SCALING
DJANGO

SCALING DJANGO WITH AMAZON WEB SERVICES

Kenneth Falck <kennu@iki.fi>

DjangoCon Finland 7.5.2011

ABOUT ME

Started with Internet apps ~1994
First C/C++, some Java/ASP, then PHP, nowadays Python and Ruby

R&D Manager at Sanoma Entertainment
(Brands: Nelonen, Ruutu.fi, Pelikone.fi, Älypää, Liigapörssi, etc.)

Interests: Scalability, NoSQL, Post-PC
(and a bunch of other stuff)

Blog at: https://kfalck.net

TOPICS

1. Introduction
2. AWS overview
3. Django apps on AWS
4. Auto-scaling
5. Load-balancing
6. Django sessions
7. Memcached
8. S3 for uploads, static files
9. RDS, multi-db sharding
10. MongoDB, SimpleDB
11. Stateless vs. Stateful
12. Email and SES
13. Boto, settings.py
14. Fabric, Puppet, AMIs
15. Logging, monitoring
16. Further reading

DISCLAIMER

- These slides are not based on production experience yet
    - We have developed Django based AWS services.
    - We have load-tested Django based AWS services.
    - We have NOT run them in production yet.
- So please consider this an introduction to AWS
    - You can always google further information about each topic.

INTRODUCTION

- Python, Django and AWS work very nicely together
  - Django doesn't require any special tweaking to use with AWS.
  - There are PyPI packages to integrate Django to Amazon S3 and RDS.
  - Boto (AWS API for Python) can be used to automate everything.
- Amazon has many scalability options for Django
  - ELB & AS (elastic load balancing & auto scaling).
  - RDS (MySQL hosted by Amazon).
  - SimpleDB (work has just started on django-nonrel).
  - MongoDB (run your own on EC2).
- AWS is easy to try out and play with
  - Just sign up with your credit card, and you have infinite cloud resources.

amazon webservices™

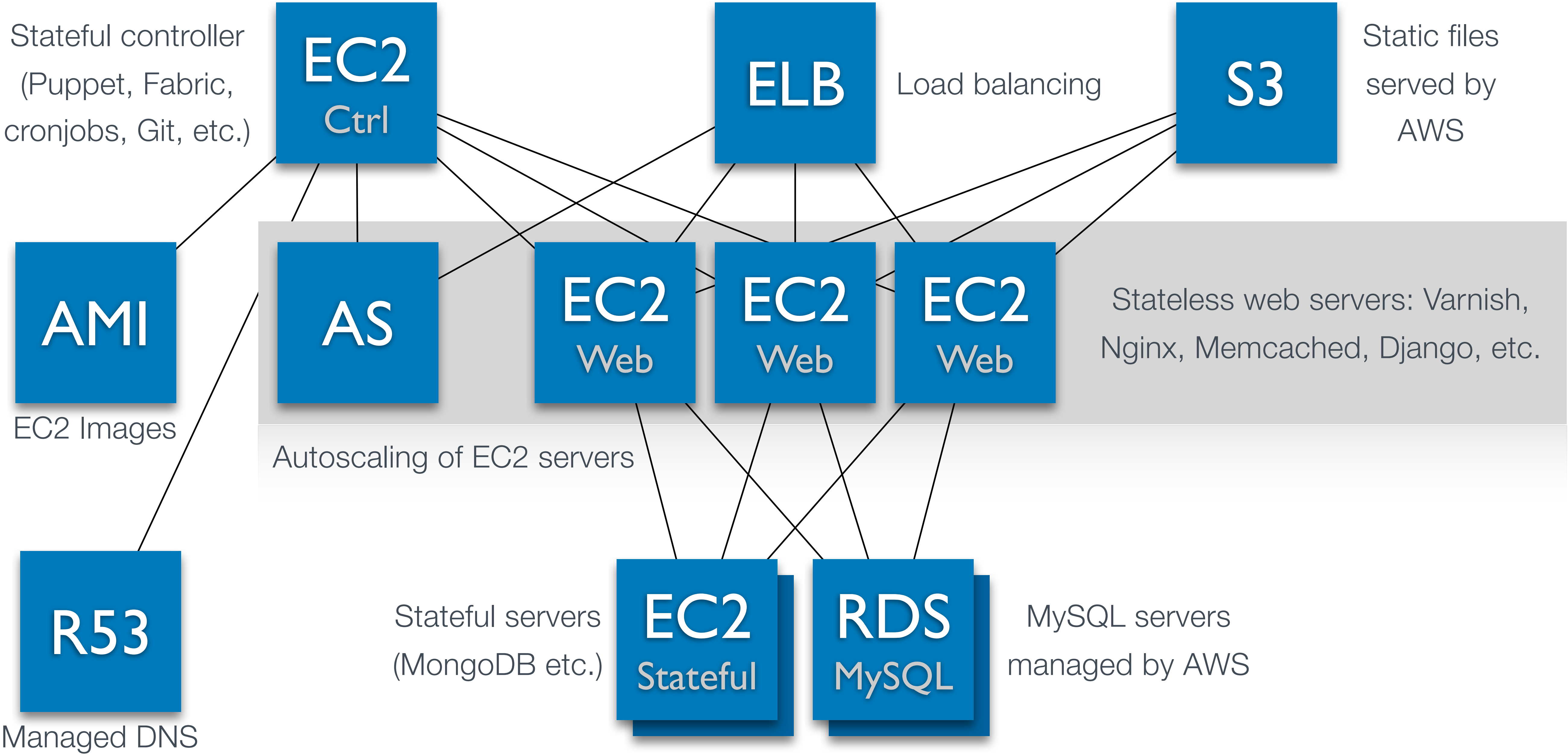
AWS OVERVIEW

- AWS is a rich collection of cloud infrastructure services
    - **EC2** - Virtual machines that run any OS (use Ubuntu).
    - **EBS** - Reliable block storage that appears as /dev/xvdX.
    - **RDS** - MySQL as a hosted “black box” service.
    - **S3** - Web-based file storage for CSS/JavaScript/images.
    - **Route 53** - Managed DNS
    - etc.
- It also provides many utility functions
    - Load balancing between EC2 instances.
    - Automatic scaling & availability for EC2 instances.
    - Security (firewalls), user management, ssh key management.
    - etc.

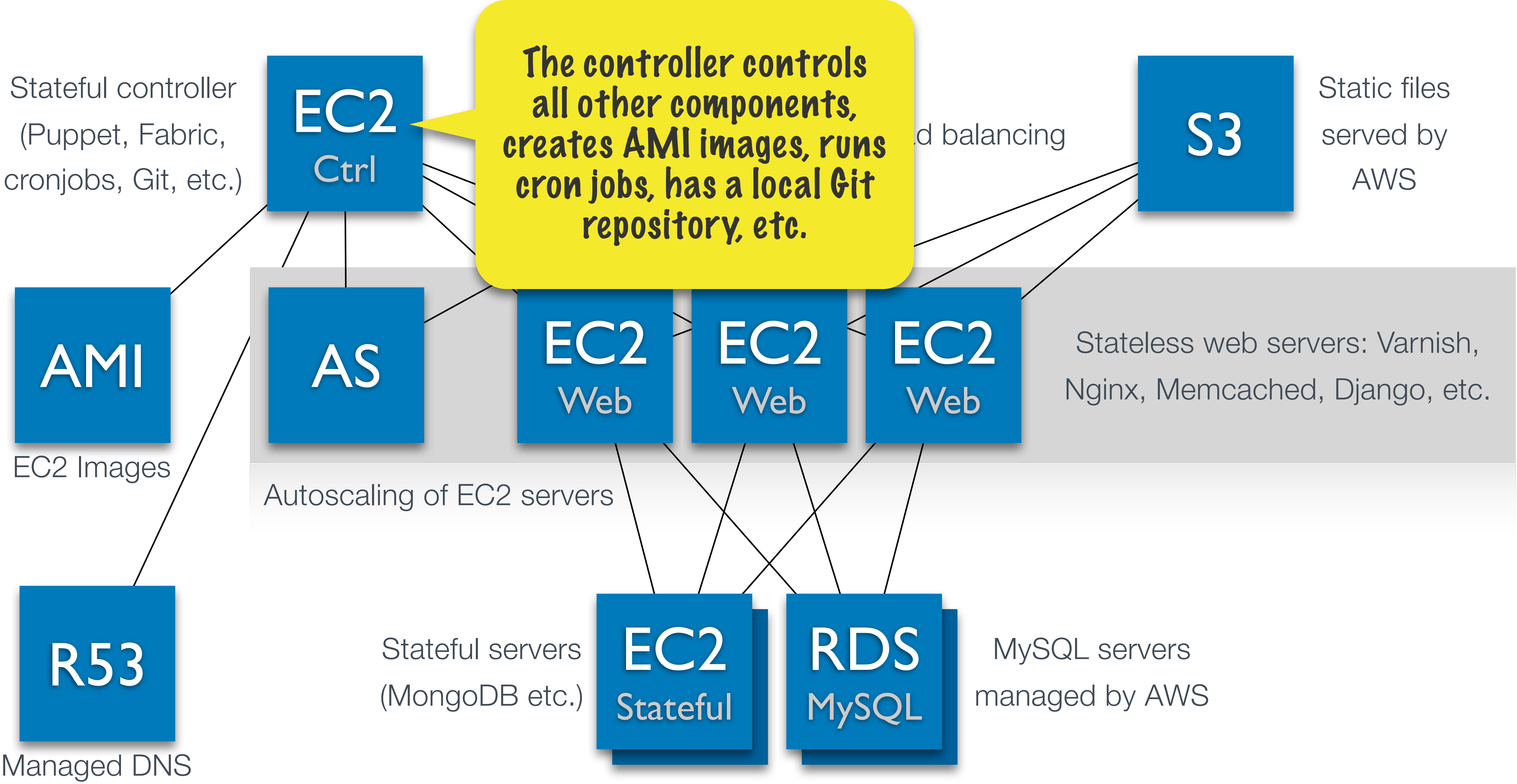
THE AWS APPROACH

- Put everything possible in Amazon’s managed services
  - AWS takes care of scaling and availability of S3, Route 53, SimpleDB, etc.
  - Compare to Google App Engine.
- Put the rest in your own EC2 instances
  - Freedom to decide which HTTP server stack you run.
  - Choose whatever framework for your app.
  - Install any additional packages you need.
  - But you have to make it scalable yourself, using AWS’s tools.
- Cost structure similar to VPS servers (Linode, etc)
  - Pay for N virtual machines, X storage, Y bandwidth.
  - But more flexibility.

GENERIC AWS ARCHITECTURE



GENERIC AWS ARCHITECTURE



The controller controls all other components, creates AMI images, runs cron jobs, has a local Git repository, etc.

Stateful controller (Puppet, Fabric, cronjobs, Git, etc.)

EC2
Ctrl

d balancing

S3

Static files served by AWS

AMI

EC2 Images

AS

EC2
Web

EC2
Web

EC2
Web

Stateless web servers: Varnish, Nginx, Memcached, Django, etc.

Autoscaling of EC2 servers

R53

Managed DNS

Stateful servers (MongoDB etc.)

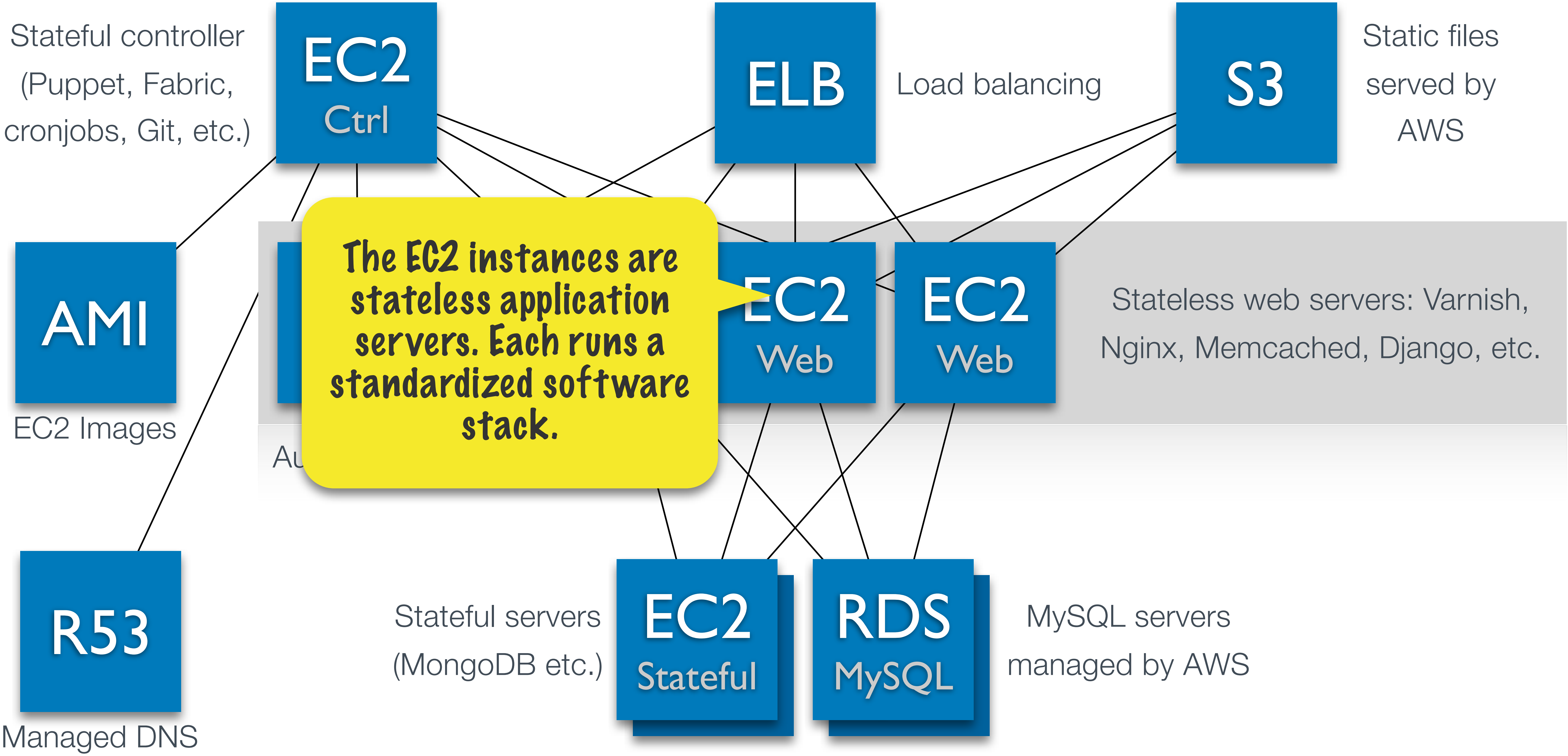
EC2
Stateful

RDS
MySQL

MySQL servers managed by AWS

SCALING
DJANGO

GENERIC AWS ARCHITECTURE



Stateful controller (Puppet, Fabric, cronjobs, Git, etc.)

Load balancing

Static files served by AWS

EC2 Images

Stateless web servers: Varnish, Nginx, Memcached, Django, etc.

Au

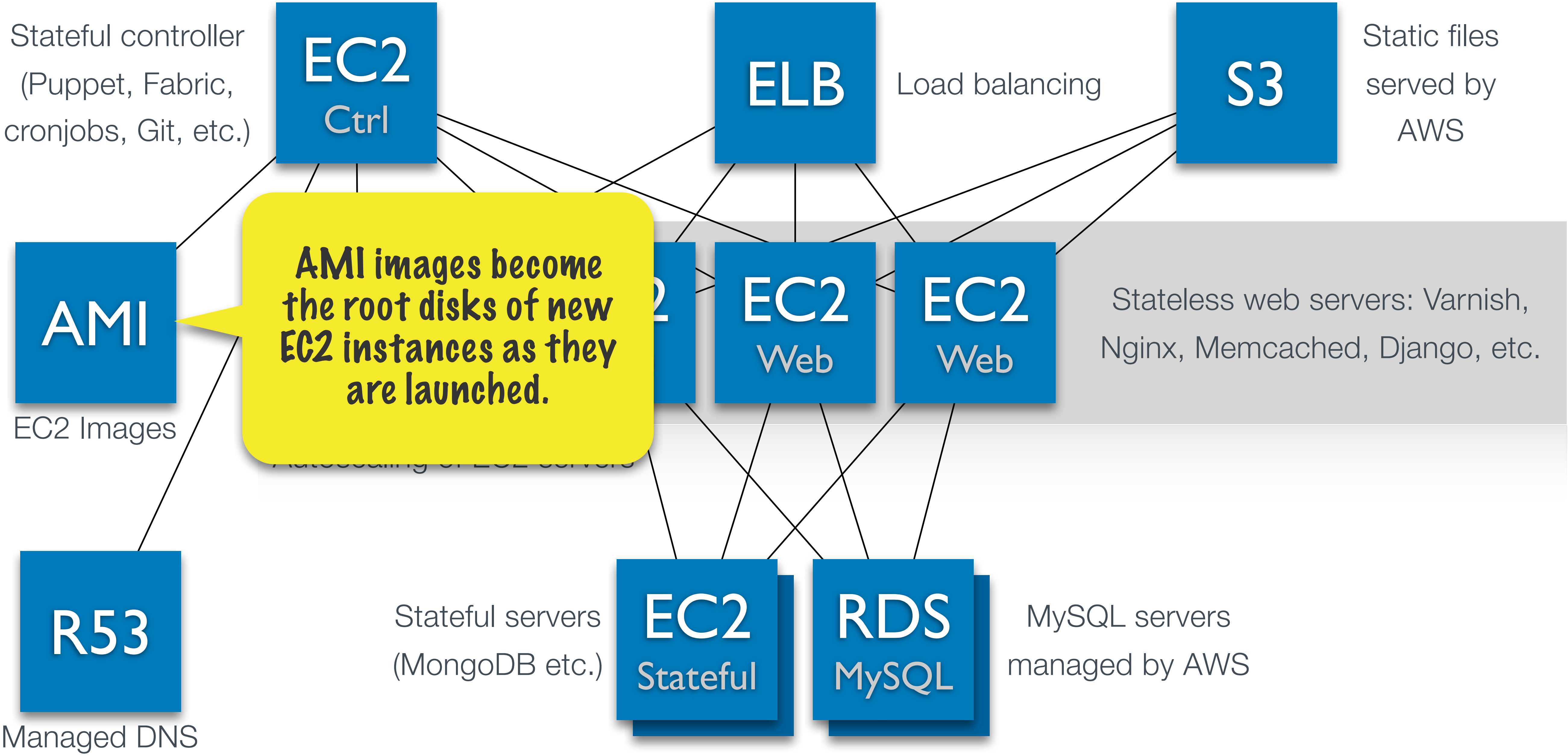
Managed DNS

Stateful servers (MongoDB etc.)

MySQL servers managed by AWS

The EC2 instances are stateless application servers. Each runs a standardized software stack.

GENERIC AWS ARCHITECTURE



Stateful controller (Puppet, Fabric, cronjobs, Git, etc.)

EC2 Ctrl

ELB

Load balancing

S3

Static files served by AWS

AMI

EC2 Images

AMI images become the root disks of new EC2 instances as they are launched.

2

EC2 Web

EC2 Web

Stateless web servers: Varnish, Nginx, Memcached, Django, etc.

Autoscaling of EC2 servers

R53

Managed DNS

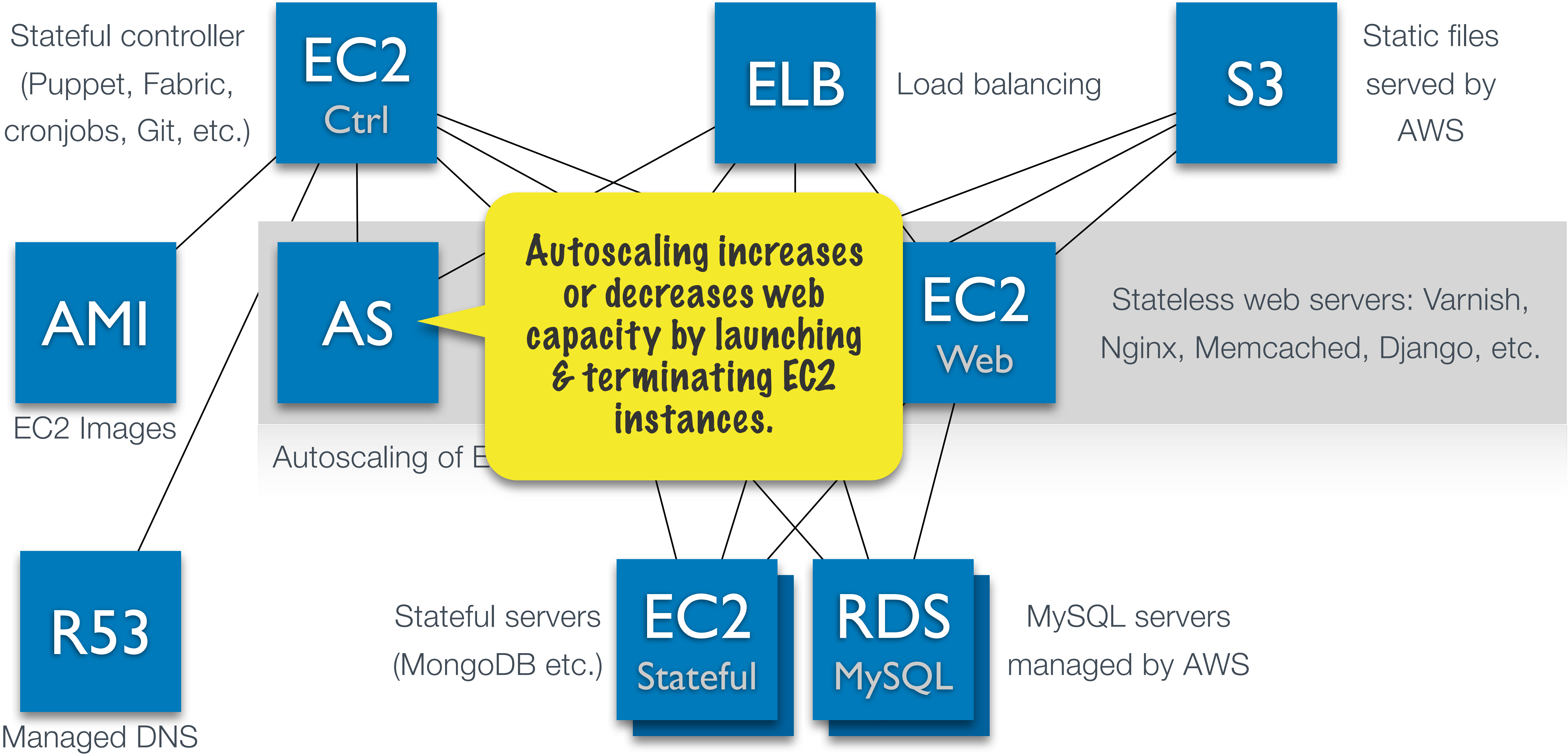
Stateful servers (MongoDB etc.)

EC2 Stateful

RDS MySQL

MySQL servers managed by AWS

GENERIC AWS ARCHITECTURE



EC2 Ctrl

Stateful controller (Puppet, Fabric, cronjobs, Git, etc.)

ELB

Load balancing

S3

Static files served by AWS

AMI

EC2 Images

AS

EC2 Web

Stateless web servers: Varnish, Nginx, Memcached, Django, etc.

Autoscaling of E

R53

Managed DNS

EC2 Stateful

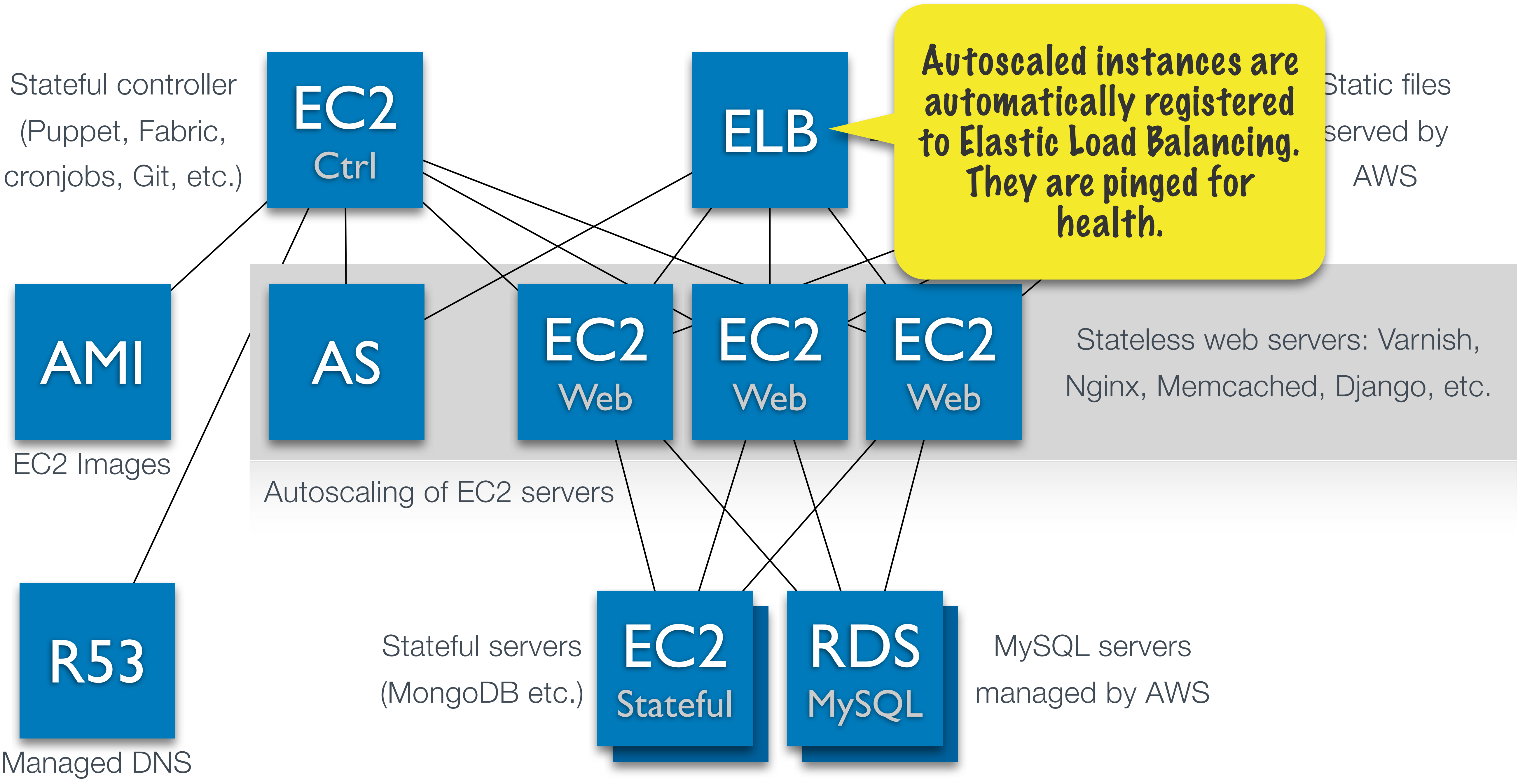
Stateful servers (MongoDB etc.)

RDS MySQL

MySQL servers managed by AWS

Autoscaling increases or decreases web capacity by launching & terminating EC2 instances.

GENERIC AWS ARCHITECTURE



Stateful controller (Puppet, Fabric, cronjobs, Git, etc.)

Autoscaled instances are automatically registered to Elastic Load Balancing. They are pinged for health.

Static files served by AWS

EC2 Images

Stateless web servers: Varnish, Nginx, Memcached, Django, etc.

Autoscaling of EC2 servers

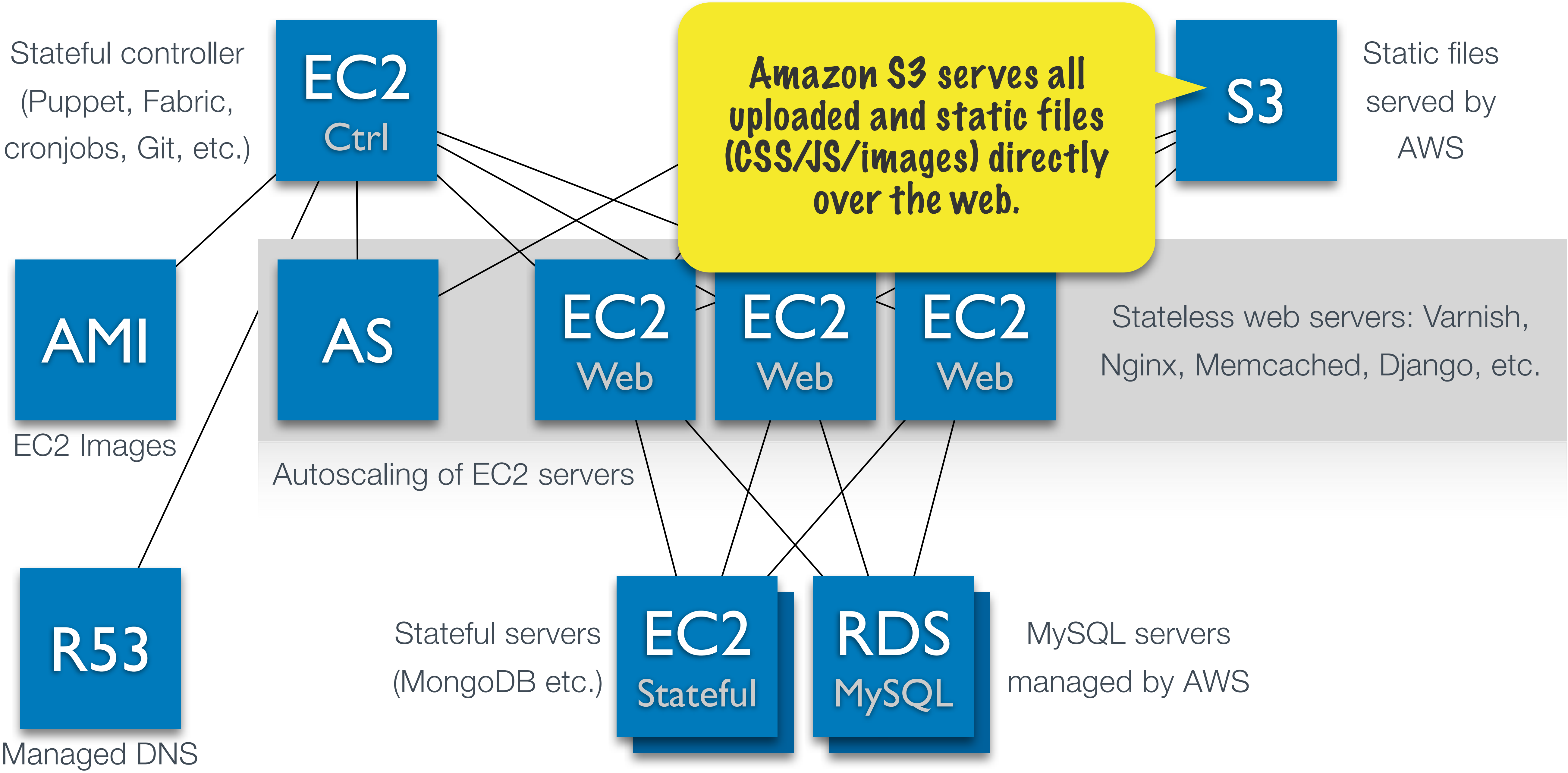
Managed DNS

Stateful servers (MongoDB etc.)

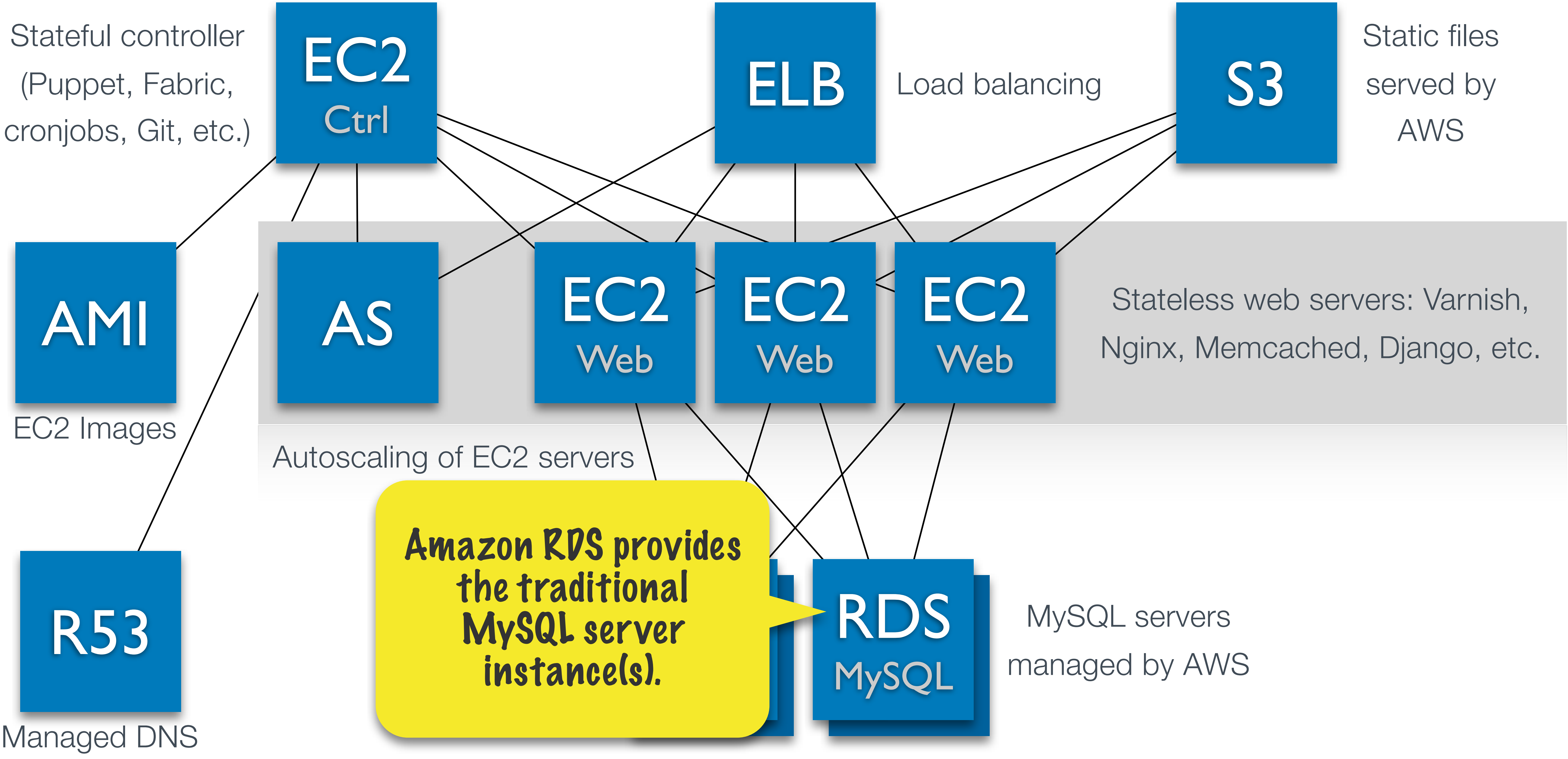
MySQL servers managed by AWS

SCALING
DJANGO

GENERIC AWS ARCHITECTURE



GENERIC AWS ARCHITECTURE



Stateful controller (Puppet, Fabric, cronjobs, Git, etc.)

Load balancing

Static files served by AWS

EC2 Images

Stateless web servers: Varnish, Nginx, Memcached, Django, etc.

Autoscaling of EC2 servers

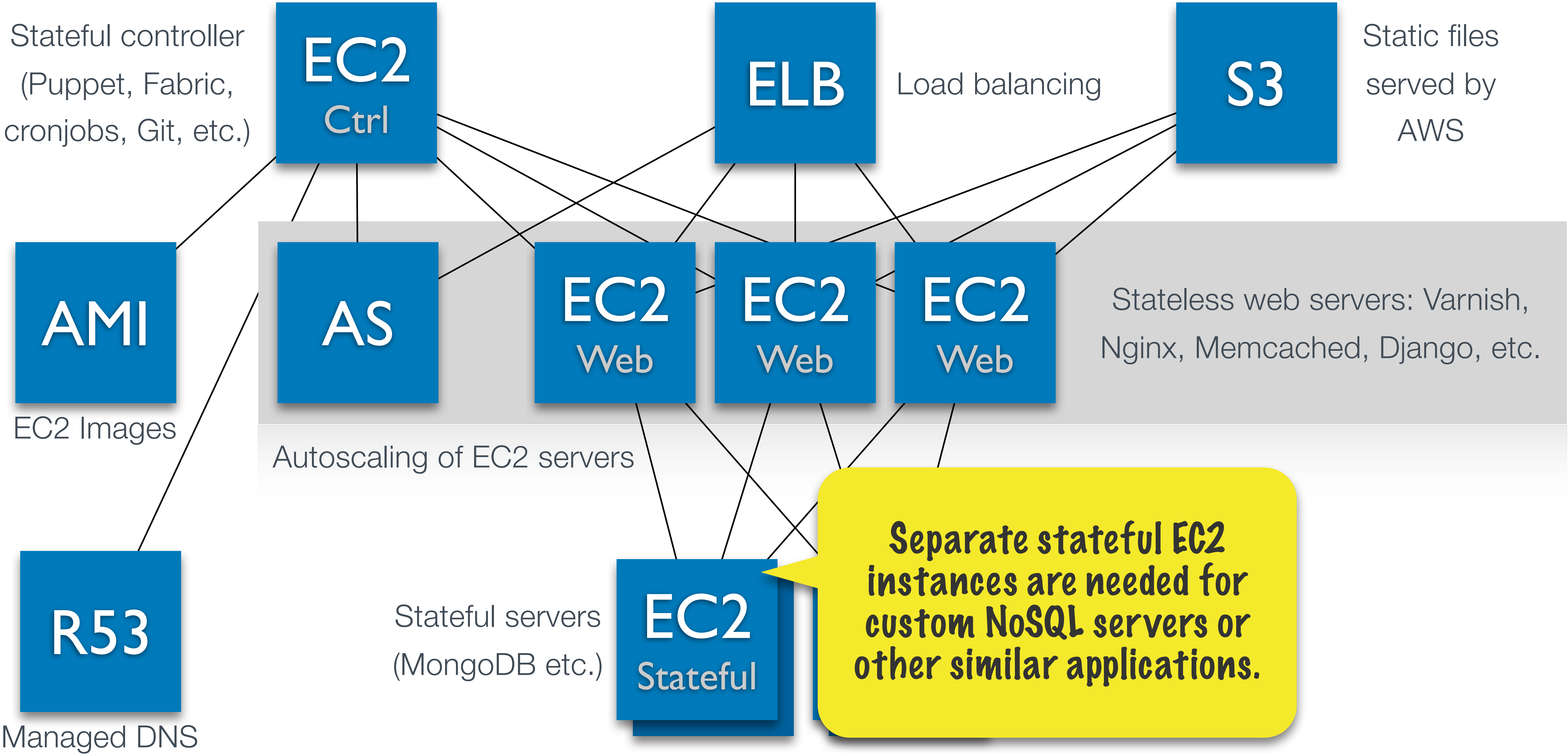
Managed DNS

MySQL servers managed by AWS

Amazon RDS provides the traditional MySQL server instance(s).

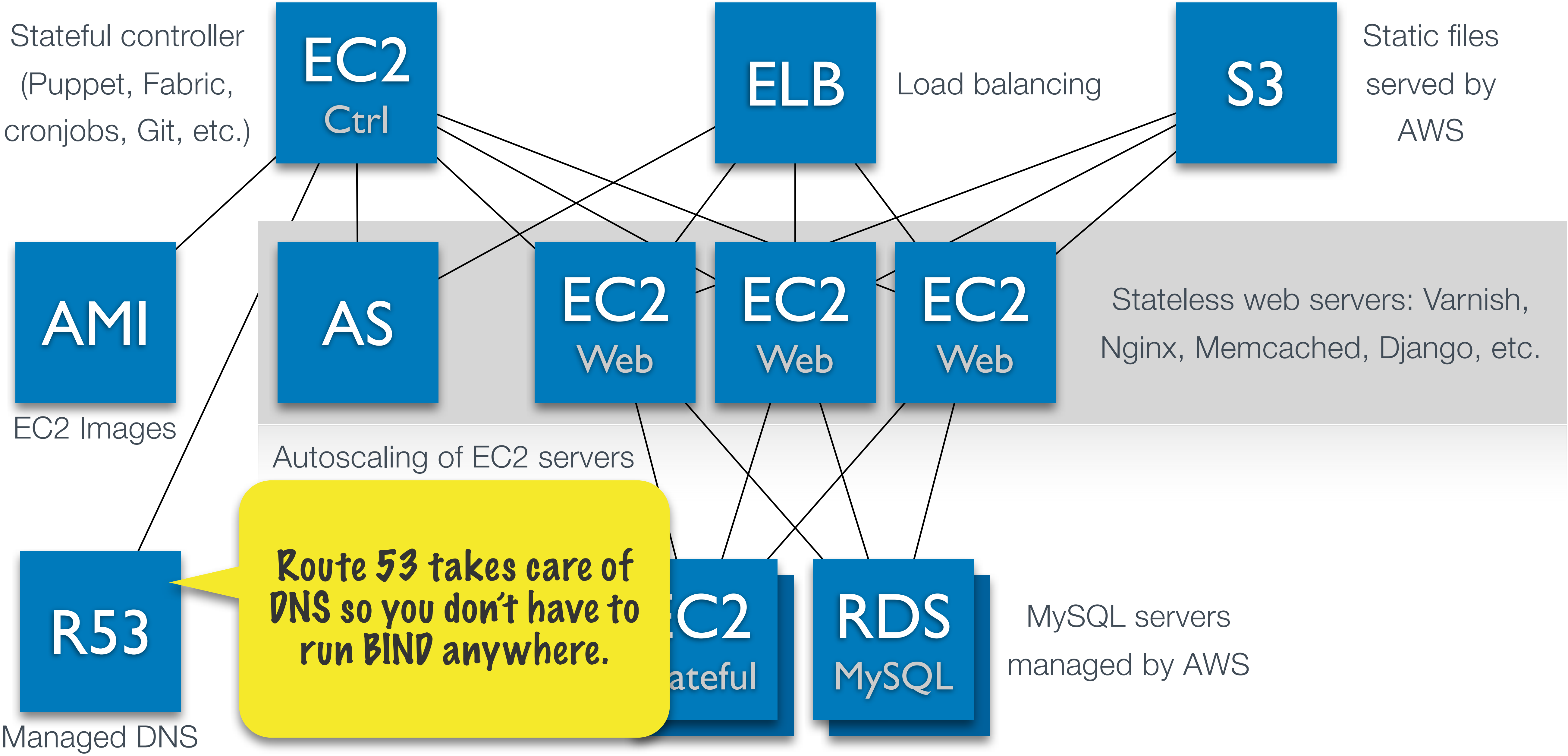
SCALING
DJANGO

GENERIC AWS ARCHITECTURE


Stateful controller (Puppet, Fabric, cronjobs, Git, etc.)
EC2 Ctrl
ELB
Load balancing
S3
Static files served by AWS
AMI
EC2 Images
AS
EC2 Web
EC2 Web
EC2 Web
Stateless web servers: Varnish, Nginx, Memcached, Django, etc.
Autoscaling of EC2 servers
R53
Managed DNS
Stateful servers (MongoDB etc.)
EC2 Stateful
Separate stateful EC2 instances are needed for custom NoSQL servers or other similar applications.

SCALING
DJANGO

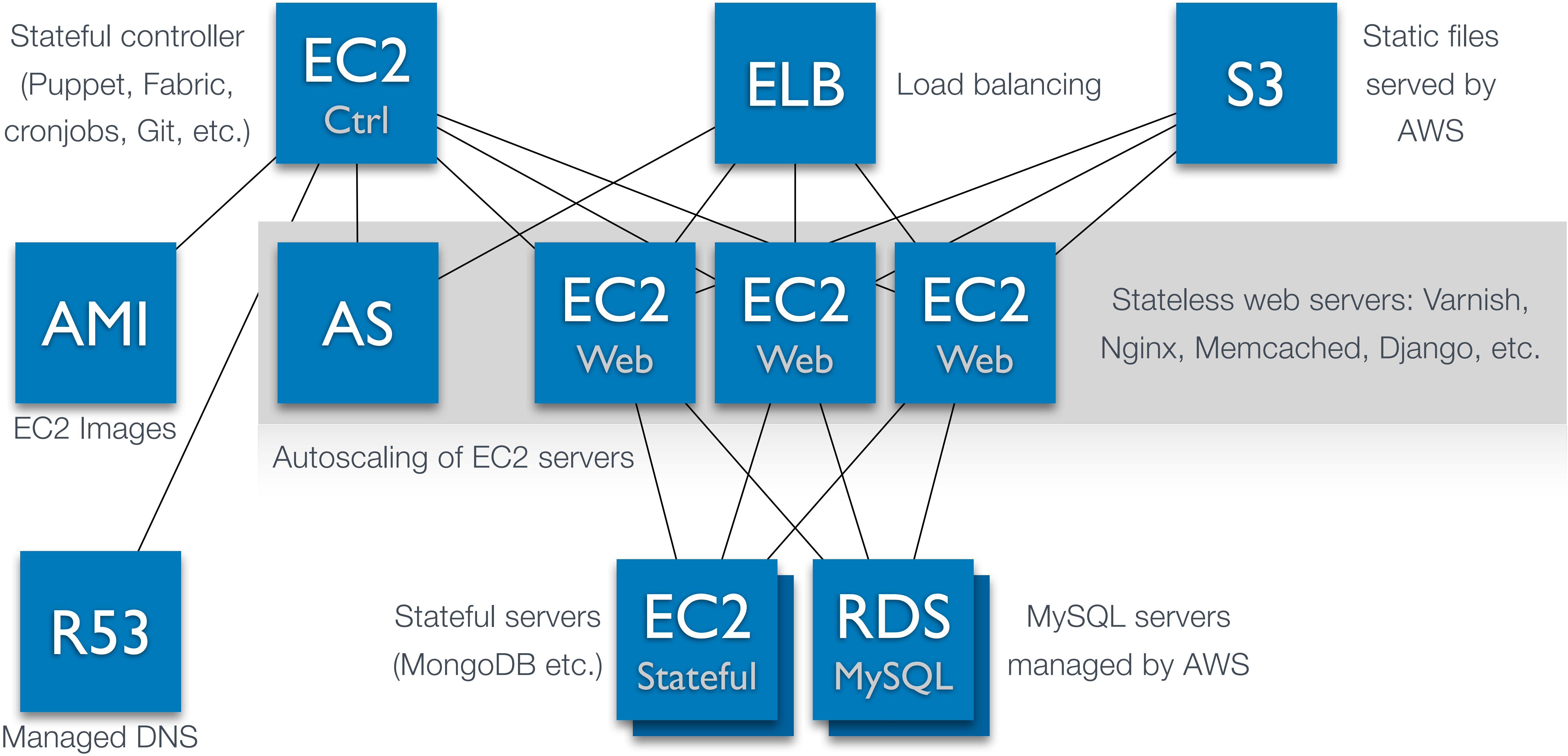
GENERIC AWS ARCHITECTURE


Stateful controller (Puppet, Fabric, cronjobs, Git, etc.)
EC2 Ctrl
ELB
Load balancing
S3
Static files served by AWS
AMI
EC2 Images
AS
EC2 Web
EC2 Web
EC2 Web
Stateless web servers: Varnish, Nginx, Memcached, Django, etc.
Autoscaling of EC2 servers
R53
Managed DNS
C2
ateful
RDS MySQL
MySQL servers managed by AWS

Route 53 takes care of DNS so you don't have to run BIND anywhere.

SCALING
DJANGO

GENERIC AWS ARCHITECTURE



EC2
Ctrl

Stateful controller
(Puppet, Fabric,
cronjobs, Git, etc.)

ELB

Load balancing

S3

Static files
served by
AWS

AMI

EC2 Images

AS

EC2
Web

EC2
Web

EC2
Web

Stateless web servers: Varnish,
Nginx, Memcached, Django, etc.

Autoscaling of EC2 servers

R53

Managed DNS

Stateful servers
(MongoDB etc.)

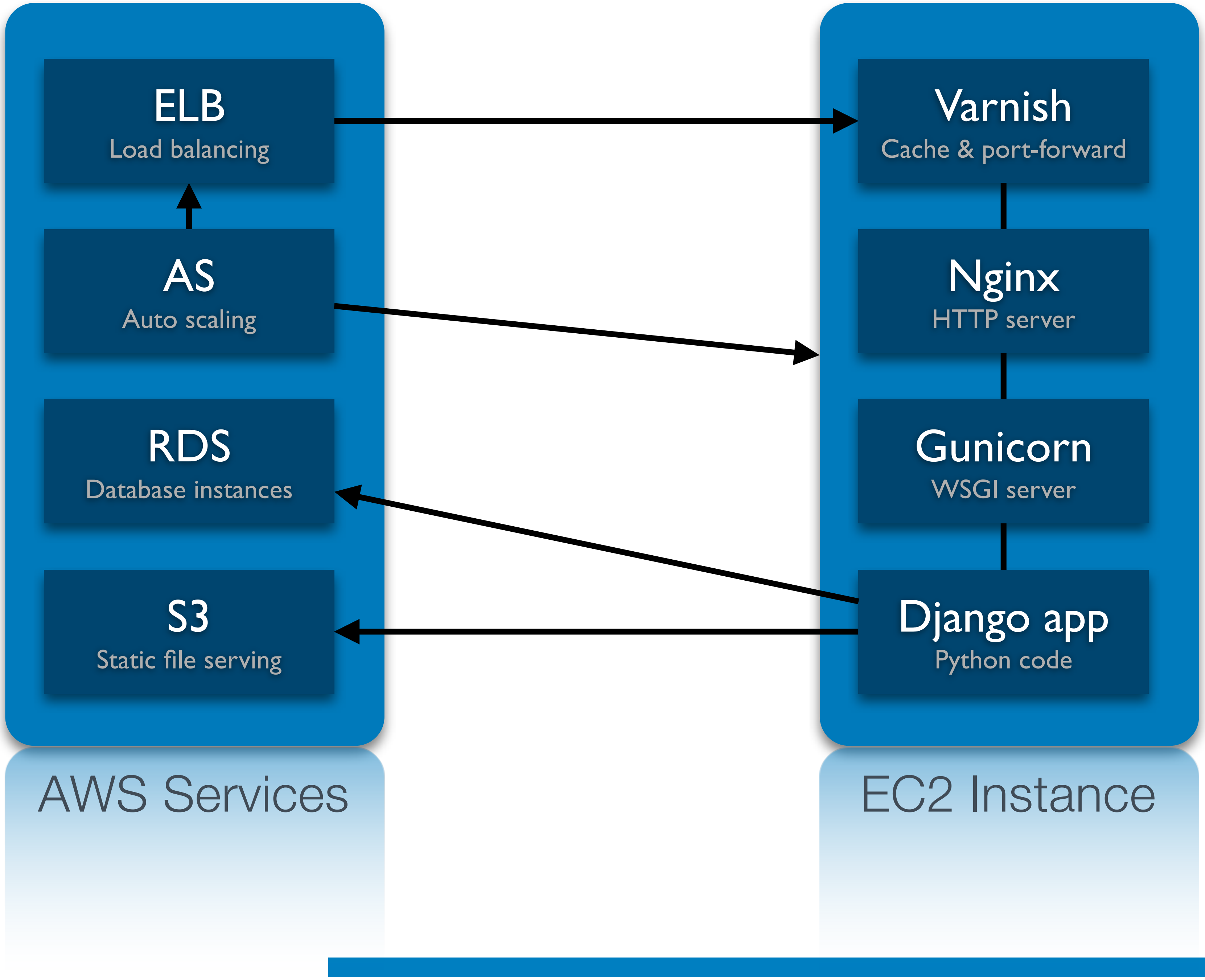
EC2
Stateful

RDS
MySQL

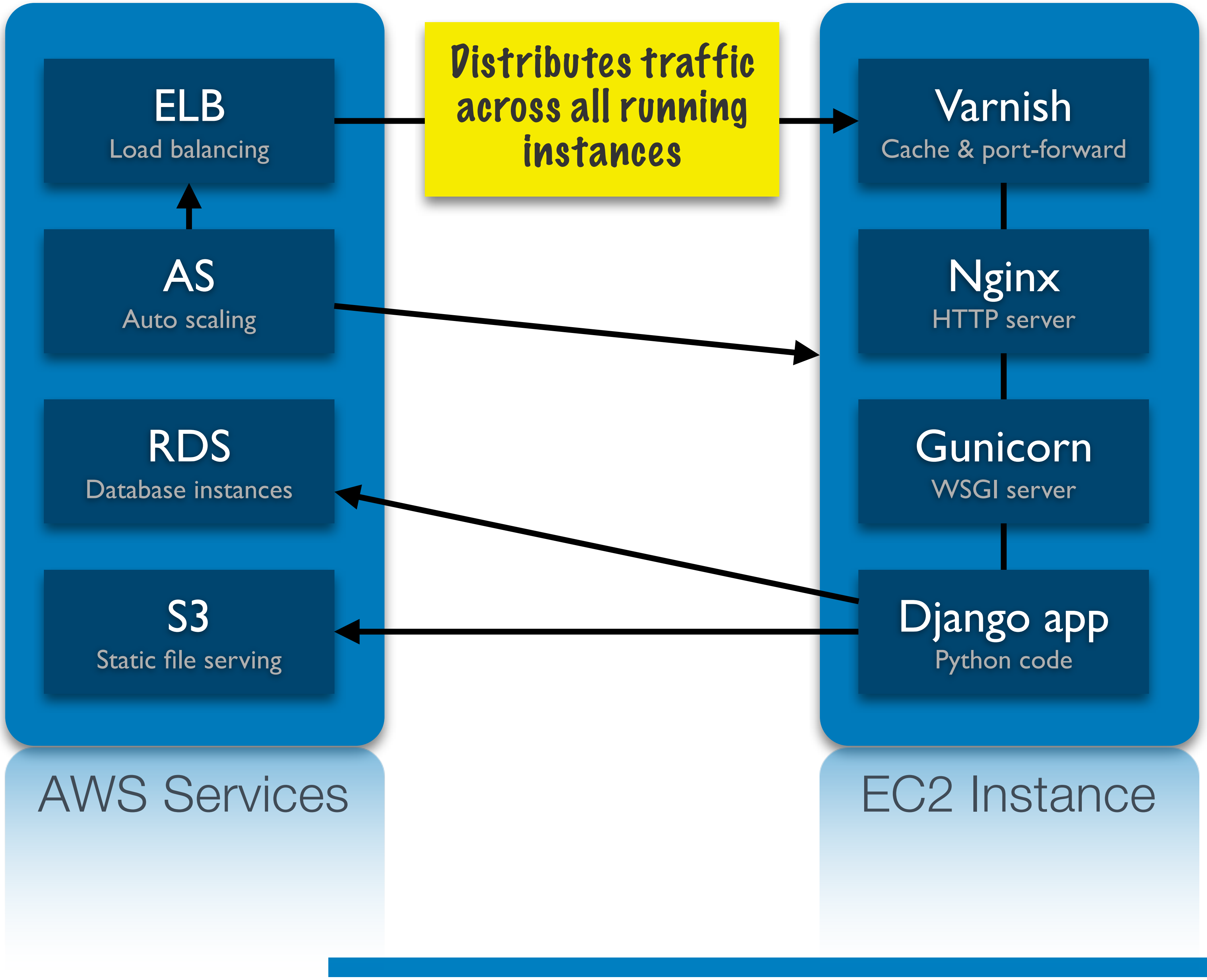
MySQL servers
managed by AWS

SCALING
DJANGO

DJANGO APPS ON AWS



DJANGO APPS ON AWS



Distributes traffic across all running instances

ELB
Load balancing

AS
Auto scaling

RDS
Database instances

S3
Static file serving

AWS Services

Varnish
Cache & port-forward

Nginx
HTTP server

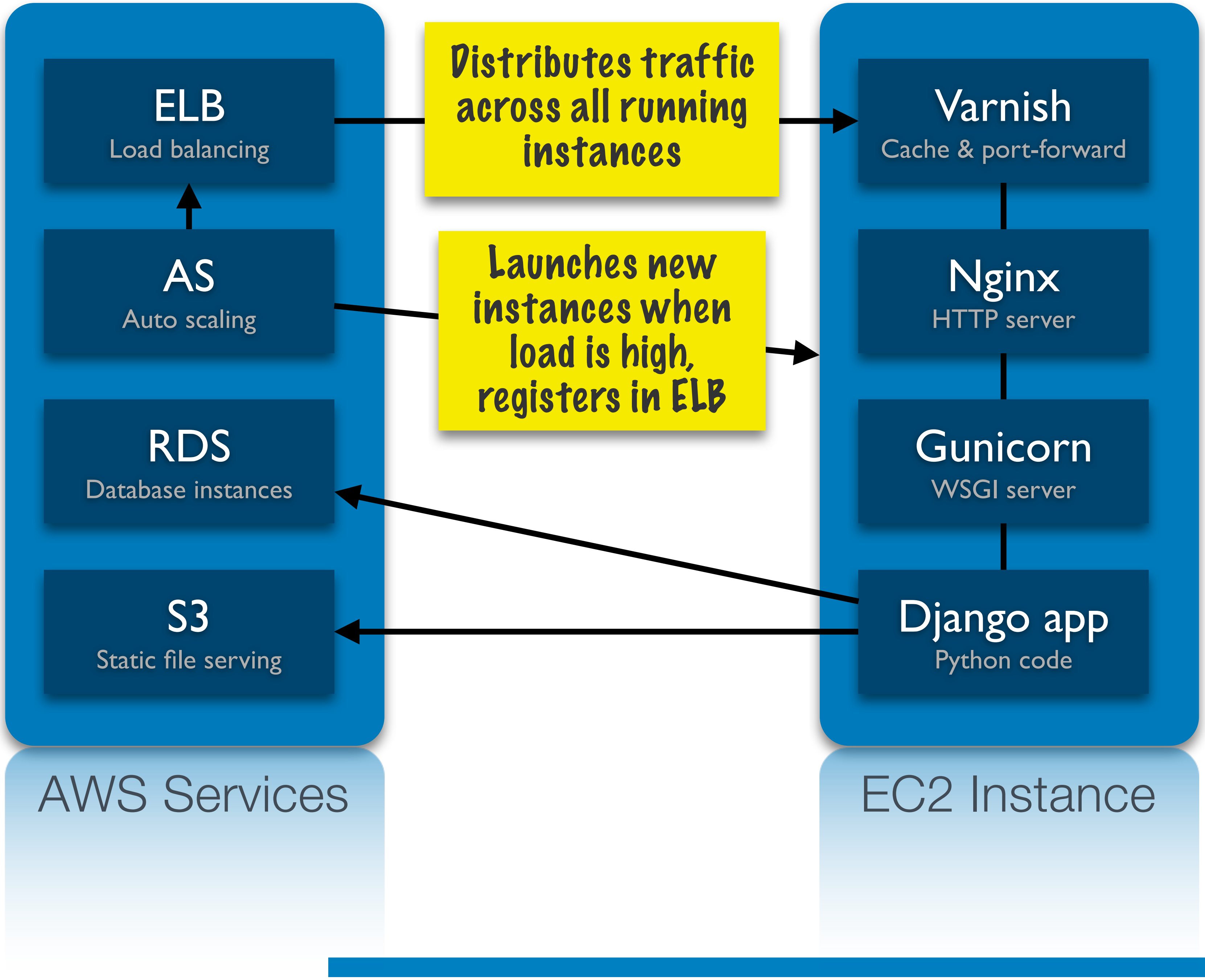
Gunicorn
WSGI server

Django app
Python code

EC2 Instance

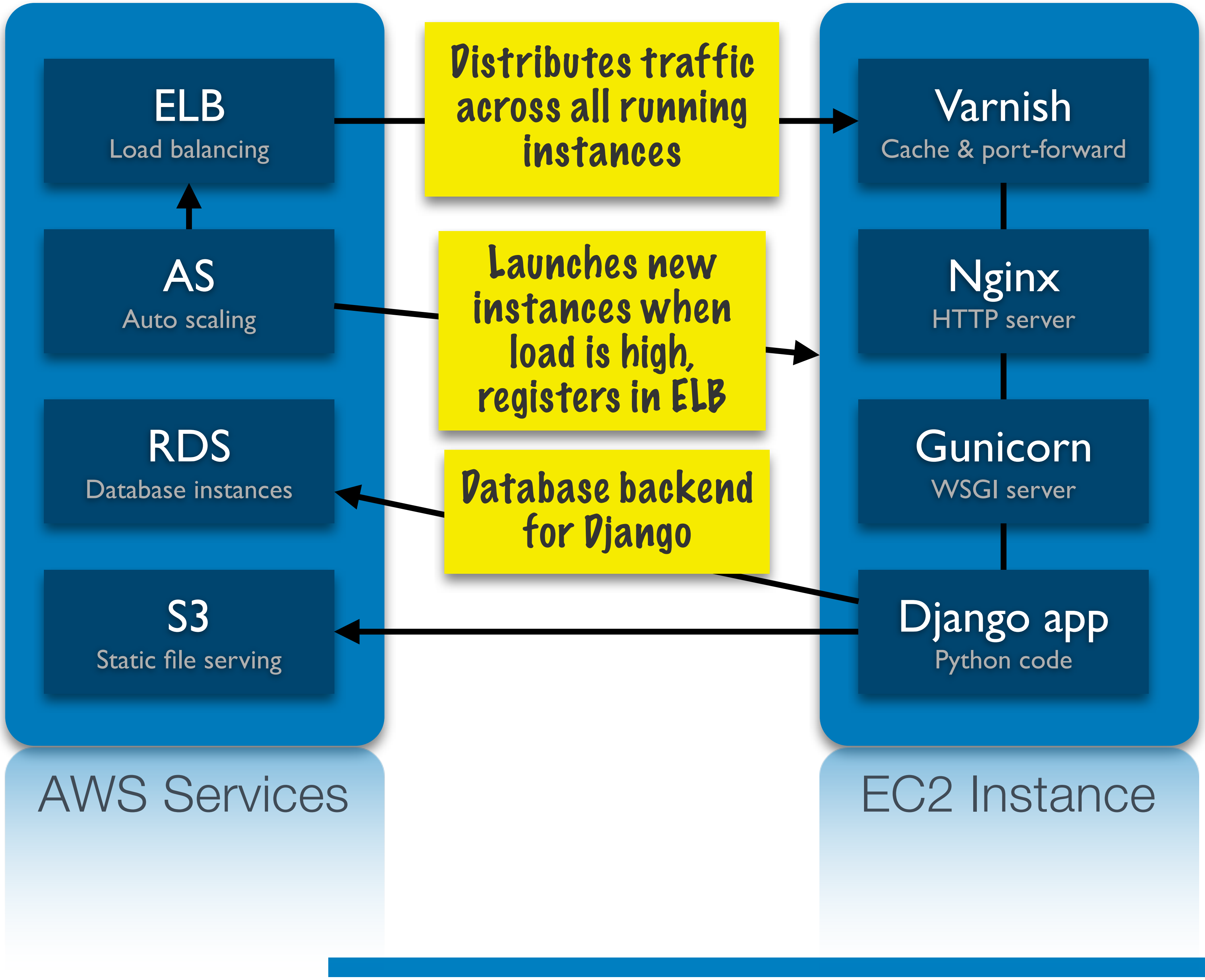
SCALING
DJANGO

DJANGO APPS ON AWS

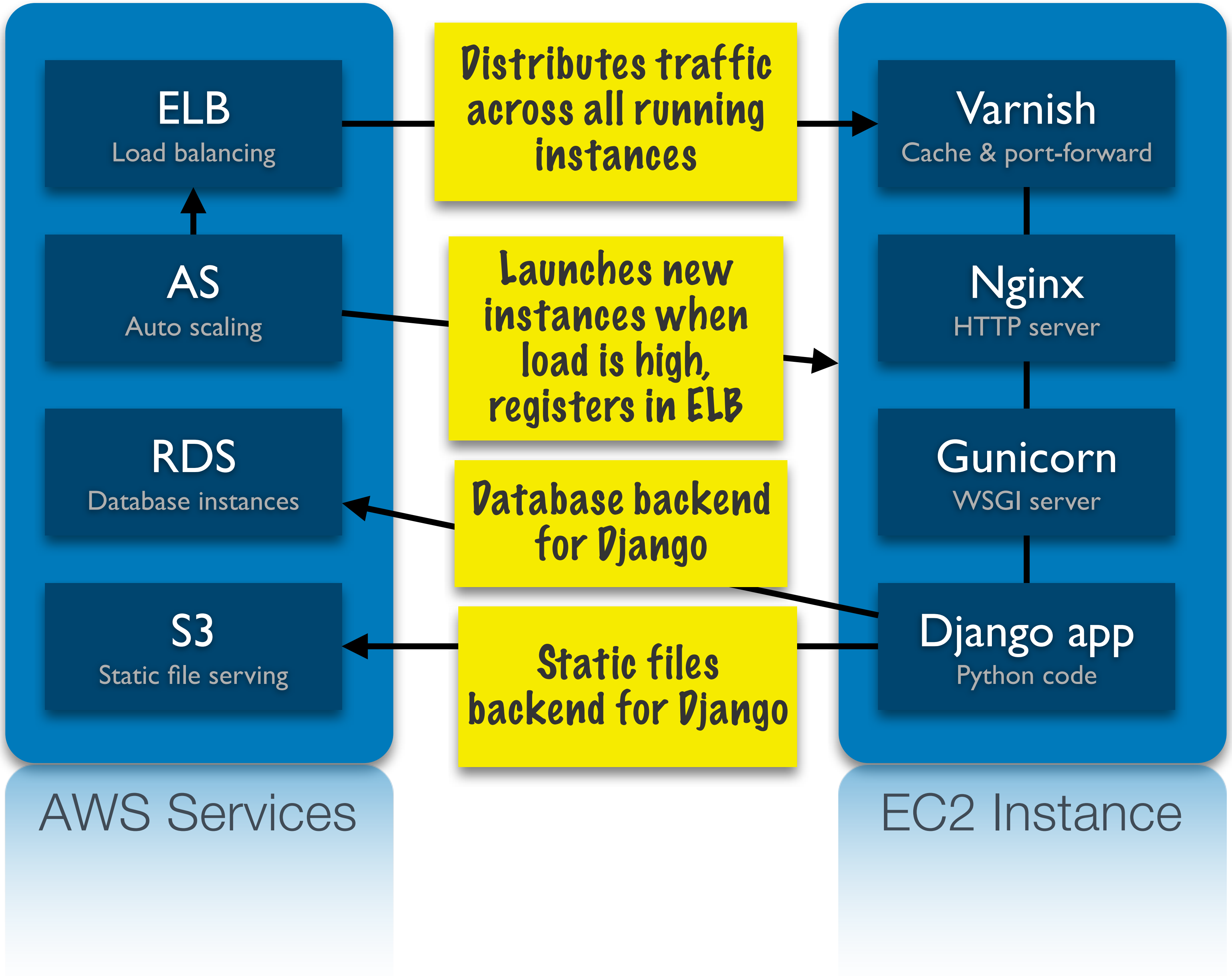


SCALING
DJANGO

DJANGO APPS ON AWS

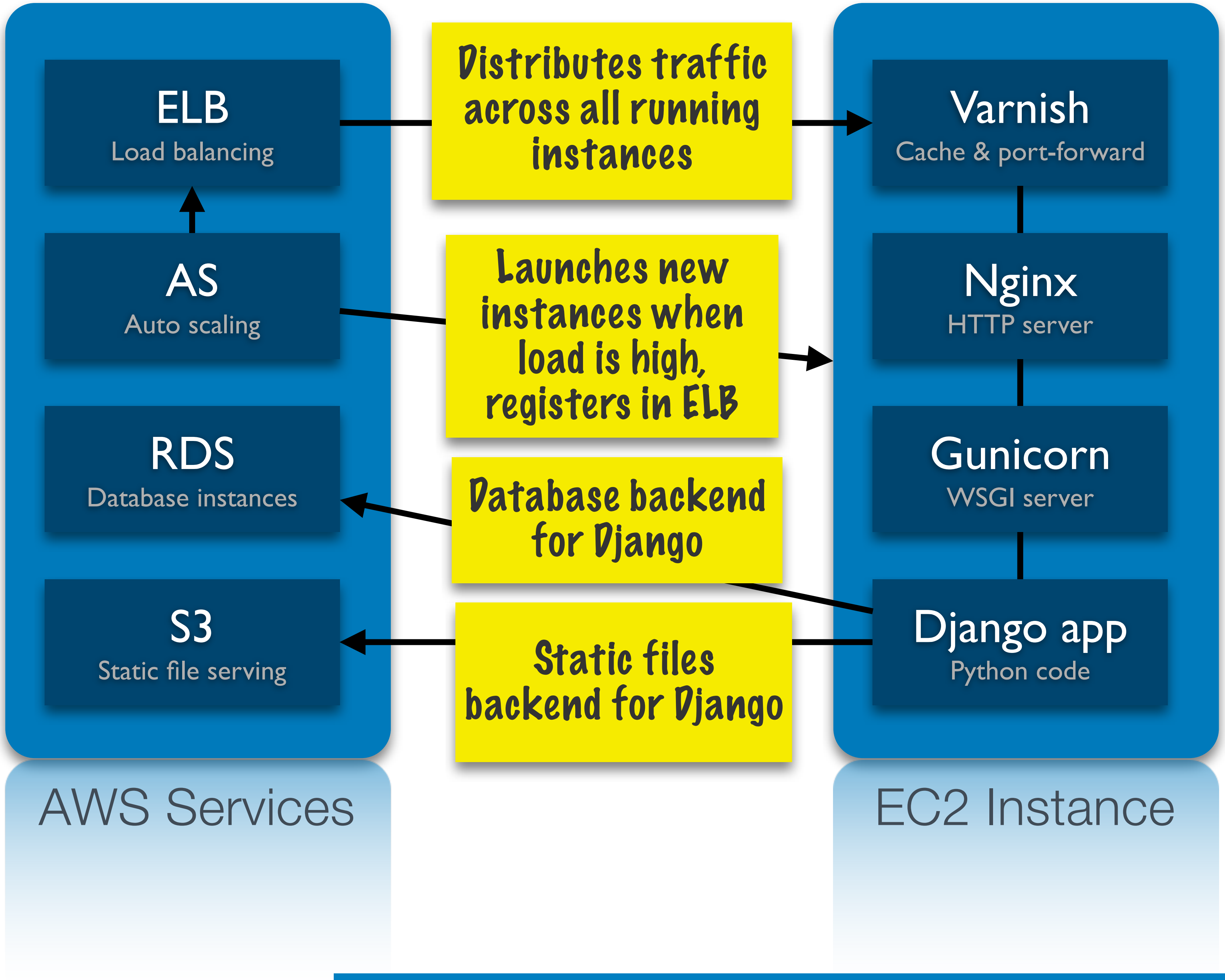


DJANGO APPS ON AWS



SCALING DJANGO

DJANGO APPS ON AWS



Stateless server stack

- Easy to add new instances.
- Standard Ubuntu & PyPI packages.
- Varnish is optional, depends on use-case. (Makes it easy to forward a URL space to another server.)
- Traditional alternative is Apache + mod_wsgi, but Nginx + Gunicorn is more efficient.

AWS AUTO-SCALING

- Auto-scaling launches and kills EC2 instances
  - You tell AWS the minimum and maximum number of instances you want.
  - And optionally the trigger conditions to scale up/down.
  - Elastic Load Balancing is automatically updated.
- Use for availability
  - Static auto-scaling configuration for exactly N instances.
  - If an instance dies, AWS will start a new one.
- Use for scalability
  - Setup triggers that scale between N to M instances.
  - Scales up and down.
  - E.g. based on CPU load.

ELASTIC LOAD-BALANCING

- AWS provides easy load-balancing with ELB
  - Setup a load-balancer and attach EC2 instances manually to it.
  - Or setup an auto-scaling group that manages ELB.
- The load-balancer is a CNAME
  - Configure your domain as a CNAME for the load-balancer.
  - CNAME cannot be for “example.com”, has to be “www.example.com”.
  - So you need to handle redirection from “example.com” separately.
- Each EC2 instance has to be stateless and independent
  - Don’t store anything on the local hard disk.
  - Assume any instance can die or start up at any time.

DJANGO SESSIONS

- Sessions tend to be a scalability bottleneck
  - Many platforms do silly things with sessions (looking at you, PHP).
- Store sessions in a shared database
  - All load-balanced EC2 instances need access to the sessions.
  - Various Django backends are available: SQL, memcached+SQL, Redis, etc.
  - (No SimpleDB session backend?)
- Or use no sessions at all
  - Implement your application with raw cookies only.
  - E.g. store object IDs in cookies, load objects from database when needed.
  - Use cookie hashing for security, encrypt if needed.
  - Check hashlib for SHA, PyCrypto for AES.

SCALING
DJANGO

DJANGO AND MEMCACHED

- Memcached is the general purpose caching system
  - Django supports it out of the box for data and session caching.
- You need to run your own memcached instances
  - Memcached just needs RAM (no CPU or I/O).
  - Can run on dedicated EC2 instances (extra cost).
  - Or run on shared web servers instances.

- Django needs some integration
  - With auto-scaling, memcached server IP addresses are always changing.
  - Enumerate with Boto and autoconfigure in settings.py, or deploy as a JSON file via Puppet.

DJANGO AND S3

- Remember, you cannot store anything on the file system
  - All uploaded or generated files must be stored on a shared server, such as S3.
  - Serving all static files from S3 is cheaper and faster than from EC2.
  - Also enables CloudFront CDN distribution.
- Django-storage: Store FileFields and ImageFields in S3
  - Drop-in replacement.
  - Supports S3, FTP, MongoDB GridFS, etc. as alternative backends.
- Django 1.3: New static files support
  - Can use S3 as a backend to copy all static files there:

    ./manage.py collectstatic

DJANGO AND RDS

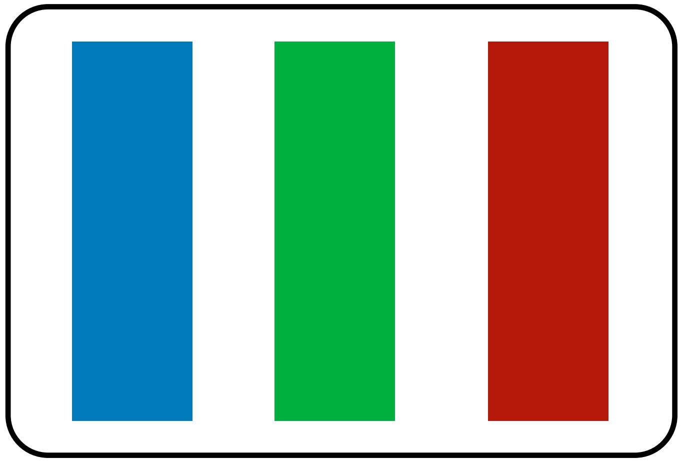
- Amazon RDS (Relational Database Service)
  - Provides you with MySQL 5.1 or 5.5 server instances.
  - Each instance can hold many databases.
  - Amazon takes care of updates and backups.

- Django sees RDS instances as ordinary MySQL servers
  - Configure hostname/username/password normally in settings.py.
- RDS provides security, availability and vertical scalability
  - DB Security Groups limit access to specific EC2 Security Groups (firewall).
  - Multi-Availability-Zone option provides availability during backups and problems.
  - Instance Classes: m1.small, m1.large, m1.xlarge, m2.xlarge, m2.2xlarge, m2.4xlarge

SCALING RDS BY INSTANCE TYPE

- RDS instances scale vertically
  - db.m1.small (1.7 GB of RAM, $0.11 per hour).
  - db.m1.large (7.5 GB of RAM, $0.44 per hour)
  - db.m1.xlarge (15 GB of RAM, $0.88 per hour).
  - db.m2.2xlarge (34 GB of RAM, $1.55 per hour).
  - db.m2.4xlarge (68 GB of RAM, $3.10 per hour).
- Database size is always limited
  - Minimum 5 GB to maximum 1 TB
- Scale up or down at any time
  - A running RDS instance can be modified and it will reboot.

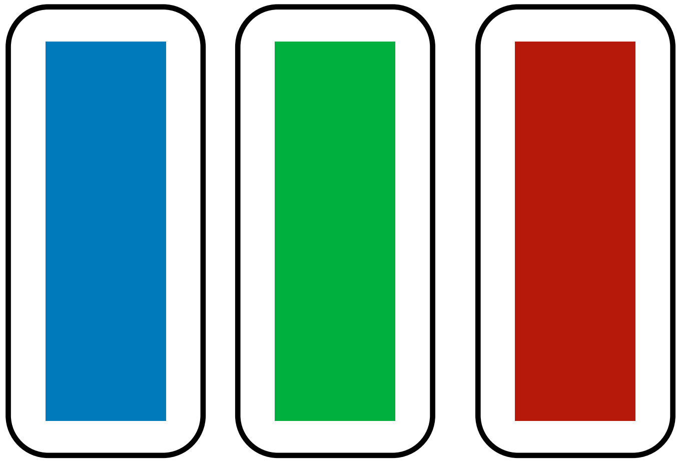
All tables in one db



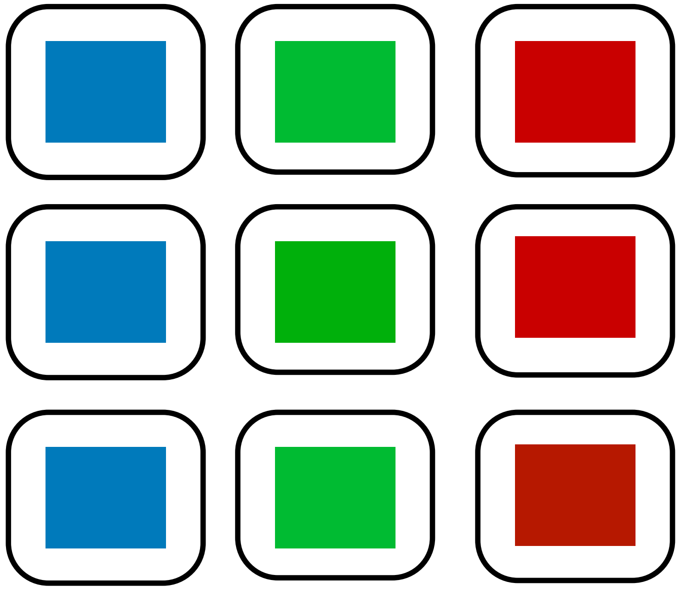
SCALING
DJANGO

SCALING RDS BY PARTITIONING

- Simple table-level partitioning
  - Put different tables in different RDS instances.
  - Configure Django's database router to use specific RDS instances for specific data models.
  - There can be no ForeignKeys between different RDS instances.



- Row-level partitioning (sharding)
  - Distribute table rows across multiple RDS instances.
  - E.g. split users into multiple shards. Scale by adding shards.
  - Need custom code in Django views or models to select RDS instance according to a sharding scheme (directory, modulo, etc).



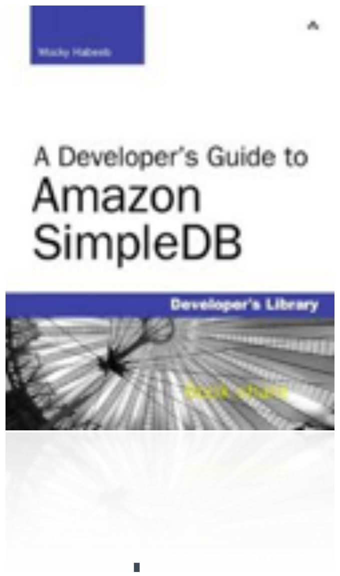
MULTI-DB & RDS

- Since 1.2, Django supports multiple databases
    - In settings.py: DATABASE ➪ DATABASES
- This works nicely together with Amazon RDS
    - RDS gives you the IP/hostname of each launched MySQL instance.
    - The instances can also be tagged and enumerated using Boto.
- Automated configuration
    - Integrate settings.py directly to AWS with Boto (enumerate RDS hosts).
    - Or pre-generate settings.py and distribute via Puppet.
    - Or pre-generate a JSON file, distribute via Puppet, and load it in settings.py.

DJANGO USER SHARDING

- Challenges with sharding users across databases
  - Built-in code assumes all users are in one auth_user table.
- Simple approach: Shard user profiles, not users
  - Keep auth_user in one huge table, only store minimum information.
  - Store the shard id of each user in e.g. User.last_name field.
  - Create separate profile model, with custom sharding logic based on shard id.
  - Make all ForeignKeys relative to profiles, not users.
- Solution is not perfect
  - Django admin won’t work with custom sharding logic.
  - The huge auth_user table will still get a lot of data & hits.

SIMPLEDB



A Developer's Guide to
Amazon
SimpleDB

Developer's Library

- SimpleDB is Amazon’s non-relational database service
  - Sort of like Google BigTable.
  - Store large collections of items, organized in domains.
  - Query language & indexing.
  - Schemaless (store attributes as key-value pairs).
  - Amazon handles scalability and availability.
- Integration to Django starting
  - Django-nonrel has a project to use SimpleDB as a database backend.
  - https://github.com/danfairs/django-simpledb
- Right now you can use SimpleDB as a simple datastore
  - Boto includes a SimpleDB API.
  - Useful when RDS is too complex for use case.

mongoDB

- Why? Scalability and schemalessness
  - Automatic sharding of collections across several MongoDB servers.
  - Just launch new servers when user base increases.
  - Data models can be modified without migrations (no South needed).
  - And they can contain lists and dictionaries!
- Django MongoDB Backend (mongodb-engine)
  - Drop-in replacement for MySQL & other RDBMS.
- MongoDB has certain limitations
  - No JOINs (use embedded objects & denormalization).
  - No transactions (use atomic updates like $inc).
  - But it works with Django admin!

MONGODB ON AMAZON EC2

- MongoDB has to run as a stateful server on EC2
  - Persistent data must be stored on an EBS data volume.
- Needs strategy for maintaining availability
  - What happens when a MongoDB EC2 instance dies?
  - You could set up an auto-scaling group to bring it up again.
  - Startup scripts can re-attach the new instance to the EBS data volume.
- Scalability is more challenging
  - Need to start new instances, create new EBS data volumes for them, configure MongoDB to shard on the new instances, etc.
  - Interesting challenge for automatization.

STATELESS VS. STATEFUL

Stateless EC2 servers

- Easy to autoscale.
- All servers boot from identical read-only image.
- Data is stored in RDS, S3, some other server.
- E.g. web servers, Django app servers, memcached.
- Prefer.

Stateful EC2 servers

- Cannot be autoscaled.
- Each server has their own set of persistent data.
- Data is stored in an EBS volume that cannot be shared.
- E.g. NoSQL servers, Git repositories, file servers.
- Avoid. Also needs custom solution for availability after an instance terminates.

RUNNING STATEFUL SERVERS

- Start with a virgin EBS AMI, such as Ubuntu
- Resize root volume to desired size
    - Detach volume, snapshot, create larger volume, attach.
    - Linux command: resize2fs /dev/xvda1
- Use a separate EBS data volume
    - Easy to move around & attach to other instances if needed.
- Regularly create an up-to-date AMI
    - If you screw up the OS or there is catastrophic failure, you can restart.
    - Detach data volume before creating AMI.
- Backup & restore via EBS snapshots
    - Might require LVM RAID to capture consistent data without shutdown.

EMAIL & AMAZON SES

- Amazon limits SMTP traffic heavily
  - If you send out emails through port 25, you’ll get blocked soon.
- AWS EC2 instances are on spam blacklists
- You need to fill out a form to request un-blacklisting and unblocking from Amazon.
- Alternative: Amazon SES (Simple Email Services)
  - Provides an API for sending email.
  - Django-ses package integrates directly to SES:

```
EMAIL_BACKEND = 'django_ses.SESBackend'
```

  - Quotas (emails/24h), send rate limits (emails/sec).
  - Separate pricing.

BOTO & SETTINGS.PY

- If you work with AWS, you need Boto
  - Full Python access to all AWS APIs.
  - Includes command line utilities, e.g. route53
- Use Git version (not PyPI) to get all latest features
  - pip install -e git+https://github.com/boto/boto.git#egg=boto
  - Ubuntu also uses Boto, so put your own version in a virtualenv.
- Use in Django settings.py to automatize configuration
  - Enumerate EC2 instances, RDS instances, get IP addresses.
  - Tag your instances in a smart way to help enumeration.
  - Risk: AWS access key has to be available. IAM may help.

FABRIC & PUPPET

firstPress
Pulling Strings with Puppet
Configuration Management Made Easy
James Turnbull
Apress

- Fabric is a SSH automatization tool
  - Run a sequence of shell commands on a remote machine.
  - Useful for running cron jobs, database migrations, etc.
  - Easy to integrate to Boto to automatically run shell commands on the currently running EC2 instances (which have dynamic IPs).
- Puppet is a configuration management tool
  - “Robot replacement for sysadmins”
  - Puppet Master holds the configuration, which is specified with a DSL.
  - Puppet Agents (EC2 instances) connect to the Master, download the configuration, and apply it to their systems.
  - Installs OS packages, copies configuration files, creates Linux users and groups, performs Git clones/pulls, etc.

CREATING AMIS

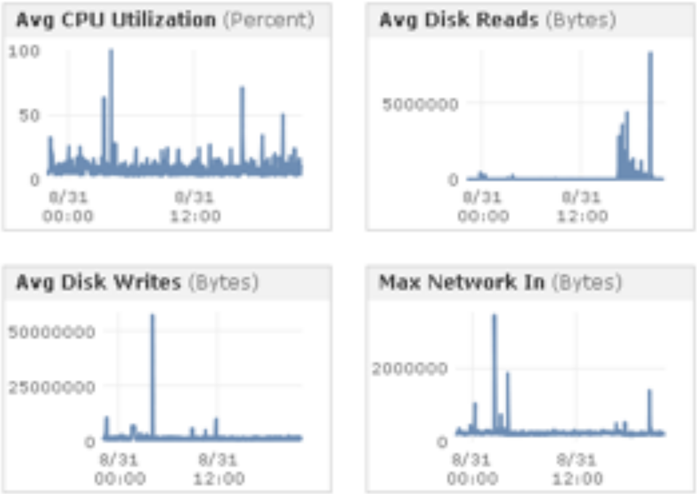
- AMIs are needed to launch new EC2 instances
  - An AMI is a snapshot of a pre-installed Linux system.
- You can create AMIs manually
  - Launch a virgin Ubuntu AMI.
  - Connect with ssh to the EC2 instance.
  - Apply your changes (install packages, modify configuration, etc.)
  - Create your own AMI from the instance using AWS Console.
- Or automatically
  - Use Boto to launch the virgin AMI instance.
  - Use a Fabric script to remotely apply the installation.
  - Use Boto to generate a new AMI.
  - Remember to clean up to avoid extra costs.

LOGGING

- You can let your EC2 instances log locally
  - The local disk is fast and will disappear when the instance is terminated.
  - It can be useful for solving problems while running.
- For important logs, use rsyslog (default on Ubuntu)
  - A central server can collect logs from all other instances.
  - Remember to set up log rotation to avoid filling hard disk.
- Django-sentry can log your Django app exceptions
  - Stores them in the database and provides a web UI.

MONITORING

- AWS provides CloudWatch option
  - External monitoring of EC2 instances (for extra cost).
  - You get a bunch of metrics like CPU, I/O, network.
  - Can send email on alarm.
  - Also used for auto-scaling triggers.



- You may want Nagios, Ganglia or similar
  - Internal monitoring of things CloudWatch can’t see.
  - Monitor application state, operating system state, etc.
- For high availability, consider outsourced monitoring
  - E.g. Pingdom.com can alert you if AWS fails completely.

MIRRORING PYPI

- A production site shouldn’t rely on PyPI
  - Deployments can fail.
  - Security updates can fail.
- There are some solutions for mirroring PyPI locally
  - See z3c.pypimirror on PyPI.
  - Problems with handling dependencies to GitHub / external links, etc.
- Any foolproof design includes staging
  - Test the whole system on a staging server first, then deploy to production.

FURTHER READING

- There’s an EC2 book on Kindle
  - Programming Amazon EC2
- Amazon has plenty of documentation
  - http://aws.amazon.com/documentation/
- New architecture center with webinars
  - http://aws.amazon.com/architecture/
- Me online
  - https://twitter.com/kennu
  - https://kfalck.net
- The end - Thanks!