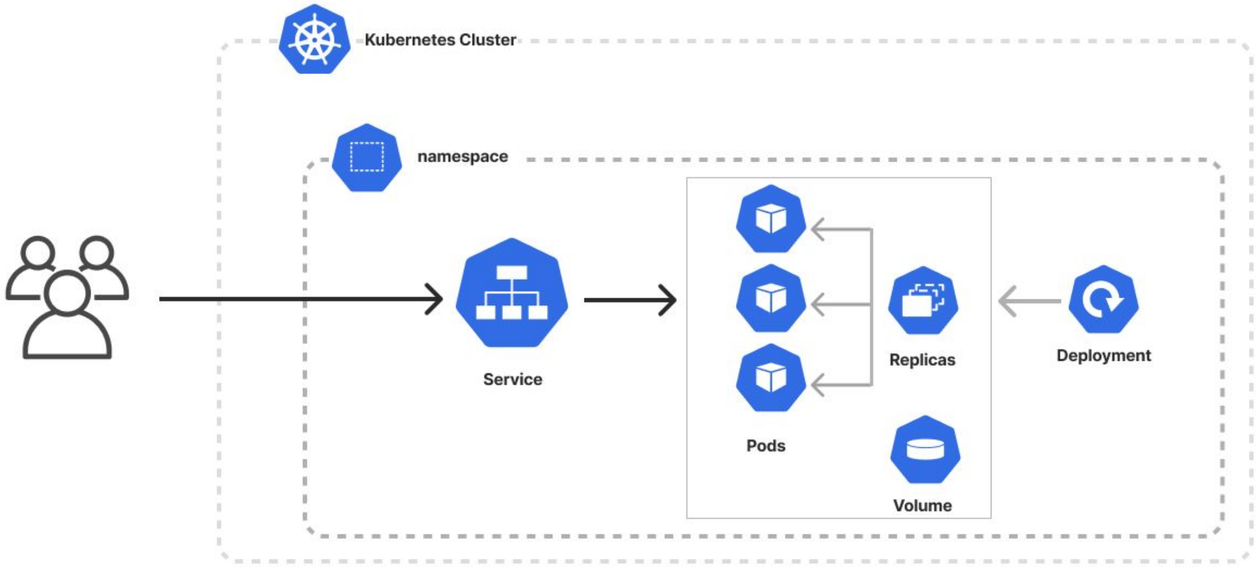
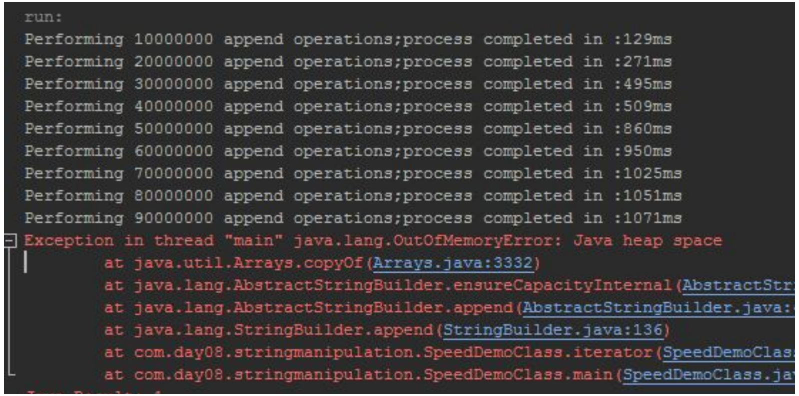
CLOUD NATIVE
COMPUTING FOUNDATION

Troubleshooting Mastery: DevOps Solutions for Java OOM Challenges

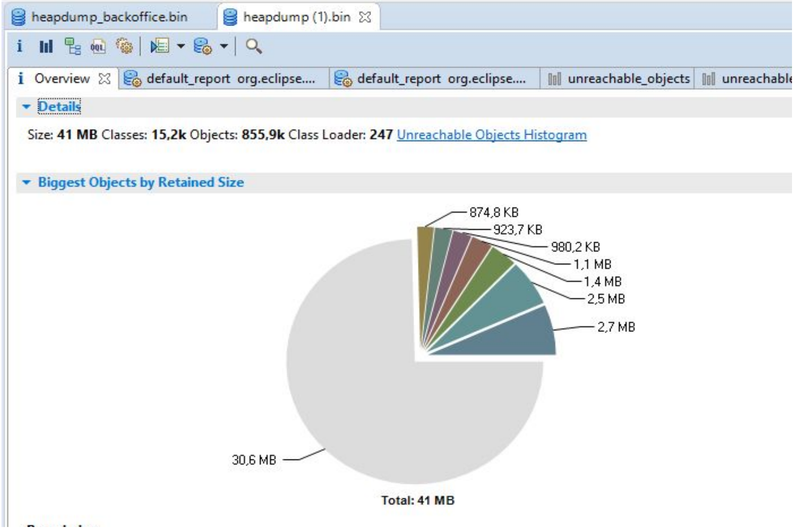
Daksh Jain

Setting the context...





```
run:
Performing 10000000 append operations;process completed in :129ms
Performing 20000000 append operations;process completed in :271ms
Performing 30000000 append operations;process completed in :495ms
Performing 40000000 append operations;process completed in :509ms
Performing 50000000 append operations;process completed in :860ms
Performing 60000000 append operations;process completed in :950ms
Performing 70000000 append operations;process completed in :1025ms
Performing 80000000 append operations;process completed in :1051ms
Performing 90000000 append operations;process completed in :1071ms
Exception in thread "main" java.lang.OutOfMemoryError: Java heap space
        at java.util.Arrays.copyOf(Arrays.java:3332)
        at java.lang.AbstractStringBuilder.ensureCapacityInternal(AbstractStr
        at java.lang.AbstractStringBuilder.append(AbstractStringBuilder.java:
        at java.lang.StringBuilder.append(StringBuilder.java:136)
        at com.day08.stringmanipulation.SpeedDemoClass.iterator(SpeedDemoClas
        at com.day08.stringmanipulation.SpeedDemoClass.main(SpeedDemoClass.ja
```



Details

Size: **41 MB** Classes: **15,2k** Objects: **855,9k** Class Loader: **247** Unreachable Objects Histogram

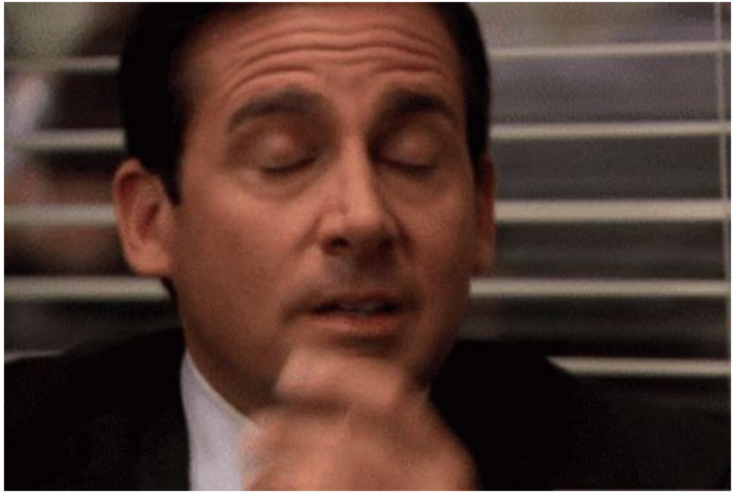
Biggest Objects by Retained Size

Problems

#1 Cost

- Enterprise tools offer robust monitoring and debugging features but come with **high costs**.
- Startups and larger companies prioritize **cost optimization**, making these solutions less attractive:
  - **Budget Constraints**: Startups and smaller companies often have **limited budgets** and prefer to allocate resources efficiently.
  - **In-House Expertise**: Companies with strong tech teams want to develop **custom solutions tailored to their specific needs**, reducing reliance on expensive third-party tools.

Problems



#2 Dependency

Even if Open-source solutions are setup, it often require DevOps oncall to provide heap dump files to developers **manually**.

This process is operation-intensive, leading to delays and inefficiencies.

Our actions should be process dependant not person dependant 🚀

Before jumping in…

After today's discussion what I want is, if a Dev team receives a pager for production apps and it has OOM error then -

- Critical files like heap dumps and thread dumps should be **auto created**.
- These files must be **available** at the time of error to **reduce MTTR**.
- Devs have **access** to these files while debugging.

Final actionables

Collect & provide heap dump from apps running in K8S pods.

↓

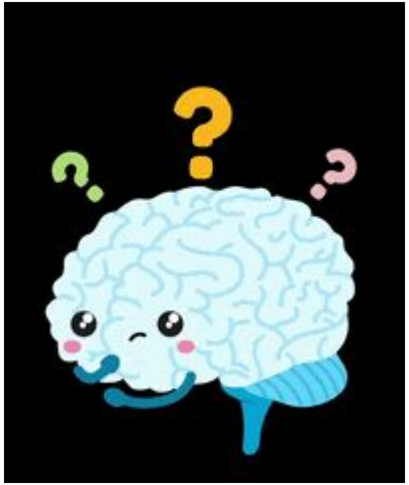
Find a solution to automate heap dump creation, collection & delivery to respective team 24x7.

FAQs

What is heap dump ?
A heap dump is a snapshot of all the objects in the Java Virtual Machine (JVM) heap at a certain point in time. The JVM software allocates memory for objects from the heap for all class instances and arrays.

What is thread dump ?
A thread dump is a snapshot of the state of all the threads of a Java process. The state of each thread is presented with a stack trace, showing the content of a thread's stack.

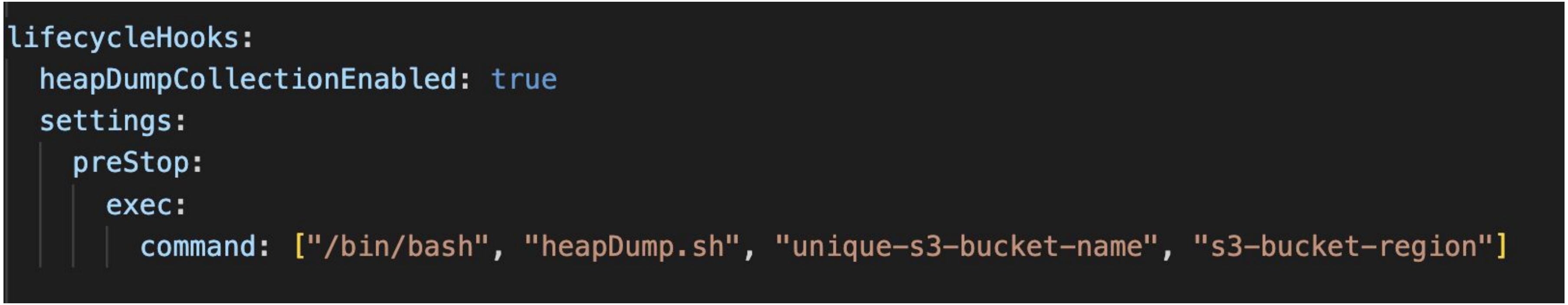
OOM Status code in K8S ?
The Kubernetes OOMKilled (Exit Code **137**) is a signal sent by the Linux Kernel to terminate a process due to an Out Of Memory (OOM) condition. This event is usually an indication that a container in a pod has exceeded its memory limit and the system cannot allocate additional memory.



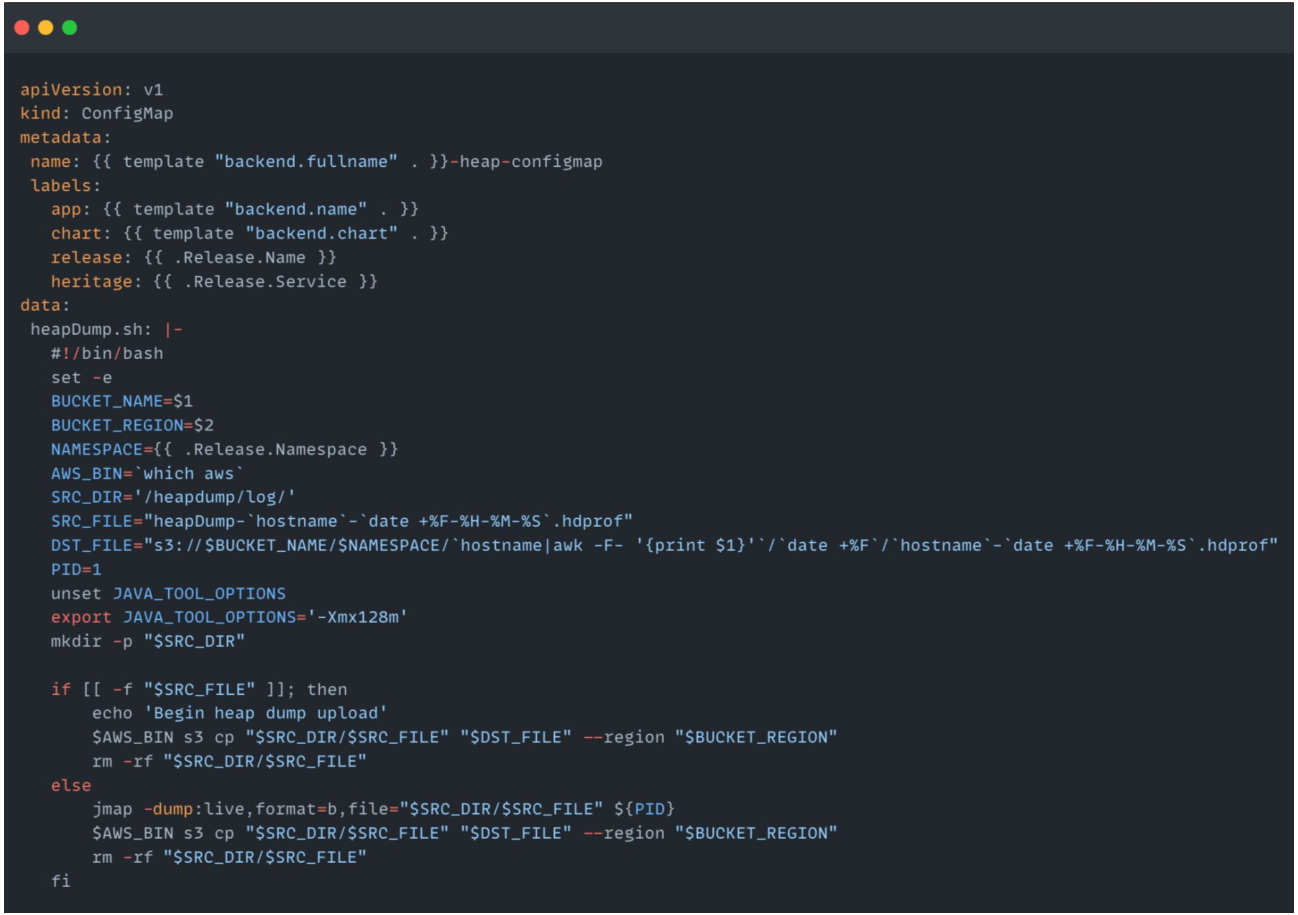
So what are the possible ways we can help Devs perform seamless debugging?

Solution #1

Use Lifecycle hook with pre stop setting in the deployment manifest.



```
lifecycleHooks:
  heapDumpCollectionEnabled: true
  settings:
    preStop:
      exec:
        command: ["/bin/bash", "heapDump.sh", "unique-s3-bucket-name", "s3-bucket-region"]
```



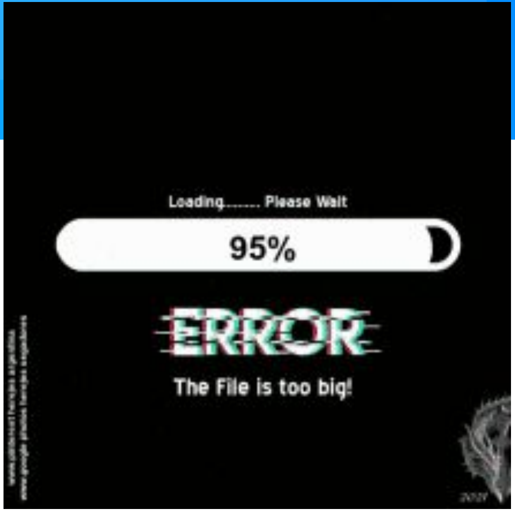
```
apiVersion: v1
kind: ConfigMap
metadata:
 name: {{ template "backend.fullname" . }}-heap-configmap
 labels:
   app: {{ template "backend.name" . }}
   chart: {{ template "backend.chart" . }}
   release: {{ .Release.Name }}
   heritage: {{ .Release.Service }}
data:
 heapDump.sh: |-
   #!/bin/bash
   set -e
   BUCKET_NAME=$1
   BUCKET_REGION=$2
   NAMESPACE={{ .Release.Namespace }}
   AWS_BIN=`which aws`
   SRC_DIR='/heapdump/log/'
   SRC_FILE="heapDump-`hostname`-`date +%F-%H-%M-%S`.hdprof"
   DST_FILE="s3://$BUCKET_NAME/$NAMESPACE/`hostname|awk -F- '{print $1}'`/`date +%F`/`hostname`-`date +%F-%H-%M-%S`.hdprof"
   PID=1
   unset JAVA_TOOL_OPTIONS
   export JAVA_TOOL_OPTIONS='-Xmx128m'
   mkdir -p "$SRC_DIR"

   if [[ -f "$SRC_FILE" ]]; then
       echo 'Begin heap dump upload'
       $AWS_BIN s3 cp "$SRC_DIR/$SRC_FILE" "$DST_FILE" --region "$BUCKET_REGION"
       rm -rf "$SRC_DIR/$SRC_FILE"
   else
       jmap -dump:live,format=b,file="$SRC_DIR/$SRC_FILE" ${PID}
       $AWS_BIN s3 cp "$SRC_DIR/$SRC_FILE" "$DST_FILE" --region "$BUCKET_REGION"
       rm -rf "$SRC_DIR/$SRC_FILE"
   fi
```

- Input
  - Bucket name
  - Region

- Steps
  - Run **jmap** cmd to collect heap dump
  - Run **aws s3 cp** to copy file to S3 bucket.

Issue in Solution #1



- “Helpful” heap dumps are usually **GBs** in size. 📈
- To transfer it to a S3 bucket we run **aws s3 cp**.
- Now the container will be killed if -

```
terminationGracePeriodSeconds < hook run time + container stop time
```

- In this case the hook run time is a **variable** + it is a heavy command to execute.
- So we are not able to determine the **exact value** for grace period.
- Eventually we will **lose** the heap dump. 👎

Solution #2

Run the same heap dump script in **JAVA_TOOL_OPTIONS**.

This helps us to remove the dependency on K8S and is **sort of a shift left approach** to the problem itself as we make the heap dump collection closer to the application.

```
JAVA_TOOL_OPTIONS: '-XX:+HeapDumpOnOutOfMemoryError
-XX:HeapDumpPath=/data/backendHeapDump.hdprof
-XX:OnOutOfMemoryError="mv /data/backendHeapDump.hdprof
/data/`hostname`-`TZ=Asia/Kolkata date +%F-%H-%M-%S`.hdprof ;
/usr/local/bin/aws s3 cp  /data/`hostname`-`TZ=Asia/Kolkata date +%F-%H-%M-%S`.hdprof
unique-bucket-url/path/`TZ=Asia/Kolkata date +%F`/`hostname`-`TZ=Asia/Kolkata date
+%F-%H-%M-%S`.hdprof --region s3-bucket-region"'
```

Issue in Solution #2

But again same issue that if the container is killed then we lose the heap dump.

Solution #3

Use **persistent storage** instead of ephemeral storage for heap dump.

Implementation Pointers

❌ Don’t use EBS.

Use EFS, to have support of **multi mount**.

In EFS use a **lifecycle rule** to move data to S3.

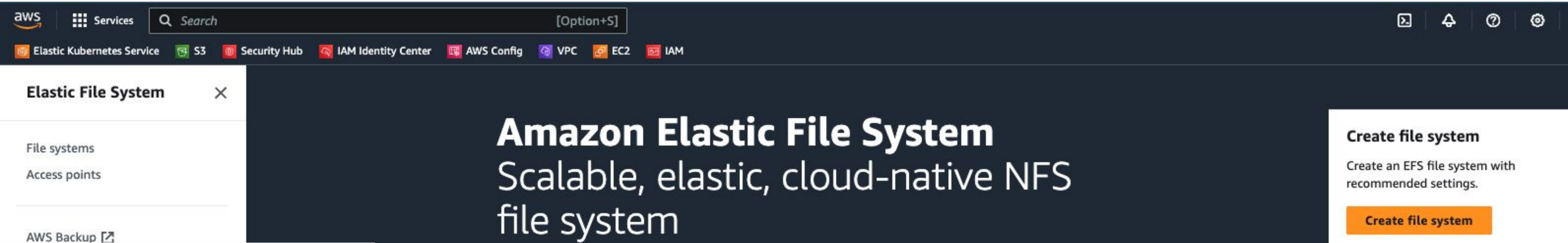
This removes all kinds of dependency –

✅ App **collects** heap dump on OOM

✅ Heap dump is **persistent** in EFS

✅ Move to S3 to reduce EFS cost + increased **accessibility**

✅ Devs can debug **independently** + **reduce MTTR**



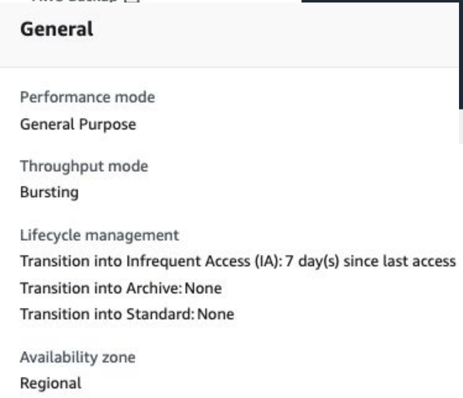
Amazon Elastic File System

Scalable, elastic, cloud-native NFS file system

Amazon Elastic File System (Amazon EFS) provides a simple, scalable, elastic file system for general purpose workloads for use with AWS Cloud services and on-premises resources.

Create file system

Create an EFS file system with recommended settings.



General

Performance mode
General Purpose

Throughput mode
Bursting

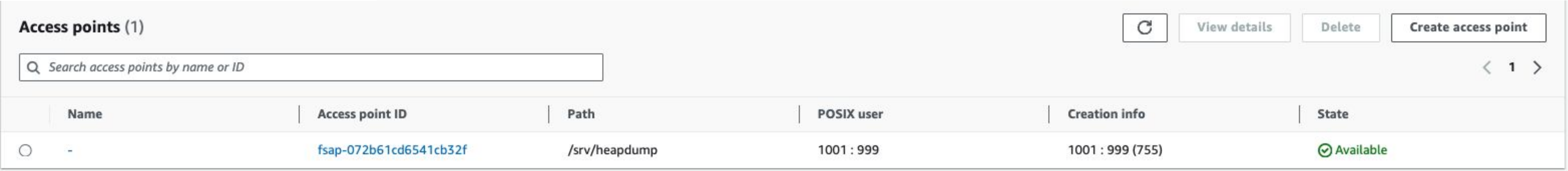
Lifecycle management
Transition into Infrequent Access (IA): 7 day(s) since last access
Transition into Archive: None
Transition into Standard: None

Availability zone
Regional

https://docs.aws.amazon.com/efs/latest/ug/getting-started.html

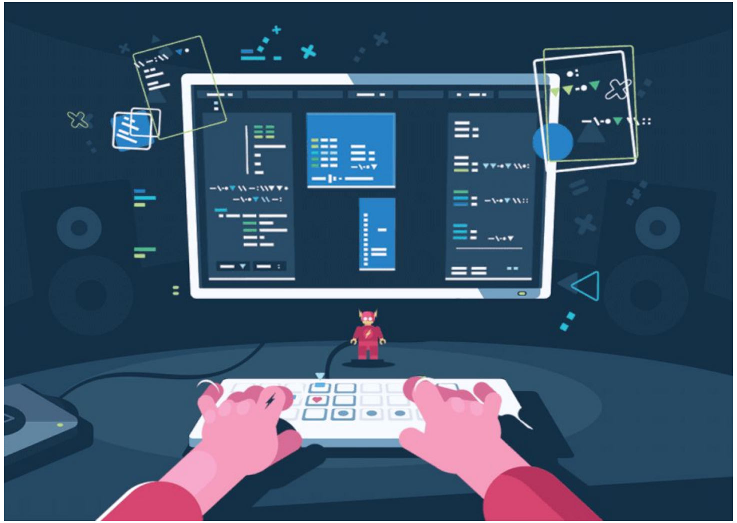
https://aws.amazon.com/efs/pricing/

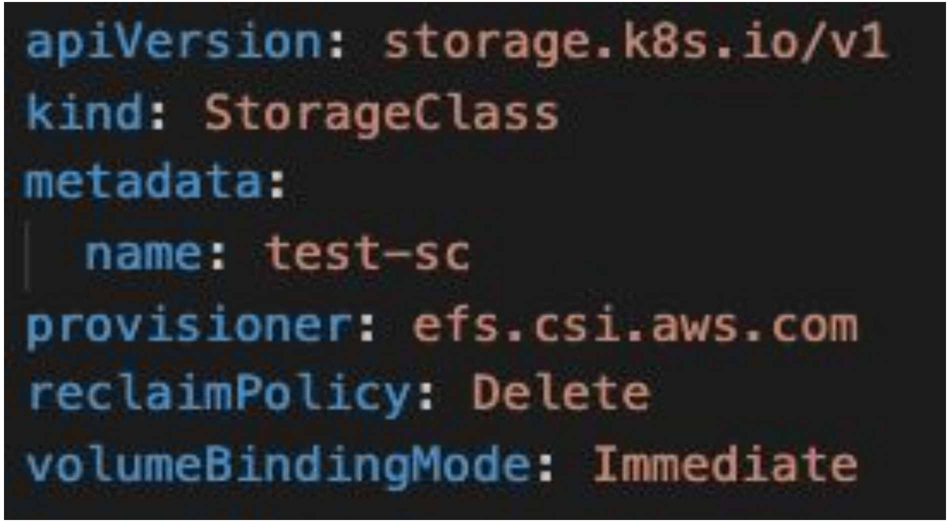
https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/efs_file_system



Access points (1)

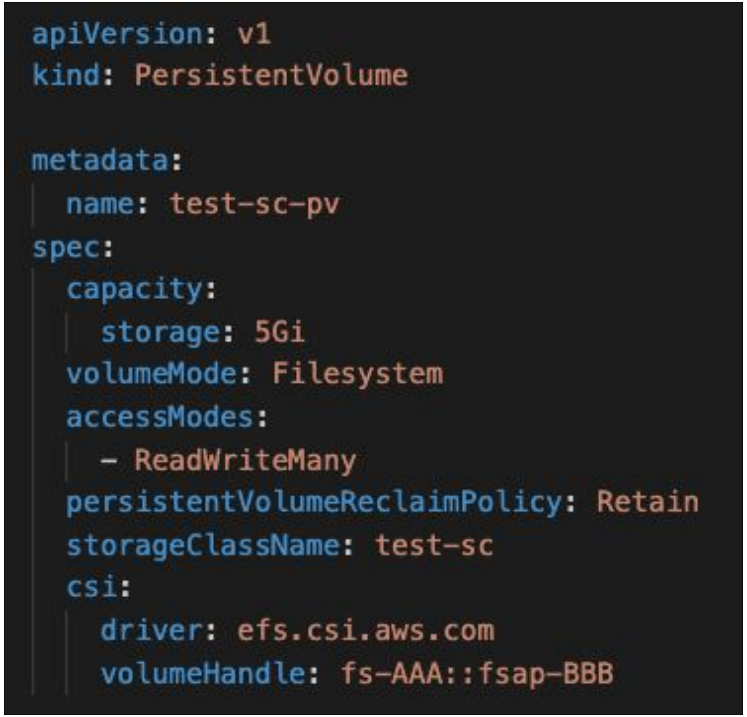
	Name	Access point ID	Path	POSIX user	Creation info	State
○	-	fsap-072b61cd6541cb32f	/srv/heapdump	1001 : 999	1001 : 999 (755)	Available





```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: test-sc
provisioner: efs.csi.aws.com
reclaimPolicy: Delete
volumeBindingMode: Immediate
```

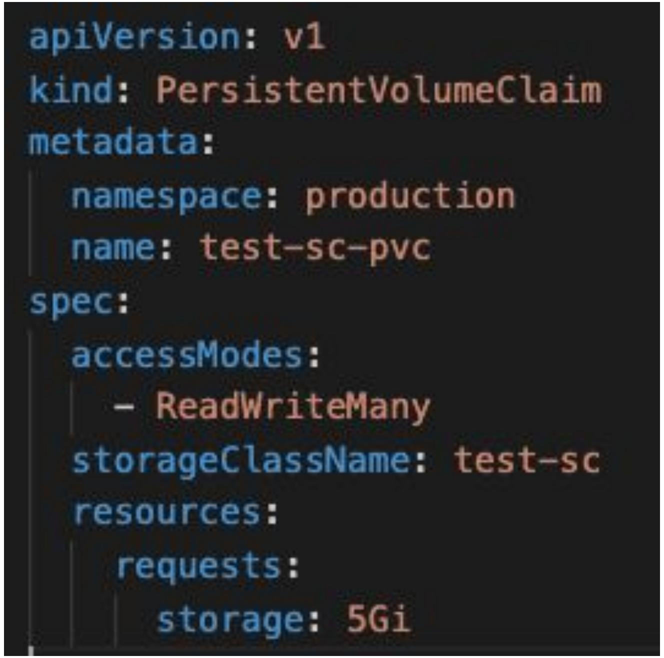
pv



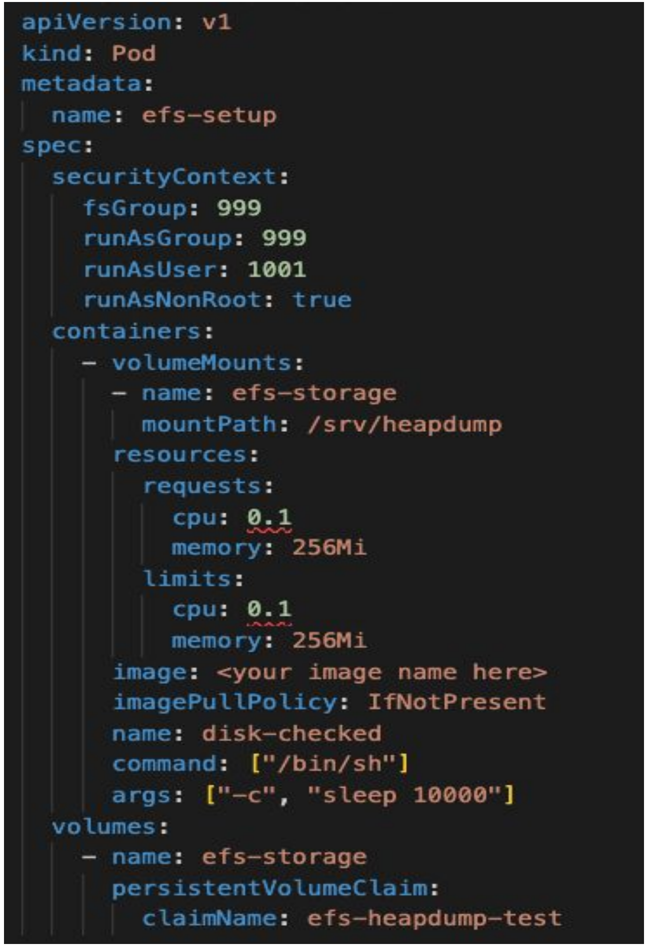
```yaml
apiVersion: v1
kind: PersistentVolume

metadata:
  name: test-sc-pv
spec:
  capacity:
    storage: 5Gi
  volumeMode: Filesystem
  accessModes:
    - ReadWriteMany
  persistentVolumeReclaimPolicy: Retain
  storageClassName: test-sc
  csi:
    driver: efs.csi.aws.com
    volumeHandle: fs-AAA::fsap-BBB
```

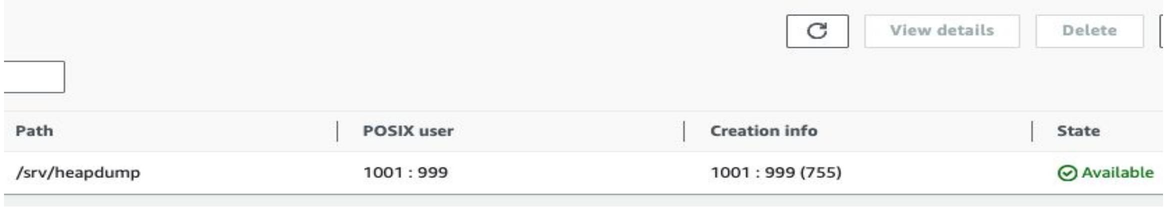
pvc



```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  namespace: production
  name: test-sc-pvc
spec:
  accessModes:
    - ReadWriteMany
  storageClassName: test-sc
  resources:
    requests:
      storage: 5Gi
```

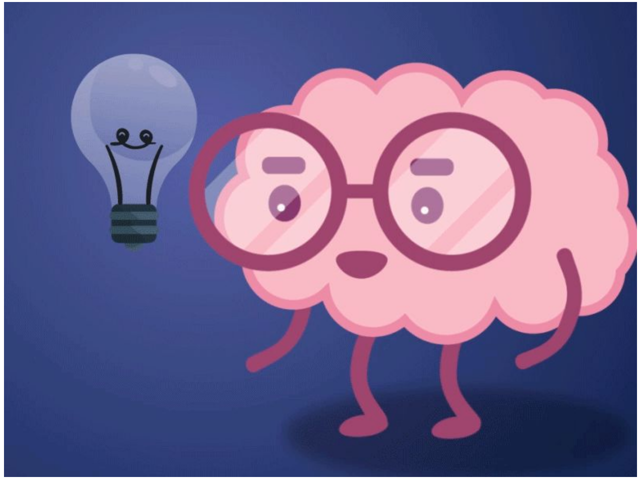


```yaml
apiVersion: v1
kind: Pod
metadata:
  name: efs-setup
spec:
  securityContext:
    fsGroup: 999
    runAsGroup: 999
    runAsUser: 1001
    runAsNonRoot: true
  containers:
    - volumeMounts:
      - name: efs-storage
        mountPath: /srv/heapdump
      resources:
        requests:
          cpu: 0.1
          memory: 256Mi
        limits:
          cpu: 0.1
          memory: 256Mi
      image: <your image name here>
      imagePullPolicy: IfNotPresent
      name: disk-checked
      command: ["/bin/sh"]
      args: ["-c", "sleep 10000"]
  volumes:
    - name: efs-storage
      persistentVolumeClaim:
        claimName: efs-heapdump-test
```



Path	POSIX user	Creation info	State
/srv/heapdump	1001 : 999	1001 : 999 (755)	✓ Available

Learnings from Solution #3



We will use a path in our pod to mount the EFS - **/srv/data**. This path is first created as an access point in EFS.

Amazon EFS access points are **application specific entry points** into an EFS file system that make it easier to manage application access to shared datasets.

Once it is created by a particular user in AWS, this is a service level limitation that you **cannot** mount the **same path** by a **different user** in the **same EFS**.