

# Lecture 18 - TaskRunners in NodeJS

---

## TaskRunner

A TaskRunner is a software component or tool designed to manage and execute tasks or jobs in a systematic and organized manner.

It plays a crucial role in automating and streamlining various processes, making it an essential part of many applications and systems.

## Need of TaskRunner

- **Task Management:** Organizes and executes tasks systematically.
- **Automation:** Reduces manual effort by automating repetitive tasks.
- **Parallel Processing:** Enhances efficiency by executing tasks in parallel.
- **Dependency Resolution:** Ensures tasks run when prerequisites are met.
- **Monitoring and Logging:** Tracks task progress and logs status.
- **Scalability:** Handles tasks across multiple servers for scalability.

## How it Works:

- **Task Definition:** Tasks are defined with attributes like name, code, and dependencies.
- **Task Queue:** Tasks are added to a queue for execution.
- **Scheduling:** Tasks are scheduled based on dependencies and priority.
- **Execution:** Tasks are executed, and results are monitored.
- **Logging:** Task execution details and errors are logged.
- **Notification:** Alerts can be sent upon completion or specific events.

- **Scaling:** Distributes tasks across multiple servers as needed.

## Applications:

- **Batch Processing:** Used for processing large volumes of data, such as data ETL (Extract, Transform, Load) jobs and image processing.
- **Workflow Automation:** TaskRunners automate workflows in various domains, including finance, healthcare, and manufacturing, to improve efficiency.
- **Continuous Integration/Continuous Deployment (CI/CD):** They are integral to CI/CD pipelines, running tests, building and deploying code automatically.
- **Server Maintenance:** TaskRunners help manage server maintenance tasks like backups, software updates, and security scans.
- **Job Schedulers:** In operating systems, they serve as job schedulers to manage system-level tasks and user-defined scripts.
- **Data Pipelines:** TaskRunners are used in creating and managing data pipelines for data analysis and reporting.
- **IoT Device Management:** TaskRunners can manage and update software on IoT devices.
- **Content Publishing:** For blogs and websites, they can automate content publishing and updates.
- **Game Servers:** In online gaming, TaskRunners can manage game server instances and matchmakings.

## Grunt Setup

Grunt is a popular JavaScript task runner that automates repetitive tasks in web development.

### Installation:

To install Grunt, follow these steps:

- Install Node.js if not already installed. Grunt runs on Node.js.
- Install Grunt's command-line interface (CLI) globally using npm (Node Package Manager) with the following command:

```
npm install -g grunt-cli
```

### Gruntfile Setup:

The Gruntfile is a configuration file used to define and configure tasks in Grunt.

Steps:

- Create a Grunt Project Directory:

Start by creating a new directory for your Grunt project.

Open your command line or terminal and navigate to this directory.

- Initialize a Node.js Project:

Run the following command to create a package.json file, which stores project metadata and dependencies: `npm init`

- Install Grunt Locally:

Install Grunt as a project dependency in the project directory using npm. This allows you to manage Grunt versions specific to your project:

```
npm install grunt --save-dev
```

- Create a Gruntfile:

Create a JavaScript file named "Gruntfile.js" in your project directory. This is where you define your Grunt tasks and configuration.

## Task Configuration in Gruntfile:

Inside the Gruntfile, you'll configure and define tasks. A basic Gruntfile structure looks like this:

```
module.exports = function (grunt) {  
  grunt.initConfig({  
    // Define tasks and configurations here  
  });  
  
  // Load Grunt plugins and tasks  
  grunt.loadNpmTasks('plugin-name');
```

```
// Define custom tasks if needed  
grunt.registerTask('custom-task', ['task1', 'task2']);  
};
```

- Load Grunt Plugins:

To use specific Grunt plugins, you need to load them using **grunt.loadNpmTasks('plugin-name')** in the Gruntfile.

- Task Registration:

You can register custom tasks using **grunt.registerTask()** to define the order in which tasks are executed.

- Run Grunt:

To execute Grunt tasks defined in the Gruntfile, simply run grunt in your project directory: `grunt`

## Gulp Installation

Gulp is a JavaScript task runner that automates repetitive tasks in web development.

Installation:

To install Gulp, follow these steps:

- Install Node.js if not already installed. Gulp runs on Node.js.
- Install Gulp's command-line interface (CLI) globally using npm (Node Package Manager) with the following command:

```
npm install -g gulp-cli
```

- Verify the installation by checking the Gulp version:

```
gulp --version
```

## Gulpfile Setup:

The Gulpfile is a configuration file used to define and configure tasks in Gulp.

Steps:

- Create a Gulp Project Directory:

Start by creating a new directory for your Gulp project.

Open your command line or terminal and navigate to this directory.

- Initialize a Node.js Project:

Run the following command to create a package.json file, which stores project metadata and dependencies: `npm init`

- Install Gulp Locally:

Install Gulp as a project dependency in the project directory using npm. This allows you to manage Gulp versions specific to your project:

```
npm install gulp --save-dev
```

- Create a Gulpfile:

Create a JavaScript file named "gulpfile.js" in your project directory. This is where you define your Gulp tasks and configuration.

- Task Configuration in Gulpfile:

Inside the Gulpfile, you'll configure and define tasks. A basic Gulpfile structure looks like this:

```
import gulp from 'gulp';
import pluginName from 'gulp-plugin-name';

gulp.task('task-name', function () {
  return gulp.src('source-files')
    .pipe(pluginName(/* plugin options */))
    .pipe(gulp.dest('destination'));
});
```

- Load Gulp Plugins:

To use specific Gulp plugins, you need to load them using **'gulp-plugin-name'** in the Gulpfile.

- Task Execution:

Gulp tasks are executed using the **gulp.task()** and **gulp.src()** functions. The task definition includes source files, plugins, and destination paths.

- Run Gulp:

To execute Gulp tasks defined in the Gulpfile, simply run gulp in your project directory: `gulp`

## Image Optimization using Gulp

### 1. Setting up Gulp for Image Optimization:

- First, set up Gulp and create a Gulpfile (as discussed in a previous response).
- Image Optimization Plugin: You'll need an image optimization plugin for Gulp. Popular choices include `gulp-imagemin` and `gulp-image-optimization`. Install it as a project dependency.

### 2. Define a Gulp Task:

In your Gulpfile, define a task for image optimization. Here's a basic structure:

```
import gulp from 'gulp';  
import imagemin from 'gulp-imagemin';
```



```
gulp.task('optimize-images', () => {  
  return gulp.src('src/images/*')  
    .pipe(imagemin())  
    .pipe(gulp.dest('dist/images'));  
});
```

In the above code, `gulp.src()` specifies the source directory, `imagemin()` is the image optimization plugin, and `gulp.dest()` is the destination directory.

## Summarising it

Let's summarise what we have learned in this module:

- We discussed TaskRunners and their importance, needs, and applications.
- We explored the Grunt task runner, including its basic setup and operation.
- We also examined another significant TaskRunner, Gulp, its functionality, and use cases.
- Furthermore, we implemented image optimization using `gulp-imagemin`.

## Some Additional Resources:

- [Grunt Plugins](#)
- [Gulp Plugins](#)