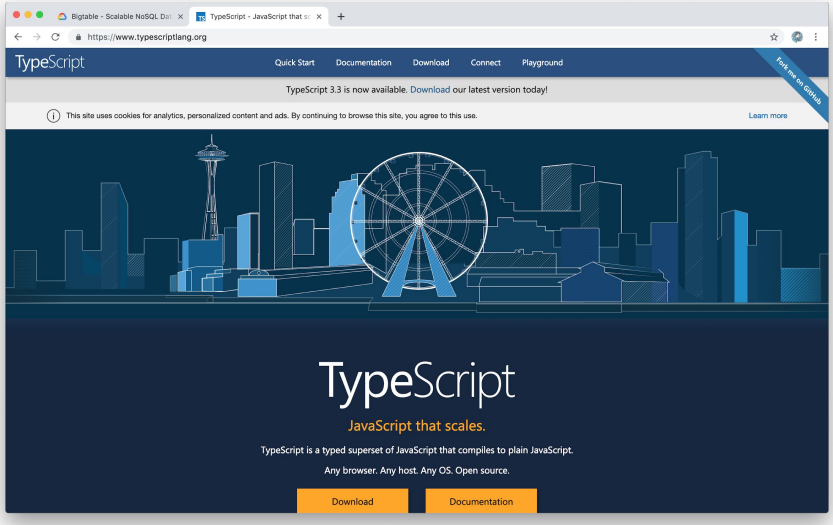
How to scale

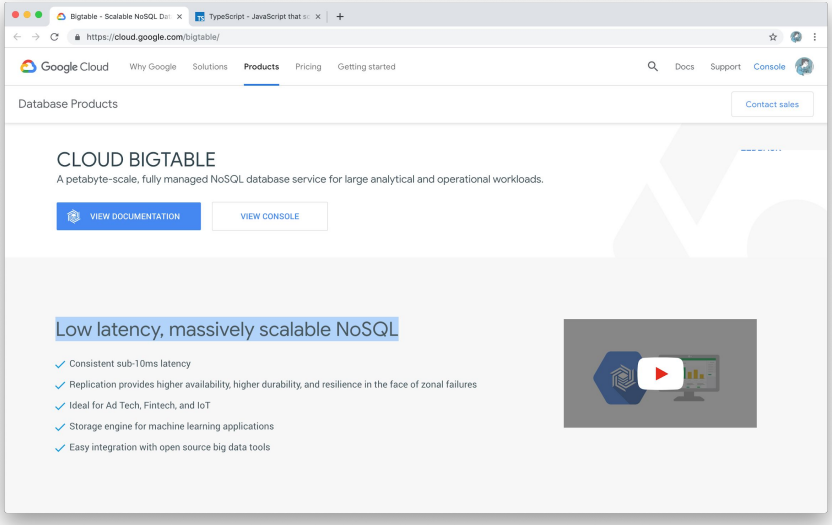
Presenter: Sam Zhou

Overloaded meaning of scale

Scale for large codebase



Scale for large workload



Overloaded meaning of scale

Scale for codebase:

Get the Bug-O right. 🐛(n!)

https://overreacted.io/the-bug-o-notation/

Scale for workload:

Get the Big-O right. O(n)

Scaling your codebase: the Bug-O notation

Not formally defined like Big-O

A sketchy measure of the complexity of code

We usually want 🐛(1) or 🐛(lg n) so things can be under control

Optimize Bug-O: better abstractions and tools

Optimize for change

Bad

“I hope our IDs won’t collide.”

“Is my UI synced with data?”

Direct DOM manipulation

Can be easily messed up by other components

(n), n = # of components in the app

```
function nestedListElement(parent) {

  var ul = document.createElement("ul");
  var li = document.createElement("li");

  li.appendChild(document.createTextNode("Placeholder Text"));
        li.setAttribute("id", "toDoList");
        li.setAttribute("class", "sortable-element");
        li.setAttribute("contenteditable", true);


        var span = document.createElement("SPAN");
        var txt = document.createTextNode("\u2713");
        span.className = "close";
        span.setAttribute("contenteditable", false);
        span.appendChild(txt);
        li.appendChild(span);

        var span2 = document.createElement("span");
        txt2 = document.createTextNode("\u2195  \u00a0 \u00a0");
        span2.className = "handle";
        span2.setAttribute("contenteditable", false);
        span2.appendChild(txt2);
        li.insertBefore(span2, li.firstChild);

        var span3 = document.createElement("span3");
        txt3 = document.createTextNode("\u2719");
        span3.className = "hider";
        span3.setAttribute("contenteditable", false);
        span3.setAttribute("data-toggle", "collapse");
        span3.appendChild(txt3);
        li.appendChild(span3);


  ul.appendChild(li);
        ul.setAttribute("class", "sortable");
        ul.setAttribute("contenteditable", false);
        parent.appendChild(ul);
}
```

Good

Use React Component to control complexity

Modular Code

Problem: What if we need to refactor?

(C*n)

n = # of component that uses this

C = some not-very-small constant

```
  render() {
    return (
      <form className={styles.NewTaskWrap} onSubmit={this.handleSave}>
        <input value={this.state.name} onChange={this.handleTaskNameChange} type="text" className={styles.NewTaskComponent} pla
        <div className={styles.NewTaskActive}>

          <div className={styles.NewTaskClass}>
            <input id="changeClassCheckbox" type="checkbox" ref={this.openClassChange} />
            <label htmlFor="changeClassCheckbox" style={{ backgroundColor: this.props.tagColorPicker[this.state.tag] }} ref={th
            <ul>
              {Object.keys(this.props.tagColorPicker).map(cTitle => <NewTaskClassPicker key={cTitle} classColor={this.props.tag
            </ul>
          </div>

          <div className={styles.NewTaskDate}>
            <label htmlFor="changeDateCheckbox">📆</label>
            <input id="changeDateCheckbox" type="checkbox" ref={this.openDateChange} />
            <div className={styles.NewTaskDatePick}>
            <Calendar
              onChange={this.handleDateChange}
              value={new Date()}

              />
              </div>
          </div>

        </div>
      </form>
    );//<input type="date" value={this.state.date} onChange={this.handleDateChange} min={new Date(new Date().getTime() - new Da
  }
}

```

Less Good

React Ref is harder to reason

Avoid it unless absolutely need it

```
15    constructor(props) {
16      super(props);
17      this.state = this.initialState();
18      this.changeClass = React.createRef();
19      this.addTask = React.createRef();
20      this.openClassChange = React.createRef();
21      this.openDateChange = React.createRef();
22    }
```

Better

Runtime type-checking components

Self-documenting

Still need good test coverage to ensure there is no prop-type errors

(c*n),

n = # of component that uses this

c = a small constant

```
35    renderCourses() {
36      if (this.props.query !== "") {
37        return this.props.allCourses.slice(0,100).map((course) => (
38          //create a new class "button" that will set the selected class to this class when it is clicked.
39          <Course key={course._id} info={course} query={this.props.query}/>
40        ));
41      }
42      else {
43        return <div />;
44      }
45    }
46
47    showDropdown = () => {
48      this.setState({showDropdown: true})
49    }
50
51    hideDropdown = () => {
52      this.setState({showDropdown: false})
53    }
54
55    render() {
56      return (
57        <div className="search-bar text-left" id="searchbar" >
58          <input className="search-text" id="search" onChange={this.props.queryFunc} placeholder="Search for classes (e.g. CS 211
59          <ul id="output" style={this.state.showDropdown ? {} : { display: 'none' }}>
60            {this.renderCourses()}
61          </ul>
62        </div>
63      );
64    }
65  }
66
67  // SearchBar requires the user's query from the parent component, a function
68  // to call when the query changes so the parent can update its copy of the query,
69  // and a list of all courses that satisfy the query.
70  SearchBar.propTypes = {
71    allCourses: PropTypes.array.isRequired,
72    loading: PropTypes.bool, // optional
73    query: PropTypes.string.isRequired,
74    queryFunc: PropTypes.func.isRequired
75  };
```

Even Better

Type-annotate the components

Type-checkers ensure you use the component correctly at compile-time

(1)

Your prop types can never go wrong once you pass the type checker

```
type DefaultProps = {|
  +className?: string;
  children?: Node;
  +newSubTaskDisabled?: boolean;
  +onFocus?: (event: SyntheticFocusEvent<HTMLElement>) => void;
  +onBlur?: (event: SyntheticFocusEvent<HTMLElement>) => void;
  +refFunction?: (HTMLElement | null) => void; // used to get the DO
|};
type Props = {|
  ...Task; // The task given to the editor at this point.
  ...DefaultProps; // Props with default values.
  // actions
  +editMainTask: (partialMainTask: PartialMainTask, doSave: boolean)
  +editSubTask: (subtaskId: string, partialSubTask: PartialSubTask,
  +addSubTask: (subTask: SubTask) => void;
  +removeTask: () => void;
  +removeSubTask: (subtaskId: string) => void;
  +onSave: () => void;
  // subscribed from redux store.
  +getTag: (id: string) => Tag;
|};

type State = {|
  +mainTaskNameCache: string | null;
  +oneSubTaskNameCache: [string, string] | null;
  +doesShowTagEditor: boolean;
  +doesShowDateEditor: boolean;
  +needToSwitchFocus: number | null; // number: expected array lengt
|};

/**
 * The component of an standalone task editor.
 * It is designed to be wrapped inside another component to extend i
 * editor itself does not remember the state of editing a task, a wr
 * You can read the docs for props above.
 */
class TaskEditor extends React.Component<Props, State> {
  state: State = {
    mainTaskNameCache: null,
    oneSubTaskNameCache: null,
```

```
/**
 * The task editor used to edit task inline, activated on
 */
export default function InlineTaskEditor({ task, className
  const [disabled, setDisabled] = React.useState(true);

  const { id } = task;
  // To un-mount the editor when finished editing.
  const onFocus = () => setDisabled(false);
  const onBlur = () => setDisabled(true);
  const taskEditorProps = {
    ...task,
    editMainTask: (partialMainTask: PartialMainTask, onSav
      editMainTask(id, partialMainTask);
      if (onSave) {
        onBlur();
      }
    },
    editSubTask: (subtaskId: string, partialSubTask: Parti
      editSubTask(subtaskId, partialSubTask);
      if (onSave) {
        onBlur();
      }
    },
    addSubTask: (subTask: SubTask) => { addSubTask(id, sub
    removeTask: () => { removeTask(task); },
    removeSubTask: (subtaskId: string) => { removeSubTask(
    onSave: onBlur,
    className,
    newSubTaskDisabled: disabled || !task.inFocus,
    onFocus,
    onBlur,
  };

  return <TaskEditor {...taskEditorProps} />;
}
```

Scaling your workload

Loosely coupled & independently scalable

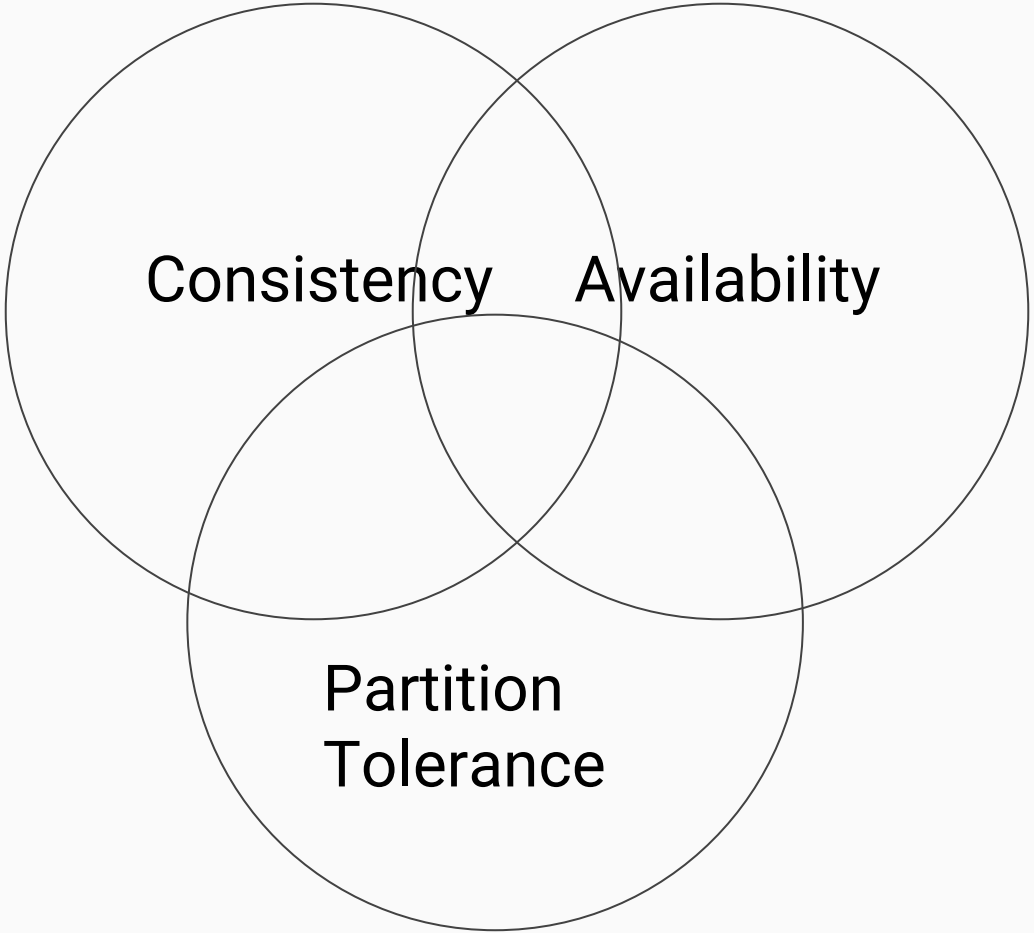
Relational vs NoSQL Database, CAP Theorem

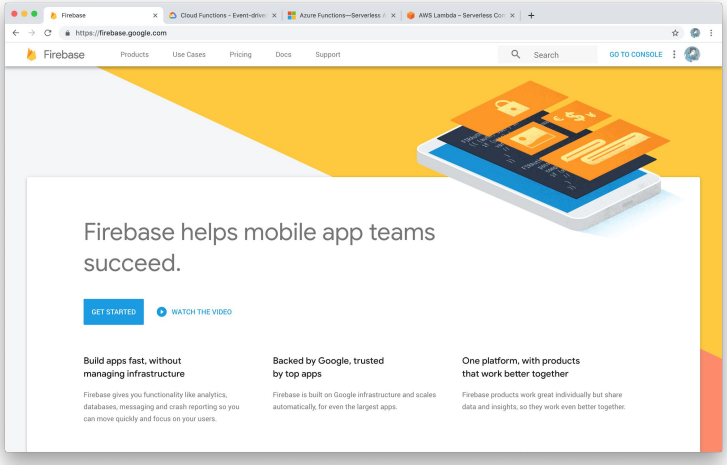
On the cloud: managed solutions

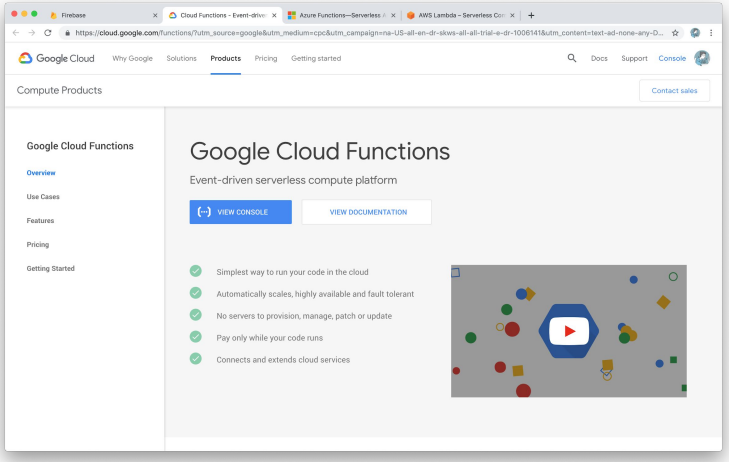
lb-ingress.yaml

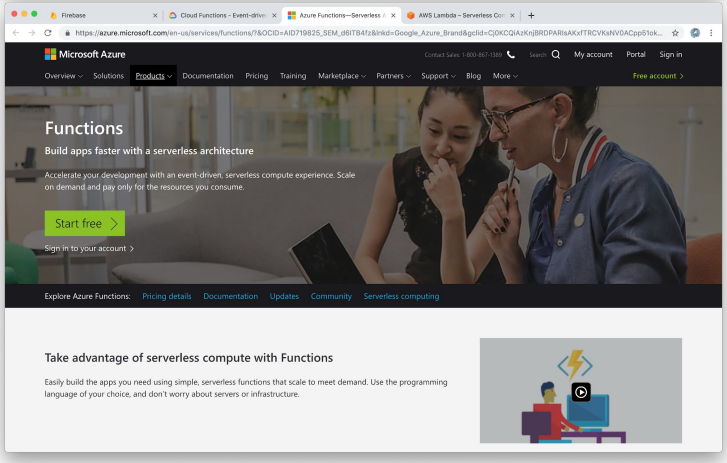
```yaml
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: lb-ingress
  annotations:
    kubernetes.io/ingress.global-static-ip-name: "web-static-ip"
spec:
  rules:
  - http:
      paths:
      - path: /apis/*
        backend:
          serviceName: backend-workload
          servicePort: 8080
      - backend:
          serviceName: frontend-workload
          servicePort: 8080

```

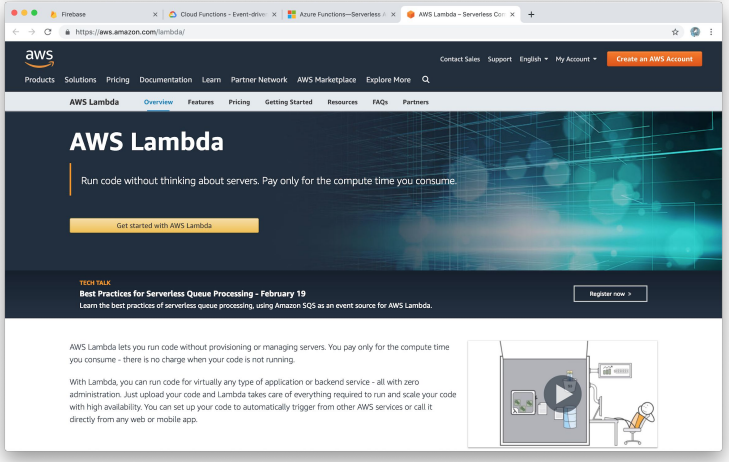
Loosely coupled & independently scalable


Consistency
Availability
Partition
Tolerance





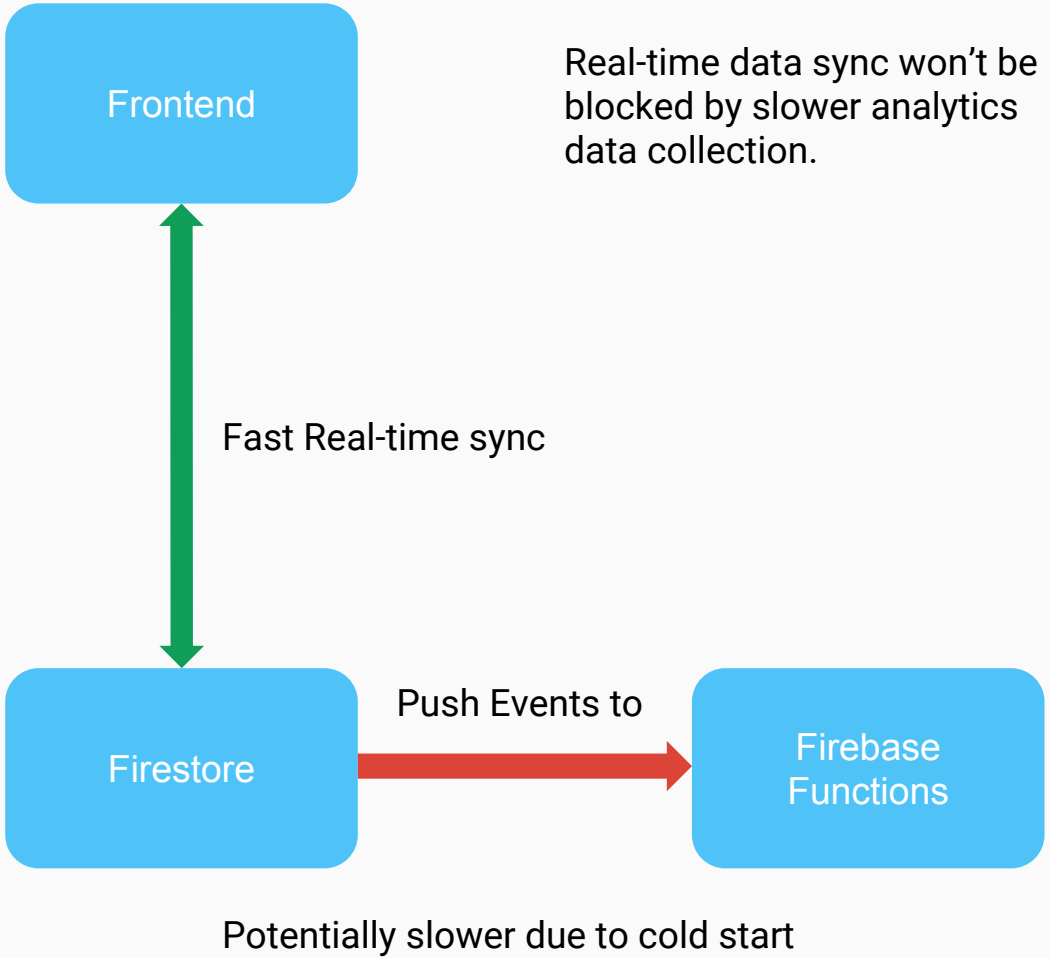




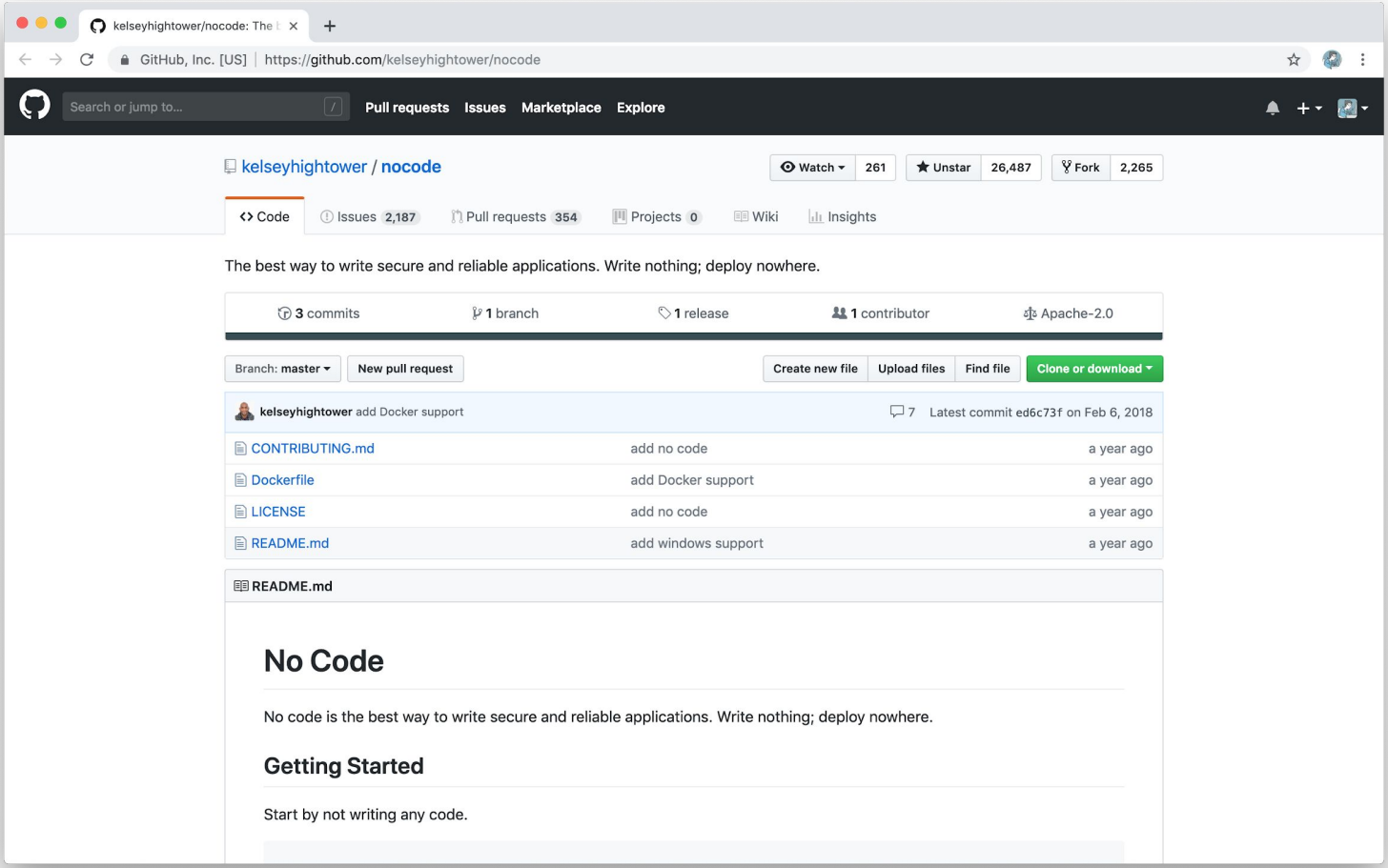
Managed FaaS solutions

Case Study

Samwise's Real-time Interactions & Analytics



However … Scaling is easy!



https://github.com/kelseyhightower/nocode