OpenChallSpec

Release 0.0.1

Jan 16, 2021

Contents

The OpenChallSpec (OCS) is a specification for jeopardy style Capture The Flag competition challenges. It defines a `challenge.yml` file which is used to describe the metadata and deployment of any CTF challenge. Usually, organizers either tediously enter metadata into their CTF platform and deploy challenges manually, or create their own challenge format that automates some or all of this work, but only covers their specific use case. The OCS aims to solve this by providing a challenge format that covers all use cases and works seamlessly with any CTF platform or tool.

The OCS has a few distinct design goals:

- Cover everyone's use cases. If a CTF platform or deployment tool has support for a certain feature, you should be able to configure this feature inside `challenge.yml` (within reason).
- Intercompatibility with every CTF platform and tool. A challenge written using the OCS for a CTF running CTFd should also work with any other CTF platform and be painless to import, granted that some features not shared between the platforms may not work.
- Challenge packageability. A folder containing a challenge should be able to be zipped, committed to a git repository, etc. and sent to another person while retaining all functionality, even if the challenge author and receiving person don't use the same tools or have the same environment.

If you have a use case the OCS doesn't cover, or a feature in your CTF platform not configurable from the OCS, don't hesitate to submit an issue so that it can be added to the spec.

CHAPTER 1

Making a challenge

Note: As you may see below, the OpenChallSpec is quite feature rich. Chances are that the CTF platform or deployment backend you may be using doesn't support all features, so be sure to research your platform or deployment backend to see what you can use.

1.1 Getting Started

A challenge is a folder containing a `challenge.yml` (or `challenge.yaml` if you prefer) configuration file. The `challenge.yml` YAML file is the heart of the challenge, defining all metadata like title, description, and categories, as well as things like which files players get to download or what services they get access to. A minimal challenge configuration file looks like this:

```
title: Example challenge
description: This is an example minimal challenge to showcase the OpenChallSpec.
authors: mateuszdrwal
categories: misc

flag_format_prefix: example{
flags: this_is_the_flag

downloadable_files:
  - challenge.txt

spec: 0.0.1
```

The first part of the example defines the metadata we are all used to: the title, description, author and category.

Then, the flag format and flag are defined. `flag_format_prefix` specifies the first part of the flag format. The variable part of the flag is defined right below in `flags`, so the resulting flag for this challenge is `example{this_is_the_flag}`.

Later, a `challenge.txt` is specified as a file that should be given to players through the `downloadable_files` option. The `challenge.txt` file has to exist in the same directory as the `challenge.yml` file.

Lastly, it is specified that the configuration follows the 0.0.1 version of OCS. This will usually be the latest version when developing challenges.

1.2 Adding a service

Many CTF challenges have a service, which is usually a website or a TCP server. In the OCS, these services are provided by Docker containers. In the configuration, it looks thusly:

```
service:
  image: ./container
  type: website
  internal_port: 8080
  external_port: 80
```

The docker image is built according to the `image` option. In this case, a directory `container` is specified, from which the image will be built. The `container` directory is assumed to have a `Dockerfile`. The `image` option can also have other values, like a docker image tag or a path to an image archive.

`type` can by default be one of `website` or `tcp`, and represents what type of service it is that is exposed. `internal_port` then defines at which port this service listens inside the container, and the optional `external_port` defines what port should be exposed on the host machine.

At this point, using a command line tool to assist with challenge creation can prove useful. The officially recommended tool is challtools, however you may use whichever you like, or none at all. Challtools can build the docker image for you as defined in the config simply by running `challtools build`. You can also start the service locally by running `challtools start`.

1.3 Adding a solve script

Note: This is a useful but uncommonly supported feature on competition infrastructure. It is usually perfectly fine to not have a solve script.

A solve script can be useful locally during development to check that the challenge is working properly, but it can also be used to check if a service is online by periodically running it against the service during the competition. To add a solve script to your challenge, add the following into your configuration:

```
solution_image: ./solution
```

This option behaves similarly to the `image` option *above*. In this case, a docker image will be built from the `solution` directory. To test if a challenge is solvable, this image will be ran with the challenge location as command line arguments. For example, a command line argument can be a string like `nc 192.0.2.69 1337` for tcp services, or the URL for website services. Therefore, if you are using a container, make sure it runs the script using `ENTRYPOINT` in exec form instead of `CMD` in the `Dockerfile` so that command line arguments get passed correctly. The container should output just the flag on STDOUT if the challenge was solved successfully.

If this seems complicated, check out the practical TODO example.

1.4 Other common options

Below is a list of other commonly used configuration options. Some of these were also explained above, but have extra functionality that was not explained above.

1.4.1 authors

The `authors` option can be a string for simplicity, but it can also be an array for when there are multiple authors.

```
authors:
  - mateuszdrwal
  - loovjo
```

1.4.2 categories

Similarly to *authors*, the `categories` option can be a string for simplicity or an array if the challenge has multiple categories. The first category in the list will be the "main" category.

```
categories:
  - web
  - forensics
```

1.4.3 flag_format_prefix and flag_format_suffix

`flag_format_prefix` and `flag_format_suffix` together define the flag format for the challenge. `flag_format_suffix` defaults to `}`, so it should rarely be needed (unless you are using a non-standard flag format, to which I say please don't). `flag_format_prefix` does not have a default so it needs to be specified in every challenge, for example `exampleCTF{`. If the challenge does not include a flag format, flag_format_prefix should be set to `null` in which case both options will be ignored.

1.4.4 tags

Tags are similar to categories, but can also include things that spoil the challenge. They are not shown to players, and are usually used for organizers own reference, but are also synonymous with tags on ctftime, so challenges can be easily added there with the right tags after a CTF. The `tags` option can be a single string, or a list of strings.

```
tags: SQL injection
```

```
tags:
  - SQL injection
  - local file inclusion
```

1.4.5 hints

Challenge hints can be configured using the `hints` option. Below is an example with two hints, one free and one that costs 100 points.

```yaml
hints:
  - content: git gud # the hint cost defaults to 0
  - cost: 100
    content: this hint costs points
```

1.4.6 score

If you are using static scoring, specify the challenge score here. A value of `null` usually means dynamic scoring. Defaults to `null`, so if you are using dynamic scoring you don't have to specify this option.

```yaml
score: 500
```

1.4.7 predefined_services

If you are deploying challenges manually or have some external unchanging service, you will want to define services using `predefined_services`. These will show to users exactly the same as the service defined in `service`, but they are not managed automatically. Usually, services are either of type `website`, in which case you need to specify `url`, or of type `tcp`, in which case you need to specify `host` and `port`. If needed, *custom types can also be defined*.

```yaml
predefined_services:
  - type: website
    url: "https://example2.com"
  - type: tcp
    host: 203.0.113.43
    port: 1337
```

1.4.8 unlocked_by

If a challenge should only be visible/available after a certain other challenge is solved, put the title of that challenge in `unlocked_by`. This option also has more advanced features and several related options, explained in the *Advanced configuration options* section.

CHAPTER 2

Advanced configuration options

Here are all other options not explained in *Making a challenge*. Some options from *Making a challenge* are also expanded upon here, in order to explain their full functionality.

2.1 flags

The *flags* option supports specifying multiple flags, and also regex flags. To do this, use the following syntax:

```
flag_format_prefix: example{
flag_format_suffix: "}"
flags:
- flag: here_is_a_text_flag
  type: text
- flag: here_is_another_text_flag # The type defaults to text, so it doesn't have to␣
↪be specified
- flag: ^here\s+is\s+some\s+flag\s+with\s+arbitrary\s+whitespace$
  type: regex
```

All of these flags will be valid when verifying against this challenge:

- `example{here_is_a_text_flag}`
- `example{here_is_another_text_flag}`
- `example{here is some flag with arbitrary whitespace}`
- `example{here is some flag with arbitrary whitespace}`

When making a regex flag, be sure to wrap the regex in `^` and `$`, which matches the beginning and end of the flag respectively. Otherwise, a regex flag `foo` will match any flag containing the word `foo`, including the flag `example{you_probably_didnt_foo_intend_this_to_match}`.

2.2 max_attempts

If a team should only have a certain number of attempts at submitting the correct flag before they are blocked, specify this number in the *max_attempts* option. In most cases, this is not something you want to do, as this leads to teams creating alt accounts and testing flags there to not be locked out of their main one. If you are looking at a way to stop people from brute forcing flags, rate limiting is usually a better idea. This option should really only be used in closed CTFs where only admins can create team accounts which they distribute to teams. The default is `null`, which signifies no limit.

```
max_attempts: 10
```

2.3 downloadable_files

Like most other options, *downloadable_files* can be either a single string or a list of strings. In addition to specifying a path to a file, a directory or a URL can also be specified. In the case of a directory, all files within it will be included. For the URL, it will be forwarded to the user raw, meaning it does not have to lead to a direct download.

```
downloadable_files: container/app.py
```

```
downloadable_files:
  - container/app.py
  - downloads/ # all files in the downloads directory will be downloadable by players
  - "https://example.com/some_file" # players will be redirected to this link to␣
↪download the file
```

2.4 custom_service_types

By default, `website` and `tcp` service types are defined, which should be sufficient for the vast majority of cases. Sometimes it maybe useful to specify a custom type, like the type `htjp` for this HTJP challenge. Doing so would look like this:

```
custom_service_types:
  - type: htjp
    user_display: "htjp://{host}:{port}"
    hyperlink: false # defaults to false
```

For the sake of example, lets say predefined_services looks like this:

```
predefined_services:
  - type: htjp
    host: 203.0.113.43
    port: 1337
```

The `user_display` option specifies how this service is displayed to players. In this case, players would receive the string `htjp://203.0.113.43:1337` as the service. If the challenge uses `deployment` to deploy a `htjp` type service, the formatting options provided will be `host` containing the ip/DNS resolvable hostname, `port` containing the port, and `url` containing the host and port formatted as a proper http URL. You can use other formatting options, but in that case you must either write a custom deployment backend that provides those options or use predefined_services, where you can define arbitrary formatting options.

The `hyperlink` option defines if a CTF platform should make the service clickable and openable in a browser.

The default `website` and `tcp` types cannot be overwritten. They are defined thusly:

```
custom_service_types:
  - type: website
    user_display: "{url}"
    hyperlink: true
  - type: tcp
    user_display: "nc {host} {port}"
    hyperlink: false
```

Keep in mind that because they cannot be overwritten, the above snippet is not valid according to the OCS if it was inserted into a configuration file.

2.5 deployment

The *deployment* option is a more complicated version of the `service` option used in *Making a challenge*, with many more features. Support for these varies depending on what you use to deploy the challenge. A full configuration, using all features, looks like this (making a challenge that looks like this is a bad idea, but we will get to that)

```
deployment:
  type: docker
  containers:
    web:
      image: container
      services:
        - type: website
          internal_port: 8080
          external_port: 80
      extra_exposed_ports:
        - internal_port: 1337
          external_port: 1337
    db:
      image: postgres:latest
  networks:
    test-network:
      - web
      - db
  volumes:
    test-volume:
      - web: /shared_volume
      - db: /shared_volume
```

Similarly to as explained in *Making a challenge*, this deployment defines a container called `web`, built from the `container` directory, which exposes a website service on external port 80.

Additionally, the `web` container exposes the port 1337 through `extra_exposed_ports`. Ports exposed through this option are "secret" ports; they are not given to players. Here, as opposed to in `services`, `external_port` is required.

After the `web` container is defined, another `db` container is defined. This challenge now consists of two separate docker containers. The `db` container is pulled from dockerhub as its image is specified as `postgres:latest`. This container could expose its own services, but it doesn't.

Now that all containers are defined, `networks` are defined. One network named "test-network" is defined to which both `web` and `db` containers will be connected when they are deployed. This network behaves the same way as a Docker bridge network. For example, the `web` container can use the hostname `db` to connect to the database and vice versa.

Lastly, shared volumes are defined. A volume called `test-volume` is created, which is mounted at `/shared_volume` in both `web` and `db` containers.

Making a deployment that looks like this is a bad idea, as support for these features is not expected to be widespread. Unless it's required, it's better to use as few deployment features as possible for simplicity, like using only one container, no networks, no volumes, etc. In practice, the `deployment` option should very rarely be used as the `service` option already provides enough functionality for the vast majority of challenges and should be used instead.

2.6 unlocked_by

Similarly to other options, *unlocked_by* can be either a string or a list of strings. For behaviour relating to when a challenge should be unlocked if it has multiple requirements, see *all_unlocked_by_required*. The recommended way to specify which challenge is a requirement for this challenge is by setting the string to its *challenge_id*. The exact title of the challenge can also be specified, however this can cause errors if the required challenge is renamed.

```
unlocked_by:
  - Example requirement challenge
  - a62c6318-9306-48c8-95ea-6a374461ac91
```

2.7 all_unlocked_by_required

This option is a boolean. If `true`, all challenges in the *unlocked_by* list must be solved in order for this challenge to be accessible. If `false`, only one challenge from the list needs to be solved. Defaults to `false`.

```
all_unlocked_by_required: true
```

2.8 release_delay

If the challenge should be automatically released/published after a certain time since the CTF started, specify the number of seconds in *release_delay*. Defaults to 0.

```
release_delay: 3600 # one hour
```

2.9 human_metadata

human_metadata unsurprisingly contains metadata intended to be read and processed by humans. Filling this in isn't in any way required, but it's nice to have for people that might look at your challenge source in the future.

```
human_metadata:
  challenge_version: "0.0.1"
  event_name: "exampleCTF 2020"
```

`challenge_version` can be used to keep track of the challenge version for yourself. Some deployment backends or CTF platforms may show this to help with knowing what version of the challenge is currently deployed. Obviously, this is only useful if you yourself keep updating it.

`event_name` is the name of the CTF this challenge is for. This is useful for archival purposes.

2.10 challenge_id

To uniquely identify a challenge in *unlocked_by* and perhaps across your infrastructure you can set *challenge_id* to a unique string. It is recommended to generate a UUID at creation time and use it, as it effectively guarantees a unique id for every challenge in existence. This is not a requirement though, and the id can be any string. It should however be something unique, even beyond the scope of your CTF. Defaults to `null`.

```
challenge_id: "3ce287f8-9c61-44c4-9113-79eb9a4d7d71"
```

2.11 custom

If you are writing your own infrastructure and have an obscure requirement the OCS doesn't support, you might find the *custom* option useful. *custom* is an object that you can format however you want, and there are no constraints on what you can put in it. If you for example want to play a different video when a team solves a challenge depending on which challenge they solve, a `solve_video_url` option would not be a good fit to include with the OCS as it's very obscure, but it can easily be configured by including it in *custom*.

custom may also be a good choice if there is a feature you are waiting for to be added into the OCS, but hasn't arrived yet and you need to use it now.

2.12 spec

This is the version of the OCS the challenge configuration is written in. These docs are written for version 0.0.1, so this is probably what the *spec* should be set to if you are following these docs.

CHAPTER 3

Challenge Examples

TODO

CHAPTER 4

Challenge specification

This document contains the full detailed technical specification for creating a challenge compliant to the OpenChallSpec version 0.0.1. For tips on how to create a reader, see writing_a_reader.

4.1 Wording and terminology

This document uses key words defined in **RFC 2119**.

The term "reader" in this document refers to any automated script that reads the challenge configuration file. It is used as an umbrella term for command line tools, CTF platforms, deployment backends or anything else that might interact with the OCS in its raw form.

4.2 Versioning

The OCS follows semantic versioning. Readers SHOULD support all versions of the OCS below the version they are written for but still at the same MAJOR version. Semantic versioning ensures that older OCS versions can be parsed without issue by readers designed for new versions, so in practice backwards compatibility should be automatic. Readers SHOULD refuse to parse versions of the OCS with a MINOR version higher than what they were designed for, at the same MAJOR version. Similarly, readers MAY also refuse to work with higher PATCH versions. This is because the *Schema* is not written in a forwards compatible manner, and new features will usually constitute schema violations when parsed with older OCS versions.

4.3 Challenge structure

A challenge is a directory containing a `challenge.yml` or `challenge.yaml` configuration file. Readers MUST support both `.yml` and `.yaml` file extensions. The configuration file MUST be written in YAML. Anything outside the challenge directory is not considered part of the challenge. All challenge files MUST be located inside the challenge directory, as this can otherwise cause packageability issues.

4.4 Schema

The OCS includes a JSON Schema for validation of the challenge configuration file structure, which can be found here. Yes, the OCS uses JSON schema to validate a YAML file, but it works the same way as both JSON and YAML files parse to the same "dictionary" representation, and JSON Schema validators in any programming language usually take this dictionary representation as input. Readers MAY validate the configuration against the schema before attempting to interpret the configuration. A configuration file that doesn't comply with the schema is considered invalid. The schema includes default values for all non-required fields, which may be a convenient method of filling in defaults.

4.5 Implementation completeness

Readers are not required to implement functionality for all fields and sub-fields below. Functionality for any field MAY be implemented. If the challenge configuration contains fields the reader does not support the user SHOULD be notified of this, unless the incompatibility is obvious, like when a standalone deployment script that only deploys challenge containers does not do anything with the challenge description. The script MAY continue running ignoring these fields. No CTF platform (probably) supports all features in the OCS, so ignoring certain fields is a very typical thing to do.

4.6 Configuration fields

Below is a list of all valid fields in the challenge configuration. Unless otherwise specified, only the described keys are valid for any object. This means that any additional unexpected keys make the configuration invalid.

4.6.1 title

Required	true
Type	String

The challenge title.

4.6.2 description

Required	true
Type	String

The challenge description.

4.6.3 authors

Required	true
Type	String, Array of strings

The challenge's authors. If the type is string, this is simplified syntax. It SHOULD be interpreted as a single element array of this one string. The first item in the array MAY be considered the main author.

4.6.4 categories

Required	true
Type	String, Array of strings

The challenge's categories. If the type is string, this is simplified syntax. It SHOULD be interpreted as a single element array of this one string. The first item in the array MAY be considered the main category. If the reader does not support multiple categories, the first one MUST be used. There MUST be at least one category.

4.6.5 tags

Required	false
Type	String, Array of strings
Default	[]

The challenge's tags. If the type is string, this is simplified syntax. It SHOULD be interpreted as a single element array of this one string. The first item in the array MAY be considered the main tag.

Tags are "private categories". They MUST NOT be shown to players. They record high level concepts that spoil the challenge, like "SQL injection".

4.6.6 hints

Required	false
Type	Array of objects
Default	[]

An array of the challenge's hints. If supported, these MUST be shown to players. Some hints cost points, in which case the action of opening a hint should subtract the price from the opening teams point total.

The objects are composed of these two sub-fields:

content

Required	true
Type	String

The hint text that is shown when opened.

cost

Required	false
Type	Number
Default	0

The hint price.

4.6.7 flag_format_prefix

Required	true
Type	String, null

The first part of the flag format that the challenge's flag(s) start(s) with. May also be `null` instead of a string signifying no flag format present for the challenge. In that case, the values of both *flag_format_prefix* and *flag_format_suffix* MUST be ignored for flag validation.

To validate a player submitted flag, a validator SHOULD first check if the flag starts with the *flag_format_prefix* and ends with the *flag_format_suffix*. If so, the prefix and suffix is stripped from the flag and rest should be matched against the list of *flags*. If it didn't, the flag's flag format is invalid.

4.6.8 flag_format_suffix

Required	false
Type	String
Default	`}`

The last part of the flag format that the challenge's flag(s) start(s) with. Defaults to `}` for convenience. For more info, see *flag_format_prefix*.

4.6.9 flags

Required	true
Type	String, Array or objects

An array of the challenge's flags. If the type is string, this is simplified syntax. It SHOULD be interpreted as this array instead: `[{"flag": "<initial string here>"}]`

Every element in the array is a separate flag, meaning that for a flag submission to be valid it must match at least one of the listed flags. If a reader doesn't support multiple flags, the first flag MUST be used.

The objects in the array are composed of these two sub-fields:

flag

Required	true
Type	String

The flag contents, without the flag format as that is defined separately.

type

Required	false
Type	String
Default	`text`

MUST be either `text` or `regex`.

If the type is `text`, the `flag` field is to be compared directly to the contents of the user submitted flag. If they are the same, the submission is considered correct.

If the type is `regex`, the `flag` field is considered to be a regex and the user submitted flag is to be matched against the regex. If it matches, the submission is considered correct. When writing challenges, the `flag` SHOULD start with `^` and end with `$`, to prevent false positives for very short flags.

4.6.10 max_attempts

Required	false
Type	Integer, null
Default	null

A positive integer signifying how many times teams may attempt to submit a flag before they are stopped from submitting any more for the challenge. If `null`, the teams have an unlimited number of tries.

Use of this option is heavily discouraged, as it often leads to a bad player experience. If you want to prevent brute-force attacks, try rate limiting instead.

4.6.11 score

Required	false
Type	Number, null
Default	null

An integer signifying how many points a team receives in reward for solving the challenge, for static scoring. For dynamic scoring, set to `null`. The dynamic scoring formula is handled by the ctf platform.

4.6.12 downloadable_files

Required	false
Type	String, Array of strings
Default	[]

An array of files downloadable by players. If the type is string, this is simplified syntax. It SHOULD be interpreted as a single element array of this one string. The string MUST be one of three things:

1. A relative path to a file in the challenge directory. Readers MUST check if the file exists, and if not, move on to the other two options.
2. A relative path to a directory in the challenge directory. All files in the directory should be included with the challenge. Readers MUST check if the directory exists, and if not, assume the string is the last option.
3. A URL to a file. When players attempt to download this file, they MUST be redirected to this URL. Therefore, it does not have to be a direct download and can be for example a google drive link.

4.6.13 custom_service_types

Required	false
Type	Array of objects
Default	[]

A list of custom service types. A service type is a concept that defines how services should be automatically shown to players. It is used in definitions of *predefined_services*, *service* and *deployment*. There are two built-in service types. they look like this:

```
- type: website
  user_display: "{url}"
  hyperlink: true
- type: tcp
  user_display: "nc {host} {port}"
  hyperlink: false
```

These built-ins MUST be treated as if they are always the first two items in the array. For example, if the custom_service_types array contains only a newly defined type `foo`, the reader MUST treat the list of defined types as containing `website`, `tcp` and `foo`. Duplicate types MUST NOT be allowed. Therefore, `website` and `tcp` cannot be redefined.

Each object in the custom_service_types array has the following 3 fields:

type

Required	true
Type	String

The name of the type that is being defined. Can be any string that is not an already defined type.

user_display

Required	true
Type	String

Defines how services with this type will be shown to players and solve scripts. Variables can be substituted in by typing the variable name immediately enclosed in curly brackets. Additional whitespace between the name and bracket MUST NOT be supported. For example, for the `tcp` service type, if the hostname of a service is `192.0.2.69` and the port is `1337`, the ctf platform will show the string `nc 192.0.2.69 1337` in the challenge details.

The variables in the substitution context depend on the environment. If the service is in *predefined_services*, all needed variables MUST be provided in that same object. Otherwise, if the service is automatically deployed with a *service* or *deployment* configuration, it is the job of the deployment script to provide a context with the required variables. Deployment scripts MUST attempt to deploy services of a custom type they don't know of and format user_display by providing the `host`, `port` and `url` variables in the substitution context.

hyperlink

Required	false
Type	Boolean
Default	false

If the resulting *user_display* string after substitution is reachable by a web browser. If this is `true`, ctf platforms MAY encase the string in an `<a>` tag.

4.6.14 predefined_services

Required	false
Type	Array of objects
Default	[]

A list of services for the challenge that are not automatically deployed. For example. if you will be manually deploying a service for a challenge, the hostname and port/URL to the challenge should be entered here.

Each predefined service object consists of the following one mandatory field, and any number of additional fields:

type

Required	true
Type	string

The service type for this service. MUST be either `website`, `tcp`, or one defined in *custom_service_types*. See *custom_service_types* for info on what a service type is.

All other fields are formatting context for formatting *user_display*. Therefore, if the service type is `website`, a `url` field must be passed. If the object is instead `tcp`, a `hostname` and `ip` field must be passed.

4.6.15 service

Required	false
Type	Object, null
Default	null

This field is a simplified syntax of the *deployment* field. It consists of 3 mandatory fields `image`, `type` and `internal_port`, and one optional field `external_port`. When this field is present, assume that the *deployment* field has the following contents where `<field name>` is replaced by the contents of this *service* field:

```
type: docker
containers:
  default:
    image: <image>
    services:
      - type: <type>
        internal_port: <internal_port>
```

(continues on next page)

(continued from previous page)

```
        external_port: <external_port>
networks: {}
volumes: {}
```

If this field is present, the *deployment* field MUST NOT be present.

4.6.16 deployment

Required	false
Type	Object, null
Default	null

Defines in detail all services that are used by the challenge. At the top level, the object consists of the following fields:

type

Required	true
Type	string

Currently, only the `docker` type is supported, so this MUST be the value. In the future more backends may be supported, like LXC or some jails.

networks

Required	false
Type	Object
Default	{}

Defines networks between containers, for multiple containers. These behave the same way as regular docker networks. A container will be able to reach another container by its container name if they have a network in common.

Each key in the *networks* object is a network name. Its value is an array of strings of container names in this network. For example, the following will put the `foo` and `bar` containers on the same network:

```
networks:
  test-network:
    - foo
    - bar
```

volumes

Required	false
Type	Object
Default	{}

Defines persistent volumes for one or multiple containers. These behave the same way as regular docker volumes. A volume can be mounted into a container at a mountpoint, and the data in it will persist between container recreations. If the volume is mounted in two containers at the same time, it behaves like a shared folder.

Each key in the *volumes* object is a volume name. Its value is an array of objects representing a mountpoint. Each mountpoint object has exactly one key, being the container name, and its value is where to mount the volume inside the container. For example, the following will mount the same volume at `/shared_volume` in both the `foo` and `bar` containers:

```
volumes:
  test-volume:
    - foo: /shared_volume
    - bar: /shared_volume
```

containers

Required	true
Type	Object

This is the last field of the *deployment* object. Defines all docker containers for this challenge. Each key in the *containers* object is a container name. Its value is a container object. These objects contain the following fields:

image

Required	true
Type	String

Defines the docker image for this container. This can be defined in one of three ways:

1. A path to a directory containing a `Dockerfile`. In this case, the image will be built from said dockerfile. Readers MUST check if the directory exists, and if not, move on to the other two options.
2. A path to a file containing an exported docker image. This file is usually obtained using the `docker save` command and results in a tarball. In this case, the exported image will be imported and used. Readers MUST check if the file exists, and if not, assume the string is the last option.
3. A docker image tag. This can be a from an image locally on the system, publically available on dockerhub, from a private container repository etc. In this case, the image will be pulled if required and used.

services

Required	false
Type	Array of objects
Default	[]

Defines the services exposed by this challenge. Each service is an object in the array. The Object has the following three fields:

type

Required	true
Type	String

The service type for this service. MUST be either `website`, `tcp`, or one defined in *custom_service_types*. See *custom_service_types* for info on what a service type is.

internal_port

Required	true
Type	Integer

The port inside the container that is exposed. This is the port your service binds to when running in the container.

external_port

Required	false
Type	String
Default	See below

The port on the host machine that the service is exposed on. If ommited, The deployment script will pick some available port. This SHOULD NOT be set unless the service requires being exposed on a specific port because this can cause issues with port collisions if the service is run on a host that also runs multiple other services.

extra_exposed_ports

Required	false
Type	Array of objects
Default	[]

Defines other ports that need to be exposed from within the container. These can be thought of as "hidden *services*". They are formatted the same way as *services*, however they do not have a `type` as they will never be shown to users or solve scripts, and `external_port` is mandatory because of this.

Here is an example of a fully utilized deployment configuration:

```
deployment:
  type: docker
  containers:
    web:
      image: ./container
      services:
        - type: website
          internal_port: 80
```

(continues on next page)

(continued from previous page)

```
        external_port: 80
    extra_exposed_ports:
      - internal_port: 1337
        external_port: 1337
  db:
    image: local_db_image:latest
networks:
  network:
    - web
    - db
volumes:
  volume:
    - web: /shared_volume
    - db: /shared_volume
```

While it is supported, it is highly RECOMMENDED that challenges are created without *volumes*, *networks*, or multiple *containers* and *services*, as these features are not expected to be widely supported and are only required in very few situations. The *service* field SHOULD be used instead unless absolutely necessary.

If this field is present, the *service* field MUST NOT be present.

4.6.17 solution_image

Required	false
Type	String, null
Default	null

A solution script that can be run to validate the challenge is functioning and solvable. This is meant mostly to test challenges with services, and could be run periodically during a CTF to validate that a challenge has not gone offline or broke in other ways. The solution is housed in a docker container so it can be run anywhere.

The string defines the docker image for this solution. This can be defined in the same ways as *the image in a service container*.

The solution container usually needs to know on which host and port a service runs on. This information is passed as a string in a command line argument when running a docker container. The string MUST be formatted using the *user_display* of service types, meaning that for example, for tcp services, a string like `nc 192.0.2.69 1337` will be given.

If a challenge has multiple services, they MUST be passed as separate command line arguments in the following order: 1. All *predefined_services*, in the order they are defined 2. For all *containers* in the order they are defined: all *services*, in the order they are defined

When creating the container, be sure to use ENTRYPOINT in exec form as otherwise the command line arguments will not be passed to the entrypoint in the container. Using `CMD` instead will not work.

The solution container MUST be run with an environment variable `FLAG`, containing the first `text`-type *flags* entry enclosed in the flag format (a valid flag). If no such entry exists, the environment variable MUST be set to an empty string.

If the challenge is functioning as expected, the solution container MUST output nothing more than a valid flag and optionally a trailing newline. Scripts that run this solution container SHOULD strip the resulting flag from whitespace on both ends before validating, in order to prevent rouge whitespace from invalidating the flag. Any output that is not a valid flag should be treated as if the service is malfunctioning.

4.6.18 unlocked_by

Required	false
Type	String, Array of strings
Default	[]

If a challenge should only be accessible to players after a certain other challenge is solved, this should be defined here. If the type is string, this is simplified syntax. It SHOULD be interpreted as a single element array of this one string. Each entry in the array can be either the exact case sensitive challenge title of another challenge, or a different challenges *challenge_id*. Referencing a challenge by its *challenge_id* has the added benefit of the link not breaking if the challenge is renamed.

Specifying multiple requirement challenges is NOT RECOMMENDED, as support in CTF platforms is uncommon. If you do specify multiple challenges, see *all_unlocked_by_required* for exact behaviour.

4.6.19 all_unlocked_by_required

Required	false
Type	Boolean
Default	false

If *unlocked_by* contains multiple challenges, defines if one or all need to be solved for this challenge to unlock. If `true`, all challenges in the array MUST be solved for this challenge to be accessible. If `false`, any one of the challenges in the array MUST be solved for this challenge to be accessible.

4.6.20 release_delay

Required	false
Type	Number
Default	0

The amount of seconds after the CTF start when the challenge should be automatically released. If `0`, the challenge is released when the CTF starts.

4.6.21 human_metadata

Required	false
Type	Object
Default	{}

Contains metadata that is designed to be read by humans, and not parsed by scripts. This can be useful for some data that you want to display in user interfaces.

This field is composed of the following two sub-fields:

challenge_version

Required	false
Type	string, null
Default	null

Defines the version of the challenge. SHOULD be shown on user interfaces for deployment backends so admins can easily see which version of the challenge is deployed, if they specified a version. The format of the string is undefined and can be decided by the challenge author.

event_name

Required	false
Type	string, null
Default	null

Defines the name of the event the challenge is for, for example `exampleCTF 2020`. For archival purposes.

4.6.22 challenge_id

Required	false
Type	string, null
Default	null

A unique identifier for this challenge. MUST be unique not only among the pool of challenges for the CTF this challenge belongs to, but among all challenges. It is therefore RECOMMENDED that this is set to a generated UUID.

This id can be used in *unlocked_by* instead of the challenge title. The advantage of this is that the link will not break if the challenge is renamed. This can also be used by readers to recognize if it is reading a challenge it already knows about, even if the title has changed.

4.6.23 custom

Required	false
Type	Object
Default	{}

An object with an undefined structure. Any custom data that is not supported by the OCS can be put here. This is useful if you have tooling that provides functionality not supported by the OCS itself, as you will be able to specify configuration values in this object in any format you like. For example, if you have implemented a feature in your CTF platform that plays an audio file when a player solves a specific challenge, you could specify which audio file to play in a custom `solve_audio` configuration field in this object.

4.6.24 spec

Required	true
Type	string

The version of the OCS the challenge is written in. The current version is `0.0.1`, so this MUST be the value if the challenge follows the version described in this document.

CHAPTER 5

Changelog

5.1 0.0.1

- Initial release

Index

R