Primitive.io Documentation

- For all the information, visit documentation.primitive.io

Docker Installation

Windows

Docker CE is available through Docker Hub Here. This will run on Windows 10 Pro or Enterprise 64-bit as it requires Hyper-V.

Linux (Ubuntu / Debian)

For *experimentation* and trials, installing Docker can be as easy as:

```
sudo apt update
sudo apt install
sudo apt install docker.io
```

Official Documenation from Docker (Recommended)

For more in-depth installation on production systems, please see Docker's Official Documentation. This will walk you through the process of installing and setting up the Docker Engine on an Ubuntu/Debian based system.

MacOS

Docker CE is also available for MacOS. This download is available at Docker Hub and will provide the information necessary for installing Docker. MacOS installs require versions 10.13 or newer (i.e. High Sierra (10.13), Mojave (10.14) or Catalina (10.15). Mac hardware must be a 2010 or a newer model). For odler versions, please see Docker Toolbox for more information.

Welcome to Primitive!

Welcome to the Primitive Documentation portal! Here you will find information related to the Primitive VR client as well as scraper information.

For more information, please visit us at Primitive.io or email us at **support@primitive.io**

Coming Soon

*# Requirements

The private scraper has a few general requirements in order to properly run.

General Requirements

- Server/machine with the ability to connect to your git hosting service.
- Administration access into Git service to create access tokens or app passwords.
    - (NOTE: The account associated with the token/password should have permissions to access all relevant repositories)
- Network capable of allowing clients to connect to server/machine hosting the private scraper.

Network Requirements

- Static IP address or ability to set up dynamic dns.
    - Non-static ip addresses can be used but this may cause issues with the clients ability to connect without frequent alterations to the client configuration file.

System Requirements (Docker-Deployment)

- Windows 10 Pro, Enterprise, and Education
- MacOS
- Linux (CentOS, Ubuntu, or Red Hat Enterprise Linux 7 (RHEL7) with kernel version 3.10 or higher)
- Recommended 2 GB of RAM
- Recommended 5 GB of available disk space *# Overview

The Private Scraper allows users to view their private repositories through the Primitive client. The preferred method of deployment is through the use of Docker. Other options are available to run the private scraper though Node.js within a daemon process manager (e.g., PM2).

Access to scraper source code or container

Please contact us at **support@primitive.io** to learn how to download the container or source code for deployment. *# Docker Installation

Windows

Docker CE is available through Docker Hub Here. This will run on Windows 10 Pro or Enterprise 64-bit as it requires Hyper-V.

Linux (Ubuntu / Debian)

For *experimentation* and trials, installing Docker can be as easy as:

```
sudo apt update
sudo apt install
sudo apt install docker.io
```

Official Documenation from Docker (Recommended)

For more in-depth installation on production systems, please see Docker's Official Documentation. This will walk you through the process of installing and setting up the Docker Engine on an Ubuntu/Debian based system.

MacOS

Docker CE is also available for MacOS. This download is available at Docker Hub and will provide the information necessary for installing Docker. MacOS installs require versions 10.13 or newer (i.e. High Sierra (10.13), Mojave (10.14) or Catalina (10.15). Mac hardware must be a 2010 or a newer model). For odler versions, please see Docker Toolbox for more information. *# Running Scraper in Docker Container

The private scraper is best run inside of Docker. This allows for consistency across platforms and removes potential dependency issues when deploying across different platforms.

Required Configuration

The private scraper relies on two main elements to run properly: a location to store the internal database and an configuration file (config.json). The configuration file will be in the data folder which is used to store data associated with the scraper.

- See information related to environment file here
- See information for required volume here

Importing Image from Archive

Docker images can be saved and archived for easier transport. The scraper image can be distributed via a ".tar" file rather than pulled from a registry.

```
docker load --input primitive_scraper.tar
```

Once loaded, the image should be viewable with 'docker images.' Once imported with the load command, the container can be run as outlined in following sections.

Running Container

Once the scraper container is pulled and available to the local machine, the container can be ran. Below is an example of the command required for the scraper to execute properly.

```
docker run -d -p 443:443 -v /home/scraper/data:/srv/primitive/data -v
/var/run/docker.sock:/var/run/docker.sock -v /tmp:/tmp --name primitive_1.0 --
restart unless-stopped private_scraper:1.0
```

Important information:

- For versions under v4.0, ports should remain 3000:3000 so the admin panel can comunicate with the scraper.
- The volume information follows the format "local:container." This means that the first path should exist on the local machine.
- The volume is used to persist data and the database. Choose a non-temporary directory.

See below for additional explaination of command flags and options:

- **'-d'** - Run the process as a daemon. Without this command, the container will execute and bind to the console. This can be removed for testing if something is failing with the run command.
- **'-p'** - Publish or expose a specific port.
- **-v** - Mount a volume to the container. Because the scraper pulls data from your git service and saves assets to be served to the Primitive client, data persistence is important. In the example above, we are mounting a local directory (/tmp/primitive/data) to a folder inside the container(/srv/primitive/data). By doing this, the database and scraped assets will be stored outside the container and available on the local machine. Make sure that the local directory is present when running the container.
- **-name** - This will be the name of the running container. It is best to name this something that represents the function and includes the versioning.
- **-restart** - This is the restart policy. We can set this to `unless-stopped` to have the container restart if the dependent application or process crashes.
- The last value is the image name. This will be the name of the image pulled from the registry and include the specific version being used (the example shows :latest but this may very well me a specific version.)

*** NOTE: Due to the websocket calls from the admin panel, port 3000 should be used for versions below 4.0 ***

Monitoring Containers

To see what container are running or available, run docker ps. The '-a' flag can be used to show all containers and not just running ones. If a container crashes, you will have to use the '-a' flag to view it.

```
docker ps
docker ps -a
```

Stopping Container

Stopping a container can be done through the 'docker stop' command. The container name can be found through the 'docker ps' command.

```
docker stop primitive_2.0
```

Starting Container

Starting a container can be done through the 'docker start' command. The container name can be found through the 'docker ps' command.

```
docker start primitive_2.0
```

Updating Container

Updating the scraper consists of stopping the current container, pulling a new version, and running the new one. Stopped containers can be maintained in case there were bugs introduced in the updates. For example:

```
docker stop primitive_1.0
docker pull <new container name>
docker run <new container name>
```

If older version of the container are no longer needed, they can be deleted.

```
docker rm primitive_1.0
```

Old images can also be viewed and removed through the 'docker image' command.

```
docker image ls
docker image rm primitive_1.0
```

Accessing Running Process

Once the container is running you can see it with the 'docker ps' command. To access the internal process, the 'exec' command can be used.

```
docker exec -it primitive_1.0 /bin/bash
```

This binds to the container primitive_1.0 and binds to an interactive bash shell giving the ability to troubleshoot and monitor the process. Within the container, we are running PM2 (a daemon process manager for Node.js). To see running logs and the process dashboard, use the following commands.

```
pm2 logs
pm2 monit
```

*# Environment File (config.json)

The config file is used to set the required variables and values for the scraper to run. This is now a JSON file that is added to the `data` folder which is mounted to the host. This will be used to authenticate to the private git services and ensure the client can properly display the content.

Below is an example of the required config file:

```
{
    "PORT": 443,
    "HOST": "{{ scraper_host }}",
    "PROTOCOL": "https",
    "VERBOSE": true,
    "SQLITE_DATABASE": true,
    "IS_PRIVATE": true,
    "PROVIDER": "{{ provider }}",
    "BASE_URL": "{{ base_url }}",
    "GITHUB_USER": "{{ github_user }}",
    "GITHUB_ACCESS_TOKEN": "{{ github_token }}",
    "BITBUCKET_USER": "{{ bitbucket_user }}",
    "BITBUCKET_PASSWORD": "{{ bitbucket_token }}",
    "GERRIT_USER": "{{ gerrit_user }}",
    "GERRIT_PASSWORD": "{{ gerrit_token }}",
    "GENERATE_DATABASE": true,
    "TEMP_DIR": "/tmp",
    "OUTPUT_DIR": "/home/scraper/data",
    "ANALYZERS":[
        {"name": "dotnet-parser", "image_name": "xxxxxxxxx.dkr.ecr.us-east-
2.amazonaws.com/dotnet-parser:prod-0.1.23", "extensions": [".cs", ".h", ".hxx",
".hpp", ".cpp", ".c", ".cc", ".m", ".py", ".py3", ".js", ".jsx", ".kt", ".ts"]}
    ],
    "UPDATE_PERMS": false,
    "ROOT_PASS":"",
    "SSL_KEY": "{{ ssl_key }}",
    "SSL_CERT": "{{ ssl_cert }}",
    "SSL_CA": "{{ ssl_ca }}"
}
```

- **PORT** - The port that the service will bind too. If used with docker, this is internal to the docker container. If the service is run through Node.js or PM2, this will be the port that will be used to connect to the running scraper. If this is left out, the port will default to 3000.
- **HOST** - The host at which the scraper will be available. This will be used to construct the raw content urls returned from the scraper.
- **PROTOCOL** - https or http. Currently, only http is supported.
- **VERBOSE** - Verbose logging. Used internally to the container. External logging is on the roadmap.
- **SQLITE_DATABASE** - True. Currently, the container uses an internal sqlite database.
- **IS_PRIVATE** - Let's the scraper know that the repositories are private and require authentication.
- **PROVIDER** - Specifies which git service provider is going to be used. Below are the supported options:
    - github
    - bitbucket

- **GITHUB_USER** - User account associated with the generated access token.
- **GITHUB_ACCESS_TOKEN** - The Github access token for api calls and authentication. Only needs read access for the repositories.
- **BITBUCKET_USER** - User account associated with the generated app password.
- **BITBUCKET_PASSWORD** - App password with read access for the repositories.
- **TEMP_DIR** - The temporary file used during various processes in the scraper container. We usually use '/tmp' but this can be configured to something different.
- **OUTPUT_DIR** - This is the local folder used to mount the container to the local file system. This will be shared between containers and therefore must be defined externally via the configuration file.
- **ANALYZERS** - This is an array of json objects representing which parsers are available for the scraper. This includes the name, the image_name, and the extensions. The name is used internally and is usually just set to the parser name without the repo URL. The image name is what the actual name of the image is in the local docker environment. The extensions specifies which filetypes should be associated with that specific parser. If the extensions array is empty, no filetypes will be associated with that parser and it will NOT be run on project files.
- **UPDATE_PERMS** - This can update the permissions of the scraper DBs if the scraper is run in a different permission that the parsers. This can be used during debugging or during development if the debugger being leveraged does not run as root.
- **ROOT_PASS** - Used primarily during testing, this is used if the above variable is true (UPDATE_PERMS). This should be empty under normal deployments.
- **SSL_KEY and SSL Variables** - Absolute file path of the ssl certs associated with the scraper. These are the files within the certs directory mounted to the container within the data directory. *# Volume and Data Persistence

The scraper pulls data from the git service and generates assets and cache entries for the client. These assets and cache entries exist outside of the container for data persistence. Within the specified folder, you will see one Sqlite database file and a series of folders that represent the scraped repositories. These assets container filenames, directory structures, and data used for preview generation.

!> NO SOURCE CODE IS SAVED OUTSIDE THE CONTAINER

During the scrape, the project is downloaded into the container into a temporary directory. This directory is deleted as soon as the files are crawled and the relevant data is saved.

Before running the container, determine where this data will live and note the location for the run command when deploying the container. If the service is run outside of docker, the 'data' folder is required in the root of the project.

*# Private Scraper Usage

Scraper Admin Interface

The scraper interface can be broken down into two main functions: scraping new repositories, and managing previously scraped ones.

Scraping New Repositories

At the top of the admin panel, there is a text field labeled 'Repository URL' with a 'Scrape Repo!' button. To scrape a new repository within your private infrastructure, input the http/https clone url in the text field and

press the scrape button. It's that easy! There are a couple of notes here...

- The http/https clone url should not include your username. The url should be in the format **[baseurl]/[organization/owner]/[project name]**.
- The base url can include the protocl (http/https) and the associated port (7990/3000/etc).

Managing Existing Repositories

Once a repository is scraped and ready for the client, you will see a "complete" message within the status bar at the bottom of the page. On a page refresh, the newly scraped repository should appear in the Repo table.

Once in the repo table, you will have the ability to edit the metadata of the repository for display within the Primitive Client, remove the repository, requeue the repository, and view the current data saved.

Requeue

Requeuing removes the current cache entry and re-scrapes the repository. This can be done to update the repo in the Primitive client to the most recent version pushed to your master branch.

Remove

Removing the repository removes the repo data from the scraper. This removes the ability for the Primitive client to graph the codebase and the repository must be re-scraped in order to restore this ability.

Repository Values

The repository values include a tag, layer, and enabled field.

Value	Description
Repo	The repository url that represents the scraped data. This should resemble the http/https clone url. There should be no username within the url and the '.git' can be removed. This is for standardization across git platforms.
Tag	The tag is the group label used to organize the repositories within the Primitive environment. Each tagged repository will be grouped based on their tag for easier navigation of the codebases
Layer	The layer is the value associated with which level the repository is placed in. The core of the Primitive structure is layer 0 and the layer number increments as you move away from the center. -1 is the default which puts it on the outer layer.
Enabled	The enabled flag allows you to remove the repository from the Primitive environment without deleting the data. A value of '0' disables the repository where a value of '1' enables it.

Repository Preview

The image that you see at the top of the model is a quick preview of what the codebase structure looks like. This is used as a starting point in the visualization process and becomes the foundation of the VR structure. If

there is no preview present, something may have gone wrong. Please check that the URL is correct and there is no connectivity issues.

Repository Links

The links included in the modal represent the data being consumed by the Primitive client. All other information pertaining to the repository are streamed into the client on demand.

*# Troubleshooting

Issue	Possible Solution
Preview is blank	The preview might be blank if the scrape was interrupted, or an error occured. This could be from the process being rate limited, the URL being incorrect, or network connectivity. Please check the URL and ensure it is in the correct format, make sure that the rate limits on your git service are appropriately set, and confirm that the scraper can access the git service
Some assets are not present (Repository links result in 404)	See above issue. This could also be caused by rate limiting issues, incorrect URLS, and/or network connectivity
Assets are valid but source code is unavailable in the client.	The base url could be incorrectly set in the scrapers configuration. Check to see that the 'url' value within the 'Package Data Url' is valid and points to the appropriate resource.
Client is unavailable to view source code and raw URLS are valid	Ensure that the client is properly authenticating to the git service. Source code is streamed from the git service to the Primitive client and is not handled by the scraper. Please check the credentials within the client to ensure proper authentication is occuring.
Raw URLS are malformed	Please check the environment file used to configure the scraper during setup. The 'provider' and 'base url' values should be validated to ensure the scraper is properly configured.

Contact Us

If issues are unabled to be resolved, please contact us at **support@primitive.io**.