

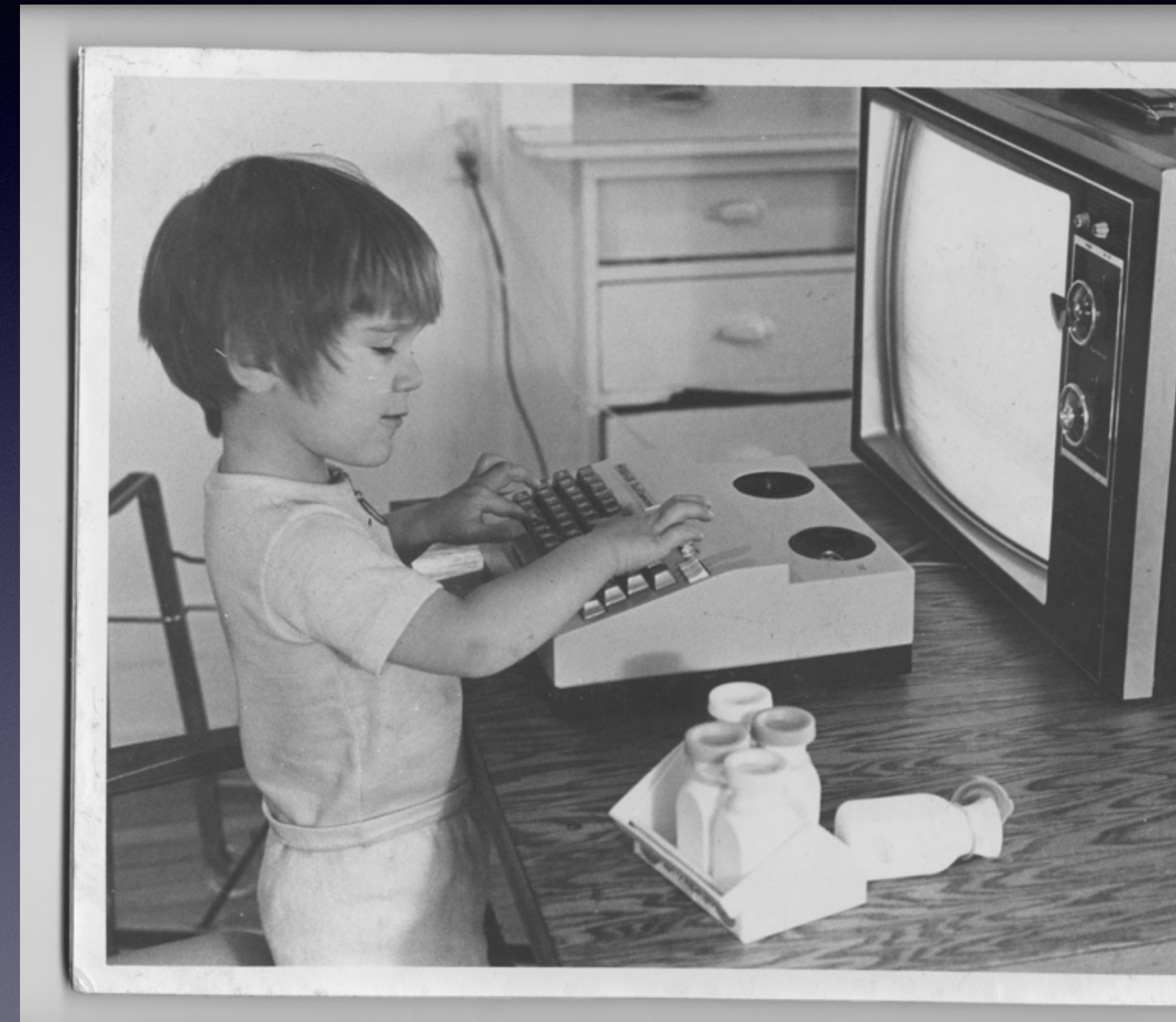
Github, Docker, and AWS: A Basic CI/CD Walkthrough

Presented by: Jon Goldman, CTO - Playbill Inc.

Presented at: PSU MacAdmins 2023

So who is this Jon guy Anyway?

- Started in computers when I was 3...
- With Playbill for the last 12+ years
- Spent 15 years as a Professional Stage Manager in theatre.
- Have been consulting and working in various tech depts. since 1986.
- Been an Apple fan since my first Apple][+ (the Darth-Vader/Bell & Howell Edition!)
- @porthosjon in MacAdmins Slack




About Playbill

- Been around as a brand since 1884
- Had a web presence since the early 1990s, originally on CompuServe
- Currently has 4+ major online properties
- 3 Offices with ~100 Macs and as few Windows machines as I can manage
- Understaffed IT department that requires creative solutions to most problems
- No, I don't get free tickets to Broadway shows, sorry.



What you should walk-away with

- Basic CI/CD Understanding
- Run Book for deploying simple docker-based apps to AWS for Immediate servicing
- Simple security precautions that can keep your app secure for the people that need it only
- List of gotchas that will prevent you from spending hours with your 

Software & Service Resources



Resources used in this presentation

- Mac Apps (Needed)
 - Docker Desktop
 - IDE (or text editor with Github)
- Mac Apps Recommended
 - Clipboard Manager
 - DDev (via Homebrew)
 - iTerm
 - AWS CLI (via Homebrew)
- Cloud Resources
 - Github
 - Docker Hub
- AWS Resources
 - Secrets Manager
 - Identity Access Manager (IAM)
 - Code Deploy
 - Elastic Beanstalk & EC2
 - Optional - RDS for Database
 - Optional - Route53 DNS
 - Optional - Certificate Manager (SSL Certs)
 - Optional - S3 & Cloudfront for Assets
 - Firmware: Duck & Dammit Doll

Software & Service Resources



Presentation Github Repo

Git Hub used for this presentation

[https://github.com/JonGoldmanPlaybill/
macadmin2023-ci-cd](https://github.com/JonGoldmanPlaybill/macadmin2023-ci-cd)



Equipped with it all -> Dive in




Github Setup

- Private or Public Repo
- Set at least 2 branches (main & staging)
-  Setup branch protection for main branch

Docker/Docker-Desktop Setup

- Setup Dockerhub
- Pull at least a generic image of NGINX (what I use)
- 🗝️ Put your credentials in a note or in your Clipboard Manager
- 🗝️ Adding a free account for docker credentials in your build lifts the rate limit on pulling images during the build to prevent errors in case of multiple rapid builds from the same IP.

Project Setup & Structure

- Project Structure
 - `infrastructure` Dir - Reference Files, don't push secure info to repo
 - `docker` Dir - All docker build files, config, and shell scripts
 - `deploy` Dir - buildspec Files for AWS Codebuild
 - `html` Dir - for app files & env files
 - `docker-compose` files should be at project root
-  if you are using a CMS software and ddev (recommended!), configure in the html subdirectory.

Project Gotchas & Considerations

- Decide early on if you want to include your web/project code in the docker image:
 - Pros for inclusion: More immutable and portable between environments, local building not really necessary, not dependent on O/S permissions
 - Cons for inclusion: Every build requires all steps (although docker will slim some of this), slower to deploy changes that are simple code, possibilities for volume dismounts and filesystem permissions issues
- Pick x86 or ARM planning early on and stick with it (prevents conflicts later). *Note that ARM can be considerably cheaper on AWS.*
- Utilize Notes & Secure Storage locally to make sure you don't lose credentials
- Tag, Tag, Tag: AWS resources can sometimes be named, but tags are the best way to keep track of your resources across services






Hey Jon, less talk– more deploy




🐦 *Remember: Generative AI can help, but it is likely out of date for the latest options*

Lets dive in to get that “hello world” nginx app running on AWS



AWS Setup

- Setup IAM User and API Key ( Copy to Clipboard and setup notes)
- Configure local AWS CLI (installed via homebrew - `aws configure`)
-  Keep tabs open with all the relevant AWS services or you're going to drive yourself batty getting back to where you were
-  & : IAM is the worst service to use and will cause you no end of headaches, but once setup it should be solid.
-  Make sure you write up a tagging strategy for AWS resources so that you can quickly find everything related to a project & environment for devOps and billing purposes.


AWS Secret Manager

- Using Secrets makes security tighter and easier at the same time
- Grant your services IAM Roles access to Secrets (either all or some depending on your setup) -  Doesn't mean individual devs have access to the secret info!
- Collect all your external API Keys & Credentials before you start

AWS Secret Manager

- Create Secret Stores
 - Cross Project Main Credentials & settings
 - Project Specific
 - All Environment Credentials & settings
 - Specific environment Credentials & settings
-  Edit using JSON for speed and copying
-  Great options to auto-manage root keys if you use RDS for database
- ***Walk through:*** Create one store as an example with JSON


AWS Elastic Container Registry [ECR]

-  Not as ubiquitous as DockerHub, easier to use ECR to store deployment envs only (can be a pain to access outside of AWS)
- Create an ECR entry for every container in every environment - this will help speed up deployments.

AWS Elastic Beanstalk

- 🗝️ This is the touchiest part of the install as many key options cannot be changed after you create an environment
- Keep a list of the options, and go through it STEP-BY-STEP *every time!*
- 🗝️ Make sure that Roles and Networks are configured before you start
- 🐣 Roles & Networks are the most common trip points, along with the environment type
- 🗝️🗝️ Pre-setup a VPC with subnets and a gateway (internet access)
 - Subnets are important because they *can* limit which types of instances you can use

AWS Elastic Beanstalk





- Optional Services that are useful with EB:
 - RDS: Database Services (MySQL is best compatibility, can use Aurora)
 - ElastiCache: Redis services - sessions and other instant pull up DB services
 - Route53
 - Point a CNAME at your Load Balancer/EB Environment
 - Point a CNAME at your RDS Endpoints (if used) for quick rebuild without deploy
 - Certificate Manager
 - FREE certs using AWS cert manager
 -  if you want a wildcard cert, use `*.domain.tld`
- S3 & Cloudfront for a quick CDN solution for assets (CSS, images, etc.)



Lets Build a Pipeline!

It's not rocket science, but it will rocket-power your workflows to deploy quickly and get those apps up and running.

Make a CodeDeploy Buildspec


-  Much cleaner to keep your deploy “code” in your VCS repo, so that you can undo the inevitable mistakes you make (but you can start directly in the console if you want)
- YAML file ( Remember your indents!)
-  Comment Away! (Modified YAML allowing comments)
- Four Phases Available: install, pre-build, build, and post-build
- Artifacts will be passed to the next CodePipeline Step
-  Remember that you need to install any special languages/extensions that you need to perform the build in the “install” phase.

Make a CodeDeploy Buildspec

- Declare Variables (and load from Secrets) in the env section at the top
- install section: Login to docker
- pre-build: Pull your ECR images (this is where the free docker credentials come in handy)
- build: All your build & push logic here
 - 🗝️ Have a different docker-compose file without a build section (or edit it out with npx) to use as an artifact to allow you to restart the environment easily
 - 🗝️ If your project need an .env file, this is the place to build it (use secrets to stream out options!)
- artifacts: make sure you have setup any files in the build that you need for the deploy step

Build the Pipeline (*console walk-through*)

SOURCE Step

- Use GitHub v2 (must set permissions)
- Pick your branch carefully
-  Don't use auto-deploy until you are sure it is working, and don't use it for production unless you protect the GitHub branch.

Build the Pipeline (*console walk-through*)

BUILD Step

- Use CodeDeploy
- 🗝️🐣 Make sure you match your build environment (x86 or ARM) to your Elastic Beanstalk environment. Sometimes it doesn't matter, and sometimes it will cost you much frustration.
- Minimize the amount of fixed information in the code-deploy console. Try to keep everything in secrets and your buildspec (it will be much easier to troubleshoot changes that break your pipeline when you have VCS)

Build the Pipeline (*console walk-through*)

DEPLOY Step

- Link to your Elastic Beanstalk environment
- 🗝️ If your environment has multiple servers, set rolling deploy to keep you online while you deploy
- 🗝️ If you use a web-app that has session information or forms, highly consider using Redis AWS Service to store. If your sessions are outside of individual servers, then load balancing works much better (probably an entire additional session for this)

Result?

Either a little while with this



Or you get this

Yes, this is an actual police, traffic, and tech support station on the Mekong.



Really. It will either work, or you start combing back through logs to see where you set an option wrong, or had a bad indent in your YAML, etc.

Notes & Tips

- If you can't find the answer quickly, it's probably IAM Permissions
- If you spend more than 5 minutes troubleshooting your Beanstalk environment, destroy it and start again, it will be faster!
- Using a shared load balancer can result in IAM permissions issues many times
- Don't stop/terminate EC2 instances manually, EB will just restart them
- Using a load balancer will allow you to have the SSL (and any security you need) at the edge instead of in the build.
- Include SSM Permissions in your IAM Roles: This allows you to get to an SSH session from within the console/browser

Use Cases

Here are some ideas for uses as opposed to local server deploys with Jenkins/Travis

- Web sites (whether using a CMS or not)
- Tools for your users
 - Email signature generator
 - Intranet
- Tools for IT
 - IT Secure information storage
 - MDM Enrollment profiles (secured with SSO)

The best part of this is that since you can place a load balancer with SSO auth in front of your app, you can secure anything without worrying about network permissions and the like

THANK YOU FOR LISTENING!

Github, Docker, and AWS:
A Basic CI/CD Walkthrough

Special Thanks to:

Playbill

(for paying me to come)

FEEDBACK: <https://bit.ly/psumac2023-120>

Presented by: Jon Goldman
@porthosjon in MacAdmins Slack
Presented at: PSU MacAdmins 2023

