HONEYCOMB & BEYOND

Order of the Bee

Martin Cosgrave

28.04.2016

martin (a) bettercode.com

SLIDE: HONEYCOMB & BEYOND

MARTIN

Welcome to our talk "Honeycomb & Beyond", and thank you all for coming.

I'm Martin Cosgrave, I'm an Alfresco enthusiast and founder member and current board member of the Order of the Bee.

In a professional capacity I play with Java, Devops and distributed systems.

DAREN

I'm Daren Firminger, I work for Digital Catalyst, we build Enterprise applications using Open Source frameworks.

We have been working with Alfresco since version 3.

SLIDE: WHY HONEYCOMB

DAREN

One of the very first goals of The Order Of The Bee was to create a distribution of Alfresco Community to serve as an ready to go, "out of the box" demonstration of the capabilities of the Community Edition.

After the successful reception of the Order of the Bee project at the London Alfresco Summit, Martin and I agreed to collaborate to create a base upon which we could build a candidate for the Honeycomb distribution.

MARTIN

We wanted it to be a fully functional product out of the box, so it should come with various extras which normally would be added on by the consultant implementing the particular Alfresco project.

Instead we were looking to create a standalone install that would allow small businesses to use Alfresco with the minimum of fuss.

I had been doing some work with puppet, and I thought that, while it was not really the usual way to use that tool, we could use it to make an installer.

SLIDE: WHY PUPPET?

DAREN

We had tried building shell-script based installers, but we were frustrated by what happened when a phase of the install failed.

We would be left with a system in an unstable state, and re-running the script would generally fail, unless you ignored the errors from each phase.

Using Puppet we could guarantee a machine's state; a puppet run will take a machine from its current state to the state described by the puppet manifests, so it didn't matter if the install had not completed due to some transient error, the next run would again attempt to bring the machine to the required state.

MARTIN

Puppet also gives you the advantage of recording your desired system state as a series of text files, hence being a suitable candidate for source control.

We were able to add additional features not present in the community installer, such as importing SSL certificates for use and setting up an apache reverse proxy in front of Tomcat.

A text based configuration tool allowed customisation of other options such as database server location and credentials.

SLIDE: SOME PUPPET EXAMPLES

DAREN

While we don't want to go into a lot of details about puppet configuration it might be useful to see an example or two of the syntax. This first slide shows a fragment of a puppet manifest which sets up some different variables based on which version of Alfresco we are trying to build. You can see for example the 'dollar-indexer' variable which is set to a different value dependent on the version. These values can then be inserted into templated files, for example we build the alfresco-global.properties file this way.

SLIDE: MINIMAL ADDON

This next slide shows a complete puppet class to install an addon. In this case the jar artefact was already pre-built as part of the share-extras project and we simply need to download the jar and store it in the right place on the filesystem.

MARTIN

Puppet turned out to have quite a learning curve, and it took us a while to find a good structure for the puppet build.

We were keen to make it easy to include addons and the structure we chose reflected that, but we soon realised that we would need to have extensive tests so that we would be able to verify the functioning of the system after adding new extensions to the build.

Daren started working on a suite of tests using Selenium and python to verify the state of the system, logging in, creating sites, checking that previews were rendered properly, etc., and then we decided we would need a framework to run the tests automatically.

We started to look at Jenkins for continuous integration, but we faced difficulties due to the fact that we were trying to test a whole system rather than an artefact of a programming language

There were some Jenkins extensions that looked promising in that regard but in the end we dropped our attempts to use Jenkins for the purpose and look around for another solution.

DAREN

Then we came across the Travis CI project. Travis was interesting since it offered a free CI tier which could watch our public github repo and automatically build the latest version and run our tests.

We were especially interested when we realised that Travis built a fresh Amazon VM for every run, it seemed like just what we needed, and we started running a matrix of our supported versions and platforms, that is 4.2.f and 5.0.x, on Ubuntu and CentOS, on every github commit.

At this point we should probably publicly apologise to the Travis team for the abuse we wrought on their servers! We soon realised however that we were abusing the Travis infrastructure when we started running up against the timeouts for a complete Travis run, and hence our test runs were often giving us false negatives.

MARTIN

Eventually I wrote a "quick and dirty" testing rig for our puppet build which used Vagrant to drive a series of Digital Ocean VMs in parallel.

This mini project was supposed to collate a build report after successfully launching and testing each combination of platform and version buy it was never quite completed, however it was good enough for us to test our development as we pleased.

We faced another interesting issue during this development. Sometimes Tomcat would take an extremely long time to start up, in the order of ten minutes or more.

This would happen more often in a VM based setup than in a vagrant based build.

After a lot of head scratching we discovered that we were exhausting the entropy on the VM host! This meant that Tomcat spent a lot of time during startup trying to gather enough entropy to initialise its random number generator.

We found an entropy generator called HAVEGED and incorporated it into the build and that frustrating issue went away.

SLIDE: LESSONS LEARNT:

DAREN

So what did we learn during this process? Well, getting collaboration going is hard. Despite there being a working group within Order of the Bee to deliver a Honeycomb distribution, we found little enthusiasm.

Perhaps this is due to the strangeness of puppet for many Alfresco developers. I think we can understand that, after all we've gone through that learning curve too.

DAREN

Perhaps it would have been better to drop the puppet stuff and use Chef instead, after all, Maurizio Pillitu had already done some work on his alfresco-boxes project at that time, which used Chef.

We tried to decipher what Maurizio had done, and perhaps we were limited by what we had already learned about puppet getting in the way of learning about Chef. In the event it would have involved another steep learning curve, at a time when we were just starting to get puppet under our belts, and so we persevered with puppet.

MARTIN

We discovered that people are obsessed with version numbers!

We were plugging away with an unversioned github repo for some time and eventually in February 2015 we released an alpha version which we called "Feature Complete".

We were not certifying it as bug-free but at that point we were not planning to add any more features. (In the event I think we did end up adding more features after that point).

However despite making a release we saw very little response, and there was a feeling that it could not be considered production ready.

My own feeling was that we were packaging up community alfresco, which we definitely considered to be a production ready product, with some additional capabilities that would ordinarily need to be implemented by a consultant, and so I didn't feel that it was any less production ready than Alfresco CE itself.

Eventually we decided that we would need to work on a release that could be considered to be production ready out of the box.

DAREN

We also made a painful realisation that many distributors and packagers of software have made before us: your upstream sources can totally screw you up if they decide to move things around or change their distribution layout.

For example, as well as supporting 4.2.f and the 5.0.x branch we also had an option to build the nightlies that Alfresco published. However, during our development the method of publishing the nightlies changed and we spent some time trying to fit in with the new changes, but in the end we never managed to stay on top of them.

MARTIN

We learnt some things about puppet, too. It actually works pretty well as an installer. The repeatability gives a sturdiness to the install process that is missing from shell script approaches.

You could make a shell script as robust as puppet but then you would end up reimplementing a lot of the features that puppet already has to keep its builds idempotent.

Puppet is actually designed to be controlled from a central puppetmaster server, and during the development I started to use a system called Foreman, which is an open source version of Puppet Enterprise, and contains a puppet master. I was encouraged to find that Foreman itself used standalone puppet to install and upgrade itself. It was nice to find we were not alone!

The development lifecycle for puppet is pretty slow, since often you need to build a system from scratch to ensure that you are getting a reliable build. However our vagrant test rig mitigates that quite well by executing the builds on relatively fast and cheap Digital Ocean VMs.

SLIDE: FUTURE PUPPET-ALFRESCO

DAREN

Currently we have not upgraded puppet-alfresco to be able to build the 5.1.x releases of Alfresco. I don't think we ever intended this to be the final release of Honeycomb, rather it was supposed to be an example upon which to build, to add new addons, etcetera.

With 5.1.x we have seen the installation change drastically, changing far more than the transition from 4.2.f to 5.0.x, and the big question is whether we bite the bullet and dive back in to implement the 5.1 changes. If we do so, are we going to have the game change from underneath us again when we hit 5.2?

MARTIN

If we had more people ready and willing to try Honeycomb and report on their successes and failures, that would be a key motivating factor to get more work done on it.

Without at least that, it can seem like we are working in a vacuum, potentially this could take up several more person months, and after that we could be faced with more changes from what seems at times to be a hostile upstream.

What would be really great though would be if we could get one or two more people to help on the puppet development.

We're both happy to mentor people through the learning curve, so it won't be as difficult as it was for us.

I'm sure that the addition of new talent into the project could ensure its being able to track future versions successfully, although we would have to stick to GA versions rather than trying to track early availability.

DAREN

Quick question. How many people in this audience are actually using the 5.1.x version in production?

(pause for response)

We know customers who are still happy with 4.2.f. While the version 5 branches definitely add some great new features, for some customers they are really not relevant or needed. We shouldn't necessarily need to commit to following Alfresco's release schedule slavishly, although obviously from some perspectives that is desirable.

SLIDE: DOGFOODING

Reliability is an important consideration for the honeycomb distribution, finding and fixing all the bugs to ensure it's reliable.

We are running Honeycomb for the speakers' materials for Beecon 2016, demonstrates we are happy to eat our own dog food, and use Honeycomb for mission critical tasks, supporting the Beecon event speakers.

Currently we have deployed honeycomb across 2 machines, on a 6 core machine, which gives us reasonable performance.

It would be interesting to understand how much Alfresco would charge us for an enterprise licence for our configuration?

SLIDE: BEYOND PUPPET

MARTIN

Puppet is great but it is slow to develop, since we often need to clear down a server to a base state and re-run all the manifests for a simple change.

This is not always required but often we need to check that the resources are always applied in order, as puppet does not guarantee resource ordering unless explicitly told, and sometimes when resources are applied out of order the build fails.

In the meantime during our development docker was rapidly becoming the next big thing.

It has some major advantages in respect of the development lifecycle since the system is built up in layers.

Once a layer has been built it is cached locally and as long as neither that layer nor the layers beneath it change, there is no need to rebuild it, leading to massive speed advantages in development compared to puppet.

There is another often quoted advantage for docker, which is that if multiple containers use the same base layer, then it is possible to have that layer exist in memory just once per host.

That does give us some benefit in that if repo, share and solr are all run in containers on a single host, and if we base those containers on a shared tomcat layer, that layer will only be present in memory once.

However, obviously the heap memory cannot be shared, and so the saving is not really great enough to make a difference. It really only comes into play when tens or hundreds of microservices which are based on the same base layer are running on the same host.

DAREN

There have been a few attempts to containerise Alfresco, but as far as we know they have all attempted to put the whole of the stack into a single container. This is not really the docker way.

Our approach was to containerise each part of the stack separately.

However, this raised another issue. When using composition tools like docker-compose or crowdr, it is not easy to control the startup order of the containers.

Again, in a microservices environment this is not a problem, but when containerising a previously monolithic stack it can cause problems if a container starts before other containers that it depends on. Martin chose to deal with this by developing a new docker composition tool called grua.

MARTIN

Grua is Spanish for 'crane', and its naming is intended to extend the docker metaphor to portray a container composition as a stack of containers, such as you would see at a busy port. It uses the analogy of filling the container instead of docker's 'build' metaphor, and 'grua stack' composes the containers into a complete application.

Grua adds explicit dependency ordering to container composition, by specifying that a given container should run before or after another container.

Furthermore, rather than just firing the next container as soon as docker has started the previous one, you can wait for a specific message in the log output before starting the next container. This can give you confidence that each dependency is ready before starting your main application.

A further advantage is the ability to use information from the environment, or from other containers in the stack, to affect the configuration of later running containers.

The current slide shows the URLs for both grua and for the docker-alfresco stack we developed. In fact within the grua documentation I explicitly refer to the docker-alfresco project since it is a complete example for grua, exercising every part of the tool.

The reason for this is simple, I only included the features that I specifically required for Honeycomb, however grua will be extended in the future to include other features and I intend to create a cookbook of container compositions to serve as examples.

SLIDE: GRUA.YAML

DAREN

Grua is controlled by a YAML configuration file. This slide shows an example of building the repo container.

The '**build**' item refers to a directory containing a Dockerfile with the build instructions for this container, along with any files that need to be copied into the container at build time.

The '***dns**' item shows an example of retrieving information from another container, in this case its IP address. The syntax for '**INSPECT**'* follows the same format as the 'docker inspect' command line tool.

'**options**' provides a way to pass arguments to docker which are not handled by grua, in this case we are exposing some ports from the container.

'**upwhen**' controls how grua determines when a container has started. Here it looks for a log message emitted when tomcat starts up, and fails if it doesn't find it within 120 seconds.

The '***before**' stanza states that this container must be stacked before attempting to stack the solr and share containers, and finally '**volumes**'* links filesystem locations between the host and the container.

SLIDE: SCALING

DAREN

Once you've containerised the component parts of Alfresco it becomes possible to think about scaling horizontally by running different parts of the stack on different hosts.

This is not something that grua supports yet although in the future it probably will.

To do this you need to work out a way to get a homogenous network to appear across all the containers on the various hosts.

I did a lot of experimentation with weave, which purports to do exactly that, however to make the weave network secure, you need to enable their encryption and unfortunately that's quite slow.

There might be faster options, perhaps openvpn, or docker's own networking capabilities which have improved to the point where weave may not be necessary any more, but as ever when scaling you need to be smart about where services go within the network.

We had great results from putting a standalone libreoffice on its own host, and it seems likely that repo and share could live on separate hosts quite well, but it's a work in progress.

It does lead however to an intriguing possibility. If we can get some kind of clustering solution into Alfresco Community, whether that is in share's tomcat, or in solr, or perhaps in libreoffice document rendering, then we are in a great position to easily scale containers, perhaps adding load balancing containers and scaling across hosts. We may even be able to dynamically move containers across hosts, using something like KOO-BER-NEAT-EES to scale the whole system up and down, adding more hosts as demand increases and removing them when load settles down. It's a really exciting proposition.

SLIDE: FUTURE STEPS: MONITORING

MARTIN

In the future we'd like to add another fundamental devops feature to Honeycomb: system monitoring using one of the many open source graphing products out there today, such as Cacti, Zabbix or OpenNMS

SLIDE: FUTURE STEPS: INTEGRATIONS

MARTIN

Furthermore, our experiments with integration with other systems have led us to explore Active MQ and Apache Camel. Active MQ is an open source, robust, enterprise-level message queueing system, and Camel is a routing engine which implements the full suite of Enterprise Integration Patterns and many more integrations on top.

DAREN

With these two products in conjunction we can create integration hooks so that any system can communicate with Alfresco, perhaps in ways not yet envisaged by the Alfresco engineers.

Active MQ in particular implements a pattern called 'competing consumer' which is a very efficient load balancer in which the least loaded node gets the next job in the queue.

Using this we can offload heavy workloads to a farm of processing nodes. Imagine an open source transformation server. With this architecture we should be able to implement it quite simply.

Does this sound useful to your organisation? Perhaps you would like to sponsor its creation?

SLIDE: GETTING INVOLVED

MARTIN

By now you're probably all thinking "What great projects, how can I get involved?".

Well they both serve slightly different needs, and they could both use collaboration from more interested parties.

We'd like to get the Honeycomb project to track the 5.1 branch of Alfresco, and we also want to get more people to download it, run it and test their use cases.

If you already know puppet, even better, we'd love you to get involved in the effort to bring Alfresco 5.1 to Honeycomb.

If you don't know puppet but would like to learn, then implementing your favourite addon is the way to go, we'll be happy to mentor you through the puppet learning process.

If you're excited about Docker, and you can tolerate grua(!), then you can get involved in the docker-alfresco project. This is not really Honeycomb, our goal with this project is to get a multi-hosted containerised Alfresco, with a view to getting some clustered features involved.

DAREN

However, docker-alfresco is not intended to be a product yet, it's a proof of concept right now. If you want to get involved in the actual Honeycomb product, puppet is where it's at for now.

But even if you don't know puppet and don't want to learn it, you can still help. You guys are the domain experts for Alfresco far more than us.

It would be really helpful to get more eyes on what we've done. Please, download and run our puppet based Honeycomb installer. It's probably easier than you think!

And then cast your eye over the installed configuration. If you see something that you would do differently on your own community installs, such as improvements you would normally add, please get in touch and let us know, and we'll work out how to include them in the distribution to make Honeycomb the most optimal, reliable build of community Alfresco.

SLIDE: WILL WORK FOR CASH

So that's the story of Honeycomb, past and with your help, future. In the meantime, if you need any help with Alfresco devops, backend development or integration with other systems don't hesitate to get in touch.

QUESTIONS???