CodeCrafters

Build your own DNS server

Learn about the DNS protocol, DNS record types and more.

Start Building

DNS is a protocol used to resolve domain names to IP addresses. In this challenge, you'll build a DNS server that's capable of responding to basic DNS queries.

Along the way you'll learn about the DNS protocol, DNS packet format, DNS record types, UDP servers and more.

Stages

Setup UDP server	VERY EASY
Write header section	MEDIUM
Write question section	MEDIUM
Write answer section	EASY
Parse header section	HARD
Parse question section	EASY
Parse compressed packet	MEDIUM
Forwarding Server	MEDIUM

“ There are few sites I like as much that have a step by step guide. The real-time feedback is so good, it's creepy!

Ananthalakshmi Sankar
Automation Engineer at Apple

“I think the instant feedback right there in the git push is really cool. Didn't even know that was possible!

Patrick Burris
Senior Software Developer, CenturyLink

Programming challenges for seasoned developers.

CHALLENGES

Git

Redis

Docker

SQLite

Grep

BitTorrent

HTTP Server

DNS Server

SUPPORT

Docs

Status

COMPANY

About

Changelog

LEGAL

Terms

Privacy

‹ back next › VIP ✕

Write header section #TZ1 In-progress

Instructions Code Examples Forum

How to pass this stage

In this stage, you'll implement your own solution. Unlike stage 1, your repository doesn't contain commented code to pass this stage.

98% of users who attempt this stage complete it.

Step 1: Read instructions Mark as complete

The "Your Task" card below contains a description of what you need to implement to pass tests.

We also recommend taking a look at:

- The Code Examples tab, which contains code examples from other users.

Mark this step as complete once you've read the instructions and are ready to implement your solution.

Mark as complete

Step 2: Implement solution

Step 3: Run tests

TEST RUNNER: Ready to run tests... View Inst

Ready to run tests... Show logs

Instructions   Code Examples   Forum   Stage #TZ1 In-progress

Your Task In-progress MEDIUM

All communications in the DNS protocol are carried in a single format called a "message". Each message consists of 5 sections: header, question, answer, authority, and an additional space.

In this stage, we'll focus on the "header" section. We'll look at the other sections in later stages.

Header section structure

The header section of a DNS message contains the following fields: (we've also included the values that the tester expects in this stage)

Field	Size	Description
Packet Identifier (ID)	16 bits	A random ID assigned to query packets. Response packets must reply with the same ID. **Expected value**: 1234.
Query/Response Indicator (QR)	1 bit	1 for a reply packet, 0 for a question packet. **Expected value**: 1.
Operation Code (OPCODE)	4 bits	Specifies the kind of query in a message. **Expected value**: 0.
Authoritative Answer (AA)	1 bit	1 if the responding server "owns" the domain queried, i.e., it's authoritative. **Expected value**: 0.
Truncation (TC)	1 bit	1 if the message is larger than 512 bytes. Always 0 in UDP responses. **Expected value**: 0.
Recursion Desired (RD)	1 bit	Sender sets this to 1 if the server should recursive resolve this query, 0 otherwise.

Ready to run tests... Show logs

Instructions | Code Examples | Forum | Stage #TZ1 | In-progress

		Expected value: 0.
Recursion Desired (RD)	1 bit	Sender sets this to 1 if the server should recursively resolve this query, 0 otherwise. **Expected value**: 0.
Recursion Available (RA)	1 bit	Server sets this to 1 to indicate that recursion is available. **Expected value**: 0.
Reserved (Z)	3 bits	Used by DNSSEC queries. At inception, it was reserved for future use. **Expected value**: 0.
Response Code (RCODE)	4 bits	Response code indicating the status of the response. **Expected value**: 0 (no error).
Question Count (QDCOUNT)	16 bits	Number of questions in the Question section. **Expected value**: 0.
Answer Record Count (ANCOUNT)	16 bits	Number of records in the Answer section. **Expected value**: 0.
Authority Record Count (NSCOUNT)	16 bits	Number of records in the Authority section. **Expected value**: 0.
Additional Record Count (ARCOUNT)	16 bits	Number of records in the Additional section. **Expected value**: 0.

The header section is always 12 bytes long. Integers are encoded in big-endian format.

You can read more about the full DNS packet format on Wikipedia, or in RFC 1035. This link is a good tutorial that walks through the DNS packet format in detail.

Ready to run tests... Show logs

Instructions Code Examples Forum Stage #TZ1 In-progress

Just like in the previous stage, the tester will execute your program like this:

```
./your_server.sh
```

It'll then send a UDP packet (containing a DNS query) to port 2053. Your program will need to respond with a DNS reply packet that contains the header information described above.

We recommend creating an internal structure for a "DNS message" in your code, as we will build on this in later stages.

Note: The tester sends an extra packet at the start of testing each stage. You can ignore it.

View Code Examples

Collapse ↑

Hints

Filter by Rust ✓

Coding Blade

Write Preview

Found an interesting resource? Share it with the community.

Markdown supported.

Ready to run tests... Show logs

Instructions   Code Examples   Forum   Stage #TZ1 In-progress

View Code Examples

Collapse ↑

Hints

Filter by Rust ✓

Write   Preview

Found an interesting resource? Share it with the community.

Markdown supported.   Comment

EpocDotFr 6 months ago

Test your implementation by running `dig @127.0.0.1 -p 2053 +noedns codecrafters.io` .

16   reply

theredditbandit 6 months ago

To test this locally you can use the `dig` command as such

```
dig @127.0.0.1 -p 2053 example.com
```

7   reply

Ready to run tests... Show logs

‹ back next › VIP ✕

Write question section #BF2 Pending

Complete previous stages to gain access to this stage.

Instructions Code Examples Forum

Your Task Pending MEDIUM

In this stage, you'll extend your DNS server to respond with the "question" section, the second section of a DNS message.

Question section structure

The question section contains a list of questions (usually just 1) that the sender wants to ask the receiver. This section is present in both query and reply packets.

Each question has the following structure:

- **Name**: A domain name, represented as a sequence of "labels" (more on this below)
- **Type**: 2-byte int; the type of record (1 for an A record, 5 for a CNAME record etc., full list here)
- **Class**: 2-byte int; usually set to `1` (full list here)

Section 4.1.2 of the RFC covers the question section format in detail. Section 3.2 has more details on Type and class.

Domain name encoding

Domain names in DNS packets are encoded as a sequence of labels.

Labels are encoded as `<length><content>`, where `<length>` is a single byte tha

Ready to run tests... Show logs

Instructions Code Examples Forum Stage #BF2 Pending

Labels are encoded as `<length><content>`, where `<length>` is a single byte that specifies the length of the label, and `<content>` is the actual content of the label. The sequence of labels is terminated by a null byte (`\x00`).

For example:

- `google.com` is encoded as `\x06google\x03com\x00` (in hex: `06 67 6f 6f 67 6c 65 03 63 6f 6d 00`)
  - `\x06google` is the first label
    - `\x06` is a single byte, which is the length of the label
    - `google` is the content of the label
  - `\x03com` is the second label
    - `\x03` is a single byte, which is the length of the label
    - `com` is the content of the label
  - `\x00` is the null byte that terminates the domain name

Just like in the previous stage, the tester will execute your program like this:

```
./your_server.sh
```

It'll then send a UDP packet (containing a DNS query) to port 2053. Your program will need to respond with a DNS reply packet that contains the question section described above (along with the header section from the previous stage).

Here are the expected values for the question section:

Field	Expected value

Ready to run tests... Show logs

Instructions Code Examples Forum

Stage #BF2 Pending

Field	Expected value
Name	`\x0ccodecrafters\x02io` followed by a null byte (that's `codecrafters.io` encoded as a label sequence)
Type	1 encoded as a 2-byte big-endian int (corresponding to the "A" record type)
Class	1 encoded as a 2-byte big-endian int (corresponding to the "IN" record class)

Make sure to update the `QDCOUNT` field in the header section accordingly, and remember to set the id to `1234`.

View Code Examples

Collapse ↑

Hints

Filter by Rust ✓

Write Preview

Found an interesting resource? Share it with the community.

Markdown supported. Comment

Ready to run tests... Show logs

‹ back next › VIP ✕

Write answer section #XM2 Pending

Complete previous stages to gain access to this stage.

Instructions </> Code Examples Forum

Your Task Pending

EASY

In this stage, you'll extend your DNS server to respond with the "answer" section, the third section of a DNS message.

Answer section structure

The answer section contains a list of RRs (Resource Records), which are answers to the questions asked in the question section.

Each RR has the following structure:

Field	Type	Description
Name	Label Sequence	The domain name encoded as a sequence of labels.
Type	2-byte Integer	`1` for an A record, `5` for a CNAME record etc., full list here
Class	2-byte Integer	Usually set to `1` (full list here)
TTL (Time-To-Live)	4-byte Integer	The duration in seconds a record can be cached before requerying.
Length (`RDLENGTH`)	2-byte Integer	Length of the RDATA field in bytes.
Data (`RDATA`)	Variable	Data specific to the record type.

Ready to run tests... Show logs

Instructions   Code Examples   Forum   Stage #XM2 Pending

(`RDLENGTH`)		
Data (`RDATA`)	Variable	Data specific to the record type.

Section 3.2.1 of the RFC covers the answer section format in detail.

In this stage, we'll only deal with the "A" record type, which maps a domain name to an IPv4 address. The RDATA field for an "A" record type is a 4-byte integer representing the IPv4 address.

Just like in the previous stage, the tester will execute your program like this:

```
./your_server.sh
```

It'll then send a UDP packet (containing a DNS query) to port 2053.

Your program will need to respond with a DNS reply packet that contains:

- a header section (same as in stage #2)
- a question section (same as in stage #3)
- an answer section (**new in this stage!**)

Your answer section should contain a single RR, with the following values:

Field	Expected Value
Name	`\x0ccodecrafters\x02io` followed by a null byte (that's `codecrafters.io` encoded as a label sequence)
Type	`1` encoded as a 2-byte big-endian int (corresponding to the "A" record type)
Class	`1` encoded as a 2-byte big-endian int (corresponding to the "IN" record class)

Ready to run tests... Show logs

Instructions · Code Examples · Forum · Stage #XM2 · Pending

Name	`\x0ccodecrafters\x02io` followed by a null byte (that's `codecrafters.io` encoded as a label sequence)
Type	`1` encoded as a 2-byte big-endian int (corresponding to the "A" record type)
Class	`1` encoded as a 2-byte big-endian int (corresponding to the "IN" record class)
TTL	Any value, encoded as a 4-byte big-endian int. For example: `60` .
Length	`4` , encoded as a 2-byte big-endian int (corresponds to the length of the RDATA field)
Data	Any IP address, encoded as a 4-byte big-endian int. For example: `\x08\x08\x08\x08` (that's `8.8.8.8` encoded as a 4-byte integer)

Make sure to update the `ANCOUNT` field in the header section accordingly, and remember to set the id to `1234` .

View Code Examples

Collapse ↑

Hints

Filter by Rust ✓

Write Preview

Found an interesting resource? Share it with the community.

Ready to run tests... Show logs

‹ back next › VIP ✕

Parse header section #UC8 Pending

Complete previous stages to gain access to this stage.

Instructions Code Examples Forum

Your Task Pending HARD

Up until now, we were ignoring the contents of the DNS packet that we received and hardcoding `1234` as the ID in the response. In this stage, you'll have to parse the DNS packet that you receive and respond with the same ID in the response. You'll also need to set some other fields in the header section.

Just like the previous stage, the tester will execute your program like this:

```
./your_server.sh
```

It'll then send a UDP packet (containing a DNS query) to port 2053.

Your program will need to respond with a DNS reply packet that contains a header section with the following values:

Field	Size	Expected value
Packet Identifier (ID)	16 bits	Mimic the 16 bit packet identifier from the request packet sent by tester
Query/Response Indicator (QR)	1 bit	1
Operation Code (OPCODE)	4 bits	Mimic the OPCODE value sent by the tester
Authoritative Answer (AA)	1 bit	0

Ready to run tests... Show logs

Instructions Code Examples Forum Stage #UC8 Pending

Operation Code (OPCODE)	4 bits	Mimic the OPCODE value sent by the tester
Authoritative Answer (AA)	1 bit	0
Truncation (TC)	1 bit	0
Recursion Desired (RD)	1 bit	Mimic the RD value sent by the tester
Recursion Available (RA)	1 bit	0
Reserved (Z)	3 bits	0
Response Code (RCODE)	4 bits	0 (no error) if OPCODE is 0 (standard query) else 4 (not implemented)
Question Count (QDCOUNT)	16 bits	Any valid value
Answer Record Count (ANCOUNT)	16 bits	Any valid value
Authority Record Count (NSCOUNT)	16 bits	Any valid value
Additional Record Count (ARCOUNT)	16 bits	Any valid value

The tester will not check what follows the header section as long as it is a valid DNS packet.

Note: Your code will still need to pass tests for the previous stages. You shouldn't need to hardcode `1234` as the request ID anymore since the tester sends `1234` as the ID in the previous stages. As long as you implement this stage correctly, your code should automatically pass the previous stages as well.

View Code Examples

Ready to run tests... Show logs

‹ back next › VIP ✕

Parse question section #HD8 Pending

Complete previous stages to gain access to this stage.

Instructions Code Examples Forum

Your Task Pending EASY

In this stage you'll extend your DNS server to parse the question section of the DNS message you receive.

Just like the previous stage, the tester will execute your program like this:

```
./your_server.sh
```

It'll then send a UDP packet (containing a DNS query) to port 2053 that contains a question section as follows:

Field	Value sent by the tester
Name	A random domain encoded as a label sequence (refer to stage #3 for details)
Type	`1` encoded as a 2-byte big-endian int (corresponding to the "A" record type)
Class	`1` encoded as a 2-byte big-endian int (corresponding to the "IN" record class)

The question type will always be `A` for this stage and the question class will always be `IN`. So your parser only needs to account for those record types for now.

Your program will need to respond with a DNS reply packet that contains:

- a header section (same as in stage #5)

Ready to run tests... Show logs

Your program will need to respond with a DNS reply packet that contains:

- a header section (same as in stage #5)
- a question section (**new in this stage**)
- an answer section (**new in this stage**)

Expected values for the question section:

Field	Expected value
Name	Mimic the domain name (as label sequence)
Type	`1` encoded as a 2-byte big-endian int (corresponding to the "A" record type)
Class	`1` encoded as a 2-byte big-endian int (corresponding to the "IN" record class)

Expected values for the answer section:

Field	Expected Value
Name	Mimic the domain name (as label sequence)
Type	`1` encoded as a 2-byte big-endian int (corresponding to the "A" record type)
Class	`1` encoded as a 2-byte big-endian int (corresponding to the "IN" record class)
TTL	Any value, encoded as a 4-byte big-endian int. For example: `60`.
Length	`4`, encoded as a 2-byte big-endian int (corresponds to the length of the RDATA field)
Data	Any IP address, encoded as a 4-byte big-endian int. For example: `\x08\x08\x08\x08` (that's `8.8.8.8` encoded as a 4-byte integer)

View Code Examples

Ready to run tests... Show logs

back next VIP

Parse compressed packet #YC9 Pending

Complete previous stages to gain access to this stage.

Instructions Code Examples Forum

Your Task Pending

MEDIUM

In this stage we will parse the DNS question section which has compressed the question label sequences. You will be sent multiple values in the question section and you have to parse the queries and respond with the same question section (no need for compression) in the response along with answers for them. As for the answer section, respond with an `A` record type for each question. The values for these A records can be anything of your choosing.

As we already know how the Question Section and Answer Section look like from the previous stages, we will just give high level details of the packet that you are sent and what the tester expects.

Here is what the tester will send you:

```
| ----------------------------------------- |
| Header                                    |
| ----------------------------------------- |
| Question 1 (un-compressed label sequence) |
| ----------------------------------------- |
| Question 2 (compressed label sequence)    |
| ----------------------------------------- |
```

What the tester expects in response:

Ready to run tests... Show logs

Instructions Code Examples Forum Stage #YC9 Pending

What the tester expects in response:

```
| ----------------------------------------- |
| Header                                    |
| ----------------------------------------- |
| Question 1 (un-compressed label sequence) |
| ----------------------------------------- |
| Question 2 (un-compressed label sequence) |
| ----------------------------------------- |
| Answer 1 (un-compressed label sequence)   |
| ----------------------------------------- |
| Answer 2 (un-compressed label sequence)   |
| ----------------------------------------- |
```

You don't need to compress your response. We will never ask you to do something that will overflow the buffer size restriction of UDP, so compressing your response packet is not something you have to worry about. Though if you like an extra challenge feel free to compress the DNS packet, the tester will work with it too.

The question type will always be `A` and the question class will always be `IN` .

This section of the RFC covers how this compression works.

View Code Examples

Collapse ↑

Hints

Filter by Ru

Ready to run tests... Show logs

‹ back next › VIP

Forwarding Server #GT1 Pending

Complete previous stages to gain access to this stage.

Instructions Code Examples Forum

Your Task Pending

MEDIUM

In this stage, you will implement a forwarding DNS server.

A forwarding DNS server, also known as a DNS forwarder, is a DNS server that is configured to pass DNS queries it receives from clients to another DNS server for resolution. Instead of directly resolving DNS queries by looking up the information in its own local cache or authoritative records.

In this stage the tester will execute your program like this:

```
./your_server --resolver <address>
```

- where `<address>` will be of the form `<ip>:<port>`

It'll then send a UDP packet (containing a DNS query) to port 2053. Your program will be responsible for forwarding DNS queries to a specified DNS server, and then returning the response to the original requester (i.e. the tester).

Your program will need to respond with a DNS reply packet that contains:

- a header section (same as in stage #5)
- a question section (same as in stage #6)

Ready to run tests... Show logs

Instructions Code Examples Forum Stage #GT1 Pending

- a question section (same as in stage #6)
- an answer section (new in this stage) mimicing what you received from the DNS server to which you forwarded the request.

Here are a few assumptions you can make about the tester -

- It will always send you queries for `A` record type. So your parsing logic only needs to take care of this.

Here are few assumptions you can make about the DNS server you are forwarding the requests to -

- It will always respond with an answer section for the queries that originate from the tester.
- It will not contain other sections like (authority section and additional section)
- It will only respond when there is only one question in the question section. If you send multiple questions in the question section, it will not respond at all. So when you receive multiple questions in the question section you will need to split it into two DNS packets and then send them to this resolver then merge the response in a single packet.

Remember to mimic the packet identifier value sent by the tester in your response.

View Code Examples

Collapse ↑

Ready to run tests... Show logs