

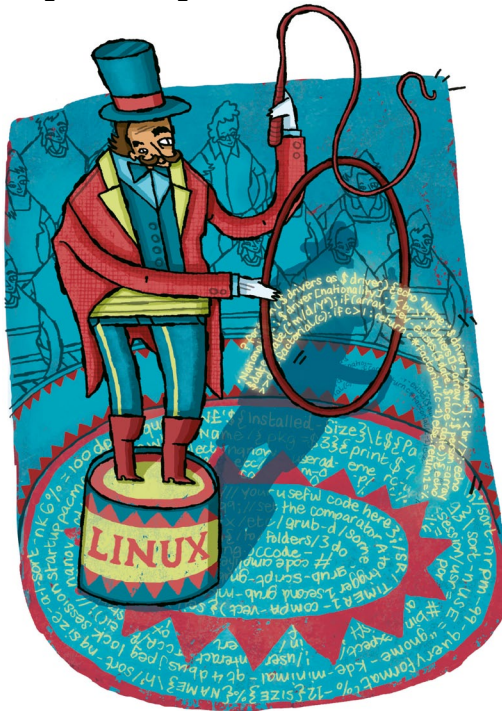
# Juju: Making

Cloud-enable your services using some West African magic and this charming man you may have met before, **Nick Veitch**.



## Our expert

**Nick Veitch** remembers he launched **LXF** with only a set of *Bash* scripts keeping the magazine going. They were replaced with 'people', a retrograde step in his opinion.



**J**uju is great. If you haven't heard of Juju, then you aren't being the best you can be at clouds (see *Juju GUI*, p81 for details on the demo). The underpinning thing about Juju is charms. A charm is a little bundle of distilled expertise that knows how to deploy a particular service – say something like *MySQL* or *WordPress*. But it is more than just a glorified install script: it can also make sense of other services and connect them together (eg joining *MySQL* to *WordPress* so the latter can use the former as a back-end). Charms are, quite simply, magic and in this tutorial, we are going to make some magic happen!

There are a few things you will need in order to create a charm. Although it is possible to do on other distros, it is much easier if you have a recent version of Ubuntu. Precise

Penguin is good (12.04 is a long term release) or Raring Ringtail (13.04). You will also need to install the **juju-core** package, or the older Juju package (for installation details for your distro/platform, see <https://juju.ubuntu.com/docs/getting-started.html>).

It is also a really good idea to install the **charm-tools** package, which has a few functions to help you along. Whatever the case, you should also have an environment where you can create instances with Juju, such as an AWS account, HP Cloud or OpenStack. If you have any difficulty setting up Juju on your cloud provider, check the link above. Assuming that's all in place, we can get started!

OK, so the first thing to do is create the directory structure for your charm, and fill it with the placeholder files we will need. Since you installed **charm-tools**, this is easy.

```
mkdir -p ~/localcharms/precise
cd ~/localcharms/precise
charm create vanilla
```

You will see an error which says 'failed to locate vanilla in apt-cache'. This is fine. If you were creating a charm for a package that was already locally installed, the charm tools would have done a bit more work and filled out some of the metadata for you, but not to worry.

## Creating charms

The first part of call is the **metadata.yaml** file. This contains all the important information that Juju will need to know about the charm, but it actually looks quite simple. The first part is straightforward:

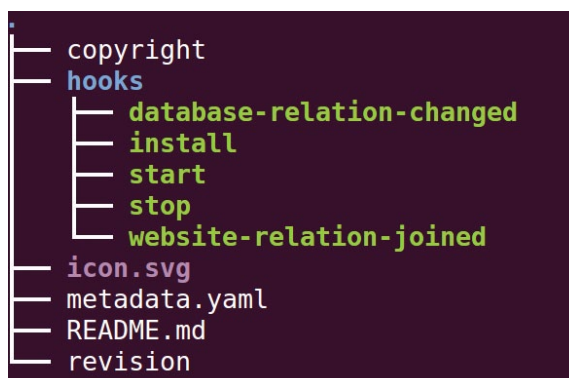
```
name: vanilla
summary: Vanilla is an open-source forum.
maintainer: my name <your@email.tld>
description: |
```

Vanilla is an extendable, multi-lingual forum system. This charm will deploy Vanilla according to the official instructions.

As you can see the YAML format is quite readable, and contains simple **key:value** pairs, which so far have been mostly simple descriptions. The next part is a little more important.

```
categories:
  applications
provides:
  website:
    interface: http
requires:
  database:
    interface: mysql
```

**Categories** is a list, as your charm can be in more than one (put each on a new line). The valid options here are databases, file-servers, applications, cache-proxy, app-servers, miscellaneous. Don't worry about this too much, it is really only for making them searchable online. There are two



» The structure of your charm directory should look like this!

# charms

important sections below though – **provides** and **requires**. The **provides** section includes services our charm will provide. In this case it is a website, so we put that down and, as an indented value, include the interface it will use to deliver this service. The names here don't matter too much – we could have called the service 'web' or 'forum' or something, but it is important to remember the name because we will need to use it for naming our scripts later.

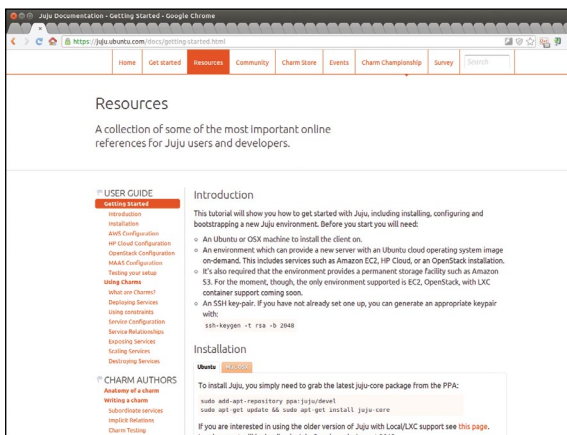
**Requires** is a little more specific, because in this case we will be consuming another service (*MySQL*). If we check the metadata of the *MySQL* charm, we see that in the provides section it states it provides a database with an interface of *MySQL*. That's exactly what we want. This information allows Juju to work out ways in which services can connect, so it knows what scripts to look for.

## Hooking up charms

Okay, now for the clever bit. The **hooks** directory contains executable scripts with specific names which are called upon when certain things happen. Think of it as event handling, but for services. These events can be broken down into two types – lifecycle and environmental. The lifecycle ones deal specifically with events that only happen to our service, and are always named the same – start, stop and install. Let's write one now. We will start with 'start', as it is pretty easy:

```
#!/bin/bash
set -e
service apache2 restart
```

As this is an executable file, it should start with the usual hash bang. An important note here – we are using *Bash* because it is simple and easily understood by most people, but you can write your scripts in any language you want, as long as it is executable on a standard Ubuntu server. We wouldn't recommend Fortran 77, but it is up to you. The **set -e** line is important, as it tells the script to exit and return



» If you need help setting up Juju, check out the online docs at <https://juju.ubuntu.com/docs>.

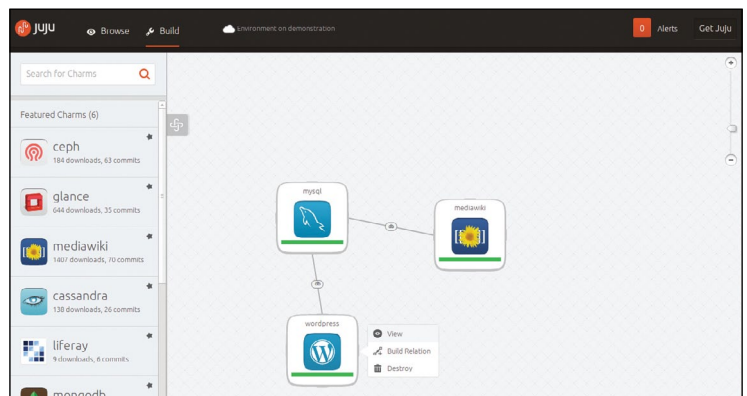
## Juju GUI

Tired of typing? You should check out the Juju GUI, a drag and drop interface for deploying charms in your cloud. The GUI runs as an instance and makes all those tedious relationships so much easier to manage. A browser on the left shows all

the available charms from the demo store, and you can drag them onto the main panel to deploy them.

Try out the live demo site to see what you are missing!

<https://jujucharms.com/sidebar>.



» The Juju GUI makes deploying cloud services look a bit too easy. Shhhh.

an error code if any of the subsequent commands fail – if something goes wrong, we want Juju to know about it. The vanilla forum runs from an *Apache* web server (in our case – you could use something else if you wanted). To start this service then, all we need to do is start *Apache*. But what if *Apache* is already running? Then we should just restart it – the restart option will start *Apache* if it is stopped, or restart it if it is running. This means our script is 'idempotent' (see *the Idempotency! box top of p82*), which means we can run it several times without changing the desired result.

The stop script is similar and simply uses a slightly different command to stop *Apache*. The real tricky one is the install script – this has to download and install everything we need to run *Vanilla*, and copy the files into the correct directory. We'll break this down to explain it as we go along...

```
#!/bin/bash
set -e
apt-get install -y apache2 php5-cgi php5-mysql curl php5-gd
wget libapache2-mod-php5
```

Yes! we can use **apt-get** to install any dependencies (remember to use the **-y** switch to disable any interaction). In this case we want *Apache*, **wget** (for fetching our *Vanilla* source) and various PHP bits we need to make *Vanilla* work properly. Your requirements may be different of course – base it on the dependencies required by your software on a plain Ubuntu server install.

[dl="https://github.com/vanillaforums/Garden/archive/](https://github.com/vanillaforums/Garden/archive/)

» If you missed last issue Call 0844 848 2852 or +44 1604 251045

## Idempotency!

Idempotency is a word you will hear a lot with regard to charm hooks (and maybe other places depending on the company you keep). It means that re-running the operation shouldn't change the intended end result. So if the operation is to start a service, it should end up started, no

matter what state it was in to begin with. It can be tricky to think of all the scenarios where relationship hooks apply, but just bear in mind that you shouldn't make assumptions about the state of services when triggering a script. This helps charms be more robust.

```

> Vanilla_2.0.18.8.tar.gz"
# Grab Vanilla from upstream.
juju-log "Fetching $dl"
wget "$dl" -O /tmp/vanilla.tar.gz

```

This part fetches the tarball from the website – it is a good idea to fetch a specific file – one you know will work!

```

# check if installed
if [ -f /var/www/vanilla/conf/config.php ]; then
cp /var/www/vanilla/conf/config.php /tmp/
rm -rf /var/www/vanilla
fi

```

What if *Vanilla* is already installed? In that case we should preserve the configuration, which is what the above does. Now we can extract the files:

```

# Extract to a known location
juju-log "Extracting Vanilla"
tar -xvzf /tmp/vanilla.tar.gz -C /var/www/
mv /var/www/Garden-Vanilla* /var/www/vanilla
if [ -f /tmp/config.php ]; then
mv /tmp/config.php /var/www/vanilla/conf/
fi

```

We should also change the permissions on relevant files:

```

chmod -R 777 /var/www/vanilla/conf /var/www/vanilla/
uploads /var/www/vanilla/cache

```

The next step is to configure *Apache* to know about *Vanilla* and edit its configuration to include an entry for it. This can easily be done by redirecting the output from **cat** to the relevant file:

```

juju-log "Creating apache2 configuration"

```

```

cat <<EOF $gt; /etc/apache2/sites-available/vanilla
<VirtualHost *:80>
ServerAdmin webmaster@localhost
DocumentRoot /var/www/vanilla
<Directory /var/www/vanilla>
Options Indexes FollowSymLinks MultiViews
AllowOverride All
Order allow,deny
allow from all
</Directory>
ErrorLog ${APACHE_LOG_DIR}/vanilla.log
LogLevel warn
CustomLog ${APACHE_LOG_DIR}/access.log combined
</VirtualHost>
EOF
a2dissite 000-default
a2ensite vanilla
service apache2 reload
juju-log "Files extracted, waiting for other events before we
do anything else!"

```

The last bits at the end there make *Apache* aware of the website and reload it so it gets the configuration changes. You may notice some **juju-log** commands. These are used to send log messages to the **/var/juju/** logfiles, and can help a lot with debugging your scripts.

## Relationships

Relationships can be tough, but the hooks relating to them are quite straightforward. The first thing to know about this type of hook is that they take their name from the relationships mentioned in the metadata. So, for example, we will need some event hooks to deal with

```

<relation name>-relation-joined
<relation name>-relation-changed
<relation name>-relation-departed
<relation name>-relation-broken

```

Some of these events are self-evident. The **relation-joined** is called when a relation is established. As this also means the relationship has changed, the **relation-changed** hook runs right afterwards. A **departed** state happens when a service disconnects, and a **broken** state when a relationship that was established is no longer available, to enable any cleanup.

So, we need a set of scripts for our database service to deal with these events. However, each time an event happens, if Juju doesn't find a script to run by the appropriate name, it just assumes everything is fine. This saves us from writing dummy scripts.

For example, there is no need to write a **database-relation-joined** script if we are going to write a **relation-changed** script which can handle the initial connection too.

```

#!/bin/bash

set -e # If any command fails, stop execution of the hook
with that error
db_user=`relation-get user`
db_db=`relation-get database`
db_pass=`relation-get password`
db_host=`relation-get private-address`
if [ -z "$db_db" ]; then
juju-log "No database information sent yet. Silently exiting"
exit 0
fi

```

➤ Browse the hook code of other charms online and check out their **Readme files** for tips on how to connect to them.

➤ **Never miss another issue** Subscribe to the #1 source for Linux on page 32.



## Local provider with LXC

You don't need to have a cloud account with HP or Amazon to mess around with charms. There is currently experimental support for LXC (Linux containers), which means you can run a cloud virtually on local instances – it saves time and money and is great for when you are developing charms. You can find the latest instructions on configuring Linux Containers at <http://juju.ubuntu.com/docs/config-LXC.html>.

```
vanilla_config="/var/www/vanilla/conf/config.php"
cat < $vanilla_config
<?php if (!defined('APPLICATION')) exit();
\${Configuration['Database']]['Host'] = '$db_host';
\${Configuration['Database']]['Name'] = '$db_db';
\${Configuration['Database']]['User'] = '$db_user';
\${Configuration['Database']]['Password'] = '$db_pass';
EOF
open-port 80
```

The **relation-get** command is a helper function for Juju, which can be used to communicate with the Juju scripts running on other instances. In this case, invoking the command:

```
juju add-relation vanilla mysql
```

will run the database-relation-joined script in our charm, but it will also run the corresponding script in the *MySQL* charm. From examining that script we can see that when something makes a connection with the *MySQL* service the script generates a database and a user with the relevant permissions. The helper function can fetch these generated values (they are random for obvious security reasons) for us, and then our script can write them into the *Vanilla* configuration file. Any time this connection is changed, the values can be fetched again (such as when the original database is replaced by a different unit) so that communication can be maintained. As we need a database, there is not much point creating the other scripts – the service will simply not work without this connection.

## Making it work

Conversely, we need to create a script to handle connections to the service we provide. So, for the file **website-relation-joined** we will have:

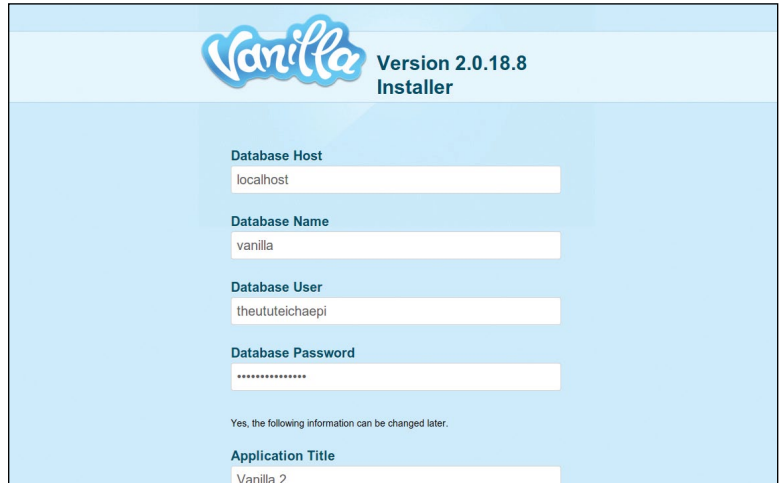
```
#!/bin/sh
relation-set hostname=`unit-get private-address` port=80
```

The **relation-set** is the corollary to **relation-get**, it exposes key=name values, in this case **hostname** and **port**. The backticked command uses another helper function, **unit-get**, to retrieve a local value (the address of the server).

You might think that this service, which is just a website after all, can operate and make connections outside of Juju – that is sort of the point of a forum. That is true, but never neglect the fact that other charms may want or need to consume your service – in this case there could be a caching proxy somewhere in your cloud for example.

Those are the basics we need to make our service work. Assuming you have installed and configured Juju, all you need now is to bootstrap an environment and deploy your charm. Oh, and *MySQL*!

```
juju bootstrap
juju deploy mysql
juju deploy --repository=~/.localcharms local:precise/vanilla
```



› The *Vanilla* startup page should appear – well done on deploying your very own charm!

```
juju add-relation mysql vanilla
juju expose vanilla
```

The special deploy line for our charm tells Juju to look in a local directory – we have also specified the 'series', in this case Precise. The **expose** command tells Juju that this is a public service and so it should make any firewall adjustments required for people to connect. But where is it? If you wait a few minutes (it takes a little time to install all that software!) you can run:

```
juju status
```

this will list all the machines and services that are running, and crucially, will also list their ports, something like this:

```
units:
  vanilla/0:
    agent-state: started
    agent-version: 1
    machine: "2"
    public-address: ec2-54-224-220-210.compute-1.
amazonaws.com
```

Then you can simply copy and paste the **public-address** into your web browser to get to the service.

## Finishing touches

If you want to distribute your charm, there are a few more things you should provide with your charm: a Readme is a must, as is some sort of copyright file. You may also provide an icon! This is used for the Juju Charm Store (an online browseable repository of available charms) and also in the GUI (see *Juju GUI*, p81).

Charms are not simply intended for deploying services, but also for managing them – particularly at scale. To go a bit further, you may want to think about in which ways your service might be made highly available. We have shown you how to make a basic, workable charm, but there are so many more things you can do. Check out the online docs for Juju at <http://juju.ubuntu.com>. 

## When things go wrong

Your Juju charms are unbelievably unlikely to work first time you set them up. To help diagnose when things go wrong, you can SSH in to the machine the charm is being deployed on, by using the machine name given from **juju status**

command. Juju will automatically add your SSH key to each deployed machine, so you just do:

```
juju ssh <machine name>
```

And then you can browse the logs or do whatever else you need to.