

Colocando meus containers em produção

Workshop - Fevereiro 2025

AGENDA

01 INTRODUÇÃO AO CLOUD RUN

O que é?

Diferenças

Vantagens

Requisitos

02 CONTAINERIZANDO A APLICAÇÃO

stateless vs stateful

testando o container

03 DEPLOY NO CLOUD RUN

Publicação da imagem

Deploy manual

Testando a aplicação

04 CONFIGURAÇÕES

Variáveis de ambiente

Secrets

Logs

Monitoramento

05 CI / CD

Integrando ao Github

Thank You

O Que é o Cloud Run?

É um serviço **serverless** da Google Cloud que permite executar **containers** sem precisar gerenciar infraestrutura.

Característica	Cloud Run	FaaS (Cloud Functions, AWS Lambda, etc.)
Modelo de execução	Executa containers completos	Executa funções específicas
Controle sobre ambiente	Total (você define o container)	Limitado ao runtime da plataforma
Suporte a linguagens	Qualquer linguagem (dentro de um container)	Linguagens suportadas pela plataforma (ex: Python, Node.js, Go)
Estado	Stateless, mas pode rodar como um serviço de longa duração	Totalmente stateless, cada execução é independente
Escalabilidade	Escala automaticamente baseado no tráfego	Escala automaticamente, mas tem cold start
Cold Start	Pode ser reduzido configurando instâncias mínimas	Pode ter latência alta devido a cold start
Conectividade	Pode acessar bancos de dados via conexões normais	Requer estratégias como connection pooling ou Cloud SQL Proxy
Duração da Execução	Até 60 minutos por requisição	Geralmente máximo de 15 minutos
Casos de Uso	Microserviços, APIs REST/gRPC, processamento assíncrono	Tarefas event-driven, funções simples, processamento esporádico

Quando escolher o Cloud Run?

- ✓ **APIs REST/gRPC** que precisam rodar continuamente.
- ✓ **Aplicações stateful** (ex.: conexão com banco de dados).
- ✓ **Executar tarefas longas** (ex.: processamento de vídeos, ML).
- ✓ **Ambiente padronizado** (ex.: usar dependências personalizadas no container).

Exemplo:

Um serviço de pagamentos que precisa processar transações continuamente e conectar-se a um banco de dados PostgreSQL.

Quando escolher um FaaS?

- ✓ **Execução de funções event-driven** (ex.: Cloud Pub/Sub, Firestore triggers).
- ✓ **Tarefas curtas e esporádicas** (ex.: redimensionamento de imagens, envio de e-mails).
- ✓ **Processamento assíncrono** (ex.: webhook listener, IoT data processing).
- ✓ **Menor custo para execuções pontuais** (FaaS cobra por milissegundos, enquanto Cloud Run tem custo mínimo quando há instâncias ativas).

Exemplo:

Um sistema que processa notificações enviadas via webhook de um provedor externo e precisa apenas rodar uma função de transformação antes de gravar no banco.

Diferença entre aplicação stateless e stateful

A diferença entre **stateful** e **stateless** está relacionada à forma como uma aplicação lida com dados e estados entre diferentes requisições.

Stateful (Com Estado)

Definição:

Uma aplicação **stateful** mantém informações sobre o usuário ou a sessão ao longo do tempo. Ou seja, **cada requisição pode depender de dados armazenados em memória ou persistidos**.

Características de Aplicações Stateful

- ✓ **Mantém informações entre requisições** (sessões, conexões abertas).
- ✓ **Requer um mecanismo de armazenamento de estado** (banco de dados, cache, arquivos).
- ✓ **Mais difícil de escalar** (requisições podem precisar ser roteadas para a mesma instância).
- ✓ **Usado para interações contínuas e persistentes** (ex.: chats, jogos online).

Exemplos de Aplicações Stateful

- 📌 **Banco de Dados** – O PostgreSQL, MySQL e MongoDB são stateful porque precisam manter e gerenciar conexões persistentes.
- 📌 **Aplicações que mantêm sessão** – Como um servidor de login que armazena sessão do usuário no servidor.
- 📌 **Filas de Mensagens** – Como Kafka ou RabbitMQ, que precisam lembrar o ponto onde um consumidor parou.

Stateless (Sem Estado)

Definição:

Uma aplicação **stateless** não mantém informações entre requisições. Cada requisição é **independente** e contém todas as informações necessárias para ser processada.

Características de Aplicações Stateless

- ✓ Cada requisição é tratada de forma independente (não precisa armazenar informações sobre clientes).
- ✓ Fácil de escalar (qualquer instância pode processar qualquer requisição).
- ✓ Menor consumo de memória e recursos.
- ✓ Melhor para microsserviços, APIs REST e serviços distribuídos.





Exemplos de Aplicações Stateless

- 📌 APIs REST/gRPC – Cada requisição contém todas as informações necessárias (ex.: `Authorization: Bearer token`).
- 📌 Serviços de autenticação via JWT – O token JWT contém todas as informações, sem necessidade de armazenar sessão.
- 📌 Cloud Run e AWS Lambda – Eles escalam porque cada requisição pode ser processada por qualquer instância.

Compartilhamento de recursos

O **Cloud Run escala automaticamente** e pode rodar **várias instâncias simultâneas** da sua aplicação. Portanto, **não compartilhe recursos entre instâncias**.

O que isso significa na prática?

-  **Não armazene estado no container** (cada instância é independente).
-  **Use bancos de dados externos** (Cloud SQL, Firestore, BigQuery, Redis, etc.).
-  **Evite usar in-memory cache** (ex.: `map` global no Go, `in-memory cache` no Node.js), pois não há garantia de que a mesma instância vai responder a uma requisição futura.
-  **Se precisar de cache, use serviços externos**, como **Redis ou Memorystore**.

```
var contador int

func handler(w http.ResponseWriter, r *http.Request) {
    contador++
    fmt.Fprintf(w, "Requisição número: %d", contador)
}
```




```
client := redis.NewClient(&redis.Options{Addr: "redis-server:6379"})

func handler(w http.ResponseWriter, r *http.Request) {
    contador, _ := client.Incr("contador").Result()
    fmt.Fprintf(w, "Requisição número: %d", contador)
}
```

Gerenciamento de Sessões

Como o Cloud Run escala dinamicamente e pode iniciar ou destruir instâncias a qualquer momento, **não use sessões armazenadas localmente**.

Alternativas seguras para autenticação e sessão:

-  Use JWT (JSON Web Tokens) para autenticação stateless.
-  Armazene sessões em um banco de dados externo (Cloud SQL, Firestore).
-  Use Firebase Authentication ou Google Identity Platform para autenticação segura.

```
sessions = {}

@app.route('/login', methods=['POST'])
def login():
    session_id = str(uuid.uuid4())
    sessions[session_id] = request.json['username']
    return jsonify({"session_id": session_id})
```

```
@app.route('/profile', methods=['GET'])
def profile():
    user = verify_firebase_token(request.headers['Authorization'])
    return jsonify({"user": user})
```

Logs

O Cloud Run **não mantém logs dentro do container**, mas captura automaticamente qualquer saída para **stdout** e **stderr**, enviando para **Cloud Logging**.

Melhores práticas para logs no Cloud Run

- ✓ Use **stdout** e **stderr** para logs (o GCP os captura automaticamente).
- ✓ Formate os logs em **JSON** para facilitar buscas e filtros.
- ✓ Evite escrever logs em arquivos dentro do container (eles serão perdidos ao reiniciar a instância).

```
logger := slog.New(slog.NewJSONHandler(os.Stdout, nil))  
logger.Info("Nova requisição recebida", "path", r.URL.Path, "method", r.Method)
```

Hands ON!