SUMMIT ♥s docker

Michael Irwin
September 14, 2016

What is SUMMIT? (marketing version)

- Joint venture between VPs of Research, IT, and Finance
- “More Research. Less Paperwork.”
- Online portal to manage research
  - Proposals (budgeting, compliance, etc.)
  - Approvals
- Collaboration Tool
  - In-team
  - Office of Sponsored Programs to Proposal Team
- Document Manager

What is SUMMIT ? (technical version)

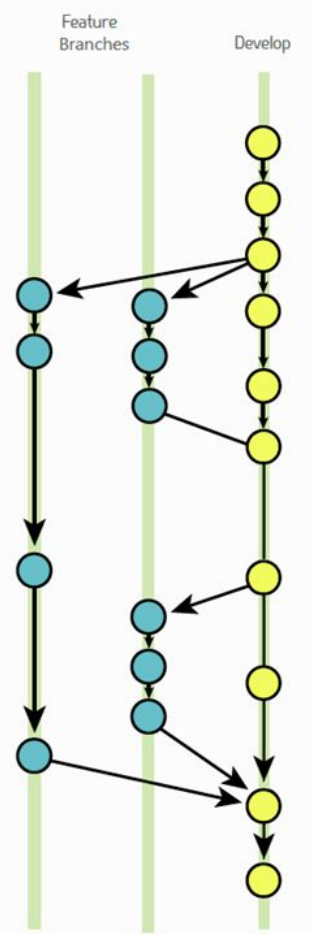
- Web-based application
  - Server-side - Java EE7 using Wildfly
  - Client-side - AngularJS 1 application
  - WebSockets keep all connected clients in-sync through published events
- First official Central IT project running in AWS
- First high-profile production application using Docker

Quick background...

- Eight years in the making
- A more “agile” attempt started just over two years ago
  - First commit on Aug 2014
  - First user on the system in January 2015
- Team consists of…
  - 4 developers
  - 2.5 functional team members, including Product Owner
  - .5 UX expert

How we do development...

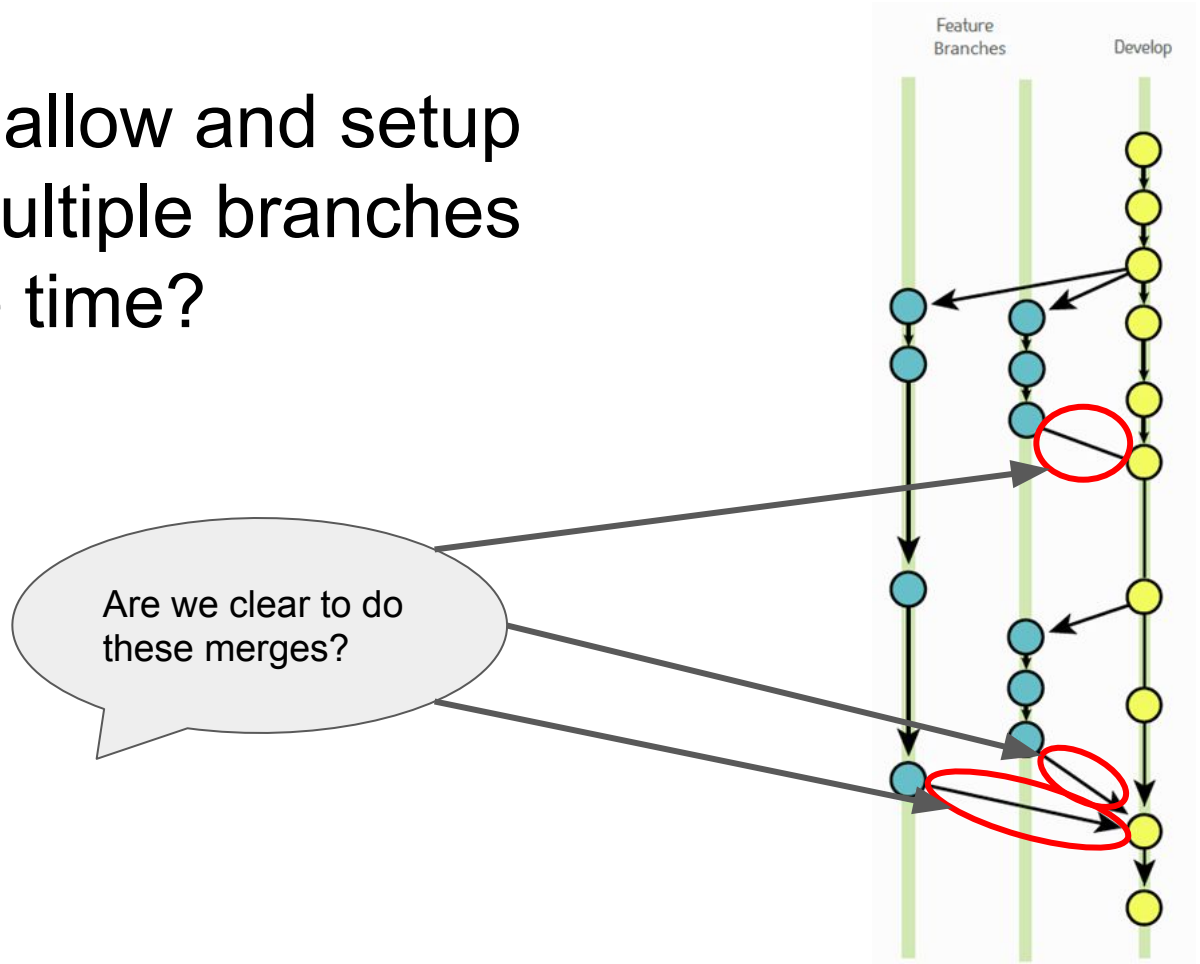
1. User story is created - CREST-1234
2. CREST-1234 is put into sprint
3. New Git branch is created named CREST-1234
4. Development done on branch and frequently pushed
5. Once work and unit/functional tests are completed, User Acceptance Testing (UAT) is performed
6. Once UAT passes, branch is merged



How we do builds...

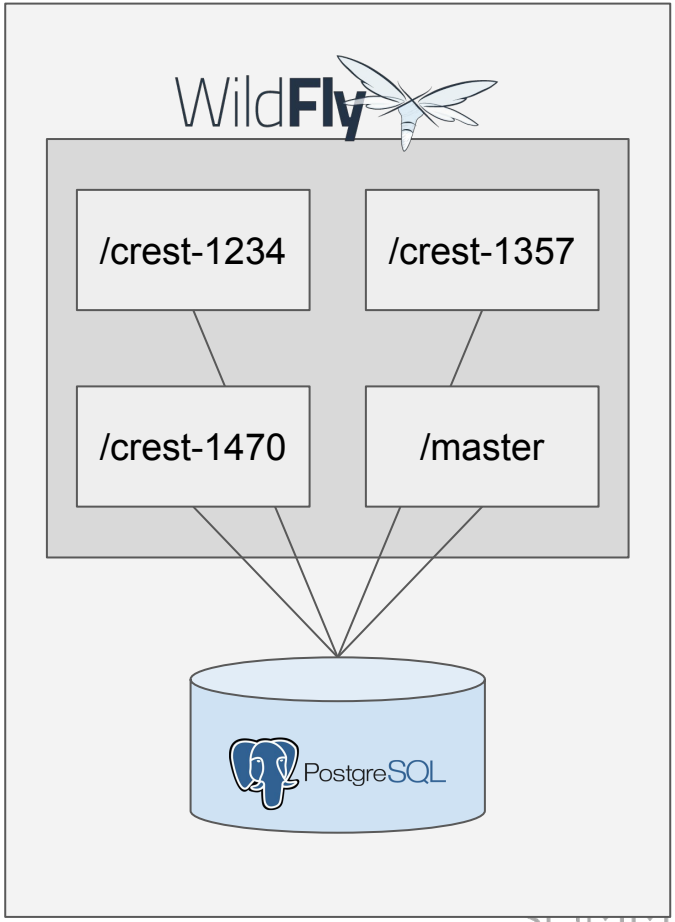
- Currently using Jenkins for CI/CD build server
  - Working to migrate to GitLab CI, but that's for another time...
- Builds occur on every push of code
- Runs on a machine that we'll name `qa.summit`
  - Yes… builds are running on same infrastructure as the QA environment

How do we allow and setup testing of multiple branches at the same time?



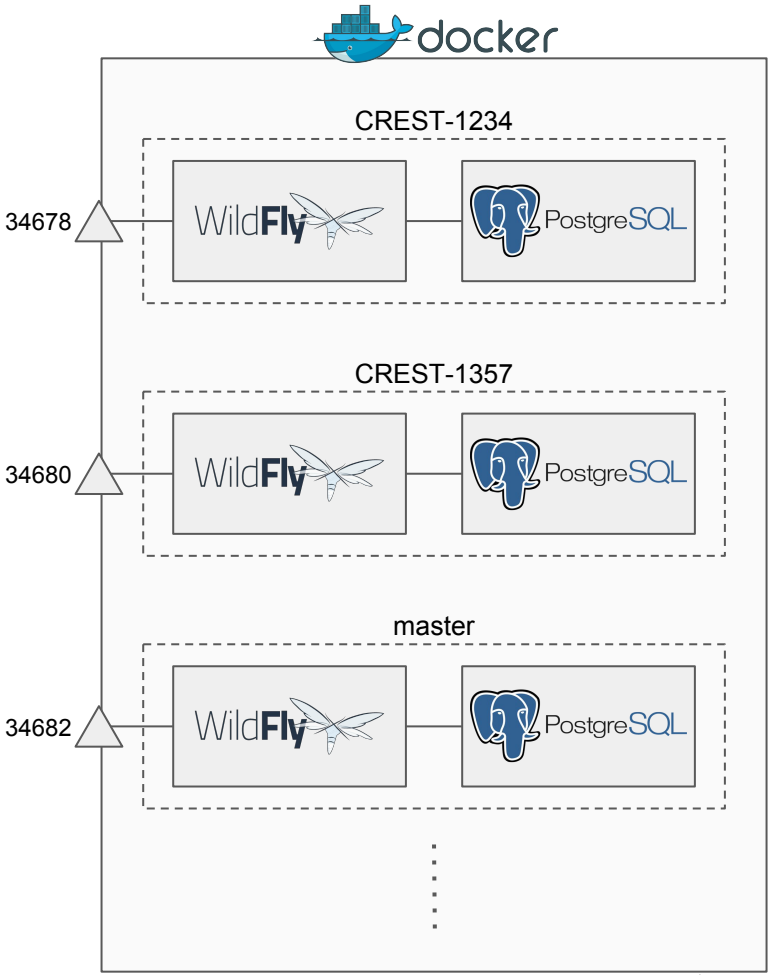
QA Round One - Dec 2014

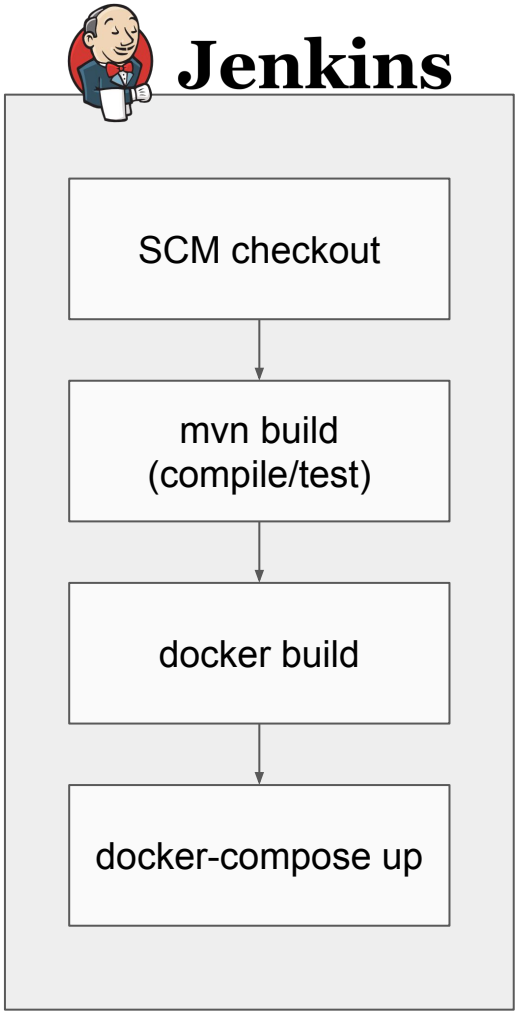
- Run single Wildfly container and single DB
- Namespace each branch
  - Has its own database - CREST-1234
  - Deployed into own context root - /crest-1234
- Pros
  - Was able to automate using Maven (it worked!!)
  - URLs were legible (what branch am I in?)
- Cons
  - LOTS of setup needed before running
  - Many deployments on a single container
  - Bound to a single host/machine
  - Updating deployment to new version was messy



QA Round Two - Oct 2015

- Transitioned to Docker
  - Container for app linked container for db
- Dynamic app port allocation
  - Exposes container's port 8080 on to host using next available port (Docker chooses)
- Pros
  - Isolated environments
  - No runtime config differences
  - Crazy easy teardown
- Cons
  - Ports in URLs are ugly (what branch am I in?) and cause browser problems





Jenkinsfile

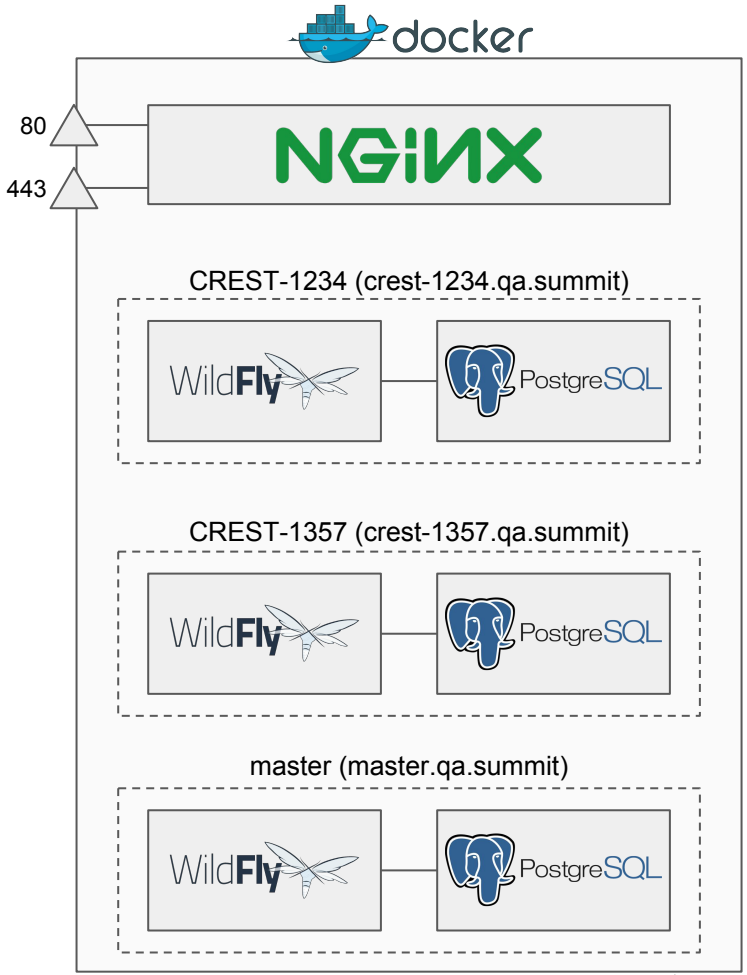
```
node {
  stage 'checkout'
  checkout scm

  stage 'build'
  docker.image('maven:3.3.3-jdk-8').inside {
    sh 'mvn -B clean install'
  }
  stash includes: 'target/*.war', name: 'warFile'

  stage 'docker build'
  dir('docker') {
    unstash 'warFile'
    docker.build "summit/qa:${env.BRANCH_NAME}"
  }

  stage 'start qa'
  dir('docker') {
    sh "docker-compose up -p ${env.BRANCH_NAME} -d"
  }
}
```

docker-compose.yml

```
summit:
  image: summit/qa:${BRANCH_NAME}
  ports:
    - 8080
    - 9990
  links:
    - postgresql
  environment:
    BRANCH: ${BRANCH_NAME}
postgresql:
  image: postgres:9.5
  environment:
    POSTGRES_USER: summit
    POSTGRES_PASSWORD: summit
    POSTGRES_DB: summit
    BRANCH: ${BRANCH_NAME}
```

QA Round Three - Jun 2016

- Switch to use `jwilder/nginx-proxy`
  - Container-aware Nginx reverse proxy
  - Uses environment variables for setup
- Each deployment on own subdomain
- Pros
  - Still all benefits of running in Docker
  - No more browser/cookie issues
  - Proxy also allows other apps to be deployed in environment (Nexus, Jenkins, etc.)
- Cons
  - Still living on one host right now



docker-compose.yml for nginx proxy

```yaml
nginx-proxy:
  image: jwilder/nginx-proxy
  restart: always
  volumes:
    - "/var/run/docker.sock:/tmp/docker.sock"
    - "/location/to/certs:/etc/nginx/certs"
  ports:
    - "80:80"
    - "443:443"
```

docker-compose.yml

```yaml
summit:
  image: summit/qa:${BRANCH_NAME}
  ports:
    - 8080
    - 9990
  links:
    - postgresql
  environment:
    BRANCH: ${BRANCH_NAME}
    VIRTUAL_HOST: ${BRANCH_LOWER}.qa.summit
    VIRTUAL_PORT: 8080
postgresql:
  image: postgres:9.5
  environment:
    POSTGRES_USER: summit
    POSTGRES_PASSWORD: summit
    POSTGRES_DB: summit
    BRANCH: ${BRANCH_NAME}
```

Round-up

- Using Docker for feature branch testing has been fantastic!
  - Quick setup/teardown
  - Much easier to automate
  - Completely isolated and consistent environments
- What’s next?
  - Scaling out to more than a single machine (auto-scaling??)
  - Include functional test suite with automated builds

Questions??