Controlling your Load Balancer in Kubernetes

Document Level Classification

200

- Introduction
- Fixed Response Rules
- Redirecting Port 80 to 443
- ALB Access Logging
- Making Changes Through the AWS Console
- Creating a Custom Certificate
- Attaching an existing Custom Certificate
- Adding a Web Application Firewall (WAF)
- Automatically Whitelist Internal Traffic
- Troubleshooting Load Balancer Errors
- Using Target Type IP

Introduction

Services running on Kubernetes typically receive traffic via an Application Load Balancer (ALB) or Network Load Balancer (NLB) manged by Kubernetes. Load balancers are controlled by annotations in your helm chart. The complete documentation for this behavior is described in the AWS Load Balancer Controller Guide.

In addition to the kube annotations provided by the AWS LB controller, Substrate also includes a security group attached and managed by the IxM command-line tool.

Continue reading for some specific examples.

Fixed Response Rules

You can configure your load balancer to respond to certain requests with a pre-configured response. This can be used to temporarily block parts of your application, or for more permanent scenarios like redirecting to another service or URL.

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  namespace: default
  name: ingress
  annotations:
    kubernetes.io/ingress.class: alb
    alb.ingress.kubernetes.io/scheme: internet-facing
    alb.ingress.kubernetes.io/actions.response-503: >
      {"Type":"fixed-response","FixedResponseConfig":{"ContentType":"te
spec:
  rules:
    - http:
        paths:
          - path: /503
            backend:
              serviceName: response-503
              servicePort: use-annotation
```

Thanks to Matt Armstrong for sharing this example!

Redirecting Port 80 to 443

How can I redirect `http` to `https`?

You can do this easily with alb-ingress-controller via the `ssl-redirect` annotation and a port 80 listener:

```
alb.ingress.kubernetes.io/listen-ports: '[{"HTTP": 80}, {"HTTPS":443}]'
alb.ingress.kubernetes.io/ssl-redirect: '443'
```

Prior versions of alb-ingress-controller required an `action` annotation along with a wildcard path pointing traffic through the redirect – this is no longer necessary if you update to ↑.

ALB Access Logging

Every substrate account has a dedicated logging bucket named as `{{ account-id }}-logs` where account-id is the numerical AWS Account ID. This bucket is connected to SumoLogic collector with the same name.

In order to enable ALB Access Logging, an annotation must be added to ingress object:

```
alb.ingress.kubernetes.io/load-balancer-attributes: access_logs.s3.enab
```

Further documentation here: https://kubernetes-sigs.github.io/aws-load-balancer-controller/latest/guide/ingress/annotations/#custom-attributes

Thanks to Former user (Deleted) for sharing this example!

Making Changes Through the AWS Console

In emergency situations, you may make these changes through the AWS console. In Substrate v2020 service owners have permission to modify the ALB attributes via the console using their UAM team role.

However, Kubernetes continuously synchronizes changes between Kubernetes and AWS, so your changes will be reverted. If you need to make manual changes, please request that the infra-platform team scale down the ALB controller deployment for your cluster.

```
$ kubectl -n epic-system scale deployment aws-alb-ingress-controller --
```

Creating a Custom Certificate

Automation exists to automatically provision certificates within ACM, this includes automatic provisioning of validation records in Route53 for any "internal.epicgames.net" subdomains.

```
loadbalancers:
  ingest:
    hosts:
      - host: service-ingest.abcd-dev.internal.epicgames.net
    port: 8080
    createSSL: true
    annotations:
      kubernetes.io/ingress.class: alb
      alb.ingress.kubernetes.io/scheme: internet-facing
      alb.ingress.kubernetes.io/success-codes: 200,404
      alb.ingress.kubernetes.io/security-groups: <sg name>
```

Attaching an existing Custom Certificate

You can create a custom CNAME for your service and attach a custom ACM certificate using the certificate-arn annotation. See https://kubernetes-sigs.github.io/aws-load-balancer-controller/v1.1/guide/ingress/annotation/#certificate-arn for details.

```
loadbalancers:
  ingest:
    hosts:
      - host: service-ingest.befa.live.use1a.on.epicgames.com
      - host: service-ingest.ol.epicgames.net
    port: 8080
    annotations:
      kubernetes.io/ingress.class: alb
      alb.ingress.kubernetes.io/scheme: internet-facing
      alb.ingress.kubernetes.io/success-codes: 200,404
      alb.ingress.kubernetes.io/certificate-arn: <cert arn>
      alb.ingress.kubernetes.io/security-groups: <sg name>
```

See the full example in the example app.

Adding a Web Application Firewall (WAF)

You can attach a WAF to your load balancer by referencing its ARN. The WAF configuration itself may be managed with Terraform. The documentation notes that only regional WAFs may be used.

File: `chart.yaml`

```
loadbalancers:
  ingest:
    hosts:
      - host: service-ingest.befa.live.use1a.on.epicgames.com
      - host: service-ingest.ol.epicgames.net
    port: 8080
    annotations:
      alb.ingress.kubernetes.io/wafv2-acl-arn: arn:aws:wafv2:us-west-2:
```

Here is a basic Terraform example for the WAF:

File: `waf.tf`

```
resource "aws_wafv2_ip_set" "my-app-whitelist" {
  name               = "my-app-whitelist"
  description        = "IP whitelist for my app"
  scope              = "REGIONAL"
  ip_address_version = "IPV4"
  addresses          = ["1.2.3.4/32", "2.3.4.5/32"]
}

resource "aws_wafv2_web_acl" "my-app-waf" {
  name = "my-app-waf"

  scope = "REGIONAL"

  default_action {
    block {}
  }

  rule {
    name     = "my-app-whitelist"
    priority = 1

    action {
      allow {}
    }

    statement {
      ip_set_reference_statement {
        arn = aws_wafv2_ip_set.my-app-whitelist.arn
      }
    }

    visibility_config {
      metric_name                = "my-app-waf-whitelist-requests"
      cloudwatch_metrics_enabled = true
      sampled_requests_enabled   = true
    }
  }

  visibility_config {
    metric_name                = "my-app-waf-all-requests"
    cloudwatch_metrics_enabled = true
    sampled_requests_enabled   = true
  }
}
```

See https://kubernetes-sigs.github.io/aws-load-balancer-controller/v2.2/guide/ingress/annotations/ for complete details.

Automatically Whitelist Internal Traffic

We use IxM prefix lists to whitelist traffic from other Epic AWS accounts, VPN, offices, and more. If you want your service to be available to all Epic networks, you can use the substrate/ixm-waf-sync lambda to manage your WAF IP set rules. See the previous WAF example for setting up the WAF and use `ixm-waf-sync` to keep the IP set up to date as new prefix lists are added or updated in IxM.

waf.tf

```
resource "aws_wafv2_ip_set" "epic-ips" {
  name               = "epic-ips"
  description        = "Known Epic IP ranges"
  scope              = "REGIONAL"
  ip_address_version = "IPV4"
  addresses          = []

  lifecycle {
    // The list of CIDRs is managed by the waf-cidr-sync lambda
    ignore_changes = [
      addresses,
      description,
    ]
  }
}
```

You can see an example of a project that uses this lambda here: https://github.ol.epicgames.net/substrate/artifacts/blob/main/terraform/waf.tf

Troubleshooting Load Balancer Errors

Start by finding the CNAME for your load balancer in your AWS account.

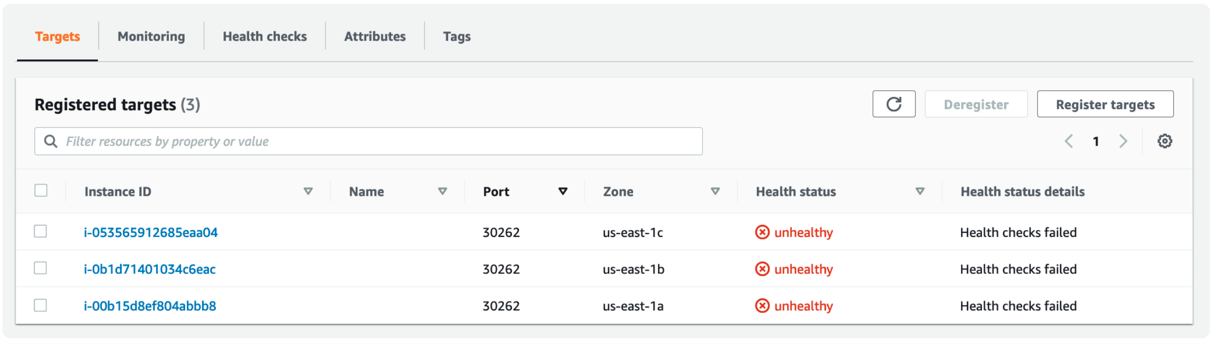
1. Start under Hosted Zones in Route53
2. Select the domain for your region, such as `*.use1a.on.epicgames.com`
3. Search for the name of your service
4. Note the value of the A record, pointing to something like `fbb8c27b-teamonlineinfrapl-a118-477313269.us-east-1.elb.amazonaws.com`. Make a note.

Now we'll need to look up the load balancer.

1. Start under Load Balancers in EC2 (adjust your region if necessary)

2. Search for the LB name we noted earlier, such as `fbb8c27b-teamonlineinfrapl-a118-477313269`
3. Select the load balancer and inspect the **Listeners** tab
4. Click **View/edit rules** under the 443 listener
5. Look for a rule that says "Forward to" and click on the target ID. This is the "target group"

Once you have identified the target group you can see details about individual service backends running in kube. For example, these backends are failing ALB health checks so the DNS for this service returns an error. In this case, you would need to resolve the health check failure in order to proceed (the **Health checks** tab has more details).



Registered targets (3)

Instance ID	Name	Port	Zone	Health status	Health status details
i-053565912685eaa04		30262	us-east-1c	unhealthy	Health checks failed
i-0b1d71401034c6eac		30262	us-east-1b	unhealthy	Health checks failed
i-00b15d8ef804abbb8		30262	us-east-1a	unhealthy	Health checks failed

If the Health status is **unhealthy** **check** the security group for the ALB ensuring that the egress has a rule that allows all traffic outbound.

Using Target Type IP

See Substrate Deploys without Downtime#ALBTargetType

Page Information:
Page ID: 81068330
Space: Cloud Developer Platform
Downloaded: 2025-07-12 04:08:15