Centralised Core Dump Processing in Kubernetes Applications in GCP

In Container World, managing centralised information like logs, core-dumps can be a challenge. Though various tools are available for managing Logs like Logstash, FluentD etc.

We don’t really have an end solution for managing Core-Dumps from our application in Container World.

In Kubernetes ecosystem, every node in a cluster can launch one or many pods. Each Pod can launch as many as one or many container applications.

There will be many circumstances where the application might fail like memory dump issue.

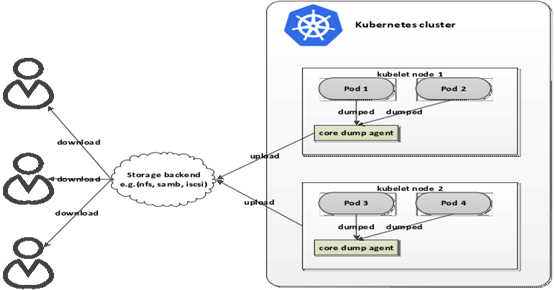
Real Definition of Core Dump is..

“*Core dumps are generated when the process receives certain signals, such as SIGSEGV, which the kernels sends it when it accesses memory outside its address space. Typically, that happens because of errors in how pointers are used. That means there's a bug in the program. The core dump is useful for finding the bug*.”

Hence, by the definition itself, it can really be made out, how important is to understand and debug the issues of Core Dump to make a robust application.

In this blog, I will be describing how we can manage core-dumps from a Kubernetes Cluster deployed in GCP.

**Managing Core Dump in Kubernetes Cluster in GCP**



**Phase 1 Approach**:

The issue with managing the core-dumps in each pod, in case the Pod/Container/Node restart... then we will be losing all the data which was generated in core-dump folder. These are important data which cannot be compromised. Also, we would like to manage the core-dumps from every node in a centralised manner to understand the behaviour of core-dumps so that we can make our application more robust and error prone.

As per the diagram above, we are proposing to introduce a Core Dump Collector Agent in each Node. This Core Dump collector agent will exist as a Daemon Pod / Sidecar container which will collect the data from the location where the core-dumps are created and upload the same data to an external piece of Storage Device.

Each Application running inside the Pod will write the core-dump’s in a said location in its file structure. This location will be Mounted Persistent Volume within the Kubernetes Cluster. The Core Dump Agent will watch the Mounted Volume Location, so as soon as some files are getting created in Persistent Volume, the same file gets uploaded in the Google Storage which is outside the cluster and hence we are able to get a centralised path where we can get all core dumps across multiple clusters.

We can also mount to an Empty directory as well rather than Persistent Volume. Persistent Volume is taken so that we do not lose any core-dumps in some unforeseen circumstances.

Later, we can create some shell scripts to do housekeeping on the Persistent Volume data as well.

The Storage device can be anything be it AWS Cloud Storage, Google Storage, Local File Storage etc. Here, we would discuss how we can manage by storing and centralising with Google Storage.

**ASSUMPTION**: We are assuming that your container is generating the core dumps in the path “*/var/lib/systemd/coredump*”. This path can vary as per the core dump pattern defined in your container. We can create the core dump location specified in the file “*/proc/sys/kernel/core\_pattern*” by doing something like below.

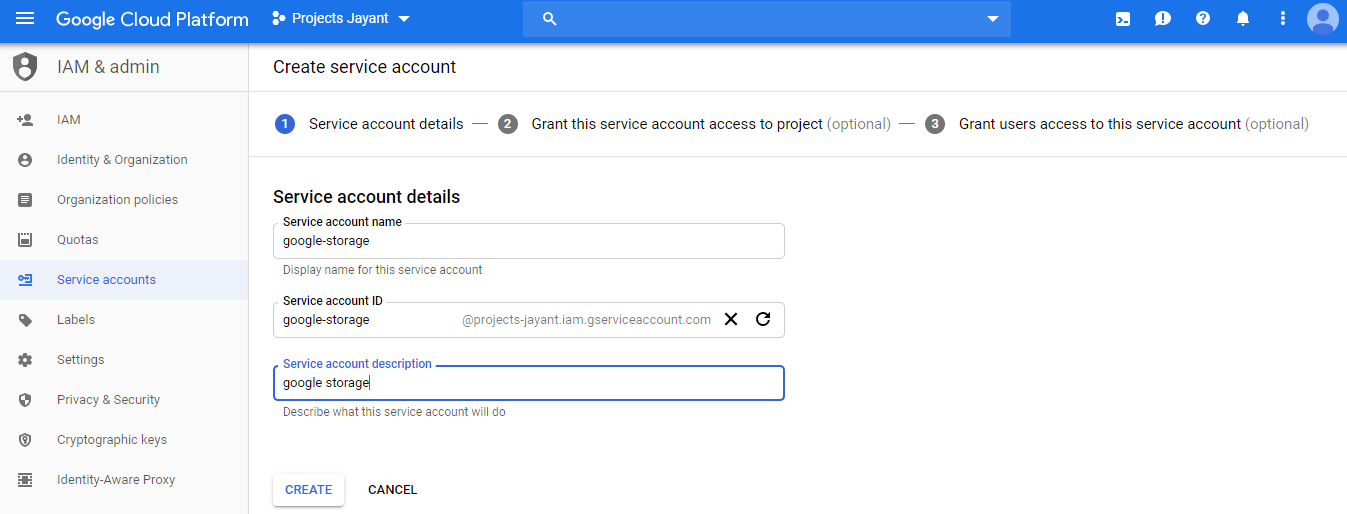
$> mkdir -p /tmp/cores  
$> chmod a+rwx /tmp/cores  
$> echo "/tmp/cores/core.%e.%p.%h.%t" > /proc/sys/kernel/core\_pattern

Here, the core dump will be created in “*/tmp/cores*” directory. In our exercise, we are assuming that core files are located in “*/var/lib/systemd/coredump*”.

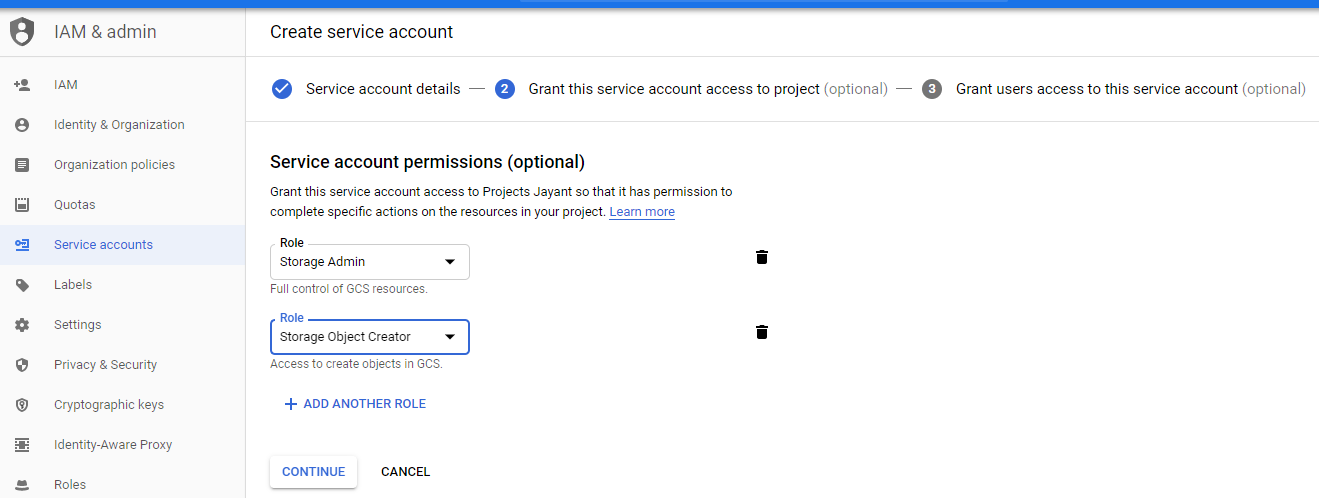
**NOTE: Most of the scripts and code, I had intentionally put as a Screenshot rather than the real code, so that you practise and understand the approach and its steps.**

**STEP 1**. **CREATION OF SERVICE ACCOUNT TO ACCESS GOOGLE STORAGE BUCKET.**

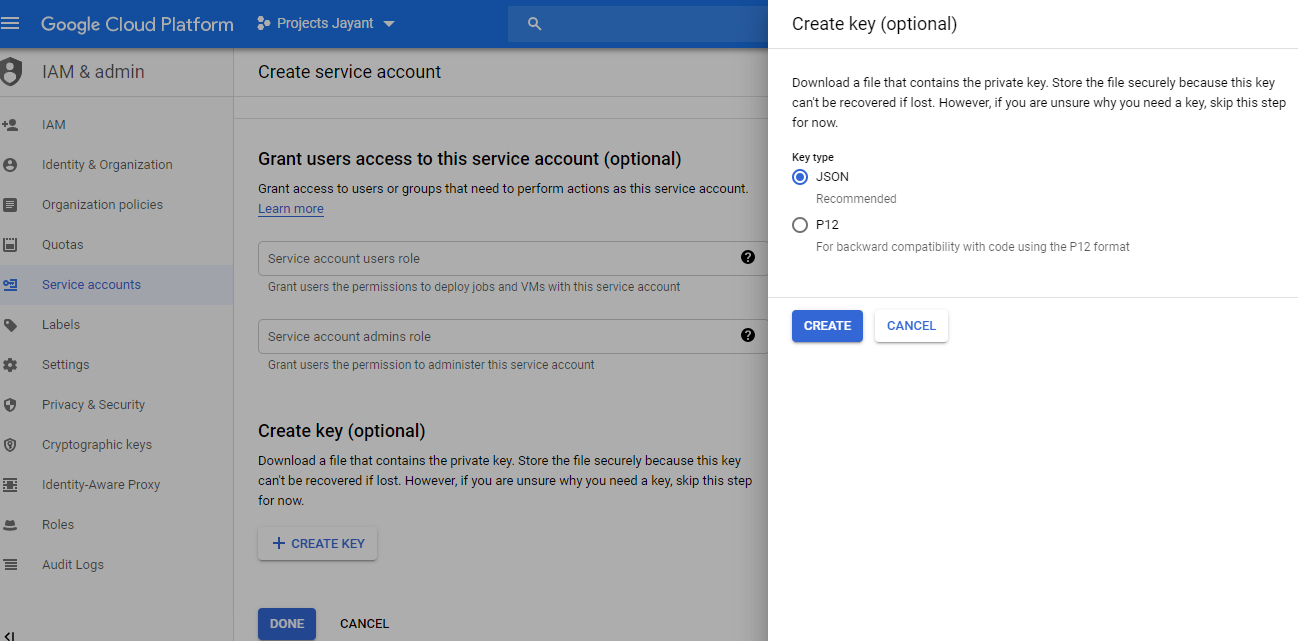
Create a Service Account in GCP console by clicking on IAM & Admin – Service Account – Create Service Account.



Select Service Account Permission Roles, do select the “*Storage Object Creator*” role.



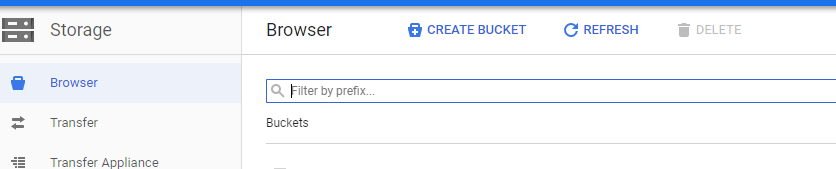
Store the key as a JSON key and save it locally in your system. In our case, we have saved the file “*projects-jayant-82544f23332b.json*” in local system.



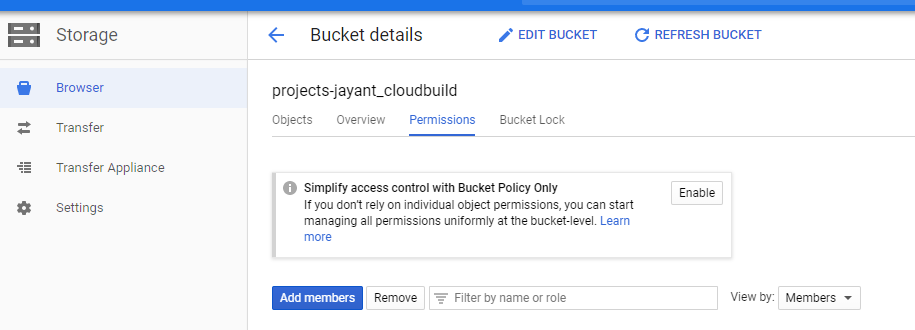
**NOTE**: Service Account can also be created with GCP scripts “*gcloud iam service-accounts create…*”.

**STEP 2**: **CREATE THE GOOGLE STORAGE EXTERNAL BUCKET.**

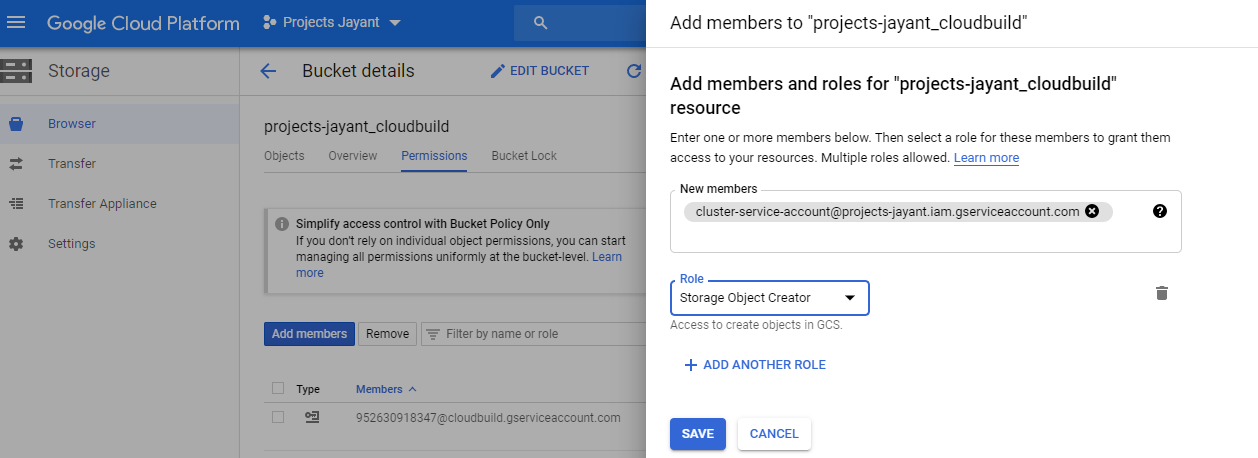
Create the Google Storage where you want to upload the Core-Dumps from multiple Pods across Multiple Nodes in different clusters.



Once the Google Storage Bucket is created. Do assign the Permission by selecting “*Add members*”. Here, you will add the Service Account information which you created above in Step 1.



Also, do assign ”*Storage Object Creator*” role to it.

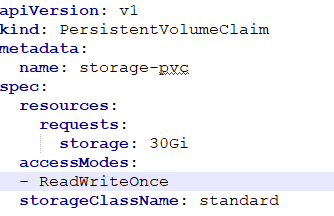


**NOTE**: Creation of Google Storage Bucket can also be done by GCP scripts like below. I went with the GUI based concepts so that it is understood by wider audience.

“*gsutil mb -p [PROJECT\_NAME] -c [STORAGE\_CLASS] -l [BUCKET\_LOCATION] -b on gs://[BUCKET\_NAME]/”*

**STEP 3.** **PERSISTENT VOLUME MOUNT ON CORE-DUMP LOCATION INSIDE APPLICATION CONTAINER.**

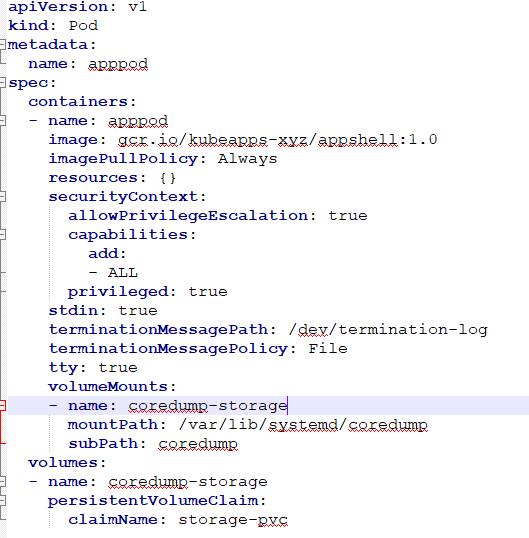
If the Persistent Volume is already created, then ignore the below yaml script. In this case, persistent volume name is “*storage-pvc”.*



Reference the Persistent Volume name in your Application Pod/Deployment yaml file. Mount the file location where core-dump is created to the Persistent Volume.

In the below Application Pod, we are mounting the core-dump path “*/var/lib/systemd/coredump*” and mapping it to a persistent volume “*storage-pvc*” as created above.

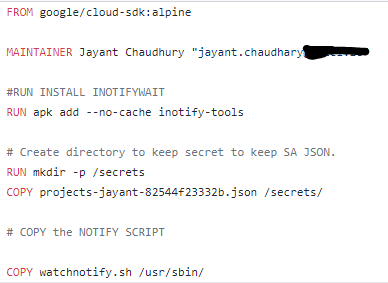
You can have your corresponding image in place of “*gcr.io/kubeapps-xyz/appshell:1.0*” below.



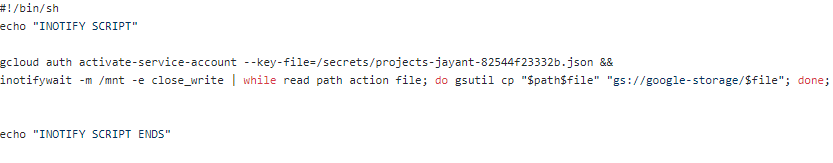
**STEP 4. CORE-DUMP AGENT COLLECTOR DOCKERFILE**.

In order to create a Core-Dump Agent Collector, we will use the Google SDK - Alpine Image and install INOTIFY-TOOLS so that core-dump agent collector can watch on the specified directory. Google SDK is used as we need to use the Google CLI command to trigger the watch notification.

In this Dockerfile, we will copy the Service Account JSON Key created in Step 1. And copy it into Alpine Directory.



If you see above in the screenshot, we also created a “*watchnotify.sh*” script (which will trigger the INOTIFY EVENT whenever there is some file created in core-dump persistent disk location. See the screenshot below) And kept inside the “*/usr/sbin”* directory.



**NOTE: If you want to sync the Storage Bucket with the Core-Dump directory, then replace the “inotify –m …xxxx..gsutil cp..” line with the below hightlighted line.**

**“*inotifywait -m /mnt -e close\_write | while read path action file; do gsutil -m rsync -d -r "$path" "gs://google-storage/"; done;”.***

Later, we can use similar “gsutil” command to copy from Google Storage to Amazon S3 bucket as well. For example:: “gsutil –m rsync –d –r “gs://<your-storage>” “s3://<amazon-bucket>”.

Just Remember, we have given the Google Storage Bucket Name in the “*watchnotify.sh*” which we have created in Step 2, in this case it is “*google-storage*”. And, we have passed the service-account JSON key which we had put in the Alpine directory in the Dockerfile. Here, it is “*secrets/projects-jayant-82xxxx.json*”. In your case, it will be different JSON file.

Now, Build the Docker Image. I have done it from my GCP console.

“*docker build . -t gcr.io/projects-jayant/coredump/coredump0.1:1*”

And, push it in Image Repository. In your case it will vary, I pushed in gcr.io repository from my GCP console.

“*docker push gcr.io/projects-jayant/coredump/coredump0.1:1*”

**STEP 5: EXECUTING CORE-DUMP AGENT AS A DAEMONSET**

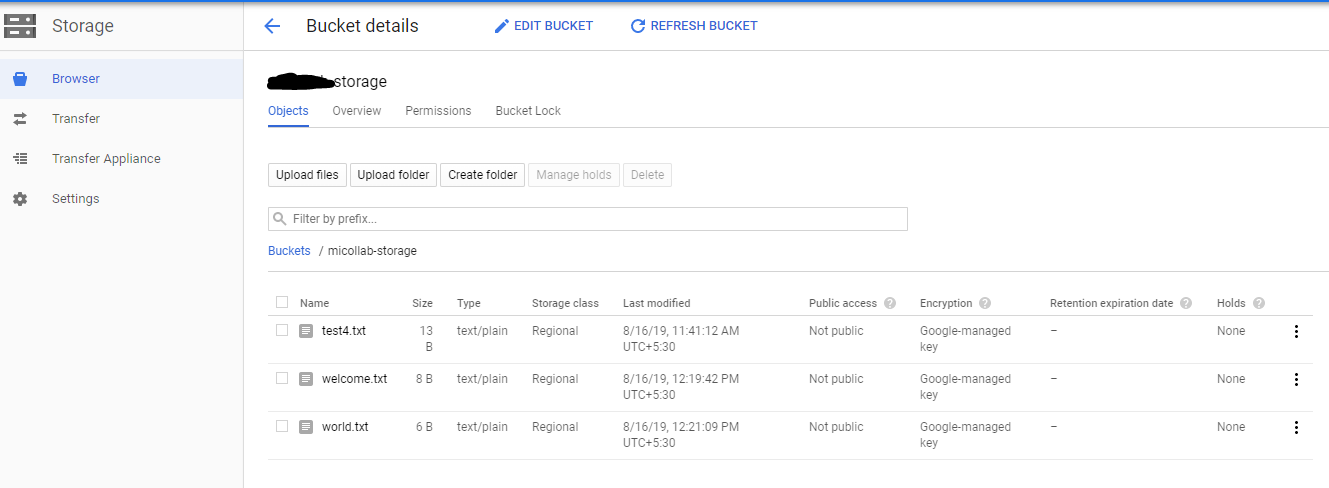
We want to run the Core-Dump Agent Collector Agent as a Sidecar container. We will create it as a Daemon Set so that Core Dump Agent Collector container is up and running as soon as the Cluster comes Alive in K8 cluster. This Daemon Set will call the Core-Dump Agent Image created in Step 4. Also, it will call the “*watchnotify.sh*” script to watch the core-dump file location. The File location (*“/mnt”*) is mapped to the same Persistent Volume created in Step 3.



Now, deploy both the Daemon Set and Application Pod in your Kubernetes Cluster.

VOLLLAA….!!!! In your Google Storage Bucket, you will start seeing the core-dumps file’s getting created.

In my case, I pushed some dummy files in my application container core-dump path *“/var/lib/systemd/coredump*” and it gets automatically created on “*gs://google-storage*”.



**Phase 2 Approach:**

In this Phase, we can plan how to consume the data captured in Google Storage and what benefits we can get from those data.

One Option, we can use Kafka Streaming to capture live data and then use this data in multiple ways, i.e. Reporting, Monitoring, Analytics.



Well… That’s on a another day.. ☺