Google Cloud Run

Serverless Containers Made Simple

The Evolution of Hosting

Traditional Servers → VPS → Cloud

Bare Metal	High upfront	Manual	You do everything
VPS	Fixed monthly	Manual	You manage OS
Cloud (IaaS)	Pay-per-use	Auto	You manage some
Serverless	Pay-per-request	Auto + Scale to 0	Fully managed

> ***The Problem:*** *Traditional hosting = paying 24/7 even with zero traffic*

What is "The Cloud"?

Key Concepts:

- **On-Demand**: Get resources when you need them
- **Pay-As-You-Go**: Only pay for what you use
- **Managed**: Provider handles infrastructure
- **Global**: Deploy worldwide in seconds

Cloud Service Models:

- **IaaS** (Compute Engine): You manage OS, runtime, app
- **PaaS** (App Engine): You manage app only
- **Serverless** (Cloud Run): You manage code only

Why Google Cloud Platform?

GCP Advantages:

- **Google's Infrastructure**: Same as Search, YouTube, Gmail
- **Developer-Friendly**: Clean APIs, excellent docs
- **Innovation**: Kubernetes, BigQuery, Cloud Run
- **Pricing**: Per-second billing, competitive rates

GCP Compute Options:

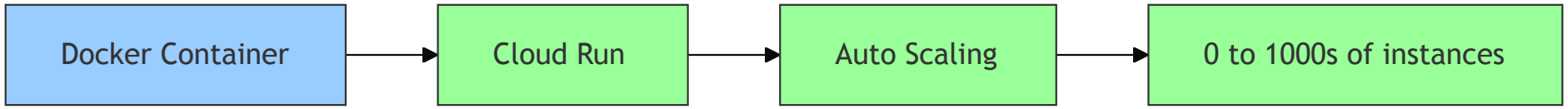
- **Compute Engine**: VMs (full control)
- **GKE**: Kubernetes (complex, powerful)
- **App Engine**: PaaS (limited flexibility)
- **Cloud Run**: Serverless containers (sweet spot!)

What is Cloud Run?

> *"Deploy any container without managing servers"*

The Perfect Balance:

- ✅ **Flexibility**: Any language, any framework (containerized)
- ✅ **Simplicity**: No Kubernetes complexity
- ✅ **Cost**: Scale to zero = $0 when idle
- ✅ **Speed**: Deploy in seconds



Cloud Run Pricing (2024)

Free Tier (Per Month):

- 2 million requests
- 360,000 GB-seconds memory
- 180,000 vCPU-seconds

Paid (Tier 1 Regions):

- **$0.40** per million requests
- **$0.000024** per vCPU-second
- **$0.0000025** per GB-second

> ***Most small apps stay FREE!***

When to Use Cloud Run?

✅
Perfect For:

- Web apps & APIs
- Microservices
- Webhooks
- Variable traffic
- Prototypes

❌
Not For:

- Long tasks (>24h)
- WebSocket-heavy apps
- Persistent local storage

Deployment Method 1: From Docker Hub

```
# 1. Build and push to Docker Hub
docker build -t your-username/my-app:v1 .
docker push your-username/my-app:v1

# 2. Deploy to Cloud Run
gcloud run deploy my-app \
  --image docker.io/your-username/my-app:v1 \
  --region us-central1 \
  --allow-unauthenticated
```

Result: Live app with HTTPS URL in ~30 seconds!

Deployment Method 2: Artifact Registry (Recommended)

Why Artifact Registry?

- Native GCP integration
- Better security & IAM
- Faster (stays in Google network)
- Automatic vulnerability scanning

```
# 1. Create repository
gcloud artifacts repositories create my-repo \
  --repository-format=docker \
  --location=us-central1

# 2. Configure Docker
gcloud auth configure-docker us-central1-docker.pkg.dev
```

Artifact Registry (Continued)

```
# 3. Tag and push
docker tag my-app:v1 \
  us-central1-docker.pkg.dev/PROJECT_ID/my-repo/my-app:v1

docker push \
  us-central1-docker.pkg.dev/PROJECT_ID/my-repo/my-app:v1

# 4. Deploy
gcloud run deploy my-app \
  --image us-central1-docker.pkg.dev/PROJECT_ID/my-repo/my-app:v1 \
  --region us-central1 \
  --allow-unauthenticated
```

Deployment Method 3: From Source Code

Buildpacks Magic!

```
# Deploy directly from source (no Dockerfile needed!)
gcloud run deploy my-app \
  --source . \
  --region us-central1 \
  --allow-unauthenticated
```

What happens?

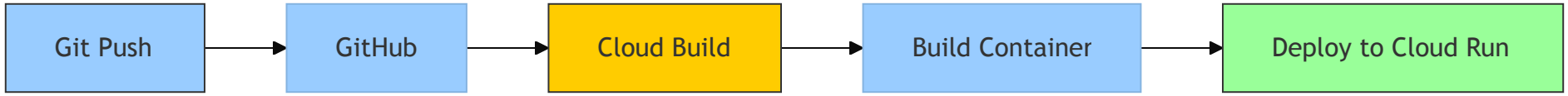
1. Cloud Run detects your language (Node.js, Python, Go, Java)
2. Automatically builds container
3. Deploys it!

Requirements: `package.json`, `requirements.txt`, or `go.mod`

GitHub Integration: Continuous Deployment

The Modern Way:

Push code → Auto-deploy to Cloud Run!



Setup GitHub CI/CD (Easy Way)

Via Cloud Run Console:

1. Go to Cloud Run → Create Service
2. Select "Continuously deploy from repository"
3. Connect GitHub account
4. Select repo and branch
5. Done!

Every push to `main` = automatic deployment!

Setup GitHub CI/CD (Advanced Way)

Create `cloudbuild.yaml`:

```yaml
steps:
  # Build container
  - name: 'gcr.io/cloud-builders/docker'
    args: ['build', '-t', 'us-central1-docker.pkg.dev/$PROJECT_ID/my-repo/my-app:$COMMIT_SHA', '.']

  # Push to registry
  - name: 'gcr.io/cloud-builders/docker'
    args: ['push', 'us-central1-docker.pkg.dev/$PROJECT_ID/my-repo/my-app:$COMMIT_SHA']

  # Deploy to Cloud Run
  - name: 'gcr.io/cloud-builders/gcloud'
    args:
      - 'run'
      - 'deploy'
      - 'my-app'
      - '--image=us-central1-docker.pkg.dev/$PROJECT_ID/my-repo/my-app:$COMMIT_SHA'
      - '--region=us-central1'
```

Create Cloud Build Trigger

```
gcloud builds triggers create github \
  --repo-name=my-repo \
  --repo-owner=my-username \
  --branch-pattern=^main$ \
  --build-config=cloudbuild.yaml
```

Now: Push to GitHub → Cloud Build runs → App deploys automatically!

Autoscaling in Cloud Run

How It Works:

- **Trigger**: Incoming requests
- **Threshold**: CPU > 60%
- **Action**: Spin up new instances
- **Scale Down**: No requests = scale to zero

Configuration:

```
# Set minimum instances (avoid cold starts)
gcloud run services update my-app --min-instances=1

# Set maximum instances (control costs)
gcloud run services update my-app --max-instances=10

# Set concurrency (requests per instance)
gcloud run services update my-app --concurrency=80
```

Cold Starts

What is it?

When scaling from 0→1, Cloud Run needs to:

1. Download container image
2. Start container
3. Initialize app

Typical time: 1-5 seconds

Solutions:

- Set `--min-instances=1` (always warm, but costs more)
- Optimize container size (smaller = faster)
- Use Alpine Linux base images

Monitoring & Logs

View Logs:

```
# Real-time logs
gcloud run services logs tail my-app --region us-central1

# Recent logs
gcloud run services logs read my-app --limit 50
```

Built-in Metrics:

- Request count & latency
- Instance count
- CPU & memory usage
- Error rates

Access: Cloud Run console → Select service → Metrics/Logs tab

Traffic Splitting & Rollbacks

Gradual Rollout:

```
# Deploy new version with 0% traffic
gcloud run deploy my-app \
  --image my-app:v2 \
  --no-traffic

# Send 10% traffic to new version
gcloud run services update-traffic my-app \
  --to-revisions=my-app-v2=10

# Full rollout
gcloud run services update-traffic my-app --to-latest
```

Instant Rollback:

```
gcloud run services update-traffic my-app \
  --to-revisions=my-app-v1=100
```

Environment Variables & Secrets

Environment Variables:

```
gcloud run services update my-app \
  --set-env-vars="NODE_ENV=production,PORT=8080"
```

Secrets (Recommended for sensitive data):

```
# Create secret
echo -n "my-api-key" | gcloud secrets create api-key --data-file=-

# Use in Cloud Run
gcloud run services update my-app \
  --set-secrets="API_KEY=api-key:latest"
```

Custom Domains & HTTPS

```
# Map custom domain
gcloud run domain-mappings create \
  --service my-app \
  --domain api.example.com \
  --region us-central1
```

Result: Automatic HTTPS with managed SSL certificate!

Hands-On: Deploy Your First App

Sample Node.js App (`index.js`):

```
const express = require('express');
const app = express();
const PORT = process.env.PORT || 8080;

app.get('/', (req, res) => {
  res.json({
    message: 'Hello from Cloud Run!',
    timestamp: new Date().toISOString()
  });
});

app.listen(PORT, () => console.log(`Server on port ${PORT}`));
```

Dockerfile

```
FROM node:18-alpine
WORKDIR /app
COPY package*.json ./
RUN npm ci --only=production
COPY . .
EXPOSE 8080
CMD ["node", "index.js"]
```

Deploy Steps

```
# 1. Build
docker build -t my-app .

# 2. Tag for Artifact Registry
docker tag my-app \
  us-central1-docker.pkg.dev/PROJECT_ID/my-repo/my-app:v1

# 3. Push
docker push \
  us-central1-docker.pkg.dev/PROJECT_ID/my-repo/my-app:v1

# 4. Deploy
gcloud run deploy my-app \
  --image us-central1-docker.pkg.dev/PROJECT_ID/my-repo/my-app:v1 \
  --region us-central1 \
  --allow-unauthenticated
```

You'll get a URL like: `https://my-app-xyz-uc.a.run.app`

Best Practices

Development:

- Use Cloud Build for CI/CD
- Implement health check endpoints
- Use structured logging (JSON)
- Version your images

Production:

- Set `min-instances=1` for critical services
- Use Secret Manager for credentials
- Configure appropriate resource limits
- Set up monitoring alerts
- Use traffic splitting for deployments

Cost Optimization

1. **Scale to Zero**: Default = free when idle
2. **Right-Size Resources**: Start with 256MB RAM, 1 vCPU
3. **Optimize Container**: Smaller = faster cold starts
4. **Use Caching**: CDN for static assets
5. **Monitor Usage**: Check metrics regularly

What's Next?

Advanced Topics:

- Cloud Run with Cloud SQL (databases)
- Pub/Sub integration (event-driven)
- VPC connectors (private resources)
- Multi-region deployment
- Cloud Run Jobs (batch processing)

Resources:

- Cloud Run Docs
- Cloud Build Docs
- Quickstart Tutorials

Questions?

Happy Cloud Running!