Development to Deployment

- Some bugs only appear under stress
- Prod environment is not the same as the dev env
- Deployment exposes your app in ways your app was not meant to be used
  - Evil forces
  - Idiots

Early Deployment Strategies

- Need a Virtual Private Server (VPS)
  - Might be in cloud eg: AWS
- Install and configure OS and all utilities needed for the app
- Fix almost-weekly security vulnerabilities
- Tune all moving parts to get most bang for buck
- Figure out how to automate horizontal scaling

Platform as a Service (PaaS)

- Easy tiers of horizontal scaling
- Component-level performance tuning
- Infrastructure-level security
- App devs still have to worry about usage and efficiency of app

Performance Metrics

- Availability / uptime $\triangleq$ percent of time site is up and accessible
- Responsiveness $\triangleq$ how long after a click does user get response
- Scalability $\triangleq$ as users increase, can you maintain responsiveness without increasing cost/user

Availability

- Measured as a percent of uptime over a given window
- Defined in terms of 9's
  - 'five nines' $\triangleq$ 99.999% uptime
- Need to consider usage and implementation metrics as well
  - Server vs cluster availability
  - Availability at load

Responsiveness

- <100ms is 'instantaneous'

- 7sec is abandonment

- Want to consider what most users see; cannot model on a standard distribution

- Scalability

  - As number of users increase, response time to each user stays the same
  - Cost to serve each user stays the same

- Service Level Objective (SLO) $\triangleq$ percent of users get acceptable performance

  - Specify percentile, target response time, and time window

- Service Level Agreement (SLA) $\triangleq$ an SLO to which provider is contractually obligated

- Application Performance Index (apdex) $\triangleq$ defines satisfactory performance

  - Given a threshold latency T for user satisfaction
  - Satisfactory requests take $t \leq T$
  - Tolerable requests take $T \leq t \leq 4T$
  - Apdex $:= \frac{|\text{satisfactory}| + \frac{1}{2}|\text{tolerable}|}{|\text{requests}|}$
  - $\mathbb{E}[\text{apdex}|\text{good}] \in [0.85, 0.93]$

- Increase responsiveness

  - Small sites: overprovision presentation and logic tier
  - Large site: worry...

- Response time $:= t(\text{render done}) - t(\text{request sent})|_{t(x) \triangleq \text{time at event x}}$

Security Metrics

- Privacy $\triangleq$ is data access limited to the appropriate users
- Authentication $\triangleq$ can we trust that user is who s/he claims to be
- Data integrity $\triangleq$ is user's sensitive data tamper-evident

Three-Tier Architecture

- Type of shared nothing architecture

1. Presentation tier
2. Logic (application) tier
3. Persistence tier

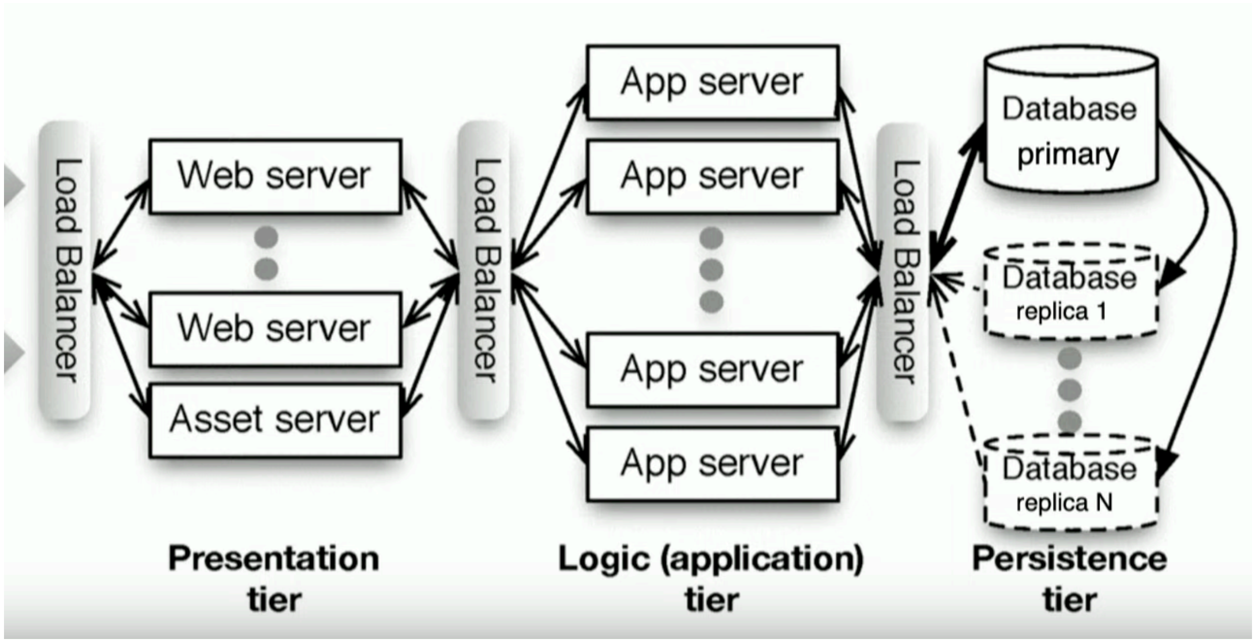

Figure 1: 475

-

Dynamic Content Generation

- Run a program to generate each 'page'
- Originally, web used template with embedded code 'snippets'
- Eventually, code moved out of the web server

Database Scaling

Sharding

- Partition across independent 'shards'
- Scales great
- Bad when operations touch > 1 table
- Eg: good for user profile info

Replication

- Replicate all data everywhere
- Good for multi-table queries
- Hard to scale b/c writes must propagate to all copies $\Longrightarrow$ temporary inconsistency in data values
- Example: social media 'wall' posts

Parts of PaaS App Controlled by Dev

- App (controller and model)
- Database access
  - Avoid abusive queries
  - Use indices wisely
- View rendering
  - Use CSS selectors wisely (`id` and `class` is fast, but DOM traversal isn't)
  - Cache 'expensive' fragments / partials in views
- Embedded assets (images, etc.) $\triangleq$ serve from dedicated static assets servers
- JavaScript $\triangleq$ load popular libraries from asset servers with generous expiration

CICD

Continuous Integration (CI)

- Integration testing the app beyond what each developer does
- Pre-release code checkin triggers CI
- Since frequent checkins, CI always running
- Common strategy: integrate with GitHub
- Good for:
  - Diagnosing differences between dev and prod envs
  - Cross-browser or cross-version testing
  - Testing SOA integration when remote services don't act as planned
  - Hardening / protecting against attacks
  - Stress testing / longevity testing of new features / code paths

Continuous Deployment (CD)

- Lots of deploys, very often
- Rational risk $\triangleq$ number of engineer-hours invested in product since last deploy
- Deployment should be a non-event; something that happens all of the time
  - Automate when possible
- Can configure to be deployed on push
  - Can be auto-integrated with CI
- Good for smaller features and non-customer facing changes
- Bad for (use releases for) big changes in how the user interacts with the app
  - Can tag commits to define releases with `git tag`
  - Have to push tags with `git push --tags`