Combine AWS CloudWatch with an EKS Cluster

In this lesson, we will combine AWS CloudWatch with an EKS cluster using Fluentd.

WE'LL COVER THE FOLLOWING

Using Fluentd for CloudWatch

Unlike GKE that has a logging solution baked into a cluster, EKS requires us to set up a solution. It does provide CloudWatch service, but we need to ensure that the logs are shipped there from our cluster.

Just as before, we'll use **Fluentd** to collect logs and ship them to **CloudWatch**. Or, to be more precise, we'll use a **Fluentd** tag built specifically for **CloudWatch**. As you probably already know, we'll also need an IAM policy that will allow **Fluentd** to communicate with **CloudWatch**.

All in all, the setup we are about to make will be very similar to the one we did with **Papertrail**, except that we'll store the logs in **CloudWatch**, and that we'll have to put some effort into creating AWS permissions.

Before we proceed, I'll assume that you still have the environment variables `AWS_ACCESS_KEY_ID`, `AWS_SECRET_ACCESS_KEY`, and `AWS_DEFAULT_REGION` used in the eks-logging.sh Gist. If you don't, please create them.

Create an IAM policy

We need to create a new AWS Identity and Access Management (IAM) policy. For that, we need to find out the IAM role, and for that, we need the IAM

For that, we need to find out the IAM role, and for that, we need the IAM profile. If you're confused by that, it might help to know that you're not the only one. AWS permissions are anything but straightforward. Nevertheless, that's not the subject of this chapter, so I will assume at least a basic understanding of how IAM works.

If we reverse engineer the route to creating an IAM policy, the first thing we need is the profile.

```
PROFILE=$(aws iam \
  list-instance-profiles \
  | jq -r \
  ".InstanceProfiles[]\
  .InstanceProfileName" \
  | grep eksctl-$NAME-nodegroup-0)

echo $PROFILE
```

The **output** should be similar to the one that follows.

```
eksctl-devops25-nodegroup-0-NodeInstanceProfile-SBTFOBLRAKJF
```

Now that we know the profile, we can use it to retrieve the role.

```
ROLE=$(aws iam get-instance-profile \
  --instance-profile-name $PROFILE \
  | jq -r ".InstanceProfile.Roles[] \
  | .RoleName")

echo $ROLE
```

With the role at hand, we can finally create the policy. I already created one we can use, so let's take a quick look at it.

```
cat logging/eks-logs-policy.json
```

The **output** is as follows.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "logs:DescribeLogGroups",

        "logs:DescribeLogStreams",
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents"
      ],
      "Resource": "*",
      "Effect": "Allow"
    }
  ]
}
```

As you can see, there's nothing special about that policy. It defines permissions required for interaction with `logs` (**CloudWatch**) from inside our cluster. So, let's move on and create it.

```
aws iam put-role-policy \
  --role-name $ROLE \
  --policy-name eks-logs \
  --policy-document file://logging/eks-logs-policy.json
```

Finally, to be on the safe side, we'll retrieve the `eks-logs` policy and confirm that it was indeed created correctly.

```
aws iam get-role-policy \
  --role-name $ROLE \
  --policy-name eks-logs
```

The `PolicyDocument` section of the output should be the same as the JSON file we used to create the policy.

Setting up Fluentd

Now that we have the policy in place, we can turn our attention to **Fluentd**.

I already prepared a YAML for it, so let's take a quick look at it.

```
cat logging/fluentd-eks.yml
```

I won't go into the details of the YAML. You should be able to understand what it does by exploring it on your own. The key resources are the `fluentd-cloudwatch` ConfigMap that contains the configuration and the *DaemonSet* `cloudwatch` ConfigMap that contains the configuration and the *DaemonSet* with the same name that will run **Fluentd** Pod in each node of your cluster.

The only difficulty you might have with that YAML is to understand the **Fluentd** configuration, especially if that is the first time you're working with it. Nevertheless, we won't go into details, and I'll let you explore **Fluentd's** documentation on your own. Instead, we'll `apply` that YAML hoping that everything works as expected.

```
kubectl apply \
  -f logging/fluentd-eks.yml
```

Before we move into Cloudwatch UI, we'll retrieve **Fluentd** Pods and confirm that there is one in each node of the cluster.

```
kubectl -n logging get pods
```

In my case, the output shows three `fluentd-cloudwatch` Pods matching the number of nodes in my EKS cluster.

```
NAME                       READY   STATUS    RESTARTS   AGE
fluentd-cloudwatch-7dp5b   1/1     Running   0          19s
fluentd-cloudwatch-zq98z   1/1     Running   0          19s
fluentd-cloudwatch-zrrk7   1/1     Running   0          19s
```

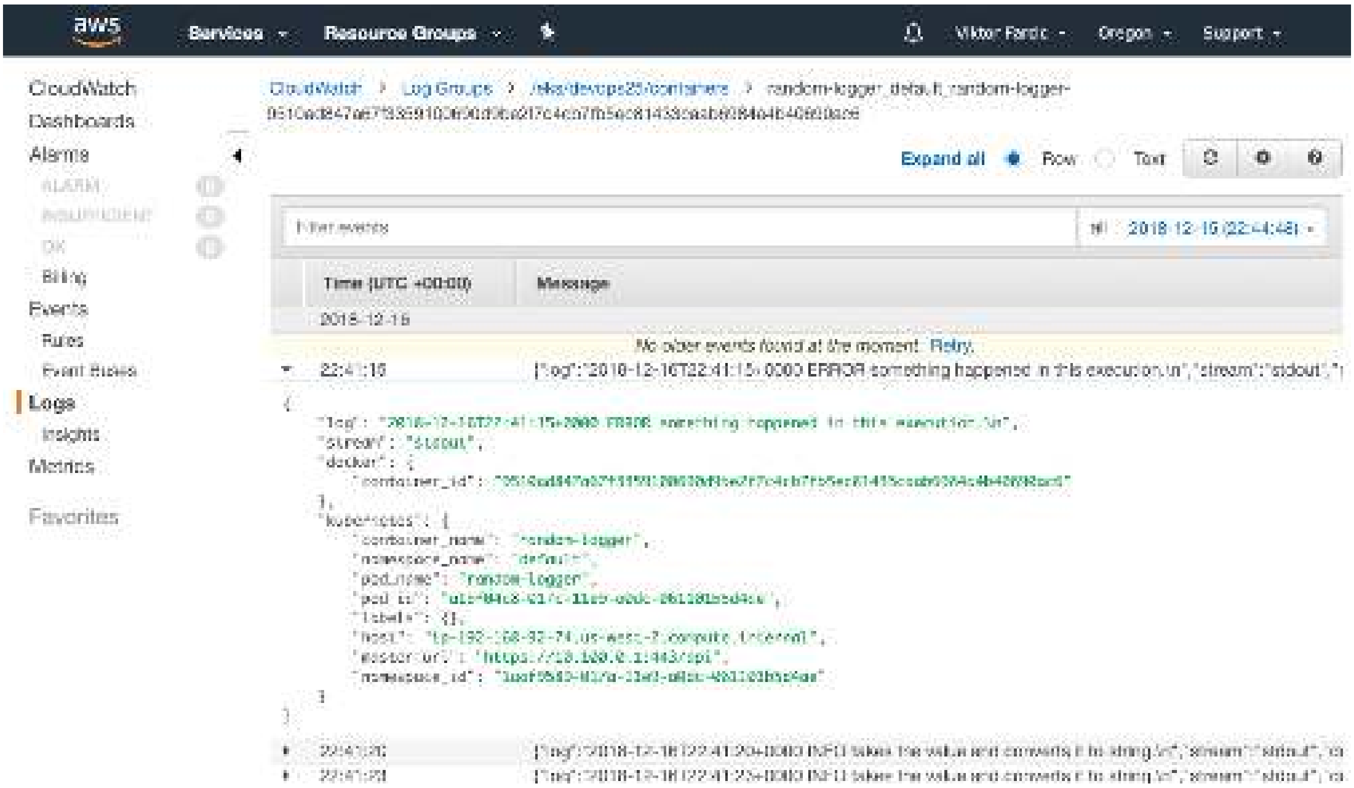
Now that everything seems to be working inside our cluster, the time has come to move into CloudWatch UI.

CloudWatch UI

```
open "https://$AWS_DEFAULT_REGION.console.aws.amazon.com/cloudwatch/home?#
logStream:group=/eks/$NAME/containers"
```

Please type *random-logger* in the *Log Stream Name Prefix* field and press the enter key. As a result, only one stream should be available. Click it.

Once inside the *random-logger* screen, you should see all the logs generated by that Pod. I'll leave it to you to explore the available options (there aren't many).



```
2018-12-16
                 No older events found at the moment. Retry.
▾ 22:41:16    {"log":"2018-12-16T22:41:15+0000 ERROR something happened in this execution.\n","stream":"stdout","
{
    "log": "2018-12-16T22:41:15+0000 ERROR something happened in this execution.\n",
    "stream": "stdout",
    "docker": {
        "container_id": "[illegible]"
    },
    "kubernetes": {
        "container_name": "random-logger",
        "namespace_name": "default",
        "pod_name": "random-logger",
        "pod_id": "[illegible]",
        "labels": {},
        "host": "ip-192-168-92-74.us-west-2.compute.internal",
        "master_url": "https://10.100.0.1:443/api",
        "namespace_id": "[illegible]"
    }
}
▸ 22:41:20    {"log":"2018-12-16T22:41:20+0000 INFO takes the value and converts it to string.\n","stream":"stdout","co
▸ 22:41:25    {"log":"2018-12-16T22:41:25+0000 INFO takes the value and converts it to string.\n","stream":"stdout","co
```

AWS CloudWatch events screen

Q

The `PolicyDocument` section of the output should be the same as the JSON file we used to create the policy.

COMPLETED 0%

1 of 1

Delete Fluentd resources

Once you're done exploring **CloudWatch**, we'll proceed by deleting the **Fluentd** resources as well as the policy and the log group. We still have more logging solutions to explore. If you choose to use **CloudWatch** with **Fluentd**, you should be able to replicate the same installation steps in your "real" cluster.

```
kubectl delete \
  -f logging/fluentd-eks.yml

aws iam delete-role-policy \
    --role-name $ROLE \
    --policy-name eks-logs

aws logs delete-log-group \
  --log-group-name \
  "/eks/devops25/containers"
```

In the next lesson, we will combine **Azure Log Analytics** with an AKS cluster.