

# Entendendo o CGI, FastCGI e WSGI



Klaus Peter Laube

2 de novembro de 2012 - 12 minutos de leitura



Pelos vários anos que programei com o *PHP* e *Apache*, nunca precisei me preocupar com o que acontecia entre esses dois. Para mim, era tudo uma “mágica” maravilhosa, que entregava as minhas páginas *web* de forma dinâmica. Era uma troca justa: Eles não me traziam preocupação, logo, eu não me preocupava.

Com o passar do tempo, o uso do *Nginx* e a necessidade de aprender *Python*, comecei a me deparar com o famoso `cgi-bin`, e entender que os truques que o *mod\_php* ocultava iam muito além do que eu imaginava.

## O Common Gateway Interface

De um modo bem simples, podemos dizer que o *Common Gateway Interface* é um “acordo” entre os servidores *HTTP* e as aplicações *web*. Por baixo dos panos, o servidor *web* vai informar uma série de parâmetros para o seu programa, e é dever do seu programa entregar uma resposta “bem formada” para o servidor *web*.

Isso quer dizer que, para o *CGI*, não importa qual linguagem ou banco de dados o seu programa está usando. Para ele, importa a passagem dos parâmetros e a resposta. Logo, é perfeitamente possível desenvolvermos nossas páginas até mesmo com a linguagem C:

```
#include <stdio.h>

int main(int argc, char *argv[])
{
    printf("Content-type:text/html\n\n");
    printf("<html>\n");
    printf("<body bgcolor=\"%s\">\n", argv[1]);
    printf("</body>");
}
```

```
printf("</html>");

return 0;
}
```

Basta compilar o código acima, jogar no `cgi-bin` do seu *Apache*, e você verá a flexibilidade do protocolo em ação. Neste exemplo, acessando nosso programa através da URL `http://localhost/cgi-bin/exemplo?red` (por exemplo), veremos apenas uma página com o fundo vermelho. Mas é importante reparar que, o parâmetro passado na URL (`?red`) está acessível através do `argv`, ou seja, o protocolo está passando para o nosso programa os parâmetros através da `STDIN`.

Através da `STDOUT`, estamos respondendo ao *Apache* utilizando de artifícios do protocolo. A nossa mensagem é composta por um cabeçalho informando o tipo da mensagem e o conteúdo. Neste exemplo, trata-se de um *HTML* extremamente simples, *James Marshall* escreveu um bom exemplo um pouco mais complexo utilizando a linguagem *C*.

Outro comportamento fundamental do *CGI* é a criação de variáveis de ambiente. Variáveis que você já deve ter usado, como `REMOTE_HOST`, `REMOTE_ADDR`, `REQUEST_METHOD` e `QUERY_STRING`, são preenchidas pelo servidor *Web* e passadas ao seu programa através do protocolo:

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
    char *addr, *method, *query_string;

    addr = getenv("REMOTE_ADDR");
    method = getenv("REQUEST_METHOD");
    query_string = getenv("QUERY_STRING");

    printf("Content-type:text/html\n\n");
    printf("Remote address: %s<br/>", addr);
    printf("Method: %s<br/>", method);
    printf("Query string: %s<br/>", query_string);
}
```

# O FastCGI

O *FastCGI* segue o mesmo princípio do *CGI*, mas possui uma série de particularidades (e vantagens) em relação ao seu “primogênito”. Para compreender a diferença entre eles, vamos analisar o ciclo de vida de uma requisição utilizando o *CGI*:

- A cada requisição, o servidor *web* **cria um novo processo**;
- Através deste processo, o servidor *web* passa informações para o “programa *CGI*” utilizando **variáveis de ambiente**;
- O servidor *web* também passa qualquer *input* de dados do usuário através da ***STDIN***;
- O programa retorna uma saída ao servidor *web* através do protocolo *CGI* (utilizando a ***STDOUT***);
- Quando o **programa acabar**, a requisição é finalizada.

Em um cenário com poucas requisições, este fluxo atende perfeitamente. Os problemas começam a aparecer quando temos que lidar com **alto consumo** (algo comum hoje em dia, mas nem tão comum quando conceberam o protocolo *CGI*). Dentre os principais problemas, temos:

- Criar e destruir um processo a cada requisição aumenta o *load* do seu servidor, o que fatalmente degrada performance;
- Não há reúso de recursos, como conexões com banco de dados e *caches* em memória (já que a cada nova requisição é iniciado um novo processo);
- Não é trivial separar a sua aplicação do seu servidor *Web*.

Foi pensando em performance e escalabilidade que o *FastCGI* foi criado. Ao contrário do *CGI*, ele utiliza “processos persistentes”, onde o servidor *web* é capaz de iniciar um processo que responde a uma série de requisições. Além disso, ele usa **multiplexação** para transmitir e receber informações dentro de uma única conexão, que pode ser um *socket* ou uma conexão *TCP*. Desse modo, você pode ter o seu servidor *web* e o seu processo *FastCGI* em máquinas diferentes.

O ciclo de vida de uma requisição *FastCGI*, é basicamente composto por:

- O servidor *web* **cria um processo *FastCGI*** para receber requisições;
- A sua aplicação é inicializada, e **aguarda por uma nova conexão** vinda do servidor *web*;

- Quando o cliente envia uma requisição, o servidor *web* abre uma **conexão com o processo *FastCGI***. O servidor envia as variáveis de ambiente e entradas de dados através desta conexão;
- O processo *FastCGI* retorna a **saída através desta mesma conexão**;
- O processo *FastCGI* fecha a conexão, e a **requisição é concluída**, porém, o **processo fica “vivo”**, esperando por outra requisição do servidor *web*.

É claro que para atingir este resultado, aplicações *FastCGI* possuem uma arquitetura mais “rebuscada” que aplicações *CGI*. Por exemplo, para suportar a **multiplexação**, o servidor *web* e o processo *FastCGI* se comunicam através de mensagens. Nestas mensagens (`BEGIN_REQUEST`, `ABORT_REQUEST`, `END_REQUEST`, `PARAMS`, `STDIN` e `STDOUT`) possuímos um cabeçalho chamado `Request ID`, que é responsável por identificar a qual requisição o pacote pertence.

Essa mudança de arquitetura acaba influenciando na escrita das aplicações *web*, trazendo alterações marcantes em comparação aos programas escritos para o bom e velho *CGI*. Por exemplo, você terá que **recompilar o seu *PHP*** com a *flag* `-enable-fast-cgi`.

O site oficial do *FastCGI* possui um bom exemplo de implementação de uma **aplicação em *C* com *FastCGI***.

## O Web Server Gateway Interface

No universo *Python* começaram a aparecer diferentes formas de comunicação entre servidor e aplicação, seja com *CGI*, *FastCGI*, ***mod python*** ou até mesmo com *APIs* próprias e não padronizadas. Isso acarretou no seguinte cenário: A escolha de um *framework* influenciava diretamente na escolha do servidor *web*, e geralmente o *framework* escolhido era “incompatível” com os demais disponíveis para uso.

O *WSGI* é uma **especificação** que tem por objetivo garantir que o desenvolvedor da aplicação não se preocupe com qual servidor *web* será escolhido, bem como o profissional responsável pelo servidor *web* não se preocupe com a arquitetura escolhida pela aplicação. Uma forma “universal” de proporcionar interoperabilidade entre servidores e aplicações escritas em *Python*.

Veja um exemplo de *script Python* utilizando o protocolo *CGI*:

```
#!/usr/bin/python

from os import environ

print "Content-Type: text/html\n\n"
print "<html><body>Hello %s!</body></html>" % environ.get('REMOTE_ADDR')
```

Seguindo a especificação do *WSGI*, devemos servir nossa aplicação da seguinte maneira:

```
#!/usr/bin/python

def application(environ, start_response):
    start_response('200 OK', [('Content-Type', 'text/html')])
    return ['<html><body>Hello %s</body></html>' % environ.get('REMOTE_ADDR')]
```

Encapsulamos a nossa entrega em uma função chamada `application`, e nela possuímos dois parâmetros: `environ` e `start_response`. O primeiro é responsável por informar quais as variáveis ambientais que temos à nossa disposição. O segundo, nomeado como `start_response`, é na verdade uma função de *callback* onde informamos o *status code* e demais cabeçalhos para resposta.

Por fim, retornamos ao servidor *web* o nosso *HTML*. O servidor *web* pode “iterar” sobre a aplicação, retornando conteúdo ao usuário conforme a aplicação for retornando conteúdo para ele. Neste caso, utilizamos na resposta um tipo *sequencial*.

Agora somos capazes de servir a aplicação através de *CGI*:

```
from wsgiref.handlers import CGIHandler

CGIHandler().run(application)
```

E até mesmo *FastCGI*:

```
from flup.server.fcgi import WSGIServer

WSGIServer(application).run()
```

A biblioteca *wsgiref* implementa as especificações do *WSGI* e provê ferramentas para a comunicação entre servidores e aplicações. No segundo exemplo utilizamos a *flup*, uma biblioteca com algumas soluções *WSGI*, incluindo a possibilidade de servir aplicações *FastCGI*.

Com esse “código de cola”, basta **configurar o seu servidor Web** favorito para servir a sua aplicação.

## mod\_wsgi

Uma vez construída a interface para a sua aplicação através do padrão *WSGI*, você pode servi-la em um servidor *Apache* através do **mod\_wsgi**. Existem soluções equivalentes para outros servidores, como por exemplo, no *Nginx* temos o *NgxWSGIModule*.

Com o *modwsgi*, você não precisa de nenhum “código de cola” (como apresentado nos exemplos de *\_CGI* e *FastCGI*), basta **configurar o seu Apache** e apontar o seu script *WSGI* através da instrução `WSGIScriptAlias`:

```
<VirtualHost *:80>

    ServerName www.example.com
    ServerAlias example.com
    ServerAdmin webmaster@example.com

    DocumentRoot /usr/local/www/documents

    Alias /robots.txt /usr/local/www/documents/robots.txt
    Alias /favicon.ico /usr/local/www/documents/favicon.ico

    Alias /media/ /usr/local/www/documents/media/

    <Directory /usr/local/www/documents>
        Order allow,deny
        Allow from all
```

```
</Directory>

WSGIScriptAlias / /usr/local/www/wsgi-scripts/wsgi.py

<Directory /usr/local/www/wsgi-scripts>
Order allow,deny
Allow from all
</Directory>

</VirtualHost>
```

Uma particularidade do `mod_wsgi` é a escolha de execução no modo `daemon`, que opera de uma forma **similar ao esquema utilizado pelo *FastCGI***.

## Servidores WSGI

Você pode utilizar servidores especialmente escritos para servir as suas aplicações *WSGI*, como por exemplo o *Gunicorn*, o *uWSGI* ou até mesmo o *Tornado*. Além da versatilidade e performance, a facilidade é outra característica marcante em muitas dessas ferramentas:

```
$ gunicorn -w 4 -b 127.0.0.1:5000 wsgi:application
```

No exemplo acima, levantamos o *Gunicorn* na `porta 5000`, e reservamos `4 workers` para servir a nossa aplicação.

Além de diminuirmos a carga do servidor *web*, e ganharmos um controle mais apurado de memória e processos, ganhamos também o uso de *workers*. Por exemplo, o *Gunicorn* trabalha com *pre-fork* de *workers*, onde um processo “master” gerencia um conjunto de processos que são de fato os responsáveis por servir a sua aplicação. Ganhamos mais uma ferramenta de baixo custo para lidar com concorrência.

Servidores *WSGI* conseguem servir as aplicações sem o auxílio de um *Apache* ou *Nginx*, mas uma prática muito comum hoje em dia é, “na frente” de um *Gunicorn* (por exemplo), termos um *Nginx* servindo estáticos, fazendo *caching* e “aguentando porrada”, enquanto que o servidor *WSGI* está totalmente focado em servir o conteúdo dinâmico. O servidor *web* acaba fazendo uma espécie de **proxy reverso** e até mesmo servindo como **balanceador**.

A comunicação entre servidores pode ser feita via *TCP* ou *socket*. Isso nos dá uma série de vantagens, que vão desde a facilidade em **escalar** e distribuir, até o *restart* individual de serviços (por exemplo, se a sua aplicação travar, você pode reiniciar apenas o servidor *WSGI* e não perder o *caching* do servidor *web*).

Um exemplo muito interessante de uso de servidores *WSGI* é fazendo **deploy de aplicações Python para o Heroku**. Configurar um servidor *Nginx* para se comunicar com servidores *WSGI* também é relativamente simples.

## Considerações finais

Um assunto muito interessante e que pretendo explorar mais aqui no *blog*, principalmente em relação a processos e *workers*.

Servir aplicações *Python* para a *web* é algo relativamente simples, limpo e elegante. Através do *WSGI*, escalar aplicações passou a ser algo quase trivial, que demanda pouco esforço. Combiná-los com o *Nginx* dão mais fôlego a sua aplicação (principalmente se estivermos falando do *uWSGI* ou **gevent**), e com um **sistema de provisionamento automático** podem facilitar e muito o seu trabalho de infraestrutura quando o consumo se tornar um problema.

## Referências

- [\*Django Documentation – How to use Django with FastCGI, SCGI, or AJP\*](#)
- [\*FastCGI – The Forgotten Treasure\*](#)
- [\*FastCGI – A High-Performance Web Server Interface\*](#)
- [\*Gunicorn – Python WSGI HTTP Server for Unix\*](#)
- [\*irt.org – Speed Thrills: CGI Please... and Fast!\*](#)
- [\*James Marshal – CGI realmente fácil\*](#)
- [\*modwsgi – Python WSGI adapter module for Apache\*](#)
- [\*PEP 333 – Python Web Server Gateway Interface\*](#)
- [\*Python Documentation – HOWTO Use Python in the web\*](#)
- [\*W3C – Common Gateway Interface\*](#)
- [\*Wikipedia – Common Gateway Interface\*](#)



- *Wikipedia – FastCGI*
- *Wikipedia – Web Server Gateway Interface*
- *XML.com – Introducing WSGI: Python's Secret Web Weapon*

[desenvolvimento-web](#)[infraestrutura](#)[python](#)[cgi](#)[fastcgi](#)[wsgi](#)[gunicorn](#)



## 18 Comentários

[1 Entrar ▼](#)

G

Participe da discussão...

FAZER LOGIN COM

OU REGISTRE-SE NO DISQUS [?](#) 4

Compartilhar

Mais votados

Mais recentes

Mais antigos

**Vanderson**

6 anos atrás

Cara, muito bom, cai aqui pelo google em uma pesquisa sobre flask e wsgi, o conteúdo é bem didático e simples.

1 0 Responder • Compartilhar ›

**Carlos Moreira**

3 anos atrás

Nem era o que eu procurava, mas li tudo. Muito bom.

0 0 Responder • Compartilhar ›

**Klaus Peter Laube** Mod

→ Carlos Moreira

3 anos atrás

Fico feliz em ajudar, Carlos :)

0 0 Responder • Compartilhar ›

T

**Thiago P. Barros**

4 anos atrás

Com relação a Variaveis de ambiente .....no trecho abaixo retirado do material que voce postou, Amigo !!.....o que está vindo para aplicação cgi, os IPs dos visitantes ou Ip do servidor Web ....quando diz que é preenchido pelo servidor ou é recebido e repassado IP de quem visita a página (Visitantes), Amigo ? .....Parabéns muito bom :)

```
=====
char *addr, *method, *query_string;

addr = getenv("REMOTE_ADDR");
method = getenv("REQUEST_METHOD");
query_string = getenv("QUERY_STRING");
=====
```

0 0 Responder • Compartilhar ›

**Klaus Peter Laube** Mod

→ Thiago P. Barros

4 anos atrás



4 anos atrás

Olá, Thiago.

O `REMOTE\_ADDR` é, em teoria, a informação do endereço IP do cliente. O que pode acontecer é que tanto cliente quanto você (servidor) estejam atrás de um proxy ou load balancer. Nesse caso, o uso do cabeçalho X-Forwarded-For (<https://en.wikipedia.org/wi...>) pode ser necessário.

Obrigado por participar.

3 0 Responder • Compartilhar ›

**Vander Vieira**

4 anos atrás

Genial! Teu jeito de explicar é muito cativante e enriquecedor. Obrigado!

0 0 Responder • Compartilhar ›



Esse comentário foi apagado.

**Klaus Peter Laube** Mod

➔ Guest

6 anos atrás

Fico feliz que o artigo tenha sido útil, Lucas! Obrigado pelo feedback.

0 0 Responder • Compartilhar ›

T

**Thiago P. Barros**

➔ Klaus Peter Laube

4 anos atrás

Parabéns !!....excelente conteúdo !!....ficou muito didático e simples a explicação .....Abraços fera!!

0 0 Responder • Compartilhar ›

**Luiz Almeida Junior**

6 anos atrás

Muito bom.

0 0 Responder • Compartilhar ›

**Wellington Gomes dos Santos**

6 anos atrás

Obrigado.

0 0 Responder • Compartilhar ›

**William Mendes**

9 anos atrás

Muito bom artigo. Gostaria de tirar uma dúvida com vocês. É possível criar shell scripts com FastCGI, tal como fazemos com CGI, com as devidas diferenças é claro?

0 0 Responder • Compartilhar ›

**Klaus Peter Laube** Mod

➔ William Mendes

9 anos atrás



7 anos atrás

William, achei a pergunta capciosa e dei uma googlada sobre. A resposta curta é: Sim! É possível. Mas por causa da natureza do Shell, é mais válido (e simples) você utilizar CGI direto. Mais detalhes, você pode conferir nesta thread do StackOverflow: <http://stackoverflow.com/qu...>

0 0 Responder • Compartilhar ›

**Flávio**

9 anos atrás

Muito bom artigo!

0 0 Responder • Compartilhar ›

**R****Renato**

9 anos atrás

Excelente.

0 0 Responder • Compartilhar ›

**Renato Souza**

9 anos atrás

Muito bom. Aprendendo a usar o CGI agora.

0 0 Responder • Compartilhar ›

**Klaus Peter Laube** Mod ➔ Renato Souza

9 anos atrás

Obrigado, Renato!

0 0 Responder • Compartilhar ›

**Wagner Junior**

10 anos atrás

Sei que o post já tem um tempo, mas, gostaria de deixar meus parabéns pela didática e pela maestria como abordou o assunto.

0 0 Responder • Compartilhar ›

**Klaus Peter Laube** Mod ➔ Wagner Junior

10 anos atrás

Wagner, muito obrigado! Fico feliz que tenha gostado.

0 0 Responder • Compartilhar ›

[Inscreva-se](#)[Privacidade](#)[Política de Proteção de Dados](#)

O conteúdo desse blog está sob a licença Creative Commons Attribution.

Fork me on Github.