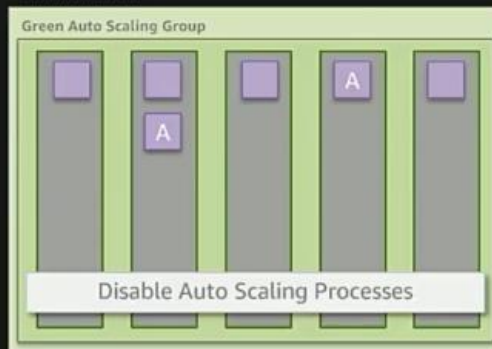
# Cluster update—Phase 2: Relocate tasks

(Batches of 3 instances)

Blue CFN Stack



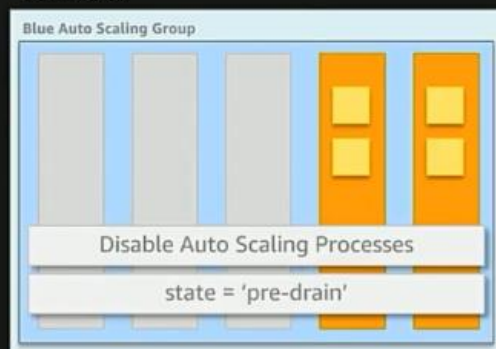
Green CFN Stack



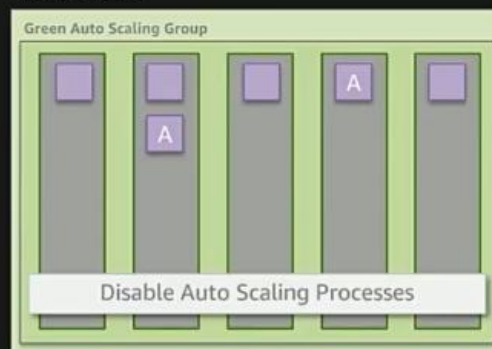
# Cluster update—Phase 2: Relocate tasks

(Batches of 3 instances)

Blue CFN Stack



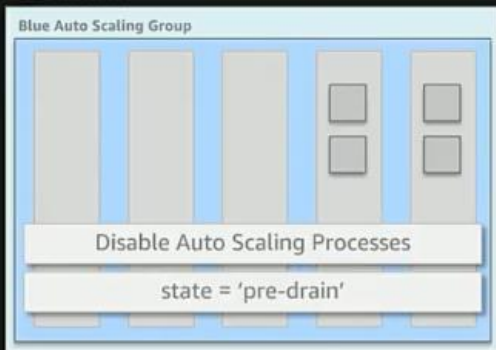
Green CFN Stack



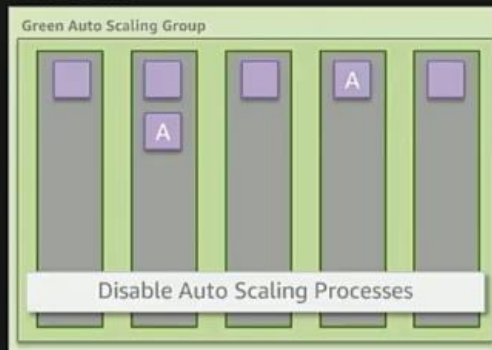
# Cluster update—Phase 2: Relocate tasks

(Batches of 3 instances)

Blue CFN Stack



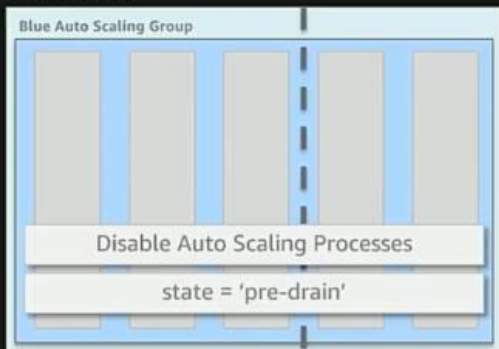
Green CFN Stack



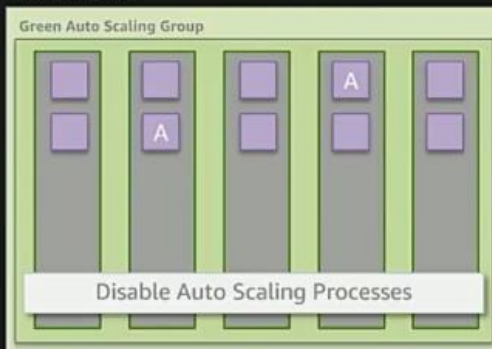
# Cluster update—Phase 2: Relocate tasks

(Batches of 3 instances)

Blue CFN Stack

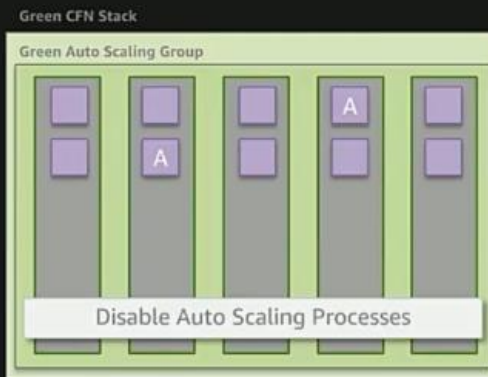


Green CFN Stack



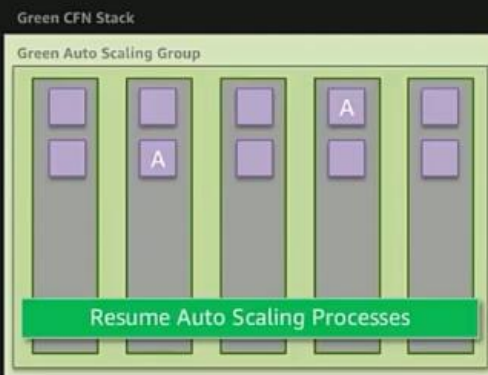
# Cluster update—Phase 2: Relocate tasks

(Batches of 3 instances)



# Cluster update—Phase 2: Relocate tasks

(Batches of 3 instances)



## Task definition placement constraint

### Constraint

Constraints allow you to filter the instances used for your placement strategies using built-in or custom attributes. The scheduler first filters the instances that match the constraints and then applies the placement strategy to place the task.

#### Type

memberOf

#### Expression

attribute:state != exists or attribute:state != pre-drain

+ Add constraint

This is what it looks like in the UI even though we use automation

# Task definition placement constraint

**Constraint**

Constraints allow you to filter the instances used for your placement strategies using built-in or custom attributes. The scheduler first filters the instances that match the constraints and then applies the placement strategy to place the task.

Type	Expression
memberOf	<code>attribute:state !exists or attribute:state != pre-drain</code>

[Add constraint](#)

# Task definition placement constraint

**Constraint**

Constraints allow you to filter the instances used for your placement strategies using built-in or custom attributes. The scheduler first filters the instances that match the constraints and then applies the placement strategy to place the task.

Type	Expression
memberOf	<code>attribute:state !exists or attribute:state != pre-drain</code>

[Add constraint](#)

- "state" is a custom ECS Instance attribute
- By default, the "state" attribute doesn't exist on instances
- Set only to "pre-drain" during cluster update
- Prevents ECS scheduling tasks on instances that are about to be drained
- Removed from instance only in case of rollback of cluster update

## Scheduling patterns

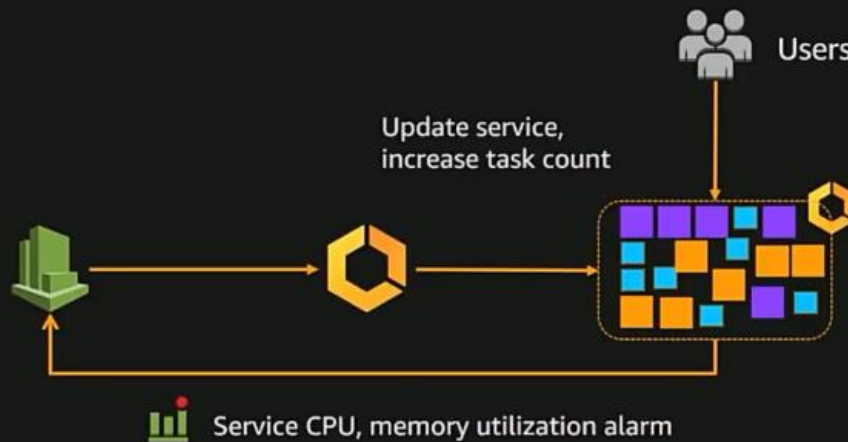
### Scheduling based on...

1. Service metrics
2. Periodic
3. Triggers
4. Dependencies
5. Priorities

Let us see an example of using some of the scheduling primitives described earlier



# Scheduling based on service metrics

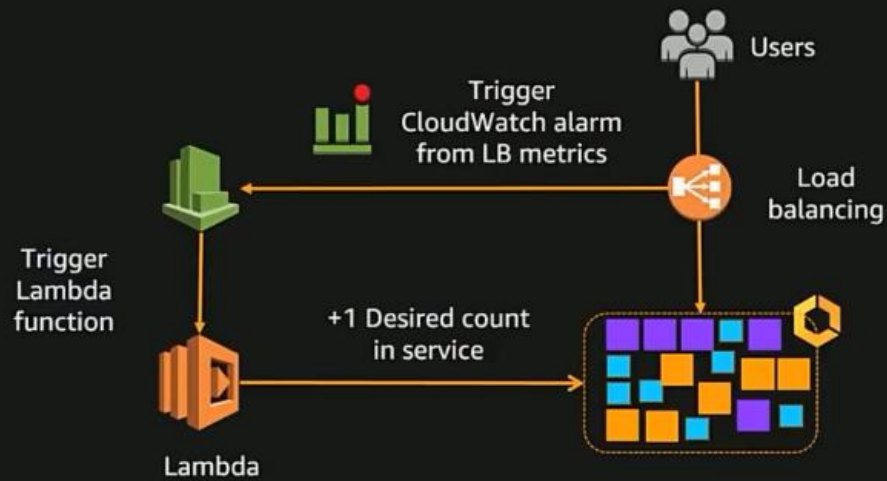


Let us see how we can use ECS and other triggers to scale up our apps. You have an app that you are running X number of tasks for, how do you know when you want to scale it out? AWS gives you primitives to trigger CloudWatch alarms based on how much CPU and memory the service is utilizing, you can then decide how much instances you want to spin up.

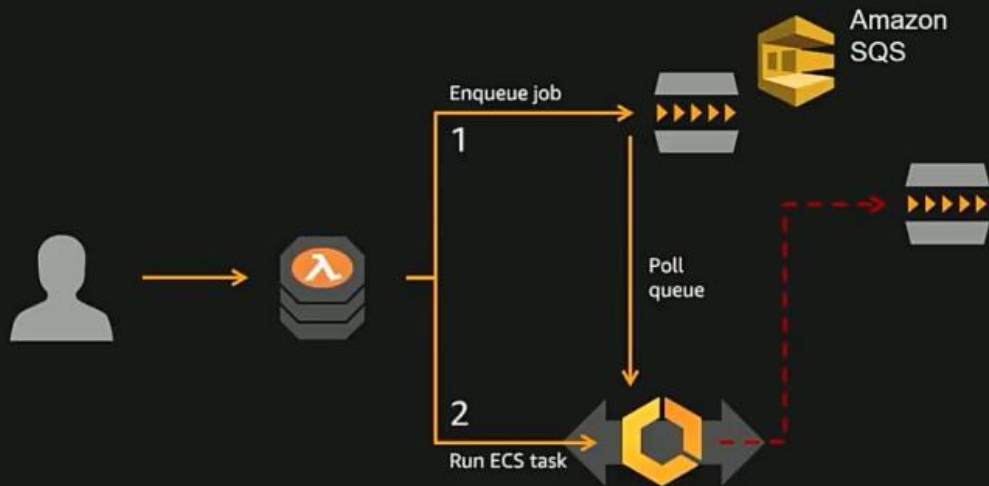
# Scheduled tasks



## Scheduling triggered by AWS services



## Scheduling triggered by AWS services



# Scheduling dependencies

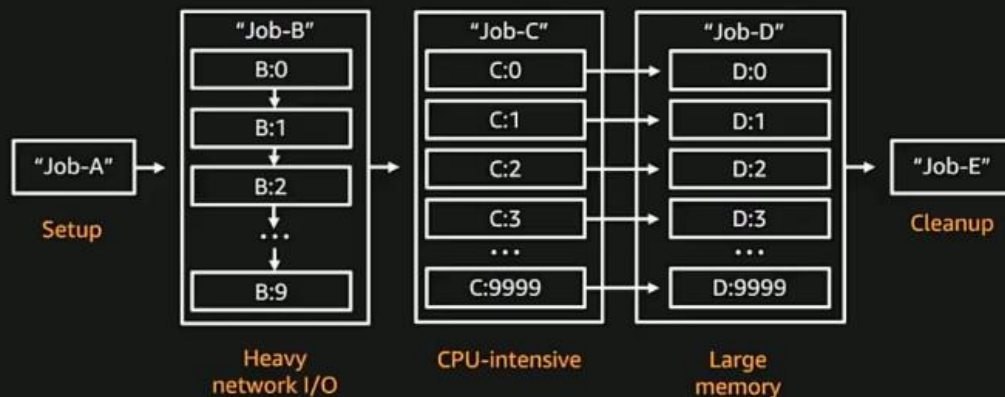


Simple job dependency modeling

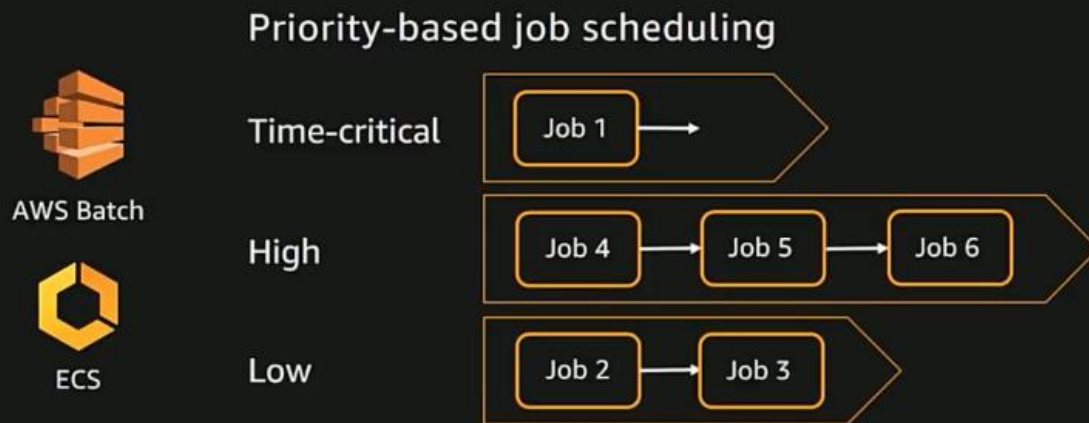
This will only run Job B after Job A completes

# Scheduling dependencies

NEW!



# Scheduling priorities



You can also put your jobs into different priority queues

## Simplifying container deployment

### CI/CD for long-running services

NEW!



# Rolling deployments

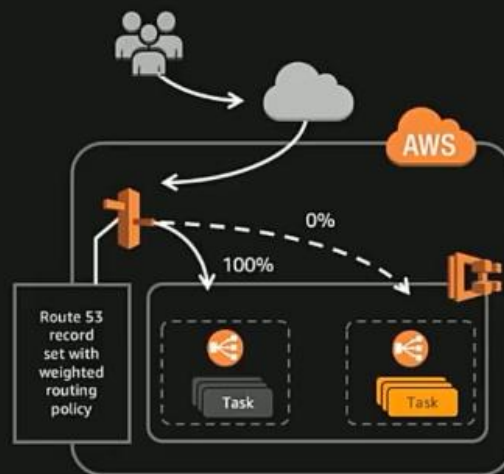


# Rolling deployments





# Canary blue-green deployments

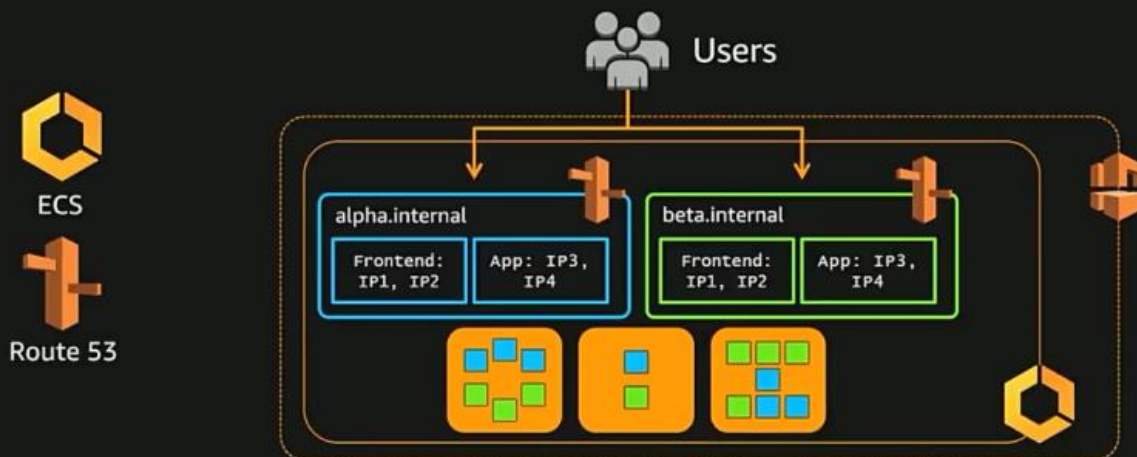


More at: [github.com/aws-labs/ecs-canary-blue-green-deployment](https://github.com/aws-labs/ecs-canary-blue-green-deployment)

This uses step functions behind the scene to roll over to the new version during the canary deployment

## ECS service discovery with Route 53

NEW!



More at: [CON403 on Friday at 10 a.m. ECS Service Discovery!](#)

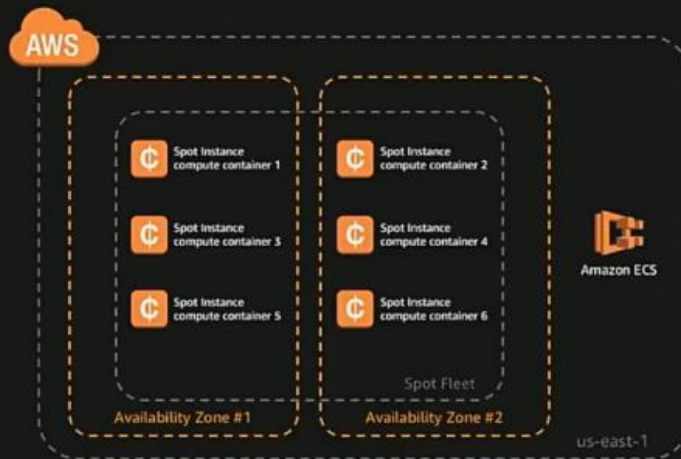
You can make services discover new versions of the other services they are interacting with. This uses the concept of registering your services with a namespace, this then allows you to have multiple versions of your services running in parallel at the same time. You clients can then do a DNS query to get the latest versions to call.

## Scheduling compute for containers

# Scheduling compute capacity

1. Spot instances
2. Auto Scaling groups—scale up
3. Auto Scaling group—scale down, draining

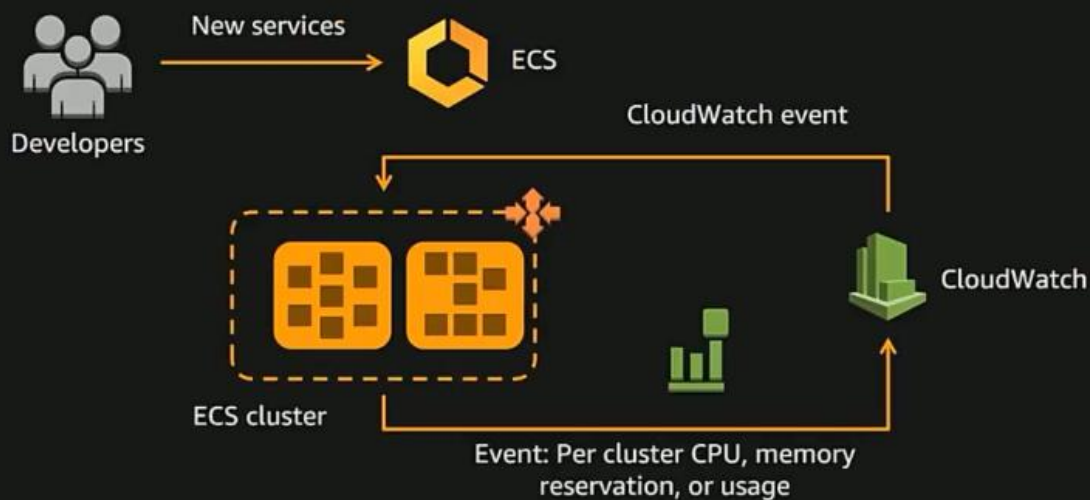
## Compute with EC2 Spot Instances



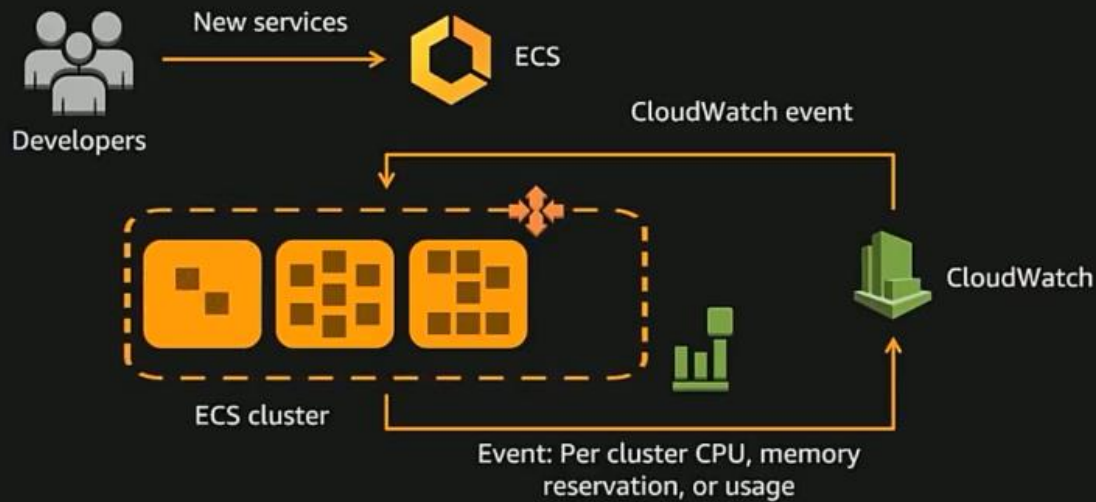
More at: <https://aws.amazon.com/blogs/compute/powering-your-amazon-ecs-cluster-with-amazon-ec2-spot-instances/>

ECS is integrated with Spot Fleet that allows you to maintain a fixed number of CPU or instances at all times using bidding processes. The instances can then be configured as part of your spot fleet cluster that can be spread across different AZs.

## Compute auto scale up



# Compute auto scale up



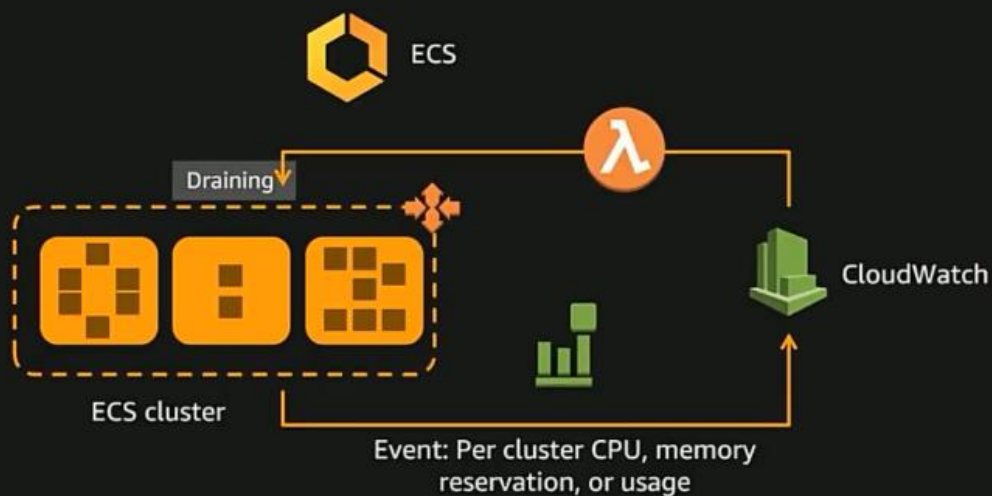
AWS  
re:Invent

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

aws

The ASG can be used to increase the size of your cluster based on some cloudwatch metric/event, then the scheduler can start placing instances in them as needed

# Compute auto scale down



AWS  
re:Invent

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

aws

You can also have the ASG drain excess capacity based on utilization metric tied to a lambda function that will set the 'draining' flag on instances that will be deleted. This allows the ECS scheduler to stop placing tasks on the draining instances

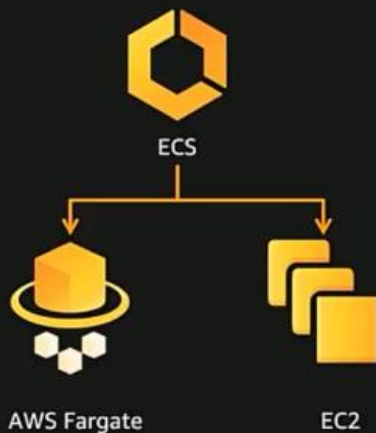
# Introducing AWS Fargate

Enable developers  
to focus 100% on  
their applications



## AWS Fargate

With AWS Fargate, instance management  
goes away



### ELASTIC

Scale services up and down seamlessly  
Pay only for what you use

### INTEGRATED

With the AWS ecosystem  
Including VPC networking, Elastic Load Balancing, IAM  
permissions, CloudWatch, and more

### MANAGED BY AWS

No EC2 instances to provision, scale, or manage

## More on AWS Fargate

CON214 – Introducing AWS Fargate

Today, 4 p.m. Aria, Level 1, Pinyon 2

CON201 – Containers on AWS – State of the Union

Tomorrow, 12:15 p.m. Aria, Level 1, Pinyon 5

CON333 – Deep Dive into AWS Fargate

Tomorrow, 1 p.m. Aria, Level 1, Pinyon 2

# AWS re:Invent

Thank you!

AWS  
re:Invent

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

