

Eight Weeks to Production with a Cloud-Native Web App



Who am I: Ken Rimple (@techcast)



- Started at Chariot in May 2007
- Director of Training/Mentoring Services
- Host of the Chariot TechCast and TechChat Tuesdays podcasts
- Present on emerging technologies at conferences
- Work on projects in AWS w/Angular, React, Spring, Node/Express, etc.

Chariot's Specialties

- Application Architecture
- Front-end SPA and Mobile dev
- Data Engineering Services
- Cloud dev services



Chariot created and hosts Philadelphia's Emerging Technologies for the Enterprise conference, and we sponsor user groups, conferences

About this talk

Goals

- Review the major tasks in developing a project to deploy to AWS
- Leverage containers for local development
- Deploy containers via ECS to AWS
- Discuss project and AWS configuration tasks

Assumptions

- User Experience / basic wireframes have been created
- Agile project w/strong issue tracking
- Documentation committed to the project as markup code
- First week: beginning of development effort

Provisos

- Your project is not my project
- Likely the timeline is not going to fit all circumstances
- This assumes a greenfield application
- The scope of this example is small
- "Production" means project is deployed in production and begins final testing



Migrations have additional considerations and take more time than expected!

The Proposed Patterns

- Docker and/or local app servers for local development
- AWS deployment via configuration-as-code
- Not focusing on Lambda application functions for this presentation
- Secure application via Cognito and app roles, JWT

Team Required Skills

Front-end Expertise

- Expertise in a SPA framework such as **Angular**, **React**, or **Vue**
- Expertise in **CSS** and a CSS framework such as **Bootstrap**, **Material Design**, or able to create / extend a vanilla template
- Comfort with configuring libraries with [npm](#) and [package.json](#)
- Calling web services via framework API or library (**Axios**, **Fetch**, etc)
- Session management via **JWT Tokens** and the **AWS Amplify SDK**

Application Server Development

- Expertise in building application servers with REST endpoints in a language / platform of choice
 - Java/Spring Boot
 - Node/Express
 - Python/Django



Ensure support for your language version via a Docker container before you begin using it...
CodeBuild makes this a bit more complex.
(CodeBuild language support)

Docker

- Use [docker-compose.yml](#) to develop a stack
- Understanding of Docker hostnames and networks, port access and forwarding
- Use of [.env](#) files and environment variables for settings
- Comfort using [Dockerfile](#) to build custom docker images



Amazon's Elastic Container Service is Docker in the cloud, so skills in Docker get translated to production success

AWS Skills

- **AWS CLI** configuration basic, use profiles
- Use **CloudFormation**, **Terraform** or **CDK** to set up your resources
- Configuring **IAM Roles/Policies**
- Distributing web applications via **CloudFront** and **S3**
- Deploying application servers with **ECS**
- Set up CI/CD via **AWS CodeBuild/CodePipeline**
- Configuring security with **Cognito**
- Configure app secrets via **Secrets Manager**



Get certified! It helps give you an overall view of AWS services

Division of Labor

Assuming a 4 person team:

- One team member focuses primarily on AWS infrastructure
- Two developers focus on application features
- One team member focuses on integrations w/AWS APIs



This example assumes one engineer will take the job of scrum master and assumes the tasks are light enough to be done within eight weeks of development

Timeline

- Pre-project inception planning
- Sprint 1 - Initial AWS scripts, Docker config, application setup, feature dev
- Sprint 2 - Build out AWS developer resources (S3 buckets, etc), Data model, feature dev
- Sprint 3 - Production install on AWS, establish CI pipeline, feature dev
- Sprint 4 - Platform guide, production delivery, complete features, build web tests

Deploy to AWS as soon as possible! App
📌 Developers will become comfortable in Docker and want to stay within it.

Week 0! Project Inception and Pre-Launch Meeting

- Set up Git Repository, JIRA/issue tracker before 1st sprint
- Set up AWS Accounts
- Follow organization preferences (language, frameworks, database type, configuration - CloudFormation -vs- Terraform is a common debate, etc.)
- Determine any security limitations or concerns
- Identify integration points with other systems



Identify all issues as early as possible and place them in your issue tracker.

Sprint 1

Configure Application
Server (Docker
execution)

Database modeling, Core Libraries and
settings in Docker

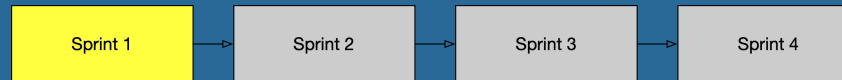
Begin AWS platform scripting

Initial SPA
configuration

UI / CSS / Navigation model

Feature 1 rough out

Set up development resources ASAP!



- Configure the development project skeleton on day 1
 - Nothing needs to be perfect
 - Nothing needs to be integrated
 - Security and UI design not yet necessary
 - Create a simple "Hello World" skeleton with a modest UI

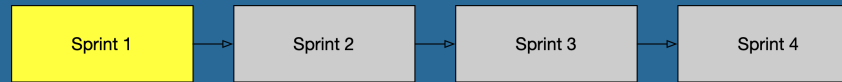
i This frees up the rest of the sprint for roughing out the UI, backend, initial meetings with the clients and sprint planning.

Pro Tip: Run your SPA from your desktop



- Run your web front-end locally, not in Docker
 - You can debug it locally
 - You can add proxy servers for backends
 - You can have it automatically reload when changes are made with hot redeloys
 - You can use local tooling
 - You'll deploy on S3 in AWS anyway

Define UI layout and components

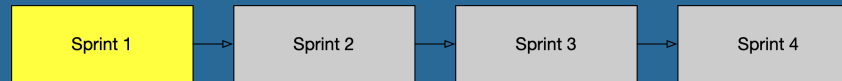


- Pick your CSS framework and stick with it for the project
- Roll out your initial UI for the SPA without tuning the colors/fonts/layout
- Key features to focus on up front
 - Navigations and routes (stub out views at first)
 - Header and Footer
 - Sign in / out buttons and fake UI for same



Twitter Bootstrap is popular for a reason. It allows for lots of customization and tailoring, while remaining very accessible to beginners.

SPA infrastructure setup



- Http API such as Axios, Angular HTTP, Fetch in this sprint
- Build a simple API calling layer in your app
- Use *site-relative* URIs ('/api') for the backend so you can deploy this anywhere w/o changing the endpoint
 - Avoids the ugliness of CORS settings
 - SPA dev tools have proxy services for this
 - Later you can expose /api via a CloudFront Origin

Database configuration on Docker during development

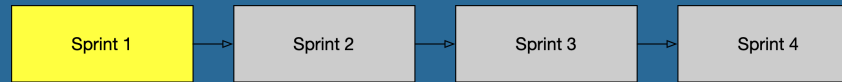


- Running relational databases for your developers on AWS can be cost-prohibitive over time
- Run a Docker container for your developer databases
- DynamoDb also has a Docker container



Use environment variables for database connection settings

Flyway: a schema management tool

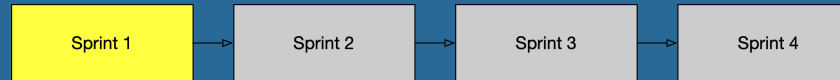


- Manages relational database schemas
- Run from a Docker container
- Using [docker-compose](#) it will synchronize your schema when you bring up the stack



Run Flyway docker container from CodeBuild to execute on RDS

Begin AWS Infrastructure build-out and developer config

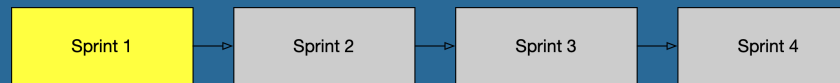


- Configure w/AWS CDK, AWS CloudFormation or Terraform
 - Version your infrastructure scripts
 - Use [Amazon's CDK](#), [CloudFormation](#) or [Terraform](#) templates
 - Break your configuration up into several chunks



Experiment in the console, but then use code to deploy for developers / production.

Managing AWS credentials

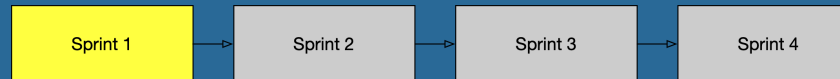


- There are three different "personalities":
 - Provisioner
 - Developer
 - Production Deployment



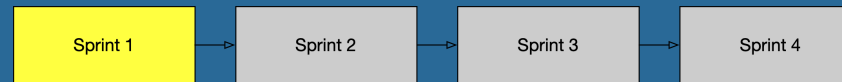
Grant permissions to **IAM Roles**, assign those roles to the personalities that need them

Consider abstractions for some services



- If you don't have an implemented service for features in your UI yet, make up a fake response and keep moving!
- Stub out AWS API calls (let them pass for now), to get customer feedback
- Then go back and begin wiring in AWS services

End of Sprint Artifacts



- Initial Git project and GitHub shared repo
- Documentation home folder
- Basic project layout
- Simple web front-end calling an app server
- AWS accounts and development sandbox
- Database via Docker and database schema migration strategy
- Established developer client tooling (what IDE, tools, etc)

Sprint 2

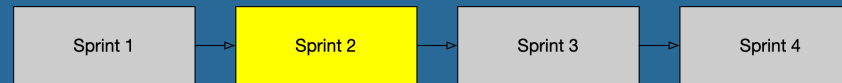
Elaborate AWS configuration

Feature #1 Completion

Feature #2 Elaboration

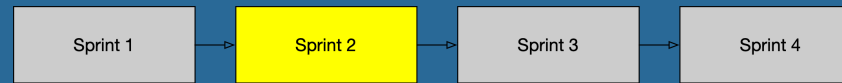
AWS security configuration (Cognito, JWT, etc)

Sprint 2 Activities



- Begin to work on tasks in parallel, for example:
 - Application security configuration via **Cognito, JWT**
 - Iterate on AWS services
 - Docker app should be able to access AWS services
 - Application development underway

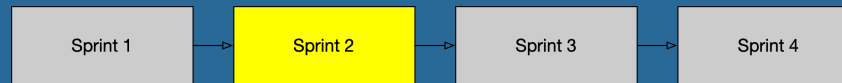
Identifying AWS Services



- Some AWS services reveal themselves up front
 - A customer may need to upload large files (S3 or SFTP)
 - Users may need email: (Simple Email Service)
 - Almost everything needs a database...

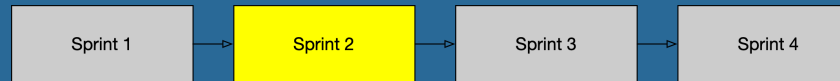
i Some are less obvious, such as monitoring and logging. You want to identify the first services you need to integrate.

AWS Database options



- This is a good time to provision AWS-hosted databases
- Use **RDS** to host most common databases
- Consider **Amazon Serverless Aurora** to keep costs down
 - Great for mostly low-frequency traffic with spikes in usage
 - Good for development databases

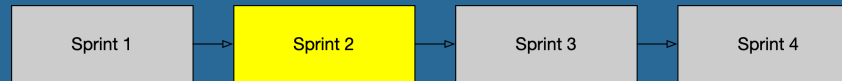
Configure App Security with Cognito



- Using the provisioning tool, add:
 - An AWS Cognito User Pool
 - An AWS Cognito User Pool Web Client
- AWS Amplify provides an Auth SDK for connecting web and mobile applications to Cognito
- Make sure to verify your JWT token sent to the app server really came from your Cognito server (see [JWT claims and Cognito](#))

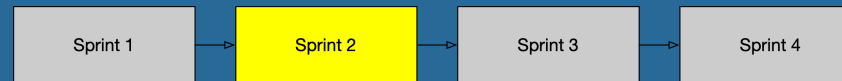
User Pools provide *authentication*, and **Identity Pools** provide AWS IAM permissions for AWS resources.

More feature elaborations



- You will need to build up UI elements, database schema elements and services
- Now this is easier, given you have:
 - a database and migrations configured
 - an application server
 - a web application
 - a navigational mechanism

Major deliverables for Sprint 2



- Developers using AWS services
- Working stack on AWS including security, database
- Improved UI via the CSS framework
- More complete navigation
- Integration w/Cognito for security
- Delivered the first major feature
- Elaboration of the second feature

 You will gain velocity after this sprint

Sprint 3

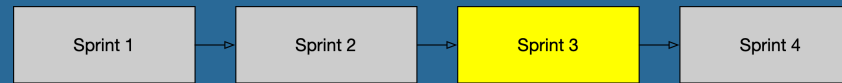
Install production AWS resources and establish CI pipeline

Feature 3 Elaboration

Feature 4 Elaboration

Feature 2 Completion

Begin customer deployment guide, feature development

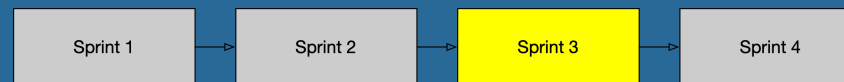


- By sprint-end:
 - Feature 2 should be complete
 - Features 3 and 4 should be in progress or near completion
 - Platform Installation guide, scripts, should be drafted for final production installation
 - The application should be deployed in both development and production AWS accounts



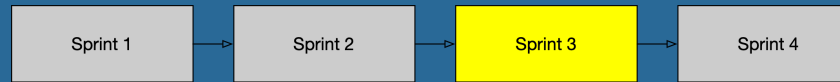
Finish the guide now, save stress at the end of the project

Developers: Just keep moving!



- Nothing new in the development process
- One of the other two developers could join in on feature development
- Any issues or updates to earlier delivered features can be performed here

Set up a CI environment

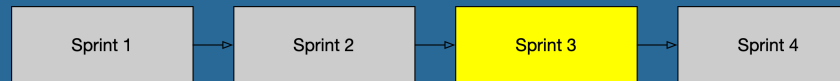


- Setting up CI in this sprint prepares for handoff to production team
- Two of the options:
 - CircleCI has support for AWS via Terraform
 - AWS CodeBuild/CodePipeline/CodeCommit/CodeDeploy runs natively on the AWS platform



Using CodeBuild provides native access to AWS resources but can be more complex

Begin documenting installation steps

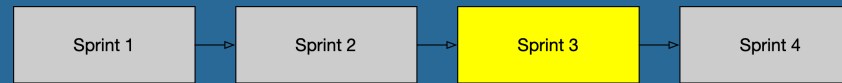


- Generating the Route53 Domain Name before installation
- Deploying ECS for the first time: deployment may stall
- Managing credentials and other info via AWS Secrets Manager
- Injecting config in ECS Task Definitions w/Secrets Manager
- Update secrets during runtime, managing ECS instances
- Triggering a build / deploy with AWS CodeBuild, ECR



Document every novel interaction. This is invaluable information.

Major Deliverables for Sprint 3



- Setup / Management guide
- Running ECS environment
- Running CloudFront /S3 web hosting environment
- Running CodeBuild CI environment
- Feature 2 completed
- Features 3 and 4 in demoable state



May or may not match your reality. Are you in production and rounding the corner on feature completion?

Sprint 4

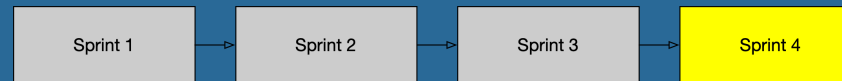
Deliver AWS platform setup guide and install in customer production environment

Feature #3 Completion

Feature #4 Completion

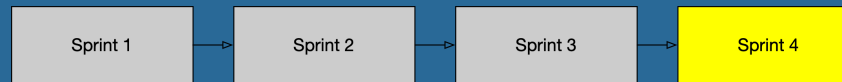
Build web tests w/Cypress and deliver w/CI pipeline

Coming around the mountain...



- Wrap up features 3 and 4
- Set up UI tests with a tool like Cypress
- Begin end-user testing in earnest
- Tune the platform, run stress tests

The benefits of Configuration-as-Code

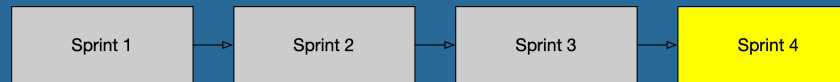


- You begin the project by delivering developers features in AWS via your CaC platform
- You set up your application stack with code
- You externalized your environment variables
- Developers built their code in Docker, not in the cloud
- Docker in the cloud works like Docker locally, so developers can monitor it
- Your CI effort uses CodeBuild to automate cloud deployments and operations like migrations



Containers make cloud development more direct

Miscellaneous Tasks




- Don't forget to use logging in your applications
 - Docker container logs are output to CloudWatch automatically in ECS
 - Use configuration to determine where these are organized
- Place CloudWatch Alarms on major services such as RDS, containers

Wrap-up and final thoughts

- Your app isn't our app
- This just shows relative timing for tasks that enable productivity
- You don't have to use ECS (try EKS, even Lambdas, but realize the differences in development workflow)
- You don't have to use SPAs (just use NuxtJS, Next.JS, Gatsby, etc)
- Look in to Docker containers for Lambdas, ECS Anywhere in the most recent AWS announcements

Thank you!

- Feedback: krimple@chariotsolutions.com
- Other media:
 - Chariot's blog: <https://chariotsolutions.com/blog>
 - Chariot, Conference vide sessions from ETE, other conferences: <https://youtube.com/chariotsolutions>
 - Join us for ETE 2021 - check [@chariotsolution](https://twitter.com/chariotsolution) and <https://phillyemergingtech.com> for details coming soon

We're available to help you with AWS
 development as an AWS Partner -
chariotsolutions.com/contact