

Create your own template (elestio.yml)

You can **create your own template to be deployed on Elestio** for yourself (private repo) or for anyone (public repo).

We recommend that you fork one of our sample repositories as a base and modify it to meet your requirements:

[Samples repositories for the most popular stacks & frameworks \(52 Apps & Frameworks\)](#)

To streamline the deployment process the application stack must be defined in [elestio.yml](#), this file contains all details about the runtime, build/run commands, env vars, ports & reverse proxy.

Here is an example [elestio.yml](#) file about a docker-compose-based stack (must be placed at the root of your repository)

```
config:
  runTime: "dockerCompose"
  version: ""
  framework: ""
  installCommand: ""
  buildCommand: "docker-compose build"
  buildDir: "/"
  runCommand: "docker-compose up -d"
  isMonoRepoPackageInRoot: true/false
environments:
  - key: "SOFTWARE_VERSION_TAG"
    value: "latest"
  - key: "SOFTWARE_PASSWORD"
    value: "random_password"
  - key: "ADMIN_EMAIL"
    value: "[EMAIL]"
  - key: "ADMIN_PASSWORD"
    value: "random_password"
ports:
  - protocol: "HTTPS"
```

```

    targetProtocol: "HTTP"
    listeningPort: "443"
    targetPort: "8001"
    targetIP: "172.17.0.1"
    public: true
    path: "/"
    isAuth: true
    login: "root"
    password: "random_password"
- protocol: "TCP"
    targetProtocol: "TCP"
    listeningPort: "26379"
    targetIP: "172.17.0.1"
    targetPort: "6379"
    public: true
lifeCycleConfig:
    preInstallCommand: "./scripts/preInstall.sh"
    postInstallCommand: "./scripts/postInstall.sh"
    preBackupCommand: "./scripts/preBackup.sh"
    postBackupCommand: "./scripts/postBackup.sh"
    preRestoreCommand: "./scripts/preRestore.sh"
    postRestoreCommand: "./scripts/postRestore.sh"
    preUpdateCommand: "./scripts/preUpdateCommand.sh"
    postUpdateCommand: "./scripts/postUpdateCommand.sh"
    preDeployCommand: "./scripts/preDeployCommand.sh"
    postDeployCommand: "./scripts/postDeployCommand.sh"
webUI:
- url: "https://[CI_CD_DOMAIN]"
  label: "Redis Insight Web UI"
  login: "[ADMIN_EMAIL]"
  password: "[ADMIN_PASSWORD]"
monoRepoWorkSpaces: []

```

"config" section

Runtime: define the base runtime image used to build your app, possible values are: **NodeJs, Java, PHP, Ruby, Python, .NET, GO, static, dockerCompose, docker**

Version: define the version of the runtime image used to build your app, a list of available versions is provided in the dropdown in our dashboard, you can find the corresponding values in

the docker hub.

Framework: based on the selected framework, our UI will auto-populate build/run commands & output directory. This is optional.

installCommand: command to be executed only for the first deployment.

buildCommand: indicate the command to execute to build your code.

runCommand: indicate the command to execute to run your code.

buildDir: indicate the directory where the build output will be placed.

isMonoRepoPackageInRoot: Not necessary unless you want to deploy a yarn/npm mono repo workspace. (Boolean type true/false)

"environments" section

If your app requires env vars for configuration, you can define them one by one in the section. For each env var, you need to indicate the key and the value. Here is an example to define an env var named "DB_TYPE" and with value "postgres":

```
[- key: "DB_TYPE"
  value: "postgres"
```

You can use those variables in the "value" field:

random_password: this will be replaced by a randomly generated password on deployment.

[EMAIL]: this will be replaced by the email address of the user deploying the pipeline on deployment.

[CI_CD_DOMAIN]: this will be replaced by the CNAME of the pipeline on deployment.

"ports" section

For each port exposed by your application you need to define a few fields:

protocol: host protocol (exposed to the internet), which can be HTTP, HTTPS, TCP, or UDP.

targetProtocol: container protocol, which can be HTTP, HTTPS, TCP, or UDP.

listeningPort: host port (exposed to the internet), any port between 1 to 65535.

targetPort: container port, any port between 1 to 65535.

targetIP: container IP address used as a reverse proxy target. You should leave it to 172.17.0.1 except if you know what you are doing.

public: can be true or false, if true, the app is exposed on the public IPV4, if not it's exposed only on the private ip address.

path: relative path to the service in the container. You should keep "/" except if you know what you are doing.

public: can be true or false, if true the app will be exposed to the internet.

isAuth: can be true or false, if true the app will require a basic authentication (define in login/password below)

login: login to be required to access the service

password: password to be required to access the service (can be any string, or **random_password** described above)

Here is an example to define an inbound route with HTTPS on port 443 pointing to a container running on port 8001 on the docker host with default IP 172.17.0.1, we also want the service to be protected with basic auth with the login "root" and a new random password generated at deploy time:

```
- protocol: "HTTPS"
  targetProtocol: "HTTP"
  listeningPort: "443"
  targetPort: "8001"
  targetIP: "172.17.0.1"
  public: true
  path: "/"
  isAuth: true
  login: "root"
  password: "random_password"
```

"lifeCycleConfig" section

Define custom scripts to be executed before or after installation, backup, or restore operations

```
lifeCycleConfig:
  preInstallCommand: "./scripts/preInstall.sh"
```

```
postInstallCommand: "./scripts/postInstall.sh"
preBackupCommand: "./scripts/preBackup.sh"
postBackupCommand: "./scripts/postBackup.sh"
preRestoreCommand: "./scripts/preRestore.sh"
postRestoreCommand: "./scripts/postRestore.sh"
preUpdateCommand: "./scripts/preUpdateCommand.sh"
postUpdateCommand: "./scripts/postUpdateCommand.sh"
preDeployCommand: "./scripts/preDeployCommand.sh"
postDeployCommand: "./scripts/postDeployCommand.sh"
```

This is optional, if you want to use it just add the section and define the relative path from the root of your repository to the script to be executed when the event occurs.

"copyCommandConfig" section

Define shortcuts pointing to the web UI. They will appear in the Elestio dashboard.

```
copyCommandConfig:
  - ./myLocalFile1 .
  - ./myLocalFile2 .
```

[CI_CD_DOMAIN] will be replaced at runtime with the pipeline CNAME address

[EMAIL] will be replaced at runtime with the email address of the user deploying the pipeline

In login and password, you can indicate the env vars to be used instead

"monoRepoWorkSpaces" section

It will be of array type. example: ["workspace-a", "workspace-b"]

Not necessary unless you want to deploy a yarn/npm mono repo workspace and you're passing *true* for *isMonoRepoPackageInRoot* in the config section.

"webUI" section

Define shortcuts pointing to the web UI. They will appear in the Elestio dashboard.

```
webUI:
  - url: "https://[CI_CD_DOMAIN]/"
    label: "Admin Web UI"
    login: "[ADMIN_EMAIL]"
    password: "[ADMIN_PASSWORD]"
```

[CI_CD_DOMAIN] will be replaced at runtime with the pipeline CNAME address

[EMAIL] will be replaced at runtime with the email address of the user deploying the pipeline

In login and password, you can indicate the env vars to be used instead

You can then add a Deploy on Elestio button to your site or readme file like this:

```
<a
href="https://dash.elest.io/deploy?source=cicd&social=Github&thirdParty=true&url=https://github.com/elestio-examples/docker-compose-redis">
  
</a>
```

It will be displayed like this:

[Deploy on Elest.io](#)

Revision #20

Created 4 July 2022 13:28:27 by Joseph Benguira

Updated 4 May 2023 15:20:41 by Amit