# Agones FleetIQ Integration Guide

This guide presents a solution that uses AWS Spot Instances and GameLift FleetIQ to deliver low cost, reliable game servers hosting. The solution is designed to support your existing Agones games with no need for modifications.

## Overview

Game session servers are usually the most resource-intensive elements of a multiplayer architecture. Game developers need tools to provision, host, and scale servers while keeping hosting costs down.

[Agones](https://agones.dev/site/) is a popular tool for running, scaling and managing containerized game servers on Kubernetes. Game servers managed by Agones are fully supported for hosting on Amazon EC2 instances. With Amazon EC2, using [Spot Instances](https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/using-spot-instances.html) is a great way to reduce your hosting costs — you can save up to 90% compared to On-demand instances — by using spare EC2 capacity in AWS Cloud. Used alone, low-cost Spot capacity is not always available, and it’s possible that Spot Instances might be [interrupted](https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/spot-interruptions.html), potentially resulting in game session interruptions. [GameLift FleetIQ](https://docs.aws.amazon.com/gamelift/latest/developerguide/gsg-intro.html) makes Spot Instances viable for game hosting by minimizing the impact of interruptions on game servers.

Normally, a game would need to be [integrated](https://docs.aws.amazon.com/gamelift/latest/developerguide/gsg-getting-started.html) with FleetIQ to benefit from its optimizations — given that FleetIQ needs to communicate with game servers to track game session utilization and to manage resources. If a customer wants to also use Agones, the game would have to integrate with both Agones and FleetIQ at the same time. These requirements introduce extra complexity and breaks the resource abstraction layer provided by Agones.

This guide describes a way to run unmodified games that are built for Agones with FleetIQ provisioning the underlying EC2 instances to the Kubernetes cluster. This allows game servers to run on any Kubernetes cluster, and makes it possible to do local testing or hybrid deployments across multiple Kubernetes clusters either on-premises or in the cloud without having to modify the game server code.

### Concepts and definitions

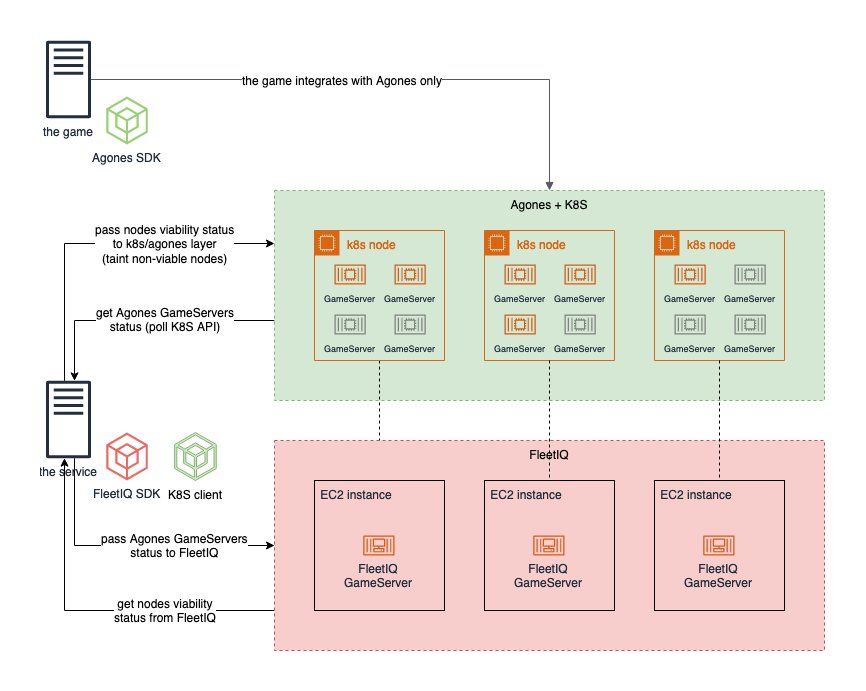
* Game server: A containerized server process that is running a customer-provided game server binary, and serves connected game clients during a game session.
* GameLift FleetIQ concepts:
  + [FleetIQ GameServer](https://docs.aws.amazon.com/gamelift/latest/developerguide/gsg-howitworks.html#gsg-howitworks-resources): FleetIQ representation of a game server process that is running on an EC2 instance in a FleetIQ-managed Auto-Scaling Group.
  + [FleetIQ GameServerGroup](https://docs.aws.amazon.com/gamelift/latest/developerguide/gsg-howitworks.html#gsg-howitworks-resources): FleetIQ representation of a set of compute resources and FleetIQ GameServer objects. It is linked to an EC2 Auto Scaling Group and contains configuration settings that define how FleetIQ works with it.
  + [FleetIQ ClaimGameServer](https://docs.aws.amazon.com/gamelift/latest/apireference/API_ClaimGameServer.html): FleetIQ operation to allocate an available FleetIQ GameServer in a FleetIQ GameServerGroupand temporarily reserve it for a game session. Typically issued by a client or a matchmaker.
  + FleetIQ instance type viability: FleetIQ concept that represents the chance of interruption for a particular EC2 Spot instance type.
* Agones concepts:
  + [Agones GameServer](https://agones.dev/site/docs/reference/gameserver/): Kubernetes resource to describe a game server’s configuration (container image, ports, etc.) and state. Corresponds to a Kubernetes pod.
  + [Agones Fleet](https://agones.dev/site/docs/reference/fleet/): Kubernetes resource to describe a set of warm Agones GameServers that are available to be allocated.
  + [Agones GameServerAllocation](https://agones.dev/site/docs/reference/gameserverallocation/):  Agones operation to allocate an Agones GameServer out of an Agones Fleet. Typically created by a client or a matchmaker.
  + [Agones Fleet Autoscaler](https://agones.dev/site/docs/reference/fleetautoscaler/): Kubernetes resource that scales the number of Agones GameServers in an Agones Fleet based on demand.
* [Kubernetes Cluster Autoscaler](https://github.com/kubernetes/autoscaler/tree/master/cluster-autoscaler): Kubernetes component that automatically adjusts the number of EC2 instances in an EC2 Auto-Scaling Group.

## Solution architecture

GameLift FleetIQ provides a logic layer to minimize the possibility of Spot Instance interruptions with your game servers. FleetIQ periodically evaluates the instance types in use and proactively replaces any instances that are not viable for game hosting. To support this process, FleetIQ needs to track active game servers to determine which instances are safe to recycle. Games servers that are integrated directly with FleetIQ through the AWS SDK regularly report their status to FleetIQ. However, if a game is integrated with Agones only, and Agones is in charge of allocating game servers, you need to set up a way to communicate game server status and instance viability between FleetIQ and Agones.

In this solution, we introduce an architecture to split responsibilities between Agones and FleetIQ layers and propose a supplementary service to pass information between them.

### Architectural layers

* Agones (and Kubernetes) provisions and scales game servers, allocates them to game sessions (through GameServerAllocation), and manages the lifecycle of game servers. The game integrates with Agones only and is unaware of GameLift FleetIQ. The Kubernetes Cluster Autoscaler scales worker nodes in an EC2 Auto Scaling Group.
* GameLift FleetIQ provides the viability status of instances, regularly updates the EC2 Auto Scaling Group configuration to use only viable instance types, and proactively drains and replaces non-viable instances.
* The supplementary service handles communication between the Agones and FleetIQ layers. It has the following responsibilities:
  + It monitors the status of instances in FleetIQ and uses Kubernetes cordons and taints to prevent Agones from using non-viable instances.
  + It registers exactly one FleetIQ GameServer on every instance and sets its status to UTILIZED in FleetIQ. This is not a real game server process, and is instead a way to tell FleetIQ when the instance might have allocated game servers and is not safe to replace.
  + It deregisters FleetIQ GameServers from non-viable instances when there are no more allocated Agones GameServers left on them. This is a way to tell FleetIQ that these instances can be recycled.

## Solution implementation

### Supplementary service

An implementation of the supplementary service is provided. It has three components packaged in a Helm chart:

* DaemonSet. A daemon that runs on every Kubernetes (k8s) worker node. It monitors FleetIQ viability status for the instance it is running on by subscribing to a Redis channel for the instance. It gracefully evicts all pods other than the currently allocated Agones GameServers from non-viable instances, and waits until all allocated GameServers finish (by polling the k8s API). It also registers and deregisters one FleetIQ GameServer on each node to notify FleetIQ when it’s being used by allocated Agones GameServers.
* Auxiliary pub/sub service (k8s deployment). It polls FleetIQ once a minute (using DescribeGameServerInstances API) to get viability status (ACTIVE, DRAINING or SPOT\_TERMINATING) for every instance in a GameServerGroup, and publishes status to the corresponding Redis channels (one channel per instance).
* Redis database. Pub/sub to communicate instance viability status to the DaemonSet. It has a separate channel for each instance.

Note: Instance viability status is communicated through Redis channels. This option was chosen over directly querying DescribeGameServerInstances API from the DaemonSet, because the throttling rate of DescribeGameServerInstances API might become a limitation in large clusters.

### Service architecture

### How non-viable instances get replaced

This is what happens when FleetIQ identifies an EC2 instance type in the Auto Scaling Group as non-viable for game hosting.

1. FleetIQ updates the Auto Scaling group configuration to temporarily remove non-viable instance type from the list of allowed instance types.
2. FleetIQ drains non-viable instances and triggers an Auto-Scaling Group to launch new instances using viable instance types.
3. The daemon receives notification through the Redis channel that the instance is in status DRAINING.
4. The daemon cordons the node in Kubernetes, adds a toleration to all allocated Agones GameServers, and then adds a taint to the node.
5. Because of taints and cordoning, no new Agones GameServers are scheduled onto the non-viable node. Available GameServers from the node are rescheduled to other nodes by k8s. Allocated Agones game servers remain running on the node until the game sessions end.
6. The daemon periodically polls the list of Agones GameServers through the k8s API and waits until all allocated Agones GameServers on the draining node have finished.
7. The daemon deregisters the FleetIQ GameServer on the draining instance, which tells FleetIQ that it can be recycled.
8. FleetIQ detaches and terminates the draining instance.

### How scaling works

Note: EC2 Auto Scaling policies are not used in this solution as the Auto Scaling Group is managed by the Kubernetes Cluster Autoscaler.

Scale-up:

1. The Agones Fleet is scaled up by Agones Fleet Autoscaler, usually based on the percentage of allocated Agones GameServers. Alternatively, you can create a webhook that implements your own scaling logic.
2. When there are not enough resources to schedule the required number of GameServers, the Kubernetes Cluster Autoscaler scales worker nodes. This is done by updating the “desired” property in an EC2 Auto Scaling Group.

Scale-down:

1. The Agones Fleet is scaled down by Agones Fleet Autoscaler or your own custom scaling logic.
2. Cluster Autoscaler can scale down nodes with no allocated Agones game servers.

## Deployment instructions

We provide a solution that helps to evaluate and start using FleetIQ and the supplementary service for games running on Agones. It includes a Cloud9 working environment, a Kubernetes cluster, Agones, a FleetIQ GameServerGroup (with corresponding EC2 Auto-Scaling Group), all necessary resources like IAM roles, and a sample game.

We provide three installation guides: full, quick and modular. Each guide deploys the same solution with the same components, but do this in different ways.

The Full Install Guide provides detailed manual steps. It covers creating a working environment (Cloud9 workspace with necessary IAM roles and Kubernetes tools), creating a EKS cluster, deploying the solution, and finally installing Agones and testing the solution. This guide can help to understand how all the pieces work together. It can also be used as a basis for further customizations to meet customers’ needs.

The Quick Install Guide provides a single script that automates the deployment of the solution. Deployed components are the same as in the Full Guide. This is a quick way to get a working sandbox environment from scratch.

The Modular Install Guide divides the deployment steps and scripts into four parts, so they can be run independently from each other. Use this guide if you already have a k8s cluster and/or a working environment with all necessary tools and permissions, and you want to selectively deploy solutions components on top.