KubeCon | CloudNativeCon

North America 2019

KubeCon | CloudNativeCon

North America 2019

Brigade Gateways and Workers

Matt Butcher and Kent Rancourt

Matt Butcher

Principal dev at Microsoft. Helm, Brigade, CNAB, OAM, and all that. Author of a bunch of tech books. Big coffee snob.

Kent Rancourt

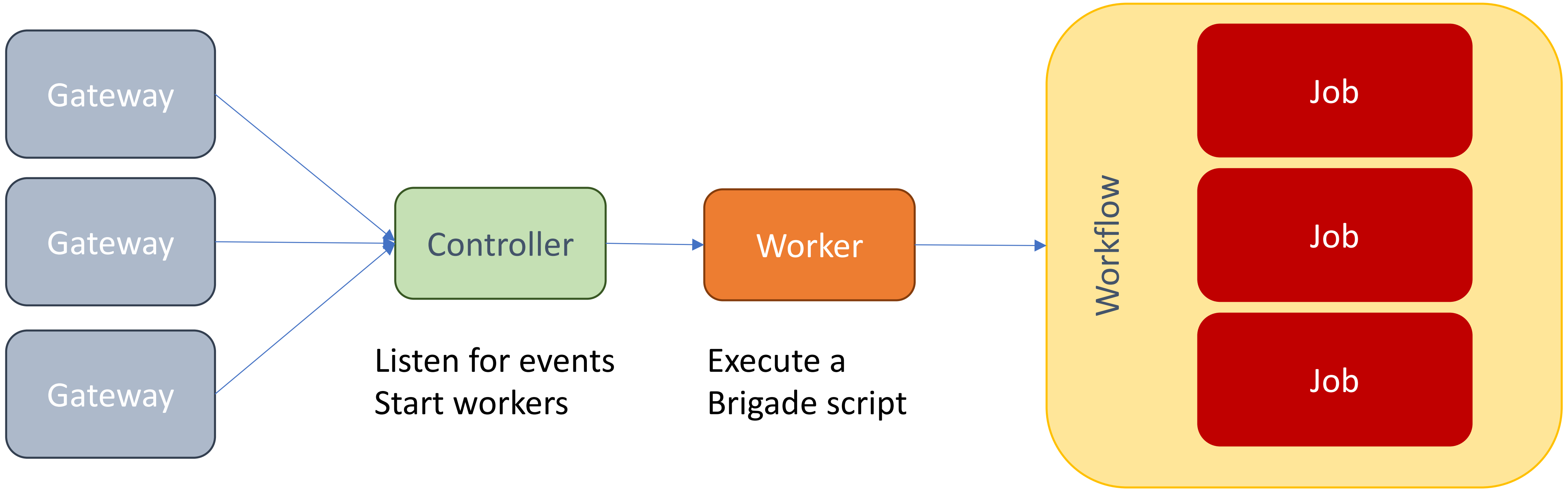
Senior engineer at Microsoft working on Brigade and other OSS. Passionate about CI/CD and automation in general. Dad, martial arts instructor, comic book nerd, lover of pub trivia, and I think Starbucks is fine coffee. Fight me.

Overview

Today we are going to cover a few ways of extending Brigade:

1. Building a custom gateway
2. Building a custom worker

Brigade Architecture



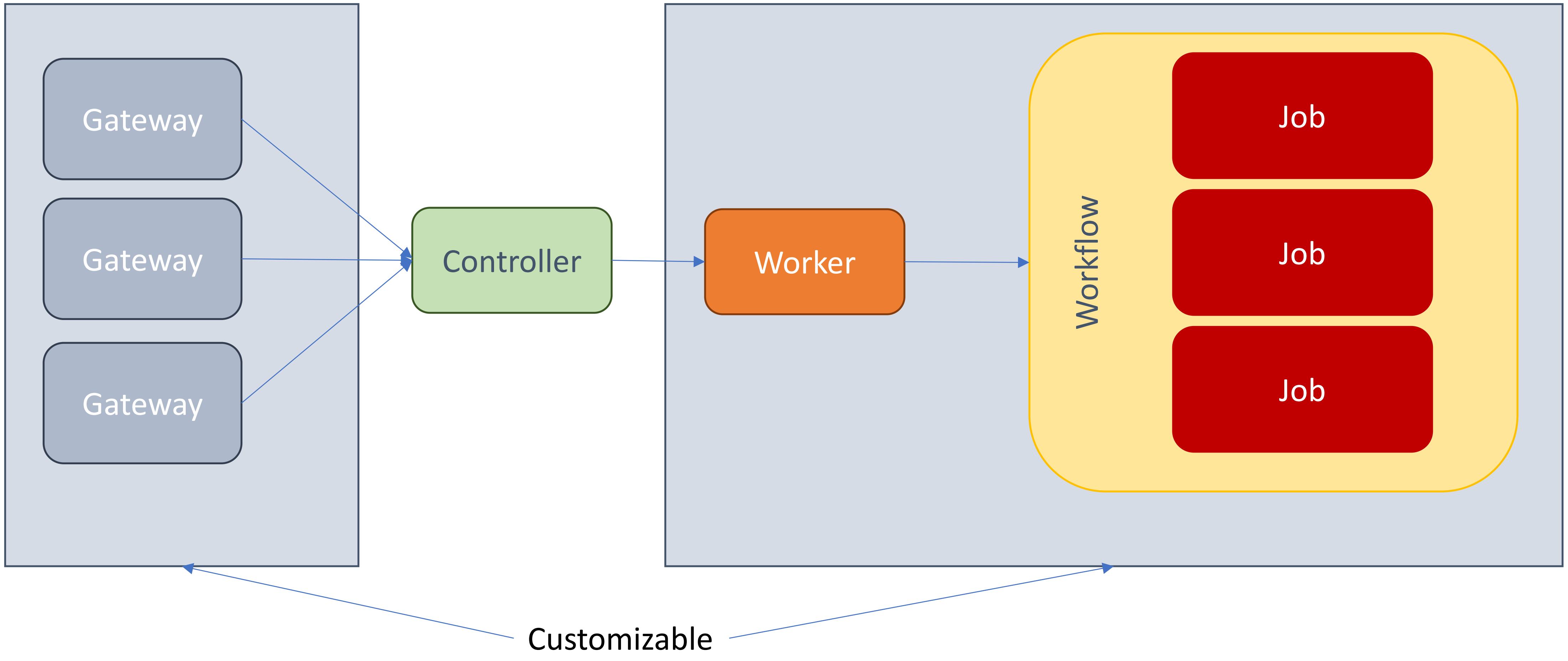
Listen for events
Start workers

Execute a
Brigade script

Based on something external create an event.

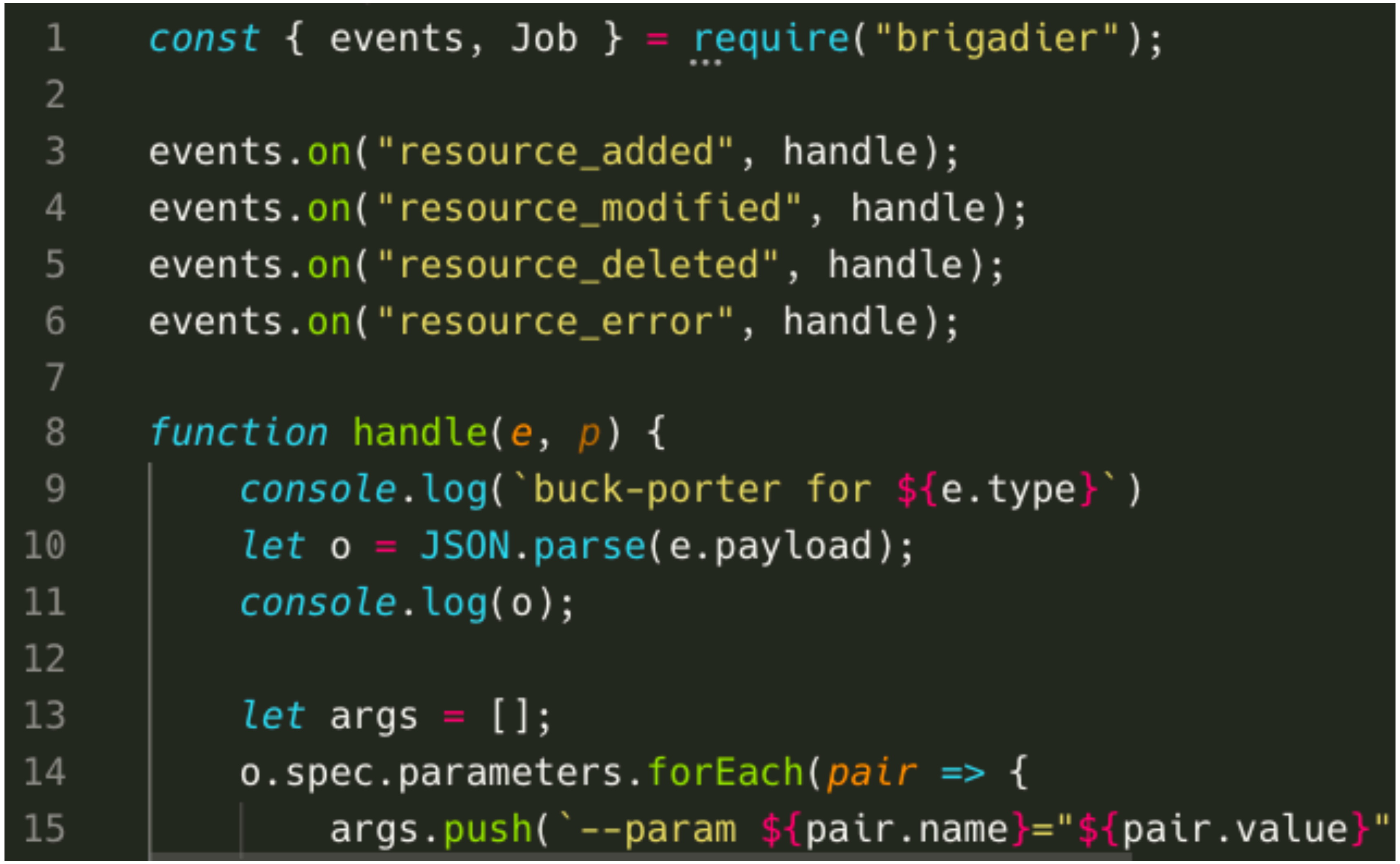
Run jobs to completion, where each job is a step in a workflow.

Brigade Architecture



Brigade.js

A workflow is typically written as a brigade.js file. And jobs are Docker containers. Thus, this part of Brigade is already highly customizable.



```javascript
const { events, Job } = require("brigadier");

events.on("resource_added", handle);
events.on("resource_modified", handle);
events.on("resource_deleted", handle);
events.on("resource_error", handle);

function handle(e, p) {
    console.log(`buck-porter for ${e.type}`)
    let o = JSON.parse(e.payload);
    console.log(o);

    let args = [];
    o.spec.parameters.forEach(pair => {
        args.push(`--param ${pair.name}="${pair.value}"
```

How to Build a Gateway

1. Write a server that watches for the external trigger (cron, webhook, event, etc)
2. That server must generate a Kubernetes secret as output
3. Typically, run this as a Kubernetes deployment

Gateways

A custom gateway makes it possible for you to trigger your own Brigade events based on whatever conditions you want.

Examples:

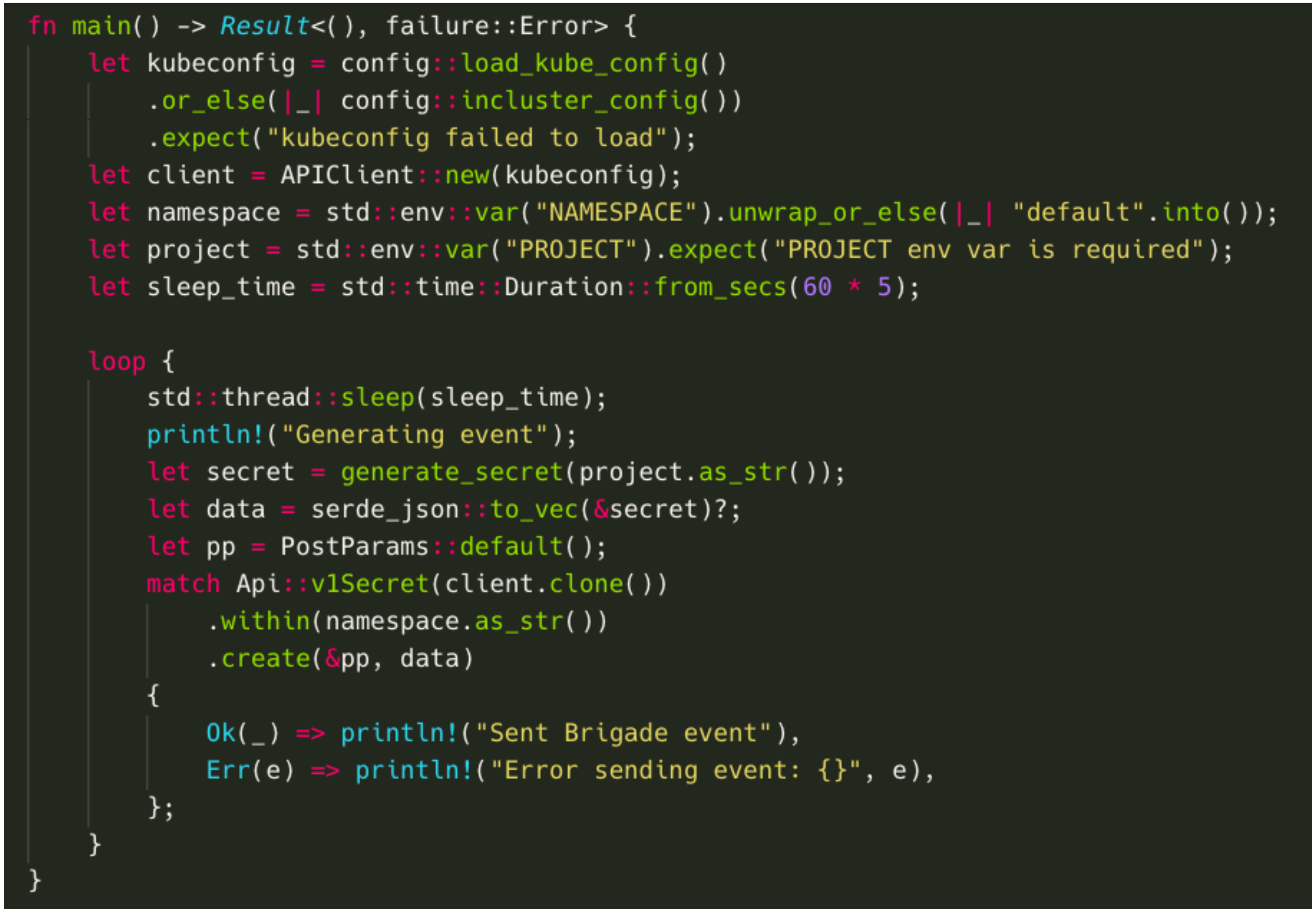
- Cron-based gateway runs a job based on time
- CloudEvents gateway hooks Brigade up to a CloudEvents emitter
- Trello gateway hooks up Trello's actions to a Brigade script

Demo: A Minimal Gateway

In this demo, we'll look at a small gateway written in Rust.

This gateway generates a new "interval" event every five minutes.

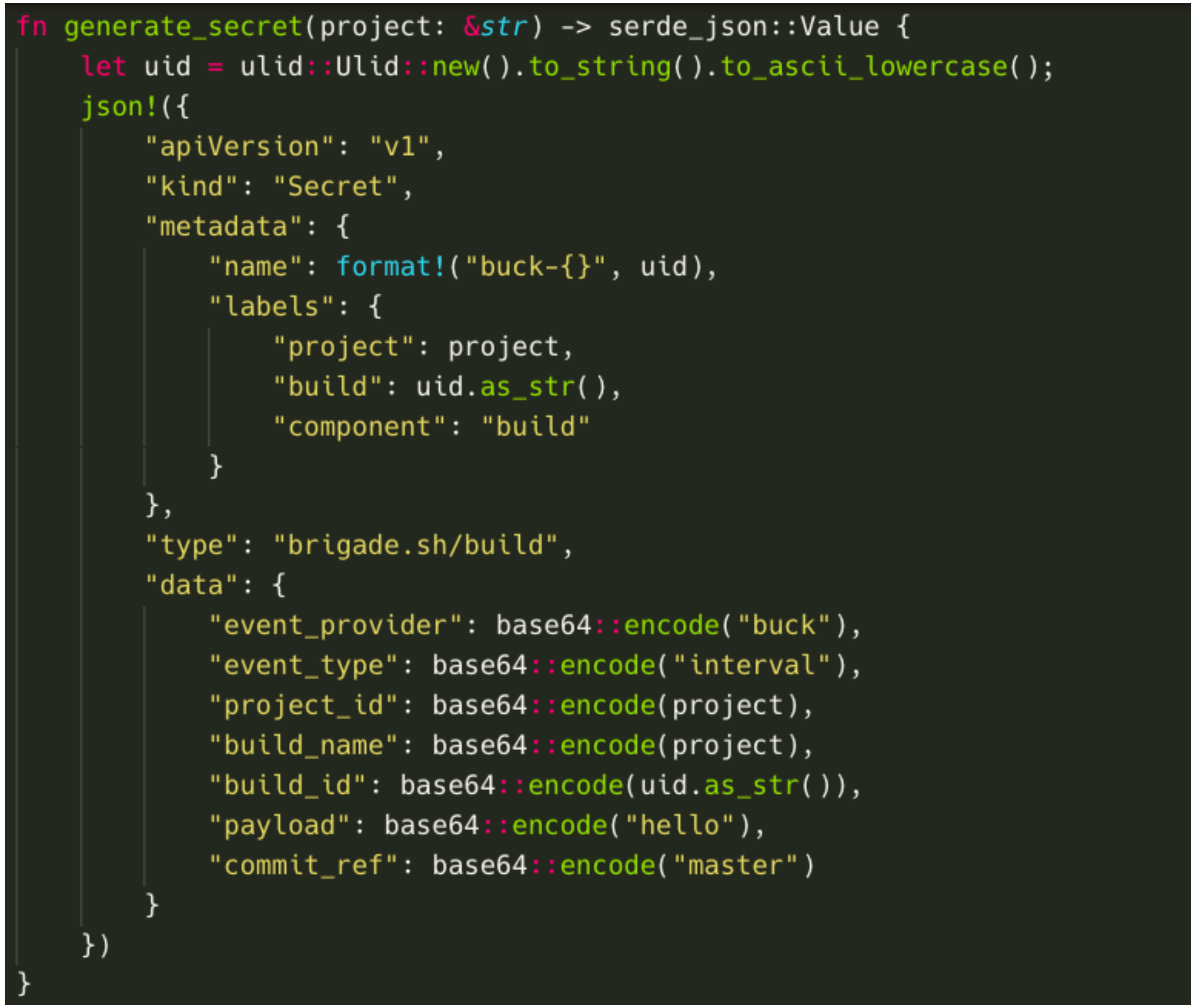
Demo: A Minimal Gateway



```rust
fn main() -> Result<(), failure::Error> {
    let kubeconfig = config::load_kube_config()
        .or_else(|_| config::incluster_config())
        .expect("kubeconfig failed to load");
    let client = APIClient::new(kubeconfig);
    let namespace = std::env::var("NAMESPACE").unwrap_or_else(|_| "default".into());
    let project = std::env::var("PROJECT").expect("PROJECT env var is required");
    let sleep_time = std::time::Duration::from_secs(60 * 5);

    loop {
        std::thread::sleep(sleep_time);
        println!("Generating event");
        let secret = generate_secret(project.as_str());
        let data = serde_json::to_vec(&secret)?;
        let pp = PostParams::default();
        match Api::v1Secret(client.clone())
            .within(namespace.as_str())
            .create(&pp, data)
        {
            Ok(_) => println!("Sent Brigade event"),
            Err(e) => println!("Error sending event: {}", e),
        };
    }
}
```

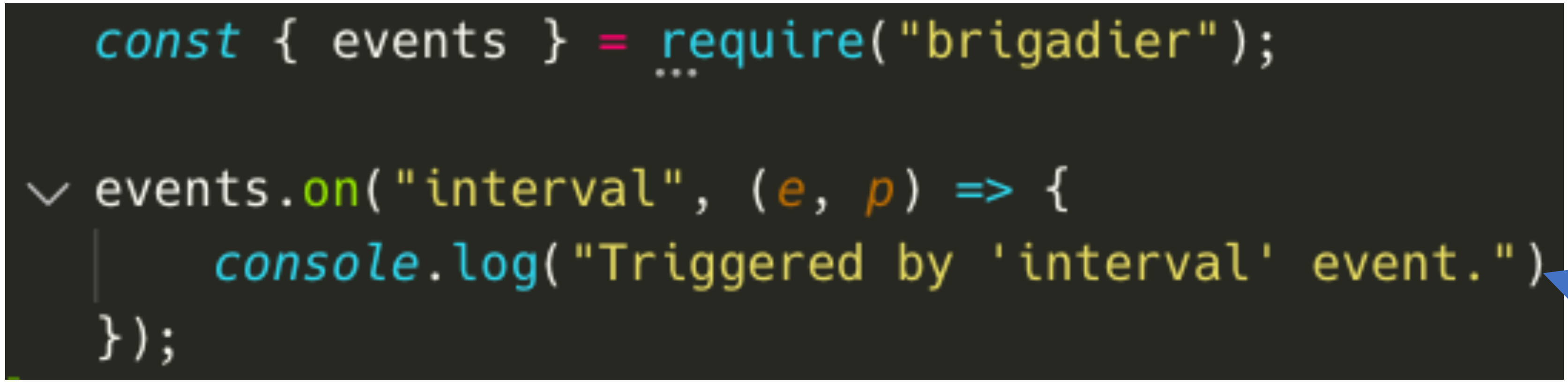
Demo: A Minimal Gateway

KubeCon CloudNativeCon North America 2019

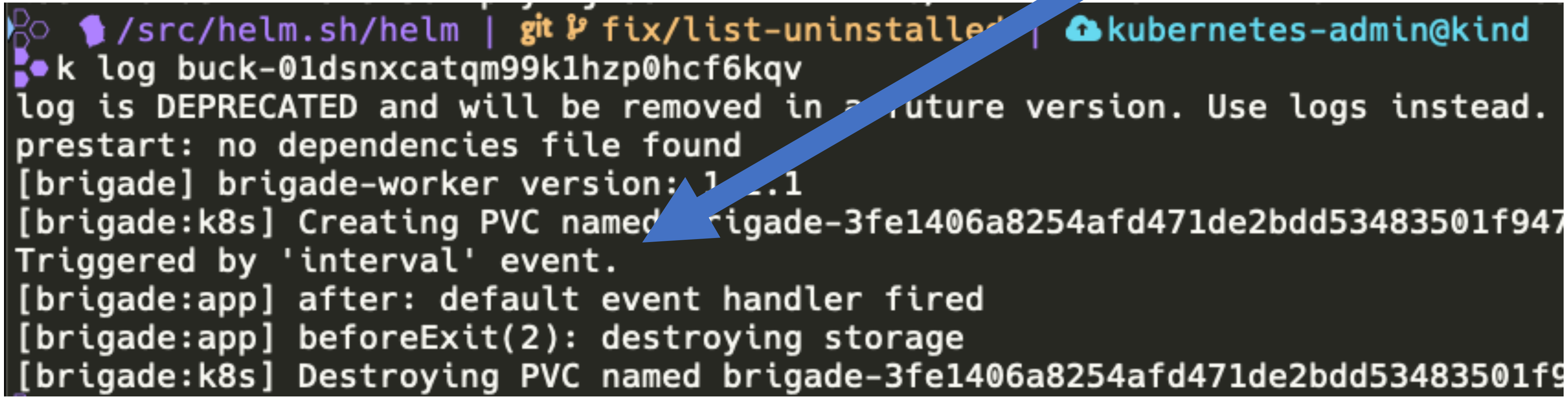


```rust
fn generate_secret(project: &str) -> serde_json::Value {
    let uid = ulid::Ulid::new().to_string().to_ascii_lowercase();
    json!({
        "apiVersion": "v1",
        "kind": "Secret",
        "metadata": {
            "name": format!("buck-{}", uid),
            "labels": {
                "project": project,
                "build": uid.as_str(),
                "component": "build"
            }
        },
        "type": "brigade.sh/build",
        "data": {
            "event_provider": base64::encode("buck"),
            "event_type": base64::encode("interval"),
            "project_id": base64::encode(project),
            "build_name": base64::encode(project),
            "build_id": base64::encode(uid.as_str()),
            "payload": base64::encode("hello"),
            "commit_ref": base64::encode("master")
        }
    })
}
```

Demo: A Minimal Gateway



```
const { events } = require("brigadier");

events.on("interval", (e, p) => {
    console.log("Triggered by 'interval' event.")
});
```



```
/src/helm.sh/helm | git fix/list-uninstalled | kubernetes-admin@kind
k log buck-01dsnxcatqm99k1hzp0hcf6kqv
log is DEPRECATED and will be removed in a future version. Use logs instead.
prestart: no dependencies file found
[brigade] brigade-worker version: 1.2.1
[brigade:k8s] Creating PVC named brigade-3fe1406a8254afd471de2bdd53483501f947
Triggered by 'interval' event.
[brigade:app] after: default event handler fired
[brigade:app] beforeExit(2): destroying storage
[brigade:k8s] Destroying PVC named brigade-3fe1406a8254afd471de2bdd53483501f9
```

KubeCon | CloudNativeCon

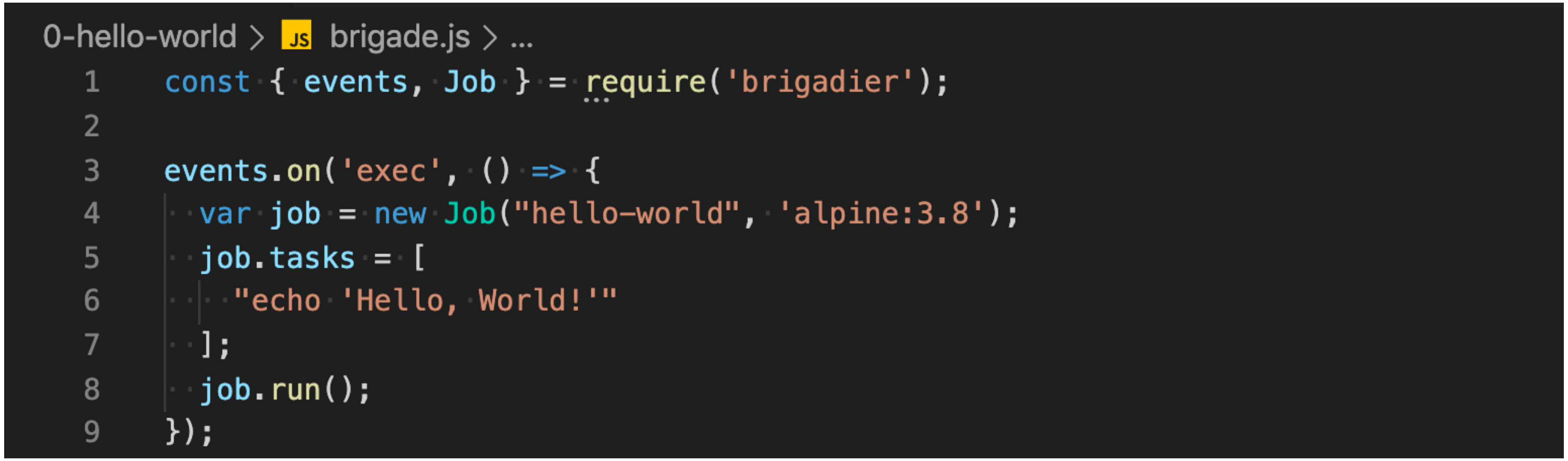
North America 2019

Part 2: Customizing the Brigade Worker

Three Approaches

1. Use `brigade.json` file.
   Add NPM packages before `brigade.js` executes.
2. "Extend" the default worker image.
   Add new NPM or system-level packages.
3. Create a worker image from scratch.
   Do something completely different that is still Brigade-compatible.

Hello, World!


0-hello-world > brigade.js > ...

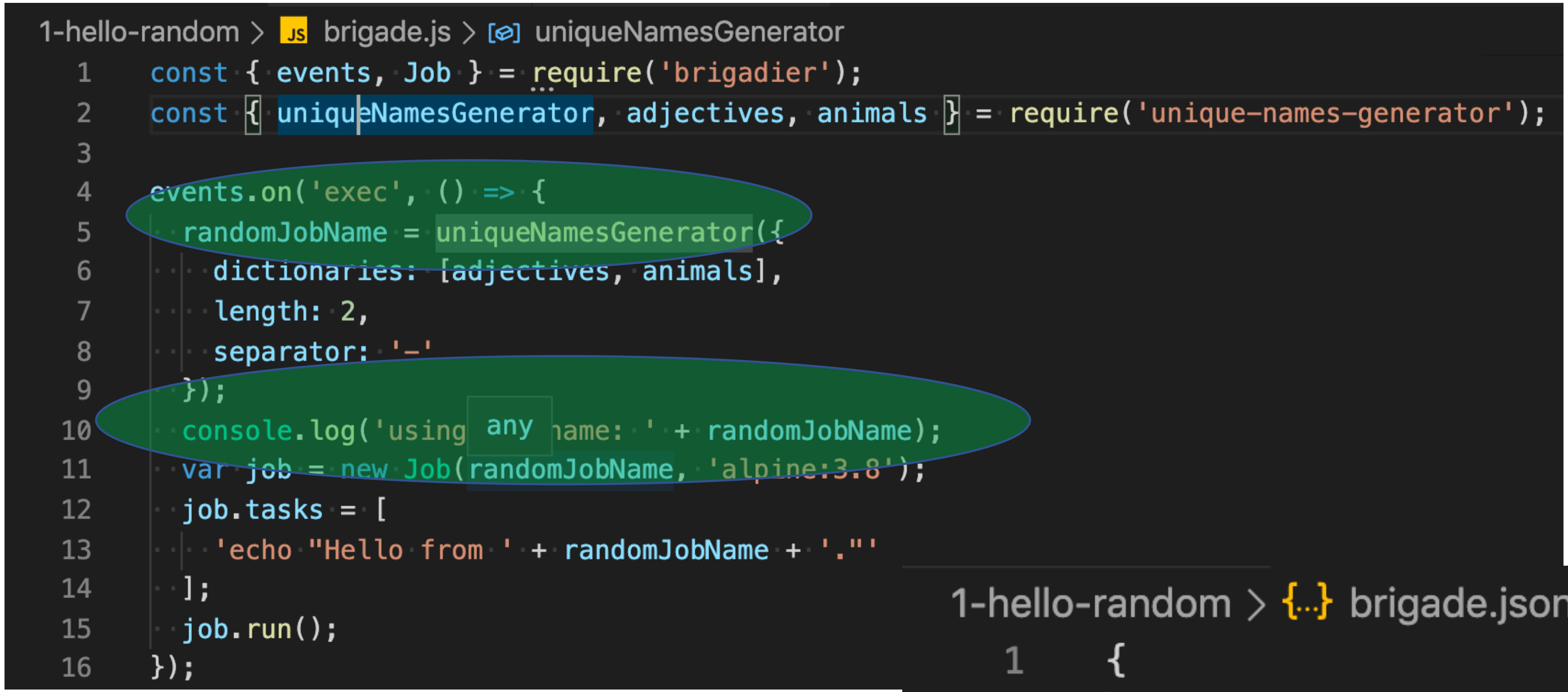
```javascript
const { events, Job } = require('brigadier');

events.on('exec', () => {
  var job = new Job("hello-world", 'alpine:3.8');
  job.tasks = [
    "echo 'Hello, World!'"
  ];
  job.run();
});
```

```
$ brig project create
```

- **VCS or no-VCS project:** `no-VCS`
- **Project Name:** `hello-world`
- **Upload a default brigade.js script:** `0-hello-world/brigade.js`
- Accept defaults for everything else.

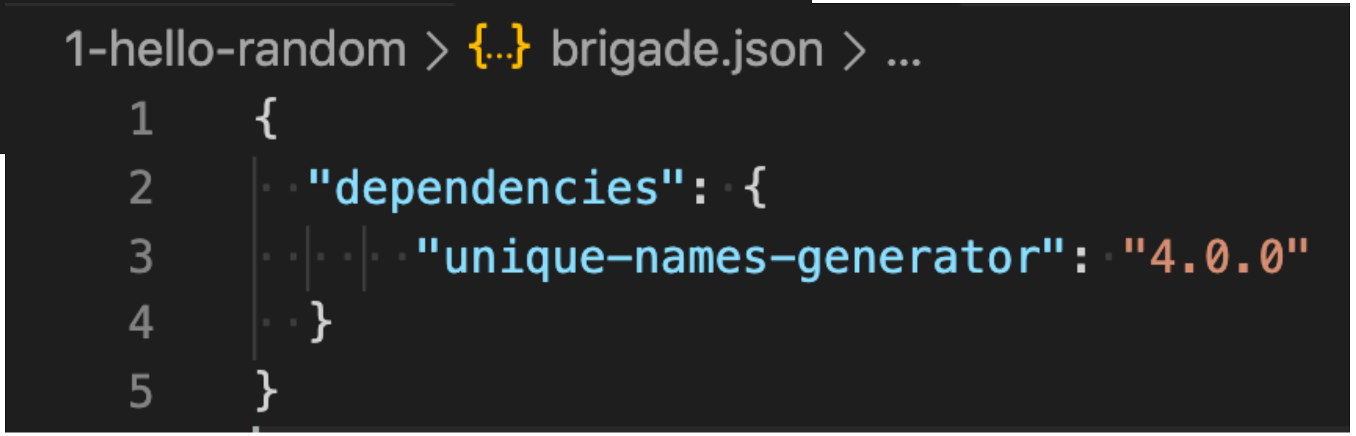
```
$ brig run hello-world
```

Hello, Random!



1-hello-random > brigade.js > uniqueNamesGenerator

```javascript
const { events, Job } = require('brigadier');
const { uniqueNamesGenerator, adjectives, animals } = require('unique-names-generator');

events.on('exec', () => {
  randomJobName = uniqueNamesGenerator({
    dictionaries: [adjectives, animals],
    length: 2,
    separator: '-'
  });
  console.log('using any name: ' + randomJobName);
  var job = new Job(randomJobName, 'alpine:3.8');
  job.tasks = [
    'echo "Hello from ' + randomJobName + '."'
  ];
  job.run();
});
```



1-hello-random > brigade.json > ...

```json
{
  "dependencies": {
    "unique-names-generator": "4.0.0"
  }
}
```

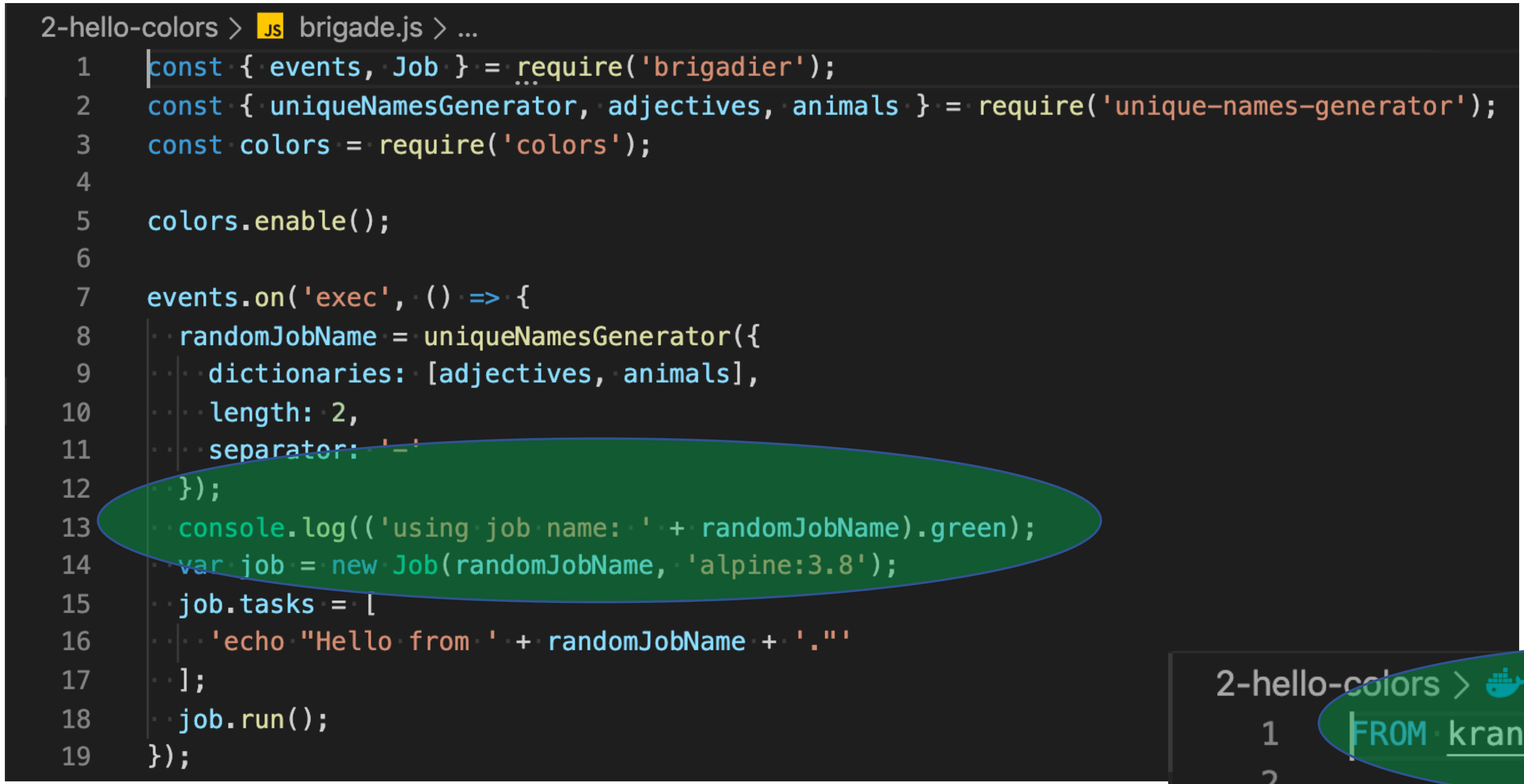
Hello, Random!

```
$ brig project create
```

- **VCS or no-VCS project:** `no-VCS`
- **Project Name:** `hello-random`
- **Upload a default brigade.js script:** `1-hello-random/brigade.js`
- Accept defaults for everything else.

```
$ brig run hello-random --config 1-hello-random/brigade.json
```

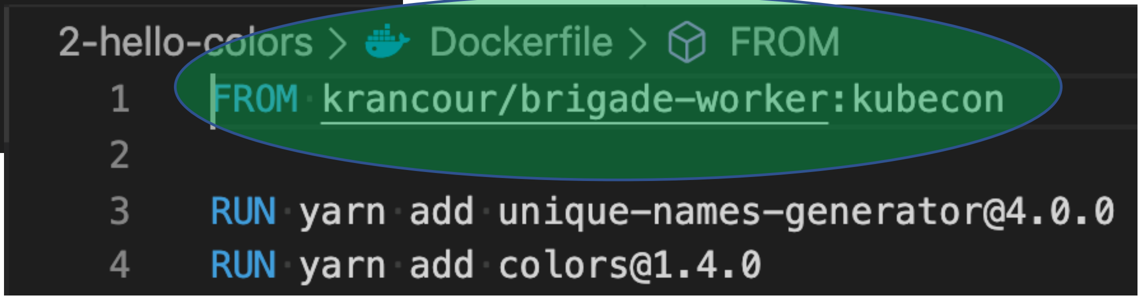
Hello, Colors!



2-hello-colors > brigade.js > ...

```javascript
const { events, Job } = require('brigadier');
const { uniqueNamesGenerator, adjectives, animals } = require('unique-names-generator');
const colors = require('colors');

colors.enable();

events.on('exec', () => {
  randomJobName = uniqueNamesGenerator({
    dictionaries: [adjectives, animals],
    length: 2,
    separator: '-'
  });
  console.log(('using job name: ' + randomJobName).green);
  var job = new Job(randomJobName, 'alpine:3.8');
  job.tasks = [
    'echo "Hello from ' + randomJobName + '."'
  ];
  job.run();
});
```

The image referenced in `FROM` was built from the head of the master branch, but you can usually just start with `brigadecore/brigade-worker:v1.2.1`



2-hello-colors > Dockerfile > FROM

```dockerfile
FROM krancour/brigade-worker:kubecon

RUN yarn add unique-names-generator@4.0.0
RUN yarn add colors@1.4.0
```

I've pre-built this and pushed it to `krancour/brigade-worker:colors`

Hello, Colors!

```
$ brig project create
```

- **VCS or no-VCS project:** `no-VCS`
- **Project Name:** `hello-colors`
- **Upload a default brigade.js script:** `2-hello-colors/brigade.js`
- **Configure advanced options:** `Y`
  - **Worker image registry or DockerHub org:** `krancour`
  - **Worker image name:** `brigade-worker`
  - **Custom worker image tag:** `colors`
- Accept defaults for everything else.

```
$ brig run hello-colors
```

Now for Something Completely Different

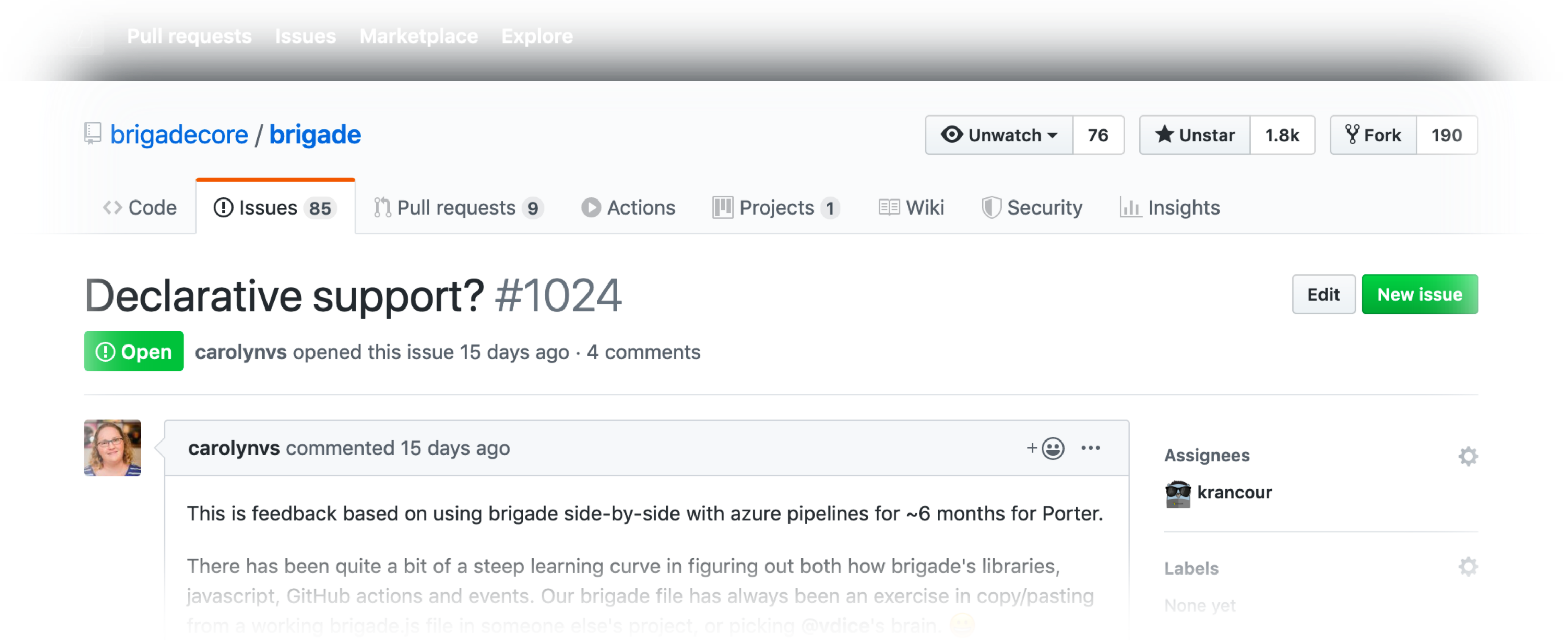
Starting from Scratch

Want to do something completely different with your worker?

The sky’s the limit as long as you:

- Consume worker configuration from the same sources as the default worker:
    - Environment variables
    - Project secrets (Kubernetes secrets)
- For each job, name and label the corresponding pod the same way the default worker would.

Declarative Pipelines?



Declarative support? #1024

carolynvs opened this issue 15 days ago · 4 comments

carolynvs commented 15 days ago

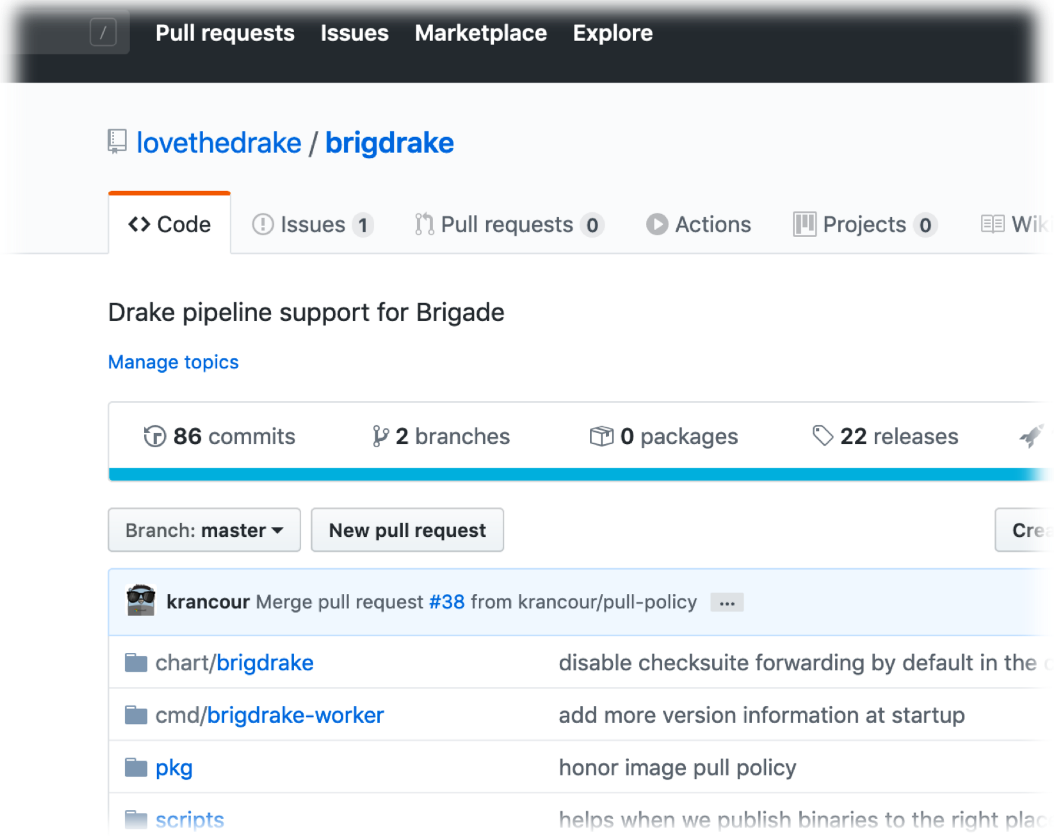
This is feedback based on using brigade side-by-side with azure pipelines for ~6 months for Porter.

There has been quite a bit of a steep learning curve in figuring out both how brigade's libraries, javascript, GitHub actions and events. Our brigade file has always been an exercise in copy/pasting from a working brigade.js file in someone else's project, or picking @vdice's brain.

Love The Drake!

The DrakeSpec is a (draft) open specification for declarative pipelines.

BrigDrake is a DrakeSpec-compliant pipeline executor that is also a Brigade-compatible worker!



lovethedrake / brigdrake

Drake pipeline support for Brigade

86 commits | 2 branches | 0 packages | 22 releases

krancour Merge pull request #38 from krancour/pull-policy ...

chart/brigdrake	disable checksuite forwarding by default in the
cmd/brigdrake-worker	add more version information at startup
pkg	honor image pull policy
scripts	helps when we publish binaries to the right plac

Hello, Drake!

This is not even JavaScript!

```
$ brig project create
```

- **VCS or no-VCS project:** `no-VCS`
- **Project Name:** `hello-drake`
- **Upload a default brigade.js script:** `3-hello-drake/Drakefile.yaml`
- **Configure advanced options:** `Y`
  - **Worker image registry or DockerHub org:** `lovethedrake`
  - **Worker image name:** `brigdrake-worker`
  - **Custom worker image tag:** `v0.21.0`
  - **Worker command:** `/brigdrake/bin/brigdrake-worker`
- Accept defaults for everything else.

```
$ brig run hello-drake --event foobar
```