AWS Compute Blog

Running Cost-effective queue workers with Amazon SQS and Amazon EC2 Spot Instances

by Ben Peven | on 25 NOV 2019 | in Amazon EC2, Amazon Simple Queue Service (SQS), Compute | Permalink |
Share

This post is contributed by Ran Sheinberg | Sr. Solutions Architect, EC2 Spot & Chad Schmutzer | Principal Developer Advocate, EC2 Spot | Twitter: @schmutze

Introduction

Amazon Simple Queue Service (SQS) is used by customers to run decoupled workloads in the AWS Cloud as a best practice, in order to increase their applications' resilience. You can use a worker tier to do background processing of images, audio, documents and so on, as well as offload long-running processes from the web tier. This blog post covers the benefits of pairing Amazon SQS and Spot Instances to maximize cost savings in the worker tier, and a customer success story.

Solution Overview

Amazon SQS is a fully managed message queuing service that enables customers to decouple and scale microservices, distributed systems, and serverless applications. It is a common best practice to use Amazon SQS with decoupled applications. Amazon SQS increases applications resilience by decoupling the direct communication between the frontend application and the worker tier that does data processing. If a worker node fails, the jobs that were running on that node return to the Amazon SQS queue for a different node to pick up.

Both the frontend and worker tier can run on Spot Instances, which offer spare compute capacity at steep discounts compared to On-Demand Instances. Spot Instances optimize your costs on the AWS Cloud and scale your application's throughput up to 10 times for the same budget. Spot Instances can be interrupted with two minutes of notification when EC2 needs the capacity back. You can use Spot Instances for various fault-tolerant and flexible applications. These can include analytics, containerized workloads, high performance computing (HPC), stateless web servers, rendering, CI/CD, and queue worker nodes—which is the focus of this post.

Worker tiers of a decoupled application are typically fault-tolerant. So, it is a prime candidate for running on interruptible capacity. Amazon SQS running on Spot Instances allows for more robust, cost-optimized applications.

By using EC2 Auto Scaling groups with multiple instance types that you configured as suitable for your application (for example, m4.xlarge, m5.xlarge, c5.xlarge, and c4.xlarge, in multiple Availability Zones), you can spread the worker tier's compute capacity across many Spot capacity pools (a combination of instance type and Availability Zone). This increases the chance of achieving the scale that's required for the worker tier to ingest messages from the queue, and of keeping that scale when Spot Instance interruptions occur, while selecting the lowest-priced Spot Instances in each availability zone.

You can also choose the capacity-optimized allocation strategy for the Spot Instances in your Auto Scaling group. This strategy automatically selects instances that have a lower chance of interruption, which decreases the chances of restarting jobs due to Spot interruptions. When Spot Instances are interrupted, your Auto Scaling group automatically replenishes the capacity from a different Spot capacity pool in order to achieve your desired capacity. Read the blog post "Introducing the capacity-optimized allocation strategy for Amazon EC2 Spot Instances" for more details on how to choose the suitable allocation strategy.

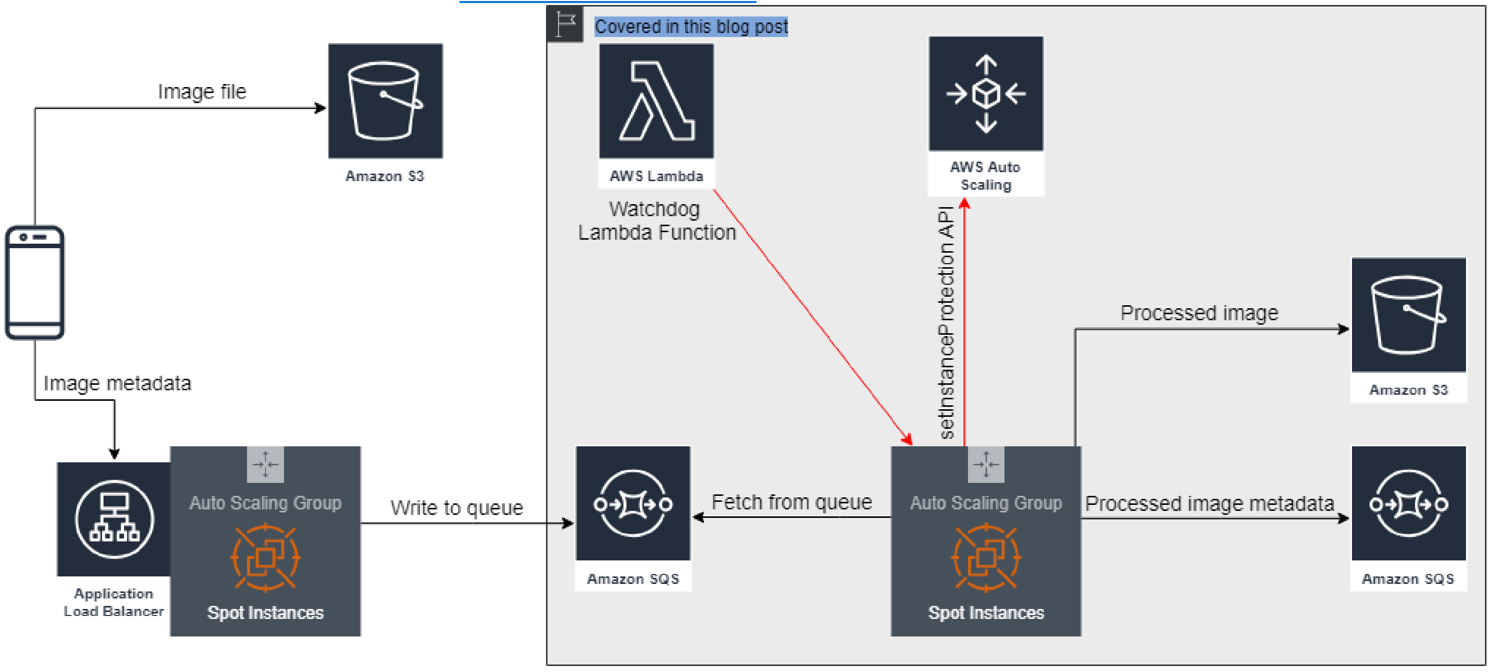
We focus on three main points in this blog:

1. Best practices for using Spot Instances with Amazon SQS
2. A customer example that uses these components
3. Example solution that can help you get you started quickly

Application of Amazon SQS with Spot Instances

Amazon SQS eliminates the complexity of managing and operating message-oriented middleware. Using Amazon SQS, you can send, store, and receive messages between software components at any volume, without losing messages or requiring other services to be available. Amazon SQS is a fully managed service which allows you to set up a queue in seconds. It also allows you to use your preferred SDK to start writing and reading to and from the queue within minutes.

In the following example, we describe an AWS architecture that brings together the Amazon SQS queue and an EC2 Auto Scaling group running Spot Instances. The architecture is used for decoupling the worker tier from the web tier by using Amazon SQS. The example uses the Protect feature (which we will explain later in this post) to ensure that an instance currently processing a job does not get terminated by the Auto Scaling group when it detects that a scale-in activity is required due to a Dynamic Scaling Policy.



AWS reference architecture used for decoupling the worker tier from the web tier by using Amazon SQS

Customer Example: How Trax Retail uses Auto Scaling groups with Spot Instances in their Amazon SQS application

Trax decided to run its queue worker tier exclusively on Spot Instances due to the fault-tolerant nature of its architecture and for cost-optimization purposes. The company digitizes the physical world of retail using Computer Vision. Their 'Trax Factory' transforms individual shelf into data and insights about retail store conditions.

Built using asynchronous event-driven architecture, Trax Factory is a cluster of microservices in which the completion of one service triggers the activation of another service. The worker tier uses Auto Scaling groups with dynamic scaling policies to increase and decrease the number of worker nodes in the worker tier.

You can create a Dynamic Scaling Policy by doing the following:

1. **Observe a Amazon CloudWatch metric.** Watch the metric for the current number of messages in the Amazon SQS queue (ApproximateNumberOfMessagesVisible).
2. **Create a CloudWatch alarm**. This alarm should be based on that metric you created in the prior step.
3. **Use your CloudWatch alarm** in a Dynamic Scaling Policy. Use this policy increase and decrease the number of EC2 Instances in the Auto Scaling group.

In Trax's case, due to the high variability of the number of messages in the queue, they opted to enhance this approach in order to minimize the time it takes to scale, by building a service that would call the SQS API and find the current number of messages in the queue more frequently, instead of waiting for the 5 minute metric refresh interval in CloudWatch.

Trax ensures that its applications are always scaled to meet the demand by leveraging the inherent elasticity of Amazon EC2 instances. This elasticity ensures that end users are never affected and/or service-level agreements (SLA) are never violated.

With a Dynamic Scaling Policy, the Auto Scaling group can detect when the number of messages in the queue has decreased, so that it can initiate a scale-in activity. The Auto Scaling group uses its configured termination policy for selecting the instances to be terminated. However, this policy poses the risk that the Auto Scaling group might select an instance for termination while that instance is currently processing an image. That instance's work would be lost (although the image would eventually be processed by reappearing in the queue and getting picked up by another worker node).

To decrease this risk, you can use Auto Scaling groups instance protection. This means that every time an instance fetches a job from the queue, it also sends an API call to EC2 to protect itself from scale-in. The Auto Scaling group does not select the protected, working instance for termination until the instance finishes processing the job and calls the API to remove the protection.

Handling Spot Instance interruptions

This instance-protection solution ensures that no work is lost during scale-in activities. However, protecting from scale-in does not work when an instance is marked for termination due to Spot Instance interruptions. These interruptions occur when there's increased demand for On-Demand Instances in the same capacity pool (a combination of an instance type in an Availability Zone).

Applications can minimize the impact of a Spot Instance interruption. To do so, an application catches the two-minute interruption notification (available in the instance's metadata), and instructs itself to stop fetching jobs from the queue. If there's an image still being processed when the two minutes expire and the instance is terminated, the application does not delete the message from the queue after finishing the process. Instead, the message simply becomes visible again for another instance to pick up and process after the Amazon SQS visibility timeout expires.

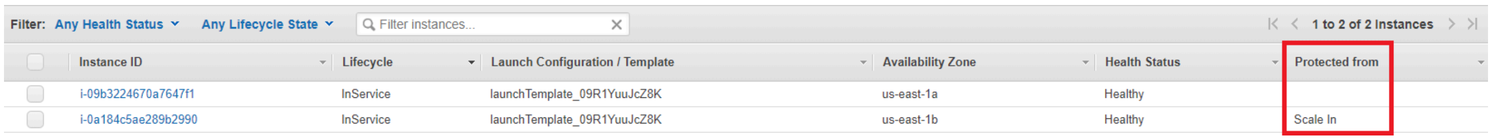
Alternatively, you can release any ongoing job back to the queue upon receiving a Spot Instance interruption notification by setting the visibility timeout of the specific message to 0. This timeout potentially decreases the total time it takes to process the message.

Testing the solution

If you're not currently using Spot Instances in your queue worker tier, we suggest testing the approach described in this post.

For that purpose, we built a simple solution to demonstrate the capabilities mentioned in this post, using an AWS CloudFormation template. The stack includes an Amazon Simple Storage Service (S3) bucket with a CloudWatch trigger to push notifications to an SQS queue after an image is uploaded to the Amazon S3 bucket. Once the message is in the queue, it is picked up by the application running on the EC2 instances in the Auto Scaling group. Then, the image is converted to PDF, and the instance is protected from scale-in for as long as it has an active processing job.

To see the solution in action, deploy the CloudFormation template. Then upload an image to the Amazon S3 bucket. In the Auto Scaling Groups console, check the instance protection status on the **Instances** tab. The protection status is shown in the following screenshot.



Filter: Any Health Status ▾ Any Lifecycle State ▾ Filter instances... 1 to 2 of 2 Instances

	Instance ID	Lifecycle	Launch Configuration / Template	Availability Zone	Health Status	Protected from
☐	i-09b3224670a7647f1	InService	launchTemplate_09R1YuuJcZ8K	us-east-1a	Healthy	
☐	i-0a184c5ae289b2990	InService	launchTemplate_09R1YuuJcZ8K	us-east-1b	Healthy	Scale In

You can also see the application logs using CloudWatch Logs:

```
/usr/local/bin/convert-worker.sh: Found 1 messages in https://sqs.us-east-1.amazonaws.com/123456789012/qtest-sqsQueue-1CL0NYLMX64OB
```

```
/usr/local/bin/convert-worker.sh: Found work to convert. Details: INPUT=Capture1.PNG, FNAME=capture1, FEXT=png
```

```
/usr/local/bin/convert-worker.sh: Running: aws autoscaling set-instance-protection --instance-ids i-0a184c5ae289b2990 --auto-scaling-group-name qtest-autoScalingGroup-QTGZX5N70POL --protected-from-scale-in
```

```
/usr/local/bin/convert-worker.sh: Convert done. Copying to S3 and cleaning up
```

```
/usr/local/bin/convert-worker.sh: Running: aws s3 cp /tmp/capture1.pdf s3://qtest-s3bucket-18fdpm2j17wxx
```

```
/usr/local/bin/convert-worker.sh: Running: aws sqs --output=json delete-message --queue-url https://sqs.us-east-1.amazonaws.com/123456789012/qtest-sqsQueue-1CL0NYLMX64OB --receipt-handle
```

```
/usr/local/bin/convert-worker.sh: Running: aws autoscaling set-instance-protection --instance-ids i-0a184c5ae289b2990 --auto-scaling-group-name qtest-autoScalingGroup-QTGZX5N70POL --no-protected-from-scale-in
```

Conclusion

This post helps you architect fault tolerant worker tiers in a cost optimized way. If your queue worker tiers are fault tolerant and use the built-in Amazon SQS features, you can increase your application's resilience and take advantage of Spot Instances to save up to 90% on compute costs.

In this post, we emphasized several best practices to help get you started saving money using Amazon SQS and Spot Instances. The main best practices are:

- Diversifying your Spot Instances using Auto Scaling groups, and selecting the right Spot allocation strategy
- Protecting instances from scale-in activities while they process jobs
- Using the Spot interruption notification so that the application stop polling the queue for new jobs before the instance is terminated

We hope you found this post useful. If you're not using Spot Instances in your queue worker tier, we suggest testing the approach described here. Finally, we would like to thank the Trax team for sharing its architecture and best practices. If you want to learn more, watch the "This is my architecture" video featuring Trax and their solution.

We'd love your feedback—please comment and let us know what you think.

About the authors

Ran Sheinberg is a specialist solutions architect for EC2 Spot Instances with Amazon Web Services. He works with AWS customers on cost optimizing their compute spend by utilizing Spot Instances across different types of workloads: stateless web applications, queue workers, containerized workloads, analytics, HPC and others.

As a Principal Developer Advocate for EC2 Spot at AWS, Chad's job is to make sure our customers are saving at scale by using EC2 Spot Instances to take advantage of the most cost-effective way to purchase compute capacity. Follow him on Twitter here! @schmutze

TAGS: Amazon EC2, Amazon EC2 Auto Scaling, Amazon EC2 Spot, Amazon SQS, microservices