Whitepaper

Skripted.

Build a Centralized Logging Solution with Amazon OpenSearch & Terraform

Solving observability challenges in a multi-account environment using Infrastructure as Code.

October 2023

Written by

Maxim Schram

IN COLLABORATION WITH:

SCRATCH FOUNDATION

Includes a complete and ready-to-run Terraform configuration.

GitHub

Table of contents

Acknowledgements **3**
Introduction **4**
Chapter 1. Why Centralized Logging Matters **5**
Multi-account strategy 5
Using CloudWatch for logging 5
CloudWatch in a multi-account environment 5
Centralized logging 6
How it works 6
Chapter 2. Infrastructure as Code **7**
The CloudFormation stack 7
Terraform implementation 7
Multi-account configuration 8
Folder structure 9
Terraform modules 9
Chapter 3. Configuration **10**
Mgt account configuration 10
Workload account configuration 10
Log-archive account configuration 11
Chapter 4. Installation & provisioning **13**
Pre-requisites 13
Create Terraform roles 13
Provisioning the Terraform backend, KMS keys and secrets 13
Provisioning the Centralized Logging Solution 14
Provisioning the Log Subscription 14
Appendix - Authentication **15**
VPN alternative 15

Acknowledgements

This whitepaper was made possible through support of the Scratch Foundation. The Scratch Foundation is the driving force behind scratch.org, a free, online application that allows kids around the world to express their ideas through coding.

I was honored to be able to help Scratch with their infrastructure challenges and, through collaboration with the Scratch team, discovered and applied many state of the art solutions along the way. One of which is the Centralized Logging Solution described in this whitepaper.

I'd like to thank the entire Scratch team for the opportunity, with special thanks to La Shauna Nichols, Colby Gutierrez-Kraybill, Christopher Willis-Ford, Ron de las Atlas, Alexis Goodfellow and Erik Lazzaro.

Introduction

When utilizing AWS in a sizable or mid-sized organization, it is highly probable that you are also implementing a multi-account hierarchy through AWS Organizations. The challenge is that CloudWatch logs are only accessible within the originating account by default, which presents various obstacles.

In this whitepaper I will walk you through the provisioning of a (modified) centralized logging solution, based on a solution made available by AWS under the Apache License Version 2.0.

My version of the solution is a derivative work that has been optimized for security in a few places, and comes as a complete Terraform configuration that can be adapted to your needs.

If you need help at any point, feel free to drop me a message at max@skripted.io.

Happy provisioning!

Maxim (Max) Schram
Founder @ Skripted | Infrastructure as Code
www.skripted.io - max@skripted.io

Github repo:
https://github.com/skripted-io/aws-centralized-logging-opensearch-terraform

Chapter 1. Why Centralized Logging Matters

Let's begin by understanding why a centralized logging solution is useful.

Multi-account strategy

If you are using AWS in a large or medium-sized organization, chances are you are also using a multi-account hierarchy with AWS Organizations.

Distributing AWS resources across accounts can be useful in many circumstances. For example, it can help reduce the blast radius of your infrastructure. Should a bad actor gain access to one account then (hopefully) this will limit the exposure of your other accounts.

It also allows you to separate workloads by environment (i.e. development, staging, production), reducing the chances of accidentally impacting a production workload while developing.

Or maybe you want to assign dedicated team members to one account or a subset of accounts, without giving them access to everything else. These are all perfect use cases for an infrastructure design involving multiple AWS accounts.

Using CloudWatch for logging

Most AWS resources can log to CloudWatch. For example you can have EC2 nginx access logs or Lambda invocation logs. When things aren't working as expected with your application, CloudWatch can be a valuable tool for finding the culprit.

Each AWS resource will log to its own "log group". For example: `/aws/lambda/<function name>` and `/ec2/nginx/logs` are possible log group names.

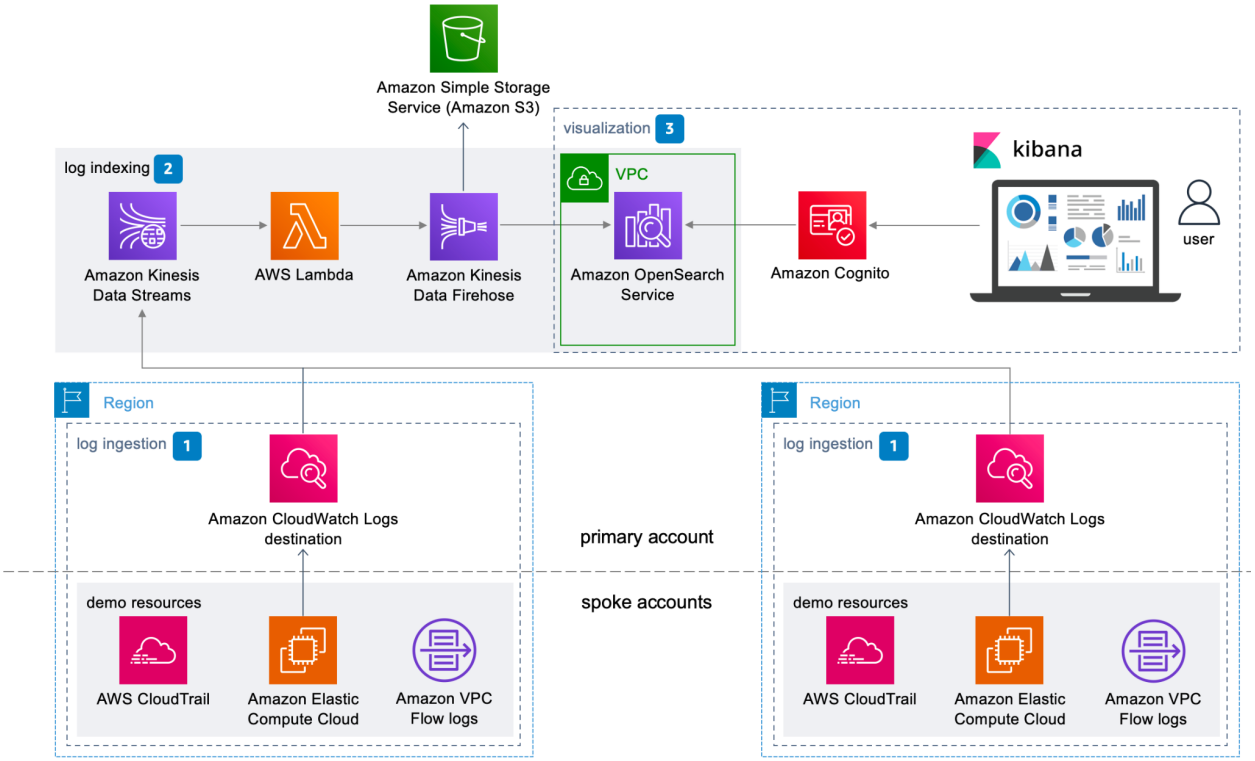
CloudWatch in a multi-account environment

CloudWatch allows you to search through large amounts of log data and find the needle in your haystack. But here's the challenge; Let's say you have development, staging and production workloads and each workload is in a dedicated AWS account. Additionally, you have workloads in 3 different AWS regions.

This means your CloudWatch log groups can be in 9 different places. To reach each of these logs, you first have to log in to the correct account and select the right region. When time is critical, you will lose valuable time looking for the right log, before you can even start looking for the cause of the issue you are investigating.

Centralized logging

As its name suggests, centralized logging is a way of aggregating your logs in one place. Luckily, through one of its many Whitepapers, AWS offers a ready solution that you can implement using CloudFormation scripts. Here's a schematic diagram of the available stack:



How it works

In a nutshell this solution provisions a Kinesis Data Stream in a dedicated Logging (primary) account. The stream ingests CloudWatch logs from other (spoke) accounts. A Lambda function in the primary account converts the logs into a unified format that can be indexed by AWS OpenSearch Service (formerly ElasticSearch). In your spoke accounts, you can "subscribe" your logs of choice to the Kinesis stream using a CloudWatch Log Destination. Finally, you can use a Kibana dashboard to search through and visualize the logs. Once set up, you will be able to find all logs for all accounts and regions in the primary account.

Chapter 2. Infrastructure as Code

As many of you already know, I'm a big fan of Infrastructure as Code. So I got excited when I saw that AWS was providing this solution as a CloudFormation stack.

The CloudFormation stack

While the provided CloudFormation script is quick and convenient, it doesn't provide us with a lot of flexibility. For starters, the script comes as one long JSON file that's difficult to navigate and break up. And the provided parameters only offer so much customization.

Additionally, it will create a VPC for the OpenSearch service, but doesn't let you specify an existing VPC. KMS keys are AWS managed and not customer managed. But more importantly, if you are already using Terraform you probably don't want to complicate your architecture management by introducing another IaC language.

Terraform implementation

In my experience, Terraform is far more suitable for more complex setups than CloudFormation, especially when you are using a multi-account hierarchy.

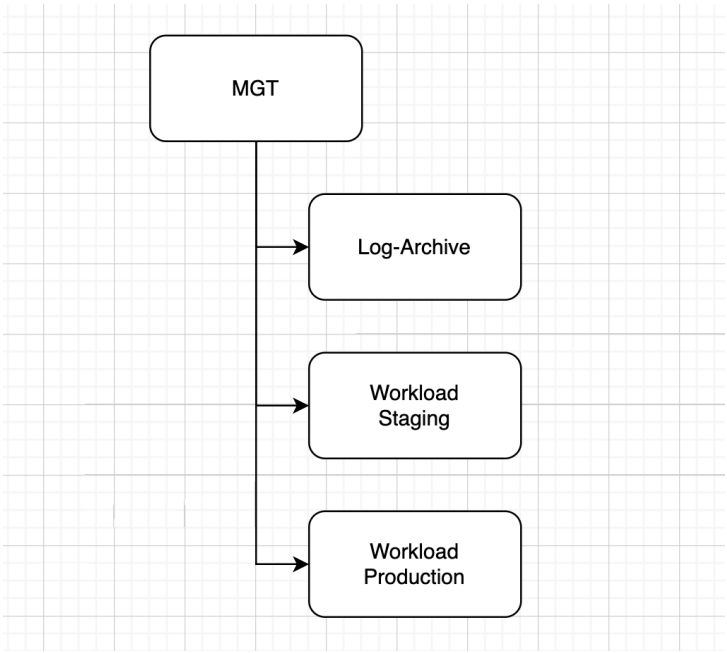
That's why I decided to reverse engineer the CloudFormation stack and break it down into separate Terraform modules. The complete Terraform configuration is available on GitHub here and can be customized further to your situation.

Need help implementing this infrastructure?

email me at max@skripted.io or
give me a call +1 (646) 543-2059.

Multi-account configuration

The Terraform configuration is organized around a hierarchy of 4 AWS accounts. Because we are using Terraform modules for each of the components, you can easily modify it to match your situation.



The setup is pretty straight forward:

- The `mgt` account stores the Terraform state (S3/DynamoDB), KMS keys and Secret Manager secrets.
- The `log-archive` account contains the AWS OpenSearch cluster and related resources.
- Finally, `staging` and `production workload` accounts generate test logs that are sent to the log-archive account.

Folder structure

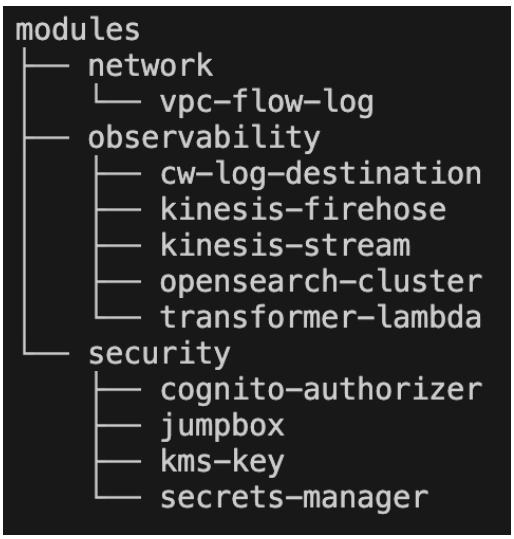
The folder structure of the Terraform configuration follows the AWS account structure with a dedicated configuration for each account. A `modules` folder contains dedicated modules for each element of the stack and is organized by purpose: `network`, `observability` and `security`.



Terraform modules

One of the great advantages of Terraform over CloudFormation is the fact that you can create modules from your code. Modules are resource components that you are planning to use often, or would like to be able to move around easily.

If you look at the modules overview of this configuration you should see familiarities with the diagram we saw earlier.



The `vpc-flow-log module` creates flow logs in our workload accounts. These will generate data that we want to capture.

The `cw-log-destination` module creates a CloudWatch log destination that we can subscribe our flow logs to.

The `kinesis-stream`, `transformer-lambda` and `kinesis-firehose` modules take care of the log ingestion and conversion.

The `opensearch-cluster` module builds the AWS OpenSearch resources. Finally, the `security` folder contains components for encrypting our data and providing secure access to OpenSearch.

It goes beyond the scope of this Whitepaper to dive deeper into the individual modules. But I went to great lengths to include comments inside the module configuration to help you understand as much as possible.

Chapter 3. Configuration

Each set of Terraform configuration files has been conveniently set up using variables. This allows us to quickly make changes without having to dive deeper into the code. Each of the account folders (mgt, log-archive, workload) contains a `terraform.tfvars` file with variables that you should modify to match your situation. Let's have a quick look...

Mgt account configuration

```
# /accounts/mgt/terraform.tfvars

aws_region                          = "us-east-1"    # Set the AWS region to deploy to
environment                         = "prod"         # Set to the environment (only used for tagging)
log_archive_account_id              = "999999999999" # Replace with the account ID of your log-archive account
workload_staging_account_id         = "999999999999" # Replace with the account ID of your staging account
workload_prod_account_id            = "999999999999" # Replace with the account ID of your production account
```

The above is the vars file for the `mgt account`. You begin by specifying the AWS region the resources need to be deployed in. The `environment` variable is used for tagging the resources so you can find them more easily and can use them as [cost allocation tags](#).

Next you specify the `account IDs` for the member accounts. This way we can create KMS keys with permissions that allow access from those accounts.

Workload account configuration

```
# /accounts/workload/us-west-2/terraform.tfvars

aws_region                      = "us-east-1"
environment                     = "staging"
account_terraform_role_arn      = "arn:aws:iam::9999999999:role/TerraformRole"
vpc_id                          = "vpc-0123456789abcdef"
```

The configuration of the workload account(s) is also relatively simple. You specify the AWS region and the environment. Next you specify the ARN of the IAM role that Terraform can assume to provision resources in this account (more on that in the next chapter). You also specify the ID of the VPC you want the VPC Flow Logs to be attached to. Of course you can also set this dynamically if you created the VPC in the same Terraform configuration.

Note that as a matter of example you will find two folders in the workload account: `us-east-1/staging` and `us-west-2/prod`. This is to illustrate how you can subscribe different environments in different regions to the central logging solution.

Log-archive account configuration

Finally, the configuration file of the log-archive account is a bit longer but it should be straightforward. Again we start by specifying region, environment and Terraform roles:

```
# /accounts/log-archive/terraform.tfvars

aws_region                                = "us-east-1"    # Set the AWS region to deploy to
environment                               = "prod"         # Set to the environment (only used for tagging)
account_terraform_role_arn                = "arn:aws:iam::999999999999:role/TerraformRole" # Replace with the ARN of your
workload_staging_account_id               = "999999999999" # Replace with the account ID of your staging account
workload_prod_account_id                  = "999999999999" # Replace with the account ID of your production account
```

Next you need to specify the VPC id, subnet IDs and number of AZs of the VPC you want to provision the OpenSearch cluster in. Again, this can also be dynamically assigned if you've created your VPC with Terraform. In that case, simply open `main.tf`, find the reference to the variables and replace them with dynamic values.

```
# VPC
vpc_id                                    = "vpc-abcdefgh0123456789" # Replace with the ID of your VPC in the log-archive account
vpc_public_subnet_ids                     = ["subnet-abcdefgh0123456789","subnet-0f5380a3d10b2cf1b"] # Replace with the public subnets of your VPC in t
vpc_private_subnet_ids                    = ["subnet-abcdefgh0123456789","subnet-abcdefgh0123456789"] # Replace with the private subnets of your VPC in
vpc_az_count                              = 2 # Update to the number of AZs in your VPC in the log-archive account
```

The next section is dedicated to configuring the OpenSearch cluster. For this example, we use small instance types for the nodes, a limited number of nodes and limited storage. You are free to change these based on expected data size. You can also provision small now and increase the values later.

```
# Centralized Logging OpenSearch
centralized_logging_os_master_nodes_type      = "t3.small.search"
centralized_logging_os_master_nodes_count     = 3
centralized_logging_os_data_nodes_type        = "t3.small.search"
centralized_logging_os_data_nodes_count       = 2 # Must be equal to the number of availability zones in the log-archive account
centralized_logging_os_ebs_volume_size        = 100
```

Need help implementing this infrastructure?

email me at max@skripted.io or
give me a call +1 (646) 543-2059.

Finally there's a section to set up the Jump box (also called a bastion host) that you can use to connect to the private cluster. I'll go into a bit more detail about this in the Appendix. But if you are just looking for what to fill out here:

- `jumpbox_ami` is the AMI id of a Microsoft Windows Server 2019 image in your region. You should be able to look that up in the EC2 console.
- `jumpbox_key_name` is the name of a keypair to connect to the box. If you have not created one yet then here are instructions.
- Finally, enter your own IP address after `client_ip` so we can configure the security group of the Jump box to only allow traffic from your ip.

```
# Jumpbox
jumpbox_ami                         = "ami-abcdefgh0123456789" # Replace with the correct one for your region - Microsoft Windows Server 2019
jumpbox_key_name                    = "ec2-keypair" # Replace with the name of your keypair in the log-archive account
client_ip                           = "8.8.8.8" # Replace with your home, office, or VPN IP address
```

Once you have configured these variables in each of these accounts, you are ready for installation.

Chapter 4. Installation & provisioning

In this fourth and final chapter, I'll walk you through the installation and provisioning of the centralized logging solution using Terraform. The repository has been set up to work with new accounts. If you are already familiar with Terraform you can integrate the configuration into your existing Terraform configuration.

Pre-requisites

- Latest AWS CLI installed
- Latest Terraform CLI installed
- 4 AWS accounts: mgt, log-archive and workloads.

Create Terraform roles

Our mgt account will store the Terraform state and provision resources in the log-archive and workload accounts. To be able to do so, we need to create a role that Terraform can assume while running in the mgt account. Here's how to set that up:

1. In the file `/terraform-role.sh` replace <Mgt-Account-ID> with the ID of your mgt account. Next, run the script in your terminal while the AWS CLI is authenticated to the log-archive and workload accounts consecutively. Note down the ARNs of the IAM roles that were created.
2. Paste the ARNs into `/accounts/log-archive/terraform.tfvars` and `/accounts/workload/terraform.tfvars` respectively where it says `account_terraform_role_arn` (like explained in the previous chapter)

Provisioning the Terraform backend, KMS keys and secrets

Make sure your local terminal is authenticated with the mgt account. Then do the following:

1. In `/accounts/mgt` run `terraform init`
2. In `/account/mgt` run `terraform apply`
3. In `/accounts/mgt/state.tf` uncomment the configuration and make sure the S3 bucket name and DynamoDB table match the resources that have been created in step 2. They default to `central-logging-demo-tf-state` and `central-logging-demo-tf-lock-table`
4. In `/accounts/mgt` run `terraform init -migrate-state`. This will move the Terraform state from your local machine to the remote bucket.

Provisioning the Centralized Logging Solution

Make sure your local terminal is authenticated with the log-archive account. Then do the following:

1. In `/accounts/log-archive` run `terraform init`
2. In `/accounts/log-archive` run `terraform apply`

Provisioning the Log Subscription

Remember that the Terraform configuration includes an example on how to subscribe your logs to the log destination that was provisioned in log-archive. To illustrate how this would work in a staging/prod environment in multiple regions, there is an example `us-east-1` and `us-west-2` folders. Each contains the same configuration with the exception of the values for the region and the environment.

To set up the staging example:

1. In `/accounts/workload/us-east-1/staging` run `terraform init`
2. In `/accounts/workload/us-east-1/staging` run `terraform apply`

To set up the production example:

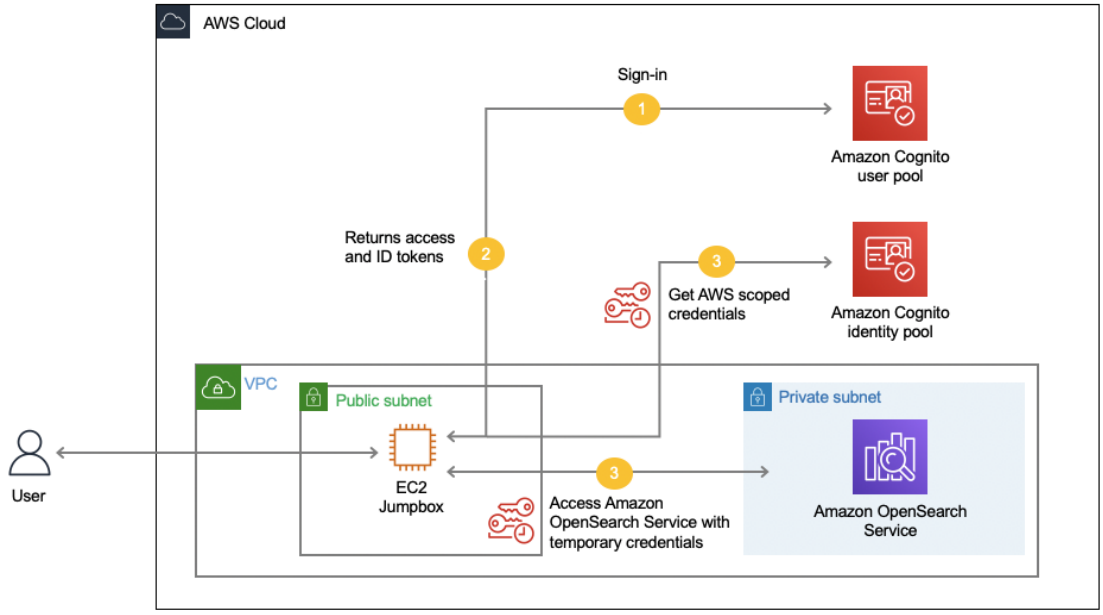
1. In `/accounts/workload/us-west-2/prod` run `terraform init`
2. In `/accounts/workload/us-west-2/prod` run `terraform apply`

Need help implementing this infrastructure?

E-mail me at max@skripted.io or give me a call +1 (646) 543-2059.

Appendix - Authentication

By default this Terraform configuration will launch a Windows Server-based Jump box (also called a Bastion host) in the public subnet of your VPC. Once provisioned, you can connect the Windows machine with RDP, open a browser on the machine and paste in the domain URL of the OpenSearch Kibana dashboard. Kibana then uses a Cognito User Pool to authenticate you and a Cognito Identity Pool to authorize your access to the Kibana dashboard.



This way you can bypass the fact that the OpenSearch cluster is a private subnet (and hence doesn't allow outside traffic). While I'm not a fan of Bastion hosts I decided to include it in the script because the original AWS example also includes it. Once the stack has been created, follow the instructions in `/cognito-user.sh` to create a username and password pair for Cognito authentication.

VPN alternative

My preference is to use VPN connections. If you also want to use a VPN connection instead of the Jump box and Cognito authorizer, then you can comment out the creation of these resources. When you are not using Cognito you can use a master username and password keypair. The Terraform for provisioning an AWS Secrets Manager resource is already enabled. And there are instructions in the OpenSearch module on how to enable those credentials instead of the Cognito authorizer.

Copyright 2023 by Maxim Schram / Skripted. All Rights Reserved.

THIS WHITEPAPER IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE WHITEPAPER OR THE USE OR OTHER DEALINGS IN THE WHITEPAPER.