Sandboxing User Node.JS Code with Docker and CoreOS

Hi!

- Developer Evangelist at Auth0
- Austinite
- Catparent
- Javascript addict

The serious question

Why on **earth** would you want to open yourself up to third-party, arbitrary code that runs on your servers?

There's a few answers to this question

- Do you have a developer-oriented product?
- Do you want to add a serious amount of flexibility to it?
- Do you want to make devs lives a little easier?

How we use Sandboxing at Auth0

It started with the rules engine

[picture of rules engine]

And we eventually spun it off into its own product!

Webtask.io is a product aimed to help developers create quick serverless architectures, without worrying about deploys and management.

[picture of webtask]

We love open source!

I used the following Auth0 open-source projects to complete the demos you'll see today:

- webtask-runtime (Which is at the heart of parsing/ transpiling and running the arbitrary code)
- express-jwt

Why we love sandboxing

- Our users can extend the product to suit their needs (and so can we!)
- Developers love to hear "We do the bulk of the work, so you can focus on code."
- Not restricting them to a few options frees us from having to say no.

But this isn't easy!

Some serious concerns with Sandboxing

- Isolation
- Security
- Scalability
- Flexibility

Isolation

We want users to run their code in an environment where even if they manage to wreak havoc, they're only wreaking havoc on their own runtime environment, not their neighbors!

How Docker helps with this

- Each container is separated-- by default no file system access shared between containers
- Each container runs a REST API to handle bits of code
- Need a db? Extra bits and pieces? Docker images and networking makes this so much easier

How CoreOS helps with isolation

- You can use chroot jails to add another layer of protection between containers
- If you need to spin down a misbehaving machine, you can leave other tenants running
- If a user manages to take their server down, we can gracefully recover

Demo: my docker image that runs Node.JS code sent to it.

Security

Not only do we have to prevent the third-party code form affecting its neighbors, we have to prevent users from accessing code bits that aren't theirs!

How Docker helps with this

- Besdies the aforementioned security in isolation, each tenant can have its own db instance
- When using JWTs, it's easy to spin up a new docker instance with that client's secret
- Security patch to a Node Module? An OS? No big deal!

How CoreOS helps with this

- CoreOS isn't just good at helping services it's running see each other-- it's great at keeping them separate!
- CoreOS and fleet make it easy to spin out new secuirty patches without downtime through automated image spinup and data movement

Side Note: JWTs

- Pretty handy way to do authentication tokens
- Web Standard RFC 7519
- Encoded and signed JSON objects
- Need encryption? No problem-- JWE and JOSE have you covered!

Demo: Adding JWT support to my Docker container, and spinning up 2 different tenants

Scalability

We need to be able to accomodate any and all user requests without latency spikes or d***time!

How docker helps

- Need a new container? Spin it up!
- We know we're getting the same environment every time
- Automation + consistency = <3

How CoreOS helps

- Extremely customizable for dynamic scaling
- Quickly scale horizontally, and not terribly hard to scale vertically.

Flexibility

It's one thing to let users run arbitrary code on our servers, but it's another to do it and let them get what they want done.

Security aids flexibility

- Because we know users probably* can't wreak havoc on any machines but their own instance, we can let them do things like:
- Make HTTP requests
- Install third-party modules

Demo: Installing modules from npm in our sandbox

Scalability aids Flexibility

- Users can use as much CPU and RAM as we allot to their container without affecting neighbors
- We can scale containers vertically and horizontally as needed, allowing folks to do bigger, better things, faster!

Some general tips and tricks for sandboxing

Watch out for infinite loops!

- Especially in single-threaded systems (like Node!)
- Have a timeout watcher, and alert the user with an error if they trip it
- (Be careful about accidentally cutting off slow HTTP requests, though!)

Issues with allowing third-party module installation

- Adds latency to the runtime of the arbitrary code
- Can be dangerous from a security perspective
  - Webtask.io whitelists npm modules
- Native modules (in Node.JS)
  - You're never quite sure every single module will compile correctly on your container image...

Thanks for listening!

- kas@auth0.com
- (Come find me for an Auth0 sticker!)
- @nodebotanist on Twitter, GitHub
- Code/Slides on GitHub