The Multiplying Architecture

Eder Ignatowicz
Principal Software Engineer
@ederign

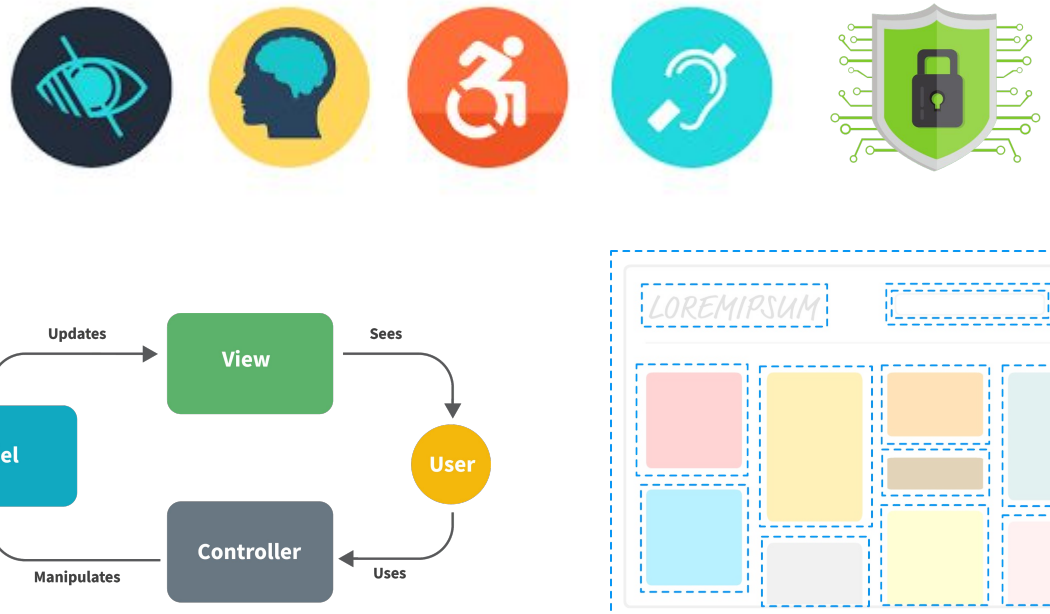
Red Hat

Landscape

We live in a world where **web technologies** have dominated software development.

Default choice for most applications.

Red Hat



Foundation

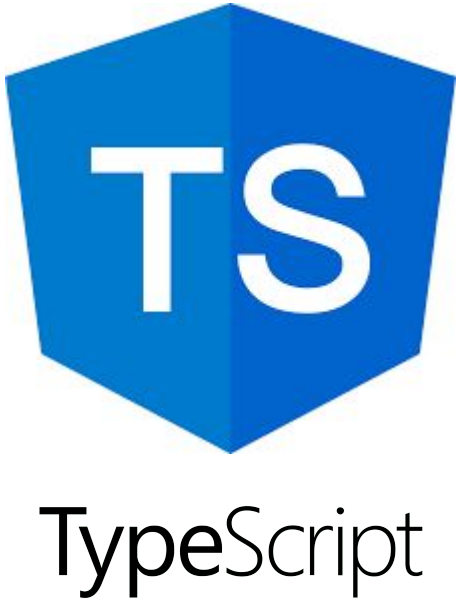
Well though and understood set of Standards, Patterns and Techniques as a strong foundation.

Red Hat

Ecosystem

Rich ecosystem that has been maturing over the years.

Red Hat



Static Typed Language

TypeScript created the perfect compromise for Static Type Languages believers.

Browser Everywhere

Browser is now more than just the window for the internet.

Browsers became part of an important trend as the mechanism to distribute any Graphical User Interface based applications.

Red Hat

Architectures

Evolutionary

An evolutionary architecture supports incremental, guided change as a first principle across multiple dimensions.

Microservices

Architectural style that structures an application as a collection of independent services.

Serverless

Incorporate third-party "Backend as a Service", and/or that include custom code run as Functions.

Micro Frontends

Design approach in which a front-end app is decomposed into individual, semi-independent "microapps" working loosely together.

Why do we need a **new** architecture?

Red Hat

Cloud Native Tooling

requirements

Multiple Distributions

The origin of multiplying architecture is rooted in the need to distribute the same set of components in a myriad of platforms.

Minimize code changes

The components to be distributed should be preserved untouched and with avoiding feature flags.

Bridge

It has to embrace different generations of technology stack.

What is Software Architecture?

Red Hat

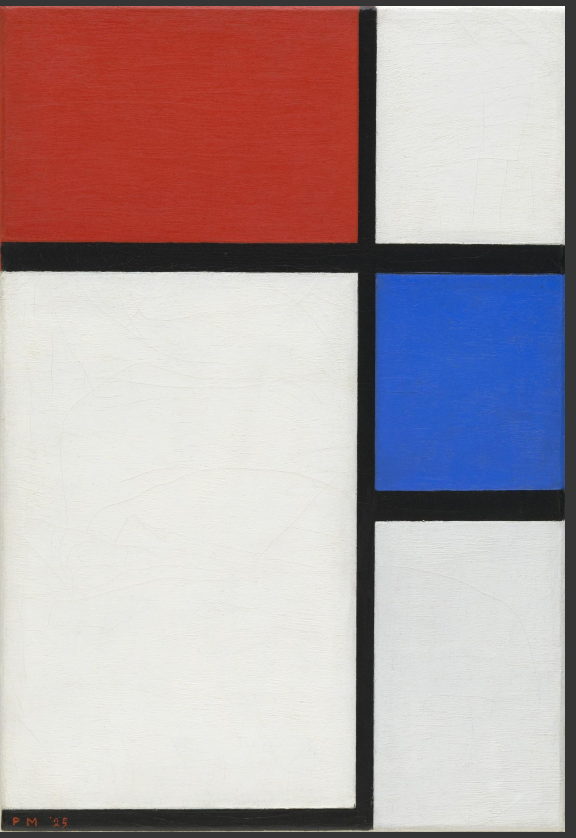
“Architecture is about the important stuff. Whatever that is.”

–
Ralph Johnson

Red Hat

Introducing Multiplying Architecture

Red Hat



What is important for the Multiplying Architecture is the ***abstraction***.

Red Hat

The Abstractions

core

Channel

Top level abstraction that represents the hosting environment, like a website or a desktop application.

Envelope

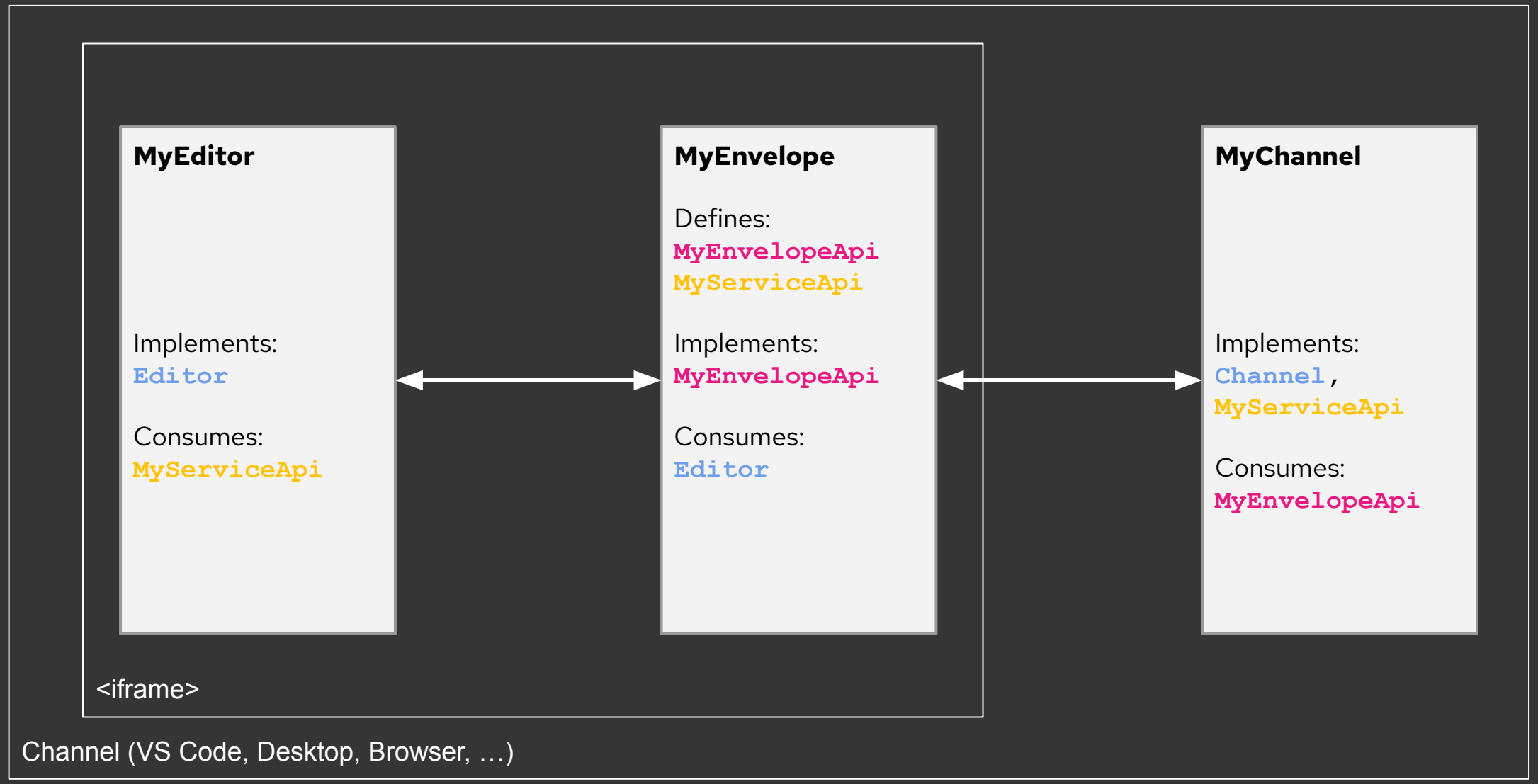
Enable transparent communication between Components (View/Editor) and Channel

View

View is a portable set of widgets that are exposed as an unit to the Channel through the Envelope.

Editor

Editor is a specialized type of View, that gets a file content as input and is able to serve the content state back to the Channel through the Envelope.


MyEditor
Implements:
Editor
Consumes:
MyServiceApi
MyEnvelope
Defines:
MyEnvelopeApi
MyServiceApi
Implements:
MyEnvelopeApi
Consumes:
Editor
MyChannel
Implements:
Channel,
MyServiceApi
Consumes:
MyEnvelopeApi
<iframe>
Channel (VS Code, Desktop, Browser, …)

Envelope Advantages

micro frontend

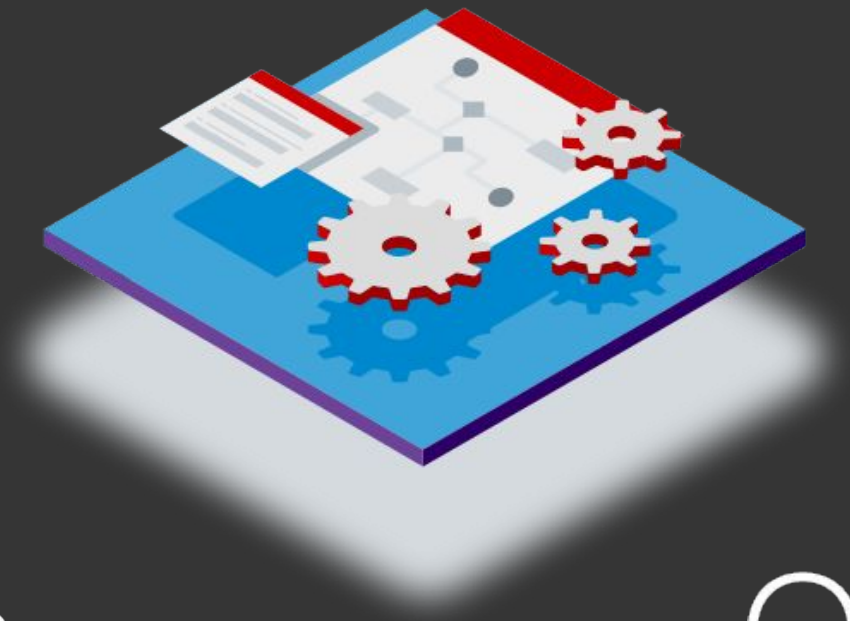
- Context Isolation (CSS and JS)
- Autonomous Teams
- Independent Release Cycles
- Type Safe Communication

Multiplying Architecture In Practice

Red Hat

KIE

jBPM



Drools

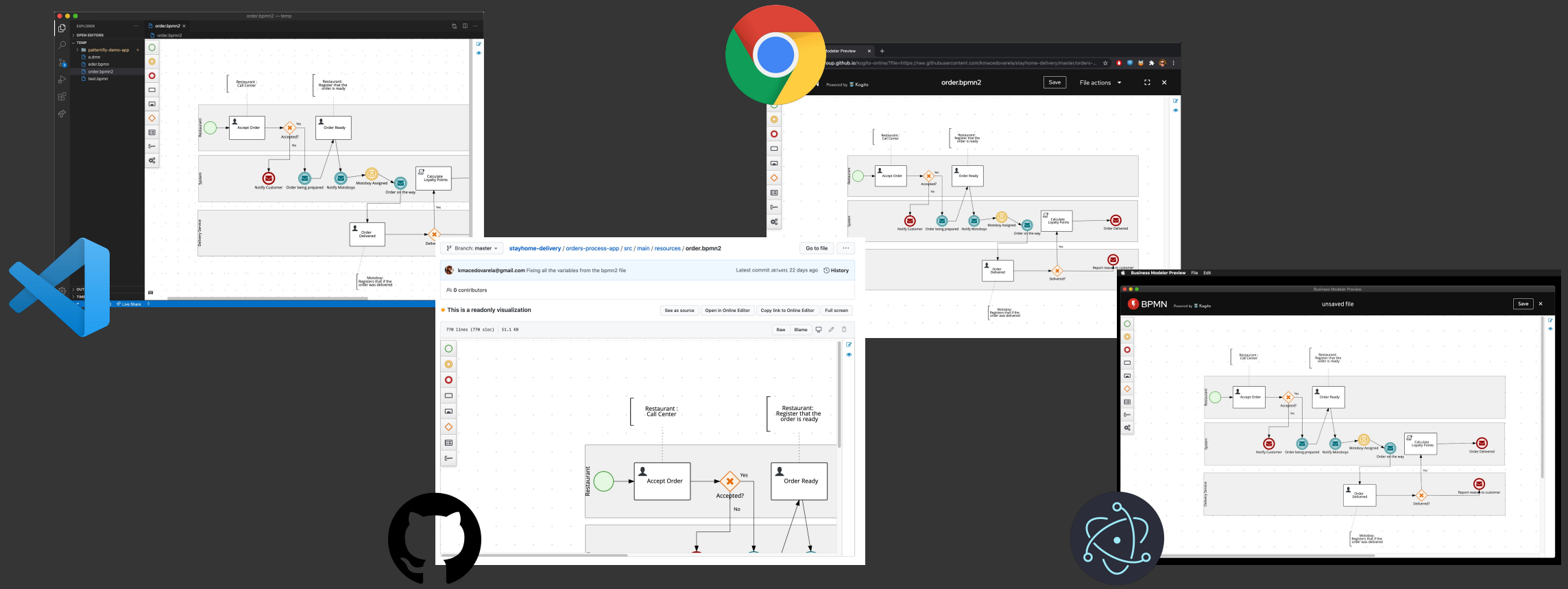
Kogito

OptaPlanner

Red Hat

KIE Tooling Channels

* Client side online editor

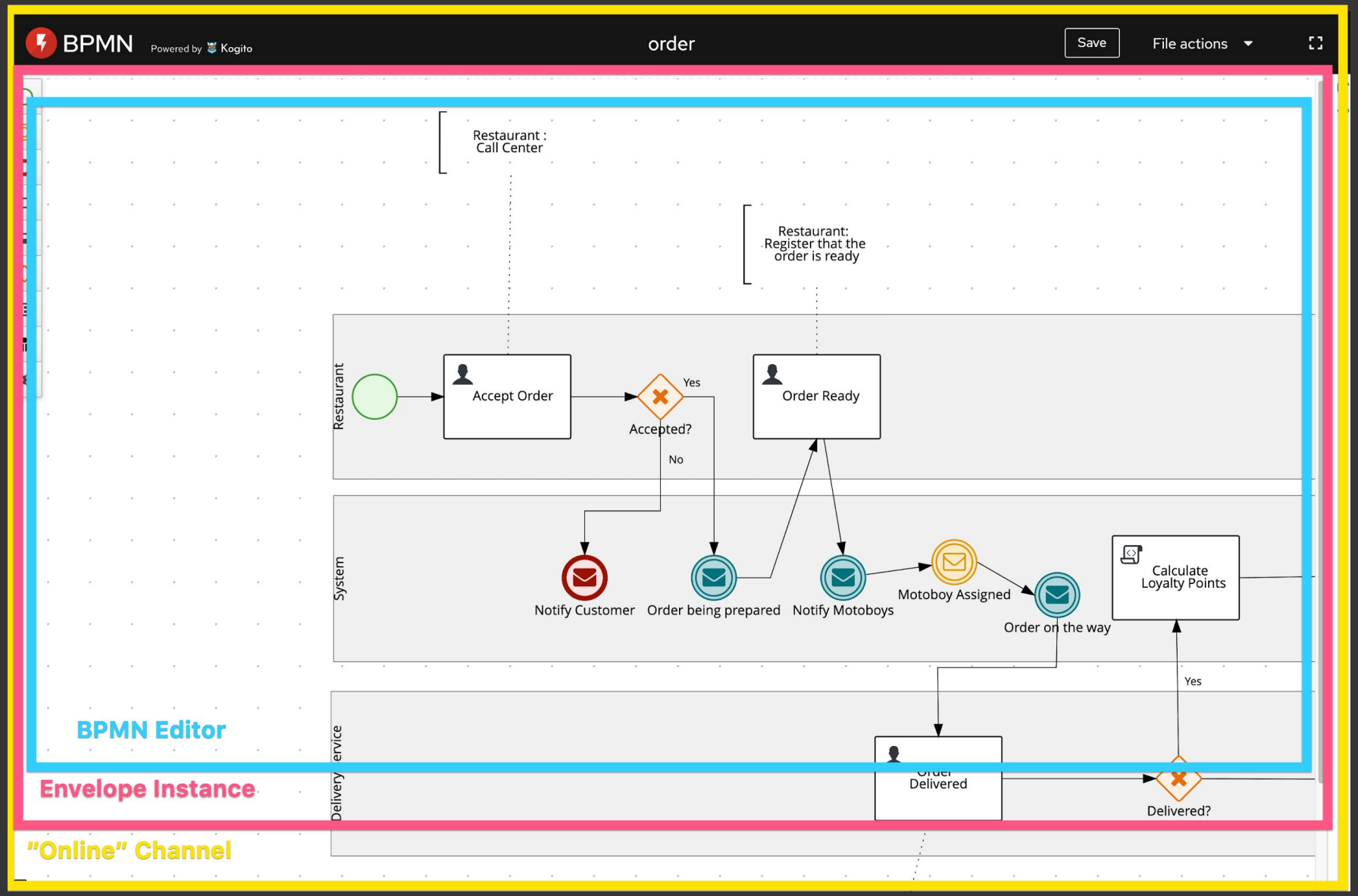


Red Hat

DEMO

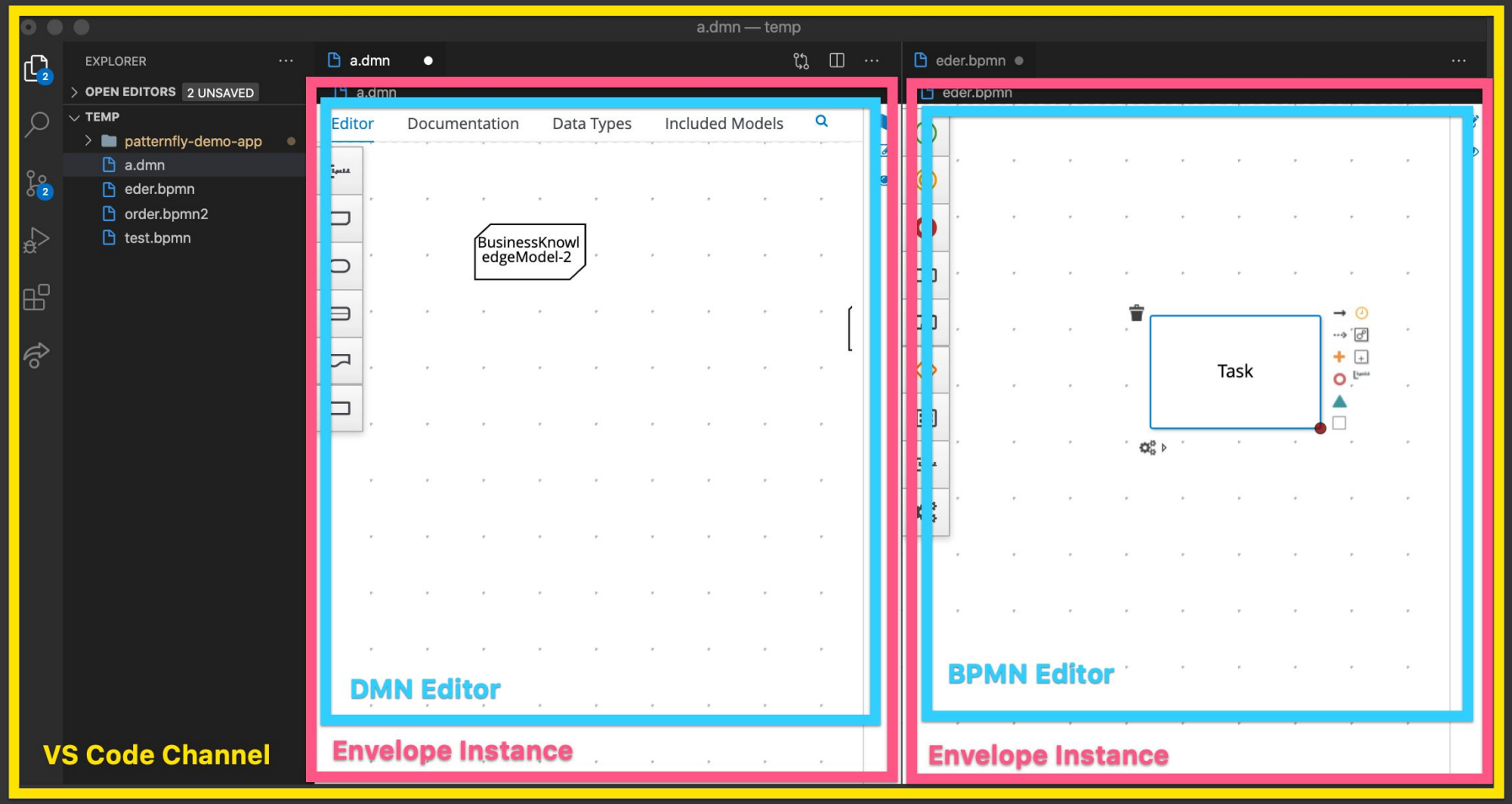
Red Hat

Online Channel



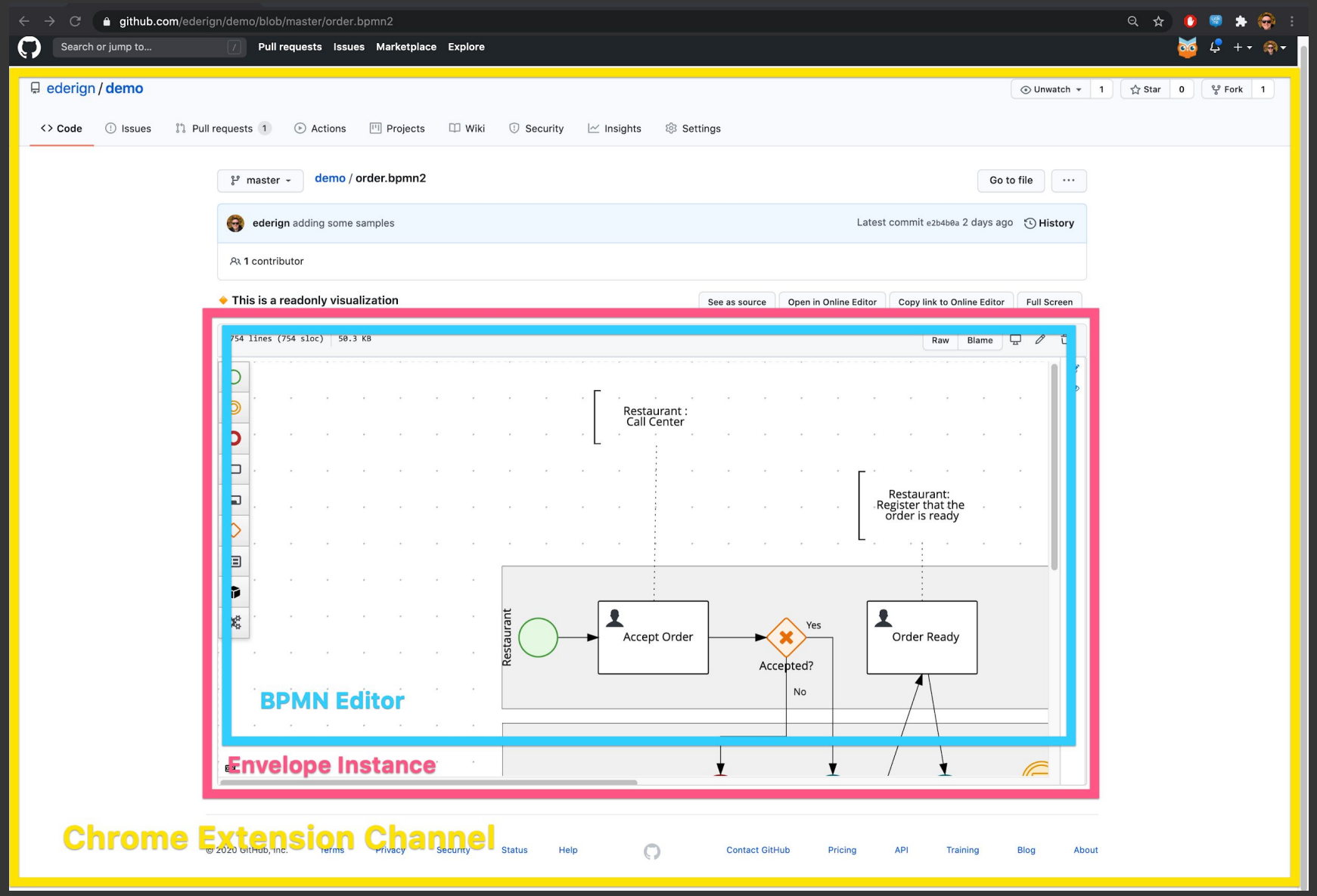
Red Hat

VSCode Channel



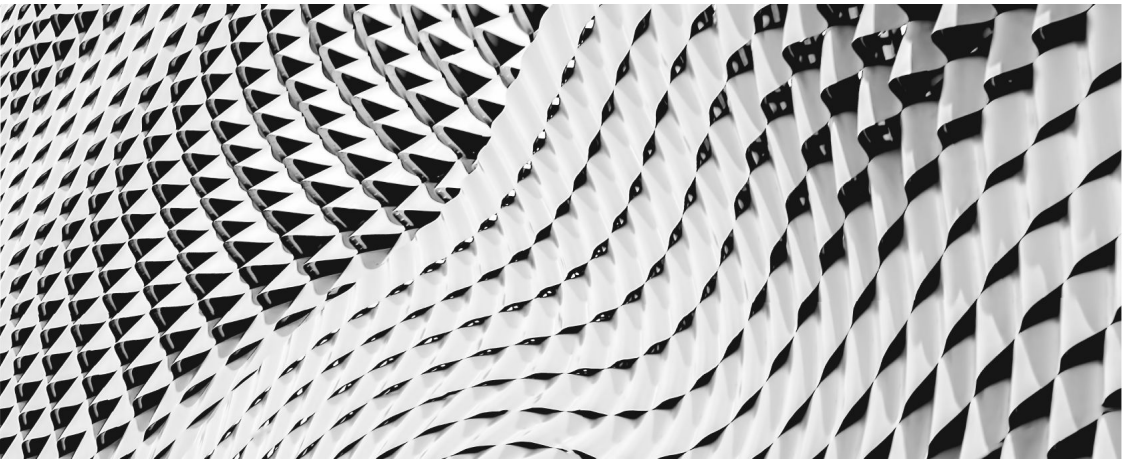
Red Hat

VSCode Channel

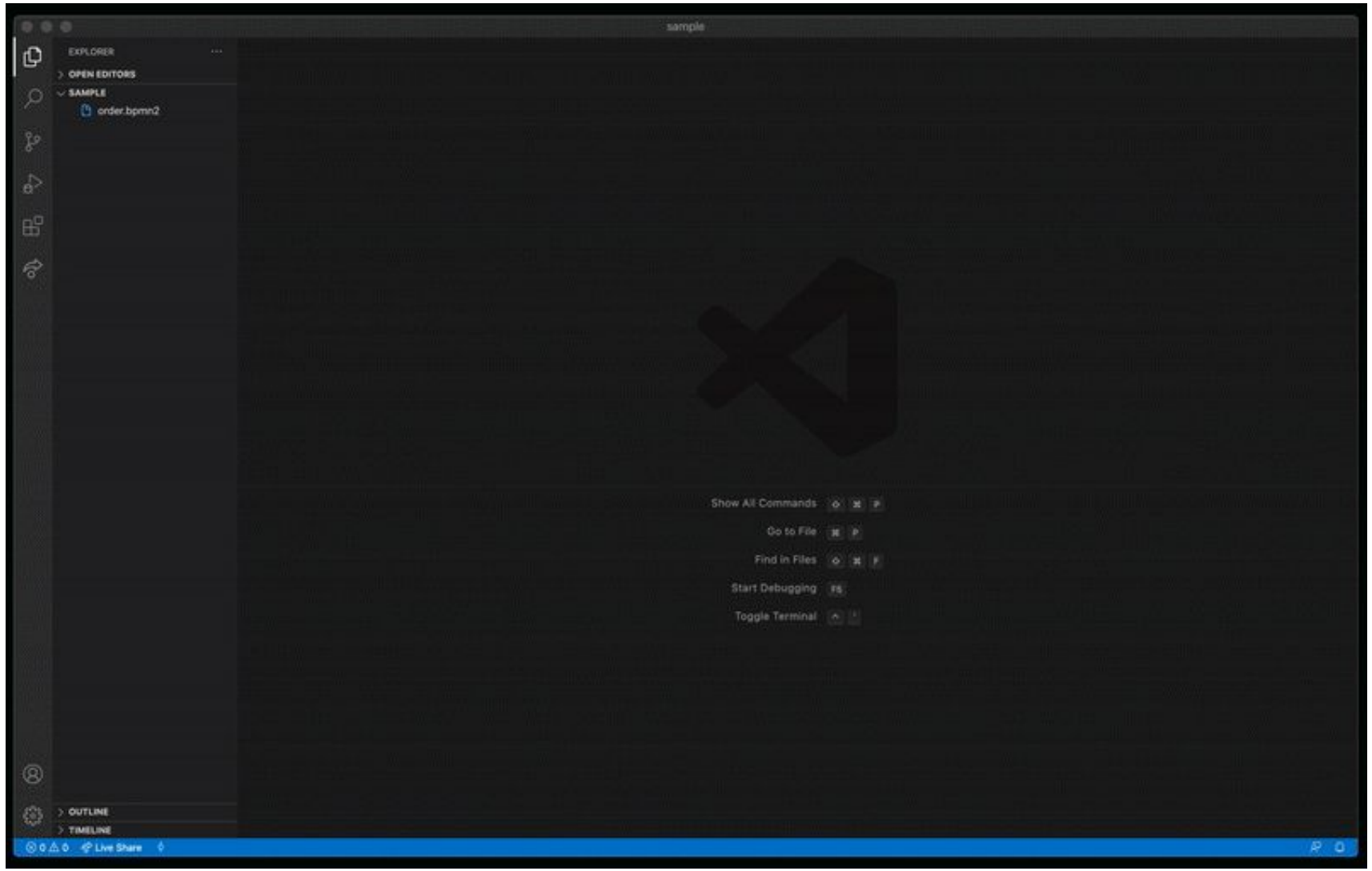


Red Hat

Feature Highlights



Red Hat

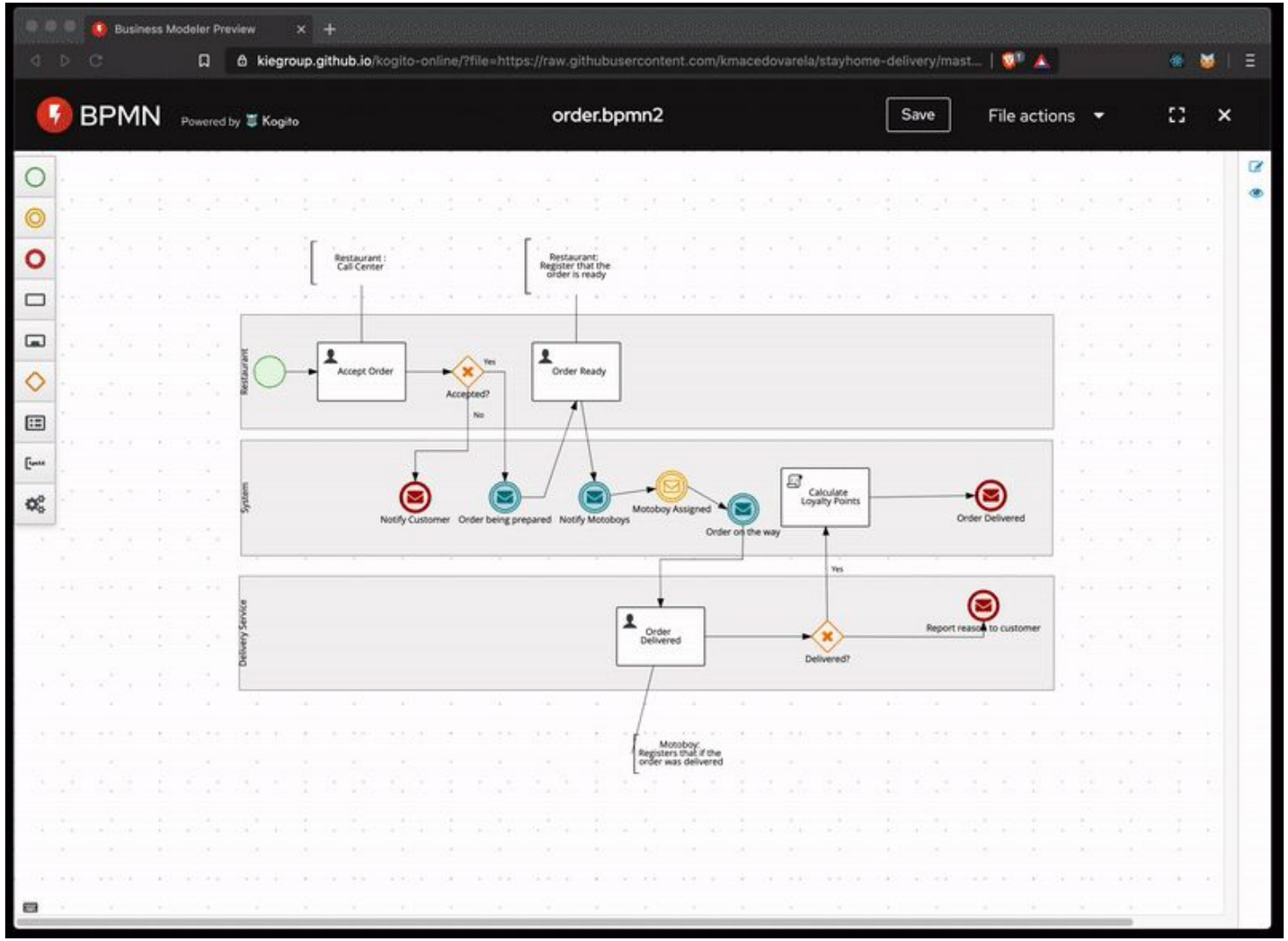


VSCode Native

Red Hat

Unified I/O API

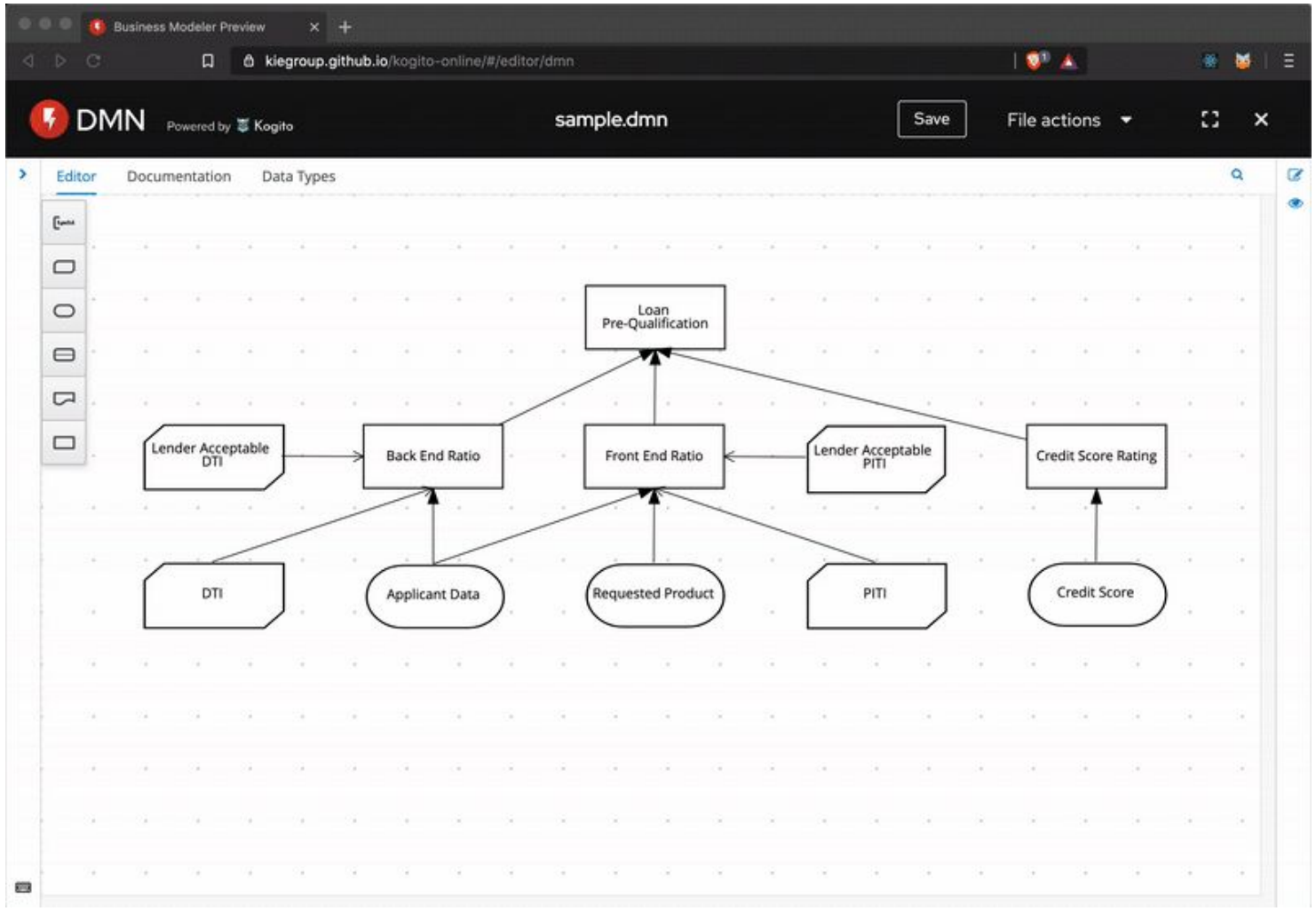
Read and Write content from multiple sources like GitHub, Gist, FileSystem, S3 (soon), etc.



State Control

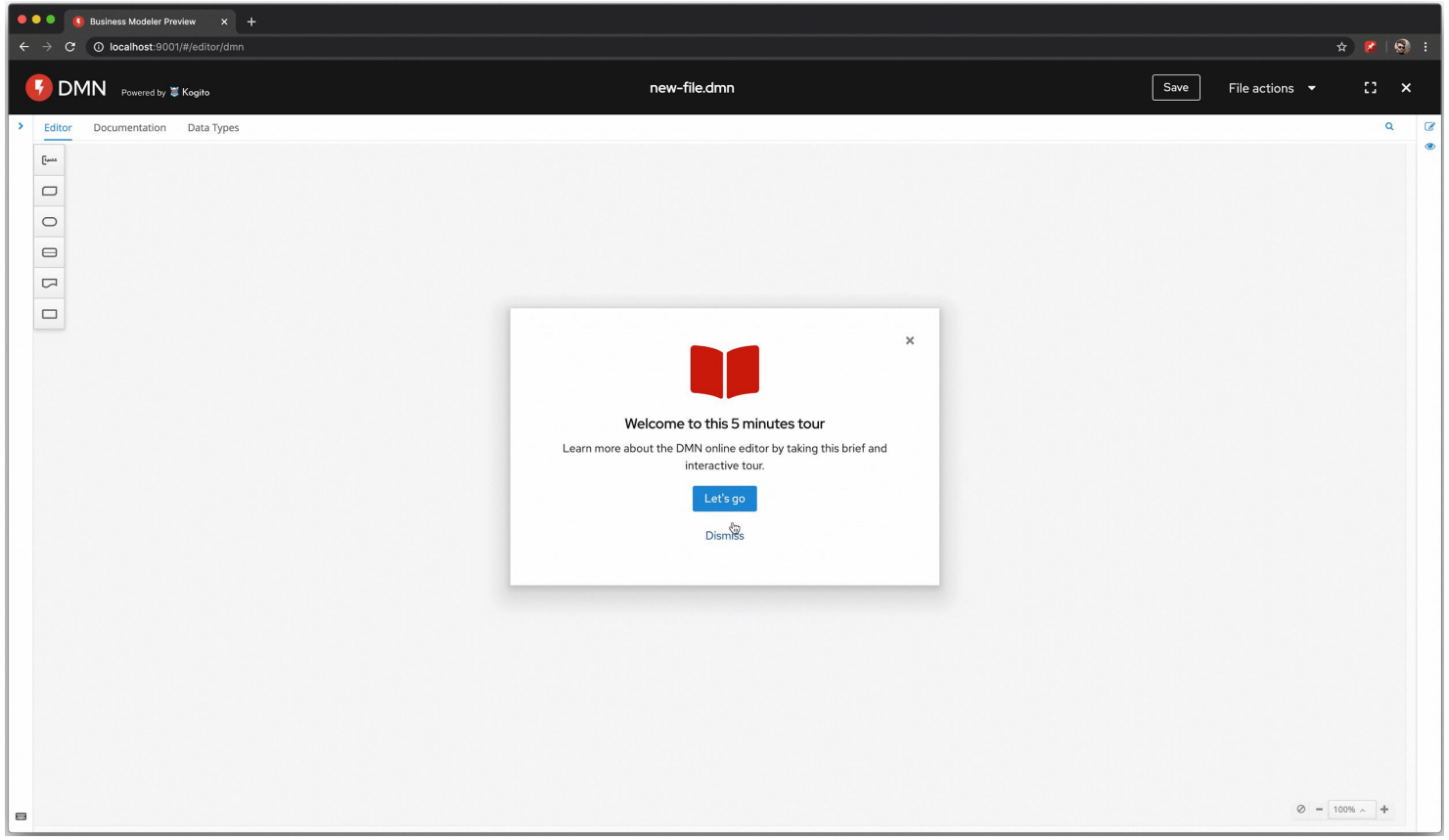
Cross channel support for Undo, Redo, and Dirty detection.

Red Hat



Keyboard Shortcuts

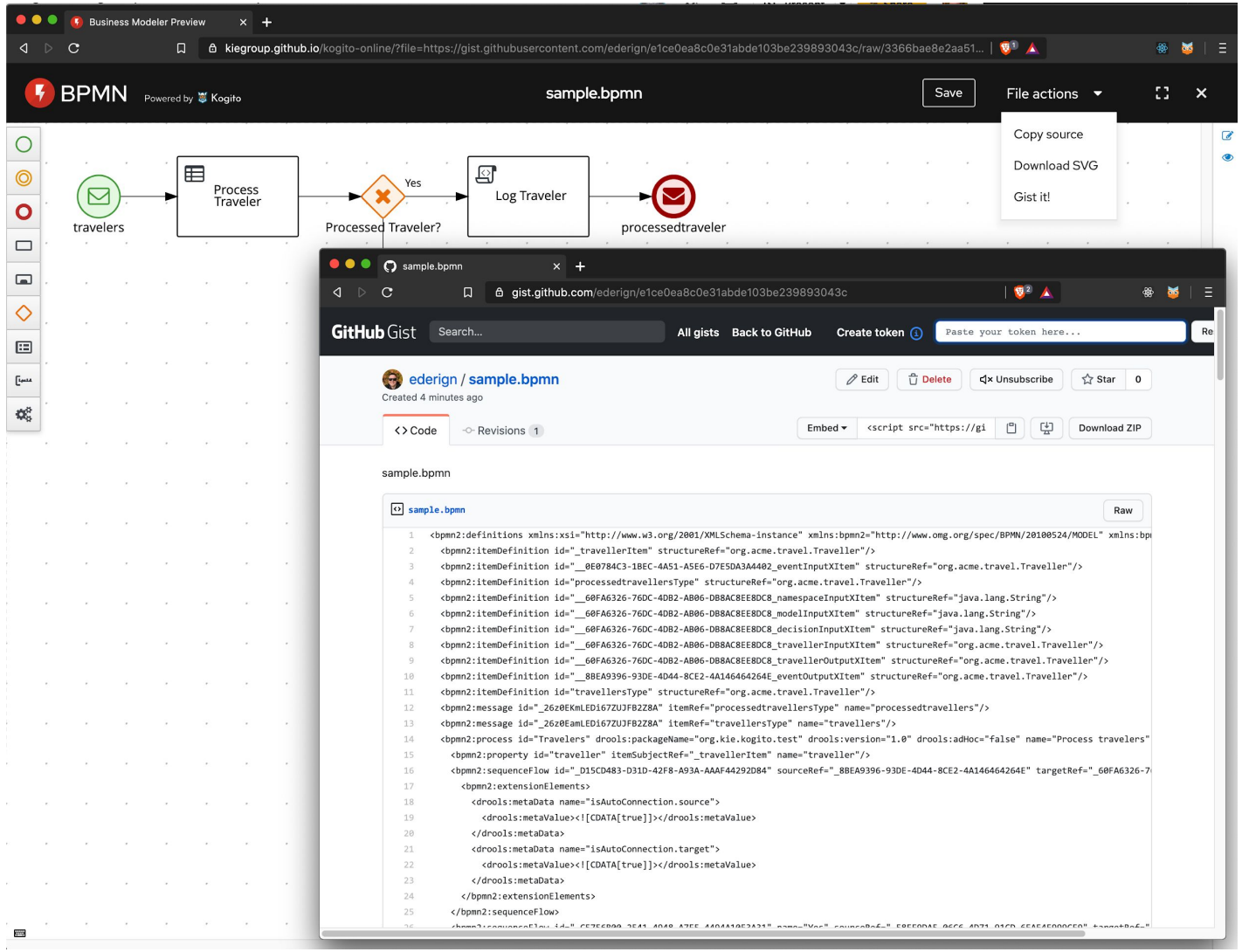
Red Hat



Guided Tour

Red Hat

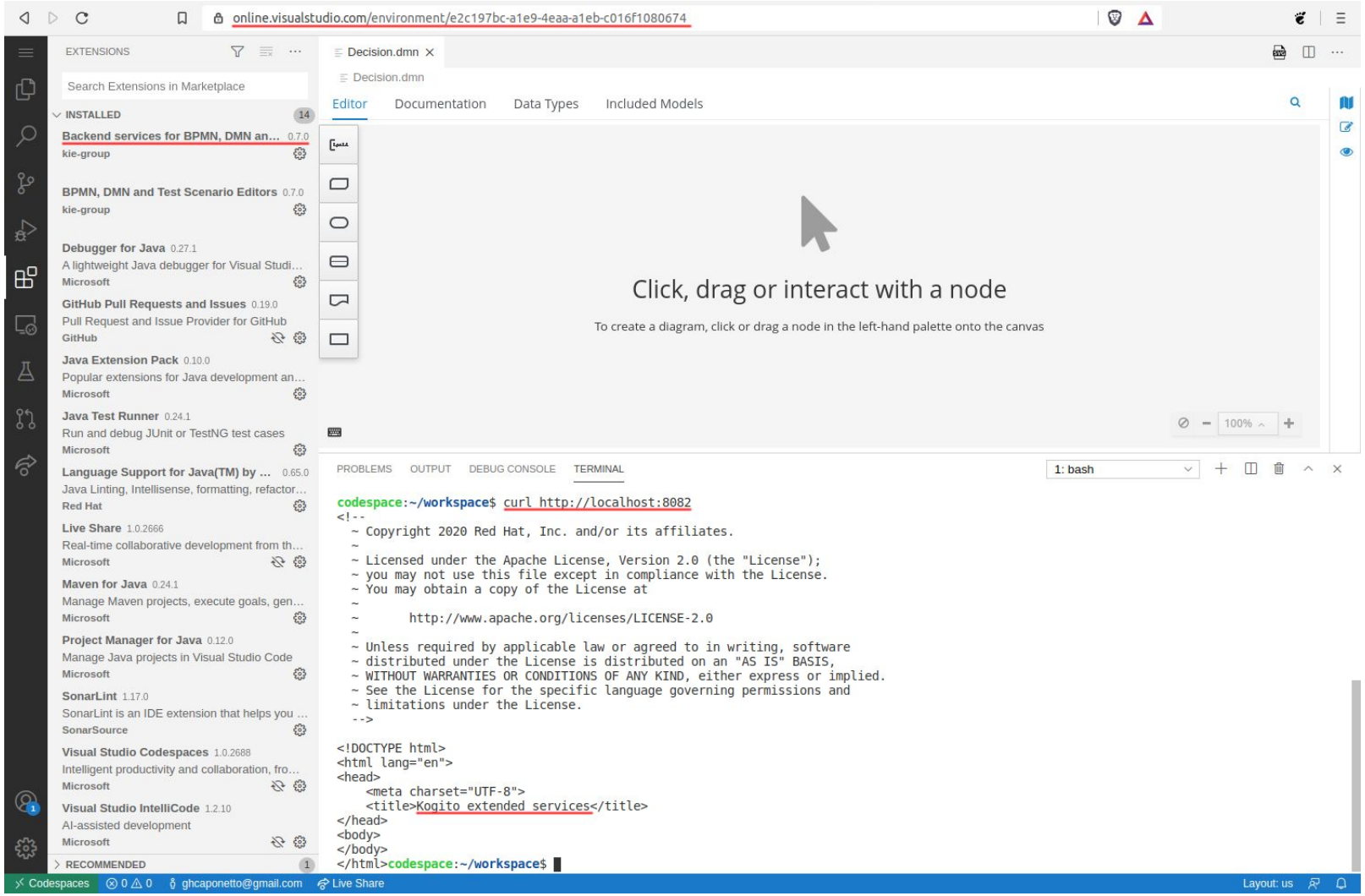
The Multiplying Architecture



Gist as storage

Pure client side mechanism to store content using GitHub Gists.

Red Hat



Backend Services

A pluggable infrastructure, able to augment the capabilities of the views and editors by enabling some backend dependent features.

Red Hat

Embedded Editors (soon)

```
const dmnEditor = kieTooling.dmnEditor('targetEmbeddedDiv', {standalone:true})

const bpmnEditor = kieTooling.bpmnEditor('targetEmbeddedDiv', {standalone:true})
```


Red Hat

Multiplying Architecture In Detail

https://github.com/kiegroup/kogito-tooling-examples

Red Hat

```typescript
/**
 * Editor component API. Basic Editor feature definitions.
 */
export interface EditorApi {
  setContent(path: string, content: string): Promise<void>;
  getContent(): Promise<string>;
  getPreview(): Promise<string | undefined>;
  getElementPosition(selector: string): Promise<Rect | undefined>;
  undo(): Promise<void>;
  redo(): Promise<void>;
}
```

```
export class SimpleReactEditorInterface extends Editor {
  private self: SimpleReactEditor;

  constructor(private readonly messageBus: EnvelopeBusInnerMessageHandler) {
    super("readonly-react-editor");
    this.af_isReact = true;
    this.messageBus = messageBus;
  }

  public getContent(): Promise<string> {
    return this.self.getContent();
  }

  public setContent(path: string, content: string): Promise<void> {
    return this.self.setContent(content);
  }
  ...
}
```

```
export class SimpleReactEditor extends React.Component<Props, State> {
  constructor(props: Props) {
    super(props);
    props.exposing(this);
    this.state = {
      content: ""
    };
  }
  ...

  public async setContent(content: string): Promise<void> {
    this.setState({ content: content });
  }

  public async getContent(): Promise<string> {
    return this.state.content;
  }

  public render() {
    return (
      <textarea
        style={{
          width: "100%",
          height: "100%",
          outline: 0,
          boxSizing: "border-box",
          border: 0,
          color: "black"
        }}
        value={this.state.content}
        onChange={(e: any) => this.updateContent(e.target.value)}
      />
    );
  }
}
```


Red Hat

```
export class SimpleReactEditorsRoutes implements Routes {
  public getRoutes() {
    return new Map<string, SimpleReactEditorsLanguageData>([
      [
        "txt",
        {
          type: "my-editor-type",
          anyData: "something"
        }
      ]
    ]);
  }
}
```


Red Hat

```
export function activate(context: vscode.ExtensionContext) {
  console.info("Extension is alive.");

  KogitoVsCode.startExtension({
    extensionName: "kogito-tooling-examples.vscode-extension-pack-simple-react",
    webviewLocation: "dist/webview/index.js",
    context: context,
    viewType: "kieKogitoWebviewSimpleEditors",
    getPreviewCommandId: "",
    router: new DefaultVsCodeRouter(context, new SimpleReactEditorsRoutes())
  });

  console.info("Extension is successfully setup.");
}
```

kogito-tooling-examples

EXPLORER

OPEN EDITORS

KOGITO-TOOLING-EXAMPLES

- node_modules
- packages
  - chrome-extension-pack-simple-react
  - simple-react-editors
  - vscode-extension-pack-simple-react
    - .vscode
    - dist
      - extension
      - fonts
      - images
      - webview
      - kogito_tooling_examples_vscode_extensi...
    - node_modules
    - src
      - extension
        - extension.ts
      - webview
    - .gitignore
    - .vscodeignore
    - LICENSE
    - package.json
    - README.md
    - tsconfig.json
    - webpack.config.js
- .gitignore
- lerna.json
- LICENSE
- package.json
- prettier.config.js
- README.md
- test.txt U
- tsconfig.json
- tslint.json
- update_version_to.js
- yarn.lock

OUTLINE

TIMELINE

NPM SCRIPTS

Show All Commands ⇧ ⌘ P

Go to File ⌘ P

Find in Files ⇧ ⌘ F

Start Debugging F5

Toggle Terminal ^ `

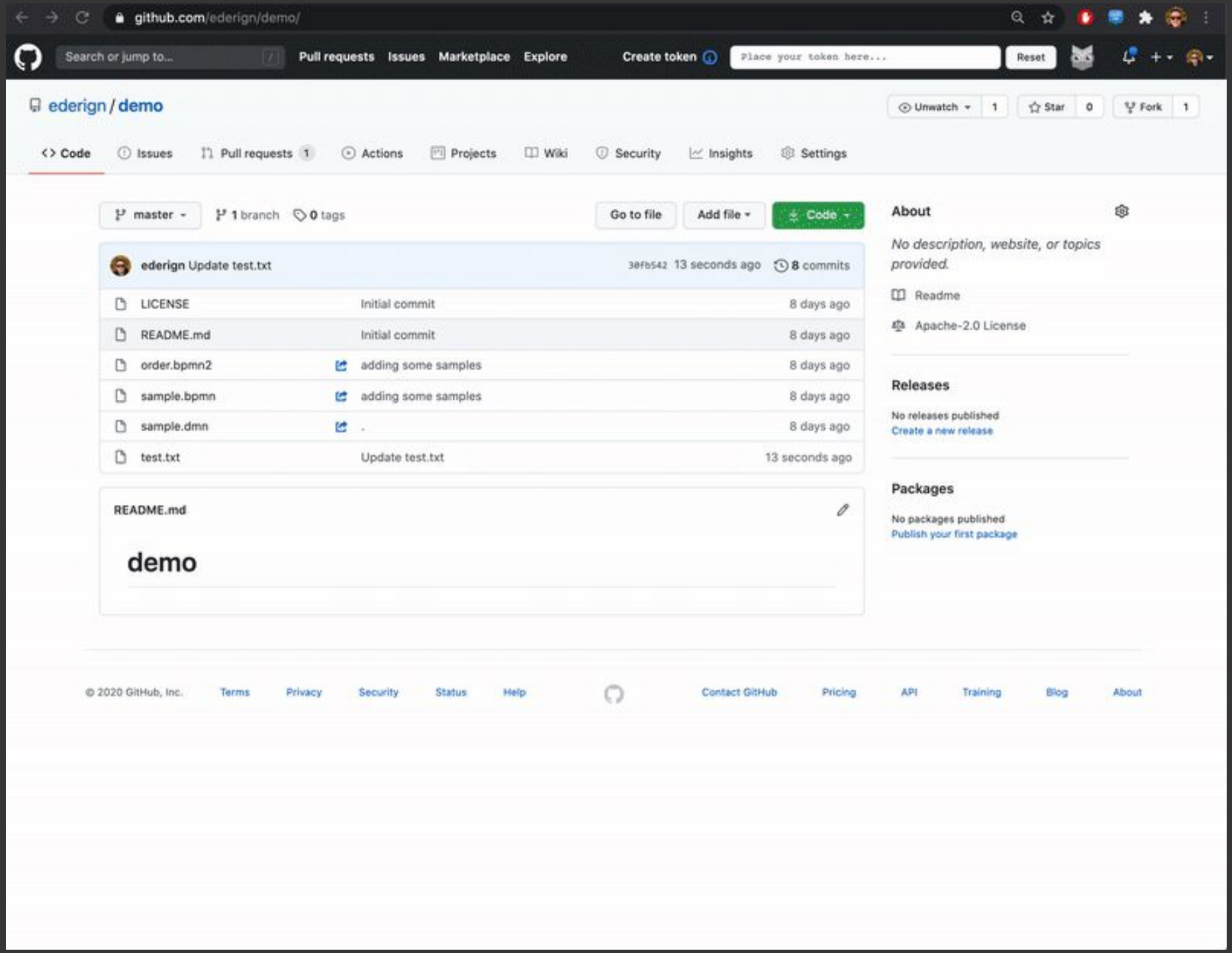
master* 0 0 Live Share Go Live

Red Hat

```
import { startExtension, DefaultChromeRouter } from "@kogito-tooling/chrome-extension";
import { SimpleReactEditorsRoutes } from "simple-react-editors";

startExtension({
  name: "KIE :: Kogito Simple React Editor",
  editorIndexPath: "envelope/index.html",
  extensionIconUrl: chrome.extension.getURL("/resources/kie-icon.png"),
  githubAuthTokenCookieName: "github-oauth-token-kie-editors",
  router: new DefaultChromeRouter(new SimpleReactEditorsRoutes())
});
```

In Detail



Red Hat

Why do we need a **new** architecture?

Red Hat

Goals of The Multiplying Architecture

solve a problem

Multiple Distributions

The origin of multiplying architecture is rooted in the need to distribute the same set of components in a myriad of platforms.

Minimize code changes

The components to be distributed should be preserved untouched and with avoiding feature flags.

Bridge

It has to embrace different generations of technology stack.

Red Hat

Questions

Red Hat

Thank you

Eder Ignatowicz

Principal Software Engineer

@ederign

linkedin.com/company/red-hat

youtube.com/user/RedHatVideos

facebook.com/redhatinc

twitter.com/RedHat

Red Hat