Compute

EC2, ECS, Fargate, lambda

AWS Certified Solutions Architect– Professional (SAP-C01) - Study notes - Sep’2019

EC2

On-Demand vs Reserved vs Spot AWS EC2

One strategy to maintain a minimum level of guaranteed compute resources for your applications is to launch a core group of On-Demand Instances, and supplement them with Spot Instances when the opportunity arises. Another strategy is to launch Spot Instances with a specified duration (also known as Spot blocks), which are designed not to be interrupted and will run continuously for the duration you select.

- **On-Demand Instances** – Pay, by the second, for the instances that you launch.
- **Reserved Instances** – Purchase, at a significant discount, instances that are always available, for a term from one to three years. *Reserved Instances are not physical instances, but rather a billing discount applied to the use of On-Demand Instances in your account.*
- **Scheduled Instances** – Purchase instances that are always available on the specified recurring schedule
- **Spot Instances** – Request unused EC2 instances, which can lower your Amazon EC2 costs significantly.
- **Dedicated Hosts** – Pay for a physical host that is fully dedicated to running your instances, and bring your existing per-socket, per-core, or per-VM software licenses to reduce costs.
- **Dedicated Instances** – Pay, by the hour, for instances that run on single-tenant hardware.
- **Capacity Reservations** – Reserve capacity for your EC2 instances in a specific AZ for any duration.

Option	**Discount**	**Description**
On-Demand	**0%**	There's no commitment from you. You pay the most with this option.
Reserved	**40%-60%**	1-year or 3-year commitment from you. You save money from that commitment.
Spot	**60%-90%**	Ridiculously inexpensive because there's no commitment from the AWS side.

	Spot Instances	**On-Demand Instances**
Launch time	Can only be launched immediately if the Spot Request is active and capacity is available.	Can only be launched immediately if you make a manual launch request and capacity is available.
Available capacity	If capacity is not available, the Spot Request continues to automatically make the launch request until capacity becomes available.	If capacity is not available when you make a launch request, you get an insufficient capacity error (ICE).
Hourly price	The hourly price for Spot Instances varies based on demand.	The hourly price is static.
Instance interruption	You can't stop and start an EBS-backed Spot Instance; only the EC2 Spot service can do this. The EC2 Spot service can interrupt an individual Spot Instance if capacity is no longer available, the Spot price exceeds your maximum price, or demand for Spot Instances increases.	You determine when an On-Demand Instance is interrupted (stopped or terminated).

ENI (Elastic Network Interface)

An elastic network interface is a logical networking component in a VPC that represents a virtual network card.

A network interface can include the following attributes:

- A primary private IPv4 address from the IPv4 address range of your VPC
- One or more secondary private IPv4 addresses from the IPv4 address range of your VPC
- One Elastic IP address (IPv4) per private IPv4 address
- One public IPv4 address
- One or more IPv6 addresses
- One or more security groups
- A MAC address
- A source/destination check flag
- A description

You can create a network interface, attach it to an instance, detach it from an instance, and attach it to another instance. The attributes of a network interface follow it as it's attached or detached from an instance and reattached to another instance. When you move a network interface from one instance to another, network traffic is redirected to the new instance. You can also modify the attributes of your network interface, including changing its security groups and managing its IP addresses.

Every instance in a VPC has a default network interface, called the *primary network interface* (eth0). You cannot detach a primary network interface from an instance. You can create and attach additional network interfaces. The maximum number of network interfaces that you can use varies by instance type.

Best Practices for Configuring Network Interfaces

- You can attach a network interface to an instance when it's running (hot attach), when it's stopped (warm attach), or when the instance is being launched (cold attach).

- You can detach secondary network interfaces when the instance is running or stopped. However, you can't detach the primary network interface (eth0).

- You can move a network interface from one instance to another, if the instances are in the same Availability Zone and VPC but in different subnets.

- Launching an Amazon Linux or Windows Server instance with multiple network interfaces automatically configures interfaces, private IPv4 addresses, and route tables on the operating system of the instance.

- A warm or hot attach of an additional network interface may require you to manually bring up the second interface, configure the private IPv4 address, and modify the route table accordingly. Instances running Amazon Linux or Windows Server automatically recognize the warm or hot attach and configure themselves.

- Attaching another network interface to an instance (for example, a NIC teaming configuration) cannot be used as a method to increase or double the network bandwidth to or from the dual-homed instance.

- If you attach two or more network interfaces from the same subnet to an instance, you may encounter networking issues such as asymmetric routing. If possible, use a secondary private IPv4 address on the primary network interface instead.

Scenarios for Network Interfaces: Attaching multiple network interfaces to an instance is useful when you want to:

- Create a management network.
- Use network and security appliances in your VPC.
- Create dual-homed instances with workloads/roles on distinct subnets.
- Create a low-budget, high-availability solution.

Reserved Instances

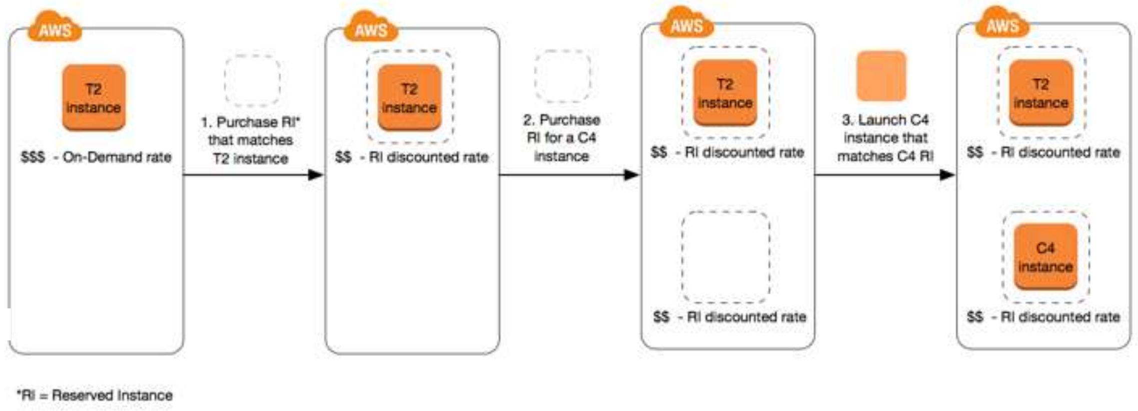
https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-reserved-instances.html

Reserved Instances provide you with a significant discount compared to On-Demand Instance pricing. Reserved Instances are not physical instances, but rather a billing discount applied to the use of On-Demand Instances in your account. These On-Demand Instances must match certain attributes in order to benefit from the billing discount. The following diagram shows a basic overview of purchasing and using Reserved Instances.

In this scenario, you have a running On-Demand Instance (T2) in your account, for which you're currently paying On-Demand rates. You purchase a Reserved Instance that matches the attributes of your running instance, and the billing benefit is immediately applied. Next, you purchase a Reserved Instance for a C4 instance. You do not have any running instances in your account that match the attributes of this Reserved Instance. In the final step, you launch an instance that matches the attributes of the C4 Reserved Instance, and the billing benefit is immediately applied.

Key Variables That Determine Reserved Instance Pricing:

1. **Instance type**: For example, m4.large. This is composed of the instance family (m4) and the instance size (large).
2. **Scope**: Whether the Reserved Instance applies to a Region (regional Reserved Instance) or specific AZ (zonal Reserved Instance).
3. **Tenancy**: Whether your instance runs on shared (default) or single-tenant (dedicated) hardware.
4. **Platform**: The operating system; for example, Windows or Linux/Unix.



Reserved instances - Standard vs. Convertible Offering Classes

https://docs.aws.amazon.com/whitepapers/latest/cost-optimization-reservation-models/standard-vs.-convertible-offering-classes.html

Standard Reserved Instance	Convertible Reserved Instance
One-year to three-year term	One-year to three-year term
Enables you to *modify* AZ, scope, network platform, and instance size (within the same instance type) of your RI	Enables you to *exchange* one or more Convertible Reserved Instances for another Convertible Reserved Instance with new attributes. These attributes include instance family, instance type, platform, scope, and tenancy, if the exchange results in the creation of a RI of equal or greater value.
Can be sold in the RI Marketplace.	Cannot be sold in the Reserved Instance Marketplace.

Standard RIs typically provide the highest discount levels. One-year Standard RIs provide a similar discount to three-year Convertible RIs.

Convertible Reserved Instances are useful when:

- Purchasing RIs in the payer account instead of a subaccount. You can more easily modify Convertible RIs to meet changing needs across your organization.
- Workloads are likely to change. In this case, a Convertible RI enables you to adapt as needs evolve while still obtaining discounts and capacity reservations.
- You want to hedge against possible future price drops.
- You don’t want to ask teams to do capacity planning or forecasting.
- You expect compute usage to remain at the committed amount over the commitment period.

Using EC2 Spot instances

Unlike Reserved Instances, Spot Instances do not require an upfront commitment. However, because Spot Instances can be terminated if the Spot price exceeds your maximum price or if no capacity is available for the instance type you've specified, they are best for flexible workloads.

Workloads that constantly save data to persistent storage—including S3, EBS, EFS, DynamoDB, or RDS—can work effectively with Spot Instances.

Spot Instances are typically used to supplement On-Demand Instances, where appropriate, and are not meant to handle 100% of your workload. However, you can use all Spot Instances for any stateless, non-production application, such as development and test servers, where occasional downtime is acceptable. They are not a good choice for sensitive workloads or databases.

The Spot price is determined by long-term trends in supply and demand for EC2 spare capacity. You pay the Spot price that's in effect at the beginning of each instance-hour for your running instance, billed to the nearest second.

With Spot Instances, you never pay more than the maximum price you specify. If the Spot price exceeds your maximum price for a given instance or if capacity is no longer available, your instance will automatically be terminated (or be stopped/hibernated, if you opt for this behavior on persistent request).

Spot offers three features to help you better track and control when Spot Instances run and terminate (or stop/hibernate).

- **Termination notices** – issued two minutes prior to interruption.
- **Persistent requests** – You can opt to set your request to remain open so that a new instance will be launched in its place when the instance is interrupted. You can also have your Amazon EBS-backed instance stopped upon interruption and restarted when Spot has capacity at your preferred price.
- **Block durations** – If you need to execute workloads continuously for 1–6 hours

Using EC2 Spot instances in an Auto scaling group

https://aws.amazon.com/about-aws/whats-new/2018/11/scale-instances-across-purchase-options-in-a-single-ASG/

EC2 Auto Scaling now lets you provision and automatically scale instances across purchase options, AZ, and instance families in a single ASG, to optimize scale, performance, and cost. Now you can include Spot Instances with On-Demand and RIs in a single ASG, to save up to 90% on compute.

Powered by EC2 Fleet, you can now create an ASG by defining which EC2 instance types work for you and how much of the desired capacity should be filled using On-Demand, RI and Spot purchase options. Auto Scaling continues to optimize and maintain the mix as and when the ASG scales out or scales back, simplifying capacity provisioning and cost optimization with automatic scaling across instances and purchase options. Auto Scaling also continues to provide lifecycle hooks, instance health checks and scheduled scaling to automate capacity management.

You can specify what percentage of your ASG capacity should be fulfilled by On-Demand instances or RIs, and what percentage with Spot Instances.

You can also indicate instances types or instances with specific amount of RAM or vCPU in the ASG configurations.

EC2 Auto Scaling then provisions the lowest price combination of instances to meet the desired capacity based on these preferences.

Allocation Strategy for Spot Instances:

lowestPrice
The Spot Instances come from the pool with the lowest price. This is the default strategy.

diversified
The Spot Instances are distributed across all pools.

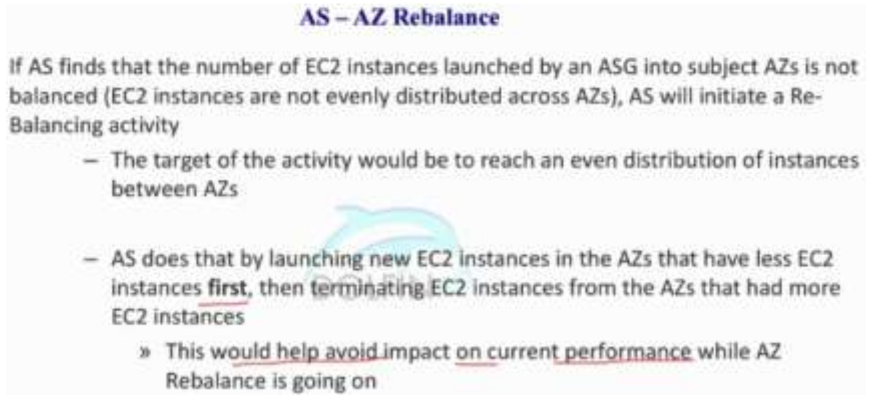
capacityOptimized
The Spot Instances come from the pool with optimal capacity for the number of instances that are launching.

InstancePoolsToUseCount
The Spot Instances are distributed across the number of Spot pools that you specify. This parameter is valid only when used in combination with lowestPrice.

Choosing an Appropriate Allocation Strategy

If your fleet is small or runs for a short time, the probability that your Spot Instances may be interrupted is low, even with all the instances in a single Spot Instance pool. Therefore, the lowestPrice strategy is likely to meet your needs while providing the lowest cost.
If your fleet is large or runs for a long time, you can improve the availability of your fleet by distributing the Spot Instances across multiple pools.
To create a cheap and diversified fleet, use the lowestPrice strategy in combination with InstancePoolsToUseCount.
If your fleet runs workloads that may have a higher cost of interruption associated with restarting work and checkpointing, then use the capacityOptimized strategy.

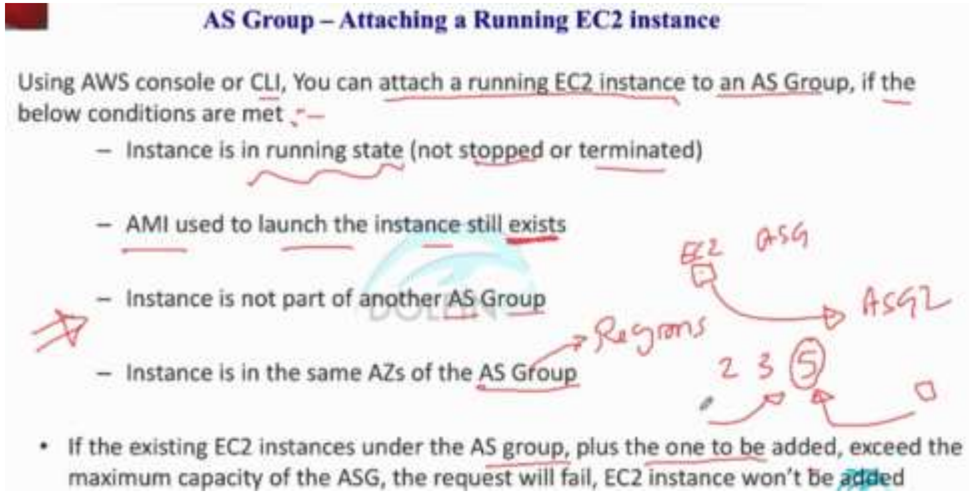
EC2 Auto-scaling



AS – AZ Rebalance

If AS finds that the number of EC2 instances launched by an ASG into subject AZs is not balanced (EC2 instances are not evenly distributed across AZs), AS will initiate a Re-Balancing activity

- The target of the activity would be to reach an even distribution of instances between AZs
- AS does that by launching new EC2 instances in the AZs that have less EC2 instances **first**, then terminating EC2 instances from the AZs that had more EC2 instances
  - » This would help avoid impact on current performance while AZ Rebalance is going on



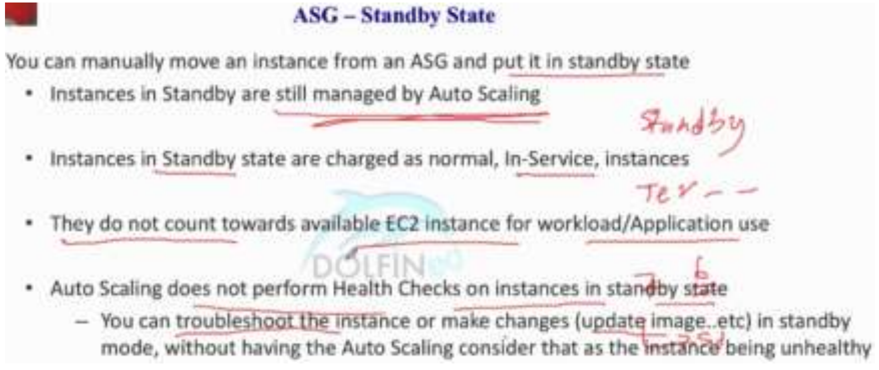
AS Group – Attaching a Running EC2 instance

Using AWS console or CLI, You can attach a running EC2 instance to an AS Group, if the below conditions are met

- Instance is in running state (not stopped or terminated)
- AMI used to launch the instance still exists
- Instance is not part of another AS Group
- Instance is in the same AZs of the AS Group

EC2
ASG
ASG2
Regions
2 3 5

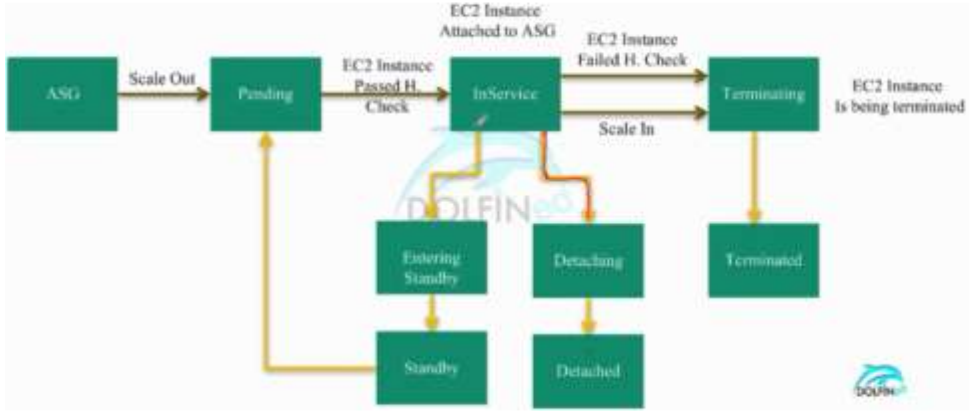
- If the existing EC2 instances under the AS group, plus the one to be added, exceed the maximum capacity of the ASG, the request will fail, EC2 instance won't be added



ASG – Standby State

You can manually move an instance from an ASG and put it in standby state

- Instances in Standby are still managed by Auto Scaling
- Instances in Standby state are charged as normal, In-Service, instances
- They do not count towards available EC2 instance for workload/Application use
- Auto Scaling does not perform Health Checks on instances in standby state
  - You can troubleshoot the instance or make changes (update image..etc) in standby mode, without having the Auto Scaling consider that as the instance being unhealthy



ASG
Scale Out
Pending
EC2 Instance Passed H. Check
InService
EC2 Instance Attached to ASG
EC2 Instance Failed H. Check
Scale In
Terminating
EC2 Instance Is being terminated
Entering Standby
Detaching
Terminated
Standby
Detached

EC2 Auto-scaling - EC2 Status + ELB Health checks

The default health checks for an Auto Scaling group are **EC2 status checks only**.

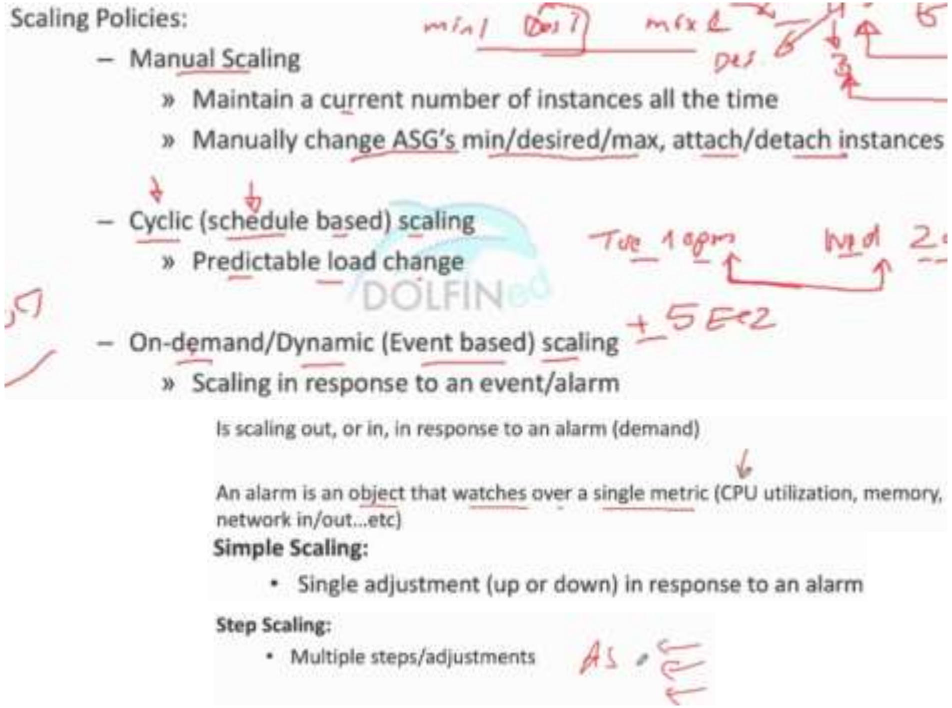
If the instance is in any state other than running or if the system status is impaired, Amazon EC2 Auto Scaling considers the instance to be unhealthy and launches a replacement instance. This includes when the instance has any of the following states:

- stopping
- stopped
- terminating
- terminated

However, you can optionally configure the Auto Scaling group to use ELB health checks. This ensures that the group can determine an instance's health based on additional tests provided by the load balancer. The load balancer periodically sends pings, attempts connections, or sends requests to test the EC2 instances. These tests are called health checks.

EC2 Auto Scaling will now determine the health status of the instances by checking both the **EC2 status checks** and the **ELB health checks**.

EC2 Auto-scaling - Scaling policies



Scaling Policies:

- Manual Scaling
  - » Maintain a current number of instances all the time
  - » Manually change ASG's min/desired/max, attach/detach instances
- Cyclic (schedule based) scaling
  - » Predictable load change
- On-demand/Dynamic (Event based) scaling
  - » Scaling in response to an event/alarm

Is scaling out, or in, in response to an alarm (demand)

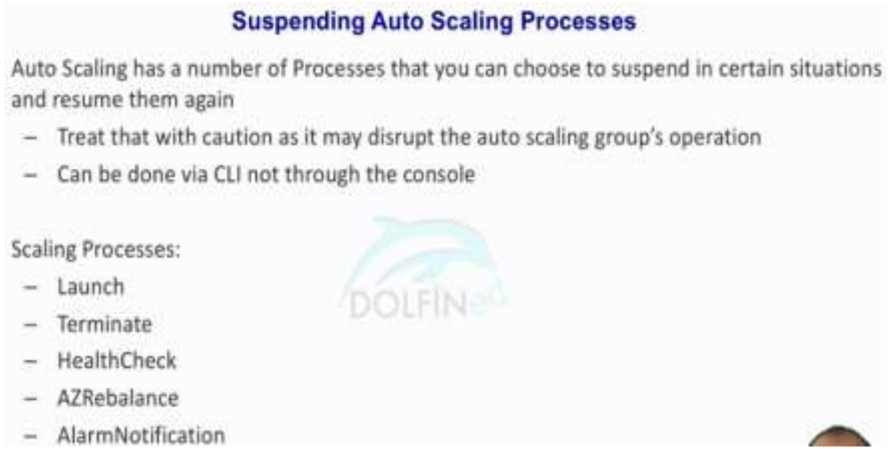
An alarm is an object that watches over a single metric (CPU utilization, memory, network in/out...etc)

Simple Scaling:

- Single adjustment (up or down) in response to an alarm

Step Scaling:

- Multiple steps/adjustments



Suspending Auto Scaling Processes

Auto Scaling has a number of Processes that you can choose to suspend in certain situations and resume them again

- Treat that with caution as it may disrupt the auto scaling group's operation
- Can be done via CLI not through the console

Scaling Processes:

- Launch
- Terminate
- HealthCheck
- AZRebalance
- AlarmNotification

EC2 Auto-scaling - AZRebalance

After certain actions occur, your Auto Scaling group can become unbalanced between Availability Zones. Amazon EC2 Auto Scaling compensates by rebalancing the Availability Zones. The following actions can lead to rebalancing activity:

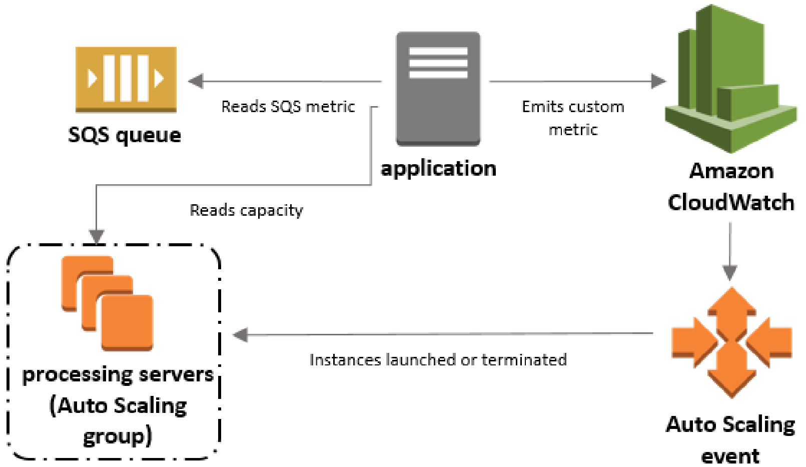
- You change the Availability Zones for your group.
- You explicitly terminate or detach instances and the group becomes unbalanced.
- An Availability Zone that previously had insufficient capacity recovers and has additional capacity available.
- An Availability Zone that previously had a Spot price above your maximum price now has a Spot price below your maximum price.

When rebalancing, Amazon EC2 Auto Scaling launches new instances before terminating the old ones, so that rebalancing does not compromise the performance or availability of your application.

Because Amazon EC2 Auto Scaling attempts to launch new instances before terminating the old ones, being at or near the specified maximum capacity could impede or completely halt rebalancing activities. To avoid this problem, the system can temporarily exceed the specified maximum capacity of a group by a 10 percent margin (or by a 1-instance margin, whichever is greater) during a rebalancing activity. The margin is extended only if the group is at or near maximum capacity and needs rebalancing, either because of user-requested rezoning or to compensate for zone availability issues. The extension lasts only as long as needed to rebalance the group typically a few minutes.

Scaling Based on Amazon SQS

https://docs.aws.amazon.com/autoscaling/ec2/userguide/as-using-sqs-queue.html



Choosing an Effective Metric and Target Value

The number of messages in your Amazon SQS queue does not solely define the number of instances needed. The solution is to use a ***backlog per instance*** metric with the target value being the ***acceptable backlog per instance*** to maintain. You can calculate these numbers as follows:

- **Backlog per instance**: To determine your backlog per instance, start with the SQS metric ApproximateNumberOfMessages (number of messages available for retrieval from the queue). Divide that number by the fleet's running capacity, which for an Auto Scaling group is the number of instances in the InService state, to get the backlog per instance.
- **Acceptable backlog per instance**: To determine your target value, first calculate what your application can accept in terms of latency. Then, take the acceptable latency value and divide it by the average time that an EC2 instance takes to process a message.

Current ApproximateNumberOfMessages - 1500
Fleet's running capacity - 10

If the average processing time is 0.1 seconds for each message and the longest acceptable latency is 10 seconds, then the acceptable backlog per instance is 10 / 0.1, which equals 100.

This means that 100 is the target value for your target tracking policy. Because the backlog per instance is currently at 150 (1500 / 10), your fleet scales out by five instances to maintain proportion to the target value.

EC2 - Enhanced networking

https://aws.amazon.com/premiumsupport/knowledge-center/enable-configure-enhanced-networking/

Enhanced networking provides **higher bandwidth**, **higher packet-per-second (PPS)** performance, and consistently **lower inter-instance latencies**.

If your packets-per-second rate appears to have reached its ceiling, you should consider moving to enhanced networking, because you have likely reached the upper thresholds of the virtual network interface driver. EC2 instances have 3 different virtual network adapters that can be used:

Adapter	Example Instance Types	Kernel Module	Windows Drivers	Performance
VIF	All	xen-netfront	Citrix or AWS PV	Low to medium
Intel 82599 VF	C3, C4, D2, I2, R3 and M4 (excluding m4.16xlarge)	ixgbevf	Intel 82599 VF	**Up to 10 Gbps**
Elastic Network Adapter	C5, C5d, F1, G3, H1, I3, m4.16xlarge, M5, M5a, M5d, P2, P3, R4, R5, R5a, R5d, T3, u-6tb1.metal, u-9tb1.metal, u-12tb1.metal, X1, X1e and z1d	ena	ena	**Up to 25 Gbps**

Placement groups are recommended for applications that benefit from low network latency, high network throughput, or both. To provide the lowest latency and the highest PPS network performance for your instances, consider using a placement group.
If you need to reach speeds up to 10 Gbps between instances, you should launch your instances into a placement group with the enhanced networking instance type.
If you need to reach speeds up to 25 Gbps between instances, you should launch instances in a placement group with ENA.

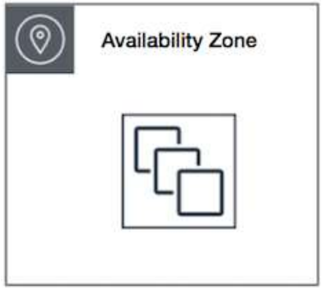
EC2 - Placement groups

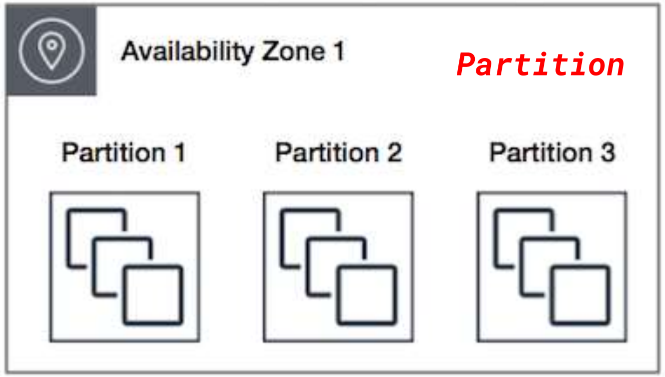
Jumbo frames allow more than 1500 bytes of data by increasing the payload size per packet, and thus increasing the percentage of the packet that is not packet overhead.
For instances that are collocated inside a cluster placement group, jumbo frames help to achieve the maximum network throughput possible, and they are recommended in this case.
Jumbo frames should be used with caution for Internet-bound traffic or any traffic that leaves a VPC. Packets are fragmented by intermediate systems, which slows down this traffic. You can use jumbo frames for traffic between your VPCs and your on-premises networks over AWS Direct Connect.

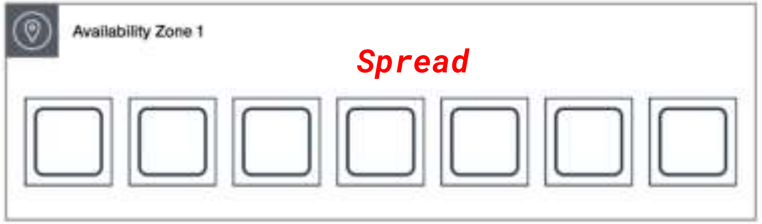
You can use *placement groups* to influence the **placement of a group of *interdependent* instances** to meet the needs of your workload. Depending on the type of workload, use following placement strategies:

- ***Cluster*** – packs instances close together inside an AZ. This strategy enables workloads to achieve the low-latency network performance necessary for tightly-coupled node-to-node communication that is typical of HPC applications.
- ***Partition*** – spreads your instances across logical partitions such that groups of instances in one partition do not share the underlying hardware with groups of instances in different partitions. This strategy is typically used by large distributed and replicated workloads, such as Hadoop, Cassandra, and Kafka.
- ***Spread*** – strictly places a small group of instances across distinct underlying hardware to reduce correlated failures.

Cluster







Spread can span different AZs

CF template creating an Auto Scaling group with a LoadBalancer, a security group that defines ingress rules, CloudWatch alarms, and Auto Scaling policies

https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/example-templates-autoscaling.html

AWS::AutoScaling::AutoScalingGroup resource WebServerGroup declares the following Auto Scaling group configuration:

- ***AvailabilityZones*** specifies the AZs. The Fn::GetAZs function call { "Fn::GetAZs" : "" } specifies all availability zones for the region in which the stack is created.
- ***MinSize*** and *MaxSize*
- ***LoadBalancerNames***

AWS::AutoScaling::LaunchConfiguration resource LaunchConfig declares the following configurations:

- ***KeyName*** takes the value of the KeyName input parameter as the EC2 key pair to use.
- ***UserData*** is the Base64 encoded value of the WebServerPort parameter, which is passed to an application .
- ***SecurityGroups***
- ***ImageId*** is the evaluated value of a set of nested maps.

AWS::ElasticLoadBalancing::LoadBalancer:

- ***AvailabilityZones*** is a list of availability zones where the LoadBalancer will distribute traffic.
- ***Listeners*** is a list of load balancing routing configurations that specify the port that the LoadBalancer accepts requests, the port on the registered EC2 instances where the LoadBalancer forwards requests, and the protocol used to route requests.
- ***HealthCheck*** is the configuration that Elastic Load Balancing uses to check the health of the EC2 instances that the LoadBalancer routes traffic to.

The AWS::AutoScaling::ScalingPolicy resource WebServerScaleUpPolicy is an Auto Scaling policy that scales up the Auto Scaling group WebServerGroup.

The AWS::CloudWatch::Alarm resource CPUAlarmHigh specifies the scaling policy WebServerScaleUpPolicy as the action to execute when the alarm is in an ALARM state (AlarmActions).

EC2 - Auto scaling group

https://docs.aws.amazon.com/autoscaling/ec2/userguide/AutoScalingGroup.html

An Auto Scaling group can launch On-Demand Instances, Spot Instances, or both. You can specify multiple purchase options for your Auto Scaling group only when you configure the group to use a launch template. (We recommend that you use launch templates instead of launch configurations to make sure that you can use the latest features of Amazon EC2.)

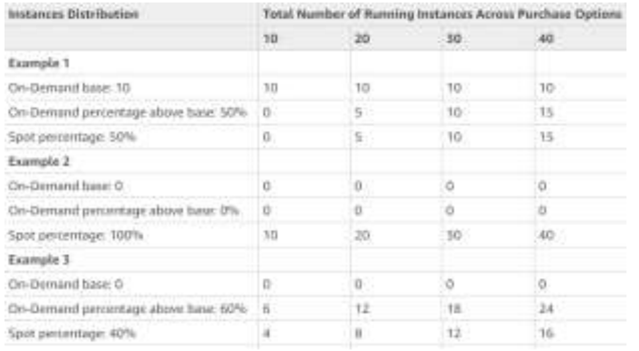
An Auto Scaling group contains a collection of Amazon EC2 instances that are treated as a logical grouping for the purposes of automatic scaling and management. An Auto Scaling group also enables you to use Amazon EC2 Auto Scaling features such as health check replacements and scaling policies. Both maintaining the number of instances in an Auto Scaling group and automatic scaling are the core functionality of the Amazon EC2 Auto Scaling service.

Auto Scaling Groups with Multiple Instance Types and Purchase Options:

You enhance availability by deploying your application across multiple instance types running in multiple Availability Zones. You can use just one instance type, but **it is a best practice to use a few instance types to avoid trying to launch instances from instance pools with insufficient capacity.** If the Auto Scaling group's request for Spot Instances cannot be fulfilled in one Spot Instance pool, it keeps trying in other Spot Instance pools rather than launching On-Demand Instances, so that you can leverage the cost savings of Spot Instances.

Allocation strategies:

On-demand instances: “prioritized”.
Spot instances: “capacity-optimized” & “lowest-price”



Instances Distribution	Total Number of Running Instances Across Purchase Options			
	10	20	30	40
Example 1				
On-Demand base: 10	10	10	10	10
On-Demand percentage above base: 50%	0	5	10	15
Spot percentage: 50%	0	5	10	15
Example 2				
On-Demand base: 0	0	0	0	0
On-Demand percentage above base: 0%	0	0	0	0
Spot percentage: 100%	10	20	30	40
Example 3				
On-Demand base: 0	0	0	0	0
On-Demand percentage above base: 60%	6	12	18	24
Spot percentage: 40%	4	8	12	16

EC2 - Multiple addresses

https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/MultipleIP.html

It can be useful to assign multiple IP addresses to an instance in your VPC to do the following:

- Host **multiple websites** on a single server by using **multiple SSL certificates** on a single server and associating each certificate with a specific IP address.
- Operate network appliances, such as **firewalls or load balancers**, that have **multiple IP addresses** for each network interface.
- **Redirect internal traffic to a standby instance** in case your instance fails, by reassigning the secondary IP address to the standby instance.

EC2 - Instance Profiles

https://docs.aws.amazon.com/IAM/latest/UserGuide/id_roles_use_switch-role-ec2_instance-profiles.html

An **Instance profile** is a container for an IAM role that you can use to pass role information to an EC2 instance when the instance starts.

If creating the EC2 role in AWS console, the instance profile is automatically created

If creating the EC2 role using CLI or API, you have to create the 2 separately.

	AWS CLI	AWS API
Create profile	aws iam create-instance-profile	CreateInstanceProfile
Add role	aws iam add-role-to-instance-profile	AddRoleToInstanceProfile
List profiles	aws iam list-instance-profiles, aws iam list-instance-profiles-for-role	ListInstanceProfiles, ListInstanceProfilesForRole
Get profile info	aws iam get-instance-profile	GetInstanceProfile
Remove role	aws iam remove-role-from-instance-profile	RemoveRoleFromInstanceProfile
Delete profile	aws iam delete-instance-profile	DeleteInstanceProfile
Attach role to running instance	aws ec2 associate-iam-instance-profile	AssociateIamInstanceProfile
Get info about profile from running instance	aws ec2 describe-iam-instance-profile-associations	DescribeIamInstanceProfileAssociations
Detach a profile	aws ec2 disassociate-iam-instance-profile	DisassociateIamInstanceProfile

```
curl http://169.254.169.254/latest/meta-data/iam/security-credentials/s3access
```

```
  "Code" : "Success",
  "LastUpdated" : "2012-04-26T16:39:16Z",
  "Type" : "AWS-HMAC",
  "AccessKeyId" : "ASIAIOSFODNN7EXAMPLE",
  "SecretAccessKey" :
"wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKE
  "Token" : "token",
  "Expiration" : "2017-05-17T15:09:54Z"
}
```

EC2 - Instance metadata

https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-instance-metadata.html

Instance metadata is data about your instance that you can use to configure or manage the running instance. Instance metadata is divided into categories. To view all categories of instance metadata from within a running instance, use the following URI

```
http://169.254.169.254/latest/meta-data/
```

Example: AMI Launch Index Value

This example demonstrates how you can use both user data and instance metadata to configure your instances.

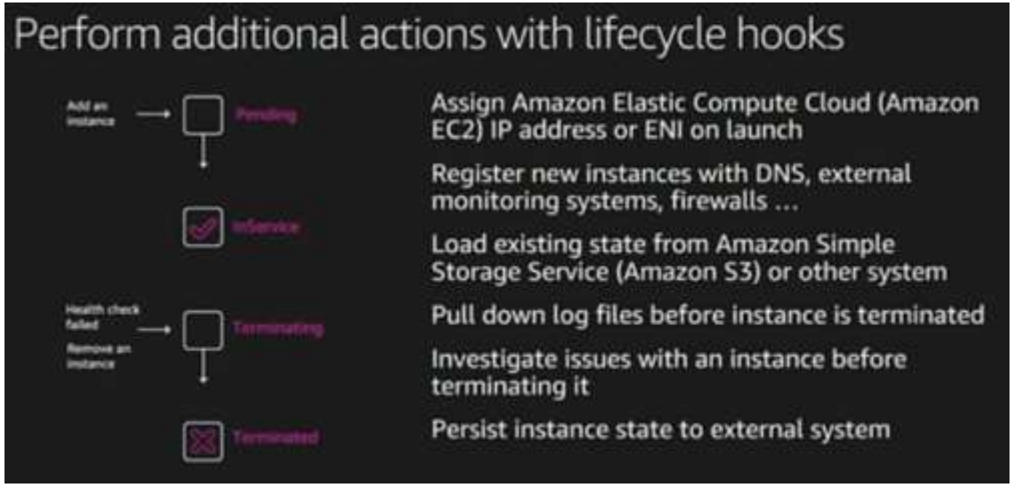
Alice wants to launch four instances of her favorite database AMI, with the first acting as master and the remaining three acting as replicas. When she launches them, she wants to add user data about the replication strategy for each replicant. She is aware that this data will be available to all four instances, so she needs to structure the user data in a way that allows each instance to recognize which parts are applicable to it. She can do this using the ami-launch-index instance metadata value, which will be unique for each instance.

Here is the user data that Alice has constructed:

```
replicate-every=1min | replicate-every=5min | replicate-every=10min
```

The replicate-every=1min data defines the first replicant's configuration, replicate-every=5min defines the second replicant's configuration, and so on. Alice decides to provide this data as an ASCII string with a pipe symbol (|) delimiting the data for the separate instances.

ami-id
ami-launch-index
ami-manifest-path
ancestor-ami-ids
block-device-mapping/ami
block-device-mapping/ebs*N*
block-device-mapping/ephemeral*N*
elastic-gpus/associations/*elastic-gpu-id*
elastic-inference/associations/*eia-id*
iam/security-credentials/*role-name* If there is an IAM role associated with the instance, *role-name* is the name of the role, and *role-name* contains the temporary security credentials associated with the role. Otherwise, not present.
instance-type
hostname



Perform additional actions with lifecycle hooks

Add an instance
Pending
InService
Health check failed
Remove an instance
Terminating
Terminated

Assign Amazon Elastic Compute Cloud (Amazon EC2) IP address or ENI on launch

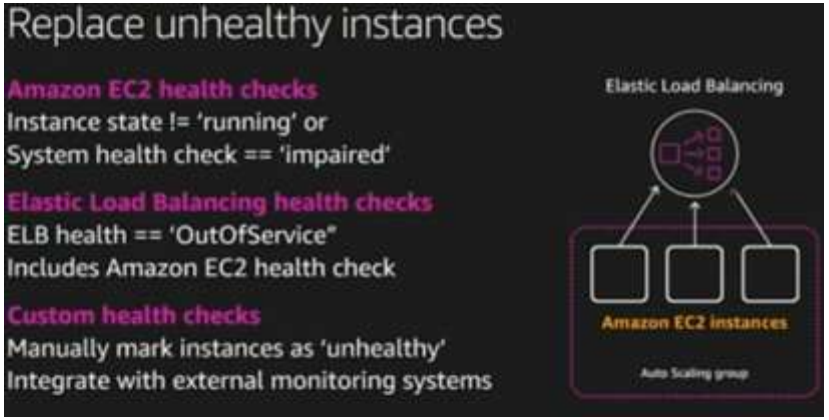
Register new instances with DNS, external monitoring systems, firewalls …

Load existing state from Amazon Simple Storage Service (Amazon S3) or other system

Pull down log files before instance is terminated

Investigate issues with an instance before terminating it

Persist instance state to external system



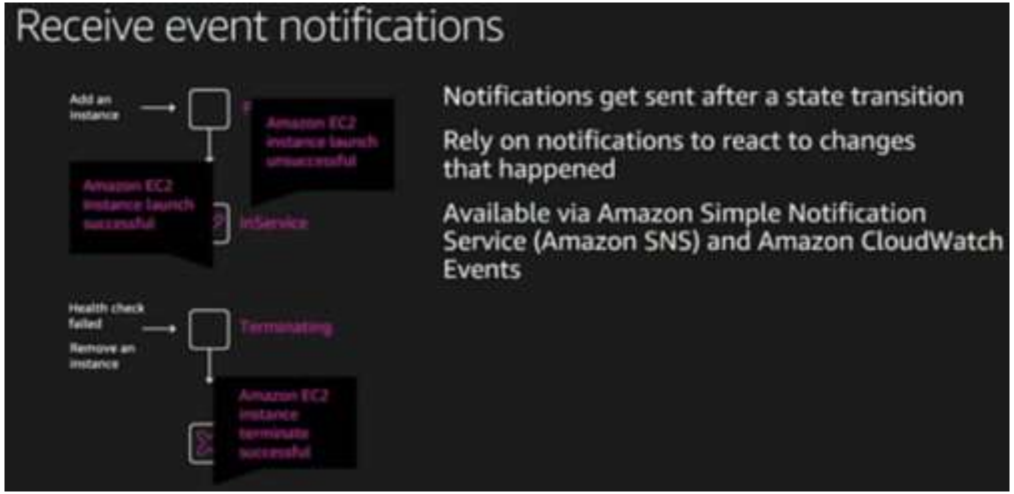
Replace unhealthy instances

Amazon EC2 health checks
Instance state != 'running' or
System health check == 'impaired'

Elastic Load Balancing health checks
ELB health == 'OutOfService'
Includes Amazon EC2 health check

Custom health checks
Manually mark instances as 'unhealthy'
Integrate with external monitoring systems

Elastic Load Balancing
Amazon EC2 instances
Auto Scaling group



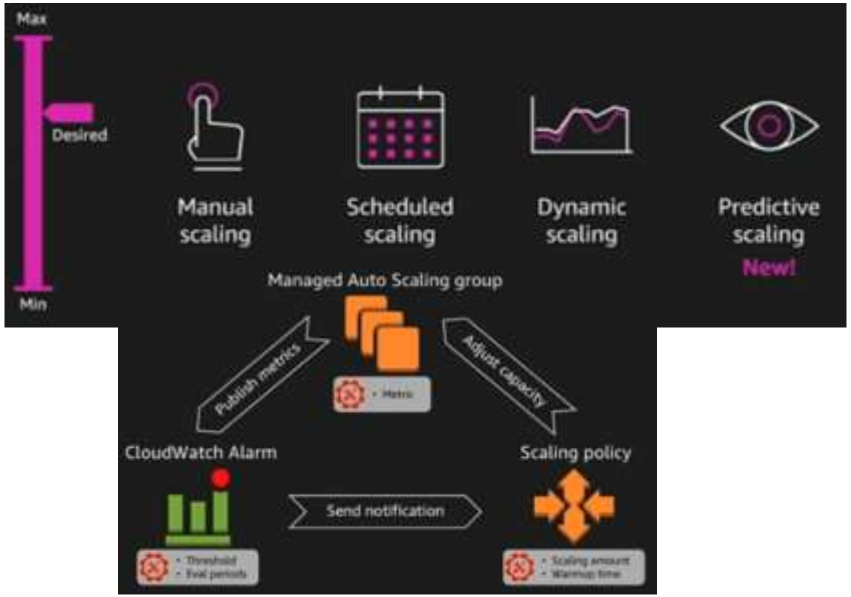
Receive event notifications

Add an instance
Amazon EC2 instance launch unsuccessful
Amazon EC2 instance launch successful
InService
Health check failed
Remove an instance
Terminating
Amazon EC2 instance terminate successful

Notifications get sent after a state transition

Rely on notifications to react to changes that happened

Available via Amazon Simple Notification Service (Amazon SNS) and Amazon CloudWatch Events



EC2Rescue Tool

https://docs.aws.amazon.com/systems-manager/latest/userguide/automation-ec2rescue.html

EC2Rescue can help you diagnose and troubleshoot problems on EC2 Linux and Windows Server instances. You can run the tool manually Or, you can run the tool automatically by using Systems Manager Automation and the **AWSSupport-ExecuteEC2Rescue** document. The **AWSSupport-ExecuteEC2Rescue** document is designed to perform a combination of Systems Manager actions, AWS CloudFormation actions, and Lambda functions that automate the steps normally required to use EC2Rescue.

You can use the **AWSSupport-ExecuteEC2Rescue** document to troubleshoot and potentially remediate different types of operating system (OS) issues.

How It Works

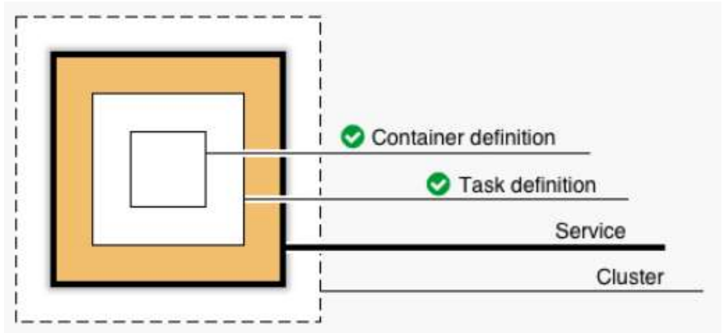
Troubleshooting an instance with Automation and the **AWSSupport-ExecuteEC2Rescue** document works as follows:

- You specify the ID of the unreachable instance and run the Automation workflow.
- The system creates a temporary VPC, and then runs a series of Lambda functions to configure the VPC.
- The system identifies a subnet for your temporary VPC in the same Availability Zone as your original instance.
- The system launches a temporary, SSM-enabled helper instance.
- The system stops your original instance, and creates a backup. It then attaches the original root volume to the helper instance.
- The system uses Run Command to run EC2Rescue on the helper instance. EC2Rescue identifies and attempts to fix issues on the attached, original root volume. When finished, EC2Rescue reattaches the root volume back to the original instance.
- The system restarts your original instance, and terminates the temporary instance. The system also terminates the temporary VPC and the Lambda functions created at the start of the automation.

ECS / Fargate

ECS / Fargate

https://medium.com/@marcus.cavalcanti/deploying-microservices-with-aws-fargate-f0578d8199a3



AWS Fargate
ECS Task

Task: A group of one or more containers launched and maintained by ECS based on a preset Task Definition. This is the smallest unit.

A **Task definition** The blueprint for a task, which is used by ECS to launch containers. It can contain following parameters:

- The Docker image to use with each container in your task (max 10 containers)
- How much CPU and memory to use with each task
- The launch type to use
- The Docker networking mode to use for the containers in your task
- The logging configuration to use for your tasks
- Whether the task should continue to run if the container finishes or fails
- The command the container should run when it is started
- Any data volumes that should be used with the containers in the task
- The IAM role that your tasks should use

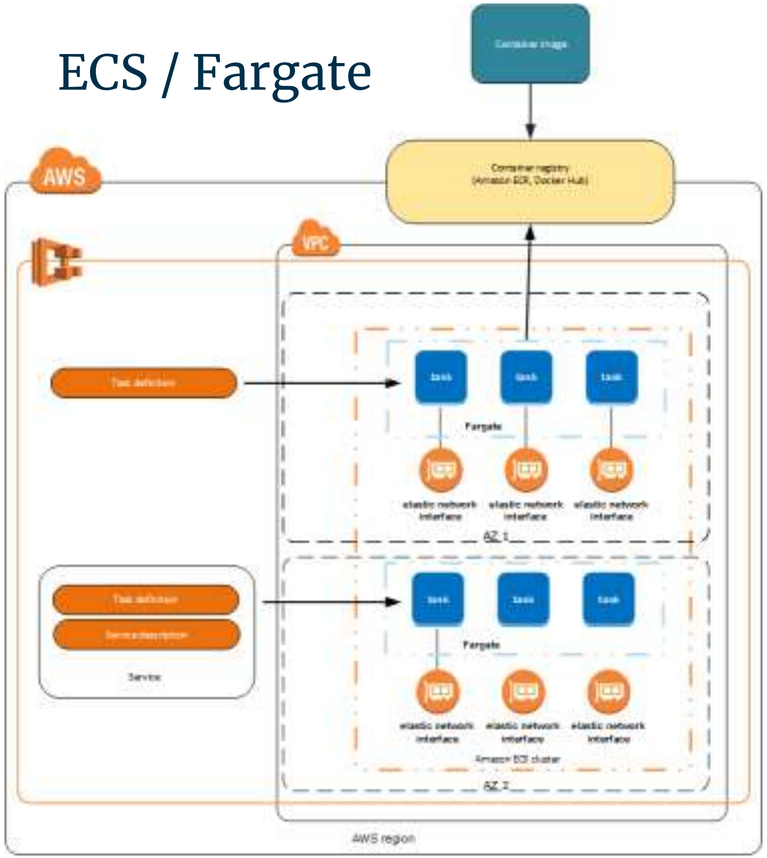
A **service** allows you to run and maintain a specified number (the "desired count") of simultaneous instances of a task definition in an ECS cluster

Cluster: A physical grouping of underlying servers that holds all of your services, tasks, etc. This is the largest unit.

ECR: Elastic Container Registry hosts images similar to Docker Hub.

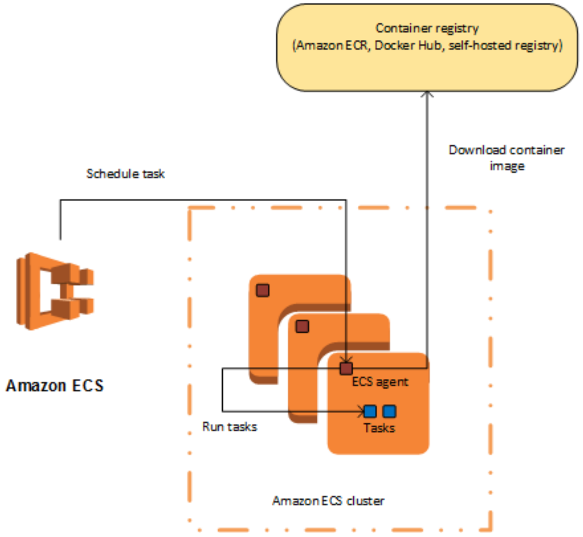
```
aws configure <access id> <secret key>
aws ecr get-login
$ docker login
Create DockerFile
$ docker build . -t learn-fargate:roughnecks // create docker image
$ docker run --rm -it learn-fargate:roughnecks // run locally
$ docker push 031780582162.dkr.ecr.us-east-1.amazonaws.com/learn-fargate:roughnecks // push to ECR
```

ECS / Fargate

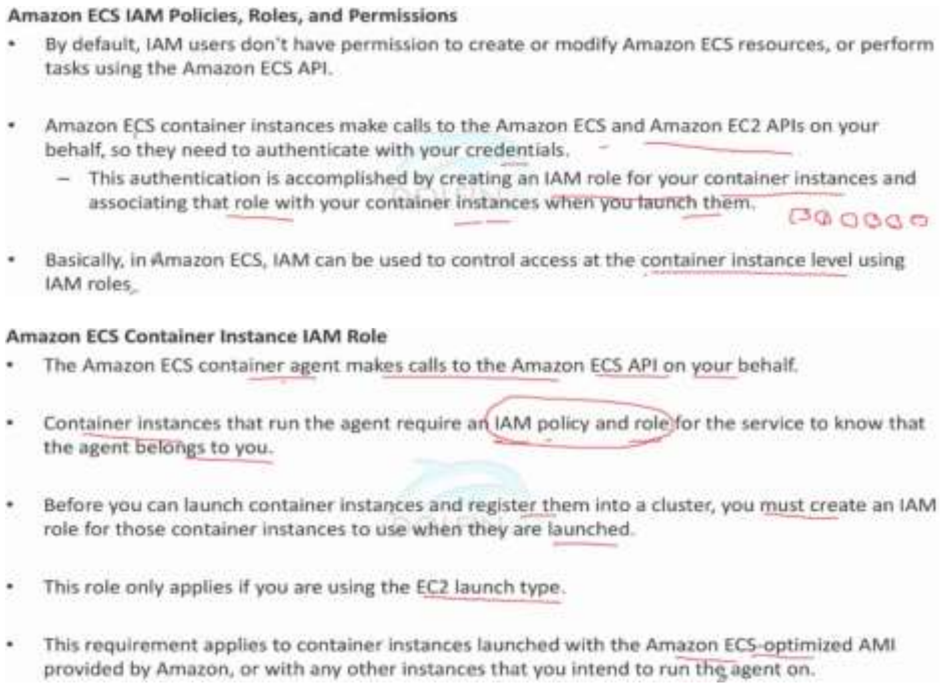

Container image
Container registry (Amazon ECR, Docker Hub)
AWS
VPC
Task definition
task
task
task
Fargate
elastic network interface
elastic network interface
elastic network interface
AZ 1
Task definition
Service description
Service
task
task
task
Fargate
elastic network interface
elastic network interface
elastic network interface
Amazon ECS cluster
AZ 2
AWS region

Container Agent

The *container agent* runs on each infrastructure resource within an Amazon ECS cluster. It sends information about the resource's current running tasks and resource utilization to Amazon ECS, and starts and stops tasks whenever it receives a request from Amazon ECS.



ECS – IAM Roles

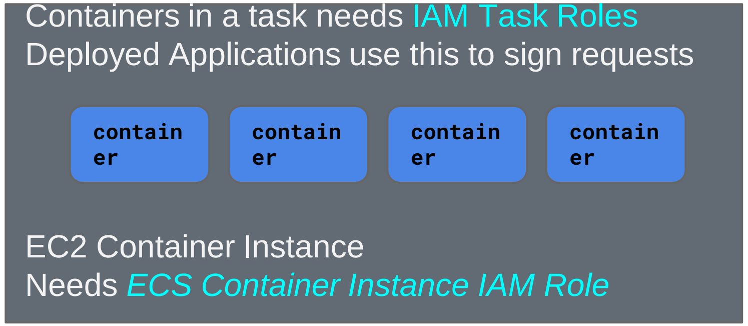


Amazon ECS IAM Policies, Roles, and Permissions

- By default, IAM users don't have permission to create or modify Amazon ECS resources, or perform tasks using the Amazon ECS API.
- Amazon ECS container instances make calls to the Amazon ECS and Amazon EC2 APIs on your behalf, so they need to authenticate with your credentials.
  - This authentication is accomplished by creating an IAM role for your container instances and associating that role with your container instances when you launch them.
- Basically, in Amazon ECS, IAM can be used to control access at the container instance level using IAM roles.

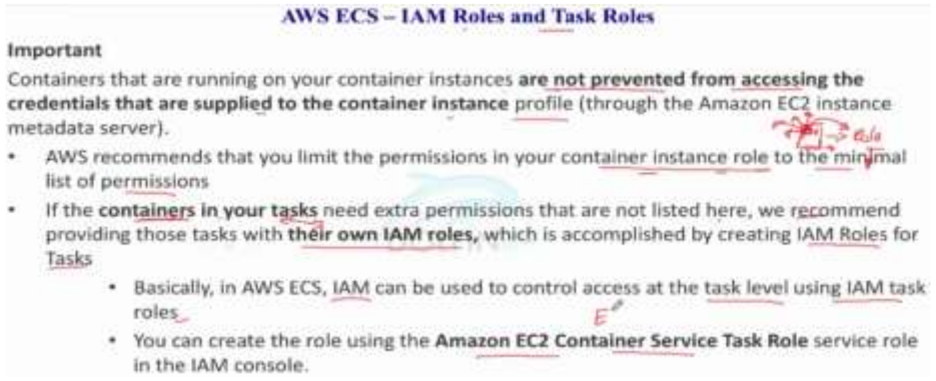
Amazon ECS Container Instance IAM Role

- The Amazon ECS container agent makes calls to the Amazon ECS API on your behalf.
- Container instances that run the agent require an IAM policy and role for the service to know that the agent belongs to you.
- Before you can launch container instances and register them into a cluster, you must create an IAM role for those container instances to use when they are launched.
- This role only applies if you are using the EC2 launch type.
- This requirement applies to container instances launched with the Amazon ECS-optimized AMI provided by Amazon, or with any other instances that you intend to run the agent on.



Containers in a task needs IAM Task Roles
Deployed Applications use this to sign requests

EC2 Container Instance
Needs *ECS Container Instance IAM Role*

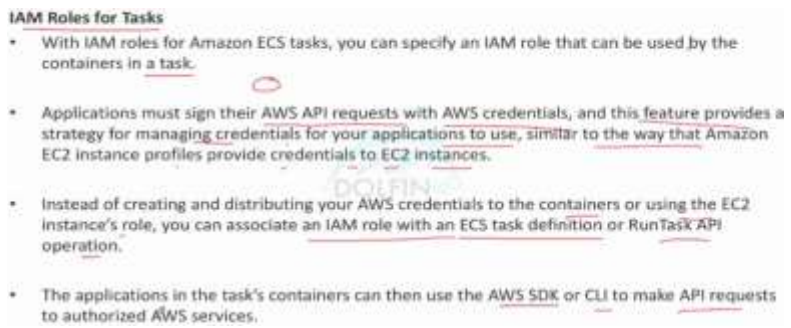


AWS ECS – IAM Roles and Task Roles

Important

Containers that are running on your container instances **are not prevented from accessing the credentials that are supplied to the container instance** profile (through the Amazon EC2 instance metadata server).

- AWS recommends that you limit the permissions in your container instance role to the minimal list of permissions
- If the **containers in your tasks** need extra permissions that are not listed here, we recommend providing those tasks with **their own IAM roles,** which is accomplished by creating IAM Roles for Tasks
  - Basically, in AWS ECS, IAM can be used to control access at the task level using IAM task roles.
  - You can create the role using the **Amazon EC2 Container Service Task Role** service role in the IAM console.

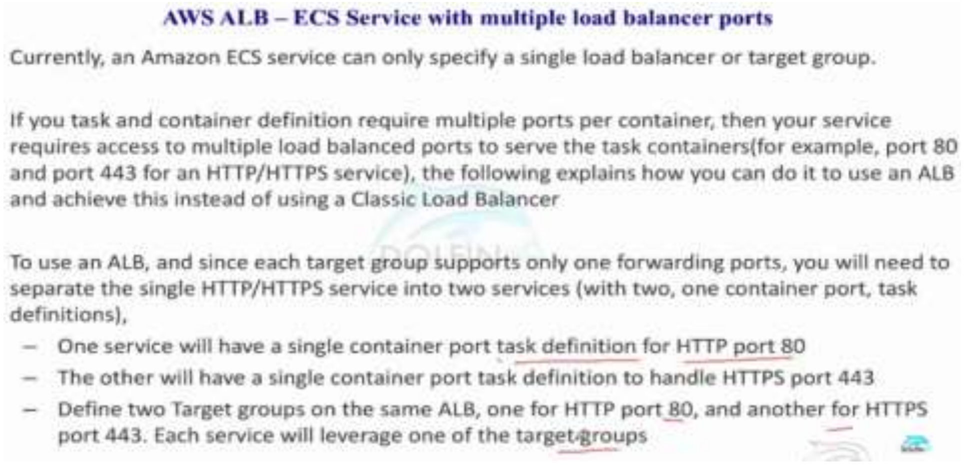


IAM Roles for Tasks

- With IAM roles for Amazon ECS tasks, you can specify an IAM role that can be used by the containers in a task.
- Applications must sign their AWS API requests with AWS credentials, and this feature provides a strategy for managing credentials for your applications to use, similar to the way that Amazon EC2 instance profiles provide credentials to EC2 instances.
- Instead of creating and distributing your AWS credentials to the containers or using the EC2 instance's role, you can associate an IAM role with an ECS task definition or RunTask API operation.
- The applications in the task's containers can then use the AWS SDK or CLI to make API requests to authorized AWS services.

ECS - Dynamic host port mapping

In short, Dynamic port mapping, allows containers on the same ECS instance, and serving different applictions to listen on the same port, and map this port from each container to a different port on the HOST that gets registered with the ALB.



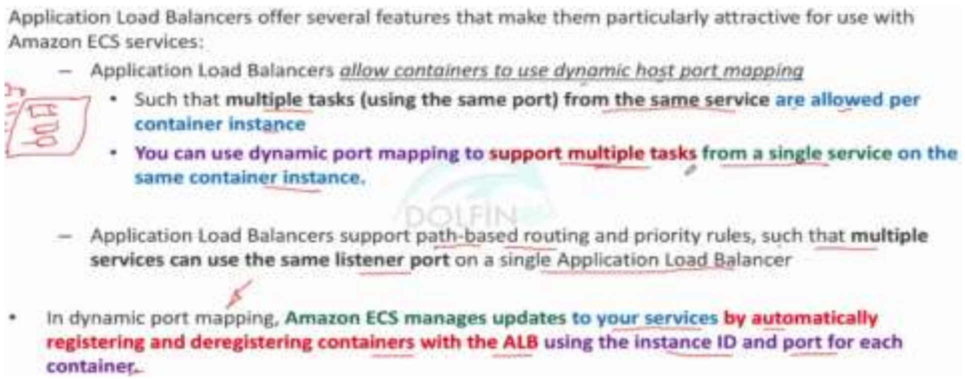
AWS ALB – ECS Service with multiple load balancer ports

Currently, an Amazon ECS service can only specify a single load balancer or target group.

If you task and container definition require multiple ports per container, then your service requires access to multiple load balanced ports to serve the task containers(for example, port 80 and port 443 for an HTTP/HTTPS service), the following explains how you can do it to use an ALB and achieve this instead of using a Classic Load Balancer

To use an ALB, and since each target group supports only one forwarding ports, you will need to separate the single HTTP/HTTPS service into two services (with two, one container port, task definitions),

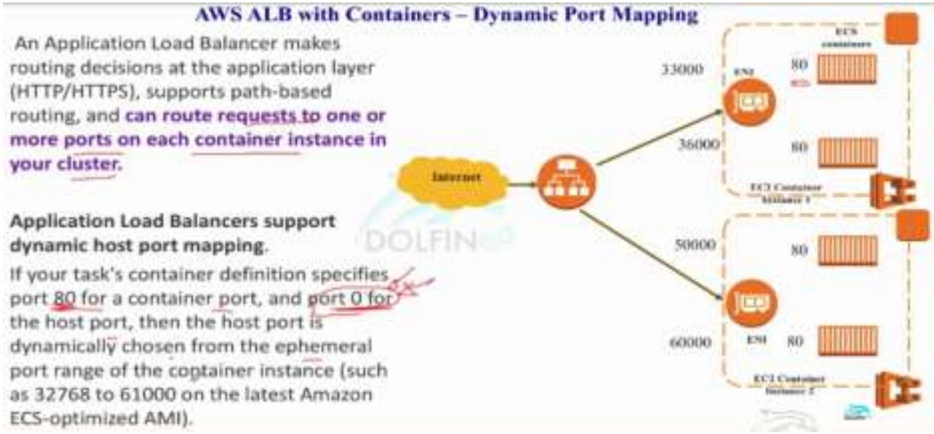
- One service will have a single container port task definition for HTTP port 80
- The other will have a single container port task definition to handle HTTPS port 443
- Define two Target groups on the same ALB, one for HTTP port 80, and another for HTTPS port 443. Each service will leverage one of the target groups



Application Load Balancers offer several features that make them particularly attractive for use with Amazon ECS services:

- Application Load Balancers *allow containers to use dynamic host port mapping*
  - Such that **multiple tasks (using the same port) from the same service** are allowed per container instance
  - **You can use dynamic port mapping to support multiple tasks from a single service on the same container instance.**
- Application Load Balancers support path-based routing and priority rules, such that **multiple services can use the same listener port** on a single Application Load Balancer

- In dynamic port mapping, **Amazon ECS manages updates to your services by automatically registering and deregistering containers with the ALB using the instance ID and port for each container.**



AWS ALB with Containers – Dynamic Port Mapping

An Application Load Balancer makes routing decisions at the application layer (HTTP/HTTPS), supports path-based routing, and **can route requests to one or more ports on each container instance in your cluster.**

Application Load Balancers support dynamic host port mapping.

If your task's container definition specifies port 80 for a container port, and port 0 for the host port, then the host port is dynamically chosen from the ephemeral port range of the container instance (such as 32768 to 61000 on the latest Amazon ECS-optimized AMI).

ECS load balancing

https://docs.aws.amazon.com/AmazonECS/latest/developerguide/service-load-balancing.html

Application Load Balancers offer several features that make them attractive for use with Amazon ECS services:

- Each service can serve traffic from multiple load balancers and expose multiple load balanced ports by specifying multiple target groups.
- They are supported by tasks using both the Fargate and EC2 launch types.
- Application Load Balancers allow containers to use **dynamic host port mapping** (so that multiple tasks from the same service are allowed per container instance).
- Application Load Balancers support **path-based routing and priority rules** (so that multiple services can use the same listener port on a single ALB).

We recommend that you use ALB for your ECS services so that you can take advantage of these latest features, unless your service requires a feature that is only available with NLB or Classic LB

Classic Load Balancer Considerations

- Services with tasks that use the awsvpc network mode, such as those with the Fargate launch type, do not support Classic Load Balancers.
- Container health checks are not supported for tasks that are part of a service that is configured to use a Classic Load Balancer.
- All of the containers that are launched in a single task definition are always placed on the same container instance. e.g, if a task definition consists of Elasticsearch using port 3030 on the container instance, with Logstash and Kibana using port 4040 on the container instance, the same load balancer can route traffic to Elasticsearch and Kibana through two listeners.
  Important
  THIS IS NOT RECOMMENDED. Because entire container instances are registered and deregistered with clb, and not with host and port combinations, this configuration can cause issues if a task from one service stops. In this scenario, a task from one service stopping can cause the entire container instance to be deregistered from the Clb while another task from a different service on the same container instance is still using it. If you want to connect multiple services to a single load balancer we recommend using an ALB.

Fargate - VPC link to connect API Gateway to Fargate internal traffic

https://www.cloudar.be/uncategorized/connect-to-private-resources-from-api-gateway-with-vpc-link/

Below steps needed for enabling Public APIG to connect to INTERNALLY exposed Fargate end point using VPC link. Older alternative was using Proxy lambdas.

1) Create Fargate cluster WITHOUT Application Load balancer
2) Create NLB
    a) Add Target Group using Elastic Private IPs (if using Fargate, else use Instances)
    b) Take the Private IPs from Fargate Task details
3) Api Gateway
    a) Create VPC Link using above NLB
    b) In Api Resource Integration Request, select VPC link and point to the above VPC Link
    c) Provide the End point URL *http://InternalNLB:8080/contacts/all*

Pricing Notes:

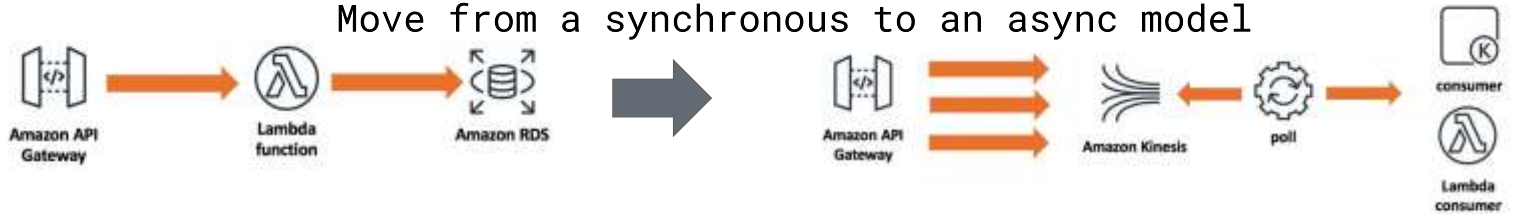
- *NLB is NOT free, Pricing based on hour ($0.0225 per hour & $0.006 per LCU-hr).*
- *In Fargate service, reduce the task count to 0 to stop getting charged for vCPU ($0.0506 per hour) and Mem ($0.0127 per hour)*

lambda

Lambda best practices

https://aws.amazon.com/blogs/architecture/how-to-design-your-serverless-apps-for-massive-sc

Move from a synchronous to an async model



```
Tip #1: When to VPC-Enable a Lambda Function
Tip #2: Deploy Common Code to a Lambda Layer (i.e. the AWS SDK)
Tip #3: Watch Your Package Size and Dependencies
Tip #4: Monitor Your Concurrency (and Set Alarms)
Tip #5: Over-Provision Memory (in some use cases) but Not Function Timeout
```

Understanding the Different Ways to Invoke Lambda Functions

https://aws.amazon.com/blogs/architecture/understanding-the-different-ways-to-invoke-lambda-functions/

Here is a list of services that invoke Lambda functions synchronously:

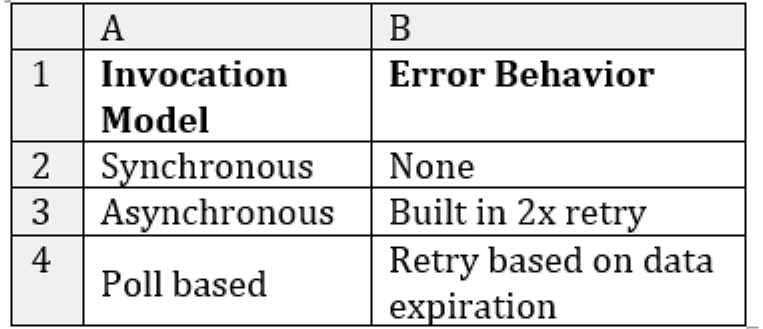
- Application Load Balancer
- Amazon Cognito
- Amazon Lex
- Amazon Alexa
- Amazon API Gateway
- Amazon CloudFront (Lambda@Edge)
- Amazon Kinesis Data Firehose

Here is a list of services that invoke Lambda functions asynchronously:

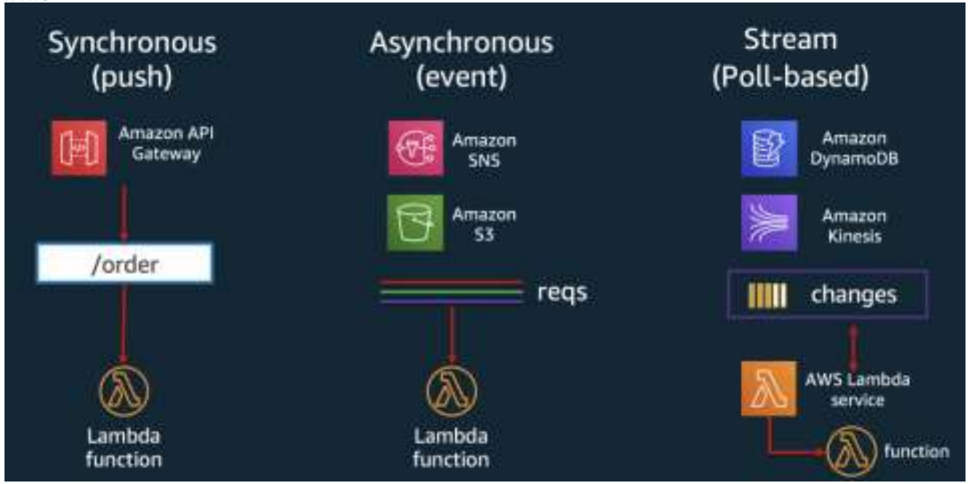
- Amazon Simple Storage Service
- Amazon Simple Notification Service
- Amazon Simple Email Service
- AWS CloudFormation
- Amazon CloudWatch Logs
- Amazon CloudWatch Events
- AWS CodeCommit
- AWS Config

Poll based invocations

- Amazon Kinesis
- Amazon SQS
- Amazon DynamoDB Streams



	A	B
1	**Invocation Model**	**Error Behavior**
2	Synchronous	None
3	Asynchronous	Built in 2x retry
4	Poll based	Retry based on data expiration



Lambda best practices

https://docs.aws.amazon.com/lambda/latest/dg/best-practices.html

Function Code:

1. Separate the Lambda handler from your core logic.
2. Take advantage of Execution Context reuse to improve the performance of your function.
3. Use AWS Lambda Environment Variables to pass operational parameters to your function.
4. Control the dependencies in your function's deployment package.
5. Minimize your deployment package size to its runtime necessities.
6. Reduce the time it takes Lambda to unpack deployment packages
7. Minimize the complexity of your dependencies.
8. Avoid using recursive code

Function Configuration

1. Performance testing your Lambda function
2. Load test your Lambda function
3. Use most-restrictive permissions when setting IAM policies.
4. Be familiar with AWS Lambda Limits.
5. Delete Lambda functions that you are no longer using
6. If you are using SQS as an event source, ur function’s expected execution time should not exceed visibility timeout value

Alarming and Metrics

1. Use AWS Lambda Metrics and CloudWatch Alarms
2. Leverage your logging library and AWS Lambda Metrics and Dimensions to catch app errors (ERROR, WARNING etc.)

Stream Event Invokes

1. Test with different batch and record sizes
2. Increase Kinesis stream processing throughput by adding shards.
3. Use Amazon CloudWatch on IteratorAge to determine if your Kinesis stream is being processed. For example, configure a CloudWatch alarm with a maximum setting to 30000 (30 seconds).

How do I give internet access to my Lambda function in a VPC?

https://aws.amazon.com/premiumsupport/knowledge-center/internet-access-lambda-function/

You might want your Lambda function to access private VPC resources (for example, RDS DB instance or Amazon EC2 instance). To configure this, you must associate the function with one or more private subnets in a VPC. To grant your function internet access, the associated VPC must have a NAT gateway (or NAT instance) in a public subnet.

Whether a subnet is private or public depends on its route table.

- A public subnet has a route pointing to an internet gateway, and
- A private subnet has a route pointing to a NAT gateway or NAT instance

All about lambda and VPCs

https://docs.aws.amazon.com/lambda/latest/dg/best-practices.html

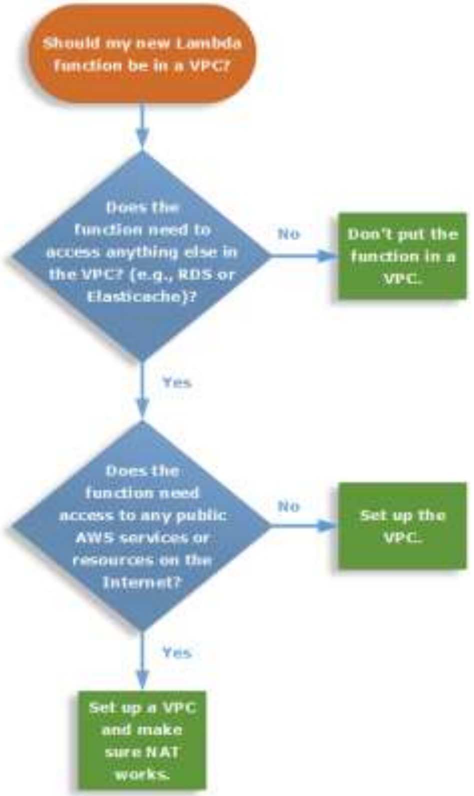
YOU DON'T NEED VPCs TO SECURE LAMBDA

Unlike EC2 instances, which need VPCs to shield them from malicious traffic. Lambda functions are protected by AWS IAM, which provides both authentication and authorization. All incoming requests have to be signed with a valid AWS credential, and the caller must have the correct IAM permission.

When should you use VPCs?

When you absolutely have to. That is, if your function needs to access a resource that runs in the VPC, for example:

- RDS database
- Elasticache cluster
- Internal APIs (running on containers or EC2)


Should my new Lambda function be in a VPC?
Does the function need to access anything else in the VPC? (e.g., RDS or Elasticache)?
No
Don't put the function in a VPC.
Yes
Does the function need access to any public AWS services or resources on the Internet?
No
Set up the VPC.
Yes
Set up a VPC and make sure NAT works.

Elastic Beanstalk

Blue/Green Deployments with Elastic Beanstalk

Because AWS Elastic Beanstalk performs an in-place update when you update your application versions, your application can become unavailable to users for a short period of time. You can avoid this downtime by performing a blue/green deployment, where you deploy the new version to a separate environment, and then swap CNAMEs of the two environments to redirect traffic to the new version instantly.

Blue/green deployments require that your environment runs independently of your production database, if your application uses one. If your environment has an Amazon RDS DB instance attached to it, the data will not transfer over to your second environment, and will be lost if you terminate the original environment.

Ebeanstalk Deployment policies

https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/using-features.rolling-version-deploy.html

All at once - Deploy the new version to all instances simultaneously. All instances in your environment are out of service for a short time while the deployment occurs.

Rolling (*like canary, the default*) - splits the environment's EC2 instances into batches and deploys the new version of the application to one batch at a time, leaving the rest of the instances in the environment running the old version of the application. During a rolling deployment, some instances serve requests with the old version of the application, while instances in completed batches serve other requests with the new version.

Rolling with additional batch - To maintain full capacity during deployments, you can configure your environment to launch a new batch of instances before taking any instances out of service. When the deployment completes, EBS terminates the additional batch of instances.

Immutable - perform an immutable update to launch a full set of new instances running the new version of the application in a separate Auto Scaling group, alongside the instances running the old version. Immutable deployments can prevent issues caused by partially completed rolling deployments. If the new instances don't pass health checks, EBS terminates them, leaving the original instances untouched.

Difference between Blue/Green and Immutable deployment in Beanstalk →
Blue/Green deployment is done by creating a new environment, deploying new EC2 instances running the new code to that new environment, doing a DNS CNAME swap to the new environment, and finally terminating the old environment. Traffic flows to one environment or the other, but not to both at the same time.

In Immutable deployment, there is no new environment. Instead, a new AutoScaling Group (ASG) is added to the original environment, new EC2 instances (with the new code) are launched into that new ASG, they are health-checked and then they are transferred out of that temporary ASG to the original ASG. Finally, the original EC2 instances and the, now empty, temporary ASG are terminated. For a certain amount of time, traffic will be flowing to both the new and the old instances.

How Rolling Deployments Work

When processing a batch, Elastic Beanstalk detaches all instances in the batch from the load balancer, deploys the new application version, and then reattaches the instances. If you enable connection draining, Elastic Beanstalk drains existing connections from the Amazon EC2 instances in each batch before beginning the deployment.

After reattaching the instances in a batch to the load balancer, Elastic Load Balancing waits until they pass a minimum number of Elastic Load Balancing health checks, and then starts routing traffic to them. If no health check URL is configured, this can happen very quickly, because an instance will pass the health check as soon as it can accept a TCP connection. If a health check URL is configured, the load balancer doesn't route traffic to the updated instances until they return a 200 OK status code in response to an HTTP GET request to the health check URL.

Elastic Beanstalk waits until all instances in a batch are healthy before moving on to the next batch. With basic health reporting, instance health depends on the Elastic Load Balancing health check status. When all instances in the batch pass enough health checks to be considered healthy by Elastic Load Balancing, the batch is complete. If enhanced health reporting is enabled, Elastic Beanstalk considers several other factors, including the result of incoming requests. With enhanced health reporting, all instances must pass 12 consecutive health checks with an OK status within two minutes for web server environments, and 18 health checks within three minutes for worker environments.

If a batch of instances does not become healthy within the command timeout, the deployment fails. After a failed deployment, check the health of the instances in your environment for information about the cause of the failure. Then perform another deployment with a fixed or known good version of your application to roll back.

If a deployment fails after one or more batches completed successfully, the completed batches run the new version of your application while any pending batches continue to run the old version. You can identify the version running on the instances in your environment on the health page in the console. This page displays the deployment ID of the most recent deployment that executed on each instance in your environment. If you terminate instances from the failed deployment, Elastic Beanstalk replaces them with instances running the application version from the most recent successful deployment.

Blue green techniques

https://d1.awsstatic.com/whitepapers/AWS_Blue_Green_Deployments.pdf#p

1. Update DNS Routing with Amazon Route 53
   - Switch all traffic or use weighted routing policy for canary deployment

1. Swap the Auto Scaling Group Behind Elastic Load Balancer
   - Attach a new green ASG , adjust the traffic at ALB to route t new ASG. As soon as the green group is scaled up without issues, you can decommission the blue group by adjusting the group size to zero.

1. Update Auto Scaling Group Launch Configurations
   - Replace the existing launch configuration with a new one. Aft a new launch configuration is in place, any new instances tha are launched use the new launch configuration parameters, but existing instances are not affected.
   - When Auto Scaling removes instances (referred to as scaling i from the group, the default termination policy is to remove instances with the oldest launch configuration.

1. Swap the Environment of an Elastic Beanstalk Application
   - Swap Environment URLs in R53

1. Clone a Stack in AWS OpsWorks and Update DNS
   - Clone a new Opsworks (green) stack, switch R53 or weighted

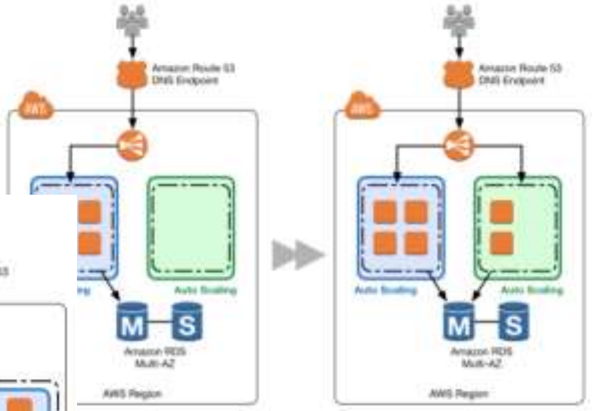


Figure 4: Swap Auto Scaling group pattern

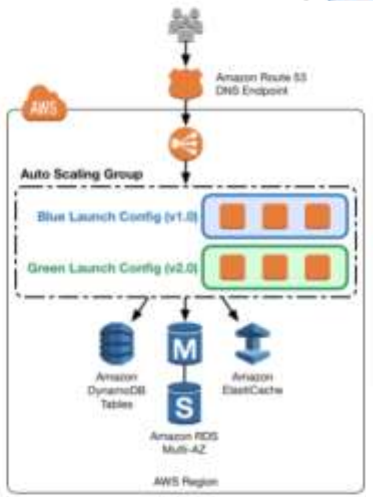


Figure 7: Scale up green launch configuration

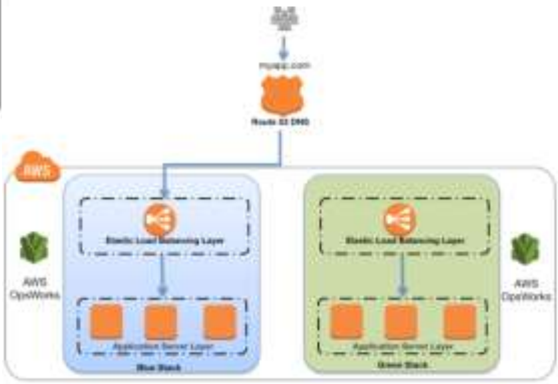


Figure 13: Clone stack to create green environment