Serverless, Schmerverless!

Philly Area Cloud & DevOps Meetup Edition

Ken Rimple, Chariot Solutions

July 29, 2020

CHARIOT SOLUTIONS

About me...

- Director of Training and Mentoring, Chariot Solutions
- Present about emerging / emergent tech topics
- Mentor consultants, clients in a wide array of technologies
- My teams develop SPA apps against AWS with ECS/Fargate

CHARIOT SOLUTIONS

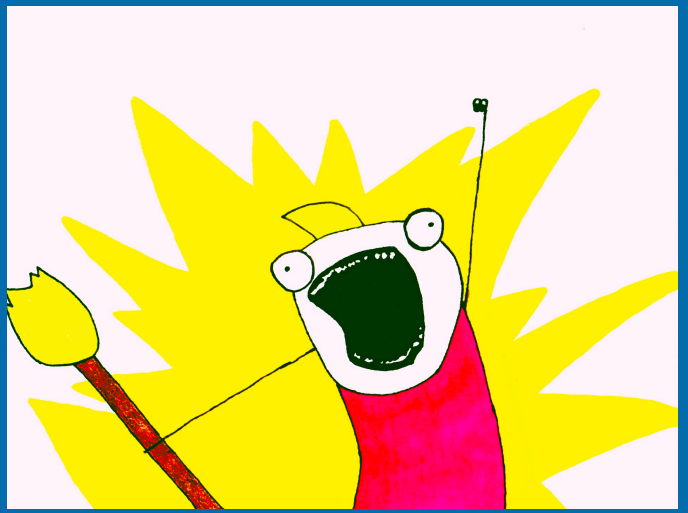
My view (personal, NOT of Chariot Solutions):

Lambda-based application stacks do not yet provide me enough developer velocity and productivity compared to container-based ones

CHARIOT SOLUTIONS

The “delight a developer” test…

- I just deploy functions?
- The functions can access all of AWS?
- COOL!



Lambda all the things!!!

In practice…

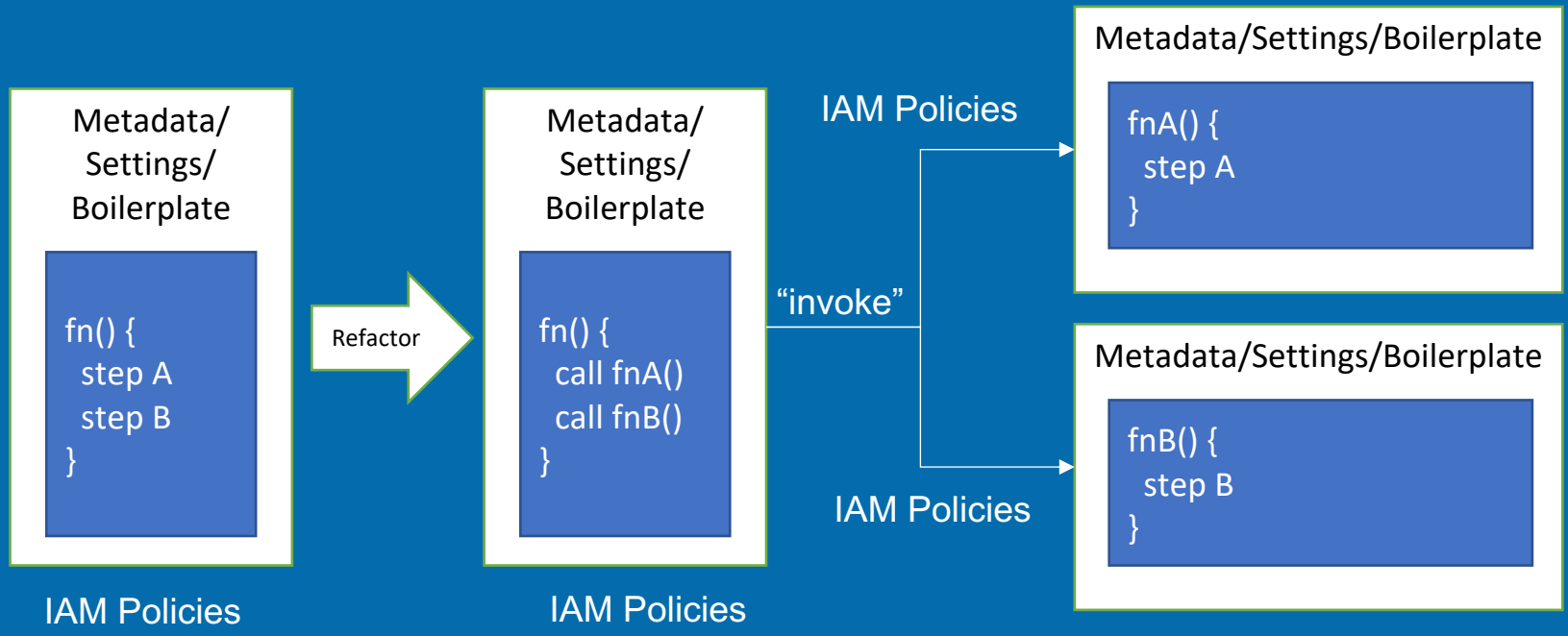
CHARIOT SOLUTIONS

Too many concerns!

- For a single web-facing Lambda, we need:
  - The Lambda function itself
  - Infrastructure to expose if front-facing
  - A good naming mechanism for inside callers
  - An IAM Role with Policies that work for the Lambda
  - Settings for the Lambda for vCPU and Memory to tune its size
  - To figure out how to inject shared resources, and when to attach/disconnect them
  - Hopefully a CloudFormation, Terraform or ADK script to deploy the Lambda again!

I just want my function!

Lambda Refactoring Expands Metadata footprint



CHARIOT SOLUTIONS

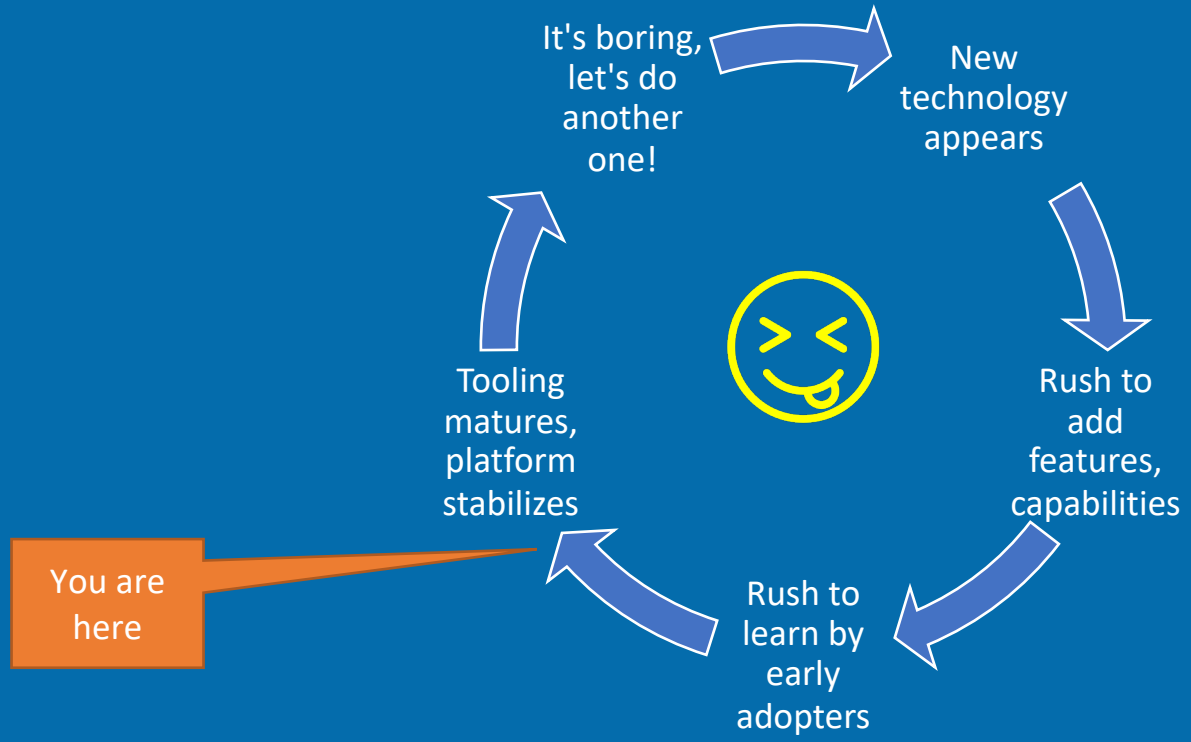
We've seen all of this before

- Java Enterprise Edition
  - Deployment Descriptor: started as complex XML-configured objects that tuned # of instances, how long to keep, etc.
- Spring defined the simple container (just an object that holds "beans") and let them all share the same thread pool
  - Separate config from code, simplified to annotations and DSL
  - Abstracted away difficult services and shared across code easily
- Many Java EE projects converted to Spring for ease of development and refactoring
- Ruby on Rails and Convention over Configuration…

More Lambdas = More Infrastructure

CHARIOT SOLUTIONS

The never-ending story…



CHARIOT SOLUTIONS

Additional Lambda App challenges…

- Is there an “application” here? (code organization)
- How big is your Lambda? Libraries included…
- Connection pooling (temporary problem)
- Tuning – your app is now diffuse across many deployed functions, and tuning just got harder
- Monitoring – need external tools – a whole universe of vendors are willing to assist here for the right price

Solving infrastructure problems too much… instead of building features

- Managing fine-grained Lambda CloudFormation stacks
- Sizing right and related cost concerns
- Pinning Lambdas for performance?
- Tooling is a marketplace, not native features
- You are deeply tied to AWS (or Azure or whatever you use)

Use Containers instead…

- We are all used to running servers
- We can keep all related code within the same container boundary
- We have one thing to tune per container
- We can monitor/debug the whole container
- We can scale at the container level
- We can use libraries to share code between containers

Containers are Portable

- Run anywhere Docker runs
  - In your local machine
  - In company hardware
  - On cloud platforms
- With proper abstraction AWS services can be replaced later with others (queues, databases, caches)

Container runtimes on AWS

- Elastic Container Service (ECS)
- Elastic Kubernetes Service (EKS)
- Elastic Container Registry (ECR)
- The Fargate runtime – Serverless runtime for ECS or EKS
  - Push the serverless infrastructure below your concern
  - Pay as you go for only what you spend
  - Tune per container, not per function

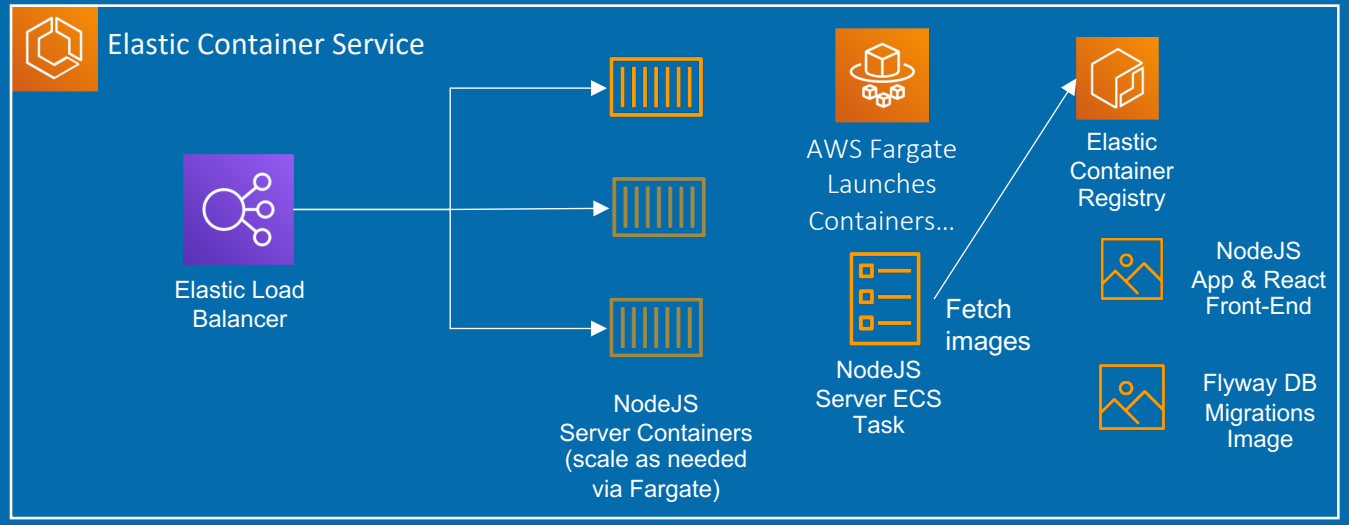
Not as easy as it sounds

- You still have to stand up the developer services you need…
- But developers can share things (like S3 buckets, Cognito, SES, etc.) to save some money
- And developers can be productive immediately without solving all problems to run in the cloud right away!

Experiences on two projects…

CHARIOT SOLUTIONS

NodeJs & React Application Stack



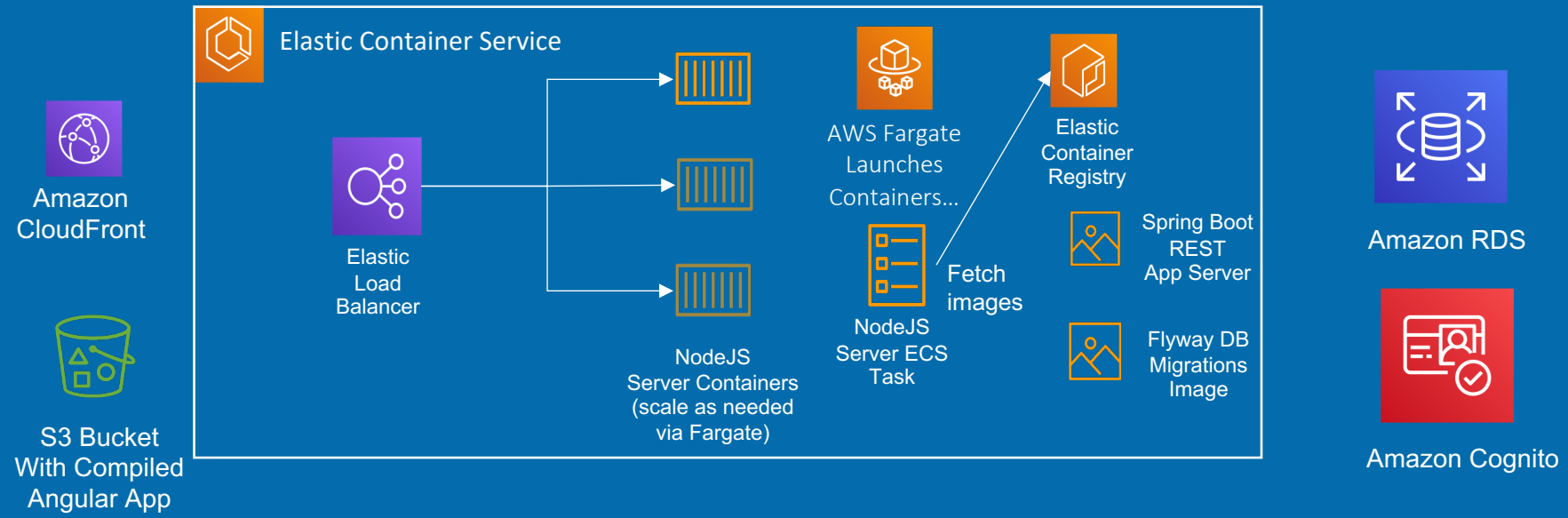
Amazon RDS

Amazon Cognito

Tools used: AWS CodeBuild/CodeDeploy, CloudFormation, ”CreateReactApp”, “React”, “Storybook”, NodeJS w/Express, Hosted Cognito for Authentication, Postgres in RDS, Docker for running Flyway migrations and Postgres locally

CHARIOT SOLUTIONS

Spring & Angular Application Stack



Tools used: AWS CodeBuild/CodeDeploy, CloudFormation, Angular, Storybook, Spring Boot, Hosted Cognito for Authentication, Postgres in RDS Aurora Serverless, Docker for running Flyway migrations and Postgres locally, Cypress for running web tests within CodeBuild

CHARIOT SOLUTIONS

Benefit: Developer productivity

- Developers can use their native tools
- They can stand up application servers locally or in Docker
- They can run databases and other services via Docker for a zero install developer setup
- They don’t need to use AWS (except for some cases such as authentication with Cognito)
- They don’t need to pay for infrastructure just to run the application

Benefit: Iterate on additional projects more easily

- IF you build a body of CloudFormation scripts and project templates
- Learn to use CodeBuild for building Docker images to deploy to ECR
- ECR, ECS w/Fargate, CloudFront/S3 hosting separates front/back ends nicely
- Swap in whatever web/app platform you want without worrying about the whole stack falling apart

It’s still hard

- CloudFormation, troubleshooting cloud problems must be endured, not enjoyed
- Wrestling networking challenges can be trying and difficult
- But the whole team does not need to play
- Focus on infrastructure infrequently, feature development frequently!

Questions?

CHARIOT SOLUTIONS

Thank you!

- More videos/sessions (incl. ETE 2020) at

  youtube.com/chariotsolutions

- Interested in training/mentoring in AWS?

  https://chariotsolutions.com/aws-training

CHARIOT SOLUTIONS

Container App Refactoring does not change metadata footprint

- Only need additional policies for new AWS features
- Policies defined at the app / container level
- Can refactor at will
- A single set of metadata for the app