

Começar o laboratório

01:00:00

Como desenvolver uma API REST com o Go e o Cloud Run

1 hora Gratuito ★★★★

GSP761	-/100
Visão geral	
Pré-requisitos	
Configuração	
Ative as APIs do Google	
Como desenvolver a API REST	
Importe os dados dos clientes fictícios para fazer os testes	
Conecte a API REST ao banco de dados do Firestore	
Teste rápido	
Como implantar uma nova revisão	
Parabéns!	
Terminar o laboratório	

GSP761
 Google Cloud Self-Paced Labs


Visão geral

Neste laboratório, você vai analisar uma situação e seguir as mesmas etapas que os personagens para resolver o problema da empresa.

Há 12 anos, Lillian fundou a rede de clínicas veterinárias Pet Theory. À medida que a rede foi crescendo, Lillian começou a passar mais tempo no telefone com corretores de seguro do que cuidando dos animais. Seria ótimo se as corretores pudessem acessar on-line o valor total dos tratamentos.

Nos laboratórios anteriores desta série, Raquel, a consultora de informática, e Pedro, o engenheiro de DevOps, migraram o banco de dados de clientes da Pet Theory para um banco de dados sem servidor do Firestore na nuvem. Depois eles disponibilizaram o acesso para os clientes podermos marcar as consultas pela Internet. Como a Pet Theory só tem uma pessoa responsável pelas operações, a empresa precisa de uma solução sem servidor que não exija manutenção contínua.

Neste laboratório, você vai ajudar a Raquel e o Pedro a disponibilizar os dados dos clientes para as corretores de seguro sem expor as informações de identificação pessoal (PII). Você vai criar um gateway seguro da API Representation State Transfer (REST) com o Cloud Run, que não precisa de servidor. Isso vai permitir que as seguradoras vejam o custo total dos tratamentos sem ter acesso às PII dos clientes.

Pré-requisitos

Para este laboratório de **nível fundamental**, é preciso ter familiaridade com o Console do Cloud e os ambientes do Cloud Shell. Este laboratório faz parte de uma série. É recomendável, mas não necessário, que você tenha concluído os laboratórios anteriores:

- Como importar dados para um banco de dados sem servidor
- Criar um app da Web sem servidor com o Firebase e o Firestore
- Criar um app sem servidor que gera arquivos PDF

Você também precisa saber editar arquivos. Use o editor de texto que preferir (como nano, vi etc.) ou inicie o editor de código do Cloud Shell, disponível na faixa de opções na parte superior:

Configuração

Antes de clicar no botão Start Lab

Leia estas instruções. Os laboratórios são cronometrados e não podem ser pausados. O timer é iniciado quando você clica em **Começar o laboratório** e mostra por quanto tempo os recursos do Google Cloud ficarão disponíveis.

Este laboratório prático do Qwiklabs permite que você realize as atividades em um ambiente real de nuvem, não em uma simulação ou demonstração. Você receberá novas credenciais temporárias para fazer login e acessar o Google Cloud durante o laboratório.

O que é necessário

Para fazer este laboratório, você precisa ter:

- acesso a um navegador da Internet padrão (recomendamos o Chrome);
- tempo para concluir as atividades.

Observação: não use seu projeto ou sua conta do Google Cloud neste laboratório.

Observação: se estiver usando um dispositivo Chrome OS, abra uma janela anônima para executar o laboratório.

Como iniciar seu laboratório e fazer login no console do Google Cloud

1. Clique no botão **Começar o laboratório**. Se for preciso pagar, você verá um pop-up para selecionar a forma de pagamento. No painel **Detalhes do laboratório** à esquerda, você verá o seguinte:

- O botão **Abrir Console do Cloud**

- Tempo restante

- As credenciais temporárias que você vai usar neste laboratório
 - Outras informações se forem necessárias
2. Clique em **Abrir Console do Google**. O laboratório ativa recursos e depois abre outra guia com a página [Fazer login](#).

Dica: coloque as guias em janelas separadas lado a lado.

Observação: se aparecer a caixa de diálogo **Escolher uma conta**, clique em **Usar outra conta**.

3. Caso seja preciso, copie o **Nome de usuário** no painel **Detalhes do laboratório** e cole esse nome na caixa de diálogo **Fazer login**. Clique em **Avançar**.

4. Copie a **Senha** no painel **Detalhes do laboratório** e a cole na caixa de diálogo **Ola**. Clique em **Avançar**.

Importante: você precisa usar as credenciais do painel à esquerda. Não use suas credenciais do Google Cloud Ensina.

Observação: se você usar sua própria conta do Google Cloud neste laboratório, é possível que receba cobranças adicionais.

5. Acesse as próximas páginas:

- Aceite os **Termos e Condições**.
- Não adicione opções de recuperação nem autenticação de dois fatores (porque essa é uma conta temporária).
- Não se inscreva em testes gratuitos.

Depois de alguns instantes, o console do GCP vai ser aberto nesta guia.

Observação: para ver uma lista dos produtos e serviços do Google Cloud, clique no **Menu de navegação** no canto superior esquerdo.



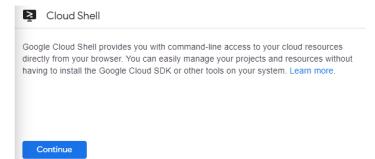
Ative o Google Cloud Shell

O Google Cloud Shell é uma máquina virtual com ferramentas de desenvolvimento. Ele conta com um diretório principal permanente de 5 GB e é executado no Google Cloud. O Google Cloud Shell permite acesso de linha de comando aos seus recursos do GCP.

1. No Console do GCP, na barra de ferramentas superior direita, clique no botão **Abrir o Cloud Shell**.



2. Clique em **Continue** (continuar):



Demora alguns minutos para provisionar e conectar-se ao ambiente. Quando você está conectado, você já está autenticado e o projeto é definido como seu **PROJECT_ID**. Por exemplo:



gcloud é a ferramenta de linha de comando do Google Cloud Platform. Ele vem pré-instalado no Cloud Shell e aceita preenchimento com tabulação.

É possível listar o nome da conta ativa com este comando:



Saída:



É possível listar o ID de projeto com este comando:



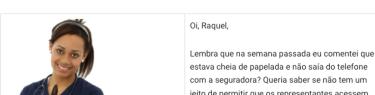
Saída:



Exemplo de saída:



A documentação completa do **gcloud** está disponível na página [Visão geral do gcloud](#) do Google Cloud.



	<p>os registros dos clientes de uma forma eficiente e segura.</p> <p>Este nível atual de carga de trabalho não é sustentável. Você pode ajudar?</p>
	<p>Lilian</p> <p>Oi, Lilian,</p> <p>Almocei com o Pedro ontem e criamos um plano para facilitar o acesso seguro de terceiros autorizados aos registros digitais da Pet Theory.</p> <p>O plano tem quatro etapas:</p> <ol style="list-style-type: none"> 1. Criar uma API REST simples 2. Importar os dados de teste dos clientes 3. Conectar a API REST ao banco de dados dos clientes 4. Adicionar autenticação à API REST <p>Eu e o Pedro já sabemos como executar as duas primeiras etapas, o que já é um bom começo. Queremos ter um protótipo operacional pronto até o final da semana.</p>
	<p>Raquel</p>

Ajude Raquel a gerenciar as atividades necessárias para criar a API REST da Pet Theory.

Ative as APIs do Google

Duas APIs foram ativadas para este laboratório:

Nome	API
Cloud Build	cloudbuild.googleapis.com
Cloud Run	run.googleapis.com

Como desenvolver a API REST

1. Ative o projeto:

```
gcloud config set project $(gcloud projects list --format='value(PROJECT_ID)' --filter="qwiklabs-gcp")
```

2. Clone o repositório da teoria pet-theory e acessar o código fonte:

```
git clone https://github.com/rosera/pet-theory.git && cd pet-theory/lab08
```

3. Use seu editor de texto favorito, ou use o botão Editor de código na faixa do Cloud Shell, para ver o arquivo `go.mod` e `go.sum`

4. Crie o arquivo `main.go` e adicione o conteúdo abaixo ao arquivo:

```
package main
import (
    "fmt"
    "log"
    "net/http"
    "os"
)
func main() {
    port := os.Getenv("PORT")
    if port == "" {
        port = "8080"
    }
    http.HandleFunc("/v1/", func(w http.ResponseWriter, r *http.Request) {
        fat.Println(w, "status: 'running'")
    })
    log.Println("Pets REST API listening on port", port)
    if err := http.ListenAndServe(":port", nil); err != nil {
        log.Fatalf("Error launching Pets REST API server: %v", err)
    }
}
```

No código acima, crie um endpoint para testar se o serviço está funcionando conforme o esperado. Se quiser, insira "/v1/" no URL do serviço para ver se o aplicativo está funcionando corretamente. Como o Cloud Run implanta contêineres, é necessário apresentar uma definição de contêiner. Um arquivo chamado "Dockerfile" informa ao Cloud Run qual versão do Go usar, quais arquivos incluir no aplicativo e como iniciar o código.

5. Agora crie um arquivo com o nome `Dockerfile` e inclua este código nele:

```
FROM gcr.io/distroless/base-debian10
COPY . /app
COPY server /app/server
CMD [ "/app/src/app/server" ]
```

O arquivo `server` é o binário de execução criado em `main.go`.

6. Execute o comando a seguir para criar o binário:

```
go build -o server
```

7. Depois de executar o comando `build`, confirme que o `Dockerfile` e o servidor estão no mesmo diretório:

```
ls -la
```

(Saida)

```
.
├── Dockerfile
└── server
```

Um Dockerfile modelo como o acima normalmente é usado sem alterações para a maioria dos apps do Cloud Run baseados em Go.

8. Para implantar sua API REST simples, basta executar:

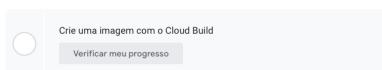
```
gcloud builds submit \
```

```
--tag gcr.io/GOOGLE_CLOUD_PROJECT/rest-api:0.1
```

Esse comando cria um contêiner com seu código e o coloca no Container Registry do projeto. Clique em **Menu de navegação > Container Registry** para ver o contêiner. Se **rest-api** não aparecer, clique em **Atualizar**.



Clique em **Verificar meu progresso** para confirmar que você concluiu a tarefa acima.



9. Depois que o contêiner for criado, implante-o:

```
gcloud run deploy rest-api \
--image gcr.io/GOOGLE_CLOUD_PROJECT/rest-api:0.1 \
--platform managed \
--region us-central1 \
--allow-unauthenticated \
--max-instances=2
```

10. Quando a implantação estiver concluída, você vai receber uma mensagem como esta:



Clique em **Verificar meu progresso** para confirmar que você concluiu a tarefa acima.



11. Clique no URL do serviço que aparece no fim dessa mensagem para abri-lo em uma nova guia do navegador. Insira `/v1/` no final do URL e pressione **Enter**. Você vai receber esta mensagem:



A API REST está configurada e em execução. Na próxima seção, a API vai ser usada para recuperar informações de clientes fictícios de um banco de dados do Firestore com o serviço de protótipo disponível.

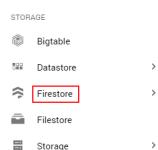
Importe os dados dos clientes fictícios para fazer os testes



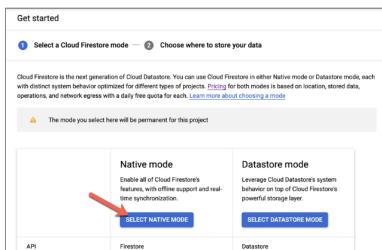
Raquel e Pedro já criaram um banco de dados de teste com 10 clientes, que inclui alguns tratamentos propostos para o gato de um cliente.

Ajude Pedro a configurar o banco de dados do Firestore e a importar os dados de teste dos clientes. Primeiro ative o Firestore no projeto.

1. Volte ao Console do Cloud e clique em **Menu de navegação > Firestore**.



2. Clique no botão **Selecionar Modo nativo**.



3. Selecione o local como a multirregião "nam5 (Estados Unidos)" no início da lista.

4. Clique no botão **Criar banco de dados**.

5. Aguarde o banco de dados ser criado.

Clique em **Verificar meu progresso** para confirmar que você concluiu a tarefa acima.



6. Migre os arquivos de importação para um bucket do Cloud Storage criado para você:

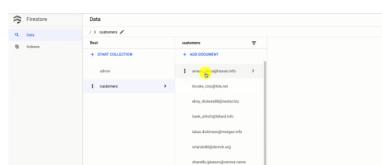
```
gsutil cp -r gs://spl/gsp445/2019-10-06T20:10:37_43617  
gs://$GOOGLE_CLOUD_PROJECT-customer
```

7. Agora importe estes dados para o Firestore:

```
gcloud beta firestore import gs://$GOOGLE_CLOUD_PROJECT-customer/2019-10-  
06T20:10:37_43617/
```

Atualize o navegador do Console do Cloud para ver os resultados do Firestore.

8. No Firestore, clique em **clientes** na seção "Raiz". Você vai ver que os dados dos animais de estimação foram importados, como mostra a captura de tela abaixo. Se quiser, navegue para ver as informações. Caso não veja os dados, atualize a página.



Bom trabalho! O banco de dados do Firestore foi criado e preenchido com os dados de teste.

Conecte a API REST ao banco de dados do Firestore



Nesta seção, você vai ajudar a Raquel a criar outro endpoint na API REST com a seguinte aparência:

```
https://rest-api-[hash].a.run.app/v1/customer/22530
```

Por exemplo, o URL deve retornar o total de tratamentos propostos e quantos foram aceitos e rejeitados pelo cliente com o ID 22530, se essas informações estiverem no banco de dados do Firestore:

```
{
  "status": "success",
  "data": {
    "proposed": 1662,
    "approved": 585,
    "rejected": 489
  }
}
```

Se o cliente não estiver no banco de dados, você vai ver o código de status 404 (não encontrado) e uma mensagem de erro.

Essa nova funcionalidade requer um pacote de acesso ao banco de dados do Firestore e outro que lida com o compartilhamento de recursos entre origens (CORS, na sigla em inglês).

1. Descubra o valor da variável de ambiente \$GOOGLE_CLOUD_PROJECT

```
echo $GOOGLE_CLOUD_PROJECT
```

2. Abra o arquivo main.go no diretório pet-theory/lab08.

Atualize o conteúdo de main.go com o valor exibido para \$GOOGLE_CLOUD_PROJECT

3. Substitua o conteúdo do arquivo pelo código abaixo e, em seguida, substitua o PROJECT_ID:

```
package main
import (
    "context"
    "encoding/json"
    "fmt"
    "log"
    "net/http"
    "os"
    "cloud.google.com/go/firestore"
    "github.com/gorilla/handlers"
    "github.com/gorilla/mux"
    "google.golang.org/api/iterator"
)
var client *firestore.Client
func main() {
    var err error
    ctx := context.Background()
    client, err = firestore.NewClient(ctx, "PROJECT_ID")
    if err != nil {
        log.Fatalf("Error initializing Cloud Firestore client: %v", err)
    }
    port := os.Getenv("PORT")
    if port == "" {
```

```

    r := mux.NewRouter()
    r.HandleFunc("/v1/", rootHandler)
    r.HandleFunc("/customer/{id}", customerHandler)
    log.Println("Pets REST API listening on port: ", port)
    cors := handlers.CORS(
        handlers.AllowedHeaders([]string{"X-Requested-With",
            "Authorization", "Origin"}))

    handlers.AllowOrigins([]string{"https://storage.googleapis.com"}, cors)
    handlers.AllowMethods([]string{"GET", "HEAD", "POST",
        "OPTIONS", "PATCH", "CONNECT"})
    if err := http.ListenAndServe(":8080", cors); err != nil {
        log.Fatalf("Error launching Pets REST API server: %v", err)
    }
}

```

4. Adicione o suporte ao gerenciador no final do arquivo:

```

func rootHandler(w http.ResponseWriter, r *http.Request) {
    fmt.Fprintf(w, "status: 'running'")
}
func customerHandler(w http.ResponseWriter, r *http.Request) {
    id := mux.Vars(r)["id"]
    ctx := context.Background()
    customer, err := getCustomer(ctx, id)
    if err != nil {
        w.WriteHeader(http.StatusInternalServerError)
        fmt.Fprintf(w, fmt.Sprintf("Customer \"%s\" not found", id))
        fmt.Fprintf(w, fmt.Sprintf("status: \"fail\", data: \"%s\"", title))
        return
    }
    if customer == nil {
        w.WriteHeader(http.StatusNotFound)
        msg := fmt.Sprintf("Customer \"%s\" not found", id)
        msg += fmt.Sprintf("status: \"fail\", data: \"%s\"", title)
        return
    }
    amount, err := getAmounts(ctx, customer)
    if err != nil {
        w.WriteHeader(http.StatusInternalServerError)
        fmt.Fprintf(w, fmt.Sprintf("status: \"fail\", data: \"Unable to fetch amounts: %s\"", err))
        return
    }
    data, err := json.Marshal(amount)
    if err != nil {
        w.WriteHeader(http.StatusInternalServerError)
        fmt.Fprintf(w, fmt.Sprintf("status: \"fail\", data: \"Unable to fetch amounts: %s\"", err))
        return
    }
    fmt.Fprintf(w, fmt.Sprintf("status: \"success\", data: %s", data))
}

```

5. Adicione o suporte ao cliente depois:

```

type Customer struct {
    Email string `firestore:"email"`
    ID    string `firestore:"id"`
    Name  string `firestore:"name"`
    Phone string `firestore:"phone"`
}
func getCustomer(ctx context.Context, id string) (*Customer, error) {
    query := client.Collection("customers").Where("id", "==", id)
    iter := query.Documents(ctx)
    var c Customer
    for {
        doc, err := iter.Next()
        if err == iterator.Done {
            break
        }
        if err != nil {
            return nil, err
        }
        err = doc.DataTo(&c)
        if err != nil {
            return nil, err
        }
    }
    return &c, nil
}
func getAmounts(ctx context.Context, c *Customer) (map[string]int64,
    error) {
    if c == nil {
        return map[string]int64{}, fmt.Errorf("Customer should be non-nil: %v", c)
    }
    result := map[string]int64{
        "proposed": 0,
        "approved": 0,
        "rejected": 0,
    }
    query := client.Collection(fmt.Sprintf("customers/%s/treatments",
        c.Email))
    if query == nil {
        return map[string]int64{}, fmt.Errorf("Query is nil: %v", c)
    }
    iter := query.Documents(ctx)
    for {
        doc, err := iter.Next()
        if err == iterator.Done {
            break
        }
        if err != nil {
            return nil, err
        }
        treatment := doc.Data()
        result[treatment["status"].(string)] += treatment["cost"].(int64)
    }
    return result, nil
}

```

6. Salve o arquivo.

Teste rápido

Qual função responde aos URLs com o padrão `/v1/customer/`?

customerHandler
 getAmounts

Qual instrução retorna "success" para o cliente?

fmt.Fprintf(w, fmt.Sprintf("status: \"success\", data: %s"))
 fmt.Fprintf(w, fmt.Sprintf("status: \"fail\", data: \"Unable to fetch amounts: %s\""))

Quais funções leem as informações no banco de dados do

Firestore?

getCustomer e getAmounts

customerHandler e getCustomer

Enviar

Como implantar uma nova revisão

1. Recrie o código-fonte:

```
go build -o server
```

2. Crie uma nova imagem para a API REST:

```
gcloud builds submit \
--tag gcr.io/$GOOGLE_CLOUD_PROJECT/rest-api:0.2
```

3. Implante a imagem atualizada:

```
gcloud run deploy rest-api \
--image gcr.io/$GOOGLE_CLOUD_PROJECT/rest-api:0.2 \
--platform managed \
--region us-central1 \
--allow-unauthenticated \
--max-instances=2
```

Clique em **Verificar meu progresso** para confirmar que você concluiu a tarefa acima.

Revisão da imagem do build 0.2

Verificar meu progresso

4. Quando a implantação estiver concluída, você vai receber uma mensagem semelhante à anterior. O URL da API REST não mudou com a implantação da nova versão.

```
Service [rest-api] revision [rest-api-00062] has been deployed and is
serving traffic at https://rest-api-[hash].a.run.app
```

5. Volte à guia do navegador que já aponta para esse URL (com /v1 no final). Atualize a guia para receber a mesma mensagem de antes, informando que a API ainda está em execução.

HTTP://rest-api-[hash].a.run.app/v1

{"status": "running"}

6. Insira /customer/22538 no URL do aplicativo na barra de endereço do seu navegador. Você vai receber esta resposta JSON com o total dos tratamentos propostos e quantos foram aceitos e rejeitados pelo cliente:

HTTP://rest-api-[hash].a.run.app/v1/customer/22538

{"status": "success", "data": {"proposed": 1162, "approved": 585, "rejected": 489}}

Aqui estão alguns IDs de cliente adicionais que você pode inserir no URL, em vez de 22530:

- 34216
- 70156 (todos os valores devem ser zero)
- 12345 (o cliente/animal não existe, e você recebe uma mensagem de erro, como **Query is null**)

Você criou uma API REST escalonável, de baixa manutenção, sem servidor e que lê as informações de um banco de dados.

Parabéns!

Raquel e Pedro criaram um protótipo de API REST para a Pet Theory.



Termine a Quest

Este laboratório autoguiado é parte da Quest [Workshop de Google Cloud Run sem servidor](#) do Qwiklabs. Uma Quest é uma série de laboratórios relacionados que formam o programa de aprendizado. Concluir esta Quest dá a você o selo acima como reconhecimento pela sua conquista. Você pode publicar os selos e incluir um link para eles no seu currículo on-line ou nas redes sociais. Caso você já tenha feito este laboratório, inscreva-se nesta Quest para ganhar os créditos de conclusão imediatamente. Veja outras [Quests do Qwiklabs](#).

Comece o próximo laboratório

Continue aprendendo com o próximo laboratório da série [Como gerar PDFs com o Go e o Cloud Run](#).

Terminar o laboratório

Após terminar seu laboratório, clique em **End Lab**. O Qwiklabs removerá os recursos usados e limpará a conta para você.

Você poderá classificar sua experiência neste laboratório. Basta selecionar o número de estrelas, digitar um comentário e clicar em **Submit**.

O número de estrelas indica o seguinte:

- 1 estrela = muito insatisfeito
- 2 estrelas = insatisfeito
- 3 estrelas = neutro
- 4 estrelas = satisfeito
- 5 estrelas = muito satisfeito

Feche a caixa de diálogo se não quiser enviar feedback.

Para enviar seu feedback, fazer sugestões ou correções, use a guia [Support](#).

Manual atualizado em 11 de agosto de 2021

Laboratório testado em 11 de agosto de 2021

Copyright 2020 Google LLC. Todos os direitos reservados. Google e o logotipo do Google são marcas registradas da Google LLC. Todos os outros nomes de produtos e empresas podem ser marcas registradas das respectivas empresas a que estão associados.

Continuar a Quest

Laboratório

Como desenvolver
uma API REST com o
Go e o Cloud Run

Iniciar