It's The Future

Hey, my boss said to talk to you - I hear you know a lot about web apps?

-Yeah, I'm more of a distributed systems guy now. I'm just back from ContainerCamp and Gluecon and I'm going to Dockercon next week. Really excited about the way the industry is moving - making everything simpler and more reliable. It's the future!

Cool. I'm just building a simple web app at the moment - a normal CRUD app using Rails, going to deploy to Heroku. Is that still the way to go?

-Oh no. That's old school. Heroku is dead - no-one uses it anymore. You need to use Docker now. It's the future.

Oh, OK. What's that?

-Docker is this new way of doing containerization. It's like LXC, but it's also a packaging format, a distribution platform, and tools to make distributed systems really easy.

Containeri.. — what now? What's LXE?

-It's LXC. It's like chroot on steroids!

What's cher-oot?

-OK, look. Docker. Containerization. It's the future. It's like virtualization but faster and cheaper.

Oh, so like Vagrant.

-No, Vagrant is dead. Everything is going to be containerized now, it's the future.

OK, so I don't need to know anything about virtualization?

-No, you still need virtualization, because containers don't provide a full security story just yet. So if you want to run anything in a multi-tenant environment, you need to make sure you can't escape the sandbox.

OK, I'm getting a little lost here. Let's back it up. So there's a thing like virtualization, called containers. And I can use this on Heroku?

-Well, Heroku has some support for docker, but I told you: Heroku's dead. You want to run your containers on CoreOS.

OK, what's that?

-It's this cool Host OS you can use with Docker. Hell, you don't even need Docker, you can use rkt.

Rocket?

-No, rkt.

Right, Rocket.

-No, it's called rkt now. Totally different. It's an alternative containerization format that isn't as bundled together as Docker is, and so it's more composable.

Is that good?

-Of course it's good. Composability is the future.

OK, how do you use it?

-I don't know. I don't think anyone uses it.

Sigh. You were saying something about CoreOS?

-Yeah, so it's a Host OS you use with Docker.

What's a Host OS?

-A Host OS runs all your containers.

Runs my containers?

-Yeah, you gotta have something run your containers. So you set up like an EC2 instance, you put CoreOS on it, then run the Docker daemon, and then you can deploy Docker images to it.

Which part of that is the container?

-All of it. Look, you take your app, write a Dockerfile, turn it into an image locally, then you can push that to any Docker host.

Ah, like Heroku?

-No, not Heroku. I told you. Heroku is dead. You run your own cloud now using Docker.

What?

-Yeah, it's real easy. Look up #gifee.

Gify?

-"Google's infrastructure for everyone else". You take some off the shelf tools and stacks, using containers, and you can have the same infrastructure Google has.

Why don't I just use Google's thing?

-You think that's going to be around in 6 months?

OK, doesn't someone else do hosting of this stuff? I really don't want to host my own

stuff.

-Well, Amazon has ECS, but you gotta write XML or some shit.

What about something on OpenStack?

-Ew.

Ew?

-Ew.

Look I really don't want to host my own stuff.

-No, it's really easy. You just set up a Kubernetes cluster.

I need a cluster?

-Kubernetes cluster. It'll manage the deployments of all your services.

I only have one service.

-What do you mean? You have an app right, so you gotta have at least 8-12 services?

What? No, just one app. Service, whatever. Just one of them.

-No, look into microservices. It's the future. It's how we do everything now. You take your monolithic app and you split it into like 12 services. One for each job you do.

That seems excessive.

-It's the only way to make sure it's reliable. So if your authentication service goes down...

Authentication service? I was just going to use this gem I've used a few times before.

-Great. Use the gem. Put it into it's own project. Put a RESTful API on it. Then your other services use that API, and gracefully handle failure and stuff. Put it in a container and continuously deliver that shit.

OK, so now that I've got dozens of unmanageable services, now what?

-Yeah, I was saying about Kubernetes. That let's you orchestrate all your services.

Orchestrate them?

-Yeah, so you've got these services and they have to be reliable so you need multiple copies of them. So Kubernetes makes sure that you have enough of them, and that they're distributed across multiple hosts in your fleet, so it's always available.

I need a fleet now?

-Yeah, for reliability. But Kubernetes manages it for you. And you know Kubernetes works cause Google built it and it runs on etcd.

What's etcd?

-It's an implementation of RAFT.

OK, so what's Raft?

-It's like Paxos.

Christ, how deep down this fucking rabbit hole are we going? I just want to launch an app. Sigh. Fuck, OK, deep breaths. Jesus. OK, what's Paxos?

-Paxos is like this really old distributed consensus protocol from the 70s that no-one understands or uses.

Great, thanks for telling me about it then. And Raft is what?

-Since no-one understands Paxos, this guy Diego...

Oh, you know him?

-No, he works at CoreOS. Anyway, Diego built Raft for his PhD thesis cause Paxos was too hard. Wicked smart dude. And then he wrote etcd as an implementation, and Aphyr said it wasn't shit.

What's Aphyr?

-Aphyr is that guy who wrote, 'Call Me Maybe.' You know, the distributed systems and BDSM guy?

What? Did you say BDSM?

-Yeah, BDSM. It's San Francisco. Everyone's into distributed systems and BDSM.

Uh, OK. And he wrote that Katy Perry song?

-No, he wrote a set of blog posts about how every database fails CAP.

What's CAP?

-The CAP theorem. It says that you can only have 2 out of 3 of Consistency, Availability and Partition tolerance.

OK, and all DBs fail CAP? What does that even mean?

-It means they're shit. Like Mongo.

I thought Mongo was web scale?

-No one else did.

OK, so etcd?

-Yeah, etcd is a distributed key-value store.

Oh, like Redis.

-No, nothing like Redis. etcd is distributed. Redis loses half its writes if the network partitions.

OK, so it's a *distributed* key-value store. Why is this useful?

-Kubernetes sets up a standard 5-node cluster using etcd as a message bus. It combines with a few of Kubernetes' own services to provide a pretty resilient orchestration system.

5 nodes? I have one app. How many machines am I gonna need with all this?

-Well, you're going to have about 12 services, and of course you need a few redundant copies of each, a few load balancers, the etcd cluster, your database, and the kubernetes cluster. So that's like maybe 50 running containers.

WTF!

-No big deal! Containers are really efficient, so you should be able to distribute these across like 8 machines! Isn't that amazing?

That's one way to put it. And with all this, I'll be able to simply deploy my app?

-Sure. I mean, storage is still an open question with Docker and Kubernetes, and networking will take a bit of work, but you're basically there!

I see. OK, I think I'm getting it.

-Great!

Thanks for explaining it.

-No problem.

Let me just repeat it back to see if I've got it right.

-Sure!

So I just need to split my simple CRUD app into 12 microservices, each with their own APIs which call each others' APIs but handle failure resiliently, put them into Docker containers, launch a fleet of 8 machines which are Docker hosts running CoreOS, "orchestrate" them using a small Kubernetes cluster running etcd, figure out the "open questions" of networking and storage, and then I continuously deliver multiple redundant copies of each microservice to my fleet. Is that it?

-Yes! Isn't it glorious?

I'm going back to Heroku.

Want to use Docker and Continuously Deliver that shit? Check out CircleCI's Docker Support.

Our followup.

- Integrations
- Users
- Industry