



MasterClass Session

Jan 06, 2024 Patna, India



Ashok Jung Bahadur

Cloud Architect

Google

[@ashokjung](#)

Serverless Foundation

- 
- Architect at Google
 - 13+ years of Industry Experience working with Various Product Organisation
 - Expertise in Application Development & Modernisation
 - Tech Speaker
 - Full Stack Developer
 - Mentor

Follow Me
@[ashokjung](#)



Disclaimer

The content and views presented during the session are author's own and not of any organizations they are associated with.

The Buzzwords



Google
Cloud Run





What is Container?

- Containers are packages of software that contain all of the necessary elements to run in any environment.
- Containers virtualize the operating system and run anywhere, from a private data center to the public cloud or even on a developer's personal laptop.
- Containerization allows our development teams to move fast, deploy software efficiently, and operate at an unprecedented scale.
- From Gmail to YouTube to Search, everything at Google runs in containers running, 2 billion containers/week.
- The Open Container Initiative (OCI), established in June 2015 by Docker and other industry leaders, is promoting common, minimal, open standards and specifications around container technology





Why do we need Container?

- 
- Isolation
 - Portability
 - Scalability
 - Resource Efficiency
 - Version Control
 - Standardisation





Types of Container?

- Stateless Containers: These types of containers do not persist data. These containers are typically used to run stateless applications such as web servers, reverse proxies, and load balancers.
- Stateful Containers: These types of containers persist data and are typically used to run stateful applications such as databases, message queues, and file servers. The data stored inside the container is persistent even if the container is stopped or recreated.
- Ephemeral Containers: These types of containers are used for short-lived tasks, such as running one-off commands, performing CI/CD pipeline tasks, etc. They are typically used for testing and debugging purposes. They are created and destroyed very quickly and are not meant to be long-lived.





What is Cloud Run?

- Cloud Run is a managed compute platform that lets you run containers directly on top of Google's scalable infrastructure.
- You can deploy code written in any programming language on Cloud Run if you can build a container image from it.
- In fact, building container images is optional. If you're using Go, Node.js, Python, Java, .NET Core, or Ruby, you can use the source-based deployment option that builds the container for you, using the best practices for the language you're using.
- Google has built Cloud Run to work well together with other services on Google Cloud, so you can build full-featured applications.
- On Cloud Run, your code can either run continuously as a *service* or as a *job*.





Cloud Run Services

- Unique HTTPS endpoint for every service
- Fast request-based auto scaling
- Built-in traffic management
- Private and public services
- Scale to zero and minimum instances
- Pricing (Request-based,Instance-based)

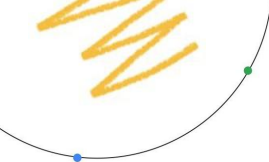




Typical Use Cases

- Microservices /API
- Event Processing
- Web Application
- GRPC
- Service to Service Communication





What would be my cost for Cloud Run?

Billing concept

- Only Pay When your Code is running for
- **CPU** (provisioned vCPU per 100 ms)
- **Memory** (provisioned GiB per 100 ms)
- **Request** (per million request)
- OR no per- request fee if CPU Throttling is disabled.






Advantage

- Deploy in seconds
- Automatic HTTPS, custom domains
- Any language, any library
- Portability
- No cluster management
- HTTP/2, WebSockets, and gRPC



Deployment

Deployment Form

 project1

Cloud Run [← Create service](#)


A service exposes a unique endpoint and automatically scales the underlying infrastructure to handle incoming requests. Service name and region cannot be changed later.

☒ Deploy one revision from an existing container image

[SELECT](#)

[TEST WITH A SAMPLE CONTAINER](#)
Should listen for HTTP requests on \$PORT and not rely on local state. [How to build a container?](#)

☐ Continuously deploy new revisions from a source repository

Region *
us-central1 (Iowa) 

[How to pick a region?](#)

CPU allocation and pricing

☒ CPU is only allocated during request processing
You are charged per request and only when the container instance processes a request.

☐ CPU is always allocated
You are charged for the entire lifecycle of the container instance.

Command line

```
$ gcloud run deploy --source . \
```

API

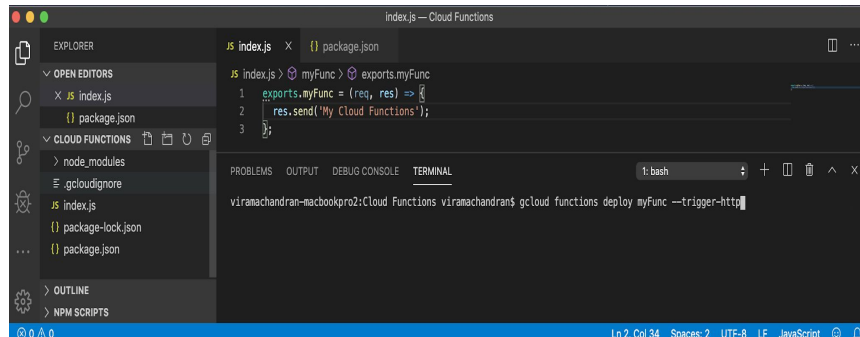


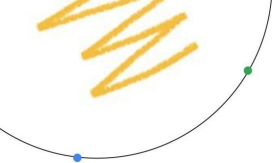
Cloud Run API

Google Enterprise API

Run stateless HTTP containers on a fully managed environment. Cloud Run is a managed compute platform that enables you to run stateless containers that are invocable via HTTP requests. Cloud Run is serverless: it abstracts away all infrastructure management, so you can focus on what matters most — building great applications.

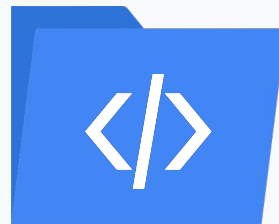
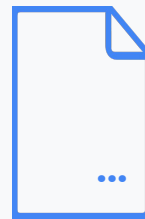
IDE

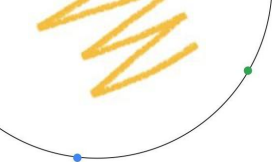




What to deploy

- A container with
- Any language
- Any library
- Any binary
- Ecosystem of base images





Google Cloud Run workflow is a three-step process

1

Write your code

2

Build and package

3

Deploy to Cloud Run

Source code



Container image

Application



Web app

Listen on a port and
accept requests





Container based deployment

- Create a **Dockerfile**, which is not always trivial for more complex applications.
- **Build the container image** (with Cloud Build or others)
- Store your image in **Artifact Registry** & Finally, **deploy to Cloud Run**

Command line

```
$ PROJECT_ID=$(gcloud config get-value project)

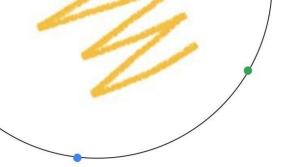
$ SERVICE_NAME=helloworld

$ REGION=us-central1

$ gcloud builds submit \
  --tag $REGION-docker.pkg.dev/    $PROJECT_ID/REPOSITORY/$SERVICE_NAME

$ gcloud run deploy $SERVICE_NAME \
  --image $REGION-docker.pkg.dev/    $PROJECT_ID/REPOSITORY/$SERVICE_NAME \
  --allow-unauthenticated \
  --region $REGION
```





Source based deployment

- You deploy source code, instead of a container, to Cloud Run. It builds the app with Google Cloud Buildpacks.
- The builder is also used internally by App Engine and Cloud Functions.
- The builder supports Go, Java, Node.js, Python, Ruby, PHP and .NET.
- The builder (gcr.io/buildpacks/builder:v1) and buildpacks are all open source.

Command line

```
$ REGION=us-central1  
  
$ gcloud run deploy $SERVICE_NAME \  
  --source . \  
  --allow-unauthenticated \  
  --region $REGION
```





Source-based vs. container-based deployment

Buildpacks



Provide a hands-off and convenient approach to building container images at the cost of transparency.

- Security centrally handled
- Builds according to Google Best Practices
- Quick and Easy Builds
(trying out new Code using **pack CLI**)
- Easy migrate between Google Cloud Products
- Very well integrated into Cloud Build

Also consider:

Enabling Container Scanning API

Dockerfiles

Offer a lower-level approach that offers transparency and flexibility at the cost of complexity.

- Reduced image size (multi-stage)
 - alpine or scratch as Baseimage
 - not impacting Cloud Run coldstarts
- Full control

Also consider:

Jib , Skaffold





Container contract

- Image Manifest V2, Schema 1 and Schema 2 image formats with Linux x86_64 ABI format binaries
- Entrypoint must listen for HTTP/1, HTTP/2 or /gRPC **requests on 0.0.0.0 without TLS** within four minutes of startup, default port 8080 can be changed in settings
- Port setting from the control plane is injected as **PORT env var** for container
- **CPU is throttled** (and unbilled) when not handling request, including for min-instances (but billed at a lower rate)
- System chooses when to fully scale to zero, **SIGTERM** sent on instance shut, and **10 second grace period** given if trapped, followed by SIGKILL
- Outbound request timeout (20 minutes to the internet and two minutes to a VPC)



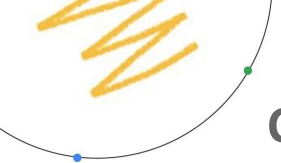


Container contract

Container instance metadata server on `/computeMetadata/v1/...`

Path	Description
<code>.../project/project-id</code>	Project ID of the project the Cloud Run service belongs to
<code>.../project/numeric-project-id</code>	Project number of the project the Cloud Run service belongs to
<code>.../instance/region</code>	Region of this Cloud Run service
<code>.../instance/id</code>	Unique identifier of the container instance (also available in logs).
<code>.../instance/service-accounts/default/token</code>	Generates a token for the runtime service account of this Cloud Run service. The Cloud Run service agent is used to fetch a token.





Container contract

Environment variable

Name	Description	Example
PORT	The port your HTTP server should listen on.	8080
K_SERVICE	The name of the Cloud Run service being run.	hello-world
K_REVISION	The name of the Cloud Run revision being run.	hello-world.1
K_CONFIGURATION	The name of the Cloud Run configuration that created the revision.	hello-world






Execution Environments

There are 2 different environments available for Cloud Run: **first and second generation**

By default, it uses the first generation which provides **fast cold start** and **emulated** system calls.

Let's check some peculiarities for each of them:

First generation

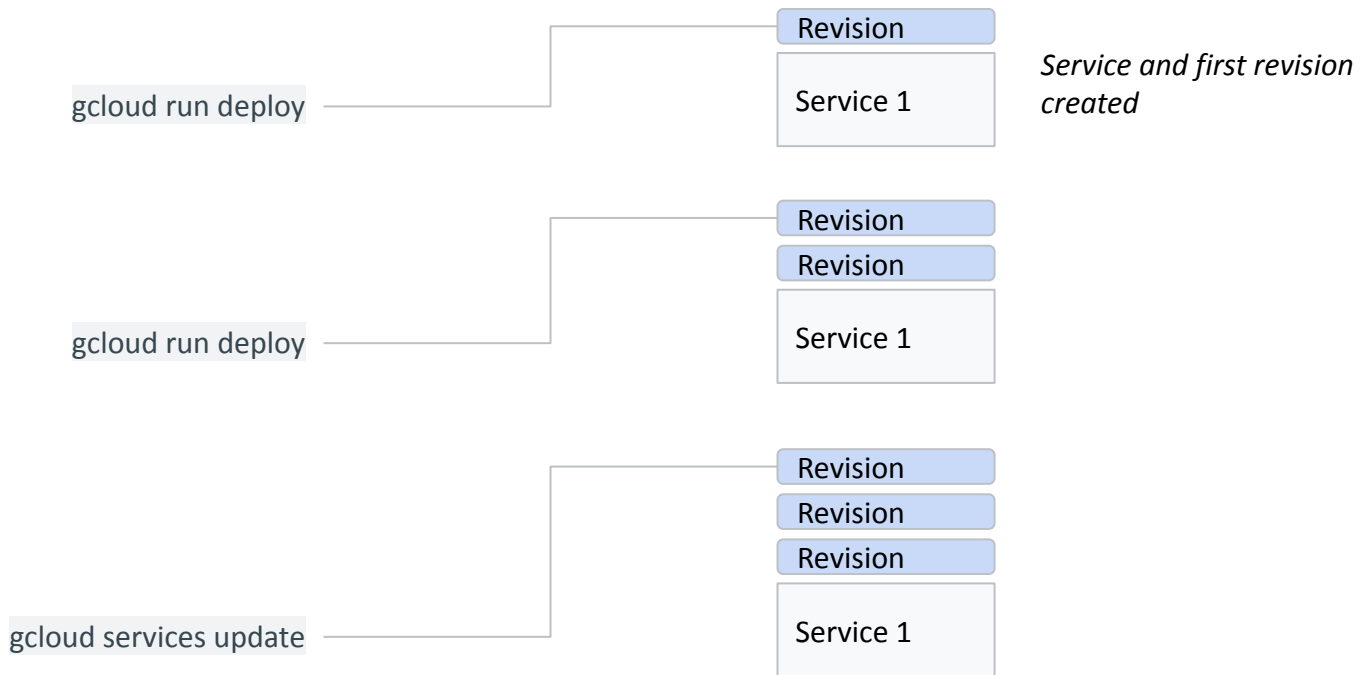
- Emulated operational system
- Fast cold starts
- Best for:
 - Bursty and scale out fast traffic
 - Scale to zero
 - Use less than 512 MiB memory
- Used by Cloud Function 

Second generation

- Full Linux compatibility (syscalls, namespaces and cgroups)
- Faster CPU and network performance
- Network file system support
- Requires at least 512 MiB memory
- Performs better at CPU-intensive steady traffic but has longer cold starts



Cloud Run Resource Model and lifecycle



Traffic Splitting

Look how easy it is to configure traffic splitting on Cloud Run:

Cloud Run Service details

demo-service Region: us-central1 URL: <https://demo-service-00001-noc.run.app>

METRICS REVISIONS LOGS DETAILS

Revisions **MANAGE TRAFFIC**

Filter revisions

	Name	Traffic	Deployed
<input checked="" type="radio"/>	demo-service-00002-xos	0%	1 minute ago
<input type="radio"/>	demo-service-00001-noc	100%	15 minutes ago

Manage traffic

Revision	Traffic percentage	
demo-service-00002-xos	10%	
demo-service-00001-noc Serving	90%	

+ ADD REVISION

CANCEL SAVE





Gradual rollouts and rollbacks

- Specify % traffic between revisions
- Blue/green deployments
- Get URLs for specific revisions by tagging
- Create separate Cloud Build trigger on tag name for update traffic

Gradual rollout

```
$ gcloud beta run deploy myservice \  
    --image us-docker.pkg.dev/.../demo \  
    --no-traffic \  
    --tag green
```

```
$ curl https://green---myservice-12345-us.a.run.app
```

```
$ gcloud beta run services update-traffic myservice \  
    --to-tags green=1
```

```
$ gcloud beta run services update-traffic myservice \  
    --to-tags green=10
```

```
$ gcloud beta run services update-traffic myservice \  
    --to-tags green=50
```

```
$ gcloud beta run services update-traffic myservice \  
    --to-tags green=100
```

Rollback

```
$ gcloud run services update-traffic myservice \  
    --to-revisions my-service-0002-joy=100
```





Local development - Cloud Run

Beside writing your own Docker file or Docker compose you can run Cloud Run services in a **local emulator** with **Hot Reload and configuration emulation** .

Allow for Keyless Secrete Reference from Secret Manager

Available in:

- gcloud
- Cloud Code VSCode
- Cloud Code IntelliJ
- Cloud Shell Editor

```
$ ls
go.mod      main.go     service.dev.yaml

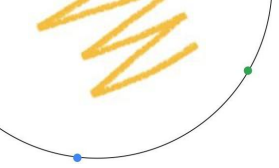
$ gcloud auth application-default login \
  --impersonate-service-account=YOUR_SERVICE_ACCOUNT

$ gcloud code dev service.dev.yaml
Starting development environment 'gcloud-local-dev' ...

Building...
Starting deploy...
Press Ctrl+C to exit
Watching for changes...

Service URL: http://localhost:8080/
```





CI/CD setup - Traffic splitting automation

- Building of some automation is relatively straightforward (consider cloud workflow orchestration) to connect the turning of this knob to metrics.
- We have an open-source reference of this



github.com/GoogleCloudPlatform/cloud-run-release-manager





CI/CD setup

cloudbuild.yaml

- Cloudbuild.yaml
- Cloud Build trigger on
 - Branch name
 - Tag name
- Also run application tests

```
steps:
# Build the container image
- name: 'docker'
  Args: ['build', '-t',
        '$REGION-docker.pkg.dev/$PROJECT_ID/REPOSITORY/XY:$COMMIT_SHA', '.']
# Push the container image to Container Registry
- name: 'docker'
  args: ['push', '$REGION-docker.pkg.dev/$PROJECT_ID/REPOSITORY/XY:$COMMIT_SHA']
# Deploy container image to Cloud Run
- name: 'gcr.io/google.com/cloudsdktool/cloud-sdk'
  entrypoint: gcloud
  args:
    - 'run'
    - 'deploy'
    - 'XY'
    - '--image'
    - '$REGION-docker.pkg.dev/$PROJECT_ID/REPOSITORY/XY:$COMMIT_SHA'
    - '--region'
    - 'REGION'
images:
- '$REGION-docker.pkg.dev/$PROJECT_ID/REPOSITORY/XY:$COMMIT_SHA'
```





CI/CD setup - Declarative



google_cloud_run_service

JUMP TO SECTION ▾

```
resource "google_cloud_run_service" "default" {
  name      = "cloudrun-srv"
  location  = "us-central1"

  template {
    spec {
      containers {
        image = "us-docker.pkg.dev/..../demo"
        env {
          name = "SOURCE"
          value = "remote"
        }
        env {
          name = "TARGET"
          value = "home"
        }
      }
    }
  }

  traffic {
    percent      = 100
    latest_revision = true
  }
  autogenerate_revision_name = true
}
```





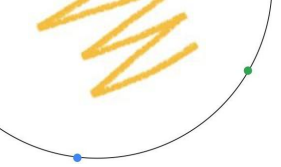
KRM Google API

Cloud Run API is a fully-managed KRM Google API

with no Kubernetes master in the mix...

```
$ gcloud beta run services describe demo-service --format  
yaml  
apiVersion: serving.knative.dev/v1  
kind: Service  
metadata:  
  name:  
    demo-service  
spec:  
  template:  
    spec:  
      containerConcurrency: 80  
      containers:  
        - image: us-docker.pkg.dev/.../demo  
          ports:  
            - containerPort: 8080  
          resources:  
            limits:  
              cpu: 1000m  
              memory: 256M  
            timeoutSeconds: 900  
status:  
  .....
```





KRM Google API

Cloud Run API is a fully-managed KRM Google API

with no Kubernetes master in the mix...

```
$ gcloud run services describe demo-service --format yaml > service.yaml  
  
# Edit yaml file with changes  
  
$ gcloud run services replace service.yaml ...
```

is very similar to kubectl apply ...



CI/CD setup - GitHub Actions



GitHub Actions

- Authenticate GitHub Worker keyless with Workload Identity Federation
- google-github-actions/deploy-cloudrun
 - Imperative Source-Based or Container
 - Declarative via metadata: service.yaml

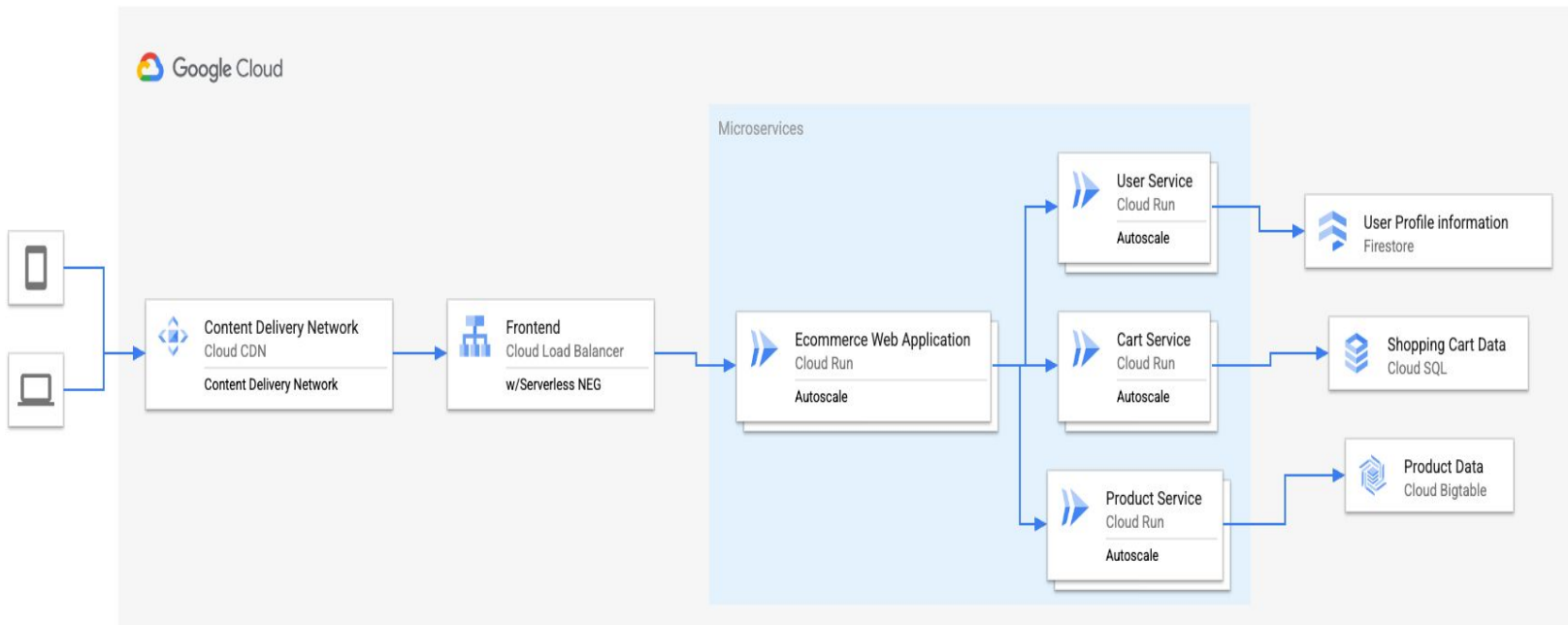
./.github/cloudbuild.yaml

```
steps:  
  - name: Checkout  
    uses: actions/checkout@v2  
  
  - name: Google Auth  
    id: auth  
    uses: 'google-github-actions/auth@v0'  
    with:  
      workload_identity_provider: '${{ secrets.WIF_PROVIDER }}'  
      service_account: '${{ secrets.WIF_SERVICE_ACCOUNT }}'  
  
  - name: Deploy to Cloud Run  
    id: deploy  
    uses: 'google-github-actions/deploy-cloudrun@v0'  
    with:  
      service: '${{ env.SERVICE }}'  
      region: '${{ env.REGION }}'  
      source: ./
```

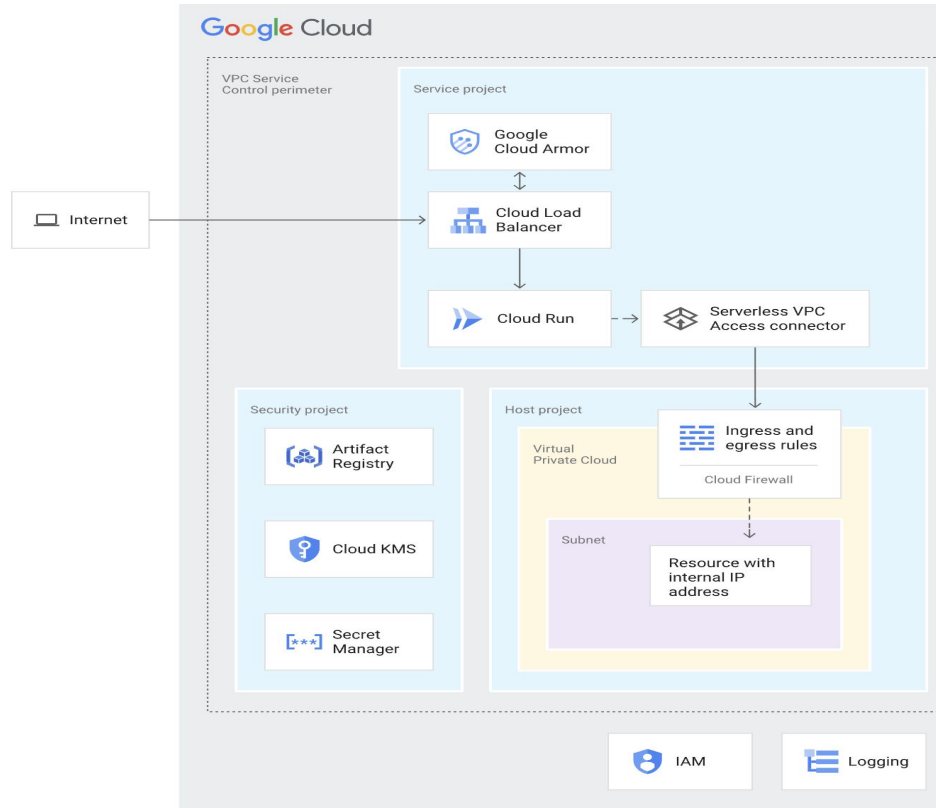


Serverless Architecture

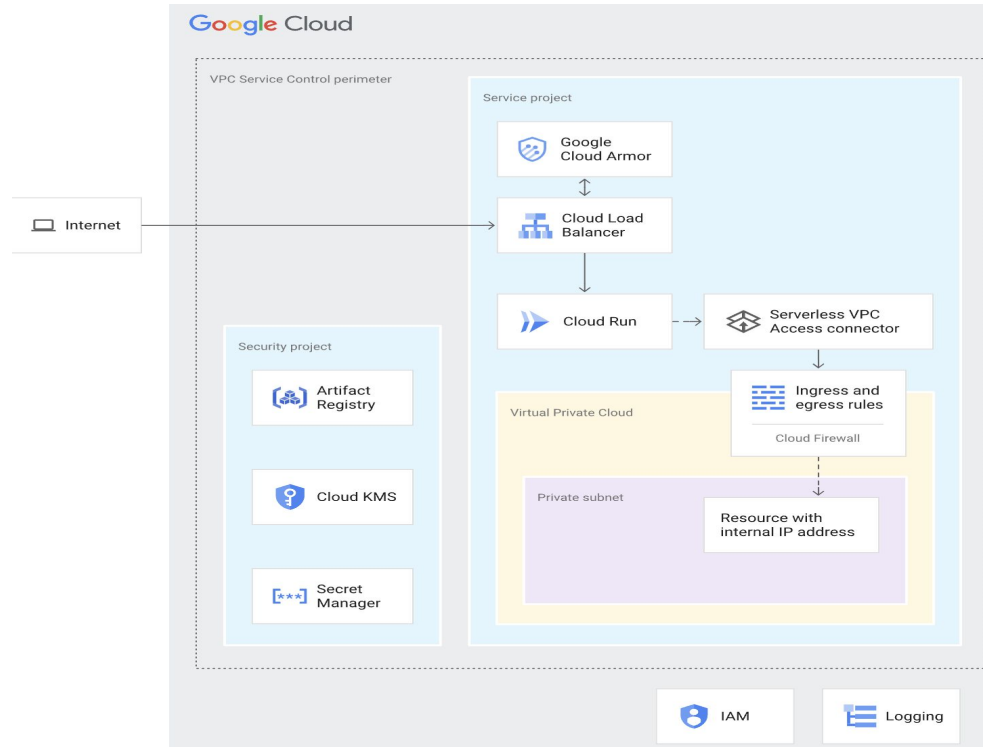
Ecommerce Web Application Architecture



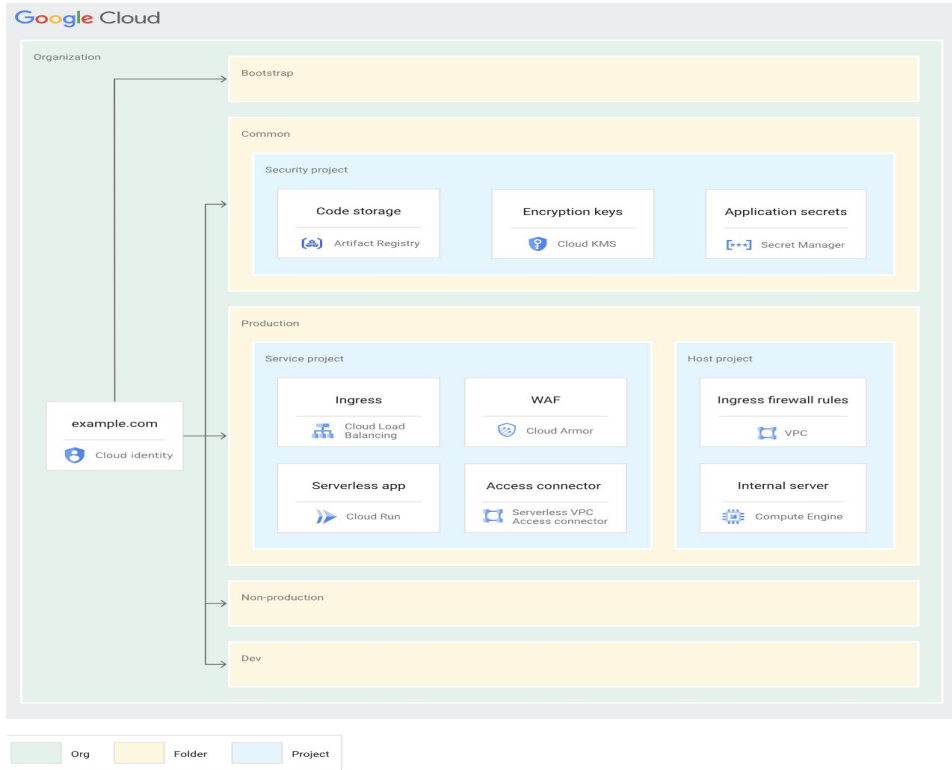
Serverless Application in Shared VPC



Serverless Application without Shared VPC



Organisation Structure



Thanks
Everyone !

<http://tinyurl.com/masterclass-session-2024-patna>



#Google
#MasterClassSession2024

#GoogleCloud
[@ashokjung](#)



Ashok Jung Bahadur

Cloud Architect, Google

[LinkedIn](#) | [Twitter](#) ([@ashokjung](#))

Follow Me
[@ashokjung](#)





Questions?