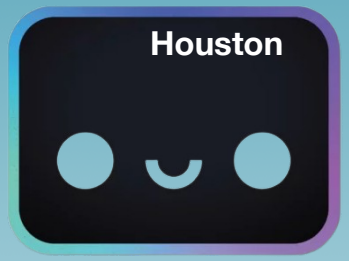


Astro - Ship Less!

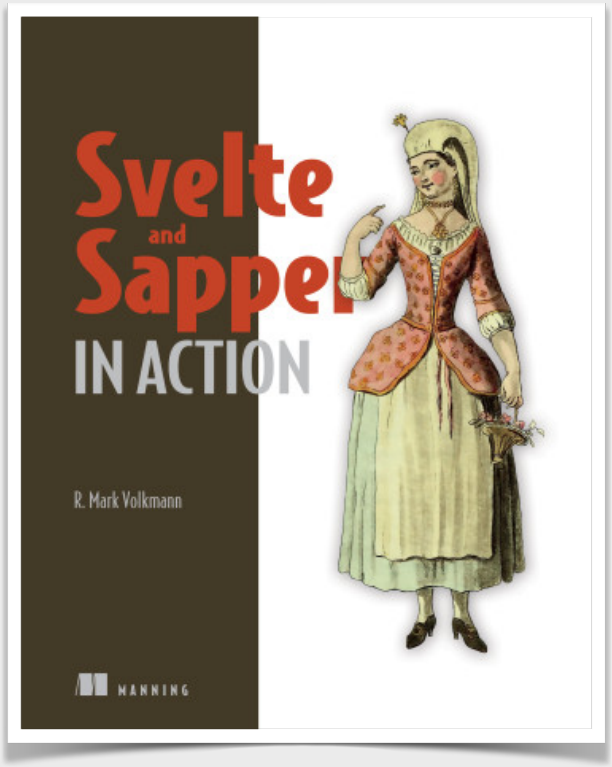
R. Mark Volkmann
Object Computing, Inc.
https://objectcomputing.com
mark@objectcomputing.com
@mark_volkmann

OBJECT COMPUTING
YOUR OUTCOMES ENGINEERED

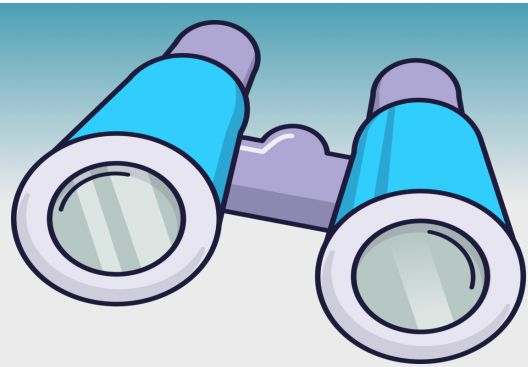
Slides at **https://github.com/mvolkmann/talks/**

About Me

- Partner and Distinguished Software Engineer at Object Computing, Inc. in St. Louis, Missouri USA
- 43 years of professional software development experience
- Writer and speaker
- Blog at https://mvolkmann.github.io/blog/
- Author of Manning book “Svelte ... in Action”



Astro Overview ...



- Free, open-source framework that can
  - generate static sites (**SSG**) at build time
  - build server-side rendered (**SSR**) sites on demand
  - define **API endpoints**
- https://alpinejs.dev/ MIT license
- Major focus is **shipping less JavaScript** code to browsers and doing more work on server
- **Can use many kinds of UI components** including Astro, Alpine, Lit, Preact, React, SolidJS, Svelte, Vue, WebComponents, and more

... Astro Overview

- Supports SSR **adapters** for Cloudflare, Netlify, Node, and Vercel
  - and many more community supported adapters
- Provides **integrations** with Tailwind for CSS styling
  - and many more packages
- Provides **file-based routing**
- Created by **Fred K. Schott**
  - previously worked on WebComponents at Google and was on Polymer team
  - created Snowpack, “a lightning-fast frontend build tool”, superseded by Vite
- Astro development is **managed by** “The Astro Technology Company”
  - founded in January 2022 with $7M in seed funding

Islands Architecture

- A way to “render HTML pages on the server, and inject placeholders or slots around dynamic regions that can then be hydrated on the client into small self-contained widgets, reusing their server-rendered initial HTML”
- Each island
  - is a bit of JavaScript-enabled interactivity and the water around them is static HTML
  - can use a different web UI framework

Pros

- Sends **less JavaScript** code (zero by default) to browsers resulting in **faster startup**
- **File-based routing** simplifies mapping pages and endpoints to URLs
- Provides **image optimization**
- Makes **static site generation (SSG)** easy
- Supports **server-side rendering (SSR)** of pages on demand
- Optimizes static content by confining dynamic behavior to "**islands**"
- Supports **TypeScript** which provides **IntelliSense** and error detection while writing code
- **Markdown** can be used to describe pages, components, and content

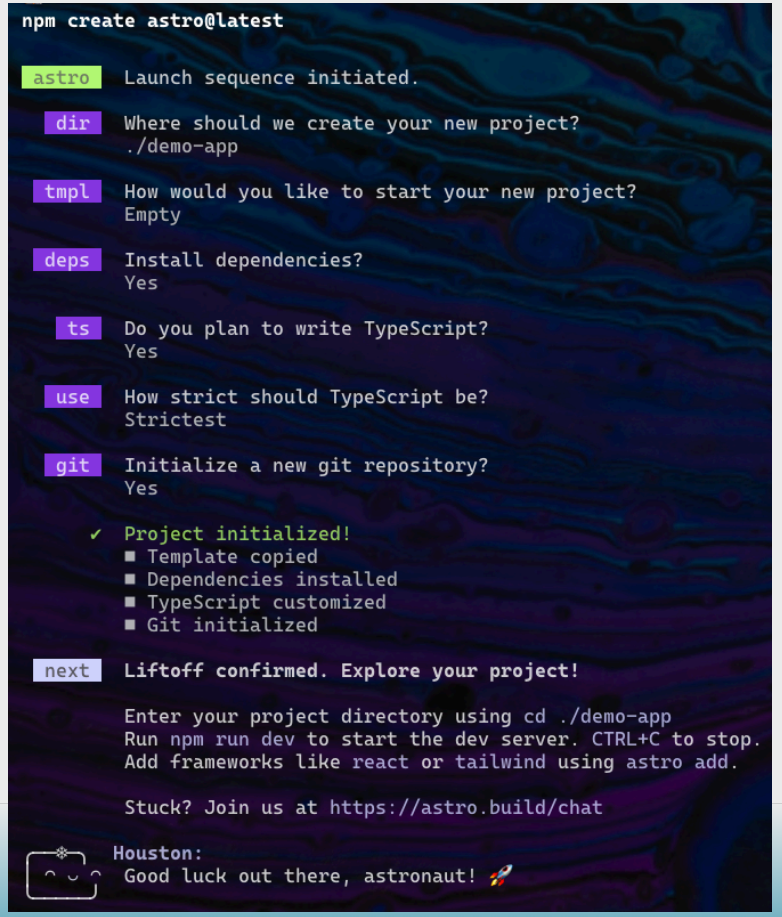
- Simple syntax for defining **Astro components** that leans into web fundamentals (HTML, CSS, and JavaScript)
- Can use components implemented in all the **popular web frameworks**
- Can use **content collections** to easily generate static pages from data at build time
- Integrates with many popular **content management systems** (CMS)
- Supports implementing **API endpoints** in JavaScript or TypeScript
- **Discord channel** is very active and helpful

Cons

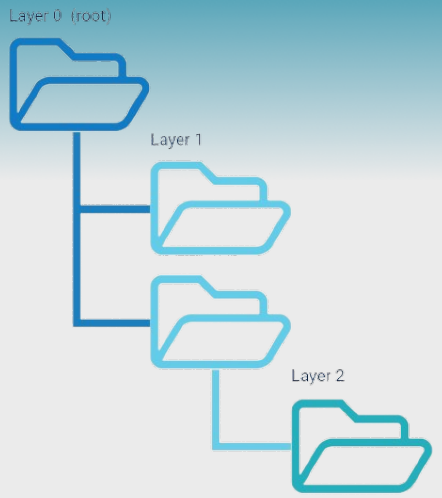
- Primitive support for client-side interactivity
  - but combining use of Alpine addresses this
- Fewer available component libraries
  than with other frameworks

Creating a Project

- Enter **`npm create astro@latest`**
- Answer questions
  - Where should we create your new project?
  - How would you like to start your new project?
    - Include sample files, Use blog template, or Empty
  - Install dependencies?
  - Do you plan to write TypeScript?
  - How strict should TypeScript be?
    - Strict, Strictest, or Relaxed
  - Initialize a new git repository?



Initial Files



- When an empty project is created, it contains
  - **`node_modules`** - holds all dependencies
  - **`public/favicon.svg`**
  - **`src/pages/index.astro`** - initial page
  - **`.gitignore`**
  - **`astro.config.mjs`** - configures adapters (ex. node) and integrations (ex. tailwind)
  - **`package.json`**
  - **`package-lock.json`**
  - **`README.md`**
  - **`tsconfig.json`** - configures TypeScript

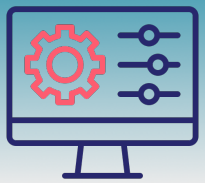
Running Project

- `cd` to project directory
- Development
  - enter `npm run dev` or `npm start`
    - provides hot reloading
  - browse localhost:4321 rocket launch countdown
- Production
  - enter `npm run build`
    - creates and populates `dist` directory
  - enter `npm run preview`
  - browse localhost:4321

Adapters & Integrations

- Install each by entering
  `npx astro add {name}`
  - updates **`astro.config.mjs`**
- Commonly used adapters
  - **`cloudflare`**, **`netlify`**, **`node`**, **`vercel`**
- Commonly used integrations
  - **`alpinejs`**, **`lit`**, **`mdx`**, **`react`**, **`solid-js`**, **`svelte`**, **`tailwind`**, **`vue`**

Astro Configuration File



- Example **`astro.config.mjs`** file
  - nearly every line results from these commands
    - **`npx astro add alpinejs`**
    - **`npx astro add node`**
    - **`npx astro add tailwind`**
- **`output`** values
  - **`static`** (default; only SSR)
    - all pages generated at build time
  - **`hybrid`** (mostly SSR)
    - pages default to build time
  - **`server`** (mostly SSG)
    - pages default to on request

```
import {defineConfig} from 'astro/config';
import alpinejs from '@astrojs/alpinejs';
import node from '@astrojs/node';
import tailwind from '@astrojs/tailwind';

export default defineConfig({
  integrations: [alpinejs(), tailwind()],
  output: 'server', // defaults to 'static'
  adapter: node({
    mode: 'standalone'
  })
});
```

`standalone` mode means it is not used with a Node library like Express or Fastify.

override in any page with
`export const pretender = false;`

override in any page with
`export const pretender = true;`

VS Code

- Extensions that help with Astro development include
  - **Astro** from astro.build
  - **Houston** from astro.build
  - **MDX** from unified
  - **Prettier - Code Formatter** from prettier.io
  - **Tailwind CSS IntelliSense** from tailwindcss.com

Prettier

- In addition to installing the VS Code extension ...
- Enter **`npm install -D prettier-plugin-astro`**
- Create file **`.prettierrc`** in project root directory

my recommended settings

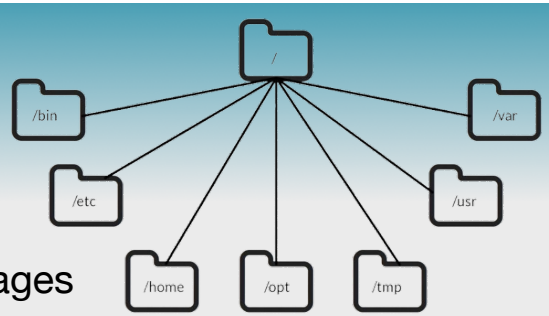
```
{
  "arrowParens": "avoid",
  "astroAllowShorthand": true,
  "bracketSpacing": false,
  "singleQuote": true,
  "trailingComma": "none",
  "plugins": ["prettier-plugin-astro"]
}
```

- Add script in **`package.json`**

```
"format": "prettier --write '{public,src}/**/*.{astro,css,html,js,ts}'",
```

- Run by entering **`npm run format`**

Common Directories



- **`src/layouts`** - holds components that provide HTML boilerplate to pages
- **`src/pages`** - holds components that define entire pages
- **`src/components`** - holds components used by pages
- **`src/styles`** - holds CSS files optionally imported by pages
- **`src/images`** - holds images to be optimized by **`Image`** component
- **`src/content`** - holds content collection Markdown files
- **`src/pages/api`** - holds JS/TS files that define API endpoints

The only mandated directory names are **`pages`** and **`content`**.

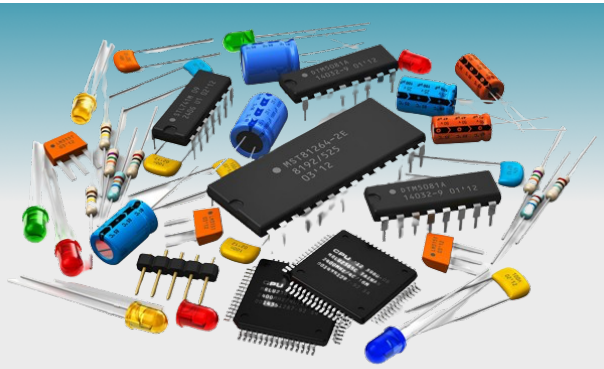
Layouts

- Example layout in **`src/layouts/Layout.astro`**

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Document</title>
  </head>
  <body>
    <slot /> ← page content is inserted here
  </body>
</html>
```

- Layouts can be nested

Components



- Example component in **`src/components/Greet.astro`**

```
---
type Props = {
  name: string;
};

const { name } = Astro.props;
---

<div>Hello, {name}!</div>

<style>
  div {
    color: purple;
    font-size: 2rem;
    font-weight: bold;
  }
</style>
```

Styles defined in a component are scoped to it by default.

Add `is:global` to change that.

Pages

- Defined by files under **`src/pages`** directory
  - can use **`.astro`**, **`.html`**, **`.md`**, and **`.mdx`** files
- Example Astro page in **`src/pages/colors.astro`**

browse at **`/colors`**

```
---
import Greet from "../components/Greet.astro";
import Layout from "../layouts/Layout.astro";

const colors = ["red", "green", "blue"];
---

<Layout>
  <Greet name="Colors" />
  <ul>
    {colors.map(color =>
      <li style={`color: ${color}`}>{color}</li>
    )}
  </ul>
<Layout>
```

front matter,
aka “component script”

uses JSX-like syntax

Hello, Colors!

- red
- green
- blue

Path Aliases

- TypeScript supports path aliases defined in **`tsconfig.json`** to simplify imports

```
{
  "extends": "astro/tsconfigs/strictest",
  "compilerOptions": {
    "baseUrl": "./src",
    "paths": {
      "@components/*": ["components/*"],
      "@images/*": ["images/*"],
      "@layouts/*": ["layouts/*"]
    }
  }
}
```

- Example
  - **`import Layout from '../../../layouts/Layout.astro';`**
    can be changed to
    `import Layout from '@layouts/Layout.astro';`

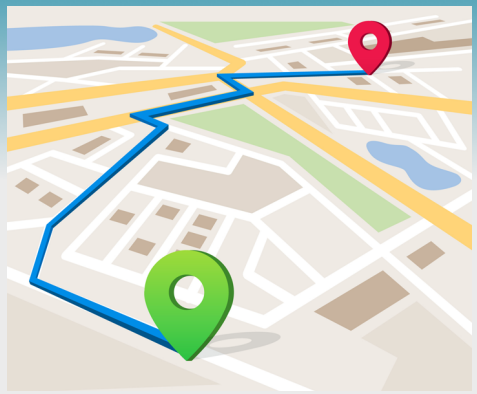
Images

- Can be placed under **`public`** or **`src`** directory
- Provided **`Image`** component optimizes images
  - performed at build time for static sites
  - must be imported from under **`src`**,
    but can also configure to optimize remote images
  - can adjust dimensions, file type (ex. WEBP), and quality
  - adds **`img`** attributes **`decoding="async"`** and **`loading="lazy"`**

Also see provided **`Picture`** component which renders an appropriate image from a provided set of formats and sizes.

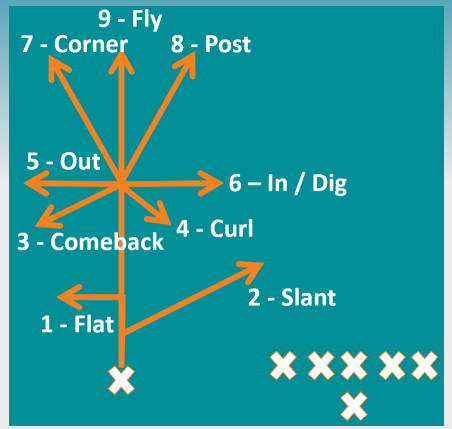
```
---
import { Image } from "astro:assets";
import logo from "../images/logo.png";
---

<Image alt="logo" src={logo} height={200} />
```

Dynamic Routes ...



- Routes defined under **`pages`** directory with directory and/or file names that contain a variable name inside square brackets
  - ex. **`src/pages/[sport]/[team].astro`**
- Can be used for both pages and API endpoints
- Inside an Astro component at a dynamic route, get matching segments from **`Astro.params`**
  - ex. if URL is **`football/chiefs.astro`** then **`Astro.params`** is the object **`{ sport: 'football', team: 'chiefs' }`**

... Dynamic Routes



- **`getStaticPaths`** function is required for SSG

  ignored by SSR

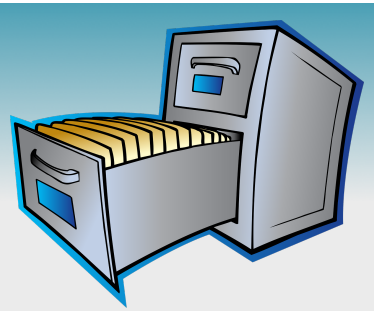
  - returns array of path params to use for generating static pages
  - example in **`src/pages/[sport]/[team].astro`**

```
---
export function getStaticPaths() {
  return [
    { params: { sport: 'baseball', team: 'Cardinals' } },
    { params: { sport: 'football', team: 'Chiefs' } },
    { params: { sport: 'hockey', team: 'Blues' } }
  ];
}

const { sport, team } = Astro.params;
---

<Layout>
  <div>The {team} play {sport}.</div>
</Layout>
```

`npm build` will generate these files in **`dist/client`**:

- **`baseball/Cardinals/index.html`**
- **`football/Chiefs/index.html`**
- **`hockey/Blues/index.html`**

Content Collections ...



- Can describe and retrieve collections of data from files under `src/content`
  - each subdirectory represents a different collection
  - files can use Markdown, MDX, YAML, or JSON format
  - all files in a collection must use same format
- Analogy between content collections and relational databases
  - each `src/content` subdirectory is **like a database table**
  - each file in these subdirectories is **like a row** in a database table
  - content collections are **like databases without SQL**,
    but can only retrieve single document or all in collection
- Content collection documents are static
  - can't update at runtime and render updated results
  - for that functionality, use a database or CMS

... Content Collections ...

- To define a collection
  - create **`src/content`** directory
  - create file **`config.ts`** inside it
    - defines properties of each collection using Zod for validation
    - https://github.com/colinhacks/zod
  - create subdirectory for each collection
  - create files inside each collection directory

```ts
import {defineCollection, z} from 'astro:content';

const dogs = defineCollection({
  type: 'content',
  schema: z.object({
    name: z.string(),
    breed: z.string()
  })
});
```

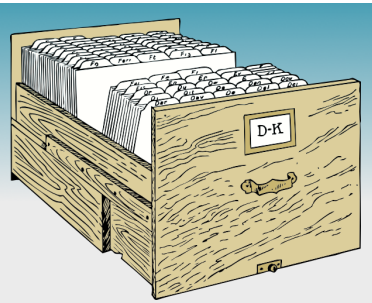
when not using Markdown or MDX, value must be `'data'`

src/content/config.ts

src/content/dogs/comet.md

```
---
name: Comet
breed: Whippet
website: https://www.akc.org/dog-breeds/whippet/
---

He loves the following:

- pool balls
- basketballs
- frisbees
```

... Content Collections



- To use a content collection in a page or component
  - call **`getCollection`** or **`getEntry`**

src/page/index.astro

```
---
import Layout from "../../layouts/Layout.astro";
import Dog from "../../components/Dog.astro";
import { getCollection, type CollectionEntry } from "astro:content";

const dogs: CollectionEntry<"dogs">[] = await getCollection("dogs");
---

<Layout title="Dogs I Know">
  <main class="m-4">
    { dogs.map(dog => <Dog {dog} /> }
  </main>
</Layout>
```

`CollectionEntry` object

`getCollection` accepts a second argument that is a function used to filter the entries.

Comet, the Whippet

He loves the following:

- pool balls
- basketballs
- frisbees

src/components/Dog.astro

```
---
import { Image } from "astro:assets";
import type {CollectionEntry} from 'astro:content';

type Props = {
  dog: CollectionEntry<'dogs'>;
};

const {dog} = Astro.props;
const {breed, name, photo} = dog.data;
const { Content } = await dog.render();
---

<div class="dog my-4">
  <p class="font-bold mb-4">{name}, the {breed}</p>
  <Image alt="breed photo" src={photo} />
  <Content />
</div>

<style>
  img {
    height: auto;
    width: 10rem;
  }
</style>
```

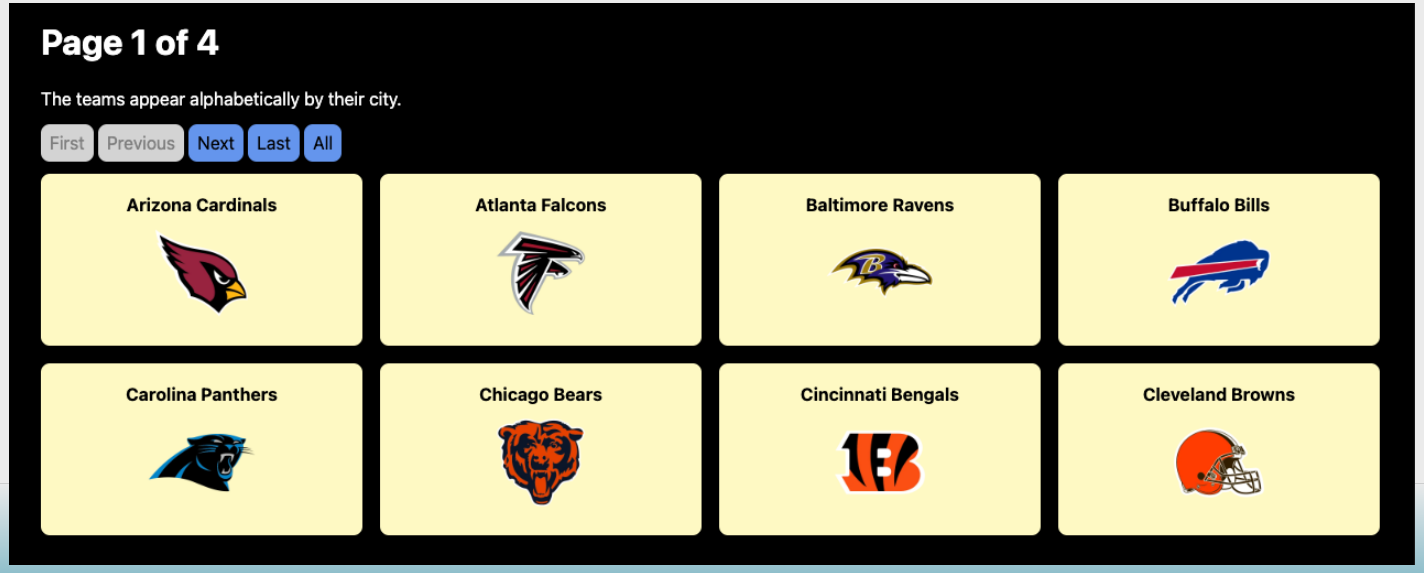
using Tailwind CSS classes

renders Markdown content, in this case “He loves ...”

Pagination

‹ 1 ... 5 ... 9 ›

- Astro provides help for implementing pagination of content collections
  - see **`paginate`** function passed to **`getStaticPaths`** functions
  - details at https://docs.astro.build/en/core-concepts/routing/#pagination
- Example at https://github.com/mvolkmann/astro-examples/tree/main/content-collections



MDX

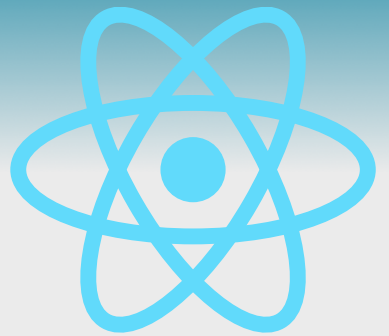
- Extends Markdown to add ability to
  - **define JavaScript variables** whose values come from JavaScript expressions
  - **insert values** of front matter properties and JavaScript variables into content using curly braces
  - **render components** implemented in any of the supported frameworks
- To enable using MDX,
  `npx astro add mdx`

MDX Examples

```
---
layout: ../layouts/Layout.astro
player: Mark
score: 19
---

This page is described by **MDX**.

The score for {frontmatter.player}
is {frontmatter.score}.

export const twoPi =
  (Math.PI \* 2).toFixed(4);

2π is approximately {twoPi}.
```

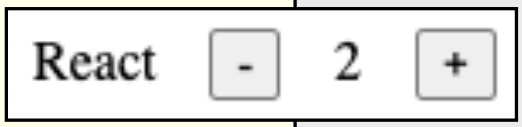
```
---
layout: ../layouts/Layout.astro
name: Comet
breed: Whippet
website: https://www.akc.org/dog-breeds/whippet/
---

import Greet from "../../components/Greet.astro";

<Greet name={frontmatter.name} />

He loves the following:

- pool balls
- basketballs
- frisbees

Learn about the
<a href={frontmatter.website}>{frontmatter.breed}</a>.
```

React



- To enable using React components, **`npx astro add react`**
- Define components in **`.tsx`** files

```tsx
import {type FC, useState} from 'react';
interface Props {
  label?: string;
  start?: number;
}

const Counter: FC<Props> = ({label = '', start = 0}) => {
  const [count, setCount] = useState(start);
  return (
    <div style={{display: 'flex', alignItems: 'center', gap: '1rem'}}>
      {label && <div>{label}</div>}
      <button disabled={count <= 0} onClick={() => setCount(c => c - 1)}>
        -
      </button>
      <div>{count}</div>
      <button onClick={() => setCount(c => c + 1)}>+</button>
    </div>
  );
};
export default Counter;
```



Svelte

- To enable using Svelte components,
  `npx astro add svelte`
- Define components in **`.svelte`** files

```
<script>
  export let label = '';
  export let start = 0;

  let count = start;
</script>

<div class="row">
  {#if label}
    <div>{label}</div>
  {/if}
  <button disabled={count <= 0} on:click={() => count--}>-</button>
  <div>{count}</div>
  <button on:click={() => count++}>+</button>
</div>
```

```
<style>
  .row {
    display: flex;
    align-items: center;
    gap: 1rem;
  }
</style>
```

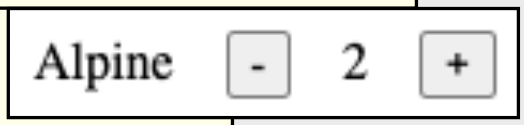
Svelte - 2 +

Alpine

- To enable using Alpine in Astro components, **`npx astro add alpinejs`**
- Define components in **`.astro`** files

```
---
interface Props {
  label?: string;
  start?: number;
}

const {label = '', start = 0} = Astro.props;
---

<div class="row" x-data={`{ count: ${start} }`}>
  {label && <div>{label}</div>}
  <button :disabled="count <= 0" @click="count--">-</button>
  <div x-text="count"></div>
  <button @click="count++">+</button>
</div>
```

```
<style>
  .row {
    display: flex;
    align-items: center;
    gap: 1rem;
  }
</style>
```



`client` Directives

- By default, components implemented in frameworks like React, Svelte, and Vue are **rendered at build time** (SSG)
- Their JavaScript is not downloaded to browsers, so they are **not interactive**
- To enable use of client-side JavaScript for a given component instance, apply a **`client:*`** directive
  - example

```
<Counter
  label="Tally"
  start={3}
  client:load
/>
```

Directive	When JS is loaded
`client:idle`	when browser is idle
`client:load`	immediately
`client:media`	when a CSS media query condition is met
`client:only`	after page load with no SSR
`client:visible`	when component becomes visible

nanostores

- Recommended way to share state between components
  - https://github.com/nanostores/nanostores
  - not specific to Astro
  - somewhat similar to Svelte stores
  - less than 1KB
  - three kinds of stores

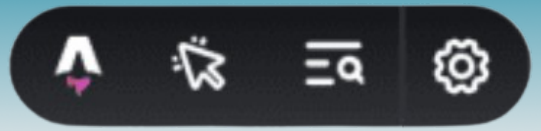
`atom`: single value

`map`: key/value pairs

`computed`: based on other stores

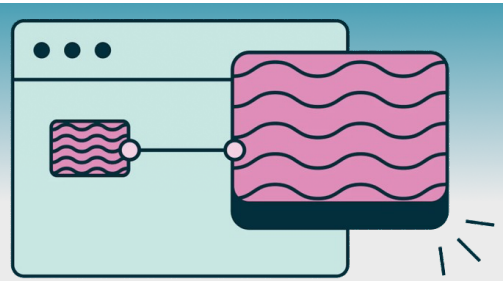
- Can use to share state between a mixture of component types in same app
  - ex. React, Svelte, Vue, and Alpine
- Demo at https://github.com/mvolkmann/astro-examples/tree/main/nanostores-demo

Dev Toolbar



- In dev mode, Astro provides a Dev Toolbar in browser
  - black oval centered at bottom of browser window
- Partially hidden from view until mouse hovers over it
- Four buttons
  - Menu (Astro icon)
    - Report a Bug, Feedback, Documentation, Community, Copy debug info
  - Inspect (arrow icon)
    - to inspect interactive components marked with `client:*` directives (islands)
  - Audit (document icon)
    - scans page for accessibility issues
  - Settings (gear icon)
    - can enable “Verbose logging” and “Disable notifications”

View Transitions ...



- Applied when navigating from one page to another
  - includes clicking `<a>` links,
    triggering browser forward and back buttons,
    and submitting forms
  - going back triggers opposite transition
    (ex. slide out vs. slide in)
- Built on View Transitions API described on MDN
- For basic fade out/in transitions between all pages,
  modify all their layout files (perhaps only one)
  - example on next slide

... View Transitions ...

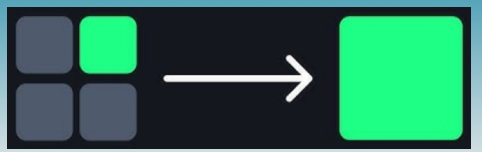
- Layout file that enables view transitions

```
---
import { slide, ViewTransitions } from "astro:transitions";
---

<html>
  <head>
    ...
    <ViewTransitions />
  </head>
  <body
    class="p-4"
    transition:animate={slide({ duration: "1s" })}
  >
    <slot />
  </body>
</html>
```

goes in **head** tag; required to enable ANY view transitions

optional customization for other than default

... View Transitions



- Builtin transitions include **`fade`**, **`initial`** (browser default), **`slide`**, and **`none`** (to disable a previously specified value)

can define custom transitions

- Can transition any kind of element that appears on consecutive pages
  - includes **`img`**, **`Image`** (Astro component), **`audio`**, **`video`**, and more
  - mark each with identifying **`transition:name`** directive value
  - morphs from current to new location and size
  - **`audio`** and **`video`** elements with **`transition:persist`** directive retain their state

```
<video controls width="200" transition:name="bunny-video" transition:persist>
  <source src="/bunny-video.mp4" type="video/mp4" />
</video>
```

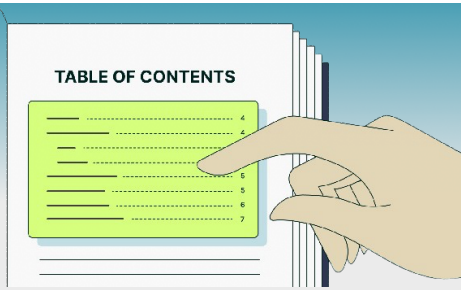
Prefetching

- Means loading resources that will be needed to render a page before navigating to it
- Triggered by hovering over link, clicking link, or scrolling into view
  - default prefetching strategy is “hover”
  - to use “tap” or “viewport” strategy, set **`data-astro-prefetch`** attribute on element to one of those
- Automatically enabled when view transitions are enabled
- When not using view transitions, enable by
  - adding **`prefetch: true`** in **`astro.config.mjs`** or
  - adding **`data-astro-prefetch`** to each element where desired

More capabilities:

- change default prefetch strategy
- make all anchor tags use prefetching by default
- manually trigger prefetching of page at a specified URL

Table of Contents



- Can add hypertext table of contents to Markdown pages
  - install with **`npm install remark-toc`**
  - update **`astro.config.mjs`**
  - add **`# Table of Contents`** in all Markdown files where desired

```
## Table of Contents

The National Football League (NFL) ...

## AFC East

### Miami Dolphins

### Buffalo Bills

### New York Jets

### New England Patriots

## AFC North

...
```

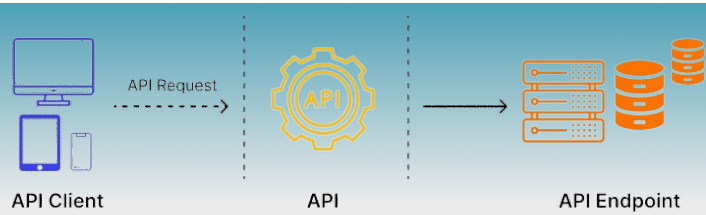
Table of Contents

1. AFC East
   1. Miami Dolphins
   2. Buffalo Bills
   3. New York Jets
   4. New England Patriots
2. AFC North
   1. Baltimore Ravens
   2. Cleveland Browns
   3. Pittsburgh Steelers
   4. Cincinnati Bengals

```
import {defineConfig} from 'astro/config';
import remarkToc from 'remark-toc';

export default defineConfig({
  markdown: {
    remarkPlugins: [
      [remarkToc, {
        maxDepth: 3,
        ordered: true,
        tight: false
      }]
    ]
  }
});
```

`true` for **`<ol>`**
`false` for **`<ul>`**

API Endpoints ...



- Invoked at build time or run time
- Defined in **`.js`** and **`.ts`** files under **`src/pages`** directory

  consider placing in **`api`** subdirectory

  - implement **`GET`**, **`POST`**, **`PUT`**, **`PATCH`**, and **`DELETE`** functions
  - all are passed an **`APIContext`** object →

`APIContext` properties:

- **`params`** object holds dynamic route segment values
- **`redirect`** is a function for redirecting to another URL
- **`request`** object has properties **`body`**, **`headers`**, **`method`**, **`url`**, and more
- **`cookies`** object provides methods to test for (**`has`**), **`get`**, **`set`**, and **`delete`** cookies
- **`url`** is a **`URL`** object for **`request.url`**

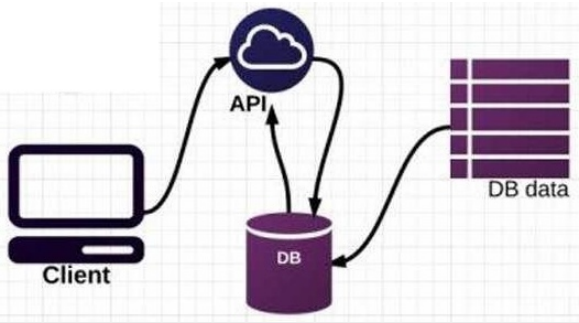
- Only used on server side, never in browsers
- Endpoint URLs are defined by file-based routing, just like UI pages
- Can have dynamic routes
  - **`Astro.params`** object contains matched segments

    alternative to **`APIContext.params`**

... API Endpoints ...

- Can return data in any format
  - including JSON and HTML (for use with HTMX)
- Uses standard web objects
  - **`Astro.request`** holds a **`Request`** object
  - **`Astro.url`** holds a **`URL`** object
  - **`Astro.response`** holds a **`ResponseInit`** object
  - **`Astro.cookies`** has methods from “cookies” API

... API Endpoints



- Example in **`src/pages/api/pets/dog.ts`**
  - invoke with **`GET`** request to **`/api/pets/dog`**
  - gets array of dog objects from a content collection

```
import {getCollection, type CollectionEntry} from 'astro:content';

export async function GET(context: APIContext) {
  const dogs: CollectionEntry<'dogs'>[] = await getCollection('dogs');
  const data = dogs.map(dog => dog.data);
  return new Response(JSON.stringify(data));
}
```

</> htmx

- **Client-side JavaScript library** that adds support for **new HTML attributes** that make HTML more expressive
- Enable responding to specific interactions (ex. click) with any HTML element by **sending an HTTP request** using any verb

  GET, POST, PUT, PATCH, or DELETE
- **Response must contain HTML**
- Rather than performing a complete page refresh, the returned HTML replaces an existing DOM element or is inserted relative to one
- **Removes need to serialize data to JSON** on server, parse JSON on client, and convert to HTML
- Server can be implemented using **any programming language and server framework**
- **Simplifies state management** because all state is in on server

HTMX in Astro

- "**Page partials**" allow Astro components to return snippets of HTML
  - only body content, not **`doctype`**, **`html`**, **`head`** (and contents), and **`body`** elements
  - ideal for use with HTMX
- Indicate by adding the following in a component script

```
export const partial = true;
```

- Example app that combines Astro and HTMX
  - https://github.com/mvolkmann/astro-htmx-todo-app
- Astro pages can be used as endpoints
  - respond to requests with any HTTP verb
  - component script can determine which verb was used and respond accordingly

Another option is implementing Astro endpoints (not pages) that redirect to an Astro page that generates the HTML to be returned.

Resources

- **Astro home page** - https://astro.build/
  - has excellent docs
- **My blog** - https://mvolkmann.github.io/blog/ (select Astro)
- **My Astro example code** - https://github.com/mvolkmann/astro-examples/
- **Astro Discord server** - https://discord.com/invite/astrodotbuild

Wrap Up

- **Astro …**
  - makes web sites faster by downloading less JavaScript code
  - has excellent support for both SSG and SSR,
    allowing each page to choose
  - implements the “islands architecture”
    to maximize the use of static content
  - has excellent support for authoring content in Markdown and MDX
  - supports implementing components using any popular web framework
  - is very easy to learn!