Creating a DevSecOps Data-Analytics Dashboard for AWS Services Using ELK stack.

Garrett Partenza
KBR, Inc.
Westminster, MD
gpartenza@sgt-inc.com

A.J. Ellis
KBR, Inc.
Ellicott City, MD
ajellis@sgt-inc.com

Nathaniel J. Ellis
KBR, Inc.
Ellicott City, MD
nellis@sgt-inc.com

***Abstract*—DevSecOps is a methodology that combines development, security, and operations into a single process of software development. This paper explores how one could create a set of dashboards that displays that data from DevSecOps AWS services in a meaningful manner.**

Keywords—Cloud computing, AWS environment, continuous integration and continuous deployment, lambda deployment package, cronjob, Elasticsearch endpoint.

I. Background

The typical process of creating an application goes beyond simply developing code and deploying into a program. When developing commercial software, two key areas of focus are *Operations* and *Security* management. *Operations* refers to the handling of the complete product, which involves monitoring and analyzing the working program. *Security* is crucial to ensure protection against data loss or compromise, but it is often handled as a mere afterthought.

However, the concept of *DevSecOps* changes the standard by integrating these two parts directly into the development process, rather than as separate tasks to be handled afterward. As a result, operations and security-related tasks are dealt with from the start. This paper will go into detail on the benefits of such an approach, as well as a proof-of-concept on how this methodology could be integrated into a cloud infrastructure dashboard using Amazon Web Services.

What is DevOps and DevSecOps?

The concept of *DevOps* is an approach focused specifically on combining *development* and *operations* tasks. This results in a cyclical software development pipeline that includes processes such as code integration and deployment, while automating the logging and monitoring of these tasks.

DevSecOps takes this a step further by integrating and automating security-related processes in between code development tasks, and logging and monitoring the results of these security scans as well.

What are the characteristics of DevSecOps?

No Silo Thinking: Because development, security, and IT operations are combined in all stages of development, there are no separate teams for each process. For example, the duty of security is directly incorporated into the role of the developer, and vice-versa.

Cyclical: A DevSecOps project is constantly being reviewed and revised over time as the software is developed (much like agile scrum software development)

Automation: As many different processes are incorporated together in the cycle, it becomes possible to automate them in a pipeline. This as a result increases efficiency, making it more beneficial to automate most tasks.

What are the advantages of DevSecOps methodology?

Increased Collaboration: Because all three teams work together, instead of separating processes, everyone is involved and aware of what is happening through all stages of development.

Increased Efficiency: Combining processes into an automated cyclical pipeline allows software to be developed much faster. Changes can be made much more easily and everyone is aware of such changes.

II. Problem Definition

The concept of *DevSecOps* sets a new standard for the methodology to use when creating and releasing software. The resulting question for the developer is one of *how to integrate* this methodology. This project targeted several users who would benefit from it and used that as a basis for how to integrate it. Overall, the user should have ready access to the data and/or tools he or she needs the most.

Program Managers: Overseeing multiple projects, these users will need the most significant information from all projects in one place, readily accessible and easy to understand.

Project Managers: The head of an individual project should have easy access to more in-depth information on their specific project.

Developers: These users should be able to focus on coding, but as per nature of DevSecOps, security and operations tasks will be integrated. So, developers should be able to understand these processes and automate what is necessary.

Security Engineers: These users will need to easily access a wide variety of security-related metrics, and easily be able to detect and prioritize issues.

System Administrators: Detailed information about all processes should be readily accessible to administrators and well-organized.

Potential Customers: In order to showcase the capabilities to potential users, a demonstration dashboard must be available. This dashboard can make use of data to showcase the benefits of the DevSecOps infrastructure.

III. Defining Terminology

A. Cloud computing

As opposed to an on-premises computer infrastructure, cloud computing is becoming more common because of its benefits with cost, maintenance, and even security.

B. AWS environment

Amazon Web Services (AWS) provides solutions for employing a cloud computing infrastructure. An AWS environment refers to this infrastructure in an AWS account. The entire DevSecOps backend framework makes use of AWS tools inside a single account. This framework could be replicated in another AWS environment, or it could be linked to tools in another environment by utilizing "cross-account access."

C. Continuous Integration/Continuous Deployment

"Integration" refers to building code from a source repository into an application, and "deployment" refers to installing that application for release. The cyclical and automated characteristics of DevSecOps allows for the concept of Continuous Integration and Continuous Deployment (CI/CD) to be employed. It allows changes in the application to be made quickly and then tested and refined before release while integrating the proper security and monitoring measures. This is typically set up as a pipeline where any change will trigger said pipeline and automatically run the required security/operations tasks, drastically reducing the time required to update code.

D. Lambda deployment package

A deployment package is a ZIP archive that contains your function code and dependencies. Lambda packages are required if you need to include libraries and dependencies other than the AWS SDK. An example of this is the Elasticsearch Python module used for indexing data from a python script to our Elasticsearch endpoint.

E. Cronjob

Cronjob is a time-based job scheduler in Unix-like computer operating systems. Users that set up and maintain software environments use cronjob to schedule jobs to run periodically at fixed times, dates, or intervals.

F. Elasticsearch endpoint

An Elasticsearch endpoint is a unique URL used to specify a destination to which data can be indexed.

IV. Tools Used

A. *Amazon CodePipeline*

Amazon CodePipeline is an AWS managed software development tool that allows for automation of builds and deployments of source code. It integrates three other AWS tools as well: CodeCommit for storing source code, CodeBuild for integrating the code into an application, and CodeDeploy for deploying the application. For the DevSecOps project, CodePipeline allows a streamlined CI/CD pipeline that can easily integrate with other AWS services and easily push logs to Elasticsearch.

B. Amazon Elastic Compute Cloud (EC2)

An EC2 is a cloud-computing service that allows users to configure and run virtual machines using Amazon-provided hardware. The DevSecOps project makes use of several EC2 instances to run open-source applications that integrate into the pipeline.

C. Amazon GuardDuty

Amazon GuardDuty is an AWS-managed security agent that watches the environment for security threats using machine learning and anomaly detection. It was chosen for this project for its vast span of information. Types of data collected ranges from descriptions of threats to coordinates of where those threats come from.

D. Amazon Inspector

Amazon Inspector is an AWS-managed vulnerability scanner for EC2 virtual machines. It offers a vast span of information as well as consumer directions on how to resolve issues found.

E. Amazon CloudTrail

Amazon CloudTrail is a service that enables governance, compliance, operational auditing, and risk auditing of an AWS account. With CloudTrail, users can log, continuously monitor, and retain account activity related to actions across an AWS infrastructure. CloudTrail also provides event history of AWS account activity, including actions taken through the AWS Management Console, AWS SDKs, command line tools, and other AWS services.

F. Elasticsearch

Elasticsearch is the heart of the ELK stack. It is the engine that stores, indexes, and aggregates real-time data for search and analysis.

G. Logstash

Logstash is an application for getting data into Elasticsearch. It is highly configurable for a wide variety of input and output data types.

H. Kibana

Kibana is a web user interface for viewing and managing all this data. It includes configurable data visualizations (such as graphs and pie charts) for creating dashboards, which can display the data for monitoring and analysis.

I. SonarQube

SonarQube is an application that scans a source code repository and notifies the user of source code vulnerabilities, as well as bugs and

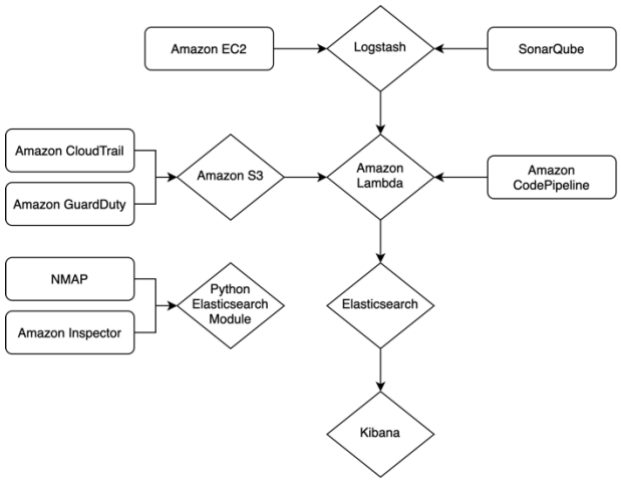
code smells, which are potential flaws that could lead to larger issues. In the DevSecOps infrastructure, it is run on a CodeCommit repository any time a change is pushed through CodePipeline.

J. NMAP

Nmap is a free and open-source network scanner. It is used to discover hosts and services on a computer network by sending packets and analyzing the responses. The responses are analyzed through a fingerprinting method, which compares the responses to known responses within NMAPs own database. Through using NMAP, security engineers can see what their machines look like to the outside.

V. High Level Architecture

Below is an overview diagram of all pipelines leading to the Elasticsearch endpoint. All tools within this architecture are Amazon services with the exception of SonarQube and NMAP. As of August 2020, AWS does not yet offer a static code analysis tool nor a network reconnaissance scanner. As a result, these two third-party tools had to be used.



VI. Implmentation

While most of the Amazon services used in this project are backed by extensive documentation and thus are relatively easy to set up, the real challenge in creating a DevSecOps dashboard is moving informative data from its origin into the dashboard in real-time. Questions like *how to push Amazon GuardDuty findings to an Elasticsearch endpoint* are severely underexplored, and thus unique pipelines had to be established in order for the dashboard to be created.

This section explores how the data from each service was moved from its origin to our Elasticsearch endpoint.

A. Amazon CodePipeline.

Amazon automatically creates logs for each event that occurs within a CodePipeline. In order to grab these logs, a CloudWatch rule was established that constantly monitors all Code Pipelines within the AWS environment. When an event is logged, this CloudWatch rule captures the log details in dictionary form and pushes that dictionary to a lambda function. This lambda function, written in Python 3.8, utilizes the Elasticsearch python module to send all incoming log details to our Elasticsearch endpoint.

B. Amazon Inspector.

As of August 2020, Amazon does not offer an automated method to move Inspector findings outside of the Inspector dashboard. Amazon does offer to publish findings to an SNS topic, however, finding details get omitted in this process and thus make it unusable for the purposes of this project.

In order to move findings from Inspector to our Elasticsearch endpoint in real-time, a unique pipeline was established. In this pipeline, the task of starting Inspector runs, retrieving findings, and sending those findings to Elasticsearch is all automated through the command line. First, cronjob is used to automated two Python scripts each an hour apart. The first script uses Amazon's CLI tools to begin an Inspector run. The second script pulls findings from the previous Inspector run to send to our Elasticsearch endpoint.

C. Amazon GuardDuty.

In the Amazon GuardDuty dashboard, an S3 bucket was specified to store all threats. Next, a lambda function was created to be triggered by this S3 bucket whenever a new threat gets stored. Upon being triggered, the function runs a Python 3.8 script that pulls the detected threat from the S3 bucket and sends its details to our Elasticsearch endpoint.

D. Amazon CloudTrail

In the Amazon CloudTrail dashboard, an S3 bucket was specified to store all CloudTrail events. Next, a lambda function was created to be triggered by this S3 bucket whenever a new event gets stored. Upon being triggered, the function runs a Python 3.8 script that pulls the detected threat from the S3 bucket and sends its details to our Elasticsearch endpoint.

E. Amazon EC2 Logs.

For most instances, logs are automatically saved in common log format to the path */var/logs*. To push these logs to Elasticsearch in real-time, Logstash was deployed on each machine and configured according. First, Java 1.8.0 was installed on the instance as a requirement of the Logstash service. Next, a Logstash configuration file was written that specified the input files to monitor, and the output Elasticsearch endpoint. Finally, Logstash was given read permissions on all specified log files and then run as a background service using *systemctl*.

F. SonarQube.

As of August 2020, Amazon does not offer a native static code analysis service. As a result, a the third-party open-source tool called *SonarQube* was integrated.

The SonarQube service was installed on an Amazon EC2 instance where data for code analysis scans are stored and then pushed to Elasticsearch through Logstash. To initiate the scan, a separate but linked application called the Sonar Scanner is run though AWS CodeBuild on a CodeCommit repository (through CodePipeline).

G. NMAP

As of August 2020, Amazon does not offer a native penetration testing service. As a result, a third-party network reconnaissance tool called NMAP had to be deployed.

To automate these network scans, an EC2 was deployed to automate command line calls and send the scan output to Elasticsearch. To accomplish this, cronjob is scheduled to automate a Python script every hour. The script holds an array of desired targets and uses Python's subprocess module to run the scan using the command line and capture the output response. The response is then sent to our Elasticsearch endpoint using the Elasticsearch Python module.

VII. Demonstration

Figure 1. and Figure 2. On the next page demonstrate a subset of the eight dashboards developed in this project.

VIII. Future work

Future work for this project remains in two main areas – *deployability and efficiency*. On the topic of deployability, the backend pipelines of this project are not ready for consumer use. Each component of the pipelines had to be manually configured to send data to our Elasticsearch endpoint. For example, the lambda functions pushing Amazon service data to our Elasticsearch endpoint have the endpoint hardcoded into them. In order for a potential customer to use this product, KBR engineers would have to manually go into their AWS environment and configure the functions to fit the customers criteria. Ideally in the future this task would be omitted through the automated development of deployment packages for each customer who purchased a dashboard. In this way, customers would be handed a pre-configured deployment package to simply upload to a lambda function themselves. On the topic of efficiency, the developers of this project have not yet explored ways to minimize the monetary costs of running this project on an AWS environment. Doing so would reduce the cost of maintaining these dashboards and provide greater incentive for its use.

IX. Acknowledgements

The authors gratefully acknowledge the significant contributions of the following individuals, in this paper:

Blake Wishard, the previous KBR developer for DevSecOps in 2019 and scrum master for the Summer 2020 intern program, for his inputs about the needed improvements of an already ongoing project. More specifically, Blake routinely assisted in role and permission-based errors encountered during the creation of this project.

Nathaniel J. Ellis and Sam Nicholson, Project Managers with KBR for their editorial review of this paper.

X. References

[1] Software. The source code for the backend pipelines is available at https://github.com/garrett-partenza-us/KBR-DevSecOps-Project.

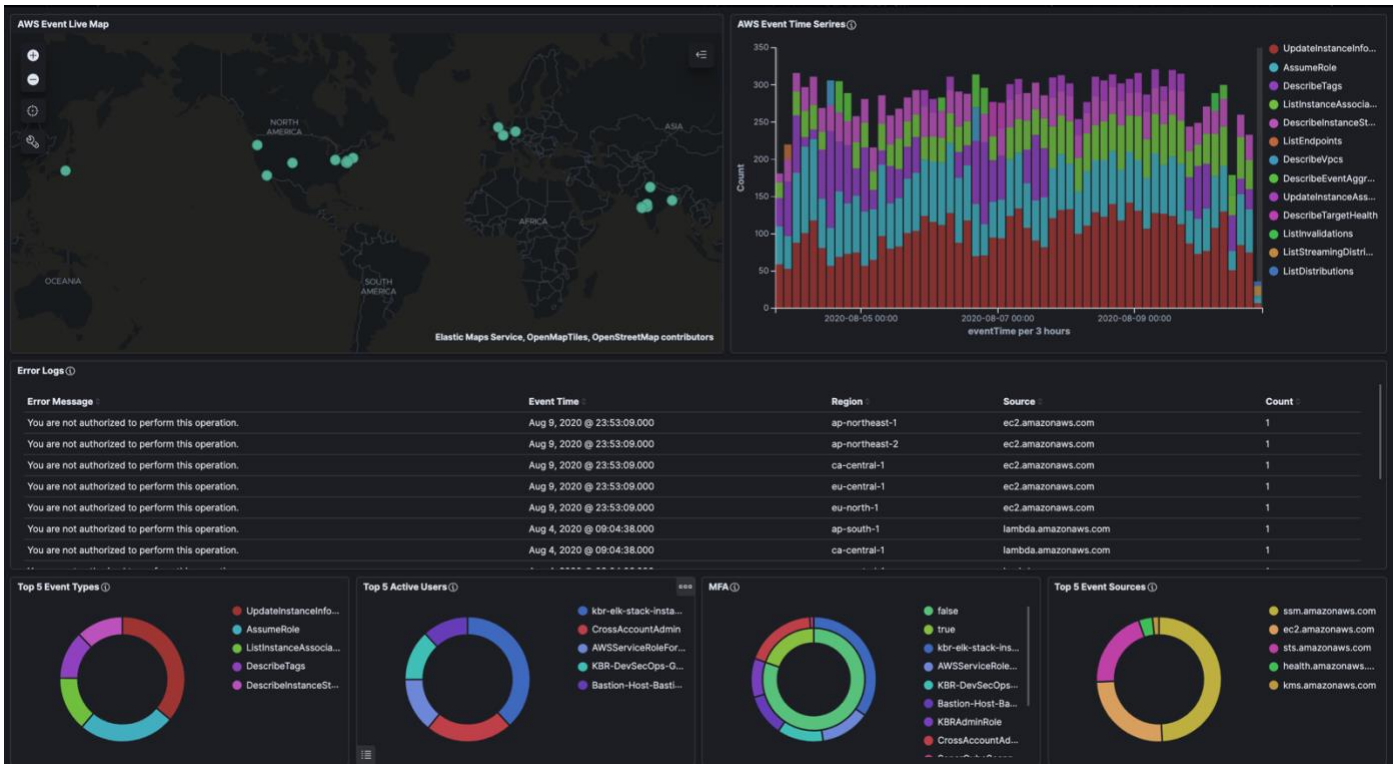


Figure 1: Amazon CloudTrail Dashboard:

Visual explanation from left to right, top to bottom – A live map of all event source locations, a time series of all events occurring with environment color-divided by event type, an error table displaying errors such as permission denials, pie chart of top five most frequent even types, top five most active users accounts, mfa authentication, top five event sources.

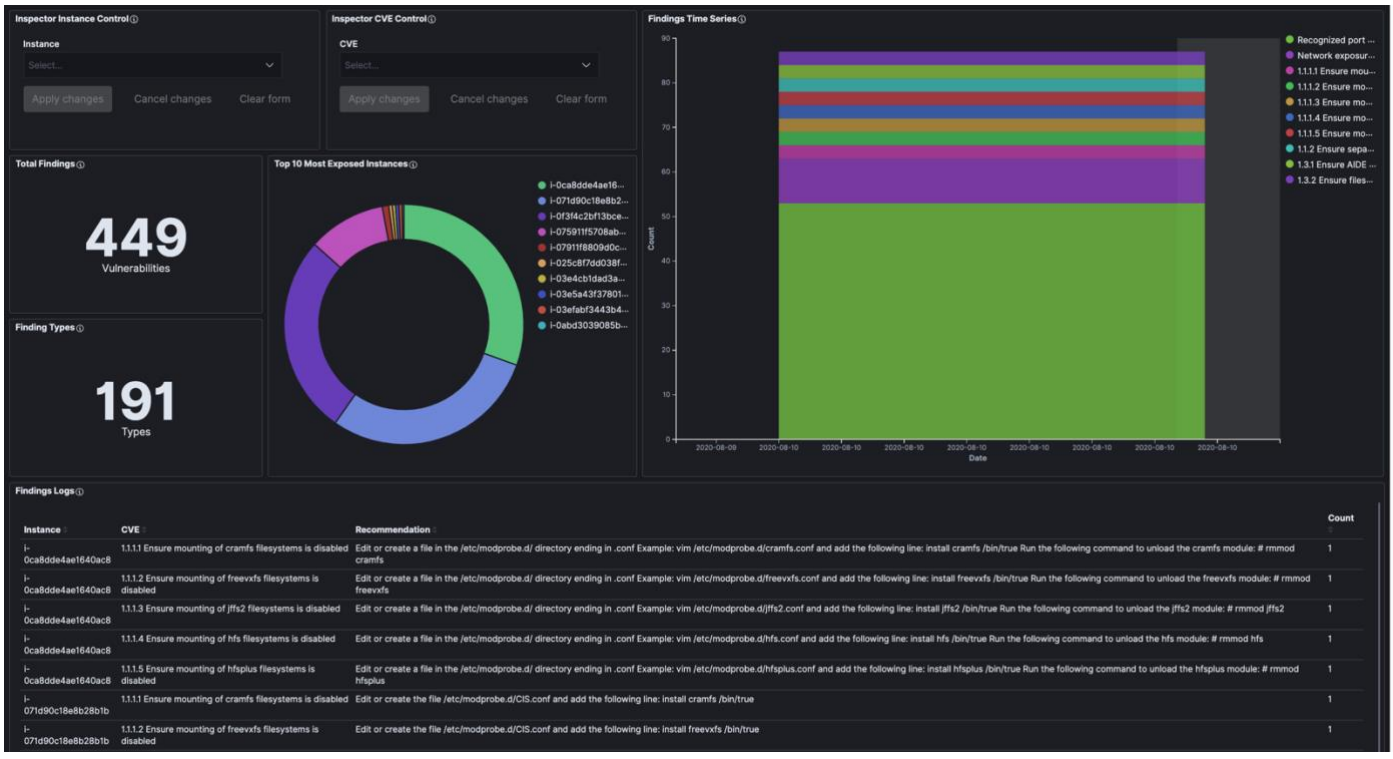


Figure 2: Amazon Inspector Dashboard:

Visual explanation from left to right, top to bottom – data aggregation controls for instance id and CVE type, time series of top ten most frequent CVEs found, total number of CVEs, total number of CVE types, top 10 most vulnerable instances, log table of discovered CVEs and recommendations to fix.