Amazon GameLift FleetIQ Integration with Amazon EKS and Agones: Full Install Guide

April 5, 2021

Table of Contents

[Introduction 3](#_Toc68535177)

[Part 1. Set Up an Environment 3](#_Toc68535178)

[Create a Cloud9 workspace 3](#_Toc68535179)

[Add an IAM role to your workspace 4](#_Toc68535180)

[Create a new IAM role 4](#_Toc68535181)

[Attach the IAM role to your Cloud9 EC2 instance 5](#_Toc68535182)

[Update IAM settings for your workspace 5](#_Toc68535183)

[Install required cluster tools 6](#_Toc68535184)

[Part 2. Create a Cluster 6](#_Toc68535185)

[Setup eksctl 7](#_Toc68535186)

[Create a FleetIQ read policy 7](#_Toc68535187)

[Create and deploy a cluster 7](#_Toc68535188)

[Part 3. Configure Your Cluster 8](#_Toc68535189)

[Create a new node group from an existing node group configuration 8](#_Toc68535190)

[Set up a FleetIQ game server group 11](#_Toc68535191)

[Create a new FleetIQ game server group 11](#_Toc68535192)

[Tag the Auto Scaling Group 12](#_Toc68535193)

[Configure the cluster autoscaler 13](#_Toc68535194)

[Set up node monitoring with a Helm chart 17](#_Toc68535195)

[Install Helm 17](#_Toc68535196)

[Give the GameLift-daemonset access to monitoring data 17](#_Toc68535197)

[Install the Helm charts from ECR 18](#_Toc68535198)

[Part 4. Set Up Agones 19](#_Toc68535199)

[Install Agones using Helm 19](#_Toc68535200)

[Deploy sample Agones fleet 19](#_Toc68535201)

[Test your FleetIQ-EKS-Agones installation 21](#_Toc68535202)

[Part 5. Add a Second Nodegroup 21](#_Toc68535203)

[Set up a second FleetIQ game server group 24](#_Toc68535204)

[Create a new FleetIQ game server group 24](#_Toc68535205)

[Tag the Auto Scaling Group 25](#_Toc68535206)

[Deploy the GameLift Daemonset on New Game Server Group 25](#_Toc68535207)

[Cluster Autoscaler Manifest Update 26](#_Toc68535208)

[Test your FleetIQ-EKS-Agones installation 32](#_Toc68535209)

[Document Revisions 33](#_Toc68535210)

# Introduction

This guide provides step-by-step instructions for setting up the FleetIQ-EKS-Agones solution.

Note: Parameters that are reused later in the process are highlighted so you can keep track of changes made to the default values.

This guide covers the install and deployment process in five parts:

1. Set up an environment.

You will deploy a Cloud9 IDE environment to deploy, configure and control the components of this solution.

1. Set up an EKS cluster.

You will configure and deploy and EKS cluster using kubectl.

1. Configure the EKS cluster with FleetIQ.

You will configure the necessary components to deploy and configure the solution (EC2 launch configuration, FleetIQ game server group, FleetIQ-daemonset Helm chart).

1. Configure Agones fleets.

You will deploy Agones along with a sample game server fleet deployment.

1. (Optional) Configure a second EKS nodegroup

# Part 1. Set Up an Environment

Set up a cloud-based IDE using AWS Cloud9 that provides an isolated CLI environment from which you can configure and control the cluster (recommended for initial sandbox tests). This section covers the following tasks:

* Create a Cloud9 workspace.
* Set up account credentials for your workspace.
* Install a required set of tools on your workspace.

## Create a Cloud9 workspace

1. In the AWS Management Console, open the Cloud9 service. Select the region where you want to run the cluster. You can use any of the following regions:

* US East (Ohio)
* US East (N. Virginia)
* US West (Oregon)
* Asia Pacific (Hong Kong)
* Asia Pacific (Mumbai)
* Asia Pacific (Seoul)
* Asia Pacific (Singapore)
* Asia Pacific (Sydney)
* Asia Pacific (Tokyo)
* Canada (Central)
* Europe (Frankfurt)
* Europe (Ireland)
* Europe (London)
* Europe (Paris)
* Europe (Stockholm)
* Middle East (Bahrain)
* South America (São Paulo)

1. Click the button **Create environment**.
2. Enter a name (such as “EKS-FleetIQ”) and optional description and click **Next step**.
3. Select the following options:
   * **Environment type**: Create a new no-ingress EC2 instance for environment (access via Systems Manager)
   * **Instance type**: t3.small
   * **Platform**: Amazon Linux 2
   * **Cost-saving setting**: After one hour
   * **IAM role**: (Leave this field blank for now.)
   * **Network settings (advanced)**: Select a VPC/Subnet option (default selections are OK), or create a new VPC and subnet based on your network environment.
4. Click **Next step** to review your environment settings. Make changes as needed.
5. When ready, click **Create environment**. It can take a few minutes to create the environment. When the development environment is completed, a Welcome screen is displayed.
6. You can customize your new environment either in the lower work area or by opening a new terminal window in the main work area by clicking the green plus (+) sign. You’ll use this work area to do additional customizations as covered in this guide.

**NOTE:** Please use the same terminal window throughout this guide to ensure environment variables are preserved between steps.

## Add an IAM role to your workspace

Create an administrator IAM role to manage permissions for all of the AWS components that are used in the FleetIQ-EKS-Agones solution. These permissions are required before you can complete the FleetIQ-EKS-Agones install process.

### Create a new IAM role

1. In the AWS Console, go to the IAM service. IAM is a global service, so you don’t need to specify a region.
2. In the sidebar, select **Roles** and click the **Create Role** button.
3. Select the following options:
   * **Select type of trusted entity**: AWS service.
   * **Choose a use case**: EC2.
4. Click **Next:Permissions**.
5. Select the permission policy **AdministratorAccess**.
6. Click **Next:Tags** to move forward, take the defaults, and click **Next:Review** to finalize the new role.
7. Enter the role name: “agonesfleetiq-admin” and add an optional role description.
8. Click **Create role**.

### Attach the IAM role to your Cloud9 EC2 instance

1. In the AWS Console, go to the EC2 service and select the region where you created your Cloud9 environment. (An EC2 instance was automatically created to run your Cloud9 environment).
2. In the **Resources** box, click **Running instances**. When you created your Cloud9 environment, an EC2 instance was automatically created for it. You should see this instance listed with a name that starts with “aws-cloud9…”.
3. Select your active Cloud9 instance (click the checkbox) and choose **Actions / Instance Settings / Modify IAM Role** (you may need to scroll the dropdown list).
4. From the **IAM Role** drop down list, choose the role “agonesfleetiq-admin” and select **Save**.

### Update IAM settings for your workspace

1. In the AWS Console, return to the Cloud9 service and open your IDE workspace. Be sure you’re working in the region where you created your workspace.
2. Click the gear icon (in top right corner), or open a new tab and choose **Open Preferences**.
3. Open **Project Settings: AWS Settings** and turn off AWS managed temporary credentials.
4. Open a terminal window and enter the following commands to finish updating your IAM settings.
   * Remove any existing credentials file:  
     rm -vf ${HOME}/.aws/credentials
   * Install jq:  
     sudo yum -y install jq
   * Configure the AWS CLI with your workspace region as default:  
     export ACCOUNT\_ID=$(aws sts get-caller-identity --output text --query Account)  
       
     export AWS\_REGION=$(curl -s 169.254.169.254/latest/dynamic/instance-identity/document | jq -r '.region')
   * Verify that AWS\_REGION is correctly set to your desired region:  
     test -n "$AWS\_REGION" && echo AWS\_REGION is "$AWS\_REGION" || echo AWS\_REGION is not set
   * Save these settings into bash\_profile:  
     echo "export ACCOUNT\_ID=${ACCOUNT\_ID}" | tee -a ~/.bash\_profile  
       
     echo "export AWS\_REGION=${AWS\_REGION}" | tee -a ~/.bash\_profile  
       
     aws configure set default.region ${AWS\_REGION}  
       
     aws configure get default.region
   * Use the [GetCallerIdentity](https://docs.aws.amazon.com/cli/latest/reference/sts/get-caller-identity.html) CLI command to validate that the Cloud9 IDE is using the correct IAM role:  
     aws sts get-caller-identity --query Arn | grep agonesfleetiq-admin -q && echo "IAM role valid" || echo "IAM role NOT valid"  
       
     If the IAM role is not valid, DO NOT PROCEED with cluster creation. Some possible reasons that your IAM role isn’t valid:
     + You named your IAM role differently. Modify the get-caller-identity command (between | grep and -q) with the correct name.
     + You had to reconnect to the instance. This may have re-applied the Managed Temporary Credentials. Restart the “Update IAM settings for your workspace“ at Step 1.

## Install required cluster tools

In your Cloud9 workspace, use your open terminal window and enter the following commands to install some required tools on your workspace.

1. Install kubectl, the command line interface for controlling Kubernetes clusters:

sudo curl --silent --location -o /usr/local/bin/kubectl \  
https://amazon-eks.s3.us-west-2.amazonaws.com/1.18.9/2020-11-02/bin/linux/amd64/kubectl

sudo chmod +x /usr/local/bin/kubectl

1. Get the latest version of the AWS command line interface (CLI):

sudo pip install --upgrade awscli && hash -r

1. Install helpful utilities:

sudo yum -y install gettext bash-completion moreutils

1. Install yq for yaml processing:

echo 'yq() {  
 docker run --rm -i -v "${PWD}":/workdir mikefarah/yq yq "$@"  
 }' | tee -a ~/.bashrc && source ~/.bashrc

1. Verify that the binaries are in the path and executable:

for command in kubectl jq envsubst aws  
do  
which $command &>/dev/null && echo "$command in path" || echo "$command NOT FOUND"  
done

# Part 2. Create a Cluster

Create and deploy your EKS cluster. This section covers the following tasks:

* Set up eksctl, a tool for managing EKS clusters.
* Create an IAM role to poll node status from FleetIQ.
* Configure and deploy an EKS cluster.

All of the steps in this part are done in your open terminal window in your Cloud9 workspace.

## Setup eksctl

Install the command line tool for creating clusters on EKS.

Download the eksctl binary:

**curl --silent --location "https://github.com/weaveworks/eksctl/releases/latest/download/eksctl\_$(uname -s)\_amd64.tar.gz" | tar xz -C /tmp  
  
sudo mv -v /tmp/eksctl /usr/local/bin**

Confirm that eksctl is set up:

eksctl version

## Create a FleetIQ read policy

Extend permissions to allow the EKS cluster to call the GameLift service and get properties and status for FleetIQ game server groups, game servers, and game server instances. To do this, create a new IAM policy with a limited permissions policy.

aws iam create-policy --policy-name FleetIQpermissionsEC2 --policy-document '{"Version": "2012-10-17","Statement": [{"Sid": "VisualEditor0","Effect": "Allow","Action": ["gamelift:DescribeGameServerGroup","gamelift:DescribeGameServerInstances","gamelift:DescribeGameServer"],"Resource": "\*"}]}'

## Create and deploy a cluster

Create a deployment file by running the following script. Cluster creation takes 10-15 minutes.

cat <<EOF > config.yaml  
---  
apiVersion: eksctl.io/v1alpha5  
kind: ClusterConfig  
metadata:  
 name: agones  
 region: ${AWS\_REGION}  
 version: "1.18"  
availabilityZones: ["${AWS\_REGION}a", "${AWS\_REGION}b", "${AWS\_REGION}c"]  
nodeGroups:  
 - name: ng-system  
 instanceType: m5.large  
 desiredCapacity: 2

ssh:

enableSsm: true

iam:  
 attachPolicyARNs:  
 - arn:aws:iam::aws:policy/AmazonEKSWorkerNodePolicy  
 - arn:aws:iam::aws:policy/AmazonEKS\_CNI\_Policy  
 - arn:aws:iam::aws:policy/ElasticLoadBalancingFullAccess

- arn:aws:iam::aws:policy/AmazonSSMManagedInstanceCore  
 - arn:aws:iam::${ACCOUNT\_ID}:policy/FleetIQpermissionsEC2  
 - name: ng-agones  
 instanceType: m5.xlarge  
 desiredCapacity: 2

ssh:

enableSsm: true

labels:  
 agones.dev/agones-system: "true"  
 taints:  
 agones.dev/agones-system: "true:NoExecute"  
 iam:  
 attachPolicyARNs:  
 - arn:aws:iam::aws:policy/AmazonEKSWorkerNodePolicy  
 - arn:aws:iam::aws:policy/AmazonEKS\_CNI\_Policy  
 - arn:aws:iam::aws:policy/ElasticLoadBalancingFullAccess

- arn:aws:iam::aws:policy/AmazonSSMManagedInstanceCore  
 - arn:aws:iam::${ACCOUNT\_ID}:policy/FleetIQpermissionsEC2  
EOF

eksctl create cluster -f config.yaml

# Part 3. Configure Your Cluster

Configure a new node group that uses an EC2 Auto Scaling Group (provisioned by FleetIQ) to provision new nodes. This section covers the following tasks:

* Configure a node group.
* Create a FleetIQ game server group and EC2 Auto Scaling Group.
* Install and configure the Cluster Autoscaler to use the FleetIQ-generated EC2 Auto Scaling Group to provision nodes for Agones.
* Install a Helm chart to monitor FleetIQ viability status for provisioning.

## Create a new node group from an existing node group configuration

In this step, you will use the configuration of an existing node group on the cluster to set up a new node group, by customizing the launch template and security groups to allow player access to these nodes on UDP 7000-8000 (Agones defaults).

In your Cloud9 workspace, use your open terminal window and enter the following commands:

1. Create launch template user data.

NG\_STACK=$(aws cloudformation describe-stacks --region ${AWS\_REGION}| jq -r '.Stacks[] | .StackId' | grep ng-system)  
  
LAUNCH\_TEMPLATE\_ID=$(aws cloudformation describe-stack-resources --region ${AWS\_REGION} --stack-name $NG\_STACK \  
| jq -r '.StackResources | map(select(.LogicalResourceId == "NodeGroupLaunchTemplate")  
| .PhysicalResourceId)[0]')  
  
aws ec2 describe-launch-template-versions --region ${AWS\_REGION} --launch-template-id $LAUNCH\_TEMPLATE\_ID \  
| jq -r '.LaunchTemplateVersions[0].LaunchTemplateData.UserData' \  
| base64 -d | gunzip > launchtemplate.yaml  
  
awk -v var="$(grep -n NODE\_LABELS=alpha ./launchtemplate.yaml | cut -d : -f 1)" 'NR==var {$0=" NODE\_LABELS=alpha.eksctl.io/cluster-name=agones,alpha.eksctl.io/nodegroup-name=game-servers-1,role=game-servers-1"} 1' launchtemplate.yaml > templt.yaml

awk -v var="$(grep -n NODE\_TAINTS= ./launchtemplate.yaml | cut -d : -f 1)" 'NR==var {$0=" NODE\_TAINTS=agones.dev/gameservers=true:NoExecute"} 1' templt.yaml > modlaunchtemplate.yaml

rm templt.yaml

base64 -w 0 modlaunchtemplate.yaml > b64modlaunchtemplate

1. Create the launch template.

VPCID=$(aws ec2 describe-vpcs --region ${AWS\_REGION} --filter Name=tag:alpha.eksctl.io/cluster-name,Values=agones | jq -r '.Vpcs[0].VpcId')

SGID=$(aws ec2 create-security-group --region ${AWS\_REGION} --description "Agones nodegroup SG" --group-name eksctl-agones-nodegroup-ng-1-SG --vpc-id ${VPCID} | jq -r '.GroupId')

SGINGRESSRULES=$(aws ec2 describe-security-groups --region ${AWS\_REGION} --filter Name=tag:alpha.eksctl.io/nodegroup-name,Values=ng-system | jq '.SecurityGroups[0].IpPermissions')

echo '{"GroupId":"'$SGID'","IpPermissions":'$SGINGRESSRULES'}' > sgingress.json

aws ec2 authorize-security-group-ingress --cli-input-json file://sgingress.json --region ${AWS\_REGION}

aws ec2 authorize-security-group-ingress --group-id ${SGID} --ip-permissions FromPort=7000,IpProtocol="udp",IpRanges=[{CidrIp="0.0.0.0/0"}],ToPort=8000 --region ${AWS\_REGION}

IAMINSTANCEPROFILE=$(aws ec2 describe-launch-template-versions --region ${AWS\_REGION} --launch-template-id $LAUNCH\_TEMPLATE\_ID | jq '.LaunchTemplateVersions[0].LaunchTemplateData.IamInstanceProfile.Arn')

NG0SG1=$(aws ec2 describe-launch-template-versions --region ${AWS\_REGION} --launch-template-id $LAUNCH\_TEMPLATE\_ID | jq '.LaunchTemplateVersions[0].LaunchTemplateData.NetworkInterfaces[0].Groups[0]')

NG0SG2=$(aws ec2 describe-launch-template-versions --region ${AWS\_REGION} --launch-template-id $LAUNCH\_TEMPLATE\_ID | jq '.LaunchTemplateVersions[0].LaunchTemplateData.NetworkInterfaces[0].Groups[1]')

NG0AMI=$(aws ec2 describe-launch-template-versions --region ${AWS\_REGION} --launch-template-id $LAUNCH\_TEMPLATE\_ID | jq '.LaunchTemplateVersions[0].LaunchTemplateData.ImageId')

B64USERDATA=$(cat b64modlaunchtemplate)

cat << EOF > ltinput.json  
{  
"LaunchTemplateName": "eksctl-agones-nodegroup-ng-1",  
"VersionDescription": "FleetIQ GameServerGroup LT",  
"LaunchTemplateData": {  
"EbsOptimized": true,  
"IamInstanceProfile": {  
"Arn": ${IAMINSTANCEPROFILE}  
},  
"BlockDeviceMappings": [  
{  
"DeviceName": "/dev/xvda",  
"Ebs": {  
"Encrypted": false,  
"DeleteOnTermination": true,  
"VolumeSize": 80,  
"VolumeType": "gp2"  
}  
}  
],  
"NetworkInterfaces": [  
{  
"DeviceIndex": 0,  
"Groups": [  
${NG0SG1},  
${NG0SG2},  
"${SGID}"  
]  
}  
],  
"ImageId": $NG0AMI,  
"Monitoring": {  
"Enabled": true  
},  
"UserData": "${B64USERDATA}",  
"TagSpecifications": [  
{  
"ResourceType": "instance",  
"Tags": [

{  
"Key": "kubernetes.io/cluster/agones",  
"Value": "owned"  
},  
{  
"Key": "k8s.io/cluster-autoscaler/agones",  
"Value": "owned"  
},

{  
"Key": "k8s.io/cluster-autoscaler/enabled",  
"Value": "true"  
},  
{  
"Key": "Name",  
"Value": "FleetIQ"  
}  
]  
}  
],  
"MetadataOptions": {  
"HttpTokens": "optional",  
"HttpPutResponseHopLimit": 2  
}  
}  
}  
EOF

GSGLTID=$(aws ec2 create-launch-template --cli-input-json file://ltinput.json --region ${AWS\_REGION} | jq '.LaunchTemplate.LaunchTemplateId')

## Set up a FleetIQ game server group

Set up a FleetIQ game server group to track the game-hosting viability of EC2 instances types in your node group. Creating a new FleetIQ game server group automatically creates a corresponding new EC2 Auto Scaling Group.

In your Cloud9 workspace, use your open terminal window and enter the following commands:

### Create a new FleetIQ game server group

1. Create an IAM FleetIQ service role [that allows GameLift to create and update EC2 Auto Scaling Groups.](https://docs.aws.amazon.com/gamelift/latest/developerguide/gsg-iam-permissions-roles.html#gsg-iam-permissions-roles-gamelift)

GSGROLEARN=$(( aws iam create-role --role-name GameLiftServerGroupRole --assume-role-policy-document '{"Version":"2012-10-17","Statement":[{"Effect":"Allow","Principal":{"Service":["gamelift.amazonaws.com","autoscaling.amazonaws.com"]},"Action":"sts:AssumeRole"}]}' | jq '.Role.Arn' ) 2>&1)

aws iam attach-role-policy --role-name GameLiftServerGroupRole --policy-arn arn:aws:iam::aws:policy/GameLiftGameServerGroupPolicy

1. Create the game server group and EC2 Auto Scaling Group.

NOTE: The instance types you select must have identical vCPU and memory allocations due to the Kubernetes Cluster Autoscaler requirements.

GSGNAME="agones-game-server-group-01"

PUBLICSUBNETIDS=$(aws ec2 describe-subnets --filters Name=vpc-id,Values=$VPCID --region ${AWS\_REGION} | jq '[.Subnets[] | {subnetid: .SubnetId, mapPublicIP: .MapPublicIpOnLaunch}]' | jq 'group\_by(.mapPublicIP)' | jq '[.[1][].subnetid]')

cat << EOF > gsgconfig.json  
{  
"GameServerGroupName": "${GSGNAME}",  
"RoleArn": ${GSGROLEARN},  
"MinSize": 1,  
"MaxSize": 10,  
"LaunchTemplate": {  
"LaunchTemplateId": ${GSGLTID},  
"Version": "1"  
},  
"InstanceDefinitions": [  
{  
"InstanceType": "c5.xlarge",  
"WeightedCapacity": "2"  
},  
{  
"InstanceType": "c4.xlarge",  
"WeightedCapacity": "1"  
}  
],  
"BalancingStrategy": "SPOT\_PREFERRED",  
"GameServerProtectionPolicy": "FULL\_PROTECTION",  
"VpcSubnets": ${PUBLICSUBNETIDS}  
}  
EOF

aws gamelift create-game-server-group --region ${AWS\_REGION} --cli-input-json <file://gsgconfig.json>

1. Check the game server group status until it reports back as ACTIVE. The describe-game-server-group command returns the game server group properties, including status.

aws gamelift describe-game-server-group --game-server-group-name ${GSGNAME} --region $AWS\_REGION | jq '.GameServerGroup.Status'

### Tag the Auto Scaling Group

Assign resource tags to the new Auto Scaling Group. The tags allow the cluster autoscaler to use the ASG to provision nodes for Agones.

aws autoscaling create-or-update-tags --tags ResourceId=gamelift-gameservergroup-${GSGNAME},ResourceType=auto-scaling-group,Key=kubernetes.io/cluster/agones,Value=owned,PropagateAtLaunch=true ResourceId=gamelift-gameservergroup-${GSGNAME},ResourceType=auto-scaling-group,Key=k8s.io/cluster-autoscaler/agones,Value=enabled,PropagateAtLaunch=true ResourceId=gamelift-gameservergroup-${GSGNAME},ResourceType=auto-scaling-group,Key=Name,Value=FleetIQ,PropagateAtLaunch=true ResourceId=gamelift-gameservergroup-${GSGNAME},ResourceType=auto-scaling-group,Key=k8s.io/cluster-autoscaler/enabled,Value=true,PropagateAtLaunch=true --region ${AWS\_REGION}

## Configure the cluster autoscaler

In your Cloud9 workspace, open a terminal window and enter the following commands:

1. Create an OIDC endpoint for the cluster.

eksctl utils associate-iam-oidc-provider --cluster agones --approve

1. Create a cluster autoscaler policy and service account/IAM role mapping.

cat << EOF > capolicy.json  
{  
"Version": "2012-10-17",  
"Statement": [  
{  
"Effect": "Allow",  
"Action": [  
"autoscaling:DescribeAutoScalingGroups",  
"autoscaling:DescribeAutoScalingInstances",  
"autoscaling:DescribeLaunchConfigurations",  
"autoscaling:DescribeTags",  
"autoscaling:SetDesiredCapacity",  
"autoscaling:TerminateInstanceInAutoScalingGroup"  
],  
"Resource": "\*"  
}  
]  
}  
EOF

aws iam create-policy --policy-name cluster-autoscaler-policy --policy-document file://capolicy.json

eksctl create iamserviceaccount --cluster agones --namespace kube-system --name cluster-autoscaler --attach-policy-arn arn:aws:iam::${ACCOUNT\_ID}:policy/cluster-autoscaler-policy --override-existing-serviceaccounts --approve

1. Create a cluster autoscaler manifest and apply it.

The Auto Scaling Group name in the Cluster Autoscaler manifest is generated based on the GSGNAME variable you set earlier. A prefix of “gamelift-gameservergroup-” is added by GameLift when it creates the Auto Scaling Group for each GameLift Game Server Group. This Auto Scaling Group name is the value that must be used in the Kubernetes Cluster Autoscaler manifest below (not the GameLift Game Server Group name).

Regarding Cluster Autoscaler priority values, the larger numerical value indicates to the Cluster Autoscaler which game server group you wish to prioritize adding compute capacity to. You can read more [here](https://github.com/kubernetes/autoscaler/blob/master/cluster-autoscaler/expander/priority/readme.md). Even though we only have one GameLift Game Server Group now, we configure the Cluster Autoscaler this way for extensibility in case you add more server groups in the future.

cat << EOF > camanifest.yaml  
---  
apiVersion: v1  
kind: ServiceAccount  
metadata:  
 labels:  
 k8s-addon: cluster-autoscaler.addons.k8s.io  
 k8s-app: cluster-autoscaler  
 name: cluster-autoscaler  
 namespace: kube-system  
---  
apiVersion: rbac.authorization.k8s.io/v1  
kind: ClusterRole  
metadata:  
 name: cluster-autoscaler  
 labels:  
 k8s-addon: cluster-autoscaler.addons.k8s.io  
 k8s-app: cluster-autoscaler  
rules:  
 - apiGroups: [""]  
 resources: ["events", "endpoints"]  
 verbs: ["create", "patch"]  
 - apiGroups: [""]  
 resources: ["pods/eviction"]  
 verbs: ["create"]  
 - apiGroups: [""]  
 resources: ["pods/status"]  
 verbs: ["update"]  
 - apiGroups: [""]  
 resources: ["endpoints"]  
 resourceNames: ["cluster-autoscaler"]  
 verbs: ["get", "update"]  
 - apiGroups: [""]  
 resources: ["nodes"]  
 verbs: ["watch", "list", "get", "update"]  
 - apiGroups: [""]  
 resources:  
 - "pods"  
 - "services"  
 - "replicationcontrollers"  
 - "persistentvolumeclaims"  
 - "persistentvolumes"  
 verbs: ["watch", "list", "get"]  
 - apiGroups: ["extensions"]  
 resources: ["replicasets", "daemonsets"]  
 verbs: ["watch", "list", "get"]  
 - apiGroups: ["policy"]  
 resources: ["poddisruptionbudgets"]  
 verbs: ["watch", "list"]  
 - apiGroups: ["apps"]  
 resources: ["statefulsets", "replicasets", "daemonsets"]  
 verbs: ["watch", "list", "get"]  
 - apiGroups: ["storage.k8s.io"]  
 resources: ["storageclasses", "csinodes"]  
 verbs: ["watch", "list", "get"]  
 - apiGroups: ["batch", "extensions"]  
 resources: ["jobs"]  
 verbs: ["get", "list", "watch", "patch"]  
 - apiGroups: ["coordination.k8s.io"]  
 resources: ["leases"]  
 verbs: ["create"]  
 - apiGroups: ["coordination.k8s.io"]  
 resourceNames: ["cluster-autoscaler"]  
 resources: ["leases"]  
 verbs: ["get", "update"]  
---  
apiVersion: rbac.authorization.k8s.io/v1  
kind: Role  
metadata:  
 name: cluster-autoscaler  
 namespace: kube-system  
 labels:  
 k8s-addon: cluster-autoscaler.addons.k8s.io  
 k8s-app: cluster-autoscaler  
rules:  
 - apiGroups: [""]  
 resources: ["configmaps"]  
 verbs: ["create","list","watch"]  
 - apiGroups: [""]  
 resources: ["configmaps"]  
 resourceNames: ["cluster-autoscaler-status", "cluster-autoscaler-priority-expander"]  
 verbs: ["delete", "get", "update", "watch"]  
---  
apiVersion: rbac.authorization.k8s.io/v1  
kind: ClusterRoleBinding  
metadata:  
 name: cluster-autoscaler  
 labels:  
 k8s-addon: cluster-autoscaler.addons.k8s.io  
 k8s-app: cluster-autoscaler  
roleRef:  
 apiGroup: rbac.authorization.k8s.io  
 kind: ClusterRole  
 name: cluster-autoscaler  
subjects:  
 - kind: ServiceAccount  
 name: cluster-autoscaler  
 namespace: kube-system  
---  
apiVersion: rbac.authorization.k8s.io/v1  
kind: RoleBinding  
metadata:  
 name: cluster-autoscaler  
 namespace: kube-system  
 labels:  
 k8s-addon: cluster-autoscaler.addons.k8s.io  
 k8s-app: cluster-autoscaler  
roleRef:  
 apiGroup: rbac.authorization.k8s.io  
 kind: Role  
 name: cluster-autoscaler  
subjects:  
 - kind: ServiceAccount  
 name: cluster-autoscaler  
 namespace: kube-system  
---  
apiVersion: apps/v1  
kind: Deployment  
metadata:  
 name: cluster-autoscaler  
 namespace: kube-system  
 labels:  
 app: cluster-autoscaler  
 annotations:  
 cluster-autoscaler.kubernetes.io/safe-to-evict: 'false'  
spec:  
 replicas: 1  
 selector:  
 matchLabels:  
 app: cluster-autoscaler  
 template:  
 metadata:  
 labels:  
 app: cluster-autoscaler  
 annotations:  
 prometheus.io/scrape: 'true'  
 prometheus.io/port: '8085'  
 spec:  
 serviceAccountName: cluster-autoscaler  
 containers:  
 - image: k8s.gcr.io/autoscaling/cluster-autoscaler:v1.18.3  
 name: cluster-autoscaler  
 resources:  
 limits:  
 cpu: 100m  
 memory: 300Mi  
 requests:  
 cpu: 100m  
 memory: 300Mi  
 command:

- ./cluster-autoscaler

- --v=4

- --stderrthreshold=info

- --cloud-provider=aws

- --skip-nodes-with-local-storage=false

- --expander=priority

- --nodes=0:10:gamelift-gameservergroup-${GSGNAME}

- --balance-similar-node-groups

- --skip-nodes-with-system-pods=false

env:

- name: AWS\_REGION

value: ${AWS\_REGION}

volumeMounts:

- name: ssl-certs

mountPath: /etc/ssl/certs/ca-certificates.crt

readOnly: true

imagePullPolicy: "Always"

volumes:

- name: ssl-certs

hostPath:

path: "/etc/ssl/certs/ca-bundle.crt"

---

apiVersion: v1

kind: ConfigMap

metadata:

name: cluster-autoscaler-priority-expander

namespace: kube-system

data:

priorities: |-

10:

- .\*-non-existing-entry.\*

20:

- gamelift-gameservergroup-${GSGNAME}  
EOF

kubectl apply -f camanifest.yaml

## Set up node monitoring with a Helm chart

Set up a Helm chart to monitor each node for game hosting viability. This component consists of a pub/sub service that polls FleetIQ for node status and loads the response into an on-cluster Redis table. This table is accessed by the daemon installed on each Agones node, and the information is used to provision, cordon, and terminate node/instances as needed.

All of the steps in this part are done in the open terminal window in your Cloud9 workspace.

### Install Helm

1. Install the Helm command line interface.

curl -sSL https://raw.githubusercontent.com/helm/helm/master/scripts/get-helm-3 | bash

1. Check that it works; you will see the Helm version installed:­

helm version --short

### Give the GameLift-daemonset access to monitoring data

Create a new IAM permissions policy that allows the daemonset to poll for node status. This is required before installing the Helm chart containing the daemonset.

cat << EOF > gameliftdaemonpolicy.json  
{  
 "Version": "2012-10-17",  
 "Statement": [  
 {  
 "Sid": "VisualEditor0",  
 "Effect": "Allow",  
 "Action": [  
 "autoscaling:DescribeAutoScalingInstances",  
 "gamelift:\*"  
 ],  
 "Resource": "\*"  
 }  
 ]  
}  
EOF

aws iam create-policy --policy-name gamelift-daemon-policy --policy-document file://gameliftdaemonpolicy.json

eksctl create iamserviceaccount --cluster agones --name gamelift-daemonset --namespace kube-system --attach-policy-arn arn:aws:iam::${ACCOUNT\_ID}:policy/gamelift-daemon-policy --override-existing-serviceaccounts --approve

### Install the Helm charts from ECR

export HELM\_EXPERIMENTAL\_OCI=1

aws ecr get-login-password --region us-west-2 | helm registry login --username AWS --password-stdin 820537372947.dkr.ecr.us-west-2.amazonaws.com

helm chart pull 820537372947.dkr.ecr.us-west-2.amazonaws.com/gamelift-daemon:0.1.1

helm chart pull 820537372947.dkr.ecr.us-west-2.amazonaws.com/gamelift-common-services:0.1.0

helm chart export 820537372947.dkr.ecr.us-west-2.amazonaws.com/gamelift-daemon:0.1.1

helm chart export 820537372947.dkr.ecr.us-west-2.amazonaws.com/gamelift-common-services:0.1.0

**NOTE:** If you chose a different value for the GSGNAME variable (i.e., something other than "agones-game-server-group-01", please update the fleetiq.conf value appropriately in gamelift-common-services/templates/deployment.yaml

helm install \

--set aws.region=${AWS\_REGION} \

gamelift-common-services ./gamelift-common-services/

helm install \

--set aws.region=${AWS\_REGION} \

--set gameliftDaemon.serviceAccount=gamelift-daemonset \

--set gameliftDaemon.nodeSelector=game-servers-1 \

--set gameliftDaemon.gameServerGroupName=${GSGNAME} \

gamelift-daemonset ./gamelift-daemonset/

# Part 4. Set Up Agones

As a final step, set up Agones on your Cloud9 workspace. With this final component in place, you can create Agones fleets and test your installation of the FleetIQ-EKS-Agones solution. This section covers the following tasks:

* Install Helm.
* Create and deploy a Agones fleet with the open-source game Super Tux Cart.
* Test your installation with the sample game deployment.

## Install Agones using Helm

1. Add the Agones stable repo:

helm repo add agones https://agones.dev/chart/stable

helm repo update

1. Install:

helm install my-release --namespace agones-system --create-namespace agones/agones

## Deploy sample Agones fleet

Create and deploy a sample Agones fleet that spawns server instances of the open-source game Super Tux Cart.

All fleets that use the FleetIQ-EKS-Agones solution must be configured with specific nodeSelector and Tolerations configurations to ensure that Agones is able to use the nodes provisioned via FleetIQ. These settings are highlighted and bolded in the following configuration file.

cat << EOF > agonesfleetconfig.yaml

kind: Fleet

apiVersion: agones.dev/v1

metadata:

annotations:

agones.dev/sdk-version: 1.8.0

name: stk-fleet

namespace: default

spec:

replicas: 5

scheduling: Packed

strategy:

rollingUpdate:

maxSurge: 25%

maxUnavailable: 25%

type: RollingUpdate

template:

metadata:

labels:

app: stk

spec:

health:

failureThreshold: 3

initialDelaySeconds: 30

periodSeconds: 10

ports:

- containerPort: 8080

name: default

portPolicy: Dynamic

sdkServer: {}

template:

metadata: {}

spec:

containers:

- command:

- /start.sh

env:

- name: WAIT\_TO\_PLAYERS

value: '120'

- name: FREQ\_CHECK\_SESSION

value: '10'

- name: NUM\_IDLE\_SESSION

value: '5'

- name: SHARED\_FOLDER

value: /sharedata

- name: WAIT\_TO\_PLAYERS

value: '120'

- name: GAME\_SERVER\_GROUP\_NAME

value: ${GSGNAME}

- name: GAME\_MODE

value: "1"

- name: POD\_NAME

valueFrom:

fieldRef:

fieldPath: metadata.name

- name: NAMESPACE

valueFrom:

fieldRef:

fieldPath: metadata.namespace

image: '163538056407.dkr.ecr.us-west-2.amazonaws.com/stk:0.4'

imagePullPolicy: Always

lifecycle:

preStop:

exec:

command:

- /bin/sh

- '-c'

- /pre-stop.sh

name: stk

resources: {}

**nodeSelector:**

**role: game-servers-1**

**tolerations:**

**- effect: NoExecute**

**key: agones.dev/gameservers**

**operator: Equal**

**value: 'true'**

**- effect: NoExecute**

**key: gamelift.status/active**

**operator: Equal**

**value: 'true'**

volumes:

- emptyDir: {}

name: shared-data

EOF

kubectl apply -f agonesfleetconfig.yaml

## Test your FleetIQ-EKS-Agones installation

Please refer to <https://agones.dev/site/docs/> for details on Agones.

Some commands you might want to run:

* Check Agones Status:  
  kubectl describe --namespace agones-system pods
* Check Agones Pod Status:  
  kubectl get pods --namespace agones-system
* Get Fleets:  
  kubectl get fleets
* Get Gameservers:  
  kubectl get gameservers
* Get Gameserver details:  
  watch kubectl describe gameserver
* Scale up Fleet:  
  kubectl scale fleet stk-fleet --replicas=100  
  watch kubectl get fleets  
  kubectl get nodes
* Scale down Fleet:  
  kubectl scale fleet stk-fleet --replicas=1  
  watch kubectl get fleets  
  kubectl get nodes

# Part 5. Add a Second Nodegroup

BEFORE YOU BEGIN: The steps that follow assume you are running the commands from the same directory you used to generate your launch templates and Kubernetes configurations for the previous nodegroup.

1. Set launch template environment variable for use in creating the second launch template

NG\_STACK=$(aws cloudformation describe-stacks --region ${AWS\_REGION}| jq -r '.Stacks[] | .StackId' | grep ng-system)  
  
LAUNCH\_TEMPLATE\_ID=$(aws cloudformation describe-stack-resources --region ${AWS\_REGION} --stack-name $NG\_STACK \  
| jq -r '.StackResources | map(select(.LogicalResourceId == "NodeGroupLaunchTemplate")  
| .PhysicalResourceId)[0]')

1. Modify your NODE\_LABELS values (NOTE: the nodegroup-name and role values must be different from your existing nodegroup. They are set to ng-2 and game-servers-2, respectively here, but you can set them to a value you prefer. The role value you set must match the nodeSelector value in the GameLift daemonset Helm deployment below and in your Agones fleet affinity configuration at the end of the document).

awk -v var="$(grep -n NODE\_LABELS=alpha ./launchtemplate.yaml | cut -d : -f 1)" 'NR==var {$0=" NODE\_LABELS=agones.dev/agones-system=false,alpha.eksctl.io/cluster-name=agones,alpha.eksctl.io/nodegroup-name=ng-2,role=game-servers-2"} 1' modlaunchtemplate.yaml > modlaunchtemplate\_ng-2.yaml

base64 -w 0 modlaunchtemplate\_ng-2.yaml > b64modlaunchtemplate2

1. Create the launch template.

VPCID=$(aws ec2 describe-vpcs --region ${AWS\_REGION} --filter Name=tag:alpha.eksctl.io/cluster-name,Values=agones | jq -r '.Vpcs[0].VpcId')

SGID=$(aws ec2 create-security-group --region ${AWS\_REGION} --description "Agones nodegroup SG" --group-name eksctl-agones-nodegroup-ng-2-SG --vpc-id ${VPCID} | jq -r '.GroupId')

IAMINSTANCEPROFILE=$(aws ec2 describe-launch-template-versions --region ${AWS\_REGION} --launch-template-id $LAUNCH\_TEMPLATE\_ID | jq '.LaunchTemplateVersions[0].LaunchTemplateData.IamInstanceProfile.Arn')

NG0SG1=$(aws ec2 describe-launch-template-versions --region ${AWS\_REGION} --launch-template-id $LAUNCH\_TEMPLATE\_ID | jq '.LaunchTemplateVersions[0].LaunchTemplateData.NetworkInterfaces[0].Groups[0]')

NG0SG2=$(aws ec2 describe-launch-template-versions --region ${AWS\_REGION} --launch-template-id $LAUNCH\_TEMPLATE\_ID | jq '.LaunchTemplateVersions[0].LaunchTemplateData.NetworkInterfaces[0].Groups[1]')

NG0AMI=$(aws ec2 describe-launch-template-versions --region ${AWS\_REGION} --launch-template-id $LAUNCH\_TEMPLATE\_ID | jq '.LaunchTemplateVersions[0].LaunchTemplateData.ImageId')

B64USERDATA2=$(cat b64modlaunchtemplate2)

cat << EOF > ltinput2.json

{

"LaunchTemplateName": "eksctl-agones-nodegroup-ng-2",

"VersionDescription": "FleetIQ GameServerGroup LT",

"LaunchTemplateData": {

"EbsOptimized": true,

"IamInstanceProfile": {

"Arn": ${IAMINSTANCEPROFILE}

},

"BlockDeviceMappings": [{

"DeviceName": "/dev/xvda",

"Ebs": {

"Encrypted": false,

"DeleteOnTermination": true,

"VolumeSize": 80,

"VolumeType": "gp2"

}

}],

"NetworkInterfaces": [{

"DeviceIndex": 0,

"Groups": [

${NG0SG1},

${NG0SG2},

"${SGID}"

]

}],

"ImageId": $NG0AMI,

"Monitoring": {

"Enabled": true

},

"UserData": "${B64USERDATA2}",

"TagSpecifications": [{

"ResourceType": "instance",

"Tags": [{

"Key": "kubernetes.io/cluster/agones",

"Value": "owned"

},

{

"Key": "k8s.io/cluster-autoscaler/agones",

"Value": "owned"

},

{

"Key": "k8s.io/cluster-autoscaler/enabled",

"Value": "true"

},

{

"Key": "Name",

"Value": "FleetIQ"

}

]

}],

"MetadataOptions": {

"HttpTokens": "optional",

"HttpPutResponseHopLimit": 2

}

}

}

EOF

GSGLT2ID=$(aws ec2 create-launch-template --cli-input-json file://ltinput2.json --region ${AWS\_REGION} | jq '.LaunchTemplate.LaunchTemplateId')

## Set up a second FleetIQ game server group

Set up a FleetIQ game server group to track the game-hosting viability of EC2 instances types in your node group. Creating a new FleetIQ game server group automatically creates a corresponding new EC2 Auto Scaling Group.

In your Cloud9 workspace, use your open terminal window and enter the following commands:

### Create a new FleetIQ game server group

1. Retrieve the previously-created IAM FleetIQ service role [that allows GameLift to create and update EC2 Auto Scaling Groups.](https://docs.aws.amazon.com/gamelift/latest/developerguide/gsg-iam-permissions-roles.html#gsg-iam-permissions-roles-gamelift)

GSGROLEARN=$(aws iam get-role --role-name GameLiftServerGroupRole --query 'Role.Arn')

1. Specify your new Game Server Group Name

GSG2NAME="agones-game-server-group-02"

1. Create the game server group and EC2 Auto Scaling Group.  
   NOTE: The instance types you select must have identical vCPU and memory allocations.

PUBLICSUBNETIDS=$(aws ec2 describe-subnets --filters Name=vpc-id,Values=$VPCID --region ${AWS\_REGION} | jq '[.Subnets[] | {subnetid: .SubnetId, mapPublicIP: .MapPublicIpOnLaunch}]' | jq 'group\_by(.mapPublicIP)' | jq '[.[1][].subnetid]')

cat << EOF > gsgconfig2.json  
{  
"GameServerGroupName": "${GSG2NAME}",  
"RoleArn": ${GSGROLEARN},  
"MinSize": 1,  
"MaxSize": 10,  
"LaunchTemplate": {  
"LaunchTemplateId": ${GSGLT2ID},  
"Version": "1"  
},  
"InstanceDefinitions": [  
{  
"InstanceType": "c5.2xlarge",  
"WeightedCapacity": "2"  
},  
{  
"InstanceType": "c4.2xlarge",  
"WeightedCapacity": "1"  
}  
],  
"BalancingStrategy": "SPOT\_PREFERRED",  
"GameServerProtectionPolicy": "FULL\_PROTECTION",  
"VpcSubnets": ${PUBLICSUBNETIDS}  
}  
EOF

aws gamelift create-game-server-group --region ${AWS\_REGION} --cli-input-json <file://gsgconfig2.json>

1. Check the game server group status until it reports back as ACTIVE. The describe-game-server-group command returns the game server group properties, including status.

aws gamelift describe-game-server-group --game-server-group-name ${GSG2NAME} --region $AWS\_REGION | jq '.GameServerGroup.Status'

### Tag the Auto Scaling Group

Assign resource tags to the new Auto Scaling Group. The tags allow the cluster autoscaler to use the ASG to provision nodes for Agones.

aws autoscaling create-or-update-tags --tags ResourceId=gamelift-gameservergroup-${GSG2NAME},ResourceType=auto-scaling-group,Key=kubernetes.io/cluster/agones,Value=owned,PropagateAtLaunch=true ResourceId=gamelift-gameservergroup-${GSG2NAME},ResourceType=auto-scaling-group,Key=k8s.io/cluster-autoscaler/agones,Value=owned,PropagateAtLaunch=true ResourceId=gamelift-gameservergroup-${GSG2NAME},ResourceType=auto-scaling-group,Key=Name,Value=FleetIQ,PropagateAtLaunch=true ResourceId=gamelift-gameservergroup-${GSG2NAME},ResourceType=auto-scaling-group,Key=k8s.io/cluster-autoscaler/enabled,Value=true,PropagateAtLaunch=true --region ${AWS\_REGION}

## Deploy the GameLift Daemonset on New Game Server Group

**NOTE:** The gameliftDaemon.nodeSelector value below must match the Kubernetes node role value you set when you added the second nodegroup.

export HELM\_EXPERIMENTAL\_OCI=1

**NOTE:** Skip the ECR login and helm chart pull/export commands if you still have the directories from the original game server group deployment

aws ecr get-login-password --region us-west-2 | helm registry login --username AWS --password-stdin 820537372947.dkr.ecr.us-west-2.amazonaws.com

helm chart pull 820537372947.dkr.ecr.us-west-2.amazonaws.com/gamelift-daemon:0.1.1

helm chart pull 820537372947.dkr.ecr.us-west-2.amazonaws.com/gamelift-common-services:0.1.0

helm chart export 820537372947.dkr.ecr.us-west-2.amazonaws.com/gamelift-daemon:0.1.1

helm chart export 820537372947.dkr.ecr.us-west-2.amazonaws.com/gamelift-common-services:0.1.0

Update the update the fleetiq.conf value in gamelift-common-services/templates/deployment.yaml to include your additional game server group’s name.

For example:

fleetiq.conf: '{"GameServerGroups": [ "agones-game-server-group-01", "agones-game-server-group-02"]}'

helm upgrade \

--set aws.region=${AWS\_REGION} \

gamelift-common-services ./gamelift-common-services/

helm install \

--set aws.region=${AWS\_REGION} \

--set gameliftDaemon.serviceAccount=gamelift-daemonset \

--set gameliftDaemon.nodeSelector=game-servers-2 \

--set gameliftDaemon.gameServerGroupName="${GSG2NAME}" \

gamelift-daemonset2 ./gamelift-daemonset/

## Cluster Autoscaler Manifest Update

The Auto Scaling Group names in the Cluster Autoscaler manifest are generated based on the GSGNAME and GSG2NAME variables you set earlier. A prefix of “gamelift-gameservergroup-” is added by GameLift when it creates the Auto Scaling Group for each GameLift Game Server Group. This Auto Scaling Group name is the value that must be used in the Kubernetes Cluster Autoscaler manifest below (not the GameLift Game Server Group name).

Regarding Cluster Autoscaler priority values, the larger numerical value indicates to the Cluster Autoscaler which game server group you wish to prioritize adding compute capacity to. You can read more [here](https://github.com/kubernetes/autoscaler/blob/master/cluster-autoscaler/expander/priority/readme.md).

cat << EOF > camanifest-updated.yaml  
---  
apiVersion: v1  
kind: ServiceAccount  
metadata:  
 labels:  
 k8s-addon: cluster-autoscaler.addons.k8s.io  
 k8s-app: cluster-autoscaler  
 name: cluster-autoscaler  
 namespace: kube-system  
---  
apiVersion: rbac.authorization.k8s.io/v1  
kind: ClusterRole  
metadata:  
 name: cluster-autoscaler  
 labels:  
 k8s-addon: cluster-autoscaler.addons.k8s.io  
 k8s-app: cluster-autoscaler  
rules:  
 - apiGroups: [""]  
 resources: ["events", "endpoints"]  
 verbs: ["create", "patch"]  
 - apiGroups: [""]  
 resources: ["pods/eviction"]  
 verbs: ["create"]  
 - apiGroups: [""]  
 resources: ["pods/status"]  
 verbs: ["update"]  
 - apiGroups: [""]  
 resources: ["endpoints"]  
 resourceNames: ["cluster-autoscaler"]  
 verbs: ["get", "update"]  
 - apiGroups: [""]  
 resources: ["nodes"]  
 verbs: ["watch", "list", "get", "update"]  
 - apiGroups: [""]  
 resources:  
 - "pods"  
 - "services"  
 - "replicationcontrollers"  
 - "persistentvolumeclaims"  
 - "persistentvolumes"  
 verbs: ["watch", "list", "get"]  
 - apiGroups: ["extensions"]  
 resources: ["replicasets", "daemonsets"]  
 verbs: ["watch", "list", "get"]  
 - apiGroups: ["policy"]  
 resources: ["poddisruptionbudgets"]  
 verbs: ["watch", "list"]  
 - apiGroups: ["apps"]  
 resources: ["statefulsets", "replicasets", "daemonsets"]  
 verbs: ["watch", "list", "get"]  
 - apiGroups: ["storage.k8s.io"]  
 resources: ["storageclasses", "csinodes"]  
 verbs: ["watch", "list", "get"]  
 - apiGroups: ["batch", "extensions"]  
 resources: ["jobs"]  
 verbs: ["get", "list", "watch", "patch"]  
 - apiGroups: ["coordination.k8s.io"]  
 resources: ["leases"]  
 verbs: ["create"]  
 - apiGroups: ["coordination.k8s.io"]  
 resourceNames: ["cluster-autoscaler"]  
 resources: ["leases"]  
 verbs: ["get", "update"]  
---  
apiVersion: rbac.authorization.k8s.io/v1  
kind: Role  
metadata:  
 name: cluster-autoscaler  
 namespace: kube-system  
 labels:  
 k8s-addon: cluster-autoscaler.addons.k8s.io  
 k8s-app: cluster-autoscaler  
rules:  
 - apiGroups: [""]  
 resources: ["configmaps"]  
 verbs: ["create","list","watch"]  
 - apiGroups: [""]  
 resources: ["configmaps"]  
 resourceNames: ["cluster-autoscaler-status", "cluster-autoscaler-priority-expander"]  
 verbs: ["delete", "get", "update", "watch"]  
---  
apiVersion: rbac.authorization.k8s.io/v1  
kind: ClusterRoleBinding  
metadata:  
 name: cluster-autoscaler  
 labels:  
 k8s-addon: cluster-autoscaler.addons.k8s.io  
 k8s-app: cluster-autoscaler  
roleRef:  
 apiGroup: rbac.authorization.k8s.io  
 kind: ClusterRole  
 name: cluster-autoscaler  
subjects:  
 - kind: ServiceAccount  
 name: cluster-autoscaler  
 namespace: kube-system  
---  
apiVersion: rbac.authorization.k8s.io/v1  
kind: RoleBinding  
metadata:  
 name: cluster-autoscaler  
 namespace: kube-system  
 labels:  
 k8s-addon: cluster-autoscaler.addons.k8s.io  
 k8s-app: cluster-autoscaler  
roleRef:  
 apiGroup: rbac.authorization.k8s.io  
 kind: Role  
 name: cluster-autoscaler  
subjects:  
 - kind: ServiceAccount  
 name: cluster-autoscaler  
 namespace: kube-system

---

apiVersion: v1

kind: ConfigMap

metadata:

name: cluster-autoscaler-priority-expander

namespace: kube-system

data:

priorities: |-

10:

- .\*-non-existing-entry.\*

20:

- gamelift-gameservergroup-${GSGNAME}

60:

- gamelift-gameservergroup-${GSG2NAME}  
---  
apiVersion: apps/v1  
kind: Deployment  
metadata:  
 name: cluster-autoscaler  
 namespace: kube-system  
 labels:  
 app: cluster-autoscaler  
 annotations:  
 cluster-autoscaler.kubernetes.io/safe-to-evict: 'false'  
spec:  
 replicas: 1  
 selector:  
 matchLabels:  
 app: cluster-autoscaler  
 template:  
 metadata:  
 labels:  
 app: cluster-autoscaler  
 annotations:  
 prometheus.io/scrape: 'true'  
 prometheus.io/port: '8085'  
 spec:  
 serviceAccountName: cluster-autoscaler  
 containers:  
 - image: k8s.gcr.io/autoscaling/cluster-autoscaler:v1.18.3  
 name: cluster-autoscaler  
 resources:  
 limits:  
 cpu: 100m  
 memory: 300Mi  
 requests:  
 cpu: 100m  
 memory: 300Mi  
 command:  
 - ./cluster-autoscaler

- --v=4

- --stderrthreshold=info

- --cloud-provider=aws

- --skip-nodes-with-local-storage=false

- --expander=priority

- --nodes=0:10:gamelift-gameservergroup-${GSGNAME}

- --nodes=0:10:gamelift-gameservergroup-${GSG2NAME}

- --balance-similar-node-groups

- --skip-nodes-with-system-pods=false  
 env:  
 - name: AWS\_REGION  
 value: ${AWS\_REGION}  
 volumeMounts:  
 - name: ssl-certs  
 mountPath: /etc/ssl/certs/ca-certificates.crt  
 readOnly: true  
 imagePullPolicy: "Always"  
 volumes:  
 - name: ssl-certs  
 hostPath:  
 path: "/etc/ssl/certs/ca-bundle.crt"  
EOF

kubectl apply -f camanifest-updated.yaml

This updated fleet YAML must include the “role” labels that you created earlier in the Autoscaling Group launch templates. If you used values that are different than the ones used in this document (i.e., game-servers-1 and game-servers-2), please update the affinity section below accordingly.

cat << EOF > agonesfleetconfig-updated.yaml

kind: Fleet

apiVersion: agones.dev/v1

metadata:

annotations:

agones.dev/sdk-version: 1.8.0

name: stk-fleet

namespace: default

spec:

replicas: 5

scheduling: Packed

strategy:

rollingUpdate:

maxSurge: 25%

maxUnavailable: 25%

type: RollingUpdate

template:

metadata:

labels:

app: stk

spec:

health:

failureThreshold: 3

initialDelaySeconds: 30

periodSeconds: 10

ports:

- containerPort: 8080

name: default

portPolicy: Dynamic

sdkServer: {}

template:

metadata: {}

spec:

affinity:

nodeAffinity:

requiredDuringSchedulingIgnoredDuringExecution:

nodeSelectorTerms:

- matchExpressions:

- key: role

operator: In

values:

- game-servers-1

- game-servers-2

containers:

- command:

- /start.sh

env:

- name: WAIT\_TO\_PLAYERS

value: '120'

- name: FREQ\_CHECK\_SESSION

value: '10'

- name: NUM\_IDLE\_SESSION

value: '5'

- name: SHARED\_FOLDER

value: /sharedata

- name: WAIT\_TO\_PLAYERS

value: '120'

- name: GAME\_SERVER\_GROUP\_NAME

value:

- name: GAME\_MODE

value: "1"

- name: POD\_NAME

valueFrom:

fieldRef:

fieldPath: metadata.name

- name: NAMESPACE

valueFrom:

fieldRef:

fieldPath: metadata.namespace

image: '163538056407.dkr.ecr.us-west-2.amazonaws.com/stk:0.4'

imagePullPolicy: Always

lifecycle:

preStop:

exec:

command:

- /bin/sh

- '-c'

- /pre-stop.sh

name: stk

resources: {}

tolerations:

- effect: NoExecute

key: agones.dev/gameservers

operator: Equal

value: 'true'

- effect: NoExecute

key: gamelift.status/active

operator: Equal

value: 'true'

volumes:

- emptyDir: {}

name: shared-data

EOF

Apply the updated fleet config

kubectl apply -f agonesfleetconfig-updated.yaml

## Test your FleetIQ-EKS-Agones installation

Please refer to <https://agones.dev/site/docs/> for details on Agones.

Some commands you might want to run:

* Check Agones Status:  
  kubectl describe --namespace agones-system pods
* Check Agones Pod Status:  
  kubectl get pods --namespace agones-system
* Get Fleets:  
  kubectl get fleets
* Get Gameservers:  
  kubectl get gameservers
* Get Gameserver details:  
  watch kubectl describe gameserver
* Scale up Fleet:  
  kubectl scale fleet stk-fleet --replicas=100  
  watch kubectl get fleets  
  kubectl get nodes
* Scale down Fleet:  
  kubectl scale fleet stk-fleet --replicas=1  
  watch kubectl get fleets  
  kubectl get nodes

# Document Revisions

|  |  |
| --- | --- |
| Date | Description |
| December 2020 | Added support for Kubernetes 1.17 and instructions for additional Kubernetes nodegroups (Part 5) |
| February 2021 | Added support for updated helm charts to deploy the gamelift daemonset, pubsub v2, and Redis components |
| April 2021 | Added support for Agones 1.13.0 and Kubernetes 1.18, and clarified nodegroup and game server group naming |