AWS Compute Blog

Sending Amazon EventBridge events to private endpoints in a VPC

by James Beswick | on 23 JUN 2022 | in Amazon EventBridge, Serverless | Permalink | Share

This post is written by Emily Shea, Senior GTM Specialist, Event-Driven Architectures.

Building with events can help you accelerate feature velocity and build scalable, fault tolerant applications. You can achieve loose coupling in your application using asynchronous communication via events. Loose coupling allows each development team to build and deploy independently and each component to scale and fail without impacting the others. This approach is referred to as event-driven architecture.

Amazon EventBridge helps you build event-driven architectures. You can publish events to the EventBridge event bus and EventBridge routes those events to targets. You can write rules to filter events and only send them to the interested targets. For example, an order fulfillment service may only be interested in events of type 'new order created.'

EventBridge is serverless, so there is no infrastructure to manage and the service scales automatically. EventBridge has native integrations with over 100 AWS services and over 40 SaaS providers.

Amazon EventBridge has a native integration with AWS Lambda, and many AWS customers use events to trigger Lambda functions to process events. You may also want to send events to workloads running on Amazon EC2 or containerized workloads deployed with Amazon ECS or Amazon EKS. These services are deployed into an Amazon Virtual Private Cloud, or VPC.

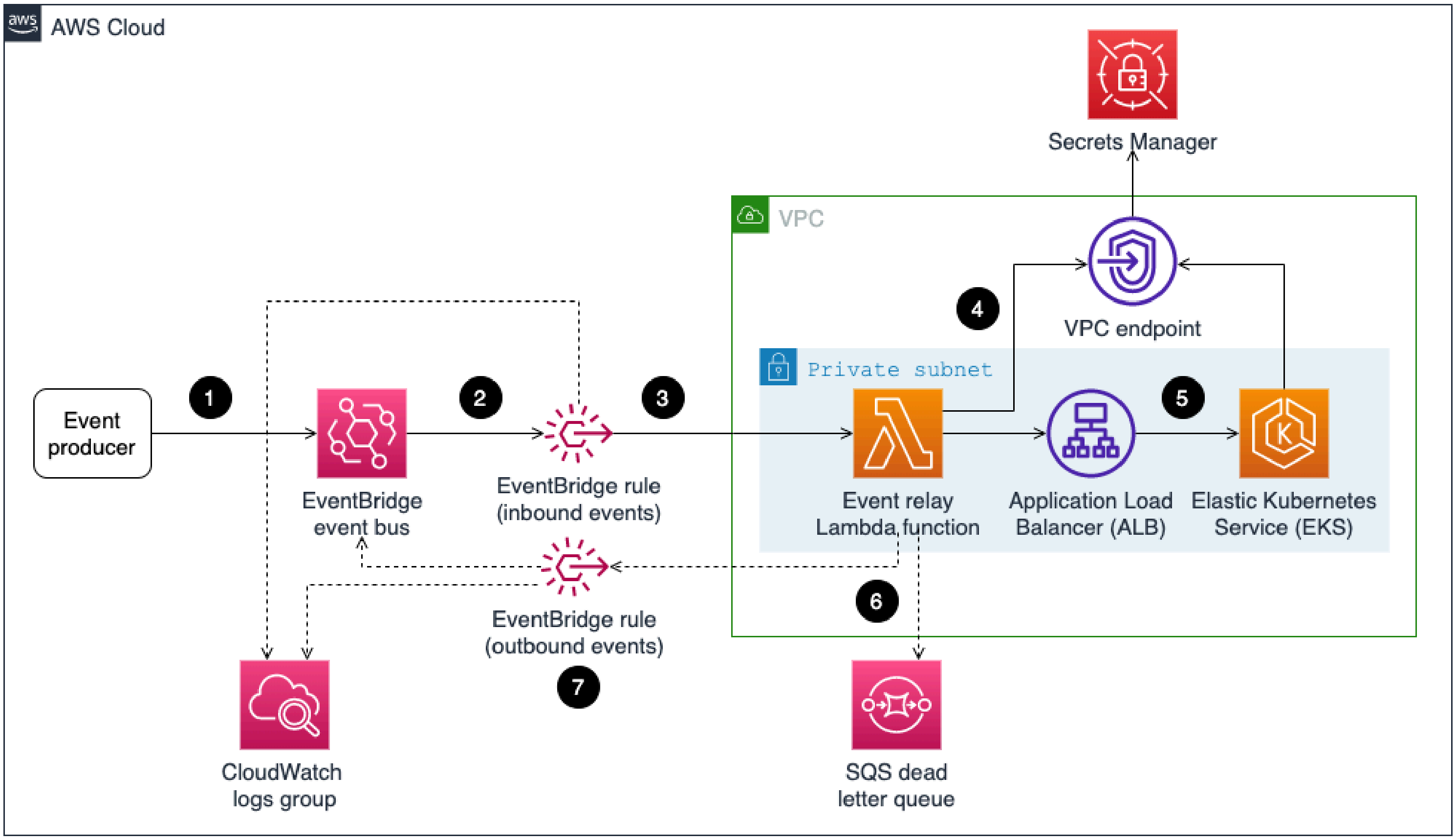
For some use cases, you may be able to expose public endpoints for your VPC. You can use EventBridge API destinations to send events to any public HTTP endpoint. API destinations include features like OAuth support and rate limiting to control the number of events you are sending per second.

However, some customers are not able to expose public endpoints for security or compliance purposes. This tutorial shows you how to send EventBridge events to a private endpoint in a VPC using a Lambda function to relay events. This solution deploys the Lambda function connected to the VPC and uses IAM permissions to enable EventBridge to invoke the Lambda function. Learn more about Lambda VPC connectivity here.

In this blog post, you learn how to send EventBridge events to a private endpoint in a VPC. You set up an example application with an EventBridge event bus, a Lambda function to relay events, a Flask application running in an EKS cluster to receive events behind an Application Load Balancer (ALB), and a secret stored in Secrets Manager for authenticating requests. This application uses EKS and Secrets Manager to demonstrate sending and authenticating requests to a containerized workload, but the same pattern applies for other container orchestration services like ECS and your preferred secret management solution.

Continue reading for the full example application and walkthrough. If you have an existing application in a VPC, you can deploy just the event relay portion and input your VPC details as parameters.

Solution overview


AWS Cloud
Secrets Manager
VPC
VPC endpoint
4
Private subnet
Event producer
1
EventBridge event bus
2
EventBridge rule (inbound events)
3
Event relay Lambda function
Application Load Balancer (ALB)
5
Elastic Kubernetes Service (EKS)
EventBridge rule (outbound events)
6
7
CloudWatch logs group
SQS dead letter queue

1. An event is sent to the EventBridge bus.
2. If the event matches a certain pattern (ex, if 'detail-type' is 'inbound-event-sent'), an EventBridge rule uses EventBridge's input transformer to format the event as an HTTP call.
3. The EventBridge rule pushes the event to a Lambda function connected to the VPC and a CloudWatch Logs group for debugging.
4. The Lambda function calls Secrets Manager and retrieves a secret key. It appends the secret key to the event headers and then makes an HTTP call to the ALB URL endpoint.
5. ALB routes this HTTP call to a node group in the EKS cluster. The Flask application running on the EKS cluster calls Secret Manager, confirms that the secret key is valid, and then processes the event.
6. The Lambda function receives a response from ALB.
    1. If the Flask application fails to process the event for any reason, the Lambda function raises an error. The function's failure destination is configured to send the event and the error message to an SQS dead letter queue.
    2. If the Flask application successfully processes the event and the 'return-response-event' flag in the event was set to 'true', then the Lambda function publishes a new 'outbound-event-sent' event to the same EventBridge bus.
7. Another EventBridge rule matches detail-type 'outbound-event-sent' events and routes these to the CloudWatch Logs group for debugging.

Prerequisites

To run the application, you must install the AWS CLI, Docker CLI, eksctl, kubectl, and AWS SAM CLI.

To clone the repository, run:

Bash
```
git clone https://github.com/aws-samples/eventbridge-events-to-vpc.git
```

Creating the EKS cluster

1. In the example-vpc-application directory, use eksctl to create the EKS cluster using the config file.

   Bash
   ```
   cd example-vpc-application
   eksctl create cluster --config-file eksctl_config.yaml
   ```

   This takes a few minutes. This step creates an EKS cluster with one node group in us-east-1. The EKS cluster has a service account with IAM permissions to access the Secrets Manager secret you create later.

2. Use your AWS account's default Amazon Elastic Container Registry (ECR) private registry to store the container image. First, follow these instructions to authenticate Docker to ECR. Next, run this command to create a new ECR repository. The create-repository command returns a repository URI (for example, 123456789.dkr.ecr.us-east-1.amazonaws.com/events-flask-app).

   Bash
   ```
   aws ecr create-repository --repository-name events-flask-app
   ```

   Use the repository URI in the following commands to build, tag, and push the container image to ECR.

   Bash
   ```
   docker build --tag events-flask-app .
   docker tag events-flask-app:latest {repository-uri}:1
   docker push {repository-uri}:1
   ```

3. In the Kuberenetes deployment manifest file (`/example-vpc-application/manifests/deployment.yaml`), fill in your repository URI and container image version (for example, `123456789.dkr.ecr.us-east-1.amazonaws.com/events-flask-app:1`)

Deploy the Flask application and Application Load Balancer

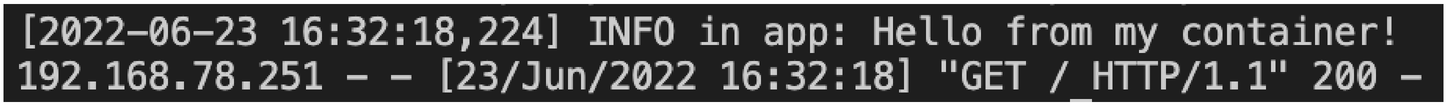
1. Within the example-vpc-application directory, use kubectl to apply the Kubernetes manifest files. This step deploys the ALB, which takes time to create and you may receive an error message during the deployment ('no endpoints available for service "aws-load-balancer-webhook-service"'). Rerun the same command until the ALB is deployed and you no longer receive the error message.

Bash
```
kubectl apply --kustomize manifests/
```

2. Once the deployment is completed, verify that the Flask application is running by retrieving the Kubernetes pod logs. The first command retrieves a pod name to fill in for the second command.

Bash
```
kubectl get pod --namespace vpc-example-app
kubectl logs --namespace vpc-example-app {pod-name} --follow
```

You should see the Flask application outputting 'Hello from my container!' in response to GET request health checks.

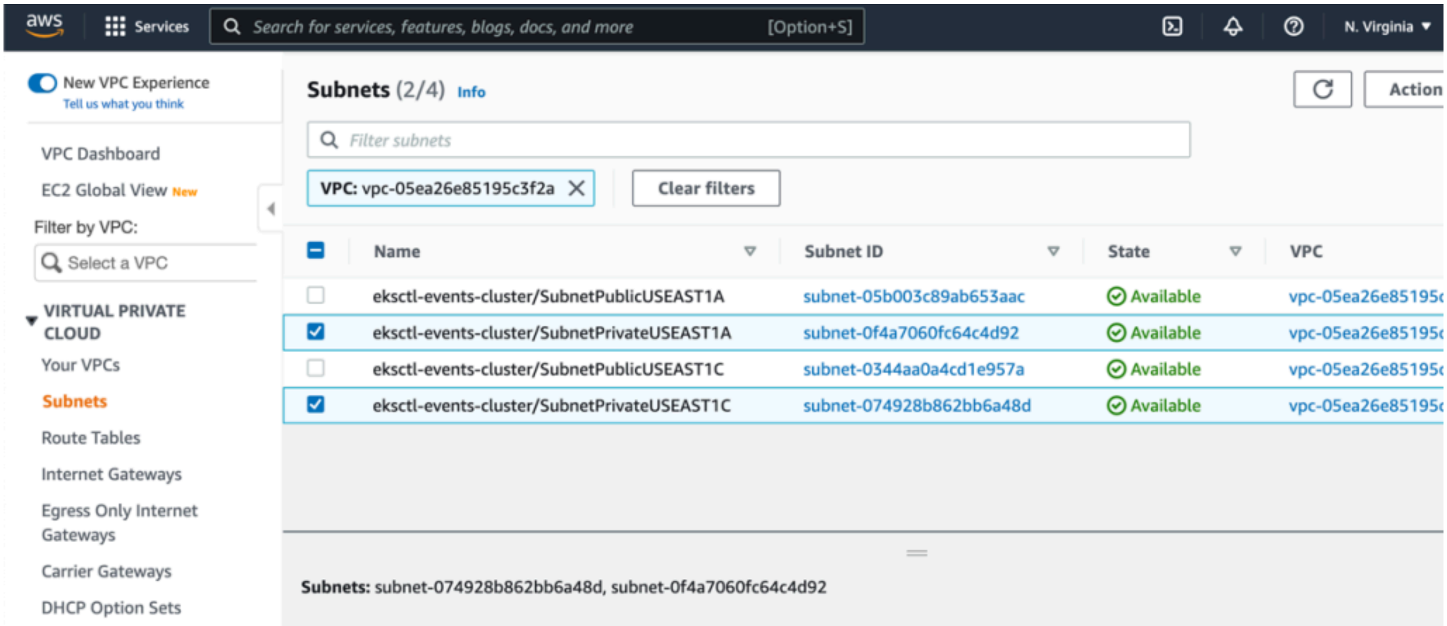


```
[2022-06-23 16:32:18,224] INFO in app: Hello from my container!
192.168.78.251 - - [23/Jun/2022 16:32:18] "GET / HTTP/1.1" 200 -
```

Get VPC and ALB details

Next, you retrieve the security group ID, private subnet IDs, and ALB DNS Name to deploy the Lambda function connected to the same VPC and private subnet and send events to the ALB.

1. In the AWS Management Console, go to the VPC dashboard and find Subnets. Copy the subnet IDs for the two private subnets (for example, subnet name 'eksctl-events-cluster/SubnetPrivateUSEAST1A').



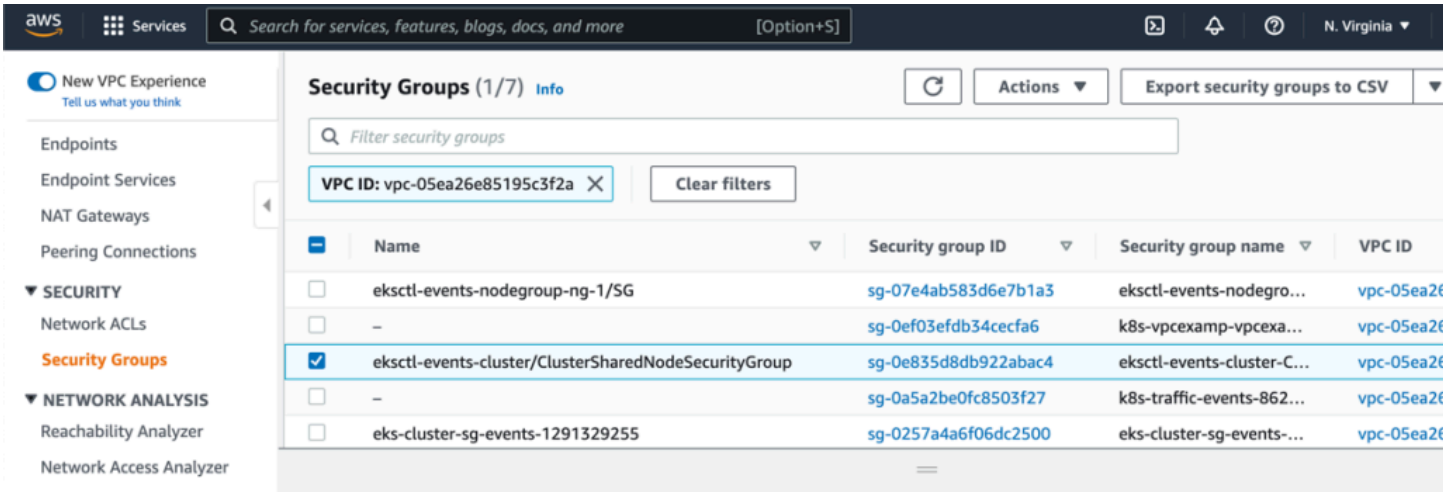
Subnets (2/4) Info

VPC: vpc-05ea26e85195c3f2a

	Name	Subnet ID	State	VPC
□	eksctl-events-cluster/SubnetPublicUSEAST1A	subnet-05b003c89ab653aac	✓ Available	vpc-05ea26e85195c
☑	eksctl-events-cluster/SubnetPrivateUSEAST1A	subnet-0f4a7060fc64c4d92	✓ Available	vpc-05ea26e85195c
□	eksctl-events-cluster/SubnetPublicUSEAST1C	subnet-0344aa0a4cd1e957a	✓ Available	vpc-05ea26e85195c
☑	eksctl-events-cluster/SubnetPrivateUSEAST1C	subnet-074928b862bb6a48d	✓ Available	vpc-05ea26e85195c

Subnets: subnet-074928b862bb6a48d, subnet-0f4a7060fc64c4d92

2. In the VPC dashboard, under Security, find the Security Groups tab. Copy the security group ID for ‘eksctl-events-cluster/ClusterSharedNodeSecurityGroup’.

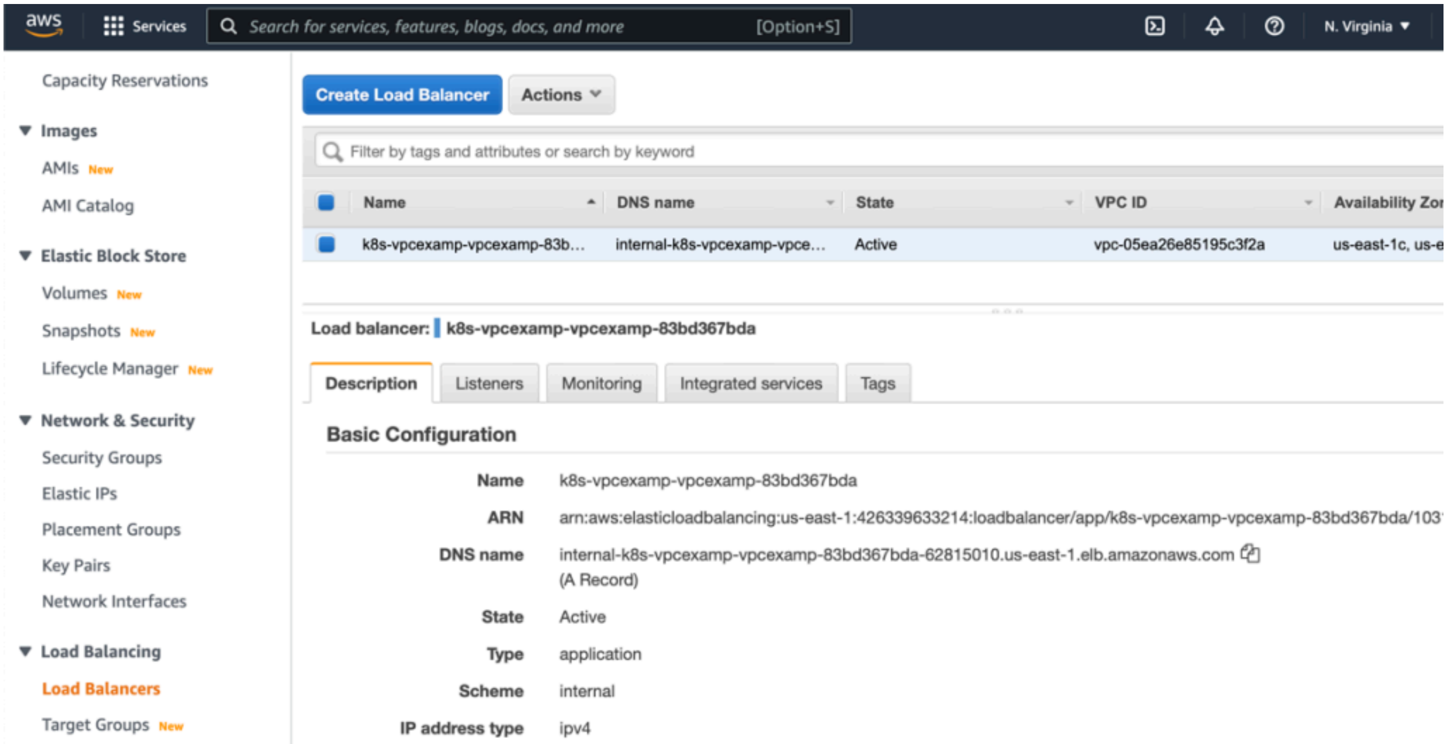


Security Groups (1/7) Info

VPC ID: vpc-05ea26e85195c3f2a

	Name	Security group ID	Security group name	VPC ID
□	eksctl-events-nodegroup-ng-1/SG	sg-07e4ab583d6e7b1a3	eksctl-events-nodegro…	vpc-05ea26
□	–	sg-0ef03efdb34cecfa6	k8s-vpcexamp-vpcexa…	vpc-05ea26
☑	eksctl-events-cluster/ClusterSharedNodeSecurityGroup	sg-0e835d8db922abac4	eksctl-events-cluster-C…	vpc-05ea26
□	–	sg-0a5a2be0fc8503f27	k8s-traffic-events-862…	vpc-05ea26
□	eks-cluster-sg-events-1291329255	sg-0257a4a6f06dc2500	eks-cluster-sg-events-…	vpc-05ea26

3. Go to the EC2 dashboard. Under Load Balancing, find the Load Balancer tab. There is a load balancer associated with your VPC ID. Copy the DNS name for the load balancer, adding ‘http://’ as a prefix (for example, http://internal-k8s-vpcexamp-vpcexamp-c005e07d1a-1074647274.us-east-1.elb.amazonaws.com).



	Name	DNS name	State	VPC ID	Availability Zo
■	k8s-vpcexamp-vpcexamp-83b…	internal-k8s-vpcexamp-vpce…	Active	vpc-05ea26e85195c3f2a	us-east-1c, us-e

Load balancer: k8s-vpcexamp-vpcexamp-83bd367bda

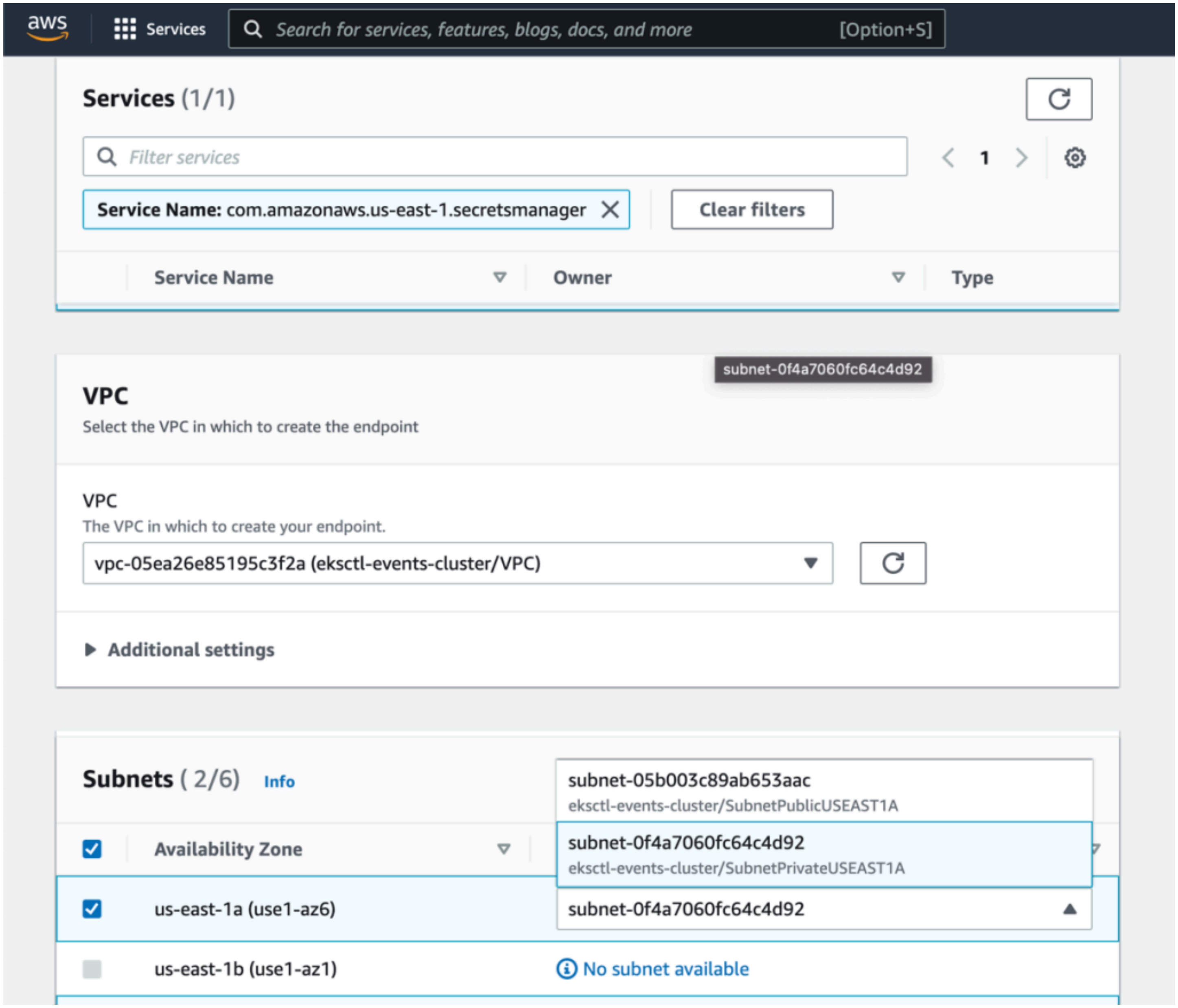
Basic Configuration

Name	k8s-vpcexamp-vpcexamp-83bd367bda
ARN	arn:aws:elasticloadbalancing:us-east-1:426339633214:loadbalancer/app/k8s-vpcexamp-vpcexamp-83bd367bda/103
DNS name	internal-k8s-vpcexamp-vpcexamp-83bd367bda-62815010.us-east-1.elb.amazonaws.com (A Record)
State	Active
Type	application
Scheme	internal
IP address type	ipv4

Create the Secrets Manager VPC endpoint

You need a VPC endpoint for your application to call Secrets Manager.

1. In the VPC dashboard, find the Endpoints tab and choose Create Endpoint. Select Secrets Manager as the service, and then select the VPC, private subnets, and security group that you copied in the previous step. Choose **Create**.



Services (1/1)

Service Name: com.amazonaws.us-east-1.secretsmanager

Service Name	Owner	Type

VPC

Select the VPC in which to create the endpoint

VPC

The VPC in which to create your endpoint.

vpc-05ea26e85195c3f2a (eksctl-events-cluster/VPC)

subnet-0f4a7060fc64c4d92

Subnets (2/6)

subnet-05b003c89ab653aac
eksctl-events-cluster/SubnetPublicUSEAST1A

subnet-0f4a7060fc64c4d92
eksctl-events-cluster/SubnetPrivateUSEAST1A

☑	Availability Zone	
☑	us-east-1a (use1-az6)	subnet-0f4a7060fc64c4d92 ▲
□	us-east-1b (use1-az1)	ⓘ No subnet available

Deploy the event relay application

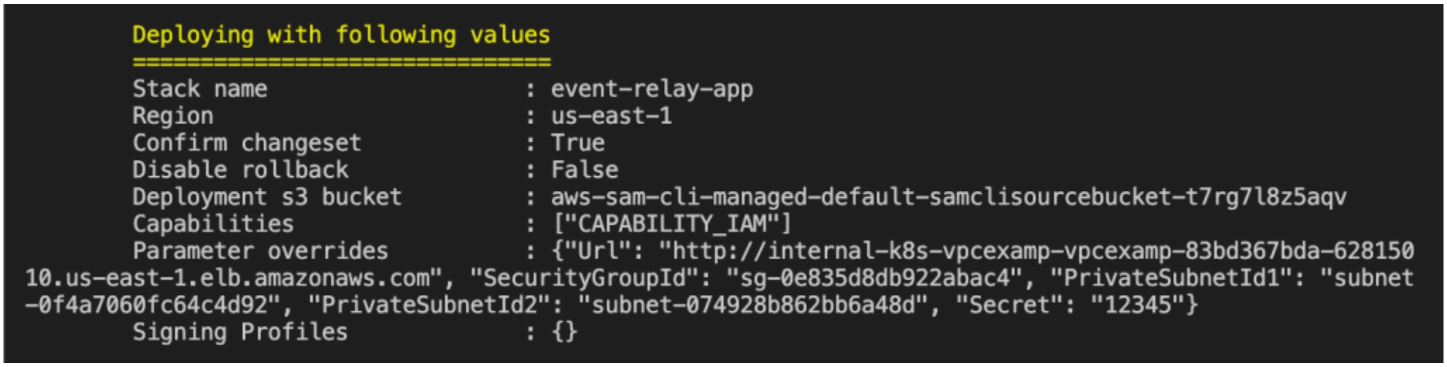
Deploy the event relay application using the AWS Serverless Application Model (AWS SAM) CLI:

1. Open a new terminal window and navigate to the event-relay directory. Run the following AWS SAM CLI commands to build the application and step through a guided deployment.

Bash
```
cd event-relay
sam build
sam deploy --guided
```

The guided deployment process prompts for input parameters. Enter ‘event-relay-app’ as the stack name and accept the default Region. For other parameters, submit the ALB and VPC details you copied: Url (ALB DNS name), security group ID, and private subnet IDs. For the Secret parameter, pass any value.The AWS SAM template saves this value as a Secrets Manager secret to authenticate calls to the container application. This is an example of how to pass secrets in the event relay HTTP call. Replace this with your own authentication method in production environments.

2. Accept the defaults for the remaining options. For ‘Deploy this changeset?’, select ‘y’. Here is an example of the deployment parameters.



```
        Deploying with following values
        ===============================
        Stack name                   : event-relay-app
        Region                       : us-east-1
        Confirm changeset            : True
        Disable rollback             : False
        Deployment s3 bucket         : aws-sam-cli-managed-default-samclisourcebucket-t7rg7l8z5aqv
        Capabilities                 : ["CAPABILITY_IAM"]
        Parameter overrides          : {"Url": "http://internal-k8s-vpcexamp-vpcexamp-83bd367bda-628150
10.us-east-1.elb.amazonaws.com", "SecurityGroupId": "sg-0e835d8db922abac4", "PrivateSubnetId1": "subnet
-0f4a7060fc64c4d92", "PrivateSubnetId2": "subnet-074928b862bb6a48d", "Secret": "12345"}
        Signing Profiles             : {}
```

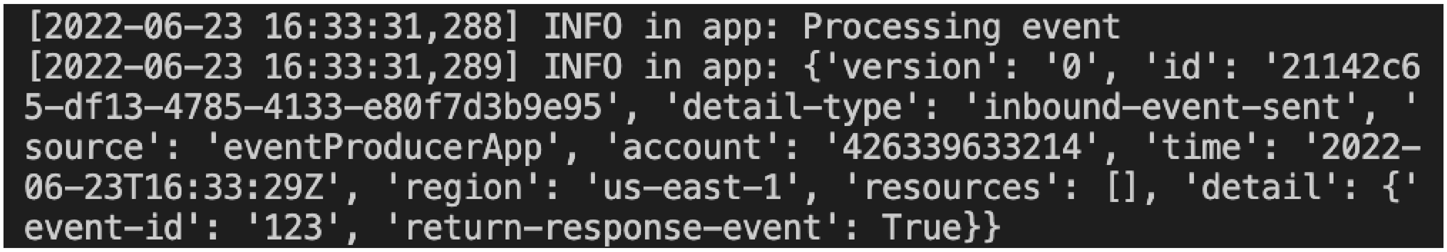
Test the event relay application

Both the Flask application in a VPC and the event relay application are now deployed. To test the event relay application, keep the Kubernetes pod logs from a previous step open to monitor requests coming into the Flask application.

1. You can open a new terminal window and run this AWS CLI command to put an event on the bus, or go to the EventBridge console, find your event bus, and use the Send events UI.

Bash
```bash
aws events put-events \
--entries '[{"EventBusName": "event-relay-bus" ,"Source": "eventProducerApp", "Detai
```

When the event is relayed to the Flask application, a POST request in the Kubernetes pod logs confirms that the application processed the event.



```
[2022-06-23 16:33:31,288] INFO in app: Processing event
[2022-06-23 16:33:31,289] INFO in app: {'version': '0', 'id': '21142c6
5-df13-4785-4133-e80f7d3b9e95', 'detail-type': 'inbound-event-sent', '
source': 'eventProducerApp', 'account': '426339633214', 'time': '2022-
06-23T16:33:29Z', 'region': 'us-east-1', 'resources': [], 'detail': {'
event-id': '123', 'return-response-event': True}}
```

2. Navigate to the CloudWatch Logs groups in the AWS Management Console. In the ‘/aws/events/event-bus-relay-logs’ group, there are logs for the EventBridge events. In ‘/aws/lambda/EventRelayFunction’ stream, the Lambda function relays the inbound event and puts a new outbound event on the EventBridge bus.

3. You can test the SQS dead letter queue by creating an error. For example, you can manually change the Lambda function code in the console to pass an incorrect value for the secret. After sending a test event, navigate to the

SQS queue in the console and poll for messages. The message shows the error message from the Flask application and the full event that failed to process.

Cleaning up

In the VPC dashboard in the AWS Management Console, find the Endpoints tab and delete the Secrets Manager VPC endpoint. Next, run the following commands to delete the rest of the example application. Be sure to run the commands in this order as some of the resources have dependencies on one another.

Bash

```bash
sam delete --stack-name event-relay-app
kubectl --namespace vpc-example-app delete ingress vpc-example-app-ingress
```

From the example-vpc-application directory, run this command.

Bash

```bash
eksctl delete cluster --config-file eksctl_config.yaml
```

Conclusion

Event-driven architectures and EventBridge can help you accelerate feature velocity and build scalable, fault tolerant applications. This post demonstrates how to send EventBridge events to a private endpoint in a VPC using a Lambda function to relay events and emit response events.

To learn more, read Getting started with event-driven architectures and visit EventBridge tutorials on Serverless Land.
TAGS: contributed, serverless