Deployment & Beyond

SECT.13.2026

- Where can we deploy our software, what are the trade-offs?
- What is different about production vs. dev environments?
- What are common deployment strategies?

The Physical Server

Rented Bare-Metal

Renting a physical machine. Someone else plugs it in, meters bandwidth, etc.

Slightly more than ownership, but with increased flexibility if needs change.

Often requires contracts: months or years.

You are responsible for everything from the operating system & up.

Examples: Hetzner, OVH, 'Dedicated' instances on AWS/GCP/Azure

Virtual Private Server

Rented bare-metal sold in fractional slices. For example: a 128GB RAM machine sold as 64 2GB VPS.

Performance is decreased somewhat due to overhead.

Allows for smaller purchases, but gets expensive at higher tiers.

Examples: EC2, GCP, Azure

Managed Application Hosts

These providers offer a hosted service (or services) and/or a way to run your applications without thinking about server hardware/OS specifics. Often even the amount of RAM/compute are only abstractly defined.

Hosted Postgres (RDS), Vercel, Railway, Heroku, etc.

Least flexibility, must tailor deployment to what is offered. May lack desired versions/features.

Least responsibility: server upgrades and even lots of security obligations offloaded to host. Highest direct cost. Paying significant markup over what a server costs.

Special Cases: Lambda Functions & Managed Docker/K8S

Server Specifications

Most services are **overprovisioned**, either intentionally to help with scaling, or accidentally due to a poor understanding of performance/needs.

If cost-sensitive, important to understand where application is limited: e.g. pay for more RAM to cache aggressively to save on CPU vs. increase workers to handle high concurrency.

Site	Traffic	Servers	Cost
CAPP Servers	Low	~1 32GB RAM / 1 TB SSD on Hetzner	~$45/mo.
Open States	1-2M API calls/day	Single 4GB RAM site/API server Separate PostgresDB ephemeral VPS for scrapers	$40/mo $100/mo $250/mo (mostly scraper costs)
Sunlight Apps	~100k/day, dozens of apps	3x32GB servers on AWS + bandwidth	$1500 + $2000/mo
Public Video Site	Traffic: 10-20M streams/day with spikes	20-60 app servers w/ CDN	compute $100k; bandwidth a few million/mo. (negotiated rate)

Production Environments

```
n|+2)
Performing system checks...

System check identified no issues (0 silenced).
May 15, 2025 - 21:14:44
Django version 5.2, using settings 'config.settings'
Starting development server at http://127.0.0.1:9933/
Quit the server with CONTROL-C.

WARNING: This is a development server. Do not use it in a production setting. Use a
 production WSGI or ASGI server instead.
For more information on production servers see: https://docs.djangoproject.com/en/5
.2/howto/deployment/
```

Production Environment

What does a production environment need?

- Forward Proxy / Web Server (software)
- Application Server
- Database and/or Cache (often separate server)

Often

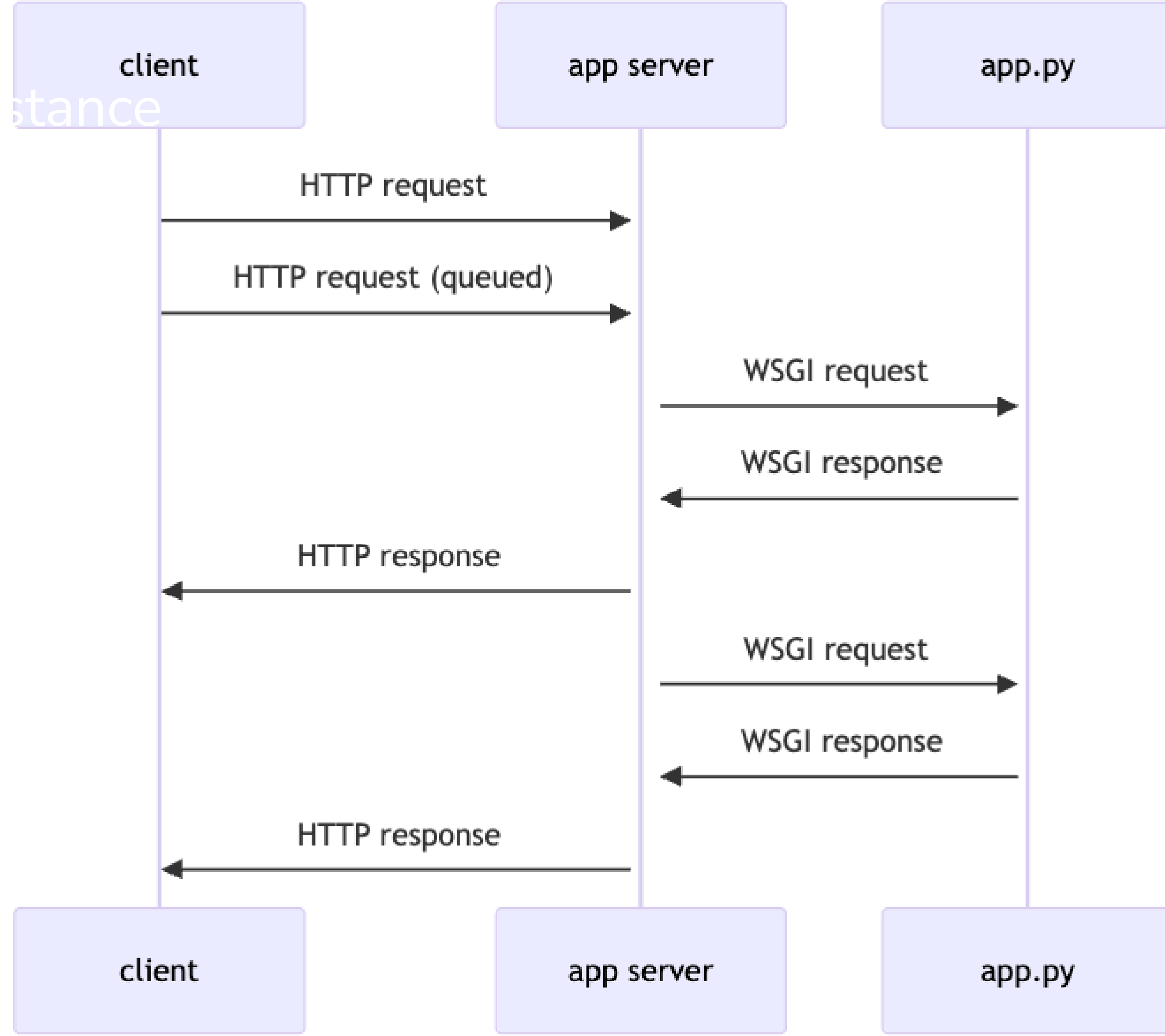
- Logging / Alerting
- Redundancy
- Containerization
- Deployment pipeline

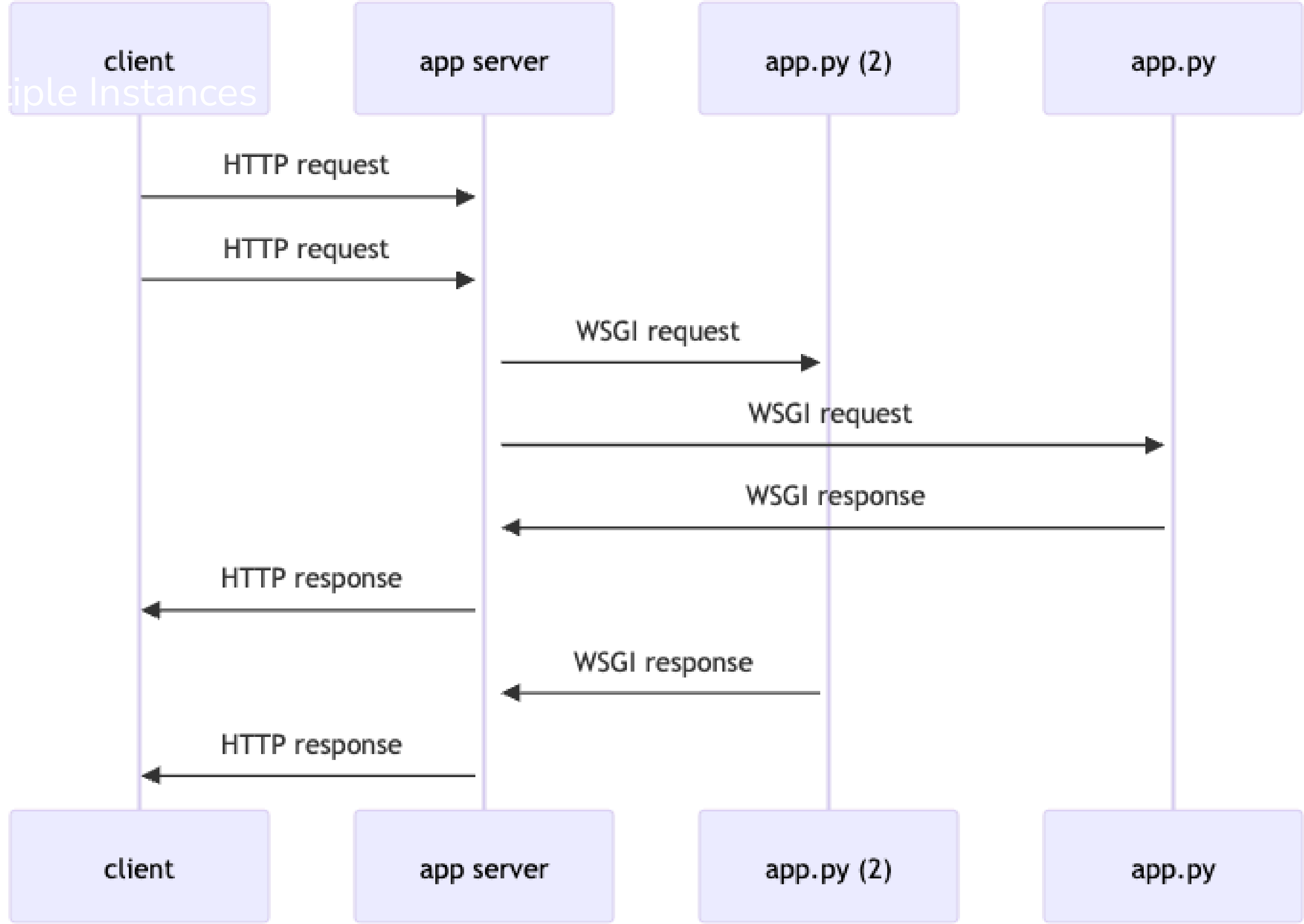
Application Server

Handles/queues HTTP requests and exchanges them with app server.

- gunicorn
- uwsgi
- uvicorn (async web apps)

Can run multiple copies to be able to serve more in parallel.


client
app server
app.py
stance
HTTP request
HTTP request (queued)
WSGI request
WSGI response
HTTP response
WSGI request
WSGI response
HTTP response
client
app server
app.py


client
app server
app.py (2)
app.py
tiple Instances
HTTP request
HTTP request
WSGI request
WSGI request
WSGI response
HTTP response
WSGI response
HTTP response
client
app server
app.py (2)
app.py

Forward Proxy / Web Server

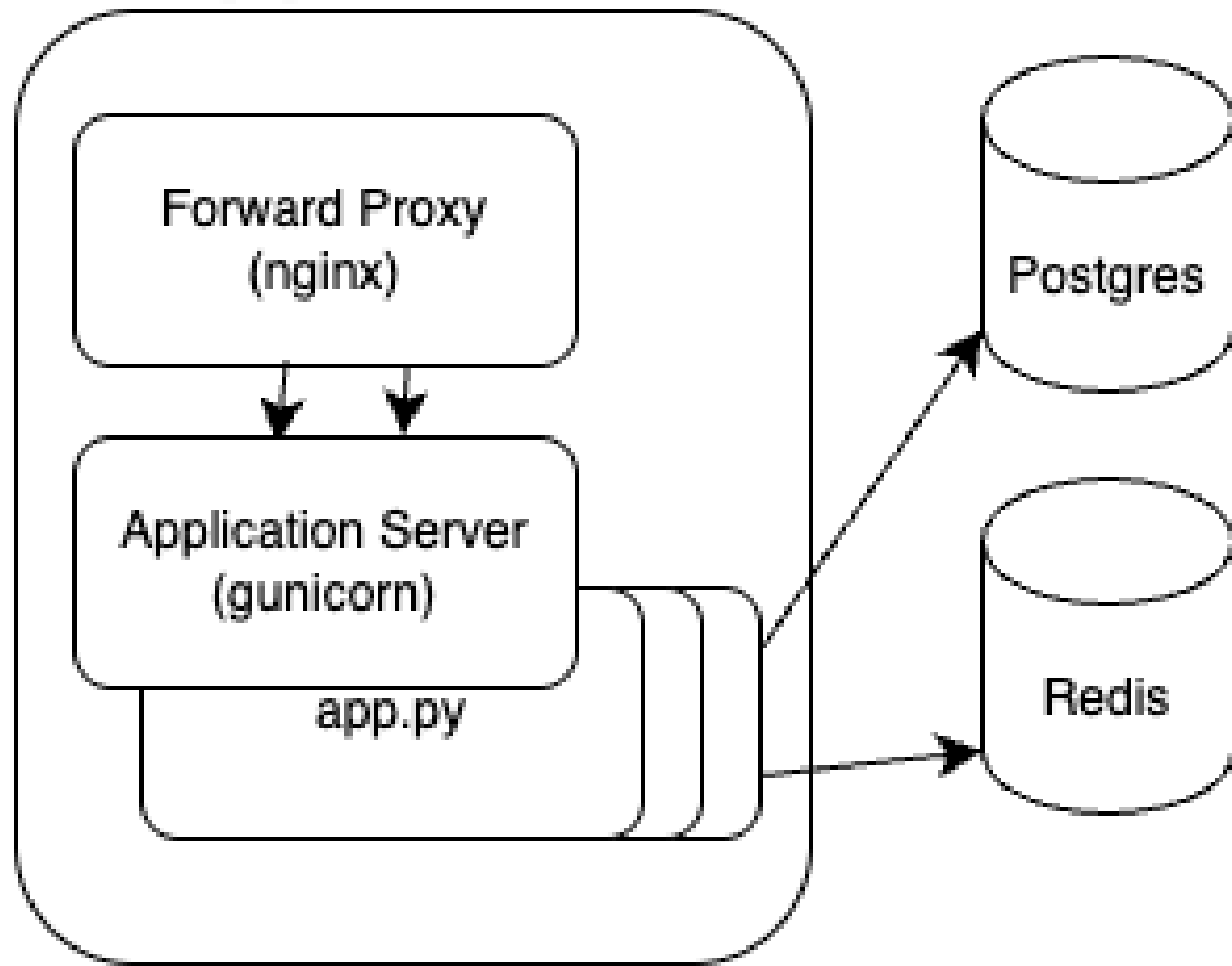
Sits in front of application server.

Handles HTTPS, caching, etc. and dispatches requests to application server(s).

Can also be used to host more than one app on same server.

- Caddy
- nginx
- Apache

app server



Staging Environments

Typically as close to production as costs will allow.

Should generally not be connected to prod infrastructure *at all*: separate DB, queue, cache, app server, etc.

Differences in architecture between staging/production can lead to bugs being missed; false confidence.

Often smaller server(s), significantly less traffic.

Exceedingly common to forgo and use local containers: advantage of true staging is ability to redirect QA/some traffic to test site.

What else do we need?

SSL

Adds layer of encryption (httpS) on top of HTTP protocol.

Should be used for all production applications.

Never Pay for SSL

Many services/servers provide certificates automatically, and if not, look for instructions on configuring on https://letsencrypt.org

Nonprofit run by Internet Security Research Group, provides services to 600 million websites.

Databases, Caches, Queues

Typically hosted on separate machine(s), with many cloud offerings available.

- sqlite3 - Local database, best for apps with low write volume.
- Postgres - Powerful scalable & extensible database.
- Redis - Key-value store, cache.
- ElasticSearch - Full text search & document database.
- RabbitMQ - High-throughput queue.
- Kafka - Event streaming.

Logging

Important for usage monitoring and debugging.

Disk usage is a surprisingly common failure point, be sure to limit/rotate logs!

- Text files.
- Text files w/ rotation.
- Log Collector
- ELK stack: Elastic; Logstash; Kibana.

Observability/Alerting

An external service, often intentionally on *someone else's infrastructure*.

These tools install hooks that monitor your code in real time. Comes at small performance cost, but can be invaluable for larger systems.

Can aggregate logs, and get real-time observability of application request timing.

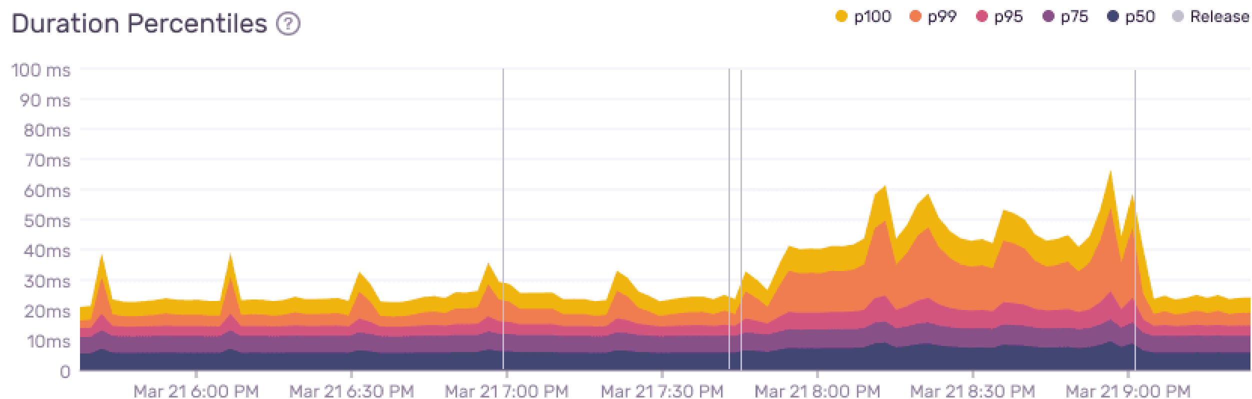
e.g. Time spent in each layer (cache; forward proxy; app server; app; DB)

- Sentry
- NewRelic
- Datadog

Performance > Transaction Summary

★ Key Transaction

/api/0/projects/ep/login_page



Total Events 1.2m

Display Percentiles **Y-Axis** ms

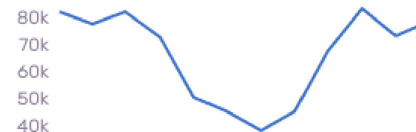
Apdex Score

0.9

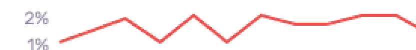
User Misery

Apdex

Throughput



Failure Rate



Filter: Slowest Transactions

Open in Discover

ID	USER.EMAIL	DURATION	TIMESTAMP
64589156756	AP aaron@datatechnology.com	190 ms	Jun 12, 2020 2:39:12 AM UTC
94587656754	AC a.carlsson126@sweden.gov	180 ms	Jun 12, 2020 2:39:11 AM UTC
12392123523	AD adiaz@usapo.com	176 ms	Jun 12, 2020 2:39:10 AM UTC

Tag S

brov

SENTRY

Redundancy & Scalability

We can scale the application by increasing the resources available to the machine (vertical scaling).

More commonly today, we scale by running additional machines with more copies of the software. (horizontal scaling).

Setup & Deployment

Infrastucture as Code

Motivation

Setup of the environment is a set of steps that need to be followed precisely.

Log into server and:

1. Install required dependencies (Python, uv, DB, etc.)
2. Check out application code.
3. Write all config files (Caddy, uwsgi, application)
4. Ensure server is secure (firewall, users, network)

Easy to miss a step/make mistakes.

Now we need a development server & production, or need to scale…

Declarative Programming

Could write each step/call functions:

```
install_dependencies(["python3", "python-uv", "postgres"])
check_out("git@github.com/ ... / ... ")
write_config("/etc/caddy/Caddyfile", "production-Caddyfile")
write_config("/etc/uwsgi/apps-enabled/app.ini", "production-app.ini")
```

When we need to add a step, run it all over again? just that step?

Declarative Programming

Key idea: **describe end state of system, not steps to get there**

Let a program figure out which steps need to be run to correct state.

```
server = Server("python3")
server.repo("git@github.com/ ... ")
server.config("Caddyfile", ... )
server.config("/etc/uwsgi/apps-enabled/app.ini", ... )

# adding a new dependency
server = Server("python3", "python-uv")
```

on next run, difference will be detected, and "install python-uv" will be run

Declarative Infastructure Tools

Ansible for setting up individual servers. Can manage every dependency on the server.

- `ansible deploy` configures a machine to have a specific set of packages and configurations.

Terraform for managing connections/relationships between servers and cloud infrastructure.

- Create/name/link AWS Instances.
- `terraform deploy` creates/removes machines from infrastructure.

Containerization

An alternative to `ansible`-style machine configuration is to run individual services in containers.

Machines only run `docker` (or `podman` or `kubernetes`) directly.

Each application is *containerized*. Set up as if it has a server all to itself!

Dockerfile

```
# base image with Python installed
FROM python:3.13

# run commands on server
RUN apt-get update && apt-get install -y --no-install-recommends \
    gcc \
    build-essential \
    && rm -rf /var/lib/apt/lists/*
RUN git clone git@github.com/ ...
RUN uv install
COPY uwsgi.ini /opt/uwsgi.ini

# expose a port to host machine
EXPOSE 8000

# command to run application
CMD ["uv", "run", "uwsgi", "--ini", "/opt/uwsgi.ini"]`
```

docker-compose.yml

```yaml
services:
  web:
    build: .
    ports:
      - "8000:8000"
    volumes:
      - .:/app
    restart: always
    environment:
      - FLASK_ENV=production
    depends_on:
      - db

  db:
    image: postgres:13
    volumes:
      - postgres_data:/var/lib/postgresql/data
    environment:
      - POSTGRES_USER=myuser
      - POSTGRES_PASSWORD=mypassword
      - POSTGRES_DB=myapp
    ports:
      - "5432:5432"
volumes:
  postgres_data:
```

Containerization Trade-Offs

Applications get pristine environment with no interference from other apps.

Harder for security issues in one application to affect rest of system.

Can ensure identical environments on dev, staging, prod, local machine.

Another layer of indirection with complexity and performance overhead.

Ephemeral services, cannot "log in and tinker/observe". Important to have logging/etc. in place.

Deployment Pipeline

Whatever the steps are, when & how do they run?

Continous Deployment

Any commit on `main` triggers a rebuild of the server via `Docker` / `ansible`.

ChatOps

A bot that lives in your team Slack that can respond to commands like 'deploy branch feature/new-login to staging'

GitOps

Various infrastructure tasks are performed by committing/tagging.

Examples:

- push to branch `demo/xyz` starts up `demo-xyz.app.example.com` so that team can see what is on that branch.
- new commits/tags on `main` create automated releases and notify team via Slack

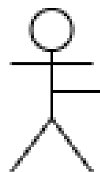
Deployment Strategies

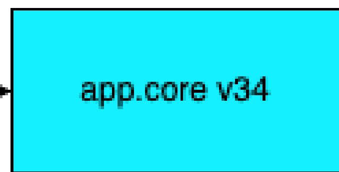
Deployment usually means a bit of downtime, and we may need to **roll back** if there are bugs or performance issues.

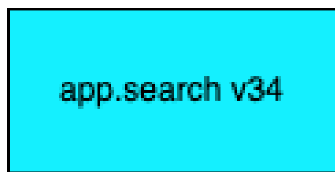
For small apps, this is fine, just redeploy and update, typically at a low-traffic time.

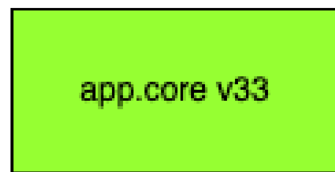
Don’t deploy on a Friday afternoon!

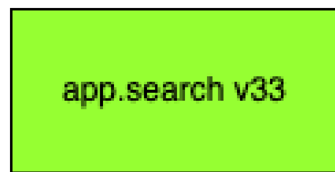
Alternative: blue/green deployment.

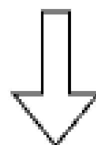


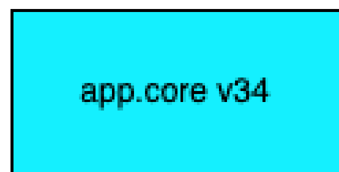


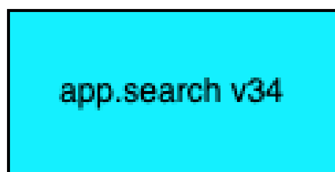


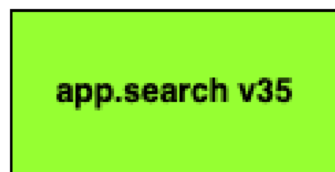


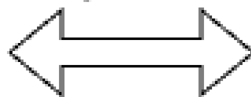


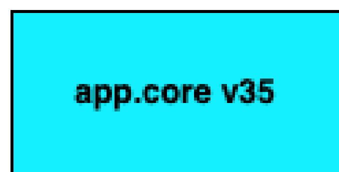


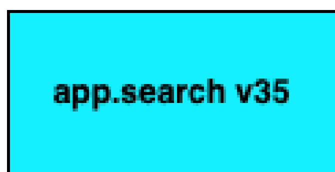


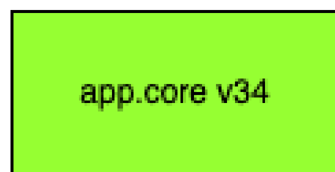


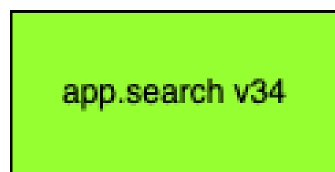












4. If issues are discovered, a rollback is simply swapping back.