

# Google Cloud Run...

A mini journey from a hard core kubernetes user to Google Cloud Run  
@javapapo - [javapapo.blogspot.com](http://javapapo.blogspot.com)

# My past ...

- Java/JVM dev background
- Exposure to AWS (loving it , I won't deny it) - less GCloud
- Started using Kubernetes in **production** almost 4 years ago- 1.4.x



# Loving Kubernetes

- I was in love with kubernetes (I still am) because
  - Gave docker a new life and adoption!
  - Indirectly contributed to the wider adoption of what people were calling microservices. You finally had a tool (platform) where you could entertain the idea of splitting code + data into smaller deployables.
  - Felt like the answer to all prayers from developers for
    - Service discovery
    - H. Scaling
    - Ingress Services - LB
    - Power to devs!!!



# Honeymoon period!



- Through this journey and after the initial honeymoon period you had to deal with
  - Kubernetes complexity
  - Upgrades/Tooling and deployment in the cloud (Aws)
  - Challenges on the application / platform architecture. How do we split services, what are their integration patterns? Is it effective?
  - Challenges on services that had to be lifted to kubernetes. Legacy code and system that were never meant to run in the cloud!
  - Somewhere along the way - this serverless thing came in!!!

# Server-less

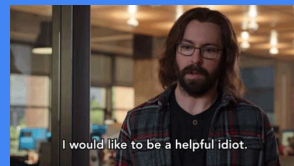
- While drinking the kubernetes kool aid + dealing with problems
- New kids on the block -> Serveless All the things!
- Too busy to play with the new kids
- The kubernetes adoption had a lot of work anyway.
- Taking a step back though
  - Serveless really had some interesting ideas and principles
  - It felt that could be a potential wave of change , like kubernetes was!
  - More developer centric - LOVE IT!



# Cloud /microservices adoption is iterative and takes time.



- You can not pivot with 1 step from deploying a monolith on prem to the cloud thinking that you are cool now and u do microservices
- You start by splitting your code (distributed monoliths)
- You lift and shift these highly coupled payloads to a modern platform (e.g kubernetes)
- Then you slowly experience the pain of distributed monoliths and you start thinking about integration patterns and how your platform can be an enabler
- Then you slowly start to chop your services to even smaller artifacts and make the integration more dynamic!
- Iteratively you fight with the increasing complexity of your platform.
- You are forced to adjust and adapt your architecture. Cascading changes across the board.
- You learn to do DevOps (this is devops)



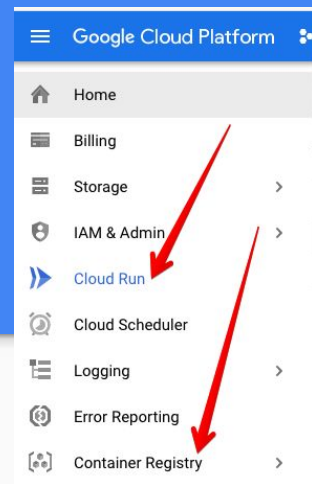
# How all this relate to Google Cloud Run?

- Google cloud run is kind of a hybrid proposal from Google cloud.
- What if we you are not ready to jump into serveless?
- What if maintaining and running full blown kubernetes clusters is too much for you?
- What if you don't have so many services but you still want some of the nice things kubernetes offers under the hood e.g scaling)
- What if you have 1 or 2 containerised apps and you still want to deploy them using a public load balancer?

# How do I start?

- Google cloud account
- Install google cli
- Create a workspace
- Enable the Google Container Registry for private builds - you can only run payloads from gcr Create 2 service accounts (SA) 1 for Google Cloud Run + 1 for GCR
- Docker-ize your application and push it to the registry

```
docker login -u _json_key --password-stdin https://gcr.io < ${GOOGLE_CONTAINER_REGISTRY_SA}
docker build -t ${GCR_IMAGE}:${BUILD_VERSION} .
docker push ${GCR_IMAGE}:${BUILD_VERSION}
echo Pushed IMAGE: ${GCR_IMAGE}:${BUILD_VERSION}
```



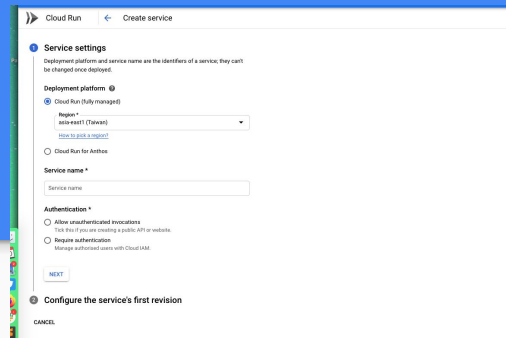


# How do I deploy?

- Easiest way through gcloud cli
- Or you can use the Gcloud web console
- Or you can use terraform :

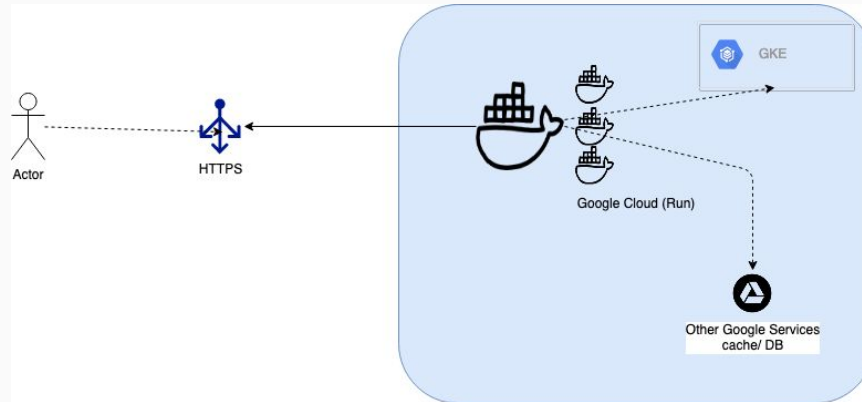
[https://www.terraform.io/docs/providers/google/r/cloud\\_run\\_service.html](https://www.terraform.io/docs/providers/google/r/cloud_run_service.html)

```
deploy:
  stage: deploy
  image: google/cloud-sdk:alpine
  before_script:
    - |
      gcloud --quiet components update
      gcloud auth activate-service-account --key-file ${GOOGLE_CLOUD_RUN_SA} --project {G_CLOUD_PROJECT}
      gcloud run deploy $SERVICE_NAME --image ${GCR_IMAGE}:${BUILD_VERSION} --platform=managed --region=europe-west1
```

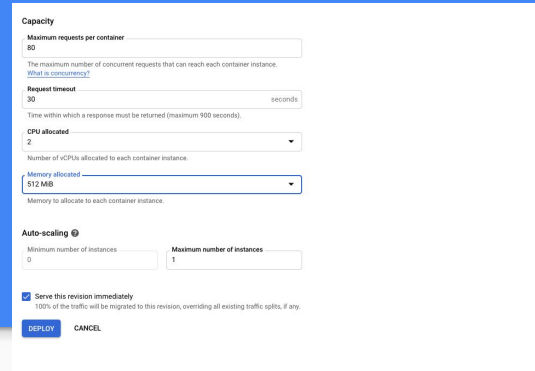


The screenshot shows the 'Create service' wizard in the Google Cloud console. The first step is 'Service settings'. It includes a note that deployment platform and service name are identifiers that cannot be changed after deployment. The 'Deployment platform' is set to 'Cloud Run (fully managed)'. The 'Region' is set to 'europe-west1'. The 'Service name' field is empty. Under 'Authentication', 'Allow unauthenticated invocations' is selected. The 'NEXT' button is visible at the bottom.

# How does it look like?



# Any limitations?



The screenshot shows a deployment configuration form with the following sections:

- Capacity**
  - Maximum requests per container**: Input field with value 80. Subtext: "The maximum number of concurrent requests that can reach each container instance. [What is concurrency?](#)"
  - Request timeout**: Input field with value 30, unit "seconds". Subtext: "Time within which a response must be returned (maximum 900 seconds)." [What is concurrency?](#)
  - CPU allocated**: Input field with value 2, unit "Number of vCPUs allocated to each container instance."
  - Memory allocated**: Input field with value 512 MB, unit "Memory to allocate to each container instance."
- Auto-scaling**
  - Minimum number of instances**: Input field with value 0.
  - Maximum number of instances**: Input field with value 1.
  - ☒ **Serve this revision immediately**  
100% of the traffic will be migrated to this revision, overriding all existing traffic splits, if any.
  - DEPLOY** button and **CANCEL** button.

- You need to be able to expose a port and respond to HTTP calls
  - EXPOSE 8080
- Stateless as much as possible
- Your container will `freeze` when there are no incoming requests!
- Be aware of the runtime limitations - currently you can spin containers with resource limits up to 2 CPUs
- Your good old fat java service maybe won't make it !
- You can now - talk to other `services` that are deployed to a kubernetes cluster on the same workspace/network

# What do I get?



- Auto scaling based on incoming requests and concurrency
- Metrics and logs!
- Public (or private) load balancer + DNS entry with HTTPS for free!
  - URL: <https://javapapo-sample-fadvfa82za-ew.a.run.app>

## Auto-scaling

Minimum number of instances

Maximum number of instances

METRICS	REVISIONS	LOGS	DETAILS	YAML	PERMISSIONS
Logs Showing 100 messages Default Filter					
2020-06-10 14:35:00.286 BST Received ping					
2020-06-10 14:35:00.288 BST GET 200 263 B 9 ms Google-Cloud-Scheduler https://					
2020-06-10 14:40:00.405 BST Received ping					
2020-06-10 14:40:00.408 BST GET 200 265 B 102 ms Google-Cloud-Scheduler https://					
2020-06-10 14:45:00.434 BST Received ping					
2020-06-10 14:45:00.436 BST GET 200 263 B 9 ms Google-Cloud-Scheduler https://					
2020-06-10 14:50:00.479 BST Received ping					

# Why should I try it?



- Dead easy way to spin / service a stateless service with a concrete set of dependencies.
- Scalability
- You will be sitting in between a Knative kubernetes cluster and before a Cloud function.
- Food for thought on how potentially you can start thinking about slimmer services - (if you are interested)

# Resources



- <https://cloud.google.com/run/docs> (official documentation)
- <https://github.com/ahmetb/cloud-run-faq> (very very good- start here!)
- <http://javapapo.blogspot.com/2020/04/the-simplest-gitlabci-pipeline-for.html>
- <https://cloud.google.com/run/pricing#cloudrun-pricing>

# Thanks

