Developing and Testing Java Microservices on Docker

Todd Fasullo
Dir. Engineering

Agenda

Who is Smartsheet + why we started using Docker

Docker fundamentals

Demo - creating a service

Demo - building service + Docker container

Demo - deploying service for testing

Versioning containers, logging, local dev environment, etc

Tips + future plans

Smartsheet

The collaborative work management tool for businesses of all sizes

82,000	Paying Organizations
8,000,000	Users
300	Employees
190	Countries
45%	International Revenue
10	Years Old

Tech Partners

Google EVERNOTE DocuSign Office 365 salesforce

Customers

airbnb Bayer Colliers INTERNATIONAL Extreme networks

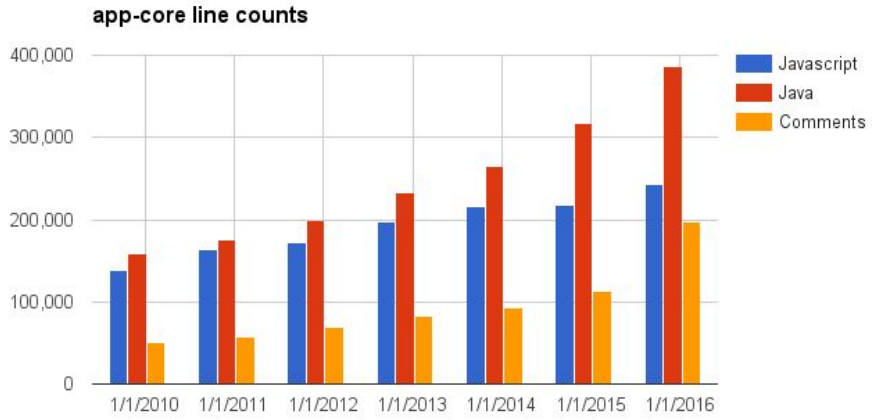
GROUPON NETFLIX PEARSON unisource

Technical Background

Primary Technologies

- Java/Tomcat
- Javascript
- MySQL Data Store
- Native Mobile iOS/Android
- Hosted in redundant colo data centers

Monolithic app with migration to services starting in 2014



Our Nomenclature

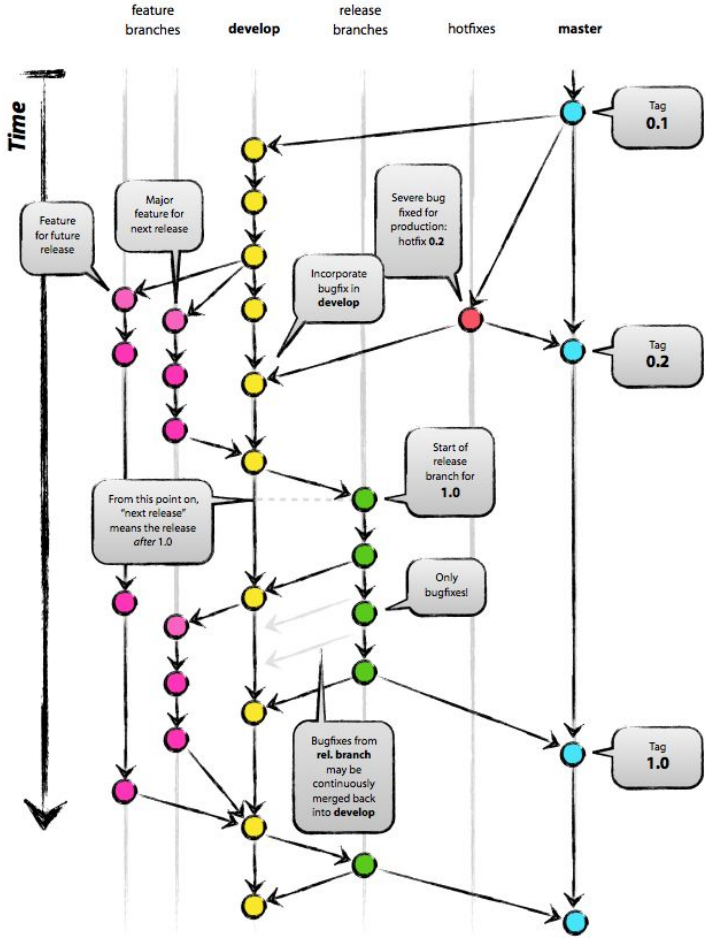
apps + services

- users interact with apps
- apps talk with services

git-flow low style development for app-core

linear (semantic) versioned development for everything else

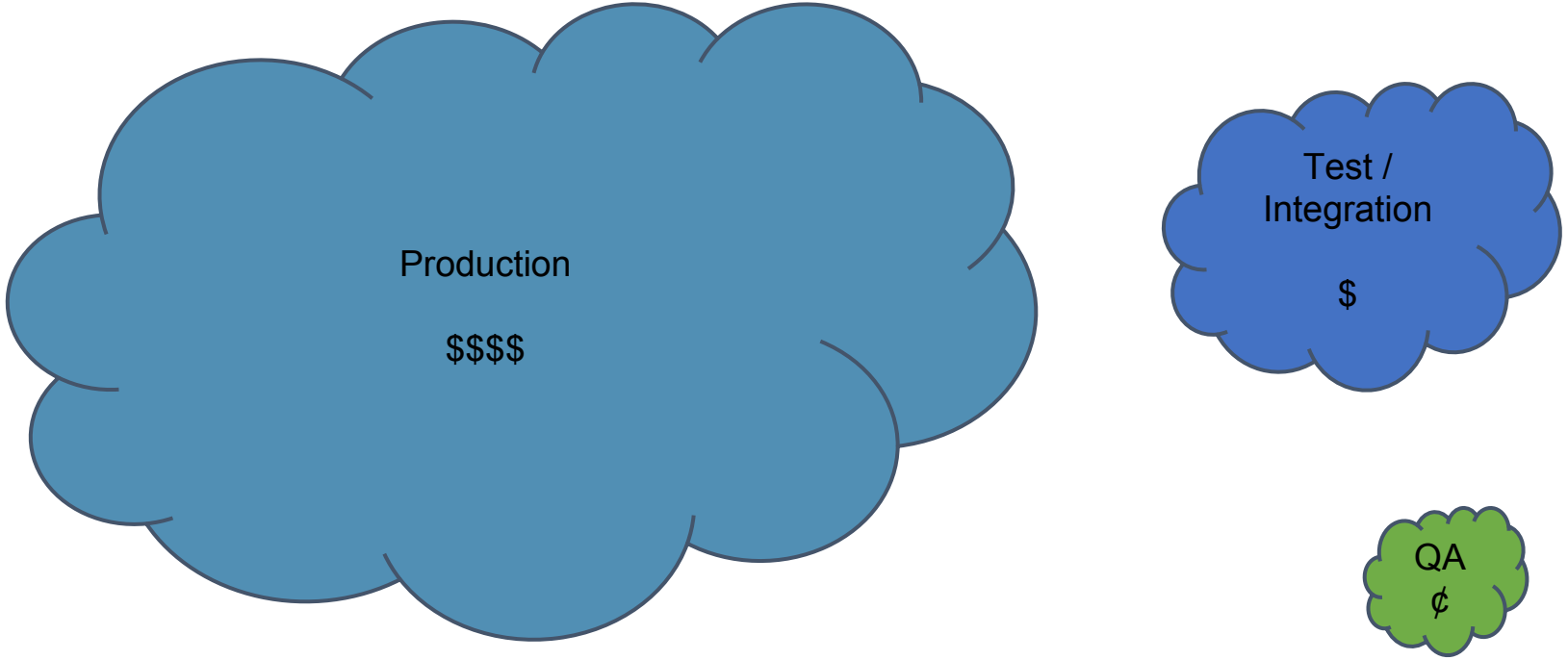
monthly release cycle

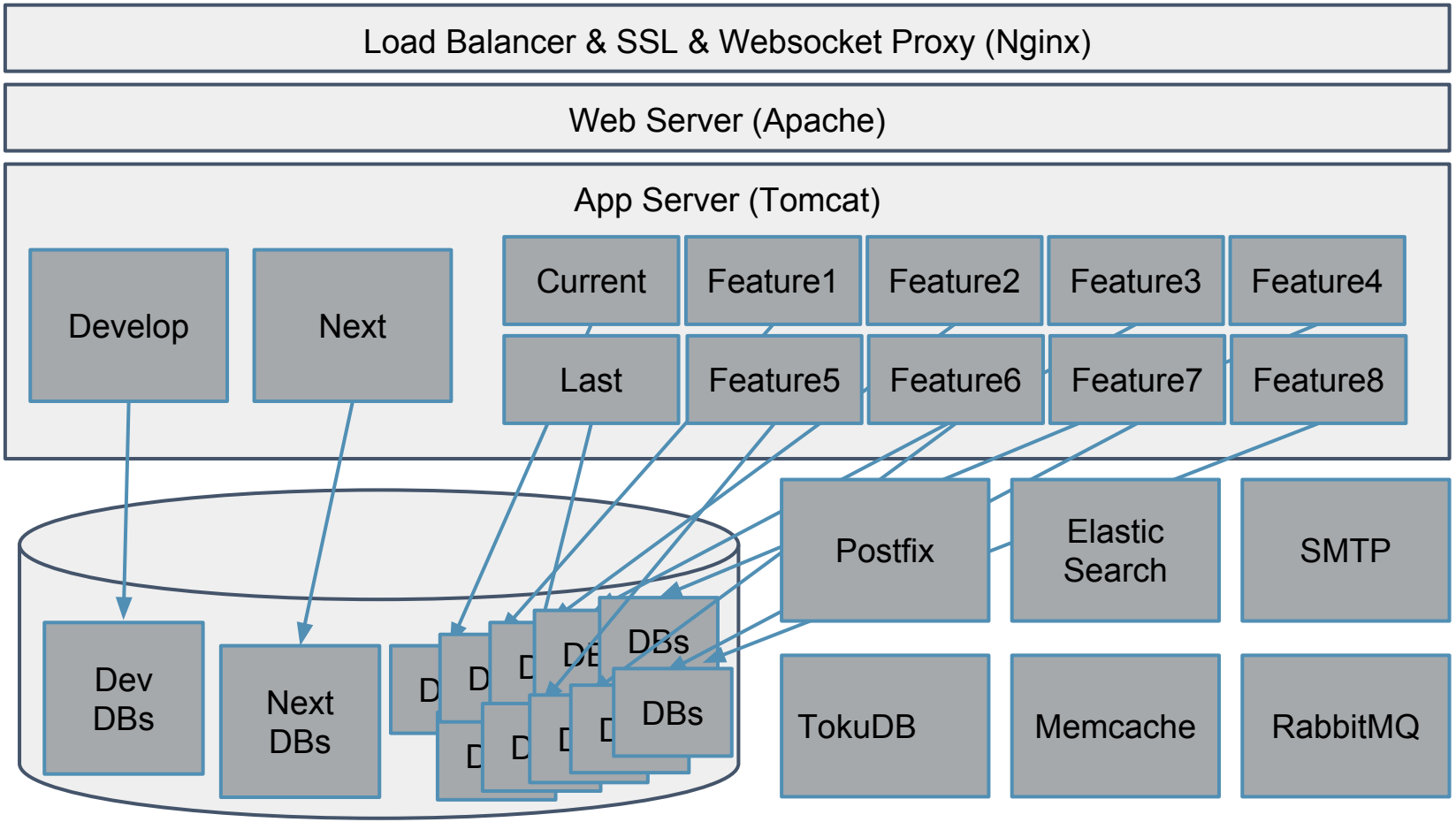


Challenges - How do you … ?

- provide flexibility for service teams to choose their technologies
- create consistent build pipeline for all apps and services
- in test environments
  - allow QA teams to deploy any combination of app/service version they want to test
  - Perform upgrade/rollback scenarios where multiple different versions of the same app/service are running at the same time
- in local development environments
  - simulate networked services on single host
  - make sure all developers are running current versions of all infrastructure, apps, and services

All environments are not created equal




Load Balancer & SSL & Websocket Proxy (Nginx)
Web Server (Apache)
App Server (Tomcat)
Develop
Next
Current
Feature1
Feature2
Feature3
Feature4
Last
Feature5
Feature6
Feature7
Feature8
Dev DBs
Next DBs
DBs
DBs
Postfix
Elastic Search
SMTP
TokuDB
Memcache
RabbitMQ

Pain Points

lack of isolation between lab slots

- rebuilds impacts everyone
- no flexibility with software versioning
- stability issues in one slot bleed into other slots

fixed hourly build schedule

lack of continuous integration testing

The solution - Docker + Jenkins

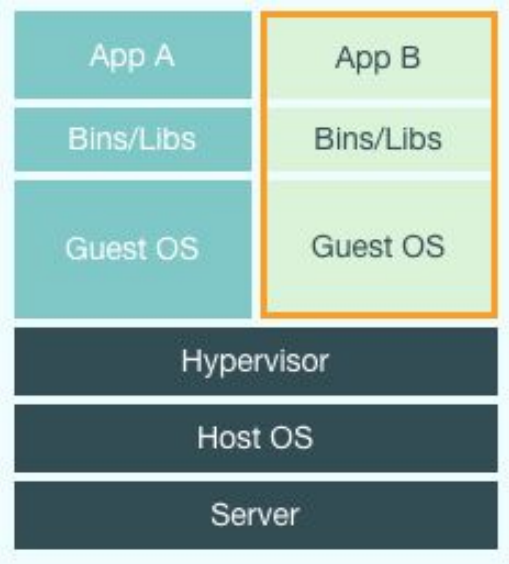
- April 2014 - start investigation and prototype with Docker 0.9.1
  - Migrate to Jenkins for build automation
- June 2014 - fully transitioned QA environment(s)
- Oct 2014 - transitioned local development environments
  - Vagrant + VirtualBox + docker + docker-compose
- July 2015 - deployed 2nd QA instance
- Sept 2015 - deployed 3rd QA instance
- March 2016 - consolidated to multi-host Swarm cluster
  - currently a 6 node cluster - 1 EC2 m4.large + 5 EC2 r3.2xlarge instances
  - 42 CPUs + 323 GB RAM
  - capacity ~20 concurrent Smartsheet environments

Docker Fundamentals

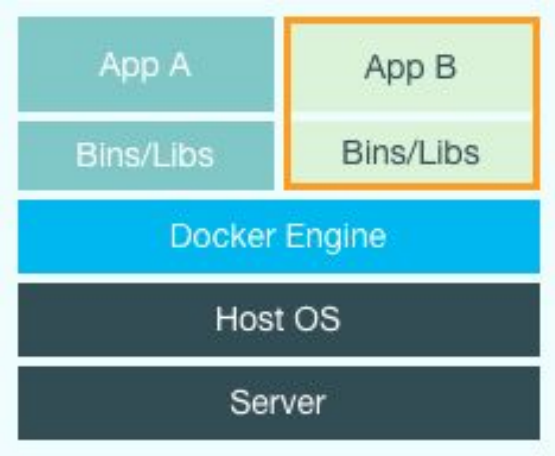
- Docker provides a lightweight runtime environment for applications called a container
- Many containers can run on a single host OS and share the same Linux kernel
- Each has its own isolated file system
- Similar to a VM but ...

Different than a VM

Traditional VM

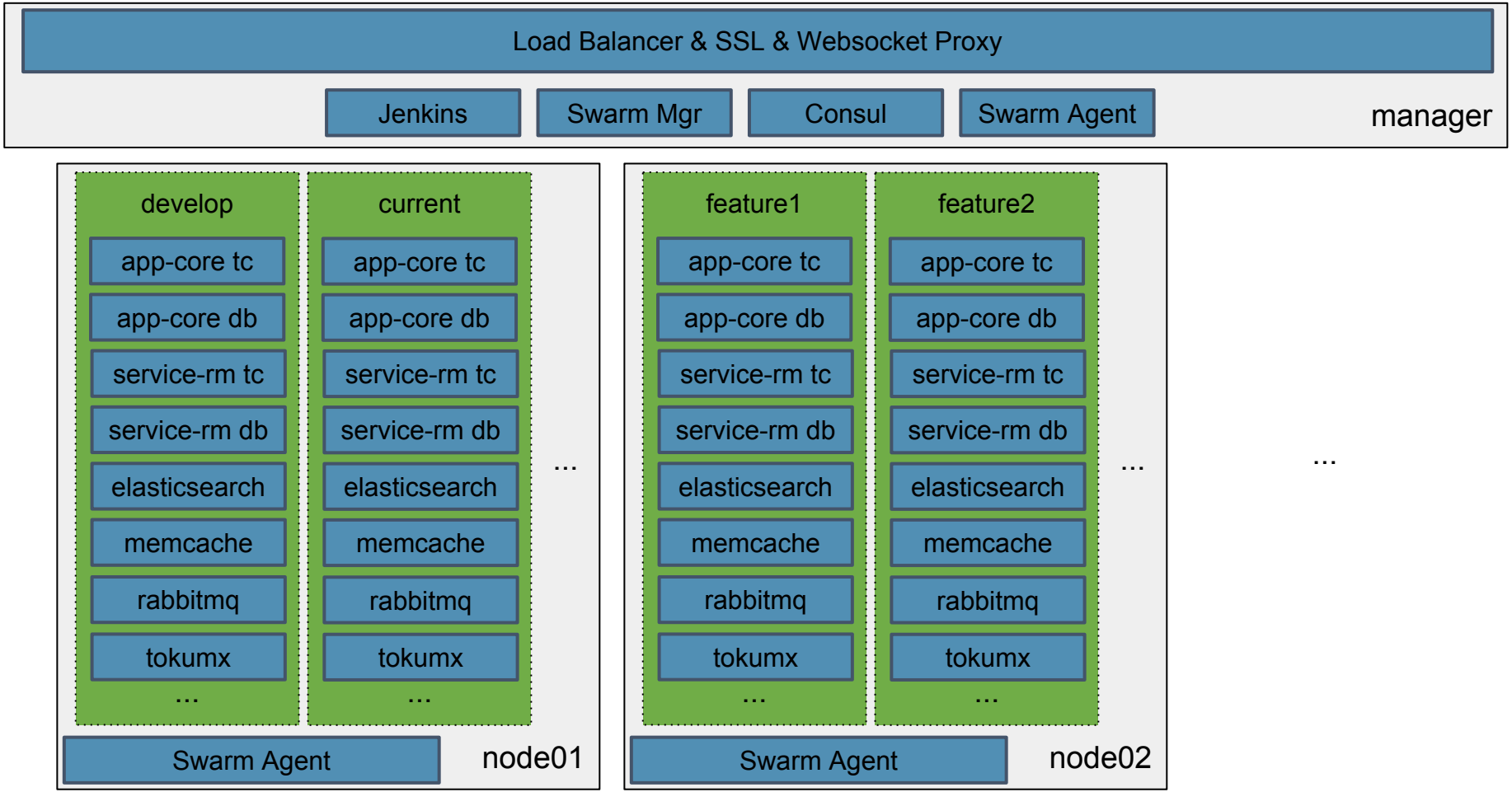


Docker



Docker Building Blocks

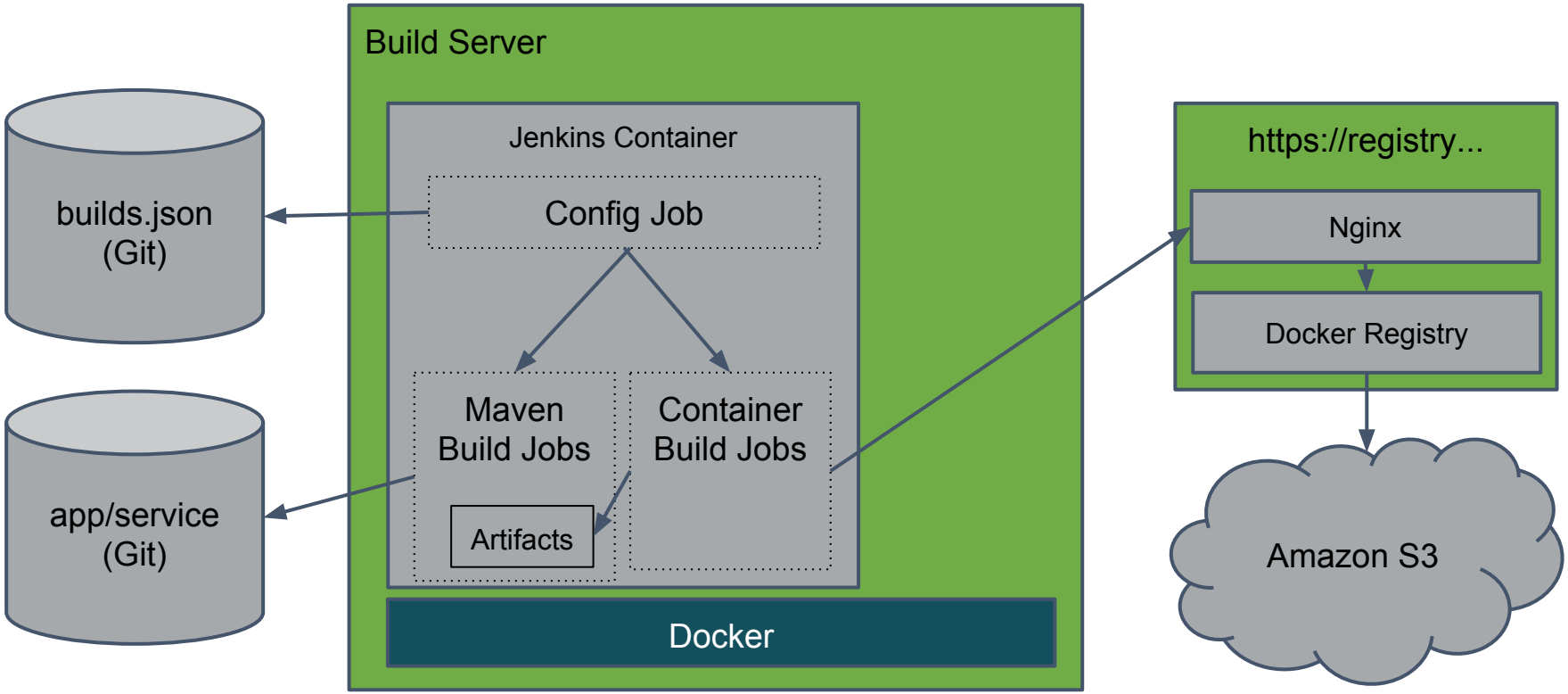
1. Docker Engine
   - runtime for containers
2. Dockerfiles
   - definition of software/config in containers
3. Docker Registry
   - central storage/sharing of containers
4. Docker Swarm
   - clustering solution to turn pool of hosts into single virtual cluster
   - separate product prior to 1.12 - now merged into Docker engine
5. Docker Compose
   - tool for defining and running multi-container applications


Load Balancer & SSL & Websocket Proxy
Jenkins
Swarm Mgr
Consul
Swarm Agent
manager
develop
app-core tc
app-core db
service-rm tc
service-rm db
elasticsearch
memcache
rabbitmq
tokumx
...
current
app-core tc
app-core db
service-rm tc
service-rm db
elasticsearch
memcache
rabbitmq
tokumx
...
...
Swarm Agent
node01
feature1
app-core tc
app-core db
service-rm tc
service-rm db
elasticsearch
memcache
rabbitmq
tokumx
...
feature2
app-core tc
app-core db
service-rm tc
service-rm db
elasticsearch
memcache
rabbitmq
tokumx
...
...
Swarm Agent
node02
...

Demo

Create a Service

App + Service Build Process



Demo

Build the Service + Container

Initial Dockerfiles

```
FROM registry.lab.smartsheet.com/smartsheetdev/java8tomcat:7
MAINTAINER smart.person@smartsheet.com

# Download our release WAR from Jenkins
RUN mkdir -p /opt/tomcat/webapps
RUN (wget -q --continue -O /opt/tomcat/webapps/servicefoo.war https://builds.lab.smartsheet.
com/jenkins/job/maven-service-builds/job/service-foo-release/lastSuccessfulBuild/artifact/target/smartsheet-
service-foo-1.0.0.war)

# Download our Tomcat context.xml
RUN mkdir -p /opt/tomcat/conf/Catalina/localhost
RUN (wget -q --continue -O /opt/tomcat/conf/Catalina/localhost/servicefoo.xml https://builds.lab.smartsheet.
com/jenkins/job/maven-service-builds/job/service-foo-
release/lastSuccessfulBuild/artifact/target/config/context.xml)

...
```

Improved Dockerfiles

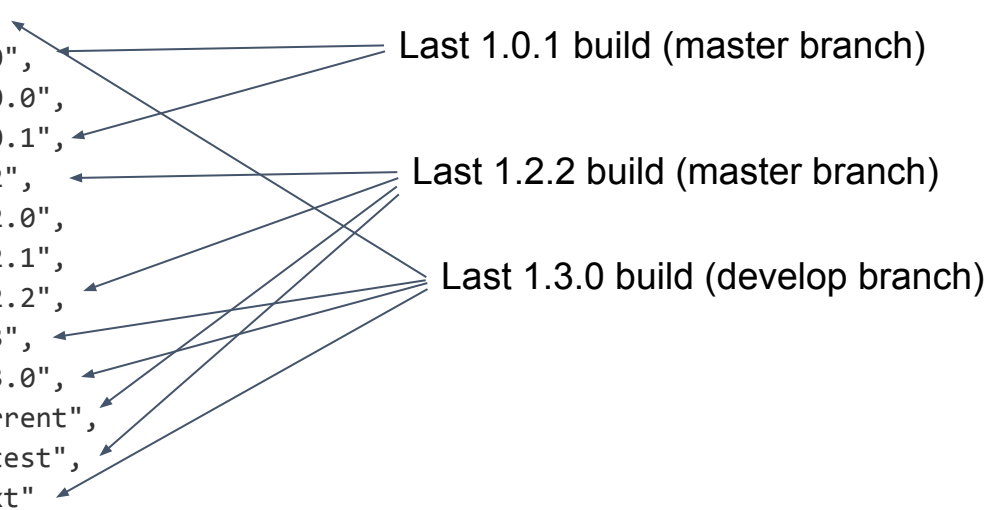
```
FROM registry.lab.smartsheet.com/smartsheetdev/java8tomcat:7
MAINTAINER smart.person@smartsheet.com

# Download our release WAR from Jenkins
RUN mkdir -p /opt/tomcat/webapps
RUN (wget -q --continue -O /opt/tomcat/webapps/servicefoo.war ${SMARTSHEET_SERVICE_FOO_WAR_FULL_URL})

# Download our Tomcat context.xml
RUN mkdir -p /opt/tomcat/conf/Catalina/localhost
RUN (wget -q --continue -O /opt/tomcat/conf/Catalina/localhost/servicefoo.xml ${CONTEXT_XML_FULL_URL})

...
```

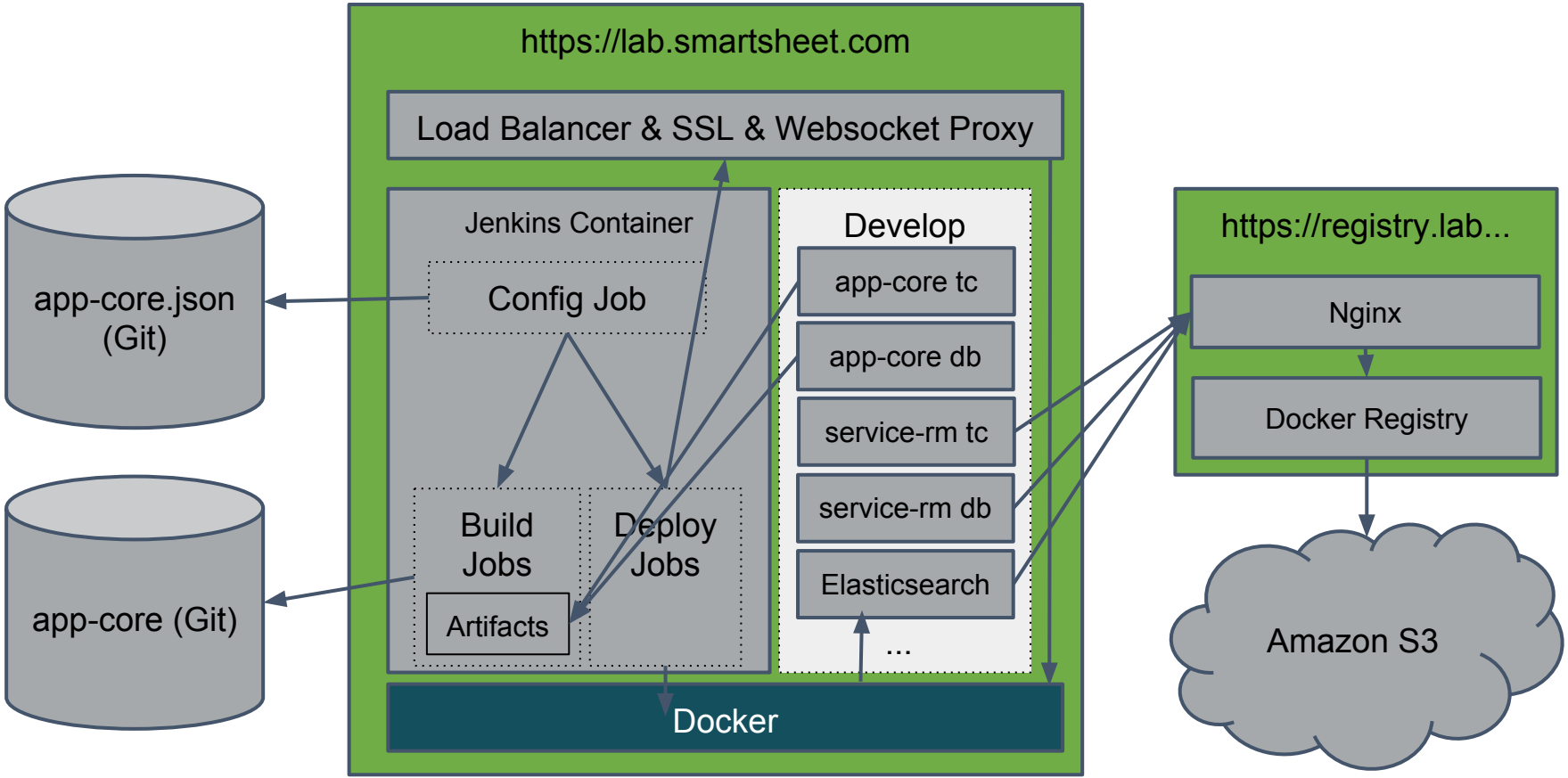
Templating Dockerfiles in Jenkins Job

```
# Get build artifacts for service-foo-snapshot
eval $(${WORKSPACE}/getArtifactEnvVars.py ${JENKINS_BUILD_URL})

# Generate Dockerfile for smartsheet-dev-service-foo next
cat ${WORKSPACE}/smartsheet-dev-service-foo/Dockerfile-next | envsubst > ${WORKSPACE}/smartsheet-dev-
service-foo/Dockerfile
${WORKSPACE}/build-images-jenkins.sh -f smartsheet-dev-service-foo -N "smartsheetdev/servicefoo" -t next

...
```

Registry Version History

```
➜  curl https://registry.lab.smartsheet.com/v2/smartsheetdev/servicefoo/tags/list | python -m json.tool
{
    "name": "smartsheetdev/servicefoo",
    "tags": [
        "1",
        "1.0",
        "1.0.0",
        "1.0.1",
        "1.2",
        "1.2.0",
        "1.2.1",
        "1.2.2",
        "1.3",
        "1.3.0",
        "current",
        "latest",
        "next"
    ]
}
```




https://lab.smartsheet.com
Load Balancer & SSL & Websocket Proxy
Jenkins Container
Config Job
Build Jobs
Deploy Jobs
Artifacts
Develop
app-core tc
app-core db
service-rm tc
service-rm db
Elasticsearch
...
Docker
app-core.json (Git)
app-core (Git)
https://registry.lab...
Nginx
Docker Registry
Amazon S3

Demo

Deploy Service + App

Logging

- everyone does logging differently - many options but no perfect one
  - mount host directory and write logs from containers there
  - log only to stdout/stderr - goes to the Docker "logs"
  - run log forwarder inside container and forward
- our approach - log forwarder in each container built into base image
  - ELK stack (Elasticsearch + Logstash + Kibana) per slot
  - events contain meta-data of container for filtering/searching
  - example

Local Development

we use Vagrant + docker-compose for local development

docker-compose.yml file checked into app-core repo

Vagrant image has Docker engine + Docker image cache

Vagrant VM syncs on startup to get latest images

```
➜  app-core git:(develop) ✗ vagrant up
....
==> develop: Pulling servicefoo (registry.lab.smartsheet.com/...)
==> develop: latest: Pulling from registry.lab.smartsheet.com/...
==> develop: Digest: sha256:59ea97dc12...
==> develop: Status: Image is up to date for registry.lab...:latest
....
```

Future Plans

Migrate to Docker 1.12 will simplify lab server deployment

- No need for Swarm Mgr, Swarm Agent, Consul, etc

Hybrid local-dev + remote EC2 environment

- current full smartsheet environment + dev tools is hitting 16 GB
- new Docker client for Mac/Windows removes need for Vagrant/VirtualBox
- securely networks across multiple hosts
- build + deploy some apps/services locally - everything else remote

Key Tips + Takeaways

Create a standard nomenclature

Generate your build jobs from scripts and config (checked into scm)

- Jenkins Job DSL Plugin is great once you get going (https://github.com/jenkinsci/job-dsl-plugin/wiki)

Version build artifacts and containers

- Use one system of record for version info - don't duplicate it in other places

Allow deploy of various combinations of artifacts for QA but default to simplest scenario

Questions?

Careers Page: http://www.smartsheet.com/careers

Dev Events: dev-events@smartsheet.com