A Bright Horizon with Vue & Vite

How new tools and techniques affect the way we view and build applications

Objectives for Today

Objectives for Today

- Introduction to Vue 3, with a focus on:
  - Composition API
  - `<script setup>`
- Comparison of traditional bundlers and dev servers
- Outlook into the future with `unplugin`

🤷 Not Part of This Talk

- Comprehensive list of changes and additions between Vue 2 and 3
- 🍍 **Pinia** as a better alternative to Vuex
  - Maybe part of another talk

😯 Notable New Features

Some of the New Features to Keep an Eye on in Vue 3

- Virtual DOM rewrite for better performance
- Rewritten in TypeScript from scratch
- Composition API
- Teleport
- Fragments
- SFC composition API syntax sugar (`<script setup>`)
- SFC state-driven CSS variables (`v-bind` in `<style>`)
- Suspense

👉 Head over to the migration page for an in-depth list

Fragments

- Components now have official support for multi-root node components
- Virtual elements won't be rendered in the DOM tree

2.X SYNTAX

```
<!— Layout.vue —>
<template>
  <div>
    <header><!— … —></header>
    <main><!— … —></main>
    <footer><!— … —></footer>
  </div>
</template>
```

3.X SYNTAX

```
<!— Layout.vue —>
<template>
  <header><!— … —></header>
  <main><!— … —></main>
  <footer><!— … —></footer>
</template>
```

🦕 Breaking Changes

The Ones Most Relevant to You

- Global API
  - Global Vue API is changed to use an application instance

```
import { createApp } from 'vue'
const app = createApp({})
```

  - Global and internal APIs have been restructured to be tree-shakable
- Template directives
  - `v-model` usage on components has been reworked
- Components
  - Async components now require `defineAsyncComponent` method to be created

👉 List of breaking changes from 2.x

The Current Horizon of Vue 2

```
<template>
  <!--  -->
</template>

<script>
import Vue from 'vue'
import Foo from './components/Foo.vue'
import { mixinBar } from './mixins/bar'

export default Vue.extend({
  components: { Foo },
  mixins: { mixinBar },
  data() {
    return {
      // ...
    }
  },
  methods: {
    // ...
  },
  created() {
    // ...
  },
  destroyed() {
    //
```

THE PROBLEM

- Logic isn't generally grouped by feature
- Scroll back and forth to follow what's happening in the SFC
- Extensibility
- TypeScript support

Composition API to the Rescue

First: Reactivity Fundamentals

DECLARING REACTIVE STATE

```
import { reactive } from 'vue'

// Returns a reactive copy of the object
const state = reactive({
  count: 0
})
```

- Objects inside `data()` are internally made reactive by `reactive()`
- Reactive conversion is "deep"—it affects all nested properties
- Based on ES2015 Proxy

CREATING STANDALONE REACTIVE VALUES AS `REFS`

```
import { ref } from 'vue'

const count = ref(0)
console.log(count.value) // 0

count.value++
console.log(count.value) // 1
```

- For standalone primitive values
- Stands for reactive **ref**erence

Composition API

OPTIONS API

```
export default {
  data() {
    return {
      dark: false,
      media: matchMedia('(prefers-color-scheme: dark)')
    }
  },
  methods: {
    toggleDark() {
      this.dark = !this.dark
    },
    update() {
      this.dark = this.media.matches
    }
  },
  created() {
    this.media.addEventListener('change', this.update)
    this.update()
  },
  destroyed() {
    this.media.removeEventListener('change', this.update)
  }
}
```

COMPOSITION API

```
import { ref, onUnmounted } from 'vue'

export default {
  setup() {
    const media = matchMedia('(prefers-color-scheme: dark)
    const dark = ref(media.matches)

    const update = () => dark.value = media.matches
    const toggleDark = () => dark.value = !dark.value

    media.addEventListener('change', update)
    onUnmounted(() => {
      media.removeEventListener('change', update)
    })

    return { dark, toggleDark }
  }
}
```

Composability

```
import { useDark } from './useDark'

export default {
  setup() {
    const { dark, toggleDark } = useDark()
    return {
      dark,
      toggleDark
    }
  }
}
```

```
import { ref, onUnmounted } from 'vue'

export function useDark() {
  const media = matchMedia('(prefers-color-scheme: dark)')
  const dark = ref(media.matches)

  const update = () => dark.value = media.matches
  const toggleDark = () => dark.value = !dark.value

  media.addEventListener('change', update)
  onUnmounted(() => {
    media.removeEventListener('change', update)
  })

  return { dark, toggleDark }
}
```

- Outsourecable into seperate hook

Desirable: Less Verbosity

`<script setup>` Syntax

- Compile-time syntactic sugar for using Composition API inside SFCs
- Recommended syntax with 👆
- Advantages over the normal `<script>` syntax:
  - Less boilerplate
  - Declare props and emitted events using pure TypeScript
  - Better runtime performance (the template is compiled into a render function in the same scope, without an intermediate proxy)
  - Better IDE type-inference performance (less work for the language server to extract types from code)

`<script setup>` Syntax

`<script>`

```
<script>
import { ref, computed } from 'vue'
import MyButton from './MyButton.vue'

export default {
  components: {
    MyButton,
  },
  setup() {
    const counter = ref(0)
    const doubled = computed(() => counter.value * 2)

    function inc() {
      counter.value += 1
    }

    return { counter, doubled, inc }
  }
}
</script>
```

`<script setup>`

```
<script setup>
import { ref, computed } from 'vue'
import MyButton from './MyButton.vue'

const counter = ref(0)
const doubled = computed(() => counter.value * 2)

function inc() {
  counter.value += 1
}
</script>
```

- Variables, functions, and components are directly available in the template
- Now stable in Vue 3.2

`v-bind()` in `<style>`

WITHOUT

```
<template>
  <button :style="{ color: buttonColor }">
    My Button
  </button>
</template>

<script>
export default {
  data() {
    return {
      buttonColor: 'green'
    }
  }
}
</script>

<style>
button {
  border-radius: 4px;
}
</style>
```

WITH V-BIND()

```
<template>
  <button>My Button</button>
</template>

<script setup>
const buttonColor = ref('green')
</script>

<style>
button {
  border-radius: 4px;
  color: v-bind(buttonColor);
}
</style>
```

The New Default Tooling – Vite

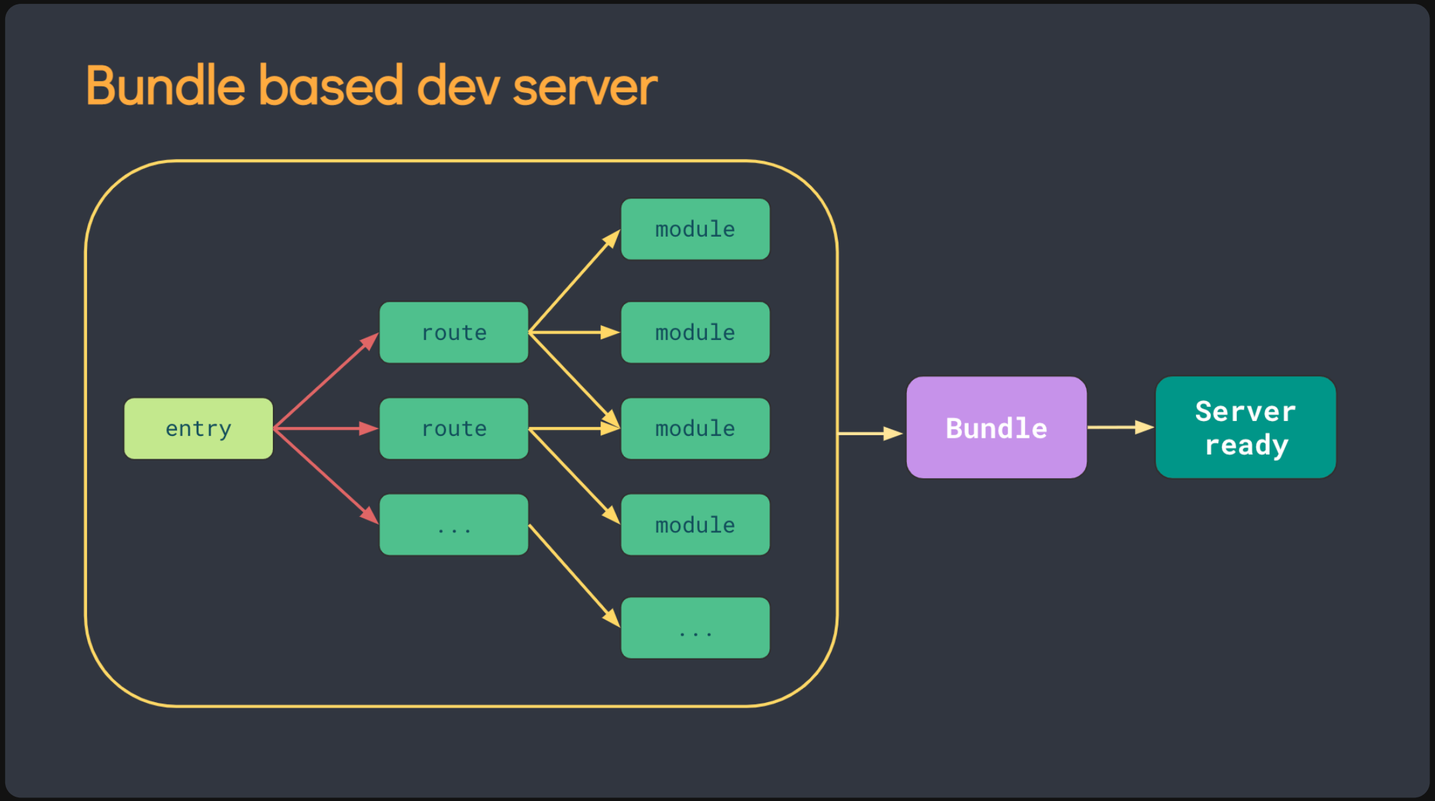
What’s Vite?

Bundlers

Webpack Rollup

BUILD FIRST

- Designed for **production build** first
- Needs to bundle the entire project to start the dev server
- Complex configuration
- HMR gets slower as projects grow

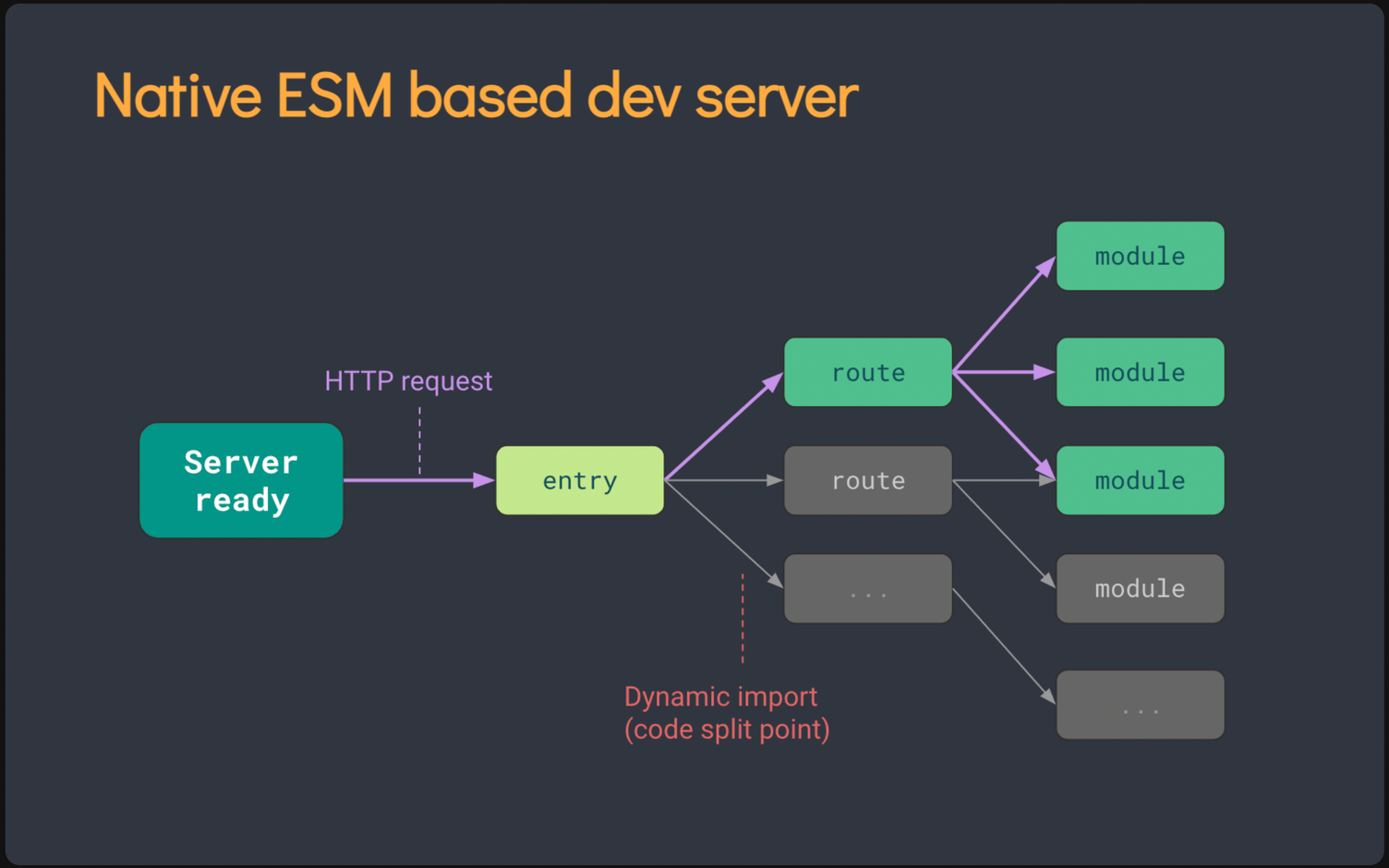


Dev Servers

Snowpack Vite

DEV FIRST

- Design for Web development
- Native ESM + unbundled
- Server is ready immediately
- On-demand
- Instant HMR
- And much more



Why use Vue 3 with Vite?

For better performance and better DX

New Ways to View Vue 👀

Using Components

```
<template>
  <my-container>
    <my-button />
    <my-input />
  </my-container>
</template>

<script>
import MyContainer from '../components/MyContainer.vue'
import MyButton from '../components/MyButton.vue'
import MyInput from '../components/MyInput.vue'

export default {
  components: {
    MyContainer,
    MyButton,
    MyInput,
  }
}
</script>
```

TO USE A COMPONENT

- Import and name it
- Register the component
- Use it in the template

THE PROBLEM

- Verbose
- Names are repeated at least 4 times

Using Components

```
<template>
  <my-container>
    <my-button />
    <my-input />
  </my-container>
</template>

<script setup>
import MyContainer from '../components/MyContainer.vue'
import MyButton from '../components/MyButton.vue'
import MyInput from '../components/MyInput.vue'
</script>
```

WITH `<SCRIPT SETUP>`

- Imports will be available directly in the template
- No longer need to register the components

BUT ...

- The component name is still repeated 3 times

Components Auto Importing

antfu/vite-plugin-components

Using antfu/vite-plugin-components

```
<template>
  <my-container>
    <my-button />
    <my-input />
  </my-container>
</template>
```

That's it!

HOW?

- **Compile-time** components resolving
- Components auto-discovery under `src/components` directory

DIFFERENCES FROM GLOBAL REGISTRATION

- Code-splitting
- No manual registration
- Skipped runtime resolving

How the compilation works

```
<template>
  <my-container>
    <my-button />
    <my-input />
  </my-container>
</template>
```

... Will be compiled by `@vue/sfc-compiler` to:

```
import { resolveComponent as _resolveComponent } from "vue"
function render(_ctx, _cache) {
  const _component_my_button = _resolveComponent("my-button")
  const _component_my_input = _resolveComponent("my-input")
  const _component_my_container = _resolveComponent("my-container")

  return (_openBlock(), _createBlock(_component_my_container, null, {
    default: _withCtx(() => [
      _createVNode(_component_my_button),
      _createVNode(_component_my_input)
    ]), _: 1 /* STABLE */
  }))
}
```

Writing the Vite plugin

```
// vite.config.ts
export default {
  plugins: [{
    name: 'my-plugin',
    enforce: 'post',
    transform(code, id) {
      if (!id.endsWith('.vue')) return

      return code.replace(
        /_resolveComponent\("(.+?)"/g,
        (_, name) => {
          const component = findComponent(name)
          // inject import for component
          return component.path
        })
    }
  }]
}
```

- Use `enforce: post` to ensure the plugin runs after Vue's compilation
- Use `transform` hook to modify the code
- Filter out files that are not Vue
- Replace the `_resolveComponent` usage to real component import

Read Vite Plugin API Documentation for more

The Result

```
import { resolveComponent as _resolveComponent } from "vue"

function render(_ctx, _cache) {
  const _component_my_button = _resolveComponent("my-button")
  const _component_my_input = _resolveComponent("my-input")
  const _component_my_container = _resolveComponent("my-container")

  return () => /* ... */
}
```

After:

```
import { resolveComponent as _resolveComponent } from "vue"
import _component_my_button from "../components/MyButton.vue"
import _component_my_input from "../components/MyInput.vue"
import _component_my_container from "../components/MyContainer.vue"

function render(_ctx, _cache) {
  return () => /* ... */
}
```

Use Icons

antfu/vite-plugin-icons

CURRENT POSSIBILITIES TO USE ICONS

- **Icon fonts**
  - Entire icon set needs to be shipped
  - Flash-of-unstyled-content (FOUC)
- **SVG**
  - Needs to be imported manually
  - Can't change color when used as images
- **Icon components**
  - Dependency limits on what the icon set provides
  - Needs to be registered manually

A BETTER SOLUTION?

Inspired by Vite's on-demand mindset, we could actually bundle the icons at compile-time when as need:

```
<template>
  <MdiAlarm />
  <FaBeer style="color: orange" />
  <TearsOfJoy />
</template>

<script setup>
import MdiAlarm from '~icons/mdi/alarm'
import FaBeer from '~icons/fa/beer'
import TearsOfJoy from '~icons/twemoji/face-with-tears-of-joy'
</script>
```

Powered by Iconify, **10,000+ icons** from 100+ popular icon sets in a unified syntax.

```
import Icon from '~icons/[collection]/[id]'
```

On-demand Icons

- Only the icons actually used will be bundled
- Almost any icons available to use
- Directly stylable via `class` and `style`
- SSR / SSG friendly

```
<template>
  <MdiAlarm />
  <FaBeer style="color: orange" />
  <TearsOfJoy />
</template>
```

Vite Ecosystem

An Excerpt

- `antfu/vite-plugin-components` - Components auto-import
- `antfu/vite-plugin-auto-import` - API auto-import
- `antfu/vite-plugin-icons` - On-demanded icons solution
- `hannoeru/vite-plugin-pages` - File-based routing
- `windicss/vite-plugin-windicss` - Windi CSS (on-demand Tailwind CSS)

… And many more

`vitejs/awesome-vite`

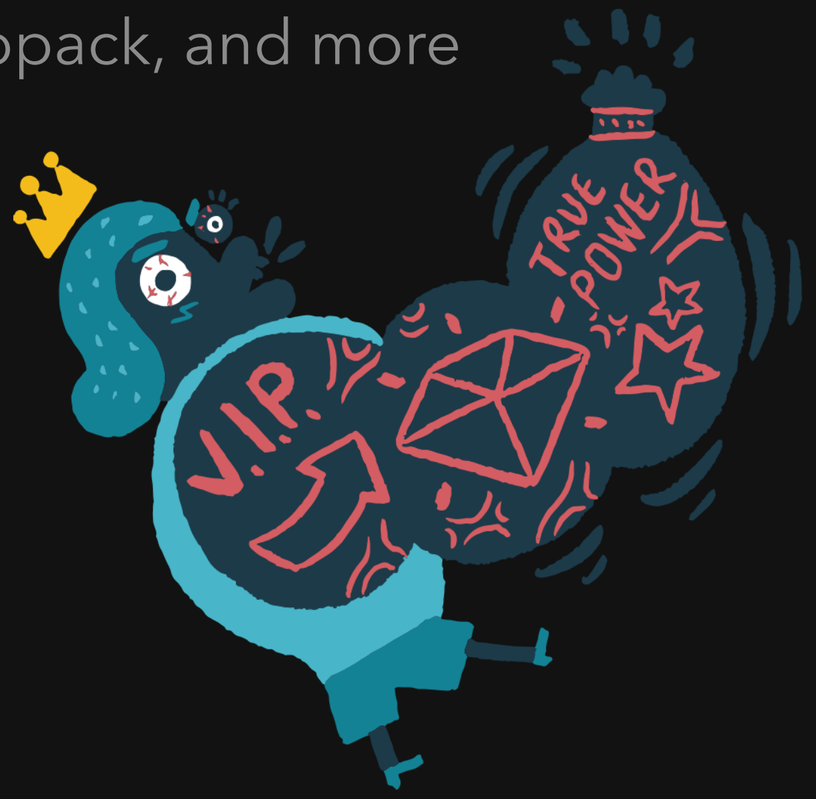
Vite has inspired many new plugins and better ways to improve DX

Available for Your Existing Projects Right Now

Introducing **unplugin**

A unified plugin system for Vite, Rollup, Webpack, and more

Written once – running on:



Vite Plugins → Unplugins

`vite-plugin-components` → `unplugin-vue-components`

`vite-plugin-auto-import` → `unplugin-auto-import`

For Vue / React / Svelte / Vanila / Any framework

`vite-plugin-icons` → `unplugin-icons`

- Vue
- React
- Preact
- Svelte
- SolidJS
- Web Components
- Vanila
- ...

+

- Vite
- Nuxt
- Next.js
- Rollup
- Vue CLI
- Webpack
- ...

+

- Carbon Icons
- Material Design Icons
- Unicons
- Twemoji
- Tabler
- BoxIcons
- EOS Icons
- ...

What about Vue 2?

Still covered!

Vue 2

POLYFILLS

- Composition API: `@vue/composition-api`
- `<script setup>` & Ref sugar: `unplugin-vue2-script-setup`

VITE SUPPORT

- `vite-plugin-vue2`
- `nuxt-vite`

DX ENHANCEMENT

- `unplugin-vue-components`
- `unplugin-auto-import`
- `unplugin-icons`

To Sum It Up

This is what you could get in Vue 2, Nuxt 2, Vue CLI, Vue 3, Vite:

```vue
<template>
  <button>
    <IconSun v-if="dark" />
    <IconMoon v-else />
  </button>
</template>

<script>
import IconSun from '@some-icon-set/sun'
import IconMoon from '@some-icon-set/moon'

export default {
  components: {
    IconSun,
    IconMoon,
  },
  data() {
    return {
      dark: false,
      media: matchMedia('(prefers-color-scheme: dark)')
    }
  },
  methods: {
```

→

```vue
<script setup>
const dark = useDark()
</script>

<template>
  <button>
    <IconSun v-if="dark" />
    <IconMoon v-else />
  </button>
</template>
```

Starter Templates

Project templates that have plugins mentioned previously

`antfu/vitesse`	Opinionated Vue 3 + Vite Starter template
`antfu/vitesse-nuxt`	Vitesse experience on Nuxt 2
`antfu/vitesse-webext`	Vitesse for Web Extensions

TRY IT NOW!

```
npx degit antfu/vitesse
```

🤫 Spoiler: Nuxt 3 has many of these features built-in directly.

LEAN ERA_

Credits to Anthony Fu

Slides forked from his New ways to Vue talk at Vue.js London 2021.

He is a Vue & Vite core team member, as well as the creator of Slidev, VueUse, Vitesse, etc.

antfu

antfu7

antfu.me