Niall Byrne

K8's for Python Projects

(Mapping a path from greenfield to production)

Some of the perspectives I have encountered in my career …

Erik Levinson, Steve Pereira, Jason Harley, Yegor Sak, Mark Ulicki, Alex Paguis, Michael St.George, Steve van Bruwaene, Kevin Falk, Boz Lenchov, Jesse Malone, James Pollard, Aaron Hurley, Darcy Clarke, Tao Huang, Lucy Kang, Hilary Watson, Eugene Taylashev, Anthony Scinocco, The Felske Trio, Pete Myers, Conan Fan, Dave Rouse, Greg Meloche, Jonathan Mazin, Patricia Tao, Can Cetin, Eric Lazam, Andrew Belomestny, Leo Tolisano, Michael Lam, Mary Broadfoot, Alexander Zbusko, Disha Patel, Stephen Williams, Alec Brunelle, Aleks Maricic, Bob Tierney, George Mihaiescu, Cecile Leung, Solomon Shorser, Brian O'Conner, Adam Wright, Denis Yuen, Matt Dao, Horace Jew, Daniel Wilcox, Seventeen Chen, Denis McEntire, Vitalii Dolia, Nataliia Dolia, Tadek Orlowski, Ivy Wang, Van Le, Susan Harmer, Jim, Francis, Gary, and even Matt from Layer 7, Stephen Hughes, Jay Thorne, Paul Wolstenholme, Li Karaian, Bill Rabjohn, Sergiu Sefer, Dmitry Drobinin, Michael Barton, Garcia Holness, Faisal Hasan, Rudy Rankine, Nina Zhang, Clarence Espejo, Irfan Hussain, Jean-Pierre Dehenne, Bruno Courcy, Steve Blackman, Sharjeel Ahmed, Vlad Dagaev, Yurui Qiu, Marg Walinga, Mike Lively, Sateish Harlal, Mubeen Qureshi, Jaya Kassiedass, Eric Uwonkunda Ngabonziza, Tanya Wang, Marc Magloire

So what is this talk about?

In one sentence:

Kubernetes rewards conforming to it’s paradigm by enforcing good development practices- and sneakily makes your code, and architecture better- and maybe even easier to work with.

Sounds Like Baloney

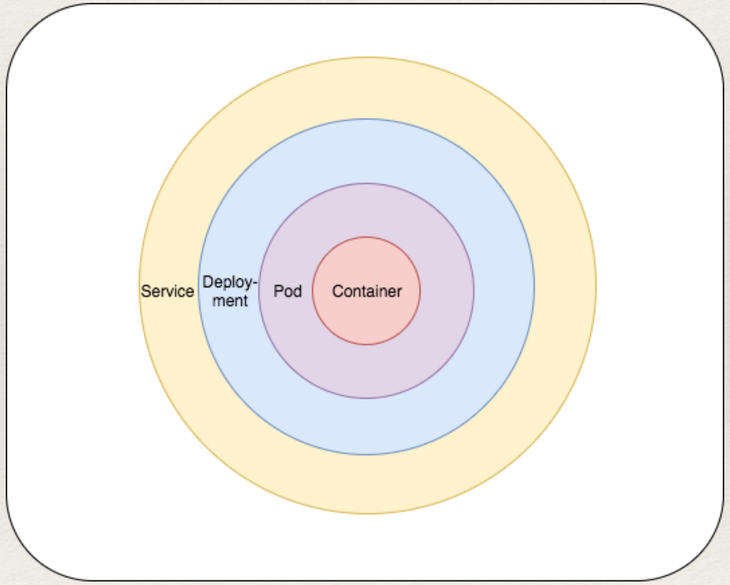
- But, just incase it's true:
  - Let's review the concepts of Containers, Pods and Deployments and Services, and look at the architecture decisions they encourage. We'll use a Django application as our model.
  - Let's look at what we can do in Development and Testing with a Kubernetes cluster, again working with a Django application.

A Django Application

- **Components**:
  - Django
  - ~~Static File Server~~
  - MySQL Database
  - Redis Server
  - Asynchronous Task Workers
  - ~~SSL Termination~~

Kubernetes Concepts

- Containers
- Pods
- Deployments
- Services



First: We Need Dockerfiles

```
FROM ubuntu:16.04

MAINTANER Your Name "youremail@domain.tld"

RUN apt-get update -y && \
    apt-get install -y python-pip python-dev

# We copy just the requirements.txt first to leverage Docker cache
COPY ./requirements.txt /app/requirements.txt

WORKDIR /app

RUN pip install -r requirements.txt

COPY . /app

ENTRYPOINT [ "python" ]

CMD [ "app.py" ]
```

- Docker Builds these Dockerfiles into **IMAGES**, which are somewhat similar to a class object.
- The **Image** can be used to spawn **Containers**, with are like instances of a class.
- Individual **Container** Instances can be customized with Environment Variables at Run Time (Frequently used to insert sets of secrets to differentiate test and production environments.)

Kubernetes 101 Part 1/4

- **Containers**:

Dictionary Definition:

A versioned OS level artifact that contains your application's runtime environment

Encouraged Behaviours:

Sneakily forces you to define your application's dependencies, and conform to a single process model. 12 Factor App Friendly.

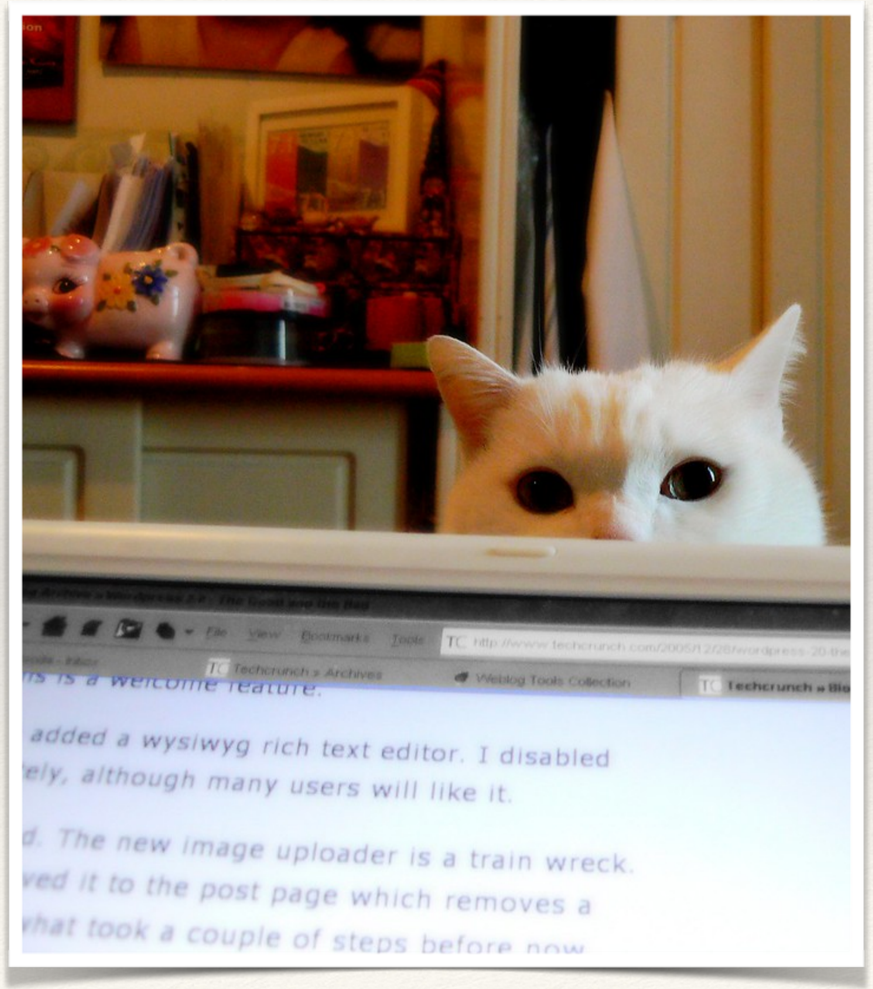
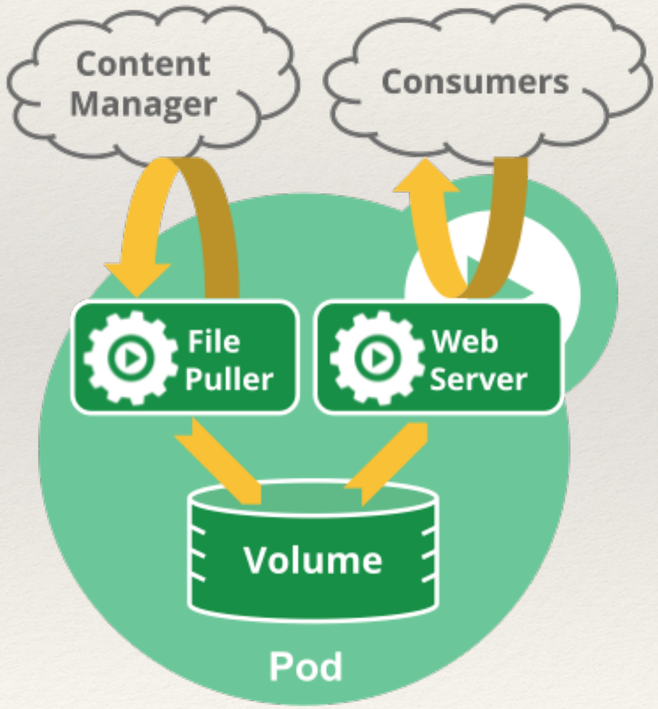
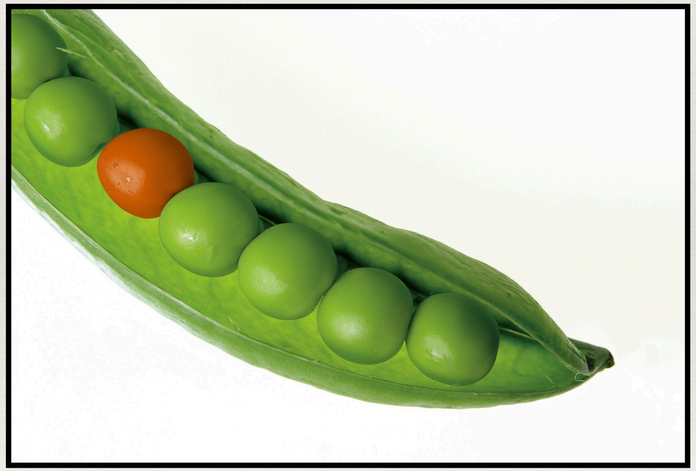


Image: r y _ _ _ _ on Flicker (Creative Commons)

Pods

Kubernetes uses the metaphor of a pod of whales, or peas in a pod to illustrate the function of the Pod Component. It's a set of containers that work together to provide a service, and as such must scale together under application load.





Kubernetes 101 Part 2/4

- **Pods**:

Dictionary Definition:

A group of containers that are always deployed together to the same K8 node. They share the same network, the same data volumes and can access each other over 'localhost' without traversing the cluster network.

Encouraged Behaviours:

Forced to consider which processes are co-dependent.

Image: Raider of Gin on Flicker (Creative Commons)

Deployments versus Pods

Kubernetes configuration is most frequently committed to source control as YAML.

```yaml
---
apiVersion: apps/v1beta1
kind: Deployment
metadata:
  name: django
spec:
  replicas: 5
  template:
    metadata:
      labels:
        app: django
    spec:
      containers:
        - name: django
          image: niallbyrne/playcounts3:latest
          ports:
            - containerPort: 8000
          command: ["gunicorn -k gevent -b 0.0.0.0 project.wsgi.application"]
          env:
            - name: DJANGO_SECRET_KEY
              value: "very_hard_to_guess_value"
```

```yaml
---
apiVersion: v1
kind: Pod
metadata:
  name: django-pod
spec:
  containers:
    - name: django-pod
      image: niallbyrne/playcounts3:latest
      ports:
        - containerPort: 8000
      command: ["gunicorn -k gevent -b 0.0.0.0 project.wsgi.application"]
      env:
        - name: DJANGO_SECRET_KEY
          value: "very_hard_to_guess_value"
```

Kubernetes 101 Part 3/4

- **Deployments**:

Dictionary Definition:

A Kubernetes Deployment, is a group of Pods that can be scaled as copies. Each Replica in the deployment will be a running set of pods that conforms to the deployment spec.

Encouraged Behaviours:

Requires us to consider how our application will need to scale, and which pieces will need to scale together

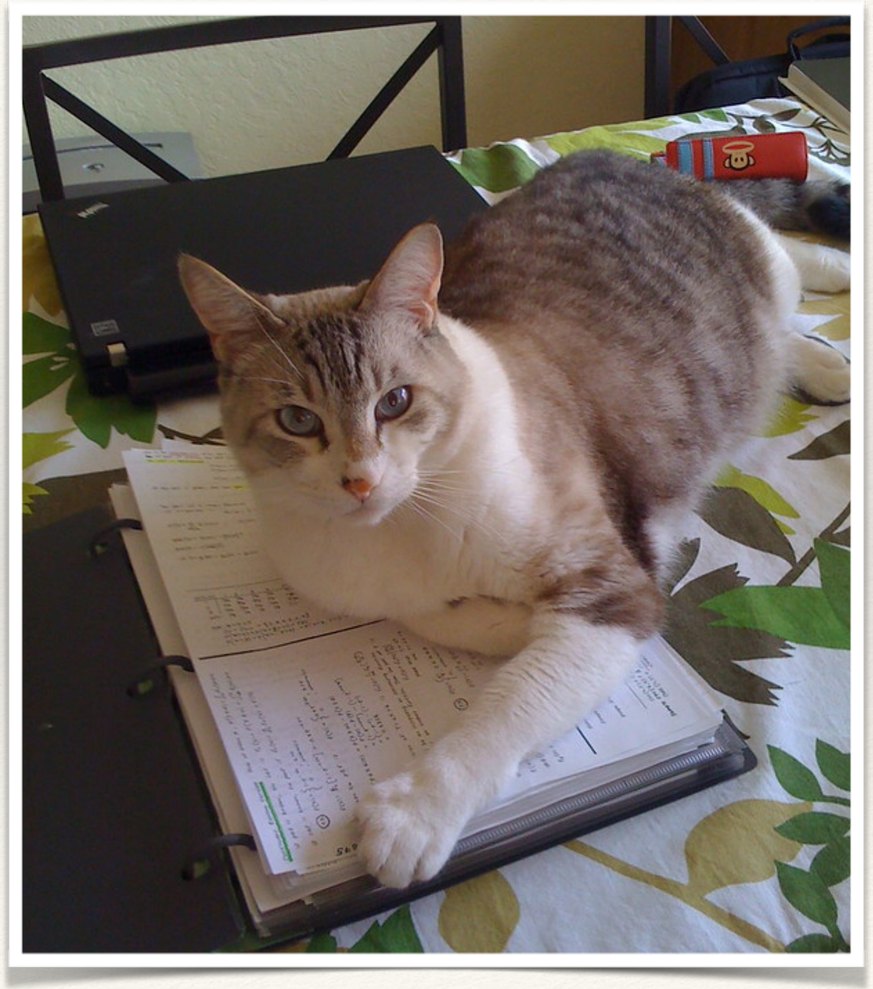
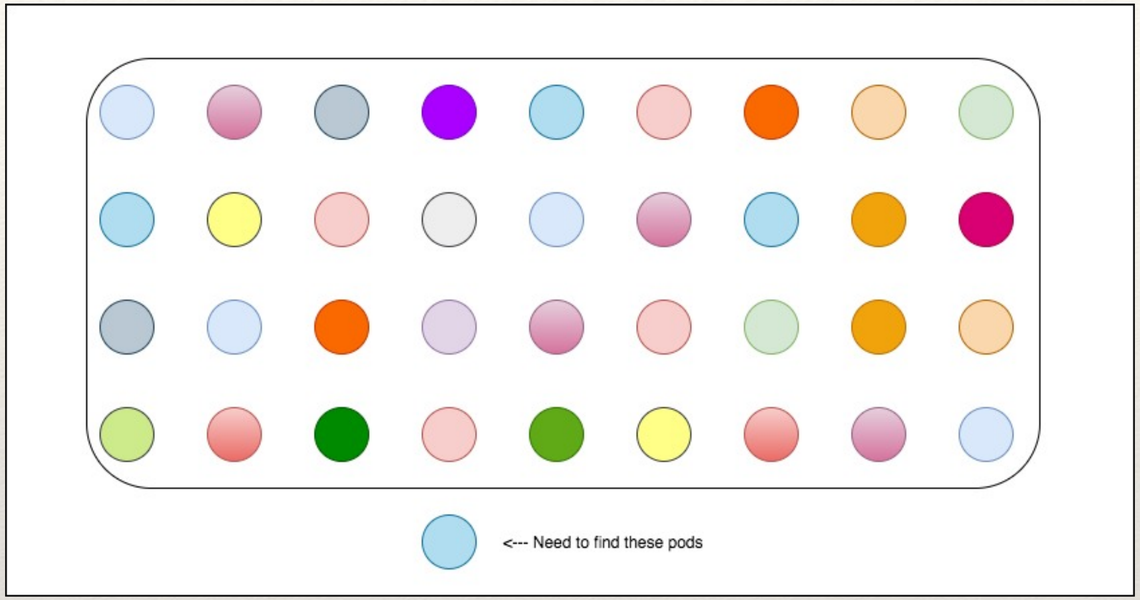
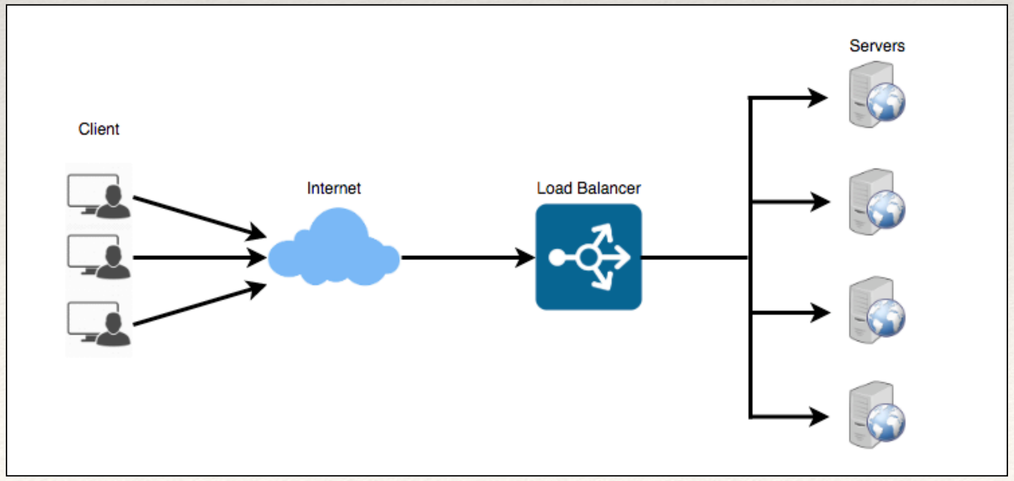


Image: Ewen Roberts on Flicker (Creative Commons)

The Problem of Service Discovery





Kubernetes 101 Part 4/4

- **Services**:

Dictionary Definition:

A K8's service is a virtual endpoint that can be used to reach one or more copies of a deployment. (Load Balancer with a Virtual IP)

Encouraged Behaviours:

We are now aware of service discovery, resiliency, and load balancing.

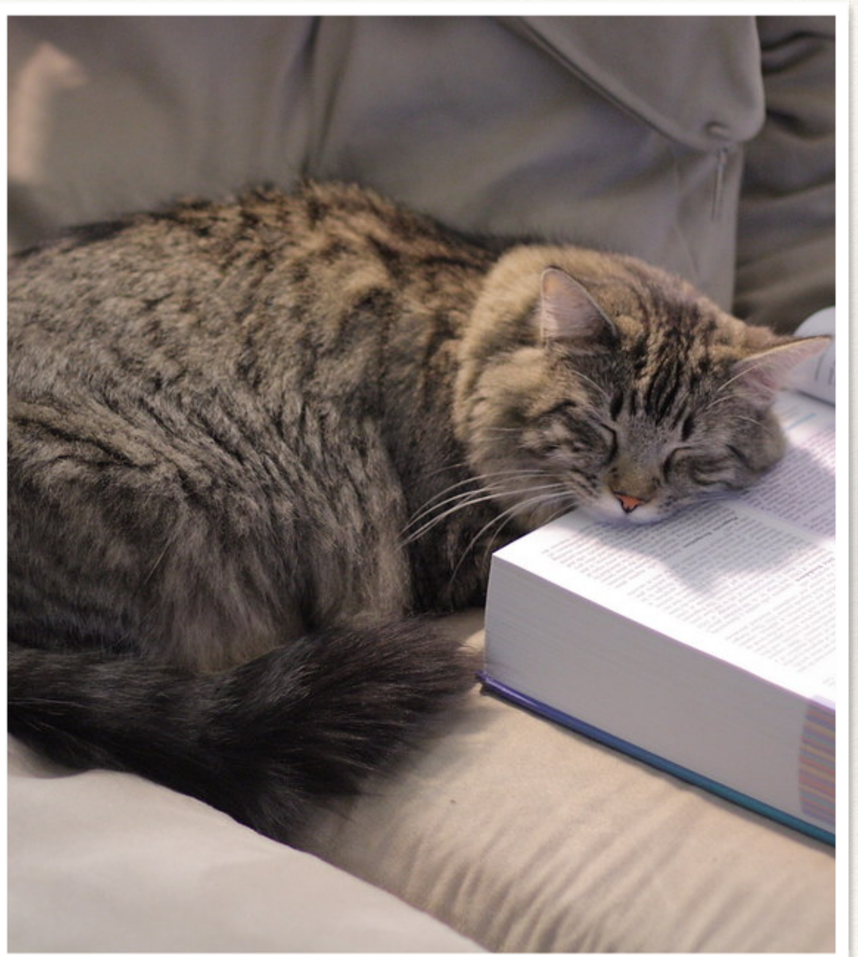
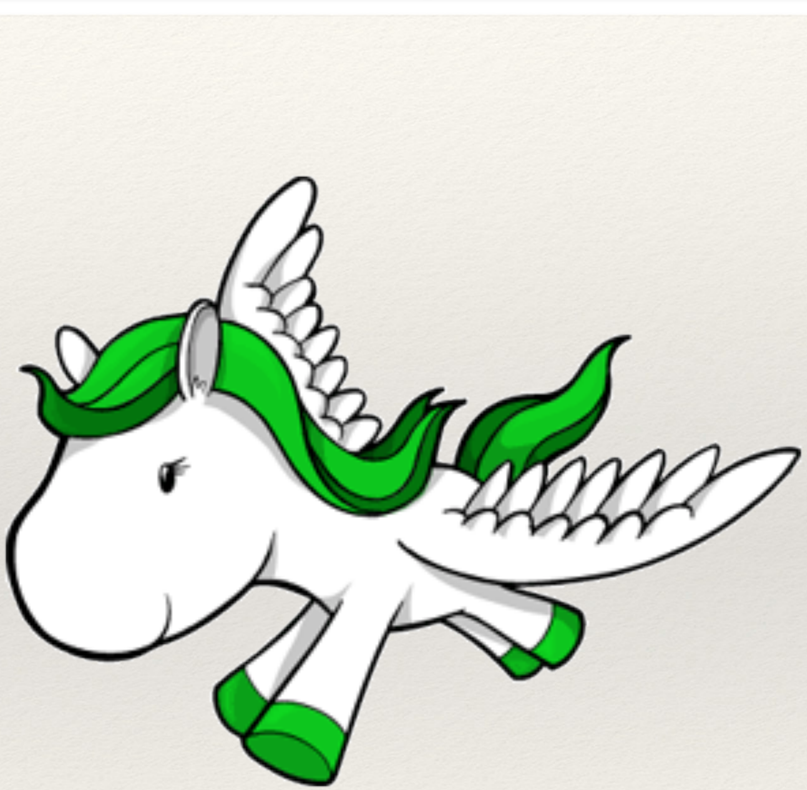


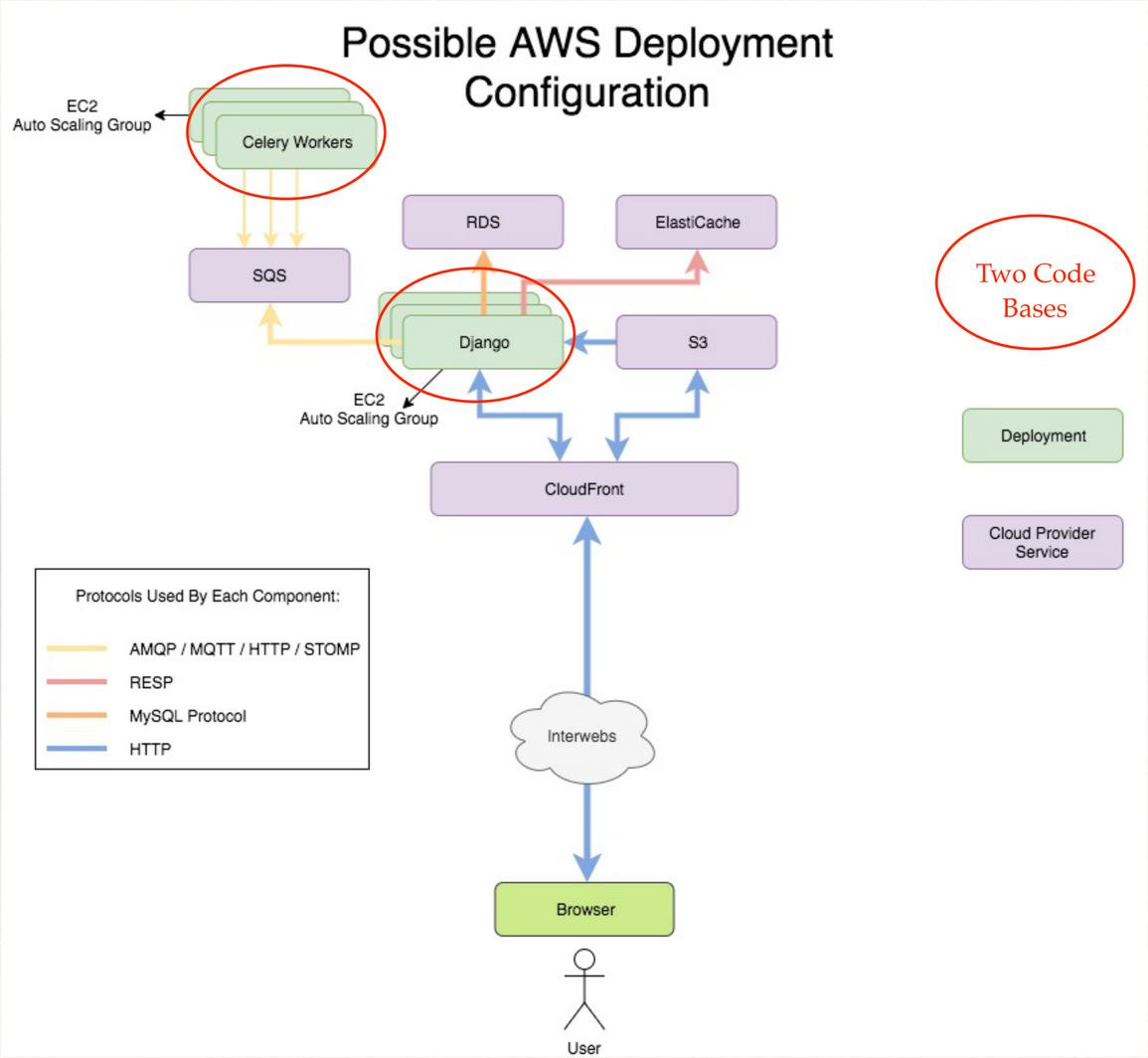
Image: Sage Ross on Flicker (Creative Commons)

Decomposing our Django Application

- **Components become services:**
  - Django: Needs to Scale!
  - ~~Static File Server~~
  - MySQL Database: Containerize
  - Redis Server: Containerize
  - Queue: RabbitMQ in Container
  - Celery Workers: Needs to Scale!
  - ~~SSL Termination~~



Production Possibilities



Wait a minute …

Quick and Easy Kubernetes Wins

- Build Pipelines: discrete build and deploy steps
- Integration Testing: you already built a QA environment

Multi Stage Builds

```
# Step 1: Container "codebase"
FROM niallbyrne/playcounts_dependencies:latest as codebase

COPY . /home/webapp/code

# Step 2: Container "unittests"
FROM codebase as unittests

RUN apk --no-cache add gcc

COPY requirements-testing.txt /home/webapp/config/requirements-testing.txt
RUN pip3 install --no-cache-dir -Ur /home/webapp/config/requirements-testing.txt

RUN ./scripts/lint.sh
RUN ./scripts/test.sh

# Step 3: Container "final", get's pushed to the repository
FROM codebase as final

RUN chown -R webapp:webapp /home/webapp

USER webapp
WORKDIR /home/webapp/code

CMD [ 'app.py' ]
```

Kubernetes = Easy Integration Testing

```
niall@Nialls-Mac-mini:~$ kubectl get services
NAME                   TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)          AGE
chrome-01-external     NodePort    10.106.159.98    <none>        5900:30005/TCP   6d2h
chrome-02-external     NodePort    10.104.54.50     <none>        5900:30006/TCP   6d2h
django                 ClusterIP   10.101.188.24    <none>        8000/TCP         6d2h
django-external        NodePort    10.110.114.132   <none>        8000:30000/TCP   6d2h
kubernetes             ClusterIP   10.96.0.1        <none>        443/TCP          6d2h
mysql                  ClusterIP   10.109.255.156   <none>        3306/TCP         6d2h
mysql-external         NodePort    10.102.177.225   <none>        3306:30001/TCP   6d2h
pubsub                 ClusterIP   10.110.164.231   <none>        8042/TCP         6d2h
pubsub-external        NodePort    10.102.95.122    <none>        8042:30002/TCP   6d2h
redis                  ClusterIP   10.108.250.216   <none>        6379/TCP         6d2h
redis-external         NodePort    10.111.54.18     <none>        6379:30003/TCP   6d2h
selenium-hub           ClusterIP   10.96.24.19      <none>        4444/TCP         6d2h
selenium-hub-external  NodePort    10.96.31.83      <none>        4444:30004/TCP   6d2h
niall@Nialls-Mac-mini:~$ curl localhost:30000/healthz
OKniall@Nialls-Mac-mini:~$
```

We can expose the cluster services as NodePorts to the host machine, access the service for further testing automation.

Recap 1/4

Containers:

Generally enforce breaking your app up into processes, make simple build pipelines possible very quickly, and enforce certain aspects of the 12 Factor App model.

Recap 2/4

Pods:

Pods ask us to consider which pieces of our application are co-dependent and must be scaled together.

No one gets left behind.

Recap 3/4

Deployments:

Deployments ask us to refactor our application into units that can be replicated. We're forced to think through how these units will scale, and what pieces we can outsource to our provider when going to production. This is all without firmly committing to a production platform.

Recap 4/4

Services:

Services ask us to consider how we'll handle discovery and load balancing in production- these are fundamental problems of any Microservices Based Architecture, so we shouldn't take them for granted.

Services also provide us with an insanely easy way of quickly setting up an integration environment- even locally.

Thanks!

niall@niallbyrne.ca

github.com/niall-byrne