FastCGI

Rob Saccoccio
robs@InfiniteTechnology.com

Overview

- Why FastCGI?
- The Protocol
- The API
- Characteristics
- mod_fastcgi
- Summary

Dynamic Content Rules

- Sites are handling unprecedented loads and generating more dynamic content
- This highlights the limitations of the Common Gateway Interface (CGI) and has spawned many solutions
- FastCGI was developed by Open Market in ’96 as an open solution.

CGI Weaknesses

- Slow
- Inherently bad for tasks that require persistent data
- Doesn’t scale

CGI Strengths

- Easy to use and understand
- Language, platform, and server independent
- Server-safe (CGIs run in a separate process)
- Open standard

FastCGI Design Goals

- Speed!
- Eliminate CGI's weaknesses
  - Scalable
  - Persistent
- Build on CGI's strengths
  - Simple, Open Standard with an easy migration path
  - Server, Language, & OS independent
  - Server Isolation

Speed!

- Common Gateway Interface (CGI) requires a fork/exec per request
- Very large and interpreted applications can be slow to start
- Initialization such as logging on to databases, connecting to remote machines, or parsing static files can be slow

Persistence

- HTTP is stateless
- Persistence eliminates per-request fork/exec and application initialization delays
- Persistence allows data caching (e.g. lookups, responses, calculations)
- Web based applications, targeted marketing, shopping carts

Server Independence

- An ideal solution isn’t server specific
- Server APIs
  - Apache, ISAPI, NSAPI, mod_perl, etc.
  - Bind the solution to the server
  - Are generally complex or undocumented
  - Embed the application in the server itself

Server Independence Cont'd

- Application Servers
  - SilverStream, HAHT, ColdFusion, etc.
  - Are often server independent (possibly via FastCGI), but bind a solution to the application server and its development environment
  - Commercial application servers can represent a non-trivial $ outlay

Language & OS Independence

- An ideal solution doesn’t require the use of a specific language
- FastCGI application development libraries in C, C++, Perl, Python, Java, TCL
- The language/library provides the OS independence

CGI-like

- Easy to use and understand
- Leverage existing knowledge, tools, and installation base
- Allow simple migration from CGI
- Isolated from server
- Ubiquitous?

Scalable

- N-tier, load distribution
- Mix-n-match OSs

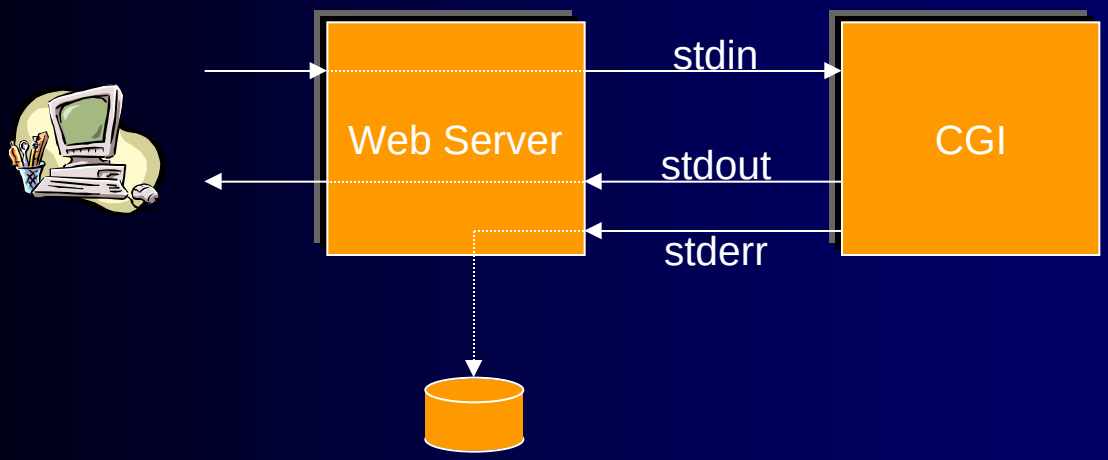
Open Standard

- Non-proprietary, no royalties
- Anyone can implement it
- Anyone can improve it

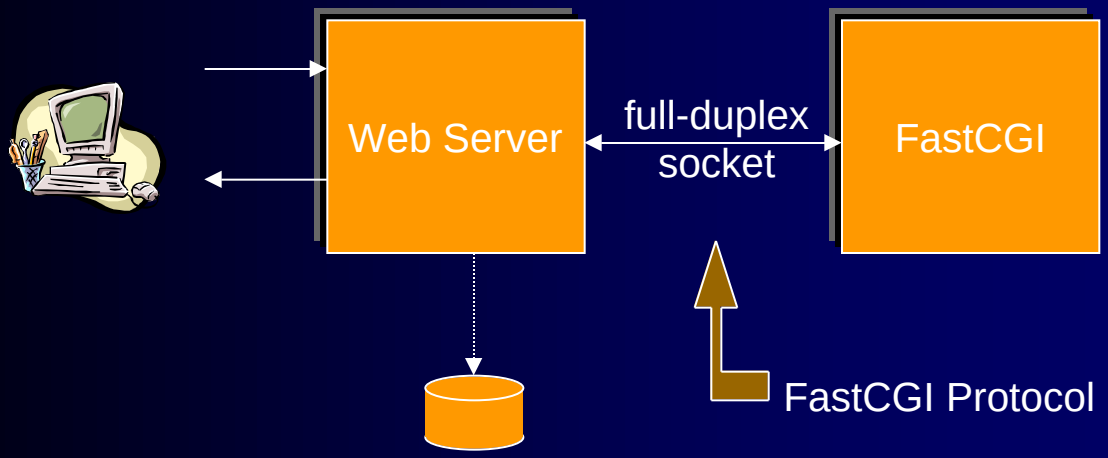
FastCGI is..

- A language and server independent, scalable, open extension to CGI that provides high performance and persistence
- A protocol for data interchange between a web server and a FastCGI application
- The set of libraries that implement the protocol

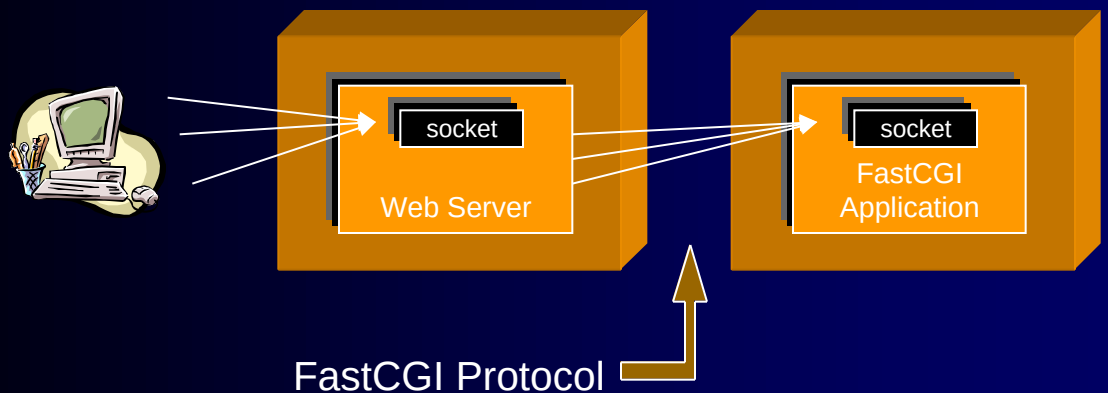
Web Server - CGI Relationship



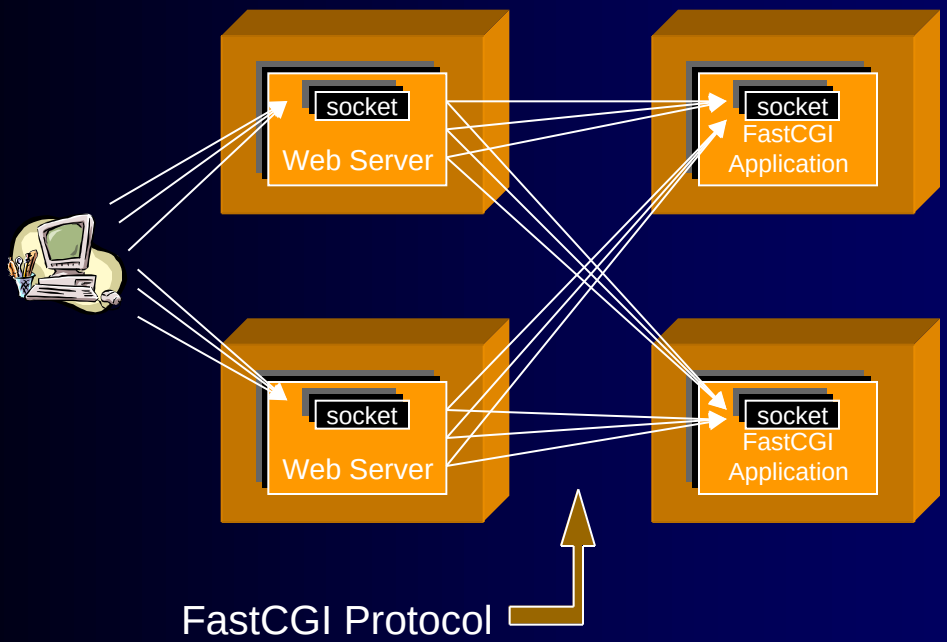
Web Server - FastCGI Relationship



Backend FastCGI Server



Piles of Servers



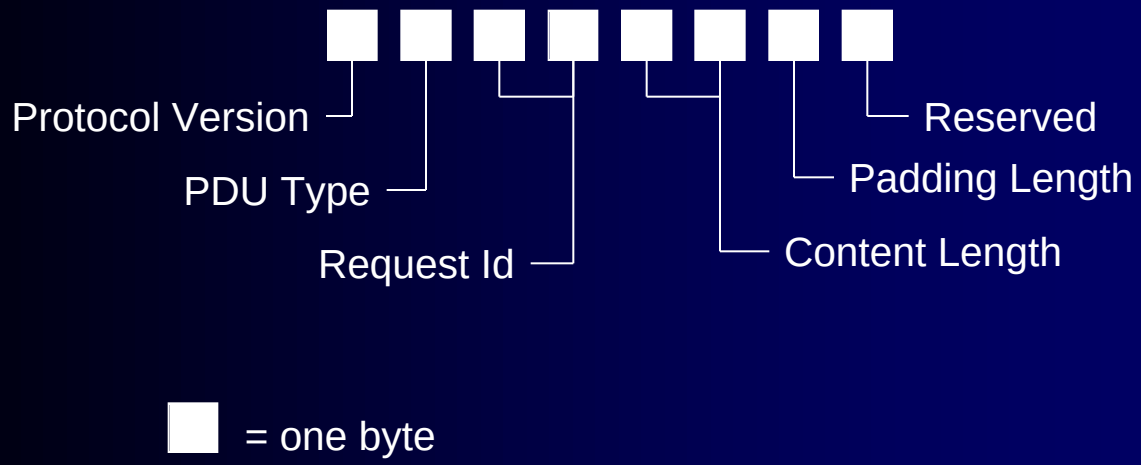
FastCGI Protocol

- All data is wrapped in the protocol
- A simple standard header precedes every Protocol Data Unit (PDU)
- The header describes the type of data and its length

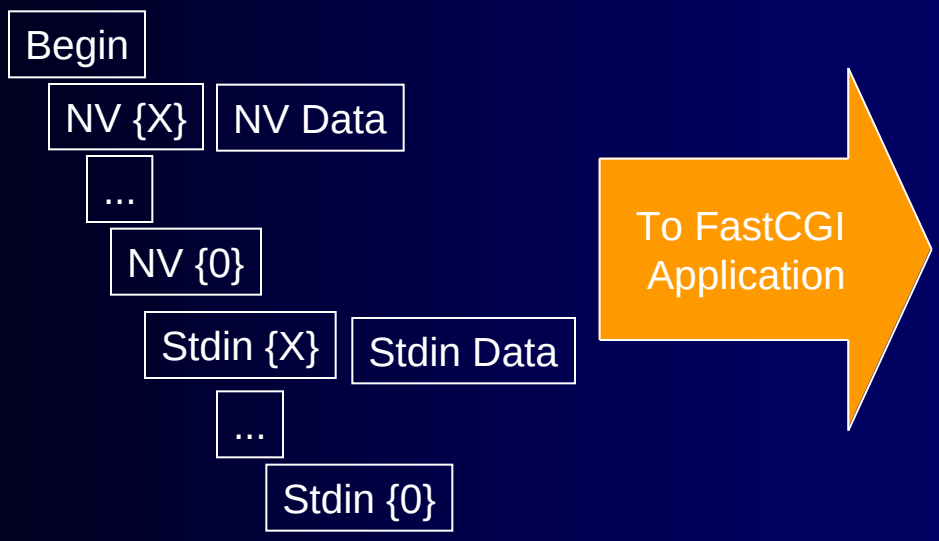
PDU Types

- Begin Request
- Name-Value Stream
- Stdin Stream
- Stdout Stream
- Stderr Stream
- End Request

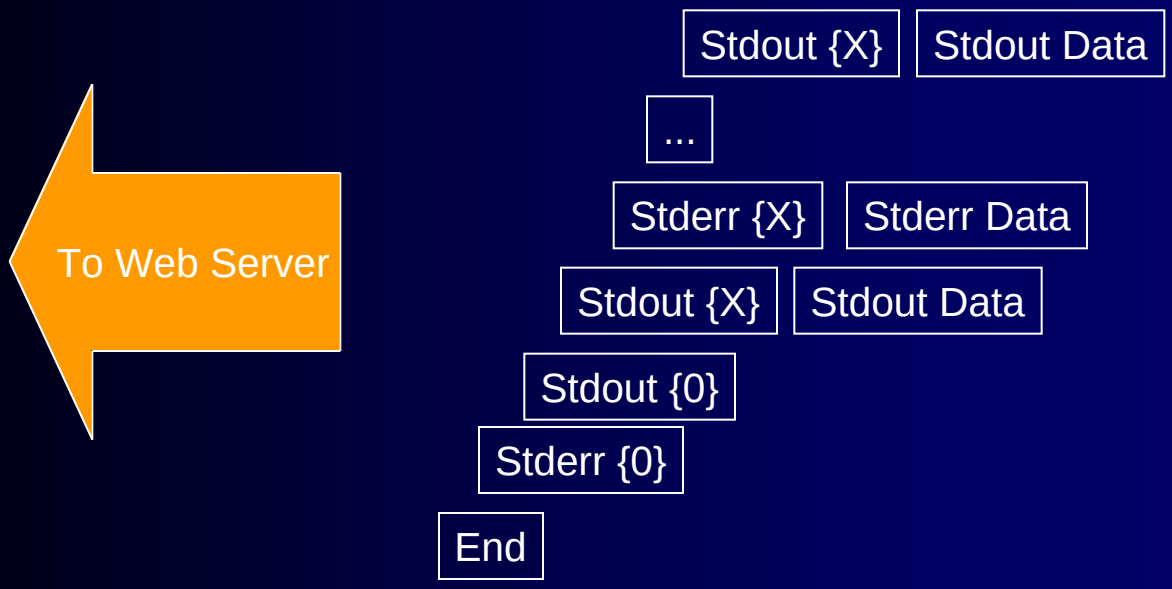
FastCGI Header



Typical PDU Flow - To FastCGI Application



Typical PDU Flow - From FastCGI Application



FastCGI Roles

- Responder
- Authorizer
- Filter

Responder Role

- The fundamental FastCGI role
- Functionally identical to CGI
- Supported by all FastCGI capable servers

Authorizer Role

- Provides a means of controlling access to a site, a page, or something in between
- Typically, this involves some form of authentication, but this isn’t required
- It has server dependent significance

Filter Role

- Allows “processing” of a file before it is sent
- Intended to support:
  - Format conversions
  - Dynamic documents (embedded code)
  - Applying templates: e.g. headers, footers, backgrounds
- Conceptually this could support dynamic content chaining, but without server support has limited utility

FastCGI Applications

- Application Organization

```
Initialization
Response Loop {
    Response Handler
}
```

- Eliminate leaks
- Review 3rd party libraries/modules
- Compress functionality

Perl FCGI API

- accept() – Accepts a new FastCGI connection request, implicitly finishes the current request
- flush() – Flushes output buffers
- finish() – Finishes the current request

Hello World - Perl

```
use FCGI;

$count = 0;

while (FCGI::accept() == 0) {
    print "Content-type: text/html\r\n“,
        "\r\n“,
        "<h1>Hello World</h1>\n“,
        "Request “, ++$count,
        “ from server “, $ENV{'SERVER_NAME'};
}
```

Hello World – Perl, CGI.pm

```
use CGI::Fast qw(:standard);

$count = 0;

while (new CGI::Fast) {
    print header,
        h1(“Hello World"),
        "Request “, ++$count,
        “ from server “, $ENV{'SERVER_NAME'};
}
```

How Fast?

- What’s the fork/exec mean to performance?
- Compare a tiny fcgi_stdio application in both CGI and FastCGI mode.

```
#include "fcgi_stdio.h“
main(void) {
    while (FCGI_Accept() == 0) {
        printf("Content-type: text/html\r\n“
               "\r\n“
               "<h1>FastCGI Hello</h1>");
    }
}
```

Pretty Useless Statistics (PUS)

- Using Apache's ab (Apache benchmark) program with 20 simultaneous requests, 100/1000 total requests
- Data represents the number of requests/sec
- tiny1 sleeps for 1 sec in the middle of handling
- 1tiny sleeps for one second at initialization

	tiny.c	1tiny.c	tiny1.c
Static File	(968)		
CGI	184	13	16
FastCGI (1 process)	516	409	1

Considerations

- How long will a typical request take? Are there any external dependencies?
- What type of load will the application have?
- How much data is sent to/from the client?
- Run as many processes/threads as you need
- Don’t send content if a redirect will do (e.g. images)

mod_fastcgi

- Authorizer and Responder Roles
- Process Management
- Static and dynamic servers
- Suexec wrapper support

mod_fastcgi Directives

- FastCgiServer
- FastCgiExternalServer
- FastCgiConfig
- FastCgiIpcDir
- FastCgiSuexec
- FastCgiAuthenticator
- FastCgiAuthorizer
- FastCgiAccessChecker

FastCgiServer

- `FastCgiServer` *`filename [option …]`*
- Defines a FastCGI application to be started and managed by mod_fastcgi
- Optional arguments for identifying:
  - Connection characteristics - socket type, backlog depth, blocking or non-blocking connect(), timeouts, flush, passed headers
  - Application invocation: process priority, process count, environment, start and restart delays

FastCgiExternalServer

- `FastCgiExternalServer` *`filename -host host:port [options..]`*
  `FastCgiExternalServer` *`filename -socket filename [options..]`*
- Defines a FastCGI application that is managed external to mod_fastcgi
- Requires identifying the whereabouts of the socket
- Optional arguments for identifying:
  - Connection characteristics - blocking or non-blocking connect(), timeouts, flush, passed headers

FastCgiConfig

- `FastCgiConfig` *`[options..]`*
- Controls the characteristics of “dynamic” handling
- Determines how and when dynamic FastCGI applications are spawned and killed based on demand (response delay)
- Optional arguments for identifying:
  - Connection characteristics – backlog depth, blocking or non-blocking connect(), timeouts, flush, start delay, passed headers
  - Application invocation – process priority, process counts, environment, start and restart delays, auto update
  - Process Manager characteristics – gain, intervals, thresholds,

FastCgiIpcDir

- `FastCgiIpcDir` *`directory`*
- Specifies a directory for the Unix Domain sockets
- The default is `/tmp/fcgi` - this is sometimes problematic (cron jobs doing cleanup)

FastCgiSuexec

- `FastCgiSuexec` *`On`* | *`Off`* | *`filename`*
- Enables the use of a wrapper that is used to invoke the FastCGI applications
- Typically the wrapper is “suid root” and is used to run the applications with different a user/group specified by a convention or file
- Works in conjunction with Apache’s suexec User and Group directives

Auth Directives

- FastCgiAuthenticator, FastCgiAuthorizer, FastCgiAccessChecker share their syntax but function in different Apache phases
- The authorizer can be dynamic or static
- The –compat option forces compliance with the FastCGI specification (less variables are made available to handlers)

Auth Directives Cont'd.

- All are considered FastCGI Authorizers, but the FCGI_APACHE_ROLE variable is set to differentiate them
- Each of the directives also have an associated Authoritative directive that allows other modules a crack at the phase

FastCgiAccessChecker

- `FastCgiAccessChecker` *`filename [-compat]`*
- AccessCheckers determine whether to grant access based on “other” data (e.g. client IP, presence of a valid cookie, time of day, etc.)
- Access is independent of and takes place before user authentication and authorization

FastCgiAuthenticator

- `FastCgiAuthenticator` *`filename [-compat]`*
- FastCgiAuthenticators authenticate users by name and password
- Uses HTTP Basic authentication
- The user's password is made available in the REMOTE_PASSWORD variable
- Can be used with the core authorizers directives, such as "`requires valid-user`"

FastCgiAuthorizer

- `FastCgiAuthorizer` *`filename [-compat]`*
- FastCgiAuthorizers determine whether to grant an authenticated user access
- Can be used in conjunction with any authenticator
- The user's password is *not* passed to the FastCGI application

Example Configuration

```
# Config used for dynamic applications
FastCgiConfig -initial-env PATH=/usr/bin -maxClassProcesses 10 -startDelay 1

# Start/manage 10 copies of my-fcgi-app
FastCgiServer fcgi-bin/my-fcgi-app -processes 10

# Send requests for my-ext-fcgi-app to db-server for handling
FastCgiExternalServer fcgi-bin/my-ext-fcgi-app -host db-server:5018

<Directory fcgi-bin/>

    # All requests in this directory are FastCGI
    SetHandler fastcgi-script

    # Dynamic FastCGI applications require this flag to be on
    Options ExecCGI

</Directory>
```

Example Configuartion Cont’d.

```
<Directory htdocs/killer-app>

    # All requests ending with these extensions are FastCGI
    AddHandler fastcgi-script .fcg .fcgi .fpl

    # Identify it as an access checker (it looks for a valid cookie)
    FastCgiAccessChecker htdocs/killer-app/authorizer

</Directory>

# Start/manage the kill-app authorizer
FastCgiServer htdocs/killer-app/authorizer

<Directory htdocs/company-policy>

    # Pass all requests through the (dynamic FastCGI) sanitizer
    Action default-handler fcgi-bin/sanitizer.fcgi

</Directory>
```

Reasons for Using FastCGI

- No learning curve, everyone knows CGI
- Same binary works with Netscape, IIS, OpenMarket, and Apache
- Same code works across SPARC and Intel
- Allows mix-n-match OS/servers
- Choice of standard programming languages
- Flexible deployment
- No vendor tie or proprietary languages

Which Solution?

- Whether to use FastCGI, an embedded interpreter, server API, or application server depends on the project, personal experiences and preferences.
- They’re all fast and persistent.

Summary

- FastCGI is a protocol and the set of libraries that implements it
- FastCGI provides a fairly low level toolkit for developing fast, persistent, and portable solutions

FastCGI Servers

- Apache - mod_fastcgi (free)
  http://www.fastcgi.com/
- Zeus - http://www.zeustech.net/
- Microsoft & Netscape – FastServ plug-in
  http://www.fastengines.com/

Thanks

- Sven Verdoolaege
- Jonathan Roy
- Bill Snapper
- Infinite Technology, Inc.

Links..

- The FastCGI Home Page
  - http://www.fastcgi.com
- These slides
  - http://www.fastcgi.com/docs/OpenSource99