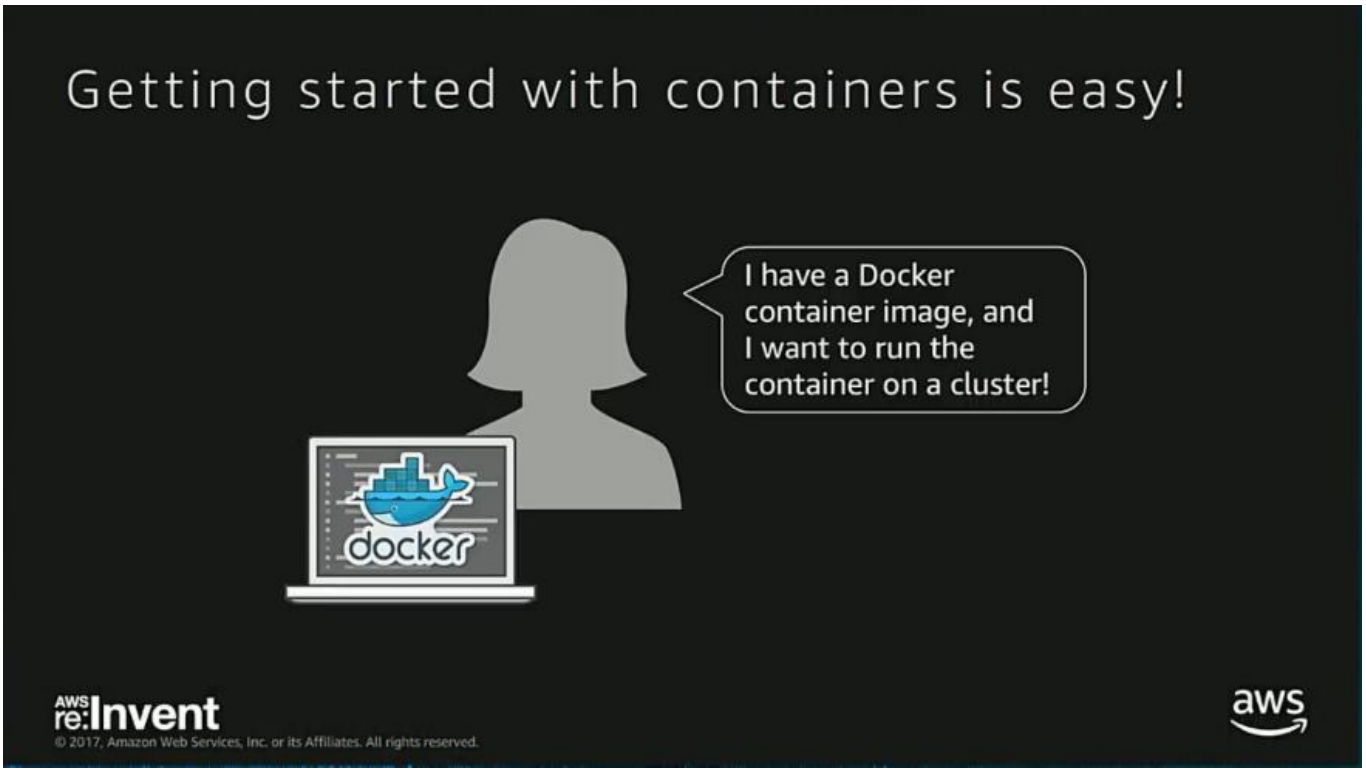
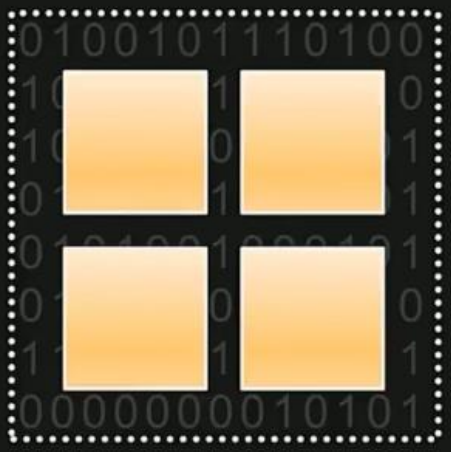
As your application's infrastructure grows and scales, well-managed container scheduling is critical to ensuring high-availability and resource optimization. In this session, we will deep dive into the challenges and opportunities around container scheduling, as well as the different tools available 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

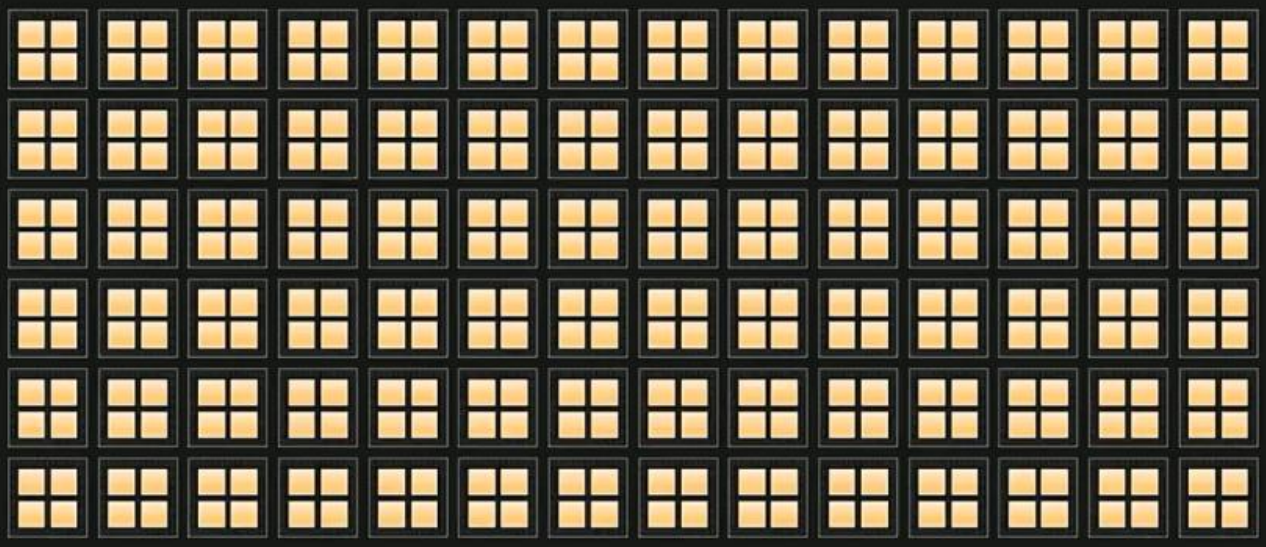


From one container...



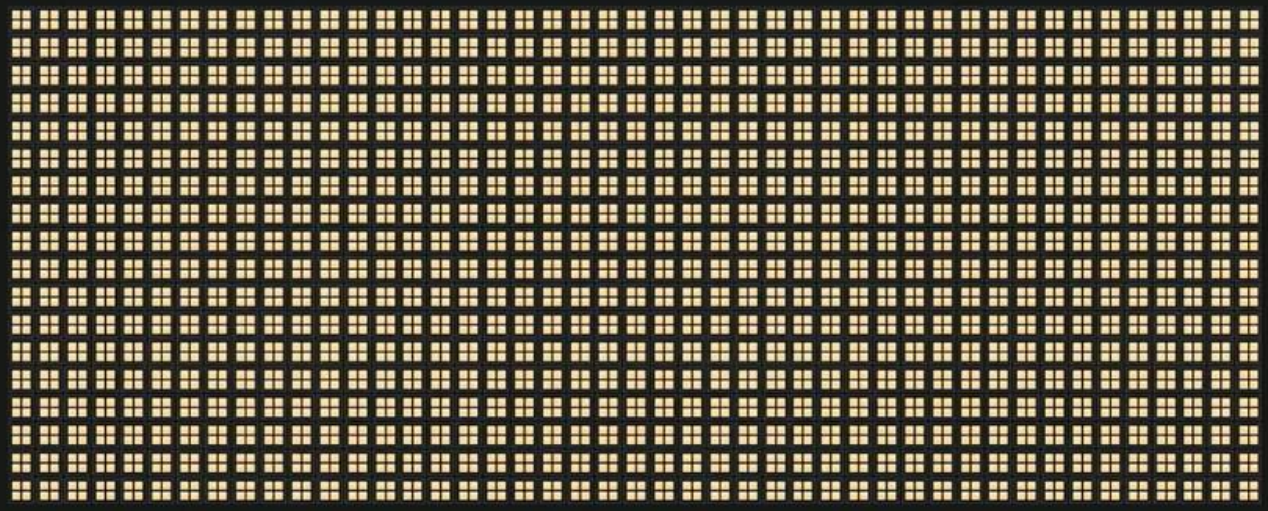
AWS re:Invent
© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.
aws

...to hundreds...



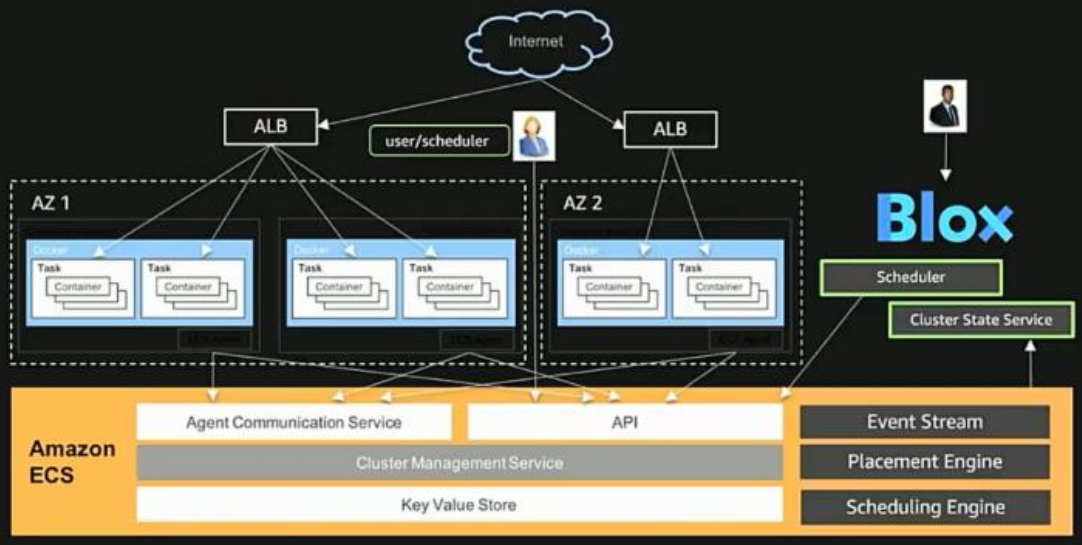
AWS re:Invent
© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.
aws

...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.
aws

Core components

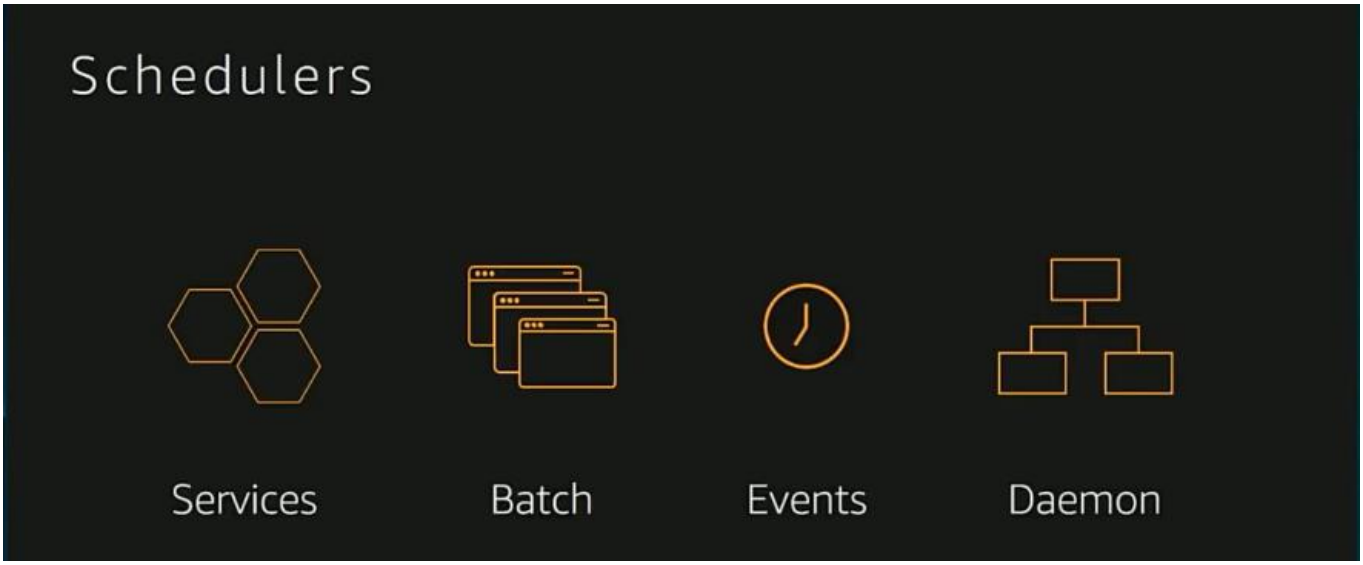
Scheduling engine

Placement engine

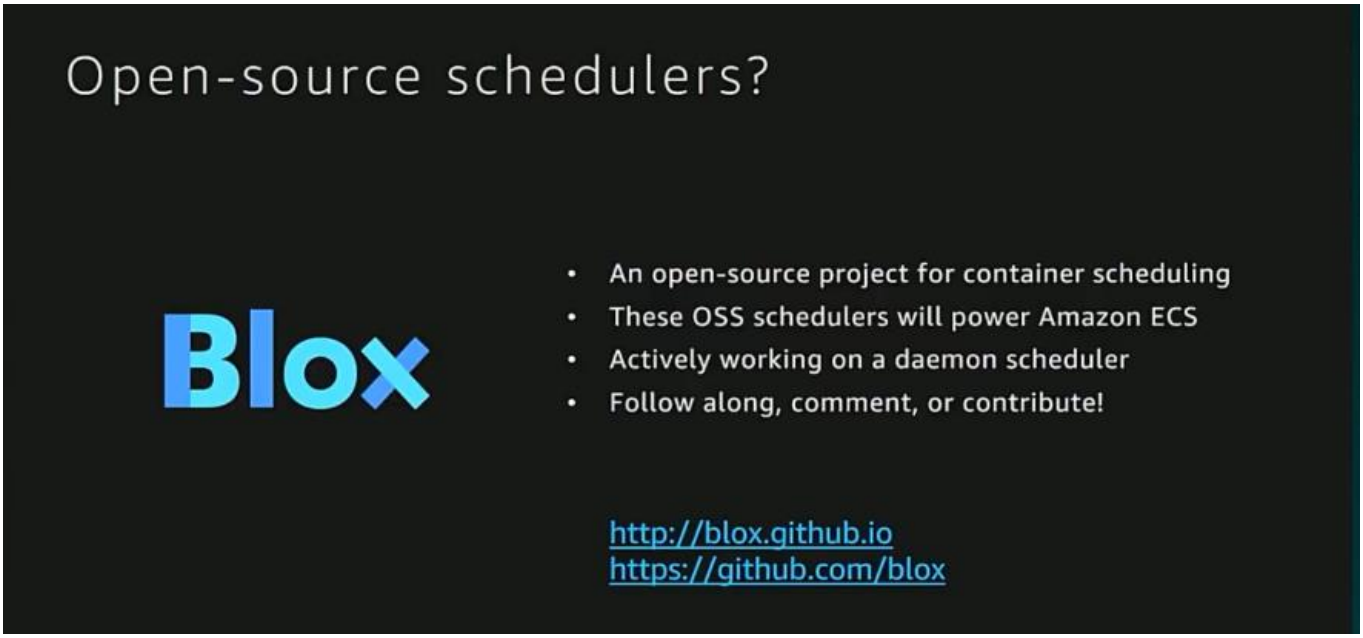
Extensions

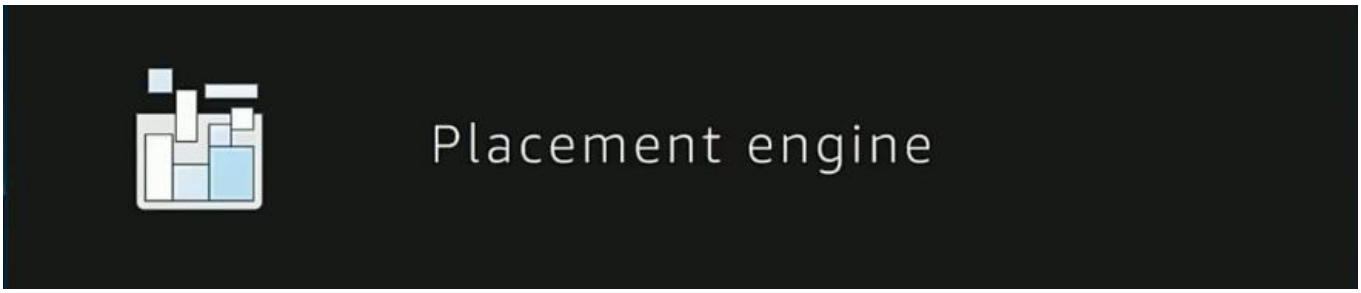
Scheduling engines

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



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***.

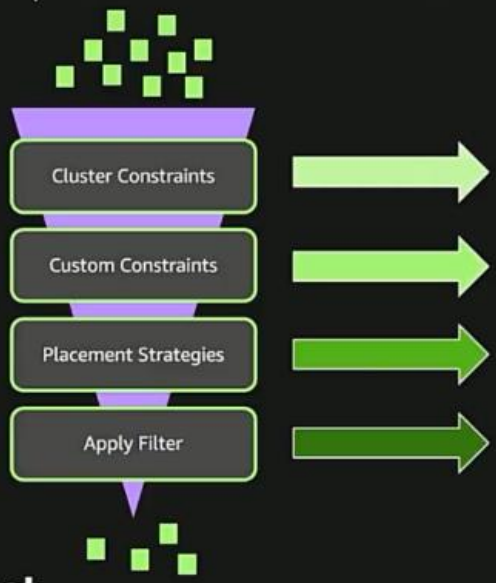




Task placement engine

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

Task placement selection



Satisfy CPU, memory, and port requirements

Filter for location, instance type, AMI, or custom attribute constraints

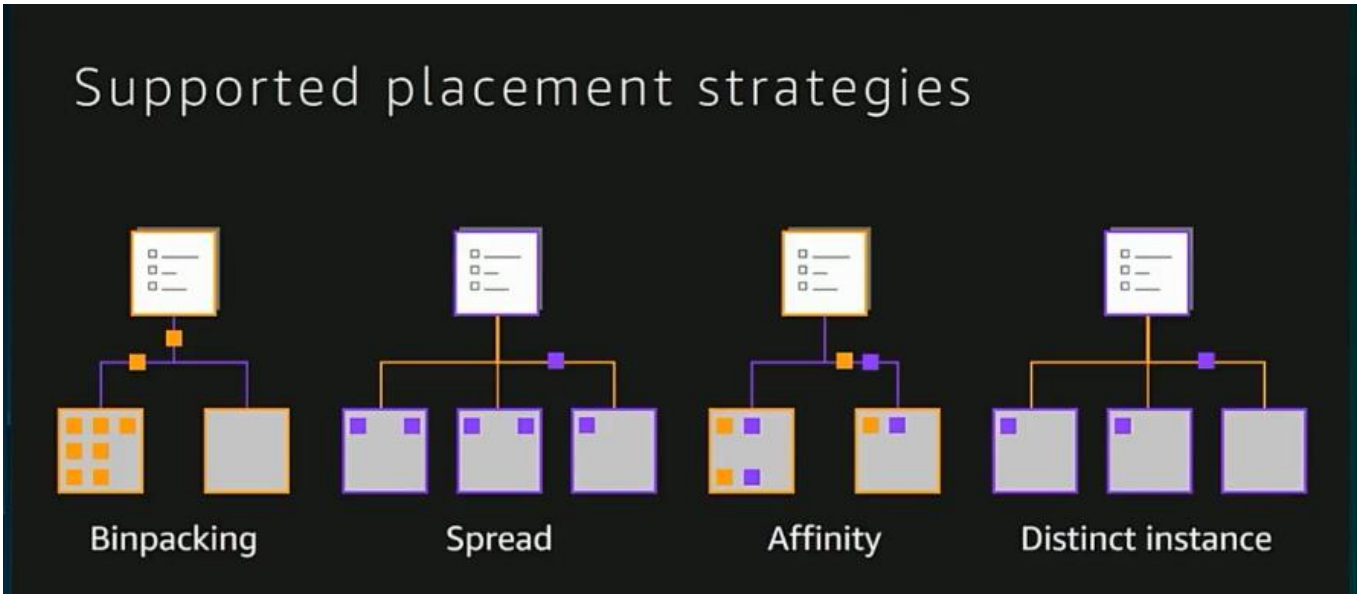
Identify instances that meet spread or binpack placement strategy

Select final container instances for placement

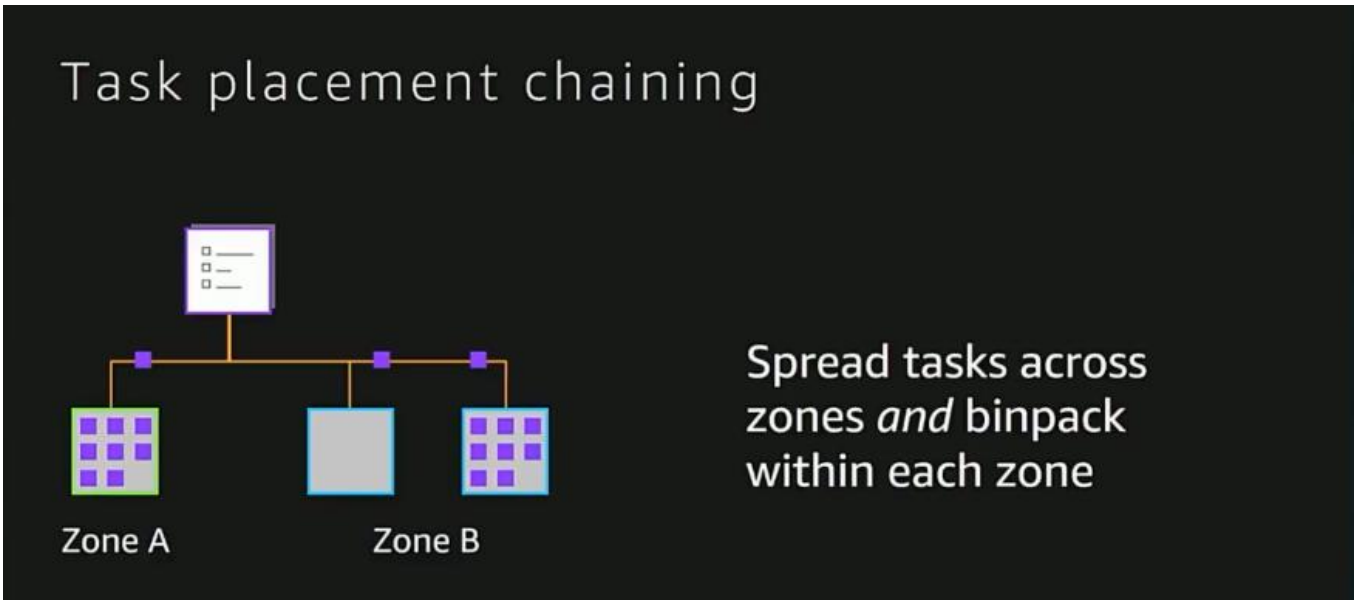
AWS re:Invent

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

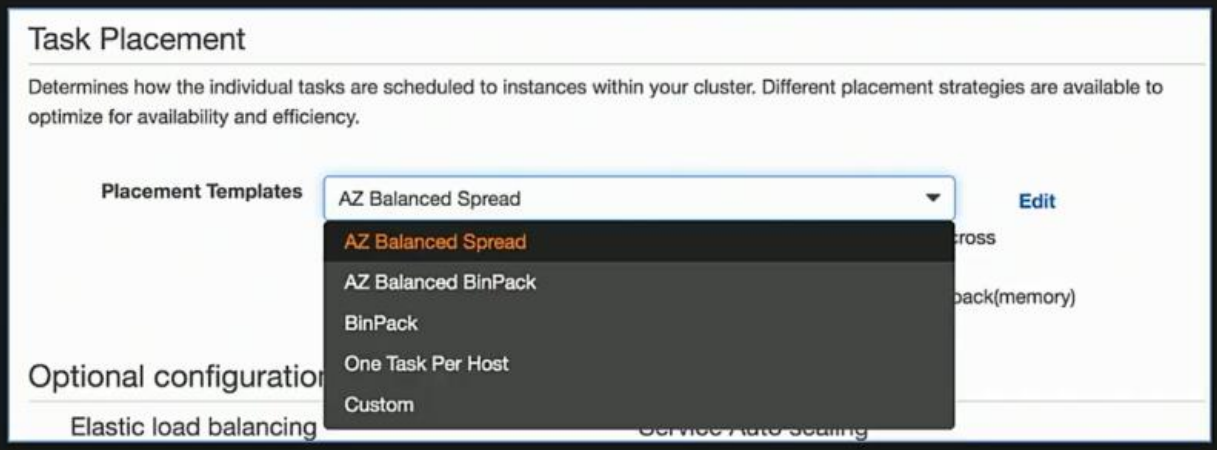
aws



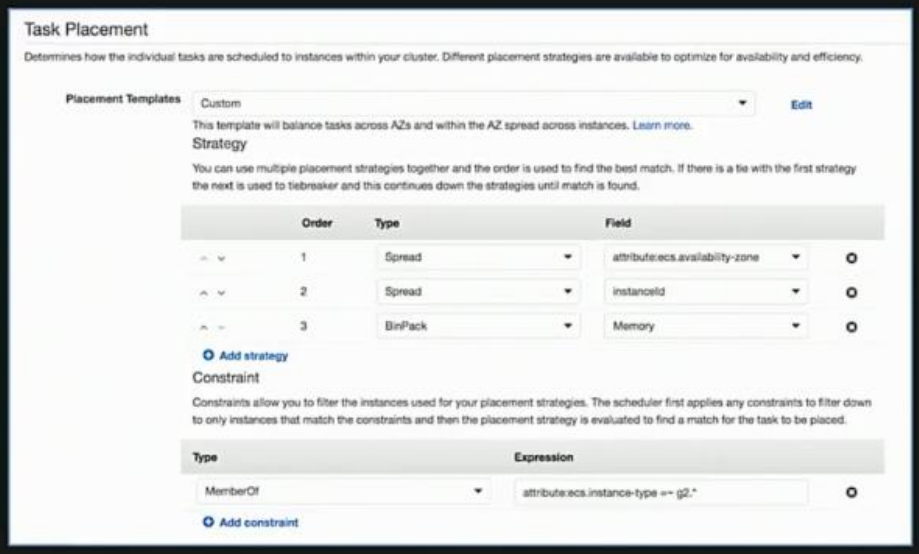
These are the default strategies offered with ECS today



Task placement: Easy to get started



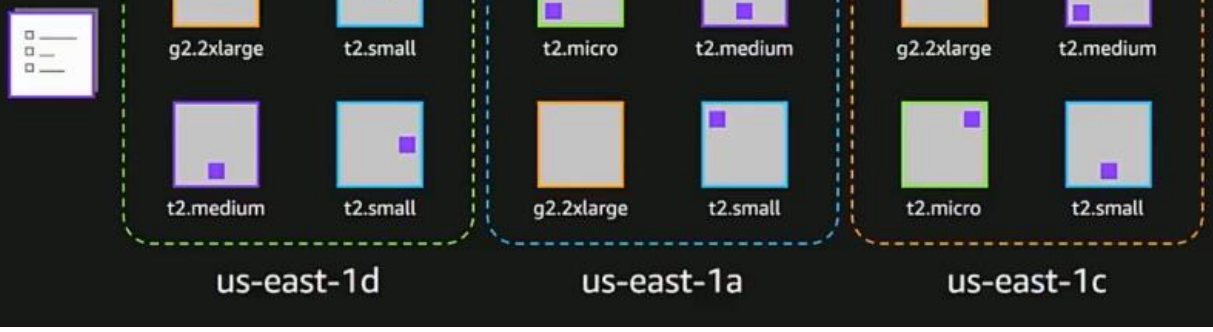
Task placement: Easy to get started



AWS re:Invent
© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.
aws

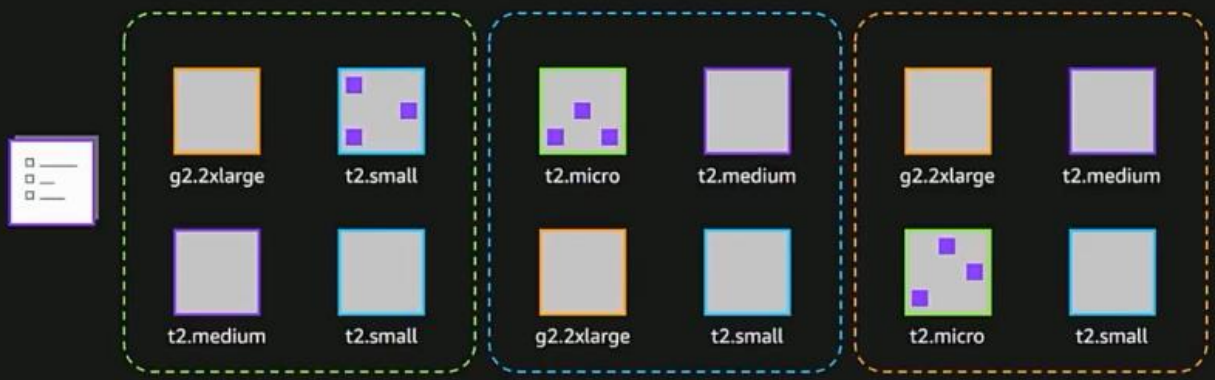
Example: Spread placement

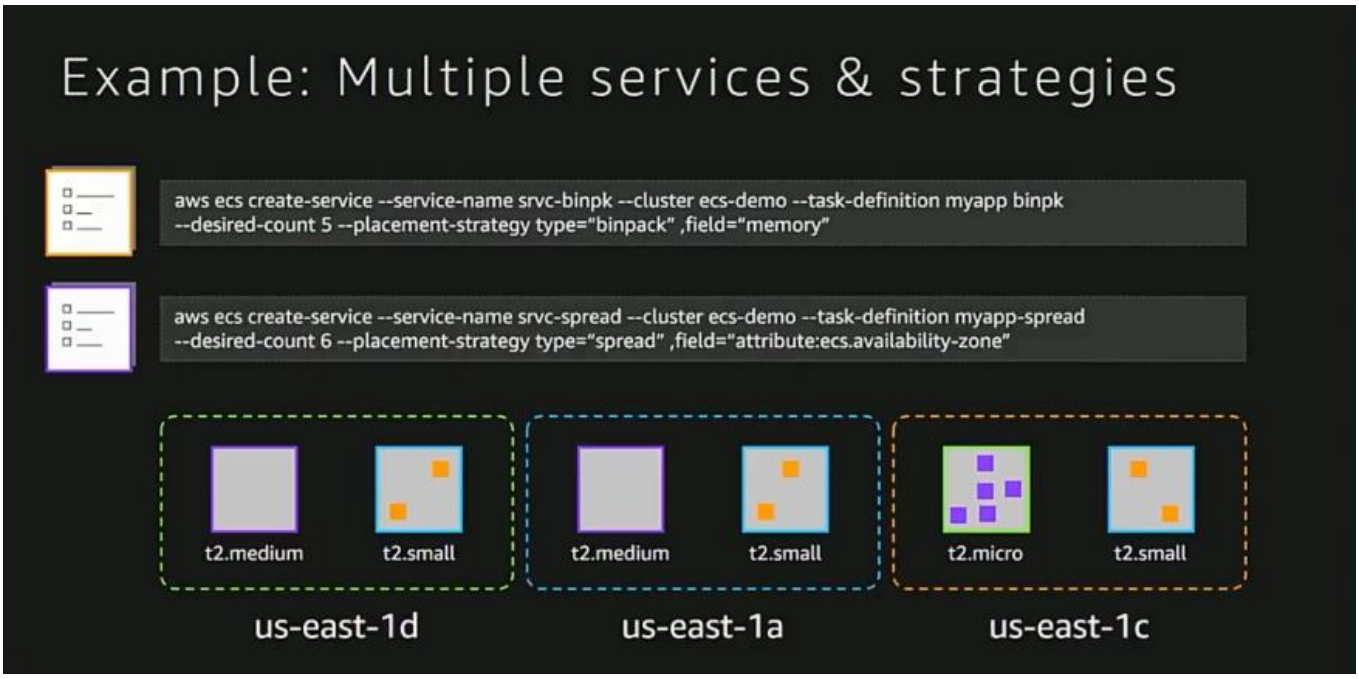
```
aws ecs run-task --cluster ecs-demo --task-definition myapp --count 9 --placement-strategy
type="spread" ,field="attribute:ecs.availability-zone"
```

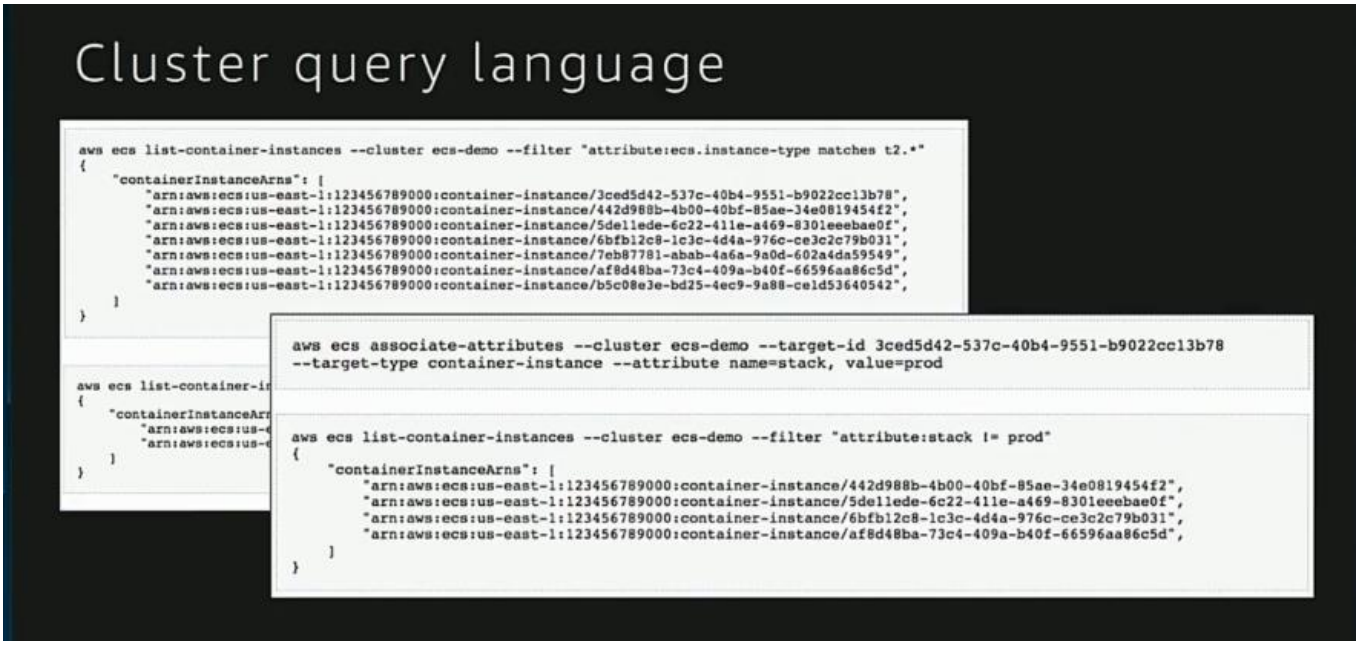


Example: Spread & binpack

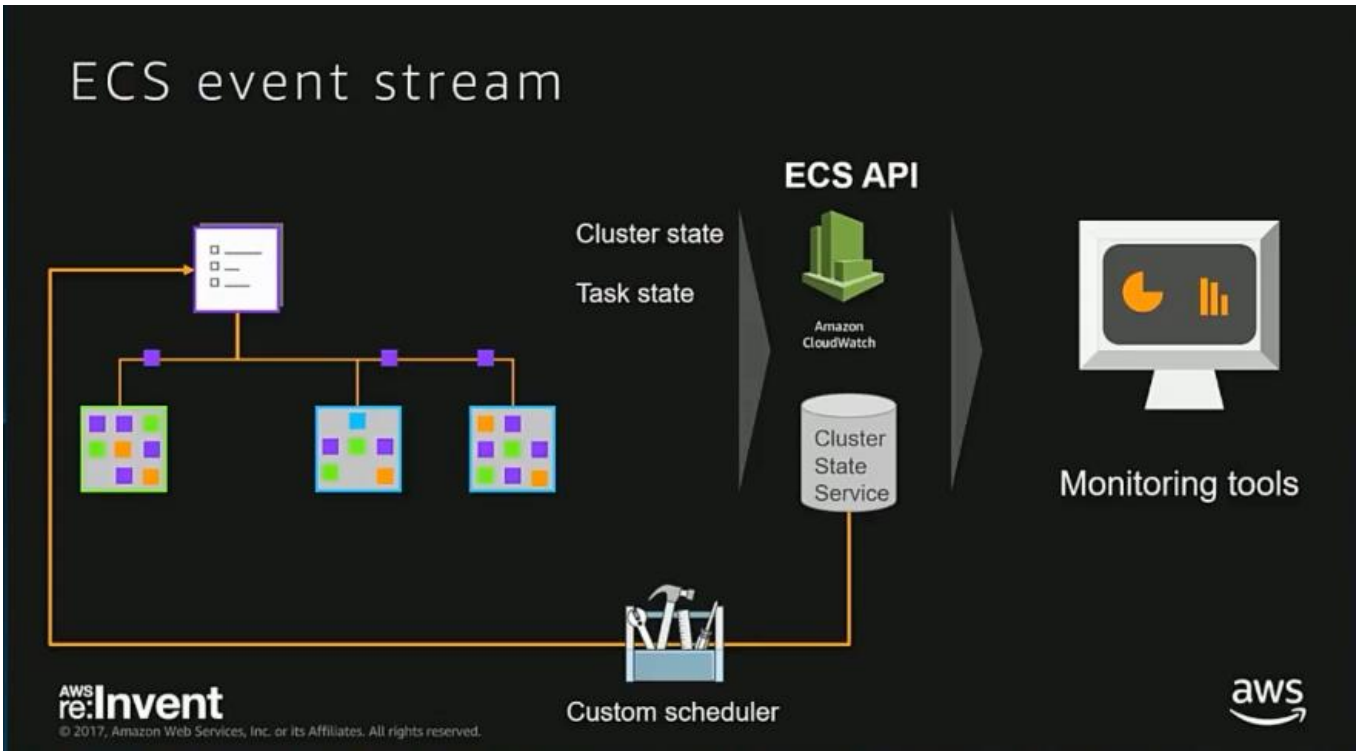
```
aws ecs run-task --cluster ecs-demo --task-definition myapp --count 9 --placement-strategy
type="spread" ,field="attribute:ecs.availability-zone" type="binpack" ,field="memory"
```







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.



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.

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

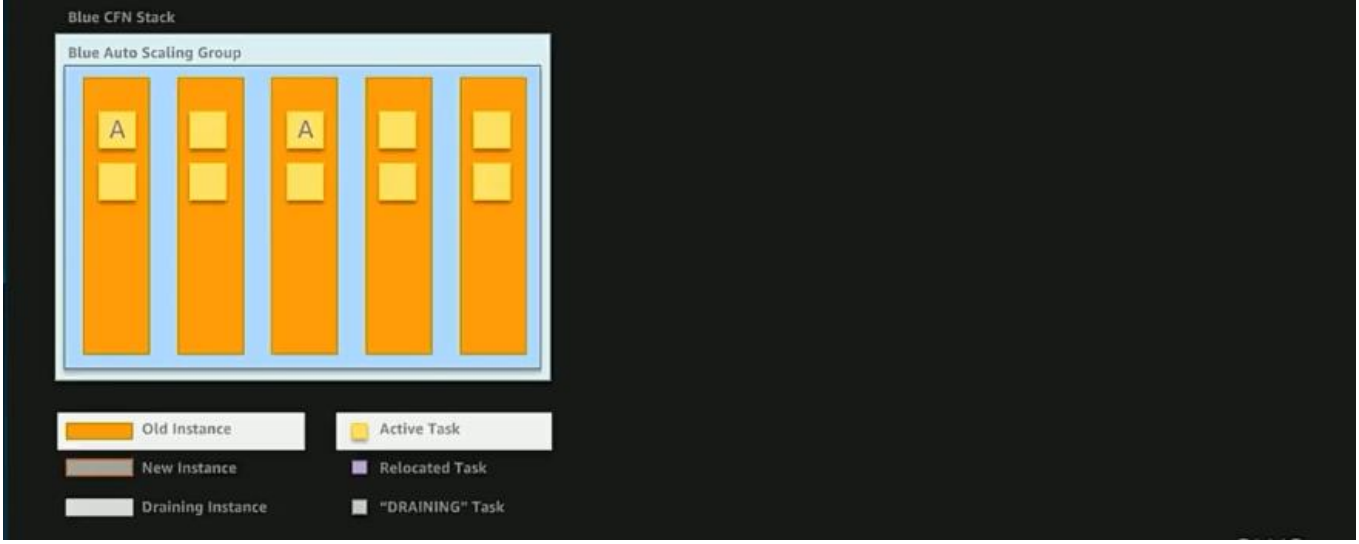
```
placement_constraints = [ {
        type: 'memberOf',
        expression: 'attribute:state !exists or attribute:state != pre-drain'
    }]
```

- 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

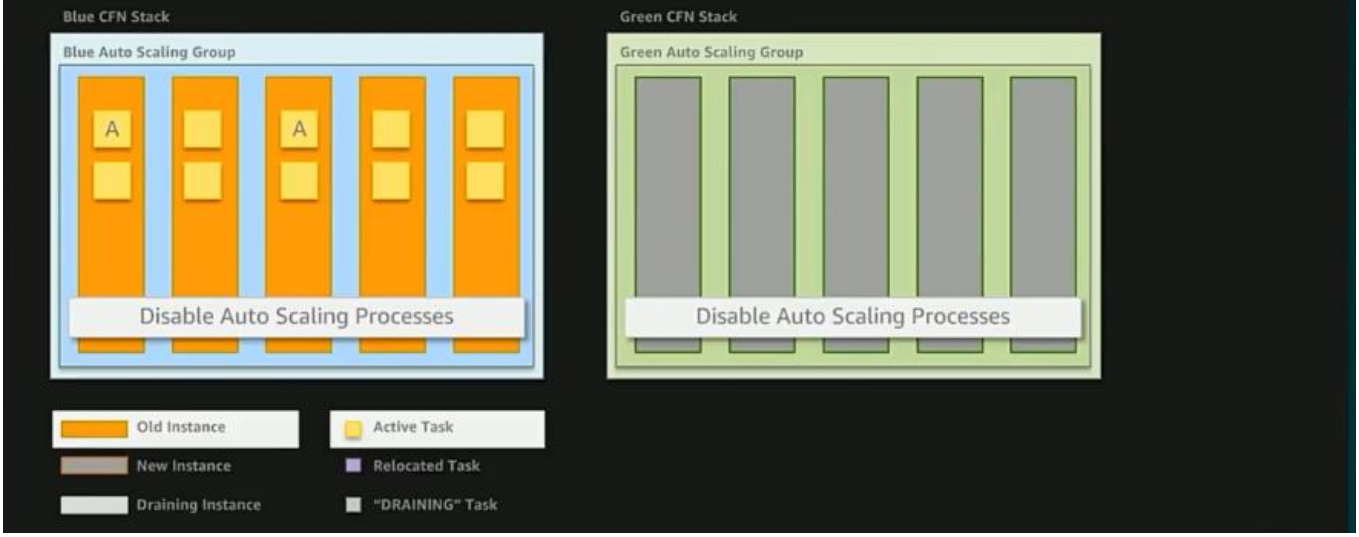
AWS re:Invent
© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.
aws

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

Cluster update—Phase 1: Expand

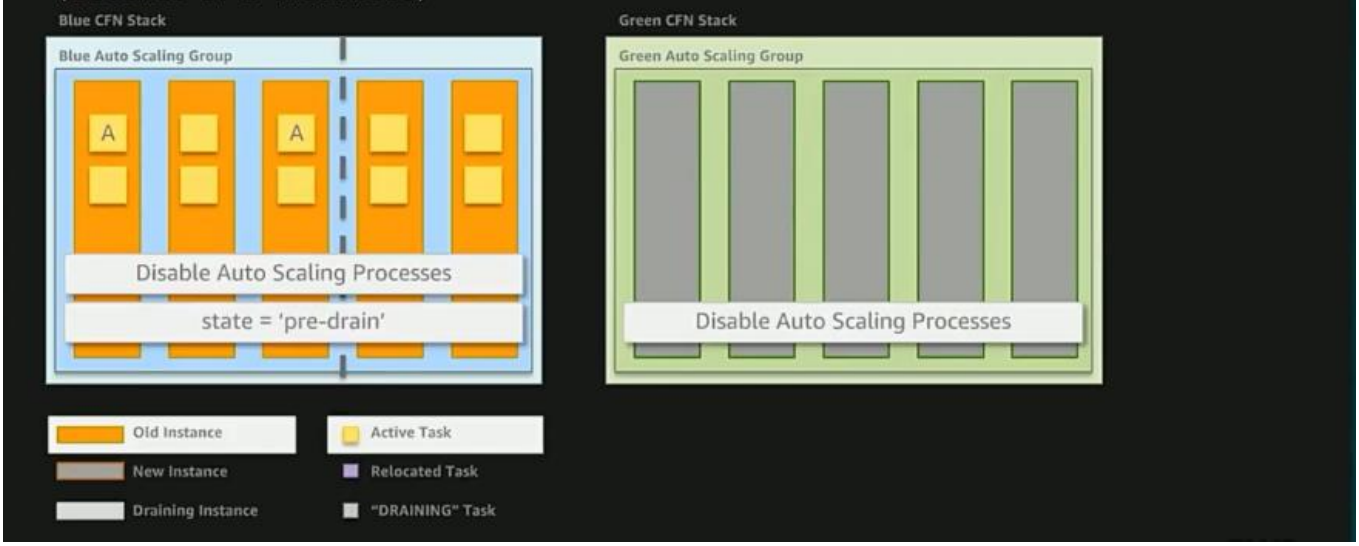


Cluster update—Phase 1: Expand



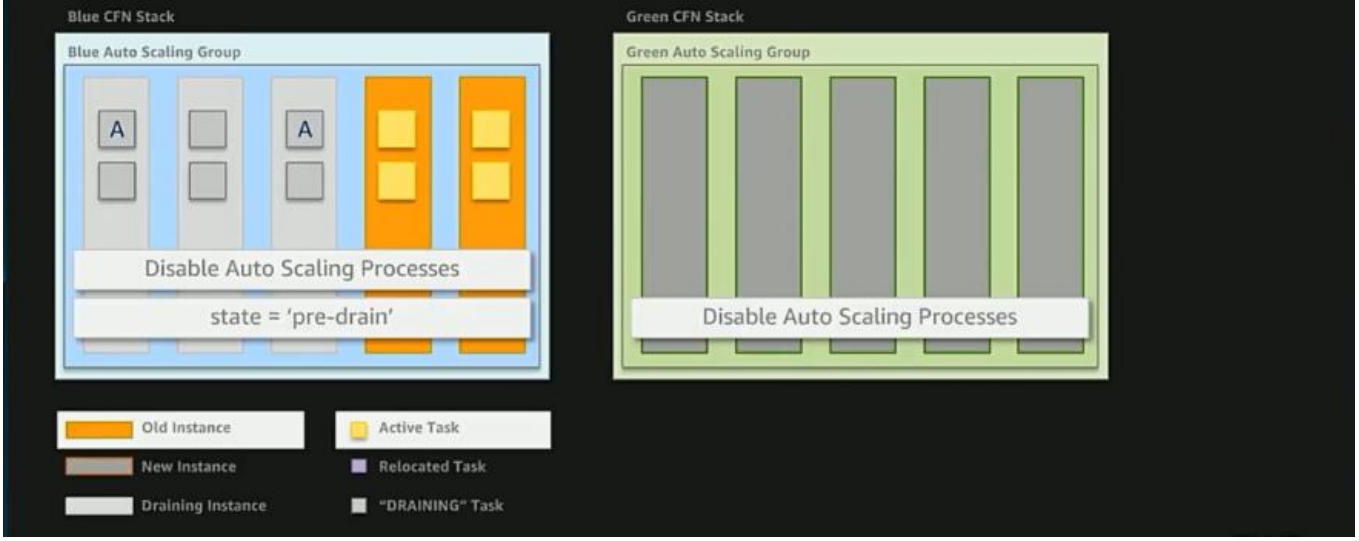
Cluster update—Phase 2: Relocate tasks

(Batches of 3 instances)



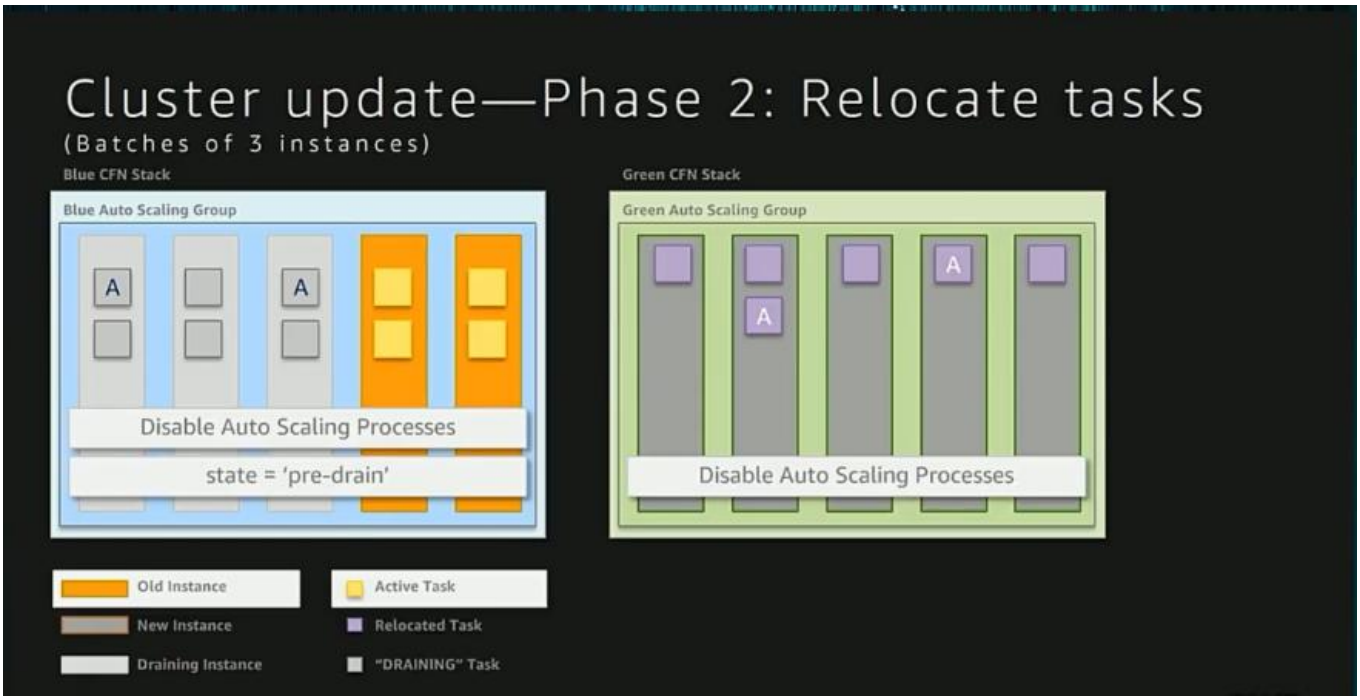
Cluster update—Phase 2: Relocate tasks

(Batches of 3 instances)



Cluster update—Phase 2: Relocate tasks

(Batches of 3 instances)



Cluster update—Phase 2: Relocate tasks

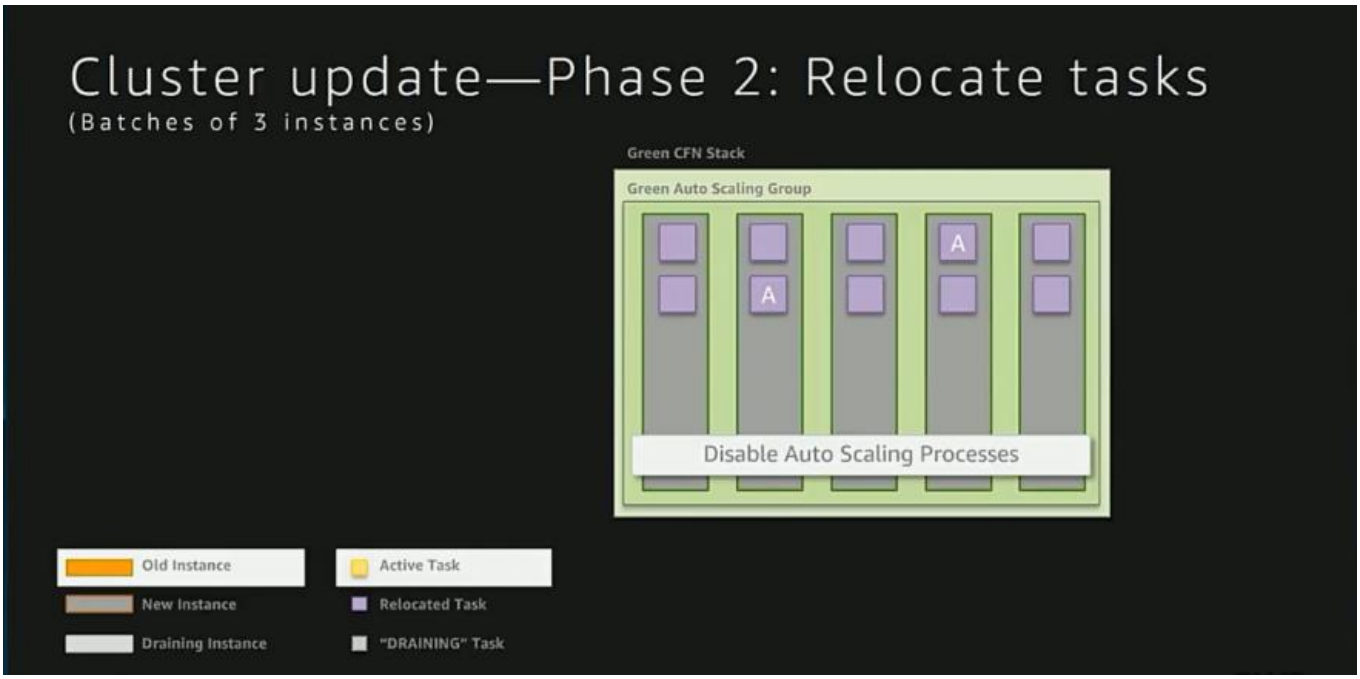
(Batches of 3 instances)

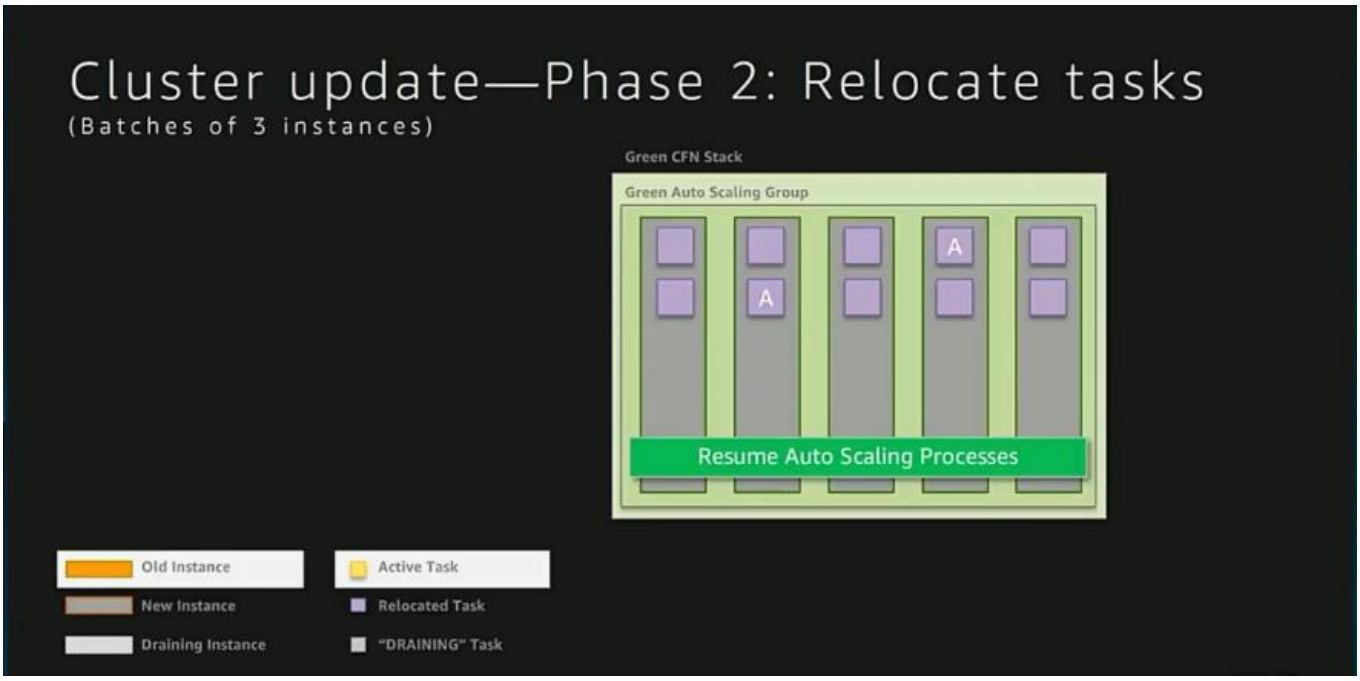
Cluster update—Phase 2: Relocate tasks

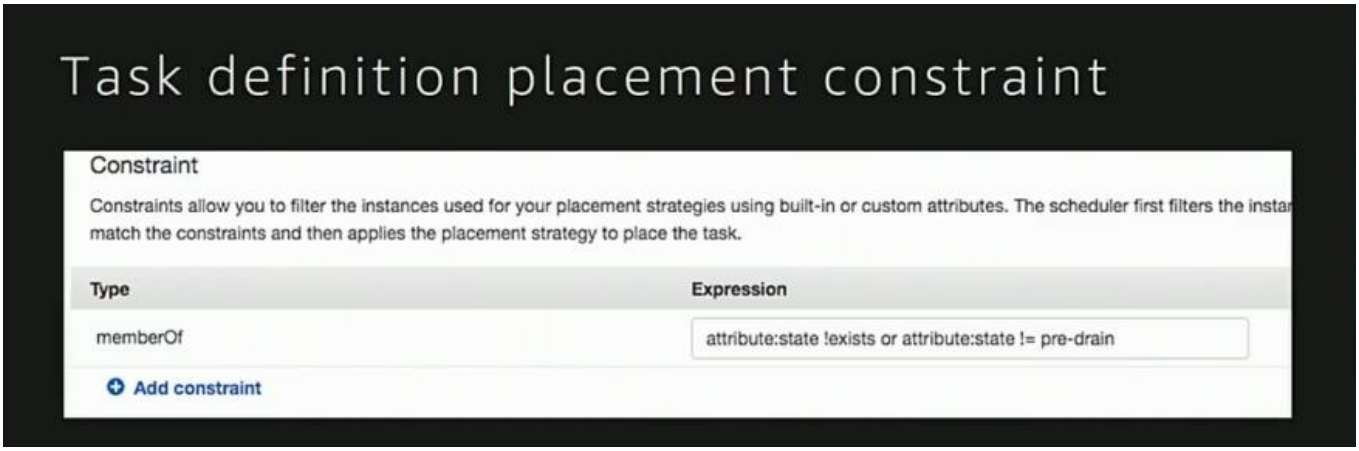
(Batches of 3 instances)

Cluster update—Phase 2: Relocate tasks

(Batches of 3 instances)

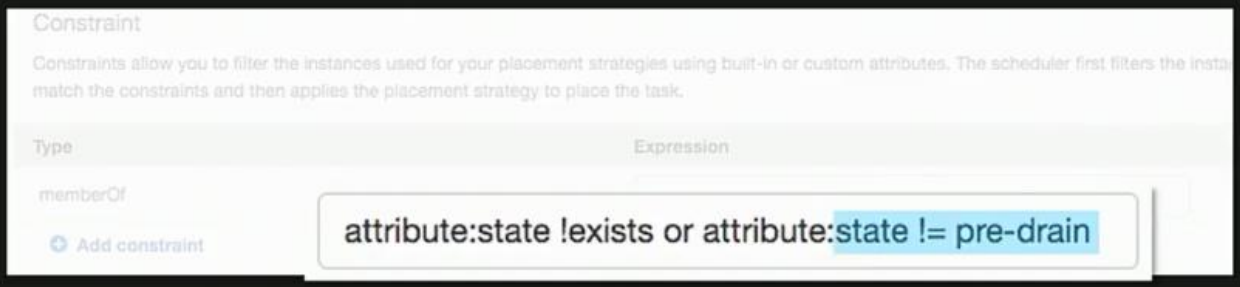




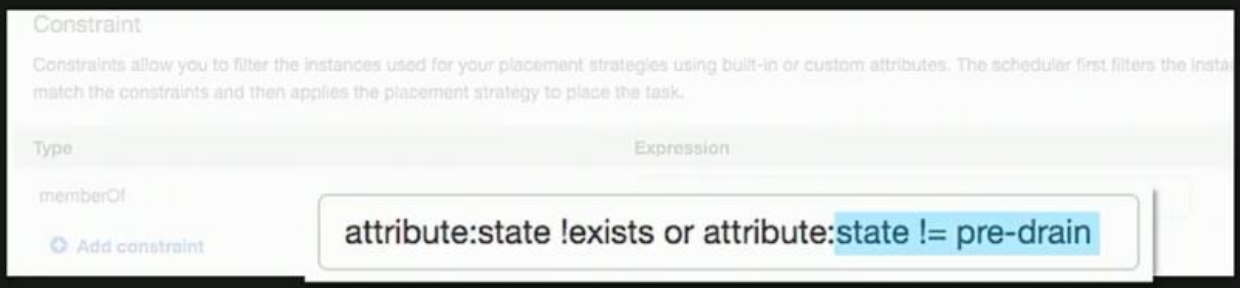


This is what it looks like in the UI even though we use automation

Task definition placement constraint



Task definition placement constraint



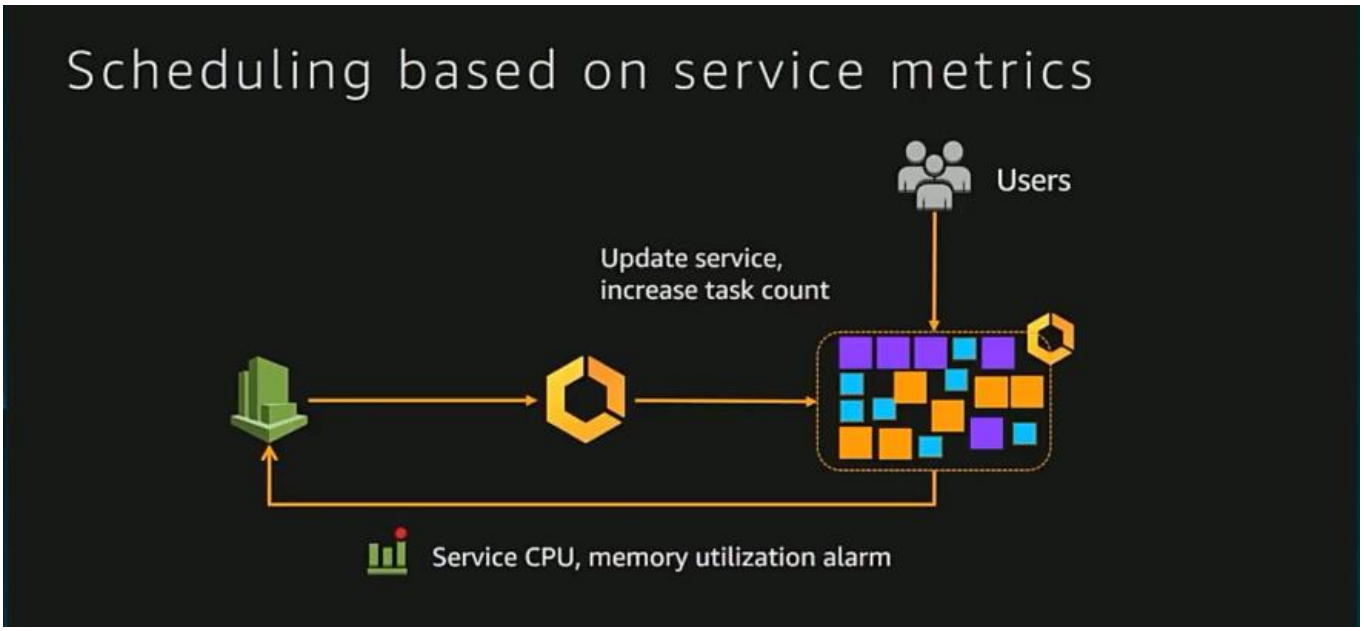
- "state" is a custom ECS Instance attribute
- By default, the "state" attribute doesn't exist on instances
- Set only to "pre-drain" during cluster update
- Prevents ECS scheduling tasks on instances that are about to be drained
- Removed from instance only in case of rollback of cluster update

Scheduling patterns

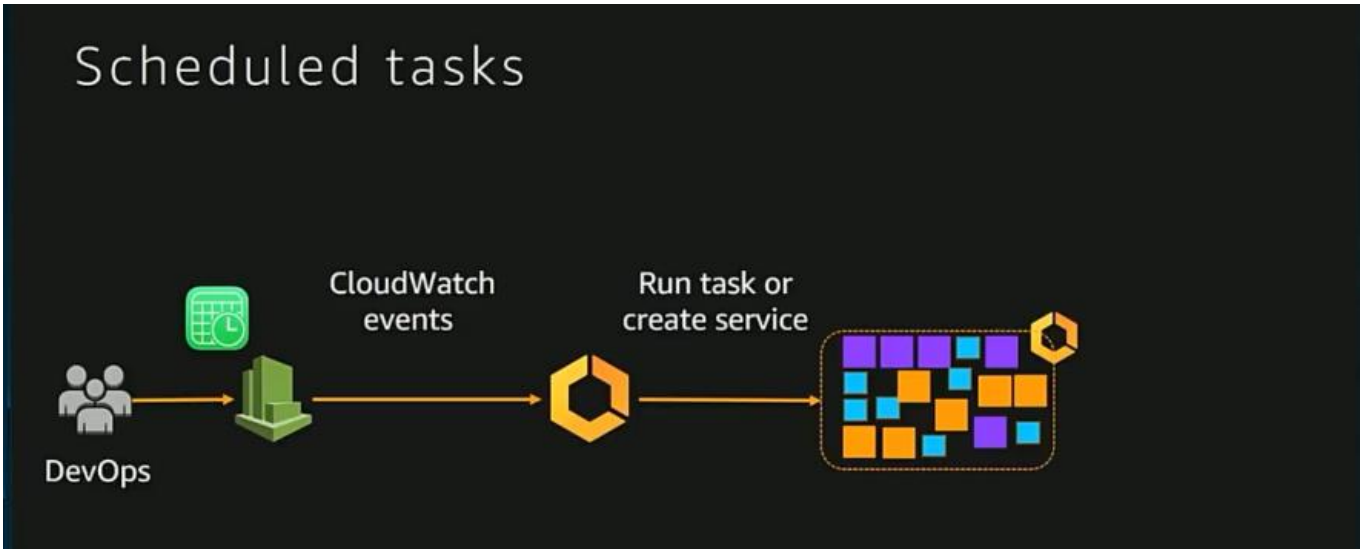
Scheduling based on...

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

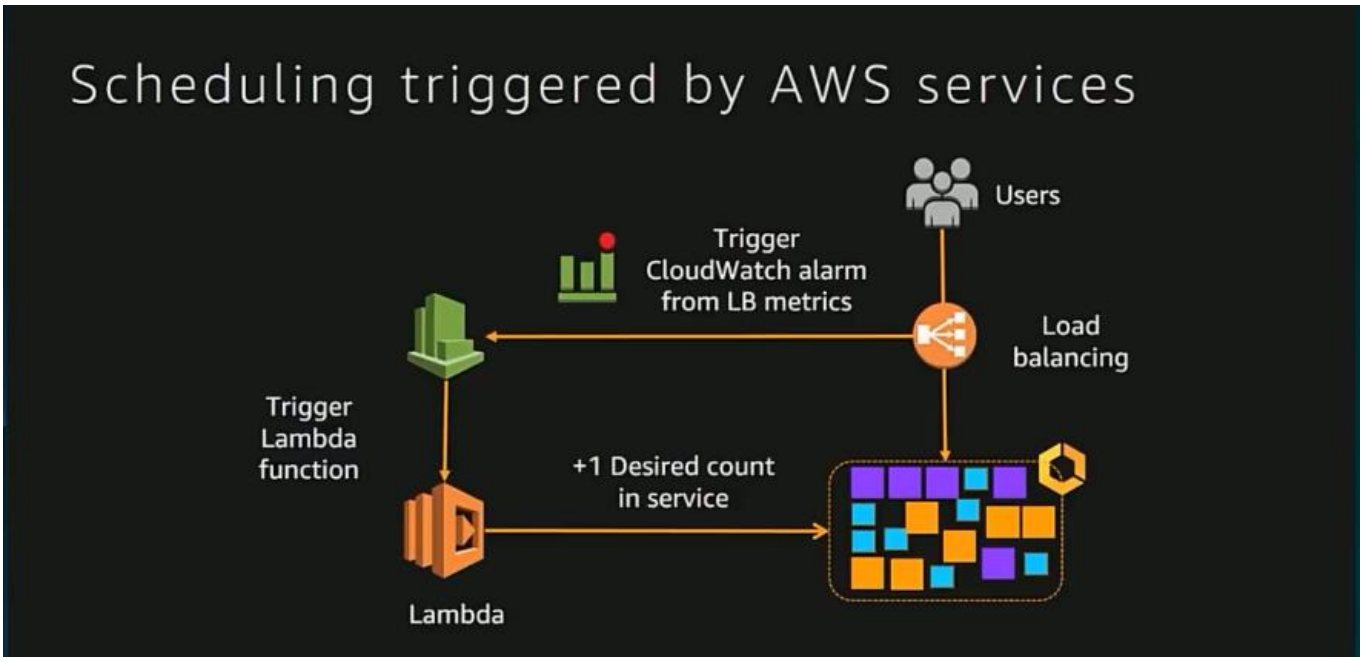
Let us see an example of using some of the scheduling primitives described earlier



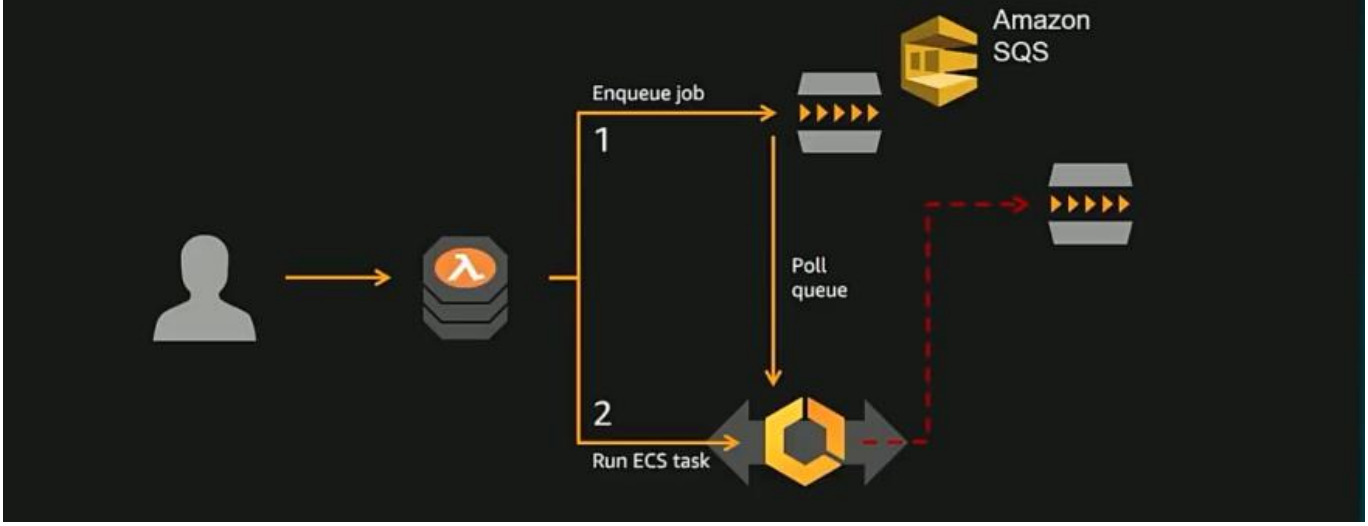
Let us see how we can use ECS and other triggers to scale up our apps. You have an app that you are running X number of tasks for, how do you know when you want to scale it out? AWS gives you primitives to trigger CloudWatch alarms based on how much CPU and memory the service is utilizing, you can then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

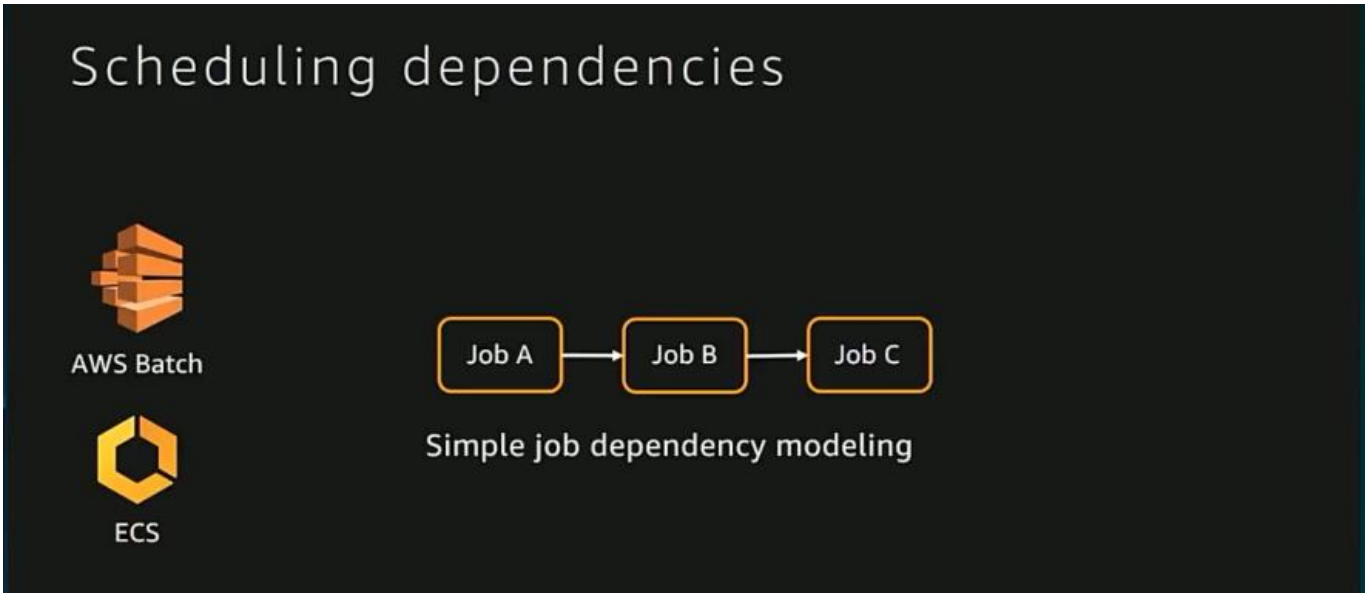


Scheduling triggered by AWS services

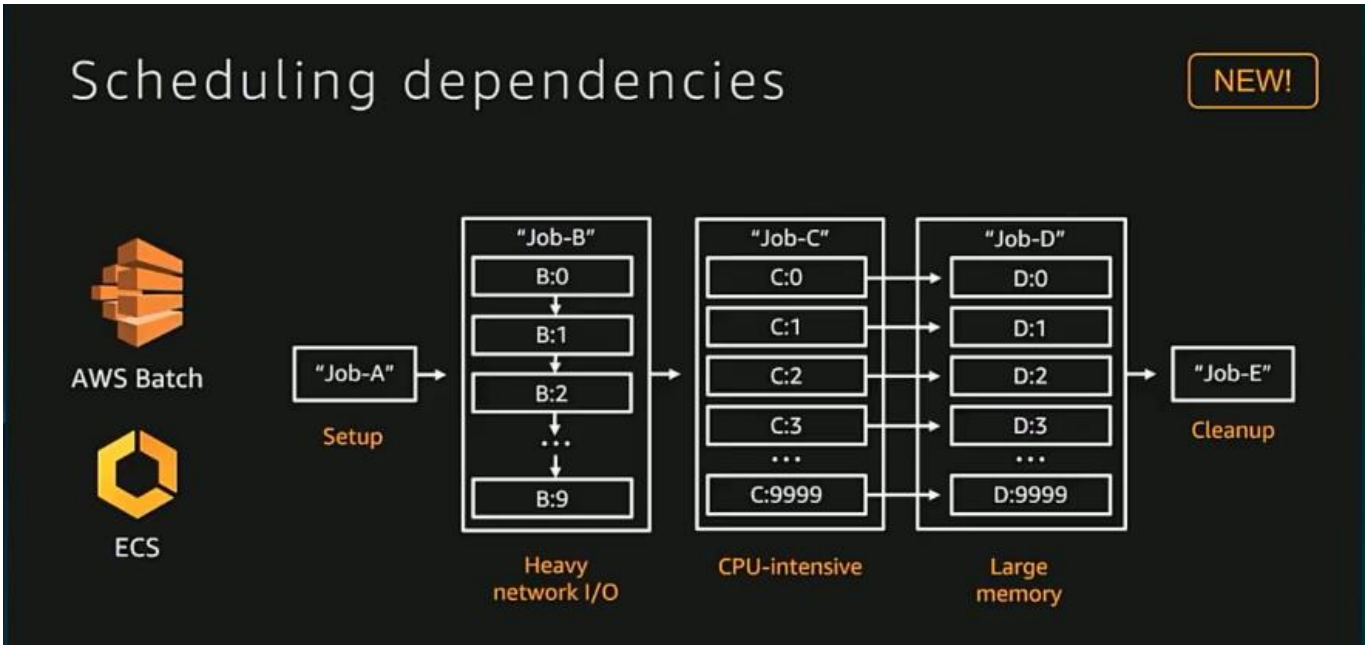


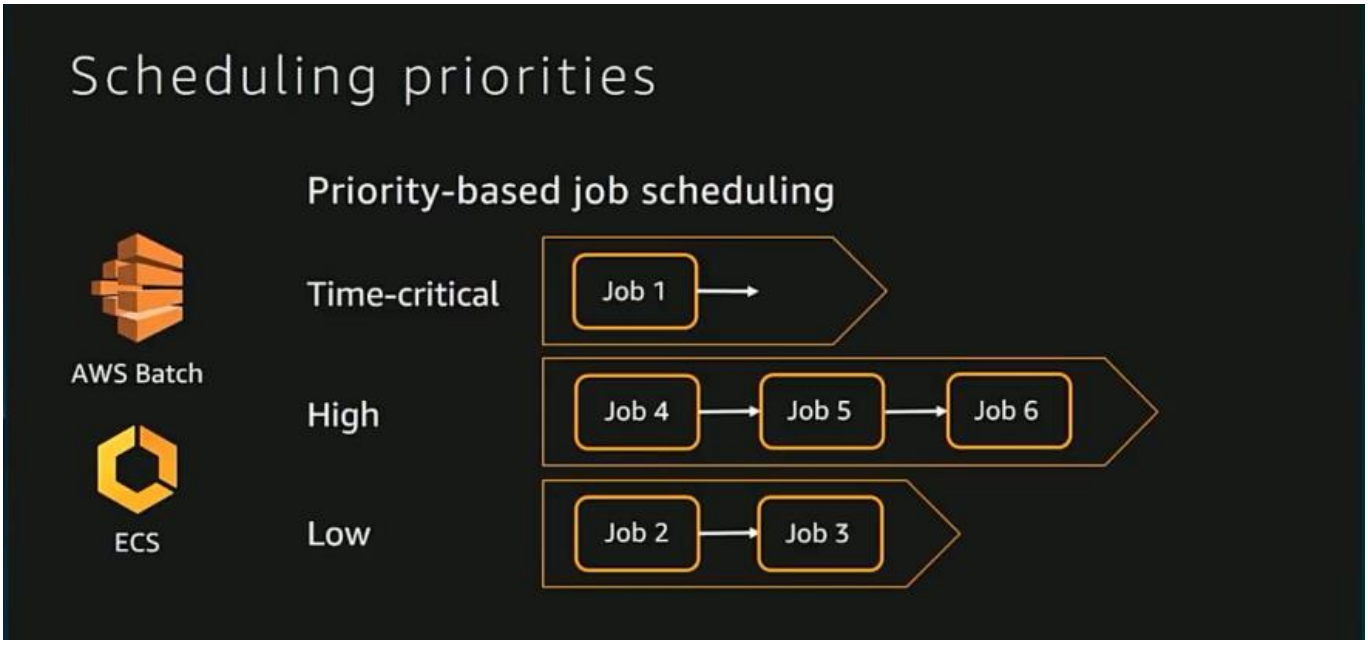
Scheduling triggered by AWS services



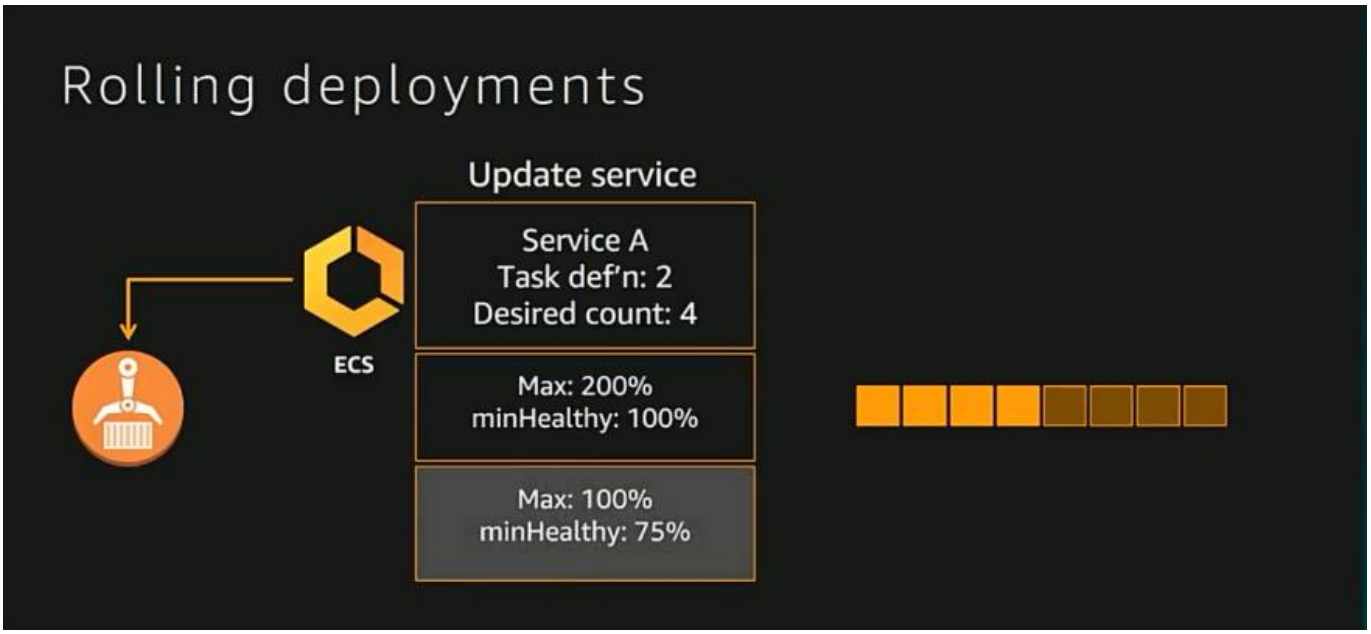


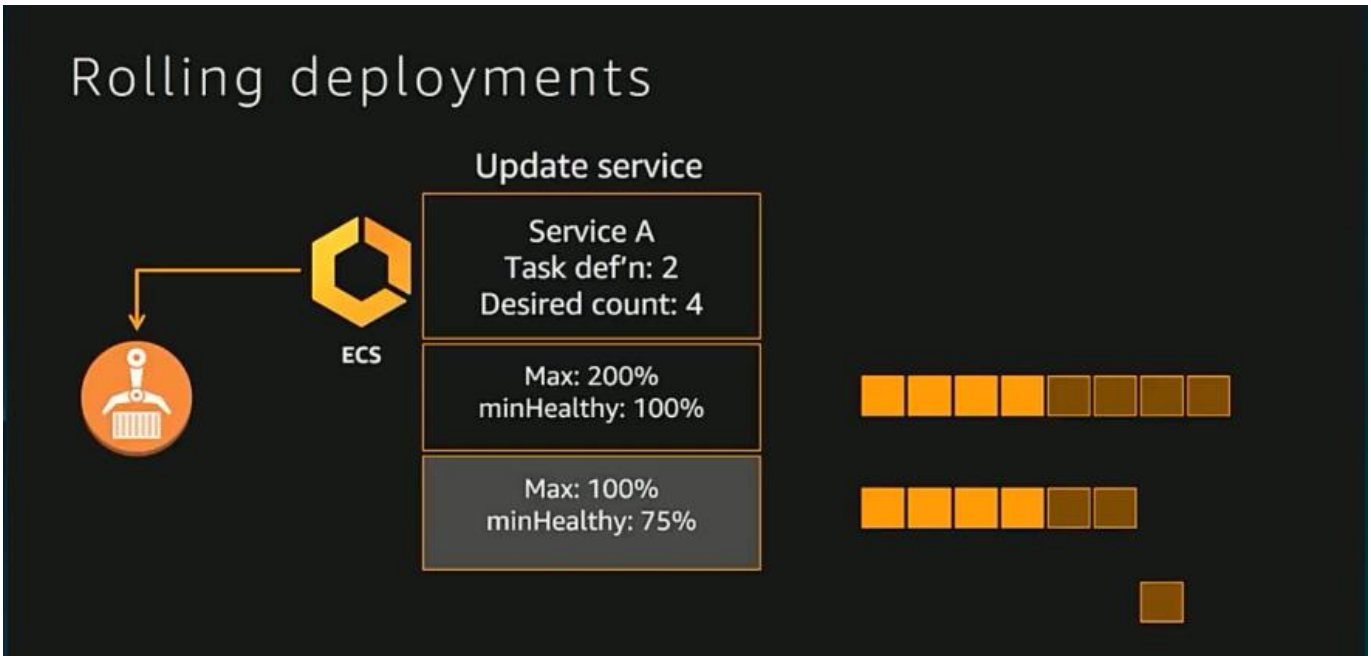
This will only run Job B after Job A completes

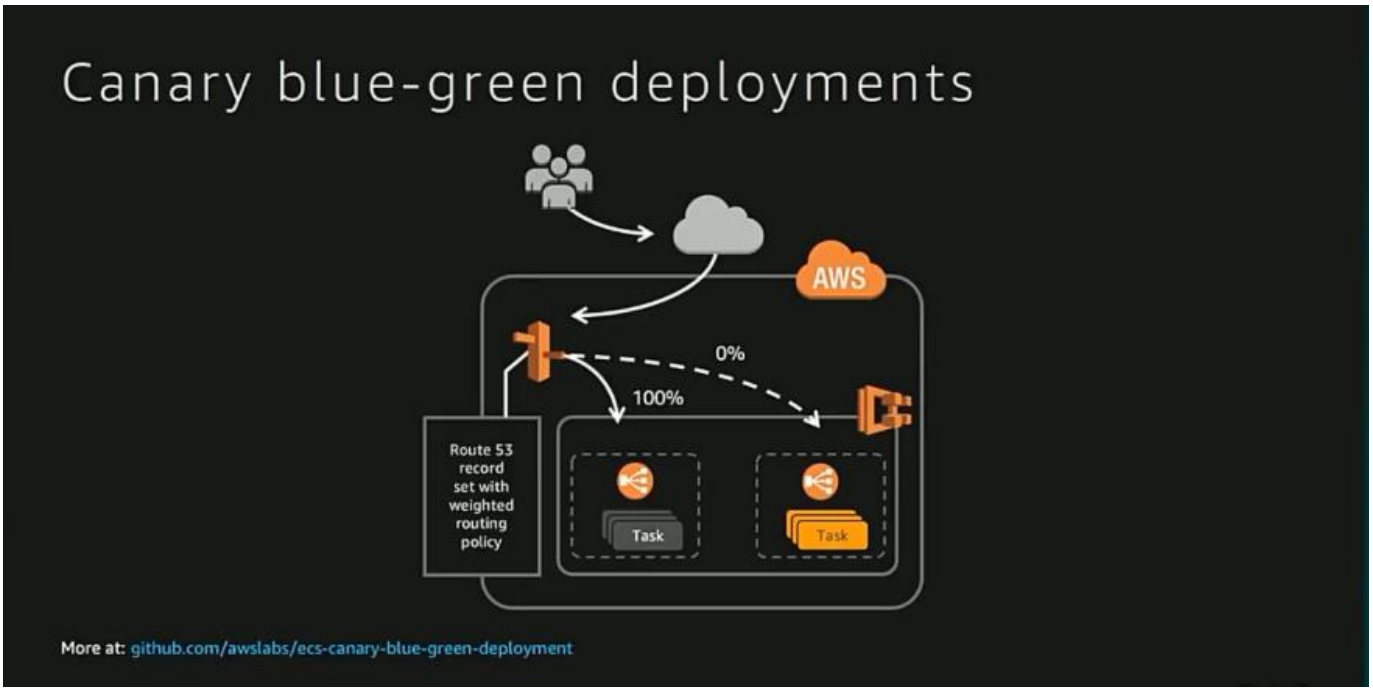




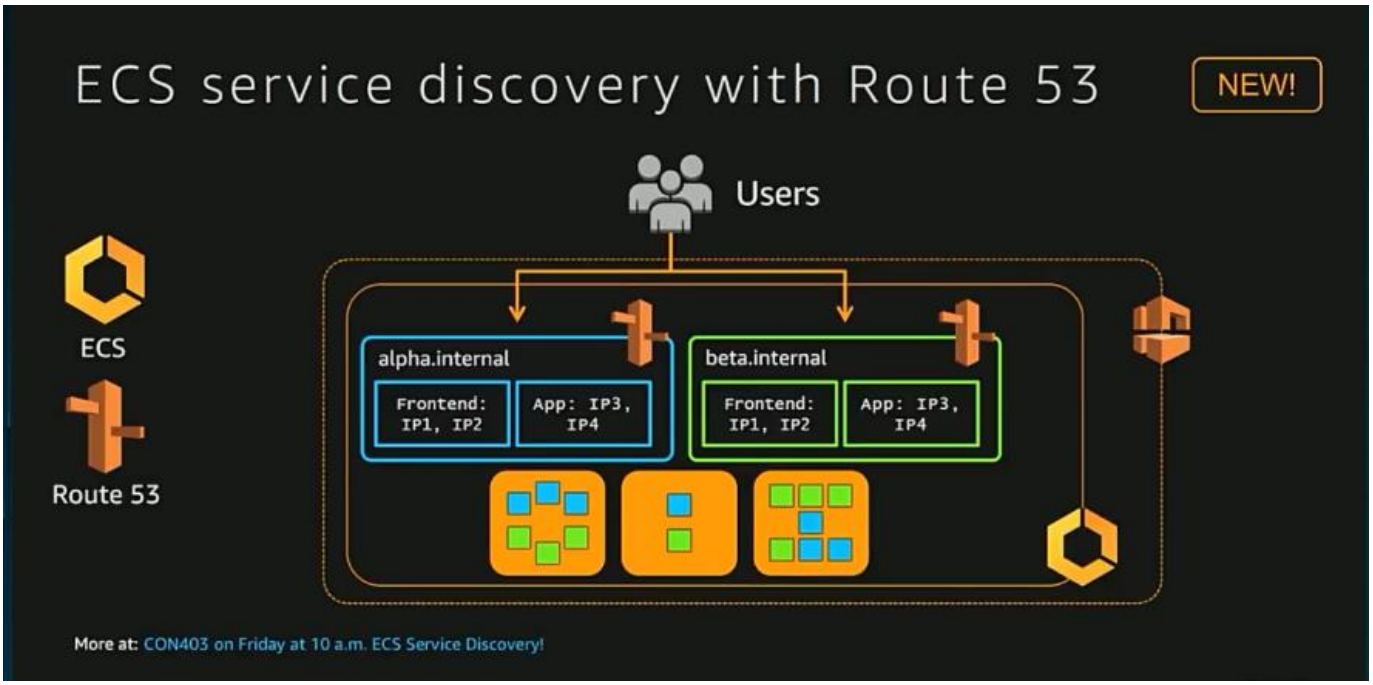
You can also put your jobs into different priority queues


Rolling deployments
Update service
Service A
Task def'n: 2
Desired count: 4
Max: 200%
minHealthy: 100%
Max: 100%
minHealthy: 75%
ECS


Rolling deployments
Update service
Service A
Task def'n: 2
Desired count: 4
Max: 200%
minHealthy: 100%
Max: 100%
minHealthy: 75%
ECS



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

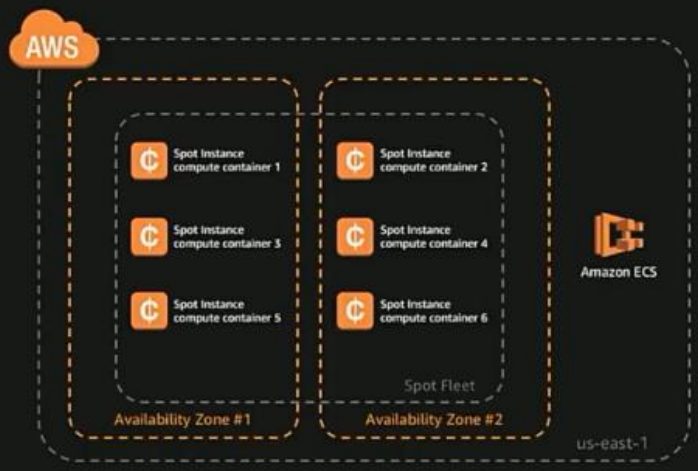


You can make services discover new versions of the other services they are interacting with. This uses the concept of registering your services with a namespace, this then allows you to have multiple versions of your services running in parallel at the same time. You clients can then do a DNS query to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

Scheduling compute capacity

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

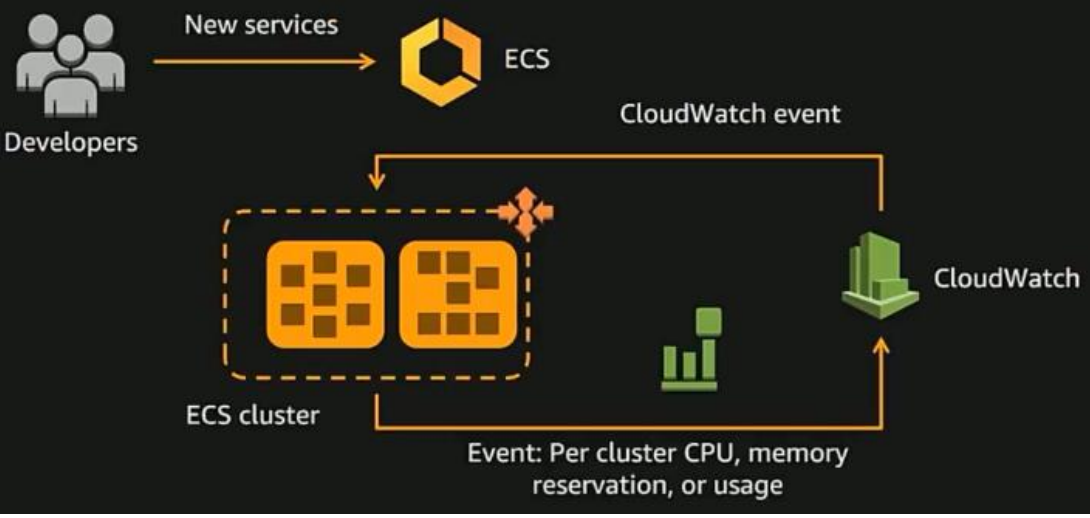
Compute with EC2 Spot Instances

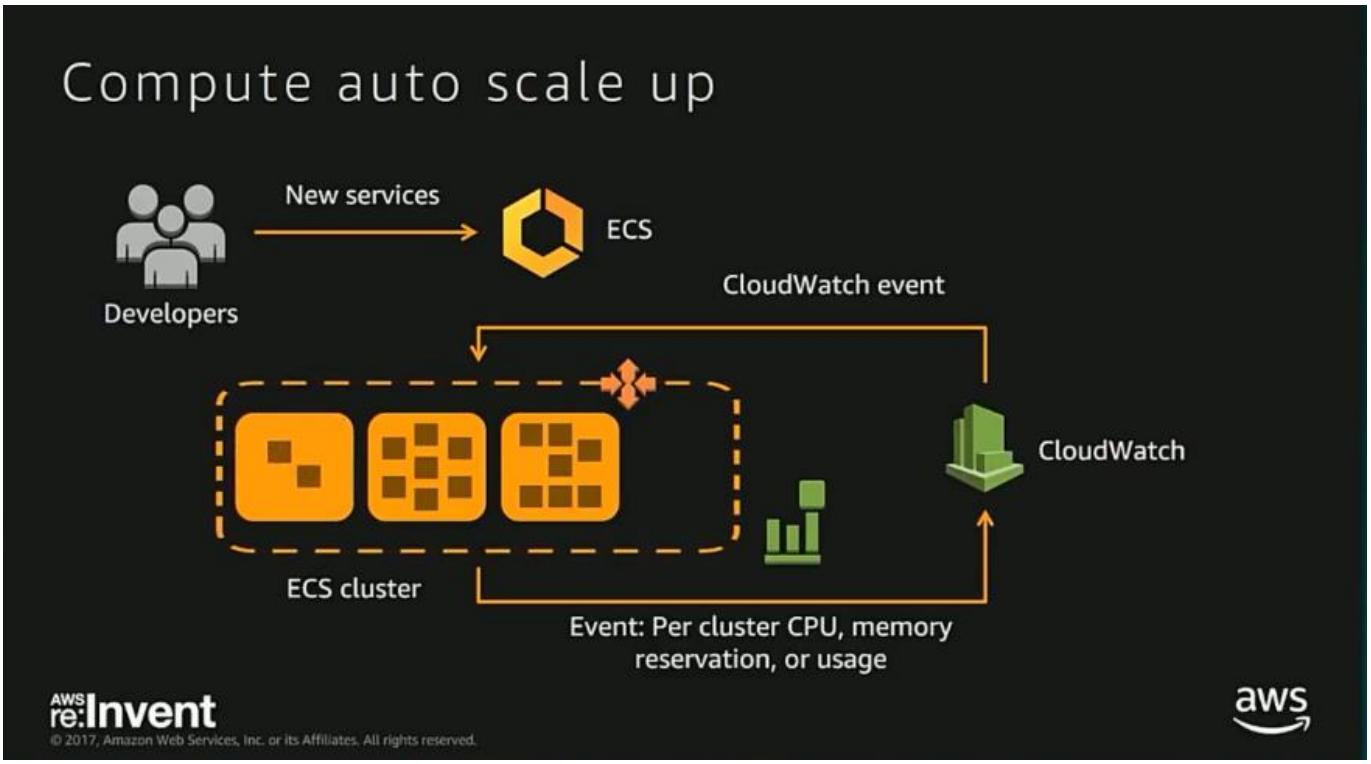


More at: https://aws.amazon.com/blogs/compute/powering-your-amazon-ecs-cluster-with-amazon-ec2-spot-instances/

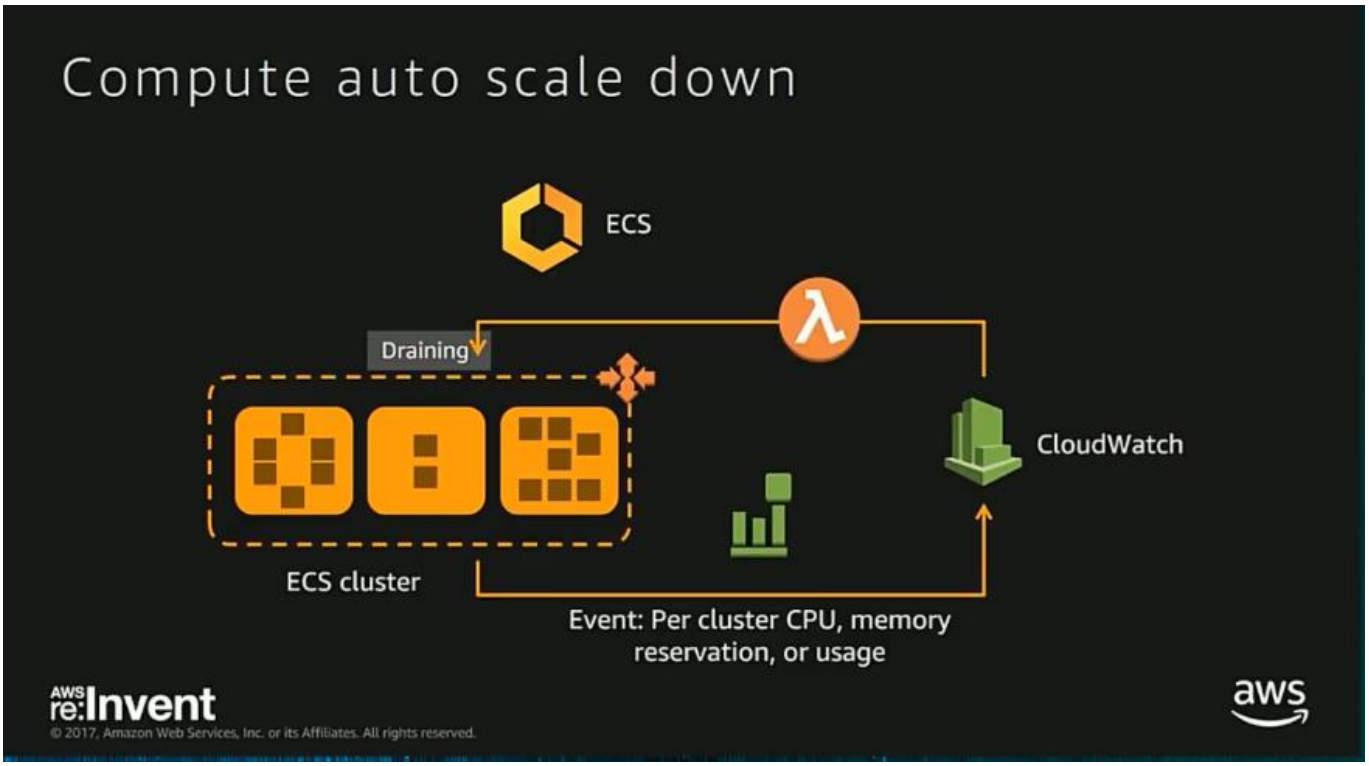
ECS is integrated with Spot Fleet that allows you to maintain a fixed number of CPU or instances at all times using bidding processes. The instances can then be configured as part of your spot fleet cluster that can be spread across different AZs.

Compute auto scale up





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



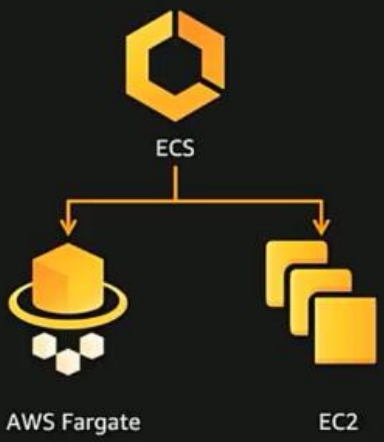
You can also have the ASG drain excess capacity based on utilization metric tied to a lambda function that will set the ‘draining’ flag on instances that will be deleted. This allows the ECS scheduler to stop placing tasks on the draining instances

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.
aws