we write about the things we build and the things we consume

written by Jamie Perkins on 1 June 2016 in devops, engineering

coming up in kubernetes

kubernetes 1.3

If you didn't know, we are currently in the middle of migrating our infrastructure from VMs on AWS to Kubernetes on VMs on AWS. We are holding off for version 1.3 which is due to be released on June 24th. In the meantime we are making all the preparations we can with testing on 1.2, planning our migration and writing all the configuration we can.

I thought I'd write about 2 major additions the new version brings to Kubernetes.

petsets

Petsets solve a problem with stateful applications and services. You may know in Kubernetes the smallest unit of deployment is a Pod. Pods are ephemeral by design, akin to a running instance of a container image and nuking it when it stops. When the Pod dies, it is permanently gone too, replaced by a new instance with a fresh filesystem, new network identity and all the rest.

This is generally fine except in the cases where an instance of your application wants to survive restarts and stops with it's filesystem and identity intact. For example in the case of a database node.

Petsets solve this problem by giving a unique and stable identity to a Pod. This is important for clustered services that need stable identities to refer to when bootstrapping a cluster or seeding additional nodes. The stable ID allows Pods to retrieve the data (volume) associated with a particular identity, meaning `db.node1` still holds the same data between restarts.

ubernetes (aka kubernetes cluster federation)

This is basically what it says on the tin. Kubernetes as of version 1.2 officially only supports single master, multi slave deployments. This works fine but leaves a single point of failure on the master node, which handles cluster state changes and hosts the Kubernetes API.

Ubernetes aims to place a control plane on top individual Kubernetes clusters to support things like failover between clusters running in different availability zone. Hopefully, in practice this means the automatic and dynamic rescaling of services and applications in response to failure of a cluster and/or availability zone.

Ubernetes actually goes a step further than that. It aims to support the use-case of

multiple Kubernetes clusters hosted across different cloud providers (e.g. GCE and AWS) and optionally on-premise bare metal. This is nice, but not something we are likely to use being quite comfortable in Amazon's warm embrace.

One more thing to mention is the changes to the script used to bootstrap a Kubernetes cluster. Titled `kube-up.sh`, it handles deploying the master and minion nodes, their network configuration and so on. In the case of AWS this means picking the AMI, setting up a VPC, gateways, subnets and more. This is being reworked in v1.3 to support Ubernetes, which should remove the manual work that is needed to setup the same in v1.2.

thats all

Minor disclaimer that anything I've written was parsed from the Github issues and discussions. The general concepts of Petsets and Ubernetes are blockers to the 1.3 release for the Kubernetes team. Their implementation and particular details may vary before release though, so do your own research on if they are right for you.

We're looking forward to lots of things from Kubernetes. Our playing around with v1.2 looked like it could remove a lot of the pain of dealing with heterogenous infrastructure and applications, each configured in different and special ways. Are you using or thinking about Kubernetes in production? Why not tell us how thats going?

If you enjoyed the read, drop us a comment below or share the article, follow us on Twitter or subscribe to our #MetaBeers newsletter. Before you go, grab a PDF of the article, and let us know if it's time we worked together.