Improving Resilience Through Service Mesh Metrics

DevSecOps Days Pittsburgh 2021
David Shepard
djshepard@sei.cmu.edu

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA  15213
This talk will cover an introduction to…

- Resilience
- Metrics
- Service Mesh
- Kubernetes automation
- Prometheus
- Basic scripting concepts

We have a lot of territory to cover.
Resilience

“an ability to recover from or adjust easily to misfortune or change” –Merriam-Webster

- Resilience is also about survival
- Recovery is variable
- Adjustment is variable
- “Easily” is the measure of how resilient something is

Resilience is not easy.

warhistoryonline.com– B-17 bombers that miraculously made it home
Metrics

“a standard of measurement” –Merriam-Webster

Metrics are how we measure progress.

Things we can measure in DevSecOps: Disk IOPS, Network throughput, IO Latency, CPU Utilization, Memory Utilization, requests per second, response latency, authorizations/failed auths/s, bad requests/s, uptime, MTTR, commits per user, deploys per day, bugs discovered/fixed, security vulns discovered/fixed, protocol overhead, and an unlimited number of additional derivative metrics.

We can measure anything, but what should you be measuring and why and for whom? What metrics are useful for measuring and improving resilience?
Service Mesh

“... makes running services easier and safer by giving you runtime debugging, observability, reliability, and security…” –LinkerD

• The service mesh exists at the interface between co-operating services, providing all of the above properties to your existing applications. For us, they provide a lot useful data. (for creating metrics)

• Technology enablers:
  • Kubernetes’ “pod” concept
  • Transparent, inject-able proxies that modify communications pathways at the edge of the pod

Service mesh proxies perform useful transformations on your data, including encryption, service instance routing, HTTP trace logging, and many others.
Service Mesh with Kubernetes

- Install your application services as normal
- Install and configure your mesh
- “Inject” the mesh into your application
- Gain insights into your operations
Where does the data come from?

- For the purposes of this discussion, the data comes from the individual services and service mesh proxies, and is reported to a central logging service, called Prometheus.
- Novel metrics can be created from data sources that are not normally collected by the mesh.
  - Application Service logs
  - Security/Access logs
  - Git Commit logs
  - Many others

The data comes from wherever it needs to come from, in order to generate useful metrics.
Plenty of data – How to use it?

You need useful user stories.

“As a security professional, I need to know how many failed authorizations we are seeing each day, so that I know when we are facing a specific kind of threat actor.”

“As a developer, I need to know what vulnerabilities exist in the dependencies my code relies on, so that we deliver safe software.”

“As an automation engineer, I need to know when the system is experiencing abnormal loads, so I can decide whether or not to scale the infrastructure.”

Anchore’s Grype tool, reporting known package vulnerabilities in a running container instance.
Automation with Kubernetes

Containers ⊆ Pods ⊆ Services ⊆ Deployments ⊆ Charts ⊆ Applications

• Kubernetes is known for adding the concepts of Pods, Services, and Deployments on top of Containers
  • Containers are the building blocks of the Application
  • Pods are the smallest deployable unit in Kubernetes (unit of organization of containers)
  • Services abstract features of your containers, such as the Service IP or endpoint
  • Deployments allow abstract reasoning about the runtime state of Services for the purposes of scaling
  • Helm Charts (and related technologies) allow you to reason about your applications, as if they were a set of packages that work together in a specific configuration
  • Applications are made up of potentially numerous Charts, configured to provide a full user experience or capability

Jobs are the Kubernetes janitor
The basic unit of cluster work is the Job

When the cluster needs to run some meta-task that isn't part of the application, it deploys a Job.

As containers are the building blocks of the Application/Cluster, Jobs are built on top of Containers

Jobs can be built for any purpose. We will use them for collecting and reporting metrics.

```
apiVersion: batch/v1
kind: Job
metadata:
  name: kube-hunter
spec:
  template:
    spec:
      containers:
      - name: kube-hunter
        image: aquasec/kube-hunter
        command: ["kube-hunter"]
        args: [--pod]
        restartPolicy: Never
        backoffLimit: 4
```

You had one job.

This Job produced the Kube-Hunter logs from slide 7.
A simple cron job illustrates the point nicely. We can use the same approach to collect and report metrics.
Creating custom automations

- A popular data source for basic metrics reporting for cluster applications today is Prometheus. (Next slide)

- Prometheus provides a number of clients for connecting to the service, reporting and querying data.

- Clients are available for various languages and can be used to create various forms of metrics and automation jobs.

Creating a metric to count current “High” CVEs.

JSON Data is extremely common. Get used to working with it.
ConfigMaps, Custom Containers, and Scripts

- ConfigMaps are a useful place to store simple scripts that can be executed from an off-the-shelf container.
- Custom containers provide ultimate flexibility for running your jobs.
  - Implement a Prometheus client metric in the language of your choice
  - Place it in a container or a ConfigMap
  - Create a job to run it
  - Create a Cron job to schedule it
  - Report results and/or push log files to your collection point.

A sample script, a starting point for developing a custom metric.

```python
from prometheus_client import start_http_server, Summary
import random
import time

# Create a metric to track time spent and requests made.
REQUEST_TIME = Summary('request_processing_seconds', 'Time spent processing request')

# Decorate function with metric.
@REQUEST_TIME.time()
def process_request(t):
    """A dummy function that takes some time."""
    time.sleep(t)

if __name__ == '__main__':
    # Start up the server to expose the metrics.
    start_http_server(8000)
    # Generate some requests.
    while True:
        process_request(random.random())
```

---

© 2021 Carnegie Mellon University

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution.
Displaying Metrics

• User engagement is crucial in developing a visual form for your data that gets the users the information they need, in a format that has meaning and facilitates decisions.

• Many tools to choose from for displaying information.

• The “dashboard” is a common tool.

You could visualize your metrics like this, but does it convey the necessary message to your user?
Never lose sight of the target for the metrics and the automations you create.

The end