Deployment

CS 485/698: AI-Assisted SE

Today’s Agenda

- Team meeting (~15 minutes)
  - Sprint planning: how are you going to deploy your course project for P6? What kind of user load can you handle?
    - I will stop by during the team meeting to ask
- Lecture on cloud computing, CI/CD
- In-class activity: deploying a simple web application

Scenario: You started a company in 2005

- Suppose that you are creating an online store for your parents' welding supply business.

Scenario: You started a company in 2005

- Suppose that you are creating an online store for your parents' welding supply business.
- What does your **tech stack** look like? What do you need to do?

Scenario: You started a company in 2005

- Suppose that you are creating an online store for your parents' welding supply business.
- What does your **tech stack** look like? What do you need to do?
  - Hook up Internet to your house.

Scenario: You started a company in 2005

- Suppose that you are creating an online store for your parents' welding supply business.
- What does your **tech stack** look like? What do you need to do?
  - Hook up Internet to your house.
  - Put a server in your garage.

Scenario: You started a company in 2005

- Suppose that you are creating an online store for your parents' welding supply business.
- What does your **tech stack** look like? What do you need to do?
  - Hook up Internet to your house.
  - Put a server in your garage.
  - Use the LAMP stack:
    - Linux
    - Apache web server (serves directories and files)
    - MySQL database (holds welding supply info)
    - All code written in Perl (e.g., for updating inventory)

Scenario: You started a company in 2005

- What’s nice about the 2005 tech stack?

Scenario: You started a company in 2005

- What's nice about the 2005 tech stack?
  - You can set up and run everything by yourself
  - Everything is relatively simple

Scenario: You started a company in 2005

- What's nice about the 2005 tech stack?
  - You can set up and run everything by yourself
  - Everything is relatively simple
- What are the downsides of the 2005 tech stack?

Scenario: You started a company in 2005

- What's nice about the 2005 tech stack?
  - You can set up and run everything by yourself
  - Everything is relatively simple
- What are the downsides of the 2005 tech stack?
  - You must set up and run everything by yourself!
  - How can you restart the server if you're not home?
  - Who rebuilds the system when the hard drive dies?
  - What if a hacker gets in and hijacks your site?
  - How do you scale?
    - What if you needed more storage? bandwidth?

Cloud Computing

- “The cloud is just someone else’s computer”
- A cloud provider (AWS, GCP, Azure, etc) builds a **big datacenter** with thousands of computers
  - They rent compute time to you
- Why would you want this?

Cloud Computing

- “The cloud is just someone else’s computer”
- A cloud provider (AWS, GCP, Azure, etc) builds a **big datacenter** with thousands of computers
  - They rent compute time to you
- Why would you want this?
  - **Specialization**: you worry about your business, they worry about the server infrastructure
    - “AWS has better ops than you”

Cloud Computing

- “The cloud is just someone else’s computer”
- A cloud provider (AWS, GCP, Azure, etc) builds a **big datacenter** with thousands of computers
  - They rent compute time to you
- Why would you want this?
  - **Specialization**: you worry about your business, they worry about the server infrastructure
    - “AWS has better ops than you”
  - **Simplifies your planning**: if you suddenly get popular, you can just rent more compute time

Cloud Computing: Selling Services

- These days, cloud providers offer many different “kinds” of compute time. Examples:

Cloud Computing: Selling Services

- These days, cloud providers offer many different “kinds” of compute time. Examples:
  - Software as a Service (SaaS).
    - E.g., Google offers an application that does mail (GMail). You log in and use the app.

Cloud Computing: Selling Services

- These days, cloud providers offer many different “kinds” of compute time. Examples:
  - Software as a Service (SaaS).
    - E.g., Google offers an application that does mail (GMail). You log in and use the app.
  - Infrastructure as a Service (IaaS)
    - E.g., AWS’s EC2 offers virtual servers, networking, and storage. You do the OS and apps.

Cloud Computing: Selling Services

- These days, cloud providers offer many different “kinds” of compute time. Examples:
  - Software as a Service (SaaS).
    - E.g., Google offers an application that does mail (GMail). You log in and use the app.
  - Infrastructure as a Service (IaaS)
    - E.g., AWS’s EC2 offers virtual servers, networking, and storage. You do the OS and apps.
  - Platform as a Service (PaaS)
    - E.g., Microsoft’s Azure offers an OS, scaling, deployment. You do apps.

Cloud Computing: Continuous Delivery

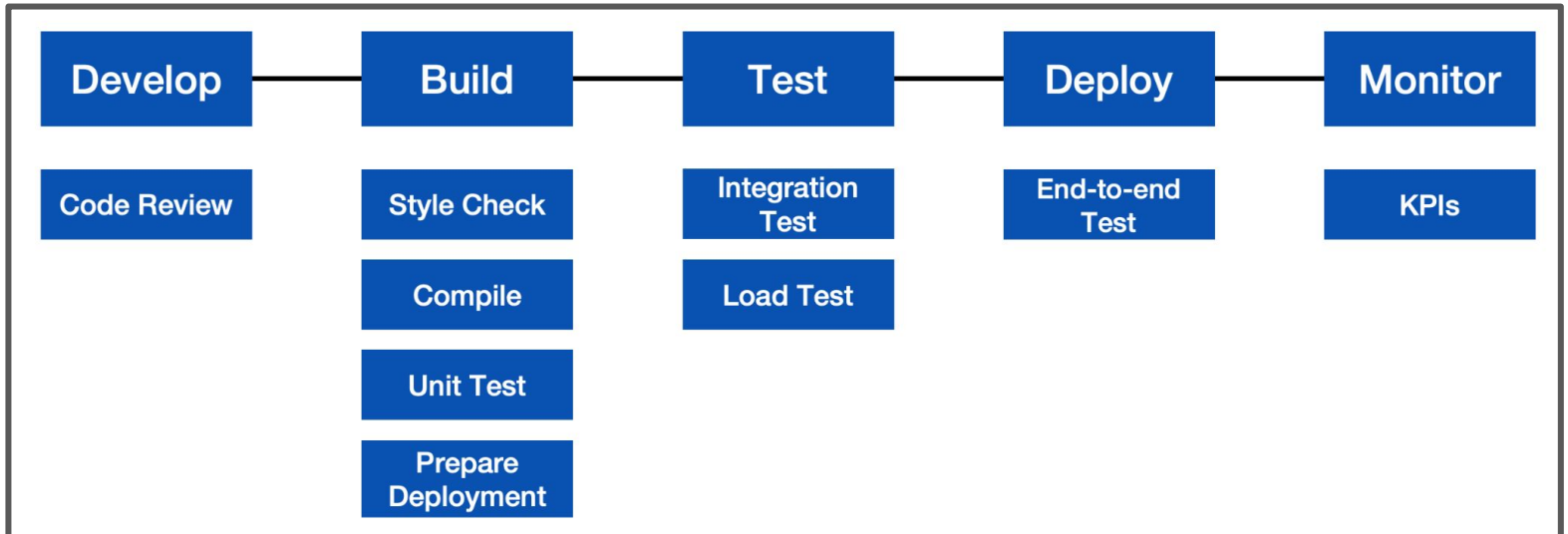
- **Continuous *integration***: Perform frequent integrations with entire codebase, running integration-scale tests

Cloud Computing: Continuous Delivery

- **Continuous *integration***: Perform frequent integrations with entire codebase, running integration-scale tests
- **Continuous *delivery***: Deploy frequently and monitor

Cloud Computing: Continuous Delivery

- **Continuous *integration***: Perform frequent integrations with entire codebase, running integration-scale tests
- **Continuous *delivery***: Deploy frequently and monitor
  - Intution: complete **pipeline** from dev to production



Cloud Computing: Continuous Delivery

- **Continuous *integration***: Perform frequent integrations with entire codebase, running integration-scale tests
- **Continuous *delivery***: Deploy frequently and monitor
  - Intution: complete **pipeline** from dev to production
  - Key values of continuous delivery:
    - “Faster is safer”
    - Release frequently, in small batches
    - Maintain key performance indicators to evaluate the impact of updates
    - Phase roll-outs
    - Evaluate business impact of new features

Continuous Delivery: Motivation

Continuous Delivery: Motivation

Knightmare: A DevOps Cautionary Tale

Published by **D7** on **April 17, 2014**

I was speaking at a conference last year on the topics of DevOps, Configuration as Code, and Continuous Delivery and used the following story to demonstrate the importance making deployments fully automated and repeatable as part of a DevOps/Continuous Delivery initiative. Since that conference I have been asked by several people to share the story through my blog. This story is true – this really happened. This is my telling of the story based on what I have read (I was not involved in this).

This is the story of how a company with nearly $400 million in assets went bankrupt in 45-minutes because of a failed deployment.

Continuous Delivery: Motivation

Knightmare: A DevOps Cautionary Tale

Published by D7 on April 17, 2014

I was speaking at a conference last year on the topics of DevOps, Configuration as Code, and Continuous Delivery and used the following story t making deployments fully automated and repeatable a Delivery initiative. Since that conference I have been a story through my blog. This story is true – this really h story based on what I have read (I was not involved in

This is the story of how a company with nearly $400 m minutes because of a failed deployment.

"In the week before go-live, a Knight engineer manually deployed the new RLP code in SMARS to its 8 servers. However, he made a mistake and did not copy the new code to one of the servers. Knight did not have a second engineer review the deployment, and neither was there an automated system to alert anyone to the discrepancy. "

https://www.henricodolfing.com/2019/06/project-failure-case-study-knight-capital.html

What could Knight have done better?

What could Knight have done better?

- Use capture/replay testing instead of driving market conditions in a test
- Avoid including “test” code in production deployments
- Automate deployments
- Define and monitor risk-based KPIs
- Create checklists for responding to incidents

What could Knight have done better?

- Use capture/replay testing instead of driving market conditions in a test
- Avoid including “test” code in production deployments
- Automate deployments
- Define and monitor risk-based KPIs
- Create checklists for responding to incidents

All elements of a good **continuous delivery** strategy!

Continuous Delivery != Immediate Delivery

- Even if you are deploying every day (“continuously”), you still have some latency
  - A new feature I develop today won't be released today

Continuous Delivery != Immediate Delivery

- Even if you are deploying every day (“continuously”), you still have some latency
  - A new feature I develop today won't be released today
  - But, a new feature I develop today can begin the **release pipeline** today (minimizes risk)

Continuous Delivery != Immediate Delivery

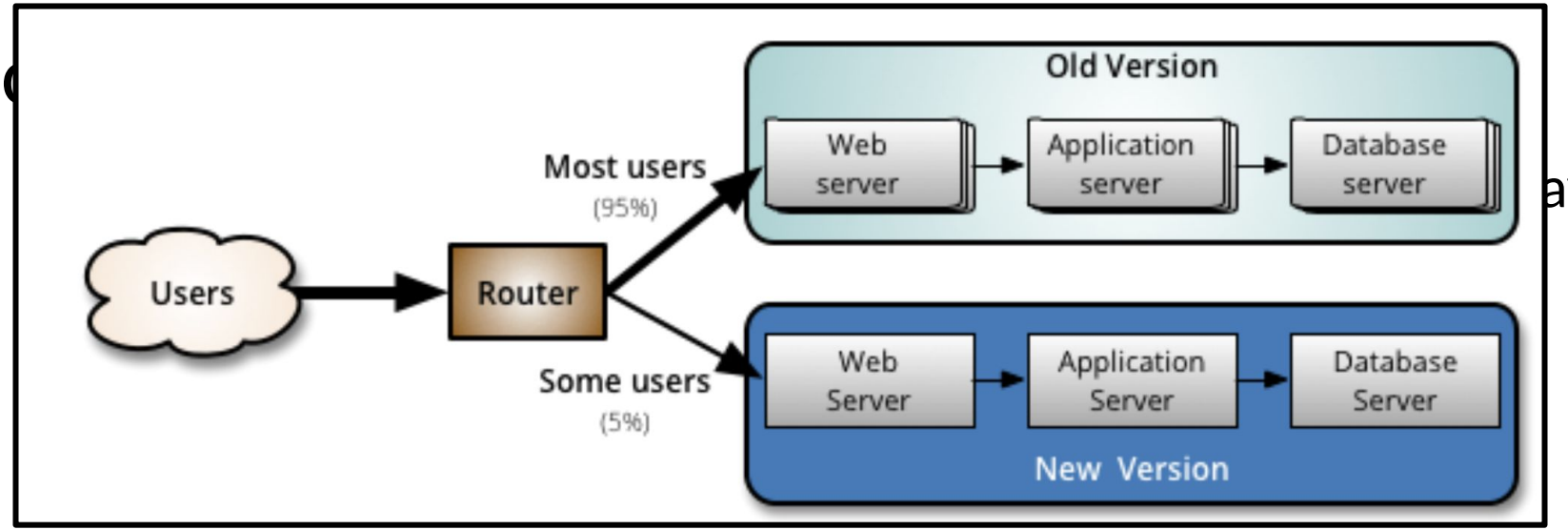
- Even if you are deploying every day (“continuously”), you still have some latency
  - A new feature I develop today won't be released today
  - But, a new feature I develop today can begin the **release pipeline** today (minimizes risk)
- Example risk mitigation strategy: ***split deployments***

Continuous Delivery != Immediate Delivery

- Even if you are deploying every day (“continuously”), you still have some latency
  - A new feature I develop today won't be released today
  - But, a new feature I develop today can begin the **release pipeline** today (minimizes risk)
- Example risk mitigation strategy: ***split deployments***
  - Idea: Deploy to a complete production-like environment, but don't have users actually use it, collect preliminary feedback

Co

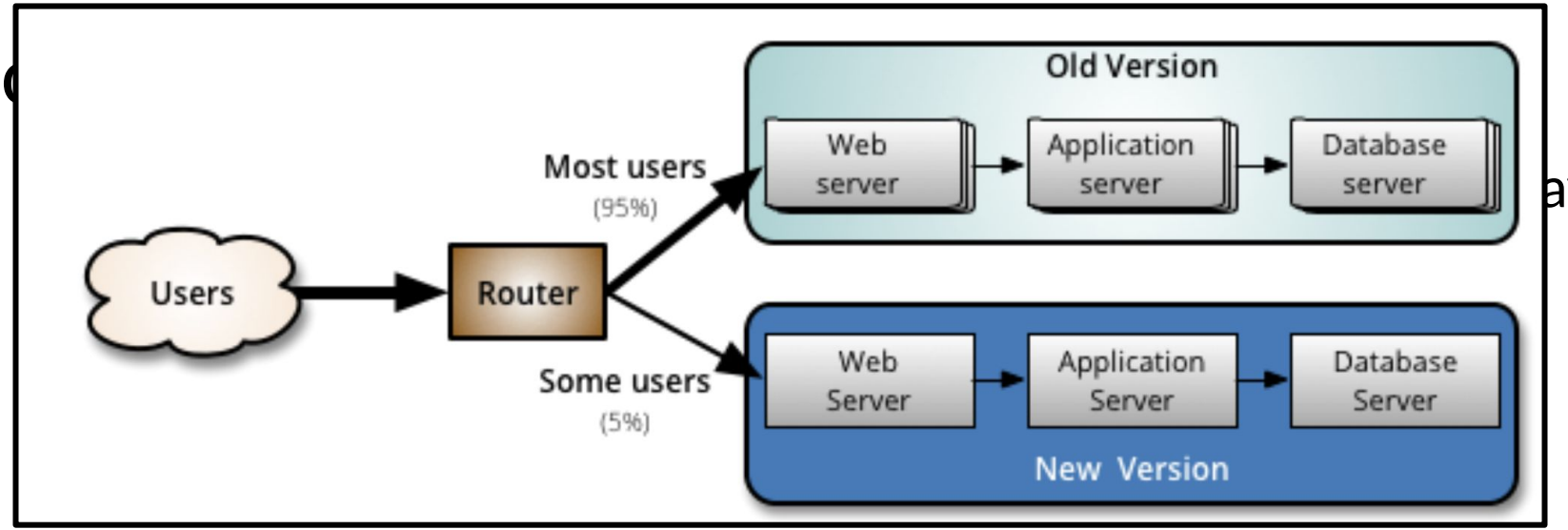
- ave



- Example risk mitigation strategy: ***split deployments***
  - Idea: Deploy to a complete production-like environment, but don't have users actually use it, collect preliminary feedback

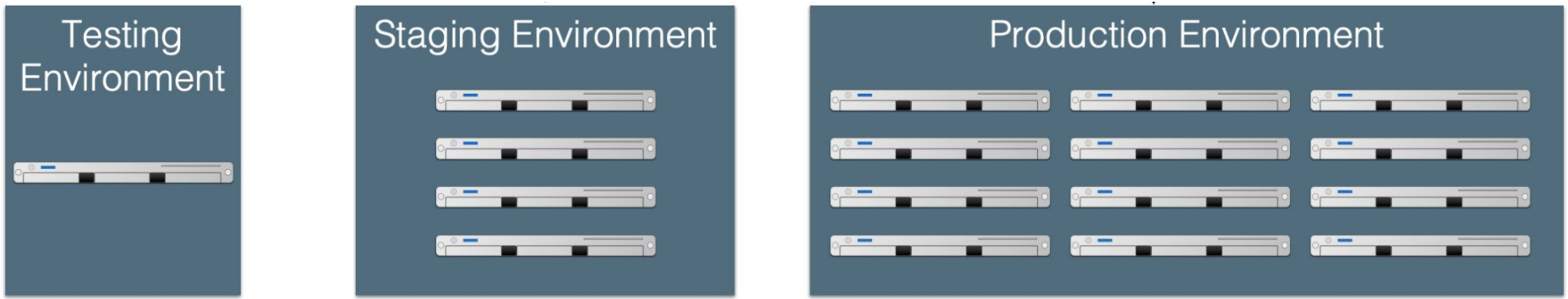
Co

- ave

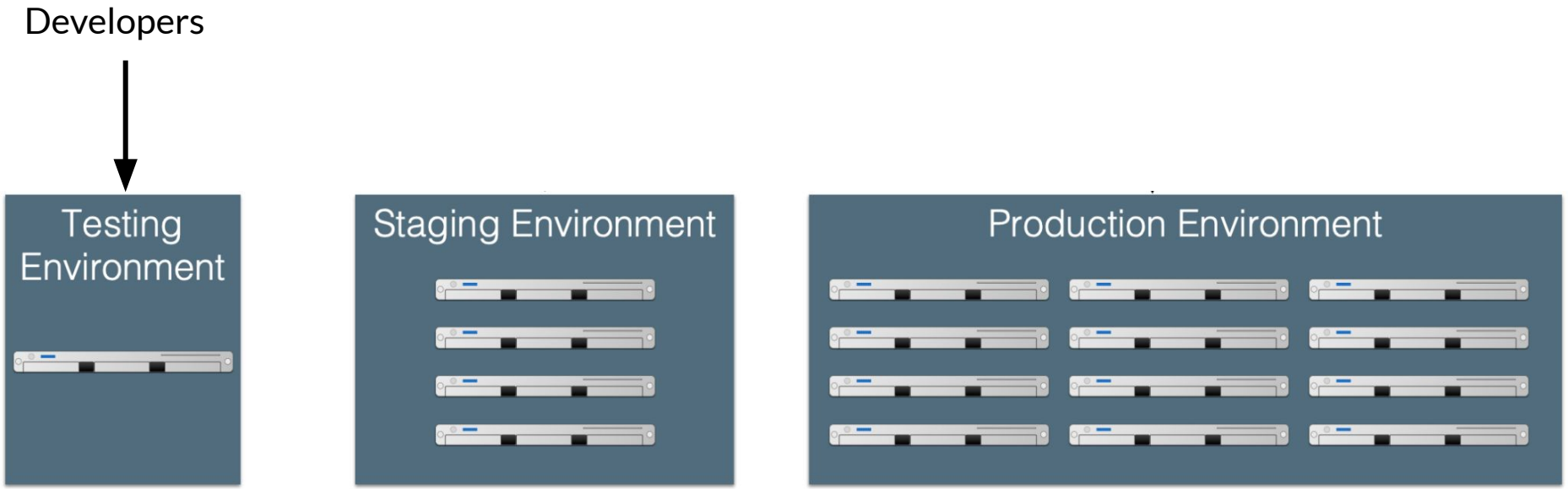


- Example risk mitigation strategy: ***split deployments***
  - Idea: Deploy to a complete production-like environment, but don't have users actually use it, collect preliminary feedback
  - Lower risk if a problem occurs in ***staging*** than in production
    - "Dogfood" your own apps in staging, use alpha testers, etc

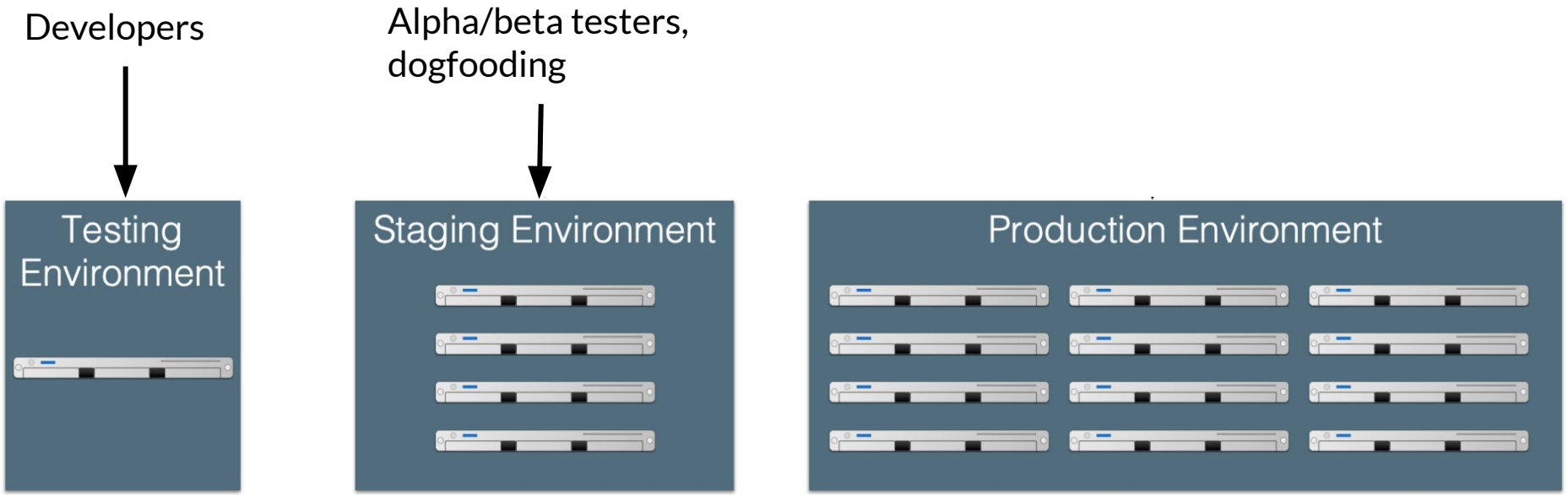
Continuous Delivery: Staging Environments



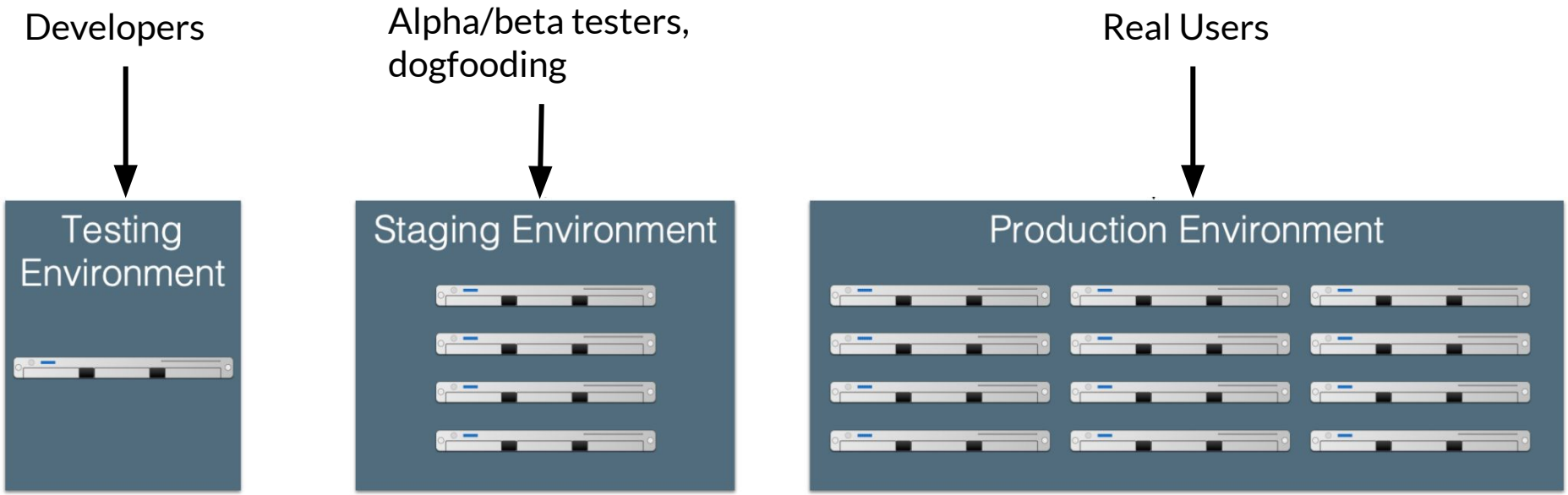
Continuous Delivery: Staging Environments



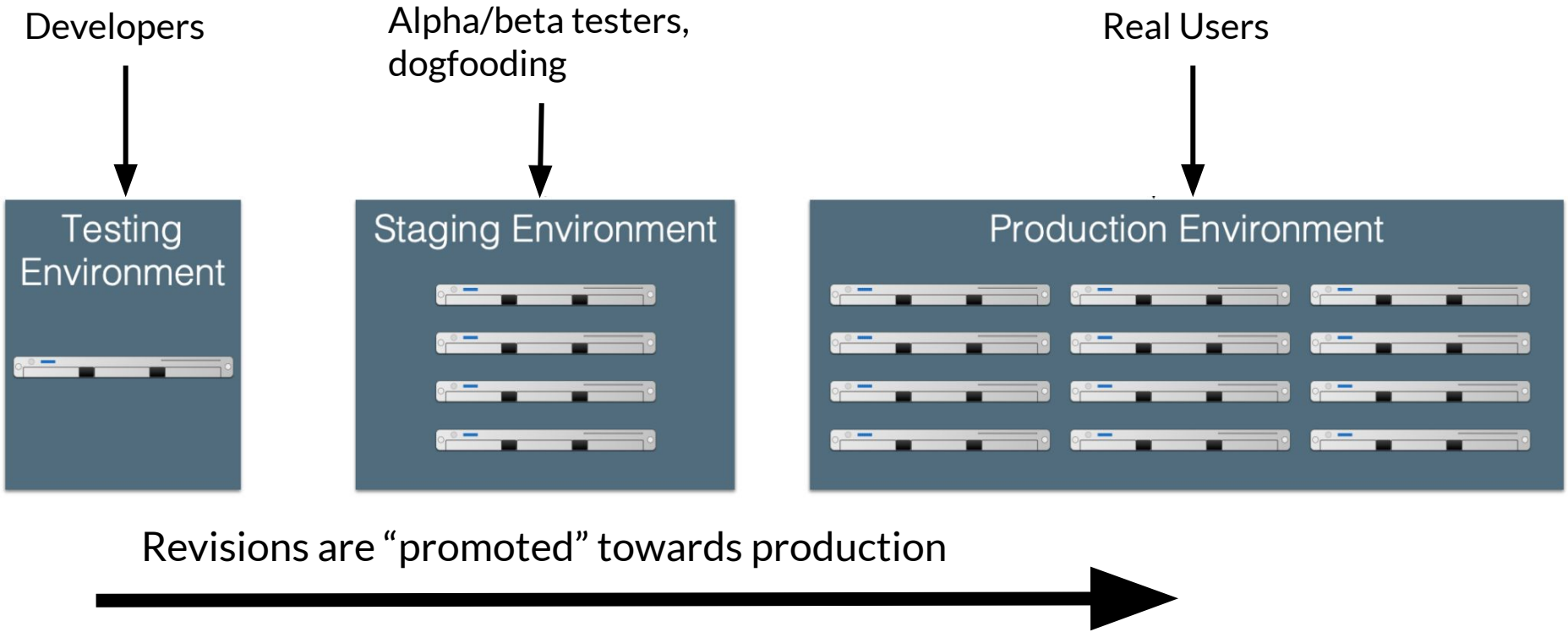
Continuous Delivery: Staging Environments



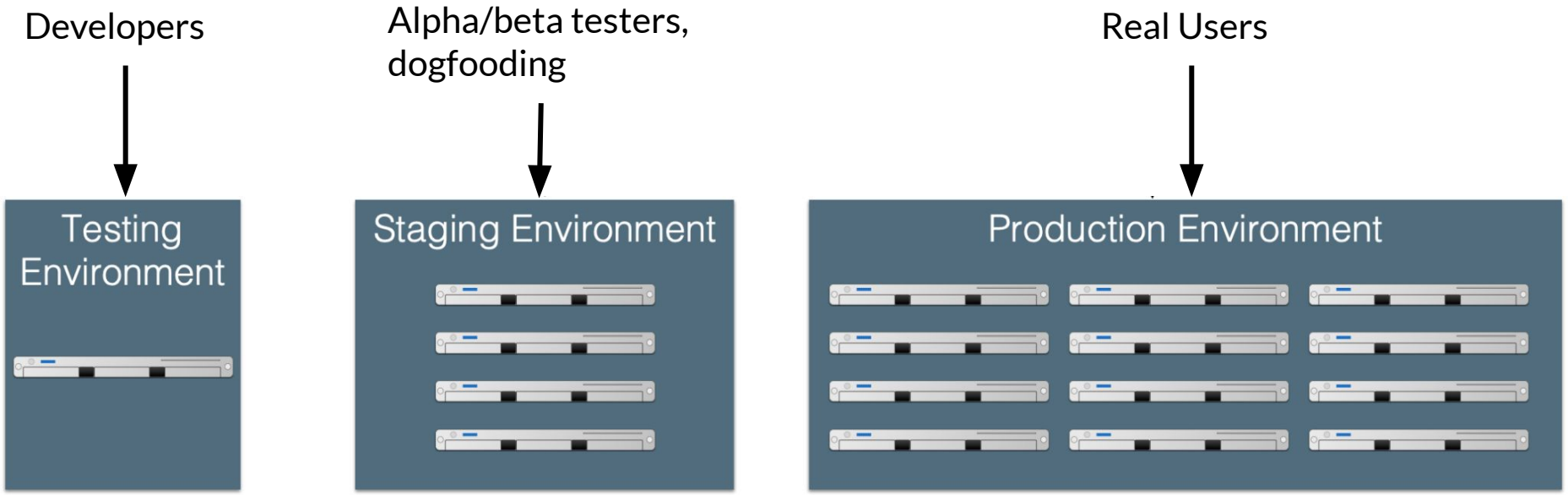
Continuous Delivery: Staging Environments



Continuous Delivery: Staging Environments



Continuous Delivery: Staging Environments



Continuous Delivery: Tooling

- Simplest tools deploy from a branch to a service (e.g. AWS Amplify, Render.com, Heroku)

Continuous Delivery: Tooling

- Simplest tools deploy from a branch to a service (e.g. AWS Amplify, Render.com, Heroku)
- More complex tools:
  - Auto-deploys from version control to a staging environment + promotes through release pipeline
  - Monitors key performance indicators to automatically take corrective actions
  - Example: “Spinnaker” (Open-Sourced by Netflix, c 2015)

In-class Activity: A Simple Deployment

- This is a pair programming activity. Find your partner.

In-class Activity: A Simple Deployment

- This is a pair programming activity. Find your partner.
- The overall goal today is to deploy a simple application on AWS.
  - We'll start by generating one with an LLM
  - Then, we'll try to actually deploy it

In-class Activity: A Simple Deployment

- This is a pair programming activity. Find your partner.
- The overall goal today is to deploy a simple application on AWS.
  - We'll start by generating one with an LLM
  - Then, we'll try to actually deploy it
- **One partner** should start **generating the calculator application** that we're going to use as an example (see next slide for the spec)

In-class Activity: A Simple Deployment

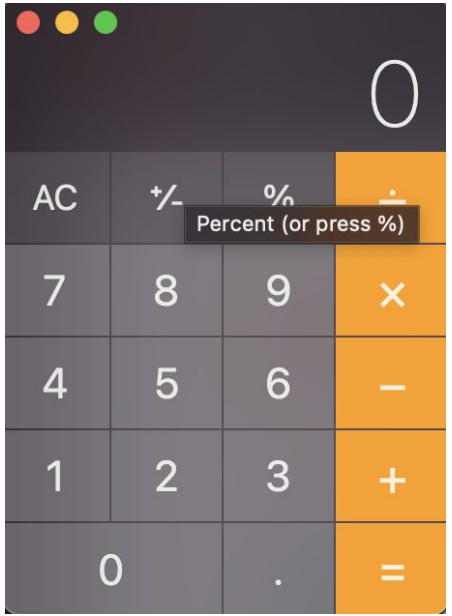
- This is a pair programming activity. Find your partner.
- The overall goal today is to deploy a simple application on AWS.
  - We'll start by generating one with an LLM
  - Then, we'll try to actually deploy it
- **One partner** should start **generating the calculator application** that we're going to use as an example (see next slide for the spec)
- The **other partner** should start **setting up AWS Amplify** for deployment (see slide after next)

In-class Activity: A Simple Deployment

- This is a pair programming activity. Find your partner.
- The overall goal today is to deploy a simple application on AWS.
  - We'll start by generating one with an LLM
  - Then, we'll try to actually deploy it
- **One partner** should start **generating the calculator application** that we're going to use as an example (see next slide for the spec)
- The **other partner** should start **setting up AWS Amplify** for deployment (see slide after next)
- Once you have the app, work together to actually get it deployed
  - Let us know once you've done so

In-class Activity: Calculator App

- Create a Node.js React app for a basic calculator.
- The frontend should contain the following buttons:
  - Numbers: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, . (decimal)
  - Functions: +, -, x, ÷, %, =, +/-, Clear, Backspace
  - It should create a string with the current expression as the user presses buttons and display it at the top.
  - Handle Clear and Backspace on the frontend.
- The backend should compute the value of the expression.
  - It has one public function: `string calculate(string)`
- When the user presses =, the frontend should call calculate() on the backend with the current expression and display the return value at the top of the calculator.
- Test out your calculator on localhost to make sure it works.



In-class Activity: AWS Amplify

- Assume that your partner will check your calculator app into a GitHub repo that you own
- Connect to AWS Console (https://console.aws.amazon.com)
  - Create an AWS account if you don't have one
- Go to AWS Amplify and connect it to your GitHub
  - Authorize AWS Amplify to use your GitHub account
- Pick your calculator repo and branch. Ask us if you don't see it in the pulldown.
  - Check the box "My app is a monorepo."
  - Fill in the root directory of your frontend app
  - Save and deploy
  - Deploy and visit the deployed URL

In-class Activity: Advanced Calculation

- Once your calculator is deployed, practice automated deployment and testing:
  - Add a small feature to your calculator (e.g., an exponentation function). Does Amplify automatically deploy it?
  - Have your LLM add automated tests for the calculator. If these tests fail, will Amplify still deploy a change?
    - Intentionally break the tests and find out!

Wrapup and Reminders

- A6 sign up are open
  - sign up by Wednesday