# Production-ready Flask & Django apps on Kubernetes

Jamie Hewland

PyConZA 2019

aruba
a Hewlett Packard Enterprise company

# Hi, I'm Jamie 👋


@jayhewland

jamie.hewland@hpe.com

Cape Town, South Africa


- ~3 years as a Site Reliability Engineer (SRE) working on container things
- Now a Backend Engineer at Aruba working on Python services
- Spoke at PyConZA 2017 about containerizing Django web apps: https://youtu.be/T2hooQzvurQ

aruba
a Hewlett Packard Enterprise company

# About this talk

- **This is not an intro to Kubernetes or Python web apps**
- **But if you are familiar with one, this talk may help you understand the other**
- **There will be more YAML than Python 😢**

**All text is in Arial**

Kubernetes resources are always in a `monospace` font, **bold**, and Title Case:

- **`Deployment`**
- **`Pod`**
- **`ConfigMap`**

# AGENDA

**INTRO**
- Why Kubernetes?
- “The WSGI Stack”

**ADAPTING TO KUBERNETES**
- Handling requests
- Static file serving
- Deployments
- Configuration
- Credentials
- Database migrations

**THE FUTURE**

aruba
a Hewlett Packard Enterprise company

# Why Kubernetes?

aruba
a Hewlett Packard Enterprise company

## Abstraction

- Compute/networking/ storage abstracted over multiple hosts
- Developers don’t think about sysadmin tasks
- Multi-cloud support
- Declarative infrastructure requirements for apps
- Improve resource usage

## Automation

- Deployments with minimal manual intervention
- Fail-over when an app misbehaves
- Tie together multiple tools

## Ecosystem

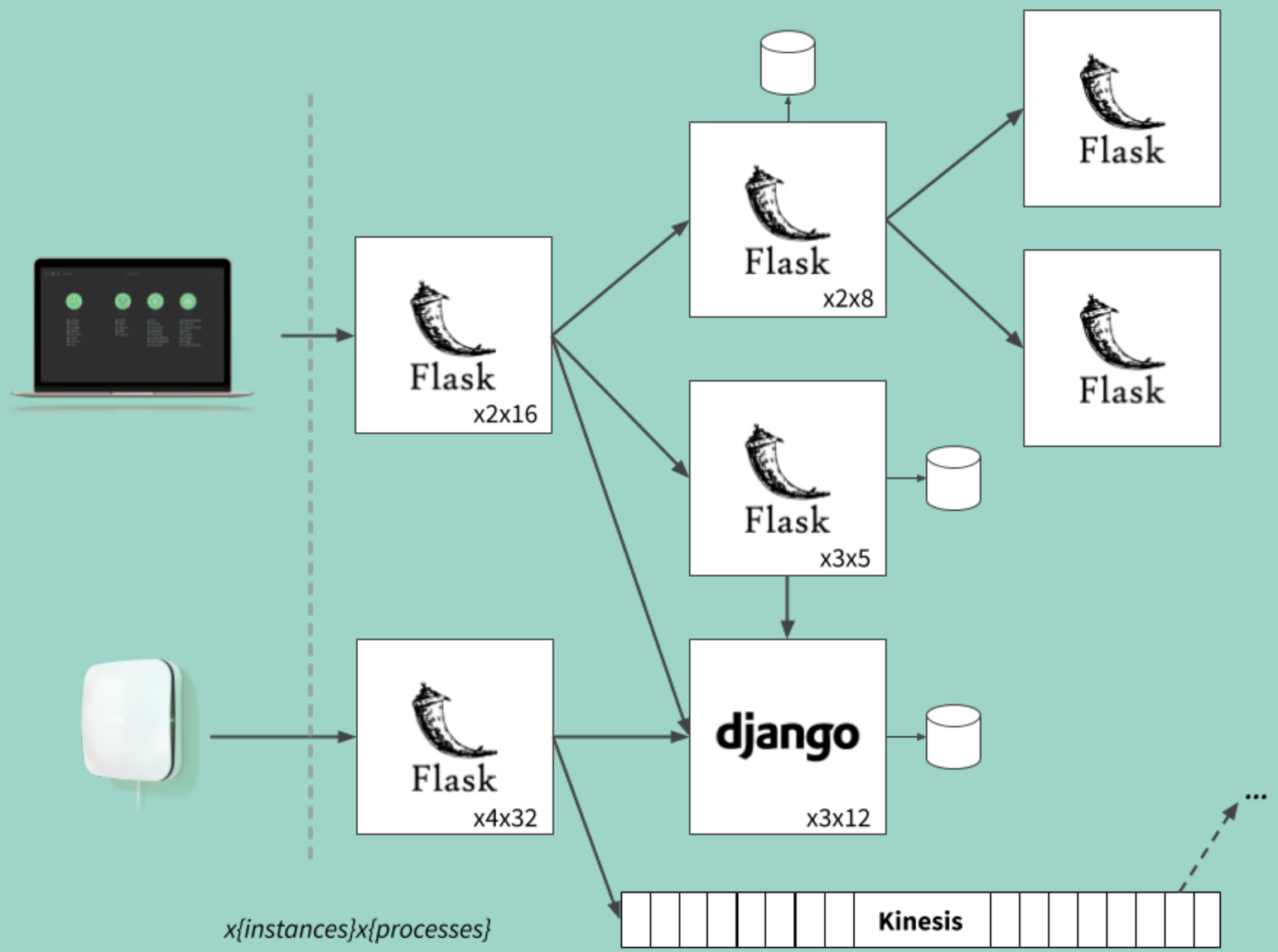
- Huge ecosystem of tools through the CNCF to extend Kubernetes
- Spend less time implementing something that has been done before

⚠

***Kubernetes is not the solution for every problem***


aruba
a Hewlett Packard Enterprise company

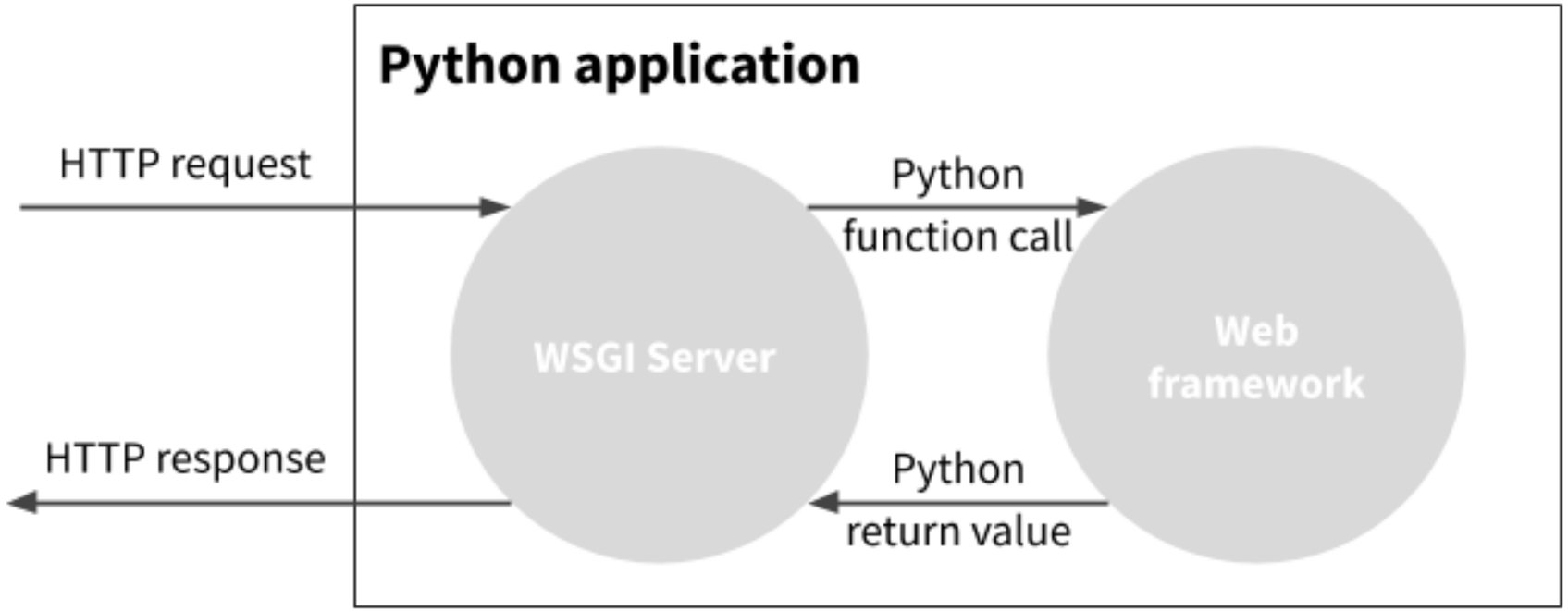
# Aruba UXI architecture



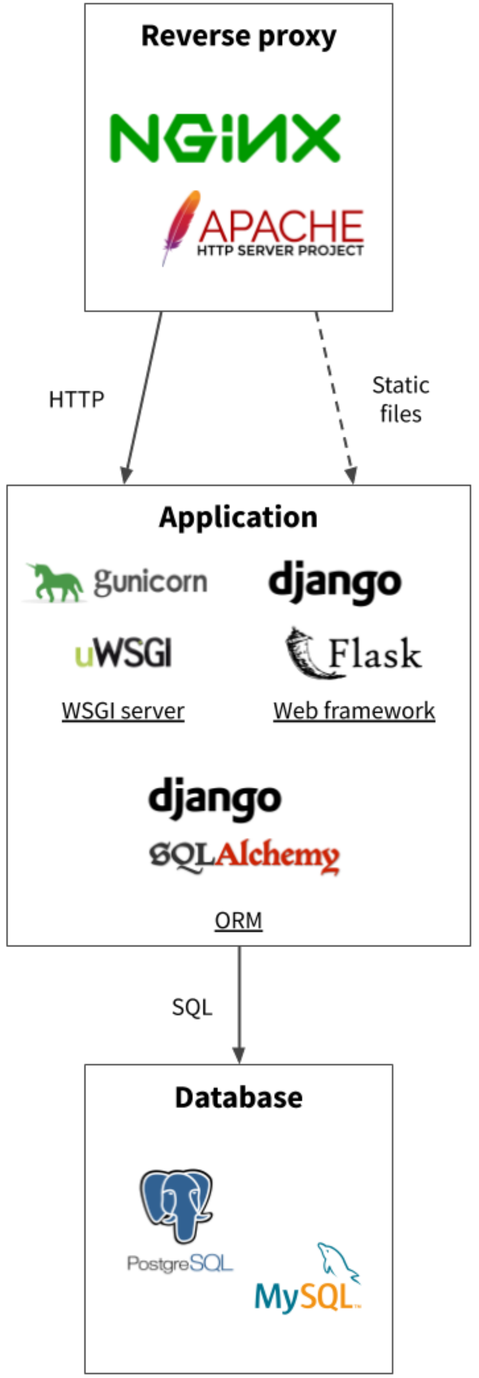

aruba
a Hewlett Packard Enterprise company

aruba
a Hewlett Packard Enterprise company

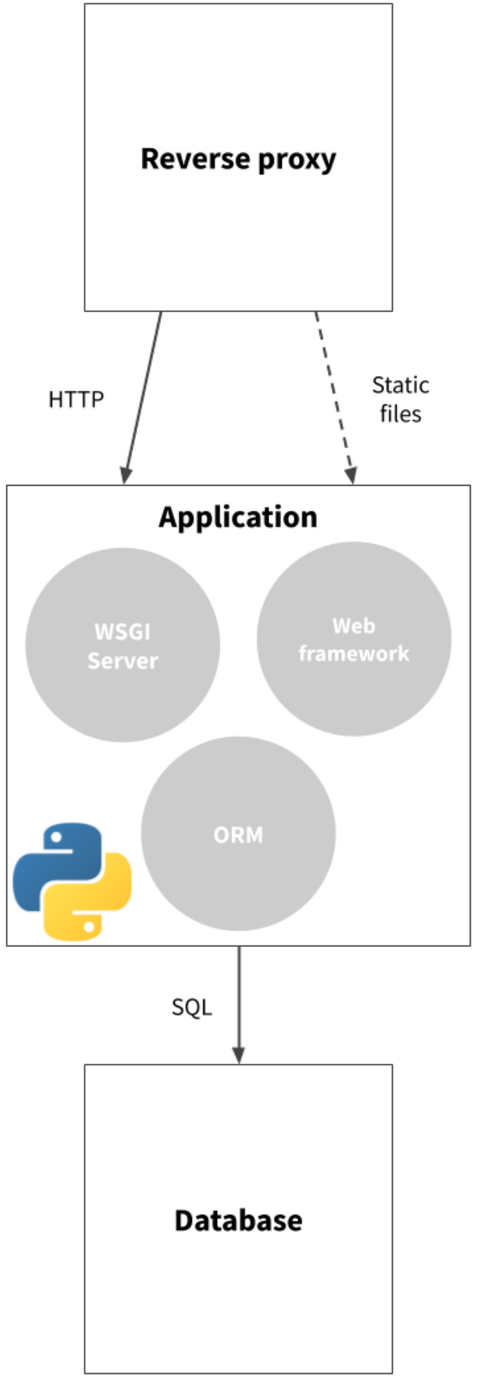
# WSGI: Web Server Gateway Interface



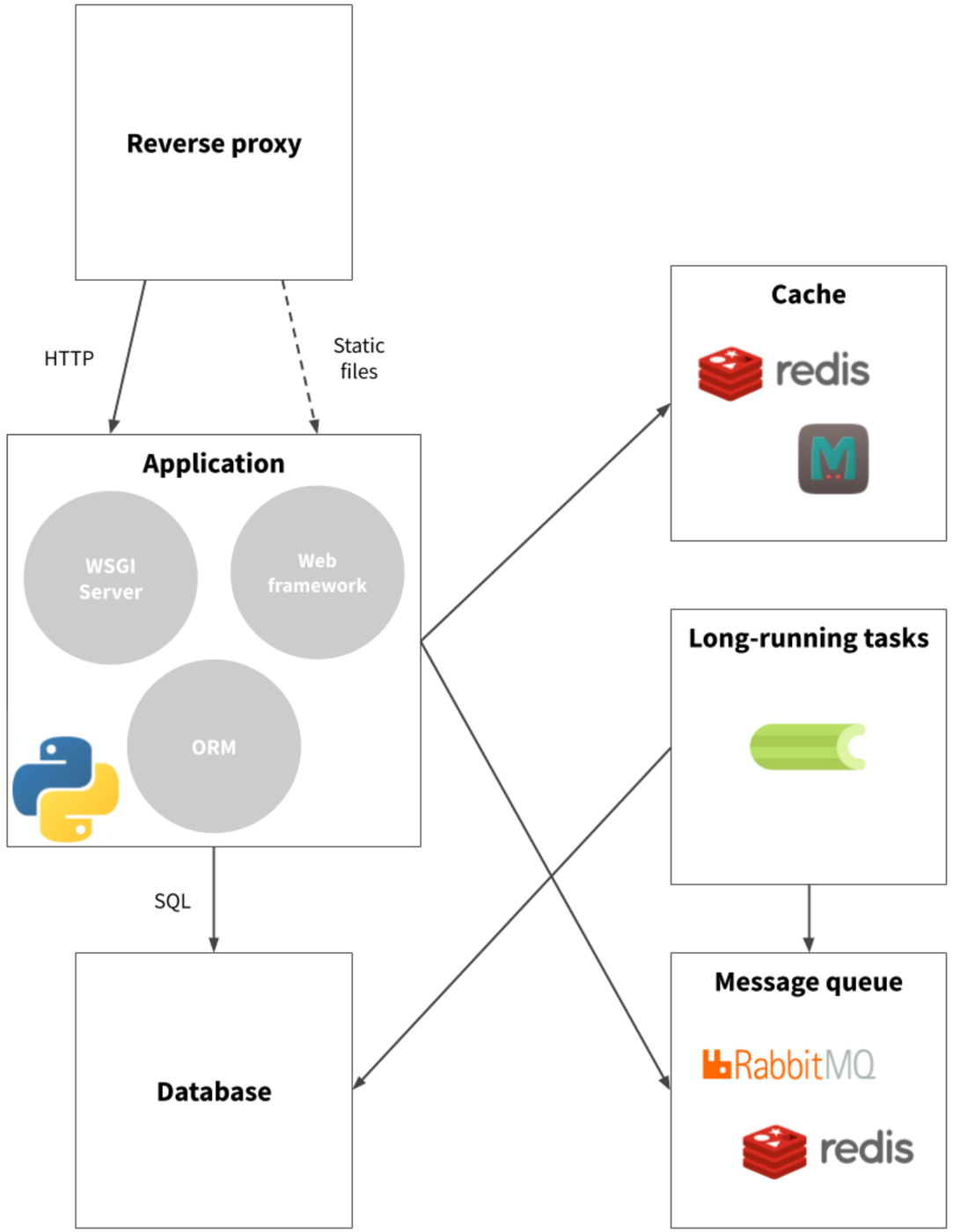
# The WSGI stack



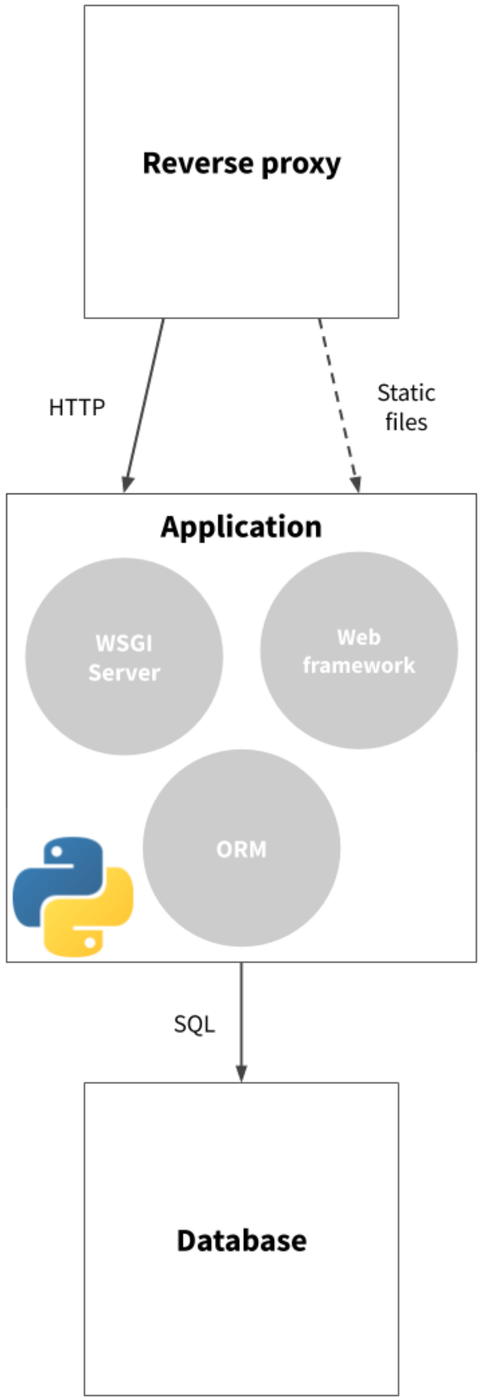
# The WSGI stack



# The WSGI stack



# The WSGI stack



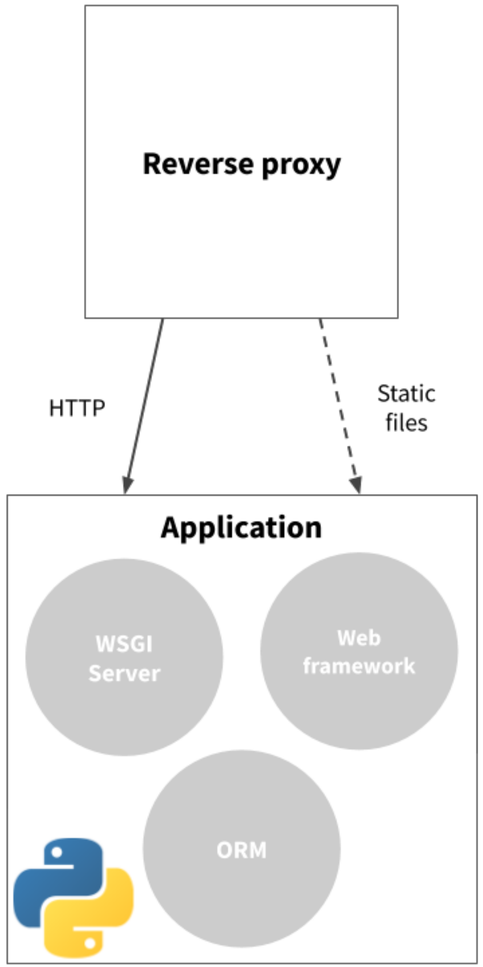

## *Out of scope*

Cache

Long-running tasks

Message queue

# The WSGI stack




## Out of scope

Cache

Long-running tasks

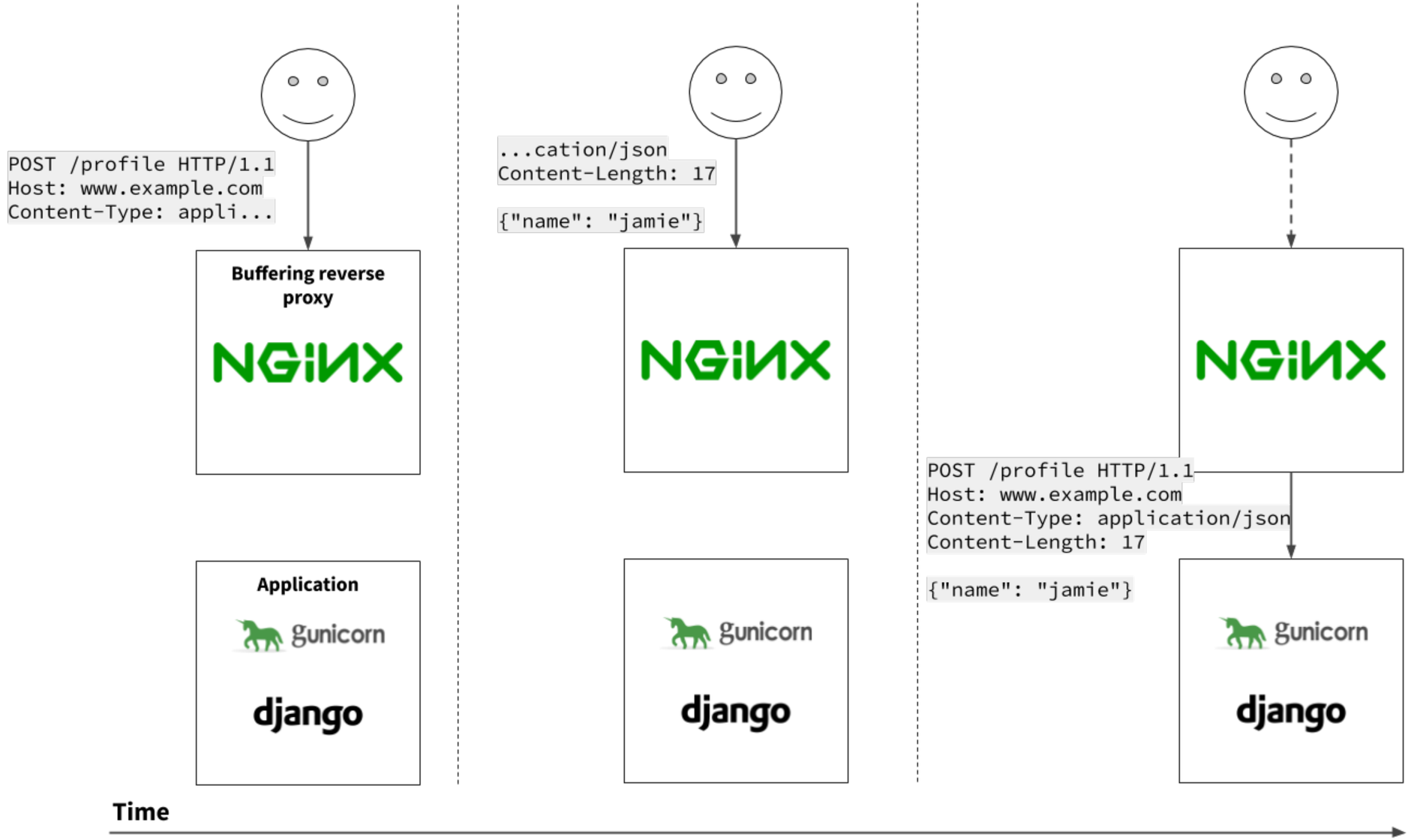
Database

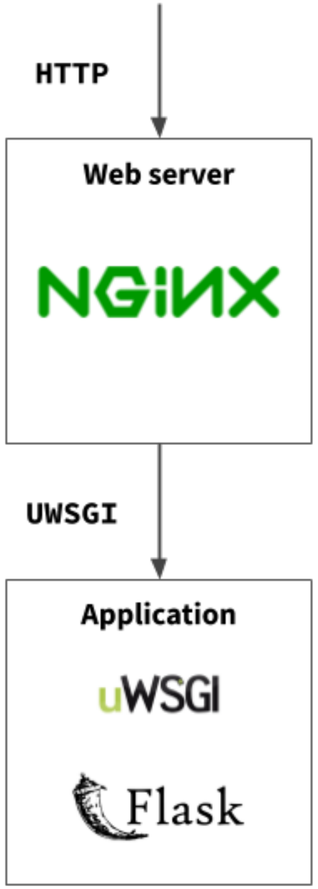
Message queue

aruba
a Hewlett Packard Enterprise company

# Handling requests

# Gunicorn’s design






**Most WSGI servers are designed like this:**

```
the web client
  -> ~~the internet~~
    -> the web server (Nginx)
      -> local socket
        -> WSGI server
          -> your application
```

**The web server…**

- Buffers slow client requests so that app code can stay synchronous (Gunicorn)
- Converts from HTTP to UWSGI protocol (uWSGI)
- “Shields” the Python program

With Kubernetes we can’t *expect* there to be a web server

aruba
a Hewlett Packard Enterprise company

# Buffering reverse proxy

aruba
a Hewlett Packard Enterprise company

SOLUTION

## Sidecar proxy in Pod

Run a buffering proxy such as Nginx in the same **Pod** as the application

- ✅ Don't modify the app
- ✅ Maintain similarity with pre-K8S environment
- ❌ Additional container for each Pod
- ❌ More K8S YAML

ALTERNATIVE

## Remove need for proxy

**Run async WSGI workers that don't require a buffering proxy**

- ✅ No proxy to set up
- ✅ Potentially lower overhead
- ✅ OK with an `Ingress`
- ❌ Potential code changes, app behaviour changes

GOLD STAR ⭐

## Service mesh

Use a service mesh to "abstract the network layer".

- ✅ No proxy specific to app to set up
- ✅ All the pros of a service mesh
- ❌ Have to run a service mesh

# Buffering reverse proxy

aruba
a Hewlett Packard Enterprise company

***RECOMMENDED***

SOLUTION

## Sidecar proxy in Pod

Run a buffering proxy such as Nginx in the same **Pod** as the application

- ✅ Don't modify the app
- ✅ Maintain similarity with pre-K8S environment
- ❌ Additional container for each Pod
- ❌ More K8S YAML

ALTERNATIVE

## Remove need for proxy

**Run async WSGI workers that don't require a buffering proxy**

- ✅ No proxy to set up
- ✅ Potentially lower overhead
- ✅ OK with an `Ingress`
- ❌ Potential code changes, app behaviour changes

GOLD STAR ⭐

## Service mesh

Use a service mesh to "abstract the network layer".

- ✅ No proxy specific to app to set up
- ✅ All the pros of a service mesh
- ❌ Have to run a service mesh

# Nginx sidecar

```yaml
containers:
- name: django
  image: jamiehewland/django-app:0aacd1d
  command: ["gunicorn"]
  args: ["--bind=127.0.0.1:8000", "django_app:wsgi"]
  env:
  - name: DJANGO_SETTINGS_MODULE
    value: django_app.settings.production

- name: nginx
  image: nginx:stable
  ports:
  - containerPort: 80
  volumeMounts:
  - name: nginx-config
    mountPath: /etc/nginx/conf.d

volumes:
- name: nginx-config
  configMap:
    name: nginx-config
```

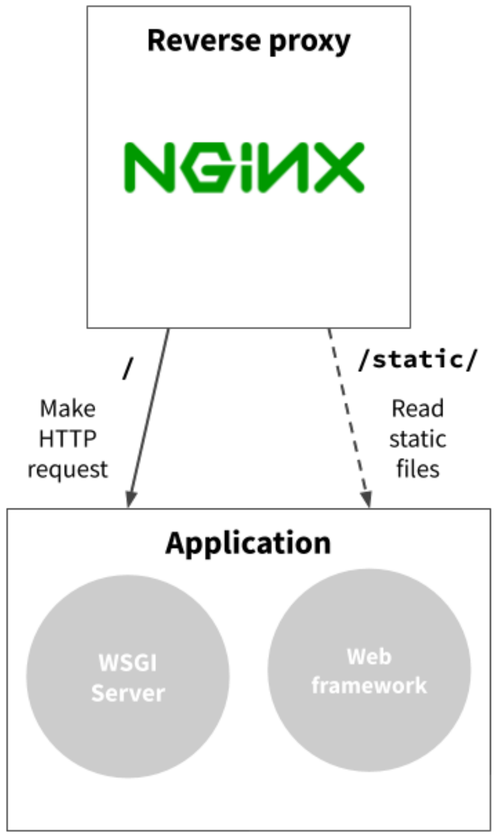
```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: nginx-config
data:
  wsgi.conf: |
upstream wsgi {
    server 127.0.0.1:8000;
}

server {
    listen 80;
    server_name _;

    location / {
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
        proxy_set_header Host $http_host;
        proxy_redirect off;
        proxy_pass http://wsgi;
    }
}
```

# Serving static files

# Serving static files




```
location /static/ {
    alias /path/to/static/files;
}

location / {
    proxy_set_header …
    proxy_pass http://127.0.0.1:8000;
}

GET /admin/                          [Django]
GET /static/admin/css/base.css       [Nginx]
```

**uWSGI:**
*“[…] it’s inefficient to serve static files via uWSGI. Instead, serve them directly from Nginx and completely bypass uWSGI.”*

# Serving static files

aruba
a Hewlett Packard Enterprise company

## SOLUTION

### Sidecar web server in Pod

Run a web server such as Nginx in the same **`Pod`** as the application, share a **`Volume`** with static files

- ✅ Don't modify the app
- ✅ Maintain similarity with pre-K8S environment
- ❌ Possibly additional container for each **`Pod`**
- ❌ **`Volume`** increases startup time

## ALTERNATIVE

### Serve static files from S3

**Use `django-storages` or similar library to upload static files to S3 or similar**

- ✅ Simple & easy
- ❌ Additional step to upload files
- ❌ Unoptimised (compression/caching headers)

## GOLD STAR ⭐

### WhiteNoise + CDN

Serve optimised static files with Python using WhiteNoise library

- ✅ Automatically compressed assets
- ✅ Smart caching headers
- ✅ App & static files always in sync
- ❌ Potentially larger container images

# Serving static files

aruba
a Hewlett Packard Enterprise company

SOLUTION

## Sidecar web server in Pod

Run a web server such as Nginx in the same **`Pod`** as the application, share a **`Volume`** with static files

- ✅ Don't modify the app
- ✅ Maintain similarity with pre-K8S environment
- ❌ Possibly additional container for each **`Pod`**
- ❌ **`Volume`** increases startup time

ALTERNATIVE

## Serve static files from S3

**Use `django-storages` or similar library to upload static files to S3 or similar**

- ✅ Simple & easy
- ❌ Additional step to upload files
- ❌ Unoptimised (compression/caching headers)

***RECOMMENDED***

GOLD STAR ⭐

## WhiteNoise + CDN

Serve optimised static files with Python using WhiteNoise library

- ✅ Automatically compressed assets
- ✅ Smart caching headers
- ✅ App & static files always in sync
- ❌ Potentially larger container images

# WhiteNoise

**Advantages**

- Serve static files directly from Python
- Compressed assets (w/correct headers): GZIP, Brotli
- Automatic far-future cache headers

**Isn't serving static files from Python horribly slow?**

- Use a CDN
- All assets generated offline
- WSGI servers can use the `sendfile` syscall

http://whitenoise.evans.io/en/stable/

```
› pip install whitenoise
```

**Django:**

1. Add middleware class
2. (Optional) Add staticfiles storage class
3. Remember to run `collectstatic`

**Flask:**

1. Add WSGI middleware
2. (Optional) Configure static file directories

# Deployments

# Deployments

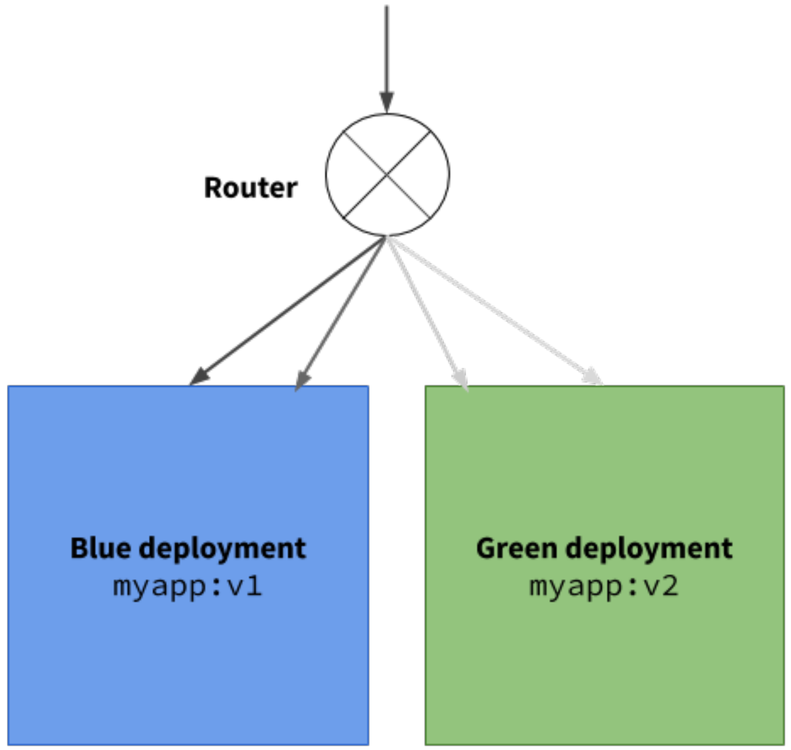
***Previously*: update the code, then:**

- Gunicorn and uWSGI support graceful reload via `SIGHUP`
- Some may just restart the process

**With containers we *never* change the running code, rather:**

- Build a new image with the new code
- Run new container with new image
- Switch traffic over to new container
- Shut down old container

https://martinfowler.com/bliki/BlueGreenDeployment.html



# Deployments

aruba
a Hewlett Packard Enterprise company

## SOLUTION

### `Deployment`

A **`Deployment`** resource with default settings such as `RollingUpdate`.

- ✅ Basic blue/green deploy
- ❌ Limited deployment history
- ❌ Limited update strategies

## NEXT STEPS

### Multiple `Deployments`

**Shift traffic between two or more `Deployments`**

- ✅ Fast roll-backs
- ✅ Open the door to more advanced deployments
- ❌ Increased resource usage (run 2 copies)
- ❌ Need extra tool to manage deployment

## GOLD STAR ⭐

### Advanced deployment strategies

Canarying, gradual red/black deployments, traffic shadowing

- ✅ “Test in production!”
- ❌ Requires sophisticated tooling & expertise

# Deployments

aruba
a Hewlett Packard Enterprise company

***RECOMMENDED***

SOLUTION

## `Deployment`

A **`Deployment`** resource with default settings such as `RollingUpdate`.

- ✅ Basic blue/green deploy
- ❌ Limited deployment history
- ❌ Limited update strategies

NEXT STEPS

## Multiple `Deployments`

**Shift traffic between two or more `Deployments`**

- ✅ Fast roll-backs
- ✅ Open the door to more advanced deployments
- ❌ Increased resource usage (run 2 copies)
- ❌ Need extra tool to manage deployment

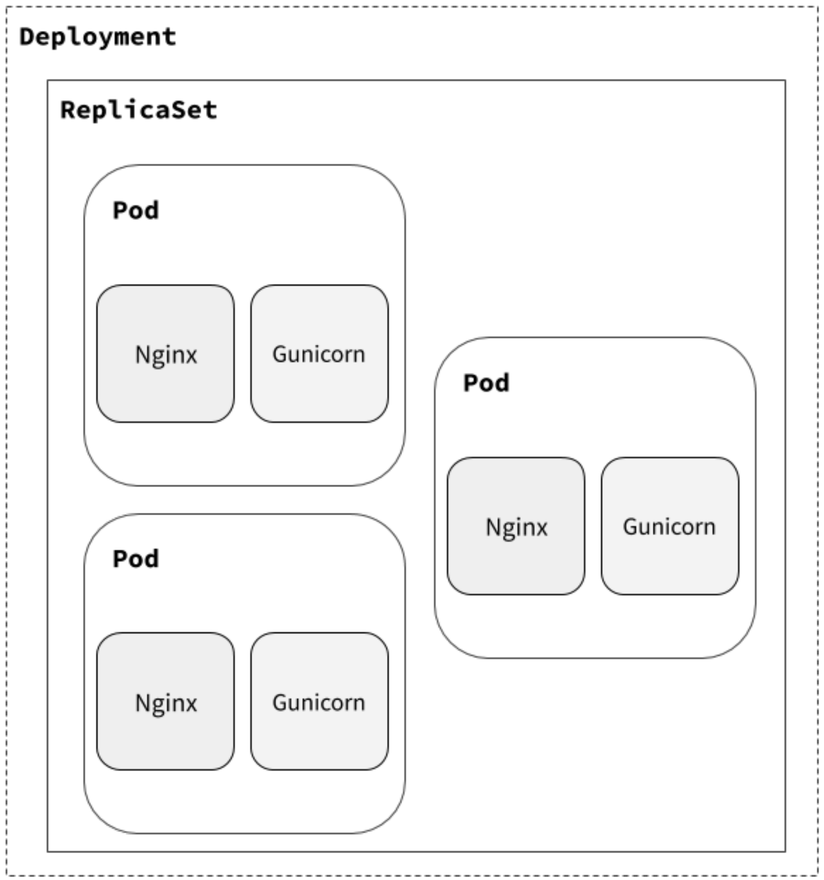
GOLD STAR ⭐

## Advanced deployment strategies

Canarying, gradual red/black deployments, traffic shadowing

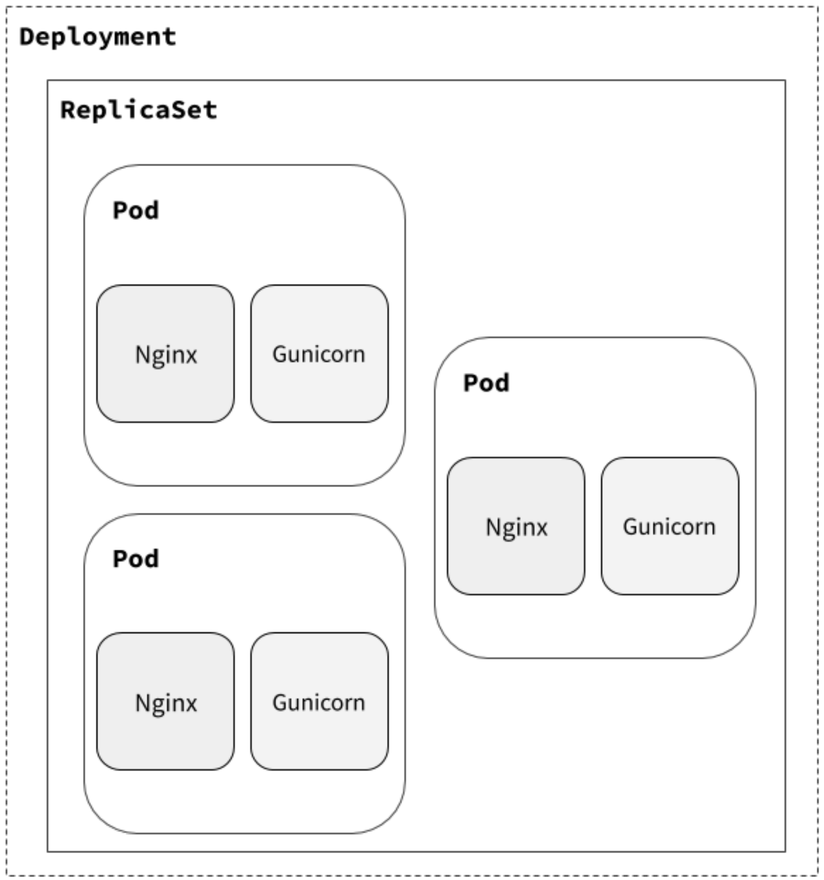
- ✅ “Test in production!”
- ❌ Requires sophisticated tooling & expertise

# Kubernetes **Deployment**




```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: django-deployment
spec:
  replicas: 3
  selector:
    matchLabels:
      app: django-app
  template:
    metadata:
      labels:
        app: django-app
    spec:
      containers:
      - name: django
        image: jamiehewland/django-app:0aacd1d
        env:
        - name: DJANGO_SETTINGS_MODULE
          value: django_app.settings.production
      - name: nginx
        image: nginx:stable
        ports:
        - containerPort: 80
        volumeMounts:
        - name: nginx-config
          mountPath: /etc/nginx/conf.d
      volumes:
      - name: nginx-config
        configMap:
          name: nginx-config
```

# Kubernetes **Deployment**




3 replicas

Django container

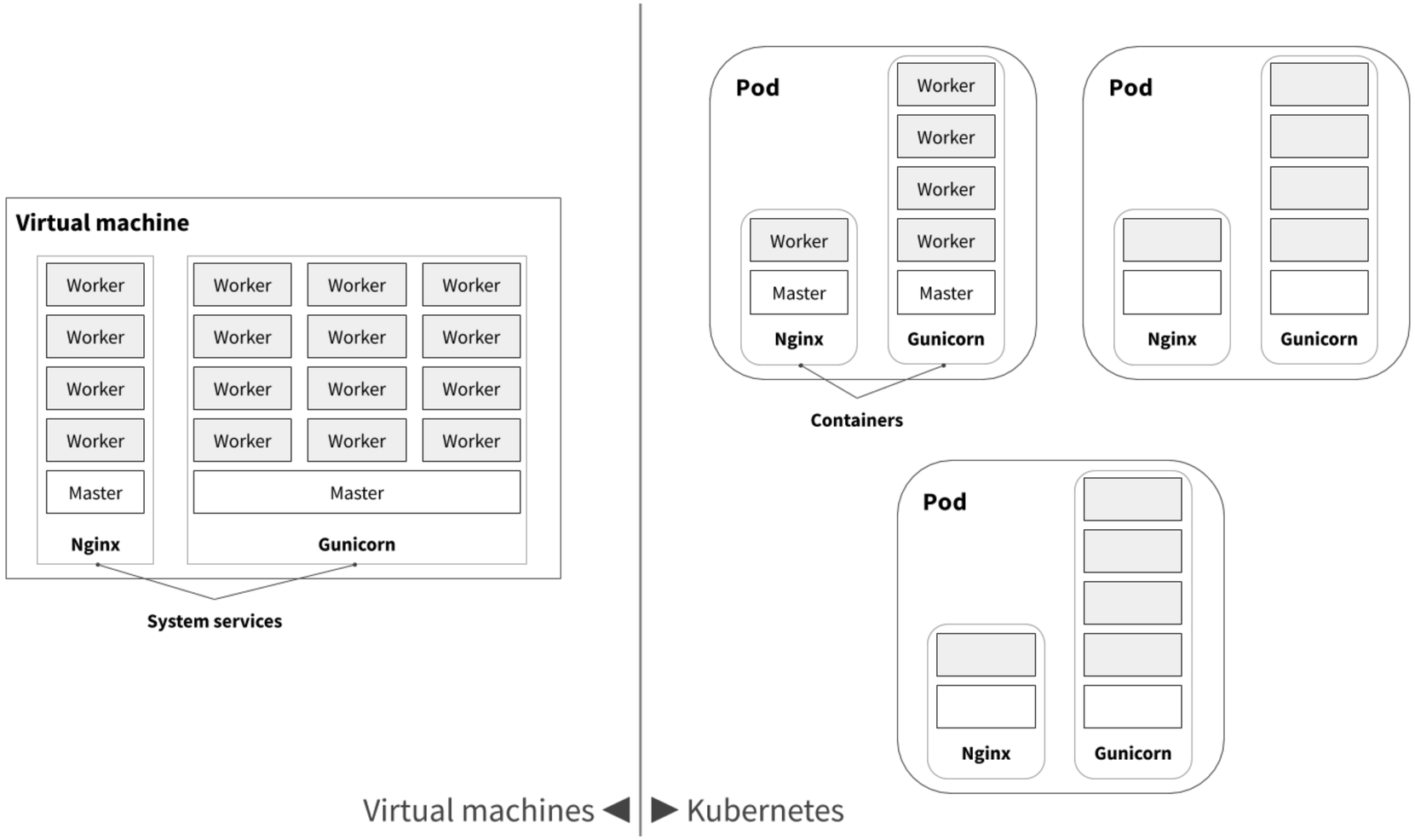
Nginx container

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: django-deployment
spec:
  replicas: 3
  selector:
    matchLabels:
      app: django-app
  template:
    metadata:
      labels:
        app: django-app
    spec:
      containers:
      - name: django
        image: jamiehewland/django-app:0aacd1d
        env:
        - name: DJANGO_SETTINGS_MODULE
          value: django_app.settings.production
      - name: nginx
        image: nginx:stable
        ports:
        - containerPort: 80
        volumeMounts:
        - name: nginx-config
          mountPath: /etc/nginx/conf.d
      volumes:
      - name: nginx-config
        configMap:
          name: nginx-config
```

# Smaller instances

**Pros:**
- Finer-grained scaling
- More redundancy
- Better for canarying

**Cons:**
- Increased overhead (especially RAM)
- Greater need for automation



# Configuration

aruba
a Hewlett Packard Enterprise company

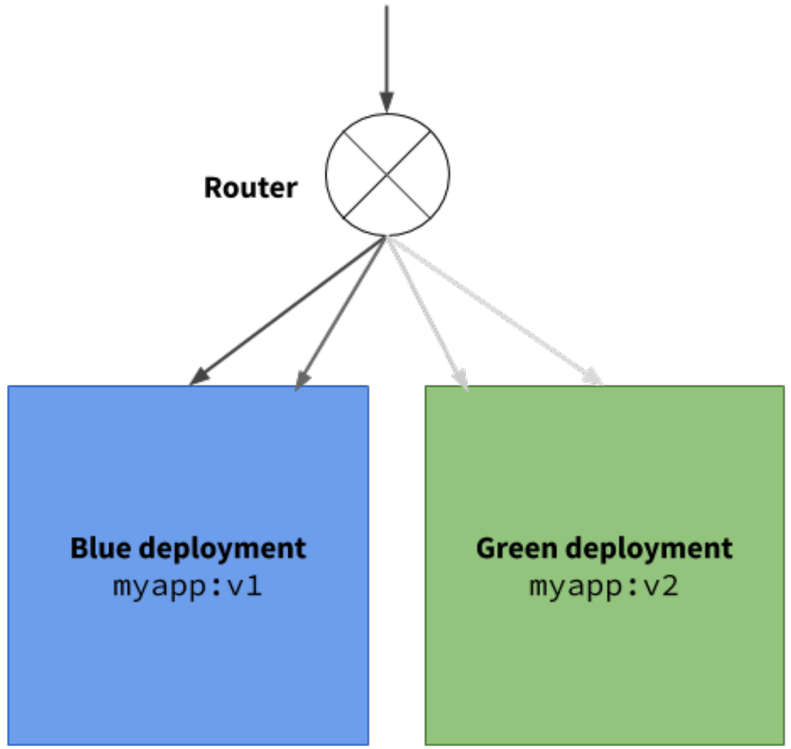
# Configuration

***Previously*: update the config by…**

- Running Configuration Management tool (e.g. Chef, Puppet, Ansible)
- Reload/restart server

**Like code, we *never* want the config a container runs with to change:**

- Run new container with the new config
- Switch traffic over to new container
- Shut down old container



# Configuration

aruba
a Hewlett Packard Enterprise company

## STARTING OUT

### Environment variables

Set environment variables for containers in the **`Deployment`**

- ✅ It works
- ❌ Lots of YAML
- ❌ Have to pack everything into strings

## BETTER

### `ConfigMap`

**Store config in a ConfigMap and import in app via env vars or file volume**

- ✅ Separate app & config
- ❌ **`ConfigMap`**s are mutable

## BEST PRACTICE

### Immutable `ConfigMaps`

Create a new **`ConfigMap`** for each config change

- ✅ Keep record of previous configs
- ✅ Not difficult to do (use Kustomize or Helm)
- ❌ Probably need additional tool

# Configuration

aruba
a Hewlett Packard Enterprise company

## STARTING OUT

### Environment variables

Set environment variables for containers in the **`Deployment`**

- ✅ It works
- ❌ Lots of YAML
- ❌ Have to pack everything into strings

## BETTER

### `ConfigMap`

**Store config in a ConfigMap and import in app via env vars or file volume**

- ✅ Separate app & config
- ❌ **`ConfigMap`**s are mutable

*RECOMMENDED*

## BEST PRACTICE

### Immutable `ConfigMaps`

Create a new **`ConfigMap`** for each config change

- ✅ Keep record of previous configs
- ✅ Not difficult to do (use Kustomize or Helm)
- ❌ Probably need additional tool

# ConfigMap

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: django-config
data:
  LOG_LEVEL: info
  DATABASE_HOST: django-app.cvqdtcqjj6f2.us-west-2.rds.amazonaws.com
  DATABASE_NAME: django-app
```

```yaml
containers:
- name: django
  image: jamiehewland/django-app:0aacd1d
  env:
  - name: DJANGO_SETTINGS_MODULE
    value: django_app.settings.production
  envFrom:
  - configMapRef:
      name: django-config
```

# Credentials

# Credentials

aruba
a Hewlett Packard Enterprise company

## STARTING OUT

### Store with the configuration

Store credentials just like you would any other configuration

- ✅ Possibly no worse than what you're doing already
- ❌ Credentials stored in plain-text in etcd

## BEST PRACTICE

### `Secret`

**Store credentials in a `Secret` and and import in app via env vars or (preferably) file volume**

- ✅ Separate sensitive secrets from config
- ❌ May have to create `Secret`s out-of-band
- ❌ Make sure your K8S is configured to use envelope encryption

## GOLD STAR ⭐

### HashiCorp Vault or similar

Store secrets (possibly dynamic ones) in a dedicated service

- ✅ About as secure as it gets right now
- ✅ Other advantages to Vault, e.g. auditing
- ❌ Additional infrastructure
- ❌ Likely requires app-level changes

# Credentials

aruba
a Hewlett Packard Enterprise company

## STARTING OUT

### Store with the configuration

Store credentials just like you would any other configuration

- ✅ Possibly no worse than what you're doing already
- ❌ Credentials stored in plain-text in etcd

***RECOMMENDED***

## BEST PRACTICE

### `Secret`

**Store credentials in a `Secret` and and import in app via env vars or (preferably) file volume**

- ✅ Separate sensitive secrets from config
- ❌ May have to create `Secret`s out-of-band
- ❌ Make sure your K8S is configured to use envelope encryption

## GOLD STAR ⭐

### HashiCorp Vault or similar

Store secrets (possibly dynamic ones) in a dedicated service

- ✅ About as secure as it gets right now
- ✅ Other advantages to Vault, e.g. auditing
- ❌ Additional infrastructure
- ❌ Likely requires app-level changes

# Secret

```
❯ kubectl create secret generic django-credentials \
    --from-literal DATABASE_USER="$DATABASE_USER" \
    --from-literal DATABASE_PASSWORD="$DATABASE_PASSWORD"
```

```
containers:
- name: django
  image: jamiehewland/django-app:0aacd1d
  env:
  - name: DJANGO_SETTINGS_MODULE
      value: django_app.settings.production
  envFrom:
  - configMapRef:
      name: django-config
  - secretRef:
      name: django-credentials
```

aruba
a Hewlett Packard
Enterprise company

# Database migrations

# Database migrations

aruba
a Hewlett Packard Enterprise company

## STARTING OUT

### `initContainers`

Add a container in the `initContainers` section to run on **Pod** startup

- ✅ Basic automation
- ❌ Run on every **Pod** start
- ❌ No post-/pre-deployment control

## BEST PRACTICE

### Job triggered manually

**Run a one-off Job manually each time a migration needs to be run**

- ✅ Single process running migration
- ✅ Separate deployment of schema changes from deployment of app
- ❌ Automation?

## GOLD STAR ⭐

### Job triggered by CD

Post- and/or pre-deployment **Job** run automatically by Continuous Deployment system

- ✅ Automated
- ❌ Requires mature CD system

# Database migrations

aruba
a Hewlett Packard Enterprise company

## STARTING OUT

### `initContainers`

Add a container in the `initContainers` section to run on **Pod** startup

- ✅ Basic automation
- ❌ Run on every **Pod** start
- ❌ No post-/pre-deployment control

***RECOMMENDED***

## BEST PRACTICE

### Job triggered manually

**Run a one-off Job manually each time a migration needs to be run**

- ✅ Single process running migration
- ✅ Separate deployment of schema changes from deployment of app
- ❌ Automation?

## GOLD STAR ⭐

### Job triggered by CD

Post- and/or pre-deployment **Job** run automatically by Continuous Deployment system

- ✅ Automated
- ❌ Requires mature CD system

## Job migrations

Run a migration using the image for the app

```yaml
apiVersion: batch/v1
kind: Job
metadata:
  name: django-migrations
spec:
  template:
    spec:
      containers:
      - name: django
        image: jamiehewland/django-app:0aacd1d
        command: ["python", "manage.py", "migrate"]
        env:
        - name: DJANGO_SETTINGS_MODULE
          value: django_app.settings.production
        - name: DATABASE_USER
          valueFrom:
            secretKeyRef:
              name: django-credentials
              key: DATABASE_USER
        - name: DATABASE_PASSWORD
          valueFrom:
            secretKeyRef:
              name: django-credentials
              key: DATABASE_PASSWORD
      restartPolicy: Never
```

| Problem | “Legacy” solution | Recommended Kubernetes solution |
|---|---|---|
| Buffering reverse proxy | Nginx on host | `Pod` with sidecar proxy |
| Static file serving | Nginx on host | WhiteNoise + CDN |
| Deployments | WSGI server reload signal | `Deployment` with blue/green strategy |
| Configuration | Configuration Management tool | `ConfigMap` (and/or 12-factor app) |
| Credentials | Configuration Management tool | `Secret` |
| Database migrations | Manual or CD step | `Job` migration container |

# tl;dl

**This is a lot to get right *(or wrong)***

Why is it so complicated?

- **Established tools like Gunicorn and uWSGI were designed before containers/Kubernetes**
  - Process management
  - Hot-code reload
  - User-switching
  - Daemonizing
  - Many tuneable settings

```
uwsgi --chdir=/path/to/your/project \
    --module=mysite.wsgi:application \
    --env DJANGO_SETTINGS_MODULE=mysite.settings \
    --master --pidfile=/tmp/project-master.pid \
    --socket=127.0.0.1:49152 \      # can also be a file
    --processes=5 \                 # number of worker processes
    --uid=1000 --gid=2000 \         # if root, uwsgi can drop privileges
    --harakiri=20 \                 # respawn processes taking more than 20 seconds
    --max-requests=5000 \           # respawn processes after serving 5000 requests
    --vacuum \                      # clear environment on exit
    --home=/path/to/virtual/env \   # optional path to a virtualenv
    --daemonize=/var/log/uwsgi/yourproject.log      # background the process
```

**This is a lot to get right *(or wrong)***

Why is it so complicated?

- **Established tools like Gunicorn and uWSGI were designed before containers/Kubernetes**
- **A lot of stuff is not necessary for containers**
  - Too many adjustable settings for beginners
  - Process management handled by Docker or Kubernetes
- **Expecting a traditional web server like Nginx is no longer a fair assumption**
  - We're not hosting LAMP servers
- **Learning curve from dev server to production-ready is steep (Kubernetes or not)**

# ASGI


- Asynchronous Server Gateway Interface
- `async/await` support: WebSockets & HTTP/2
- Early ASGI server implementations:
  - Daphne
  - Uvicorn
  - Hypercorn
- Needs new frameworks:
  - Django Channels, Django 3 (coming soon)
  - Quart (async Flask port)
  - Starlette, Sanic, …
- Async ecosystem still evolving


@jayhewland

jamie.hewland@hpe.com


**Further reading:**

- ***Deploying Django web applications in Docker containers***: youtu.be/T2hooQzvurQ
- ***Kubernetes, Local to Production with Django*** **by Mark Gituma:** bit.ly/k8slocaltoprod

# Thank you

aruba
a Hewlett Packard Enterprise company