Adopting Theia: Insights from an Initial Contributor

EclipseCon 2022
Ludwigsburg, Germany
Overview

- **Eclipse Theia** ramp-up tour
  - What is **Eclipse Theia**?
  - **Theia Extensions** and VS Code Extensions
  - Common pitfalls
- Usage/deployment examples and breakdown
  - **Docker** images (e.g., deprecated [github.com/theia-ide/theia-apps](https://github.com/theia-ide/theia-apps))
  - **Eclipse Blueprint**
  - Proof of concept authentication portal
What is Eclipse Theia?

It’s a framework allowing developers to make browser and desktop IDE-like applications.

**Eclipse Theia** comes as a collection of NPM packages

- Lot of **Theia Extension** packages (45?)
- Few development packages such as the **Theia CLI**
- All under the `@theia/*` NPM namespace
- Cannot easily tell what’s a **Theia Extension** or not from the name alone...
<table>
<thead>
<tr>
<th>Package Name</th>
<th>Description</th>
<th>Version</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>@theia/editor</td>
<td>Thia - Editor Extension</td>
<td>1.38.0</td>
<td>22 days ago</td>
</tr>
<tr>
<td>@theia/markers</td>
<td>Thia - Markers Extension</td>
<td>1.38.0</td>
<td>22 days ago</td>
</tr>
<tr>
<td>@theia/callhierarchy</td>
<td>Thia - Call Hierarchy Extension</td>
<td>1.38.0</td>
<td>22 days ago</td>
</tr>
<tr>
<td>@theia/file-search</td>
<td>Thia - File Search Extension</td>
<td>1.38.0</td>
<td>22 days ago</td>
</tr>
</tbody>
</table>
What is a Theia app?

It’s an NPM application that’s also a collection of Theia Extensions.

Install `@theia/cli` as dev dependency to bootstrap your applications:

- It will crawl your `node_modules` looking for Theia Extensions to include
- It will generate the frontend and backend scripts and assets
- You can check for dependency issues
- You can start your app quickly
What is a Theia Extension?

It’s an NPM package that defines a “theiaExtensions” field in its `package.json`.

```javascript
export interface Extension {
  frontend?: string;
  frontendElectron?: string;
  secondaryWindow?: string;
  backend?: string;
  backendElectron?: string;
  electronMain?: string;
}
```

```json
"theiaExtensions": [
  {
    "frontend": "lib/browser/terminal-frontend-module",
    "secondaryWindow": "lib/browser/terminal-frontend-module",
    "backend": "lib/node/terminal-backend-module"
  }
],
```
What can a Theia Extension do?

Virtually anything: every feature is implemented through some Theia Extension.

```javascript
bind(MonacoToProtocolConverter).toSelf().inSingletonScope();
bind(ProtocolToMonacoConverter).toSelf().inSingletonScope();
bind(MonacoLanguages).toSelf().inSingletonScope();
rebind(LanguageService).toService(MonacoLanguages);
bind(WorkspaceSymbolCommand).toSelf().inSingletonScope();
for (const identifier of [CommandContribution, KeybindingContribution,
                         MenuContribution, QuickAccessContribution]) {
    bind(identifier).toService(WorkspaceSymbolCommand);
}
bind(MonacoWorkspace).toSelf().inSingletonScope();
```
Example Theia app

```json
{
    "private": true,
    "name": "my-theia-app",
    "version": "1.0.0",
    "main": "src-gen/backend/main.js",
    "dependencies": {
        "@theia/plugin-ext-vscode": "latest"
    },
    "devDependencies": {
        "@theia/cli": "latest"
    }
}
```
Transitive dependencies

- **Theia Extensions** are deeply interdependent
- Installing one **Theia Extension** may pull several other **Theia Extensions** (as per its dependencies)
- [@theia/cli](https://theia.io) sees it all 😎
Bogus Theia app

```json
{
  "private": true,
  "name": "my-theia-app",
  "version": "1.0.0",
  "main": "src-gen/backend/main.js",
  "dependencies": {
    "@theia/new-dependency": "latest",
    "@theia/plugin-ext-vscode": "latest"
  },
  "devDependencies": {
    "@theia/cli": "latest"
  }
}
```
Bogus Theia app

```json
{
    "private": true,
    "name": "my-theia-app",
    "version": "1.0.0",
    "main": "src-gen/backend/main.js",
    "dependencies": {
        "@theia/new-dependency": "1.30.0",
        "@theia/plugin-ext-vscode": "1.29.0"
    },
    "devDependencies": {
        "@theia/cli": "1.29.0"
    }
}
```
Dependency mishaps

- First, be aware about your NPM dependencies
- Second, use `@theia/cli` to spot common issues with Theia Extensions
  - `npx theia check:theia-version`
Theia Extension or VS Code Extension?

- First try implementing your feature as a VS Code Extension
  - Make sure Theia supports the APIs you plan on using
- Otherwise consider writing a Theia Extension
Theia Extensions stability

- **Stable**
  - Contribution points (e.g., `FrontendApplicationContribution`)

- **Somewhat stable**
  - Some services (e.g., `RequestService`)

- **Not stable**
  - Arbitrary classes and protected methods (implementation details)
  - We still make a best effort to not break these, but it may hinder maintenance
VS Code Extensions

- Theia’s implementation of the VS Code API
- Theia Community Release 1.29.1 supports VS Code API 1.53.2
  - See the Theia Community Release: theia-ide.org/releases/
    - Latest was released on September 29th, 2022
  - See the API comparator: eclipse-theia.github.io/vscode-theia-comparator/status.html
- Ongoing efforts to support more recent APIs
<table>
<thead>
<tr>
<th>Method</th>
<th>Theia v1.30.0</th>
<th>Theia v1.29.1-community</th>
<th>VSCode 1.72.2</th>
<th>VSCode 1.71.2</th>
<th>VSCode 1.70.2</th>
<th>VSCode 1.69.2</th>
<th>VSCode 1.55.2</th>
<th>VSCode 1.53.2</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>onDidExpandElement</td>
<td>Supported</td>
<td>Supported</td>
<td>Supported</td>
<td>Supported</td>
<td>Supported</td>
<td>Supported</td>
<td>Supported</td>
<td>Supported</td>
<td></td>
</tr>
<tr>
<td>reveal</td>
<td>Supported</td>
<td>Supported</td>
<td>Partial</td>
<td>Partial</td>
<td>Partial</td>
<td>Partial</td>
<td>Partial</td>
<td>Partial</td>
<td></td>
</tr>
<tr>
<td>selection</td>
<td>Supported</td>
<td>Supported</td>
<td>Supported</td>
<td>Supported</td>
<td>Supported</td>
<td>Supported</td>
<td>Supported</td>
<td>Supported</td>
<td></td>
</tr>
<tr>
<td>title</td>
<td>Supported</td>
<td>Supported</td>
<td>Supported</td>
<td>Supported</td>
<td>Supported</td>
<td>Supported</td>
<td>Supported</td>
<td>Supported</td>
<td></td>
</tr>
<tr>
<td>visible</td>
<td>Supported</td>
<td>Supported</td>
<td>Supported</td>
<td>Supported</td>
<td>Supported</td>
<td>Supported</td>
<td>Supported</td>
<td>Supported</td>
<td></td>
</tr>
<tr>
<td>TreeViewExpansionEvent</td>
<td>Supported</td>
<td>Supported</td>
<td>Partial</td>
<td>Partial</td>
<td>Partial</td>
<td>Partial</td>
<td>Partial</td>
<td>Partial</td>
<td></td>
</tr>
<tr>
<td>element</td>
<td>Supported</td>
<td>Supported</td>
<td>Partial</td>
<td>Partial</td>
<td>Partial</td>
<td>Partial</td>
<td>Partial</td>
<td>Partial</td>
<td></td>
</tr>
<tr>
<td>TreeViewOptions</td>
<td>Partial</td>
<td>Partial</td>
<td>Partial</td>
<td>Partial</td>
<td>Partial</td>
<td>Partial</td>
<td>Partial</td>
<td>Partial</td>
<td></td>
</tr>
<tr>
<td>canSelectMany</td>
<td>Unsupported</td>
<td>Unsupported</td>
<td>Unsupported</td>
<td>Unsupported</td>
<td>Unsupported</td>
<td>Unsupported</td>
<td>Unsupported</td>
<td>Unsupported</td>
<td></td>
</tr>
<tr>
<td>dragAndDropController</td>
<td>Unsupported</td>
<td>Unsupported</td>
<td>Unsupported</td>
<td>Unsupported</td>
<td>Unsupported</td>
<td>Unsupported</td>
<td>Unsupported</td>
<td>Unsupported</td>
<td>#10102</td>
</tr>
<tr>
<td>showCollapseAll</td>
<td>Supported</td>
<td>Supported</td>
<td>Supported</td>
<td>Supported</td>
<td>Supported</td>
<td>Supported</td>
<td>Supported</td>
<td>Supported</td>
<td></td>
</tr>
<tr>
<td>treeDataProvider</td>
<td>Supported</td>
<td>Supported</td>
<td>Supported</td>
<td>Supported</td>
<td>Supported</td>
<td>Supported</td>
<td>Supported</td>
<td>Supported</td>
<td></td>
</tr>
<tr>
<td>TreeViewSelectionChangeEvent</td>
<td>Supported</td>
<td>Supported</td>
<td>Partial</td>
<td>Partial</td>
<td>Partial</td>
<td>Partial</td>
<td>Partial</td>
<td>Partial</td>
<td></td>
</tr>
<tr>
<td>selection</td>
<td>Supported</td>
<td>Supported</td>
<td>Partial</td>
<td>Partial</td>
<td>Partial</td>
<td>Partial</td>
<td>Partial</td>
<td>Partial</td>
<td></td>
</tr>
<tr>
<td>TreeViewVisibilityChangeEvent</td>
<td>Supported</td>
<td>Supported</td>
<td>Supported</td>
<td>Supported</td>
<td>Supported</td>
<td>Supported</td>
<td>Supported</td>
<td>Supported</td>
<td></td>
</tr>
<tr>
<td>visible</td>
<td>Supported</td>
<td>Supported</td>
<td>Supported</td>
<td>Supported</td>
<td>Supported</td>
<td>Supported</td>
<td>Supported</td>
<td>Supported</td>
<td></td>
</tr>
<tr>
<td>TypeDefinitionProvider</td>
<td>Supported</td>
<td>Supported</td>
<td>Supported</td>
<td>Supported</td>
<td>Supported</td>
<td>Supported</td>
<td>Supported</td>
<td>Supported</td>
<td></td>
</tr>
<tr>
<td>provideTypeDefinition</td>
<td>Supported</td>
<td>Supported</td>
<td>Supported</td>
<td>Supported</td>
<td>Supported</td>
<td>Supported</td>
<td>Supported</td>
<td>Supported</td>
<td></td>
</tr>
<tr>
<td>TypeHierarchyItem</td>
<td>Supported</td>
<td>Unsupported</td>
<td>Unsupported</td>
<td>Unsupported</td>
<td>Unsupported</td>
<td>Unsupported</td>
<td>Unsupported</td>
<td>Unsupported</td>
<td>#11516</td>
</tr>
<tr>
<td>constructor</td>
<td>Supported</td>
<td>Unsupported</td>
<td>Unsupported</td>
<td>Unsupported</td>
<td>Unsupported</td>
<td>Unsupported</td>
<td>Unsupported</td>
<td>Unsupported</td>
<td></td>
</tr>
<tr>
<td>detail</td>
<td>Supported</td>
<td>Unsupported</td>
<td>Unsupported</td>
<td>Unsupported</td>
<td>Unsupported</td>
<td>Unsupported</td>
<td>Unsupported</td>
<td>Unsupported</td>
<td></td>
</tr>
</tbody>
</table>
The different kinds of integration

Integrated Development Environment (IDE)
- Supports a variety of workflows OOTB
- Lots of views
- Wizards?
- May feel heavy or bloated?
- More things may still be contributed

(Smart) Editors
- Main purpose: Work on code
- Somewhat dumb OOTB
- Smartness is contributed
- Tools are used, not often embedded
The VS Code way

Support common denominators:

- Code navigation
- Code completion
- Live static analysis
- In-editor debug
- Tasks
- Tests
- …
The VS Code way: LSP

- **Technical feat**
  - Specify a reliable *Language Server Protocol (LSP)* to transmit language information in real time
  - Ensures the *Language Server (LS)* logic is isolated from the main process
  - The protocol can run on a variety of transports (streams: pipes, TCP sockets, WebSocket, etc)

- **Collaboration feat**
  - *VS Code* can focus on implementing *editor core features*
  - The “communities” will happily develop language servers thanks to their *expert knowledge*
  - Servers can be implemented in any language, **not just JavaScript**
The VS Code way: DAP

• Technical feat
  – Just like the LSP, the Debug Adapter Protocol (DAP) is well specified and versatile

• Collaboration feat
  – Debuggers come in all shapes and sizes
  – Some can be remote controlled
  – Others must be driven from the command-line
  – Debugger developers not always keen on supporting this protocol
  – Instead, people may implement wrappers called **Debug Adapters (DA)**
The VS Code way: Plugins

- The editor provides common features
- VS Code Extensions (Plugins) leverage these features by providing actual intelligence
- Plugins may contribute “meta workflows”, but I would argue this is not the norm
  - Most plugins are just UI integrations of command-line tools
Modularity = Independence

- If you implement complex workflow as part of extensions, how do you execute it on CI?
- **Modularity** comes from **re-usable components**
- **Re-usable components** come with **minimal dependencies**
- Avoid dependence on your editor, or anything else really
Eclipse Theia
Image from [github.com/eclipse-theia/theia](https://github.com/eclipse-theia/theia)

Godot Editor
Image from [github.com/godotengine/godot](https://github.com/godotengine/godot)
Examples
Theia app breakdown

- **Theia CLI** will generate multiple artifacts
  - **src-gen/**
    - Entry points referencing each **Theia Extension** module statically
      - **backend/main.js**
      - **frontend/index.js**
  - **webpack.config.js**
    - Required boilerplate configuration to bundle the frontend
  - **lib/**
    - The bundled frontend from the **src-gen/frontend/index.js** entry point
BackendApplicationServer mechanism

If you bind something to the `BackendApplicationServer` symbol from `@theia/core` in one of your Theia Extension, then the Theia backend won’t try to serve the default Theia frontend generated next to it.

This means you can have a Theia backend that only acts as a dedicated server to a Theia frontend served from a Content Delivery Network (CDN), for example.
Docker images

• Pros
  – Easy to consume*

• Cons
  – Images prone to become monolithic
  – Hard to compose use cases as you need a new image for new combinations

*or is it?
Monolithic Docker images

Required when building a Docker image that embeds a Theia app:

- Build tools for Theia
  - These can be removed from the final image (pkg-config, libx11-dev, libxkbfile-dev, etc.)
  - E.g., use a multi-stage build to build Theia and get rid of the build tools

- Runtime binaries for Theia
  - Node.js

- Development tools to be used by/with Theia
  - Want to develop Java applications? Add Java
  - Want to develop Python applications too? Add Python
  - Want to develop X? Add Y. 😃
Theia in Docker

- Let users create their own development environment
- Somehow get a Theia app to run and have access to this container
  - Build the Theia app in the image?
  - Inject a pre-built Theia app in the container?
  - Run multiple containers for one workspace?
Eclipse Theia Blueprint

github.com/eclipse-theia/theia-blueprint
Eclipse Theia Blueprint browser

- Goal was to create an executable server ready to download
- Draft: github.com/eclipse-theia/theia-blueprint/pull/168
Eclipse Theia Blueprint browser

It is difficult to create executables from complex Node.js applications (hello Theia backend):

- While the Theia frontend is designed with bundling in mind, the Theia backend is not...
- Dynamic imports are problematic
- Sub-processes are problematic
  - Need to create a bundle for each spawned sub-process
- `vercel/pkg` is amazing but has an issue with “execArgv” when forking scripts
  - VS Code Extensions very often fork scripts (LS, DA, etc), and hence sometime fail
A simple coding interview portal?
A simple coding interview portal...

I want to see how interviewees fare when in front of code relevant to the expected role:

- I have an Ubuntu 20.04 virtual machine running at home
- I configured my home NAT to redirect a TCP port to this VM

Problems:

- If I run one Theia instance, what user should it have? Good ol’ www-data?
- If using the same user, do I need to manually clean up between interviews?
- If interviewees remember the hostname I gave them, can they interfere with future interviews?
A simple coding interview portal...

Interviewees should not be able to access anything outside of the allocated time for the interview.

When they do have access it should be limited to a one-time user’s privileges.
A simple coding interview portal...

- Register a session in advance
  - Must be privileged to create users
  - Required: `username` + interview `date/time` + connection `token`
  - Note that I decided to prepare the workspace files in the user’s home at registration time...
- Run the portal
  - Must be privileged to impersonate users
  - Check authentication token validity based on time
  - Act as a reverse proxy to local services when authenticated
  - Respond with 401 Unauthorized otherwise
A simple coding interview portal!

- **GET** `/portal/auth/:userToken`
  - Moves `<userToken>` into a cookie
  - Redirects to `/portal/loading`
- **GET** `/portal/loading`
  - Displays a simple html page with JavaScript code waiting for the workspace to boot
  - A button appears to open the workspace once ready (querying `/portal/wait`)
- **GET** `/portal/wait`
  - Responds with **200** when workspace is ready
- **ALL** `/`
  - The Theia application and workspace for `<user-token>`
A simple coding interview portal!

- Already used in a couple of interviews with great success
- Repository: [github.com/paul-marechal/theia-local-portal](https://github.com/paul-marechal/theia-local-portal)