
THE UNIVERSITY OF
WESTERN
AUSTRALIA
Topic 15: Deploying the Application
CITS3403/5505 Agile Web Development
The Flask Mega-Tutorial
Miguel Grinberg
Chapters 17 and 18
Semester 1, 2022

Deploying the project

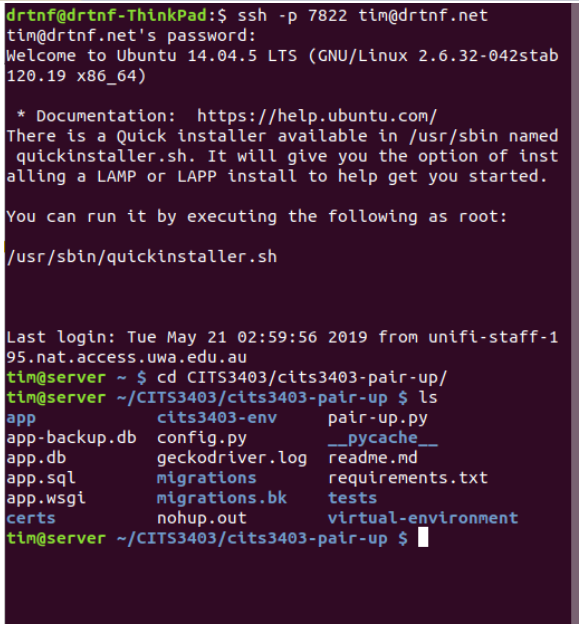
- In this unit we have gone through the process of building a web application, and deploying it via the computers local host.
- However, web applications work best on the web, so in this lecture we'll go through some options for deploying the application via a url accessible world wide.
- We will consider deploying via a linux virtual machine, your own server, or Heroku.

HEROKU

A linux virtual machine

THE UNIVERSITY OF WESTERN AUSTRALIA

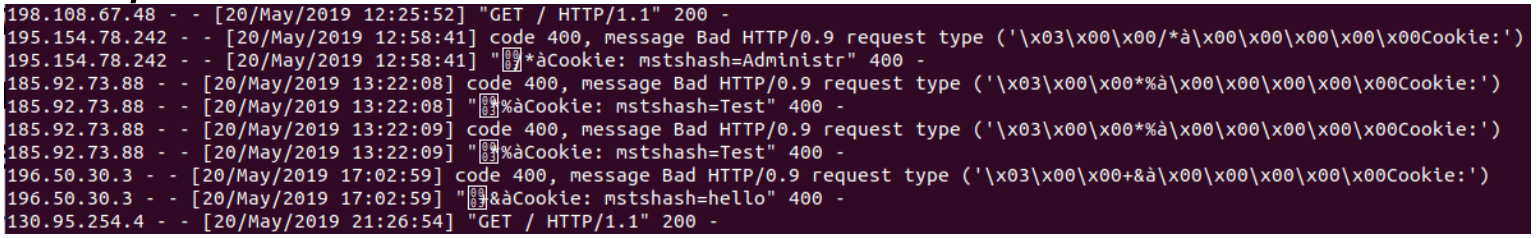
- A linux server is the traditional way of deploying a web application.
- The server runs applications listening to ports for requests and serves those requests.
- As most people don't want to worry about the physical infrastructure they use hosting solutions.
- Amazon, Digital Ocean, A2, Azure, AliBaba all offer hosted virtual machines that you can rent for about $5 a month.
- Typically, you register an account and request an instance. You have a user account with login details, that allows you to ssh to the virtual machine and configure and deploy your application from the command line.



A2 HOSTING
OUR SPEED, YOUR SUCCESS

Securing a web-server

- It's sensible to be reasonably paranoid when using a publicly accessible machine.
- Steps taken to secure to server are:
  - Removing passwords for login, and use key files instead. *You can generate a public-private key pairing that is stored in your personal machine.*
  - Disabling root logins. *Someone with root user credentials will have complete access to your server.*
  - Using a firewall to only accept traffic on ports 22 (ssh) 80 (http) and 443 (https). *People will scan open ports for any vulnerabilities.*
  - Routing all web requests through https. *Http traffic is transmitted in plaintext, and is visible to intermediate nodes.*



Production grade tools

- Flask uses it's own web-server and sqlite to allow fast and simple development.
- However these don't scale well either in terms of security, or handling many requests.
- As we have been using SQL-Alchemy, we will look at the databases mysql or PostgreSQL.
- We will use gunicorn as a web server to run the flask app, and nginx as the outward facing proxy server (Apache is an alternative to nginx).

Running mysql on the web server.

- We can install mysql on our web server using apt-get (a linux package manager).
- We open mysql and create a special user to handle the database transactions.
- Set the username to your app name, and insert an appropriate password.
- We need to install a driver for mysql, and then set the DATABASE_URL to the new database.
- As Flask automatically reads the DATABASE_URL variable we can simply run flask db migrate and flask db upgrade to create the database.
- Now the app will work as before, but we now have a full database server.

```
$ sudo apt-get -y update
$ sudo apt-get -y install python3 python3-venv python3-dev
$ sudo apt-get -y install mysql-server postfix supervisor nginx git
```

```
$ mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 6
Server version: 5.7.19-0ubuntu0.16.04.1 (Ubuntu)
```

```
mysql> create user 'pair-up'@'localhost' identified by 'top_secret';
Query OK, 0 rows affected (0.01 sec)

mysql> grant all privileges on pairup.* to 'pair-up'@'localhost';
Query OK, 0 rows affected (0.00 sec)

mysql> flush privileges;
Query OK, 0 rows affected (0.00 sec)

mysql> quit;
Bye
```

```
(cits3403-env) tim@server ~/CITS3403/cits3403-pair-up $ pip install pymysql
DEPRECATION: Python 3.4 support has been deprecated. pip 19.1 will be the la
it. Please upgrade your Python as Python 3.4 won't be maintained after March
.
Requirement already satisfied: pymysql in ./cits3403-env/lib/python3.4/site-
You are using pip version 19.0.3, however version 19.1.1 is available.
You should consider upgrading via the 'pip install --upgrade pip' command.
(cits3403-env) tim@server ~/CITS3403/cits3403-pair-up $
```

```
(cits3403-env) tim@server ~/CITS3403/cits3403-pair-up $ export DATABASE_URL="mysql+pymysql://pair-up:top_secret@localhost:3306/pairup"
(cits3403-env) tim@server ~/CITS3403/cits3403-pair-up $ flask db migrate
INFO  [alembic.runtime.migration] Context impl MySQLImpl.
INFO  [alembic.runtime.migration] Will assume non-transactional DDL.
INFO  [alembic.env] No changes in schema detected.
(cits3403-env) tim@server ~/CITS3403/cits3403-pair-up $ flask db upgrade
INFO  [alembic.runtime.migration] Context impl MySQLImpl.
INFO  [alembic.runtime.migration] Will assume non-transactional DDL.
(cits3403-env) tim@server ~/CITS3403/cits3403-pair-up $
```

A better webserver.

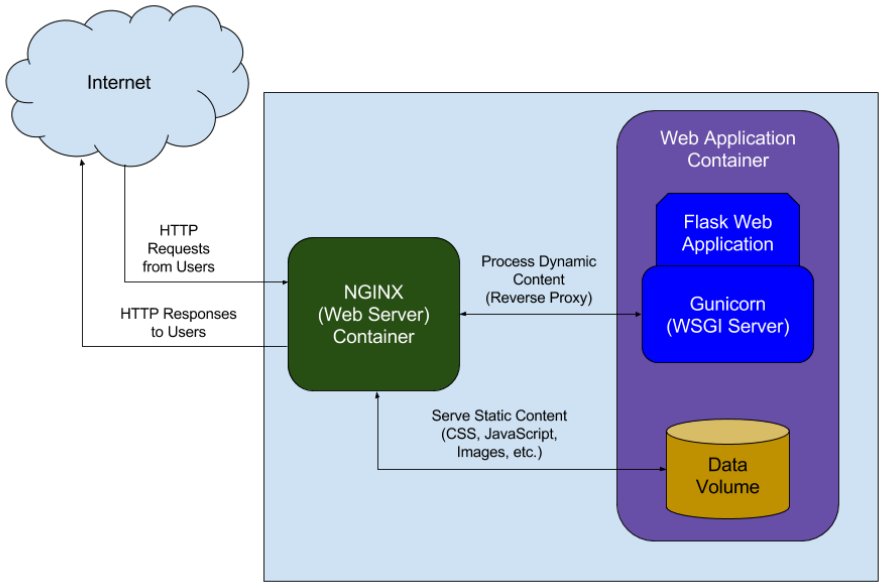
- The Flask webserver is a lightweight server used for rapid prototyping.
- Web Service Gateway Interface is a python standard for accessing web-requests, and is implemented in Gunicorn (*install with pip)*
- For dealing with external traffic and serving static content, we will use nginx as a proxy server (*install with apt-get)*
- We will also use a supervisor to restart these services when the server restarts *(install with apt-get)*

```
(venv) $ gunicorn -b localhost:8000 -w 4 microblog:app
```

/etc/supervisor/conf.d/microblog.conf: Supervisor configuration.

```
[program:microblog]
command=/home/ubuntu/microblog/venv/bin/gunicorn -b localhost:8000 -w 4 microblog:app
directory=/home/ubuntu/microblog
user=ubuntu
autostart=true
autorestart=true
stopasgroup=true
killasgroup=true
```

```
$ sudo supervisorctl reload
```



Configuring NGINX

THE UNIVERSITY OF WESTERN AUSTRALIA

- NGINX is the external facing web server. It does several things:
  - It routes all traffic through https (port 443) so it is encrypted.
  - It caches any static data served, to improve efficiency.
  - It includes public-key encryption, using a secure certificate.
  - Certificates can be generated locally, or you can get an externally signed one from LetsEncrypt.org.
  - This requires a domain name for your server.

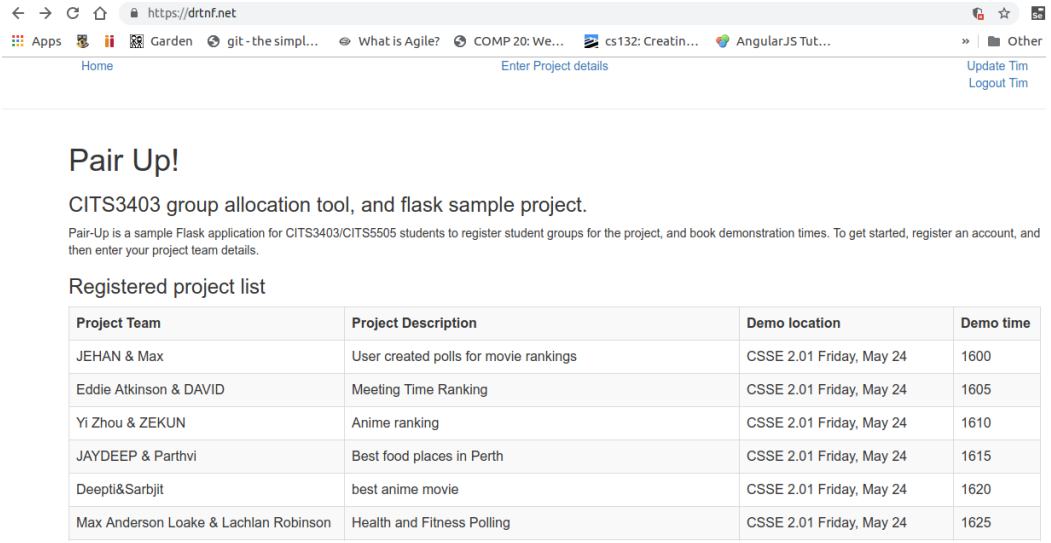
/etc/nginx/sites-enabled/microblog: Nginx configuration.

```
server {
    # listen on port 80 (http)
    listen 80;
    server_name _;
    location / {
        # redirect any requests to the same URL but on https
        return 301 https://$host$request_uri;
    }
}
server {
    # listen on port 443 (https)
    listen 443 ssl;
    server_name _;

    # location of the self-signed SSL certificate
    ssl_certificate /home/ubuntu/microblog/certs/cert.pem;
    ssl_certificate_key /home/ubuntu/microblog/certs/key.pem;

    # write access and error logs to /var/log
    access_log /var/log/microblog_access.log;
    error_log /var/log/microblog_error.log;

    location / {
        # forward application requests to the gunicorn server
        proxy_pass http://localhost:8000;
        proxy_redirect off;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    }

    location /static {
        # handle static files directly, without forwarding to the application
        alias /home/ubuntu/microblog/app/static;
        expires 30d;
    }
}
```

```
$ sudo service nginx reload
```

Deploying the website

- When we deploy the website, with the command `sudo nginx reload` (or `sudo nginx start`) the server will listen for requests on port 80 and forward them through port 443 so they are encrypted end to end.
- This uses the public key registered with the certificate authority, and the private key to decrypt traffic. The client uses the public key to encrypt traffic, but has to have confidence in the identity of the origin of the public key, which is why a 3rd party is required.
- The servers run in a daemon thread so they persist after the session has ended (i.e. you log out).

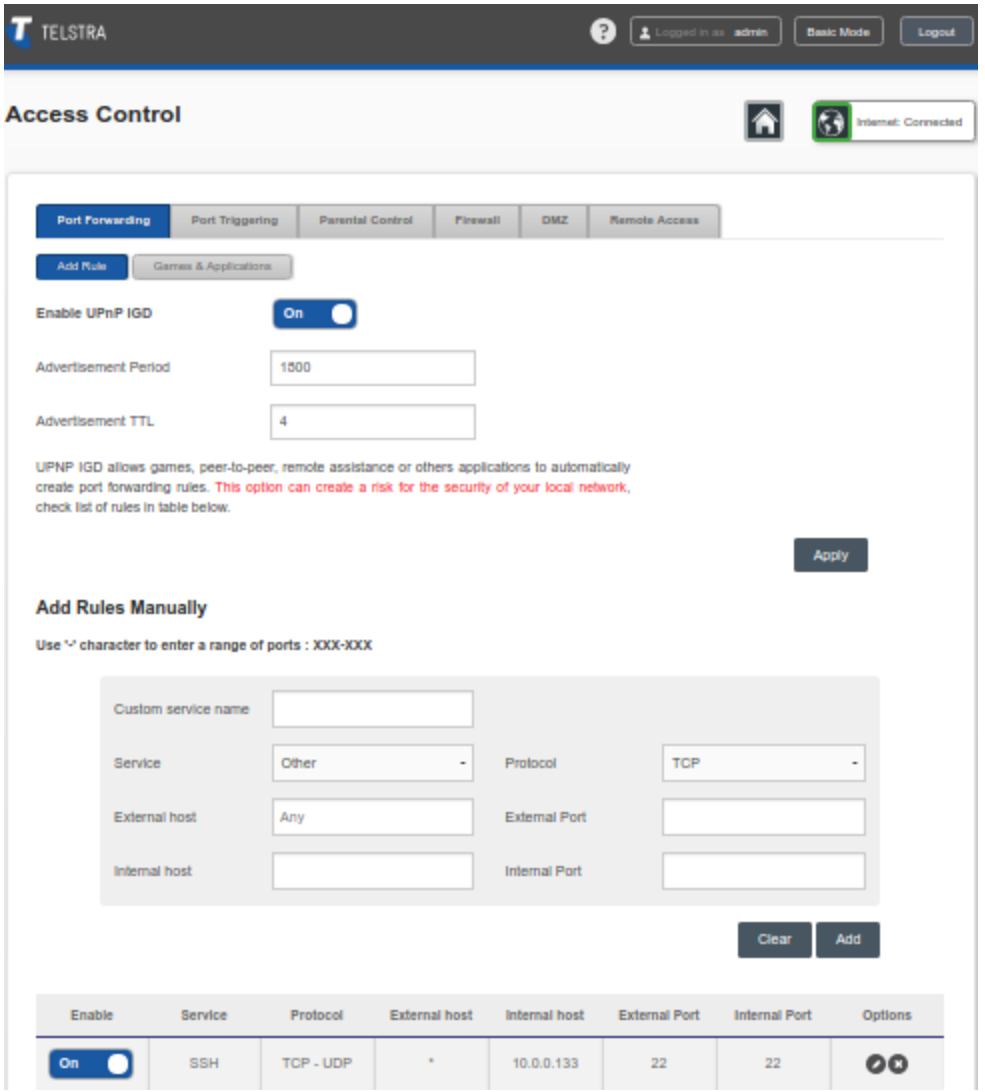


Running your own server

- In the past running your own server would have been an expensive proposition.
- It is now much cheaper, using a wireless router and home broadband connection and raspberry pi, or similar small computer.
- A raspberry pi is a single board computer costing less than $60, with 1GB Memory and 1.4 Ghz processor. This is easily enough to power a small web server.
- Raspberry Pi's come with a variation of linux (Raspbian) and can be configured in the same way as the linux servers we have discussed.
- You normally need a keyboard and monitor to configure the raspberry pi, but once you have it on your home network, you can use ssh and treat it like any normal linux server.

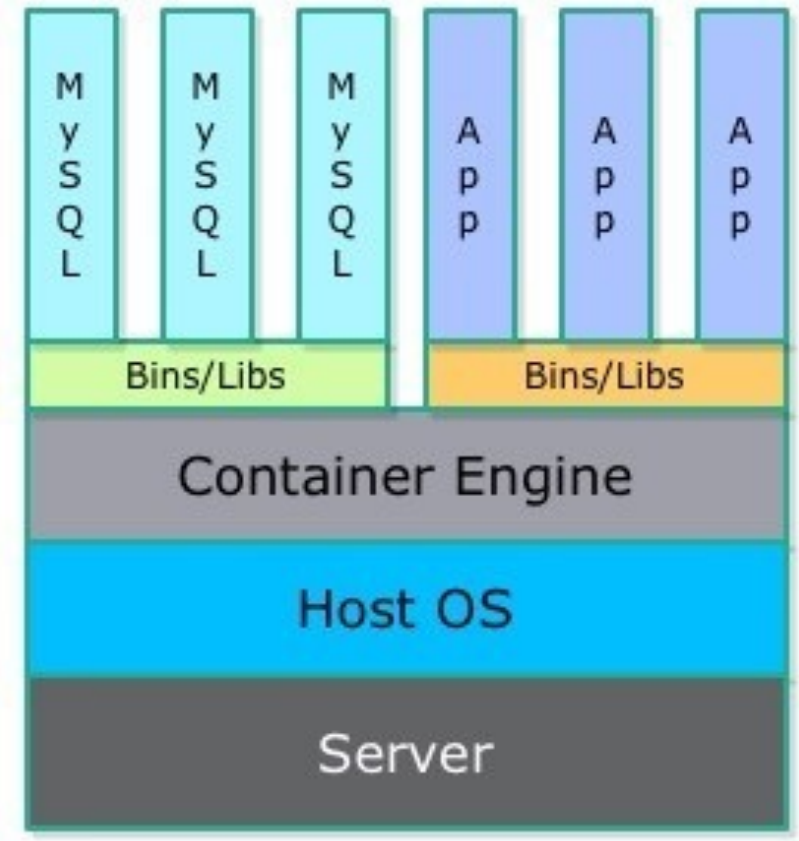
Deploying to raspbery pi

- You can connect a raspberry pi (or any computer) to a wireless router, and then acces the administrator interface of the router.
- Using that you can open ports to the computer (80, 23 and 443 are usually enough), and then access the application via the ip address of the router.



Containerising your web app

- When creating software such as a web app, there can be issues with versioning, package management and cross-platform reliability
- Solution: put your web app in it's own little transportable box that stays the same no matter what
- This is basically a **container**
- Containers standardise the software you've written, along with all of the dependencies, so it runs the same from one environment to another



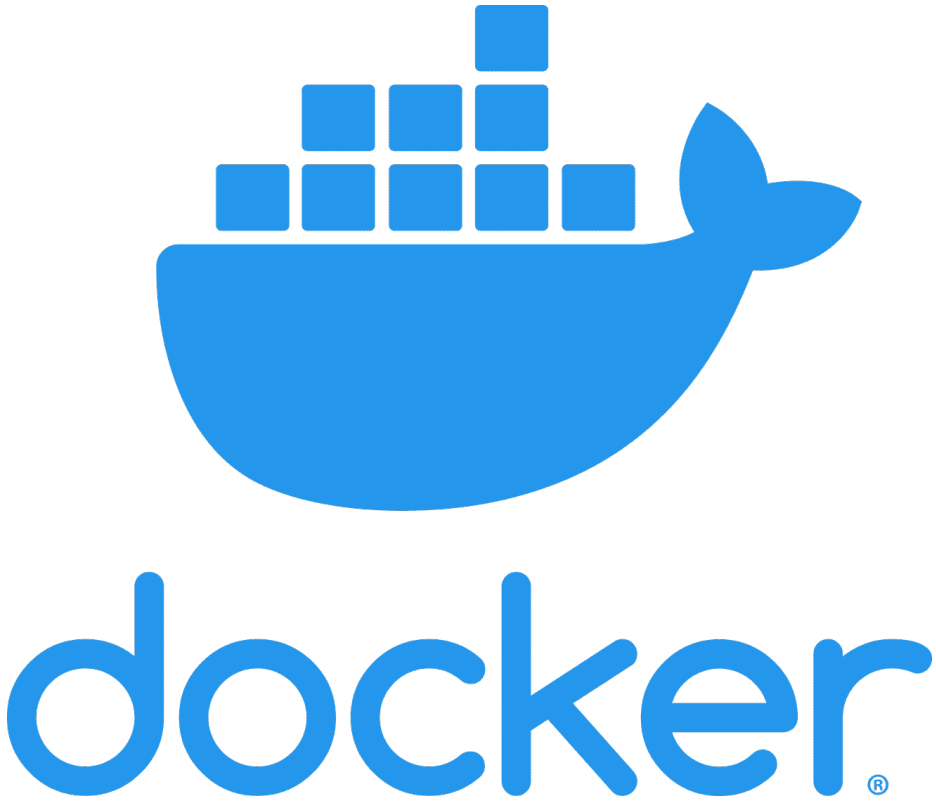
An example of a container stack: anything above is bundled into the container, to be run on the Host OS and deployed on the server [1]

[1] https://www.msystechnologies.com/blog/a-complete-guide-to-cloud-containers/

Docker: Easy way to create containers

THE UNIVERSITY OF WESTERN AUSTRALIA

- Docker made your favourite containers favourite container
- Docker software includes functionality to create Docker container images, as well as the Docker Engine as a run environment
- These container images are run on the Docker Engine, and no matter what the environment is the app will be the same
- Docker has different tiers of containers depending on your needs (security, space etc.)



Dockerfiles

- Docker (as a program) can create an image automatically from a script called a Dockerfile
- Each project can contain multiple files (depending on environments such as development and production), and these files are basically a script to create the container
- `docker build' can run this script, and the container will be created
- For further information on Dockerfile scripts and best practices, go here

Dockerfile > ...

```
1   # Get the base image
2   FROM python:3.10.3-slim-buster
3
4   # Set the working directory
5   WORKDIR /usr/src/app
6
7   # Set any relevant environment variables
8   ENV PYTHONDONTWRITEBYTECODE 1
9   ENV PYTHONUNBUFFERED 1
10
11  # Add any system dependencies
12  RUN apt-get update \
13      && apt-get -y install netcat gcc postgresql \
14      && apt-get clean
15
16  # Install requirements from file
17  COPY ./requirements.txt .
18  RUN pip install -r requirements.txt
19
20  # Add the Flask app
21  COPY . .
22
23  # Set the entrypoint with a script
24  COPY ./entrypoint.sh .
25  RUN chmod +x /usr/src/app/entrypoint.sh
```

Dockerfile from a deployed Flask project

Hosting web apps on the world wide web

- When deploying apps (especially in less-mature / early stage projects), it's easiest at time of writing to use the cloud
- There exist many Platform-as-a-Service (PaaS) products, largely doing away with having to set up a server, manage storage and maintain infrastructure
- There are almost always pay to play and scale with playtime, but almost all have free or hobby tiers
- They also often are limited to different OS / languages

Fly.io

There are many services available: the choice is an individual one, but we've been using Heroku the longest so we'll be using that for demonstration. At time of writing HN hates Heroku and espouses Fly.io

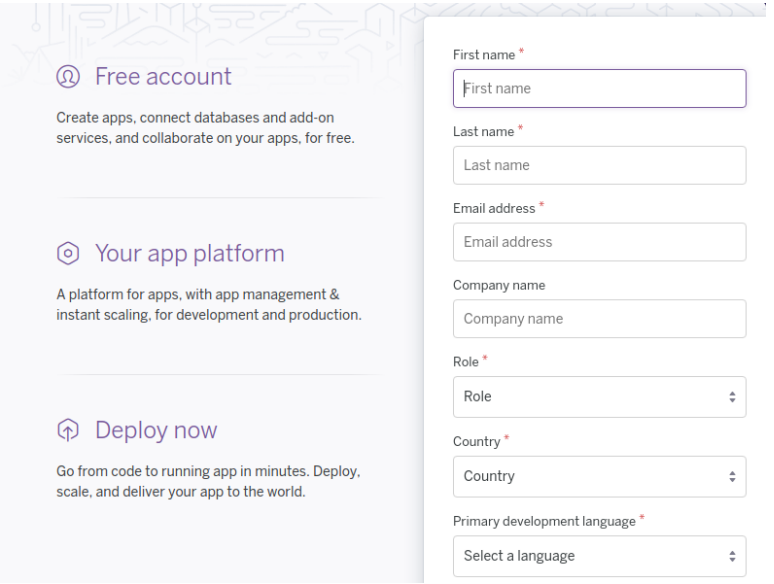
Heroku: A heroic haiku owned by Salesforce

THE UNIVERSITY OF WESTERN AUSTRALIA

- Heroku is a cloud platform that you can upload your (ideally containerised) app to
- You can then use built-in functionality to build, maintain and scale your project
- Popular as very easy to use: you basically commit and push your project to the host URL like you would a Git repo
- Free tier (hobby dev), but can get expensive quite quickly
- Still probably the most popular / recognised even though reputation for being expensive

Setting up Heroku

- To launch your app on Heroku, you need a Heroku account (free), and a git repository of your project.
- Once you register an account you should install Heroku CLI, a command line interface that lets you set up and configure your heroku instance from your local machine.
- Build your app as usual, in a git repo.
- Heroku does not have persistant memory, but offers free postgress databases as a service, so we need to include psycopg2, and gunicorn.
- Freeze the requirements and then create the app. This initialises a git remote to push our code to.




```
$ heroku apps:create flask-microblog
Creating flask-microblog... done
http://flask-microblog.herokuapp.com/ | https://git.heroku.com/flask-microblog.git
```

Database as a service

THE UNIVERSITY OF WESTERN AUSTRALIA

- As the Heroku container does not offer persistant memory, we require an external database.
- We can use Heroku addons to add a postgresql database. If we initialise it as hobby-dev, it is free.
- When we request a database for our project, Heroku initialises it, and sets DATABASE_URL in our project settings to point to it, so we don’t have to do anything expect migrate our database structure.
- There are many other Database As A Service providers we could use: MLab for mongo databases, Azure and AWS offer every type of database as a service.

```
$ heroku addons:add heroku-postgresql:hobby-dev
Creating heroku-postgresql:hobby-dev on flask-microblog... free
Database has been created and is available
 ! This database is empty. If upgrading, you can transfer
 ! data from another database with pg:copy
Created postgresql-parallel-56076 as DATABASE_URL
Use heroku addons:docs heroku-postgresql to view documentation
```

Deploying on Heroku

- Given we went through many steps to set up the Flask environment on our local machine deploying to Heroku is surprisingly easy.
- We set any system variables we need (in this case `FLASK_APP`), and Heroku has already set others we require (`DATABASE_URL` and `PORT`)
- We give it the basic commands to run on initilaisation in a `Procfile`, that is stored in the root of our git repo.
- To launch the app we just push our git remote to the remote that was created with the project. Heroku will detect what language we are using (python), install python, pip and all our requirements from reuqirements.txt, and then run the commands in the `Procfile`.

```
$ heroku config:set FLASK_APP=microblog.py
Setting FLASK_APP and restarting flask-microblog... done, v4
FLASK_APP: microblog.py
```

Procfile: Heroku Procfile.

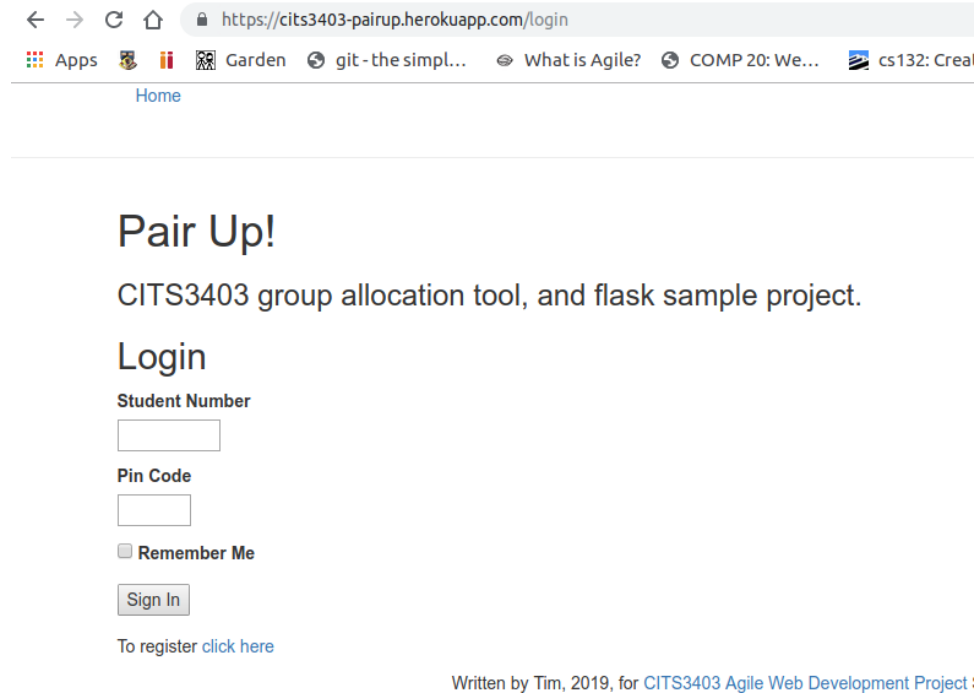
```
web: flask db upgrade; flask translate compile; gunicorn microblog:app
```

```
$ git push heroku deploy:master
Counting objects: 247, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (238/238), done.
Writing objects: 100% (247/247), 53.26 KiB | 3.80 MiB/s, done.
Total 247 (delta 136), reused 3 (delta 0)
remote: Compressing source files... done.
remote: Building source:
remote:
remote: -----> Python app detected
remote: -----> Installing python-3.6.2
remote: -----> Installing pip
remote: -----> Installing requirements with pip
...
remote:
remote: -----> Discovering process types
remote:        Procfile declares types -> web
remote:
remote: -----> Compressing...
remote:        Done: 57M
remote: -----> Launching...
remote:        Released v5
remote:        https://flask-microblog.herokuapp.com/ deployed to Heroku
remote:
remote: Verifying deploy... done.
To https://git.heroku.com/flask-microblog.git
 * [new branch]      deploy -> master
```

Full Deployment

```
drtnf@drtnf-ThinkPad:$ heroku apps:create cits3403-pairup
Creating ⬢ cits3403-pairup... done
https://cits3403-pairup.herokuapp.com/ | https://git.heroku.com/cits3403-pairup.git
drtnf@drtnf-ThinkPad:$ heroku addons:add heroku-postgresql:hobby-dev
Creating heroku-postgresql:hobby-dev on ⬢ cits3403-pairup... free
Database has been created and is available
 ! This database is empty. If upgrading, you can transfer
 ! data from another database with pg:copy
Created postgresql-shaped-80812 as DATABASE_URL
Use heroku addons:docs heroku-postgresql to view documentation
drtnf@drtnf-ThinkPad:$ heroku config:set FLASK_APP=pair-up.py
Setting FLASK_APP and restarting ⬢ cits3403-pairup... done, v5
FLASK_APP: pair-up.py
drtnf@drtnf-ThinkPad:$ git push heroku master
Counting objects: 3376, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (3267/3267), done.
Writing objects: 100% (3376/3376), 14.57 MiB | 499.00 KiB/s, done.
Total 3376 (delta 544), reused 0 (delta 0)
remote: Compressing source files... done.
remote: Building source:
remote:
remote: -----> Python app detected
remote: -----> Installing python-3.6.8
remote: -----> Installing pip
remote: -----> Installing SQLite3
remote: -----> Installing requirements with pip
remote:        Collecting alembic==1.0.8 (from -r /tmp/build_6a52ac60a414523900bbf25c
remote:          Downloading https://files.pythonhosted.org/packages/d6/bb/ec1e21f2e3
99474c4d3535c8/alembic-1.0.8.tar.gz (1.0MB)
remote:        Collecting certifi==2019.3.9 (from -r /tmp/build_6a52ac60a414523900bbf
))
remote:          Downloading https://files.pythonhosted.org/packages/60/75/f692a584e8
1e598828d380aa/certifi-2019.3.9-py2.py3-none-any.whl (158kB)
remote:        Collecting chardet==3.0.4 (from -r /tmp/build_6a52ac60a414523900bbf25c
```

```
remote:        Successfully installed Click-7.0 Flask-1.0.2 Flask-HTTPAuth-3.2.4 Flas
ask-SQLAlchemy-2.3.2 Flask-WTF-0.14.2 Jinja2-2.10 Mako-1.0.7 MarkupSafe-1.1.1 PyMySQL
.1 WTForms-2.2.1 Werkzeug-0.14.1 alembic-1.0.8 certifi-2019.3.9 chardet-3.0.4 gunicor
ngerous-1.1.0 python-dateutil-2.8.0 python-dotenv-0.10.1 python-editor-1.0.4 requests
urllib3-1.24.3
remote:
remote: -----> Discovering process types
remote:        Procfile declares types -> web
remote:
remote: -----> Compressing...
remote:        Done: 65.5M
remote: -----> Launching...
remote:        Released v6
remote:        https://cits3403-pairup.herokuapp.com/ deployed to Heroku
remote:
remote: Verifying deploy... done.
To https://git.heroku.com/cits3403-pairup.git
 * [new branch]      master -> master
```



The future of the web

THE UNIVERSITY OF WESTERN AUSTRALIA

- In this course we have tried to focus on the fundamental technologies of the web: HTML, CSS, Javascript, and web application frameworks, RESTful architectures, AJAX, Sockets.
- We have also looked at the agile software development process, and key tools like git.
- While the core technologies such as REST, HTTP, and AJAX are reasonably persistent, the web is a rapidly changing domain, with many trends, and many new emerging technologies.
- What will the web look like in the future?

Cloud computing

THE UNIVERSITY OF WESTERN AUSTRALIA

- As with deployment on Heroku… cloud services are everywhere.
- High bandwidth permanent online devices and the principles of REST between the line between your personel device and a cloud service is increasingly blurred.
- Interestingly cloud is blurring into desktop apps, such as word processing, and the desktop SOE is disappearing.
- Many music/movie streaming services are changing the notion of libaries and owning media entirely.
- Software as a service is a huge industry and many of the emerging business models use this concept.

Want to know more?
Study CITS5505, Cloud Computing

The Internet of Things

- The internet of things is the extension of the internet to smart devices. It focuses on the message passing interfaces and infrastructure, rather than the end user interface.
- Smart homes, adaptive lighting, smart vehicles, and connected devices all use basic web technologies for remote control.
- Similarly they are able to gather analytics and adapt to their usage to improve user experience.
- But this has big implications for privacy: the web extends seamlessly to our day to day life without us being conscious of it.

Want to know more?
Study CITS5506, Internet of Things

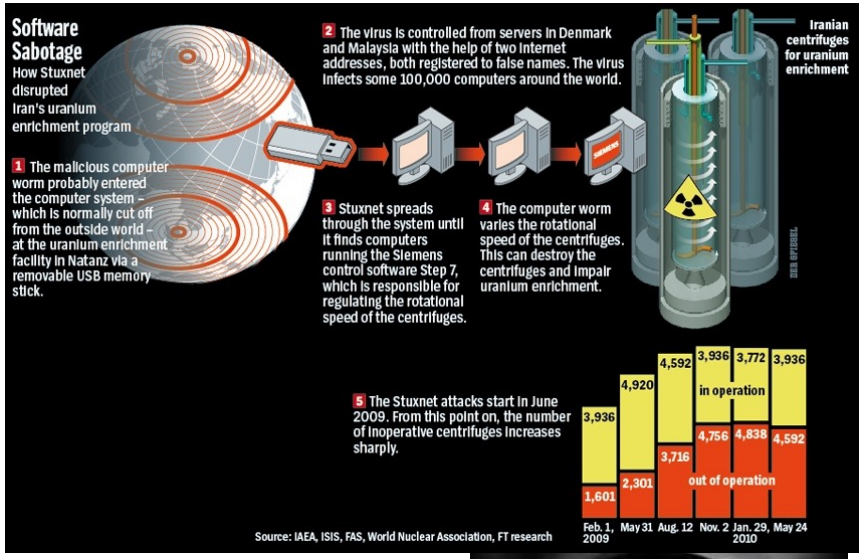
Semantic web

- Sir Tim Berners-Lee was one of the creators of the web in 1989, but has for a long time been championing the semantic web.
- This highlights the difference between content and meaning. Semantics is meaning. We use HTML and CSS to differentiate content and presentation, but if we also have a standard for meaning, then a smart search engine, can understand whether web services are selling tickets, or promoting events, or documenting history etc.
- This has big implications for artifical intelligence and automatic assistants.

Want to know more? Study: CITS3001 Agents, Algorithms and Artificial Intelligence; and CITS3005: Knowledge Representation.

Cyber-Security

- Security is the ongoing challenge of the web. The web has evolved and wasn't designed for security, so securing services is a constant battle.
- The prevelance of the web means that small flaws can have major implications in defence and society and cyberwarfare is consuming significant defence spending.
- Big examples are the Stuxnet attacks on Iran's nuclear program, the Ashley Madison hack, and Anonymous's attack on HB Gary.



Want to know more? Study:
CITS3004 Cybersecuritry;
CITS3005 Penetration Testing and
CITS3007 Secure Coding.

Final Take home test

- This Exam is worth **50%** of your final grade and must be done individually.
- The test will be released at 8am on Tuesday May 31, via LMS and the unit web page and is due at 5pm Wednesday June 1.
- Your answer should be presented as an HTML document (including raw JS and CSS) meeting a given specification, and answering specific questions.
- The content of the HTML document will be worth approximately 60%, with the format and functionality accounting for the other 40%. You maybe be required to answer questions about any aspect of the course
- Good luck!