Web Science Application Development

Jason Kuruzovich, PhD

kuruzj@rpi.edu
https://biggermatrix.com

October 8, 2025

John Schroeder
VP, Management Plane Engineering

VP, Management Plane Engineering
Mimic · Full-time
Sep 2023 - Present · 2 yrs 7 mos
Palo Alto, California, United States · Remote

Mimic is a category defining last line of defense solution built to defend and deflect attacks after ransomware has breached... more

Chief Technology Officer
Qwire Inc. · Full-time
Mar 2022 - Sep 2023 · 1 yr 7 mos
New York City Metropolitan Area · Remote

Qwire is an enterprise SaaS solution for music clearance, licensing, and cue sheet reporting for major studios, sports networks,... more

Principal
Fulcrum Growth
Jun 2018 - Apr 2022 · 3 yrs 11 mos
Saratoga Springs, NY

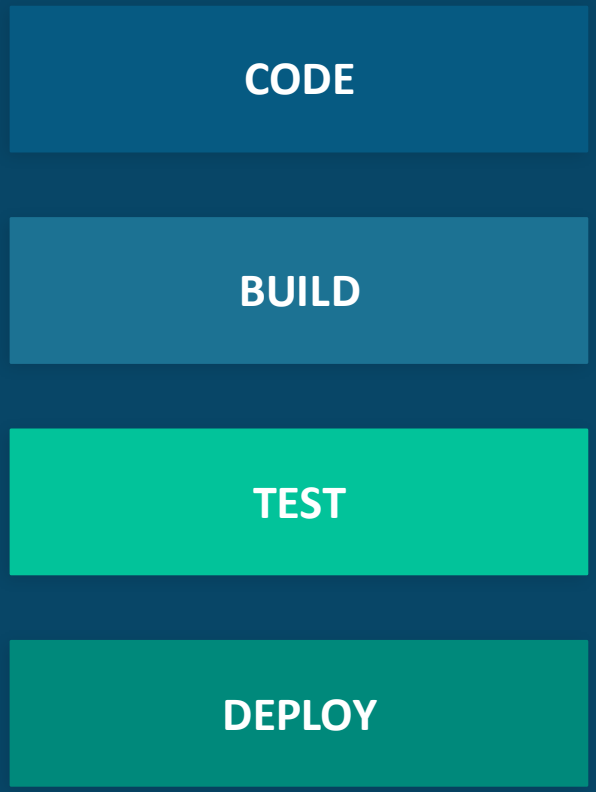
Providing product management, software development, and ... operations consulting to high growth technology companies inmore

Chief Information Officer
Pet Partners LLC
May 2015 - Jun 2018 · 3 yrs 2 mos

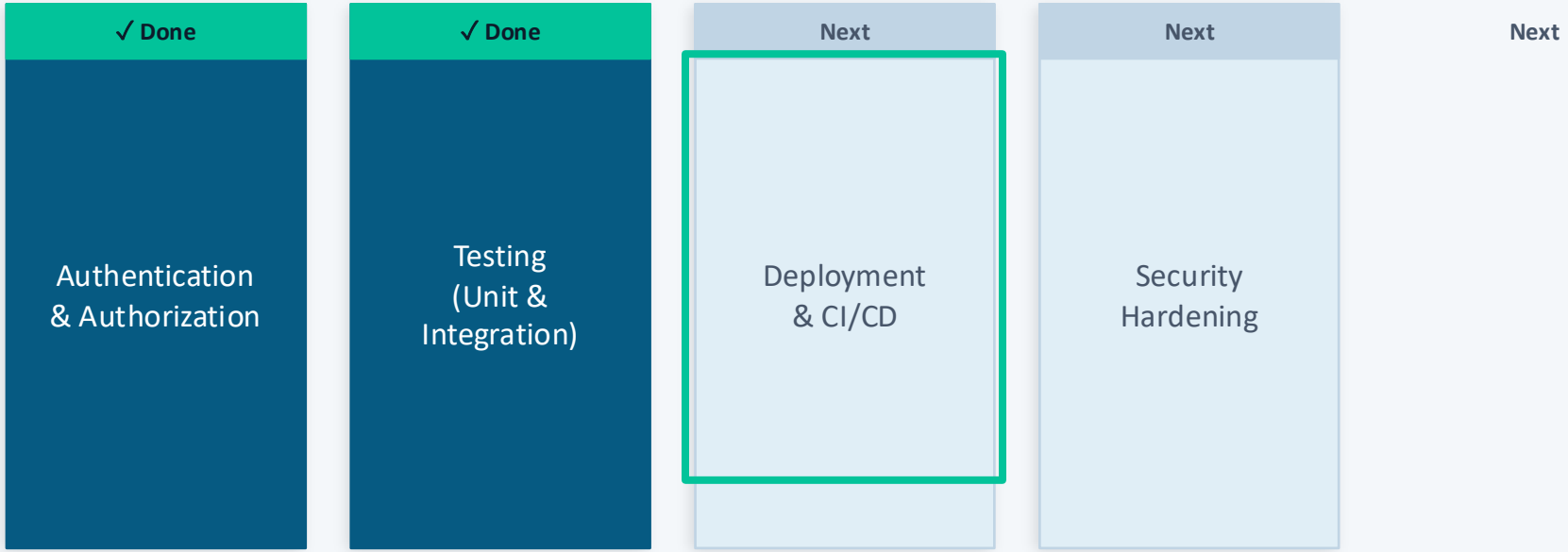
Deployment &

CI/CD Pipelines

From Code Commit to Running Application



Where We Are in the Course



Today's focus: taking tested code all the way to running in production.

The Deployment Problem

Manual Deployment Risks

- Human error in each step
- "Works on my machine" syndrome
- Long release cycles (fear of shipping)
- No rollback plan
- Dev and prod environments diverge
- Testing skipped under time pressure

CI/CD Solves This

- Automated, repeatable pipeline
- Tests run on every commit
- Ship small changes frequently
- Automatic rollback on failure
- Infrastructure defined as code
- One-click or fully automatic deploy

Understanding CI / CD

Continuous Integration (CI)

Continuous Delivery (CD) — with manual gate before deploy

Continuous Integration

Merge code often. Automatically build and run tests on every commit. Goal: catch integration bugs early.

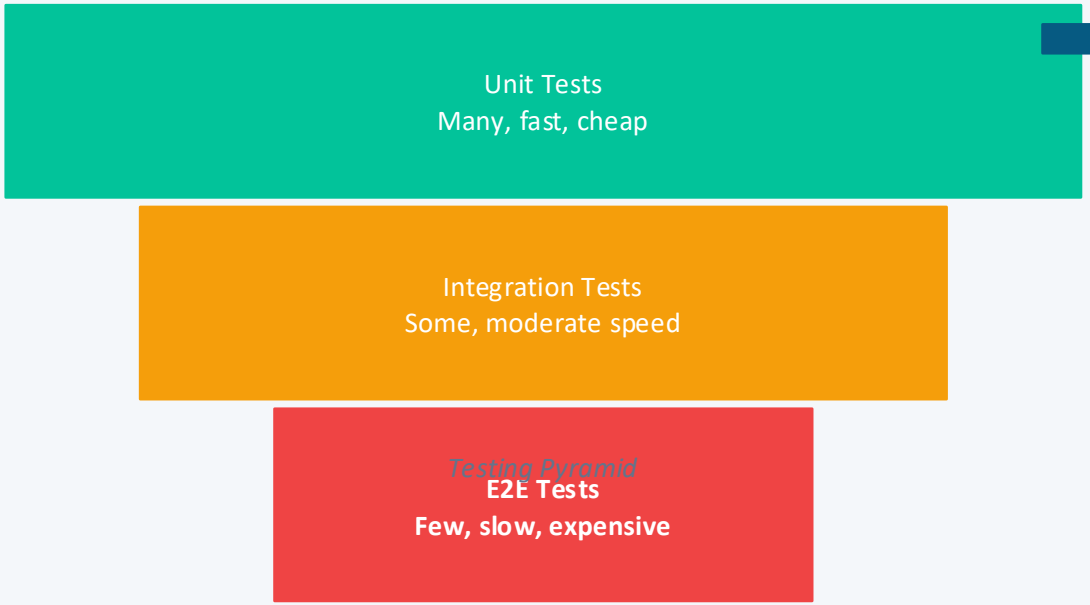
Continuous Delivery

Code is always in a deployable state. Deployment to production requires a manual approval step.

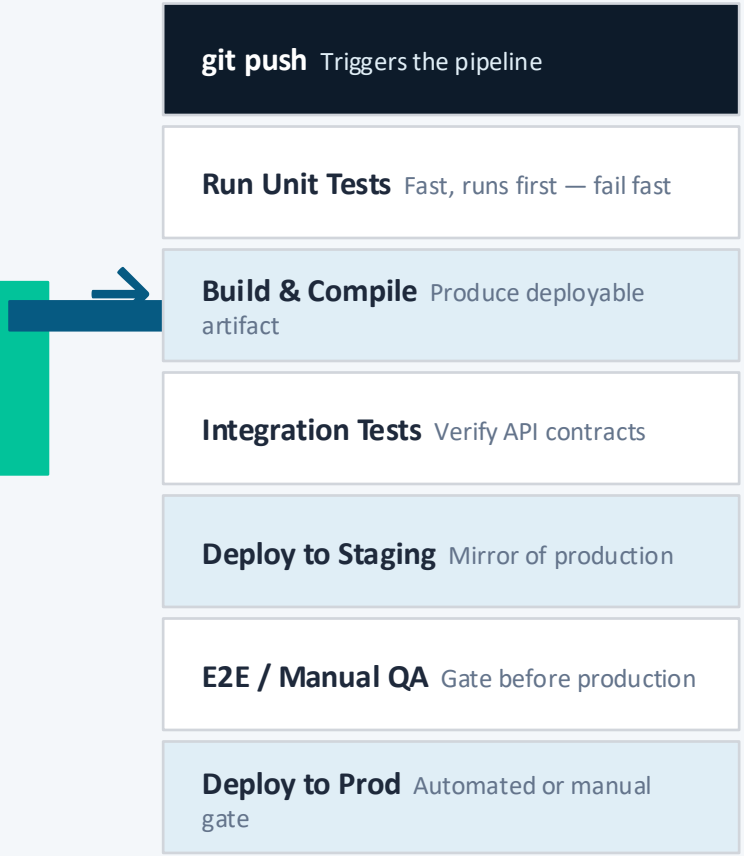
Continuous Deployment

Every change that passes tests is deployed automatically to production. No human gate.

From Testing Pyramid to CI/CD Pipeline



Testing Pyramid



CI/CD Pipeline Stages

Environments: Dev → Staging → Production

💻 Development	🔧 Staging	🚀 Production
Local machines or shared dev server	Mirrors production exactly	Live users, real data
Frequent changes, breaking OK	Real infrastructure, safe data	Zero tolerance for downtime
Mock data / services	Integration & E2E tests run here	Monitored & alerting active
Fast feedback loop	Pre-approval for release	Only tested code gets here

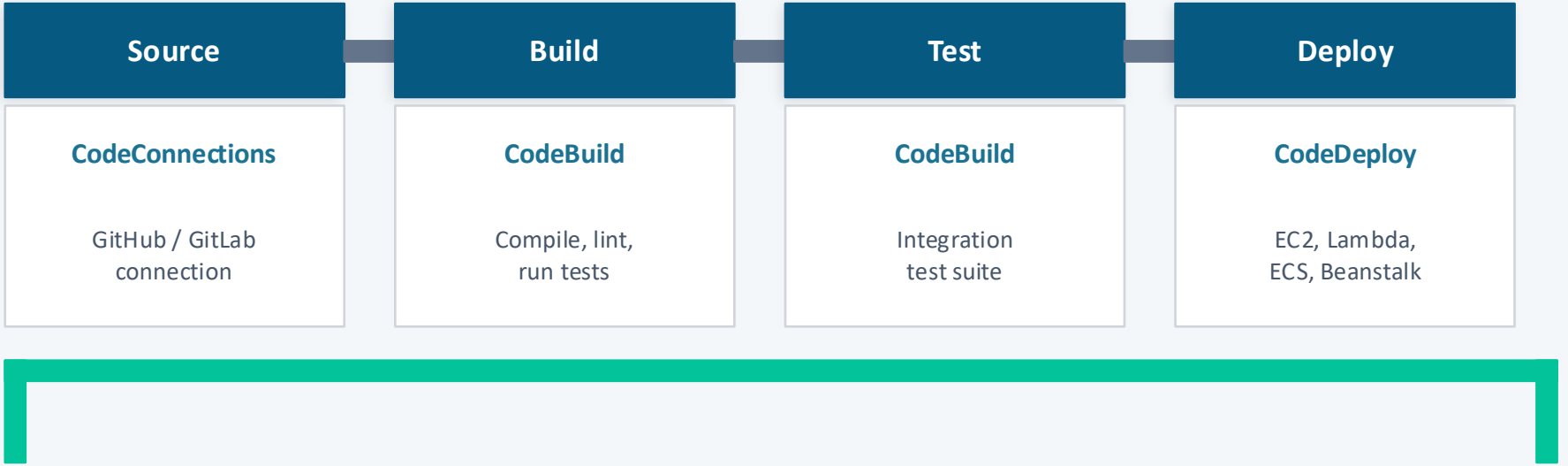
Infrastructure as Code ensures each environment is identical — no more 'works on my machine'

PART 2

AWS CI/CD Services

CodeCommit/Connections • CodeBuild • CodeDeploy • CodePipeline • CloudFormation

AWS DevOps Toolchain



CloudFormation

Infrastructure as Code — provision all AWS resources from templates

AWS SAM

Simplified CloudFormation syntax for serverless (Lambda, API GW, DynamoDB)

Elastic Beanstalk

Platform-as-a-Service — deploy without managing EC2 instances directly

CloudWatch

Monitor logs, metrics, and set alarms for deployed applications

AWS CodePipeline: The Orchestrator

What CodePipeline Does

- Triggered automatically on source change (push to main)
- Connects source → build → test → deploy stages
- Each stage can succeed, fail, or require approval
- Failed stages block downstream stages
- Full audit trail of every pipeline run
- Integrates with GitHub, CodeBuild, CodeDeploy, Lambda
- Parallel action groups for faster pipelines
- Supports multiple deployment targets per stage

pipeline.yml (GitHub Actions equivalent)

```
stages:
  - source:
      provider: GitHub
      branch: main
  - build:
      provider: CodeBuild
      buildspec: buildspec.yml
  - test:
      provider: CodeBuild
      run: npm test
  - approval:
      type: Manual
  - deploy:
      provider: CodeDeploy
      target: EC2 / ECS
```

AWS CodeBuild: Build & Test Automation

buildspec.yml

```yaml
version: 0.2

phases:
  install:
    commands:
      - npm install

  pre_build:
    commands:
      - npm run lint
      - npm test

  build:
    commands:
      - npm run build

artifacts:
  files: ['**/*']
  base-directory: dist
```

Key Concepts

Managed build server
No infrastructure to maintain — AWS spins up containers per build

buildspec.yml
YAML file in your repo that defines install, test, and build phases

Artifacts
Build output uploaded to S3 for CodeDeploy to use

Environment vars
Inject secrets from Parameter Store or Secrets Manager

Parallel builds
Multiple builds can run simultaneously

AWS CodeDeploy: Deployment Strategies

⚡ All-at-Once

Deploy to all instances simultaneously

✓ Fast, simple

✗ Full downtime if deploy fails

Use when: Dev/testing environments

🔄 Rolling

Deploy batch-by-batch, keeping capacity

✓ No full downtime

✗ Two versions run simultaneously

Use when: Moderate risk tolerance

🔵🟢 Blue/Green

Deploy to new fleet, switch traffic

✓ Instant rollback, zero downtime

✗ Double infrastructure cost

Use when: Production — recommended

CodeDeploy also supports Lambda (traffic shifting) and ECS (task replacement)

Infrastructure as Code: CloudFormation

Why Infrastructure as Code?

Reproducible environments

Spin up identical dev, staging, and prod with the same template

Version controlled

Infrastructure changes go through code review like application code

Self-documenting

The template IS the architecture diagram

Rollback support

CloudFormation can revert a stack to its previous state

No snowflake servers

Destroy and recreate instead of patching in-place

template.yml (CloudFormation)

```yaml
AWSTemplateFormatVersion: '2010-09-09'
Description: Web App Stack

Parameters:
  Environment:
    Type: String
    AllowedValues: [dev, staging, prod]

Resources:
  AppServer:
    Type: AWS::EC2::Instance
    Properties:
      InstanceType: t3.micro
      ImageId: ami-0abcdef123456789

  Database:
    Type: AWS::RDS::DBInstance
    Properties:
      DBInstanceClass: db.t3.micro
```

AWS SAM: Serverless Application Model

Built on CloudFormation

SAM transforms into full CloudFormation before deploying — same reliability

Serverless-first syntax

One line defines a Lambda + API Gateway + IAM role that would take 50 lines in CF

Local development

`sam local invoke` lets you test Lambda functions on your laptop before deploying

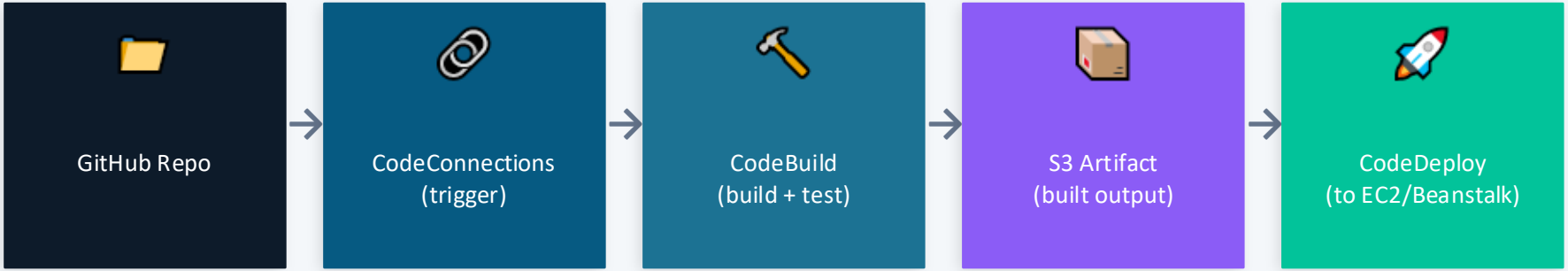
CI/CD integration

Works natively with CodeBuild + CodeDeploy + CodePipeline

SAM template (vs 50+ lines of CloudFormation):

```
Transform: AWS::Serverless-2016-10-31
Resources:
  TodoApi:
    Type: AWS::Serverless::Function
    Properties:
      Handler: index.handler    Runtime: nodejs20.x    Events:
        Api: {Type: Api, Properties: {Path: /todos, Method: get}}
```

Demo: CI/CD Pipeline on AWS



CodePipeline orchestrates all stages above

Demo Tasks

1. Create a GitHub repository with a simple Node/React app + buildspec.yml
2. Connect the repo to AWS via CodeConnections (OAuth to GitHub)
3. Create a CodePipeline with Source → Build → Deploy stages
4. Configure CodeBuild with the buildspec.yml (npm install, test, build)
5. Deploy to Elastic Beanstalk (managed environment — no EC2 config needed)
6. Push a change to GitHub → watch the pipeline run end-to-end

Elastic Beanstalk: Easiest AWS Deployment

You provide: your app code AWS provides: everything else

Auto-provisioned EC2

No manual server setup — Beanstalk picks the right instance type and launches it

Load Balancer + Auto Scaling

Automatically scales up under load, scales down when traffic drops

Managed Deployments

Rolling, immutable, blue/green deployments with automatic health checks

Built-in Monitoring

CloudWatch metrics, logs, and health dashboard out of the box

RDS Integration

Attach a database directly to your environment — lifecycle managed together

Easy Rollback

One click to deploy the previous version if something goes wrong

Perfect for students — skip infrastructure complexity, focus on your app code

GitHub Actions vs AWS CodePipeline

Feature	GitHub Actions	AWS CodePipeline
Trigger	git push, PR, schedule	git push, webhook, manual
Build runner	GitHub-hosted or self-hosted	CodeBuild (managed containers)
Config format	YAML in .github/workflows/	YAML or AWS Console / CDK
AWS integration	Requires IAM credentials setup	Native — no extra config
Cost	2,000 min/mo free, then pay	Pay per pipeline run
Best for	Open source, GitHub-centric teams	AWS-native production apps
Learning curve	Easier — more tutorials	Steeper but more enterprise

For the lab we use CodePipeline — understanding both prepares you for industry.

Key Takeaways

1

CI/CD is not optional

Manual deployments don't scale. Automate early and often.

2

Test ↔ Deploy are linked

Tests run in CI. A failing test blocks the deploy. This is the point.

3

Environments must match

Dev, staging, prod should be identical — use IaC (CloudFormation/SAM).

4

AWS gives you the whole stack

CodePipeline → CodeBuild → CodeDeploy handles source-to-prod.

5

Blue/Green is production-grade

Zero-downtime deploys with instant rollback. Default to this pattern.