Kubernetes on AWS:

Cost optimization and reducing operational effort

Eryan Ariobowo

28 July 2020

AWS Container Day at KubeCon

August 17, 2020 | 8:00AM - 4:00PM PDT

Register now

https://bit.ly/2BC0461

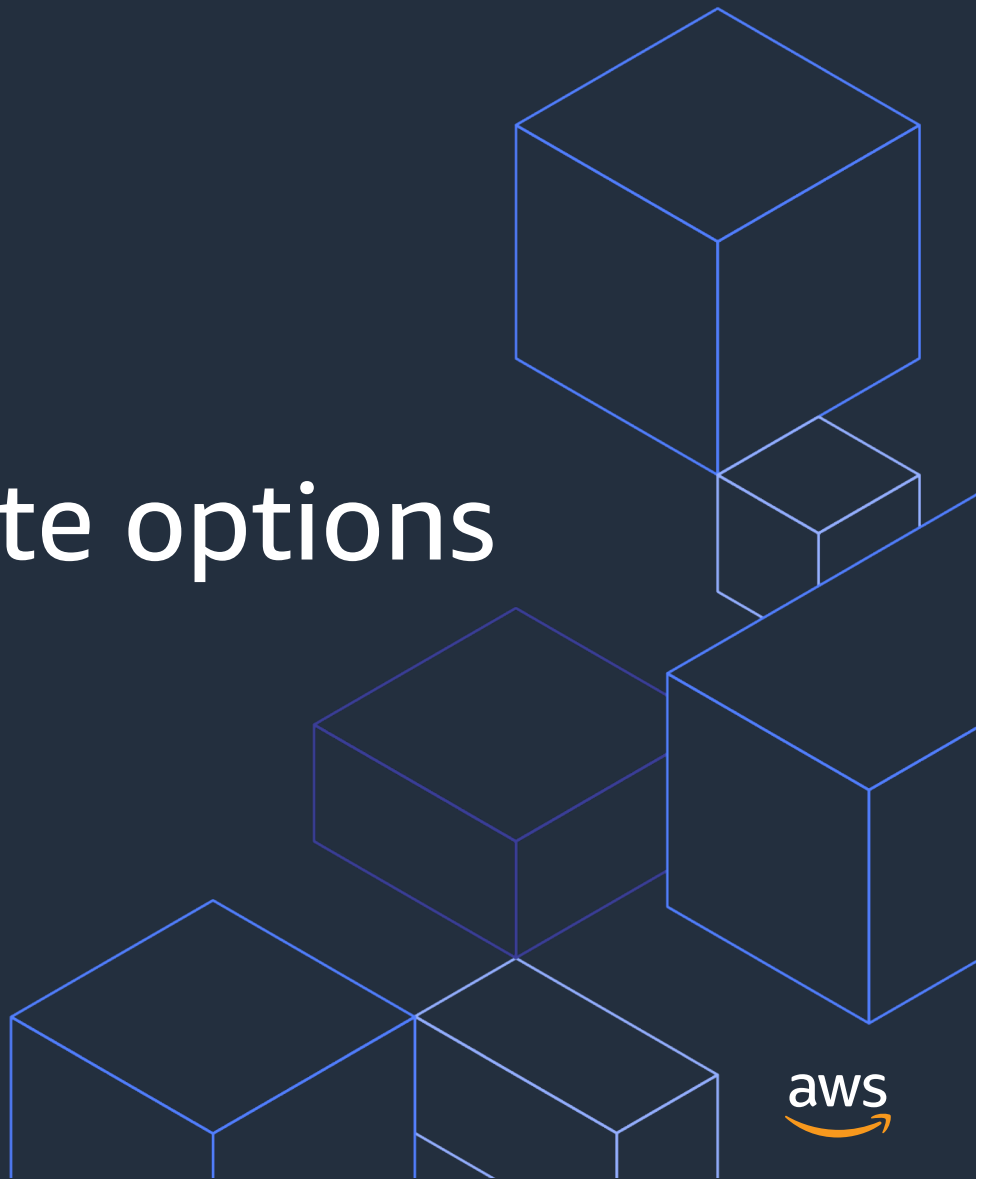
aws

Agenda

- Containers and Compute options
- Compute Cost and Performance
- Maximize Price/Performance
- Cost Optimization and Autoscaling
- Spot with Amazon **EKS**

aws

Containers and compute options



aws

Kubernetes on AWS and compute

Running K8s Options

Deployment, scheduling, scaling, and management of containerized applications

Self-managed Kubernetes on EC2

Amazon Elastic Kubernetes Service (Amazon EKS)

3rd Party-managed Kubernetes

Compute Engine

Where the containers run

Amazon EC2 Self-Managed Worker Nodes

AWS Fargate for EKS

Amazon EC2 Managed Worker Nodes on EKS

aws

Amazon Elastic Kubernetes Service

Upstream Kubernetes

Use standard Kubernetes APIs.
Works with community tools.

Highly available

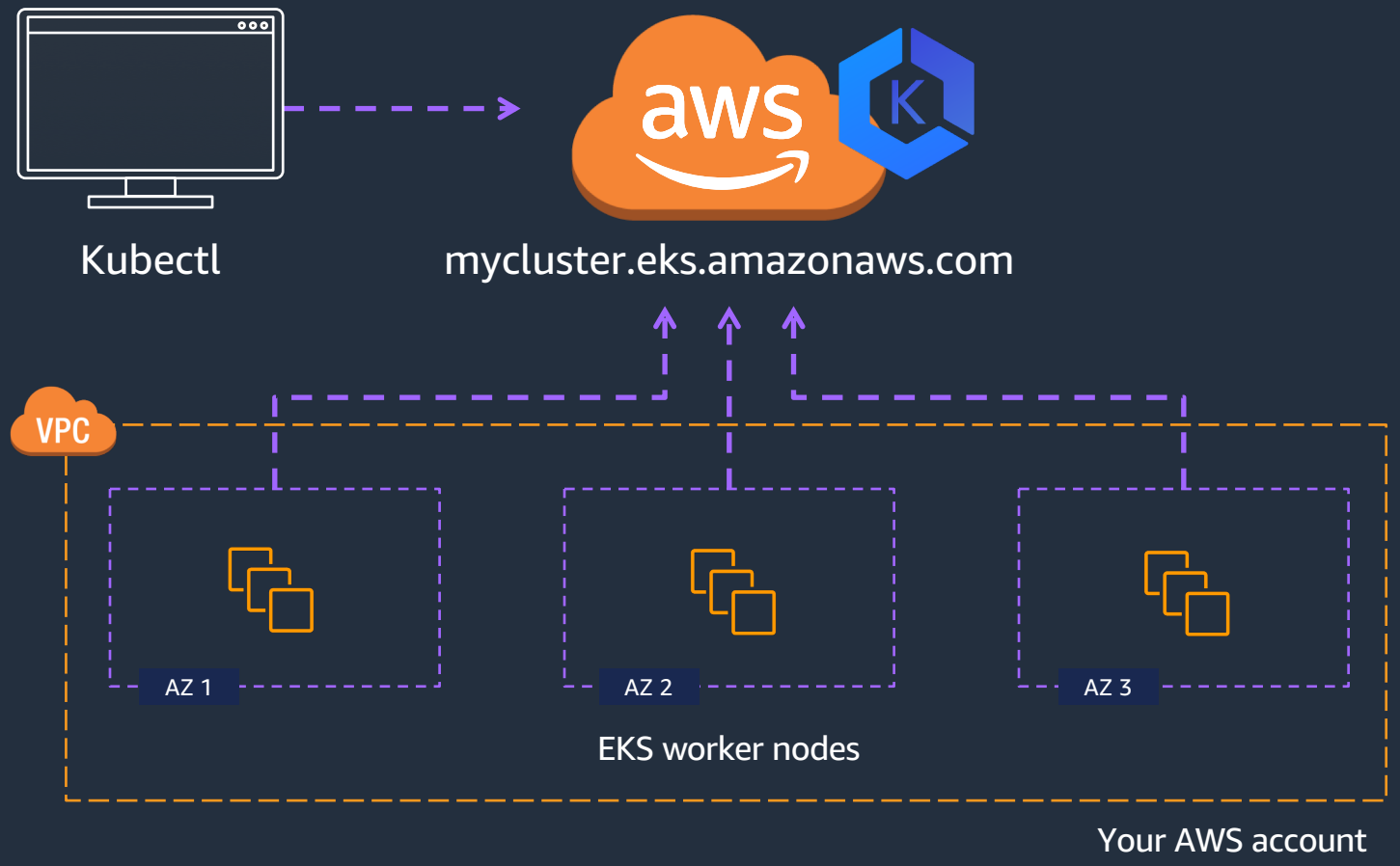
Built for production workloads, all clusters are highly available. Backed by a 99.9% SLA.

Integrated

with the AWS ecosystem: VPC Networking, Elastic Load Balancing, IAM Permissions, CloudWatch and more

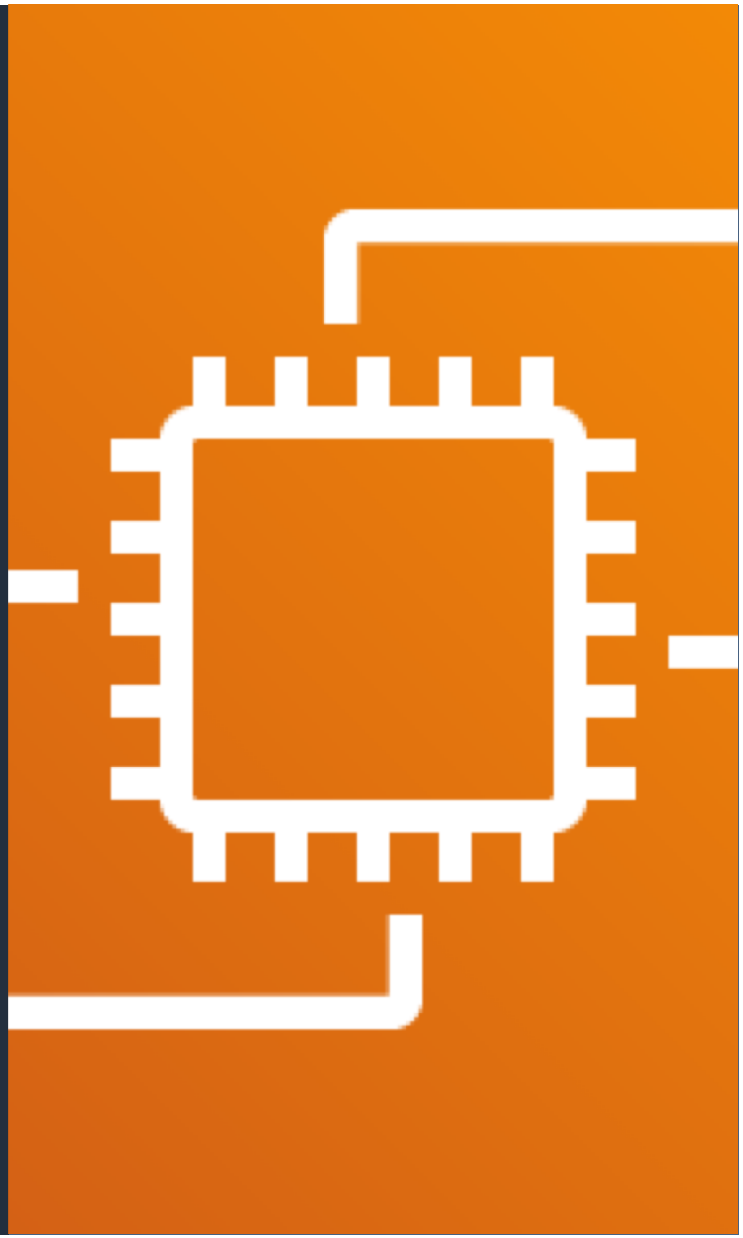
aws

Amazon Elastic Kubernetes Service

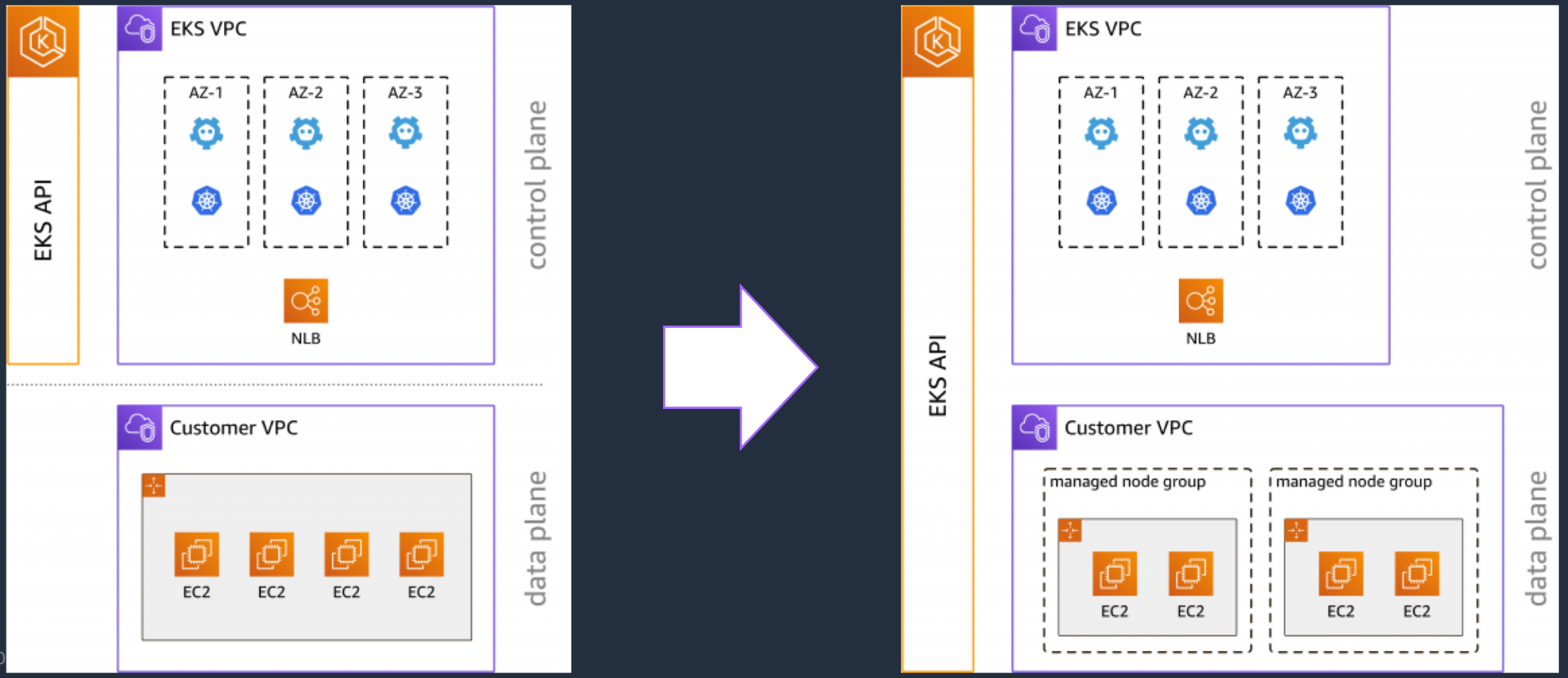


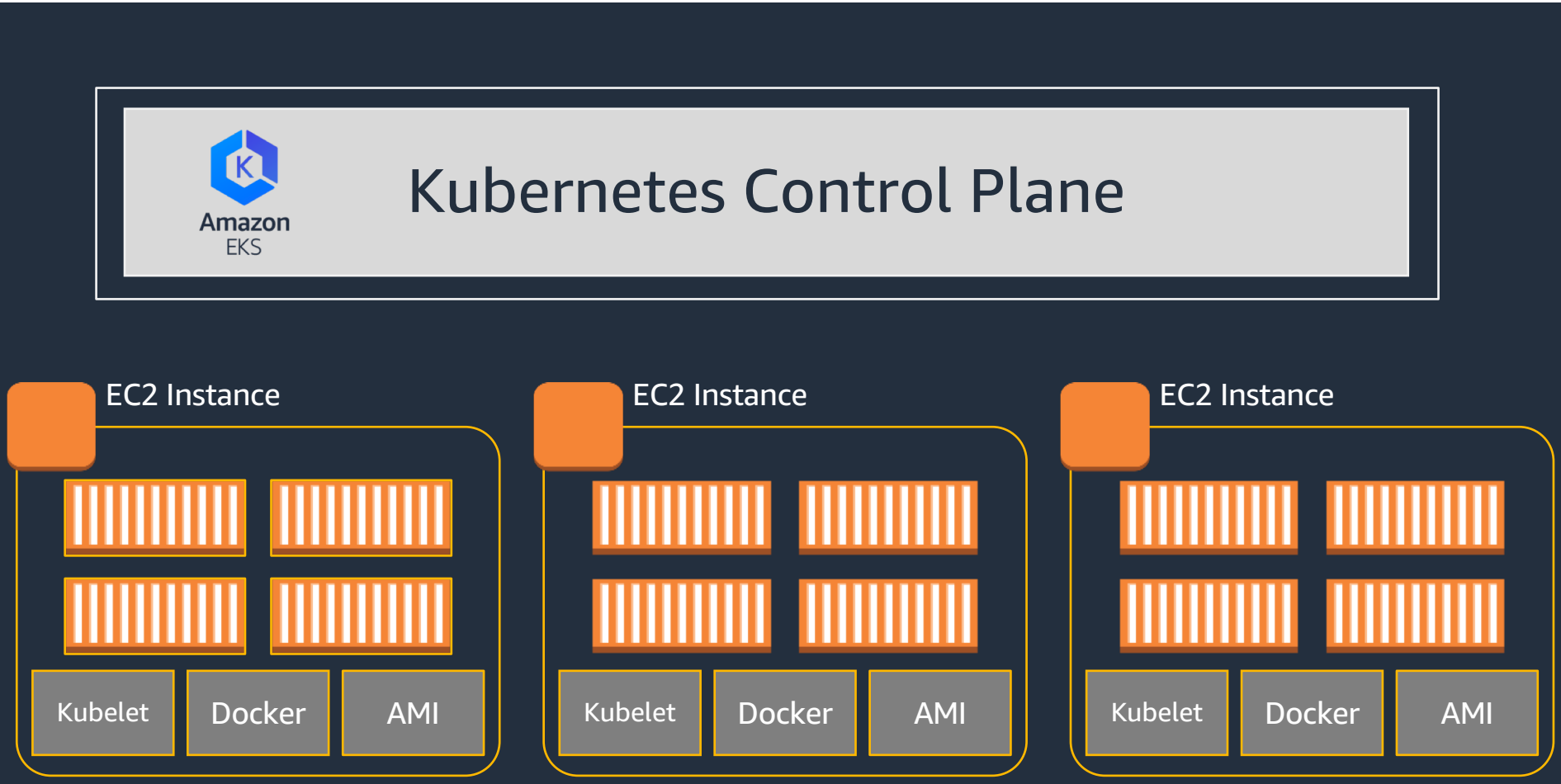
EC2 Mode – Customer Responsibilities

- Instance type and quantity to choose?
- Which OS to choose?
- Hardening the OS (e.g. against CIS benchmark) & Patching of the OS, Docker, kubelet etc.



EKS managed worker nodes





aws

Kubernetes Control Plane

aws

AWS Fargate

Managed by AWS

No EC2 Instances to provision, scale or manage

Elastic

Scale up & down seamlessly. Pay only for what you use

Integrated

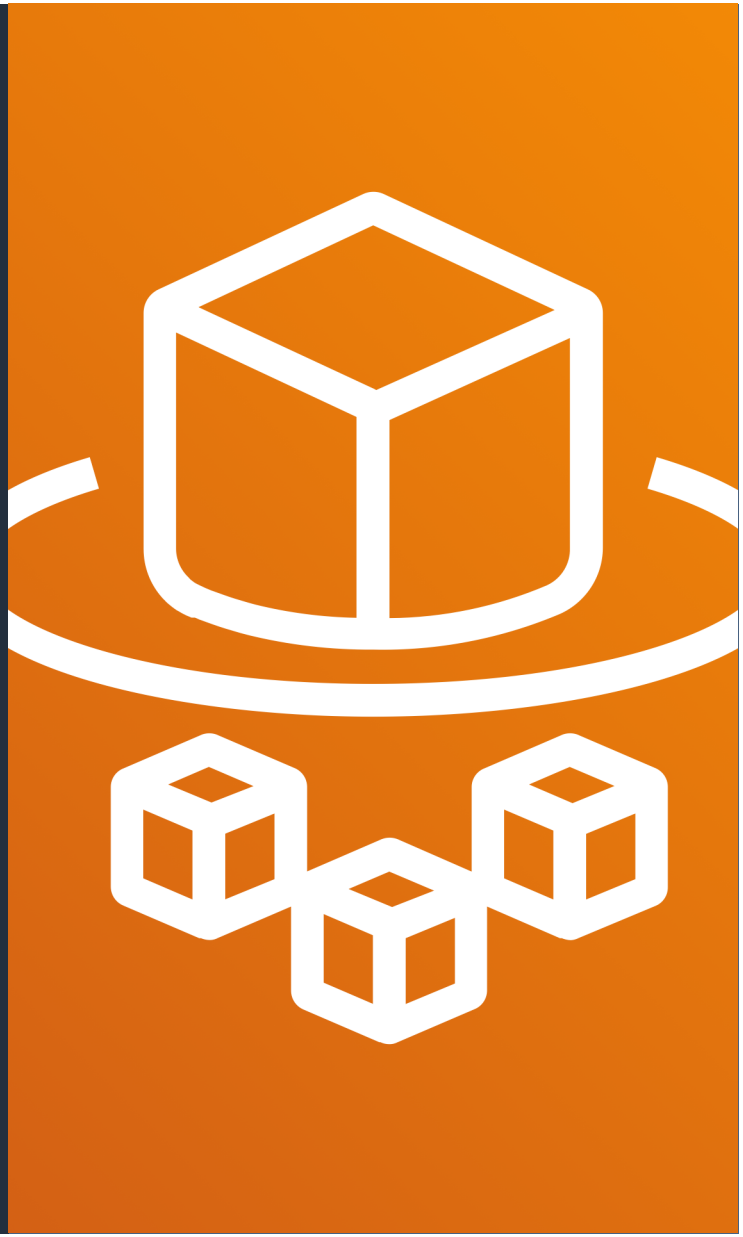
with the AWS ecosystem: VPC Networking, Elastic Load Balancing, IAM Permissions, CloudWatch and more. Run Kubernetes pods or ECS tasks.

aws

Security Benefits Of Fargate

We do more, you do less:

- Patching of the OS, Docker, ECS Agent, etc.
- Task isolation (via separate Clusters)
- Requires awsvpc network mode so ENI/SG per Task
- No runtime access for users (ssh or interactive Docker)



Cost Optimization

Cost

Performance

Cost optimization is about trading off cost and performance:
How can you achieve your performance goal at minimum cost?

aws

Amazon EC2 purchase options

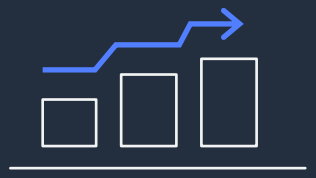
On-Demand

Pay-for-compute capacity **by the second** with no long-term commitments

Spiky workloads, to define needs

Savings Plans & Reserved Instances

Make a commitment and receive a **significant discount** off compute



Committed & steady-state usage

Spot Instances

Spare Amazon EC2 capacity at **savings of up to 90%** off On-Demand prices

Fault-tolerant, flexible, stateless workloads

aws

Fargate Purchase Options

Fargate

Pay for containers **per-second** with no long-term commitment

Capacity needs can change rapidly

Compute Savings Plan

Make a 1 or 3-year commitment and receive a **significant discount**

Baseline compute needs known in advance

New

Fargate Spot

Spare capacity with **savings up to 70%** off Fargate standard pricing

Fault-tolerant, flexible workloads

aws

Containers + Spot Instances

- Containers are often stateless, fault-tolerant, and a great fit for Spot Instances
- Deploy containerized workloads and easily manage clusters at any scale at a fraction of the cost with Spot Instances
- Spot instances can be used with Kubernetes to run any containerized workload

skyscanner

Skyscanner is a travel fare aggregator website and travel metasearch engine based in Edinburgh, Scotland

"We are currently tracking **74% saving** over all regions."

- Paul Gillespie,
Principal Architect/Tribe Lead

lyft yelp MapBox Expedia Caltech

aws

Amazon EC2 Spot and Fargate Spot

Amazon EC2 Spot

Spare EC2 Capacity

Save up to 90% over On-Demand

Can be *reclaimed* by Amazon EC2 (with two minute warning)

Follow EC2 Spot best practices – instance flexibility and use capacity optimized allocation strategy for ASGs.

AWS Fargate Spot

Spare compute Capacity

Save up to 70% over standard Fargate

Can be *reclaimed* (with two minute warning)

Automatic diversification

aws

The Simple Rules of Spot

70 - 90% off

Spot infrastructure

Is same as On-Demand and RIs

Spot pricing

Smooth, infrequent changes no spikes, more predictable

Interruptions

Only happen when OD needs capacity (**no bidding**)

Diversify

Choose different instance types, size and AZ in a single fleet

aws

EC2 Spot instance pools explained

C4	1a	1b	1c	On Demand
8XL	$0.50	$0.27	$0.29	$1.76
4XL	$0.21	$0.30	$0.16	$0.88
2XL	$0.08	$0.07	$0.08	$0.44
XL	$0.04	$0.05	$0.04	$0.22
L	$0.01	$0.01	$0.04	$0.11

- Each instance family
- Each instance size
- Each Availability Zone
- In every region
- Is a separate ***Spot pool***

aws

Fargate Spot Pricing

Fargate		Fargate Spot	
per vCPU per hour	**$0.04**048	per vCPU per hour	**$0. 012144**
per GB per hour	**$0.004**445	per GB per hour	**$0. 0013335**

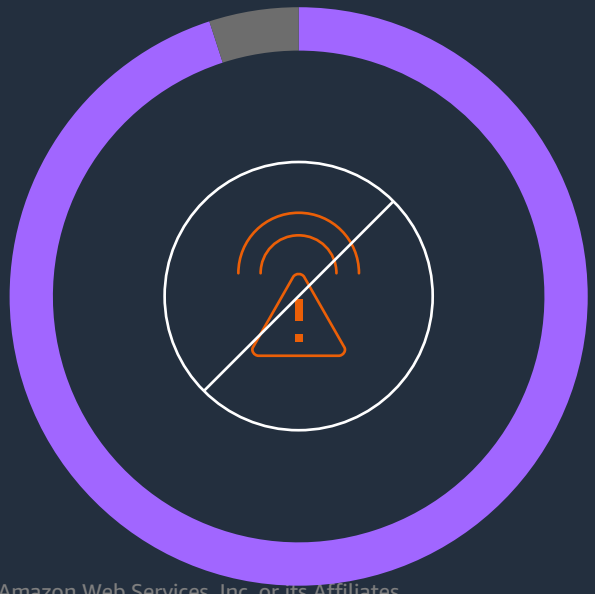
Supported Configurations	
CPU	Memory Values
0.25 vCPU	0.5GB, 1GB, and 2GB
0.5 vCPU	Min. 1GB and Max. 4GB, in 1GB increments
1 vCPU	Min. 2GB and Max. 8GB, in 1GB increments
2 vCPU	Min. 4GB and Max. 16GB, in 1GB increments
4 vCPU	Min. 8GB and Max. 30GB, in 1GB increments

aws

What about interruptions?

Minimal interruptions

Over **95%** of the instances were not interrupted in the last 3 months



Leverage Spot Instance Advisor to select the suitable instance types to your workload with the lowest interruption rates

Catch the interruption notifications by polling EC2 metadata or using Cloudwatch events and automate response to interruptions:

- ELB connection draining and graceful shutdown
- Checkpointing
- Cordon your node and drain containers

aws

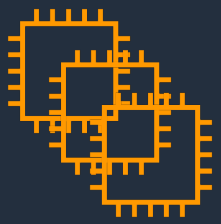
Compute Cost and Performance

aws

The Components of Compute Cost

=

(total compute cost)



x

x

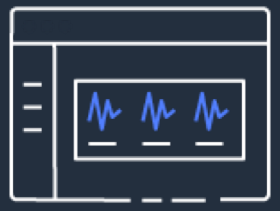
(# of compute units) (time in use) (cost per unit)

aws

What are Compute Units?

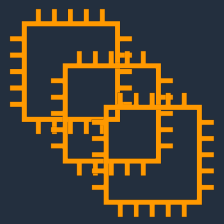
aws

The Components of Compute Performance



=

(total compute performance)



x

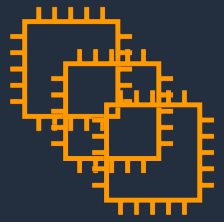
(# of compute units)

(performance per unit)

aws

Minimize Cost While Meeting Performance Goals

(1) Choose compute units with the best *price/performance* for your workload.



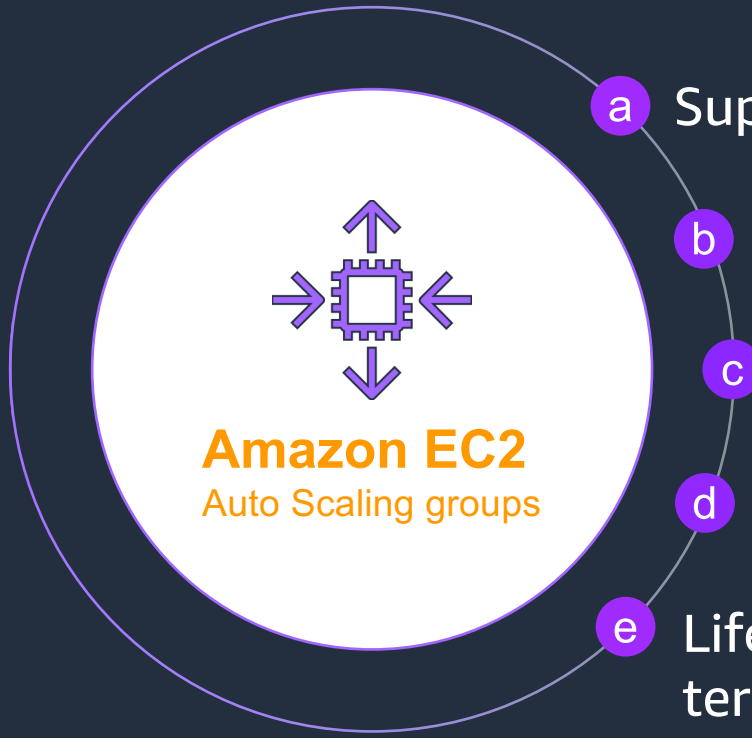
(2) Run the *right number* for only *as long as you need* them to meet performance goals.

aws

Cost Optimization and Autoscaling

aws

Amazon EC2 Auto Scaling groups



a. Support for mixed instance types & Purchase Options

b. Replacement of unhealthy & interrupted instances

c. Balanced capacity across availability zones

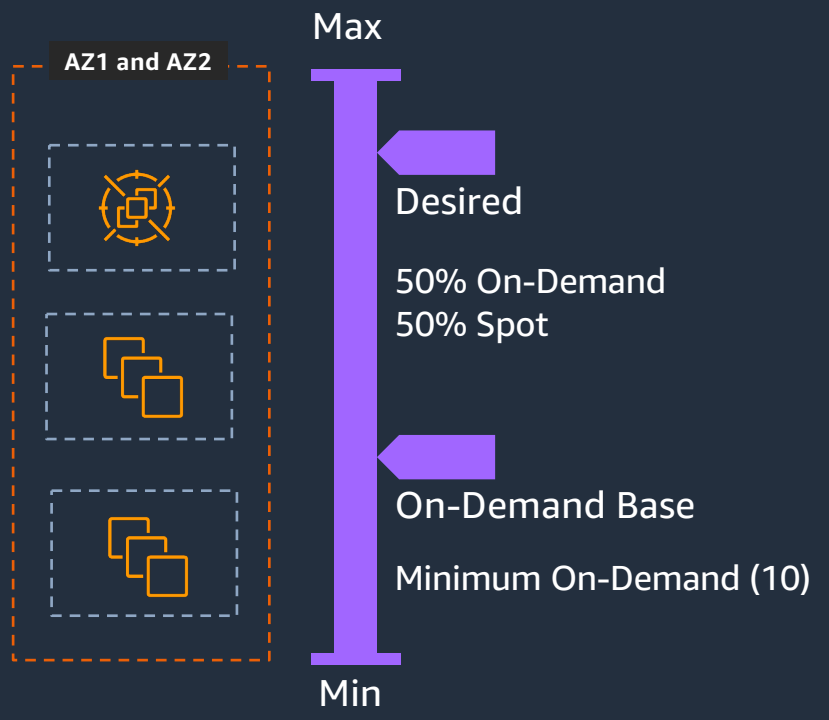
d. Elastic Load Balancer integration

e. Lifecycle hooks, termination policies, instance termination protection, scaling processes…

aws

API Parameters

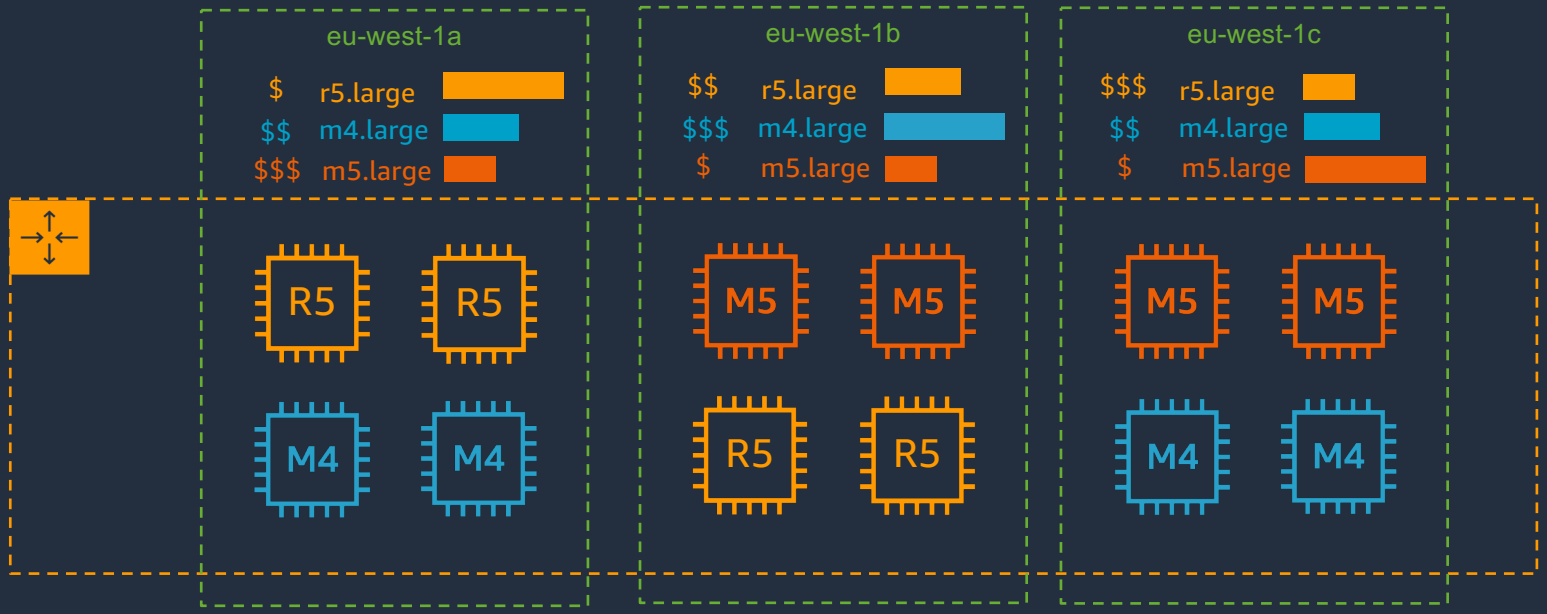
```
"MixedInstancesPolicy": {
    "LaunchTemplate": {
       "LaunchTemplateSpecification": {
         "LaunchTemplateName": ”MyLaunchTemplate"
       },
       "Overrides": [
         { "InstanceType": "c5.large" },
         { "InstanceType": "c4.large" }
       ]
     },
     "InstancesDistribution": {
       "OnDemandAllocationStrategy": "prioritized",
       "OnDemandBaseCapacity": 10,
       "OnDemandPercentageAboveBaseCapacity": 50,
       "SpotAllocationStrategy": ”capacity-optimized"
    }
}
```



aws

ASG SpotAllocationStrategy: lowest-price (across N pools)

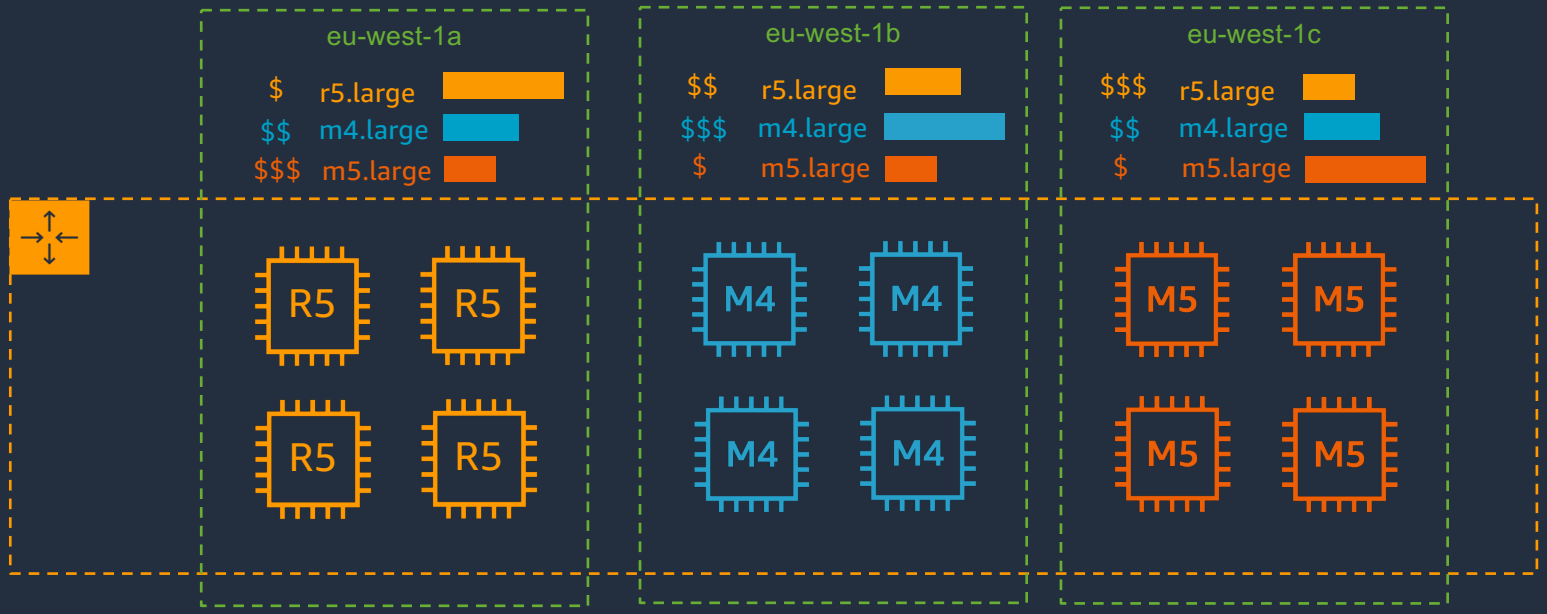
```
Desired capacity: 12        OnDemandBaseCapacity: 0        OnDemandPercentageAboveCapacity: 0
Overrides: ["r5.large", "m4.large", "m5.large"]        SpotAllocationStrategy: lowest-price        SpotInstancePools: 2
```



aws

ASG SpotAllocationStrategy: capacity-optimized

```
Desired capacity: 12        OnDemandBaseCapacity: 0        OnDemandPercentageAboveCapacity: 0
Overrides: ["r5.large", "m4.large", "m5.large"]        SpotAllocationStrategy: capacity-optimized
```



aws

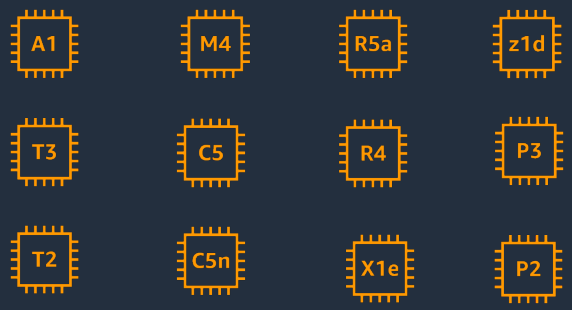
(1) Maximizing Price/Performance of Compute Units

EC2 Instances

Size task/pod vCPU and Memory

Choose Purchase Type

Choose Instance Type(s)



Fargate Tasks/Pods

Size task/pod vCPU and Memory

Choose Purchase Type

aws

(1) Maximizing price/performance with 100% Spot

Good fit for 100% Spot:

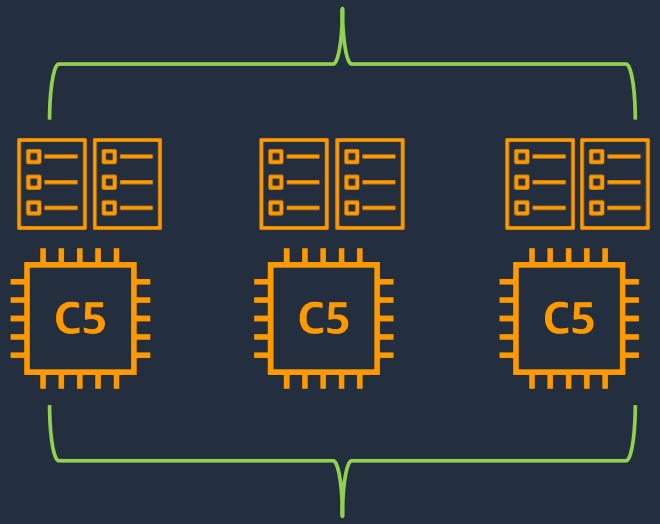
- ETL
- Batch processing
- Dev/test environments

aws

(2) Run the *right number* for only as long as needed

EKS/EC2

k8s Horizontal Pod Autoscaler



k8s Cluster Autoscaler

EKS/Fargate

k8s Horizontal Pod Autoscaler

aws

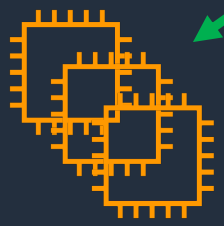
The Components of Compute Cost (revisited)

(total compute cost)

=

Number of pods/tasks over time

Purchase type of compute



(# of compute units)

x

(time in use)

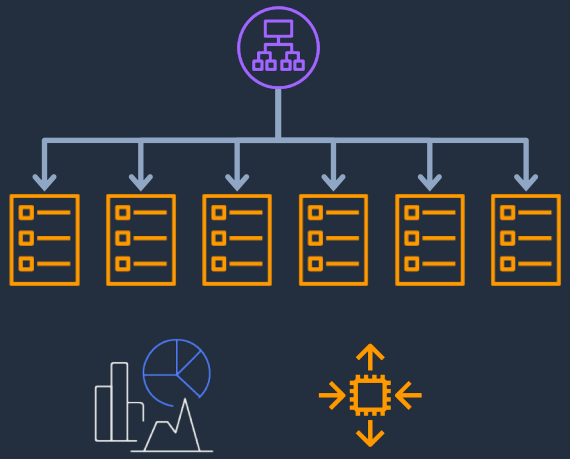
x

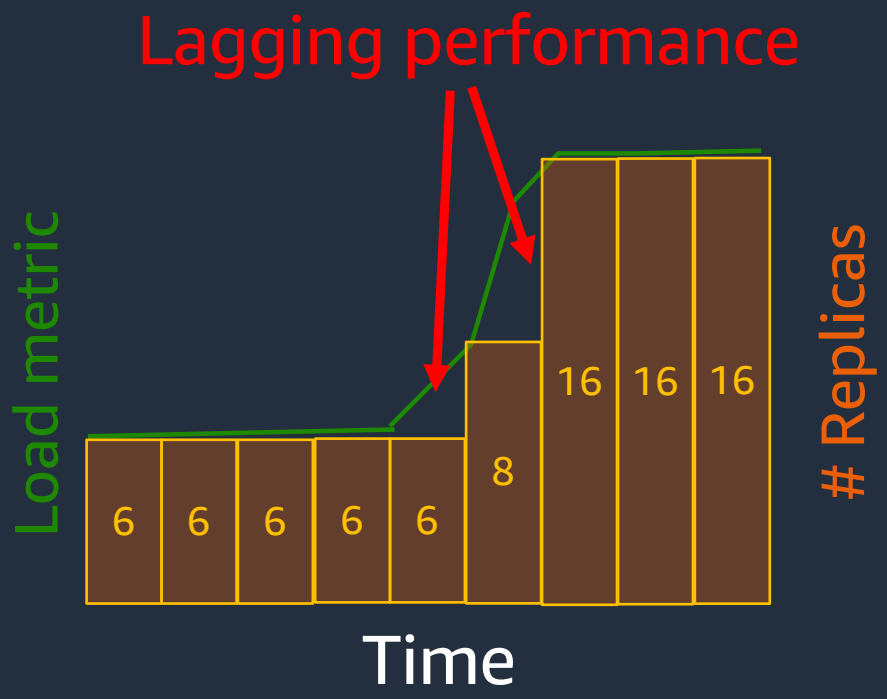
(cost per unit)

aws

(2) Run the *right number* for only as long as needed

Kubernetes Horizontal Pod Autoscaler

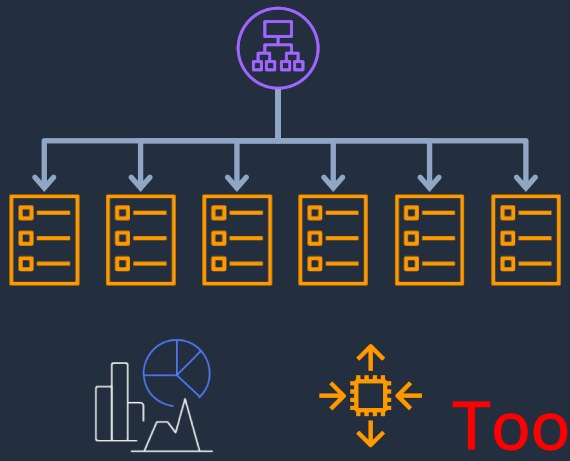




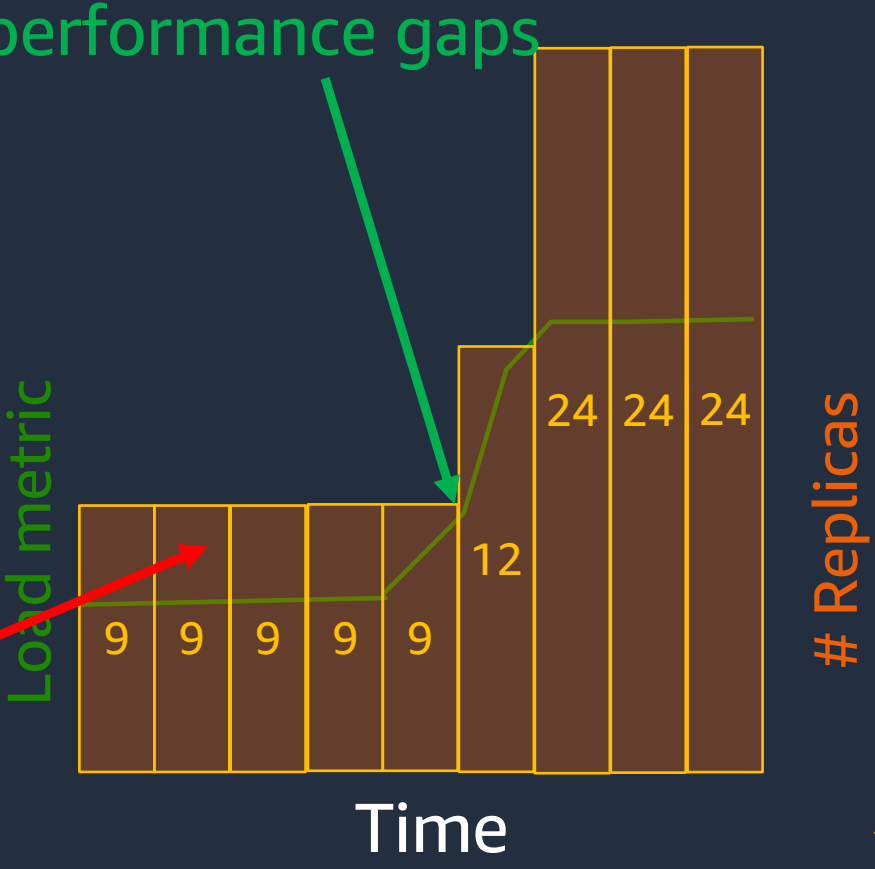
aws

(2) Run the *right number* for only as long as needed

Overprovision by 50%:
Reduce metric target value by 1/3



Too many, most of the time

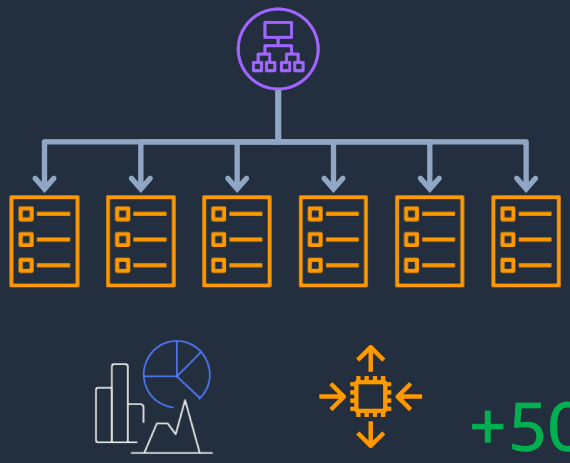


aws

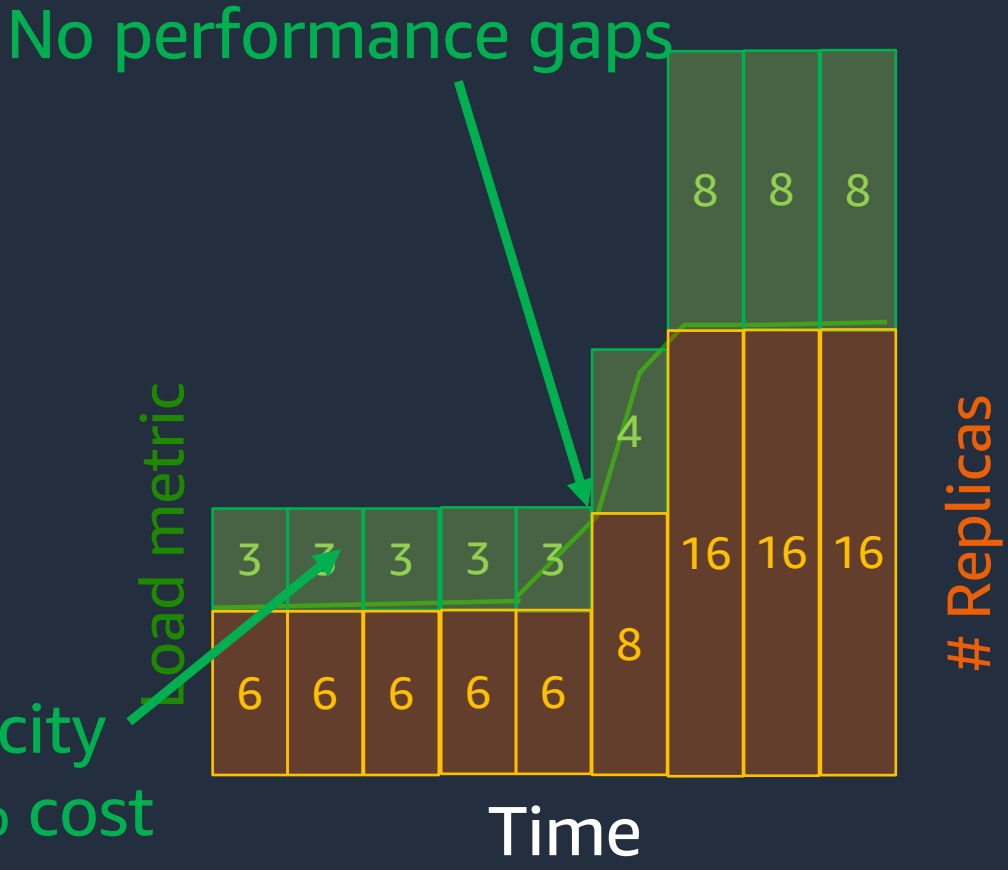
(2) Run the *right number* for only as long as needed

Overprovision by 50%:
Reduce metric target value by 1/3

Run 2/3 On-Demand, 1/3 on Spot



+50% capacity
for +5-10% cost



aws

Amazon EKS & EC2 Spot Instances

aws

Using EC2 Spot instances with Amazon EKS

1. Acquiring Capacity
2. Scaling Mechanisms: Horizontal Pod AutoScaler (HPA), Cluster Autorscaler (CA)
3. Interruption handling with a DaemonSet
4. Scheduling: Node Affinity, Taints and Tolerations

aws

eksctl – Adding a diversified node group

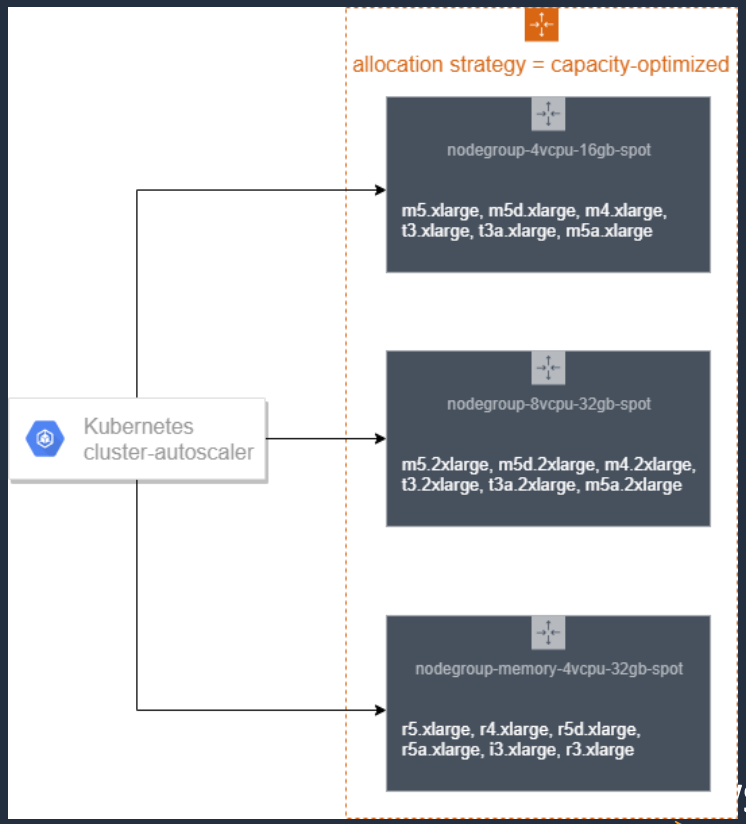
```yaml
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig
metadata:
    name: test-cluster
    region: us-west-2
nodeGroups:
    - name: dev-4vcpu-16gb-spot
      availabilityZones: ["us-west-2a","us-west-2b","us-west-2c"]
      minSize: 1
      maxSize: 100
      instancesDistribution:
        instanceTypes: ["m5.xlarge", "m5d.xlarge", "m4.xlarge","t3.xlarge","t2.xlarge"]
        onDemandBaseCapacity: 0
        onDemandPercentageAboveBaseCapacity: 0
        spotInstancePools: 4
      labels:
        lifecycle: Ec2Spot
        environment: dev
        costcenter: engineering
        project: default
      taints:
        spotInstance: "true:PreferNoSchedule"
```


aws

Auto Scaling the App & Cluster

CA Nodegroups are still expected to be homogeneous. Implement diversification !

```
command:
  - ./cluster-autoscaler
  - --v=4
  - --stderrthreshold=info
  - --cloud-provider=aws
  - --skip-nodes-with-local-storage=false
  - --nodes=0:100:nodegroup-dev-4vcpu-16gb-spot
  - --nodes=0:100:nodegroup-dev-8vcpu-32gb-spot
  - --nodes=0:100:nodegroup-dev-4vcpu-16gb-ondemand
  - --nodes=0:100:nodegroup-dev-8vcpu-32gb-ondemand
  - --expander=priority
```



AWS Interruptions Handler DaemonSet

https://github.com/aws/aws-node-termination-handler

```
log.Println("Kubernetes Spot Node Termination Handler has started successfully!")
waitForTermination(nthConfig)
drainer.Drain(nthConfig)
log.Printf("Node %q successfully drained.\n", nthConfig.NodeName)
```

https://github.com/aws/eks-charts/tree/master/stable/aws-node-termination-handler

```
helm repo add eks https://aws.github.io/eks-charts
helm install --name aws-node-termination-handler \
  --namespace kube-system eks/aws-node-termination-handler \
  --set nodeSelector.lifecycle=Ec2Spot
```


aws

Taints, Toleration & Affinity

```
requiredDuringSchedulingIgnoredDuringExecution:
  nodeSelectorTerms:
  - matchExpressions:
    - key: environment
      operator: In
      values:
      - dev
```

```
affinity:
  nodeAffinity:
    preferredDuringSchedulingIgnoredDuringExecution:
    - weight: 1
      preference:
        matchExpressions:
        - key: lifecycle
          operator: In
          values:
          - OnDemand
```

Multi-tenant cluster. Group affinity

Life-cycle affinity & Toleration

```
tolerations:
- key: "spotInstance"
  operator: "Equal"
  value: "true"
  effect: "PreferNoSchedule"
```


aws

Main Takeaways

- **Use Fargate to reduce operational effort**
- **Get discount, use RI & Saving Plans**
- **Right Sizing**
- **Use only when you need it**
- **Cluster Auto-scaling**
- **Use Spot Instances**

aws

Thank you!